Collaboration en temps réel pour la production de films d'animation numérique

Swann Martinez

▶ **To cite this version:**

Swann Martinez. Collaboration en temps réel pour la production de films d'animation numérique. Multimédia [cs.MM]. Université Paris 8, 2022. Français. ⟨NNT : ⟩. ⟨tel-04018090⟩

HAL Id: tel-04018090

https://hal.science/tel-04018090v1

Submitted on 7 Mar 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons CC BY-NC 4.0 - Attribution - Non-commercial use - International License

Université paris 8 - Vincennes Saint-Denis
École doctorale esthétique, Sciences et Technologies des Arts
Laboratoire Arts des Images et Art Contemporain (EA 4010)
Équipe de recherche Image Numérique et Réalité Virtuelle

Thèse

Pour obtenir le grade de
Docteur de l'Université Paris 8
Discipline : Esthétique, sciences et technologies des arts
Spécialité : Images numériques

Collaboration en temps réel pour la production de films d'animation numérique

Thèse Soutenue le 13/01/2022 par Swann Martinez
sous la direction de Mme Chu-Yin Chen

Membres du jury

Chu-Yin Chen (PR)	Université Paris 8
Réjane Hamus-Vallée (PR)	Université d'Évry
Rémi Ronfard (DR)	Inria Grenoble Rhone-Alpes
Cédric Plessiet (MCF HDR)	Université Paris 8
Anne-Laure George-Molland (MCF)	Université Paul Valéry Montpellier 3

« *Tout seul on va plus vite, ensemble on va plus loin.* »

Proverbe africain

Résumé

Un film d'animation est une œuvre collective. Il prend vie tout au long du processus de production. Du storyboard au compositing en passant par différentes étapes, son workflow s'apparente à une chaîne d'assemblage composé d'une succession linéaire de tâches. Chaque maillon représente presque un département. La collaboration s'effectue à travers des fichiers acheminés entre les tâches individuelles par une structure logicielle sous-jacente : le pipeline. Mais en échangeant un résultat et non le cheminement de la création entre les étapes, les artistes ne sont pas sensibilisés aux problématiques des départements voisins. Cette absence de communication nuit à l'anticipation des erreurs. Ainsi, un problème peut passer inaperçu sur plusieurs étapes, et, lorsqu'il ressort, la rigidité de la chaîne oblige à repasser à travers toutes les étapes à partir de son origine, un processus coûteux humainement et financièrement. L'univers du jeu vidéo se rapproche du cinéma. Les moteurs de rendus temps réels accélèrent les productions traditionnelles en réduisant considérablement les temps d'itérations des différentes étapes. Mais ils pourraient également devenir un nouvel outil de communication au sein des productions de film d'animation en amenant les graphistes 3D à collaborer entre eux en temps réel à travers une interface multi-utilisateur.

Cette thèse questionne l'apport du temps réel dans les processus de fabrication des films d'animation et propose une méthode de création plaçant le groupe artistique au cœur de la création à travers la recherche, le développement et l'expérimentation d'un outil de collaboration temps réel.

Mots-clefs : production, animation 3D, collaboration temps réel, multi-utilisateur, pipeline, interface homme-machine

Abstract

An animated film is a collective work. It comes to life throughout the production process. From storyboard to compositing through various stages, its workflow is like an assembly line composed of a linear succession of tasks. Each step is almost like a department. The collaboration is done through files carried between individual tasks by an underlying software structure : the pipeline. But, by exchanging a result and not the journey of creation between steps, the artists are not aware of the adjacent departments problematics. This lack of communication prevents the anticipation of errors. Thus, a problem can go unnoticed at several stages, and when it comes to light, the rigidity of the chain requires going through all the stages again from its origin, a costly process in human and financial terms. The world of video games is getting closer to cinema. Real-time rendering engines accelerate traditional productions by drastically reducing the iteration time of the different steps. But they could also become a new communication tool within animation film productions by bringing 3D graphic designers to collaborate with each other in real time through a multi-user interface.

This thesis questions the contribution of real time in the manufacturing process of animated films and proposes a creation method placing the artistic group at the heart of the creation through the research, development and experimentation of a real time collaboration tool.

Key words : production, 3D animation, real-time collaboration, multi-user, pipeline, human-machine interface

Remerciements

Je tiens à remercier ma directrice de thèse Chu-Yin Chen et Valentin Moriceau, mon encadrant chez Cube Creative Computer Company pour leurs encouragements et conseils qui m'ont guidé durant ces trois années. Mes remerciements vont aussi aux membres du jury pour l'intérêt porté à cette thèse.

Merci aux gens de Cube Creative que j'ai interrogés et que je cite dans ma thèse : Maxime Harenczyk, Tanguy Weyland.

À mes collègues de Cube Creative sans qui rien de tout cela n'aurait été possible, avec qui nous avons fait de belles choses, partagé ensemble des moments forts. Merci à Valentin, Cécile, Axel, Sylvain, Majid, Pascale, Fabian, Maël, Adrien, Guillem, Maxime (x2), Maximilien, Line, Anthoine, Anthony, Clémence, Bruno, Gorgio, Tristan, Claire, Marine, Romain.

Aux membres de l'INREV à qui ce travail doit beaucoup. En particulier merci à Cédric Plessiet, Rémy Sohier, Jean-François Jégo, Anne-Laure George-Molland, Marie-Hélène Tramus et Vincent Meyrueis. Aux doctorants Céline, Isadora, Ponara, Maëlys, Julien, Gaëtan, Guofan, Laeticia, Tsovinar pour leur soutien et conseils.

Merci au collectif OpenPipe de m'avoir inspiré et en particulier à Yannick, Laurent, Philippe pour leur soutien sur le *Multiuser*.

À la communauté du *Multiuser* pour son support. En particulier à Fabian Adam, NotFood, Poochyc, Ultr-X, Wuaieyo, Softyoda pour leur participation à cette aventure collaborative.

À l'équipe du projet Corps Infini.

Aux copains pour leur soutien, merci à Mathieu, Colin, Quentin, Iris, Nilo, Nicolas, Adrien, Stéphane, Manon, Marion, Étienne, Thibault, Pierre, Marc et Samuel.

À mes parents et mes frères qui m'ont supporté dans les moments difficiles. À Laure pour ses bulles d'oxygène dans les périodes délicates.

Table des matières

Résumé **5**

Remerciements **9**

Table des matières **11**

Introduction Générale **15**

Partie I : Évolution du processus de production des films d'animation **19**

Introduction **21**

I.1 Production d'images de synthèse précalculées **23**
I.1.1 Étapes de production 25
I.1.1.1 Préproduction : planification, recherche 25
I.1.1.2 Production : fabrication, assemblage 27
I.1.1.3 Postproduction : affinage, export 28
I.1.2 Segmentation du travail 30
I.1.2.1 Une grande variété de corps de métiers 30
I.1.2.2 Départements 30
I.1.2.3 Chaîne de communication 34
I.1.3 Au cœur de la chaîne de fabrication : le pipeline 37
I.1.3.1 À l'origine, un ensemble de besoins et de contraintes 38
I.1.3.2 Au cœur de la collaboration : des fichiers 40
I.1.3.3 Un ensemble d'outils logiciels 44

I.2 Vers une fusion des méthodes de fabrication **49**
I.2.1 Spécificités des moteurs de jeu 50
I.2.1.1 Structure des projets 50
I.2.1.2 Types de données 50
I.2.1.3 Structure de fichiers 51
I.2.1.4 Dataflow 51
I.2.1.5 *Asset manager* 52
I.2.1.6 Systèmes de gestion de versions pour l'image de synthèse . . . 53
I.2.2 *Virtual production*, carrefour du temps réel et du précalculé 56
I.2.2.1 Techniques de virtual production 56
I.2.2.2 Un workflow compatible avec le film d'animation ? 60
I.2.3 Explorations initiales 68
I.2.3.1 *Smartphone Remote*, une collaboration multi *device* 68

I.2.3.2 Le Corps Infini, une collaboration humaine et interdisciplinaire pour la création d'une narration en réalité virtuelle 71
I.2.3.3 *Open Pipe* : réflexions avec des vétérans du pipeline 73

I.3 Enjeux de la co-création temps réel pour l'image de synthèse 77
I.3.1 Co-construction temps-réel dans d'autres disciplines 78
I.3.1.1 Dans le Bâtiment et Travaux Publics 78
I.3.1.2 Dans le développement informatique 81
I.3.2 Collaboration en temps réel pour l'image de synthèse 84
I.3.2.1 Enjeux techniques et humains 84
I.3.2.2 Historique des solutions de collaboration 86

Conclusion 89

Partie II : Recherche et développement d'un outil de co-création en temps réel pour l'animation 91

Introduction 93

II.1 *Multiuser* : la collaboration en temps réel au cœur de la création 95
II.1.1 Spécificités d'un DCC 3D 96
II.1.2 Un framework pour la création collaborative de contenu numérique 3D 98
II.1.2.1 Architecture du framework 99
II.1.2.2 Protocole de définition des données répliquées 102
II.1.2.3 Contrôle d'accès concurrentiel 103
II.1.2.4 Fonctions conscientes de la collaboration 103
II.1.2.5 Interfaces conscientes de la collaboration 104
II.1.2.6 Sauvegarde des sessions 106
II.1.3 Intégration du framework dans *Blender* : l'*add-on Multiuser* 107
II.1.3.1 Définition des *datablocks* répliqués 108
II.1.3.2 Présence des utilisateurs 110
II.1.3.3 Outils de collaboration 111
II.1.3.4 Mécanismes de droit 113
II.1.3.5 Une gestion des mises à jour tumultueuse 113
II.1.3.6 Historique des actions 114

II.2 Mise en situation de l'outil : Expériences de co-création 117
II.2.1 Protocole d'expérimentation 118
II.2.2 Expérimentations académiques 120
II.2.2.1 Enseignement de la modélisation 3D 120
II.2.2.2 Enseignement du pipeline complet de création d'*asset* 3D . . 122
II.2.2.3 Enseignement du *lighting* 124
II.2.3 Expérimentations industrielles 126
II.2.3.1 Un début mouvementé 126
II.2.3.2 Efficacité de la collaboration 127
II.2.3.3 Intégration d'élément du pipeline 128
II.2.3.4 Observations qualitatives 130
II.2.3.5 Compatibilité avec l'animation *keyframe* ? 132
II.2.4 Expérimentations publiques 135
II.2.4.1 Émergence d'une communauté 135

II.2.4.2 Support de session sur internet . . . 135
II.2.4.3 Un développement participatif . . . 136
II.2.4.4 Un *workflow* collaboratif à distance . . . 138
II.2.4.5 Un espace de co-création persistant ? . . . 143
II.2.4.6 Organisation de la scène . . . 145
II.2.4.7 Superposition du travail . . . 146

II.3 Collaboration en temps réel au cœur d'une production traditionnelle 151
II.3.1 Applications du *Multiuser* dans un pipeline de série animées . . . 152
II.3.1.1 *Briefs layout* et *storyboard* . . . 152
II.3.1.2 Formation des nouveaux graphistes . . . 153
II.3.1.3 *Look development* . . . 155
II.3.1.4 Découpage de séquence . . . 155
II.3.2 Recherches et développements futurs . . . 156
II.3.2.1 *Rigging* avancé . . . 156
II.3.2.2 gestion de versions . . . 156
II.3.2.3 Système de cache . . . 157
II.3.2.4 Système de rôles . . . 157
II.3.2.5 Administration serveur . . . 158

II.4 Vers des productions agiles ? 161
II.4.1 Méthodes agiles . . . 163
II.4.1.1 Définition . . . 163
II.4.1.2 Scrum . . . 164
II.4.2 Méthodes agiles pour l'image de synthèse . . . 169
II.4.2.1 Production agile . . . 169
II.4.2.2 Vers une création agile ? . . . 174

Conclusion 181

Conclusion Générale 183

Liste des acronymes 191

Glossaire 195

Bibliographie 207

Table des figures 211

Liste des tableaux 215

Annexe A Interview de Maxime Harenzyk 217

Annexe B Extrait du planning du projet Mush Mush 221

Annexe C Liste des outils développés à CUBE Creative 225

Annexe D Script de dessin du graphe de réplication 229

Annexe E Projets collaboratifs menées avec le Multiuser 231

Introduction Générale

L'image de synthèse telle que nous la connaissons aujourd'hui est apparue dans les années 1970. À leurs débuts, les ordinateurs n'étant guère plus puissants qu'une calculatrice actuelle, plusieurs heures voire plusieurs jours de calculs étaient nécessaires pour obtenir le rendu d'une seule image. C'est de cette période qu'est né le rendu dit précalculé [1]. Toutefois, bien avant d'effectuer le calcul d'une image de synthèse, on doit la concevoir. Cette étape de création demande un certain nombre de ressources humaines en fonction du domaine d'application. Dans le cadre du cinéma d'animation, de nombreux corps de métier collaborent pour traiter et créer tous les aspects (modélisation, animation, lumière, etc.) de l'image ou du film en conception. Chacun de ces domaines de création nécessite des logiciels et des outils spécifiques. Afin de coordonner et d'articuler tous ces acteurs, tous ces outils et tous ces flux de données, un solide pipeline [2] de production est nécessaire. Ce dernier garantit la fluidité et l'efficacité du travail des différents artistes.

Dès les premiers films d'animation en images de synthèse dans les années 90 comme *Toy's Story*[36], les premiers outils de création et les premiers processus de production (que l'on peut appeler méthodes de création) furent donc conçus afin de fonctionner en adéquation avec des systèmes de rendus précalculés nécessitant des parcs entiers d'ordinateurs pour calculer chaque image.

Parallèlement, l'image de synthèse dite interactive fait son apparition dans les années 1980 avec le jeu vidéo notamment. Elle se développe avec l'évolution de la puissance de calcul des ordinateurs ; sa particularité étant de faire des rendus dits temps réel [3] rendant possibles les interactions avec le joueur. Ce type de rendu nécessitant beaucoup de puissance de calcul, des compromis qualitatifs étaient alors nécessaires (par exemple l'utilisation de *sprites*, de lumières précalculées, etc.). Le niveau de détail de l'image était bien moins avancé en comparaison des images précalculées de l'époque.

Mais aujourd'hui, les nouvelles technologies de rendu temps réel arrivent à un niveau qualitatif sans précédent. Avec une résolution élevée et des algorithmes de rendu photoréaliste s'approchant de la réalité physique, la distance existant jadis entre précalculé et temps réel s'amincit.

Mon premier contact avec l'utilisation du temps réel pour le cinéma se produisit lors de mon alternance de Master 2 Art et Technologie de l'Image (ATI) [4] dans la société

1. Le rendu de précalculé se caractérise par une absence d'interactivité, car le calcul d'une image demande du temps.

2. Un pipeline représente l'ensemble des outils et processus logiciels conçus pour répondre aux besoins d'une production.

3. On parle de rendu temps réel lorsque le temps de calcul des images est en dessous d'un quinzième de seconde. Cette catégorie englobe les technologies du jeu vidéo et du multimédia interactif nécessitant de calculer un nombre élevé d'images par seconde.

4. Formation Art et Technologie de l'Image à l'université Paris 8. `https://www.ati-paris8.fr/`

SolidAnim [5]. En y développant des outils de *previs on set* [6] sur des moteur de jeu [7], j'ai été sensibilisé aux possibilités offertes par ces technologies émergentes. C'est avec la volonté d'approfondir leurs usages pour le cinéma d'animation que je suis rentré en contact avec Valentin Moriceau et Sylvain Grain respectivement directeur Recherche et Dévelopement (R&D) [8] et producteur à Cube Creative. Par des échos d'anciens étudiants et des enseignants d'ATI, j'avais pris connaissance de leurs recherches d'utilisation de moteur de jeu pour faire du rendu sur leur série *Kaeloo*. Cette ouverture d'esprit me plaisait beaucoup. Nous avons rapidement constaté notre convergence d'intérêts pour l'exploration de ces méthodes au cœur de la fabrication d'image animée et nos discussions donnèrent naissance à ce projet de thèse.

J'ai donc intégré l'équipe de R&D de Cube Creative à travers une Convention Industrielle de Formation par la Recherche (CIFRE) [9]. Fondée en 2002, cette société produit de l'animation 2D, 3D et des effets visuels numériques. Par la diversité de ses projets, et surtout parce qu'elle apporte un grand soin à développer des projets écrits en interne, elle lie étroitement art et technologies.

Ma mission au sein de l'équipe a été double. D'une part, j'ai été ponctuellement amené à concevoir, tester et déployer certains outils pour les productions audiovisuelles (lorsqu'ils partageaient des liens avec mon sujet de recherche) ; ceci me permit d'acquérir une vision globale de la chaîne de fabrication en place. D'autre part et principalement, j'ai recherché et développé une méthode de création exploitant le temps réel à des fins de collaboration. J'ai également eu la chance de participer à de nombreuses rencontres et conférences qui œuvrèrent grandement à l'émergence de nos travaux. L'ouverture de Cube Creative sur la communication nous offrit d'excellentes conditions pour mener notre recherche de façon ouverte et de publier les outils développés sous licence *open source*. Ainsi, nous avons régulièrement partagé notre avancement lors de conférences et colloques.

La pluralité des contextes offerts par le dispositif CIFRE a constitué un terreau propice à l'émergence d'une méthode de création. Les expérimentations menées dans l'enseignement et l'industrie ont directement nourri nos développements et orienté notre recherche vers de nouveaux horizons collaboratifs pour la création d'images de synthèse. Notre démarche s'inscrit donc dans un processus de recherche création.

La démocratisation des moteurs de rendu temps réel et leurs utilisations dans les productions de films d'animation numérique remettent en question leurs processus de

5. `https://www.solidanim.com`

6. La *previs on set* est une technique de prévisualisation interactive. Elle rassemble plusieurs unités selon le budget et les besoins de la production. Généralement, elle compte une unité de *motion capture* responsable de la capture des mouvements des acteurs, une unité de rendu responsable de la visualisation temps réel de l'aperçu des effets visuels.

7. Un moteur de jeu ou *game engine* est un logiciel qui offre une suite d'outils et de fonctionnalités aux développeurs de jeux afin de construire efficacement leurs jeux vidéo. Depuis quelques années, certains moteurs de jeu proposent aussi des outils pour la création de films en image de synthèse.

8. Dans l'industrie du cinéma d'animation, la recherche et développement (R&D) est une pratique consistant à mener des recherches fondamentales ou appliquées sur un sujet donné, et à développer des procédés, algorithmes et programmes. Dans un studio d'animation, l'équipe R&D fait de la veille technologique et développe des solutions adaptées aux problématiques de production.

9. Une Convention Industrielle de Formation par la Recherche permet à un doctorant de réaliser sa thèse en entreprise en menant un programme de recherche et développement en liaison avec une équipe de recherche universitaire.

conception et de création. Aussi, cette recherche s'est attachée à questionner, tout d'abord, quel pourrait être l'apport de ces technologies pour que les artistes aient une meilleure maîtrise de la création de l'image dans le cinéma d'animation numérique. Puis, quelles seraient les méthodes de création temps réel permettant un travail collaboratif créatif parallèle et simultané dans une production de cinéma d'animation. Enfin, quelles seraient les technologies les plus adaptées pour expérimenter ces nouvelles méthodes.
Les domaines du cinéma et du jeu vidéo se rapprochent. Cependant, bien que ces techniques de rendu temps réel soient abouties d'un point de vue esthétique, les méthodes actuelles de création des productions de cinéma d'animation ne sont pas adaptées à une production exploitant les capacités du temps réel : elles n'en supportent pas les spécificités et ne sont pas prévues pour fonctionner avec elles. Inversement, les pipelines de production issus du temps réel ne sont pas conçus pour supporter les spécificités d'une production de cinéma d'animation.

Ainsi, l'introduction de techniques de rendu temps réel remet profondément en question le fonctionnement de la production, de manière globale et sur tous ses pôles, notamment en termes de potentialités créatives. L'immédiateté du rendu des techniques temps réel offrirait aux artistes une meilleure maîtrise de la création de l'image et de son esthétique dans le cinéma d'animation numérique. Il nous semblait aussi que des méthodes innovantes de création favorisant un travail collaboratif, créatif et simultané émergeraient d'expérimentations de production temps réel.

Ces techniques n'auraient pas pour seule finalité de réaliser du rendu livrable en un temps record. Pour chacun des différents départements impliquant le travail de l'image ou du son, il faudra remettre en question les anciennes pratiques au profit de cette nouvelle méthode. En effet, les processus de fabrication d'images actuellement en place en production sont séquentiels dans leurs exécutions (organisés chronologiquement étape après étape), ce qui limite aux artistes la possibilité d'un travail simultané sur une même scène. Cette limitation est remise en cause par l'interactivité qu'apporte l'arrivée du temps réel au cœur même des différents départements de la production.

Cette thèse se développe en deux parties. La première partie pose le contexte de la recherche. Un premier état des lieux décrit les processus créatifs traditionnels du cinéma d'animation numérique et leurs verrous méthodologiques. Ces éléments nous permettront de comprendre les leviers technologiques liés à l'intégration des moteurs de jeu dans ces processus et nous conduira naturellement à envisager les enjeux d'une expérience collaborative temps réel au sein des logiciels de création de contenu numérique (ou Digital Content Creation software (DCC)).

La seconde partie décrit la recherche, le développement et l'expérimentation d'une méthode de co-création en temps réel pour l'animation. Nous exposerons l'outil de collaboration ainsi que les expérimentations en milieu industriel, académique et communautaire qui permirent son émergence à travers une démarche de recherche-création. Nous ferons ensuite état des cas d'application concrets envisagés au sein du pipeline de Cube Creative dans la continuité de ces travaux.

Enfin, notre thèse prendra du recul et proposera des pistes de réflexion pour une restructuration plus profonde de l'appareil de production à travers l'utilisation de notre méthode.

L'enjeu majeur de cette recherche a été de s'affranchir des contraintes du précalculé

pour explorer une nouvelle logique créative parallèle, organisée et efficace, en centrant les différents axes de recherche autour d'expériences collaboratives. Et comme l'écrit Anne George-Molland :« *par les technologies temps réel* [on peut accéder à] *une immédiateté dans le processus à la fois de création et de décision, retrouver une spontanéité de l'imaginaire et en même temps une forme d'improvisation, d'expérimentation en direct* » ([9]).

Première partie

Évolution du processus de production des films d'animation

Introduction

Un film d'animation est le fruit d'un savant mélange d'art et de technologie et d'huile de coude. À travers l'histoire, la production d'images animées a toujours eu un lien étroit avec l'évolution technologique. Cette partie étudie la transformation des modes de productions d'images de synthèse précalculées portée par la montée en puissance des technologies temps réel.

Nous commençons par établir un état de l'art des procédés de fabrication actuellement en place dans les studios d'animation et de leurs limites.

Cette base technique et méthodologique facilitera la compréhension du second chapitre qui explore la profonde remise en cause débutée par l'arrivée des moteurs de jeu dans les productions audiovisuelles. Nous relèverons ainsi différents verrous technologiques qui nous conduirons à questionner la capacité des logiciels de création 3D à supporter une expérience collaborative temps réel.

Enfin, nous exposerons les enjeux techniques et humains impliqués par la création d'une solution de collaboration en temps réel pour le cinéma d'animation.

Chapitre I.1

Production d'images de synthèse précalculées

Introduction

Un film d'animation est une idée matérialisée tout au long du processus de production par de multiples acteurs différents, elle se cristallise à travers de nombreuses étapes définies et interdépendantes. Tout au long de ce chapitre, nous décrirons cette structure à la fois humaine et logicielle qui porte la collaboration au sein des studios de création. Souvent qualifiée de « chaîne de fabrication », nous dépeindrons ses bénéfices et inconvénients à travers sa similarité avec les chaînes de montage automobiles issues du fordisme.

Bien que le procédé de fabrication varie selon le format de distribution (série, film, publicité, etc.), il n'en reste pas moins similaire sur de nombreux aspects. Néanmoins, il est important de préciser que notre exposé sera principalement nourri d'éléments issus de la production des séries d'animations tirés de notre expérience à Cube Creative.

Les individus derrière la création d'un film peuvent être segmentés en trois grandes familles : les équipes de production, les équipes artistiques et les équipes techniques. Les équipes de production (aussi appelée *prod* dans le milieu) sont au cœur de la gestion de l'humain et du suivi de la fabrication. Elles sont garantes de la planification et du respect du budget initialement fixé. La création des images et des *assets*[10] est assurée par les équipes artistiques qui appliquent des procédés mis en place par l'équipe technique. Ces procédés sont ensuite ancrés et automatisés par les outils fournis par cette dernière.

Maxime Harenczyk, chargé de production chez Cube Creative associe la nature des échanges entre ces trois familles à une relation de co-dépendance. Selon lui :

« *Ce sont les équipes artistiques et techniques qui créent la série. L'équipe de production, elle, sert d'huile dans les rouages. Ou de ficelle qui aide le tout à coulisser ensemble.* » (Voir Annexe A)

La Figure I.1.1 esquisse les relations entre les trois équipes. La production s'apparente à une membrane assurant les échanges entre les demandes des éléments extérieurs au studio et les équipes artistiques et techniques internes responsables de la création. Les éléments externes au studio ne sont pas les mêmes suivant les projets. Sur les séries produites par le studio, ils sont principalement des organes de distributions (chaînes

10. Les *assets* sont utilisés pour le suivi de ce qui doit être réalisé pour le projet. Nous ne parlons pas des fichiers numériques, mais du concept abstrait de ce qui doit être accompli, comme un personnage, un véhicule, un objet, un décor, etc. Définition par LePipeline.org.

de télévision, marques, etc.). Si le studio joue le rôle de sous-traitant sur une phase du projet, la société de production et même le réalisateur pourront être externes au studio. Nous détaillerons la relation étroite existant entre ces trois familles en Sous-section I.1.2.3.

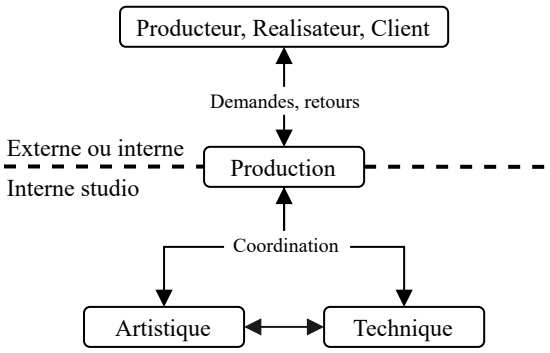


FIGURE I.1.1 – Schéma simplifié des relations entre les grandes familles opérant au sein de la production d'un film d'animation.

Ce chapitre fait état du processus de production industrielle actuel à l'origine des films d'animation. Un premier survol des phases de fabrication dépeint les grands enjeux des différents acteurs tout au long de la production. Notre attention se portera ensuite sur ces acteurs ainsi que leurs relations structurelles au sein de la chaîne de fabrication afin de mettre en lumière la circulation de l'information et ses limites au sein du studio. Enfin, nous nous attarderons sur la nature de cette information ainsi qu'aux outils utilisés pour la distribuer.

I.1.1 Étapes de production

Le processus de création d'un film d'animation en images de synthèse se décompose en trois grandes étapes interdépendantes et linéaires :

1. La préproduction.
2. La production.
3. La postproduction.

Selon l'échelle du projet, il est courant que chacune de ces phases soit sous-traitée par de multiples sociétés spécialisées dans chaque aspect de la création (telle que la prévisualisation [11] par exemple). Les plus grandes productions peuvent regrouper parfois plusieurs dizaines de compagnies spécialisées pour répartir l'importante charge de travail que représente la phase de production.

La société Cube Creative étant elle-même souvent productrice, j'ai eu la chance de suivre la progression de projets du début à la fin pour appréhender les enjeux et limites de la collaboration au sein du modèle de production actuellement en place.

Dans cette section, nous introduirons les principales phases de création d'un film d'animation en image de synthèse. Nous analyserons cet enchaînement d'étapes de progression (qualifié de *workflow* [12]) à travers le prisme des enjeux des différentes équipes créatives (production, artistique et technique) afin de peindre une fresque temporelle et humaine du processus de création d'un film.

I.1.1.1 Préproduction : planification, recherche

Le cycle de création d'un film débute par une phase de recherche et de planification : la préproduction. C'est une période critique durant laquelle les différentes équipes vont bâtir les fondations graphiques et techniques du projet. Si le film était une maison, la préproduction représenterait l'esquisse des plans [13]. Durant cette période, les équipes vont identifier au mieux les difficultés techniques (par exemple, les premiers concepts graphiques représentent parfois des éléments trop complexes au regard du budget fixé). Une bonne anticipation des problèmes permettra de minimiser les risques et le coût de la production à venir.

La Figure I.1.2 fait état des principales étapes de création artistique présentes en préproduction en fonction du temps. Les trames narratives sont développées durant la phase de scénario. Lorsque l'univers scénaristique est assez développé, les dialogues sont écrits au moment du script puis enregistrés. Parallèlement, les premières représentations graphiques 2D de l'univers prennent vie lors de la phase de design. Ces

11. La prévisualisation désigne l'ensemble des outils permettant de prévoir, de manière visuelle, le résultat d'un tournage à venir. C'est une technique de description et d'assistance à la mise en scène. Étape fondamentale dans la conception d'un film en prise de vue réel, la prévisualisation permet à un réalisateur, un directeur de photographie ou encore un superviseur Special Effect (FX) d'expérimenter différentes mises en scène ou directions artistiques telles que l'éclairage, le cadrage et le montage pour un budget réduit. Plus communément, on appelle aujourd'hui prévisualisation ce qui porte en fait le nom d'animatique, c'est-à-dire une séquence d'images en mouvement simulant la version finale d'un plan ou d'une scène de film telle qu'elle apparaîtra à l'écran. Défini partiellement par Benoit Melançon [15].

12. Le *workflow* décrit les différentes étapes de progression d'un projet du point de vue des départements de travail. Définition par LePipeline.org

13. Un plan au sens cinématographique du terme est une prise de vue, comprise entre la mise en marche de la caméra et son arrêt. Le plan est la plus petite unité temporelle utilisée pour segmenter le travail de création d'un film.

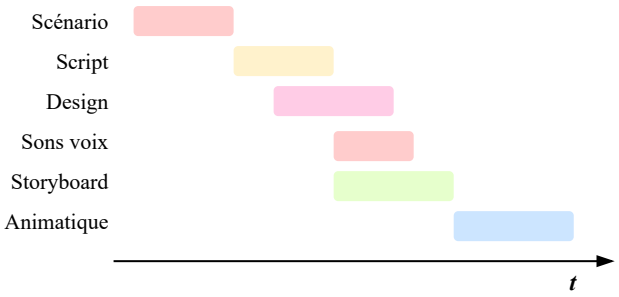


FIGURE I.1.2 – Étapes artistiques de préproduction en fonction du temps

premières phases sont fondamentales pour bâtir les fondations de l'univers graphique du projet. Elles faciliteront également les décisions techniques quant à leur réalisation. Dès lors que les designs sont entamés, le *storyboard*[14] (ou scénarimage) va dérouler des plans clefs qui seront utilisés comme référence pour créer l'animatique. En fonction des projets, l'animatique peut prendre la forme d'une séquence[15] 2D ou 3D qui apportera au *storyboard* une dimension sonore (principalement les voix) et temporelle. Elle validera les raccords ainsi que les timings des différents plans. L'animatique peut être considérée comme la dernière phase de la préproduction.

Parallèlement, les équipes techniques testeront et adapteront le pipeline existant pour le préparer au mieux à la phase de production. Nous aborderons une description détaillée du pipeline en Section I.1.3.

Lors de la préproduction, l'enjeu principal de l'équipe de production sera la construction de solides fondations pour que le projet se déroule sereinement. Maxime Harenzyk explique que ces fondations se font par ordre chronologique avec la prise en main de la série :

« *Il faut établir budget, planning et mettre tous les jalons en place pour que la suite se passe bien, que l'écriture se lance et se fasse, qu'on ait un processus de validation avec le ou les producteurs* » (Voir Annexe A). Nous avons mis le planning de la série *MushMush* à titre d'exemple en Annexe B.

Maxime décrit également un ensemble de tâches plus concrètes de dépouillements[16] (ou *breakdown*) d'*asset*, de scripts ainsi que leur gestion pour préparer les enregistrements, etc.(cf. Annexe A)

Plus généralement, selon Maxime, l'enjeu majeur de l'équipe de production « sera de tout caler sur des rails pour que la suite se passe de manière saine » (Voir Annexe A).

Comme on le constate dans la Figure I.1.2, dès la préproduction le *workflow* suit une structure en cascade (aussi appelé *waterfall*). Autrement dit, les étapes sont interdépendantes dans leur enchaînement. En planifiant la fabrication du film de cette

14. Le scénarimage ou *storyboard* est une suite de dessins correspondant chacun à un plan et permettant (lors de la préparation d'un film) de visualiser le découpage. Définition par Larousse

15. Séquence est un terme technique employé au cinéma et à la télévision pour désigner un ensemble de plans se déroulant dans un même lieu et dans un même temps. En animation les séquences sont définies à l'écriture et au *storyboard*. Elles sont mises en scène en 3D par les équipes de *layout*. Les artistes *layout* travaillent majoritairement 'à la séquence' ce qui leur permet de travailler plus facilement leurs raccords caméra entre les différents plans et s'abstenir de tâches redondantes comme reconstituer ou importer les décors et personnages. Définition par LePipeline.org.

16. Cette étape consiste à étudier chaque plan ou séquence et à définir les éléments qui y figurent, comme le nombre de personnages ou de props. Chaque élément sera défini en tant qu'*asset* de production et pourra ainsi être budgétisé. C'est souvent ce qui va déterminer la difficulté des plans, le volume des éléments à fabriquer va permettre de donner un ordre de grandeur de la taille des équipes et du niveau de séniorité requis. Définition par LePipeline.org.

façon, les équipes de production bénéficient d'un calendrier complet à avancer au client. Cependant, malgré les marges de temps prises sur chacune des étapes de fabrication, il est très compliqué d'anticiper tous les problèmes qui pourraient survenir durant la production. Or, avec le modèle en cascade, toutes occurrences de retard ou changement impactent la suite de la chaîne de fabrication. On se retrouve alors à décaler toutes les étapes suivantes. Il serait cependant difficile de concevoir une structure autre qu'en cascade durant la phase de préproduction, car paralléliser les étapes entrainerait plus d'erreurs et donc plus de retard. Seule l'étape de design peut se faire en même temps que les autres, ce qui est déjà plus ou moins le cas dans les studios.

Pour la fabrication de série, on observe sur l'Annexe B que la création des épisodes suit le même schéma. Cela permet aux différentes équipes artistiques d'enchaîner les épisodes de la même saison. C'est une cascade à deux niveaux de détails, à l'échelle d'un épisode et entre les épisodes. Par conséquent, les épisodes s'impactent directement, un retard au *layout* de l'épisode 1 risque d'affecter la fabrication de l'épisode 2.

I.1.1.2 Production : fabrication, assemblage

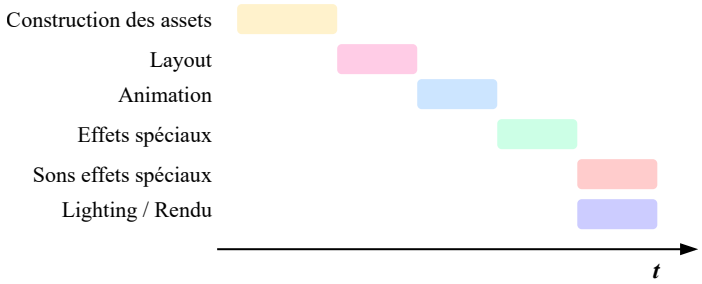


FIGURE I.1.3 – Étapes de production en fonction du temps

Une fois les bases établies par la préproduction, le moment est venu de passer en production. C'est la phase la plus longue durant laquelle les équipes auront le plus d'effectifs. Au début de cette phase, le pipeline se doit d'être prêt pour garantir l'efficacité et la bonne marche de la chaîne. La production est au film ce qu'est le gros œuvre à la construction d'une maison.

L'œuvre prendra corps tout au long des différentes étapes de cette phase. D'un point de vue macroscopique, la production d'un film peut s'apparenter à la production d'une voiture. Au cœur de la chaîne de fabrication, les équipes artistiques procèdent dans un premier temps à la création des *assets* puis à leur assemblage dans des plans et des séquences. Notons que selon les studios, la construction des *assets* peut s'étendre en parallèle du *layout*.

Dès lors qu'une première version des *assets* est disponible, d'autres équipes de graphistes débutent l'assemblage des plans lors du *layout*. Le *layout* consiste à positionner les *assets* dans les scènes 3D, à cadrer les plans et établir les mouvements de caméra en suivant le *storyboard* et l'animatique. Dès lors que les plans sont assemblés et validés, que les *assets* sont préparés, les plans sont prêts à être animés. Durant cette phase, les équipes d'animation donnent vie à certains *assets* comme les personnages en leur enregistrant des poses clefs en fonction du temps. Les équipes d'effets spéciaux (aussi dénommé FX) ajoutent ensuite différents effets simulés pour animer tout type de matière (exemples : simulation d'eau, de fumée, fractures d'objets).

Comme illustré sur la Figure I.1.3, l'enregistrement des effets sonores se développe après les effets spéciaux.

Une fois enrichis de l'animation et des FX, les plans sont mis en lumière et rendus lors de la phase de *lighting*[17]. Notons que sur un projet de série, les ambiances de lumière sont arrêtées dès la construction des sets de décor.

Durant cette phase, l'équipe technique se focalise généralement sur la stabilisation du pipeline. Mais il arrive fréquemment que de nouvelles fonctionnalités soient développées durant cette période pour répondre à des besoins non anticipés. Comme l'artiste écrivaine Renee Dunlop le relève dans son livre, dans un pipeline « la seule chose qui est constante : c'est le changement »[18].

Tout au long de la fabrication, l'équipe de production doit « s'assurer que la barque ne tangue pas trop, écoper et être réactif. » (Voir Annexe A) Mais elle assure également la validation du travail (décrit en Sous-section I.1.2.3) avec les acteurs extérieurs.

I.1.1.3 Postproduction : affinage, export

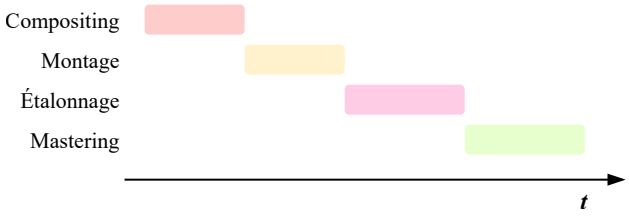


FIGURE I.1.4 – Étapes de postproduction en fonction du temps.

Au sortir de la production, toutes les pièces sont usinées : chaque plan est créé, rendu, transformé en information 2D. La postproduction va venir polir le résultat reçu de la phase précédente et l'exporter. C'est la dernière occasion d'apporter des corrections au produit avant la livraison au client.

La Figure I.1.4 fait état des différentes étapes artistiques qui composent la postproduction. Les sorties d'images récupérées en fin de production sont les rendus multicouches (passes de couleur, d'ombre, de profondeur, etc.) des différents plans pour faciliter la postproduction. Le *compositing*[19] a pour but de traiter et d'améliorer ces séquences brutes sorties du rendu. Les plans sont ensuite assemblés entre eux avec le son lors du montage[20]. Après cette étape, leur colorimétrie est homogénéisée par les équipes d'étalonnage (*grading*). Le film sera enfin adapté aux formats physiques de distribution lors du *mastering*.

Lorsque vient la postproduction, les équipes techniques commencent à faire une analyse rétrospective du déroulement de la production et anticipent la préparation des outils pour la prochaine production. C'est également un temps pour mener une veille technologique afin d'évaluer si de nouvelles technologies sont intéressantes à ajouter au pipeline de fabrication.

17. Le *lighting* est la tâche de mise en lumière de la scène.
18. *« The only thing that is constant is change. »* [2, p. 29]
19. L'étape de *compositing* consiste à assembler différentes couches de rendu pour créer l'image finale.
20. En série d'animation, l'étape de montage est généralement faite dès de l'animatique. Verrouiller toute la fabrication sur ce montage évite d'animer pour rien.

Pour l'équipe de production, les principaux enjeux seront d'assurer la livraison du produit fini et de vérifier que rien n'a été oublié en route.

Dans cette section nous avons donné un aperçu macroscopique du déroulement de la fabrication d'un film d'animation. Nous avons mis en lumière les enjeux des trois grandes familles d'équipes œuvrant à la création du film. Nous avons constaté que le modèle actuel de production reposait en grande partie sur une structure en cascade.

I.1.2 Segmentation du travail

Dans la section précédente, nous avons découvert les grands enjeux des trois équipes responsables de la création d'un épisode de série en image de synthèse, au cours du temps. En réalité, ces équipes ou familles sont composées d'une multitude de corps de métiers formant le cœur de cette chaîne de fabrication hybridant l'humain et le logiciel. Le facteur humain, situé au cœur de la création fera l'objet de cette section. Nous verrons la diversité des rôles et métiers représentant les maillons de la chaîne. Une étude de l'organisation humaine mettra ensuite en lumière la relation verticale qui connecte les acteurs de la chaîne de fabrication. Enfin, nous nous attarderons sur les processus qui font circuler l'information à travers ce maillage humain que constitue la chaîne de production.

I.1.2.1 Une grande variété de corps de métiers

Comme survolé en Section I.1.1, produire un film d'animation requiert énormément d'étapes différentes. Chacune de ces étapes est elle-même constituée de nombreux aspects. Si l'on prend l'exemple de la construction d'un *asset* (première étape de la Figure I.1.3) de voiture, il faudra au minimum trois corps de métiers différents pour lui donner vie. Un graphiste commencera par la modéliser, c'est-à-dire créer sa forme géométrique en trois dimensions. Viendra ensuite la définition de l'aspect de la matière de sa surface (carrosserie métallique), ainsi que les détails (couleurs, reliefs, égratignures, etc.), on parle alors de *shading*[21] et *texturing*[22]. Puis, si l'*asset* ne contient pas de vêtements ou de poils, on peut considérer que son aspect visuel est terminé. En revanche, si l'objet est destiné à être animé, un squelette d'animation (ou *rig*[23]) est requis pour l'articuler. La création du *rig* (*rigging*) représente souvent le dernier aspect de la construction d'un *asset*, mais selon les studios, il peut se tenir avant le *shading* et *texturing*.

Avec la multiplication des logiciels de création spécialisés, chaque aspect de la création des *assets* tend à devenir un métier à part entière. Cette individualisation des tâches dépend également de l'échelle des productions. Dans les petits studios (d'une dizaine de personnes) on retrouvera des profils plus généralistes qui seront qualifiés sur plusieurs aspects de la création (tels la modélisation, le *surfacing* et le *rig*). Dans les gros studios, les postes sont plus spécialisés, on trouvera par exemple des artistes spécialisés dans la création des poils et fourrures (aussi dénommés *grooming artists*).

Le Tableau I.1.1 énumère quelques métiers qui composent les familles énoncées dans la Section I.1.1, il donne ainsi un bref aperçu des différents acteurs qui composent la chaîne de fabrication. Mais comment ces divers métiers communiquent-ils ? Comment la collaboration se structure-t-elle entre eux ?

I.1.2.2 Départements

À partir d'une certaine échelle, les équipes sont segmentées en départements. Cette organisation structurelle, liée à l'individualisation du travail, permet de paralléliser les

21. Le *shading* est la tâche qui consiste à définir les *shaders* d'un *asset*.

22. Le *texturing* est la tâche qui consiste à générer les textures utilisées par le *shading* d'un objet 3D. Définition par LePipeline.org

23. Le *rig* désigne l'ensemble des *bones* qui permet de déplacer un modèle. Cela inclut les contraintes, les contrôleurs, voir même les *pickers*(interfaces de sélection des *bones*).

Famille	Métier	Mission
Technique	Infographiste développeur (TD)	Conçoit des modules complémentaires aux logiciels de création et de production d'images utilisées dans le cadre de la production.
	Ingénieur.e R&D	Conçoit des logiciels de création adaptés aux besoins de la production.
	I.T.	Gère l'infrastructure logicielle et matérielle du parc informatique
Production	Directeur.rice de production	Gestion de l'aspect administratif, juridique et budgétaire du film.
	Chargé.e de production	Gestion du projet, de l'assignation des tâches et du recrutement.
	Assistant.e de production	Suivis de l'aspect administratif, juridique et budgétaire du film.
Artistique	Scénariste	Écriture du scénario et du script.
	Réalisateur	Garant de la cohérence narrative et graphique du projet.
	Storyboarder	Responsable du développement du *storyboard*.
	Infographiste modélisation	Création de la forme 3D des assets.
	Technicien.ne setuper 3D	Création de *rig* d'*asset*.
	Infographiste *surfacing*	Création de textures et de *shaders* pour les *assets*.
	Layout man / woman	Dispose les assets et les caméras dans les scènes 3D
	Animat.eur.rice 2D 3D	Mise en mouvement des assets dans la scène.
	Infographiste effets spéciaux	Création d'effets visuels.
	Infographiste rendu et éclairage	Mise en lumière des scènes 3D.
	Technicien.ne compositing	Intégration des effets visuels
	Infographiste monteur	Montage des plans.

TABLE I.1.1 – Différents métiers d'une production de série d'animation à Cube Creative

tâches d'une étape de production à travers plusieurs artistes et ainsi de réduire les temps de production.

Les départements sont constitués d'un superviseur et de graphistes spécialisés dans une étape de la production (par exemple, au sein du département *layout* on retrouvera des *layout* man/woman et un superviseur *layout*). La structure hiérarchique d'un département varie selon sa taille.

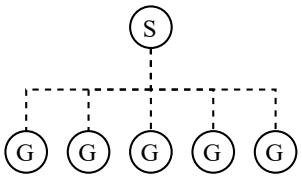


FIGURE I.1.5 – Structure hiérarchique d'un petit département.

Dans le cadre de petits effectifs (voir la Figure I.1.5) un superviseur (S) suffira pour gérer une dizaine de graphistes (G). Le rôle de superviseur peut correspondre à différents échelons selon la configuration du studio, il peut y avoir un superviseur par département et/ou un superviseur général au projet (ou *CG Supervisor*).

En plus de bénéficier d'un niveau élevé de compétence dans la spécialisation du département et de chapeauter ses équipes, le superviseur a plusieurs responsabilités. Il peut assister et conseiller le réalisateur sur la faisabilité de ses demandes en évaluant

les scripts. Pour ce faire, il met en place des solutions techniques et artistiques adaptées pour répondre au mieux aux problématiques narratives et budgétaires. Il est aussi en relation étroite avec le responsable de production sur les processus de recrutements et peut avoir à gérer la formation d'un nouvel arrivant.

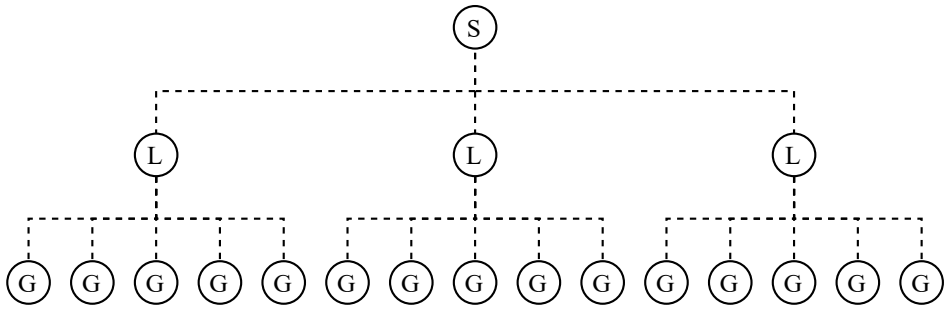


FIGURE I.1.6 – Structure hiérarchique d'un grand département.

Pour des départements aux effectifs élevés (voir la Figure I.1.6), la tendance sera de subdiviser les effectifs en équipes dirigées par des leads (représentés par un L). Ces derniers rendront leurs comptes au superviseur du département. Le lead fait tampon entre le superviseur et l'équipe de graphistes en répartissant le travail parmi ces derniers. En canalisant les demandes des graphistes, il agit comme un ambassadeur auprès du superviseur.

À Cube Creative, nous avons un *CG Supervisor* par projet ainsi qu'un superviseur d'animation. Le *CG Supervisor* va porter et orienter les choix des méthodes de création tout au long du projet. Il est présent sur le projet dès la préproduction pour échanger avec les réalisateurs, directeurs artistiques et chargés de production afin de planifier les besoins du projet. En fonction de ces éléments, il décidera de la méthode de fabrication des images. À ce moment-là, si le calendrier le permet, il développera les premiers *assets* et les premières images de la série pour valider les méthodes de fabrication, de concert avec l'équipe technique. Tout au long de cette élaboration, il fait le lien avec l'équipe de R&D pour lui faire part des besoins du projet en matière d'outils et de développement pipeline (en collaboration étroite avec les autres superviseurs du studio). Toujours en préproduction, il valide techniquement les étapes de développement pour anticiper les problèmes survenant en production. Ainsi, il peut agir dès l'écriture du pitch [24] lorsque certains aspects sont alarmants d'un point de vue technique. Il donne également des recommandations techniques lors de la validation des scripts et de l'animatique 2D. Durant la production, il effectue le suivi des équipes et fait du soutien technique (l'expertise sur les logiciels utilisés est très importante). Le *CG Supervisor* est également en contact avec les entreprises externes (si il y en a) sur les points techniques. Par exemple sur la série *Tangranimo*, où l'animation s'est en partie déroulée en Inde, Tanguy Weyland alors *CG Supervisor* de la série, les assista dans le déploiement du pipeline.

Les départements peuvent être comparés aux maillons de la chaîne de fabrication. Au sein de la chaîne d'assemblage, on peut les représenter par des entités opaques et interdépendantes coordonnées par l'équipe de production et le superviseur. Comme

24. Le pitch, c'est l'histoire formulée en une seule phrase. C'est le lien le plus simple entre le personnage et l'intrigue, qui est généralement constitué d'un évènement déclenchant l'action, d'une indication sur le personnage principal et d'une indication sur le dénouement de l'histoire. John Truby dans l'*L'anatomie du scénario*

illustré dans la Figure I.1.7, chacun d'entre eux récupère des éléments en entrée pour les traiter, transformer ou assembler afin d'en générer de nouveaux en sortie.

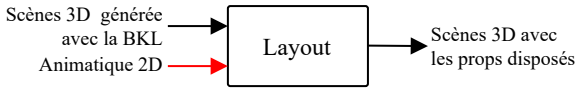


FIGURE I.1.7 – Exemple des flux d'entrée et de sortie du département *layout*.

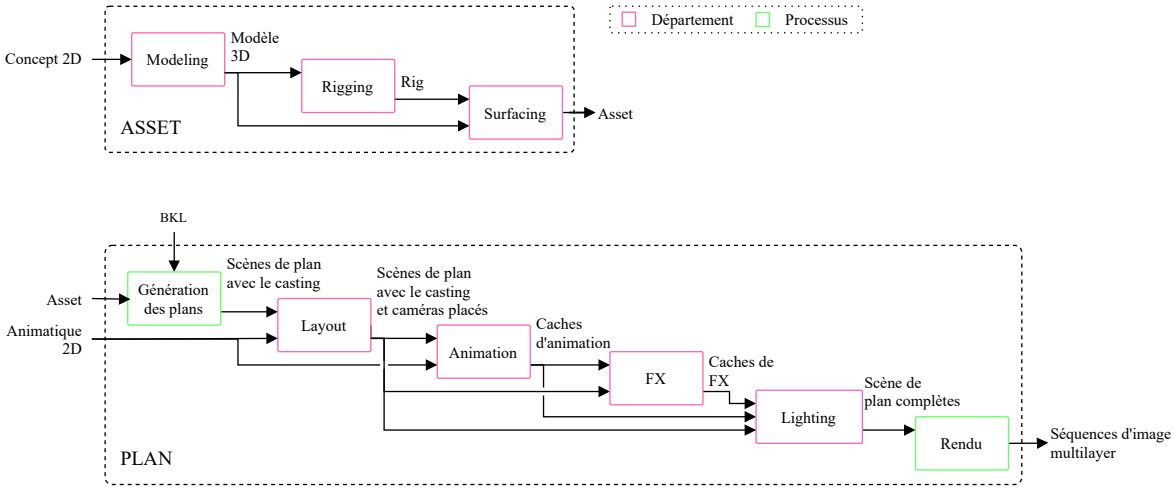


FIGURE I.1.8 – Diagramme d'analyse fonctionnelle descendante d'une production d'un film d'animation classique.

Département	Entrées	Sorties
Storyboard	Script	Planches 2D de *storyboard*
Modélisation	Concept 2D	Modèle 3D
Surfacing	Modèle 3D	Modèle 3D + UV + Matériaux + Textures
Rigging	Modèle 3D + Matériaux + Textures	Modèle 3D + Matériaux + Textures + *Rig*
Layout	Animatique 2D & Scène 3D autogénérée à partir de la BKL	Scènes 3D incluant tout les assets du casting et les caméras placées
Animation	Scènes 3D incluant assets et caméras placés	Scène 3D animée
FX	Scène 3D avec les caches d'animation	Caches de FX
Lighting	Scènes 3D avec cache de FX et d'animation	Scènes 3D complètes avec lumières et ciel
Rendu	Scènes 3D complètes	Séquences d'images *multi-layer*

TABLE I.1.2 – Entrées et sorties des différents départements que l'on retrouve au sein de la phase de production d'une série.

Les flux de données connectent chaque département de la chaîne. Ainsi, comme vu précédemment, le *layout* prendra en entrée une scène autogénérée composée du *casting*[25] entré par les équipes de production dans la Break Down List (BDL)[26] et

25. Le *casting* décrit tout ce qui va jouer dans un plan (personnage, décor, etc.). Il résulte généralement du dépouillement du scénario, d'un *storyboard* ou d'une séquence. C'est à partir du *casting* que des outils de pipeline vont pouvoir initialiser ou mettre à jour des scènes pour que les graphistes puissent se focaliser sur leur travail sans se préoccuper d'assembler leur scène. Fondamentalement, c'est une simple liste de tous ces éléments, mais on y adjoindra assez rapidement des informations supplémentaires pour spécifier les versions de rig utilisées, le nombre d'éléments castés, le nom des fichiers correspondants, etc. Définition par LePipeline.org.

26. La Breakdown List est une liste des assets présents par plan par épisode.

l'animatique 2D. Dans la production de série d'animation, les studios ont tendance à privilégier le *layout* à la séquence pour travailler plus facilement les raccords caméra et gagner en efficacité (car les *castings* changent rarement au cours d'une séquence). Lorsque cette séquence sortira du département *layout*, elle sera automatiquement découpée en plans pour pouvoir être ingérée dans le pipeline puis passée au département d'animation. Nous détaillerons l'ensemble de ces processus automatisés en Section I.1.3.

Les départements sont donc des entités séparées, qui forment les maillons de la chaîne de fabrication. Mais comment fonctionnent-ils ? Comment communiquent-ils entre eux et comment collaborent les acteurs qui y coexistent ?

I.1.2.3 Chaîne de communication

Cette section détaillera les deux mécanismes de communication qui forment le cœur de la collaboration au sein d'une production de films d'animation : l'**assignation** et la **validation**. Nous explorerons comment ces deux phases interviennent tout au long de la chaîne de fabrication au sein des départements pour mettre en mouvement le flux de travail.

Durant la fabrication d'un épisode de série ou d'un film, tout le travail à effectuer est divisé en une multitude de tâches [27]. En fonction de l'étape de fabrication, cette entité peut être une séquence, un plan ou encore un *asset* (exemple de tâche : faire le *layout* de la séquence 3 sur l'épisode 4). Les tâches décrivent tout le travail artistique à faire sur le projet et permettent de suivre son état de progression. Leur cycle de vie est rythmé par les mécanismes d'assignation et validation.

Le cycle de vie d'une tâche (voir la Figure I.1.9) débute lors de sa création par l'équipe de production. À sa naissance, une tâche porte généralement le statut TODO [28] et n'a pas d'artiste attitré, cela se produira lors de l'assignation. Généralement, les chargés de production contactent des équipes de création et leur donnent un certain nombre de tâches à réaliser dans un laps de temps donné. Pour de la série, on parle de quotas à respecter. Sur le *layout* de *MushMush* par exemple, les artistes avaient un quota de dix plans par jour, sur *Kaeloo* les animateurs devaient finir six secondes d'animation par jour.

Une fois le travail transmis à l'équipe, le lead ou superviseur assignera les tâches aux membres de son équipe. Selon le studio, ce processus peut être plus ou moins collégial. À Cube Creative, cette étape se produit lors d'un *brief* [29] durant lequel les artistes et le lead discutent de la répartition ouvertement. Ce fonctionnement permet aux artistes de se faire entendre et leur donne de l'initiative. La répartition se fait généralement selon les niveaux d'expérience des artistes allant de junior à senior en passant par mid. Les tâches sont assignées à l'issue de ce *brief* qui se produit une fois par étape de fabrication d'épisode (par exemple, un *brief layout* par épisode).

Le travail de l'artiste débute lorsqu'il passe une tâche qui lui est assignée en Work In Progress (WIP). En fonction de l'étape de fabrication, la durée d'une tâche varie de quelques heures à plusieurs jours. Lorsque le travail est terminé, l'artiste passera

27. Généralement, une tâche (ou *task*) est un travail à faire dans un temps déterminé et à certaines conditions. Définition par Larousse. Plus spécifiquement au cinéma d'animation, la tâche de travail est une étape individuelle qui s'intègre dans le processus de fabrication d'une entité. Définition par LePipeline.org.

28. Selon les studios et les productions, on peut aussi retrouver les status initiaux HOLD (on attend), NR (pas prêt) ou encore N/A (pas nécessaire)

29. Le *brief* est une réunion d'information pour définir les objectifs, les méthodes, les moyens, etc.

le statut de la tâche en Wait For Approval (WFA) (aussi appelé TOCHECK). Ce qui marque le début de validation du travail sous forme de *review*[30].

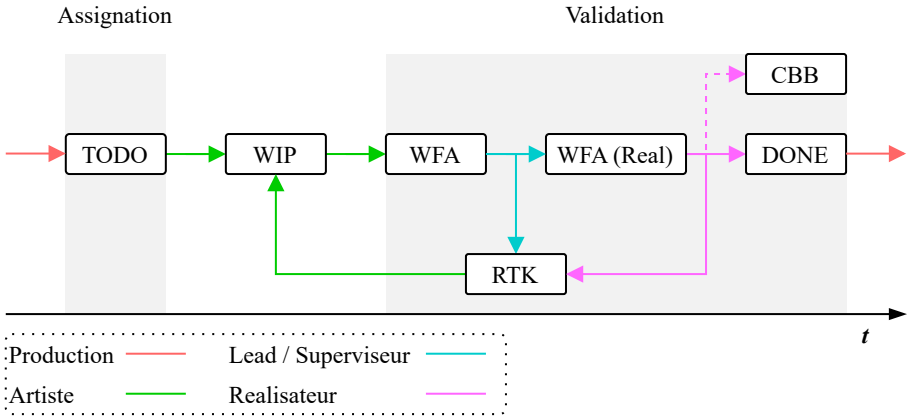


FIGURE I.1.9 – Cycle de vie d'une tâche à travers son statut.

La validation consiste à passer le travail en revue par les personnes responsables (lead, superviseur, réalisateur ou encore production). Selon l'échelle des équipes et la nature du travail, la validation peut s'effectuer à un, deux ou trois niveaux. On notera qu'il est nécessaire d'avoir un statut différent par niveaux (WFA et WFA(Real) sur la Figure I.1.9) afin de savoir où en est la validation. Au terme de ce processus, le responsable peut valider le travail et passer le statut de la tâche en DONE ou équivalent pour son niveau (WFA(Real) pour la validation du Lead dans la Figure I.1.9) ou encore demander des modifications et passer la tâche en Retake (RTK). Dès lors qu'une reprise (Retake) est demandée, l'artiste repassera la tâche en WIP pour effectuer les modifications nécessaires puis en WFA lorsque la modification peut être évaluée. Le processus de validation est donc itératif par nature. Cependant, le nombre de *retake*[31] est strictement réglementé dans le contrat de production pour éviter les débordements de budget et de temps. Dans certains types de productions tels que les longs ou courts métrages, un troisième type de statut peut être assigné au terme d'une *review* : Could Be Better (CBB)[32]. Ce statut précise que le travail accompli remplit les critères de revue, mais pourra être amélioré s'il reste du temps au terme de l'étape de fabrication. Il est rare de retrouver ce statut dans le cadre d'une série car le budget et les plannings sont serrés.

Comme constaté dans le Sous-section I.1.2.2, la structure hiérarchique au sein d'un département est verticale. Par conséquent, les processus d'assignation et de validation qui rythment le cycle de vie du travail partagent la même verticalité. On peut ainsi décrire l'assignation comme une relation hiérarchiquement descendante (le travail est assigné au niveau hiérarchique inférieur) et la validation est à la fois montante (WFA) et descendante (RTK)). Cette organisation de la collaboration est actuellement la norme au sein des studios.

Si cette structuration du flux de travail a l'avantage de s'adapter aisément à de grandes productions, elle n'en présente pas moins quelques inconvénients. Tout d'abord, nous avons vu qu'une tâche était associée à un travail spécifique d'une entité. Ce faisant,

30. *Review* est le mot utilisé pour définir l'étape de validation d'une tâche. Le travail est passé en revue par les personnes concernées telles que les superviseurs, la production et le réalisateur. C'est au terme de cette étape qu'il est décidé si la tâche est valide et peut être considérée comme terminée. Définition par LePipeline.org.

31. Une *retake* est une correction demandée à l'issus d'une *review*.

32. Could Be Better est un statut désignant un travail validé mais qui pourrait être amélioré.

elle est limitée au département, créant ainsi une première barrière de communication entre les artistes amenés à travailler sur la même entité au sein de départements différents. Un artiste chargé du *surfacing* sur un *asset* n'aura que le résultat du travail des artistes précédents. Il ne connaitra pas le cheminement de la construction (en dehors du suivi des étapes de fabrication dans l'*asset manager*) par les différents artistes qui ont travaillé sur le même *asset*. La structure en départements représente ainsi un silo de communication. L'organisation verticale de la chaîne de communication induit un ruissellement de l'information à travers les strates hiérarchiques. Chaque niveau agit comme un filtre qui ne laisse passer que l'information utile au niveau inférieur ou supérieur. Finalement, les artistes d'un département sont eux-mêmes cloisonnés dans des silos individuels.

Dans cette section, nous avons décrit la structure humaine de la chaîne de fabrication, ses forces et ses faiblesses. Cette structure sur laquelle se repose le *workflow* représente l'une des deux clefs garantissant la collaboration tout au long de la création du film. Mais ce modèle de production ne saurait exister sans la structure logicielle sous-jacente traitant les énormes flux de données et automatisant les traitements sur ces dernières. Cet organe vital que représente l'ensemble de la brique logicielle derrière la chaîne de fabrication fait l'objet de la prochaine section.

I.1.3 Au cœur de la chaîne de fabrication : le pipeline

Dans les sections précédentes, nous avons décrit le *workflow* classique de production d'un épisode de série d'animation (cf. Section I.1.1) ainsi que la structure humaine sous-jacente (voir la Section I.1.2). Mais il est impossible de capturer la fabrication des films en images de synthèse dans sa globalité sans passer par la structure logicielle supportant la chaîne de fabrication : le pipeline.

De par la diversité de ses formes, on lui prête de nombreuses définitions. Renee Dunlop le définit comme « la colle qui unifie le travail de tous les artistes impliqués dans une production » [2]. La colle représente ici les processus logiciels qui structurent la collaboration entre les artistes.

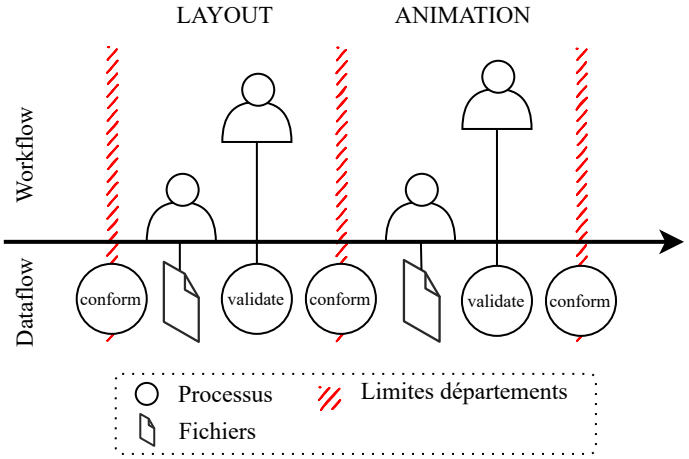


FIGURE I.1.10 – Relation en dataflow et workflow.

John Carey sur le *google-group Global VFX pipeline* situe la notion primordiale de flux de données au sein du pipeline (que nous dénommerons *dataflow*[33] par la suite) : « Un pipeline est un moyen formalisé de transformer des données d'un état à un autre à travers une série de processus de gestion. » En effet, *workflow* et *dataflow* sont intimement liés à travers une relation de co-dépendance : un *dataflow* supporte un *workflow* donné. Or, comme nous le constaterons tout au long de cette section, le pipeline sculpte le *dataflow* à travers un ensemble de logiciels et processus, par conséquent il modélise le workflow sous-jacent. La Figure I.1.10 illustre cette relation presque fusionnelle. On peut noter qu'une personne se trouve derrière chaque étape du dataflow, que ce soit pour déclencher un processus où le nourrir en données. Comme le décrit Flavio Perez, « c'est un ensemble de procédures très organiques qui n'a d'existence que par les artistes et techniciens qui le nourrissent en données que d'autres graphistes transforment » [19].

Pour appréhender la place et la nature des pipelines au sein de la collaboration dans la production, nous débuterons par l'étude des besoins et contraintes à leur origine. Ainsi, nous constaterons qu'à l'image d'un organisme vivant s'adaptant aux contraintes de son environnement, le pipeline peut prendre des formes très différentes selon les particularités du projet. Nous décrirons ensuite les principaux organes qui le composent

33. Le *dataflow* (« flux de données », en français) est l'organisation des données dans une structure de fichiers. Contrairement au *workflow* qui décrit des étapes de travail, le *dataflow* décrit les dépendances de fichiers induites par le pipeline. Définition par LePipeline.org.

en faisant écho au workflow décrit dans les sections précédentes tout en les illustrant de cas concrets issus du pipeline en place à Cube Creative.

I.1.3.1 À l'origine, un ensemble de besoins et de contraintes

Le pipeline est co-conçu en amont par l'équipe de développement, les superviseurs et l'Information technology Technician (IT) [34]. L'enjeu est de taille : concevoir et développer un ensemble d'outils adaptés aux besoins humains, graphiques, techniques du film, mais également à ses contraintes budgétaires. Nous ferons ici état des différents facteurs décisionnels conduisant au choix des logiciels de création qui porteront la réalisation des étapes de production.

Au commencement, les tests techniques menés par les superviseurs détermineront les logiciels de création (que nous abrégerons Digital Content Creation software (DCC)) adaptés aux problématiques graphiques (ou esthétiques) du projet. Lorsque les solutions logicielles choisies nécessitent des licences d'utilisation, c'est-à-dire des coûts supplémentaires, les contraintes budgétaires de la production entrent en jeu et peuvent nécessiter une réorientation des choix. Le besoin de réalisme dans le rendu des animaux pour la série *Athléticus* [33], par exemple, a nécessité une phase de veille de la part des superviseurs. Cette phase requiert une bonne connaissance des logiciels de création disponibles sur le marché ainsi qu'une vision claire des technologies déjà en place au sein de l'entreprise. Dans le cas des animaux réalistes, l'enjeu fût de trouver une solution pour gérer le *grooming* [35] dans *Blender* [36], l'un des principaux DCC déjà en place dans la chaîne de fabrication à Cube. On note ici l'importance de l'existant au sein de ces décisions. Dans notre cas, un *add-on* [37] *Blender* payant répondant au besoin existait déjà sur le marché. Une fois cette solution sélectionnée, l'équipe de recherche et développement (R&D) entre en jeu pour intégrer cette brique logicielle au pipeline existant.

Les besoins techniques, logiciels et matériels, sont directement formulés par les spécificités structurelles du projet. Est-ce que la production est divisée sur plusieurs sites ? Les graphistes sont-ils en télétravail ? Toutes ces questions auront un impact important sur les choix d'infrastructure réseau, logicielle et matérielle. La question du multi-sites s'est notamment posée sur la série *Pfffirates* éclatée sur trois sites différents, dont un à l'étranger. La création des *assets* se tenait à Levallois, le *layout* dans les locaux de Xilam à Lyon, l'animation en Inde et à Lyon et enfin le rendu et *compositing* se firent sur Paris et Lyon. Ces contraintes très fortes ont nécessité le développement d'outils et de processus qui ont permis de faire collaborer les trois sites. Nous détaillerons certains de ces outils dans la Sous-section I.1.3.2. Le cas de la crise sanitaire Covid-19 est un exemple marquant de contrainte technique avec l'arrivée massive et brutale du télétravail. Sur un laps de temps très court (un mois environ), les équipes techniques des studios d'animation ont adapté leur pipeline afin de permettre aux équipes de télétravailler avec une perte minimale d'efficacité. La complexité d'adaptation dépendait de l'échelle des studios et des normes de sécurité en place. À Cube Creative par exemple,

34. Personne responsable du parc machine, de la gestion du réseau et du stockage.

35. Le *grooming* (ou toilettage en français) est la tâche qui vise à placer et coiffer des poils et cheveux d'un personnage ou d'une créature. Le *grooming* peut aussi s'appliquer sur d'autres objets comme les arbres (feuilles) ou des sols (fleurs, herbe). Le *grooming* est aussi appelé *hair* dans certains studios. Définition par LePipeline.org

36. *Blender* est un logiciel de création 3D open source. `https://www.blender.org/`

37. Un add-on ou un plug-in est un programme ou un script tiers qui est ajouté à un programme pour lui donner des fonctionnalités et des capacités supplémentaires.

deux solutions ont été nécessaires :

— le travail sur bureau à distance : les graphistes, en télétravail, accèdent à leur poste situé dans les locaux de Cube avec une solution de contrôle à distance ;

— le travail « local » : les graphistes travaillent sur des fichiers locaux à leurs machines.

La majorité des équipes bénéficiaient du bureau à distance, plus simple puisqu'il ne nécessite pas de modification du pipeline, les artistes contrôlant leurs postes locaux à Cube. On notera que c'est également sécurisé du fait que les fichiers ne quittent pas les machines du site de production. Initialement, les équipes ont privilégié la solution Teamviewer [38] pour sa fiabilité, mais son coût augmente rapidement avec le nombre d'utilisateurs. Par conséquent, nous sommes passés à son concurrent gratuit Anydesk [39]. Ici encore, on constate l'impact des contraintes budgétaires qui orientent la décision finale. Cependant, pour certaines étapes de fabrication précises, la solution de bureau à distance s'est avérée insuffisante. La latence induite par la connexion réseau n'était pas acceptable pour les équipes d'animateurs qui ont besoin de performances proches du temps réel pour visualiser les animations avec fluidité. Le flux vidéo compressé de ces solutions peut également poser problème aux équipes de *shading* et de *compositing* qui ont besoin de couleurs calibrées. Finalement, ces limitations induites par le bureau à distance ont conduit à la création d'un outil étendant le pipeline chez les télétravailleurs. Un logiciel d'automatisation synchronisant une partie de l'arborescence de fichiers du serveur de stockage de Cube avec celle de la machine locale des télétravailleurs. Les animateurs pouvaient ainsi accéder à leurs fichiers de travail localement et bénéficier de performances viewport meilleures qu'en bureau à distance. En revanche, cette architecture nécessite de sortir les fichiers du site de production, ce qui pose des problèmes de sécurité.

Au gré des productions qui s'enchaînent, de sa croissance, un studio peut capitaliser des briques technologiques ré-exploitables d'une production à une autre. Lors de la mise en place d'une nouvelle production, ces solutions existantes pèseront lourd dans le choix du pipeline. Cela s'accompagne souvent d'une dette technologique [40], une contrainte technique non négligeable. L'utilisation d'Autodesk *3Ds Max* [41] à Cube en est un bon exemple. Historiquement, *3Ds Max* a été le principal DCC 3D utilisé au sein des productions dans les débuts de la société. Au gré des projets, les équipes techniques ont développé un ensemble d'outils pipeline prévus pour ce dernier dans un langage de script qui lui est propre : le MaxScript [42]. Le temps passant, *3Ds Max* est devenu de moins en moins mis à jour et *Blender* commença à devenir une alternative gratuite et open source [43] sérieusement viable. Et finalement, le jour où fut prise la décision de remplacer Max par *Blender* en production, il a été nécessaire de réécrire tous les outils conçus en MaxScript en Python [44] pour *Blender*. Ce travail qui peut s'apparenter à

38. `https://www.teamviewer.com`

39. `https://anydesk.com`

40. On parle de "dette technologique" lorsque des choix « anciens » effectués généralement dans le but de « se simplifier la vie » se retournent contre les concepteurs et empêchent d'apporter des évolutions pourtant prévisibles au projet. Lorsque cela se produit, il faut recoder ou jeter l'existant pour avancer.

41. `https://www.autodesk.com/products/3ds-max`

42. Le MaxScript est un langage de script propriétaire à *3Ds Max*. Documentation du MaxScript :`http://docs.autodesk.com/3DSMAX/14/ENU/MAXScript`

43. Un logiciel open source est un programme dont les sources sont librement accessibles pour être éventuellement étudiées, utilisées, modifiées et redistribuées.

44. `https://www.python.org`

une mue du pipeline, n'est ni plus ni moins que l'absorption de la dette technologique. Dans le cas du passage à *Blender*, il a fallu approximativement trois membres de l'équipe R&D à temps plein pendant dix mois pour développer une première itération du pipeline sur *Blender*[45].

Au sein d'un studio créé depuis plusieurs années, l'existant, c'est-à-dire l'ensemble des entités logicielles, matérielles et humaines déjà en place, pèsera lourd dans le choix du pipeline.

Les besoins humains sont formulés par l'équipe de production lors des phases de recrutements. Ils représentent le nombre de personnes requises ainsi que leurs compétences nécessaires pour la réalisation d'une étape de production. Ce besoin influencera le choix des DCC. Il doit y avoir assez de personnes formées à *Blender* sur le marché du travail pour l'utiliser sur l'étape donnée d'une production sans quoi les effectifs seront insuffisants. Pour pallier ce souci lors des débuts de l'utilisation de *Blender* à Cube, les équipes de production ont dû prévoir un temps de formation au logiciel au sein de la mission des artistes.

Les besoins et contraintes représentent un grand nombre de variables dans l'équation du choix des outils et technologies d'un pipeline. En fonction du projet et du studio, ces variables peuvent avoir de grandes amplitudes, et par conséquent les différences entre les pipelines aussi. Il peut aller d'un simple outil de nomenclature pour un court métrage à une suite complexe d'outils d'*asset* management pour une série d'animation.

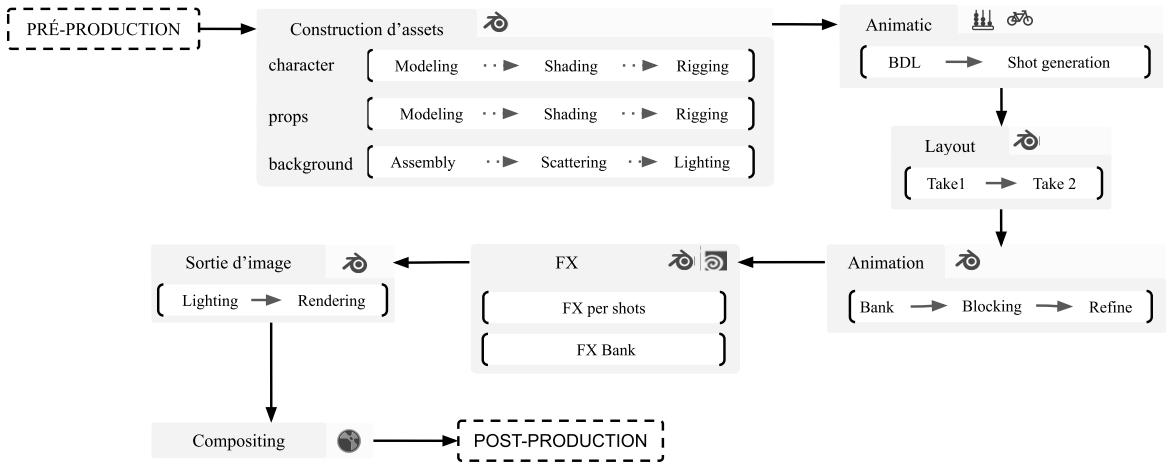


FIGURE I.1.11 – Dataflow d'une production de série à Cube

I.1.3.2 Au cœur de la collaboration : des fichiers

S'il existe une notion commune à tout pipeline existant, c'est celle du *dataflow*. Pour supporter le *workflow* étudié en Section I.1.1, ce sont des milliers de fichiers qui

45. Plus d'information sur cette transition lors de l'intervention *Preparing a Blender pipeline for 52x11' TV series* de Valentin Moriceau et Tanguy Weyland à la *Blender Conference 2018* : `https://youtu.be/4R8PduY_vV4`

sont générés, modifiés, échangés. Ces données échangées représentent le support de vie de la collaboration au sein de la chaîne de fabrication. On peut les assimiler à ce que représente l'eau dans un circuit de plomberie : sans eau, les tuyaux et robinets que sont les processus et logiciel ne servent à rien ! L'inverse est également vrai.

L'ensemble complexe de besoins et de contraintes présenté dans la section précédente a dessiné l'esquisse de la structure du pipeline en arrêtant les choix techniques (tels que le choix des logiciels de création) pour chaque étape de fabrication. La question des données (c'est-à-dire le choix des formats de fichiers) et de leur structure viendra ensuite consolider le *workflow* en figeant certaines décisions. Cela permettra de déterminer la complexité des outils à développer qui peut être assimilée au nombre de tuyaux à poser par rapport à la comparaison ci-dessus.

Ainsi que le précise Flavio Perez dans son mémoire [19, p. 29], la nature des données est étroitement liée aux entrées/sorties (aussi communément appelé *input/output* dans le milieu) des différentes étapes de fabrication. Les entrées/sorties parcourues dans le Tableau I.1.2 en Sous-section I.1.2.2 représentent les spécifications des types d'informations que les fichiers stockeront.

Pour une étape de fabrication donnée, le format des fichiers de travail doit être compatible au logiciel de création utilisé. Si le *rig* se fait sur *Blender*, les fichiers de travail seront des `.blend`, le seul format supporté par le DCC *Blender* pour stocker les informations de *rig* qui lui sont propres tels que les *drivers*[46] ou encore des contraintes d'animation. Comme le relève Flavio Perez, « le résultat d'une étape de fabrication génère souvent un fichier utile à l'étape suivante » ([19]). Sur la Figure I.1.11, on observe que la fabrication des *assets* se déroule entièrement sur *Blender*. Cela simplifie la gestion des fichiers en utilisant le même format de fichier à travers les étapes de modélisation, *shading* et *rigging* : des fichiers `.blend`.

Pour faciliter la compréhension des enjeux des différents formats, nous les catégoriserons en fonction de leur finalité : les fichiers de travail et les fichiers de cache. Les formats de travail, directement exploitables par les DCC stockeront tous les aspects de la création d'un artiste sur une tâche, sans perte d'information. Les formats natifs des DCC (par exemple, les `.blend` pour *Blender*, les `.psd` pour *Photoshop*[47]) entrent dans cette catégorie. À l'inverse, les fichiers de cache, plus simples, ne stockeront que le résultat du travail et requièrent une phase d'import-export pour être exploités dans les logiciels.

Lorsque les fichiers de travaux natifs aux DCC assurent les échanges entre les étapes de fabrication on parle de *dataflow* non-destructif. À l'inverse, lorsque ce sont des fichiers de cache (qui occasionnent une perte d'information) on parle de *dataflow* destructif.

En fonction de leur finalité, les formats ont un impact décisionnel important dans la chaîne de collaboration. Les fichiers de travail sont utilisés dans tout le cycle de vie d'une tâche de fabrication (voir la Figure I.1.9) ; ils supportent le processus itératif de création en donnant aux artistes la capacité de revenir sur un état précédent de leur travail sans perte. Ils sont aussi privilégiés entre certaines étapes de fabrication qui impliquent des échanges réguliers (telles que les phases comprises dans la construction des *assets* sur la Figure I.1.11). Les fichiers de cache sont voués à un tout autre objectif : figer l'information. On y fait appel lorsque l'on souhaite figer une décision artistique

46. Dans *Blender*, les *drivers* sont un moyen de contrôler les valeurs des propriétés au moyen d'une fonction, ou d'une expression mathématique.

47. *Photoshop* est un logiciel de traitement d'image. `https://www.photoshop.com`

pour toute la suite de la production (par exemple, au sortir de l'étape d'animation, tous les objets animés sont exportés dans des fichiers de cache). Cela représente deux avantages :

— une sécurité : le cache verrouille l'aspect exporté, évitant des erreurs de modifications par les maillons suivants ;

— une optimisation des scènes : en ne stockant que le résultat, le cache est plus léger et allège ainsi les scènes de travail des étapes suivantes.

Si le choix du format joue un rôle important dans la chaîne décisionnelle, il a également vocation à simplifier l'interopérabilité entre les DCC. C'est la raison pour laquelle des formats standards (visant à être compatibles entre plusieurs DCC) fleurissent depuis plusieurs années.

	Alembic	VDB	USD	OpenEXR	Ptex	MaterialX	OSL	GLtf	OTIO
Animations	•		•					•	
Particules	•							•	
Rigs									
Nurbs	•		•					•	
Meshes	•		•					•	
Curves	•		•					•	
Matériaux	•		•			•	•	•	
Textures	•		•	•	•			•	
Cameras	•		•					•	
Lights	•		•					•	
Séquences									•
Volumes		•	•					•	
Scene graph			•					•	
Simulations									

TABLE I.1.3 – Informations supportées par les formats standards en 2021.

Dans le Tableau I.1.3, nous avons fait l'état des principaux standards utilisés dans les productions de films en images de synthèse avec les informations qu'ils supportent. On observe que hormis pour le *rig*, la majorité des aspects d'une production en image de synthèse est couverte par un standard. L'*alembic*[48] et le VDB[49] sont des formats de cache tandis que l'Open Shading Language (OSL)[50], MaterialX[51], Open Timeline IO (OTIO)[52] et Ptex[53] sont des fichiers de travaux. L'Universal Scene Description (USD)[54] se situe entre les deux, car bien qu'il puisse être utilisé comme fichier de travail, il occasionne des pertes d'information et nécessite souvent un import-export. Le

48. Alembic est un format d'échange de données graphiques open source originellement développé par Sony Pictures Imageworks et Industrial Light & Magic. `http://www.alembic.io/`

49. VDB est un format d'échange de données volumétriques open source originellement développé par DreamWorks Animation. `https://www.openvdb.org/`

50. Open Shading Laguage est un langage de shader open source originellement développé par Sony Pictures Imageworks. `https://github.com/AcademySoftwareFoundation/OpenShadingLanguage`

51. MaterialX est un format d'échange de matériaux open source. `http://www.materialx.org`

52. Open Timeline IO est un format d'échange de données de montage open source développé par Pixar. `https://github.com/PixarAnimationStudios/OpenTimelineIO`

53. Ptex est un format d'échange de *mapping* de textures développé par Disney. `https://ptex.us/`

54. Universal Scene Description est un format d'échange open source de scène 3D. `https://github.com/PixarAnimationStudios/USD`

GL Transmission Format (glTF) [55] est un format plus répandu dans l'univers du temps réel, mais qui serait tout à fait légitime à des fins de visualisation d'*asset*. Finalement, pourquoi ne pas utiliser que des standards à travers la chaîne de fabrication ? Dans un monde idéal, utiliser des standards tout au long de la fabrication simplifierait l'intégration de nouveaux logiciels de création au pipeline et faciliterait les échanges de données avec des studios externes. Ils pérenniseraient aussi les *assets* qui ne seraient plus archivés dans des formats propriétaires obsolètes. Cependant, l'utilisation d'un standard est limitée aux DCC bénéficiant d'une intégration (brique logicielle permettant au DCC de l'importer et l'exporter). Plus un standard est intégré, plus il sera adopté au sein des studios. Mais l'arrivée d'une intégration dépend de la réactivité de l'éditeur ou de la communauté développant le logiciel de création ; cela peut arriver plusieurs années après la sortie du standard : par exemple dans *Blender*, l'intégration de l'USD est encore en chantier à l'heure où nous écrivons ces lignes alors que ce dernier est utilisé en production depuis 2016).

	Alembic	VDB	USD	OpenEXR	Ptex	MaterialX	OSL	GLtf	OTIO
Blender	•	•	WIP	•			•	•	
Maya	•	•	•	•	•			•	
3Ds Max	•		•				•		
Modo	•		•					•	
ZBrush	•			•					
MudBox					•				
Mari	•			•	•				
Marvelous Designer	•				•	•			
Substance Painter	•			•				•	
Substance Designer						•			
Houdini	•	•	•	•	•			•	
Realflow	•	•							
Maxwell	•	•		•					
Guerilla	•	•		•	•				
Renderman	•	•	•	•	•		•		
Vray	•	•		•	•		•		
Arnold	•	•		•		•	•		
Clarisse	•	•	•	•			•		
Katana	•		•	•	•		•		
Nuke	•		•	•					
Krita				•					
Photoshop				•					

Table I.1.4 – Compatibilité entre DCC et standards en 2021.

Le Tableau I.1.4 donne une vision d'ensemble du support des standards au sein des logiciels de création que l'on retrouve fréquemment au sein des productions en 2021 [56]. On constate que l'*alembic* et OpenEXR [57] sont largement adoptés à travers les logiciels

55. glTF est un format de fichier open source pour les scènes et les modèles tridimensionnels. Il a été créé par Khronos Group. `https://github.com/KhronosGroup/glTF`

56. Version à jour de ce tableau pouvant fournir une source information pour orienter le choix des standards : `https://docs.google.com/spreadsheets/d/1-XJscWD-hLPsLMFkxQszHUc6jAfhNpEJHmcLqpYf_Aw`

57. OpenEXR est un format d'échange d'image multicouche open source. Il fut originellement développé par Industrial Light & Magic. `https://www.openexr.com`

listés. On peut également considérer VDB et OSL comme largement adoptés au sein des solutions au vu de leurs spécifications (voir Tableau I.1.3). En revanche, l'USD ne fait pas l'objet de suffisamment d'intégrations pour justifier une forte adoption ; seuls les gros studios disposent des moyens nécessaires pour l'intégrer.

Les fichiers représentent donc l'une des bases fondamentales pour la collaboration entres les équipes de production/artistiques. L'utilisation des fichiers supporte directement la division du travail en tâches individuelles tel qu'exposées en Section I.1.2 : dans une tâche, un artiste est amené à modifier un ou plusieurs fichiers. Ces fichiers seront ensuite évalués lors du processus de *review*. Si une modification est nécessaire, l'artiste produira une nouvelle itération [58] de ses fichiers de travail. Lorsque la tâche est finie (passée en DONE sur l'*asset manager*), le fichier est utilisé par l'étape suivante.

I.1.3.3 Un ensemble d'outils logiciels

Dans un petit projet, un pipeline peut se résumer à échanger manuellement des fichiers nommés correctement. Mais la nécessité d'automatiser se fait ressentir très rapidement lorsqu'on parle de millions de fichiers sur des productions plus importantes. Cette automatisation de la chaîne de production se cristallise à travers un ensemble d'outils et de processus logiciels développé par les équipes techniques. Pour reprendre la métaphore de la plomberie, elle représente les tuyaux et les composants qui transportent et transforment l'eau tout au long du circuit. Cette composante clef du pipeline transparaît dans la définition de Mike Stein lorsqu'il le décrit comme « un ensemble de logiciels et technologies qui permettent aux artistes de travailler sur une grande quantité de tâches de façon consistante, efficiente et répétée ». En automatisant les traitements des données en amont et en aval du travail des artistes, le pipeline optimise la fabrication du film et connecte efficacement les acteurs de la chaîne de production.

Cette section trace le portrait des différents outils à travers des exemples présents au sein de Cube Creative. Pour ce faire, nous utiliserons la classification établie par Flavio Perez dans son mémoire [19, p. 40], il y distingue trois grandes catégories d'outils :

— les outils d'automatisation et de confort ;

— les outils qui vont de la collecte d'information au suivi de la fabrication ;

— les outils qui structurent la gestion des fichiers : les *assets* manager.

Outils de confort et d'automatisation

Les outils de confort et d'automatisation sont destinés à soulager les graphistes des tâches redondantes, fluidifier leur travail. Ils interviennent dans toutes les étapes de fabrication et représentent ainsi une grande partie des outils du pipeline. Le langage de programmation utilisé pour créer les outils dépendra de leurs contextes d'utilisation et de leurs niveaux de complexité. Dans *Blender* par exemple, pour un outil de préréglage de rendu, un *add-on* écrit en Python suffit amplement tandis que pour ajouter l'*importer* de VDB nécessaire à l'import des FX sur la série *MushMush*, nous avons dû modifier le code source en C/C++. L'Annexe C reflète la proportion d'outils de confort et d'automatisation à Cube Creative. Selon les données présentées, cette catégorie représente 92% des outils développés. *Kitsudi* (cf. Figure I.1.12) est un exemple d'outil

58. Une itération est une modification d'un fichier ou d'un asset qui est enregistrée (publiée) et versionnée. Le but, la plupart du temps, est une amélioration technique ou artistique. Définition par Lepipeline.Org.

développé à Cube pour automatiser différents traitements, telle que la génération des scènes de *layout* (*generate layout* sur la Figure I.1.12).

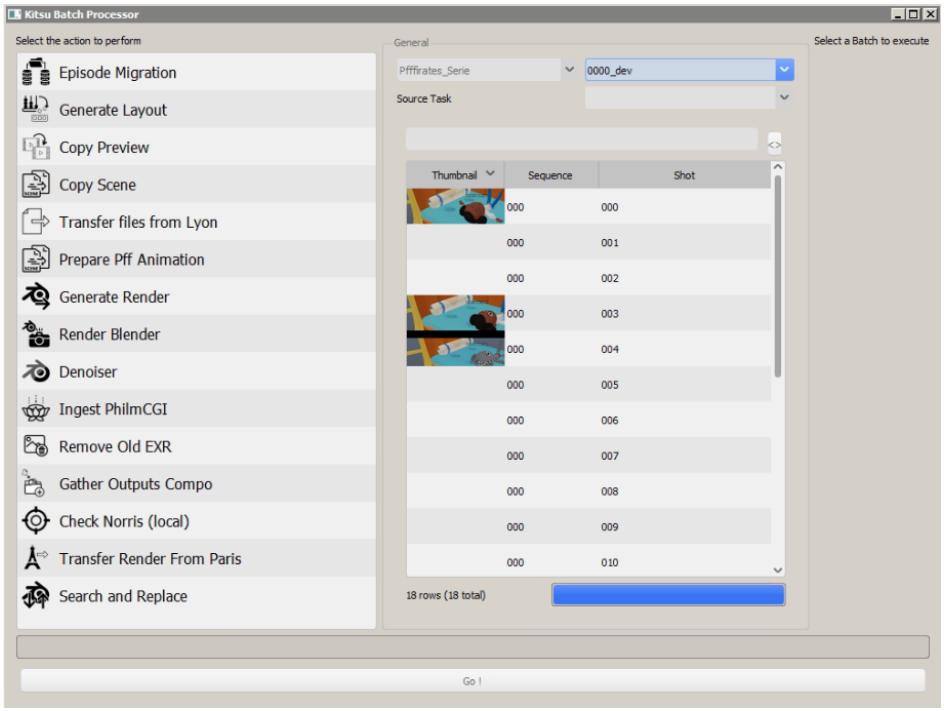


FIGURE I.1.12 – Le logiciel de moulinette *Kitsudi* développé à Cube Creative par l'équipe R&D

Outils de collecte d'information et de suivi de production

Les outils de seconde catégorie se destinent à extraire des informations de la chaîne de fabrication et à les exploiter dans les outils de suivi de production (ou *production tracker*[59]). Comme le note Flavio [19, p. 42], les outils allant de la collecte d'informations au suivi de production vont généralement de pair : l'un va nourrir l'autre. Le logiciel de suivi de production, communément appelé *production tracker* dans le milieu, matérialise entre autres, le cycle de vie des tâches exposé en Section I.1.2. C'est un véritable hub où production, artistes et supervision se retrouvent pour collaborer à travers les processus d'assignation et de validation exposés en Sous-section I.1.2.3. En ce sens, il représente l'un des principaux leviers de communication au sein de la chaîne de fabrication, ce qui en fait un objet particulièrement intéressant dans le cadre de notre état de l'art. Dans les coulisses, le cœur du *production tracker* est une Base De Données (BDD)[60] dotée d'une interface web qui permet de la visualiser et manipuler. En général, il met à disposition une Application Programming Interface (API)[61]

59. Le *production tracker* est le logiciel (ou ensemble d'outils) qui permet d'harmoniser les départements artistiques et techniques avec les départements de la production. Il permet de gérer un projet, suivre son avancement, faire de l'assignation, faire des plannings, des budgets ou encore faire des *reviews*. Définition par LePipeline.org.

60. Une Base De Données est un ensemble structuré et organisé de données qui représente un système d'informations sélectionnées de telle sorte qu'elles puissent être consultées par des utilisateurs ou par des programmes. Définition du Larousse

61. Une *Application Programming Interface* est un ensemble particulier de règles et de spécifications qu'un programme logiciel peut suivre pour accéder aux services et aux ressources fournis par un autre programme logiciel particulier qui met en œuvre cette API et pour les utiliser.

pour écrire et lire dans la BDD depuis des outils extérieurs. Par exemple, certains outils d'automatisation l'exploiteront pour afficher ou changer le statut d'une tâche. Véritable mine d'information, il confère à l'équipe de production une vision claire de la progression du travail pour anticiper les retards et planifier la suite de la fabrication.

Sur *Tangranimo*, le *productions tracker* en place est *Kitsu*[62], une solution open-source développé par CG-Wire[63]. La Figure I.1.13 expose l'une des vues principales de l'interface web de Kistu. Le statut des tâches, affiché par plan et par étape de fabrication donne un aperçu de l'avancement de la fabrication sur l'ensemble d'un épisode. D'autres aspects de la production sont disponibles tels que le planning, le budget ou encore la visualisation de statistiques de fabrication. Le processus d'assignation est simple : une fois créée dans le *production tracker* (ici *Kitsu*), l'assignation se fait directement par un chargé de production ou un lead. L'artiste assigné peut alors mettre à jour le statut de la tâche depuis l'interface web ou depuis un outil d'automatisation (en interne, l'outil exploitera l'API de Kistsu : Gazu[64]). Dans Kistu, le processus de *review* se formalise dans un fil de commentaires lié à une tâche (encadré en jaune sur la Figure I.1.13). Au moment de passer sa tâche en WFA, l'artiste met en ligne le résultat de son travail à valider. En fonction du type de tâche, ce résultat peut prendre la forme d'une image ou d'une séquence d'image (encadré en vert dans la Figure I.1.13). À travers l'interface web, l'organe de validation bénéficie d'une interface pour annoter l'image en dessinant (cf. à la fleur rouge sur la Figure I.1.13) ou avec du texte. On peut constater que la *review* se base sur une information 2D à l'opposé du travail de l'artiste qui a lieu en grande partie en 3D. Ce changement de dimension peut être problématique dans l'interprétation spatiale des annotations et commentaires de la *review*. C'est une limitation présente dans beaucoup de production tracker à l'heure actuelle.

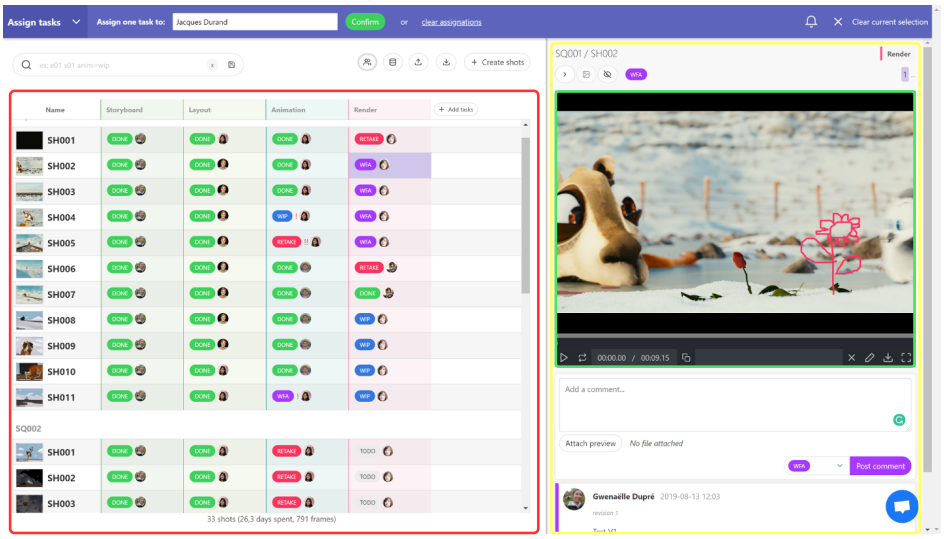


FIGURE I.1.13 – Interface du *production tracker Kitsu* sur le projet Caminandes.

Asset managers

Les *asset managers* représentent une couche d'abstraction logicielle entre les fichiers et leurs utilisateurs vouée à structurer et à simplifier la gestion des fichiers tout au long

62. `https://www.cg-wire.com/en/kitsu`
63. `https://www.cg-wire.com`
64. Documentation de l'API de *Kitsu* : `https://gazu.cg-wire.com`

du pipeline. De par leur complexité, les *asset managers* sont majoritairement exploités dans les studios ayant des équipes et des compétences suffisantes en IT, et en R&D. Flavio Perez les définit comme des « programmes chargés de gérer les fichiers d'une production » [19, p. 46] qui répondent aux problématiques suivantes :

— Qui peut accéder aux fichiers ?
— Quelles sont les relations entre les *assets* ?
— Comment modifier un *asset* ? (Avec quel DCC ? Quel environnement ?)
— Comment stocker et gérer les versions d'*asset* ?

On peut identifier l'*asset manager* à un agent de la circulation qui fluidifie la collaboration entre artistes. En verrouillant intelligemment l'accès aux *assets*, il empêche que deux artistes modifient le même fichier simultanément.

À Cube Creative, l'outil *Tube* a été développé pour répondre à ces questions. C'est un *asset manager* conçu en Python qui expose une API pour manipuler les *assets* d'une production. Chaque DCC utilisé dans la chaîne de fabrication a une intégration pour naviguer et manipuler les *assets* gérés dans l'*asset manager*. Ce dernier est tentaculaire par définition puisqu'il touche à toutes les étapes de fabrication.

À travers cette section nous avons illustré les coulisses d'un pipeline de fabrication de séries d'animation ainsi que ses enjeux vis-à-vis de la collaboration qu'il supporte entre artistes, production et technique. De par sa relation avec le *workflow*, nous avons constaté qu'il entretenait les murs existant entre les départements et les artistes.

Conclusion

Dans ce chapitre, nous avons esquissé un portait rapide de l'« organisme vivant » que représente la chaîne de fabrication d'un film d'animation en images de synthèse. Un premier survol du *workflow* a dessiné les contours des enjeux créatifs des familles de métiers agissant au cœur d'un processus linéaire et rigide, composé d'étapes interdépendantes. Nous avons ensuite étudié ces familles à la loupe pour comprendre leur relation, noyau de la collaboration humaine au sein de la fabrication. Il a été constaté que l'organisation humaine, structurellement verticale, était cloisonnée en départements et représentait une barrière à la communication entre les différentes étapes de fabrication. Au fil de la production, les artistes travaillent sur des tâches individuelles, les rendant insensibles aux problématiques artistiques et techniques des départements suivants ou précédents. Ce manque de vision globale du projet en cours de construction chez les artistes peut naturellement conduire à des erreurs (par exemple, un personnage placé hors champs lors de l'animation peut poser des soucis d'ombres durant l'étape de *lighting*) pouvant émerger après plusieurs étapes de fabrication et coûter cher à résorber. Finalement, le pipeline, véritable structure logicielle qui supporte et modélise le *workflow* semble partager des caractéristiques du fordisme connu pour son efficacité, mais qui ne laisse aucune place aux imprévus.

En industrialisant la production de film d'animation, nous avons mécanisé la création, un processus à la base très organique. Cette approche moderne de la fabrication de film est aujourd'hui omniprésente, car elle répond au système de financement de l'industrie du film. Les investisseurs veulent être rassurés, avoir des calendriers de production complets avant d'engager les frais. Par conséquent, les équipes de production construisent des plannings à long terme qui laissent peu de place à l'inattendu.

Bien que nos méthodes de production actuelles tendent à brider la communication entre départements, l'arrivée de nouvelles technologies commence à faire bouger les choses. Depuis quelques années, un secteur en pleine effervescence se rapproche technologiquement de l'industrie du cinéma : le jeu vidéo. Dans quel domaine a lieu cette fusion technologique ? Quels en sont les bénéfices pour l'univers du cinéma d'animation ?

Chapitre I.2

Vers une fusion des méthodes de fabrication

Introduction

Ce chapitre étudie la remise en question des méthodes de production, engendrée par la convergence technologique de l'univers du jeu vidéo et du cinéma.

Pour contextualiser ce phénomène, nous commencerons par exposer les contraintes techniques qu'implique l'utilisation d'un moteur de jeu.

Les points exposés constitueront une base pour parcourir et comprendre le vaste paysage que représente la production virtuelle (que nous dénommerons *virtual production*[65] par la suite). Celle-ci, en pleine expansion, est un véritable point de fusion entre l'univers du jeu vidéo et du cinéma. Nous confronterons ensuite la viabilité de ces technologies et méthodes face aux spécificités des productions de film d'animation.

Nos premières expérimentations paveront ensuite la route de recherches de nouveaux modes de collaboration au cœur des outils de création. Enfin, nous enrichirons ce questionnement du modèle de production actuel grâce aux sessions de réflexion organisées par le collectif de studio *Open Pipe*[66].

65. La production virtuelle ou *virtual production* est un terme général qui désigne un éventail de méthodes de production et de visualisation de films assistée par ordinateur.

66. *Open Pipe* est un groupe fondé par Supamonks, Cube Creative, Ubisoft Animation Studio et Karlabs dans l'optique de mettre en commun des réflexions et des outils innovants de pipeline de production.

I.2.1 Spécificités des moteurs de jeu

La philosophie qui organise le fonctionnement des moteurs de jeu provient directement du jeu vidéo. Il est donc important de connaître quelques subtilités structurelles et techniques liées à la création de jeu vidéo pour comprendre les contraintes et avantages de l'utilisation d'un moteur de jeu dans une production de cinéma d'animation.

I.2.1.1 Structure des projets

Comme présenté dans le Chapitre I.1, le cinéma d'animation s'appuie sur un découpage en plans et séquences pour organiser la fabrication des éléments qui compose un film. À l'opposé, dans le jeu vidéo la fabrication suit un découpage en niveaux [67] et sections. Les sections sont simplement des portions de niveau dont le chargement peut être fait individuellement.

Le film suit un découpage temporel tandis qu'il est spatial pour le jeu vidéo.

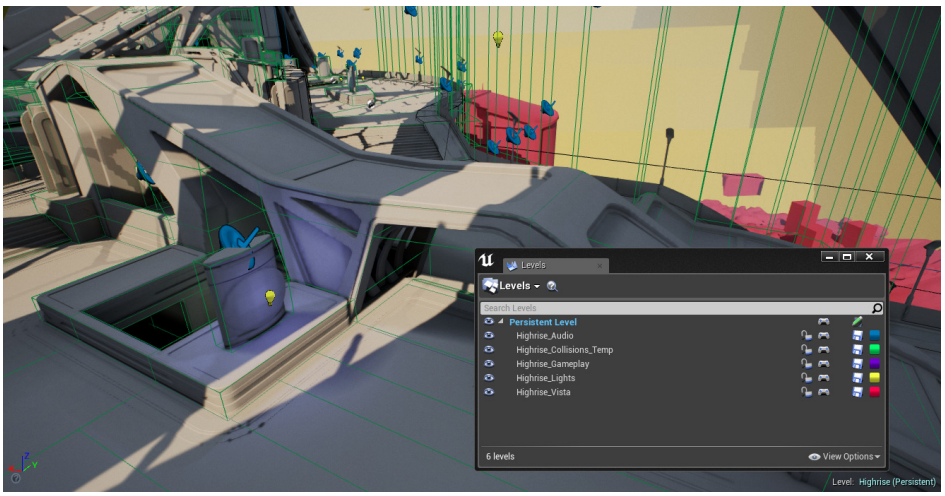


FIGURE I.2.1 – Exemple d'organisation d'un niveau en section sur *Unreal Engine* avec une coloration des *assets* par section. Source : Documentation d'Unreal Engine

I.2.1.2 Types de données

Un film est fait d'*assets* construits par les équipes artistiques, et de ce fait, ce sont principalement des éléments graphiques. Basé sur l'interactivité, le jeu vidéo implique en plus des équipes de développeurs qui construisent des *assets* logiques (un script pour activer un piège par exemple). Par conséquent, le moteur de jeu jongle avec plus de types de ressources qu'une production de cinéma. Cela augmente aussi le niveau de complexité des dépendances entre *assets* car le code est lié aux éléments graphiques. Par exemple, changer la longueur d'une animation pourrait involontairement influencer l'efficacité d'un personnage lors d'un combat.

La création des *assets* graphiques se fait à l'extérieur du moteur de jeu, dans des DCC similaires à ceux utilisés dans le film d'animation. Pour échanger les *assets* entre

67. Un niveau (ou *level* ou *stage* en anglais) ou carte (en anglais *map*) est une étape dans un jeu vidéo. Généralement, il correspond à une unité de lieu dans la progression. Les niveaux se distinguent par diverses caractéristiques : environnement, topographie, ennemis, objectifs, difficulté, etc. Définition par Wikipédia.

DCC et moteur de jeu, deux phases sont nécessaires : une phase d'export du DCC et une phase d'import dans le moteur de jeu. Un format intermédiaire devient nécessaire pour faire tampon entre le moteur et le DCC.

Finalement, dans une production de jeux vidéo, on retrouve trois grandes catégories de fichiers :

— les fichiers de travaux des DCC (par exemple, les fichiers `.blend`) ;

— les fichiers tampons (par exemple, les fichiers `.fbx`) ;

— les *game assets*[68] exploités par le moteur de jeu.

Ces différences fondamentales ont directement une influence sur la nomenclature et la structure de fichiers utilisés.

I.2.1.3 Structure de fichiers

La nomenclature va dépendre du moteur de jeu utilisé, car les types d'*assets* présents relèvent de ce dernier. En général chaque éditeur de moteur donne ses propres *guidelines*[69], nous n'avons donc pas représenté la nomenclature dans la Figure I.2.2.

En prenant en compte les différences expliquées dans les paragraphes précédents, on obtient la structure de fichiers représentée sur la Figure I.2.2. On qualifie cette configuration de *structure en miroir*, car l'agencement des fichiers est dupliqué à la racine en suivant la logique fichier de travail/. La hiérarchie suit ensuite un découpage par type d'*asset* et par niveau.

I.2.1.4 Dataflow

Pourquoi faire un export et un import ?

Les fichiers de travaux des DCC fournissent trop d'informations par rapport aux besoins du moteur de jeu, l'export du DCC permet d'enlever les données inutiles. L'import convertit les données exportées en un nouveau format de travail optimisé pour ses besoins (`.uasset` pour *Unreal Engine* par exemple). Cette étape est responsable d'un formatage des données incluant entre autres la génération des collisions, des données de *lighting*, etc.

En soi, ce processus d'exports/imports n'est pas obligatoire, mais sans lui, on perd en flexibilité. Il permet de reformater les fichiers tampons sans demander aux artistes de les ré-exporter. Cela laisse également la possibilité d'exploiter les *assets* pour différentes plateformes sans d'autres efforts de l'équipe artistique.

Cependant, dans certaines situations, il peut être fort utile d'éviter ce processus fastidieux ; pour obtenir la prévisualisation rapide d'un *asset* en cours de création, par exemple. Selon le moteur de jeu, il existe des outils qui exportent ou diffusent en flux continu directement du contenu depuis les DCC. L'*add-on BlenderTools*[70] en est un parfait exemple, il transfère les éléments d'une scène *Blender* directement dans *Unreal Engine* sans créer de fichiers tampons en passant par le réseau.

68. Un *game asset* est un *asset* utilisé dans un moteur de jeu. Ce terme désigne aussi bien des entités logiques (exemple : un script) que graphiques (exemple : un modèle 3D).

69. Dans le jeu vidéo, une *guideline* est une liste de règles de nomenclatures et d'organisation de fichiers dont l'objectif est d'uniformiser les pratiques. Deux exemples de *guideline* pour *Unreal Engine* : la *Marketplace Guideline* (`https://www.unrealengine.com/en-US/marketplace-guidelines`) et la *Gamemaking Guideline* (`https://github.com/Allar/ue4-style-guide`).

70. `https://github.com/epicgames/blendertools`

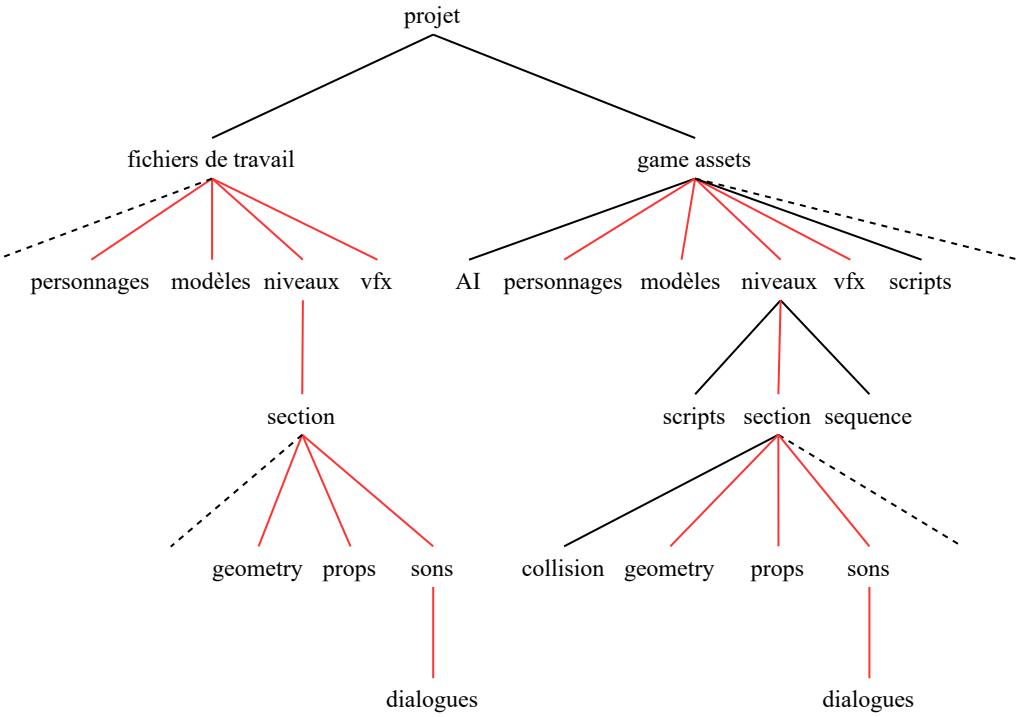


FIGURE I.2.2 – Exemple de structure de dossier en miroir (représentée en rouge).

Quel format de fichier utiliser pour le fichier tampon ?

Historiquement, le Filmbox format (FBX) s'est imposé comme standard pour les échanges de données 3D dans le jeu vidéo, bien qu'Autodesk ne l'ai jamais publié comme tel[71]. Il supporte un certain nombre d'aspects des fichiers de travaux 3D tels que la géométrie, les matériaux, les animations ou encore les *shapekeys*[72] (ou *blendshapes*). Son principal avantage réside dans le fait qu'il soit binaire, ce qui rend son parsing rapide au moment de l'import. Pour les données 2D (textures), les moteurs de jeu supportent la majorité des formats standard (png, jpeg, etc.). Nous reviendrons sur les formats des fichiers tampon dans le cadre spécifique des films d'animation.

I.2.1.5 *Asset manager*

Comme expliqué un peu plus tôt, chaque moteur de jeu possède ses types d'*assets*. Ces derniers étant généralement stockés dans un format binaire, propriétaire et optimisé[73], il est nécessaire d'avoir une interface dédiée pour les manipuler et les visualiser : un *asset manager*.

Tout comme ceux utilisés en cinéma d'animation, les *asset managers* de moteur de jeu ont pour objectif d'afficher, modifier et stocker les données et leurs dépendances.

71. Cela pose problème pour les projets open source par exemple, les utilisateurs doivent accepter une *End-user license agreement* (EULA). Pour cette raison, *Godot* et *Blender* ont écrit leur propre import/export, un processus très fastidieux.

72. Les clefs de formes ou *shapekeys* représentent la position d'un ou plusieurs points de géométrie à un temps donné. Dans d'autres termes, les clés de forme peuvent être appelées « cibles de morphing » (*morph target*) ou « formes de mélange » (*blendshape*). Documentation de *Blender* : `https://docs.blender.org/manual/fr/latest/animation/shape_keys/introduction.html`

73. *Godot* est l'un des rares contre-exemples qui utilisent un format de fichiers lisibles pour stocker les scènes. Documentation du format TSCN de *Godot* : `https://docs.godotengine.org/en/3.4/development/file_formats/tscn.html`

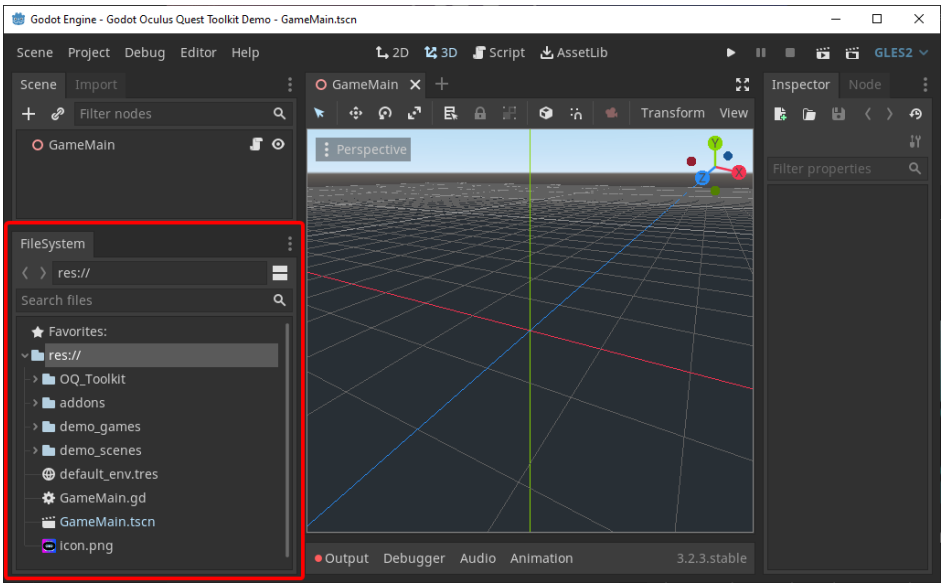


FIGURE I.2.3 – *Asset manager* (encadré en rouge) dans *Godot*.

Ils gèrent en plus l'import/export des *assets* et leur audit [74]. Sur la Figure I.2.4 on observe que l'outil d'audit de l'*asset manager* d'*Unreal Engine* affiche de nombreuses statistiques sur les *assets*. En se basant sur les informations de la colonne `Total Usage` on pourrait par exemple supprimer les *assets* inexploités pour alléger le projet.

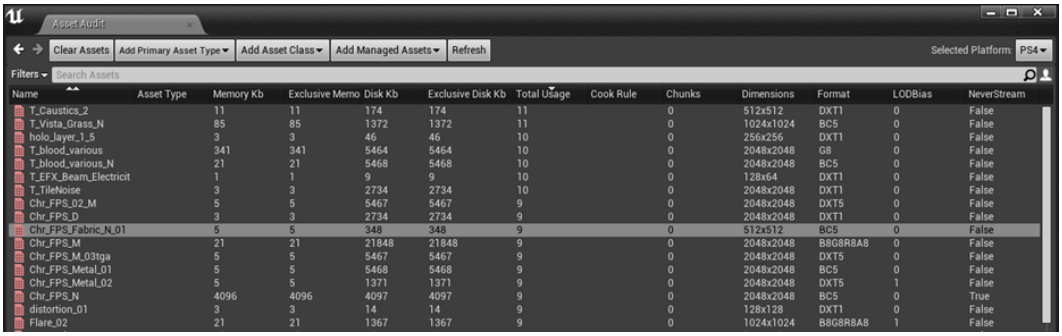


FIGURE I.2.4 – Audit d'*assets* dans *Unreal Engine*.

L'un des avantages d'un moteur de jeu réside dans sa capacité à faire travailler plusieurs personnes parallèlement sur différents aspects d'un même projet. Pour éviter au maximum les conflits entre artistes, leurs *asset managers* se reposent généralement sur l'intégration d'un Version Control System (VCS) [75].

I.2.1.6 Systèmes de gestion de versions pour l'image de synthèse

Les VCS sont des applications destinées à enregistrer les changements qui se produisent sur un ou plusieurs fichiers en fonction du temps, de manière à pouvoir rétablir une version spécifique plus tard en cas de besoin.

74. Dans le jeu vidéo, l'audit est un processus d'inspection qui permet de trouver d'éventuels *assets* problématiques.

75. Un système de gestion de versions ou *Version Control System* est une catégorie de logiciels chargés de gérer les modifications apportées aux programmes informatiques, aux documents ou à d'autres collections d'informations.

Au sein d'un projet, l'utilisation d'un VCS représente une toile de sécurité pour l'individu en conservant des sauvegardes de son travail. Pour l'équipe, il permet de garder les ressources du projet à jour.

Aujourd'hui on distingue deux grandes catégories de VCS utilisées sur les moteurs de jeu : centralisé et distribué.

Comme son nom l'indique, le système centralisé consiste à stocker la totalité des modifications sur un serveur. Les clients ne gardent localement qu'une version souhaitée du projet. Les avantages de cette architecture sont nombreux :

— chacun connaît jusqu'à un certain degré le travail des autres ;

— les administrateurs peuvent contrôler avec une grande précision ce qui peut être fait par quelqu'un ;

— il est possible d'avoir localement qu'une partie du projet (aussi appelée vue), minimisant ainsi l'espace disque utilisé chez les artistes.

Cependant, il y a aussi certains inconvénients :

— si le serveur ou la connexion s'interrompent, plus personne ne peut collaborer, sauver ou mettre à jour sa version ;

— si le disque du serveur lâche, tout l'historique du projet est perdu.

Malgré tout, c'est actuellement l'architecture la plus utilisée dans la production de jeux vidéo, car elle est plus efficiente sur les fichiers binaires (*game assets*). Aujourd'hui, deux solutions sont très employées dans le secteur : *Subversion*[76] et *Perforce*[77].

Les VCS distribués ont été créés pour répondre aux faiblesses de leurs homologues centralisés. Dans un système distribué, chaque client fait une copie complète du projet et de tout l'historique de modification. Par conséquent, si un serveur tombe en panne, n'importe lequel des clients peut remettre en ligne sa copie miroir pour le restaurer. Il permet également de continuer à sauvegarder des modifications sans connexion. *Git*[78] et *Mercurial*[79] sont les deux principales solutions utilisées pour ce système. Cependant, cette architecture n'est pas largement adoptée dans les projets d'image de synthèse, car stocker tout l'historique de gros projets localement devient très rapidement lourd. En revanche, Git bénéficie d'une extension dédiée à soulager ce problème : *Large File Storage* (LFS)[80]. Cette dernière imite le comportement des VCS centralisés pour gérer les fichiers binaires en ne conservant localement que les dernières versions.

Sur un projet de jeu vidéo, le VCS est principalement utilisé sur les *game assets* mais peut aussi être étendu aux fichiers de travaux. C'est un point de pivot qui centralise et supporte le travail parallèle en permettant aux équipes de travailler en continu. Il fournit des mécanismes de mise à disposition des fichiers de travail et de verrouillage qui, s'ils sont exploités correctement, évitent les conflits. Cependant, l'utilisation d'une telle solution requiert de la discipline, car elle implique que chaque artiste travaille dans son univers déconnecté du flux de la production (une version locale du projet). Les artistes œuvrent certes en parallèle, mais doivent mettre à jour régulièrement leur

76. `https://subversion.apache.org`
77. `https://www.perforce.com`
78. `https://git-scm.com`
79. `https://mercurial-scm.org`
80. *Git Large File Storage* est une extension de *Git*. Elle lui ajoute la capacité de gérer correctement les fichiers volumineux en stockant des références au fichier dans le repository, mais pas le fichier lui-même. Pour contourner l'architecture de *Git*, *Git* LFS crée un fichier pointeur qui agit comme une référence au fichier réel (qui est stocké ailleurs). Définition de GitHub.

projet pour avoir constamment les dernières versions. Par conséquent, comme le relève Renee Dunlop [2, p. 137], les décisions artistiques risquent d'être basées sur des *assets* ayant déjà changés, il en est de même pour les décisions techniques.

Cette section a mis en valeur les spécificités et les contraintes liées à l'utilisation d'un moteur de jeu. Ces notions faciliteront la compréhension de l'état de l'art qui va suivre sur leur utilisation dans le secteur du cinéma.

I.2.2 *Virtual production*, carrefour du temps réel et du précalculé

Un film est souvent le fruit de la rencontre du réel et du fictif, que ce soit en prise de vues réelles avec des effets spéciaux ou dans l'animation avec la rotoscopie [81] par exemple. Jusqu'au début des années 2000, cette rencontre se faisait toujours à retardement dans le processus de fabrication linéaire des films : on filme puis on intègre les effets spéciaux, on filme puis on anime avec la référence filmée. Mais depuis quelques années, un lieu de convergence temporelle est apparu grâce aux progrès technologiques logiciels et matériels de l'informatique graphique : la *virtual production*, portée par les moteurs de jeu. On peut assimiler la *virtual production* à une membrane où réel et virtuel se rencontrent dans la même temporalité. Cette notion englobe un ensemble de techniques à la croisée des mondes, qui parsèment la fabrication du film en allant de la préproduction à la postproduction.

Dans cette section, nous parcourrons l'éventail de techniques qui composent la *virtual production*. Nous localiserons chacun de ces procédés au sein du processus de production, ainsi que leurs implications en matière de collaboration pour la création d'un film. Enfin, nous confronterons ces outils aux spécificités de la production de films d'animation.

I.2.2.1 Techniques de virtual production

La *virtual production* étant un domaine en pleine expansion, nous n'avons pas encore beaucoup de recul dessus. Cependant, les techniques qui la composent ne sortent pas de nulle part ; comme de nombreuses méthodes dans le cinéma, ce sont les sociétés du domaine qui en sont à l'origine. Pour dépeindre ce paysage en pleine évolution, nous nous sommes appuyés sur les définitions formulées par l'industrie et par une société à but non lucratif : la *Previsualization Society* qui héberge un comité dédié à l'exploration et à la définition de la *virtual production* depuis 2009 [82]. À ce jour, les *Virtual Production Field Guides* (Volume 1 et 2) [3] constituent les ouvrages les plus complets sur cette discipline émergente.

Bien que la convergence se produise principalement dans le secteur du film en prise de vues réelles, certaines de ces techniques sont utilisées dans des productions de cinéma d'animation. En étudiant l'impact du temps réel sur la fabrication des films, nous explorerons les possibilités d'évolution de ce dernier.

Malgré leurs différences fondamentales, les projets d'animation et de prise de vues réelles ont des phases de productions similaires. Tous deux présentent un cycle en trois temps : préproduction, production et postproduction. Cependant, leur signification, c'est-à-dire le travail effectué durant ces trois étapes, est différente. Historiquement, la phase de production d'un film en prise de vues réelles représente principalement la phase de tournage, tandis que dans un film d'animation numérique, elle inclut toute la conception des images de synthèse. En postproduction d'un film, on retrouve la création de tous les effets numériques ajoutés aux plans tournés en prise de vues réelles. On y

81. La rotoscopie est une technique cinématographique qui consiste à relever image par image les contours d'une figure filmée en prise de vue réelle pour en transcrire la forme et les actions dans un film d'animation. Ce procédé permet de reproduire avec réalisme la dynamique des mouvements des sujets filmés. Définition par Wikipédia.

82. Annonce de la formation du comité : `https://www.awn.com/news/virtual-production-committee-launched`

retrouve donc les mêmes étapes que dans la partie production qu'un film d'animation. En cinéma d'animation, la postproduction est beaucoup plus courte, car elle ne fait que traiter les séquences d'images conçues en production.

La Figure I.2.5 représente une tentative de classification des techniques de *virtual production* en fonction de leur place au sein du cycle de production et de leur affinité au type de projet cinématographique (animation ou prise de vues réelles). Dans cette section, nous décrirons et situerons chacune d'entre elles dans l'ordre de la temporalité du schéma.

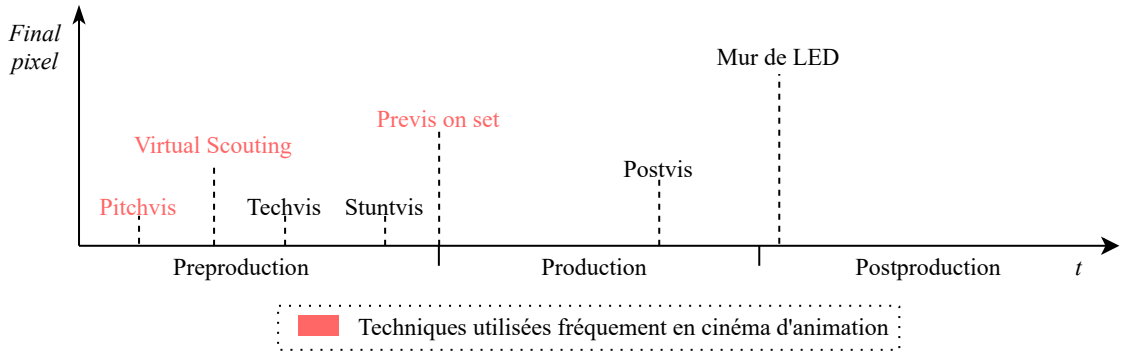


FIGURE I.2.5 – Illustration des différentes techniques de *virtual production* en fonction de leur temporalité d'utilisation dans une production.

Pitchvis **:** dans ce cycle, la *pitchvis* (visualisation de pitch) arrive généralement en première. Cette technique de tournage consiste à créer des séquences clefs du film en associant des prises de vues réelles avec des éléments virtuels pour communiquer l'essence d'un projet en développement à des investisseurs potentiels. Par exemple, la *pitchvis* a été utilisée sur *Alita : Battle Angel* [39] [83] pour montrer le style de combat du long-métrage aux investisseurs. Cette technique à aussi bien été utilisée pour des films d'animation impliquant de la *performance capture* [84] que des films en prise de vues réelles. [85]

Virtual scouting **:** le *virtual scouting* (repérage virtuel) est une technique utilisée pour assister le réalisateur et le directeur de photographie dans la construction des décors réels ou fictifs. Immergés dans les scènes en réalité virtuelle, les artistes ont la possibilité de placer en temps réel des *props* [86] créées au préalable pour composer les décors. Dans certains outils de *virtual scouting*, il est possible d'enregistrer des points de vue qui guideront les départements suivants dans la réalisation du projet. En créant un pont entre la fabrication des décors et la prise de vue, le *virtual scouting* permet de préciser les *sets* [87] à fabriquer et évite le gaspillage de ressources (décors inutiles). Dans l'animation, cette méthode est idéale pour faire du concept de décors. À Cube Creative, le directeur artistique Tristan Michel a utilisé cette technique pour créer certains prototypes de décors en réalité virtuelle sur l'application Medium [88], qui permet de sculpter en trois dimensions.

83. Making of de la *pitchvis* d'*Alita : Battle Angel* : `https://youtu.be/bvD-6ROSE70`
84. Le processus d'enregistrement des mouvements d'un acteur, utilisé par exemple dans le cinéma et les jeux vidéo. Traduit du Cambridge Dictionnary.
85. Quelques exemples vidéo de *pitchvis* : `https://vimeo.com/225877552`
86. Éléments de décor d'une scène. Exemple : une casserole.
87. En cinéma, le *set* correspond à l'enceinte dans laquelle est tournée une scène de film ; il comprend les décors et les accessoires.
88. `https://www.adobe.com/products/medium.html`

Techvis **:** la *techvis* (visualisation technique) est la combinaison d'éléments virtuels avec des équipements du monde réel pour le processus de planification des prises de vue ainsi que la combinaison de séquences déjà capturées avec des éléments virtuels. C'est aussi le domaine où les mouvements de caméra, le placement de la caméra et les choix d'objectifs peuvent être validés, ce qui atténue le risque de choix virtuels physiquement invraisemblables. La Figure I.2.6 montre les détails typiques d'une *techvis* (le placement des acteurs, des caméras, etc.). Cette visualisation guide également la création des effets spéciaux en postproduction en apportant un référentiel précis pour les positionner. La *techvis* crée donc un pont entre la préproduction et la postproduction.

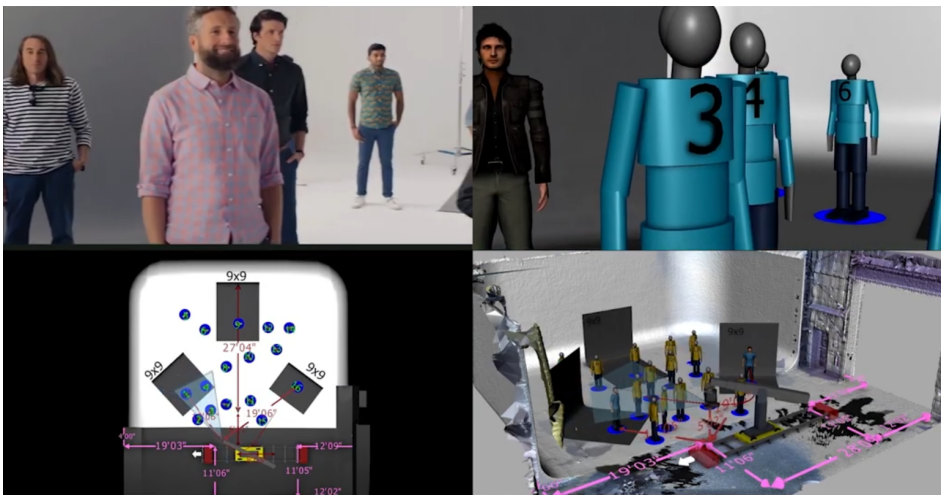


FIGURE I.2.6 – Exemple de *techvis* tirée de la vidéo explicative du studio The Third Floor. Source : `https://youtu.be/dHm_o83tW6o`

Stuntvis **:** la *stuntvis* (visualisation des cascades) permet aux chorégraphes de créer des chorégraphies réalistes à travers des simulations physiques menées dans les moteurs de jeu. Les équipes de cascadeurs peuvent ensuite répéter et exécuter ces plans simulés, ce qui permet d'améliorer l'efficacité de la production et de réduire le nombre de jours de tournage.

Previs on set **:** la *previs on set* est issue de la prévisualisation, mais contrairement à cette dernière, elle est employée directement sur le tournage et rassemble plusieurs unités selon le budget et les besoins de la production. Son objectif consiste à fournir en temps réel une vue des éléments virtuels du film en extension de décors (incrustation à la place d'un fond vert) ou en réalité augmentée au moment de la capture des plans. Généralement, elle compte une unité de *performance capture* responsable du traitement de la capture des mouvements des acteurs, une unité de rendu gérant la visualisation temps réel des aperçus des effets visuels et une unité de tracking chargée de la reconstruction de la position de la caméra pour l'unité de rendu. Certains estiment que l'on peut aujourd'hui considérer la *previs on set* comme le début de la production, car elle fournit des éléments exploités directement par les équipes d'effets spéciaux.

Postvis **:** à la différence de la *previs on set*, la *postvis* (post visualisation) arrive plus tard dans la chaîne de fabrication [89]. Lorsque le département éditorial commence à monter les plans capturés durant la phase de production, il est souvent compliqué de comprendre l'action sur les shots représentant uniquement des acteurs capturés sur

89. Vidéo de comparaison par *Pixel Liberation Front* entre *previs* et *postvis* sur le film *La Planète des singes : Les Origines* [41] : `https://vimeo.com/28203887`

fond vert. Pour pallier ce problème, la *postvis* enrichit ces plans problématiques d'effets spéciaux en cours de production. Cela permet au monteur d'avoir des informations pour le guider dans son travail. Cette technique améliore la communication entre l'équipe de production et l'équipe éditoriale.

Murs de LED : la visualisation sur murs de LED en direct ou *Live LED Wall* consiste à filmer une scène en utilisant un mur de LED pour afficher les éléments d'arrière-plan en image de synthèse (remplaçant ainsi le traditionnel fond vert ou bleu). En exploitant les moteurs de jeu pour le rendu, l'environnement virtuel d'arrière-plan est calculé en temps réel selon le point de vue de la caméra réelle grâce au *tracking*[90] de caméra. L'image filmée comporte ainsi les décors finaux avec les acteurs au moment du tournage. Cette technique est une version interactive du célèbre principe cinématographique de transparence[91] utilisé depuis les années 1930. En obtenant une image définitive au moment du tournage, c'est aujourd'hui la consécration de la philosophie du *final pixel*[92]. Elle se situe à mi-chemin entre la production et la postproduction, car les étapes de montage et de *mastering* sont toujours requises.

FIGURE I.2.7 – Exemple de plan du *Mandalorian* [38] utilisant le plateau d'ILM StageCraft. Source : FXGuide [27]

Toutes ces techniques créent de nouveaux canaux de communication entre les artistes et départements de la fabrication du film, évitant le gaspillage de ressources et laissant plus de place à l'exploration. Mais qu'impliquent ces nouvelles passerelles pour le processus de fabrication ?

En intervenant majoritairement en phase de préproduction/production des films, elles viennent remodeler profondément leur processus de fabrication. Pour permettre à

90. Le *camera tracking* consiste à reconstituer la position de la caméra en trois dimensions pour reproduire son mouvement virtuellement.

91. La transparence est un procédé mêlant, dans un plan, une projection de film à une prise de vues. Employée en studio pour donner l'illusion de voir derrière les comédiens une scène (filmée auparavant) dont on veut faire croire qu'elle se déroule en même temps que le jeu des comédiens. Définition de Wikipédia.

92. *Final pixel* désigne la possibilité d'obtenir la qualité finale de l'image en direct, dans la caméra, sans avoir à recourir à la postproduction.

ces techniques d'exister, les *assets* doivent être préalablement conçus. Par conséquent, la construction des *assets* se faisant habituellement en postproduction débute maintenant dès la préproduction. Cependant, les moteurs de jeu utilisés au cœur de ces techniques de visualisation ont l'« estomac sensible », ils ne peuvent pas ingérer n'importe quel type d'*asset*. Ces derniers se doivent d'être optimisés et transmis dans un format qu'ils comprennent.

À l'opposé du précalculé, les modèles 3D utilisés dans les moteurs temps réel sont moins lourds, et donc moins détaillés[93]. Cela garanti un temps de calcul d'image suffisamment court pour être interactif. Le niveau d'optimisation des modèles dépend directement des capacités matérielles de l'ordinateur utilisées pour le rendu (mémoire graphique disponible sur le Graphical Processing Unit (GPU), fréquence du Compute Process Unit (CPU), etc.). Il se traduit à travers des limitations sur certains aspects des *asset* tel que :

— les *meshes*[94] doivent comporter un nombre réduit de triangles ;

— la résolution des textures ne doit pas dépasser une certaine taille ;

— les effets physiques ne doivent pas dépasser une certaine quantité de particules.

Ces *assets* sont donc différents de ceux utilisés par les équipes d'effets spéciaux et peuvent nécessiter un travail supplémentaire pour être exploitable dans les DCC en postproduction.

Cette étude nous a donné un aperçu des principales méthodes de visualisation utilisées en *virtual production*. Bien que la majorité de ces techniques soit apparue dans les productions incluant de la prise de vues réelles, nous avons observé que certaines d'entre elles (la *previs on set*, le *virtual scouting*, etc.) se prêtent également aux films d'animation. Ces dernières années, les productions de films en prise de vues réelles ont adopté massivement la *virtual production*. La *virtual production* tisse de véritables passerelles entre les phases de création, cassant la barrière entre tournage et Visual Effects (VFX). En ce sens, elle représente ainsi un véritable outil de communication qui place la collaboration au cœur de la création, permettant aux artistes de créer à travers un *workflow* non linéaire.

Mais, le pipeline de production d'un film d'animation diffère sur de nombreux aspects à celui d'une série d'animation. Dans la section suivante, nous interrogerons les apports de l'utilisation d'un moteur de jeu au centre de sa fabrication à travers quatre cas concrets de l'industrie.

I.2.2.2 Un workflow compatible avec le film d'animation ?

Si les exemples d'usage de la *virtual production* sont de plus en plus fréquent dans l'univers de la prise de vues réelles, ils ne sont pas légion dans le cinéma d'animation. Comment expliquer cette différence ? Qu'est-ce que l'utilisation d'un moteur de jeu induit vis-à-vis du *workflow* et *dataflow* présenté dans le Chapitre I.1 ?

À travers cette section nous nous intéresserons aux cas concrets d'usage des moteurs de jeu et donc du temps réel au cœur des pipelines de production de films et séries

93. Ces règles d'optimisation tendent à devenir caduques avec les progrès technologiques. Par exemple, le système de géométrie virtuelle Nanite développé par Epic Game tend à rendre caduc la limite en nombre de triangles des *meshes*. Documentation de Nanite : `https://docs.unrealengine.com/5.0/en-US/RenderingFeatures/Nanite/`

94. Un *mesh* est un maillage composé de polygones constituant un modèle 3D.

FIGURE I.2.8 – Productions d'animation étudiées. De gauche a droite et de haut en bas : *Adam* [31] par *Unity Technologies*, *Zafari* [34] par Digital Dimension, *Fortnite Trailer* [37] par Epic Games et *Ada* [40] par BlueZoo Animation.

d'animation en images de synthèses. Une étude des expériences menées par différents studios de l'industrie quantifiera l'apport de ces technologies au cœur des méthodes de créations. Nous naviguerons entre quatre exemples représentant différents types de projets d'animation réalisés partiellement sur les moteurs de jeu *Unreal Engine*[95] et *Unity*[96].

Le Tableau I.2.1 liste les caractéristiques des productions étudiées. Nous avons sélectionné ces projets pour leur diversité (format, style de rendu et d'animation) et pour leur usage des moteurs de jeu. On notera qu'aucun projet de long métrage ne figure dans la liste. À notre connaissance, *La Véritable Histoire du Chat Botté* (2009) [32] et *Cendrillon au Far West* (2009) [35] sont les deux seuls longs métrages d'animation rendus avec un moteur temps réel. Ils ont tous deux utilisé *Fantasmagorix*, un outil basé sur *Render Box*[97], le même moteur graphique conçu par Nadeo[98] pour *Trakmania* notamment. Nous ne les avons pas inclus dans notre étude de cas car l'outil n'est plus développé.

Pour cette analyse, il est également important de relever la différence de culture des studios à l'origine de ces projets. BlueZoo Animation et Digital Dimension sont des studios d'animation et d'effets spéciaux tandis qu'Epic Game et *Unity Technologies* sont des studios de jeux vidéo. Nous constaterons tout au long de cette étude l'influence du cœur de métier du studio sur l'orientation des choix d'intégration du temps réel dans la chaîne de fabrication.

95. `https://www.unrealengine.com`
96. `https://unity.com`
97. Article détaillé sur l'utilisation du moteur pour le film : `https://www.pixelcreation.fr/3d-video/animation-3d-vfx/delacave-fantasmagorix`.
98. `https://www.nadeo.com`

	Adam	Zafari	Fortnite Trailer	Ada
Format	Série	Court métrage	Court métrage	Court métrage
Année	2016	2017	2017	2019
Moteur de jeu	Unity	Unreal Engine	Unreal Engine	Unreal Engine
Style Visuel	Photoréalisme	Cartoon	Cartoon	Stylisé
Technique d'animation	Mocap	Keyframe	Mocap	Keyframe
Fichiers tampons	FBX, Alembic	NC (Non communiqué)	FBX, Alembic	NC
Système de gestion de versions	Perforce	NC	Perforce	Perforce
Layout	Unity	Unreal Engine	Unreal Engine	Unreal Engine
Modeling	Maya	Maya	Zbrush, Modo, Maya, 3Ds Max	Maya
Texturing	Substance Painter	Unreal Engine	Substance Painter	Unreal Engine (shaders)
Shading	Unity	Unreal Engine	Unreal Engine	Unreal Engine
Rigging	3DsMax	Maya	Maya	Maya
Animation	Motion Builder, 3DsMax	Maya	Motion Builder, Maya, Live Link	Maya
Système Mocap	Vicon Blade	-	Vicon Blade	-
Cloth	Marvelous Designer	-	AnimDynamics	-
Destruction	Houdini	-	Blender	-
Smoke	Houdini	Unreal Engine	Unreal Engine	-
Lighting	GI Pré calculé	Temps réel	Temps réel	Temps réel
Compositing	Post process	Post process	Post process	Post process
Montage	Unity	NC	Adobe Première	NC

TABLE I.2.1 – Caractéristiques techniques des productions d'*Adam*, de *Zafari*, du *Fortite Trailer* et d'*Ada*. Sources : *Adam* : [21] [29] - *Zafari* : [23] - *Fortnite Trailer* : [17] [6] - *Ada* : [4, p. 114-116] [24]

Construction des *assets*

En regardant les outils de ces projets dans le Tableau I.2.1, on constate qu'ils ont un point commun majeur : la construction et l'animation des *assets* restent externes au moteur de jeu. Sur le *Fortnite Trailer* l'utilisation de la *previs on set* (voir la Figure I.2.9) a permis de créer un *layout* brut très tôt sur *Unreal Engine* (l'équivalent d'une animatique 3D). Mais cela a scindé la création des *assets* en deux parties : une phase de récupération des *assets* issus du jeu vidéo[99] pour la *previs on set* puis leur transformation en *asset* en haute résolution en vue de l'animation et du rendu. À l'inverse d'un pipeline traditionnel où la création des *assets* est monolithique (cf. Section I.1.3), elle devient poreuse et fragmentée lorsqu'un moteur de jeu est utilisé. Poreuse, car elle nécessite des itérations fréquentes entre le moteur et le DCC pour permettre aux artistes de bénéficier d'un rendu de leur travail très tôt.

Les productions en animation *keyframe* (*Zafari* et *Ada*) n'ont pas eu recours à la *previs on set*. La création des *assets* n'était pas fragmentée, elle était similaire à celle d'un pipeline traditionnel (cf. Chapitre I.1). Dans leur cas, la phase d'export des *assets* vers le moteur a débuté lors du *shading*.

99. On parle alors de réutilisation des *assets*. En ce sens, l'utilisation d'un moteur de jeu peut faciliter les projets cross-média.

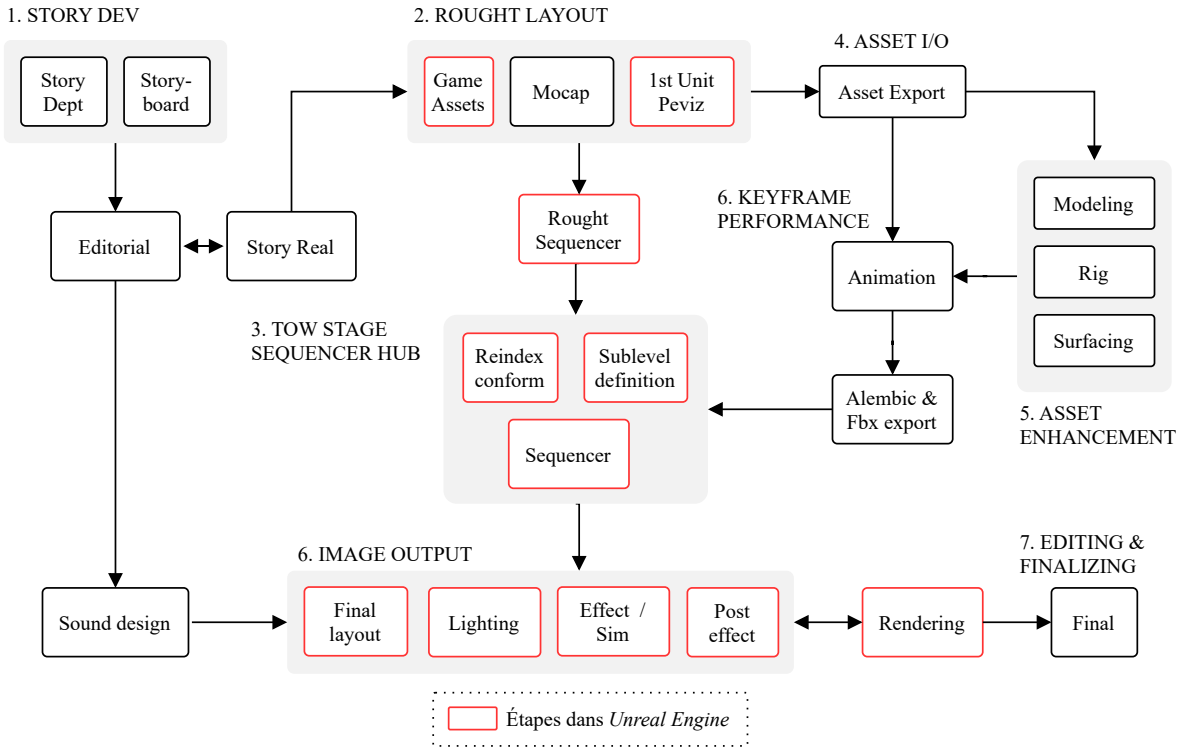


FIGURE I.2.9 – Pipeline du *Fortnite Trailer*. Source : Conférence *Fortnite Trailer Pipeline | Unreal Dev Day Montreal 2017*.

Un pipeline d'animation complexe

Comme observé précédemment, les quatre productions présentées ont développé l'animation à l'extérieur des moteurs de jeu. À l'heure actuelle, ces derniers n'ont pas des capacités de *rig* suffisantes pour répondre aux besoins complexes d'un film d'animation. Par conséquent, sur les quatre exemples de production, le *rig* fut créée dans le DCC où l'animation s'est faite : *Maya*. Dans une interview, Dan Winn alors réalisateur sur *Ada* explique qu'il manque principalement des outils de *rigging* aux moteurs de jeu [4, p. 114].

En effet, il est intéressant de noter que les éditeurs des moteurs de jeu tendent à développer des outils d'animation et de *rigging* dans l'optique de pouvoir animer directement dedans. On peut par exemple citer le *ControlRig*[100] sur *Unreal Engine* qui va dans ce sens.

En attendant la maturation de ces efforts, les studios peuvent recourir à des *plug-ins*[101] DCC similaires au *BlenderTools* (cf. Section I.2.1) pour créer des passerelles avec le moteur. Par exemple, le *plug-in Maya Live Link*[102] développé par Epic Game permet aux animateurs de diffuser en temps réel les animations de *Maya* ou de *Motion Builder* dans *Unreal Engine* pour les visualiser. Ce système a été utilisé sur *Fortnite Trailer* pour visualiser les personnages animés en *motion capture*[103] lors de la *previs*

100. `https://docs.unrealengine.com/4.26/en-US/AnimatingObjects/SkeletalMeshAnimation/ControlRig/`
101. Un *plug-in* ou un *add-on* est un programme ou un script tiers qui est ajouté à un programme pour lui donner des fonctionnalités et des capacités supplémentaires.
102. `https://docs.unrealengine.com/4.27/en-US/AnimatingObjects/SkeletalMeshAnimation/LiveLinkPlugin`
103. La capture de mouvement ou *motion capture* est un processus qui consiste à enregistrer les mouvements d'objets ou d'acteurs, ces données sont ensuite utilisées pour animer des modèles numériques.

on set.

Que ce soit dans *Adam*, *Ada*, *Fortnite Trailer* ou *Zafari*, on retrouve deux formats d'échanges majoritairement utilisés : le FBX et l'*alembic*. Pour *Fornite* ou *Adam*, le format FBX fut principalement utilisé pour transmettre les *props* statiques ou ceux nécessitant leurs squelettes de déformation pour y appliquer les données de *motion capture* en temps réel lors de la *previs on set*. À l'opposé, l'*alembic*, spécialisé dans la transmission de caches d'animation (décrit en Sous-section I.1.3.2), fût principalement exploité comme fichier tampon pour transmettre les animations et les FX vers le moteur de jeu. Son usage au regard du rendu temps réel présente deux avantages :

— figer l'aspect exporté pour la suite de la chaîne de fabrication (détaillé en Sous-section I.1.3.2) ;

— optimiser le *framerate*[104] des scènes : lors de la lecture d'une animation d'objet, le moteur de jeu interpole en temps réel la position de la géométrie en fonction du mouvement du squelette de l'objet. Ce traitement est très lourd lorsqu'il est appliqué aux *assets* haute définition utilisés en animation. Lorsqu'un cache *alembic* est utilisé, l'objet n'a pas de squelette, la position de la géométrie en fonction du temps est directement lue dans le fichier de cache. Le *framerate* n'est donc plus dépendant du temps de calcul de l'interpolation, mais du temps de lecture du cache sur le disque. Par conséquent, il est recommandé d'utiliser des disques de type Solid-state drive (SSD)[105] pour bénéficier de performances optimales.

L'animation d'objet physique non complexe peut être menée en temps réel en exploitant le moteur physique du moteur de jeu, épargnant du travail d'animation manuelle. Sur *Fortnite Trailer*, les personnages ont étés importés en deux parties dans le moteur afin d'exploiter la simulation physique tout en gardant un *framerate* décent. L'animation des têtes fut importée en cache *alembic* et les corps en FBX pour conserver le squelette d'animation dans le but d'y attacher des objets physiques directement dans *Unreal Engine* [17, p. 2].

Ces dernières années, les moteurs de jeu ont commencé à intégrer la gestion des principaux standards de transmission du cinéma tel que l'USD ou l'*alembic*, mais leur support est encore incomplet. Par conséquent, il est régulièrement nécessaire de passer par des formats tel que le FBX pour les intégrer aux pipelines de fabrication, compliquant ainsi leur mise en place.

L'utilisation de moteurs de jeu implique d'utiliser au moins deux types de fichiers supplémentaires au sein du pipeline de fabrication (cf. Sous-section I.2.1.2) : un format tampon (FBX par exemple) et un format de travail propre au moteur (les *game assets*). À l'échelle d'une production d'animation, cela revient presque à doubler voir tripler la quantité de fichiers à gérer. On peut alors se questionner si cet accroissement des fichiers est viable dans le cadre de productions importantes. Sur un projet utilisant le temps réel, ce *dataflow* complexifie la gestion des fichiers même sur de petits projets et nécessite à minima l'utilisation d'un VCS (cf. Sous-section I.2.1.6). L'ajout d'un tel outil au pipeline nécessite un travail d'intégration non négligeable qui peut être hors de portée pour les petits studios. C'est aussi une question complexe pour les studios ayant déjà un système de gestion de versions en place.

104. Le *framerate* ou taux de rafraîchissement désigne la fréquence à laquelle des images consécutives sont affichées ou capturées.

105. Un *solid-state drive* est un matériel informatique permettant le stockage de données sur de la mémoire flash.

Un nouveau pipeline de rendu

Si l'intégration des *game assets* représente un défi majeur en termes de *dataflow*, l'utilisation du temps réel dès le *layout* amène des changements de *workflow* importants.

David Dozoretz, alors réalisateur de *Zafari* déclare dans une interview à Cartoon-Brew [106] « Ce qui est fascinant, c'est qu'ils [les moteurs de jeu] prennent l'ensemble du pipeline d'éclairage et de rendu, le déplacent plus tôt et le raccourcissent. Je vois l'éclairage commencer à être développé alors que le *layout* et le *blocking* [107] sont en cours. » [108]. Au sein des quatre productions le moteur de jeu a joué le rôle d'un hub de collaboration entre les phases de *layout*, de *lighting* et de rendu. Dan Winn explique que l'effet stylisé d'*Ada* était composé « d'un amalgame de *post process* [109] (équivalent temps réel du *compositing*), lumières et *shaders* [110] » [111]. Le moteur de jeu leur permettait de modifier tous ces aspects en même temps.

En se tenant en parallèle, les métiers traditionnellement spécialisés dans chaque aspect du rendu fusionnent au profit de profils plus généralistes aptes à jongler entre *shading*, *lighting* et *compositing* temps réel. La limite existant jadis entre production et postproduction se retrouve bousculée, le *compositing* disparaissant presque totalement. Sur *Zafari*, David Dozoretz estime que les équipes de rendu sont réduites de presque 75 % par rapport à une production traditionnelle. Sur les quatre productions, aucun *compositing* après export n'a été nécessaire.

Une fois prêts à l'export, les plans sont rendus par le moteur de jeu en très peu de temps, chaque *frame* [112] se rend en quelques secondes (ce temps étant lié à l'écriture des fichiers). Aucune *renderfarm* [113] n'a été utilisée sur *Zafari*. Jérôme Chen, superviseur d'effets spéciaux à Sony Pictures Imageworks émet quelques réserves sur la disparition de la *renderfarm* : « Le workflow de la renderfarm ne va pas disparaître, mais avoir du temps réel dans certaines parties de votre processus va devenir une nécessité pour de nombreuses de raisons. » [114]. Il voit donc les moteurs de jeu occuper une place centrale dans la collaboration, mais pas nécessairement jusqu'au bout de la chaîne de rendu.

Une efficacité accrue ?

Lorsque le moteur de jeu est utilisé pour faire les rendus, le gain de temps de calcul est évident. Mais quelle est l'efficacité globale d'un tel pipeline ? D'un point de vue temps de production/coûts, Dan Winn estime que [4, p. 116] le coût reste probablement le même, mais que le temps de production est divisé par deux grâce au nouveau pipeline de rendu.

106. `https://www.cartoonbrew.com`
107. Le *blocking*, première étape de création d'une animation, consiste à créer les positions les plus importantes du mouvement (aussi appelées poses clefs).
108. *« What's so fascinating about this, is that they're taking the entire lighting and rendering pipeline and moving it earlier and shortening it. »* [23]
109. Les *post-process* sont des *shaders* appliqués à la fin du pipeline de rendu temps réel ; ils sont utilisés pour changer l'image dans sa globalité.
110. Un *shader* est un type de programme destiné à décrire l'aspect visuel d'un *vertex* ou d'un pixel.
111. « That was a real amalgamation of comping, lighting, and shader. » [4, p. 114]
112. Une frame est une image.
113. Une *renderfarm* ou ferme de rendu est un groupe d'ordinateurs dédiés au calcul d'images, de simulations, ou d'autres données. Définition par LePipeline.org.
114. *« The render farm workflow is not going to go away, but having real-time in parts of your process is going to become a necessity for a variety of reasons. »* [4, p. 101]

Pour avoir un ordre d'idée de l'impact des moteur de jeu sur le temps de production (du layout au rendu) d'un épisode d'animation, nous avons comparé deux projets de série au format similaire (52 épisodes de 11 minutes) : *Zafari* et *Tangranimo*.

Le Tableau I.2.2 fait états du nombre d'artistes travaillant sur les temps de fabrication étapes décrites dans le Tableau I.2.3. En prenant en compte les différences d'effectifs, on observe ainsi un temps de layout similaire. En revanche, pour l'animation on note qu'elle est plus efficace sur *Tangrimo* (15 jours / 9 personnes contre 14 jours / 24 personnes sur *Zafari*). On peut se demander si ces différences sont liées aux observations du paragraphe précédent (cf. Figure I.2.2.2) sur la complexification du pipeline d'animation. Cependant, plus de données seraient nécessaires pour confirmer cette hypothèse. En effet, de telles variations dépendent de nombreux facteurs tels que nombre de personnages par épisode, le temps passé sur l'étape de *rigging* par exemple.

Bien que le niveau qualitatif de *Tangranimo* soit plus élevé, on note que son rendu dure huit jours de plus que *Zafari*, une différence non négligeable.

	Zafari	Tangranimo
Layout	6	3
Animation	24	9
Rendu	3	4

TABLE I.2.2 – Nombre d'artiste par étape de production (en comptant les *leads*) sur les saisons 1 de *Tangranimo* et *Zafari*.

	Zafari	Tangranimo
Layout	7	12
Animation	14	15
Rendu	7	15

TABLE I.2.3 – Temps en jour par étape de production sur un épisode de 11 minute pour *Tangranimo* et *Zafari*.

Au cours de cette étude de cas, nous avons survolé les avantages et inconvénients techniques liés à l'usage d'un moteur de jeu au cœur de la fabrication de film d'animation. Mais certaines questions n'ont pas trouvé de réponses dans les cas étudiés. Si les moteurs de jeu permettent d'avoir un temps d'itération réduit, ne risque-t-on pas d'avoir beaucoup plus de *review* ? Le réalisateur deviendra-t-il un facteur limitant à cette accélération ? Finalement, quel est le processus de *review* utilisé au sein de ces productions ?

Cette section a exposé les phénomènes de convergences actuellement à l'œuvre au sein des productions audiovisuelles. Nous avons constaté que la consécration de ce rapprochement se manifestait dans la philosophie du *final pixel* utilisé principalement dans le milieu du film en prise de vues réelles.

En intégrant les moteurs de jeu comme hub collaboratif dans une production de cinéma d'animation, le pipeline de rendu débute dès le *layout*. La mise en place d'une chaîne de fabrication non linéaire du *layout* à l'export des images devient alors envisageable.

Actuellement la série et le court métrage sont les principaux formats du cinéma d'animation dans lesquels on retrouve des productions appliquant ce concept. Les flux

tendus liés aux formats et budgets des séries en font des candidats idéaux. Le pipeline de Digital Dimension pour la série *Zafari* en fut un parfait exemple, leur permettant de livrer des épisodes en un temps record.

Bien que les avantages économiques soient évidents, certains axes de cette convergence sont encore jeunes. Notre étude de cas d'utilisation de moteur de jeu dans les productions de cinéma d'animation a mis en lumière certains verrous liés à leur intégration dans les pipelines de fabrication :

— le manque d'outils de *rig* pour supporter l'animation ;

— une augmentation considérable du nombre de fichiers, complexifiant le *dataflow* ;

— la nécessité d'utiliser un VCS ;

— une gestion de production plus sophistiquée est requise ;

— l'obligation de changer le pipeline déjà en place ;

— la nécessité de recruter ou de former des gens sur le temps réel ;

— la qualité graphique des rendus temps réel peut être insuffisante pour certaines productions. [115]

Parallèlement aux moteurs de jeu, on observe que les éditeurs de DCC 3D comme *Blender*, *Maya* améliorent les capacités de visualisation de leur solution en ajoutant des moteurs de rendu temps réel (par exemple, Eevee pour *Blender*). Ces nouvelles briques technologiques ouvrent la fenêtre de l'interactivité au cœur de solutions de création déjà en place dans les productions de cinéma d'animation. Par conséquent, est-il envisageable d'utiliser ces technologies au cœur d'un workflow non linéaire ?

115. Cette limite tend à devenir caduque avec les évolutions technologiques comme l'arrivée récente de l'accélération matérielle pour faire du *raytracing* en temps réel.

I.2.3 Explorations initiales

Bien que l'utilisation de moteurs de jeu au sein de la production de films d'animation soit encore très jeune, nous avons constaté qu'elle remet profondément en question la chaîne de fabrication. Durant la première année de cette thèse, nous avons questionné l'apport du temps réel dans les méthodes de création à travers trois projets :

— *Être en apesanteur* [14] : une œuvre en réalité virtuelle sur un moteur de jeu explorant une organisation du travail non linéaire ;

— *Smartphone Remote* [12] : un outil d'assistance à la création pour *Blender* explorant les capacités interactives de son moteur de rendu temps réel ;

— *Open Pipe* : une rencontre avec des spécialistes du pipeline questionnant les méthodes actuelles (cf. Chapitre I.1).

Cette section fait état de ces différents projets et de l'impact qu'ils ont eu sur l'orientation de nos recherches de nouvelles méthodes de création temps réel pour la production d'images animées.

I.2.3.1 *Smartphone Remote*, une collaboration multi *device*

La *Smartphone Remote* trouve ses premières origines dans une conversation avec Samuel Berdou et Laure Le Sidaner. L'étincelle qui motiva sa réalisation peut se résumer en une question que nous nous posâmes en janvier 2018 autour d'une bière : « peut-on manipuler en temps réel les éléments d'une scène 3D dans *Blender* en utilisant un *smartphone* ? » À travers cette question se cachaient en réalité plusieurs problématiques d'ordre technique :

— Comment reconstruire la position du *smartphone* en trois dimensions en temps réel ?

— Comment communiquer cette information à *Blender* en temps réel ?

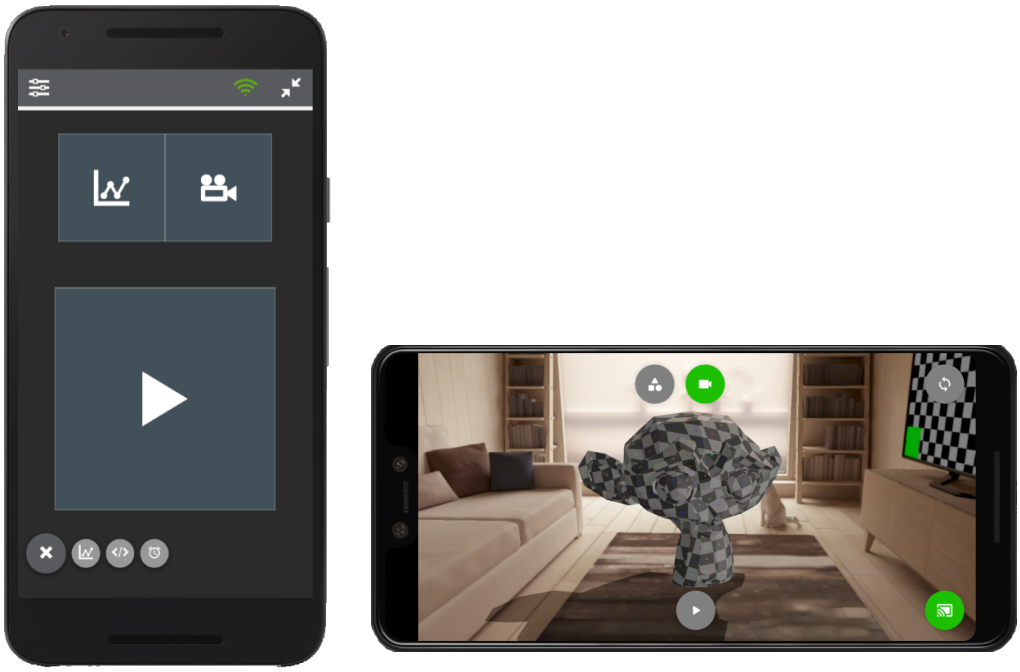
— Quelle interface graphique adopter pour placer l'utilisateur du *smartphone* au cœur de la scène ?

En répondant à ces questions à travers deux itérations logicielles, ce projet représente notre première tentative d'intégration et d'utilisation d'une brique collaborative temps réel au sein d'un DCC utilisé au cœur des chaînes de production. Durant l'année 2018, nous avons apporté une solution en deux temps, deux itérations. Au cours de cette section, nous aborderons ces deux versions en mettant l'accent sur l'architecture logicielle et réseau au cœur de la gestion des flux temps réels, garante de l'interactivité de l'outil.

FIGURE I.2.10 – Architecture de la première (a) et de la seconde (b) itération de la *Smartphone Remote*.

Le *tracking* consiste à reconstituer de la position de la caméra en trois dimensions. La *previs on set* utilise cette technique pour créer une une caméra virtuelle reproduisant le point de vue réel utilisé dans l'unité de rendu (cf. Section I.2.2). Pour reconstruire la position du téléphone dans l'espace avec précision, nous avions un problème similaire avec la possibilité d'exploiter le flux vidéo de la caméra mono objectif présente sur le *smartphone*.

Par conséquent, dans la première itération, nous avons basé le tracking optique sur un algorithme communément utilisé en *previs on set* : l'*ORB-SLAM* [16]. En prenant un flux d'images continues en entrée, l'*ORB-SLAM* construit en temps réel une représentation en trois dimensions de l'espace filmé et localise la position de la caméra au sein de cette représentation. Cet algorithme, très gourmand en ressource, ne pouvait fonctionner directement sur *smartphone*, il fut donc intégré à un programme géré depuis l'addon *Blender* pour déporter les calculs sur l'ordinateur (cf. Figure I.2.10a). Le reste de l'architecture découla directement de ce choix qui, nous le verrons dans quelques lignes, est très discutable. L'interface utilisateur, servie à travers un serveur web depuis l'*add-on*, fut développée sous la forme d'une page web statique afin de garantir une interopérabilité avec le maximum de *smartphones* existants. En procédant ainsi, les artistes n'avaient qu'à se connecter à l'adresse du serveur depuis un navigateur de leur *smartphone* pour manipuler les objets de la scène 3D.



(a) Première itération (début 2018), une page web static en HTML5 / CSS / Javascript

(b) Seconde Itération (fin 2018), une application android en Java.

FIGURE I.2.11 – Interface utilisateur des deux itérations de la *Smartphone Remote*.

Sur la première itération, nous n'avions pas moins de cinq langages de programmation utilisés : un programme en C++ pour l'*ORB-SLAM*, une interface web en JavaScript, HyperText Markup Language (HTML) [116] et Cascading Style Sheets (CSS) [117] et

116. Le HTML est le langage de balisage conçu pour représenter les pages web. Définition de Wikipédia.

117. Le Cascading Style Sheets est un langage informatique utilisé pour la mise en forme de pages

l'*add-on Blender* en Python 3 chargé de l'orchestration. La difficulté fut donc d'assurer la transmission des données entre ces acteurs du pipeline de tracking. Heureusement, un protocole standard de communication bénéficiait d'intégration dans ces différents langages : WebSocket [118]. Comme on le constate dans la Figure I.2.10a, tout le pipeline de transmission des données s'opère en WebSocket : les images de la caméra sont récupérées depuis l'interface web, envoyées au programme *ORB-SLAM* qui calcule les poses de caméra et les envoie sous leur forme matricielle à l'*add-on* dans *Blender* qui les applique dans la scène. À l'utilisation, nous avons constaté deux défauts importants à cette première itération :

— la transmission des images entre le *smartphone* et le programme d'*ORB-SLAM* est lourde et induit une latence non négligeable ;

— l'envoi des images à haute fréquence draine rapidement la batterie du *smarthpone* ;

— l'interface web (cf. Figure I.2.11a) n'affichait aucun retour, l'artiste devait regarder son écran d'ordinateur durant la manipulation du *smartphone* pour visualiser son mouvement.

Courant 2018, deux nouveaux *frameworks* [119] dédiés à la réalité augmentée sur *smartphone* ont été publiés : *ARCore* [120] pour les systèmes Android et *ARKit* [121] pour les systèmes iOS. Ces derniers ont l'avantage de faire le tracking directement sur *smartphone* avec un algorithme tirant parti de la fusion de capteurs (ils exploitent simultanément la centrale inertielle [122] et le flux de la caméra). Cela motiva la création d'une seconde itération. *ARCore* nous permit de faire d'une pierre deux coups : déporter le tracking directement sur le *smartphone* et y visualiser la scène en réalité augmentée. Bien que l'utilisation de ces *frameworks* ait nécessité le développement d'une application Android pour le client, on peut observer sur la Figure I.2.10b que le pipeline de transmission des données en est ressorti grandement simplifié : le client Android calcule localement la position de la caméra et envoie directement sa matrice de pose en Transmission Control Protocol (TCP) [123] avec *ZeroMQ* [124], une bibliothèque de fonctions réseau open source et multiplateforme. En supprimant la transmission des images de l'équation, les problèmes de latence ont disparu. Dans la seconde itération, l'artiste scanne un QR Code avec l'application Android pour se connecter à son instance de *Blender*. Au moment de la connexion, la scène 3D est transmise au *smartphone* en GLTF pour y être affichée en réalité augmentée. Le GLTF nous a permis de transmettre les lumières, les *meshes*, les matériaux et l'animation.

Initialement, cette application devait être un simple outil à destination de l'artiste souhaitant utiliser un contrôleur spatialisé en trois dimensions pour manipuler des objets virtuels, mais nous avons rapidement découvert que l'ajout de la réalité augmentée

HTML

118. Spécification du standard WebSocket : `https://datatracker.ietf.org/doc/html/rfc6455`

119. Un *framework* est un ensemble de composants logiciels structurés facilitant le développement de programmes en fournissant un niveau d'abstraction plus élevé.

120. `https://developers.google.com/ar/`

121. `https://developer.apple.com/augmented-reality/arkit`

122. Une centrale à inertie ou centrale inertielle est un instrument capable d'intégrer les mouvements d'un mobile pour estimer son orientation (angles de roulis, de tangage et de cap), sa vitesse linéaire et sa position. Définition par Wikipédia.

123. Le protocole de contrôle de transmissions ou *transmission control protocol* est un protocole de transport de données.

124. `https://zeromq.org`

désolidarisait l'artiste de sa machine de travail. Ainsi, lorsqu'un artiste cherche des cadrages depuis le smartphone, une seconde personne pouvait ajuster des éléments de décors selon les besoins de composition en s'installant sur le poste libéré. Des membres de la communauté de *Blender* ont aussi essayé de l'utiliser pour faire de l'extension de set sur un plateau [125].

En entrouvrant une porte collaborative au sein de *Blender*, ce projet a confirmé que nous pouvions explorer l'impact du temps réel dans la création d'images animées ailleurs que dans un moteur de jeu : directement dans les DCC utilisés en production. Tout le projet est disponible en ligne sous licence open source [126].

I.2.3.2 Le Corps Infini, une collaboration humaine et interdisciplinaire pour la création d'une narration en réalité virtuelle

En 2018 ainsi qu'en 2019, j'ai eu la chance de participer aux éditions #2 et #3 du projet de recherche Corps Infini [127].

Au croisement des arts numériques, de la danse et du cirque, ce projet interroge la possibilité d'un espace en trois dimensions qui recrée les conditions de l'apesanteur. En immergeant le public dans un espace visuel, sonore et virtuel, il opère un basculement de la perception du corps et de l'environnement.

Notre travail sur ce programme consista à développer une œuvre de réalité virtuelle plongeant le public du point de vue empathique des circassiens (cf. Figure I.2.12).

FIGURE I.2.12 – Image de l'œuvre en Réalité Virtuelle (RV) : *Être en apesanteur*, développée dans le cadre du projet *Corps Infini*.

Dans cette section, notre intérêt se portera principalement sur les méthodes de collaboration à l'origine de l'expérience. Nous débuterons par une description du terreau propice à l'émergence de ces méthodes : le plateau expérimental. Puis nous analyserons

125. Vidéo d'utilisation de la *Smartphone Remote* pour de l'extension de set : `https://youtu.be/0CHxEdtC_sA`
126. Sources de l'outil : `https://gitlab.com/slumber/smartphoneremote`
127. Projet soutenu par le Labex Arts-H2H sous la direction de Kitsou Dubois et la participation de l'INREV. `http://www.lecorpsinfini.labex-arts-h2h.fr`

les processus collaboratifs qui y ont germé.

Le plateau expérimental composé d'une dizaine de personnes à plein temps (l'équivalent d'un petit studio d'animation) prit place dans les locaux de l'Académie Fratellini [128]. Au sein de cet organisme, les artistes circassiens, musiciens, cinéastes et numériques oeuvraient de concert à la construction d'une œuvre vivante, à cheval entre réel et virtuel.

D'un point de vue structurel, les artistes formaient de petites équipes par unité de recherche (cf. Figure I.2.13) coordonnées par Kitsou Dubois, chorégraphe et porteuse du projet. On retrouve ainsi :

— une équipe pour le son (Département Musique de l'Université Paris 8) ;

— une équipe de captation vidéo (École Louis Lumière) ;

— une équipe de circassiens (Académie Fratellini) ;

— une équipe de réalité virtuelle (Équipe Image Numérique et Réalité Virtuelle (INREV) [129] de l'Université Partis 8).

Ces équipes avaient la particularité de travailler intégralement en parallèle. Chacune d'entre elles fonctionnait indépendamment sur l'un des aspects du projet (Son, Captation, etc.). Kitsou, garante de la vision globale du projet, passait régulièrement valider ou orienter le travail des équipes d'étudiants, doctorants et professeurs.

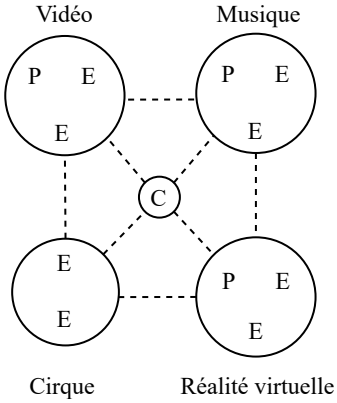


FIGURE I.2.13 – Organigramme du projet *Corps Infini* montrant la relation entre les élèves (E), les professeurs (P) et la chorégraphe Kitsou (C).

Un aspect particulièrement intéressant de ce projet réside dans l'organisation du travail en cycle hebdomadaire. Chaque début de semaine, un ensemble d'objectifs était établi collégialement pour les différentes équipes. Par exemple, dans notre équipe de réalité virtuelle, l'une des étapes initiales fût de produire une simulation de fluide interactive (visible sur la Figure I.2.12). Durant la semaine, les professeurs des différentes équipes suivaient l'avancement des étudiants. En fin de semaine ou en début de semaine suivante, une présentation globale était faite avec toutes les équipes afin que chacun soit au courant du travail de tous. C'était également une phase de validation critique où Kitsou approuvait ou réorientait le travail des différentes unités.

La durée de ce processus cyclique et itératif s'ajustait en fonction des objectifs. Par exemple, lorsque les principales briques techniques de l'expérience en RV furent

128. L'Académie Fratellini est un centre de formation qui délivre le Diplôme national supérieur professionnel d'artiste de cirque. `https://www.academie-fratellini.com`

129. `https://inrev.univ-paris8.fr`

en place, nous avons fait des cycles d'une heure avec les circassiennes afin d'ajuster l'expérience au plus proche de leur vécu. L'utilisation d'un moteur de jeu nous a permis d'obtenir des cycles d'itérations extrêmement courts puisqu'aucun temps de calcul ne causait de délais entre deux simulations. Ce constat appuie donc les observations relevées en Sous-section I.2.2.2. On note ici qu'à l'inverse d'une chaîne de fabrication en animation, les échanges ne passaient pas tous par une seule personne (ici Kitsou). La communication était horizontale entre les équipes. La nature des relations variait selon le besoin des équipes. De notre point de vue (équipe RV), l'équipe des circassiens nourrissait notre travail de leur vécu et des nombreux retours qu'ils faisaient de notre installation. L'équipe musique nous fournissait différentes pistes audio, nous étions donc leurs clients. Enfin, Kistou validait l'avancement général de notre travail.

À l'opposé de la chaîne de fabrication décrite dans la Section I.1.1, l'expérience du *Corps Infini* s'appuie sur une collaboration organique, horizontale qui favorise le travail en groupe (unité de recherche) et non individuel. Les présentations globales de suivis apportaient aux équipes une vision du projet en cours d'avancement. La découverte de cette méthodologie au cours du projet *Corps Infini* nous conduisit à interroger si une telle organisation était applicable dans une production de cinéma d'animation. Placer le groupe au cœur de la création d'images animées améliorerait-il la vision globale du projet au sein des artistes ?

I.2.3.3 *Open Pipe* : réflexions avec des vétérans du pipeline

Entre décembre 2018 et février 2019, j'ai eu la chance de participer à deux sessions du groupe de travail *Open Pipe*. Nous avons abordé de nombreux sujets tels que la synthétisation des données de suivis de production, la souplesse d'un pipeline au sein d'un studio de série d'animation ou encore des présentations d'outils de pipeline tel que *Kabaret Studio*[130].

Certains constats analysés durant ces rencontres font écho avec des observations du Chapitre I.1. Par exemple, nous avons relevé une observation de Pierrot Jacquet alors directeur de production à Ubisoft Animation Studio :

> *« Aujourd'hui l'animation suit les préceptes du fordisme avec un compartimentage des métiers qui freine la création artistique, le financement et les clients formatent la production. »*

En spécialisant de plus en plus les métiers dans la chaîne de fabrication, nous avons mis des limites strictes à la créativité. Durant la phase de production, lorsqu'une tâche arrive entre les mains d'un artiste, il n'a le droit d'influer qu'un aspect très spécifique de l'œuvre. Or, dans une image, tous les aspects sont connectés, la modélisation d'un *props* influencera le *shading*, qui influencera le comportement de la lumière. Par conséquent, nous avons questionné l'efficacité des méthodes actuelles :

> *« Une spécialisation en département est peut-être moins efficace qu'un compartimentage artistique »*

Ici, le compartimentage artistique désignerait une segmentation des artistes en groupes pluridisciplinaires qui travailleraient de concert à la création d'un shot par exemple. À l'opposé de la spécialisation en départements qui scindera la création de ce même shot à travers plusieurs départements. Au final, le constat a été fait qu'il « *serait formidable d'essayer de mettre en avant le groupe artistique au lieu du département spécialisé.* » Mais dans cette situation, comment aborder la répartition du travail ?

130. `https://www.kabaretstudio.com`.

Pierrot suggéra une piste intéressante : utiliser un système de scores. En laissant les artistes choisir leur tâche en fonction de scores qui leur sont attribués durant leur création. Ces scores fourniraient aux artistes un indicateur précis de la difficulté de la tâche.
Une organisation du travail en groupe pluridisciplinaire privilégierait-elle une généralisation des compétences des artistes ?

Ces interrogations nous conduisirent à envisager des pistes pour pallier le manque de communication entre départements exposé dans le Chapitre I.1. En tissant de nouveaux canaux de collaboration temps réel entre les étapes de fabrication, deviendrait-il possible de faire travailler ensemble des artistes issus de différents départements ? Si oui, comment établir ces liens de communication et à quel niveau opérer ? Ces rencontres avec des professionnels du pipeline en début de thèse ont joué un rôle crucial dans l'orientation de nos travaux de recherches. Les sujets abordés nous ont conduits à beaucoup d'interrogations qui précisèrent certains de nos axes de recherches.

Conclusion

Au début de ce chapitre, nous avons défriché le point de fusion entre l'univers du jeu vidéo et du cinéma. Nous avons ensuite précisé l'apport de cette brique technologique au regard du milieu plus restreint du cinéma d'animation en nous basant sur une étude de quatre cas. L'utilisation de moteur de jeu au sein des chaînes de fabrication représente un nouveau hub de collaboration qui intervient dès le *layout*. Ce dernier réduit les coûts de l'étape de rendu en diminuant les effectifs et en accélérant les itérations. Cependant, l'intégration d'un moteur de jeu au pipeline complexifie le *dataflow* en ajoutant un niveau de données et requiert le développement de passerelles complexes avec les DCCs d'animation. Bien qu'ils se développent dans cette direction, les moteurs de jeu ne sont pas encore assez matures pour contenir intégralement l'étape d'animation. Si ces moteurs de jeu ne supportent pas la conception des *assets* graphiques, est-il possible d'avoir un *workflow* non linéaire directement au sein des DCC utilisés pour leur fabrication ?

La *Smartphone Remote* nous a permis de confirmer que l'affichage et l'interactivité en temps réel ne se bornait pas aux moteurs de jeu. En développant un pipeline de communication réseau à faible latence entre le *smartphone* et *Blender*, nous avons créé un outil d'animation de caméra en temps réel directement dans le DCC. Cette expérience valide ainsi techniquement nos interrogations quant au support du traitement et de l'interprétation de flux de données temps réel au sein de *Blender*, un DCC utilisé dans la chaîne de fabrication de film d'animation. En revanche, en se bornant à la manipulation d'éléments 3D, elle n'aborde pas les problématiques de collaboration pluridisciplinaire.

En apportant un *workflow* non linéaire, les technologies temps réel induisent une complexification de la coordination des artistes. Les exemples de notre étude de cas (cf. Sous-section I.2.2.2) laissent beaucoup d'interrogations quant à l'évolution du processus collaboratif qui encadre l'utilisation de ces méthodes de créations émergentes. Comment organiser l'assignation et la validation du travail au sein d'une production non linéaire ? Au cours de la création de l'œuvre *Être en apesanteur*, au sein du plateau expérimental du projet de recherche le *Corps Infini*, nous avons ouvert les yeux sur un processus de création organique extrêmement réactif de par ses cycles d'itération courts. Ce processus de collaboration proche des méthodes agiles (cf. Infra Section II.4.1) apporte une première piste de méthodes de collaboration qui pourraient être employées pour améliorer la communication au sein des équipes de création exploitant le temps réel.

Notre expérience au sein des réunions du collectif Open Pipe fut un carrefour dans nos recherches. En confirmant nos observations sur le manque de communication lors de la fabrication de films, les échanges avec des séniors du milieu ont fait converger nos recherches vers un cap plus précis : l'exploration du temps réel comme outil de collaboration.

Chapitre I.3

Enjeux de la co-création temps réel pour l'image de synthèse

Introduction

Au début du chapitre précédent, il a été observé que l'usage des moteurs de jeu permettait d'accélérer les productions traditionnellement linéaires en réduisant les temps d'itération entre chaque étape. Mais il s'agit là d'une vision microscopique (par tâche/-solution) des bénéfices du rendu temps réel pour la production de films d'animation. En regardant à l'échelle macroscopique, on peut se demander quel serait l'apport de systèmes temps réel au *workflow* global ? Pourrait-il amener de nouvelles façons de travailler ensemble donnant plus de visibilité entre les départements tout au long de la production ?

> *Que se passerait-il si plusieurs artistes étaient amenés à travailler ensemble en même temps sur une même scène ?*

Telle fut la question qui émergea naturellement suite à toutes les réflexions du chapitre précédent en novembre 2018. Ainsi, nous allions essayer d'exploiter le temps réel comme support de collaboration, de co-création pour le cinéma d'animation.

Avant d'aller plus loin dans notre exposé, il est important de délimiter le sens qui est derrière les termes collaboration ou co-création en temps réel dans nos travaux.

> *Selon nous, une expérience de collaboration ou de co-création en temps réel en cinéma d'animation consiste à amener plusieurs artistes à travailler dans le même espace virtuel en même temps.*

Au début de cette thèse, la co-création en temps réel était très peu explorée dans le cinéma d'animation. Nous débuterons donc par exposer l'usage de cette technique dans d'autres disciplines. Ces exemples, matures de plusieurs années dans leurs secteurs respectifs, furent une source d'inspiration non négligeable pour nos expérimentations. Dans un second temps, nous établirons une fresque temporelle catégorisant les solutions existantes dans notre secteur en y situant nos travaux présentés en seconde partie.

I.3.1 Co-construction temps-réel dans d'autres disciplines

Si la collaboration en temps réel est un concept récent dans la création d'images de synthèse, elle est largement exploitée dans d'autre secteurs. Étudier ses usages dans d'autres domaines nous a aidé à comprendre à la fois ses bénéfices et les problématiques qui y sont associés.

Dans cette section, nous étudierons l'usage et l'impact de ces méthodes dans l'industrie du génie civil et de l'informatique.

I.3.1.1 Dans le Bâtiment et Travaux Publics

Le processus de construction d'un bâtiment partage certaines analogies avec la fabrication d'un film exposée dans le Chapitre I.1. La Figure I.3.1 décrit le *workflow* de construction d'un bâtiment. Similairement au cinéma d'animation, il se compose d'une suite d'étapes linéaires et interdépendantes menées par des corps de métiers spécialisés éclatés dans différentes entreprises (exemple, une agence d'architectes). Par conséquent, le milieu du Bâtiment et Travaux Publics (BTP) [131] fait face aux mêmes problématiques communicationnelles (silos, mauvaise anticipation des erreurs, etc.) que le cinéma d'animation. Cependant, de par son ancienneté, ce secteur apporte des solutions plus avancées à ce problème parmi lesquelles on retrouve *Building Information Modeling* (BIM) [132].

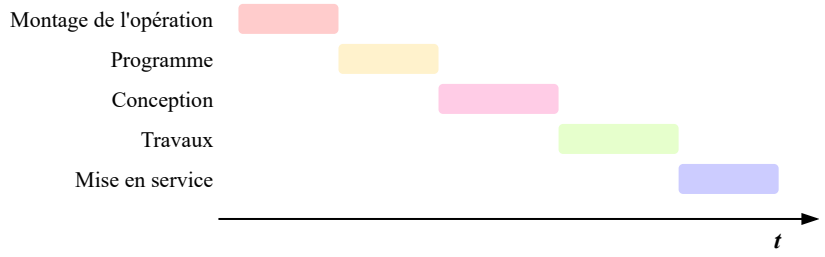


Figure I.3.1 – *Workflow* de construction dans le BTP.

Selon Wikipédia, le BIM est né à partir du travail mené par quelques entreprises du bâtiment en 1995 dans le but de faciliter l'interopérabilité des applications utilisées entre les différents corps de métiers. Il est défini dans les normes internationales comme :

> *L'utilisation d'une représentation numérique partagée d'un actif bâti pour faciliter les processus de conception, de construction et d'exploitation de manière à constituer une base fiable permettant les prises de décision. Actifs bâtis = bâtiments, ponts, routes, tunnels, voies de chemin de fer, usines, etc.* (ISO 19650-1:2018)

Ce terme désigne des processus de collaboration supportés par un format d'échange informatique. Actuellement, il existe plusieurs formats d'échange, certains propriétaires

131. Le secteur économique du bâtiment et des travaux publics, ou BTP, regroupe toutes les activités de conception et de construction des bâtiments publics et privés, industriels ou non, et des infrastructures telles que les routes ou les canalisations. Définition de Wikipedia

132. *Building information modeling* est un processus soutenu par divers outils, technologies et contrats impliquant la génération et la gestion de représentations numériques des caractéristiques physiques et fonctionnelles des lieux. Les *Building information models* sont des fichiers informatiques qui peuvent être extraits, échangés ou mis en réseau pour soutenir la prise de décision concernant un bien construit.

et d'autres ouverts. Ce manque de consensus est l'un des freins à l'adoption de BIM. Contrairement au secteur du cinéma d'animation, les entreprises du BTP ont peu de développement informatique en interne et dépendent donc des solutions fournies par les éditeurs de logiciels propriétaires. Cependant, pour permettre l'émergence d'un standard neutre, plusieurs sociétés mirent en commun leur effort à travers la création d'une organisation à but non lucratif : BuildingSmart [133]. Aujourd'hui, cet organisme maintient deux formats d'échange ouverts complémentaires pour supporter le BIM :

— Industry Foundation Classes (IFC) [134] ;

— BIM Collaboration Format (BFC) [135].

Le format IFC supporte une grande partie des aspects de la construction. Par analogie, il est au BTP ce qu'est l'USD au cinéma d'animation. Cependant, l'IFC existe depuis 1997, il est plus mature que son homologue et intégré dans plus d'une centaine de logiciels [136].

Les éditeurs logiciels de BTP l'utilisent au cœur du *dataflow* de leurs solutions *cloud* [137] (par exemple, BIM 360 pour Autodesk ou BIMCloud pour Graphisoft). Ces plateformes se transforment alors en hub de collaboration interdisciplinaire où chaque responsable de la construction (architecte, ingénieur et entrepreneur) partage en temps réel un état commun du bâtiment en construction à travers des applications telles que *Revit* [138]. Ainsi, lorsqu'un architecte déplace une fenêtre, ce changement sera répercuté directement dans le modèle 3D utilisé par les ingénieurs structures. Grâce à ce nouveau moyen de communication, les différents corps de métiers mesurent l'impact de leur travail sur celui des autres.

En partageant une vision commune du projet en construction, BIM apporte beaucoup d'avantages :

— une meilleure prise de décision : les acteurs de la fabrication sont amenés à prendre des décisions collégialement et non individuellement tout au long de la conception et de l'exécution de la construction ;

— une anticipation des erreurs accrue : des outils de détection de conflits entre les schémas fournis par les différentes disciplines permettent de résoudre les problèmes avant qu'ils ne se produisent sur le terrain ;

— une réduction des coûts : l'amélioration de la coordination des équipes sur le terrain évite les erreurs de construction qui coûtent cher en argent et en temps ;

— une pérennisation des informations : en se basant sur un format « standard », les données de construction sont facilement réexploitables tout au long de la durée

133. `https://www.buildingsmart.org/`

134. Le format Industry Foundation Classes est un format de fichier utilisé dans le BTP pour échanger et partager des informations entre logiciels. Il permet de décrire des objets (murs, poteaux, etc.), leurs caractéristiques et leurs relations. Les IFC ont pour but d'assurer l'interopérabilité entre les logiciels de conception assistée par ordinateur et les logiciels d'ingénierie. Ce format est décrit dans la norme ISO 16739.

135. Le *BIM Collaboration Format* est un format de note pour IFC. Les fichiers BCF, permettent d'échanger des commentaires, éventuellement accompagnés d'une vue, ou d'une sélection d'éléments de la maquette IFC. Ces fichiers contiennent aussi toutes les informations de traçabilité : auteur, date, version.

136. Liste des implémentations : `https://technical.buildingsmart.org/resources/software-implementations`

137. Le *cloud* est un modèle informatique où les traitements traditionnellement effectués sur des serveurs locaux ou sur des postes clients sont externalisés sur des serveurs distants. Définition par Antidote.

138. `https://www.autodesk.fr/products/revit`

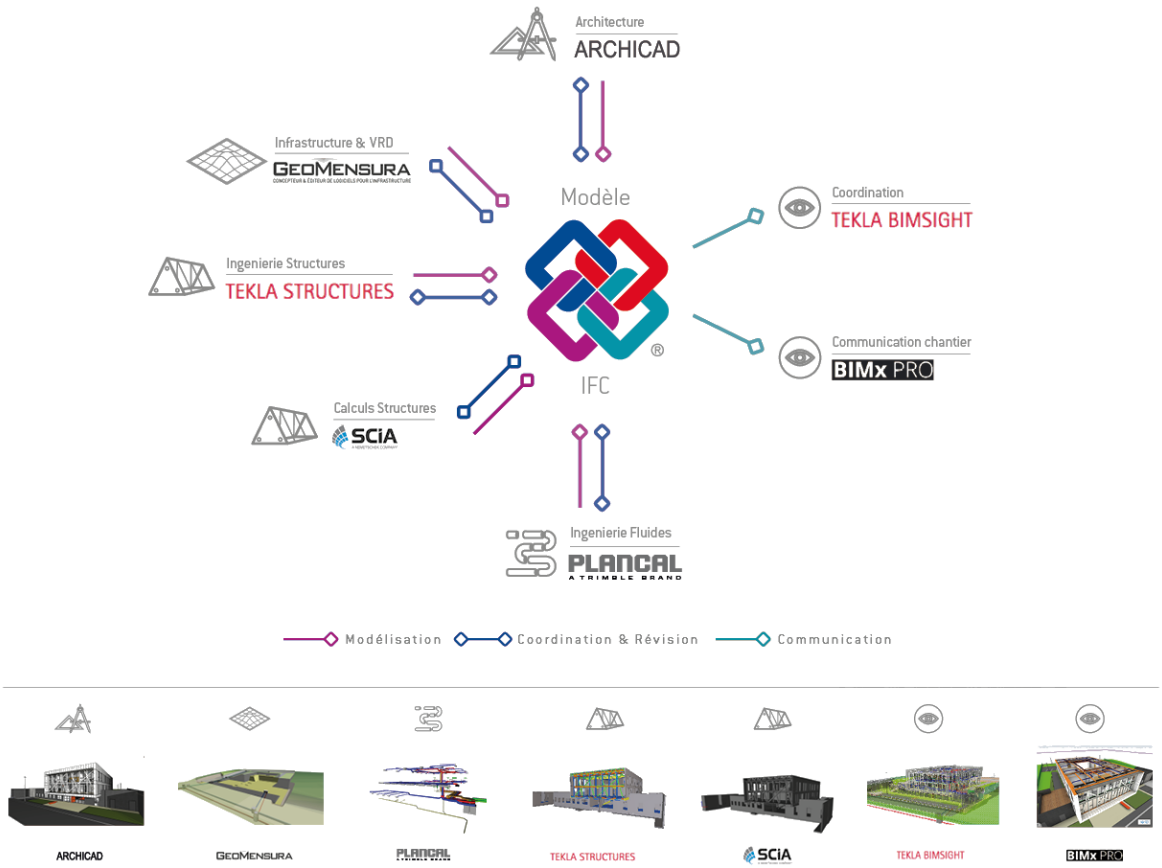


FIGURE I.3.2 – Exemple d'utilisation de BIM en se basant sur l'échange de fichiers IFC dans différents corps de métiers du BTP. Source : `https://openbim.fr`

de vie du bâtiment construit (lors de la phase d'entretien et même lors de la démolition).

En plus de 20 ans de développement, ce standard est arrivé à un stade de maturité où il est enseigné dans les écoles d'architecture [139]. Un métier a même émergé autour de sa pratique : BIM Manager [140]. Le BIM Manager est responsable de la centralisation des données des différents corps de métiers dans le BIM, il est ainsi garant de la modélisation centrale du projet.

À l'heure actuelle, l'adoption de BIM dans les projets de construction est freinée par trois aspects :

— l'infrastructure logicielle propriétaire freine l'établissement d'un standard ;

— la méthode ne peut être adoptée dans un projet que si les entreprises participantes à la construction y sont compatibles (l'intègrent dans leur processus) ;

— passer par un format d'échange visant l'interopérabilité (comme l'IFC) implique de supporter seulement une partie des données liées à chaque solution (et donc à chaque corps de métier). Un tri est fait au moment de la mise à disposition du travail, limitant la collaboration aux aspects supportés par le format.

139. Par exemple, BIM est enseigné dans le Master parcours Design Numérique Architecture de l'école nationale supérieure d'architecture de Nancy : `http://www.nancy.archi.fr/fr/specialite-ame.html`

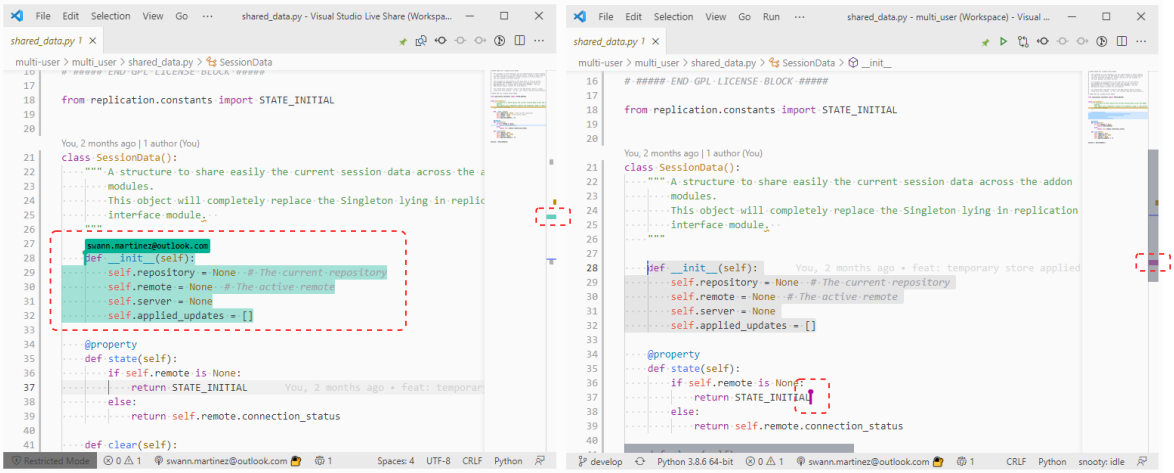
140. Fiche de description du métier de BIM Manager : `https://www.onisep.fr/Ressources/Univers-Metier/Metiers/bim-manager`

À travers l'exploration de BIM, nous avons vu l'utilité d'un outil de collaboration interdisciplinaire utilisé au cœur d'un processus de production. En centralisant et superposant les informations des différents corps de métiers, il fournit à tous les acteurs une vision globale de l'avancement du projet en temps réel. Cet ensemble d'outils et de méthodes crée de nouveaux canaux de communication entre les corps de métier qui composent toute la chaîne de construction. Serait-il envisageable d'appliquer un concept similaire à notre secteur en superposant les travaux des différents départements au même endroit tout en gardant une infrastructure légère ?

I.3.1.2 Dans le développement informatique

En phase avec le progrès technologique, le milieu de l'informatique est en constante évolution. Le développement logiciel ne coupe pas à la règle. Au fil des années, les méthodes de conception logicielle ont évolué pour gagner en souplesse et permettre l'adaptation du produit développé. Au sortir de cette mue, un ensemble de pratiques collaboratives est apparu : les méthodes agiles (elles seront étudiées plus en détail en Section II.4.1). Dans cette section, nous nous intéresserons principalement à l'une de ces techniques : la programmation en binôme (*pair programming*[141]).

Historiquement, une session de *pair programming* consiste à faire travailler deux développeurs sur le même poste de travail. Le conducteur écrit le code tandis que l'observateur évalue chacune des lignes. Mais l'évolution des outils de développement a modifié cette configuration. L'arrivée d'outils tels que *Teletype*[142] en 2018 a permis aux développeurs de se connecter en réseau depuis leur éditeur à la même *codebase*[143] et d'y apporter des modifications simultanément en temps réel.



Développeur conducteur — Développeur observateur

FIGURE I.3.3 – Pair programming en temps réel entre deux développeurs sur l'éditeur *VSCode*.

141. Le *pair programming* (ou programmation en binôme), est une méthode de travail dans laquelle deux développeurs travaillent ensemble sur un même poste de travail. La personne qui rédige le code est appelée conducteur (*driver*). La seconde personne, appelée observateur (*observer*), assiste le conducteur en décelant les imperfections, en vérifiant que le code implémente correctement le design et en suggérant des alternatives de développement. Définition par Wikipédia.
142. `https://teletype.atom.io/`
143. La *codebase* désigne l'ensemble du code source d'un logiciel.

Expérience multi-utilisateur dans un programme non conscient de la collaboration.

La Figure I.3.3 illustre une session de *pair programming* entre deux développeurs en réseau sur l'éditeur *VSCode*[144] avec le *plug-in LiveShare*[145] (un outil similaire à *Teletype*).

Du point de vue de l'architecture, *VSCode* est un éditeur initialement mono-utilisateur. Lawer et al. qualifient ces logiciels comme étant non conscients de la collaboration (ou *collaboration-transparent programs*) [11]. L'extension *LiveShare* ajoute un ensemble de fonctionnalités pour supporter la collaboration.

Sur la Figure I.3.3, on peut observer les utilisateurs connectés à la session et leur localisation dans le fichier à travers la représentation de leur curseur et d'une marque dans la barre de défilement (encadré en rouge pointillé). Ces éléments graphiques représentant les collaborateurs dans la session sont qualifiés d'interfaces conscientes de la collaboration.

LiveShare expose également un ensemble d'outils pour aider les développeurs à se coordonner. Par exemple, un utilisateur peut se verrouiller sur le curseur d'un participant pour partager le même point de vue et suivre ses déplacements dans le code. On qualifie ces instruments collaboratifs de fonctions conscientes de la collaboration.

Notre méthode de création collaborative exposée en partie II s'est largement inspirée de cette architecture car les DCC existants sont également des *collaboration-transparent programs*.

Apports de la méthode

Durant les sessions de programmation en binôme, l'observateur apporte un point de vue critique sur le code produit par le conducteur. Cela permet de détecter un certain nombre de bugs que le conducteur ne remarque pas. En ce sens le *pair programming* améliore la qualité du code écrit en apportant une *review* en temps réel[146].

La communication représente l'essence du *pair programming*. C'est une activité sociale qui développe des liens entre les employés. Cockburn et al. [7, p. 8] expliquent qu'il contribue au *team building*[147] en apprenant aux équipes à travailler ensemble. À l'issue de plusieurs sessions, les employés communiquent plus facilement sur les problèmes et solutions qu'ils rencontrent.

Une autre force de cette méthode réside dans ses capacités liées à l'apprentissage. En entreprise, il accélère le transfert de connaissances entre les développeurs de différents niveaux d'expérience. Le novice se familiarisera plus rapidement à la *codebase* et apprendra les bonnes pratiques grâce à l'expert. À titre d'exemple, à l'arrivée de Fabian Adam sur le développement de l'*add-on* Multiuser (cf. Chapitre II.1), nous avons utilisé le *pair programming* pour le familiariser avec l'organisation du code de l'outil. Dans le milieu universitaire, le *pair programming* est utilisé dans l'enseignement de la programmation informatique. En fonctionnant par binôme durant les cours, les étudiants apprennent mieux et plus rapidement les éléments du cours.

144. `https://code.visualstudio.com/`
145. `https://visualstudio.microsoft.com/services/live-share/`
146. Cf. à l'étude menée par Dyba et al. [8, p. 15].
147. Le *team building* ou renforcement d'équipe, est une méthode qui vise à resserrer des liens sociaux au sein d'un groupe d'individus appartenant à une entreprise ou à une institution.

Les spécificités des solutions de développement collaboratif furent un point de départ important dans notre processus de recherche et développement d'outils dédiés à la mise en place d'une interface multi-utilisateur. Ainsi, le *pair programming* améliore la communication en tissant de nouveaux liens sociaux au sein des équipes, et conduit à un code de meilleure qualité et à une transmission de savoir naturelle. La co-création en temps réel aura-t-elle des apports similaires pour le cinéma d'animation ?

Dans cette section, nous avons parcouru deux cas emblématiques d'utilisation de la collaboration en temps réel au service de la production. Ces explorations initiales ont dégagé beaucoup de pistes pour nous guider dans l'application de la collaboration en temps réel au sein de notre propre discipline. Pour le BTP, nous avons constaté qu'elle était exploitée pour donner une vision commune du projet en cours aux principaux acteurs (équivalents aux départements) de la construction durant tout le cycle de vie du bâti. Est-il envisageable de construire un *workflow* similaire dans la fabrication de film d'animation ? En conception logicielle, le *pair programming* intervient au sein d'un métier spécifique : le développement. Il fournit aux programmeurs une méthode pour collaborer sur la même *codebase*. En ajoutant l'aspect social au cœur du développement, cette technique améliore l'anticipation des erreurs et le transfert de connaissances. Elle trouve donc une place de choix dans les méthodes d'apprentissage de programmation informatique utilisées en étude supérieure. Est-ce que ce concept est transposable au cinéma d'animation pour l'enseignement de la création d'images de synthèse ? Enfin, Pourrait-on imaginer plusieurs artistes travailler en binôme sur certains aspects de la fabrication ?

I.3.2 Collaboration en temps réel pour l'image de synthèse

Ainsi, la collaboration en temps réel n'est pas étrangère au BTP et au développement informatique qui l'exploitent à des fins similaires : améliorer la communication au sein des travailleurs. Ces cas d'usage ont montré les potentiels apports de telles techniques pour les acteurs de la production.

En cinéma d'animation, l'avènement de technologies de rendu temps réel au sein des DCC ouvre de nouvelles portes collaboratives. Au début de cette thèse, le concept de co-création en temps réel était encore embryonnaire et non sans raison. Nous commencerons par exposer les différents enjeux techniques liés à l'utilisation de systèmes collaboratifs temps réel dans notre secteur. Cela nous permettra de catégoriser les approches existantes exposées ensuite tout en positionnant nos travaux présentés en partie II.

I.3.2.1 Enjeux techniques et humains

Amener la collaboration temps-réel au cœur des applications de création dans le cinéma d'animation est un problème complexe. D'un point de vue technique, nous avons vu au cours du Chapitre I.1 que les scènes 3D d'une production sont constituées d'une grande variété de données volumineuses de création (*meshes*, *shaders*, *rigs*, textures, etc.) qui peuvent être spécifiques à chaque logiciel. Répliquer ces informations avec une réactivité suffisante pour garantir une interaction en temps réel représente un défi majeur. Humainement, en structurant verticalement la chaîne de communication (cf. Sous-section I.1.2.2) et en basant le processus de validation sur des travaux individuels (cf. Sous-section I.1.2.3) la chaîne de fabrication actuelle ne se prête pas ou peu à la collaboration multi-utilisateur.

En développement logiciel, on distingue deux stratégies principales pour implémenter une expérience de collaboration en temps réel :

1. programme conscient de la collaboration : développer un logiciel un conçu spécifiquement pour cet usage ;
2. programme transparent à la collaboration (ou non conscient de la collaboration) : étendre les fonctionnalités d'un logiciel conçu initialement pour une expérience mono-utilisateur à travers un *plug-in* (cf. Section).

Appliquée au cinéma d'animation, la première stratégie équivaut à créer un nouveau DCC dédié. En plus de développer le système qui supportera la collaboration, il faudrait concevoir tous les outils de création. Une telle approche à l'avantage du contrôle total des données gérées, ce qui permet d'atteindre de meilleures performances. Mais le coût de cette stratégie est très élevé, car la solution développée ne repose pas sur de l'existant. Par exemple, en imaginant une telle solution pour de la modélisation, il serait nécessaire de créer tous les outils de modélisation en partant de zéro, un travail de plusieurs années. Une telle approche éprouvera également des difficultés à être adoptée dans les chaînes de fabrication, car de la même façon que les moteurs de jeu, elle ajoute un maillon complexe à intégrer dans le pipeline en place.

La seconde stratégie, similaire à l'approche utilisé dans *VSCode* pour le *pair programming* (cf. Figure I.3.1.2) revient à créer un *plug-in* pour un logiciel de création déjà existant en utilisant son API. Les performances et fonctionnalités collaboratives

de cette approche dépendent directement de l'architecture du logiciel ciblé. En revanche, cette méthode permet de gagner énormément en temps de développement, car seule la brique collaborative est développée. Si la solution 3D est déjà en place dans un studio, l'intégration d'un *plug-in* dans son pipeline en sera facilitée.

La seconde méthode est davantage en phase avec notre problématique d'intégration du processus collaboratif au cœur de la chaîne de fabrication.

Les travaux exploratoires présentés en seconde partie de cette thèse sont nés de nombreuses sessions de création collaboratives (cf. supra Chapitre II.2).

Au fil de ces expériences, nous avons identifié différents besoins concrets nécessaires à l'établissement d'une collaboration multi-utilisateur efficace au sein d'un logiciel de création 3D non-conscient de la collaboration :

— indépendance face à l'interface utilisateur : ne pas interférer dans les interfaces du DCC, laisser les utilisateurs l'utiliser comme il est prévu pour fonctionner ;

— édition synchrone de la scène : permettre à différents utilisateurs d'éditer différentes parties et aspects de la scène simultanément et sans conflits ;

— indépendance des erreurs : une erreur causée par un artiste localement ne doit pas influencer les autres utilisateurs connectés ;

— flexibilité d'intégration : laisser au programme de création le choix de l'endroit d'exécution des fonctions de collaboration ;

— *dataflow* non destructif : laisser à l'intégration la responsabilité du choix des données de création répliquées ;

— *workflow* adaptatif : adapter le comportement de la collaboration aux besoins des artistes (par exemple, pouvoir mettre en pause la réplication de certaines données) ;

— chargement dynamique : permettre aux artistes de rejoindre et quitter une session de création collaborative à tout moment ;

— présence utilisateur : Permettre aux artistes de visualiser qui travaille sur quoi, comment et où.

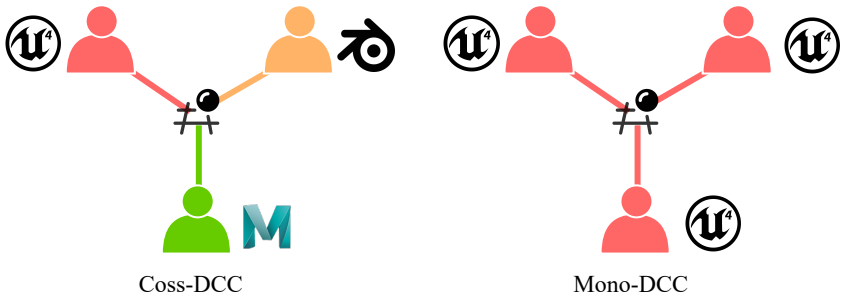


FIGURE I.3.4 – Exemple de configuration cross-DCC et mono-DCC.

On peut catégoriser les solutions de collaboration en temps réel de cross-DCC ou mono-DCC (cf. Figure I.3.4). Une démarche mono-DCC consiste à répliquer les informations au sein de plusieurs instances de la même application. Hormis pour les logiciels de création généralistes comme *Blender* ou *Maya*, cette approche favorise la collaboration entre plusieurs artistes de la même spécialisation (par exemple, concept 2D). Étant donné qu'un seul logiciel est utilisé, cette architecture rend possible la collaboration sur l'ensemble des données de création qui y sont liées et favorise donc un

dataflow non destructif. À l'inverse une application cross-DCC propagera la collaboration entre plusieurs applications différentes (par exemple, une application de *texturing*, de modélisation, etc.) supportant ainsi les échanges entre plusieurs métiers. Ici l'enjeu principal est l'interopérabilité entre logiciels. Comme nous l'avons vu avec le BIM, cette architecture exploite généralement un format de fichier qui assure les échanges entre les solutions et qui limitera la collaboration aux aspects qu'il supporte, impliquant un *dataflow* destructif.

	Mono-DCC	Cross-DCC
Interopérabilité	non	oui
Dataflow	non destructif	destructif
Complexité de développement	moyenne	élevée

Table I.3.1 – Résumé des caractéristiques des approches cross-DCC et mono-DCC.

Par sa nature, une architecture cross-DCC nécessitera le développement d'une intégration (sous la forme de *plug-in* par exemple) pour chaque solution supportée. Cela représente un effort considérable de développement. De plus, les interfaces graphiques varient énormément d'un DCC à un autre, il devient donc complexe de maintenir une présence utilisateur consistante. Enfin, cette approche est tributaire des changements de versions de la part des éditeurs pouvant remettre beaucoup de choses en cause, et empêcher les migrations d'upgrade pour avoir de nouvelles fonctionnalités.

Ainsi, du fait de son *dataflow* destructif et de la difficulté à maintenir une présence utilisateur consistante, l'architecture cross-DCC ne répondait pas à nos besoins. Nous avons donc basé notre approche sur une architecture mono-DCC.

I.3.2.2 Historique des solutions de collaboration

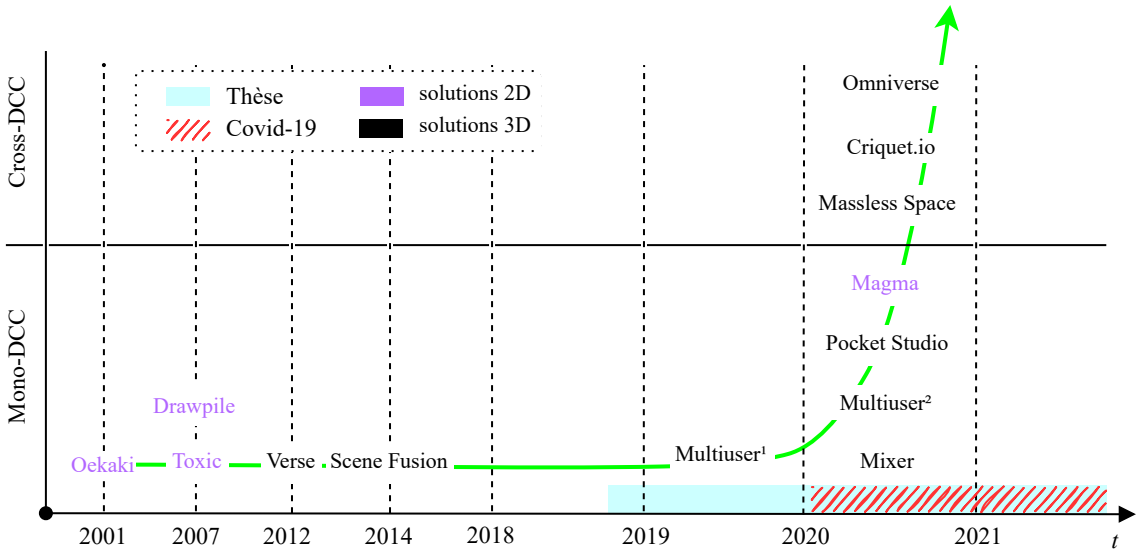
La Figure I.3.5 établit un historique des solutions de création collaborative en temps réel par date de sortie coloriée selon le type de solution (2D ou 3D). *Oekaki*[148], un logiciel de dessin collaboratif en ligne est l'une des premières expériences de co-création en temps réel. Quelques années plus tard, on retrouve *Autodesk Toxic* en 2007 qui propose également des outils de collaboration 2D, mais appliqués au *compositing*. On observe que ce sont les solutions de co-création 2D qui sont apparues en premier.

Il faudra attendre le protocole *Verse* [10] en 2012 pour voir la première tentative d'implémentation d'une interface multi-utilisateur en 3D dans un logiciel de création. Du point de vue de l'architecture, le protocole proposé par Hnídek apporte une solution efficace sur les délais de transmission des données 3D, mais a l'inconvénient de se reposer sur une architecture centralisée. Durant nos expérimentations (cf. Infra Chapitre II.2), les artistes nous ont demandé très tôt la possibilité de mettre en pause la réplication de certaines données à la demande pour itérer sans impacter les autres utilisateurs. Cela requiert d'enregistrer les mises à jour localement, ce qui est impossible lorsque les opérations sont centralisées. De plus, la solution n'était plus maintenue et nécessitait une modification du code source de *Blender* pour fonctionner.

Scene Fusion[149] apparaît ensuite en 2014. C'est un *plug-in* majoritairement utilisé

148. `http://en.oekaki.nl/`
149. `https://www.kinematicsoup.com/scene-fusion`

dans la création de jeux vidéo pour faire de l'édition collaborative de niveau (*level design* en anglais) dans *Unity* et *Unreal Engine*.



1. Outil que nous avons développé à CUBE Creative pour *Blender*.
2. Outil développé par Epic Game pour *Unreal Engine*.

FIGURE I.3.5 – Historique de sortie des outils de co-création en temps réel.

Les solutions précédemment exposées constituaient les seules pistes existantes au moment où nous avons mené notre état de l'art en novembre 2018. Aucune de ces approches ne nous permit d'explorer la collaboration en temps-réel au sein de plusieurs instances d'un même outil de création 3D utilisé en production de manière non destructive et sans conflit d'édition. Cela motiva la création d'un *framework* et de son intégration dans *Blender* (l'*add-on* Multiuser) exposé en partie II dont la première version fut publiée en septembre 2019 à l'occasion de la *Blender Conference*[150].

En 2020, on observe une explosion du nombre de solutions, montrant un intérêt certain de l'industrie pour ces méthodes naissantes. Cette augmentation peut être corrélée avec l'arrivée de la crise de la Covid-19 (en rouge sur la Figure I.3.5). Comme nous le constaterons dans le Chapitre II.2, la collaboration en temps réel permet à plusieurs artistes de travailler ensemble tout en respectant les règles de distanciation sociale.

En s'appuyant sur l'USD pour gérer les données des scènes 3D, *Nvidia Omniverse*[151] constitue une plateforme collaborative cross-DCC intéressante. Cependant, comme nous l'avons constaté en Sous-section I.1.3.2, la standardisation des formats de transmission de scènes 3D est encore limitée. Certains types de données tels que les *rigs* ne sont pas encore supportés. De ce fait, l'approche de Nvidia est limitée aux données compatibles avec l'USD. Il fut également relevé en Sous-section I.3.2.1 qu'il était complexe d'obtenir une interface d'interaction homogène dans l'approche cross-DCC à cause de la diversité des expériences utilisateurs (UX) des solutions. À titre d'exemple, les intégrations DCC d'*Omniverse* ne représentent pas les clients collaborant dans le viewport, il n'y a donc pas de présence utilisateur. Pour pallier ces limitations, la solution exposée en seconde partie de cette thèse se focalise sur une approche mono-DCC et n'impose pas un format de données 3D spécifique. Au contraire, il fournit une interface pour spécifier les données de création répliquées ainsi que leurs relations pour s'adapter aux spécificités de l'application.

150. Vidéo de la présentation : `https://youtu.be/v6XyFJLAIyM`
151. `https://developer.nvidia.com/nvidia-omniverse-platform`

Une autre approche intéressante est l'*add-on Mixer*[152] développé pour *Blender* par Ubisoft Animation Studio. Bien qu'initialement conçu en interne à l'entreprise comme un outil cross-DCC assurant un pont entre *Blender* et *Unity*, on peut le considérer comme mono-DCC car seule l'intégration à *Blender* a été publiée. Son architecture centralisée basée sur les commandes s'appuie sur de petits transferts, ce qui se traduit par une bonne réactivité du point de vue des performances. Il gère la représentation de la scène comme une pile linéaire de commandes (stockées sur le serveur). La pile croît pendant la création de la scène. Cette architecture ne fournit pas les données relationnelles nécessaires pour éviter les conflits d'édition entre utilisateurs, ce qui est un besoin critique que nous avons identifié dans nos premières expériences. Comme les artistes modifient généralement plusieurs aspects d'un *asset*, le système de contrôle d'accès doit connaître les relations entre *assets* afin de les verrouiller. De plus, son architecture centralisée ne permet pas l'exécution d'opérations locales, car il se base entièrement sur l'ordre des commandes pour reconstruire la scène.

Le *plug-in* Multiuser développé par Epic Game apporte une expérience mono-DCC similaire à *Scene Fusion*, mais dans *Unreal Engine* et avec des outils de gestion de versions ainsi qu'une représentation des utilisateurs plus poussée. Il assure la collaboration sur tous les aspects supportés par le moteur de jeu. Il est aujourd'hui omniprésent sur les plateaux de *virtual production* (cf. Figure I.2.5) pour faire communiquer les différents postes de travail entre eux.

Croquet.io[153] est une API web JavaScript qui apporte une interface multi-utilisateur dans les applications web l'intégrant. À ce jour, trop peu d'informations et d'exemples ont été publiés pour situer son intérêt dans la chaîne de fabrication d'un film.

Pocket Studio[154] est l'exemple type de solution consciente de collaborations. C'est un hub collaboratif temps réel conçu pour aider à la création de plans en pré-production. En développement depuis plusieurs années, l'application est actuellement en bêta publique, mais ne supporte que partiellement les standards, ce qui freine son adoption dans le milieu.

À travers cette section, nous avons inspecté le paysage actuel de la proposition de systèmes collaboratifs temps réel pour la création d'images de synthèse. L'étude des problématiques humaines et techniques de création d'une expérience multi-utilisateur a permis de préciser et préparer les travaux présentés en partie II.

À travers l'historique des solutions existantes, nous avons parcouru les tentatives de plusieurs sociétés tels qu'Ubisoft Animation Studio ou encore Nvidia, démontrant l'intérêt montant de l'industrie pour la pratique. Cependant, il a été constaté qu'aucune de ces solutions ne remplissait nos besoins pour l'étude de la pratique au sein de la chaîne de fabrication.

152. `https://github.com/ubisoft/mixer`
153. `https://croquet.io/`
154. `https://www.pocketstudio.io`

Conclusion de la partie I

De nos jours, la création de films d'animation suit les préceptes du fordisme en s'appuyant sur une chaîne de production. Cette structure est composée d'un ensemble d'étapes linéaires et interdépendantes qui forment des départements spécialisés. Au sein de ces entités, les artistes travaillent à créer les différents aspects du film sur des fichiers de travail individuels. La collaboration entre les départements se produit à travers l'échange de ces fichiers qui sont créés, assemblés, modifiés par une structure logicielle sous-jacente : le pipeline. En coulisse, les outils du pipeline tirent les ficelles du *workflow*, automatisant l'acheminement des données d'un poste à un autre à travers un ensemble de processus informatiques.

Mais cette structure logicielle et humaine tend à isoler les artistes dans des tâches individuelles. En empêchant la prise de conscience de l'avant et de l'après, elle nuit à l'obtention d'une vision globale du projet en cours de création. Ainsi, une erreur peut passer inaperçue pendant plusieurs étapes de fabrication. Lorsqu'elle émerge, la rigidité de ces méthodes oblige à repasser à travers toutes les étapes depuis l'origine du problème. Au final, le processus qui avait alors été adopté en partie pour sa rentabilité perd en efficacité, car il s'avère coûteux et très rigide face aux imprévus.

Finalement, les méthodes de production actuelles prônent la passation d'un résultat entre les artistes et non un partage de processus. Mais la convergence technologique entre le jeu vidéo et le cinéma remet en question ce paradigme.

L'univers de la prise de vues réelles s'est approprié les moteurs de jeu à travers la *virtual production*. Cet ensemble de techniques basées sur l'utilisation des moteurs de jeu exploite l'interactivité pour tisser des liens entre les étapes de fabrication, en resculptant la chaîne de production pour obtenir des images définitives au moment du tournage. Leur usage dans les productions de cinéma d'animation restructure le pipeline de rendu, le faisant débuter dès le *layout*.

Mais si les avantages économiques sont évidents, d'autres axes de cette convergence sont encore jeunes. Au cœur d'une production, les fichiers constituent le pivot de la communication entre les départements. Si les moteurs de jeu tendent vers l'adoption de formats standards utilisés dans le cinéma, leur intégration au flux de production induit actuellement une complexification non négligeable du *dataflow*, freinant leur adoption dans les studios d'animation.

Les verrous technologiques liés à l'utilisation des moteurs de jeu nous ont conduits à interroger les capacités interactives des logiciels de création 3D déjà établis en productions. La création de la *Smartphone Remote* interrogea et valida l'aptitude de *Blender* à gérer et interpréter des flux de données haute fréquence au sein de son moteur de rendu temps réel *Eevee*. Mais l'usage du temps réel questionne également les méthodologies de travail. L'expérience du *Corps Infini* nous ouvrit les yeux sur une organisation de travail alternative favorisant la communication en plaçant le groupe artistique au cœur de la création. La rencontre de spécialistes du pipeline nous confirma une remise en

question latente des processus de production actuels. Nous avons alors envisagé d'utiliser le temps réel comme outil pour rapprocher les artistes autour de la création.

Si le concept de collaboration en temps réel est encore très jeune dans le cinéma d'animation, il n'est pas étranger aux secteurs du génie civil et du développement informatique. Ceux-ci l'utilisent depuis plusieurs années pour placer les travailleurs autour d'une vision commune et partagée du projet en cours de fabrication. Ces deux cas d'application ont mis en valeur deux architectures logicielles distinctes qui se manifestent également dans le paysage du cinéma d'animation à travers les initiatives cross-DCC et mono-DCC. L'interopérabilité est au cœur de la stratégie cross-DCC. Cependant, si cette dernière facilite l'échange d'information entre départements, elle détruit également les données de création spécifiques à chacune des solutions. De plus, les artistes n'étant pas représentés visuellement au sein de la session de collaboration, l'ampleur de la communication supportée par cette stratégie est limitée. Ainsi, nous allions privilégier une approche mono-DCC pour mener notre exploration de la co-création en temps réel.

Deuxième partie

Recherche et développement d'un outil de co-création en temps réel pour l'animation

Introduction

La partie précédente a fait état d'un paysage en plein questionnement face à l'arrivée du temps réel dans le cinéma d'animation. Notre cheminement dans ce milieu très changeant nous a conduits à l'imaginer en tant qu'outil de collaboration dans la chaîne de fabrication. Cependant, parmi les solutions que nous avons exposées, aucune ne répondait aux besoins spécifiques de cette recherche.

Le premier chapitre de cette partie établit une description technique de l'outil multi-utilisateur qui a émergé suite aux nombreuses sessions expérimentales. Nous aborderons les deux strates qui composent notre approche : un *framework* Python et son intégration dans *Blender* à travers un *add-on* que j'ai appelé le *Multiuser*. Ces deux outils [155] ont été développés sous licence open source et sont disponibles sur *Gitlab*.

Dans un second temps, nous rendrons compte, au travers d'expérimentations, de la mise en situation de cet outil en milieu académique, industriel et public. Nous constaterons alors le lien étroit qui unit les expériences et la création de l'outil à travers une démarche de recherche-création. En analysant les expériences menées, nous étudierons l'apport de la collaboration en temps réel, d'une part, dans les processus de fabrication d'images animées, et d'autre part pour l'enseignement de la discipline.

Les sessions à Cube Creative furent un terrain favorable pour imaginer des applications concrètes pour la suite cette thèse. C'est pourquoi le troisième chapitre décrit ces différents cas d'usage envisagés à court terme pour améliorer la communication entre artistes dans un pipeline de production traditionnel.

Mais regrouper les artistes dans le même espace de création induit un changement de paradigme profond : ce ne sont plus des individus séparés qui créent, mais un groupe. Cette idée est au cœur des méthodes agiles. Aussi, le dernier chapitre de la thèse envisage une autre façon d'aborder la production d'images animées et étudie comment la collaboration en temps réel s'y inscrit.

155. *Framework* : `https://gitlab.com/slumber/replication` - *Multiuser* : `https://gitlab.com/slumber/multi-user`

Chapitre II.1

Multiuser : la collaboration en temps réel au cœur de la création

Introduction

Aujourd'hui, il n'existe aucun consensus de *workflow* pour la collaboration en temps réel dans le cinéma d'animation. Pour cette raison, il existe de nombreuses directions à explorer.

L'outil multi-utilisateur développé au cours de cette thèse a été conçu dès le début avec une idée de souplesse. Le code métier a été isolé dans une brique logicielle, appelée *framework*, indépendante du DCC pour faciliter son développement et sa maintenance. Ce dernier expose un ensemble de fonctions haut niveau pour définir le *dataflow* et le *workflow* collaboratif directement depuis une intégration DCC.

Au cours de ce chapitre, nous décrirons les deux composants développés à partir de 2018 pour répondre aux besoins énoncés en Sous-section I.3.2.1. Tout d'abord, nous aborderons le noyau logique qui orchestre et modélise l'architecture de l'expérience multi-utilisateur : le *framework*. Puis, nous décrirons ensuite son intégration à *Blender* : l'*add-on* Mulitiuser.

II.1.1 Spécificités d'un DCC 3D

Cette section explique brièvement des notions relatives au fonctionnement interne des logiciels de création 3D nécessaires à la compréhension des outils exposés dans ce chapitre.

Dans les DCC 3D génériques tels que *Blender* ou *Maya*, les scènes sont composées d'une grande variété de types d'*assets*. Par exemple, on retrouve des *meshes*, des systèmes de particules, des lumières, des *shaders*, etc.

Ces logiciels suivent généralement une architecture interne orientée objet pour définir les types d'*assets*. Les objets sont des structures logicielles qui contiennent un ensemble d'attributs (variables stockant des données relatives à l'objet) et de méthodes (fonctions relatives à l'objet).

Dans *Blender* par exemple, chaque type d'*asset* composant la scène est défini à travers un objet développé en C dans le dossier `blenkernel` (pour *Blender Kernel*) [156] s'appuyant sur des structures [157] décrites dans le dossier `makesdna` [158] des sources. Pour les *meshes* par exemple [159], on retrouvera une structure `Mesh` définissant un groupe de variables correspondants aux données du mesh (les arrêtes, les points, les faces, les matériaux, etc.). Elle est accompagnée d'un ensemble de fonctions pour initialiser, manipuler et accéder à ces données (par exemple, la fonction `BKE_mesh_add` retournera un pointeur vers une nouvelle instance de la structure `Mesh` et est utilisée lorsqu'un artiste crée un objet de type *mesh*).

Les structures comme `Mesh` déterminent donc les données de création employées par les différents types d'*assets*. Pour chaque *mesh* présent dans la scène 3D, une instance de cette structure est créée en mémoire pour stocker ses données. Ce sont ces instances que nous qualifierons de *datablock* [160] par la suite.

En plus d'avoir une grande diversité de types de *datablocks*, une scène 3D peut également contenir un très grand nombre d'objets 3D, donc de *datablocks*, et par ce fait, occuper une place importante en mémoire ou sur disque, et devenir ainsi plus longue à manipuler. Aussi, pour effectuer efficacement les mises à jour de leur représentation dans le *viewport* [161] lorsqu'ils sont édités par l'artiste, il est critique de connaître leurs relations. C'est pourquoi, une grande partie des DCC hiérarchisent les *datablocks* composants la scène 3D à travers un graphe acyclique dirigé (communément appelé graphe de dépendance, *dependency graph* ou encore *scene graph*). Si son objectif reste similaire, le contenu et la forme de ce graphe varient beaucoup d'un logiciel à un autre.

Dans *Maya* par exemple, le graphe de dépendance est composé de nœuds stockant

156. Ces objets sont accessibles à l'adresse `https://developer.blender.org/diffusion/B/browse/blender-v2.93-release/source/blender/blenkernel/`

157. Dans le langage C, une structure (écrite `struct` dans le code source) est un type de donnée composite qui permet de définir une liste physiquement groupée de variables sous un seul nom dans un bloc de mémoire. Les différentes variables d'une structure sont appelées des membres ou des champs et peuvent être de différent type.

158. Ces structures sont explorables à l'adresse `https://developer.blender.org/diffusion/B/browse/blender-v2.93-release/source/blender/makesdna/`

159. Définis dans les fichiers `BKE_mesh.h`, `DNA_mesh_types.h` et `mesh.c`

160. Un bloc de données ou *datablock* désigne toute unité d'information de la scène 3D stockée par le logiciel de création 3D (par exemple : un *shader*, un *mesh*, un système de particule, etc.).

161. En informatique graphique 3D, le *viewport* désigne le rectangle 2D utilisé pour projeter la scène 3D à l'emplacement d'une caméra virtuelle. C'est une région de l'écran utilisée pour afficher une partie (ou la totalité) de l'image à montrer. Définition par Wikipédia.

chacune des opérations et des données associées utilisées pour construire la scène (par exemple, un nœud *transform* permet de déplacer spatialement les objets parents via des attributs de position, de rotation et de taille) [162]. Ainsi, dans *Maya* les *datablocks* sont stockés dans les nœuds du graphe de dépendance.

Dans *Blender*, une opération effectuées par un artiste modifie directement les données dans les *datablocks* mais ne sera visible dans le *viewport* qu'après évaluation (par exemple, une opération de déplacement d'un point de *mesh* ira directement modifier sa matrice de transformation dans le *datablock*). Le graphe de dépendance comporte un nœud par *datablock* qui stocke principalement le résultat de son évaluation [163] (par exemple, le *mesh* résultant après déplacement).

Dans les deux cas, l'évaluation du graphe de dépendance se déclenche dès qu'un artiste effectue une modification : elle consiste à interpréter les données des *datablocks* modifiés et leurs dépendances pour mettre à jour leurs représentations. Les données relationnelles contenues dans le graphe évitent une réévaluation complète de la scène en fournissant efficacement les dépendances des éléments modifiés pour les évaluer. Cette opération est propre à chaque type de *datablock*, elle est donc généralement définie directement dans une fonction de son fichier source. Dans *Blender*, lorsqu'un *mesh* est modifié, la fonction `BKE_mesh_eval_geometry` définie dans le fichier `mesh.c` sera appelée pour l'évaluer. Dans *Maya*, c'est la méthode `compute()` de chacun des nœuds qui sera appelée pour les évaluer en fonction des entrées et données qui y sont stockées.

Dans le contexte de la collaboration en temps réel, les *datablocks* représentent donc les données de création à partager, à répliquer entre les artistes ; tandis que le graphe de dépendance permet de connaître précisément quels *datablocks* ont changé lorsqu'une modification locale de la scène se produit.

162. Documentation du graphe de dépendance de *Maya* : `https://knowledge.autodesk.com/support/maya/learn-explore/caas/CloudHelp/cloudhelp/2022/ENU/Maya-Basics/files/GUID-51096BC4-32B7-4391-BE39-21641B374745-htm.html`

163. Documentation du graphe de dépendance de *Blender* : `https://wiki.blender.org/wiki/Source/Depsgraph`

II.1.2 Un framework pour la création collaborative de contenu numérique 3D

Construire un outil de collaboration en temps réel s'apparente à créer un pipeline pour connecter les artistes entre eux et les amener à travailler ensemble. On y retrouve les problématiques centrales de *dataflow* et de *workflow*. Nous articulons ces deux notions critiques au cœur du *framework* à travers une interface de définition de données ainsi qu'un ensemble de fonctions de collaboration conscientes. Ces dernières adaptent le *workflow* collaboratif aux besoins des artistes.

L'approche exposée est conçue pour être intégrée dans des *collaboration-transparent programs*, tels que les logiciels de création 3D qui ignorent totalement qu'ils interagissent avec plusieurs utilisateurs actifs sur la même scène. Ce *framework* répond à certains des besoins énumérés en Sous-section I.3.2.1 concernant la mise en œuvre de l'expérience multi-utilisateur pour la création de contenu 3D en animation grâce aux caractéristiques suivantes :

— un protocole de définition des données répliquées aussi dénommé Replicated Data definition Protocol (RDP) [164] : agissant comme une interface pour la définition des *datablocks* répliqués ;

— un système de session : gérer la connexion/déconnexion des utilisateurs à tout moment de la création de la scène ;

— un système de contrôle d'accès : prévention des conflits d'édition simultanée de *datablock* ;

— fonctions conscientes de la collaboration : fournir des fonctions pour gérer le comportement du pipeline de réplication à partir de l'interface de programmation du DCC ;

— interface consciente de la collaboration : fournir une interface pour la création d'outils dédiés à la coordination de la collaboration (par exemple, pour aider les utilisateurs à s'organiser).

Dans l'animation, la majorité des solutions de création intègre une API Python pour étendre leurs fonctionnalités et les intégrer dans un pipeline de production. Par conséquent, le *framework* multi-utilisateur a été conçu comme un module Python pour faciliter son intégration.

Dans les DCC 3D, les utilisateurs peuvent accéder à de nombreux opérateurs afin de créer et de modifier une scène 3D (par exemple, il existe des opérateurs pour créer les points d'un maillage, pour les déplacer, pour les diviser, etc.). En fonction du logiciel de création 3D, le *workflow* de création est plus ou moins destructif. Dans un pipeline linéaire classique, l'exportation entre les tâches entraîne une perte d'informations due au verrouillage de l'aspect artistique. Mais dans le contexte de la collaboration multi-utilisateur en temps réel, les tâches sont parallélisées. Ainsi, il est important de conserver toutes les informations de création de manière non destructive. Les opérateurs sont souvent liés à un contexte d'exécution. La réplication d'une opération nécessiterait de synchroniser le contexte associé chez tous les clients et de l'imposer lors de l'application de l'opération, ce qui entraînerait un comportement potentiellement inattendu de l'interface, une dégradation de l'expérience utilisateur et des erreurs.

164. Le *Replicated Data Protocol* est une interface de notre *framework* (cf. Sous-section II.1.2.2) permettant au développeur de définir avec précision les données de création du DCC répliquée à travers la session de collaboration.

Par conséquent, nous avons opté pour une approche de réplication orientée données (*data based replication* en anglais). Celle-ci présente plusieurs avantages par rapport à nos besoins initiaux :

— elle permet de collaborer sur des données procédurales (par exemple, un réseau nodal pour générer une géométrie) ;

— elle facilite la mise en œuvre d'une architecture distribuée, pour appliquer aux données, des opérations locales à l'utilisateur ;

— elle permet d'étendre les données de création avec un système de contrôle d'accès.

Dans le *framework*, un client et une instance d'application ne sont pas différenciés. Ils sont représentés par le même objet. Il y a donc un client par instance d'application. Les clients se connectent à une session de travail collaboratif gérée par le serveur de réplication. La session est un objet à état sur lequel le serveur a autorité ; on peut l'assimiler au *game state*[165] dans le jeu vidéo. Elle stocke les informations des clients connectés et le statut de la session de création actuelle. Le serveur est un script léger pouvant être démarré sur l'ordinateur de l'un des clients ou sur un serveur dédié.

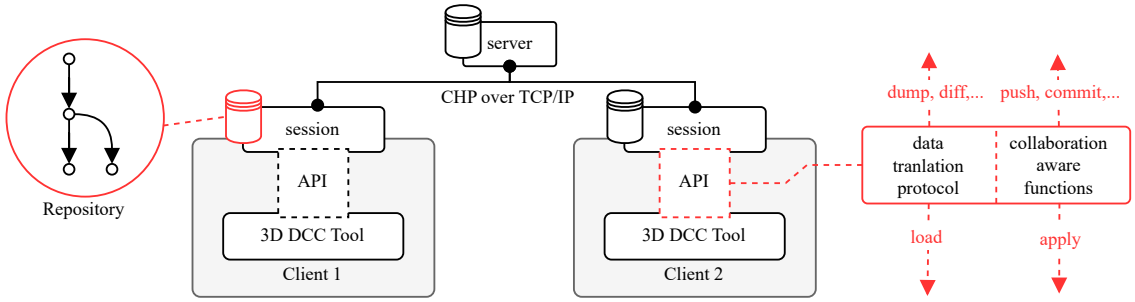


FIGURE II.1.1 – Architecture globale du *framework*

II.1.2.1 Architecture du framework

Le *framework* se structure comme une combinaison d'architectures distribuées et centralisées. L'approche distribuée est utilisée pour répliquer les données de la scène tandis que la gestion des commandes (verrouillage, clone, etc.) est effectuée de manière centralisée pour garantir l'autorité du serveur.

Réplication distribuée de données

Les données de la scène sont stockées dans un graphe de réplication généré par le protocole de définition des données. Comme montré dans la Figure II.1.4, ce graphe est stocké dans un objet *repository*[166] sous la forme d'un dictionnaire d'entrées clefs-valeur (*key-pair values*). Une clef représente l'identifiant unique d'un nœud et la valeur représente ses données. Le graphe de réplication est différent du graphe de dépendance interne au DCC dans sens où chaque nœud contient non seulement un miroir de *datablock* mais surtout des métadonnées propres à la collaboration (par exemple, le propriétaire, représenté par le champ `owner` sur la Figure II.1.4).

165. En jeu vidéo, le *game state* est un composant logiciel responsable du stockage de l'état global du jeu. Dans un jeu multijoueur par exemple, il peut être utilisé pour stocker les joueurs connectés.
166. Dans le *Multiuser* (cf. Chapitre II.1), le *repository* est un objet qui stocke une version locale des données de création répliquées.

En clonant le graphe de réplication lors de leurs connexions, les clients obtiennent une copie locale complète de ces données (similaire à l'opération clone sur *Git*). Par la suite, les modifications qui y seront apportées seront appliquées localement puis répliquées. Cela permet de résoudre le problème de l'indépendance des erreurs. En exécutant localement certaines des fonctions de collaboration (cf. infra Sous-section II.1.2.4), les erreurs n'affectent pas les autres utilisateurs.

Réplication centralisée des commandes

Une commande contient un ensemble d'instructions à exécuter sur le *repository* (par exemple, verrouiller/déverrouiller un nœud). Le serveur, qui a autorité sur le contrôle d'accès, valide si une commande peut être appliquée. Par conséquent, toutes les commandes sont d'abord envoyées au serveur pour un contrôle d'autorisation (par exemple, pour savoir si le client a le droit de verrouiller/déverrouiller un nœud précis). Elles sont ensuite exécutées sur le *repository* du serveur et relayée aux autres clients qui les appliqueront. Les commandes suivantes sont utilisées à travers les différents composants du *framework* :

— `Authenticate` : connexion au serveur ;

— `Clone` : transfère une copie complète du *repository* du serveur au client local ;

— `Lock/Unlock` : utilisé par le système contrôle d'accès (voir Sous-section II.1.2.3) pour acquérir/relâcher les droits de modification d'un ou plusieurs nœuds ;

— `Kick` : enlève un utilisateur de la session ;

— `UpdateUserMetadata` : utilisé par les interfaces conscientes de collaborations (cf. infra Sous-section II.1.2.5) pour mettre à jour les métadonnées [167] d'un utilisateur ;

— `Delete` : supprime un nœud du *repository* pour permettre aux artistes de stopper manuellement le suivi d'un *datablock* ;

— `RequestServerInfo` : interroge le statut du serveur cible, utilisé pour interroger l'état des serveurs en ligne sans être dans une session.

Architecture réseau

La couche réseau a été développé avec la même bibliothèque que l'application *Smartphone Remote* : *ZeroMQ*.

Le *framework* implémente une version modifiée du *Clustered Hashmap Protocol* (CHP) [25] pour échanger les données du *repository* à travers le réseau. Ce protocole est prévu pour la réplication d'états (sous la forme d'un ensemble de paires clefs-valeurs) partagés entre un serveur et un grand nombre de clients. Par exemple, pour répliquer une configuration entre un groupe de serveurs ou encore un *game state* entre plusieurs joueurs.

Le CHP prévoit trois *sockets* [168] pour assurer la communication entre les clients et le serveur. À sa connexion, un client télécharge l'ensemble des états partagés à travers un premier *socket*. Une fois à jour, il réceptionne en temps réel les mises à jour que le

167. Une métadonnée (ou *metadata*) est une information qui décrit des données qu'elle accompagne. Dans le cadre du *Multiuser*, les métadonnées sont utilisées pour enrichir les informations liées à un utilisateur (par exemple, pour décrire son objet sélectionné).

168. Un *socket* ou « connecteur réseau » est une interface logicielle qui permet au développeur d'exploiter les services d'un protocole réseau.

serveur relaye à tous les clients connectés à travers un second *socket*. Enfin, un troisième *socket* permet à chacun des clients de pousser sur le serveur une mise à jour de l'un des états. Si elle est acceptée, le serveur la relayera à tous les clients.

Ce protocole nous est apparu adapté pour échanger les mises à jour du graphe de réplication qui n'est autre qu'un dictionnaire d'entrées clefs-valeur. Cependant, pour éviter l'envoi de données inutiles, nous avons modifié la publication des mises à jour de façon à ce que le serveur ne relaye pas une mise à jour au client qui en est à l'origine, car ce dernier l'aura déjà appliquée localement (cf. infra Sous-section II.1.2.4). Nous avons également fusionné les sockets de publication et de distribution des mises à jours en un seul (`DATA` sur la Figure II.1.2) pour éviter la multiplication des ports requis pour communiquer.

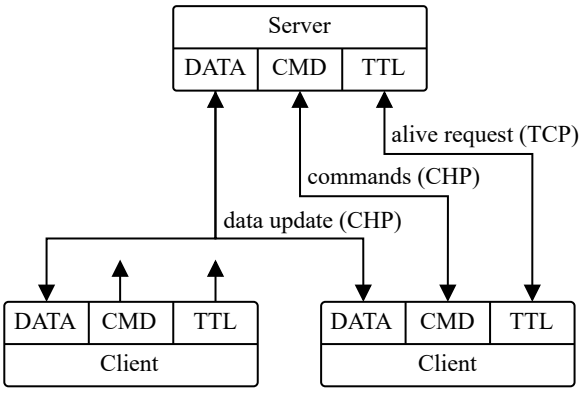


Figure II.1.2 – Couche de communication réseau

Pour réduire la bande passante utilisée par les transferts de données, les changements de nœuds sont transmis sous forme de *deltas* (ou différentiels) calculés pendant les `COMMIT` (cf. infra Sous-section II.1.2.4). La taille de ces *deltas* varie fortement en fonction de la nature du changement. Par exemple, le déplacement d'un objet génère un petit *delta* alors que la mise à jour d'un maillage complexe est beaucoup plus importante. Par conséquent, la granularité des données transmises dépend directement des modifications apportées par les utilisateurs.

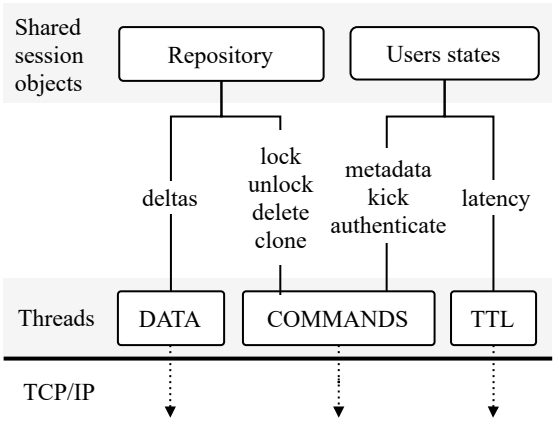


Figure II.1.3 – Architecture interne du serveur avec une réception synchrone des paquets réseau.

Comme le montre la Figure II.1.2, les données et les commandes sont transmises par deux *sockets* différents. Cela permet de les traiter de manière simultanée. En séparant

ces deux types de transactions au niveau du serveur (cf. Figure II.1.3), nous garantissons la réactivité du système droit basé sur les commandes, même lorsque de grands *deltas* sont transférés.

Chaque canal est basé sur TCP pour limiter la perte de paquets. Le *socket Time To Leave* (TTL) permet de mesurer le *ping*[169] et de définir si les clients sont en ligne grâce à un *heartbeat*[170] régulier. Côté client, ce mécanisme est intégralement géré dans un processus séparé pour ne pas être influencé par les calculs lourds, typiques des outils de création 3D (rendu, etc.). La même stratégie est appliquée côté serveur.

II.1.2.2 Protocole de définition des données répliquées

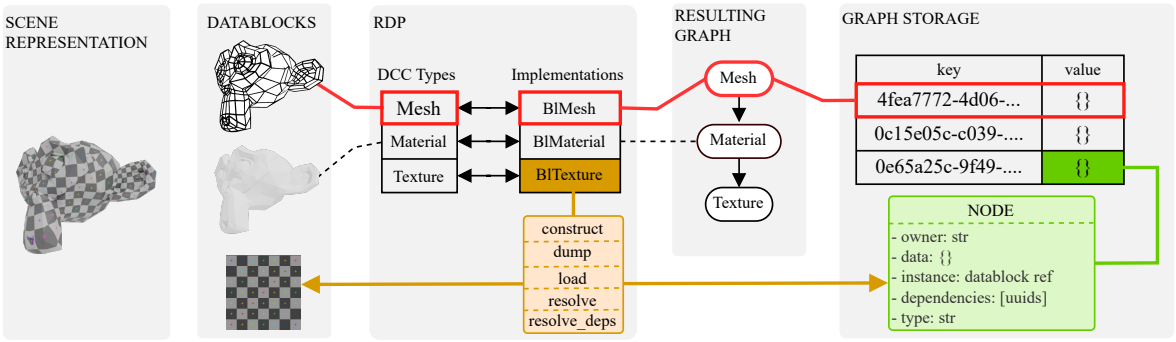


FIGURE II.1.4 – Traduction des données avec le RDP

Le protocole de définition des données répliquées (RDP) représente la clef de voûte entre le DCC et le *framework*. Il est utilisé comme un dictionnaire de traduction pour faire transiter les données entre eux (voir Figure II.1.4).

Pour ajouter un type de données supporté par le *framework*, il est nécessaire de définir une implémentation de la classe abstraite `ReplicatedDatablock` offrant les fonctions suivantes :

- `dump` : traduit l'objet ciblé en dictionnaire utilisant des types standards ;
- `load` : charge les données du dictionnaire dans le *datablock* du DCC ;
- `construct` : instancie le *datablock* vide ;
- `resolve` : cherche une instance déjà existante du *datablock* ;
- `resolve_dependencies` : retourne les dépendances du *datablock* (par exemple, textures, *meshes*, etc.) ;
- `needs_update` : vérifie si une mise à jour du *datablock* est requise ;
- `compute_delta` : calcule la différence entre un *datablock* et son nœud stocké dans le *repository*.

Les méthodes ci-dessus sont utilisées par les fonctions de collaboration (cf. infra Sous-section II.1.2.4) à travers le pipeline de réplication pour échanger des données entre le *repository* local et les *datablocks* du DCC.

La définition de ces implémentations n'a lieu qu'une seule fois. Elle se produit pendant la phase d'intégration du *framework* dans le DCC. Comme elles interagissent

169. Ping est le nom d'une commande informatique permettant de tester l'accessibilité d'une autre machine à travers un réseau IP. Définition par Wikipédia.
170. En informatique, un *heartbeat* est un mécanisme logiciel qui surveille la disponibilité d'un ou plusieurs programmes sur le réseau en leur envoyant des requêtes à intervalle régulier.

profondément avec le logiciel de création 3D, une connaissance solide de l'API Python du DCC est nécessaire. Dans le *Multiuser* (cf. infra Section II.1.3), nous avons développé ces implémentations en tant que sous-module de l'*add-on* responsable de l'intégration du *framework*.

Une fois définies, ces implémentations sont stockées dans un dictionnaire clef-valeur (voir Figure II.1.4). La clef représente le type du *datablock* et la valeur représente l'implémentation correspondante. Ce dictionnaire est ensuite transmis au *framework* lors de la création du *repository*. De cette façon, le *framework* est capable d'interagir avec les données du DCC pour les répliquer.

Le dictionnaire d'implémentations sera ainsi utilisé tout au long de la session pour synchroniser le graphe de réplication et les *datablocks* de la scène 3D dans les deux sens. Par exemple, lorsqu'un *datablock* est ajouté dans la scène, un nœud correspondant sera instancié dans le graphe de réplication grâce aux fonctions fournies par l'implémentation correspondante au type du *datablock*.

II.1.2.3 Contrôle d'accès concurrentiel

Initialement, le *framework* ne bénéficiait d'aucun système de contrôle d'accès, plusieurs utilisateurs pouvaient modifier le même *datablock*, conduisant naturellement à des conflits d'éditions (cf. infra Sous-section II.2.3.1). Pour les prévenir, nous avons limité la modification d'un *datablock* à un seul utilisateur. Le système de contrôle d'accès gère la propriété de chaque nœud du graphe de réplication. Il va permettre ou non la modification de ces derniers en fonction de leur propriétaire.

La fonction `change_owner` permet de changer les droits sur un nœud et ses dépendances. Lorsqu'un nœud appartient à un utilisateur, seul cet utilisateur a la capacité d'y apporter des modifications.

Par défaut, la politique de gestion de droit considère qu'un nœud appartient à ce que nous surnommons le « droit commun ». Cela signifie que lorsqu'un utilisateur déverrouille un nœud, il le cède au droit commun. Seuls les nœuds appartenant au droit commun peuvent être verrouillés par un utilisateur. Ce système de contrôle d'accès garantit une collaboration sans conflits en permettant aux utilisateurs de travailler de manière simultanée sur différents *datablocks* de la scène 3D.

II.1.2.4 Fonctions conscientes de la collaboration

Les fonctions conscientes de la collaboration (abrégée fonction-CC)) forment le cœur du pipeline de réplication du *framework*. Elles sont exposées au DCC pour lui permettre de contrôler et d'adapter le rythme des mises à jour à ses besoins. Chacune de ces fonctions peut être appelée automatiquement (par le programme) ou manuellement (par l'utilisateur). La Figure II.1.5 montre la place de ces fonctions dans la chaîne de réplication. Nous nous sommes inspirés de *Git* pour développer ces mécanismes de mise à disposition collaborative du travail en cours ou finalisé par les artistes.

Comme montré dans la Figure II.1.5, la fonction-CC `ADD` ajoute d'abord le *datablock* ciblé au *repository* local en instanciant un nouveau nœud correspondant à son type (avec le RDP présenté dans la Sous-section II.1.2.2). C'est le point d'entrée du pipeline de réplication. Lorsqu'un *datablock* est ajouté au *repository*, il est considéré comme suivi. Au cours d'une session, `ADD` doit être appelé pour enregistrer chaque nouveau *datablock*.

Ensuite, la fonction-CC `COMMIT` peut être exécutée sur n'importe quel nœud. Tout

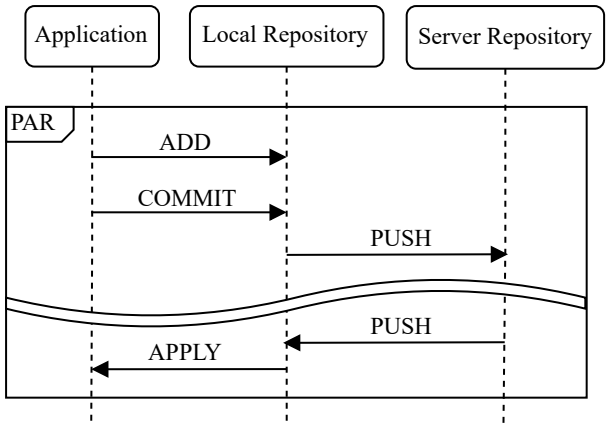


FIGURE II.1.5 – *Pipeline* des fonctions primitives de collaboration

d'abord, elle vérifiera l'état des dépendances du *datablock* pour assurer l'intégrité des données publiées. Pour chaque *datablock* renvoyé par `resolve_dependencies` (cf. Sous-section II.1.2.2), elle vérifiera s'il est suivi et à jour. Si c'est le cas, ce dernier fera l'objet d'un `ADD` ou d'un `COMMIT`. Dans un deuxième temps, il calculera les modifications et les appliquera au nœud correspondant du *repository* local. Les changements consistent en un différentiel des données extraites (avec `DUMP`, cf. Sous-section II.1.2.2) du *datablock* dans son état actuel et de la dernière version commitée (stockée dans le champ `data` du nœud correspondant, cf. Figure II.1.4). Nous utilisons *DeepDiff*[171] pour calculer la différence entre les deux versions.

La fonction-CC `PUSH` est ensuite appelée pour transférer les modifications locales au serveur qui les appliquera à son propre *repository* et les transmettra directement aux clients connectés.

Dès que les modifications sont reçues par les clients connectés, elles peuvent être chargées avec `APPLY`. Cette fonction-CC vérifie l'existence du *datablock* correspondant avec `resolve` (voir la Sous-section II.1.2.2). S'il n'est pas résolu, cela signifie qu'il s'agit d'un nouveau *datablock* correspondant à un nouvel élément de scène. Dans ce cas, il sera instancié à l'aide de `construct` (voir la Sous-section II.1.2.2). Une fois l'instance de *datablock* trouvée, la méthode `load` (cf. Sous-section II.1.2.2) y chargera ses données mises à jour.

Le comportement par défaut est de faire un `COMMIT` puis de faire un `PUSH` juste après pour avoir la mise à jour en temps réel. En revanche, dans certaines situations (par exemple, sculpture de gros volumes de géométries) il est très important de donner la capacité à un artiste de stopper temporairement l'envoi des données, pour éviter de dégrader les performances des autres clients. En séparant les étapes de réplication dans quatre fonctions conscientes de la collaboration, ce schéma de réplication répond au problème de débrayage des mises à jour et de *workflow* adaptatif.

II.1.2.5 Interfaces conscientes de la collaboration

Lors de l'édition de scène de création 3D pour l'animation, l'utilisateur édite les *assets* dans un contexte spatial et temporel. Si l'on extrapole cela dans un environnement multi-utilisateur, il est important que chaque utilisateur ait conscience des actions des autres utilisateurs présents sur la même scène 3D.

L'objet `user_states` du *framework* est destiné à supporter de telles interfaces.

171. `https://zepworks.com/`

Clef	Valeur

`view_matrix`	`[[0.0, 0.9, 0.0, 0.0],`
	`[-0.4, 0.0, 0.9, 0.0],`
	`[0.9, 0.1, 0.4,-5.0],`
	`[0.0, 0.0, 0.0, 1.0]]`

`color`	`[0.0, 0.3, 0.2, 1.0]`
`frame_current`	`91`
`scene_current`	`main_stage`
`selected_objects`	`['object_1', 'object_2']`

TABLE II.1.1 – Échantillon d'un dictionnaire de métadonnées utilisateur, utilisé dans l'intégration du *framework* dans Blender

Plus concrètement, il s'agit d'un dictionnaire de métadonnées (exemple dans le Tableau II.1.1) spécifiques à chaque utilisateur, qui sont stockées et répliquées sur chaque client. Le *framework* fournit deux fonctions pour interagir avec cet objet :

— `update_user_metadata()` permet à un client de mettre à jour un ou plusieurs de ses champs de métadonnées sur tous les clients connectés ;

— `get_users_states()` permet de lire le dictionnaire de métadonnées de tous les clients connectés (y compris le client local).

Pour maintenir la conscience de la collaboration, il était crucial de prendre en charge un taux de rafraîchissement élevé des métadonnées. Par exemple, l'opérateur de synchronisation temporelle exigeait un taux de rafraîchissement minimal de 30 Hz pour le champ `frame_current` (*frame* actuelle du client). Pour ce faire, nous avons limité la taille des mises à jour au strict minimum. Ainsi, lorsqu'un client demande à mettre à jour un champ avec `set_metadata`, seul ce dernier sera relayé à travers le réseau encapsulé dans une commande `UpdateUserMetadata` (cf. Sous-section II.1.2.1) qui patchera tous les objets `user_states` des clients connectés.

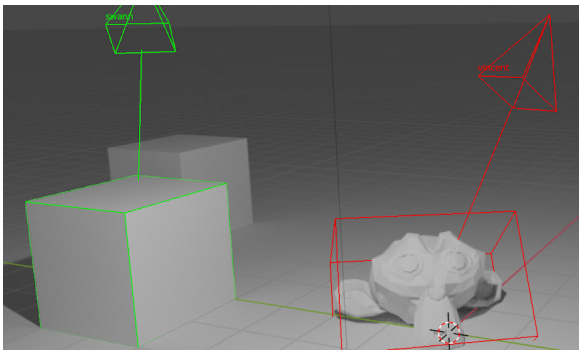


FIGURE II.1.6 – Interface consciente de collaboration dans la vue 3D de Blender

Par exemple, l'implémentation du *framework* dans *Blender* (cf. Section II.1.3) s'appuie sur ces interfaces pour stocker le point de vue de l'utilisateur (voir le champ `view_matrix` dans le Tableau II.1.1) pour dessiner son point de vue dans le *viewport* des autres clients connectés (cf. Figure II.1.6). Le même concept est appliqué à la représentation de la sélection active (voir le champ `selected_objects` du Tableau II.1.1) de chaque utilisateur. Ces deux éléments de l'interface utilisateur permettent aux artistes connectés de savoir en temps réel où se trouvent les autres et sur quoi ils travaillent.

II.1.2.6 Sauvegarde des sessions

Au cours des expérimentations (cf. infra Section II.2.1), nous avons pris rapidement conscience du besoin d'enregistrer les informations de session pour sauvegarder et analyser le travail collaboratif. Pour supporter cette fonctionnalité, deux fonctions ont été ajoutées au *repository* :

— `dumps` : sauvegarde tout le *repository* de la session dans un fichier `.db` ;

— `loads` : charge un *repository* le depuis un fichier `.db`.

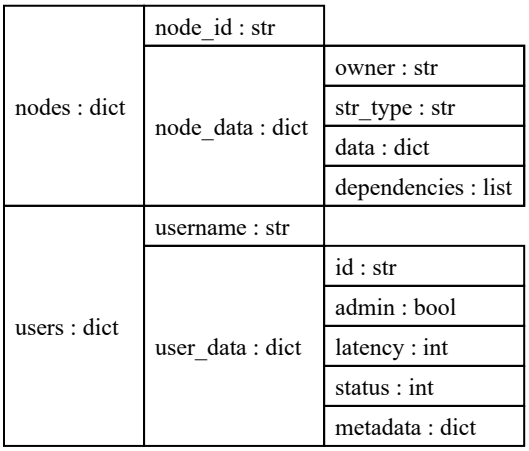


FIGURE II.1.7 – Structure des fichiers de sauvegarde de session.

La Figure II.1.7 montre la structure du fichier exporté par la fonction `dumps`. On constate qu'elle sauvegarde toutes les informations de la session avec, d'une part, les données de création brutes (dans le champ `data`) et leur appartenance (dans le champ `owner`), et d'autre part, les données des utilisateurs connectés incluant leurs métadonnées.

En permettant de restaurer une session à un instant t avec toutes les informations de collaboration, cette fonctionnalité d'import/export fut une aide précieuse pour extraire les points saillants des sessions expérimentales.

II.1.3 Intégration du framework dans *Blender* : l'*add-on Multiuser*

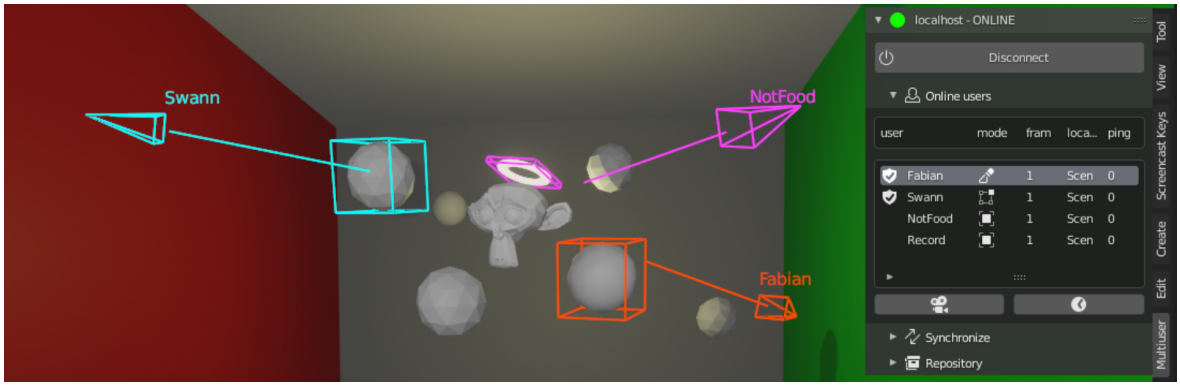


FIGURE II.1.8 – *Multiuser* en action.

Lorsque vint la question du DCC sur lequel créer l'intégration, notre choix se porta sur *Blender* pour de nombreuses raisons. Tout d'abord, c'est un véritable couteau suisse pour la création 3D. À travers ses très nombreuses fonctionnalités, il supporte toutes les étapes de fabrication d'un film d'animation. Il permit ainsi l'exploration de la collaboration au sein du même ou de plusieurs métiers (cf. Figure II.1.9).

De plus, Cube Creative l'utilise dans une grande partie de son pipeline de fabrication de série d'animation depuis la construction des *assets* (modélisation, *texturing*, *shading* et *rigging*) jusqu'au rendu en passant par l'animation. Un des ses principaux atouts pour la collaboration en temps réel réside dans son système de *shader* unifié entre son moteur de rendu temps-réel (*Eevee*) et son moteur précalculé (*Cycles*) permettant de prévisualiser avec une haute fidélité les rendus des matières en temps réel. Appliqué au contexte multi-utilisateur, cela donne la possibilité à tous les artistes de visualiser en temps réel avec *Eevee* les changements visuels de la scène tout en gardant une fidélité proche du rendu définitif que produira *Cycles*.

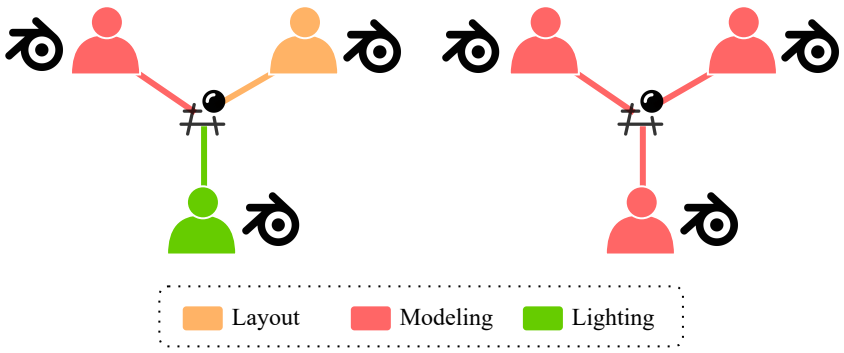


FIGURE II.1.9 – Exemple de configurations de collaboration permises avec *Blender*.

Pour faciliter la compréhension de cette section, nous introduirons préalablement quelques termes spécifiques au développement d'*add-on* pour *Blender*. À travers son API Python nommée *Blender PYthon* (BPY), *Blender* expose différentes interfaces pour étendre ses fonctionnalités et l'intégrer dans les chaînes de fabrication. Presque toutes les données exposées (par exemple, un *mesh*) sont accessibles et modifiables par

des appels à BPY. Nous étudierons la subtilité de ce « presque » un peu plus loin dans notre exposé.

Dans *Blender*, les outils exposés à l'utilisateur sont des opérateurs. L'utilisateur y accède généralement à partir de boutons, d'éléments de menu ou de raccourcis clavier. Par exemple, lorsque l'on déplace un objet, *Blender* fait appel à son opérateur `bpy.ops.transform.translate`. Mais cet opérateur peut également être appelé depuis un script Python via le module `bpy.ops`. Ce dernier permet également la définition de nouveaux opérateurs.

Durant une session, il est crucial de détecter les événements provoqués par l'utilisateur durant son interaction avec le logiciel pour répliquer les changements résultants. Afin de permettre à un *add-on* Python de réagir à certains évènements, *Blender* propose un système de notification. L'*add-on* enregistre auprès d'un *handler* (accessible dans le module `bpy.app.handlers`) une fonction de *callback*, qui sera appelée automatiquement lorsque l'évènement se produit. Un système de *handler* similaire est disponible pour dessiner dans le *viewport* 3D. À titre d'exemple, le *handler* `bpy.app.handlers.render_post` se déclenchera juste après le rendu d'une *frame*.

Depuis la version 2.8, BPY propose une interface pour faciliter la gestion de tâches périodiques : les *timers*. Ces derniers permettent d'enregistrer des fonctions s'exécutant à intervalles réguliers.

Dans cette section et les suivantes, nous décrirons les aspects importants de la création de l'*add-on Multiuser* aux yeux des problématiques d'implémentation d'une expérience de collaboration en temps réel.

II.1.3.1 Définition des *datablocks* répliqués

Comme nous l'avons présenté lors de la description du *framework* en Sous-section II.1.2.2, la définition des implémentations supporte l'échange de données entre le *repository* local et le DCC. Ainsi, on retrouve une implémentation pour chaque type de *datablocks* supporté dans le *Multiuser*. Dans *Blender*, il existe à minima un type de *datablock* pour chaque aspect de la création. Or, étant donné que ce dernier supporte presque tous les aspects de la création d'un film d'animation, il contient beaucoup de types de données. Par conséquent, nous avons dû développer un grand nombre d'implémentations. Le Tableau II.1.2 expose la liste des types de *datablocks* supportés dans le *Multiuser* (chaque type ayant une implémentation correspondante).

Le support des différents types de données a beaucoup évolué avec le temps en fonction des besoins des sessions expérimentales, tendant progressivement vers la prise en charge de l'intégralité des aspects que *Blender* permet de manipuler. La première version supportait très peu d'aspects (cf. Tableau II.1.2). Désormais, il manque très peu de type de données à couvrir.

Pour chaque implémentation, il faut se questionner sur le contenu que l'on veut répliquer. Ce choix influencera directement les performances et le comportement de la collaboration. Dans un logiciel de création, on observe deux grandes catégories de données exposées dans l'API :

— les données « utilisateur » directement manipulées par ce dernier (par exemple, les points d'un *mesh*) ;

— les données générées par le DCC en fonction des actions utilisateur (par exemple, le cache d'une simulation de fluide).

Type	Aspect	Date de développement
action	Courbes d'animation	09-2019
armature	Squelette d'animation	09-2019
camera	Point de vue	07-2019
collection	Hiérarchise les objets de la scene	03-2019
curve	Courbe paramétrique	09-2019
file	Fichiers externe à *Blender*	09-2020
greasepencil	Dessin dans le *viewport* 3D	03-2019
image	Image utilisée comme texture	07-2019
lattice	Boite de déformation	04-2019
light	Lumières	04-2019
lightprobe	Point de mesure du lighting (Eevee)	11-2019
material	Matériau d'un objet	03-2019
mesh	Maillage d'un objet	03-2019
metaball	Surfaces organiques	09-2019
node_group	Sous-groupe de nœud	08-2019
objects	Élément instantiable dans la scène	03-2019
particle	Système de particules et de génération de poils et cheveux	03-2020
scene	Élément d'organisation de l'espace	03-2019
sound	Fichier de son	09-2020
speaker	Son spatialisé dans la scène (utilise le sound)	11-2019
texture	Image paramétrique 2D ou 3D	11-2019
volume	Volumes 3D (par exemple, fumée)	11-2019
world	Arrière-plan et atmosphère de la scène	05-2019

TABLE II.1.2 – Liste des implémentations actuellement en place dans le *Multiuser*.

Dans un souci de performance, nos implémentations ne répliquent que les données de création utilisateur. Cela permet de déléguer la responsabilité de la génération des caches au DCC de chaque client et d'éviter d'envoyer des *deltas* trop lourds à travers le réseau. Ces implémentations permettent aussi de construire des filtres stoppant temporairement la réplication du *datablock*. Dans le *Multiuser* nous avons exploité cela pour permettre à un artiste de stopper temporairement la réplication des paramètres de rendu (cf. Sous-section II.1.3.3).

Le développement ainsi que la maintenance de ces implémentations n'ont pas toujours été aisés, car *Blender* est un logiciel en constante évolution. Au cours de ce processus, deux difficultés principales ont été rencontrées :

— l'absence ou le manque de fonctions dans l'API pour manipuler certains types de *datablocks* ;

— les changements réguliers de l'API Python.

L'API de *Blender* représente le seul point d'entrée dans le DCC pour manipuler

programmatiquement les *datablocks* (si l'on exclut la modification du code source qui requiert la création d'une version différente de la version officielle). Par conséquent, si BPY ne permet par de manipuler certains types de données, il est impossible de créer une implémentation correspondante pour les supporter dans le *Multiuser*. Ce problème est dû au rythme élevé du développement de *Blender* : en ajoutant de nouvelles fonctionnalités très régulièrement, les développeurs ne pensent pas toujours ou manquent de temps pour créer des *bindings*[172] Python nécessaires exposer ces fonctionnalités dans l'API. Nous avons rencontré ce souci pour quelques *datablocks* tels que les *lightprobes* qui étaient alors impossibles à créer correctement depuis BPY. Mais *Blender* étant open source, nous avons pallié ce problème en proposant un correctif[173] [174] ajoutant le code manquant pour compléter l'API. Par la suite, les *lightprobes* ont été supportés dès que le correctif fut accepté.

Dans le cycle de vie du logiciel, l'API est régulièrement amenée à changer pour accompagner les nouvelles fonctionnalités et modifications développées. Avec l'arrivée de la version 2.8 en 2018 et 2019, *Blender* a connu des changements très importants qui ont impliqué un nombre élevé de modifications de son API. Au regard du *Multiuser*, ces transformations de l'API étaient susceptibles de casser les implémentations existantes. Il était donc critique de trouver un moyen de les détecter rapidement. Pour y pallier, nous avons configuré l'intégration continue[175] sur l'hébergeur du code source (ici *GitLab*) pour lancer régulièrement des tests unitaires sur chacune des implémentations avec les dernières versions de *Blender* publiées. Ainsi, lorsqu'un changement d'API se produisait, les tests ne passaient plus et nous pouvions rapidement agir en conséquence.

II.1.3.2 Présence des utilisateurs

Au sein du *Multiuser*, la présence des utilisateurs connectés est assurée par deux éléments d'interfaces : les *widgets*[176] 3D et un panneau d'interface 2D. Pour récupérer et mettre à jour les informations représentées, ils s'appuient principalement sur les fonctions `update_user_metadata()` et `get_users_states()` du *framework* (cf. Sous-section II.1.2.5)

Les *widgets* 3D sont responsables de la représentation des informations clients dans le *viewport* 3D (un exemple de *widgets* est donné en Sous-section II.1.2.5). Dans la pratique, chaque *widget* contient un code de dessin (*shader*) exécuté à chaque mise à jour des métadonnées client. À l'heure actuelle, pour chaque client connecté, les *widgets* suivants sont exécutés :

— User Frustum : dessine le point de vue de l'utilisateur ;

172. En programmation, un *binding* est un point d'entrée permettant à une API d'accéder à des fonctionnalités ou structures internes du logiciel.

173. Un correctif ou *patch*, est une section de code que l'on ajoute à un logiciel pour y apporter des modifications.

174. Correctif proposé pour étendre l'API des *lightprobes* : `https://developer.blender.org/D6396`

175. L'intégration continue (ou *continuous integration*) est une pratique consistant à automatiser l'intégration des modifications du code provenant de plusieurs contributeurs dans un seul projet logiciel. Par exemple, lorsqu'un développeur publie le code d'une nouvelle fonctionnalité, l'intégration continue permettra d'automatiser le lancement de tests sur ce code pour le vérifier puis de le déployer en production.

176. En informatique, un *widget* désigne un élément visuel d'une interface graphique (par exemple, un bouton).

— User Selection : dessine la sélection active de l'utilisateur en se basant sur la *bounding box*[177] des objets ;

— User Mode : dessine le mode actif dans lequel se trouve l'utilisateur ;

— Username : dessine le pseudo de l'utilisateur.

Le panneau d'interface (voir la Figure II.1.10) prend son inspiration des tableaux de score utilisés dans l'univers du jeu vidéo. Il est divisé en une zone d'affichage (encadré en rouge) et d'une zone d'action (encadrée en vert), que nous décrirons en Sous-section II.1.3.3). La zone d'affichage liste les utilisateurs connectés avec leurs informations importantes (stockées dans les métadonnées, cf. Sous-section II.1.2.5) :

— *user* : affiche le pseudo de l'utilisateur et une icône de bouclier si ce dernier est administrateur ;

— *mode* : décrit sur quel aspect travail l'utilisateur en affichant son mode actif (par exemple, modélisation, animation, etc.) ;

— *frame* : *frame* actuelle de l'utilisateur, situe l'artiste dans la temporalité de la scène ;

— *location* : scène dans laquelle travaille l'utilisateur (dans *Blender*, un même projet peut avoir plusieurs scènes - espaces de travail). Cette entrée permet de localiser l'utilisateur dans le cas où le *widget* 3D est hors champ.

— *ping* : latence de l'utilisateur en millisecondes, utile pour déceler des problèmes de connexion.

Toutes ces informations situent les utilisateurs connectés dans toutes les dimensions de l'espace de création en répondant aux questions : où, quand et comment les artistes travaillent en temps réel. Ces outils ont la responsabilité de maintenir la conscience de la collaboration au sein de la session en assurant la présence de tous les utilisateurs connectés.

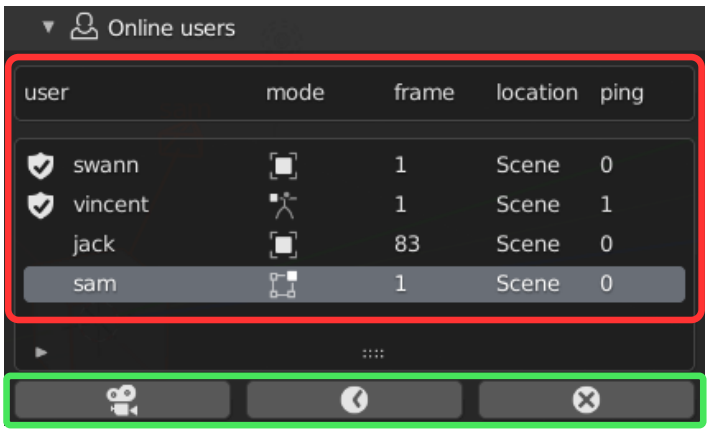


FIGURE II.1.10 – Panneau affichant la liste des utilisateurs connectés à la session.

II.1.3.3 Outils de collaboration

Les outils de collaboration sont des opérateurs développés dans l'optique de créer des interactions entre les utilisateurs connectés. Ils ont directement émergé du besoin

177. La *bounding box* est la boîte minimale encadrant un ensemble de points en N dimensions.

des artistes durant les sessions expérimentales (cf. Chapitre II.2).

L'outil de suivi spatial (ou *spatial snapping* en anglais) est accessible via le bouton caméra dans la zone d'action (encadrée en vert dans la Figure II.1.10). Il s'inspire du principe du mode spectateur utilisé en jeu vidéo qui consiste à suivre le point de vue d'un des joueurs en temps réel. Lors de l'activation de l'outil, l'artiste voit son *viewport* synchronisé avec celui de l'utilisateur qu'il a sélectionné dans la liste (encadrée en rouge dans la Figure II.1.10). En synchronisant les points de vue en temps réel, cet outil facilite les discussions entre artistes. Il permet aussi de retrouver facilement quelqu'un dans la scène, car l'utilisateur se téléporte dans la scène auprès de la représentation symbolique de la personne.

L'outil de synchronisation temporelle (ou *time snapping* en anglais), accessible via le bouton d'horloge dans la zone d'action (cf. Figure II.1.10), a été ajouté fin 2019 lorsque nous avons commencé à explorer l'animation collaborative. Il est né du besoin qu'avaient les animateurs à observer la scène dans la même temporalité pour l'étudier ensemble. Cet opérateur se calque sur le même concept que celui de suivi spatial mais appliqué au temps : il donne la capacité aux artistes de synchroniser leur *timeline* les uns sur les autres.

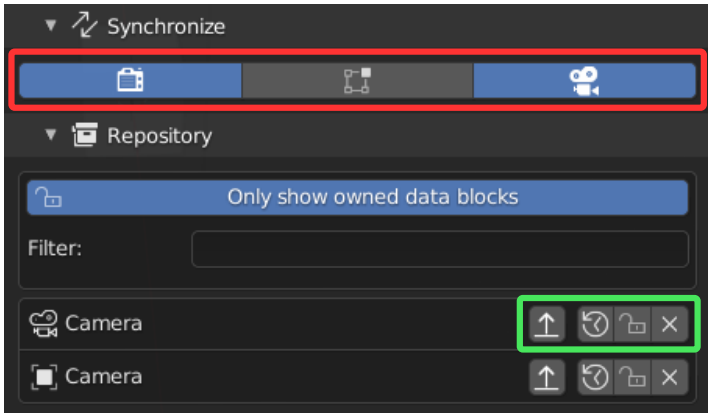


FIGURE II.1.11 – Outils de synchronisation et de gestion de droits.

La Figure II.1.11, montre les panneaux qui contiennent des opérateurs relatifs à la synchronisation des données. Leur usage est moins fréquent, mais peut s'avérer fort utile dans certaines situations. Dans l'onglet *synchronize* (en rouge sur le schéma), on retrouve les options qui permettent d'ajuster globalement le comportement de la réplication. De gauche à droite se trouvent les boutons pour activer/désactiver la synchronisation des *datablocks* suivants :

— les paramètres de rendus de la scène : les désactiver permet aux artistes de travailler sur les rendus sans impacter les autres ;

— les objets sélectionnés en mode d'édition : la désactivation permet d'éviter d'envoyer les mises à jour des objets pendant leur édition, utile pour optimiser les performances lorsqu'on travaille sur des objets complexes ;

— la caméra active de la scène : permet aux artistes d'essayer des points de vue sans influencer les caméras existantes synchronisées.

Lorsqu'un niveau de détail plus précis est nécessaire, l'onglet *repository* (en vert dans la

Figure) permet d'effectuer certaines opérations sur des nœuds spécifiques du *repository*. On retrouve de gauche à droite les opérations suivantes :

— `push` : cf. Sous-section II.1.2.4 ;

— `revert` : réinitialise le *datablock* sur la version du *repository* local ;

— `lock/unlock` : cf. Sous-section II.1.2.4 ;

— `delete` : cf. Sous-section II.1.2.4.

II.1.3.4 Mécanismes de droit

Comme expliqué précédemment (cf. Sous-section II.1.2.3), par défaut, tous les nœuds du *repository* sont soumis au droit commun. Pour récupérer/relâcher les droits de modification, le *framework* met à disposition deux commandes : `lock` et `unlock`. Dans les premiers prototypes du *Multiuser*, les utilisateurs devaient faire appel manuellement à ces commandes avant de commencer à travailler sur un *datablock*. Ce processus étant extrêmement fastidieux, nous avons rapidement basculé vers une stratégie automatique basée sur la sélection.

Le système se repose sur un *timer* responsable de la synchronisation des droits des nœuds avec la sélection. Si un objet est sous le droit commun, il est sélectionnable ; dans le cas contraire, il sera verrouillé à toute sélection. Cela évite la situation conflictuelle où deux personnes sélectionnent le même objet. Avec ce système, lorsqu'un objet de la scène est sélectionné, la fonction `lock` est appelée. Si cet objet n'est pas déjà verrouillé, le client récupérera les droits de modification sur ce dernier (cet objet ne sera alors plus sélectionnable par les autres clients) et ses dépendances (si elles n'appartiennent pas déjà à un autre client). À l'inverse, la désélection d'un *asset* déclenche un appel à `unlock` qui cède l'objet et ses dépendances au droit commun. Ce mécanisme agit sur chacun des objets composant la sélection, il fonctionne donc avec la multisélection.

En automatisant la gestion de droit dans l'intégration du *framework*, nous sommes parvenus à la rendre plus transparente et intuitive pour les artistes. Cependant, dans certaines situations, il peut être utile de reprendre possession de certains *datablocks* manuellement (par par exemple, verrouiller toute la scène pour éviter qu'un utilisateur présent à titre de spectateur ne provoque des modifications accidentelles). L'opérateur `lock/unlock` (dans l'encadré vert de la Figure II.1.11) a été conçu à cet effet.

II.1.3.5 Une gestion des mises à jour tumultueuse

Pour guider automatiquement le pipeline de réplication du *framework* (cf. Sous-section II.1.2.4), il a été crucial de trouver un moyen de détecter les mises à jour des *datablocks*. Au sein du *Multiuser*, nous avons eu deux approches : l'approche orientée surveillance et l'approche orientée évènements.

Depuis la version 2.8 de *Blender*, les mises à jour du *viewport* sont guidées par le graphe de dépendance de la scène. Ainsi, lorsqu'un *datablock* de la scène, visible au *viewport*, est modifié (le maillage d'un *mesh* par exemple), la scène est réévaluée et le *viewport* est actualisé en conséquence. Par chance, *Blender* expose les mises à jour du graphe de dépendance dans un *handler* : `bpy.app.handlers.depsgraph_update_post`. Mais jusqu'à mi-2020, ce *handler* était inexploitable, car il ignorait certains évènements critiques déclenchés par les utilisateurs.

Par conséquent, dans la version initiale, la méthode orientée surveillance consista à développer un *timer* chargé de vérifier à intervalles réguliers si les *datablock* sélectionnés étaient modifiés. L'intervalle d'exécution de ces vérifications pouvait être modifié à la volée par l'utilisateur. La fréquence ajustable de réplication représentait l'avantage principal de cette technique, mais s'avéra bien plus lourde qu'une approche orientée événement.

À partir de mai 2020, le graphe de dépendance de blender avait acquis assez de maturité pour être utilisé. Le système de mise à jour du *Multiuser* fut donc basculé vers une approche orientée événement. Cette approche nous permit, d'une part, de gagner en réactivité (car le *handler* notifie le *framework* dès qu'un changement se produit), d'autre part, de gagner en performance (car aucune vérification de changement n'est nécessaire).

II.1.3.6 Historique des actions

Dans un DCC, l'historique est une fonctionnalité importante qui permet à l'artiste d'annuler ou de rétablir une ou plusieurs actions précédentes. Sans l'historique, l'artiste n'a pas le droit à l'erreur. Lors de la création d'une expérience collaborative la question de l'historique utilisateur arrive très tôt, car les artistes utilisent quotidiennement cette fonctionnalité partout dans leur DCC.

Dans le *Multiuser*, nous avons exploré deux stratégies d'historique :

— global : tous les utilisateurs partagent le même ;

— local : chaque utilisateur a son propre historique.

L'historique global est devenu problématique dès les premières sessions expérimentales. Les artistes annulaient régulièrement des changements qui n'étaient pas les leurs. Ce comportement était naturellement contre-productif. Nous avons donc opté pour un historique local à chaque utilisateur, mais ce fut la fonctionnalité la plus compliquée à stabiliser (stable depuis seulement décembre 2020).

Pour comprendre nos difficultés sur le support de l'*undo* (annuler)/*redo* (rétablir), une brève explication du fonctionnement de la gestion historique de *Blender* est nécessaire. Pour chaque opérateur exécuté par l'artiste, *Blender* ajoute une étape à la pile d'historique au moment de son exécution.

Jusqu'à juillet 2020, *Blender* utilisait uniquement un système d'*undo* global : chaque étape d'historique stockait une version complète de tous les *datablocks* qui composaient la scène à un instant *t*. Ainsi, lors d'un *undo*, tous les *datablocks* de l'étape précédente étaient simplement restaurés en mémoire. Cette stratégie de gestion de l'historique a le bénéfice d'être rapide à développer, mais demeure sous-optimale : pour chaque retour en arrière, *Blender* rechargeait l'intégralité des éléments de la scène, même ceux n'ayant pas changé. Sur des scènes de taille importante, cet algorithme est très lent [178], par conséquent il cause beaucoup de problèmes dans les productions.

Au regard du *Multiuser*, ce système causait beaucoup d'erreurs et d'instabilités, car les *datablocks* répliqués devenaient systématiquement invalidés. Par conséquent, nous recommandions de ne pas utiliser l'*undo* pendant les sessions. En juillet 2020, le système d'*undo* a connu un correctif notable qui consista à détecter les *datablocks* non

178. Comme en témoignent les utilisateurs dans ce ticket : `https://developer.blender.org/T60695`

changés et à les réutiliser au lieu de les recharger. Cette optimisation fût l'élément déclencheur qui nous permit de stabiliser l'utilisation de l'*undo* local au sein du *Multiuser* en décembre 2020.

Conclusion

Ce chapitre a fait état des développements menés pour supporter la mise en place d'une expérience de création multi-utilisateur au sein d'un logiciel de création numérique 3D.

La première section a introduit la notion de *datablock* et de graphe de dépendance, deux composants logiciels communément utilisés au cœur des DCC 3D que nous exploitons dans nos outils de collaboration.

Le *framework* exposé en seconde section supporte les fondations nécessaires à la création d'une expérience collaborative. En s'appuyant sur une approche mono-DCC, le protocole RDP (voir Sous-section II.1.2.2) supporte un *dataflow* non-destructif adaptant ainsi les possibilités de la collaboration aux spécificités des logiciels de création. Les fonctions conscientes de collaborations concilient le *workflow* collaboratif aux contraintes des logiciels de création et aux besoins des artistes. Cet outil fut présenté dans le cadre de la conférence WSCG 2021 [13].

Lors de la troisième section, nous avons détaillé les spécificités de l'intégration du *framework* dans *Blender*. Nous avons ainsi constaté les forces et faiblesses de la conception d'une expérience multi-utilisateur au sein d'un programme transparent à la collaboration. Bien que tributaire des défauts du logiciel sur des problématiques d'historique, d'API et de performances, cette approche nous permit d'avoir une première version fonctionnelle en six mois de travail, supportant de nombreux aspects de la fabrication d'un film d'animation.

Cependant, le *framework* et le *Multiuser* n'auraient pas existé sans les nombreuses sessions expérimentales qui en dessinèrent le besoin et la forme.

Chapitre II.2

Mise en situation de l'outil : Expériences de co-création en temps réel

Introduction

L'*add-on Multiuser* présenté dans le chapitre précédent a été développé pour explorer l'impact de la collaboration en temps réel dans le processus de création d'images de synthèses pour le film d'animation. Nous avons démarré l'aventure des sessions expérimentales dès le premier prototype utilisable en juillet 2019.

Au cours de cette thèse, nous avons mené 129 sessions de création (soit environ 300 heures) qui ont abouti à la création de 38 scènes. Ces sessions ont pris place dans trois milieux différents :

— industriel : à Cube Creative avec des artistes de la société ;

— académique : dans le cadre de cours à l'Université Paris 8 ;

— public : sur internet avec une communauté internationale.

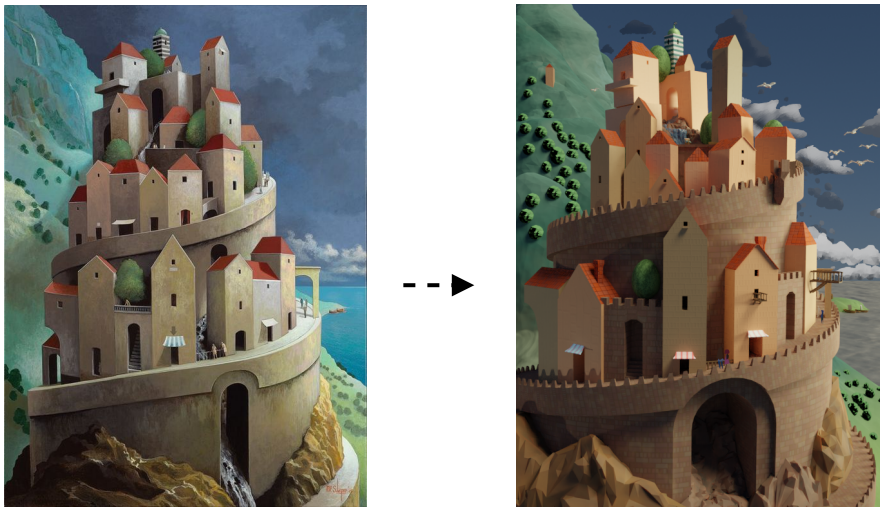
Ces trois contextes ont questionné les apports de la collaboration dans le processus de fabrication de films (milieu industriel), pour l'enseignement des techniques de création 3D (milieu académique) et pour la création en image de synthèse plus générale avec des artistes qui ne se connaissent pas et qui ne sont pas réunis physiquement au même endroit (milieu public).

Dans ce chapitre, nous exposerons le protocole qui structure ces sessions puis explorerons les points remarquables qui sont ressortis au regard des problématiques de chacun des milieux explorés.

II.2.1 Protocole d'expérimentation

Nous distinguons deux grandes catégories parmi les projets réalisés en session : ceux en création libre et ceux en création orientée (ou reproduction de scènes en 3D).

Les projets de création libre se veulent sans contraintes, les artistes ont la liberté de créer ce qu'ils veulent dans la scène. Leur but était d'étudier l'émergence d'un consensus artistique. À l'inverse, les exercices de recréation, beaucoup plus cadrés, visent à créer une scène à partir d'une référence 2D (cf. Figure II.2.1). Cette référence est choisie par un vote collégial entre artistes. Selon le niveau de détail de la référence, l'exercice de recréation laissera plus ou moins de place à l'interprétation. Dans les deux exercices, les artistes créent une œuvre en partant d'une scène vide.



Référence 2D — Scène créée durant la session

FIGURE II.2.1 – Exemple du résultat de l'un des exercices de recréation (*Surreal Architecture*). Crédits en Annexe E.

Lors des sessions de création, nous avons enregistré le plus de données possibles en vue de leur analyse. Ces informations se présentent en trois catégories :

— les captures vidéo d'écran ;
— les fichiers de travail ;
— les exports du *repository* de la session.

Les captures vidéo sont les enregistrements d'un client spectateur connecté à la session et verrouillé sur le point de vue d'une caméra fixe. Elles permettent de rejouer la session avec le même point de vue que la caméra fixe et de constater visuellement l'évolution de la scène en évaluant qui travaille sur quel *asset* et à quel moment. Elles ont également été utilisées pour réaliser des *timelapses*.

Les fichiers de travail (des `.blend`) sont des sauvegardes des scènes *Blender*. Elles ont été utilisées pour évaluer la distribution effective du travail ainsi que l'évolution de la scène en fonction du temps. En revanche, ces informations sont insuffisantes pour garder une trace de l'activité précise des utilisateurs (le comportement de la collaboration), car ces fichiers ne stockent que les données de création relatives à la scène *Blender*.

Pour pallier ce problème, en novembre 2020 nous avons ajouté au *framework* la capacité d'exporter le *repository* dans un fichier (cf. Sous-section II.1.2.6). Durant chaque

session, nous avons fait un export automatique du *repository* toutes les vingt secondes. Le *repository* stockant toutes les informations relatives à la collaboration, cela nous permit d'analyser a posteriori quelles étaient les activités créatives entreprises par chaque utilisateur et à quel moment. Cette étude s'est majoritairement faite directement dans *Blender* en important la sauvegarde via le *Multiuser*. Mais nous avons aussi expérimenté d'autres méthodes de visualisation via la création de scripts Python lisant directement les fichiers `.db` (sauvegardes du *repository*, cf. Sous-section II.1.2.6). À titre d'exemple la Figure II.2.2 montre l'évolution du graphe de réplication basée sur les sauvegardes (nous avons sélectionné des sauvegardes à trente minutes d'intervalle dans un souci de lisibilité). Nous avons utilisé les bibliothèques *NetworkX* et *Mathplotlib* pour créer une représentation visuelle des données enregistrées (le script Python figure en Annexe D). Cette vue schématique donne une idée assez précise de qui travaille sur quel *datablock* à quel moment.

De nombreux paramètres pourraient y être ajoutés, par exemple le diamètre des nœuds pourrait varier fonction de la taille des données de création associées.

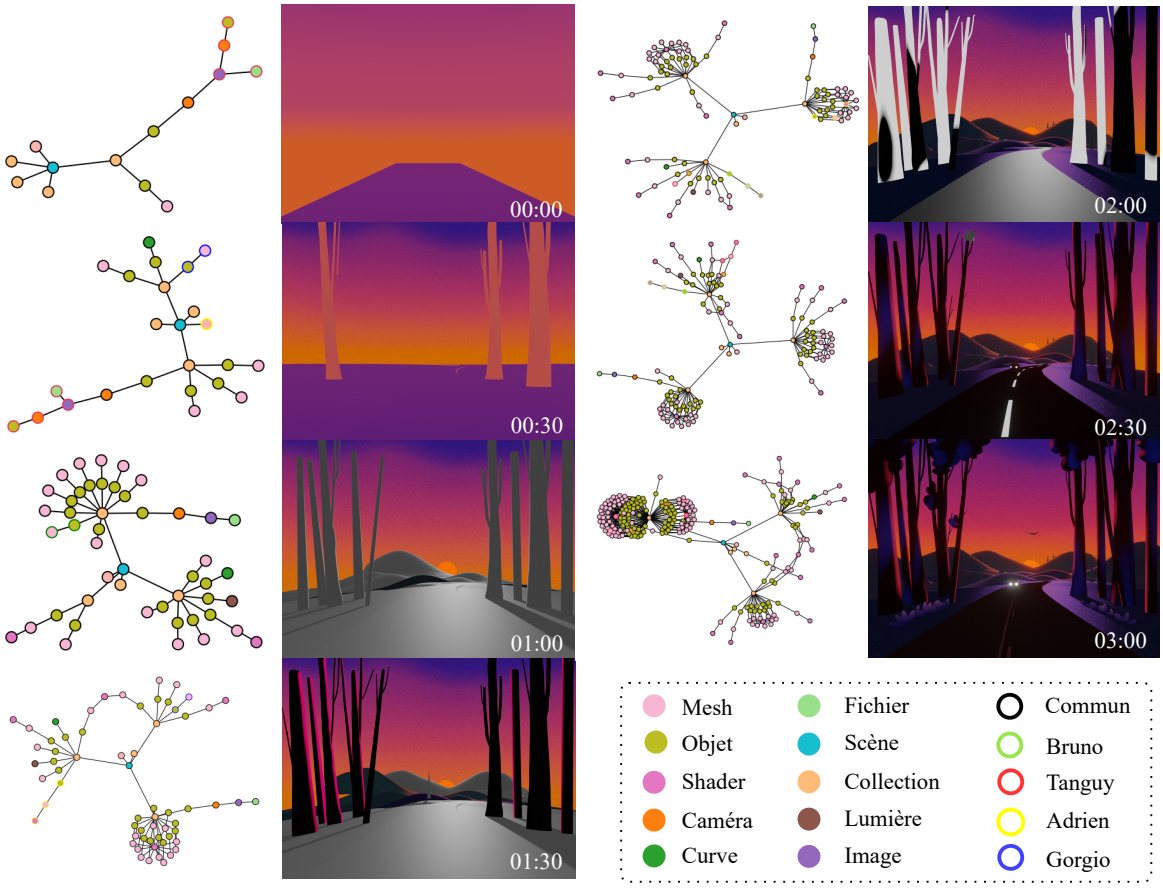


FIGURE II.2.2 – Visualisation de l'évolution du graphe de réplication générée à partir des sauvegardes du *repository* du projet *Travel*. Crédits en Annexe E.

Au total, nous avons accumulé 283 Go de données de création collaborative entre 2019 et 2021. Les observations établies dans les sections suivantes se basent directement sur ces matériaux (fichiers de scènes, enregistrements vidéos et exports du *repository*).

II.2.2 Expérimentations académiques

Actuellement, dans une classe d'enseignement de création en image de synthèse (par exemple, lors d'une leçon de modélisation sur *Maya*), le professeur dispense généralement le cours depuis son ordinateur projeté sur un mur. Les étudiants, derrière leurs postes de travail, suivent ce qui est montré et (selon le cours) l'appliquent ensuite dans le logiciel de création enseigné. Ce processus d'enseignement peut poser plusieurs problèmes. Les concepts enseignés sont parfois complexes, or lorsque la période entre démonstration et application est longue, les étudiants peuvent se trouver rapidement égarés et ne pas réussir à appliquer correctement le concept.

Nous pensons qu'utiliser la collaboration en temps réel au cœur de l'enseignement en image de synthèse réduirait la distance théorie-pratique et faciliterait ainsi la mémorisation du concept enseigné.

L'utilisation d'ordinateurs crée naturellement une distance communicationnelle entre le professeur et les étudiants, mais également entre les étudiants. Pour l'enseignant, cette barrière empêche de voir facilement le travail des élèves, compliquant la détection des élèves en difficulté. Pour les étudiants, cette forme d'isolement ne favorise pas l'entraide et l'émulation, car les élèves ne voient que leurs travaux individuels. Effectuer le cours au sein d'un outil multi-utilisateur amènerait le professeur et les étudiants à partager le même espace virtuel ce qui pourrait faciliter la communication professeur-élève et élève-élève.

Pour explorer ces hypothèses, nous avons organisé trois ateliers de formation à la création d'images de synthèse sur *Blender* pour les doctorants du laboratoire de l'équipe INREV et les étudiants en master 2 ATI. Ces sessions se déroulèrent entre juillet 2019 et février 2020 à l'Université Paris 8 dans les locaux de l'INREV. Le *Multiuser* a été exploité comme outil pédagogique pour enseigner différentes notions relatives à la création 3D telles que :

— la modélisation ;
— le *shading* ;
— le *lighting* ;
— le travail créatif en équipe.

II.2.2.1 Enseignement de la modélisation 3D

Le premier cours se déroula en juillet 2020, l'objectif était d'apprendre les bases de la modélisation dans *Blender* à huit élèves. Le résultat de ce premier cours est visible dans la Figure II.2.3. Étant donné que certains élèves avaient déjà des bases en 3D, nous avons mis en place un *workflow* pouvant s'adapter au niveau de tous. L'idée fut de créer un environnement 3D en partant de zéro. Nous avons commencé par créer ensemble une représentation schématique (en vue de dessus) de la future scène destinée à servir de référence, au tableau. Le travail fut ensuite réparti collégialement parmi les étudiants en suivant un découpage spatial en sections (plages, mer, volcan, etc.). Les élèves choisissaient la section qu'ils voulaient créer en fonction de la difficulté de réalisation (et donc de leur niveau). Une fois la répartition des tâches effectuée, les élèves se connectèrent tous à une session hébergée sur l'ordinateur du professeur avec le *Multiuser* depuis *Blender*. Un écran spectateur était projeté au mur. Nous partagions tous ainsi le même espace virtuel et observions le travail de chacun en temps réel.

Nous avons enseigné chaque concept (la manipulation de *mesh*, la topologie, la modélisation non destructive) directement dans la scène partagée. Par conséquent, les étudiants avaient à disposition l'aperçu projeté au mur, plus le résultat de nos actions affiché en temps réel dans *Blender* sur leurs machines. En modélisation 3D, on est amené à enseigner les bonnes pratiques sur la création de la topologie de *mesh*. La scène 3D partagée en temps réel a permis aux étudiants d'observer les *meshes* étudiés depuis leur propre ordinateur tout en ayant la liberté de choisir d'autres angles de vue que celui imposé par la rediffusion de l'écran sur le mur.

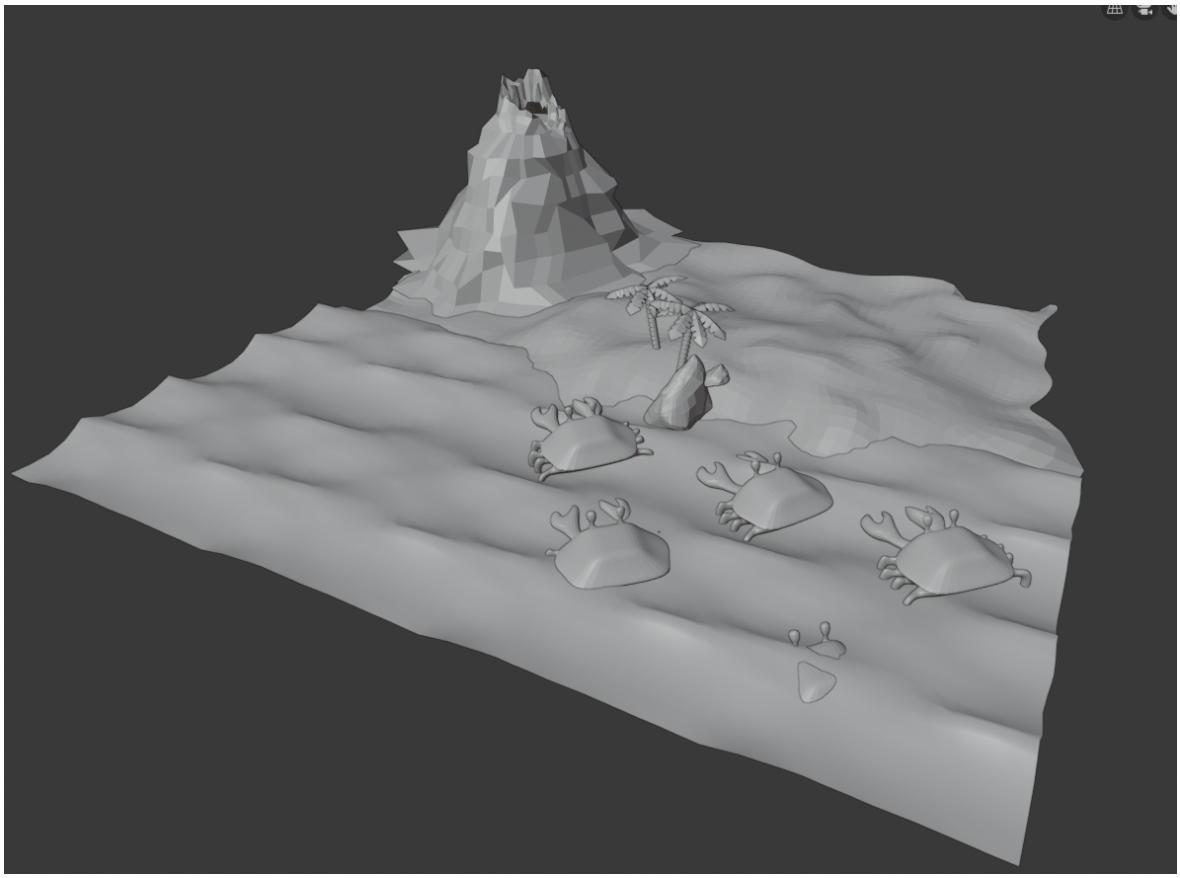


FIGURE II.2.3 – Résultat du cours d'enseignement collaboratif de la modélisation 3D sur *Blender*

Au fur et à mesure de la progression de l'exercice, les élèves rompaient le silence pour discuter entre eux et échanger des remarques constructives. En partageant la même scène, les étudiants déjà à l'aise constataient les difficultés de leurs camarades et leurs venaient naturellement en aide. Ce phénomène de transmission de savoir s'apparentait à ce que nous avions observé pour le *pair programming* (cf. Sous-section I.3.1.2). Cependant, ce constat n'a été fait qu'avec de petits effectifs. Il serait important d'expérimenter avec une classe de plus de quinze étudiants pour étudier si le niveau sonore reste viable.

Ce genre de retour est rare dans une production de cinéma d'animation car la hiérarchie verticale induit un phénomène de 'silo'. C'est généralement au superviseur de faire des *reviews* sur les travaux des artistes. Mais dans notre cas, les étudiants dirigeaient naturellement entre eux les *reviews* sans aucune intervention de ma part.

Mais l'ajout du travail collaboratif en temps réel dans ce cours a également apporté une dimension importante de coordination dans le rôle de l'enseignant. Certains élèves

débutants avaient du mal à se déplacer dans l'espace 3D, par conséquent nous devions gérer l'organisation spatiale du travail en indiquant à ces élèves où se localiser pour travailler. Un autre aspect de la coordination fut de verrouiller les droits de modification des objets de la scène situés dans les secteurs où le travail des étudiants se superposait pour éviter les changements non consentis. Pour ce faire, nous utilisions le mécanisme de verrouillage manuel expliqué en Sous-section II.1.3.3.

À travers ce premier cours exploitant le *Multiuser*, nous avons constaté que la collaboration en temps réel était très intéressante pour l'enseignement, en revanche, cela s'accompagna d'un effort de coordination non négligeable.

II.2.2.2 Enseignement du pipeline complet de création d'*asset* 3D

Le second cours se déroula en octobre 2019 avec cinq élèves de la première leçon. En ce qui concerne le contenu, il faisait suite au premier en proposant de compléter l'apprentissage de la création de modèles 3D jusqu'au *shading*. Comme pour le premier cours, l'objectif fut de concevoir une scène à partir d'une référence. En revanche la nature de la référence et son processus de sélection furent différents.

Durant la préparation du cours, j'ai sélectionné un ensemble de références en 2D (Illustrés sur la Figure II.2.4) qui correspondait aux notions enseignées et pouvait se réaliser en 3 heures sur *Blender*. Comme on le constate, ces références donnent des informations de matériaux, lumières et formes. Le but de cette démarche était de positionner les étudiants dans une situation similaire à une production où l'on crée les *assets* à partir de concept en 2D (cf. Section I.1.1).

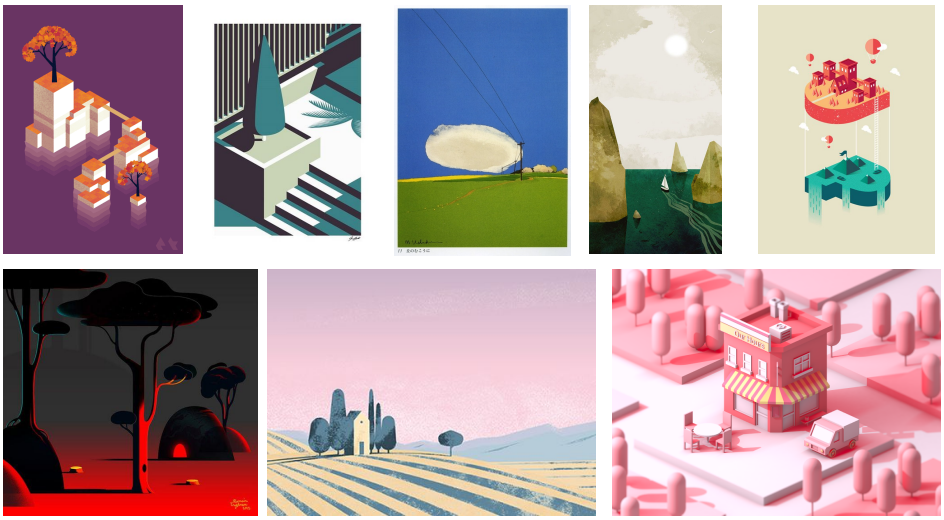


FIGURE II.2.4 – Sélection de référence proposée aux étudiants.

Au début du cours, nous avons fait un vote pour déterminer l'univers graphique que les étudiants préféreraient réaliser. Pour la division du travail, nous avons exploré une répartition basée sur segmentation d'image (voir Figure II.2.5b). Nous avons d'abord sélectionné une couleur distincte par étudiant qui correspondait à celle assignée dans le *Multiuser*. Puis nous avons coloré des zones dans l'image de référence, correspondant au travail que chaque élève devait effectuer sur la scène. Par exemple, l'étudiant bleu devait créer l'arbre central (modélisation et *shading*). Les sections d'images ont été assignées en fonction du niveau des étudiants.

(a) Référence retenue pour l'exercice. (b) Répartition du travail dans la scène.

FIGURE II.2.5 – Préparation du second cours avec les étudiants.

Le cours dura deux heures et demie. Le constat fut similaire à celui du premier cours : partager le même espace virtuel leur permit d'échanger naturellement et leur apprit à s'organiser mutuellement. Sur la Figure II.2.6 on peut observer que l'enseignant est situé très proche des élèves. Cette configuration, bien qu'elle soit compliquée à déployer pour une classe de plus de vingt étudiants, nous est apparue comme optimale pour les cours exploitant la collaboration en temps réel. En positionnant les élèves dans une configuration où ils sont tous proches, ils pouvaient s'entraider sans avoir besoin de se déplacer. Il en est de même pour le professeur qui peut assister les élèves en difficulté sans se déplacer.

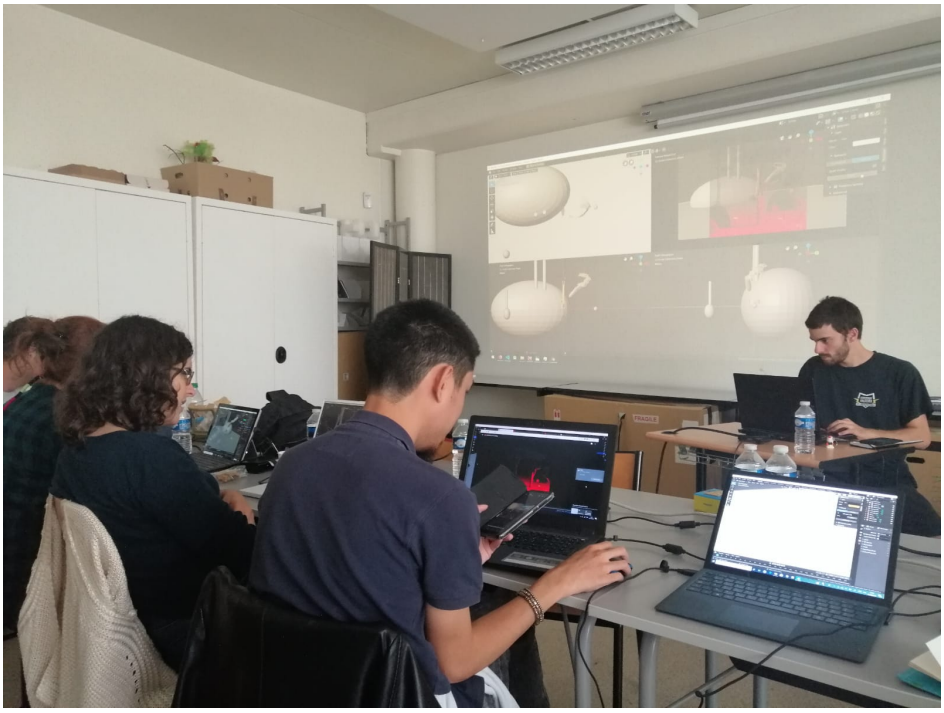


FIGURE II.2.6 – Étudiants durant le cours.

Sur la Figure II.2.7 on peut observer une capture d'écran du poste de travail de l'enseignant durant le cours. Le découpage de la fenêtre en quatre vues (ou *quadview*)

s'est avéré très pratique pour suivre les étudiants dans leur travail. La vue de la caméra (en haut à droite) permet de constater des erreurs de composition de la scène. Les trois autres vues orthographiques permettent de visualiser l'activité des étudiants hors champ.

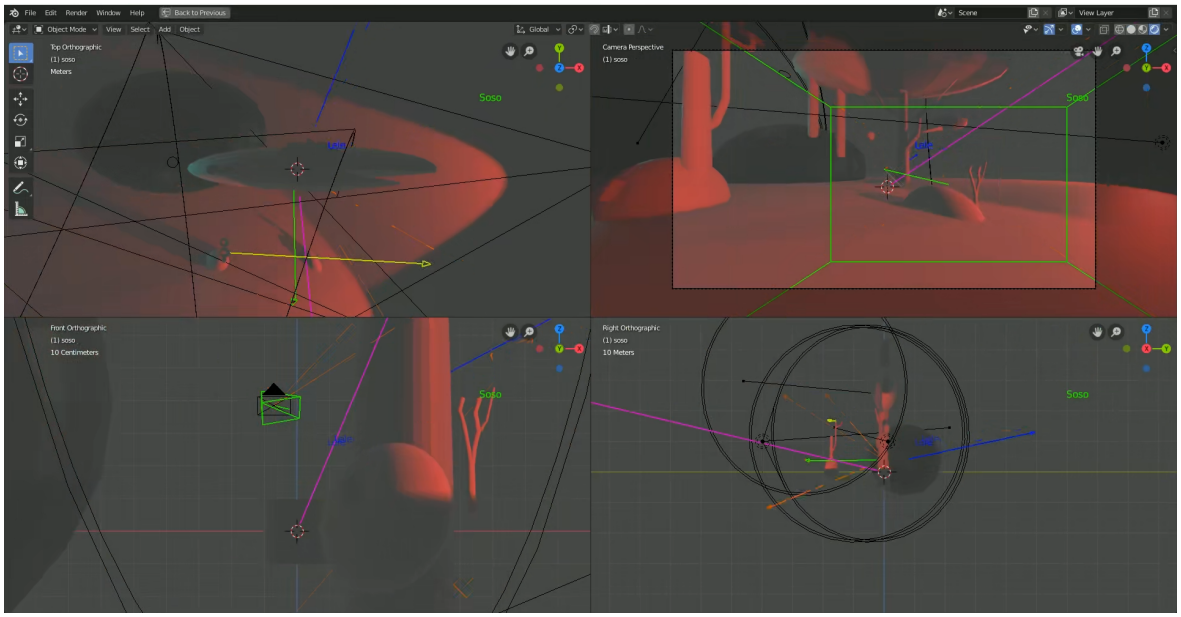


FIGURE II.2.7 – Scène en cours de création, du point de vue de l'enseignant.

Les élèves, ayant déjà assisté au cours précédent, étaient familiers du *Multiuser* et avaient déjà travaillé ensemble. Ce simple fait réduisit considérablement le besoin de coordination au regard du cours précédent.

II.2.2.3 Enseignement du *lighting*

Le troisième cours se déroula en février 2020 avec quatre élèves du cours précédent. L'objectif était d'aborder les bases du *lighting* sur *Blender*.

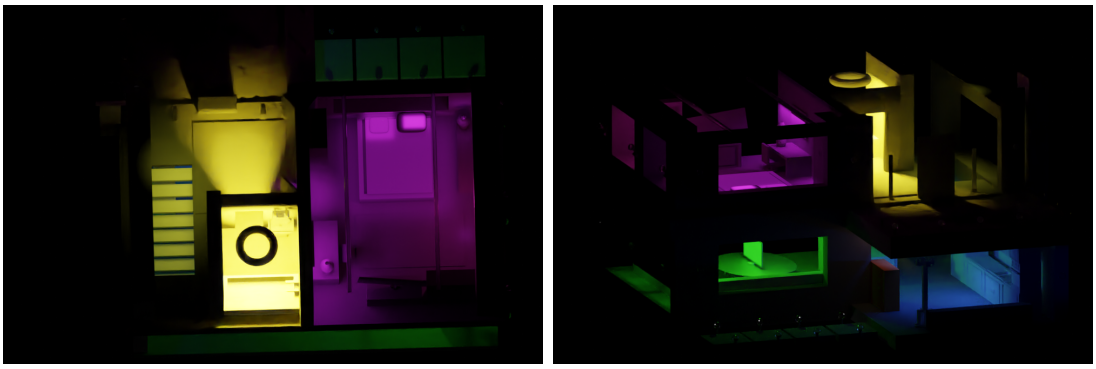
Pour recentrer les étudiants sur la réalisation d'une seule étape de fabrication, nous avons récupéré une modélisation 3D préexistante d'une maison qui constituerait l'environnement à éclairer sur *Blender*.

Avant de débuter l'exercice collectif, nous avons brièvement introduit les différentes catégories de lumières à disposition dans *Blender*. Pour répartir le travail, nous avons opté pour une division spatiale (comme pour le premier cours). Les élèves choisirent quelle pièce ils voulaient mettre en lumière. Pour visualiser la répartition du travail au terme de l'exercice, nous avons colorisé les lumières créées selon la couleur de représentation des élèves dans le *Multiuser* (cf. Figure II.2.8).

Sur la Figure II.2.9 on observe les étudiants travaillant dans la scène. Tout au long de l'exercice, *Eevee* (moteur de rendu temps réel de *Blender*) occupa un rôle crucial : il permit de visualiser en direct l'impact des lumières des uns sur celles des autres.

En explorant la collaboration temps-réel sous l'angle de l'éducation, nous n'avons qu'effleuré la surface d'une vaste possibilité d'applications tangibles pour l'enseignement en image de synthèse.

En utilisant l'*add-on Multiuser* au cœur du cours, l'enseignant partage le même espace virtuel que les étudiants. Il peut interagir avec leurs travaux pratiques en temps



(a) Répartition du travail vue de dessus. (b) Répartition du travail vue perspective.

FIGURE II.2.8 – Résultat de l'exercice de *lighting* en fonction de la répartition du travail dans la scène.

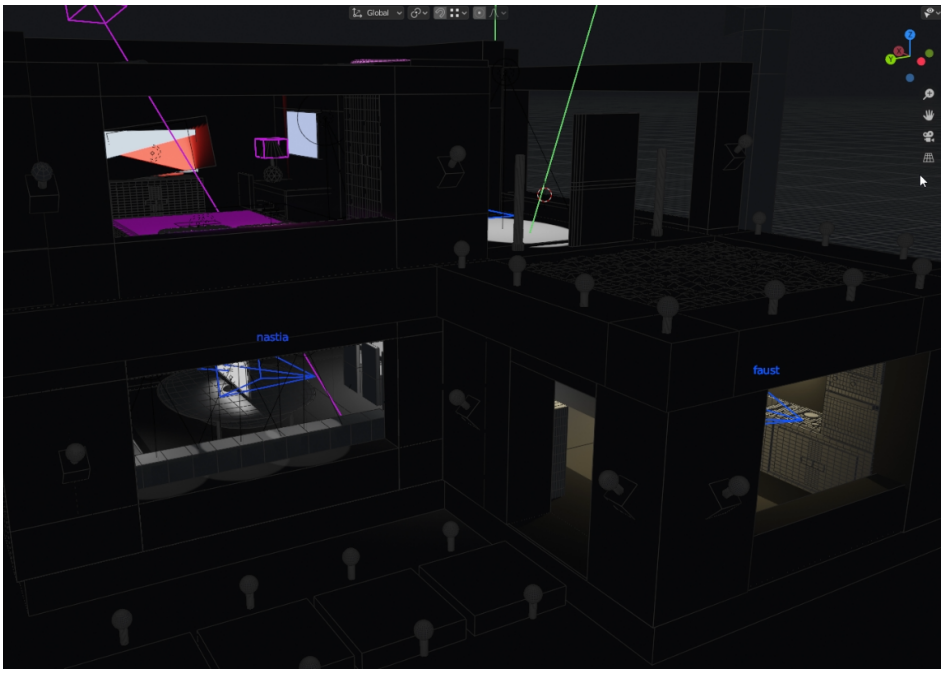


FIGURE II.2.9 – Création de l'éclairage par les étudiants.

réel et réagir promptement aux questions et difficultés de chacun. Pour les étudiants, la collaboration en temps réel offre un socle commun qui favorise l'entraide entre eux et conduit à une transmission de connaissance entre personnes de différents niveaux.

Nos travaux trouvent donc une application pertinente dans le milieu académique d'un point de vue pédagogique. Mais ce constat se fonde principalement sur des observations de la pratique. Il serait intéressant de mener une étude qualitative pour quantifier les différents apports énoncés et valider ces observations.

II.2.3 Expérimentations industrielles

Cette section fait état des expérimentations menées à Cube Creative. Entre septembre 2019 à juillet 2021, nous avons eu la chance de faire un total de trente sessions de création collaborative qui ont mené à la création de huit scènes. Elles nous ont permis de mettre en situation notre outil, dans le cadre concret de l'entreprise et avec des professionnels du cinéma d'animation, en vue d'explorer quelles seraient ses applications possibles dans la fabrication d'images animées.

Dans le Chapitre I.1 nous avions constaté que la production d'un film passe linéairement par de nombreuses étapes. Chaque étape est isolée et validée individuellement. Au sein de la chaîne de production, la collaboration en temps réel peut s'envisager au sein d'une ou de plusieurs de ces étapes de fabrication. De ces deux configurations vont découler deux compositions d'équipe très différentes. La collaboration entre plusieurs étapes de production nécessite une équipe pluridisciplinaire (par exemple, un.e artiste *layout*, un.e animateur.rice, un.e artiste rendu). À l'inverse, la réalisation collective d'une tâche donnée (par exemple le *layout*) requiert une équipe spécialisée.

Deux types d'exercices ont été conçus pour étudier l'apport de la collaboration au sein de ces deux configurations d'équipe :

— *background concept*[179] : la création d'un décor en partant d'une scène vide ;

— *recréation* : la création d'une scène en partant d'un concept 2D.

Pour l'exercice de *background concept*, les équipes sont spécialisées. À l'inverse, la recréation de scène englobe tous les aspects de la création et nécessite donc différents métiers.

Au fur et à mesure de ces exemples, nous observerons si de telles méthodes sont compatibles avec le pipeline exposé dans le Chapitre I.1 ou si elles nécessitent une réorganisation de l'appareil de production. Nous nous appuierons ensuite sur ces deux catégories d'exercices afin d'évaluer notre approche en termes d'efficacité et de qualité.

II.2.3.1 Un début mouvementé

Dans les premières sessions menées en septembre 2019, nous n'avions aucune idée vers où nous mèneraient nos pérégrinations avec le *Multiuser*, qui dans ses premières versions était très rustique. Il ne supportait alors que quelques types de données (cf. Sous-section II.1.3.1) et la visualisation des utilisateurs connectés. Notre première session en *background concept* consista à plonger des superviseurs dans une scène préexistante (Figure II.2.10a). Les participants n'étaient alors pas familier de l'outil ; de plus nous avions donné qu'un seul conseil très vague :

Tentez d'embellir la scène ensemble.

L'aspect ludique de l'outil collaboratif s'est révélé dès le commencement de l'atelier avec de nombreuses questions échangées entre les participants. (« Comment as-tu fait telle chose », etc.) Mais très rapidement, en constatant que leur travail influençait celui des autres, les superviseurs se sont mis à jouer ensemble pour déconstruire le travail et reconstruire autre chose. La Figure II.2.10b montre la scène chaotique résultant de la session. On constate que le visuel (cf. Figure II.2.10a) a été nettement dégradé.

Sans autorité pour canaliser leur créativité, les superviseurs se gênaient mutuellement. Nous avons nommé ce phénomène de dérive observé l'effet « bac à sable » ou

179. Le *background concept* ou concept de décor désigne l'étape de prototypage de nouveaux décors.

« *sandbox* ». Les constats établis durant cette session ont conduit à la création du module de gestion de droits dans le *framework* (cf. Sous-section II.1.2.3). En ajoutant un champ d'appartenance (stockant l'identifiant du propriétaire) aux *datablocks* de la scène, il était désormais possible de restreindre leurs droits de modification pour éviter les conflits. La synchronisation du système avec la sélection active est arrivée au même moment dans le *Multiuser* (cf. Sous-section II.1.3.4)

(a) Scène initiale (b) Résultat de la session

FIGURE II.2.10 – Première expérimentation avec les superviseurs.

Cette session initiatique pour les superviseurs pava le chemin pour les suivantes au cours desquelles l'effet *sandbox* ne s'est plus manifesté.

II.2.3.2 Efficacité de la collaboration

Pour étudier l'impact de la collaboration en temps-réel sur l'efficacité d'exécution du travail, nous avons mené une série de sessions de recréation de scène à partir de références choisies pour leurs diversités. Ces sessions se sont déroulées à distance sur internet suite à la crise de la Covid-19 (son influence sur nos recherches est étudiée en Section II.2.4).

Chacune de ces scènes a été créée deux fois sur *Blender* : une version a été créée par une seule personne exécutant séquentiellement les différentes tâches, et l'autre version a été créée par une équipe collaborant en temps réel avec l'*add-on Multiuser* via internet.

Dans les deux cas, les artistes devaient créer les aspects suivants :

— modélisation : création de la surface géométrique des *assets* ;
— *shading* : définition des matériaux ;
— *layout* : placement du point de vue et des *assets* ;
— FX : création d'effets de particules et de simulations ;
— *lighting* : mise en lumière de la scène ;
— *compositing* : traitement du rendu de la scène.

Bien qu'il s'agisse d'une notion abstraite, les qualifications de l'artiste pour ces genres de tâches sont un facteur très important vis-à-vis des résultats à obtenir. Dans la Sous-section I.1.2.3, nous avons distingué trois niveaux d'expérience dans les équipes d'animation : les artistes juniors, mids (pour *middles* ou intermédiaires) et seniors. Dans l'expérience de création mono-utilisateur, les artistes avaient une expérience de généraliste mid de façon à pouvoir gérer tous les aspects de la création de la scène. Dans la partie multi-utilisateur de l'expérience, l'équipe de collaboration était composée à parts égales d'artistes juniors et mids.

Le Tableau II.2.1 montre dans quelle mesure la collaboration en temps réel peut accélérer le temps de création d'une scène. Ce gain varie en fonction de la complexité de la

Scene			SU	MU	
name	*tris*	*Oc*	*t*	*t*	*Ts*
Campsite	3430	161	63	45	3
Paper Summit	22215	58	90	60	5
All seing monolith	59000	146	177	90	4
Xbox clubs image	16112	726	238	175	4
Abstract city	2267256	139	472	295	5

TABLE II.2.1 – Temps des sessions de recréation (t) en minutes en configuration mono-utilisateur(SU) et multi-utilisateur(MU) avec la taille d'équipe correspondante(Ts), le nombre total de triangles ($tris$) et le nombre d'objets (Oc)

scène et de la taille des équipes. D'après nos données, les équipes de quatre artistes obtiennent le meilleur taux d'efficacité sur les scènes complexes. On observe qu'une équipe est plus rapide qu'un artiste seul. Mais proportionnellement au nombre de personnes, l'équipe est moins efficiente. L'efficacité de la méthode pourrait être grandement améliorée par une optimisation de la distribution du travail avant la session. Pendant les sessions de reproduction, nous n'avons pas imposé cela, et par conséquent, nous avons constaté que les artistes, par habitude, avaient tendance à travailler individuellement sur les *assets*. Ainsi, nous avons observé que le principal facteur limitant la parallélisation du travail était d'ordre organisationnel. L'objectif serait de faire se chevaucher la création de plusieurs aspects d'un même *asset* afin d'améliorer le parallélisme du travail collaboratif. Par exemple, un artiste pourrait construire un *asset* tandis qu'un autre le placerait dans la scène (approfondit en Sous-section II.2.4.7).

Cette étude aurait pu être approfondie en menant une analyse comparative avec un pipeline traditionnel où chaque étape de la reproduction aurait été linéairement effectuée par un individu différent. Elle n'a malheureusement pas été conduite pour raison de disponibilité des équipes de production à cette époque.

Les sessions de recréation ont mis en évidence d'autres avantages de la co-création en temps réel. Bien que travaillant individuellement sur les *assets*, les artistes ont été naturellement amenés à travailler simultanément sur plusieurs aspects de la scène. Par exemple, il était courant qu'un artiste commence l'éclairage pendant que d'autres modélisaient et disposaient les éléments. Par conséquent, tous étaient conscients de leur impact sur le travail des autres, en temps réel, ce qui a considérablement amélioré la gestion des erreurs en rapport à un pipeline industriel linéaire.

II.2.3.3 Intégration d'élément du pipeline

Les sessions de recréation exposées précédemment étaient complètement détachées du pipeline de Cube Creative. En revanche, en exploitant des *assets* de production, les sessions de *background concept* nous ont conduit à intégrer la *Resource Library* au sein du *Multiuser*. Par extension, nous en sommes venus à nous questionner sur l'intégration du *Multiuser* au sein du pipeline.

La *Resource Library* est un asset manager simplifié développé par Cube Creative. Elle permet aux artistes de sauvegarder et d'importer rapidement des *assets* stockés dans un disque réseau. Au sein des productions, elle est massivement utilisée pour créer des banques d'animation (un ensemble d'expressions faciales et corporelles animées

destinées à être utilisées comme base pendant l'animation des plans). L'objectif fut d'utiliser cet outil pour mettre les *assets* de la série *Tangranimo* à disposition des artistes durant la session. À l'époque, la *Resource Library* était prévue pour fonctionner avec la version 2.7 de *Blender* utilisée en production à Cube Creative. Un premier travail fut de la porter pour la version 2.8 *Blender* sur laquelle a été développé le *Multiuser*.

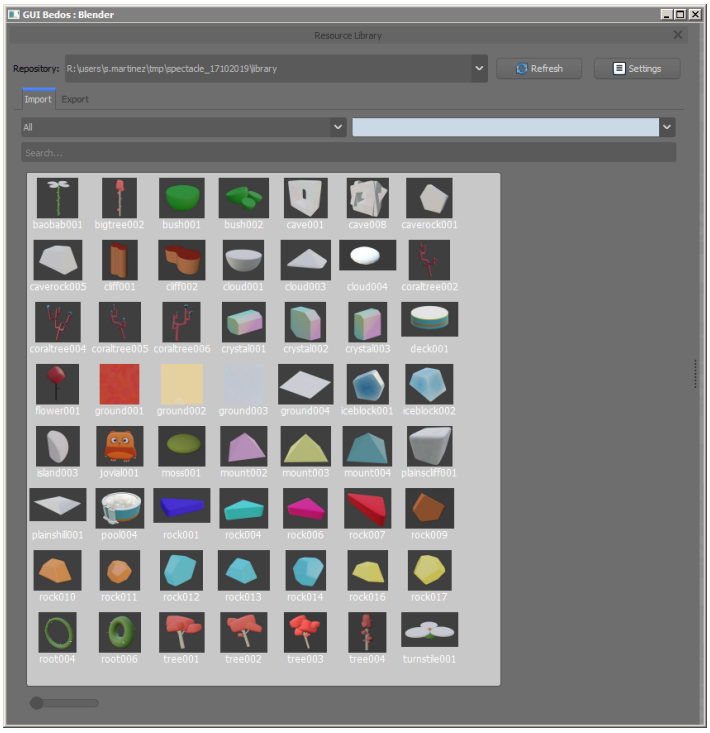


FIGURE II.2.11 – *Resource Library* utilisée pour les sessions de *background concept*.

Amener cet outil dans le *workflow* collaboratif posa des questions sur l'organisation structurelle des fichiers. En interne, la *Resource Library* fonctionne avec des références [180] d'*assets*, qui sont omniprésentes dans les scènes de production. Lorsqu'un artiste sélectionne un objet dans la librairie, une référence à cet *asset* est ajoutée dans la scène 3D. Dans *Blender*, ce mécanisme dénommé *link*, importe l'*asset* en lecture seule prévenant toute modification. Pour utiliser la *Resource Library*, il fallut donc ajouter le support des références au *Multiuser* qui n'était à l'origine prévu pour fonctionner qu'avec des *assets* stockés dans la scène. La fonctionnalité prit la forme d'une implémentation très simple stockant uniquement le chemin des références dans des nœuds dédiés au sein du graphe de réplication. Lors de l'ajout d'un *asset* de la *Resource Library*, le *Multiuser* réplique la référence en envoyant le chemin de stockage de l'*asset*, la responsabilité de charger la référence est ensuite donnée au mécanisme de *link* de *Blender* qui est automatiquement appelé au moment du `load()`(cf. Sous-section II.1.2.4) du nœud.

180. On parle de référence lorsqu'on pointe sur une donnée préexistante. Cela se dit souvent d'un fichier appelé depuis un autre fichier. Les références de fichiers (ou *file reference*) permettent d'assembler des objets 3D sans être contenu directement dans la scène 3D. Si le fichier en référence est modifié, les fichiers l'utilisant bénéficieront des modifications. Si le fichier en référence n'est plus disponible (supprimé, inaccessible, etc.), les fichiers l'utilisant ne pourront plus s'ouvrir correctement et/ou entièrement. Un objet référence contient simplement le chemin d'un autre fichier. Son utilisation apporte de la souplesse et facilite le travail collaboratif. Définition par LePipeline.org.

Nous avons alors découvert un bénéfice inattendu. L'utilisation des *assets* liés par référence dans les sessions améliore considérablement les performances du *Multiuser* , car il stocke et suit que le chemin du fichier de l'*asset* et non ses informations de création. Par exemple, lorsqu'un artiste instancie un *mesh* très lourd depuis la *Resource Library*, seule l'adresse du fichier de ce *mesh* sera répliquée à travers le réseau puis utilisée par les clients connectés pour laisser *Blender* le charger. Cet avantage a été exploité dans une optique parallélisation des taches, ainsi qu'expliqué Infra en Sous-section II.2.4.7.

II.2.3.4 Observations qualitatives

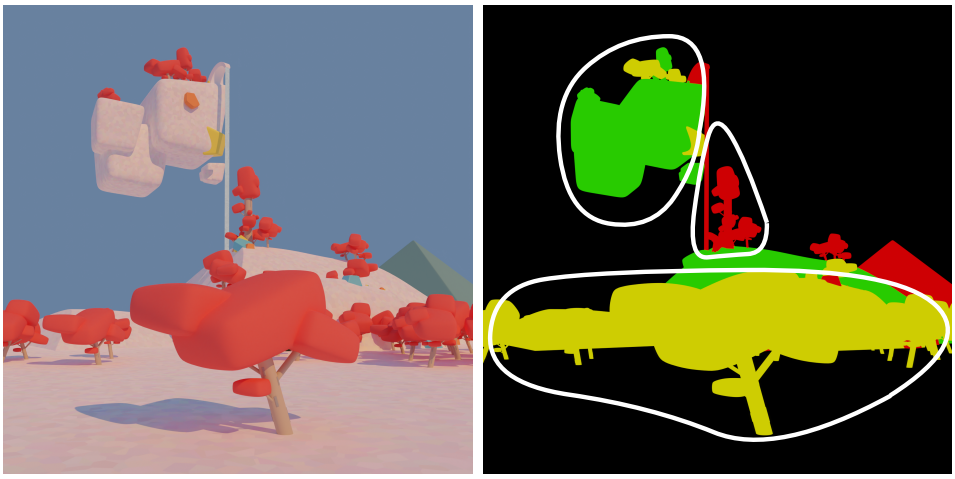
Si l'analyse des sessions de recréation a étudié l'efficacité de notre outil pour le travail collaboratif dans un contexte concret (guidé par des références), nous nous sommes également interrogés sur la capacité d'un tel *workflow* lors de la création de nouveaux designs sans références (*background concept*).

FIGURE II.2.12 – Tanguy Weyland (*CG Supervisor* sur *Tangranimo*) et Clémence Ottevaere (artiste *layout* sur *Tangranimo*) durant la seconde session de *background concept*.

Au cours des sessions de *background concept* collaboratif, les équipes étaient composées de deux à trois membres de niveau mid à senior. Ces expérimentations ont eu lieu dans les locaux de Cube Creative, les artistes étaient placés à côté (cf. Figure II.2.12) pour faciliter la communication orale.

Afin d'être aussi proche que possible des conditions réelles, nous avons utilisé un ensemble d'*assets* existants provenant de la série *Tangranimo* actuellement en production chez Cube Creative (en exploitant l'intégration de la *Resource Library* expliqué en section précédente). L'exercice se concentre sur le *layout* collaboratif pour la conception de nouveaux décors basés sur un style visuel existant. Au début, une caméra est placée par l'un des artistes pour verrouiller un point de vue. Puis chacun d'entre eux compose la scène parallèlement en ajoutant et en déplaçant des *assets*.

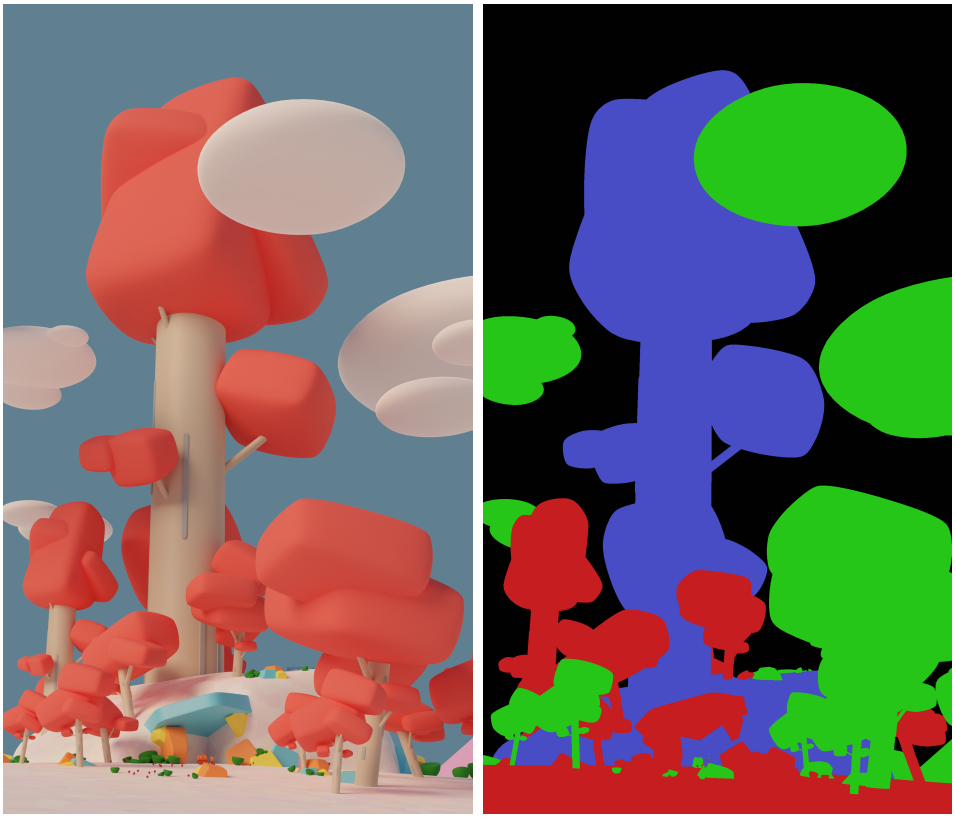
La répartition spatiale du travail est un critère particulièrement intéressant à étudier à la suite de ce type d'exercice d'improvisation lorsque les artistes travaillent sur la même étape de fabrication (ici le *layout*). Nous avons recréé une image en classant par couleurs les régions de l'espace occupées par les objets, en fonction de l'utilisateur qui y a travaillé (ce schéma ne prend en compte que les *assets* - pas les lumières). Cette image



(a) Scène résultante.

(b) Distribution du travail des utilisateurs par couleur.

FIGURE II.2.13 – Résultat de la première session de *background concept*.



(a) Scène résultante.

(b) Distribution du travail des utilisateurs par couleur.

FIGURE II.2.14 – Résultat de la deuxième session de *background concept*.

fournit une indication visuelle du degré de collaboration entre les utilisateurs. Lorsque la collaboration n'est pas fluide, on observe des îlots très distincts qui ne s'intersectent pas ainsi qu'une dominance du taux d'occupation de certaines couleurs.

C'est notamment le cas de la première session qui a été réalisée sans aucune phase préparatoire. Dans la distribution du travail par utilisateur (cf. Figure II.2.13b), on observe des groupes isolés (entourés en blanc sur l'image) qui ne se croisent pas, ainsi

Artiste	Taux d'occupation de l'image
Trisan Michel	15.3 %
Clémence Ottevaere	7.7 %
Tanguy Weyland	3.1 %
Assets non manipulés	73.9 %

(a) Première session

Artiste	Taux d'occupation de l'image
Clémence Ottevaere	26.0 %
Tanguy Weyland	13.7 %
Base commune	25.0 %
Assets non manipulés	35.3 %

(b) Seconde session

Table II.2.2 – Pourcentage d'occupation du travail des utilisateurs sur les deux sessions de *background concept*.

qu'une dominance du jaune sur le taux d'occupation spatiale (15.3 % pour l'utilisateur jaune contre 7.7 % pour le vert et 3.1 % pour le rouge, cf. Tableau II.2.2). Cela signifie que les artistes travaillaient de manière isolée dans l'espace, révélant une collaboration peu fluide, comme s'ils éprouvaient une certaine timidité à interagir les uns avec les autres. Cela se manifeste par un manque de cohérence dans la scène résultante illustrée sur la Figure II.2.13a.

Dans la deuxième session expérimentale, nous avons introduit une période de *brief* de cinq minutes au début de la session, consacrée à la mise en place d'une base commune. Durant cette phase, les artistes discutent et placent ensemble la caméra et les fondations géométriques de la scène (en bleu dans la Figure II.2.14b). Comme le montre la répartition du travail des utilisateurs dans la Figure II.2.14b, la conception commune de l'espace a entraîné une distribution beaucoup plus uniforme du travail. La scène qui en résulte (voir Figure II.2.14a) est cohérente.

Lorsque des maladresses se produisaient (par exemple, un objet projetant une ombre involontaire), nous avons observé que la nature temps réel de la collaboration permettait aux participants de les remarquer et de les corriger instantanément. Dans un pipeline de production traditionnel, ce processus aurait été beaucoup plus long et fastidieux (comme montré dans la Figure I.1.10 du Chapitre I.1). Ainsi, l'utilisation du *Multiuser* augmenta considérablement la communication entre les artistes, conduisant à une meilleure anticipation des erreurs. Enfin, les artistes ont commencé à parler pendant la création, ce qui n'était pas possible auparavant en raison de l'effet de silo (cf. Sous-section I.1.2.3).

II.2.3.5 Compatibilité avec l'animation *keyframe* ?

Les expérimentations parcourues dans les sections précédentes ont exploré l'application de la collaboration au sein de toutes les étapes de la phase de production à l'exception de l'animation. L'exploration de la collaboration en animation arriva plus

tardivement dans le cheminement de cette recherche, en avril 2021. Deux projets répartis sur huit sessions expérimentales à distance ont été menées pour étudier la viabilité de cet exercice. Comme le support des *rigs* était encore en développement, les sessions se sont focalisées sur la recréation de boucles de motion design sans *rig* complexe (seules des armatures simples furent utilisées).

(a) (b)

FIGURE II.2.15 – Rendus extraits des vidéos des projets *Mstrd II* (a) et *Elmo* (b) d'animation collaborative. Crédit en Annexe E.

Des images tirées de la vidéo des deux projets réalisés sont visibles sur la Figure II.2.15. Les vidéos résultantes sont archivées librement en ligne, un lien d'accès est fourni en Annexe E.

Avant le début de chaque projet, superviseurs et artistes se sont collégialement répartis les *assets* à construire afin que tous puissent travailler sur un *asset* qui était animé en parallèle. Il s'agissait ici d'animation *keyframe* manuelle.

Le premier projet (cf. Figure II.2.15a) a été réalisé par trois artistes pendant deux sessions. Le but était de reproduire une séquence animée. À l'exception de la balle entièrement bleue, tous les objets de la scène sont animés sur une boucle de 2,5 secondes. Le second projet (cf. Figure II.2.15b) qui dura six sessions n'est pas une reproduction complète. Les quatre artistes participants se sont principalement inspirés du mouvement de la balle de la référence inspirant l'exercice et ont modélisé un décor original. La scène comprend huit objets animés sur une boucle de huit secondes.

Pour se coordonner dans le temps, les artistes participants utilisaient l'outil de synchronisation temporelle (cf. Sous-section II.1.3.3) conçu pour l'occasion. Au cours de leur travail de création, les artistes n'ont eu aucune difficulté particulière à animer parallèlement les différents *assets* de la scène. Sur le second projet d'animation (cf. Figure II.2.15b), les superviseurs ont utilisé le même processus itératif d'animation que dans un pipeline traditionnel à savoir : *blocking* puis *refinment*[181]. Bloquer les poses

181. Le *refinment* est la seconde et dernière étape du processus d'animation qui précise les mouvements en ajoutant des interpolations et de nouvelles poses entre les positions clefs définies lors du

clefs des animations a permis aux artistes de synchroniser aisément entre eux les principaux mouvements de la balle par rapport aux décors. Le *blocking* représente donc une technique de coordination importante au sein de sessions de co-création d'animation.

Un artiste n'ayant pas de connaissances en animation se greffa au cours du second projet pour faire de la modélisation. Nous avons alors été témoins d'un exercice de transmission de savoir : le superviseur a apprit en direct le fonctionnement des armatures à l'artiste. En exploitant le retour temps réel des actions de l'initié dans la scène, le superviseur était en capacité de l'accompagner pour la création d'un squelette d'animation. Ces situations d'apprentissage spontanées ont été observées dans beaucoup des projets menés lors des sessions. Le fait que le *Multiuser* était conçu sur une version de *Blender* plus récente que celle utilisée en production était également en cause. Les superviseurs se sont retrouvés en situation de veille technologique, et communiquaient beaucoup entre eux sur les fonctionnalités qu'ils découvraient ensemble.

En menant la collaboration directement au sein d'un DCC utilisé en production, nous avons pu mener une exploration initiale de l'animation *keyframe* dans un contexte multi-utilisateur. Cette phase de production n'avait pas jusque-là été explorée dans des solutions de collaboration en temps réel à cause de sa complexité d'intégration dans les moteurs de jeu (cf. Sous-section I.2.2.2). Bien que nous ayons découvert que cette étape de fabrication était compatible avec la co-création, il reste beaucoup à explorer sur les interactions entre animateurs. Nous avons observé qu'il n'y avait aucun obstacle à l'animation simultanée d'objets simples, mais ce constat est-il applicable à des personnages avec des *rigs* complexes ?

Ces sessions ont permis l'exploration de différentes configurations d'application de la collaboration en temps réel dans un contexte industriel. Des équipes pluridisciplinaires ont questionné l'efficacité de notre méthode à l'échelle de la création d'une scène. Si l'on assimile chaque artiste de l'équipe à un département spécialisé, cela représente une collaboration inter-département. Bien qu'intéressants, l'analyse des sessions montra que les résultats pourraient être améliorés en optimisant la répartition des tâches pour maximiser la superposition du travail.

Le *Multiuser* a aussi été intégré avec succès dans un *micropipeline* d'asset pour les sessions de *background concept*. Ces dernières ont mis en valeur la nécessité d'un *brief* en début de session pour guider l'obtention d'un résultat cohérent.Enfin, nous avons brièvement évalué l'utilisation de cette méthode pour l'animation, maillon fondamental dans la création d'images animées.

blocking.

II.2.4 Expérimentations publiques

Dans cette section, nous décrirons l'impact de la pandémie du Covid-19 sur notre recherche. En provoquant la création d'une communauté internationale autour du *Multiuser*, elle nous guida vers des développements et questionnements sur la co-création à distance qui furent étudiés durant 27 projets créés au cours d'une centaine de sessions publiques qui prirent place entre juillet 2020 et août 2021.

II.2.4.1 Émergence d'une communauté

Au début de la crise sanitaire, nous avons reçu plusieurs demandes de personnes de la communauté de *Blender* pour créer un serveur *Discord*[182] dédié au *Multiuser* à travers le forum BlenderArtist[183]. Nous avons donc crée cet espace[184] initialement dédié aux utilisateurs du *Multiuser*. Progressivement, de plus en plus de personnes l'ont rejoint pour dépasser trois cents membres aujourd'hui.

Ce lieu de rencontre virtuel offre à tout un chacun un endroit pour échanger sur l'outil, les pratiques associées et trouver de l'aide. Au début, de nombreuses personnes venaient chercher conseils pour utiliser le *Multiuser* afin de télétravailler sur des scènes 3D sur internet. Mais en janvier 2020, l'outil était essentiellement prévu pour fonctionner sur un réseau Local Area Network (LAN) gigabit, par conséquent sa mise en place pour internet était fastidieuse et non optimisée. Parallèlement, l'idée d'organiser des sessions de création publiques émerge dès mars 2020 au cours de discussions ouvertes sur les espaces créatifs participatifs.

En réaction à tous ces évènements, entre février et avril 2020, nous avons effectué une passe d'optimisation et d'adaptation de l'architecture du *Multiuser* aux contraintes d'une utilisation sur internet.

II.2.4.2 Support de session sur internet

Pour optimiser les performances du pipeline de réplication sur internet, nous avons minimisé la taille des données transmises en utilisant des différentiels (cf. Sous-section II.1.2.5).

Côté architecture, le serveur a été isolé pour muer en serveur dédié. La séparation entre le *framework* et l'intégration simplifia énormément le développement et le déploiement de cet outil en le réduisant à un simple module Python à installer. Manipulable depuis le terminal, il permet de créer une session très rapidement depuis n'importe quelle machine sans interface graphique. Cela aida l'hébergement des sessions sur le *cloud*. Pour simplifier encore le processus, nous avons développé une image *Docker*[185] qui évite la configuration de l'environnement Python nécessaire au lancement du serveur dédié.

182. Un serveur *Discord* est une plateforme de communication sur internet dans laquelle les utilisateurs peuvent parler entre eux à travers des canaux textuels et vocaux. `https://discord.com/`

183. `https://blenderartists.org/`

184. Serveur discord du *Multiuser* : `https://discord.gg/mHtkYDW`

185. *Docker* est une plateforme open source permettant de lancer certaines applications dans des conteneurs logiciels. Ces conteneurs comprennent l'application et toutes ses dépendances. Ils peuvent ainsi être exécutés sur n'importe quel serveur. Par exemple, l'image *Docker* du *Multiuser* comprend le *framework* accompagné de la version appropriée de Python. Grâce à ce mécanisme, un utilisateur n'aura pas besoin de réinstaller le *framework* et Python sur chaque machine dans laquelle il voudra déployer le serveur dédiée du *Multiuser*. Plus d'informations sur *Docker* : `https://www.docker.com`

Cependant, l'ajout d'un serveur dédié posa une question majeure : comment démarrer la session sur une scène existant uniquement chez un des clients ?

Nous nous sommes inspirés du fonctionnement des systèmes de matchs utilisées dans les jeux vidéo en ligne [5, p. 533] pour apporter une réponse. Au démarrage du serveur, la session s'initialise en mode salle d'attente (ou *lobby*). Comme montré sur la Figure II.2.16, les utilisateurs peuvent se connecter, mais aucune modification n'est autorisée avant que l'un d'entre eux démarre la session en initialisant la scène. Pour empêcher n'importe qui d'initialiser accidentellement la session, nous avons ajouté un rôle d'administrateur. Pour être administrateur de la session, l'utilisateur doit fournir un mot de passe lors de sa connexion au serveur dédié.

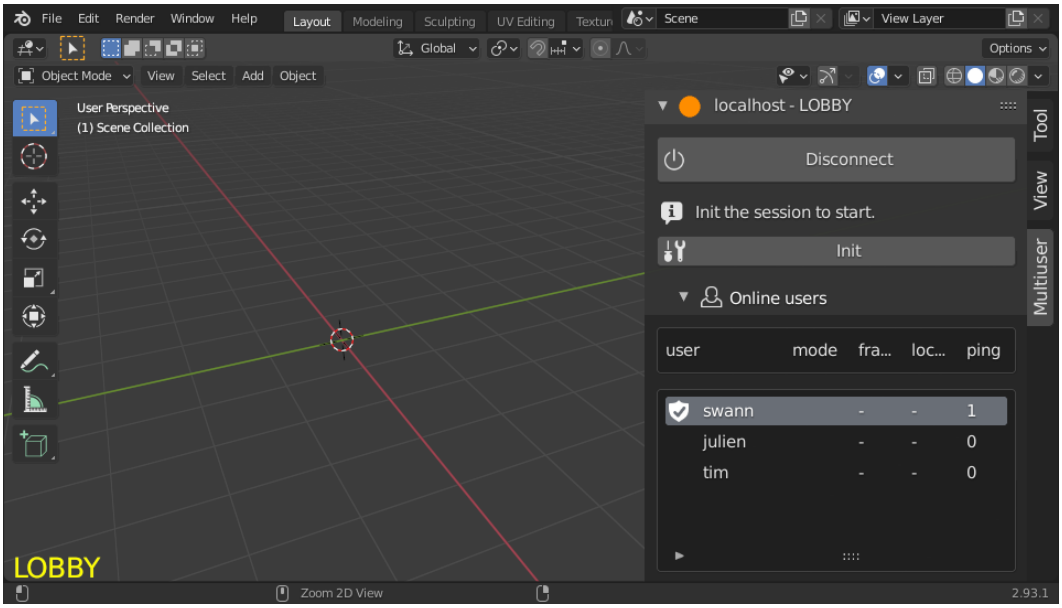


FIGURE II.2.16 – Session en mode *lobby* avant son initialisation.

Au terme de ces améliorations, la création d'une session dans le *cloud* pouvait se résumer à une commande à exécuter côté serveur :

```
replication.serve -p 5555 -pwd admin
```

Une documentation [28] complète co-construite par la communauté accompagna cette mise à jour majeure pour guider les artistes à travers l'utilisation de l'outil.

II.2.4.3 Un développement participatif

Nous avons rendu public le *Multiuser* au cours d'une intervention à la *Blender Conference* en 2019. Lors de cet évènement, le code source de l'outil fut publié sous licence open source et nous avons ouvert tout le processus de développement (outils de suivis, etc.) afin que toute personne ou société intéressée puise participer à cette aventure. Suscitant un vif intérêt auprès de la communauté de *Blender*, nous avons eu la chance de compter de nombreuses participations de particuliers et d'Ubisoft Animation Studio au processus de développement de l'outil.
Entre janvier et mars 2020, Ubisoft Animation Studio à mené une série de tests du *Multiuser* en interne. Cela a donné lieu à de nombreuses participations très constructives qui ont beaucoup apporté à la stabilité de l'*add-on*. Les artistes d'Ubisoft Animation Studio menaient leurs tests sur des scènes de production pour pousser l'outil dans

ses retranchements et n'hésitaient pas à rédiger des rapports d'erreurs dans le *bugtracker*[186] du *Multiuser* sur *GitLab*[187]. Voici quelques développements qui ont émergé des discussions avec Yannick Castaing (*CG Supervisor* chez Ubisoft Animation Studio) et Laurent Noël (*Lead R&D* chez Ubisoft Animation Studio) :

— le support de la collaboration à travers plusieurs scènes ;

— la division des paquets volumineux en petits morceaux (ou *chunks*) pour éviter d'atteindre la taille limite des paquets supportés par la bibliothèque *ZeroMQ* ;

— le *refactoring*[188] de la sérialisation des *meshes* pour améliorer les performances de leur réplication ;

— l'amélioration de la gestion des droits sur les *datablocks* d'arrière-plan.

Cependant, une autre équipe d'Ubisoft avait déjà développé en interne un protocole pour assurer la communication entre *VRtist* (une application RV interne basée sur Unity) et *Blender*. Finalement, ils ont opté pour la réexportation de ce protocole pour faire communiquer plusieurs instances de *Blender* à travers un nouvel *add-on* qu'ils ont nommé *Mixer* et qui a été déjà présenté en Sous-section I.3.2.2.

Après qu'Ubisoft ait publié *Mixer*, nous sommes restés en contact à travers des réunions où nous partagions nos avancées et difficultés communes vis-à-vis de l'implémentation d'une expérience collaborative dans *Blender*.

La communauté blender participa également dès 2019 au développement de l'*add-on Multiuser*, mais cela s'accentua avec la création du serveur *Discord* (cf. Sous-section II.2.4.1). Elle s'est impliquée dans tous les processus liés à l'émergence de l'outil tel qu'il se présente dans cette thèse, à savoir :

— la documentation : rédaction d'une grande partie de la documentation de l'outil ;

— le support : certains utilisateurs se chargent d'aider ceux en difficulté grâce aux canaux de discussions dédiés sur *Discord* ;

— la stabilisation : certains membres publient régulièrement des rapports d'anomalies ;

— l'orientation de développement : beaucoup de fonctionnalités sont nées d'idées formulées par la communauté ;

— l'infrastructure : certains membres ont participé à la création de scripts simplifiant le déploiement de serveurs dédiés sur le *cloud* ;

— la recherche : les membres qui prennent part aux sessions publiques participent directement à l'étude de nos problématiques de recherches.

186. Un *bugtracker* est un outil de suivi des dysfonctionnements (ou erreurs) associés au fonctionnement d'un logiciel. Les problèmes y sont généralement représentés sous forme de tickets qui canalisent le suivi de leur résolution dans un fil de commentaires.

187. Quelques exemples de rapport de bug d'Ubisoft Animation Studio : bug de déconnexion sur les scènes importantes : `https://gitlab.com/slumber/multi-user/issues/59`, *crash* avec des scènes utilisant des images comme texture : `https://gitlab.com/slumber/multi-user/issues/58`, l'arrière-plan non synchronisé : `https://gitlab.com/slumber/multi-user/issues/57`, *crash* lors de l'utilisation de *speakers* : `https://gitlab.com/slumber/multi-user/issues/56`

188. En développement logiciel, le *refactoring* désigne le processus de restructuration du code d'un logiciel sans changer son comportement externe. Cette opération vise généralement à améliorer le design, la structure interne du logiciel tout en conservant ses fonctionnalités.

II.2.4.4 Un *workflow* collaboratif à distance

Dès que ce fut techniquement envisageable, nous avons commencé à organiser les sessions publiques. Il fut compliqué de trouver une heure commune pour laquelle les personnes de différents pays puissent participer à cause du décalage horaire. C'est en juillet 2020 que nous sommes parvenu à trouver les créneaux de 17-18h (Paris) le mardi et vendredi chaque semaine. À partir de là, nous avons organisé ces sessions durant plus d'une année, jusqu'à la rédaction de cette thèse. Cela permit d'aborder différents aspects du *workflow* de co-création en temps réel avec la communauté tel que :

— la segmentation du travail ;
— le suivi de la fabrication de la scène ;
— le consensus artistique dans l'exercice de la création libre collaborative.

Ces sessions internet se coordonnaient sur un canal de discussion écrit sur *Discord*, un second canal vocal était mis à contribution pour diffuser en direct une capture du client témoin de la scène en construction. Régulièrement des membres se connectaient au canal vocal pour suivre la scène en cours de création. Paradoxalement, la communication orale fut très peu utilisée durant ces sessions (peut être dû à la barrière de la langue et du fait que les gens ne se connaissaient pas). Pour chaque projet mené, le thème (création libre ou orientée avec une référence) était choisi collégialement par vote. Au début de chaque projet, une période de préparation de 5 minutes était observée similairement aux sessions de *background concept* (cf. Section II.2.3) pour mettre en place la structure de la scène (point de vue, etc.) et répartir les tâches initiales.

Segmentation du travail

À travers les sessions de recréation, nous avons exploré les segmentations suivantes :

— libre : laisser chacun choisir son travail sans contrainte ;
— stricte : imposer une division en se basant sur un ou plusieurs critères :
  par zone,
  par aspect (*shading*, *lighting*, etc.),
  par *asset*.

La segmentation libre consiste à laisser les artistes se répartir le travail à mesure de l'avancement du projet. Elle a été utilisée au cours de nombreuses sessions. Cependant, nous avons constaté qu'elle ne pouvait fonctionner que lorsque les participants avaient déjà appris à travailler ensemble. En connaissant les forces et faiblesses de chacun sur les différents aspects de la création, les artistes sont à même de se diviser les tâches au fur et à mesure de la création. L'efficacité de cette méthode dépendra donc directement de la communication entre artistes. Elle peut éventuellement nécessiter un coordinateur qui arrête les décisions artistiques importantes et garde toujours un œil sur la progression de la scène pour anticiper les problèmes (par exemple, un *asset* qui dévie trop du style de la référence). Nous reviendrons sur ce rôle dans la Figure II.2.4.4.

La segmentation par zone fut mise en application dans un projet de modélisation. L'objectif consistait à reproduire une image de référence en 3D (représenté sur la Figure II.2.17). Le découpage des zones fut proposé par un membre de la communauté (softyoda). Quatre personnes participèrent au projet.

La modélisation de la scène fut partiellement terminée en cinq sessions (le rendu est visible en Figure II.2.18a). Comme nous n'étions que quatre au lieu des six initialement

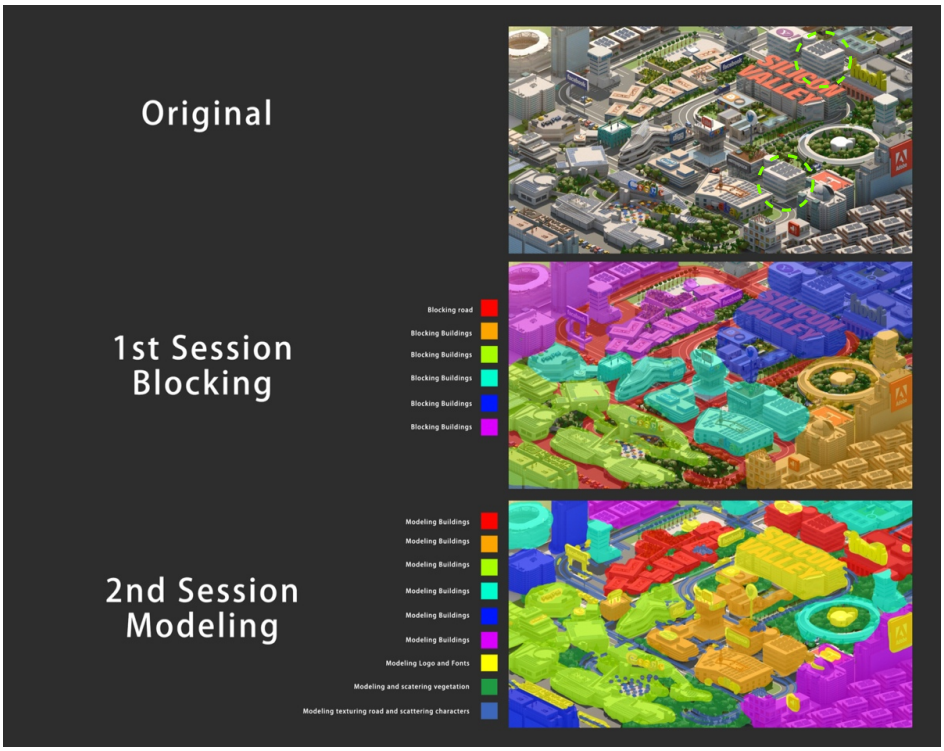
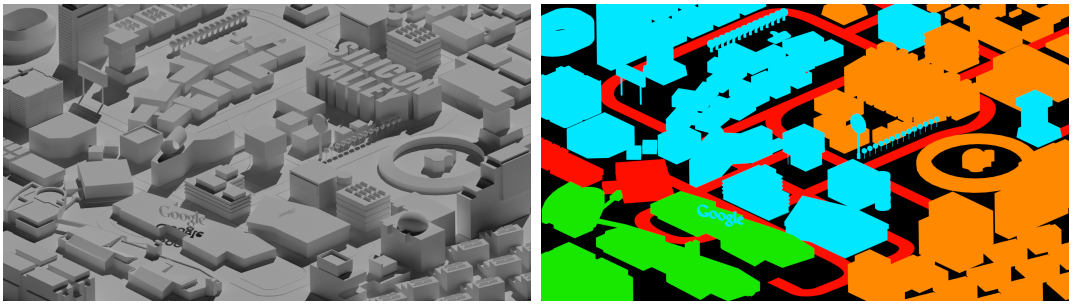


FIGURE II.2.17 – Schéma d'organisation préalable à la session (créée par softyoda). Une couleur représente un utilisateur.



(a) (b)

FIGURE II.2.18 – Résultat de la segmentation par zone sur le projet *Silicon Valley* avec le rendu (a) et la distribution effective (b). Crédits en Annexe E.

prévus dans la répartition (cf. Figure II.2.17), les deux zones vacantes (bleue et violette sur la Figure II.2.17) ont été respectivement données aux utilisateurs orange et bleu clair.

Au cours des sessions, nous avons noté qu'il n'y avait que très peu de conflits : en répartissant les utilisateurs par zone, ils avaient très peu d'*asset* interagissant entre eux. En comparant les distributions du travail théorique (cf. Figure II.2.17) et effectif (cf. Figure II.2.18b), on note que les utilisateurs bleu et rouge ont créé des *assets* débordants de leur zone de travail. Cela s'explique par la différence de niveaux des artistes composant l'équipe. Ces deux utilisateurs, plus expérimentés, ont fini leur zone avant les autres. Par conséquent ils ont entamé les *assets* des zones des utilisateurs moins expérimentés. Cette intersection du travail a induit des temps morts, car les artistes ont dû se coordonner à nouveau pour définir les bâtiments sur lesquels ils allaient commencer à travailler. Les tâches définies avec cette méthode n'étaient pas suffisamment simples pour que les artistes transitent entre elles avec fluidité. On note

également que cette segmentation n'est pas adaptée aux équipes contenant des artistes de différents niveaux. Cependant, cette limitation pourrait être levée en déterminant des zones plus petites. Les artistes pourraient ainsi évoluer entre les zones à leur rythme sans nouvel effort de coordination. On peut également imaginer des zones de taille variable adaptées au niveau d'expériences des artistes.

Par ailleurs, les deux *assets* entourés de pointillés verts sur la Figure II.2.17 sont exactement les mêmes, mais ils n'appartiennent pas à une même zone. Par conséquent, ils ont été recréés une fois par chaque utilisateur comme en témoigne la répartition effective (cf. Figure II.2.17). Pour sauvegarder des efforts (et pour une meilleure optimisation), cet *asset* aurait du n'être modélisé qu'une fois, le second aurait alors été une simple instance [189] du premier. La technique de répartition par zone ne prend pas en compte la redondance des *assets* ce qui induit donc une mauvaise superposition du travail.

La segmentation par étapes de fabrication implique des artistes spécialisés. Au cours des sessions, nous n'avons pas eu l'occasion d'avoir une équipe composée entièrement de membres spécialisés dans chacun des domaines requis à la création d'une scène complète. En revanche, il est souvent arrivé qu'une personne de l'équipe ne fasse qu'un seul aspect de la scène. Sur les quatre sessions de reproduction de l'œuvre *Wind mill in Němčice* de Febin Raj par exemple, j'ai fait la majorité du *shading / lighting* de la scène. Pendant ce temps, les trois autres artistes modélisaient et disposaient les *assets*. Le résultat de la session est visible sur la Figure II.2.19. L'utilisation de la segmentation par aspect nous permit de créer aisément un effet de grain homogène sur l'ensemble des *assets*.

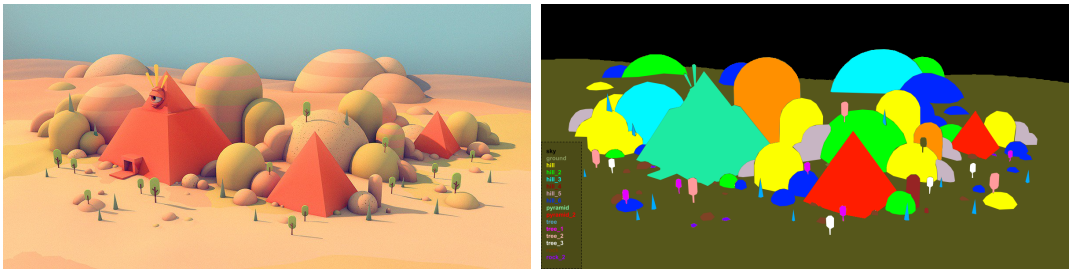
La segmentation par *asset* fut initialement testée sur le projet de recréation de *All Seing Monolith* (cf. Figure II.2.20a). La Figure II.2.20b montre le document utilisé pour guider les artistes à travers un découpage des différents *assets* de l'image de référence. Contrairement à la segmentation par zone, on constate que cette technique met en valeur les *assets* identiques en les colorant de la même couleur. Grâce à ces informations, nous avons été en capacité de paralléliser la fabrication des *assets* et leur *layout* dans la scène en créant des instances. En guidant les artistes dans l'utilisation d'instances d'*assets*, cette méthode optimise la superposition du travail dans la création de scène guidée par une référence. Durant ce projet, les six artistes participants choisissaient au fur et à mesure les *assets* qu'ils souhaitaient faire parmi ceux listés dans le document. La session dura 90 minutes. En séparant les tâches au niveau des *assets*, les artistes expérimentés pouvaient enchaîner le travail sans perturber les novices.

Consensus artistique dans la co-création

Parmi les 27 projets menés, seul deux d'entre eux furent des exercices de création libre. Cela témoigne des difficultés que nous avons rencontrées avec ce type de projet. Pour les sessions de recréation, le consensus artistique était naturellement conduit par la référence 2D. Mais sans ces références, il était compliqué de ne pas s'éparpiller dans toutes les directions. C'est pourquoi un thème était collégialement fixé au début de ces deux projets : ce fut respectivement *zen garden in a snowball* pour la première et

189. Une instance est un pointeur vers un *asset* préexistant. Ce mécanisme permet de placer plusieurs fois cet asset dans la scène pour le coût d'un seul, car il est dupliqué au moment du rendu. Ainsi, une modification visuelle effectuée sur l'*asset* (par exemple, une amélioration de son *mesh*) sera automatiquement répercutée sur ses instances.

FIGURE II.2.19 – Résultat de la recréation de *Wind mill in Němčice*. Crédits en Annexe E.



(a) (b)

FIGURE II.2.20 – Image de référence (a) du projet de recréation « *All Seing Monolith* » exploitant la segmentation par *asset* (b). Crédits en Annexe E.

outrun pour la seconde.

La Figure II.2.21 montre le résultat de la première session. Parmi les quatre artistes participants, deux sont partis dans des directions différentes : un travaillant sur les *assets* liées au jardin zen et l'autre travaillant sur la maison type chalet lié à l'univers de la montagne. Parmi eux, le bonzaï et le chalet ressortent beaucoup, car ils ne forment pas un ensemble visuel homogène et ne sont pas à une échelle cohérente (le bonzaï en pot est trop grand, et la maison n'a pas le même niveau de détail que l'arbre). Ce problème découle de la difficulté qu'ont eue les participants à faire des choix artistiques cohérents sur l'échelle de l'ensemble de la scène. Pour homogénéiser un peu l'image en fin de projet, un artiste a eu la bonne idée de créer une couche de neige sur l'ensemble de les objets. Mais finalement, personne n'a pris de décisions assez fortes pour guider la création globale de la saynète.

Au cours du second projet, nous avons constaté l'importance du rôle de coordinateur. L'artiste qui proposa le thème d'*outrun* coordonna les décisions artistiques en validant les choix importants. Par ailleurs, il n'hésitait pas à faire des remarques lorsque

FIGURE II.2.21 – Résultat de la première session de création libre sur le thème : *zen garden in a snowball*. Crédits en Annexe E.

les *assets* dérivaient du style voulu. On constate sur la Figure II.2.22 qu'au terme du projet, nous avions une scène bien plus homogène en terme de style graphique que dans la première session. Régulièrement, certains artistes demandaient ouvertement ce qu'ils pouvaient créer sur la scène en début de session. Ici, le coordinateur, qui avait une vision d'ensemble put leur indiquer très rapidement sur quel *asset* se lancer.

FIGURE II.2.22 – Image tirée de l'animation produite durant la seconde session de création libre sur le thème : *outrun*. Crédits en Annexe E.

Le rôle de coordinateur est donc à mi-chemin entre un chargé de production et un

directeur artistique. C'est une personne qui, en plus de son travail de création, se rend garant de décisions artistiques et organisationnelles. Il s'est avéré vital dans le cadre des projets de création libre pour obtenir un consensus artistique sur le style visuel. Il réorientait les artistes lorsqu'ils s'éloignaient trop de l'identité globale du projet.

II.2.4.5 Un espace de co-création persistant ?

Au fur et à mesure des sessions publiques, nous constations que les créneaux horaires initialement choisis posaient problème pour les participants des pays d'Asie qui devaient veiller tard pour être présents. L'idée de session persistante émergea en écho à ce problème. En laissant la session publique en ligne de façon permanente, il devenait alors possible pour chacun de venir créer à n'importe quel moment.

Dès novembre 2020, les projets furent menés à travers des sessions persistantes. En laissant le serveur en ligne jour et nuit, nous avons pu le stabiliser en décelant certains bugs liés à l'accumulation de données mal libérées. Cependant, la persistance de l'espace de co-création entre les sessions posa très rapidement un problème important : comment un artiste rejoignant la session à un moment où personne n'est connecté peut-il savoir qui a travaillé sur quoi et ce qu'il reste à faire ? Cela revient à dire, comment assurer la continuité du travail collaboratif sur une scène ?

Pour trouver une solution, nous nous sommes inspirés d'un outil alors utilisé pour le développement du *Multiuser* : le tableau kanban ou *kanboard*[190]. Sur le *Multiuser*, cet outil permet un suivi temporel des différentes tâches menées en parallèle (cf. Figure II.2.23). Un *kanboard* contient un certain nombre de cartes ; représentant les tâches à réaliser. Ces cartes sont réparties entre plusieurs colonnes représentant des états d'avancement bien définis. Au fur et à mesure de l'avancement, les développeurs actualisent le statut des tâches qui leur sont assignées en les changeant de colonne. Pour le développement open source, cette organisation du travail à l'avantage de visualiser le travail fait et à faire, permettant aux développeurs bénévoles de participer facilement. Nous l'utilisons aussi comme une *roadmap*[191] pour communiquer à la communauté l'avancement des différentes fonctionnalités en cours de développement. Mais comment l'appliquer dans la création collaborative ?

Pour utiliser un *kanboard*, il faut avant tout avoir un ensemble de tâches à réaliser. Pour ce faire, nous avons segmenté la référence par *assets* (étudiée en Sous-section II.2.4.4). Une fois les *assets* définis, nous les avons disposés dans des cartes individuelles sur un *kanboard* hébergé par *Cryptpad*[192], un outil gratuit et open source. Trois colonnes ont étés utilisées pour organiser les cartes :

— To Do : *assets* ou tâche à faire ;

— In Progress : *assets* ou tâche en cours de création ;

190. Un tableau Kanban ou *kanboard* est un outil inventé par Toyota dans les années 1960 pour mettre en œuvre la méthode agile Kanban afin de gérer le travail. Il décrit visuellement le travail à différentes étapes d'un processus en utilisant des cartes pour représenter les éléments du travail et des colonnes pour représenter chaque étape du processus. Les cartes sont déplacées de gauche à droite pour montrer la progression et aider à coordonner les équipes qui effectuent le travail. `https://leanmanufacturingtools.org/kanban/`

191. La *roadmap* d'un logiciel est une représentation graphique communiquant les objectifs de livraison de ses différentes fonctionnalités à travers le temps.

192. Adresse du *kanboard* utilisé sur le projet *Random House #5* : `https://cryptpad.fr/kanban/#/2/kanban/edit/Zj2dLA-SdJ37UpgSRJsTMQTl/`

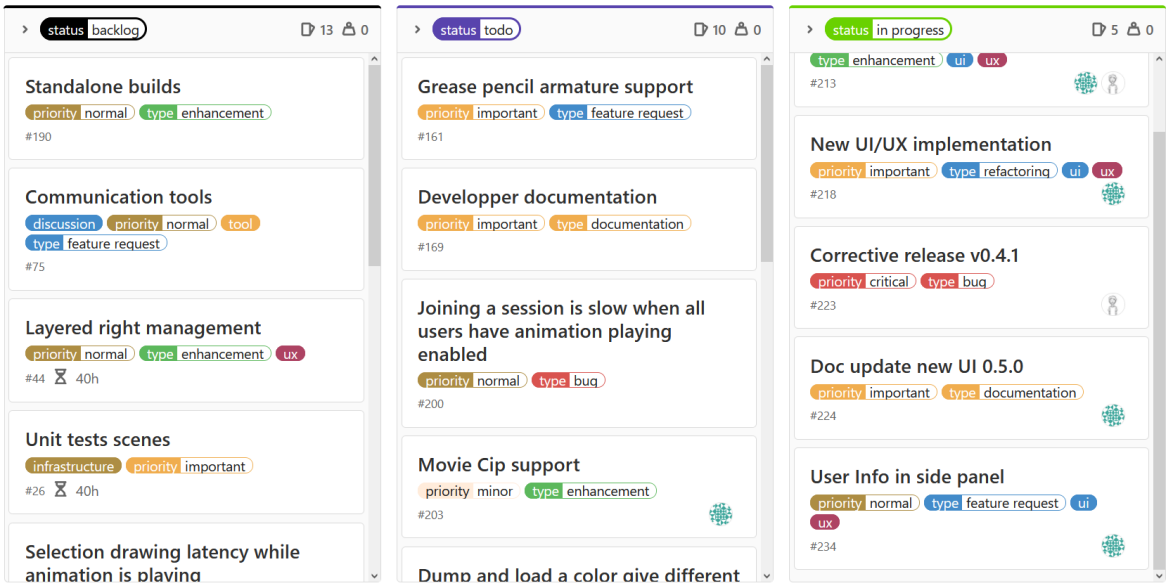
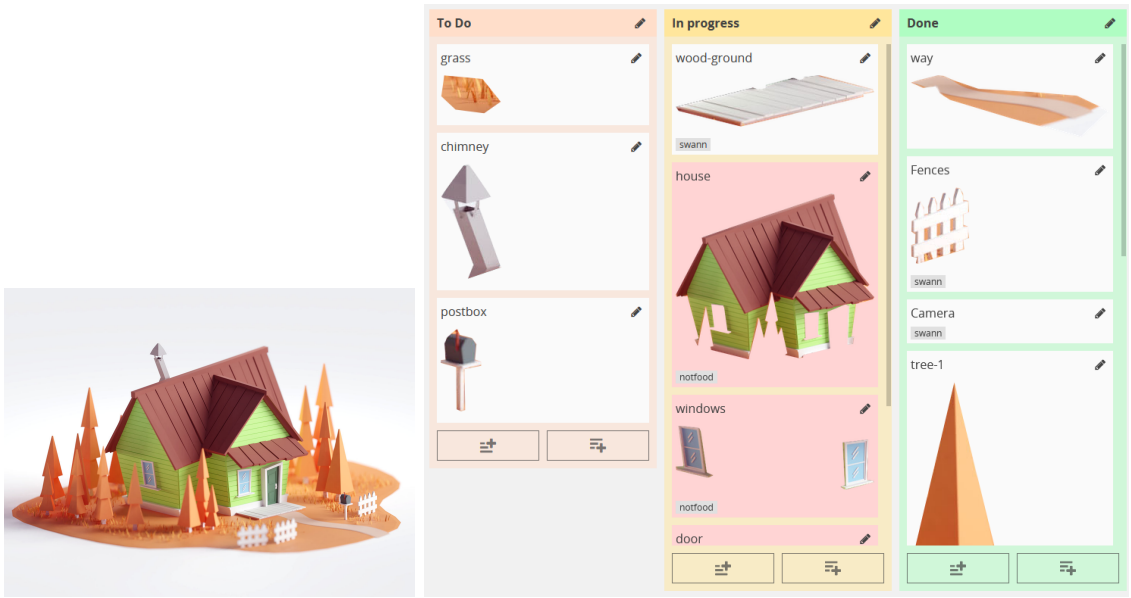


FIGURE II.2.23 – *kanboard* utilisé sur la conception du *Multiuser*.

— Done : *assets* ou tâche finie.

Au départ, toutes les cartes figuraient dans la colonne To Do. Lors du lancement du projet, la consigne a été donnée aux artistes de choisir ce qu'ils voulaient faire sur le tableau en s'assignant sur les cartes et de les mettre à jour dans les bonnes colonnes au fur et à mesure de la création. La référence et le *kanboard* sont exposés sur la Figure II.2.24.



(a) Référence initiale. (b) *kanboard* du projet.

FIGURE II.2.24 – Utilisation du *kanboard* pour organiser le travail durant le projet *Random House #5*.

En montrant aux artistes participants à la création le travail que faisait actuellement chacun, où il en était et ce qu'il restait à faire, le *kanboard* permit d'organiser les tâches à travers la session persistante. Cet outil accéléra grandement leur répartition en exposant clairement le travail qu'il restait à faire. Par la suite, nous l'avons

régulièrement utilisé pour coordonner les sessions publiques et certaines des sessions industrielles.

FIGURE II.2.25 – Rendu de la première session exploitant le *kanboard*. Crédits en Annexe E.

L'ouverture du *Multiuser* à la communauté apporta énormément à nos travaux en matière de développement et de recherche. Le support des sessions en ligne (sur internet) a servi de socle à de nombreuses expérimentations publiques. Au cours de ces sessions, la communauté s'est impliquée dans l'exploration de nos problématiques de recherche sur l'organisation du travail pour la co-création en temps réel. Ces sessions ont également étudié le suivi du travail dans le contexte de la collaboration en temps réel. Pour centraliser le statut du travail, nous avons adopté un outil des méthodes agiles : le *kanboard*.

II.2.4.6 Organisation de la scène

Dans un pipeline traditionnel (cf. Section I.1.3), le *dataflow* implique presque toujours un fichier de travail différent par artiste. Dans une session collaborative temps réel, tous les artistes travaillent dans le même fichier. La problématique classique de rangement des fichiers s'applique donc aussi à l'échelle de la scène. Comment structurer la hiérarchie de la scène pour que plusieurs personnes y naviguent facilement ?

Au cours des sessions, nous avons exploré deux structures hiérarchiques de scène différentes :

— orientée artiste ;

— orientée *asset*.

La structure orientée artiste décrite dans la Figure II.2.26a consiste à ranger les *assets* dans des collections [193] portant le nom des artistes participants à la racine. Elle permet de savoir sur quels *assets* les artistes ont travaillé. Pas contraste, la structure

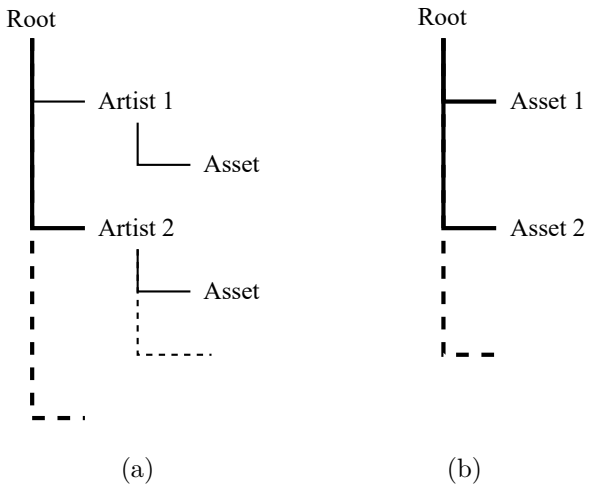


FIGURE II.2.26 – Hiérarchies de scène orientée artiste (a) et centré *asset* (b).

orientée *asset* consiste à ranger les *assets* directement à la racine de la scène.

La structure orientée artiste s'est révélée particulièrement utile dans le cadre des sessions académiques. En isolant le travail de chaque apprenant dans un dossier à son nom, il nous était facile de voir à qui apporter de l'aide. Nous avons également exploité cette structure pour éviter tout conflit de modification accidentelle entre étudiants en verrouillant les droits de modification de chacun de ces dossiers à l'étudiant correspondant.

Sur les sessions publiques et industrielles, nous avons testé les deux configurations. L'organisation orientée artiste fut utile au début pour éviter le chaos lorsque les graphistes n'avaient pas l'habitude de ranger leurs scènes. Mais avec des équipes rigoureuses, l'approche orientée *asset* fut plus intéressante pour deux raisons. Tout d'abord, la hiérarchie est plus lisible, car elle permet de faire des rangements plus logiques (par exemple, une collection végétation, une collection bâtiment, etc.). Ensuite, cet arrangement favorise le travail de groupe : en retirant toute notion de propriété sur les *assets*, les artistes ont plus de facilité à aller travailler sur un *asset* qu'ils n'ont pas créé.

II.2.4.7 Superposition du travail

Lorsque plusieurs artistes travaillent ensemble dans le même environnement virtuel en temps réel, la superposition du travail (c'est-à-dire la capacité à paralléliser la création des différents aspects), devient une variable importante pour éviter le gaspillage d'efforts (les *assets* recrées deux fois par exemple, cf. Sous-section II.2.4.4). Pour aborder cet aspect important, nous avons étudié le processus de fabrication de jeu vidéo qui se heurte à une problématique similaire.

La phase de préproduction d'un jeu vidéo consiste à définir un prototype jouable pour le vendre à l'éditeur. On parle de développement en tranche (ou *slice*) [1, p. 16] lorsque chaque aspect est développé en parallèle (cf. Figure II.2.27).

Une tranche horizontale représente l'ensemble de l'expérience dans un environnement de prototypage surnommé *whitebox level*, c'est-à-dire avec des ressources artistiques temporaires et très probablement du code temporaire. Ils ont pour objectifs

193. Dans *Blender*, les collections sont des éléments qui permettent de structurer la hiérarchie de la scène. Elles peuvent s'apparenter à des dossiers de rangement.

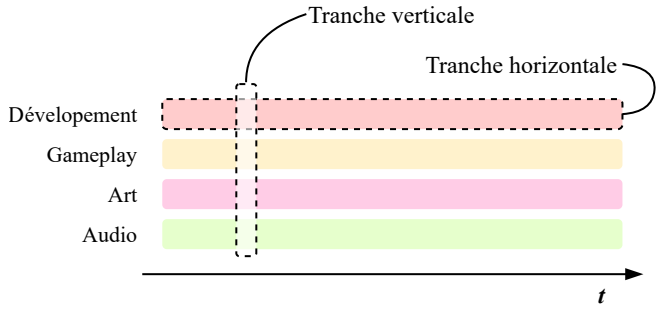


FIGURE II.2.27 – *Workflow* de la préproduction d'un jeu vidéo.

d'aider à voir comment le *gameplay*[194] se déroulera, combien de temps le jeu durera. À l'inverse, les tranches verticales sont de petites sections du jeu aussi proches que possible de la qualité du livrable.

FIGURE II.2.28 – Exemple de *whitebox level* de design de niveau. Source : David Shaver - *Game Developers Conference* 2018

Pour permettre un tel niveau de parallélisation du développement de la logique du jeu et de la création des *assets* artistiques, les développeurs s'appuient sur des *placeholders*. Les *placeholders* sont des versions extrêmement simplifiées des *assets* créées par les artistes. Par exemple, le *whitebox level* de la Figure II.2.28 contient de nombreux *placeholders* d'éléments graphiques du niveau. Ces derniers permettent au *gameplay designer* de développer les mécaniques du jeu pour que les artistes complètent les *assets* graphiques. Tout au long de l'avancement du projet, les *placeholders* sont progressivement remplacés par des versions plus avancées des *assets*.

Nous avons essayé d'appliquer ce concept à la co-création en temps réel pour paralléliser différents aspects de la création (*layout*, modélisation, etc.). Dans la session *All Seing Monolith*, nous avons utilisé cette technique pour paralléliser le *layout* et la construction des *assets*. Les *placeholders* furent créés puis placés grâce au système d'instance d'*asset* de *Blender*. Sur la Figure II.2.29, on observe que l'arbre encadré en jaune est en cours de construction par Staz tandis que Wuaieyo positionne l'une des instances de l'arbre (encadrée en cyan). Grâce à cette configuration de travail, lorsque Staz fait des changements sur l'arbre, Wuaieyo les constatera en temps réel sur toutes les instances qu'il a positionnées. Pendant ce temps-là, Softyoda était en train de créer

194. Le *gameplay* ou « jouabilité » désigne l'ensemble des possibilités d'action offertes au joueur par un jeu vidéo. Définition par la Commission d'enrichissement de la langue française.

l'ambiance lumineuse. Notons que cette organisation a été rendue possible grâce à la segmentation des tâches par *assets* (cf. Sous-section II.2.4.4).

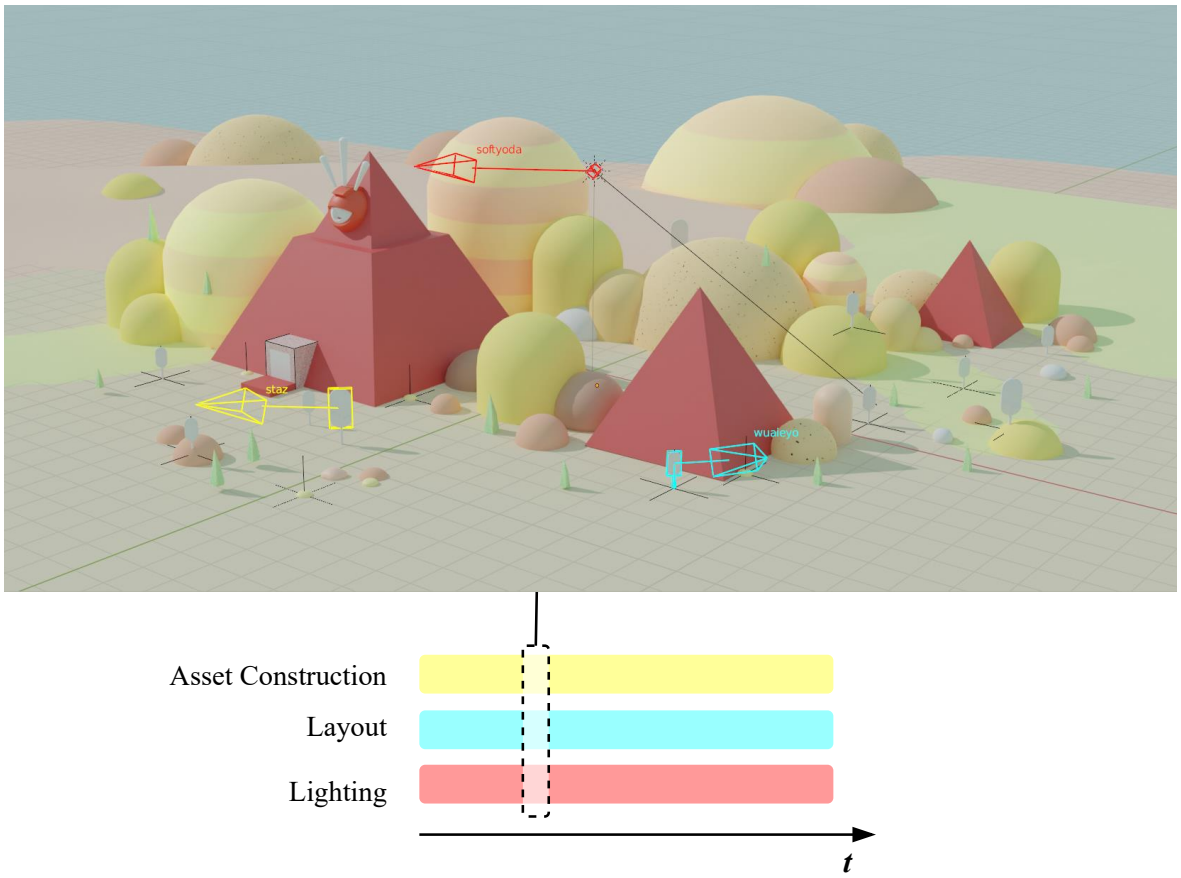


FIGURE II.2.29 – Parallélisation du travail dans la session *All Seing Monolith*.

Dans *All Seing Monolith*, la superposition du travail se produit au sein de la même scène, du même espace virtuel. Lorsque les artistes ont besoin d'être isolés dans un espace de travail séparé, cette logique de *placeholder* peut s'étendre à l'échelle des scènes : les *assets* peuvent être conçus dans une scène et disposés dans une autre. Pendant la session *Outrun*, cette logique fut utilisée pour permettre aux artistes de travailler sur le *layout* sans gêner la création des *assets* qui le composent (cf. Figure II.2.30).

Néanmoins, il fut constaté que le système de droit actuel limite la parallélisation des tâches sur la construction d'*asset* (modélisation, *shading* et *rig*). Au moment de la sélection d'un objet, l'opération de synchronisation des droits avec la sélection active (effectuée au niveau de l'*add-on Multiuser*, cf. Sous-section II.1.3.4) verrouille les droits sur l'objet et ses dépendances (*datablocks* de matériaux, textures, *mesh*, etc.), empêchant deux artistes de construire le même objet ensemble. Nous proposons une piste de solution dans la suite de cette thèse (cf. infra Sous-section II.3.2.4).

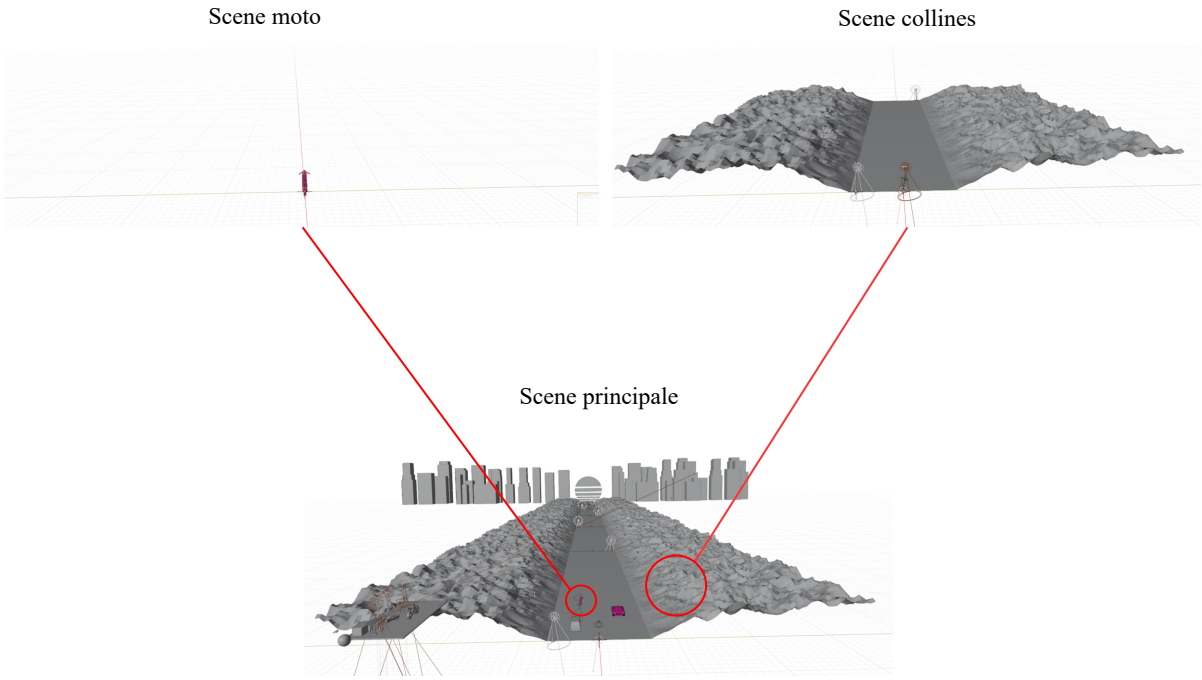


FIGURE II.2.30 – Parallélisation du travail dans la session *Outrun*.

Conclusion

Ce chapitre a fait état du travail d'exploration mené à partir du *Multiuser* (cf. Chapitre II.1). La diversité des cadres d'application de cette CIFRE a permis l'exploration de notre méthode dans le milieu académique pour l'enseignement de l'image de synthèse et dans le milieu industriel pour le processus de fabrication de films d'animation. Nous avons ensuite constaté le tournant inattendu qu'a provoqué la crise de la Covid-19. À travers ce contexte économique, une communauté internationale s'est formée autour de l'outil et s'est activement impliquée dans l'exploration de nouveaux horizons collaboratifs.

Chapitre II.3

Collaboration en temps réel au cœur d'une production traditionnelle

Introduction

Dans l'optique d'une utilisation progressive du *Multiuser* au sein d'un studio, une première étape serait de l'exploiter dans les processus déjà en place. Nous nous sommes interrogés sur la manière d'insérer la collaboration en temps réel au sein d'une production traditionnelle d'un studio de création de films d'animation.

Au terme des sessions expérimentales, nous avons réfléchi à cette question avec l'ensemble des acteurs décideurs de Cube Creative. Les applications concrètes envisagées suite à cette thèse sont étudiées en première section.

Bien qu'il permette de collaborer sur de nombreux aspects de la création, le *Multiuser* est encore en développement. Avant d'être exploités dans certaines étapes de fabrication, des modules de l'outil méritent d'être étendus. Nous ferons le point sur ces pistes de développement en seconde section.

II.3.1 Applications du *Multiuser* dans un pipeline de série animées

En juin 2021, nous avons organisé une réunion avec l'ensemble des superviseurs participant aux sessions industrielles et les dirigeants de Cube Creative dans l'objectif de situer de potentiels cas d'application du *Multiuser* dans le pipeline de fabrication déjà en place. Au terme de cet évènement, les cas suivants ont été envisagés :

— le *look development*[195] ;

— les *briefs layout* et *storyboard* ;

— former les nouveaux graphistes à *Blender* ;

— faire du découpage de séquence.

Pour chacune de ces applications, cette section fait état du processus existant et de ce que lui apporte le *Multiuser*.

Actuellement, nous avons débuté les tests d'utilisation du *Multiuser* pour les *briefs* et la formation, les autres cas sont encore prospectifs.

II.3.1.1 *Briefs layout* et *storyboard*

Actuellement, pour faire le *brief storyboard* d'un épisode, le directeur artistique Tristan Michel exporte le modèle 3D du décor et l'importe dans SketchFab [196]. Il l'utilise ensuite pour naviguer dans le décor afin de montrer en partage d'écran les différents points de vue aux équipes de *storyboard*. Durant le *brief*, il prend des captures d'écran des différents points de vue intéressants.

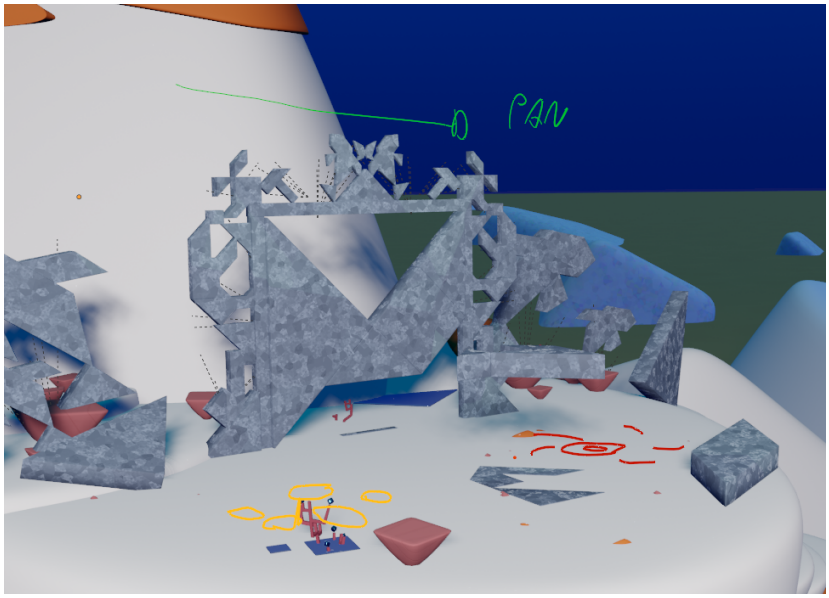


FIGURE II.3.1 – Annotation faite par Tristan lors d'une session de test de *brief* avec le *Multiuser*.

195. Le *look development*, ou *lookdev*, est la tâche de chercher à quoi ressemble le rendu final d'un asset. C'est donc l'ensemble des aspects techniques et artistiques nécessaire à la définition de l'aspect et du ressenti visuel qu'offre un projet. Le lookdev s'intéresse donc particulièrement aux deux aspects fondamentaux du rendu : la matière et la lumière. Définition par Lepipeline.Org.

196. `https://sketchfab.com/`

Tristan relève plusieurs problèmes avec cette méthodologie. Tout d'abord, exporter les décors pour passer d'une solution à une autre est un processus fastidieux. Il faut s'assurer d'avoir la dernière version et préparer la scène en ajoutant des références des personnages un peu partout pour donner une idée de l'échelle du décor aux storyboarders. Dans un second temps, les images sauvegardées par Tristan ne comprennent pas les informations techniques de la caméra (focale, position dans la scène, etc.). On perd donc beaucoup d'informations qui pourraient être extrêmement utiles aux départements du *layout* pour guider la création des caméras.

Porter les *briefs storyboard* et *layout* sur *Blender* avec le *Multiuser* permettrait à Tristan et aux artistes storyboard/*layout* de partager leurs observations dans le même espace virtuel. Les storyboarders et artistes *layout* pourraient poser des caméras ensemble et avoir de meilleurs timings ainsi qu'une meilleure visualisation de l'espace. Les annotations pourraient être utilisées pour écrire des indications spatiales et temporelles (par exemple, un mouvement de pan. cf. Figure II.3.1). Ces caméras pourraient ensuite servir comme base au *layout* et ainsi l'accélérer.

Durant les sessions de test, Tristan et Jérôme Auffret (lead *layout*) ont déterminé que les *briefs* nécessiteraient un peu plus de temps de préparation mais que ce temps serait amorti par la suite grâce à la réutilisation des informations qu'ils généreront.

II.3.1.2 Formation des nouveaux graphistes

Comme nous le relevions dans le Chapitre I.1, il n'est pas toujours aisé de recruter des graphistes formés à *Blender* pour les projets ; les superviseurs sont souvent amenés à les former eux-mêmes.

Lors des expérimentations industrielles et académiques, nous avions observé l'apport de notre méthode pour la transmission de connaissance. Nous avons donc envisagé d'utiliser le *Multiuser* pour enseigner *Blender* aux nouveaux artistes ainsi que ses usages en production. Le bénéfice premier de cette application est évident : paralléliser l'apprentissage entre nouveaux arrivants. L'ajout de l'aspect interactif et social permettra également de briser la glace d'une éventuelle timidité pour accélérer l'intégration des nouveaux dans les équipes.

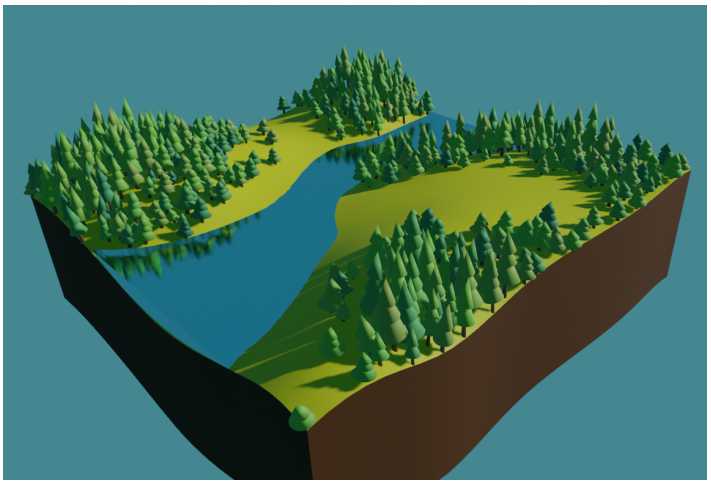


FIGURE II.3.2 – Résultat attendu à la fin de la formation aux *geometry nodes*.

Pour explorer cette application, Fabian Adam (stagiaire développeur sur le *Multiuser*) et Tanguy Weyland ont organisé une session de formation en ligne, ouverte aux

artistes de Cube Creative, pour enseigner les *geometry nodes*, un outil de modélisation procédurale nodale publiée dans la version 2.92 de *Blender*. La formation comptabilisa six apprenants, dont trois présents virtuellement dans la session. Plus précisément, l'objectif du cours était d'apprendre à faire une distribution d'arbres sur une colline en contrôlant leur densité, leur orientation, et leur taille avec les *geometry nodes* (cf. Figure II.3.2).

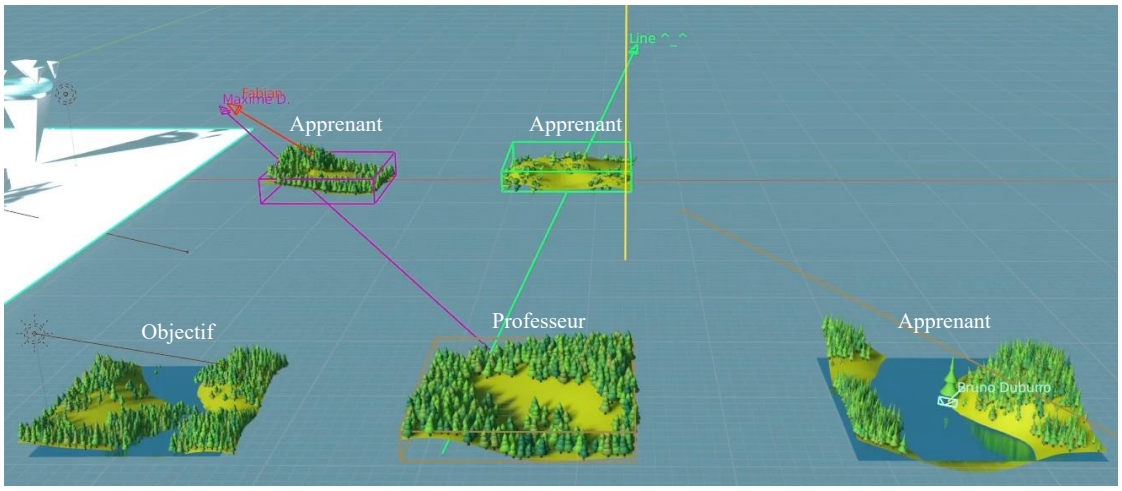
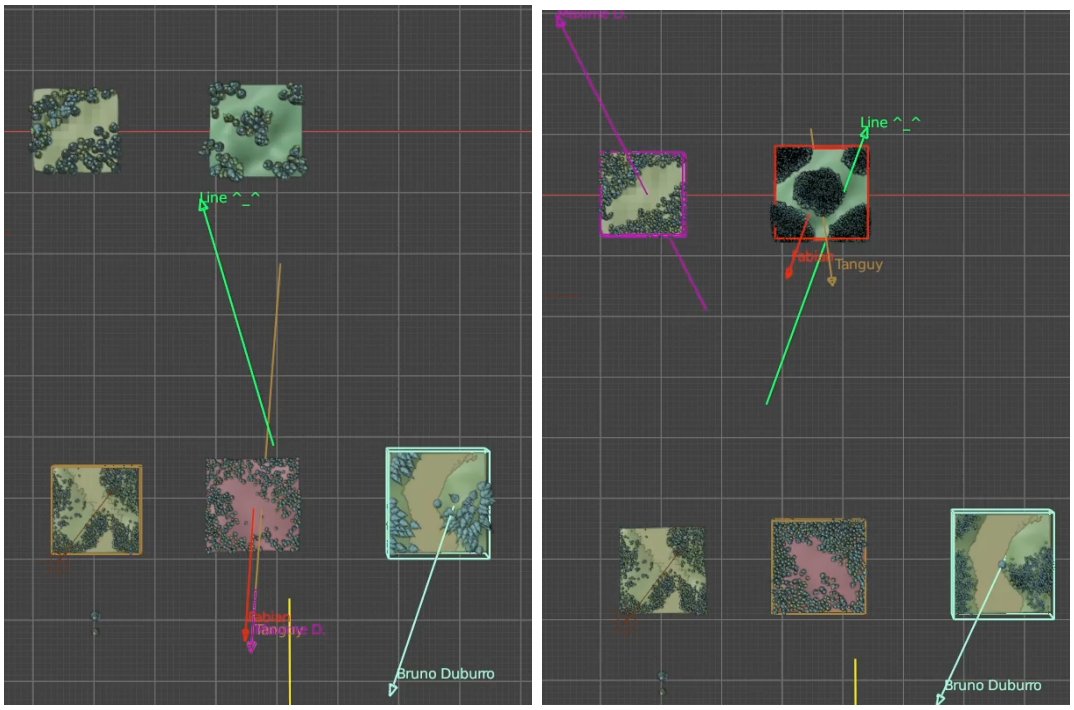


FIGURE II.3.3 – Organisation de l'espace.



(a) Phase théorique. (b) Phase pratique.

FIGURE II.3.4 – Localisation des participants lors des différents cycles du cours.

La scène a été découpée de façon à ce que chaque apprenant ait son espace pour reproduire directement les éléments enseignés (noté 'Apprenant' sur la Figure II.3.3). Tout au long du cours, Tanguy reconstruisait progressivement l'« Objectif » sur son

espace (« Professeur » sur la Figure II.3.3) avec les étudiants. Pour chaque nouvelle notion, le cycle suivant a été observé :

— théorie : les étudiants se regroupent virtuellement autour de l'enseignant qui montre la notion dans son espace. Ils observent ses actions en temps réel (cf. Figure II.3.4a) ;

— application : les étudiants reproduisent l'exercice dans leur espace. À tout moment, ils peuvent retourner voir la saynète du professeur en cas de besoin (cf. Figure II.3.4b).

Sur la Figure II.3.4b, on observe Tanguy entrain d'aider Line sur un problème avec la densité de sa distribution. En organisant la formation de cette façon, Tanguy avait une vision d'ensemble du travail des apprenants. Cela lui permit de détecter très rapidement une personne en difficulté pour venir l'aider.

II.3.1.3 *Look development*

Au cours des sessions de *background concept*, nous observions que le *Multiuser* facilitait l'émergence d'une vision commune. Les superviseurs ont donc proposé de tenter de l'utiliser dans le processus de recherche de style visuel 3D. En s'appuyant sur *Eevee*, le directeur artistique et ses *concept artists* seront en capacité de prototyper ensemble des décors de séries et les ambiances lumineuses associées.

II.3.1.4 Découpage de séquence

Le découpage de séquence consiste à établir la succession de plans de caméra qui compose la séquence. En cinéma, nous avons constaté que la *previs on set* permettait au réalisateur, directeur de photographie et superviseur VFX d'explorer la mise en scène et le cadrage pour définir une première version des plans.

Nous envisageons d'appliquer un concept similaire pour la création des plans en amenant le réalisateur et le directeur artistique à travailler ensemble dans la même scène avec le *Multiuser*. L'application *Smartphone Remote* (cf. Sous-section I.2.3.1) pourrait être mise à contribution pour se déplacer intuitivement dans la scène et mettre en place des mouvements de caméra.

Au moment où nous avons développé l'*add-on* pour *Blender*, il ne supportait aucune forme de RV, ce qui empêcha l'exploration de son utilisation pour la collaboration en temps réel. Cependant, dans la version 3.0 de *Blender* prévue pour décembre 2021, il est annoncé qu'il sera possible de manipuler les objets de la scène en RV. Dès lors, nous serons en mesure de plonger le réalisateur en RV pour étendre cette application à du *virtual scouting* collaboratif.

Dans son état actuel de développement, le *Multiuser* est à même de gérer ces applications. En revanche, durant nos expérimentations, nous avons relevé certaines pistes qui nécessiteraient d'étendre certaines fonctionnalités existantes ou d'en apporter de nouvelles.

II.3.2 Recherches et développements futurs

Tout au long de notre chemin exploratoire de la collaboration en temps réel, nous n'avons cessé de découvrir de nombreux embranchements de développement pouvant conduire vers d'autres horizons de recherches. Les fonctionnalités présentées dans cette section sont des pistes d'approfondissement qui étendraient le champ des possibilités de co-création en temps réel pour le cinéma d'animation.

II.3.2.1 *Rigging* avancé

Dans nos expérimentations publiques et industrielles, nous avons déterminé qu'il n'y avait aucun frein majeur à l'animation *keyframe* collaborative. Le support initial de l'animation a permis un défrichage important des spécificités de la gestion du temps entre les utilisateurs.

Cependant, nous n'avons pas terminé l'intégration de la gestion des *rigs* complexes. La finalisation de leur support dans le *Multiuser* permettrait, d'une part, d'étudier les capacités de superposition de cette étape de fabrication avec les autres, et d'autre part, d'explorer l'animation *keyframe* collaborative de personnages interagissant entre eux. Par exemple, serait-il possible d'animer simultanément deux personnages se serrant la main avec un animateur responsable de chaque personnage ?

II.3.2.2 gestion de versions

En ajoutant l'aspect temps réel au sein de la collaboration, nous nous sommes éloignés de la nature incrémentale du travail. De ce fait, nous avons laissé de côté le mécanisme de *review* qui se repose dessus. Tels que présentés, le *framework* ainsi que le *Multiuser* ne stockent que la version actuelle des données de création. Ce seul état global de la scène est insuffisant supporter des *retake*s sélectifs d'une scène créée par une équipe pluridisciplinaire (par exemple, il n'est pas possible de revenir en arrière uniquement sur le *lighting*).

Pour pallier ce problème, il serait intéressant de considérer la validation artistique dans le contexte d'une scène créée en temps réel par une équipe. L'ajout d'un système de gestion de versions des changements (cf. Figure II.3.5) pourrait aider les équipes de production à suivre le travail de l'équipe artistique dans le temps.

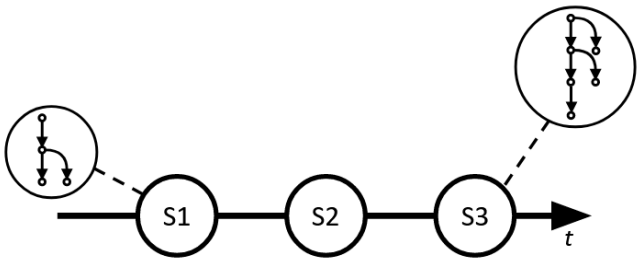


Figure II.3.5 – Illustration d'une proposition de système de gestion de versions pour le *Multiuser*.

On peut imaginer un système sauvegardant une version complète du *repository* à un instant *t*. Ces sauvegardes pourraient être conduites de façon automatique ou manuelle.

La stratégie automatique, déclenchée à un intervalle de temps régulier répond aux exigences de sauvegarde et de sécurité.

À l'opposé, dans la stratégie manuelle, l'artiste piloterait la création des points de sauvegarde depuis le DCC. Cette méthode serait idéale pour itérer précisément sur un aspect de la scène. Elle servirait de base pour effectuer des revues de validation en fonction d'un état donné de l'œuvre.

Dans le concept, ce système se rapprocherait d'une version très simplifiée de *Git*.

II.3.2.3 Système de cache

Actuellement, le *framework* stocke l'ensemble du *repository* en mémoire, limitant la taille des scènes prises en charge à quantité la mémoire disponible sur l'ordinateur du client.

Une perspective de travail consisterait à ajouter un système de cache du *repository* sur disque pour supporter des scènes 3D plus importantes. En le configurant via RDP, chaque type de *datablocks* bénéficierait d'une stratégie de mise en cache adaptée à ses besoins (par exemple, les fichiers externes comme les images seraient immédiatement mis en cache, car ils sont très peu modifiés au sein de *Blender*).

II.3.2.4 Système de rôles

Lorsqu'un artiste sélectionne un objet de la scène, la gestion automatique des droits décrite en Sous-section II.1.3.4 va verrouiller le *datablock* correspondant et toutes ses dépendances. Par conséquent, l'artiste récupère les droits sur tous les aspects de l'objet (cf. Figure II.3.6) tel que son *mesh*, son matériau, ses textures, etc.

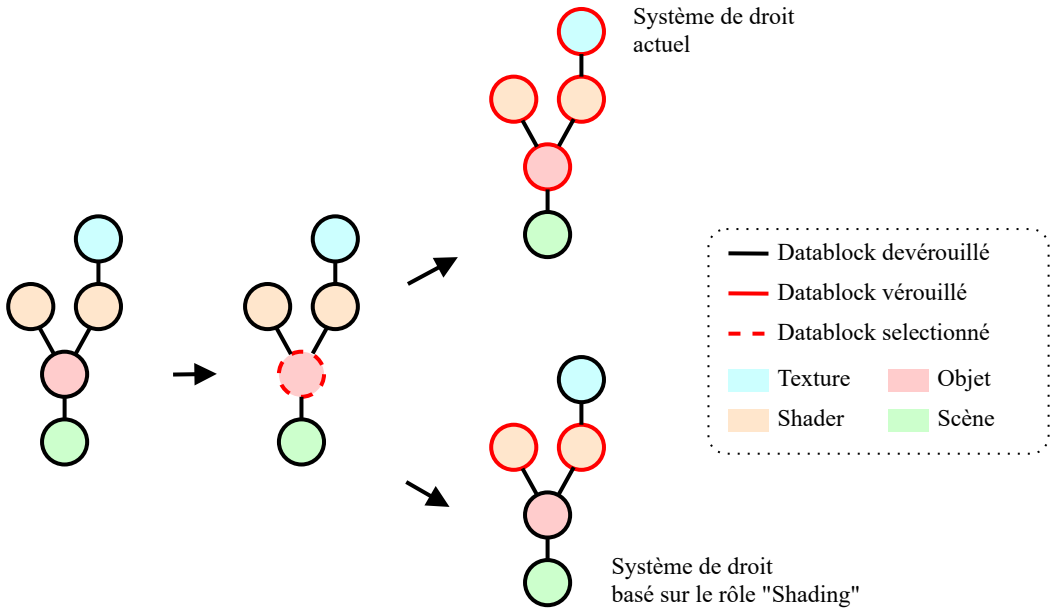


FIGURE II.3.6 – Comportement du système de droit actuel et idéal lors de la sélection d'un objet dans la scène.

En l'état actuel, il est donc impossible de superposer la création de différents aspects du même objet à travers plusieurs artistes [197].

197. À moins de contourner la limitation de façon sous optimale en développant les aspects tels que le *shading* sur un autre objet puis de les reporter sur l'objet initial.

L'ajout d'un mécanisme de rôle au système de droit de l'*add-on* viendrait aisément à bout de cette limitation. Le principe serait que chaque artiste choisisse un rôle (*lighting*, modélisation, etc.) qu'il pourrait changer tout au long de la session. Ce rôle sera ensuite exploité comme un filtre par le système de droit qui ne verrouillera que les *datablocks* directement reliés au rôle (cf. Figure II.3.6). Cette approche permettrait par exemple à un artiste de faire le *shader* d'un objet tandis qu'un autre l'anime, poussant ainsi la superposition des tâches à l'échelle des *datablocks*.

II.3.2.5 Administration serveur

La création de sessions sur internet a été simplifiée une première fois lors de l'adaptation du *Multiuser* aux contraintes du réseau (cf. Sous-section II.2.4.2) mais de nombreux aspects pourraient être améliorés pour faciliter les sessions persistantes.

Actuellement, les données d'une session ne peuvent être sauvegardées que depuis le client ; la fonctionnalité n'est pas exposée depuis la console serveur. Ajouter cette possibilité côté serveur ouvrirait la porte à un système de sauvegarde automatique sur ce dernier. Ce système pourrait être intelligemment géré en fonction des évènements de connexion au serveur (par exemple, une sauvegarde serait faite à chaque déconnexion).

Une seconde limitation réside dans la gestion de la session : cette dernière ne peut être lancée que lorsqu'un administrateur se connecte et l'initialise (cf. Sous-section II.2.4.2). Ainsi, lorsque l'on veut démarrer la session depuis une sauvegarde, il faut l'importer cette sauvegarde dans *Blender*, se connecter au serveur dédié en tant qu'administrateur et initialiser la session. Il serait extrêmement pratique de pouvoir démarrer le serveur dédié directement sur la sauvegarde pour éviter ce processus fastidieux.

Dans les premières sessions publiques, nous avons régulièrement fait face à des problèmes de scène corrompue par des erreurs de réplication. Pour les résoudre, il était nécessaire de se connecter à distance au serveur pour redémarrer le container [198] *Docker* afin de réinitialiser la session. La création d'une commande dans le *framework* pour purger le graphe de réplication corrompu éviterait d'avoir à effectuer un *reset* du container côté serveur. Cette commande pourrait ensuite être directement exposée depuis l'interface graphique de l'*add-on* sur *Blender* pour réinitialiser la session à distance en un clic.

Tous ces détails faciliteraient la création et la maintenance d'espaces de co-créations persistants sur internet.

198. Dans *Docker*, un conteneur est une unité logicielle qui regroupe le code et toutes ses dépendances afin que l'application fonctionne rapidement et de manière fiable d'un environnement informatique à un autre.

Conclusion

Ce chapitre a abordé concrètement les suites envisagées à notre recherche. La première section a montré les applications à court terme de nos travaux dans le pipeline de fabrication de Cube Creative. L'objectif sera de l'exploiter dans des étapes charnières du processus de fabrication pour faciliter la communication et la transmission de connaissances au sein des artistes. La seconde section a exposé les développements envisagés à long terme. Ces propositions d'approfondissement de la recherche et du développement du *Multiuser* sont basées sur les limitations observées lors des sessions expérimentales.

Mais amener plusieurs artistes de différents départements à travailler ensemble pose une remise en question profonde de l'appareil de production. En propulsant le groupe à la place de l'individu, la collaboration en temps réel est-elle en train d'« agilifier » la création ?

Chapitre II.4

Vers des productions agiles ?

Introduction

Dans le premier chapitre de la première partie, nous avons constaté que la production des films d'animation reposait sur une méthode en cascade. Ce modèle de planification est aujourd'hui utilisé dans de nombreux secteurs tels que la construction, le développement, etc. En anticipant tout le cycle de production, il représente l'avantage de fournir un cahier des charges et un budget clair. Mais, nous avons également relevé un certain nombre d'inconvénients :

— pas de place pour les imprévus ;

— une rigidité face aux changements ;

— des ajustements coûteux.

Le second chapitre de la première partie marqua la remise en cause de ce système avec l'arrivée du temps réel dans les productions.

Dans le chapitre précédent, nous avons décrit les prochaines applications envisagées du *Multiuser*, dans un pipeline traditionnel. Il n'a pas été poussé au cœur des étapes de production, car dans le processus actuel, les méthodes d'assignation et de validation sont individuelles (cf. Sous-section I.1.2.3). En effet, pour exploiter la collaboration au cœur de la production, il faudrait utiliser des méthodes qui remplacent l'individu par le groupe.
Comme Anne-Laure George-Molland l'énonce dans la conclusion de sa thèse :

> *L'amélioration du travail collectif ne trouvera donc pas uniquement écho dans les développements logiciels. Elle devra aussi passer par la prise en considération et l'amélioration de problématiques humaines, la canalisation des conflits et la recherche de stratégies.* [18, p. 330]

Nos expérimentations (cf. Chapitre II.2) nous ont menés à utiliser des outils issus des méthodes agiles. Il fut également constaté que notre approche partageait de nombreux avantages avec le *pair programming*, une technique tirée de ces méthodes. Mais que sont les méthodes agiles ? Et surtout, sont-elles adaptées au secteur du cinéma d'animation ?

Dans le milieu du développement informatique, le problème de la méthode en cascade a déjà été étudié. Auparavant, la majorité des logiciels étaient conçus avec un cahier des charges très strict et des étapes prévues longtemps à l'avance. En plus des inconvénients listés précédemment, le développement logiciel faisait également face à un risque de déphasage entre le moment de l'analyse du besoin et la livraison finale

du produit. La méthode en cascade n'était donc pas apte à prendre en compte l'évolution des besoins du client, notamment dans le cas de projets de longue durée, ou bien lorsque les besoins clients avaient été mal définis ou mal interprétés.

Dans ce chapitre, nous décrirons les méthodes agiles qui furent créées en réaction à cette problématique ; nous nous attarderons sur le *framework* Scrum (cf. infra Sous-section II.4.1.2) particulièrement intéressant pour l'image de synthèse. Nous dresserons ensuite une étude de cas de productions de cinéma ayant exploré de telles méthodes pour réorganiser la fabrication d'images autour de groupes d'artistes. Enfin, nous décrirons comment le *workflow* de création agile qui a émergé des nombreuses sessions expérimentales s'inscrit particulièrement bien dans cette démarche.

II.4.1 Méthodes agiles

II.4.1.1 Définition

Les concepts agiles ont étés formalisés dans un manifeste agile [199] en 2001 aux États-Unis par un groupe de 17 experts en développement logiciel. Chacun de son côté avait expérimenté des alternatives aux méthodes en cascade. Ce manifeste définit les dénominateurs communs à leurs expériences. Les méthodes agiles existaient donc avant le manifeste agile, mais étaient peu connues.

Ce manifeste prône les valeurs suivantes :

— **les individus et leurs interactions** priment plus que plus que les processus et les outils ;

— **des logiciels opérationnels** priment plus que plus qu'une documentation exhaustive ;

— **la collaboration avec les clients** prime plus que plus que la négociation contractuelle ;

— **l'adaptation au changement** prime plus que plus que le suivi d'un plan.

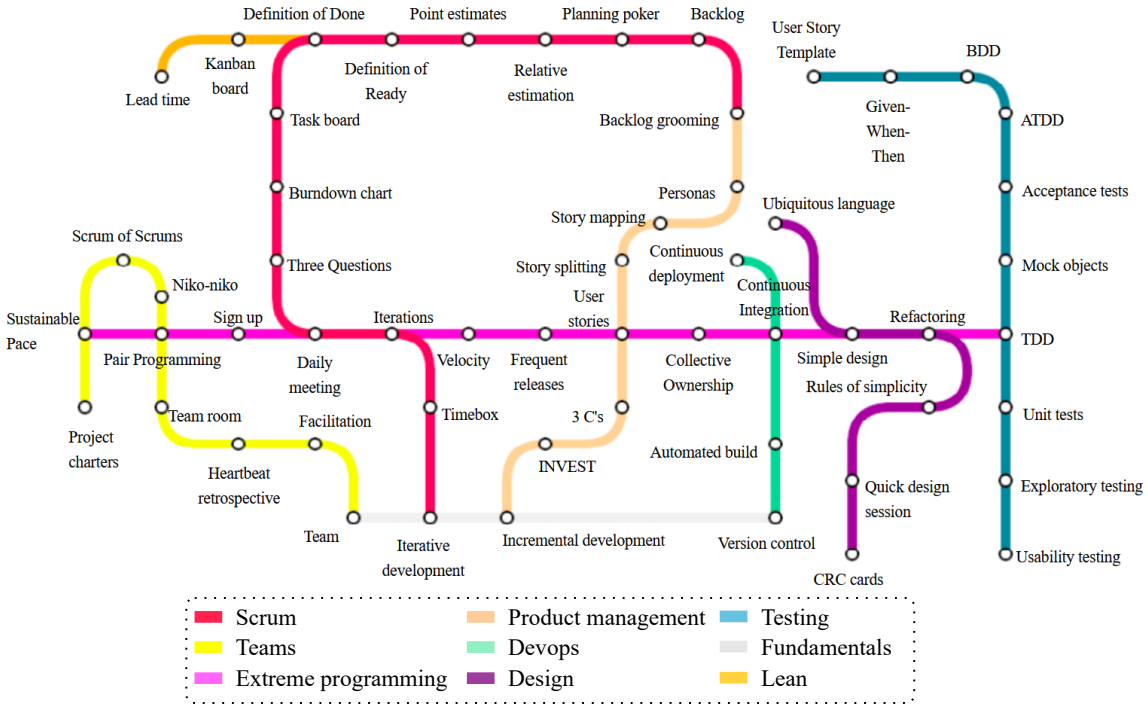


FIGURE II.4.1 – Carte des différentes méthodes agiles. Source : `https://www.agileallia nce.org`

Il décline ces quatre valeurs en douze concepts [200]. Ces principes mettent en avant la collaboration entre des équipes auto-organisées pluridisciplinaires et leurs clients. Centrés sur l'humain et la communication, ils favorisent une planification adaptative et un développement évolutif. En utilisant des livraisons précises tout au long de la fabrication, ils encouragent des réponses flexibles au changement. Les méthodes agiles se

199. `http://agilemanifesto.org/iso/fr/manifesto.html`
200. Liste des douze principes du manifeste agile : `http://agilemanifesto.org/iso/fr/principl es`

veulent plus pragmatiques que les méthodes traditionnelles, elles impliquent au maximum le demandeur (client) et permettent une grande réactivité à ses demandes. Méthodologiquement parlant, les principes proposent des cycles de développement itératif, incrémental et adaptatif.

Si l'on assimile l'agilité à un gâteau, il existe un certain nombre de recettes pour en faire. La Figure II.4.1 propose un aperçu des principales recettes utilisées pour mettre en place des productions agiles. Chaque couleur représente alors une recette et les stations sont différents ingrédients. Comme on peut le constater, certaines recettes utilisent des ingrédients en commun.

Naturellement, il n'est pas dans notre intérêt d'aborder toutes ces recettes. Nous irons à l'essentiel en abordant la plus utilisée dans le milieu du cinéma d'animation. Lors de nos recherches sur l'utilisation de méthodes agiles dans les productions d'animation (exposé dans la section suivante), la méthode Scrum a fait l'unanimité.

II.4.1.2 Scrum

On pourrait penser que, rendues populaires par le monde de l'ingénierie logicielle, les méthodes agiles seraient principalement destinées aux développeurs et ingénieurs, mais ce n'est pas le cas. Par exemple Scrum a été largement utilisé au Federal Bureau of Investigation (FBI), dans des agences de publicité ou encore dans le génie civil. Les méthodes agiles peuvent s'appliquer sur tout projet comptant un produit à développer. Les films et les jeux ne font pas exception. Nous nous sommes appuyés sur le Scrum Guide [201] pour décrire la méthode.

Les fondements de Scrum se résument à deux concepts :

— l'**empirisme**, qui affirme que la connaissance vient de l'expérience et que la prise de décisions est basée sur des faits observés ;

— le **lean thinking**, qui se focalise sur l'essentiel et réduit le gaspillage d'effort.

Ces concepts sont cristallisés dans trois grandes valeurs :

— la transparence : chacun doit pouvoir voir ce sur quoi les autres travaillent ;

— l'inspection : les artefacts (cf. infra paragraphe II.4.1.2) et les progressions doivent être inspectés fréquemment avec diligence et efficacité pour trouver les écarts ou les problèmes ;

— l'adaptation : s'il y a des aspects du processus qui s'écartent de la limite acceptable, on doit s'adapter le plus rapidement possible.

Nous décrirons la méthode Scrum à travers les rôles, artefacts et éléments du *framework* Scrum. Nous illustrerons notre exposé par des exemples en place au sein de l'équipe R&D de Cube Creative qui l'utilise pour organiser le développement du pipeline au quotidien.

Rôles

Le *framework* Scrum établit deux principaux rôles dans le *workflow* : une équipe de développement et un *Product Owner* (décrit plus bas).

201. `https://scrumguides.org/scrum-guide`

Équipe de développement : une équipe Scrum est auto-organisée, aucun intervenant extérieur ne lui indique comment travailler sur les tâches. Idéalement, il n'y a pas de hiérarchie, aucun des membres n'a de titre particulier. Cependant, si la méthode n'est pas adoptée dans toute l'entreprise, un membre de l'équipe doit la représenter et la coordonner pour qu'elle s'insère correctement dans le reste de la société. À Cube Creative, c'est le directeur R&D Valentin Moriceau qui occupe ce rôle.
En cas de problème, l'équipe entière est tenue responsable, et doit trouver collégialement une solution. Il est arrivé que le pipeline casse à Cube Creative et que toute l'équipe soit sur le pont pour éteindre le feu. Notons que cela dépend de l'ampleur du problème : sur de petits bugs très ciblés (par exemple un outil qui ne démarre pas à cause d'un commit problématique) un développeur ou deux suffisent amplement.

Pour éviter de compliquer la gestion du travail, une équipe ne doit pas dépasser neuf membres. Ceux-ci sont pluridisciplinaires afin de lui permettre d'achever des itérations complètes du produit qu'elle développe.

***Product owner* :** le *product owner* est généralement externe à l'équipe de développement. Il représente les intérêts du client et à ce titre, il a l'autorité pour définir les fonctionnalités du produit final.

À Cube Creative, Valentin occupe ce rôle en canalisant les demandes de la production et des superviseurs à travers un ensemble de fonctionnalités. Dans notre cas, il fait aussi partie de l'équipe de développement, car il représente ses intérêts auprès des autres équipes du studio.

Artefacts

Dans les méthodes agiles, les artefacts sont des documents produits et utilisés dans le *workflow*. Dans Scrum, on distingue principalement des artefacts de type *backlog*. Concrètement, un *backlog* est une liste de tickets représentants les tâches et spécifications du produit. Il y a le *product backlog* et le *sprint*[202] *backlog*. Ces deux artefacts sont liés : le *sprint backlog* est un sous-ensemble du *product backlog*.

***Product backlog* :** le *product backlog* résume toutes les fonctionnalités du projet à développer. Le *product owner* en a la responsabilité et y priorise chacune des tâches en fonction de leur importance. À titre d'exemple, dans le *kanboard* du *Multiuser* (cf. Figure II.2.23), une colonne dédiée au *backlog* permet d'y visualiser et prioriser les tâches. La Figure II.4.2 illustre une petite section du *backlog* de la R&D maintenu sur *GitLab*.

C'est l'artefact principal de la méthode Scrum, utilisé pour coordonner les tâches à travers les évènements qui rythment le développement.

Sprint backlog : le *sprint backlog* contient une partie du *product backlog* à réaliser pendant un temps donné aussi appelé *sprint* (que nous définirons un peu plus bas). C'est un document fait par et pour les développeurs, qui est mis à jour tout au long du *sprint* pour suivre l'avancement du travail. Dans le *backlog* de Cube Creative (cf. Figure II.4.2), le *sprint backlog* est constitué des tâches classées *This Week* (cette semaine). Pour le *Multiuser*, il est représenté à travers la colonne *To Do* (cf. Figure II.2.23).

202. Dans le *framework* Scrum, un *sprint* représente un intervalle de temps durant lequel l'équipe de développement va planifier puis compléter une certaine quantité de tâches.

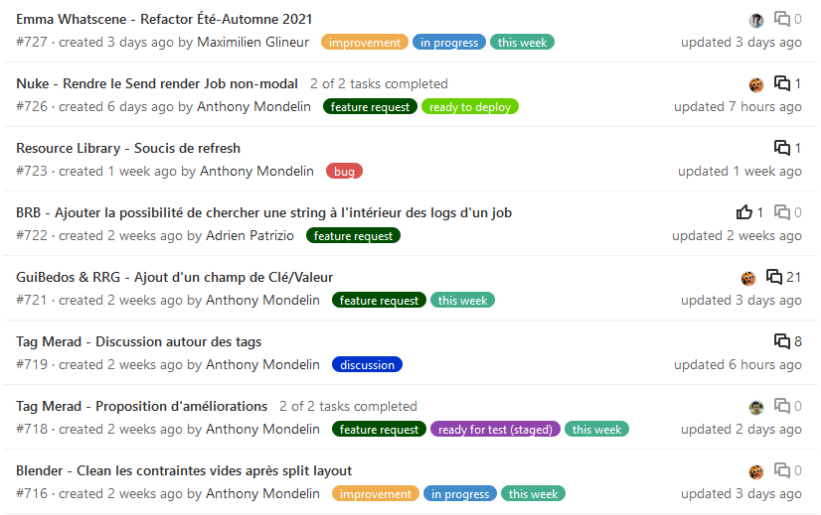


FIGURE II.4.2 – Exemple d'une partie du *backlog* de Cube Creative

Évènements

La temporalité d'un cycle de production d'une équipe Scrum est rythmé en *sprints*. Les *sprints* sont des intervalles de temps pendant lesquels l'équipe va compléter les tâches du *sprint backlog*. C'est le cœur du réacteur où les idées se matérialisent. La durée du *sprint* dépend des besoins de l'équipe, mais deux semaines est une durée communément utilisée. Dans l'équipe R&D, les *sprints* durent une semaine. Dans cette section, nous décrirons linéairement les principaux évènements qui formalisent le *sprint*.

Planification du *sprint* : la planification du *sprint* marque le début du cycle (cf. Figure II.4.3). Elle est effectuée par toute l'équipe de développement. Son but est de définir ce qui peut être terminé au cours de ce *sprint* et comment ce travail sera effectué. Durant cet évènement, l'équipe de développement est amenée à prendre des fonctionnalités du *product backlog* pour les porter dedans. Ces fonctionnalités peuvent être divisées en sous-tâches.

Au sein de la R&D, nous appelons cet évènement le « Tour Ticket ». Tous les lundis matin, un développeur différent énonce les différents tickets en cours, puis dans un second temps, il est déterminé collégialement ceux qui feront l'objet du *sprint* à venir. Ces tickets sélectionnés sont classés *this week* (cf. Figure II.4.2).

Daily Scrum : les équipes se rencontrent tous les jours en début de matinée pour le "daily scrum", une réunion quotidienne d'environ quinze minutes où chacun fait part aux autres de son avancement. Tour à tour, chaque membre énonce :

— ce qu'il a fait le jour d'avant ;

— ce qu'il fera le jour même ;

— d'éventuels problèmes rencontrés.

Cette rencontre tient tout le monde informé du travail de chacun et permet de réagir vite aux problèmes qui peuvent se produire.

À l'origine, cette réunion était appelée le « Matin Debout » (*standup meeting*) à la R&D, nous nous réunissions tous debout en cercle pour aborder ces points tour à tour. Avec la crise sanitaire, cet évènement a évolué sous la forme d'un canal textuel dénommé « Avance Assis » dans lequel nous écrivons chaque jour les informations énoncées initialement au « Matin Debout ».

Sprint Review : la *sprint review* arrive en fin de *sprint*. C'est une réunion incluant le *product owner* qui vise à faire l'état des lieux : on présente le travail effectué et on prépare les tickets probables pour le prochain *sprint*. Suite à cette rencontre, le *product owner* peut être amené à préciser certaines fonctionnalités du *product backlog*.

À Cube Creative, cette réunion surnommée « Pestacle », se tient toutes les deux semaines. Elle regroupe superviseurs, chargés de production et l'équipe R&D. Pendant ce temps de partage, l'équipe présente les avancements du développement des outils et introduit les nouveautés par des démos. À la fin du « Pestacle », les participants sont invités à faire des retours sur leurs besoins et les éventuels problèmes qu'ils rencontrent sur des outils déployés en production.

Sprint Retrospective : La *sprint retrospective* conclut le cycle du *sprint*. C'est la réunion la plus importante d'un point de vue humain, car elle resserre l'équipe à propos de ce qui fonctionne, ne fonctionne pas et ce qu'il faut améliorer, non techniquement, mais en termes de manière de travailler ; C'est un outil de construction de l'équipe et de la collaboration.

Dans l'équipe R&D, cette réunion est fusionnée avec les autres en fonctions des sujets. Lorsque cela concerne uniquement l'équipe de développement, ils sont abordés lors du « Avance Assis » (*sprint planning*) ; lorsqu'ils portent sur des mécanismes de collaboration entre R&D et superviseurs/chargés de production, ils sont abordés lors du « Pestacle » (*sprint review*).

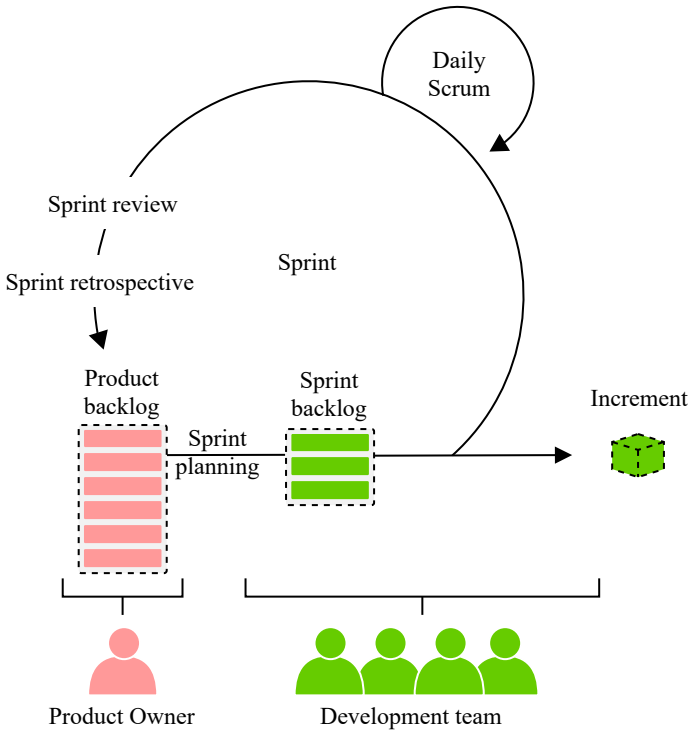


FIGURE II.4.3 – *Workflow* Scrum.

Après avoir brièvement décrit la méthode Scrum, on retrouve deux idées principales : l'itération et l'amélioration.

À la fin d'un *sprint*, le délivrable doit être prêt à être livré au client. Attention, le but ici n'est pas d'avoir un projet fini, mais plutôt d'avoir atteint un stade suffisamment avancé pour pouvoir montrer quelque chose au client. Pour l'équipe R&D, cela peut se traduire par la fonctionnalité d'un outil par exemple. Cette notion de livrable en

fin de *sprint* est très importante, car elle permet de collecter des retours d'expérience utilisateur plus tôt pour guider le développement. C'est aussi utile pour obtenir un résultat en phase avec les attentes du client.

Par conséquent, on évite l'un des problèmes courant dans la méthode en cascade où le résultat ne satisfait pas les attentes d'un client. Car au lieu de délivrer seulement le produit final en fin de production, on délivre de petits tronçons du produit à la fin de chaque *sprint* (cf. Figure II.4.4).

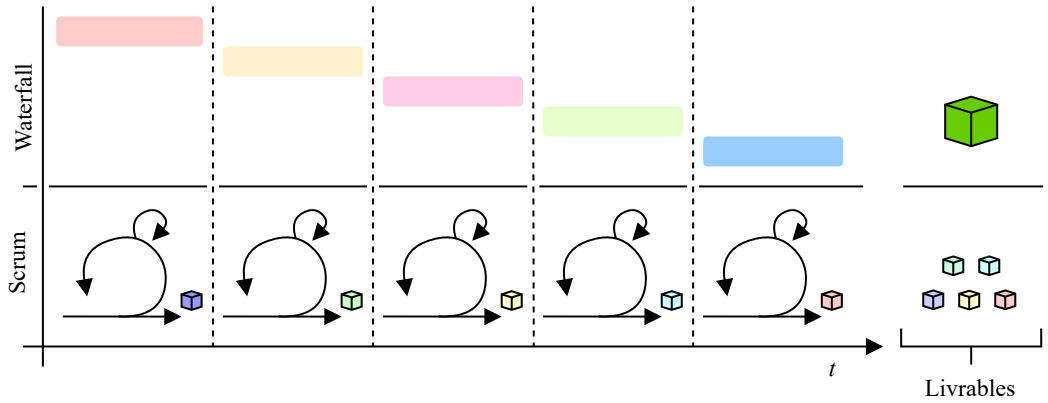


FIGURE II.4.4 – Comparaison de la méthode *Scrum* et de la méthode en cascade.

Au-delà de l'importance de l'itération, la méthode s'attache également à améliorer le processus en lui-même à chaque nouveau cycle. Pendant la rétrospective les équipes discutent des éléments qui pourraient les faire gagner en efficience.

Finalement, à l'inverse de la méthode en cascade qui focalisent les membres de l'équipe sur des tâches individuelles (par exemple, l'artiste sur une tâche de *layout*), Scrum tente de focaliser l'équipe sur un but commun : la réalisation du projet.

Mais la mise en place d'une méthode agile n'est pas sans risques. Elle nécessite que tous les acteurs soient impliqués dans le processus et acceptent de jouer le jeu. En favorisant des cycles de développement court et adaptatif, cette méthode laisse peu de place à la création de documentation pour les outils.

II.4.2 Méthodes agiles pour l'image de synthèse

La section précédente décrit les principales valeurs agiles et l'une des méthodes les appliquant : Scrum. Durant l'exposé, nous avons ainsi observé son emploi dans développement des fonctionnalités nécessaires au pipeline de production en image de synthèse. Mais l'application de ces méthodes à la production d'un film d'animation pourrait changer également notre regard sur la manière de les fabriquer.

Dans un premier temps, nous étudierons comment la méthode Scrum peut restructurer la chaîne de fabrication autour du groupe artistique à travers l'étude de trois cas d'application de l'industrie.

Nous décrirons ensuite comment le travail collaboratif en temps réel s'intègre dans cette méthode à l'échelle d'une équipe, au cœur de la création.

II.4.2.1 Production agile

S'il est vrai que les équipes R&D des studios changent leur modèle de développement logiciel pour exploiter des méthodes agiles, les productions sont encore loin de les adopter largement. Jusqu'à maintenant les recherches scientifiques sur l'utilisation de ces méthodes en audiovisuel sont quasiment inexistantes, par conséquent, nous nous sommes basés sur trois cas concrets de l'industrie :

— Cinnamon VFX : un studio d'effets spéciaux ukrainien [26] ;

— Main Road Post : un studio d'effets spéciaux russe [30] ;

— l'expérience de Gwenaëlle Dupré (*product manager* chez CGWire) dans une production de série d'animation [22].

Dans les trois cas, l'adoption des méthodes agile s'est faite par nécessité. Cinnamon VFX a été fondé pour explorer ces méthodes suite à une lassitude face aux problèmes induits par la méthode traditionnelle (telles que les périodes de *rush* - précipitation). Chez Main Road Post, l'adoption s'est faite suite à leur croissance. Le studio a démarré avec une équipe de six personnes, la communication était alors facile. Mais lorsqu'ils sont devenus cinquante, des départements ont été mis en place et la communication se complexifia énormément entre les équipes. Ils ont basculé vers les méthodes agiles en réaction à ces problèmes. De son côté, Gwenaëlle Dupré a utilisé la méthode à son arrivée sur une production qui avait accumulé beaucoup de retard, autrement dit pour « éteindre un incendie qui avait bien débuté ».

Nous utiliserons ces trois cas pour analyser le nouveau *workflow* qu'induisent ces méthodes pour les productions d'images de synthèse. Pour faciliter l'analogie avec Scrum, nous avons suivi un découpage des explications similaire.

Équipes

Lorsque la méthode Scrum est appliquée dans un studio de création, on parle d'équipe artistique et non d'équipe de développement. Dans les différents cas étudiés, deux schémas d'équipes agiles se détachent :

— pluridisciplinaire ;

— spécialisée.

Chez les deux studios en VFX, on retrouve des équipes pluridisciplinaires. Tandis qu'en série d'animation, Gwenaëlle a utilisé des équipes spécialisées. Cette différence peut s'expliquer par l'échelle d'application de la méthode. À Main Road Post et Cinnamon VFX, la méthode est appliquée à l'ensemble du studio tandis que dans le cas de Gwenaëlle, elle est appliquée au niveau des départements (layout, animation, etc.).

La configuration pluridisciplinaire en place à Main Road Post est représentée sur la Figure II.4.5. La notion de département et de superviseur n'existe plus. Le réalisateur (R sur le schéma) s'adresse directement à chacune des équipes pour les *reviews*. Ici, le livrable est devenu un plan rendu, comprenant ainsi toutes les étapes de fabrication. Chaque membre de l'équipe travaille sur une partie différente du plan.

Dans une intervention au *meetup* de CG-Wire[203], le directeur du studio, Arman Yahin, précise que les membres des équipes ne sont pas forcément généralistes, c'est une somme de spécialités. À Main Road Post les équipes sont autonomes au point de décider du recrutement de leurs membres.

Dans cette configuration, les équipes représentent un petit studio dans le studio. Pour avoir une idée de l'échelle de l'application, en 2019, Main Road Post comptait quatre-vingt-dix personnes réparties en quatorze équipes (dont une de renfort, ce qui fait des équipes d'environ six personnes).

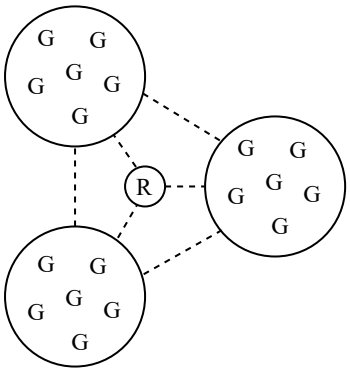


FIGURE II.4.5 – Relations au sein du studio Main Road Post entre les graphistes (G) répartis en équipes pluridisciplinaires et le réalisateur (R).

La configuration spécialisée est représentée sur la Figure II.4.6. On observe que les départements sont conservés. De ce fait, le livrable attendu de chaque équipe est similaire à celui d'un *workflow* traditionnel : l'aspect traité par la spécialisation (par exemple, pour l'animation, ce sera une itération d'un plan animé : *animation T1*, *animation t2*, etc.). Il en va de même pour le cycle de validation : les superviseurs et le réalisateur effectuent respectivement les validations techniques et artistiques. Cependant, les *retake*s résultantes sont gérées par l'équipe (cf. infra II.4.2.1). Ici, chaque équipe travaille sur une séquence et se divise les différents plans à traiter. Par conséquent, les membres d'une équipe travaillent sur des plans qui se suivent, cela facilite la communication et la détection de problème lors des *daily Scrum* (cf. infra Figure II.4.2.1).

Dans son article [22], Gwenaëlle précise que les équipes de quatre à cinq artistes étaient composées de seniors, mids et juniors. Cela permettait d'avoir un niveau global correct et de ne pas laisser les juniors derrière. Pour communiquer avec le superviseur, chaque équipe avait un ambassadeur chargé de canaliser les discussions et faire gagner

203. Vidéo de l'intervention : `https://youtu.be/NpvWOuxXVhg`

du temps au superviseur. On pourrait penser que ce rôle s'apparente à celui du lead exposé dans le premier chapitre en Sous-section I.1.2.2, mais il est très différent, car il n'implique pas de responsabilité particulière hormis celle de résumer le travail effectué. Contrairement au lead, l'ambassadeur a le même pouvoir décisionnel que les autres.

En fin de compte, pour les équipes spécialisées, la différence avec un pipeline traditionnel réside principalement dans l'organisation du travail que nous aborderons dans les sections suivantes.

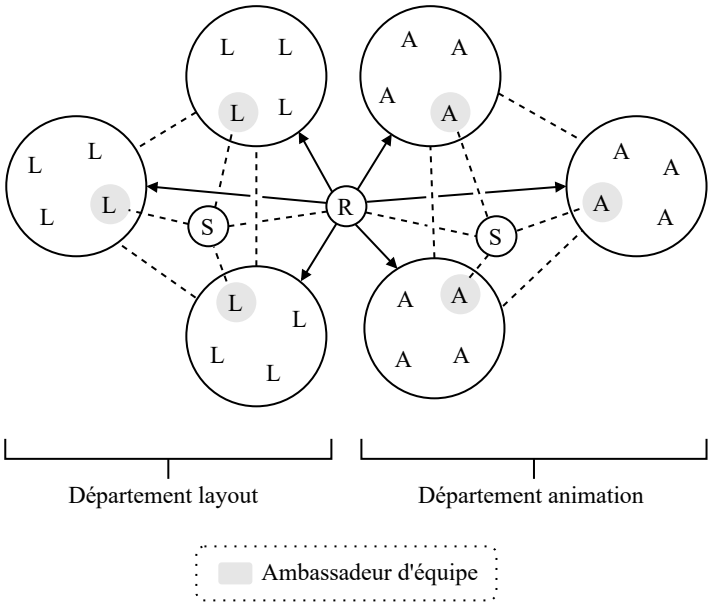


FIGURE II.4.6 – Structure d'équipe spécialisée utilisée par Gwenaëlle pour le *layout* et l'animation. Les équipes de graphistes layout(L) et d'animateur (A) interagissent avec les superviseurs (S) de leur département et le réalisateur (R).

Selon Scrum les équipes doivent être pluridisciplinaires pour être capables de produire en autonomie une itération de leur produit (cf. paragraphe II.4.1.2). Si l'on considère que le produit fini est le film, les équipes des studios VFX se rapprochent plus de cette philosophie.

Cependant, que les équipes soient spécialisées ou pluridisciplinaires, on constate qu'à l'inverse de la structure humaine décrite dans le Chapitre I.1, elles induisent une communication entre tous les acteurs. Les artistes ne sont pas isolés dans une bulle, mais rassemblés autour d'un objectif commun dans des équipes autogérées et composées de petits effectifs. On remarque ici que durant les expérimentations menées avec le *Multiuser*, et qui ont été décrites dans le chapitre précédent (cf. Section II.2.3), nous avions des configurations d'équipes très proches.

Après avoir examiné la structure humaine, nous allons observer comment le travail s'organise.

Segmentation des tâches

Selon les dirigeants de Cinnamon VFX [26], en premier lieu il est important de revoir l'échelle du temps. Découper les choses à faire en tâches réalisables en une journée permettra de faire des prévisions plus précises et de moins perdre de temps. En suivant cette logique, Alex Prihodko explique qu'il est plus facile d'estimer la durée de mille tâches simples plutôt qu'une seule complexe (cf. Figure II.4.7).

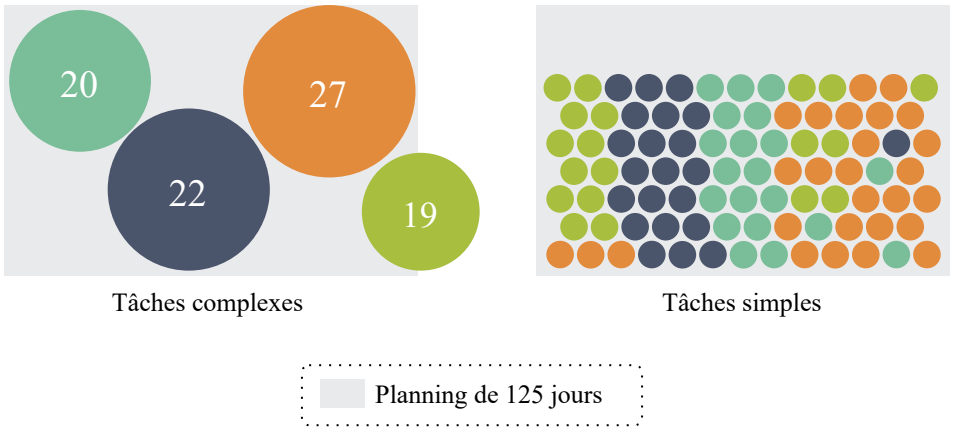


FIGURE II.4.7 – Différence de gestion entre des tâches complexes et simples ; leur planification est plus simple et précise lorsqu'elles sont petites et de durée similaire. Source : FXGuide [26]

Cette augmentation du niveau de détail des tâches entre en écho avec les conclusions de nos expérimentations sur la segmentation du travail (cf. Section I.1.2).

Estimation des tâches

Dans les trois cas étudiés, ce sont les équipes qui estiment combien de temps prendront les tâches. Seul Cinnamon fut plus explicite sur la méthode utilisée : le *sprint* poker.

Cette technique consiste à évaluer collégialement la complexité du travail par vote. Les options de vote sont en jours, on retrouve par exemple : 1, 2, 3, 5, 8, 13, 21 ou 34 jours. L'idée étant qu'au fur et à mesure que la durée augmente, l'erreur d'estimation augmente. Il est donc impossible qu'une estimation soit précise au-delà d'un jour ou deux. Ce procédé a de nombreux points positifs :

— l'exactitude des estimations est perceptible (plus une estimation est élevée, moins elle est précise) ;

— les membres de l'équipe « possèdent » leurs propres estimations ;

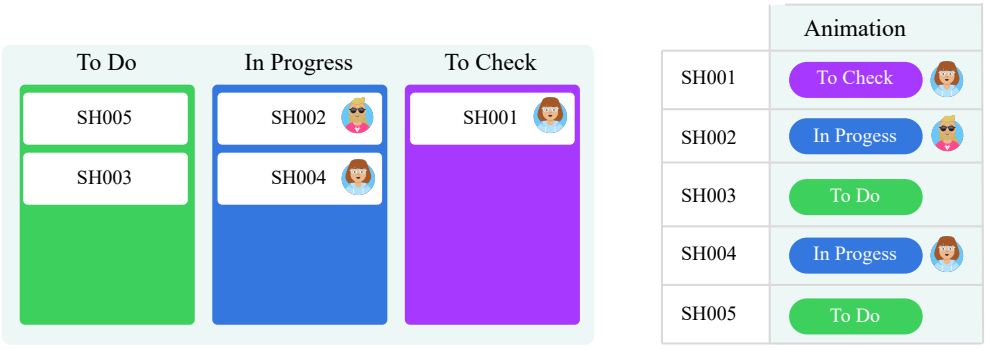
— si une personne estime une durée très différemment, alors peut-être qu'elle sait quelque chose que d'autres ignorent et pourra le faire remonter ;

— au fur et à mesure que le processus est répété et revu par *sprint*, l'équipe apprend et améliore rapidement les estimations.

Backlogs et suivi de la fabrication

Les trois studios ont utilisé le même outil pour gérer les tâches et le *backlog* : un *kanboard*.

Les tâches sont organisées et priorisées dans les colonnes de ce tableau. Par exemple, Gwenaëlle a utilisé les colonnes *To Do*, *In Progress*, *To Check* et *Question*. On note la similarité avec les statuts utilisés dans les *production trackers* exposés en Sous-section I.1.2.3. La différence principale entre un *kanboard* et un *production tracker* réside dans le sens de présentation des informations. À titre d'exemple, la Figure II.4.8 schématise le suivi de l'animation sur un *kanboard* et sur un *production tracker*. On observe que le *kanboard* liste les tâches (SH001, SH002, etc.) selon leur statut. À l'inverse le *production tracker* liste le statut de chaque tâche. Si le *production tracker* permet de suivre plusieurs étapes de fabrication, le *kanboard* est visuellement plus efficace



Kanboard — Production tracker

FIGURE II.4.8 – Exemple de suivi de l'animation sur un *kanboard* et sur un *production tracker*.

pour constater le travail qu'il reste à faire sur une étape (notamment parce qu'il est moins redondant). En représentant les tâches en « mouvement », il permet de visualiser clairement le flux de travail.

Tout au long des *sprints*, les équipes mettent à jour leurs tâches sur le *kanboard*. Ce dernier représente donc l'assignation et la progression du travail.

Dans l'expérience de Gwenaëlle, cet outil fut très utile pour les superviseurs. L'information étant centralisée dans le tableau, ils ne perdaient plus de temps à aller la collecter (interroger les équipes pour savoir où elles en étaient) et pouvaient se focaliser sur les validations et les plans qui posaient des questions (colonne *Question*). Elle note que le tableau représentait également un avantage pour les équipes de production permettant d'obtenir une vision d'ensemble de la progression de la fabrication en temps réel.

Sprints

En transitant vers Scrum, la production se mue en une succession de *sprints*. Dans les trois cas, les équipes coordonnaient leur *sprint* de manière similaire en suivant les trois phases exposées en Sous-section II.4.1.2 :

— *sprint planning* (cf. paragraphe II.4.1.2) : les équipes artistiques choisissent leur travail pour le *sprint* à venir ;

— *daily Scrum* (cf. paragraphe II.4.1.2) : les membres de l'équipe artistique montrent leur travail effectué la veille à leur pair et énoncent ce qu'ils feront le jour même. En se montrant mutuellement le travail lors du *daily Scrum*, les artistes donnent leurs opinions. Pour une équipe de cinq membres, il y avait donc quatre paires d'yeux pour trouver d'éventuels problèmes ou faire des suggestions positives. Gwenaëlle observe que ce processus permit de faire chuter le nombre de *retakes* techniques.

— *Sprint review* (cf. paragraphe II.4.1.2) : les équipes artistiques reviennent sur le travail effectué la semaine et évaluent si le processus peut être amélioré.

La durée des *sprints* dépend du type de production. À Main Road Post, la durée exacte n'est pas précisée, mais Arman Yahin précise qu'ils sont courts. Pour Cinnamon VFX, les *sprints* durent entre une et quatre semaines selon le type de projet (VFX ou télévision). Dans sa production, Gwenaëlle utilisa des *sprints* d'une semaine. Lorsqu'elle est arrivée sur le projet, la production fonctionnait sur des cycles de six semaines. Mais cela provoquait des oublis d'éléments donnés lors du *brief* réalisateur. Les cycles courts

de Scrum permirent aux artistes de garder les informations en mémoire.

En fonction de la structure des équipes, les *sprints* peuvent dépendre les uns des autres. Comme les produits sur lesquels travaillent les équipes pluridisciplinaires (Main Road Post et Cinnamon VFX) sont indépendants (un plan par exemple), les *sprints* le seront également (on peut produire ces plans dans l'ordre que l'on veut). À l'inverse, les *sprints* sont dépendants pour les équipes spécialisées, car elles conservent la séparation entre les étapes de fabrication principales.

Gestion des *retakes*

Dans une chaîne de production traditionnelle, une *retake* est menée individuellement par l'artiste ayant travaillé sur l'aspect remis en question (cf. Sous-section I.1.2.3). Les méthodes agiles donnent la responsabilité à l'équipe et non à l'individu. Par conséquent, dans les trois cas d'application, les *retake*s étaient prises en charge par l'ensemble de l'équipe et l'individu qui en était à l'origine.

Cela représente plusieurs avantages :

— les autres membres de l'équipe apportent leurs regards sur ce qui a provoqué le retake ; ceci permet d'avoir des résultats plus homogènes au sein de l'équipe ;

— les juniors vont passer moins de temps à reprendre leurs plans, car ils sont aidés par les mids et les seniors ;

— la communication et l'apprentissage sont améliorés, car les artistes discutent autour de la *retake* et les plus expérimentés diront aux novices ce qui dysfonctionne.

À travers cette étude de cas, nous avons constaté que les méthodes agiles étaient compatibles avec les productions de cinéma d'animation. En revanche, cela demande une réorganisation profonde des relations humaines, de la temporalité des cycles de production et l'usage d'outils de suivis adéquats.

Selon l'échelle d'application de la méthode, la notion de départements peut être conservée via la formation d'équipes spécialisées ou disparaître au bénéfice d'équipes pluridisciplinaires. Comme le relève Gwenaëlle dans son expérience, ce mode d'organisation est difficile à faire accepter à l'organe de financement habitué des plannings de production à long terme. Ce manque de confiance est l'une des causes de son faible taux d'adoption. La méthode doit encore faire ses preuves pour rentrer dans la culture de l'entreprise.

On pourrait s'interroger sur la place de l'équipe technique dans cet environnement. Les études de cas n'ont pas donné de détails sur cet aspect. Cependant, on peut s'appuyer sur notre vécu à Cube Creative où l'équipe R&D (qui suit cette méthode) considère la production et les superviseurs comme des clients qui lui commandent des outils. Par conséquent, elle devrait être en capacité de s'insérer dans les structures énoncées en Sous-section II.4.2.1 sans difficulté.

Les nombreuses similarités relevées avec les observations de nos expérimentations nous ont poussés à prendre du recul sur notre pratique, en découvrant un schéma agile au cœur de la création.

II.4.2.2 Vers une création agile ?

Tout comme Scrum, la collaboration en temps réel au sein de la fabrication des images de synthèse privilégie le groupe artistique. Ainsi, au même titre que *LiveShare*

supporte le *pair programming* (cf. Figure I.3.1.2) on peut considérer le *Multiuser* comme un outil agile supportant la co-création.

Cette section prend du recul sur les expérimentations et dessine les contours de la méthode de création agile qui en a émergée.

Équipes de création

Dans la section précédente, nous avions constaté qu'en fonction de l'échelle d'application de Scrum, les équipes d'artistes étaient pluridisciplinaires dans les studios VFX ou spécialisées pour le projet de série d'animation. Dans nos sessions de création, nous avions observé que ces deux catégories d'équipes fonctionnaient dans un contexte de collaboration en temps réel. On peut donc envisager d'appliquer cet outil à ces deux échelles.

Cependant, seules les équipes pluridisciplinaires tireront pleinement profit des avantages de la co-création en temps réel. Nous avions en effet constaté que les équipes pluridisciplinaires avaient la capacité, avec une segmentation correcte des tâches, d'atteindre une meilleure superposition du travail (en parallélisant la construction des *assets* et le *layout* par exemple). Cette configuration aura également une meilleure anticipation des erreurs, car les artistes mesurent en temps réel l'impact des différents aspects de la scène en cours de création. Par exemple, il est régulièrement arrivé au cours d'une session que l'artiste développant l'ambiance lumineuse se rende compte que le placement de certains objets (*layout*) était problématique et les déplace en conséquence. Une telle interaction entre les aspects de création n'est possible qu'avec une équipe pluridisciplinaire.

Dans les deux configurations d'équipe, nous avons observé l'apparition d'un nouveau rôle : le coordinateur (représenté sur la Figure II.4.9). L'artiste chargé de la coordination a deux missions en plus de sa tâche créative :

— il doit garder un œil sur le projet (par exemple le plan) dans sa globalité pour s'assurer qu'il ne dévie pas de l'objectif fixé. Lorsqu'il détecte une anomalie, il la signale à l'équipe qui prendra le problème en compte collégialement ;

— lorsque des artistes s'interrogent sur ce qu'ils doivent faire, il peut les orienter vers des tâches.

FIGURE II.4.9 – Composition fictive d'une équipe de co-création en temps réel.

Dans les sessions de reproduction, le coordinateur est principalement là pour vérifier que la scène ne dévie pas trop de l'objectif. Si cela se produit, il prévient le reste de l'équipe qui prend une décision collégiale pour remédier au problème. Son rôle est donc très éloigné de ceux du lead ou du superviseur qui sont des acteurs décisionnels importants à la chaîne de validation. Cependant, dans les sessions de création non guidées, il détient également le mot de la fin sur les décisions artistiques et se rapproche ainsi plus du rôle du réalisateur.

Plus globalement, l'analyse de performance montra que les sessions devenaient plus fastidieuses à coordonner au-delà de six personnes ; la limite suggérée par Scrum est donc applicable aux sessions. Enfin, pour que les équipes tirent pleinement parti du mécanisme de la transmission de savoir permise par la méthode (cf. Section II.2.3), il est important de mélanger différents niveaux d'expériences (junior, mid et senior).

Planification du travail

La préparation du travail dépendra beaucoup de sa nature. En préproduction, lorsqu'il n'y a pas de matériaux de référence (par exemple, dans le cas de la création de *background concepts* (cf. Sous-section II.2.3.4), la distribution du travail dans la scène ne peut être planifiée avec précision. En revanche, cela est possible pour la majorité des étapes de la phase de production. Cette préparation se tient une fois par projet (ou livrable, par exemple, un plan pour une équipe pluridisciplinaire), elle consiste à :

— segmenter le travail en tâches ;

— créer le *kanboard* associé.

Selon nos expériences (cf. Section I.1.2), la segmentation du travail par *asset* est à privilégier pour optimiser la superposition du travail. On note que cette technique est compatible avec des équipes pluridisciplinaires et spécialisées. Une équipe spécialisée en animation pourrait par exemple ainsi diviser le travail par personnage. Pour l'équipe pluridisciplinaire, cela dépend de sa composition. Si l'équipe est composée d'artistes généralistes, cette distribution est directement utilisable. Si ses membres sont tous spécialisés, une sous-segmentation par aspect pourrait être requise. Par exemple, pour un *asset* de voiture, on pourrait faire une sous-tâche modélisation, *shading* et *rigging*.

Une fois définies, les tâches peuvent être portées dans la colonne *To Do* d'un *kanboard* en vue de leur suivi. Une colonne *product backlog* (cf. paragraphe II.4.1.2) additionnelle est envisageable pour gagner en lisibilité lorsque la quantité de tâches est importante. L'idée étant de considérer chaque session comme un *sprint* en portant les tâches que l'on compte faire dans la journée du *product backlog* au *To Do*.

Temporalité du travail

La Figure II.4.10 représente l'évolution des tâches de la session en fonction du temps. Au début du projet, les artistes commencent par un *brief*. Comme dans un orchestre, c'est le moment d'accordage des instruments, on peut l'assimiler au *sprint planning*. C'est une étape fondamentale pour l'efficacité de la session qui suivra. Sans cela, les artistes risquent d'être déphasés et dirigés dans de mauvaises directions. Durant cet évènement, on pose les fondations, les règles de la session et les artistes choisissent collégialement une tâche initiale.

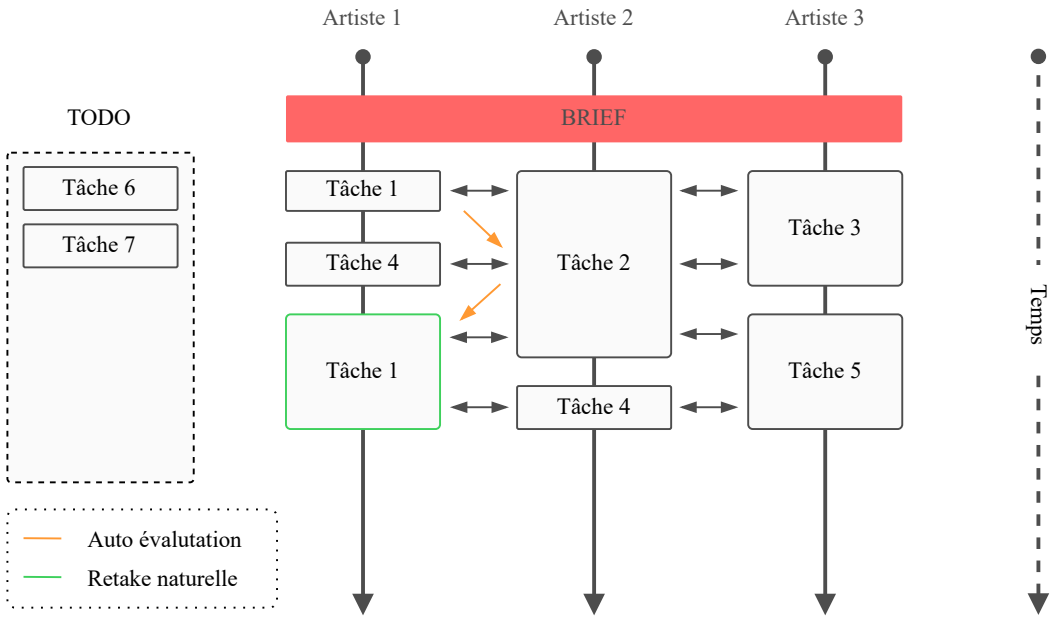


FIGURE II.4.10 – Temporalité des projets menés avec le *Multiuser*.

Tout au long de la session, les artistes sélectionnent d'autres tâches au fur et à mesure qu'ils en finissent. En travaillant tous dans la scène en cours de construction, les artistes constatent en temps réel le travail mené par le reste de l'équipe. Deux phénomènes vont émerger de cette situation :

— une évaluation des pairs ;

— une auto-évaluation.

L'évaluation des pairs provient de l'observation passive du travail en construction. Quand un artiste constate un problème dans la scène, il le communique au reste de l'équipe qui le corrige très rapidement. Nous l'avons notamment relevé dans la seconde session de *background concept* lorsque Clémence a déplacé un nuage projetant une ombre involontaire sur le travail de *layout* de Tanguy. Tanguy l'a alors directement signalé à Clémence qui a très rapidement ajusté la position de son nuage [204].

L'auto-évaluation (en orange sur la Figure II.4.10) va pousser les artistes à recalibrer leur propre travail en fonction de celui des autres. Cette réflexivité les conduit à faire d'eux-mêmes des changements sur leur travail, que nous qualifions de *retake*s naturelles (par exemple, la tâche 1 sur la Figure II.4.10). Ceci se passe presque automatiquement au cours de la session de travail collaboratif. Le caractère temps réel est intrinsèque à cet évènement. Si de nombreuses *retake*s naturelles ont été observées durant les sessions, nous ne sommes pas parvenus à les quantifier. Il serait intéressant de mener une étude quantitative et comparative pour connaître leurs impacts sur la qualité globale du travail. Ce phénomène pourrait-il faire baisser le nombre de *retake* en production ?

Cette section a mis en lumière la collaboration en temps réel en tant que méthode de création agile. En ajoutant un aspect social au cœur de la création, nous avons réenvisagé la fabrication d'images animées en remplaçant l'individu par le groupe. Ce changement de paradoxe a induit des changements profonds de *workflow* : une coor-

204. Observable à partir de la quarantième seconde du *timelapse* `3-tangra_concept_2.mp4` accessible à l'adresse suivante : `https://doi.org/10.5281/zenodo.5549504`

dination et une organisation plus précise. Actuellement, cette méthode a été explorée à l'échelle d'une équipe composée de deux à dix artistes (cf. Annexe E). Il serait intéressant d'appliquer ce principe à un groupe d'équipes pour étudier si ce concept serait généralisable à une plus grande échelle.

On pourrait envisager d'élargir l'exercice pour une séquence par exemple. Si on considère que le décor est le même sur toute une séquence, il faudrait alors créer :

— une scène d'*asset* ;

— une scène de décor ;

— une scène pour la séquence (avec une caméra par plan) ;

— une scène de montage.

La scène d'*asset* serait commune à toutes les équipes. Ces *assets* seraient instanciés sous forme de *placeholders* dans la scène du décor. Un *placeholder* du décor serait lui-même instancié dans la scène de séquence. Il serait possible d'assigner une équipe d'animation par plan. Cependant, pour tirer pleinement profit de cette configuration de scènes, il serait recommandé de faire l'animation à la séquence et ainsi d'assigner un artiste par personnage par exemple. Enfin, le réalisateur pourrait faire le montage des différentes séquence dans la scène de montage en exploitant le Video Sequence Editor (VSE) de *Blender*.

Conclusion

Ce chapitre a remis en question le mode de fabrication traditionnel d'images animées en envisageant de remplacer l'individu par le groupe à travers l'utilisation des méthodes agiles. Une brève introduction des concepts agiles a posé les bases importantes pour la compréhension du *framework* Scrum qui a ensuite été décrit.

Son application directe dans notre discipline implique d'organiser la gestion des tâches à l'échelle des équipes et de restructurer le déroulement de la fabrication en cycles courts de quelques semaines rythmés par des évènements de planification et de rétrospective.

Il fut observé que la collaboration en temps réel s'intégrait parfaitement dans cette démarche. En rapprochant l'ensemble des membres de l'équipe autour de l'œuvre, elle augmente la conscience de groupe et favorise la transmission de connaissances entre artistes de différents niveaux. Un phénomène nouveau d'auto-évaluation apparaît alors et conduit les artistes à ajuster leur travail de façon autonome en fonction de la scène en cours de création.

Conclusion de la partie II

Cette seconde partie a exploré le potentiel de la collaboration en temps réel pour le cinéma d'animation.

Cette recherche a conduit au développement d'un *framework*, répondant aux problématiques de création d'une expérience multi-utilisateur dans un programme transparent à la collaboration. Elle a également pris corps à travers le développement de l'*add-on Multiuser* pour *Blender*. Ces deux outils open source ont étroitement évolué avec les besoins découverts lors de leurs mises en situation dans les expérimentations.

Les sessions expérimentales à l'université ont mis en valeur l'apport de notre outil pour l'enseignement de la création d'images de synthèse. La collaboration en temps réel crée un espace virtuel partagé entre le professeur et les étudiants, favorisant le suivi, les interactions et la transmission de savoir.

Les sessions à Cube Creative ont permis d'évaluer notre approche dans le contexte d'une production industrielle d'images de synthèse. Les sessions de recréation ont exploré la collaboration interdépartement. L'analyse comparative de ces sessions avec leur équivalent linéaire mit en évidence un léger gain de temps intéressant, mais également une efficacité qui gagnerait à être améliorée en optimisant l'organisation du travail. Le cadre plus abstrait des sessions de *background concept* a mis en valeur l'impact bénéfique d'une période de *brief* observée en début de session sur le résultat final.

Nous avons ensuite décrit l'impact de la Covid-19 avec l'émergence d'une communauté internationale impliquée dans la recherche et le développement du *Multiuser*. Dès lors, des sessions publiques hebdomadaires ont enrichi la recherche de méthodes de segmentations et d'organisations du travail, adaptées au contexte non linéaire de la création collaborative.

Nous avons ensuite exposé, d'une part, les cas d'usages concrets envisagés à court terme dans le pipeline de Cube Creative, d'autre part, les axes de développement à long terme pour approfondir la recherche et le développement de l'outil.

Enfin, nos expérimentations nous ont conduits à envisager l'appareil de production sous un autre angle. En supportant le travail de groupe, notre outil s'insère dans une démarche d'« agilification » de la création replaçant le facteur humain et social au cœur de l'œuvre.

Conclusion Générale

Cette thèse menée en entreprise au cours d'un dispositif CIFRE a permis d'explorer l'utilisation du temps réel pour la collaboration dans la production d'images animées. En développant un outil de co-création et en étudiant la manière de l'utiliser, nous avons pu dégager des conclusions sur deux axes. Le premier axe a étudié les processus de fabrication actuels de film d'animation et les changements qu'y apportent les moteurs de jeu. Le second axe, le plus fondamental, est celui de la recherche et du développement d'une méthode de collaboration en temps réel.

Évolution des processus de production de films d'animations

Une première étape de cette thèse a été d'exposer la convergence entre deux univers : le cinéma d'animation et le jeu vidéo. Celle-ci dévoile le contexte de notre recherche, à savoir, les changements apportés par l'arrivée de méthodes de création temps réel dans les productions précalculées.

Production industrielle d'images animées

Aujourd'hui, les films d'animation résultent d'une chaîne de production complexe. Pour saisir son fonctionnement, nous avons décrit ses composantes humaines et logicielles. La présentation des différents acteurs et leurs enjeux pour chacune des phases de fabrication mirent en valeur la linéarité et l'interdépendance des étapes de fabrication. La relation entre ces différents maillons humains dans cette organisation du travail est cloisonnée. Cela est dû à la forte spécialisation des métiers et leur départementalisation qui canalise la collaboration verticalement. Les artistes travaillent ainsi sur des tâches individuelles et sur des fichiers individuels.

Mais ce *workflow* est étroitement lié au pipeline sous-jacent qui automatise les traitements et fait transiter des données tout au long de la production. En transmettant uniquement le résultat de chaque étape vers la suivante, il empêche les artistes de connaître le cheminement de la construction de l'objet sur lequel ils travaillent. Par conséquent, ils demeurent aveugles aux problématiques associées. or, ce silo de communication peut être une source d'erreur : de par sa nature rigide, la chaîne de fabrication laisse très peu de place aux imprévus. Pour envisager de changer notre *workflow* de fabrication, il faut donc modifier le *dataflow* qui le supporte.

Vers une fusion des méthodes de fabrications

Pour contextualiser et comprendre le rapprochement de l'univers du temps réel et celui du précalculé, il a fallu exposer les spécificités des moteur de jeu. La fusion

des deux univers se matérialise à travers la *virtual production*, un regroupement de techniques de visualisation qui exploitent le temps réel comme hub de collaboration pour rapprocher les différents pôles de la production. Si ces technologies sont largement adoptées dans le milieu des VFX en prise de vue réelle, nous avons vu à travers une analyse de cas que certains verrous technologiques freinaient leur intégration dans le cinéma d'animation. Nous avons donc questionné la possibilité d'exploiter le temps réel au cœur des logiciels de création déjà adoptés par l'industrie en s'appuyant sur la création d'un outil de caméra virtuelle pour *Blender*. Cette expérience a validé la capacité du DCC à interpréter et rendre des flux de données en temps réel. Cependant, cela amena des interrogations quant aux méthodologies à adopter pour collaborer dans la même temporalité. Notre participation au projet *Corps Infini* nous ouvrit les yeux sur une piste intéressante pour y répondre : les méthodes agiles.

Finalement, cette étude nous a alors permis d'arrêter le choix du logiciel dans lequel nous allions mener notre exploration de co-création en temps réel et nous donna des pistes organisationnelles pour gérer l'aspect humain au sein de la collaboration.

Enjeux de la co-création en temps réel pour l'image de synthèse

Au début de cette thèse, la collaboration en temps réel était embryonnaire dans le cinéma d'animation. Nous avons donc analysé son utilisation par deux autres secteurs d'activité afin d'en déterminer ses enjeux et de potentielles pistes d'applications. En génie civil, elle est employée pour partager une vision commune du projet entre les acteurs de la construction. En informatique, elle facilite la transmission de connaissances, la réflexion et l'évaluation au sein des équipes de développeurs. Nous avons observé que deux architectures logicielles étaient utilisées. Dans le premier cas, c'est un format de transmission garantissant l'interopérabilité entre les logiciels qui est à la base de la collaboration, dans l'autre, c'est une extension qui réplique le projet au sein de la même application chez chaque client.

En explicitant la signification de ces deux architectures dans le domaine du cinéma d'animation sous les approches cross-DCC et mono-DCC, nous en avons conclu que la seconde était plus adaptée à notre démarche. Nous avons ensuite positionné notre recherche dans l'historique des solutions de co-création en temps réel existantes. Cette chronologie mit en évidence l'intérêt de l'industrie pour ces méthodes émergentes. De plus, elle permit de constater qu'aucune solution existante ne supportait la collaboration temps-réel au sein de plusieurs instances d'un même outil de création 3D utilisé en production de manière non destructive et sans conflit d'édition.

Recherche et développement d'un outil de co-création en temps réel pour l'animation

Le cœur de nos travaux se situe dans le développement d'un outil portant la collaboration en temps réel au cœur de la création et de son expérimentation pour la recherche de nouvelles méthodes de fabrication.

Multiuser : la collaboration en temps réel au cœur de la création

Nous avons ainsi développé un *framework* mono-DCC qui fournit les fondations nécessaires à la création d'une expérience multi-utilisateur au sein de logiciels de création 3D non conscients de la collaboration.

Ce dernier repose sur un graphe de réplication acyclique dirigé pour stocker et structurer les données de création afin de les répliquer de manière distribuée à travers le réseau local de l'entreprise ou internet. Un protocole de définition des données répliquées apporte un *dataflow* non destructif et adaptable aux spécificités des différents DCC. Dès lors qu'un élément de la scène 3D est supporté par une implémentation, le *framework* va le répliquer et l'appliquer à travers les clients connectés.

Pour éviter les conflits d'accès, chaque nœud est soumis une gestion de droits de modification stricte et centralisée. Cette approche nodale offre une granularité de mise à jour répondant à la complexité des données propres aux scènes 3D utilisées dans le cinéma d'animation. Le rythme de ces mises à jour est géré grâce aux fonctions conscientes de collaborations qui permettent concilier le *workflow* collaboratif avec les contraintes de ces logiciels et les besoins des artistes.

Enfin, une interface consciente de collaborations supporte la création d'outils destinés à maintenir la présence des utilisateurs dans la session de collaboration.

Le *framework* fut intégré dès le début dans *Blender* à travers l'*add-on Multiuser*. Il permet à plusieurs utilisateurs de se connecter, depuis différents postes de travail, au sein d'une même scène 3D et d'en modifier presque tous les aspects en temps réel. La conception de l'*add-on* mis en lumière les difficultés liées à l'utilisation de programmes conçus pour être mono-utilisateur. Ici, la philosophie *open source* de Blender fut cruciale pour résoudre ces problèmes et parvenir à la création d'un prototype en quelques mois afin de démarrer les expérimentations.

Expériences de co-création en temps réel

La diversité des cadres d'application du dispositif CIFRE nous a permis d'étudier la collaboration en temps réel dans deux contextes : l'enseignement et l'industrie.

Ainsi, le *Multiuser* fut mis à contribution pour donner des cours de modélisation, de *shading* et de *lighting* sur *Blender*. L'analyse de ces cours montra que partager le même espace virtuel améliorait les interactions et la communication au sein de la classe, d'une part en aidant le professeur à détecter les élèves en difficulté afin qu'il vienne les aider, d'autre part en favorisant l'entraide entre étudiants de niveaux différents.

Mais ces expériences collaboratives d'apprentissage ont seulement touché la surface des applications possibles pour l'enseignement. Il serait intéressant d'approfondir cet axe de recherche pour étudier les capacités de l'outil pour l'enseignement à distance et l'apprentissage de la coordination dans le travail.

Dès 2019, les sessions industrielles prirent place avec des artistes et des superviseurs de Cube Creative. Elles interrogèrent d'une part les capacités et la qualité de notre méthode, et, d'autre part sa place dans le processus de fabrication d'images animées. Une analyse des sessions de recréations montra que le principal facteur limitant l'efficacité du travail parallèle, était d'ordre organisationnel. Cela entrait en écho avec l'analyse qualitative des sessions de *background concept* qui démontrèrent qu'un temps de *brief*

en début de projet améliorait la cohésion des artistes. Avec ces deux exercices, nous avons montré que la collaboration en temps réel pouvait aussi bien s'appliquer à des équipes spécialisées que pluridisciplinaires.

Une exploration partielle des capacités de notre méthode dans l'étape d'animation permit de valider qu'elle ne représentait aucun verrou technique en vue d'un approfondissement dans de futures recherches.

Par ailleurs, il fut observé que la collaboration temps-réel donnait aux artistes la pleine mesure de l'impact de leur travail sur celui des autres, augmentant ainsi considérablement l'anticipation des erreurs. Enfin, l'ajout de l'aspect social dans la création a poussé les artistes à communiquer sur leur pratique, créant une transmission naturelle du savoir entre les différents niveaux d'expérience.

L'ouverture du projet en *open source* et la crise de la Covid-19 apportèrent une nouvelle dimension à nos recherches. Une communauté internationale hybridant particuliers et professionnels du secteur de l'animation émergea autour du *Multiuser*. L'*add-on* avait le potentiel de rapprocher les artistes autour de la création malgré les règles de distanciation sociale. De fil en aiguille, nous avons adapté l'outil aux contraintes liées au réseau internet ce qui conduisit à l'émergence de sessions publiques : des moments de création collaborative où tout un chacun pouvait venir apporter sa pierre à l'édifice.

Cette communauté s'est impliquée dans de nombreux aspects de notre recherche, que ce soit sur le plan du développement de l'outil à travers des correctifs et rapports de d'anomalies ou sur le plan expérimental à travers la centaine de sessions publiques qui prirent place.

Les sessions publiques représentèrent un défi de coordination important, car elles se déroulaient à distance et majoritairement sans échanges vocaux. Cela nous conduisit à concevoir et à expérimenter des stratégies d'organisation du travail pour faciliter sa gestion. Nous avons ainsi montré que la répartition du travail par *asset* était une approche efficace pour diviser et distribuer les tâches au sein d'une équipe en collaboration. L'analyse des sessions publiques de création libre mirent en avant l'importance du rôle de coordinateur pour parvenir à un consensus artistique. Mais la communauté grandissant, certains membres ne pouvaient pas participer à cause d'un trop grand décalage horaire. Cela nous conduisit à rendre continues les sessions publiques. La persistance de l'espace de création dans le temps motiva la recherche d'une méthode de suivi donnant aux artistes la capacité de connaître l'état d'avancement de la scène et de choisir des tâches en conséquence. Nous avons ainsi montré que les *kanboards* se prêtaient très bien à cette problématique et représentaient un outil de suivi adapté au travail collaboratif en temps réel.

Les contraintes propres aux sessions distancielles sur internet se sont en fin de compte avérées très stimulantes dans l'étude de la question organisationnelle du travail collaboratif.

Utiliser la collaboration en temps réel au cœur d'une production traditionnelle

Nous nous sommes questionnés sur la suite envisagée pour ces travaux avec, d'une part leur application à court terme dans le pipeline de fabrication de Cube Creative et d'autre part, les développements à long terme pour élargir le champ de la méthode.

Applications du *Multiuser* dans un pipeline de série animée

Dans la continuité de cette thèse CIFRE, nous envisageons d'utiliser les outils développés à différentes étapes de la fabrication de séries d'animation. Nous pensons ainsi utiliser le *Multiuser* durant les *briefs layout* et *storyboard* pour faciliter leur déroulement à distance et générer des données qui guideront les artistes *layout*.

Les superviseurs enseignent régulièrement *Blender* aux nouveaux artistes. Or, nos sessions expérimentales ont montré l'apport de l'outil pour l'apprentissage. Il est donc naturellement apparu que nos travaux faciliteraient la formation des nouveaux arrivants. Enfin, nous envisageons également d'appliquer la collaboration en temps réel au cœur de deux étapes de création : pour faire du *look development* et du découpage de séquences avec le réalisateur. Cependant, notons que ces deux derniers cas n'ont pas encore été testés.

Développements futurs

Les sessions expérimentales ont levé de nombreux axes de développements futurs pour le *Multiuser*.

Actuellement, l'étape d'animation n'est que partiellement supportée, mais elle a beaucoup de potentialités en termes de collaboration. Achever le support des *rigs* complexes permettrait d'explorer la co-animation de personnages par exemple.

En nous focalisant sur le temps réel, nous avons développé un outil permettant de collaborer uniquement dans le présent (sur la version actuelle d'une scène). En ce sens, nous nous sommes éloignés de la logique incrémentale du travail que l'on retrouve dans les *workflows* traditionnels, utilisés pour la validation notamment. Pour supporter le suivi et la validation du travail nous avons envisagé l'ajout d'un système de gestion de versions s'inspirant de Git en stockant des images du *repository* à un instant t.

Actuellement, le système de droit du *Multiuser* est l'un des principaux facteurs limitant la superposition du travail dans les sessions collaboratives. En verrouillant toutes les dépendances d'un *datablock*, il empêche la parallélisation du travail sur les aspects qui le composent. En ajoutant un mécanisme de rôle, il deviendrait possible de verrouiller un aspect précis associé à un rôle, rendant ainsi accessible le travail parallèle au niveau d'un objet.

Vers une création agile ?

La production d'un film s'organise autour d'une répartition individuelle du travail. En mettant en avant l'équipe, la co-création en temps réel change profondément ce paradigme au cœur de la fabrication. Pour en tirer pleinement parti, il faut donc envisager de réorganiser l'appareil de production autour du groupe artistique.

Nous nous sommes donc intéressés aux méthodes agiles qui embrassent ce concept et plus particulièrement au *framework* Scrum que nous appliquons au sein de l'équipe de R&D de Cube Creative. Dans cette méthode, le groupe de travail est responsable et autogéré. Aussi, la temporalité de production s'organise en cycles courts qui impliquent des évènements réguliers afin de garantir la transparence du travail et l'amélioration du processus.

L'étude de trois cas concrets de l'industrie du VFX et de l'animation a montré comment cette méthode pouvait s'appliquer à l'échelle d'un studio dans le domaine. D'un point de vue structurel, les artistes sont regroupés en équipes spécialisées ou pluridisciplinaires avec de préférence plusieurs niveaux d'expérience au sein d'une même

équipe. C'est elle qui décide de la répartition et de l'estimation du travail. Typiquement, à l'issue d'une *review*, la *retake* n'est plus adressée à l'artiste qui en est à l'origine, mais à l'équipe entière. Les membres de l'équipe participent tous à sa résolution et, en apportant chacun leur vision, le processus d'apprentissage s'en retrouve ainsi amélioré. La conséquence directe observée est une baisse du nombre de *retake*s. Cette structure rassemble les artistes autour d'une suite de petits objectifs communs à l'inverse d'une production classique qui considère le projet dans sa globalité. Grace à cette méthode, la production mue d'une longue chaîne d'étapes interdépendantes à une suite de cycles courts, indépendants et réactifs aux changements. En perdant une estimation approximative à long terme, on gagne une estimation plus précise à moyen terme ainsi qu'une réactivité accrue face aux erreurs.

En rapprochant les artistes autour de l'œuvre, la collaboration en temps réel favorise la transmission de connaissances et crée une auto-évaluation qui conduit les artistes à ajuster leur travail en fonction de la scène en cours de création. Par conséquent, ce phénomène pourrait jouer un rôle important au cœur de cette démarche d'« agilification » de la fabrication des images de synthèse.

Ouverture

Cette thèse ouvre plusieurs perspectives de recherche.

Ajouter de nouvelles dimensions à la collaboration ?

Impliquer des médiums immersifs tels que la réalité virtuelle dans la co-création pourrait ajouter une nouvelle dimension amenant les artistes à collaborer sur plusieurs échelles. Pour l'enseignement, le professeur pourrait guider avec plus de précision les élèves durant les travaux pratiques. Dans la fabrication d'image animée, on peut imaginer qu'un artiste produise le *layout* de *placeholder* en RV tandis que les autres développent les différents *assets* parallèlement.

Vers l'émergence d'un métavers[205] *créatif ?*

Notre recherche ouvre des questionnements vis-à-vis de l'évolution des espaces créatifs, et notamment sur le développement d'interactions sociales dans la création.

Aujourd'hui, le *Multiuser* permet à plusieurs artistes de se retrouver dans un même espace virtuel en trois dimensions et d'y développer un univers visuel. Une session restant accessible à n'importe quel moment devient un lieu d'échange social. Nous avons ainsi observé des artistes qui utilisaient cet espace pour se retrouver et discuter. Peut-être est-ce un premier par vers l'émergence d'un métavers dédié à la création. Cependant, les interactions sociales y sont encore très limitées. En représentant les utilisateurs avec un avatar personnalisable doué d'expression, les artistes pourraient s'approprier cet univers et y avoir des interactions plus profondes.

Vers la création de films participatifs en ligne ?

Si les sessions publiques ont montré que la collaboration en temps réel pouvait fédérer une communauté autour de la création d'une œuvre ouverte, est-ce applicable à l'échelle d'un film ? Pour explorer cette question, il faudrait étudier comment orchestrer plusieurs espaces de création. Il serait intéressant d'intégrer au *framework* des outils participatifs existants pour permettre le suivi et la gestion d'un tel projet.

205. Un métavers est univers virtuel en deux ou en trois dimensions créé artificiellement par un logiciel et pouvant héberger une communauté d'utilisateurs qui y interagissent sous forme d'avatars.

Enfin vient l'épineuse question de la licence du projet. Comment protéger les droits de cette œuvre ? Dans sa thèse Damien Picard [20, p. 92] propose une piste intéressante avec l'utilisation de licences libres de diffusion comme celles de la famille *Creative Commons*.

Mais bien avant les problématiques de distribution, en remontant à l'origine d'un projet de film, les méthodes de fabrication sont formatées par le modèle de financement actuel. Aussi, pour permettre aux studios d'innover, de faire évoluer la fabrication des films, il faudrait interroger des alternatives au modèle économique productiviste actuel, et promouvoir des modes de financement plus collaboratifs.

Liste des acronymes

Application Programming Interface (API)

Une *Application Programming Interface* est un ensemble particulier de règles et de spécifications qu'un programme logiciel peut suivre pour accéder aux services et aux ressources fournis par un autre programme logiciel particulier qui met en œuvre cette API et pour les utiliser. 45–47, 84, 88, 98, 103, 107–110, 116, 196, 197

Art et Technologie de l'Image (ATI)

Formation Art et Technologie de l'Image à l'université Paris 8. `https://www.ati-paris8.fr/` 15, 16, 120

Base De Données (BDD)

Une Base De Données est un ensemble structuré et organisé de données qui représente un système d'informations sélectionnées de telle sorte qu'elles puissent être consultées par des utilisateurs ou par des programmes. Définition du Larousse 45, 46

Break Down List (BDL)

La Breakdown List est une liste des assets présents par plan par épisode. 33, 225, 226

BIM Collaboration Format (BFC)

Le *BIM Collaboration Format* est un format de note pour IFC. Les fichiers BCF, permettent d'échanger des commentaires, éventuellement accompagnés d'une vue, ou d'une sélection d'éléments de la maquette IFC. Ces fichiers contiennent aussi toutes les informations de traçabilité : auteur, date, version. 79

***Building Information Modeling* (BIM)**

Building information modeling est un processus soutenu par divers outils, technologies et contrats impliquant la génération et la gestion de représentations numériques des caractéristiques physiques et fonctionnelles des lieux. Les *Building information models* sont des fichiers informatiques qui peuvent être extraits, échangés ou mis en réseau pour soutenir la prise de décision concernant un bien construit. 78–81, 86

***Blender PYthon* (BPY)**

107, 108, 110

Bâtiment et Travaux Publics (BTP)

Le secteur économique du bâtiment et des travaux publics, ou BTP, regroupe toutes les activités de conception et de construction des bâtiments publics et privés, industriels ou non, et des infrastructures telles que les routes ou les canalisations. Définition de Wikipedia 78–80, 83, 84, 192, 212

Could Be Better (CBB)

Could Be Better est un statut désignant un travail validé mais qui pourrait être amélioré. 35

***Clustered Hashmap Protocol* (CHP)**

100

Convention Industrielle de Formation par la Recherche (CIFRE)

Une Convention Industrielle de Formation par la Recherche permet à un doctorant de réaliser sa thèse en entreprise en menant un programme de recherche et développement en liaison avec une équipe de recherche universitaire. 16, 150, 183, 185, 187

Compute Process Unit (CPU)

60

Cascading Style Sheets (CSS)

Le Cascading Style Sheets est un langage informatique utilisé pour la mise en forme de pages HTML 69

Digital Content Creation software (DCC)

17, 38–43, 47, 50, 51, 60, 62, 63, 67, 68, 71, 75, 82, 84–87, 95, 96, 98, 99, 102, 103, 107–109, 114, 116, 134, 157, 184, 185, 193, 203, 204, 215

***End-user license agreement* (EULA)**

52

Federal Bureau of Investigation (FBI)

164

Filmbox format (FBX)

52, 62, 64

Special Effect (FX)

25, 27, 28, 44, 64, 127, 205

Graphical Processing Unit (GPU)

60

HyperText Markup Language (HTML)

Le HTML est le langage de balisage conçu pour représenter les pages web. Définition de Wikipédia. 69, 70, 192

Industry Foundation Classes (IFC)

Le format Industry Foundation Classes est un format de fichier utilisé dans le BTP pour échanger et partager des informations entre logiciels. Il permet de décrire des objets (murs, poteaux, etc.), leurs caractéristiques et leurs relations. Les IFC ont pour but d'assurer l'interopérabilité entre les logiciels de conception assistée par ordinateur et les logiciels d'ingénierie. Ce format est décrit dans la norme ISO 16739. 79, 80

Image Numérique et Réalité Virtuelle (INREV)

`https://inrev.univ-paris8.fr` 72, 120

Information technology Technician (IT)

Personne responsable du parc machine, de la gestion du réseau et du stockage. 38, 47

Local Area Network (LAN)

135

***Large File Storage* (LFS)**

Git Large File Storage est une extension de *Git*. Elle lui ajoute la capacité de gérer correctement les fichiers volumineux en stockant des références au fichier dans le repository, mais pas le fichier lui-même. Pour contourner l'architecture de *Git*, *Git* LFS crée un fichier pointeur qui agit comme une référence au fichier réel (qui est stocké ailleurs). Définition de GitHub. 54

Open Shading Language (OSL)

Open Shading Laguage est un langage de shader open source originellement développé par Sony Pictures Imageworks. `https://github.com/AcademySoftwareFoundation/OpenShadingLanguage` 42, 44

Open Timeline IO (OTIO)

Open Timeline IO est un format d'échange de données de montage open source développé par Pixar. `https://github.com/PixarAnimationStudios/OpenTimelineIO` 42

Replicated Data definition Protocol (RDP)

Le *Replicated Data Protocol* est une interface de notre *framework* (cf. Sous-section II.1.2.2) permettant au développeur de définir avec précision les données de création du DCC répliquée à travers la session de collaboration. 98, 102, 103, 116, 157, 212

Retake (RTK)

35

Réalité Virtuelle (RV)

71–73, 137, 155, 188, 211

Recherche et Dévelopement (R&D)

Dans l'industrie du cinéma d'animation, la recherche et développement (R&D) est une pratique consistant à mener des recherches fondamentales ou appliquées sur un sujet donné, et à développer des procédés, algorithmes et programmes. Dans un studio d'animation, l'équipe R&D fait de la veille technologique et développe des solutions adaptées aux problématiques de production. 16, 47, 137, 164–167, 169, 174, 187

Solid-state drive (SSD)

Un *solid-state drive* est un matériel informatique permettant le stockage de données sur de la mémoire flash. 64

Transmission Control Protocol (TCP)

Le protocole de contrôle de transmissions ou *transmission control protocol* est un protocole de transport de données. 70, 102

***Time To Leave* (TTL)**

102

Universal Scene Description (USD)

Universal Scene Description est un format d'échange open source de scène 3D. `https://github.com/PixarAnimationStudios/USD` 42–44, 64, 79, 87

Version Control System (VCS)

Un système de gestion de versions ou *Version Control System* est une catégorie de logiciels chargés de gérer les modifications apportées aux programmes informatiques, aux documents ou à d'autres collections d'informations. 53, 54, 64, 67

Visual Effects (VFX)

60, 155, 170, 173, 175, 184, 187

Video Sequence Editor (VSE)

178

Wait For Approval (WFA)

35, 46

Work In Progress (WIP)

34, 35

GL Transmission Format (GLTF)

glTF est un format de fichier open source pour les scènes et les modèles tridimensionnels. Il a été créé par Khronos Group. `https://github.com/KhronosGroup/glTF` 43, 70

Glossaire

Docker

Docker est une plateforme open source permettant de lancer certaines applications dans des conteneurs logiciels. Ces conteneurs comprennent l'application et toutes ses dépendances. Ils peuvent ainsi être exécutés sur n'importe quel serveur. Par exemple, l'image *Docker* du *Multiuser* comprend le *framework* accompagné de la version appropriée de Python. Grâce à ce mécanisme, un utilisateur n'aura pas besoin de réinstaller le *framework* et Python sur chaque machine dans laquelle il voudra déployer le serveur dédiée du *Multiuser*. Plus d'informations sur *Docker* : `https://www.docker.com` 135, 158

Open Pipe

Open Pipe est un groupe fondé par Supamonks, Cube Creative, Ubisoft Animation Studio et Karlabs dans l'optique de mettre en commun des réflexions et des outils innovants de pipeline de production. 49, 68, 73

add-on

Un add-on ou un plug-in est un programme ou un script tiers qui est ajouté à un programme pour lui donner des fonctionnalités et des capacités supplémentaires. 38, 44, 51, 69, 70, 82, 87, 88, 93, 95, 103, 107, 108, 117, 124, 127, 136, 137, 148, 155, 158, 181, 185, 186

alembic

Alembic est un format d'échange de données graphiques open source originellement développé par Sony Pictures Imageworks et Industrial Light & Magic. `http://www.alembic.io/` 42, 43, 64

asset manager

Un *asset manager* est une couche d'abstraction logicielle entre le système de fichiers et leurs utilisateurs qui sert à structurer et à simplifier la gestion des fichiers tout au long du *pipeline*. 36, 44, 46, 47, 52, 53, 211

asset

Les *assets* sont utilisés pour le suivi de ce qui doit être réalisé pour le projet. Nous ne parlons pas des fichiers numériques, mais du concept abstrait de ce qui doit être accompli, comme un personnage, un véhicule, un objet, un décor, etc. Définition par LePipeline.org. 23, 26, 27, 30–32, 34, 36, 38, 40, 41, 43, 44, 47, 50–53, 55, 60, 62, 64, 75, 88, 96, 104, 107, 113, 118, 122, 127–130, 133, 138–148, 175, 176, 178, 186, 188, 195, 198–200, 202, 203, 211, 213, 225, 227

Un *game asset* est un *asset* utilisé dans un moteur de jeu. Ce terme désigne aussi bien des entités logiques (exemple : un script) que graphiques (exemple : un modèle 3D). 51, 54, 64, 65

background concept

Le *background concept* ou concept de décor désigne l'étape de prototypage de nouveaux décors. 126, 128–132, 134, 138, 155, 176, 177, 181, 185, 212, 215

binding

En programmation, un *binding* est un point d'entrée permettant à une API d'accéder à des fonctionnalités ou structures internes du logiciel. 110

blocking

Le *blocking*, première étape de création d'une animation, consiste à créer les positions les plus importantes du mouvement (aussi appelées poses clefs). 65, 133, 134, 200

bounding box

La *bounding box* est la boîte minimale encadrant un ensemble de points en N dimensions. 111

brief

Le *brief* est une réunion d'information pour définir les objectifs, les méthodes, les moyens, etc. 34, 132, 134, 152, 153, 173, 176, 181, 185, 187, 213

bugtracker

Un *bugtracker* est un outil de suivi des dysfonctionnements (ou erreurs) associés au fonctionnement d'un logiciel. Les problèmes y sont généralement représentés sous forme de tickets qui canalisent le suivi de leur résolution dans un fil de commentaires. 137

casting

Le *casting* décrit tout ce qui va jouer dans un plan (personnage, décor, etc.). Il résulte généralement du dépouillement du scénario, d'un *storyboard* ou d'une séquence. C'est à partir du *casting* que des outils de pipeline vont pouvoir initialiser ou mettre à jour des scènes pour que les graphistes puissent se focaliser sur leur travail sans se préoccuper d'assembler leur scène. Fondamentalement, c'est une simple liste de tous ces éléments, mais on y adjoindra assez rapidement des informations supplémentaires pour spécifier les versions de rig utilisées, le nombre d'éléments castés, le nom des fichiers correspondants, etc. Définition par LePipeline.org. 33, 34

cloud

Le *cloud* est un modèle informatique où les traitements traditionnellement effectués sur des serveurs locaux ou sur des postes clients sont externalisés sur des serveurs distants. Définition par Antidote. 79, 135–137

codebase

La *codebase* désigne l'ensemble du code source d'un logiciel. 81–83

compositing

L'étape de *compositing* consiste à assembler différentes couches de rendu pour créer l'image finale. 28, 38, 39, 65, 86, 127

datablock

Un bloc de données ou *datablock* désigne toute unité d'information de la scène 3D stockée par le logiciel de création 3D (par exemple : un *shader*, un *mesh*, un système de particule, etc.). 96–100, 102–104, 108–110, 112–114, 119, 127, 137, 148, 157, 158, 187

dataflow

Le *dataflow* (« flux de données », en français) est l'organisation des données dans une structure de fichiers. Contrairement au *workflow* qui décrit des étapes de travail, le *dataflow* décrit les dépendances de fichiers induites par le pipeline. Définition par LePipeline.org. 37, 40, 41, 60, 64, 65, 67, 75, 79, 85, 86, 89, 95, 98, 116, 145, 183, 185

driver

Dans *Blender*, les *drivers* sont un moyen de contrôler les valeurs des propriétés au moyen d'une fonction, ou d'une expression mathématique. 41

final pixel

Final pixel désigne la possibilité d'obtenir la qualité finale de l'image en direct, dans la caméra, sans avoir à recourir à la postproduction. 59, 66

framerate

Le *framerate* ou taux de rafraîchissement désigne la fréquence à laquelle des images consécutives sont affichées ou capturées. 64

framework

Un *framework* est un ensemble de composants logiciels structurés facilitant le développement de programmes en fournissant un niveau d'abstraction plus élevé. 70, 87, 93, 95, 98–100, 102–105, 108, 110, 113, 114, 116, 118, 127, 135, 156–158, 181, 185, 188, 195, 212, 215

frame

Une frame est une image. 65, 105, 108, 111

game state

En jeu vidéo, le *game state* est un composant logiciel responsable du stockage de l'état global du jeu. Dans un jeu multijoueur par exemple, il peut être utilisé pour stocker les joueurs connectés. 99, 100

gameplay

Le *gameplay* ou « jouabilité » désigne l'ensemble des possibilités d'action offertes au joueur par un jeu vidéo. Définition par la Commission d'enrichissement de la langue française. 147

grooming

Le *grooming* (ou toilettage en français) est la tâche qui vise à placer et coiffer des poils et cheveux d'un personnage ou d'une créature. Le *grooming* peut aussi s'appliquer sur d'autres objets comme les arbres (feuilles) ou des sols (fleurs, herbe). Le *grooming* est aussi appelé *hair* dans certains studios. Définition par LePipeline.org 38

handler

Dans l'API Python de *Blender*, un *handler* est une interface qui permet d'enregistrer des fonctions pour les appeler automatiquement à un évènement donné. 108, 113, 114

heartbeat

En informatique, un *heartbeat* est un mécanisme logiciel qui surveille la disponibilité d'un ou plusieurs programmes sur le réseau en leur envoyant des requêtes à intervalle régulier. 102

kanboard

Un tableau Kanban ou *kanboard* est un outil inventé par Toyota dans les années 1960 pour mettre en œuvre la méthode agile Kanban afin de gérer le travail. Il décrit visuellement le travail à différentes étapes d'un processus en utilisant des cartes pour représenter les éléments du travail et des colonnes pour représenter chaque étape du processus. Les cartes sont déplacées de gauche à droite pour montrer la progression et aider à coordonner les équipes qui effectuent le travail. `https://leanmanufacturingtools.org/kanban/` 143–145, 165, 172, 173, 176, 186, 213

layout

Le *layout* consiste à positionner les *assets* dans les scènes 3D, à cadrer les plans et établir les mouvements de caméra en suivant le *storyboard* et l'animatique. 27, 31, 33, 34, 38, 45, 62, 65, 66, 75, 89, 126, 127, 130, 140, 147, 148, 152, 153, 168, 171, 175, 177, 211–213

lighting

Le *lighting* est la tâche de mise en lumière de la scène. 28, 48, 51, 65, 120, 124, 125, 127, 138, 140, 156, 158, 185, 212

lightprobe

Les *lightprobe* sont des positions dans la scène où la lumière est mesurée pendant le processus de *bake*. C'est une technique utilisée par les moteurs de rendu temps réel (principalement dans le jeu vidéo) pour précalculer le *lighing* d'une scène. 110

look development

Le *look development*, ou *lookdev*, est la tâche de chercher à quoi ressemble le rendu final d'un asset. C'est donc l'ensemble des aspects techniques et artistiques nécessaire à la définition de l'aspect et du ressenti visuel qu'offre un projet. Le lookdev s'intéresse donc particulièrement aux deux aspects fondamentaux du rendu : la matière et la lumière. Définition par Lepipeline.Org. 152, 187

mastering

Le *mastering* est l'étape d'adaptation du film sur le format physique utilisé pour sa distribution (DVD, Blu-Ray, etc.). 28, 59

mesh

Un *mesh* est un maillage composé de polygones constituant un modèle 3D. 60, 70, 84, 96, 97, 102, 107, 108, 113, 121, 130, 137, 148, 157, 226

motion capture

La capture de mouvement ou *motion capture* est un processus qui consiste à enregistrer les mouvements d'objets ou d'acteurs, ces données sont ensuite utilisées pour animer des modèles numériques. 63, 64

pair programming

Le *pair programming* (ou programmation en binôme), est une méthode de travail dans laquelle deux développeurs travaillent ensemble sur un même poste de travail. La personne qui rédige le code est appelée conducteur (*driver*). La seconde personne, appelée observateur (*observer*), assiste le conducteur en décelant les imperfections, en vérifiant que le code implémente correctement le design et en suggérant des alternatives de développement. Définition par Wikipédia. 81–83, 121, 161, 175

performance capture

Le processus d'enregistrement des mouvements d'un acteur, utilisé par exemple dans le cinéma et les jeux vidéo. Traduit du Cambridge Dictionnary. 57, 58

ping

Ping est le nom d'une commande informatique permettant de tester l'accessibilité d'une autre machine à travers un réseau IP. Définition par Wikipédia. 102

pitchvis

La *pitchvis* est une méthode de tournage créée pour aider un projet en cours de développement à obtenir le feu vert d'un studio ou à susciter l'intérêt d'investisseurs potentiels. Les pitchvis peuvent inclure des séquences caractéristiques spécifiques du projet proposé ou une bande-annonce générale pour aider à transmettre l'intention créative des réalisateurs. Traduit du Virtual Production Field Guide. 57

placeholder

En jeu vidéo, les *placeholders* sont des versions extrêmement simplifiées des *assets* créées par les artistes. 147, 148, 178

plug-in

Un *plug-in* ou un *add-on* est un programme ou un script tiers qui est ajouté à un programme pour lui donner des fonctionnalités et des capacités supplémentaires. 63, 82, 84–86, 88

post process

Les *post-process* sont des *shaders* appliqués à la fin du pipeline de rendu temps réel ; ils sont utilisés pour changer l'image dans sa globalité. 65

postvis

Les *postvis* impliquent la création d'images fusionnant des éléments réels avec des effets visuels temporaires pour fournir des substituts à l'éditorial. Par exemple, Halon Entertainment a fourni des plans de *postvis* sur *La planète des Singes* en ajoutant des versions temporaires de singes en images de synthèse. La *postvis* fournit au réalisateur et au monteur des scènes visuellement plus développées pour guider leur travail éditorial, en particulier pour les séquences avec des décors partiels et dans lesquelles les effets visuels sont le moteur pour la narration. Elle fonctionne également comme un outil de communication visuel permettant au réalisateur de communiquer avec l'équipe des effets spéciaux. Traduit du Virtual Production Field Guide. 58, 59

previs on set

La *previs on set* est une technique de prévisualisation interactive. Elle rassemble plusieurs unités selon le budget et les besoins de la production. Généralement, elle compte une unité de *motion capture* responsable de la capture des mouvements des acteurs, une unité de rendu responsable de la visualisation temps réel de l'aperçu des effets visuels. 16, 58, 60, 62–64, 69, 155

production tracker

Le *production tracker* est le logiciel (ou ensemble d'outils) qui permet d'harmoniser les départements artistiques et techniques avec les départements de la production. Il permet de gérer un projet, suivre son avancement, faire de l'assignation, faire des plannings, des budgets ou encore faire des *reviews*. Définition par LePipeline.org. 45, 46, 172, 227

props
Éléments de décor d'une scène. Exemple : une casserole. 57, 64, 73

refactoring
En développement logiciel, le *refactoring* désigne le processus de restructuration du code d'un logiciel sans changer son comportement externe. Cette opération vise généralement à améliorer le design, la structure interne du logiciel tout en conservant ses fonctionnalités. 137

refinment
Le *refinment* est la seconde et dernière étape du processus d'animation qui précise les mouvements en ajoutant des interpolations et de nouvelles poses entre les positions clefs définies lors du *blocking*. 133

renderfarm
Une *renderfarm* ou ferme de rendu est un groupe d'ordinateurs dédiés au calcul d'images, de simulations, ou d'autres données. Définition par LePipeline.org. 65

repository
Dans le *Multiuser* (cf. Chapitre II.1), le *repository* est un objet qui stocke une version locale des données de création répliquées. 99, 100, 102–104, 106, 108, 112, 113, 118, 119, 156, 157, 187, 212

retake
Une *retake* est une correction demandée à l'issus d'une *review*. 35, 156, 170, 173, 174, 177, 188

review
Review est le mot utilisé pour définir l'étape de validation d'une tâche. Le travail est passé en revue par les personnes concernées telles que les superviseurs, la production et le réalisateur. C'est au terme de cette étape qu'il est décidé si la tâche est valide et peut être considérée comme terminée. Définition par LePipeline.org. 35, 44, 46, 66, 82, 121, 156, 170, 188, 200

rigging
Le *rigging* est la tâche qui consiste à créer le *rig* d'un *asset*. 30, 41, 63, 66, 107, 176

rig
Le *rig* désigne l'ensemble des *bones* qui permet de déplacer un modèle. Cela inclut les contraintes, les contrôleurs, voir même les *pickers*(interfaces de sélection des *bones*). 30, 31, 33, 41, 42, 63, 67, 84, 87, 133, 134, 148, 156, 187, 200

roadmap
La *roadmap* d'un logiciel est une représentation graphique communiquant les objectifs de livraison de ses différentes fonctionnalités à travers le temps. 143

set
En cinéma, le *set* correspond à l'enceinte dans laquelle est tournée une scène de film ; il comprend les décors et les accessoires. 57

shader
Un *shader* est un type de programme destiné à décrire l'aspect visuel d'un *vertex* ou d'un pixel. 65, 84, 96, 107, 110, 158

shading
Le *shading* est la tâche qui consiste à définir les *shaders* d'un *asset*. 30, 39, 41, 62, 65, 73, 107, 120, 122, 127, 138, 140, 148, 157, 176, 185

shapekeys

Les clefs de formes ou *shapekeys* représentent la position d'un ou plusieurs points de géométrie à un temps donné. Dans d'autres termes, les clés de forme peuvent être appelées « cibles de morphing » (*morph target*) ou « formes de mélange » (*blendshape*). Documentation de *Blender* : `https://docs.blender.org/manual/fr/latest/animation/shape_keys/introduction.html` 52

socket

Un *socket* ou « connecteur réseau » est une interface logicielle qui permet au développeur d'exploiter les services d'un protocole réseau. 100–102

sprint

Dans le *framework* Scrum, un *sprint* représente un intervalle de temps durant lequel l'équipe de développement va planifier puis compléter une certaine quantité de tâches. 165–168, 172–174, 176

storyboard

Le scénarimage ou *storyboard* est une suite de dessins correspondant chacun à un plan et permettant (lors de la préparation d'un film) de visualiser le découpage. Définition par Larousse 26, 27, 31, 33, 152, 153, 198

stuntvis

La *stuntvis* est un type de *techvis* spécialement conçu pour la planification des cascades. Elle est généralement guidée par le coordinateur des cascades ou le chorégraphe de l'action. Elle comprend l'élaboration de la mise en scène, de la chorégraphie et des essais de cascades et de leur d'accessoires. On y explore également les placements et mouvements de caméra, ainsi que la configuration du *lighing*. Elle s'effectue en collaboration avec le directeur de la photographie. En tirant parti de la simulation physique réaliste disponible dans les moteurs en temps réel, les coordinateurs de cascades peuvent directement traduire les résultats numériques en résultats concrets. Les coordinateurs de cascades peuvent directement transposer les résultats numériques dans le monde réel. Traduit du Virtual Production Field Guide. 58

team building

Le *team building* ou renforcement d'équipe, est une méthode qui vise à resserrer des liens sociaux au sein d'un groupe d'individus appartenant à une entreprise ou à une institution. 82

techvis

La *techvis* est la combinaison d'éléments virtuels avec des équipements du monde réel pour le processus de planification des prises de vue ainsi que la combinaison de séquences déjà capturées avec des éléments virtuels. Cette visualisation permet de valider les mouvements et le placement de la caméra ainsi que les choix d'objectifs, ce qui atténue le risque de choix virtuels physiquement invraisemblables. Par exemple, sur le premier Avengers, la *techvis* a été utilisé pour aider à tracer l'animation des explosions capturées en partie sur le plateau et ensuite considérablement élargies par les effets visuels pendant la postproduction. Elle a permis de déterminer la forme, le volume et le taux d'expansion des explosions par rapport au mouvement de la caméra, afin de s'assurer que les effets visuels ajoutés en postproduction s'intègrent de manière crédible aux séquences en direct. Cette technique de visualisation s'attache davantage à la précision physique ainsi qu'aux données spécifiques de la caméra, et moins à la fidélité visuelle. Traduit du Virtual Production Field Guide 58, 201, 211

texturing

Le *texturing* est la tâche qui consiste à générer les textures utilisées par le *shading* d'un objet 3D. Définition par LePipeline.org 30, 86, 107

timeline

Dans *Blender*, la *timeline* est une interface graphique pour visualiser et manipuler la temporalité de la scène. 112, 226

timer

Dans l'*API* Python de *Blender*, un *timer* est une interface qui permet d'enregistrer des fonctions afin qu'elles s'exécutent automatiquement de façon récurrente (par exemple, toutes les cinq secondes). 108, 113, 114

tracking

Le *camera tracking* consiste à reconstituer la position de la caméra en trois dimensions pour reproduire son mouvement virtuellement. 59, 69

viewport

En informatique graphique 3D, le *viewport* désigne le rectangle 2D utilisé pour projeter la scène 3D à l'emplacement d'une caméra virtuelle. C'est une région de l'écran utilisée pour afficher une partie (ou la totalité) de l'image à montrer. Définition par Wikipédia. 96, 97, 105, 108–110, 112, 113

virtual production

La production virtuelle ou *virtual production* est un terme général qui désigne un éventail de méthodes de production et de visualisation de films assistée par ordinateur. 49, 56, 57, 60, 88, 89, 184, 211

virtual scouting

Le repérage virtuel (ou *virtual scouting*) présente une version entièrement numérique d'un lieu ou d'un décor proposé avec laquelle les membres de l'équipe peuvent interagir. L'interaction peut avoir lieu dans un casque de réalité virtuelle ou simplement sur un écran d'ordinateur. La version en rv du *virtual scouting* peut inclure des accessoires repositionnables et des caméras virtuelles. Cette technique est utilisée pour planifier des prises de vues, définir des constructions de plateau et tourner des séquences entières sans construire un seul décor. Elle permet également aux cinéastes de se concentrer sur des zones d'importance spécifique et de filtrer les zones de moindre importance pour éviter de créer des ressources qui ne seront pas utilisées. À ce jour, de nombreux films ont fait appel a du *virtual scouting* durant leur préproduction. Traduit du Virtual Production Field Guide. 57, 60, 155

whitebox level

En jeu vidéo, un *white box level* est un niveau contenant des *assets* graphiques et logiques temporaires pour donner une idée globale du gameplay et de la progression du jeu. 146, 147, 213

widget

En informatique, un *widget* désigne un élément visuel d'une interface graphique (par exemple, un bouton). 110, 111

workflow

Le *workflow* décrit les différentes étapes de progression d'un projet du point de vue des départements de travail. Définition par LePipeline.org 25, 26, 36, 37, 40, 41, 47, 48, 60, 65, 75, 77, 78, 83, 85, 89, 95, 98, 104, 116, 120, 129, 130, 138, 147, 162, 164, 165, 167, 169, 170, 177, 183, 185, 187, 212, 213, 221

centrale inertielle

Une centrale à inertie ou centrale inertielle est un instrument capable d'intégrer les mouvements d'un mobile pour estimer son orientation (angles de roulis, de tangage et de cap), sa vitesse linéaire et sa position. Définition par Wikipédia. 70

collection

Dans *Blender*, les collections sont des éléments qui permettent de structurer la hiérarchie de la scène. Elles peuvent s'apparenter à des dossiers de rangement. 145

container

Dans *Docker*, un conteneur est une unité logicielle qui regroupe le code et toutes ses dépendances afin que l'application fonctionne rapidement et de manière fiable d'un environnement informatique à un autre. 158

correctif

Un correctif ou *patch*, est une section de code que l'on ajoute à un logiciel pour y apporter des modifications. 110, 114, 186

cross-DCC

Le terme cross-DCC désigne les solutions de collaborations temps réel propageant la collaboration entre plusieurs logiciels de création numérique. 85–88, 90, 212, 215

dette technologique

On parle de "dette technologique" lorsque des choix « anciens » effectués généralement dans le but de « se simplifier la vie » se retournent contre les concepteurs et empêchent d'apporter des évolutions pourtant prévisibles au projet. Lorsque cela se produit, il faut recoder ou jeter l'existant pour avancer. 39, 40

dépouillement

Cette étape consiste à étudier chaque plan ou séquence et à définir les éléments qui y figurent, comme le nombre de personnages ou de props. Chaque élément sera défini en tant qu'*asset* de production et pourra ainsi être budgétisé. C'est souvent ce qui va déterminer la difficulté des plans, le volume des éléments à fabriquer va permettre de donner un ordre de grandeur de la taille des équipes et du niveau de séniorité requis. Définition par LePipeline.org. 26

instance

Une instance est un pointeur vers un *asset* préexistant. Ce mécanisme permet de placer plusieurs fois cet asset dans la scène pour le coût d'un seul, car il est dupliqué au moment du rendu. Ainsi, une modification visuelle effectuée sur l'*asset* (par exemple, une amélioration de son *mesh*) sera automatiquement répercutée sur ses instances. 140, 147

intégration continue

L'intégration continue (ou *continuous integration*) est une pratique consistant à automatiser l'intégration des modifications du code provenant de plusieurs contributeurs dans un seul projet logiciel. Par exemple, lorsqu'un développeur publie le code d'une nouvelle fonctionnalité, l'intégration continue permettra d'automatiser le lancement de tests sur ce code pour le vérifier puis de le déployer en production. 110

itération

Une itération est une modification d'un fichier ou d'un asset qui est enregistrée (publiée) et versionnée. Le but, la plupart du temps, est une amélioration technique ou artistique. Définition par Lepipeline.Org. 44

MaterialX

MaterialX est un format d'échange de matériaux open source. `http://www.materialx.org` 42

mono-DCC

Le terme mono-DCC désigne les solutions de collaborations temps réel propageant la collaboration au sein de plusieurs instances du même logiciel de création numérique. 85–88, 90, 212, 215

moteur de jeu

Un moteur de jeu ou *game engine* est un logiciel qui offre une suite d'outils et de fonctionnalités aux développeurs de jeux afin de construire efficacement leurs jeux vidéo. Depuis quelques années, certains moteurs de jeu proposent aussi des outils pour la création de films en image de synthèse. 16, 21, 49–56, 58–68, 71, 73, 75, 77, 84, 88, 89, 134, 183, 195

métadonnée

Une métadonnée (ou *metadata*) est une information qui décrit des données qu'elle accompagne. Dans le cadre du *Multiuser*, les métadonnées sont utilisées pour enrichir les informations liées à un utilisateur (par exemple, pour décrire son objet sélectionné). 100, 105, 106, 110, 111, 225

métavers

Un métavers est univers virtuel en deux ou en trois dimensions créé artificiellement par un logiciel et pouvant héberger une communauté d'utilisateurs qui y interagissent sous forme d'avatars. 188

niveau

Un niveau (ou *level* ou *stage* en anglais) ou carte (en anglais *map*) est une étape dans un jeu vidéo. Généralement, il correspond à une unité de lieu dans la progression. Les niveaux se distinguent par diverses caractéristiques : environnement, topographie, ennemis, objectifs, difficulté, etc. Définition par Wikipédia. 50, 87

open source

Un logiciel open source est un programme dont les sources sont librement accessibles pour être éventuellement étudiées, utilisées, modifiées et redistribuées. 39, 42, 43, 52, 70, 71, 93, 110, 135, 136, 143, 181, 193–195, 204, 206

OpenEXR

OpenEXR est un format d'échange d'image multicouche open source. Il fut originellement développé par Industrial Light & Magic. `https://www.openexr.com` 43

pipeline

Un pipeline représente l'ensemble des outils et processus logiciels conçus pour répondre aux besoins d'une production. 15, 26–28, 32, 34, 37–41, 43, 44, 47–49, 60, 62–67, 70, 73–75, 84, 85, 89, 93, 98, 107, 126, 128, 132, 133, 145, 152, 159, 161, 164, 165, 181, 186, 195, 211

pitch

Le pitch, c'est l'histoire formulée en une seule phrase. C'est le lien le plus simple entre le personnage et l'intrigue, qui est généralement constitué d'un évènement déclenchant l'action, d'une indication sur le personnage principal et d'une indication sur le dénouement de l'histoire. John Truby dans l'*L'anatomie du scénario* 32

plan

Un plan au sens cinématographique du terme est une prise de vue, comprise entre la mise en marche de la caméra et son arrêt. Le plan est la plus petite unité temporelle utilisée pour segmenter le travail de création d'un film. 25–28, 34, 46, 50, 56, 58, 59, 65, 88, 198

prévisualisation

La prévisualisation désigne l'ensemble des outils permettant de prévoir, de manière visuelle, le résultat d'un tournage à venir. C'est une technique de description et d'assistance à la mise en scène. Étape fondamentale dans la conception d'un film en prise de vue réel, la prévisualisation permet à un réalisateur, un directeur de photographie ou encore un superviseur FX d'expérimenter différentes mises en scène ou directions artistiques telles que l'éclairage, le cadrage et le montage pour un budget réduit. Plus communément, on appelle aujourd'hui prévisualisation ce qui porte en fait le nom d'animatique, c'est-à-dire une séquence d'images en mouvement simulant la version finale d'un plan ou d'une scène de film telle qu'elle apparaîtra à l'écran. Défini partiellement par Benoit Melançon [15]. 25, 58

Ptex

Ptex est un format d'échange de *mapping* de textures développé par Disney. `https://ptex.us/` 42

rotoscopie

La rotoscopie est une technique cinématographique qui consiste à relever image par image les contours d'une figure filmée en prise de vue réelle pour en transcrire la forme et les actions dans un film d'animation. Ce procédé permet de reproduire avec réalisme la dynamique des mouvements des sujets filmés. Définition par Wikipédia. 56

référence

On parle de référence lorsqu'on pointe sur une donnée préexistante. Cela se dit souvent d'un fichier appelé depuis un autre fichier. Les références de fichiers (ou *file reference*) permettent d'assembler des objets 3D sans être contenu directement dans la scène 3D. Si le fichier en référence est modifié, les fichiers l'utilisant bénéficieront des modifications. Si le fichier en référence n'est plus disponible (supprimé, inaccessible, etc.), les fichiers l'utilisant ne pourront plus s'ouvrir correctement et/ou entièrement. Un objet référence contient simplement le chemin d'un autre fichier. Son utilisation apporte de la souplesse et facilite le travail collaboratif. Définition par LePipeline.org. 129

structure

Dans le langage C, une structure (écrite `struct` dans le code source) est un type de donnée composite qui permet de définir une liste physiquement groupée de variables sous un seul nom dans un bloc de mémoire. Les différentes variables

d'une structure sont appelées des membres ou des champs et peuvent être de différent type. 96

séquence

Séquence est un terme technique employé au cinéma et à la télévision pour désigner un ensemble de plans se déroulant dans un même lieu et dans un même temps. En animation les séquences sont définies à l'écriture et au *storyboard*. Elles sont mises en scène en 3D par les équipes de *layout*. Les artistes *layout* travaillent majoritairement 'à la séquence' ce qui leur permet de travailler plus facilement leurs raccords caméra entre les différents plans et s'abstenir de tâches redondantes comme reconstituer ou importer les décors et personnages. Définition par LePipeline.org. 26, 27, 34, 50, 152, 155, 170

transparence

La transparence est un procédé mêlant, dans un plan, une projection de film à une prise de vues. Employée en studio pour donner l'illusion de voir derrière les comédiens une scène (filmée auparavant) dont on veut faire croire qu'elle se déroule en même temps que le jeu des comédiens. Définition de Wikipédia. 59

tâche

Généralement, une tâche (ou *task*) est un travail à faire dans un temps déterminé et à certaines conditions. Définition par Larousse. Plus spécifiquement au cinéma d'animation, la tâche de travail est une étape individuelle qui s'intègre dans le processus de fabrication d'une entité. Définition par LePipeline.org. 34

VDB

VDB est un format d'échange de données volumétriques open source originellement développé par DreamWorks Animation. `https://www.openvdb.org/` 42, 44

Bibliographie

Livres

[1] Adams, Ernest, Dormans, Joris. *Game Mechanics : Advanced Game Design*. New Riders, 2012. 442 p. isbn : 978-0-321-82027-3. Google Books : `_Azio0txIdAC`.

[2] Dunlop, R. *Production Pipeline Fundamentals for Film and Games*. CRC Press, 2014. isbn : 978-1-317-93622-0. url : `https://books.google.fr/books?id=a2PMAgAAQBAJ`.

[3] Kadner, Noah. *Virtual Production Field Guide Volume 1*. Epic Game, 2019. url : `https://cdn2.unrealengine.com/Unreal+Engine%2Fvpfieldguide%2FVP-Field-Guide-V1.2.02-5d28ccec9909ff626e42c619bcbe8ed2bf83138d.pdf`.

[4] Kadner, Noah. *Virtual Production Field Guide Volume 2*. Epic Game, 2021. url : `https://cdn2.unrealengine.com/Virtual+Production+Field+Guide+Volume+2+v1.0-5b06b62cbc5f.pdf`.

[5] Kirmse, Andrew. *Game Programming Gems 4*. Charles River Media, 2004. 703 p. isbn : 978-1-58450-295-1. Google Books : `h8J1QgAACAAJ`.

[6] Pohl, Brian J., Brakensiek, Tim, Lombardo, Simone. *Fortnite Trailer : Developing a Next Gen Pipeline for a Faster Workflow*. 2017. url : `https://cdn2.unrealengine.com/Unreal+Engine%2Fresources%2FVirtual-Production-white-paper-62e08e13d5ff8b76f17286b33297e741bdca5db5.pdf`.

Articles

[7] Cockburn, A., Williams, L. « The Costs and Benefits of Pair Programming ». *undefined* (2001). url : `https://www.semanticscholar.org/paper/The-costs-and-benefits-of-pair-programming-Cockburn-Williams/5ff7b75b20fdbfae23587b660b7093aec2f48e69` (visité le 18/08/2021).

[8] Dybå, Tore et al. « Are Two Heads Better than One ? On the Effectiveness of Pair Programming ». *IEEE Software* 24.6 (nov. 2007), p. 12-15. issn : 1937-4194. doi : `10.1109/MS.2007.158`.

[9] George-Molland, Anne-Laure. « L'imaginaire des auteurs-réalisateurs à l'épreuve des processus de création numériques ». *Arts et Technologies de l'Image, Images numériques et Réalité Virtuelle* (1er déc. 2011). url : `https://halshs.archives-ouvertes.fr/halshs-02105973` (visité le 07/09/2021).

[10] Hnídek, Jiří. « Network Protocols for Applications of Shared Virtual Reality ». Dans : Václav Skala - UNION Agency, 2011. isbn : 978-80-86943-82-4. url : http://dspace5.zcu.cz/handle/11025/10817 (visité le 19/08/2021).

[11] Lauwers, J. Chris, Lantz, Keith A. « Collaboration Awareness in Support of Collaboration Transparency : Requirements for the next Generation of Shared Window Systems ». Dans : *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '90. New York, NY, USA : Association for Computing Machinery, 1er mars 1990, p. 303-311. isbn : 978-0-201-50932-8. doi : 10.1145/97243.97301. url : https://doi.org/10.1145/97243.97301 (visité le 22/02/2021).

[12] Maritnez, Swann, Chen, Chu-Yin. « Le périphérique mobile, interface de création interactive pour le cinéma d'animation ». Revue ATI-INREV.6 (2019), p. 9. url : https://inrev.univ-paris8.fr/IMG/pdf/revue_inrev-ati_n6.pdf (visité le 05/01/2021).

[13] Martinez, Swann, Chen, Chu-Yin. « A Framework Enabling Real-Time Multi-User Collaborative Workflow in 3D Digital Content Creation Software ». Dans : *Computer Science Research Notes*. WSCG. Václav Skala - UNION Agency, 2021. isbn : 978-80-86943-34-3. doi : 10.24132/CSRN.2021.3101.10. url : http://dspace5.zcu.cz/handle/11025/45013 (visité le 04/09/2021).

[14] Martinez, Swann, Chen, Chu-Yin. « Être en apesanteur :. Une approche diégétique en réalité virtuelle ». *Entrelacs. Cinéma et audiovisuel* 17 (17 27 juin 2020). issn : 1266-7188. doi : 10.4000/entrelacs.5918. url : http://journals.openedition.org/entrelacs/5918 (visité le 04/09/2020).

[15] Melançon, Benoit. « La prévisualisation 3D comme outil de gestion de risque en production cinématographique : état des lieux et dérapages possibles ». *Cahiers d'histoire* 33.1 (2014), p. 69-80. issn : 0712-2330, 1929-610X. doi : 10.7202/1029362ar. url : https://www.erudit.org/fr/revues/histoire/2014-v33-n1-histoire01788/1029362ar/ (visité le 22/09/2021).

[16] Mur-Artal, Raul, Montiel, J. M. M., Tardos, Juan D. « ORB-SLAM : A Versatile and Accurate Monocular SLAM System ». *IEEE Transactions on Robotics* 31.5 (oct. 2015), p. 1147-1163. issn : 1552-3098, 1941-0468. doi : 10.1109/TRO.2015.2463671. url : http://ieeexplore.ieee.org/document/7219438/ (visité le 15/07/2018).

[17] Pohl, Brian J. et al. « Fortnite : Supercharging CG Animation Pipelines with Game Engine Technology ». Dans : *Proceedings of the ACM SIGGRAPH Digital Production Symposium*. DigiPro '17. New York, NY, USA : ACM, 2017, 7 :1-7 :4. isbn : 978-1-4503-5102-7. doi : 10.1145/3105692.3114816. url : http://doi.acm.org/10.1145/3105692.3114816 (visité le 04/07/2018).

Thèses et mémoires

[18] George-Molland, Anne-Laure. « La collaboration au cœur du processus de création des œuvres audiovisuelles numériques : analyse des transformations apportées par le développement des technologies et par l'évolution des savoir-faire. » Thèse de doct. Université Paris 8 Vincennes. Saint-Denis, 12 déc. 2007. url : https://hal.archives-ouvertes.fr/tel-01627358 (visité le 20/05/2020).

[19] Perez, Flavio. « Le Pipeline de l'image de Synthèse : Définitions et Enjeux Pour Les Œuvres Collaboratives ». Mémoire de master. Université Paris 8 Vincennes, Saint-Denis.

[20] Picard, Damien. « Le logiciel libre dans un studio d'animation ». Thèse de doct. Université Paris 8 Vincennes, Saint-Denis, 2 déc. 2020.

Pages web

[21] Bagard, Alexandre. *Adam – Animation for the Real-Time Short Film – Unity Blog*. Unity Technologies Blog, 2017. URL : https://blogs.unity3d.com/2016/08/31/adam-animation-for-the-real-time-short-film/ (visité le 05/04/2017).

[22] Dupré, Gwenaëlle. *How To Apply The Scrum Methodology To a Cartoon TV Show Production*. 10 avr. 2019. URL : https://medium.com/cgwire/how-to-apply-the-scrum-method-to-a-cartoon-tv-show-production-bc09c72e40b0 (visité le 04/01/2020).

[23] Ian, Failes. *Upcoming Animated Series 'Zafari' Is Being Rendered Completely With The Unreal Game Engine*. 9 déc. 2017. URL : https://www.cartoonbrew.com/tools/upcoming-animated-series-zafari-rendered-completely-unreal-game-engine-153123.html (visité le 04/02/2021).

[24] Lumsden, Ben. *Blue Zoo Uses Real-Time Technology to Create Distinctive Animated Short "Ada"*. Unreal Engine, 20 nov. 2019. URL : https://www.unrealengine.com/en-US/spotlights/blue-zoo-uses-real-time-technology-to-create-distinctive-animated-short-ada (visité le 28/09/2020).

[25] Pieter, Hintjens. *12/CHP*. URL : http://rfc.zeromq.org/spec/12/ (visité le 19/08/2021).

[26] Semour, Mike. *Scrum in VFX*. 30 mai 2016. URL : https://web.archive.org/web/20210827112013/https://www.fxguide.com/fxfeatured/scrum-in-vfx/ (visité le 04/02/2020).

[27] Seymour, Mike. *Art of LED Wall Virtual Production, Part One : 'Lessons from the Mandalorian' – Fxguide*. FXGuide. URL : https://web.archive.org/web/20210804090201/https://www.fxguide.com/fxfeatured/art-of-led-wall-virtual-production-part-one-lessons-from-the-mandalorian/ (visité le 04/08/2021).

[28] Swann Martinez et al. *Documentation de l'add-on Multiuser*. 2020. URL : https://slumber.gitlab.io/multi-user/ (visité le 17/10/2021).

[29] Tamnev, Plamen. *ADAM – Assets Creation for the Real Time Short Film – Unity Blog*. Unity Technologies Blog, 2016. URL : https://blogs.unity3d.com/2016/08/09/adam-assets-creation-for-the-real-time-short-film/ (visité le 05/04/2017).

[30] Yahin, Arman. *From Russia with Agility : The Studio Pioneering a New Way of Working*. 17 oct. 2017. URL : https://www.foundry.com/insights/film-tv/pioneering-studio (visité le 01/10/2019).

Films

[31] Blomkamp, Neill et al. *Adam : The Mirror*. 2017.

[32] Deschamps, Jérôme et al. *La Véritable Histoire Du Chat Botté*. 2009.

[33] Deveaux, Nicolas. *Athléticus*. 2018.

[34] Froud, Brian, Dumont, Richard M., Little, Pauline. *Zafari*. 2018.

[35] Hérold, Pascal et al. *Cendrillon Au Far West*. 2012.

[36] Lasseter, John. *Toy Story*. 1995.

[37] Michael Clausen, Gavin Moran. *Fortnite Trailer*. 2017.

[38] Pascal, Pedro, Carano, Gina, Esposito, Giancarlo. *The Mandalorian*. 2019.

[39] Rodriguez, Robert et al. *Alita : Battle Angel*. 2019.

[40] Winn, Dane. *Ada*. 2019.

[41] Wyatt, Rupert et al. *La Planète Des Singes : Les Origines*. 2011.

Table des figures

I.1.1 Schéma simplifié des relations entre les grandes familles opérant au sein de la production d'un film d'animation. 24
I.1.2 Étapes artistiques de préproduction en fonction du temps 26
I.1.3 Étapes de production en fonction du temps 27
I.1.4 Étapes de postproduction en fonction du temps. 28
I.1.5 Structure hiérarchique d'un petit département. 31
I.1.6 Structure hiérarchique d'un grand département. 32
I.1.7 Exemple des flux d'entrée et de sortie du département *layout*. 33
I.1.8 Diagramme d'analyse fonctionnelle descendante d'une production d'un film d'animation classique. 33
I.1.9 Cycle de vie d'une tâche à travers son statut. 35
I.1.10 Relation en dataflow et workflow. 37
I.1.11 Dataflow d'une production de série à Cube 40
I.1.12 Le logiciel de moulinette *Kitsudi* développé à Cube Creative par l'équipe R&D 45
I.1.13 Interface du *production tracker Kitsu* sur le projet Caminandes. . . . 46

I.2.1 Exemple d'organisation d'un niveau en section sur *Unreal Engine* avec une coloration des *assets* par section. Source : Documentation d'Unreal Engine 50
I.2.2 Exemple de structure de dossier en miroir (représentée en rouge). . . 52
I.2.3 *Asset manager* (encadré en rouge) dans *Godot*. 53
I.2.4 Audit d'*assets* dans *Unreal Engine*. 53
I.2.5 Illustration des différentes techniques de *virtual production* en fonction de leur temporalité d'utilisation dans une production. 57
I.2.6 Exemple de *techvis* tirée de la vidéo explicative du studio The Third Floor. Source : `https://youtu.be/dHm_o83tW6o` 58
I.2.7 Exemple de plan du *Mandalorian* [38] utilisant le plateau d'ILM StageCraft. Source : FXGuide [27] 59
I.2.8 Productions d'animation étudiées. De gauche a droite et de haut en bas : *Adam* [31] par *Unity Technologies*, *Zafari* [34] par Digital Dimension, *Fortnite Trailer* [37] par Epic Games et *Ada* [40] par BlueZoo Animation. 61
I.2.9 Pipeline du *Fortnite Trailer*. Source : Conférence *Fortnite Trailer Pipeline | Unreal Dev Day Montreal 2017*. 63
I.2.10 Architecture de la première (a) et de la seconde (b) itération de la *Smartphone Remote*. 68
I.2.11 Interface utilisateur des deux itérations de la *Smartphone Remote*. . . 69
I.2.12 Image de l'œuvre en RV : *Être en apesanteur*, développée dans le cadre du projet *Corps Infini*. 71

I.2.13 Organigramme du projet *Corps Infini* montrant la relation entre les élèves (E), les professeurs (P) et la chorégraphe Kitsou (C). 72

I.3.1 *Workflow* de construction dans le BTP. 78
I.3.2 Exemple d'utilisation de BIM en se basant sur l'échange de fichiers IFC dans différents corps de métiers du BTP. Source : `https://openbim.fr` 80
I.3.3 Pair programming en temps réel entre deux développeurs sur l'éditeur *VSCode*. 81
I.3.4 Exemple de configuration cross-DCC et mono-DCC. 85
I.3.5 Historique de sortie des outils de co-création en temps réel. 87

II.1.1 Architecture globale du *framework* 99
II.1.2 Couche de communication réseau 101
II.1.3 Architecture interne du serveur avec une réception synchrone des paquets réseau. 101
II.1.4 Traduction des données avec le RDP 102
II.1.5 *Pipeline* des fonctions primitives de collaboration 104
II.1.6 Interface consciente de collaboration dans la vue 3D de Blender . . . 105
II.1.7 Structure des fichiers de sauvegarde de session. 106
II.1.8 *Multiuser* en action. 107
II.1.9 Exemple de configurations de collaboration permises avec *Blender*. . 107
II.1.10 Panneau affichant la liste des utilisateurs connectés à la session. . . . 111
II.1.11 Outils de synchronisation et de gestion de droits. 112

II.2.1 Exemple du résultat de l'un des exercices de recréation (*Surreal Architecture*). Crédits en Annexe E. 118
II.2.2 Visualisation de l'évolution du graphe de réplication générée à partir des sauvegardes du *repository* du projet *Travel*. Crédits en Annexe E. 119
II.2.3 Résultat du cours d'enseignement collaboratif de la modélisation 3D sur *Blender* 121
II.2.4 Sélection de référence proposée aux étudiants. 122
II.2.5 Préparation du second cours avec les étudiants. 123
II.2.6 Étudiants durant le cours. 123
II.2.7 Scène en cours de création, du point de vue de l'enseignant. 124
II.2.8 Résultat de l'exercice de *lighting* en fonction de la répartition du travail dans la scène. 125
II.2.9 Création de l'éclairage par les étudiants. 125
II.2.10 Première expérimentation avec les superviseurs. 127
II.2.11 *Resource Library* utilisée pour les sessions de *background concept*. . . 129
II.2.12 Tanguy Weyland (*CG Supervisor* sur *Tangranimo*) et Clémence Ottevaere (artiste *layout* sur *Tangranimo*) durant la seconde session de *background concept*. 130
II.2.13 Résultat de la première session de *background concept*. 131
II.2.14 Résultat de la deuxième session de *background concept*. 131
II.2.15 Rendus extraits des vidéos des projets *Mstrd II* (a) et *Elmo* (b) d'animation collaborative. Crédit en Annexe E. 133
II.2.16 Session en mode *lobby* avant son initialisation. 136
II.2.17 Schéma d'organisation préalable à la session (créée par softyoda). Une couleur représente un utilisateur. 139

II.2.18 Résultat de la segmentation par zone sur le projet *Silicon Valley* avec le rendu (a) et la distribution effective (b). Crédits en Annexe E. . . 139
II.2.19 Résultat de la recréation de *Wind mill in Němčice*. Crédits en Annexe E.141
II.2.20 Image de référence (a) du projet de recréation « *All Seing Monolith* » exploitant la segmentation par *asset* (b). Crédits en Annexe E. . . . 141
II.2.21 Résultat de la première session de création libre sur le thème : *zen garden in a snowball*. Crédits en Annexe E. 142
II.2.22 Image tirée de l'animation produite durant la seconde session de création libre sur le thème : *outrun*. Crédits en Annexe E. 142
II.2.23 *kanboard* utilisé sur la conception du *Multiuser*. 144
II.2.24 Utilisation du *kanboard* pour organiser le travail durant le projet *Random House #5*. 144
II.2.25 Rendu de la première session exploitant le *kanboard*. Crédits en Annexe E. 145
II.2.26 Hiérarchies de scène orientée artiste (a) et centré *asset* (b). 146
II.2.27 *Workflow* de la préproduction d'un jeu vidéo. 147
II.2.28 Exemple de *whitebox level* de design de niveau. Source : David Shaver - *Game Developers Conference* 2018 147
II.2.29 Parallélisation du travail dans la session *All Seing Monolith*. 148
II.2.30 Parallélisation du travail dans la session *Outrun*. 149

II.3.1 Annotation faite par Tristan lors d'une session de test de *brief* avec le *Multiuser*. 152
II.3.2 Résultat attendu à la fin de la formation aux *geometry nodes*. 153
II.3.3 Organisation de l'espace. 154
II.3.4 Localisation des participants lors des différents cycles du cours. . . . 154
II.3.5 Illustration d'une proposition de système de gestion de versions pour le *Multiuser*. 156
II.3.6 Comportement du système de droit actuel et idéal lors de la sélection d'un objet dans la scène. 157

II.4.1 Carte des différentes méthodes agiles. Source : `https://www.agilealliance.org` 163
II.4.2 Exemple d'une partie du *backlog* de Cube Creative 166
II.4.3 *Workflow* Scrum. 167
II.4.4 Comparaison de la méthode *Scrum* et de la méthode en cascade. . . 168
II.4.5 Relations au sein du studio Main Road Post entre les graphistes (G) répartis en équipes pluridisciplinaires et le réalisateur (R). 170
II.4.6 Structure d'équipe spécialisée utilisée par Gwenaëlle pour le *layout* et l'animation. Les équipes de graphistes layout(L) et d'animateur (A) interagissent avec les superviseurs (S) de leur département et le réalisateur (R). 171
II.4.7 Différence de gestion entre des tâches complexes et simples ; leur planification est plus simple et précise lorsqu'elles sont petites et de durée similaire. Source : FXGuide [26] 172
II.4.8 Exemple de suivi de l'animation sur un *kanboard* et sur un *production tracker*. 173
II.4.9 Composition fictive d'une équipe de co-création en temps réel. 175
II.4.10 Temporalité des projets menés avec le *Multiuser*. 177

B.1 Extrait d'un épisode de *Mush Mush*. Source : La Cabane Production 221

Liste des tableaux

I.1.1 Différents métiers d'une production de série d'animation à Cube Creative 31
I.1.2 Entrées et sorties des différents départements que l'on retrouve au sein de la phase de production d'une série. 33
I.1.3 Informations supportées par les formats standards en 2021. 42
I.1.4 Compatibilité entre DCC et standards en 2021. 43

I.2.1 Caractéristiques techniques des productions d'*Adam*, de *Zafari*, du *Fortite Trailer* et d'*Ada*. Sources : *Adam* : [21] [29] - *Zafari* : [23] - *Fortnite Trailer* : [17] [6] - *Ada* : [4, p. 114-116] [24] 62
I.2.2 Nombre d'artiste par étape de production (en comptant les *leads*) sur les saisons 1 de *Tangranimo* et *Zafari*. 66
I.2.3 Temps en jour par étape de production sur un épisode de 11 minute pour *Tangranimo* et *Zafari*. 66

I.3.1 Résumé des caractéristiques des approches cross-DCC et mono-DCC. 86

II.1.1 Échantillon d'un dictionnaire de métadonnées utilisateur, utilisé dans l'intégration du *framework* dans Blender 105
II.1.2 Liste des implémentations actuellement en place dans le *Multiuser*. . 109

II.2.1 Temps des sessions de recréation (t) en minutes en configuration mono-utilisateur(SU) et multi-utilisateur(MU) avec la taille d'équipe correspondante(Ts), le nombre total de triangles ($tris$) et le nombre d'objets (Oc) 128
II.2.2 Pourcentage d'occupation du travail des utilisateurs sur les deux sessions de *background concept*. 132

C.1 Outils de confort et automatisation développés à CUBE Creative. . . 226
C.2 Outils de collecte d'information et suivis développés à CUBE Creative. 227
C.3 Asset managers développés à CUBE Creative. 227

E.1 Liste des projets réalisés avec le Multiuser lors des sessions expérimentales académiques (aca), industrielles (ind) avec le nombre de participant (Ts). 232

Annexe A

Interview de Maxime Harenzyk sur le rôle de l'équipe de production au sein du Studio

Swann Martinez : En matière de relations internes, peut-on dire que l'équipe de production est cliente des équipes artistiques et techniques ? (Cliente dans le sens où elle va commander des assets aux équipes artistiques et des outils/données de suivis aux équipes de dev)

Maxime Harenzyk : J'aime pas trop le terme client (mais j'aime pas son utilisation dans le vocabulaire agile, pour le coup). Mais pour considérer les relations entre l'équipe de production et les équipes artistiques et techniques, je dirais que c'est une relation plus latérale ou de codépendance. Tel que je me le représente, ce sont les équipes artistiques et techniques qui créent la série (ou la bande-annonce, le film, la pub, peu importe). L'équipe de production, elle sert d'huile dans les rouages. Ou de ficelle qui aide le tout à coulisser ensemble. Notre travail, ça va être de prendre les données « qu'est-ce qu'on doit produire, pour quand et quels moyens (budget, matériel, locaux) on a pour le produire ? » et établir un plan d'action à partir de ces données, en dialogue avec les autres départements, pour que ceux-ci puissent produire ce qu'on est censé produire, dans les temps impartis. Qu'ils aient les moyens, qu'ils sachent ce qu'ils ont à faire, etc.. Donc en ça, par rapport à ce qui est créé, je ne dirais pas qu'on a ce rôle de client. En revanche, on peut l'avoir vis-à-vis de l'équipe de développement quand on demande un outil particulier pour nous aider à faire notre travail précisément. Mais quand on se fait relais pour demander un outil de prévisualisation des décors à destination des storyboarders, admettons, je dirais que c'est plutôt l'équipe de storyboard et/ou le superviseur technique qui sont clients. Pour moi, dans tout ce type de relations, on a plutôt un rôle intermédiaire.

Swann Martinez : L'équipe de production est-elle bien le principal relais entre les départements artistiques et le client externe ?

Maxime Harenzyk : Pour le coup, oui, l'équipe de production est celle qui fait la jonction entre l'interne et l'externe. Que cet externe le soit vraiment (le réalisateur de Mush Mush) ou pas vraiment (un réalisateur de CUBE Creative, de Xilam ou même un producteur). Les graphistes créent les images, etc., mais c'est la production qui va faire en sorte que la personne externe, quelle que soit son profil, y ait accès, puisse exprimer son retour, et c'est la production qui va rendre

le retour accessible, le discuter ou non, etc.. Bon, dans les grandes lignes et avec des exceptions, bien sûr.

Swann Martinez : Est-ce que vous vous appuyez sur des diagrammes de gantt pour la planification de la production ? Si oui, en aurais-tu un vieux sous la main que je puisse utiliser à des fins d'illustration pour montrer un exemple concret ?

Maxime Harenzyk : On n'utilise pas de diagramme de gantt. Il y a longtemps, il y avait un chargé de production qui utilisait un peu du PERTT dans Microsoft Project, mais globalement on ne s'en sert pas directement. La raison à ça, c'est qu'on fait des gros tableaux / plannings Excel qui sont en général plus compliqués et rarement compatibles. Comme on est sur des trucs avec beaucoup de détails et qui changent beaucoup, un diagramme de gantt va demander beaucoup de boulot d'entretien. Surtout, l'antériorité des tâches est assez « naturelle » une fois qu'on connaît un peu. En série, on fait storyboard > layout > animation > rendu > compositing > étalonnage > postproduction, donc quand tu regardes ton planning tu sais que si le layout est devant le storyboard ça va pas aller. Et du coup... Pas de gantt sous la main.

Swann Martinez : Du coup vis-à-vis des méthodes de la planification vous vous êtes mis d'accord sur un format/modèle commun de planning au sein de l'équipe de production ou c'est chacun à son propre format ?

Maxime Harenzyk : Alors sur la planification, on va dire qu'on a une religion commune et de grosses bases communes, et qu'après au sein de celles-ci on ajuste un peu à notre sauce. Pendant un moment le planning était intégré au document de budget, personnellement je l'en ai sorti assez vite. Avec les documents demandés par Xilam en plus, ça renforce un peu le côté « chacun arrange un poil à sa sauce » ; maintenant ça reste quand même très homogène, on est plus sur du détail et vraiment de l'apparence.

Swann Martinez : Quand tu dis que vous êtes sur des plannings qui changent beaucoup, est-ce des changements induits par le retard que peuvent prendre les étapes de productions ?

Maxime Harenzyk : Pour les changements sur les plannings, ça peut venir de beaucoup de choses. Des retards sur une étape de production (ou de préproduction) : écriture, assets, layout... Si on prend l'exemple de Mush, le layout et l'animation ont pris beaucoup de retard et ensuite au début du rendu on s'est bouffé des mois de retard, donc ça demande fatalement ajustements... Conséquents. Et c'est des dominos après, si une étape prend du plomb dans l'aile, ça affecte tout ce qu'il y a après. Il peut aussi y avoir des changements liés à d'autres choses : problèmes éditoriaux, demandes des producteurs ou distributeurs... Sur TGN Rico pourrait te donner des exemples plus précis. Sur Mush on avait eu le distrib' international qui avait en cours de production décrété que les cordes qui servaient de sangle aux escargots n'étaient pas tolérables pour un public jeune, donc il a fallu repasser dans plein d'épisodes pour modifier, par exemple.

Swann Martinez : Lorsque vous êtes amenés à estimer la durée des étapes de production, combien de temps prévoyez-vous en rab pour éviter les débordements ?

Maxime Harenzyk : Alors ça, c'est une vaste question. Je pense qu'il serait plus juste de parler en pourcentage qu'en durée parce que des étapes peuvent être très différentes en longueur. Le vrai truc, c'est qu'en prod' on aime bien se caler

des sécurités autant qu'on peut, mais ça dépend des exigences du planning. Et souvent, on n'a pas trop le loisir d'en mettre beaucoup parce qu'on se cogne des plannings ultra-serrés. L'idée, c'est de mettre un « buffer » entre les étapes, plus que du rab', parce que sur 52 épisodes d'une série, si tu fais 6 jours au lieu de 5 pour une étape à tous les épisodes, aucun planning ne tiendra. Par contre, on peut prévoir du temps pour que un épisode donné puisse prendre du retard sans que ça affecte le reste. Donc il est important que l'étape suivante (mettons layout et animation) ne démarre pas juste après. Si t'as une semaine de layout de prévue, t'as envie qu'il y ait au moins une bonne semaine entre la fin du layout et le début de l'animation, en cas de souci.

Swann Martinez : Si sur un projet CUBE Creative se charge de la préproduction, production et postproduction, quels seraient les enjeux de l'équipe de prod pour chacune de ces étapes. Pour la préprod par exemple j'imaginais qu'il y avait l'estimation du budget, l'asset breakdown et le recrutement, mais je me trompe sûrement)

Maxime Harenzyk : La préproduction, c'est toute la mise en place, à noter que la limite peut être floue entre préproduction et développement/éditorial. Sur la partie préproduction (on va dire, tout ce qui vient avant le layout et la création d'assets épisodiques), il y a deux parties. D'abord chronologique, sur la prise en main de la série : il faut établir budget, planning et mettre tous les jalons en place pour que la suite se passe bien, que l'écriture se lance et se fasse, qu'on ait un processus de validation avec le ou les producteurs. Il faut préparer le recrutement de l'équipe, etc. Il y a aussi des parties plus concrètes d'asset breakdown, de gestion des scripts validés pour préparer les enregistrements, etc. ; du dépouillement sur les scripts validés Si on parle d'enjeu, le gros enjeu en préproduction c'est tout caler sur des rails pour que la suite se passe de manière saine. Sur l'étape « production », c'est s'assurer que la barque ne tangue pas trop, écoper et être réactif. Sur l'étape « postproduction », c'est assurer la livraison du produit fini, vérifier que rien n'a été oublié en route, etc. En très gros

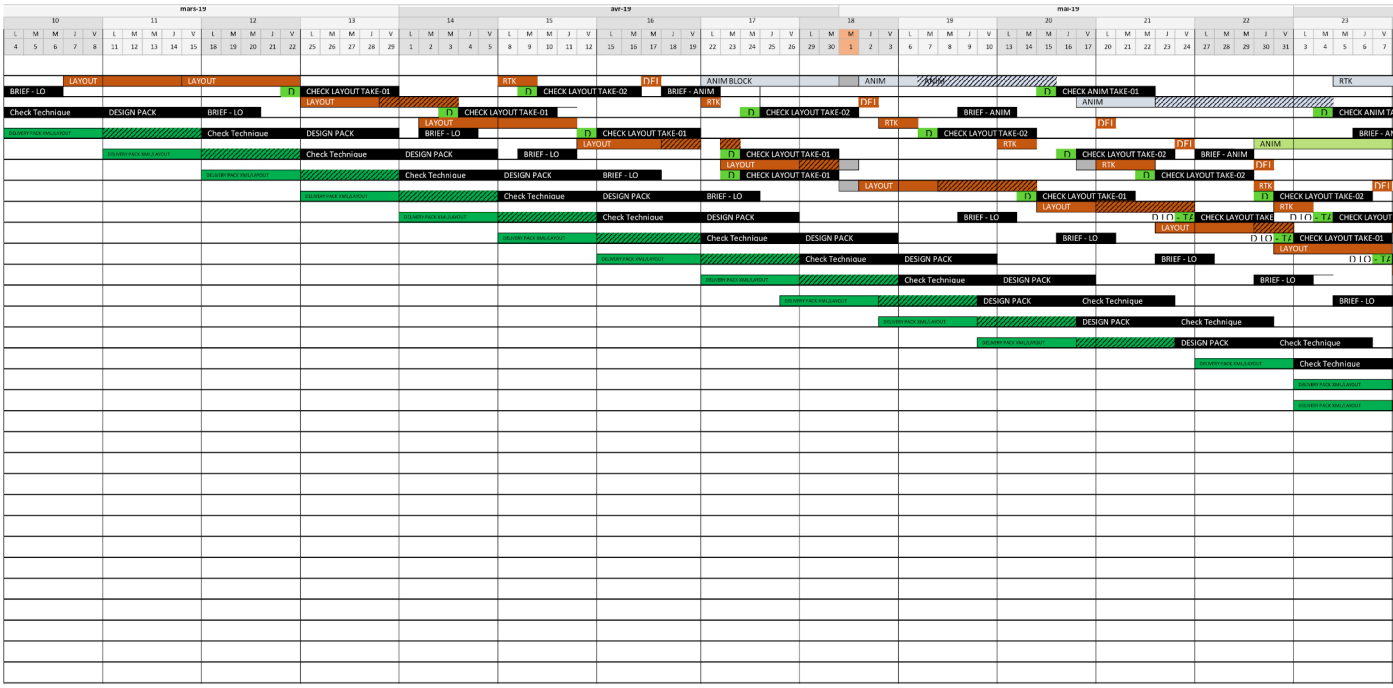
Annexe B

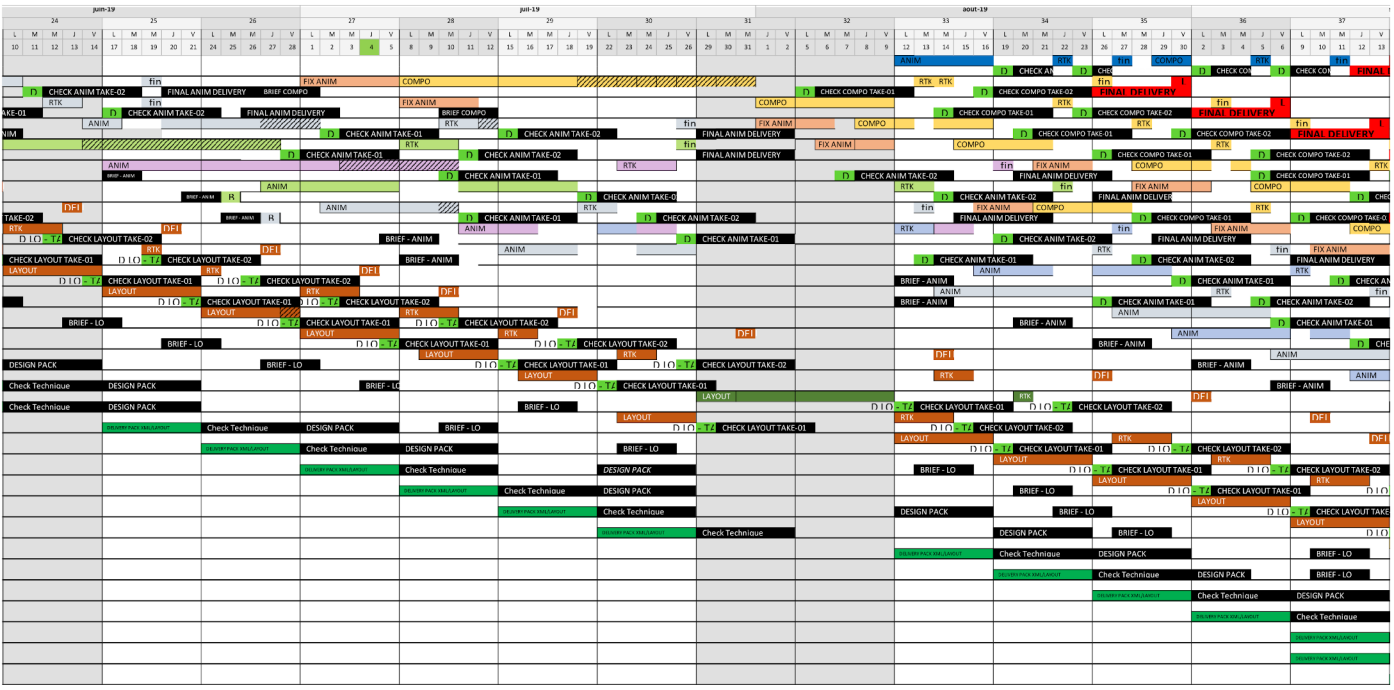
Extrait du planning du projet Mush Mush

L'extrait du planning du projet *Mush Mush* présenté à la page suivante montre le calendrier prévisionnel établi par les chargés de production avant le démarrage de la production. C'est l'un des documents requis par les sociétés qui financent le projet pour pouvoir débloquer les fonds nécessaires à la fabrication.

En ce sens, il témoigne de l'influence du modèle de financement actuel sur le *workflow* de la fabrication de séries et fait écho aux observations du Chapitre I.1.

FIGURE B.1 – Extrait d'un épisode de *Mush Mush*. Source : La Cabane Production


mars-19
avr-19
mai-19
BRIEF - LO
LAYOUT
CHECK LAYOUT TAKE-01
CHECK LAYOUT TAKE-02
BRIEF - ANIM
ANIM BLOCK
ANIM
CHECK ANIM TAKE-01
RTK
DFI
Check Technique
DESIGN PACK


juin-19
juil-19
août-19
BRIEF - ANIM
BRIEF - LO
BRIEF COMPO
ANIM
LAYOUT
DESIGN PACK
Check Technique
FIX ANIM
COMPO
RTK
FINAL ANIM DELIVERY
FINAL DELIVERY
CHECK ANIM TAKE-01
CHECK ANIM TAKE-02
CHECK COMPO TAKE-01
CHECK COMPO TAKE-02
CHECK LAYOUT TAKE-01
CHECK LAYOUT TAKE-02

Annexe C

Liste des outils développés à CUBE Creative

Nom	Type	Description
Add Displace	add-on (*Blender*)	Ajoute un modifier *space wrap*
Additional data kit	add-on (*Blender*)	Enregistrement des nodesgroups génériques en vue de les réutiliser
Alembic Importer	add-on (*Houdini*)	Import des caches d'animation
Align Origin	add-on (*Blender*)	Alignement des origines
Align transform	add-on (*Blender*)	Alignement des objets en position et rotation
Asset Manager	add-on (*Blender*)	Actions globales pour les *assets* présent dans la scène
Au secours manager	standalone	Éditeur pour scinder et préparer les fichiers de rendus
Auto Layer	add-on (*Blender*)	Configuration automatique des layer de la scène
AutoLayer	script (*3Ds Max*)	Configuration automatique des layer de la scène
Barack au bas mot	standalone	Instance Kabana de statistiques de renderfarm
Base Rig	add-on (*Blender*)	Génération automatique de rig
Benedict Renderbatch	standalone	Outil de gestion des jobs de rendus (connecté à Coalition)
Breakdown Importer	add-on (*Blender*)	Import automatique des asset à partir du breakdown
Cc_substance	add-on (*Blender*)	Import / export de texture de Substance Painter
ChainsTools	script (*3Ds Max*)	Création d'une chaine de bones FK en fonction d'une spline
Character Picker	add-on (*Blender*)	Interface de sélection des contrôleurs d'animation
Check Norris	add-on (*Blender*)	Vérification de la nomenclature des objets de la scène
Check Tool	script (*3Ds Max*)	Vérification de la nomenclature des objets de la scène
CHSelect Tool	script (*3Ds Max*)	Character Picker
ClearSkinBones	script (*3Ds Max*)	Réinitialise le skin des bones séléctionnés
CmViewMrg	script (*3Ds Max*)	Supprime les éléments situés en dehors du champ de vision de la caméra afin d'alléger la scène.
Composition Tree	add-on (*Blender*)	Import / export d'arbre de compositing en JSON
ConformTool	script (*3Ds Max*)	Création d'une projection d'un mesh sur un rig existant
Constraint Tool	add-on (*Blender*)	Utilisé en animation pour contraindre un objet à un autre.
Constraint Tool	script (*3Ds Max*)	Utilisé en animation pour contraindre un objet à un autre.
Copy Paste transfo	script (*3Ds Max*)	Copier/coller les transformations d'un contrôleur à un autre
Copy paste transform	add-on (*Blender*)	Copier/coller les transformations d'un contrôleur à un autre
Copy paste weight	add-on (*Blender*)	Copier/coller les influences de skin d'un bone à un autre
Create Cryptomatte manifest	add-on (*Blender*)	Écrit les métadonnées cryptomatte dans un fichier JSON situé dans les dossiers de rendu
CreateBaseRig	script (*3Ds Max*)	Créer une base de setup pour un asset simple
Custom Pop	add-on (*Blender*)	Interface de gestion les options IK/FK des membres d'un rig.
Data Users	add-on (*Blender*)	Renseigne les utilisateurs de l'image ou matériaux actif
Drivers tools	add-on (*Blender*)	Automatisation de traitement sur les drivers
Easy Copy	standalone	Interface de copie de fichiers pour l'IT
Emma Whatscene	add-on (*Blender*)	Gestionnaire de scènes
ExportShaders	script (*3Ds Max*)	Permet d'exporter les shaders d'un asset lorsque celui-ci est termine
External data tracker	add-on (*Blender*)	Liste les données externes utilisées dans un fichier Blender
Frustum culling	add-on (*Blender*)	Supprime les éléments situés en dehors du champ de vision de la caméra afin d'alléger la scène.
Generate Clean	script (*3Ds Max*)	Génère une scène sans rig avec les caches d'animation
Generate layout	add-on (*Blender*)	Génère une scène de layout en utilisant la BDL
Homeoffice -> SFTP	standalone	Synchronisation des disques réseaux pour le télétravail
Jamel Debug	standalone	Sauvegarde de logs pour fournir des informations de plantage

Nom	Type	Description
Jean Bauchefort	standalone	Carnet de contacts des collaborateurs de Cube, gestion des candidatures
Jean-Paul Start	standalone	Lanceur d'application
Keith Harig	add-on (*Blender*)	Rig automatique
Kistudi	standalone	Gestion de traitement de lot pour Kitsu
L'Haddock de Cube	standalone	Documentation des processus et outils de CUBE Creative
Manual ABC Export	add-on (*Blender*)	Export automatique d'Alembic
Motion Path	add-on (*Blender*)	Création de chemins de déplacement
Outline mode	add-on (*Blender*)	Switch en mode *outline*
Paint Clones (Cube)	add-on (*Blender*)	Peinture d'instance d'objets
Particle system density	add-on (*Blender*)	Assignation de la densité de particule par lot
Plot sélection	script (*3Ds Max*)	Bake les animations des éléments sélectionnés
PointCache tool	script (*3Ds Max*)	Mise à disposition et récupération des caches d'animation
Pommier	add-on (*Nuke*)	Boite à outils pour Nuke
Preview	script (*3Ds Max*)	Création de preview d'animation
Preview Maker	add-on (*Blender*)	Création automatique de preview
Process Diana	standalone	Automatisation de processus d'organisation réccurents
Puppet_tool	script (*3Ds Max*)	Boite à outils pour gérer les rig de marionnetes
Push/Pull	add-on (*Blender*)	Mise à disposition et récupération du travail
Remote debugger	add-on (*Blender*)	Debugger pour *PyCharm*
Render as thumbnail	add-on (*Blender*)	Sauvegarde le rendu actuel comme Thumbail pour Tube
Render Layer Settings	add-on (*Blender*)	Export / import des paramètres de layers de rendu en JSON
Render output	add-on (*Houdini*)	Prépare des fichiers de configuration pour le rendu
Render Settings	add-on (*Blender*)	Export / import des paramètres de rendu en JSON
Reset transfo	add-on (*Blender*)	Réinitialise les transformation de tout les objets
RiggedSpline	script (*3Ds Max*)	Créér une chaine de bones contraintes à une spline, pour les sangles ou les collier (entre autres)
Ronald Reglage	standalone	Éditeur de fichiers de configuration de projets et d'utilisateurs
SaveSkinBones	script (*3Ds Max*)	Sauvegarde du skin des bones
Scene_manager	add-on (*Blender*)	Gestionnaire de scène interfacé avec Tube
SceneInfo	script (*3Ds Max*)	Rassemble toutes les infos d'un shot ou d'une scène d'asset
SceneManager	script (*3Ds Max*)	Gestionnaire de scène interfacé avec Tube
Select hierarchy	add-on (*Blender*)	Sélection de toute la hiérarchie en un clic
Send Render Job	add-on (*Blender*)	Envois un job de rendu
Set Render Output	add-on (*Blender*)	Assigne automatiquement les chemins de sortie de rendu
Set sélection	add-on (*Blender*)	Interface pour faire ses propres set de sélection
Set sélection	script (*3Ds Max*)	Interface pour faire ses propres set de sélection
Setup particle system texture	add-on (*Blender*)	Création automatique de systèmes de particules pour Mush Mush
Shader viewport update	add-on (*Blender*)	Mise à jour automatique du viewport pour les matériaux animés
Shading Tool	script (*3Ds Max*)	Boite à outils pour automatiser le workflow du shading
Shading workflow	add-on (*Blender*)	Boite à outils pour automatiser le workflow du shading
Sigourney Viewer	standalone	Visionneuse 3D de backgrounds et de props
Simple Renaming Panel	add-on (*Blender*)	Boite à outils de renommage par lot
Sitcky	script (*3Ds Max*)	Système de sticky cluster pour imiter celui de *Maya*
SkinTool	script (*3Ds Max*)	Boite à outils pour gérer le skin des squelettes s'animation
Smooth tools	add-on (*Blender*)	Boite à outils pour gérer le smooth des *meshes*
Split Armature	add-on (*Blender*)	Sépare les armatures après le layout à la séquence
Sprite component	add-on (*Blender*)	Gestion de texture animée pour les matériaux
StretchBones	script (*3Ds Max*)	Raccourcis pour créer un simple bone, contraint entre deux contrôleurs
Stylo CA	script (*3Ds Max*)	Édition d'interfaces personnalisées sur les rig
SyncMagic	add-on (*Blender*)	Mixe des poses de la bibliothèque d'animation et du presse papier
TBDL	standalone	Création et édition de BDL
The Walking Mushrooms	add-on (*Blender*)	Génération automatique de marche
Timeline Tools	add-on (*Blender*)	Boite à outil pour gérer la *timeline*
Worlds Center Shot	add-on (*Blender*)	Déplacement de toute l'action d'un shot au centre du monde pour éviter les artefacts au rendu.
XRef setup	script (*3Ds Max*)	Met à jour un setup ou un xref d'un asset présent dans un shot d'animation
Yves Montage	standalone	Création de shots à partir d'animatiques, vidéos à partir d'images de rendu et génération de Bout à Bouts

TABLE C.1 – Outils de confort et automatisation développés à CUBE Creative.

Nom	Type	Description
Pierre Desprods	standalone	Feuille de temps
CommentEditor	script (*3Ds Max*)	Ajout de commentaire dans la scène
Tags	script (*3Ds Max*)	Gestion des tags des *assets*
Tag Merad	standalone	Ajout ou et modification des utilisateurs ainsi que leurs rôles pour l'accès aux applications de Jean-Paul Start
George W Push	standalone	Renseigne ce qui a été pushé (mis à disposition) et quand

TABLE C.2 – Outils de collecte d'information et suivis développés à CUBE Creative.

Nom	Type	Description
Resource Library	add-on (*Blender*)	Gestion de collection d'*assets*
Tube	standalone	Gestion des *assets* et *production tracker*

TABLE C.3 – Asset managers développés à CUBE Creative.

Annexe D

Script de dessin du graphe de réplication

Le script suivant extrait et dessine le graphe de réplication stocké dans une sauvegarde de session.

Dans un premier temps, le fichier de sauvegarde est décompressé avec *gzip* ❶ puis désérialisé avec *pickle*. Un graphe directionnel[206] est ensuite construit en lisant les nœuds ❷ et leurs dépendances ❸.

Une fois le graphe prêt, on assigne couleur prédéfinie par nœud en fonction de son type ❹ et au contour de chaque nœud selon son propriétaire ❺. On effectue ensuite la répartition des nœuds en fonction d'une force d'attraction ❻. Enfin, on déclenche ensuite le dessin du graphe ❼.

```
1   import gzip
2   import pickle
3   import sys
4   from pathlib import Path
5
6   import matplotlib.pyplot as plt
7   import networkx as nx
8
9   node_color = {
10      'BlAction': "#1f77b4",    'BlArmature': "#aec7e8",
11      'BlCamera': "#ff7f0e",    'BlCollection': "#ffbb78",
12      'BlCurve': "#2ca02c",     'BlFile': "#98df8a",
13      'BlFont': "#d62728",      'BlGpencil': "#ff9896",
14      'BlImage': "#9467bd",     'BlLattice': "#c5b0d5",
15      'BlLight': "#8c564b",     'BlLightprobe': "#c49c94",
16      'BlMaterial': "#e377c2", 'BlMesh': "#f7b6d2",
17      'BlMetaball': "#7f7f7f", 'BlNodeGroup': "#c7c7c7",
18      'BlObject': "#bcbd22",    'BlParticle': "#dbdb8d",
19      'BlScene': "#17becf",     'BlSpeaker': "#9edae5",
20      'BlTexture': "#ffff33",   'BlVolume': "#6a3d9a",
21      'BlWorld': "#fbb4ae"}
22
23  edge_color = {
24      'COMMON': "black"}
25
26  user_color = ["#7fb486", "#2f90d4", "#e83753", "#ddf265",
27                "#d0a8a8", "#ff8800", "#6e8ca4"]
28
29
30  def extract_graph(filepath):
31      with gzip.open(filepath, "rb") as f:
32          snapshot = pickle.load(f)
```

206. https://networkx.org/documentation/stable/reference/classes/digraph.html#networkx.DiGraph

```
33
34          nodes = snapshot.get('nodes')
35          G = nx.DiGraph()
36          # Creation des noeuds
37 ❷        for node, node_data in nodes:
38              G.add_node(node,
39                         owner=node_data.get('owner'),
40                         str_type=node_data.get('str_type'))
41          # Creation des edges
42 ❸        for n, nd in nodes:
43              deps = nd.get('dependencies')
44              if deps:
45                  G.add_edges_from([(n, d) for d in deps])
46
47          # Recuperation d'une couleur par utilisateur
48          for username, user_metadata in snapshot['users'].items():
49              edge_color[username] = user_color.pop()
50
51          return G
52
53
54  def render_graph(graph):
55      fig = plt.figure()
56
57      # Couleur des noeuds en fonction de leur type
58      node_color_map = []
59 ❹    for node, node_data in G.nodes.items():
60          ntype = node_data.get('str_type')
61          node_color_map.append(node_color[ntype])
62
63      # Couleur des edges en fonction du proprietaire du noeud
64      edge_color_map = []
65 ❺    for node, node_data in G.nodes.items():
66          username = node_data['owner']
67          color = edge_color.get(username)
68
69          if color is None:
70              color = user_color.pop()
71              edge_color[username] = color
72
73          edge_color_map.append(color)
74
75      # Layout automatique des noeuds
76 ❻    pos = nx.spring_layout(G,
77                             seed=2342,
78                             iterations=1050,
79                             scale=1.0,
80                             center=[0, 0])
81
82      options = {
83          "node_color": node_color_map,
84          "linewidths": 2,
85          "width": 2,
86          "node_size": 300,
87          'edgecolors': edge_color_map
88      }
89
90      # Dessin du graphe et affichage
91 ❼    nx.draw(G, **options)
92      plt.show()
93
94
95  if __name__ == "__main__":
96      args = sys.argv[1]
97      filepath = Path(args)
98
99      G = extract_graph(filepath)
100     render_graph(G)
101     exit(0)
```

Annexe E

Projets collaboratifs menées avec le Multiuser

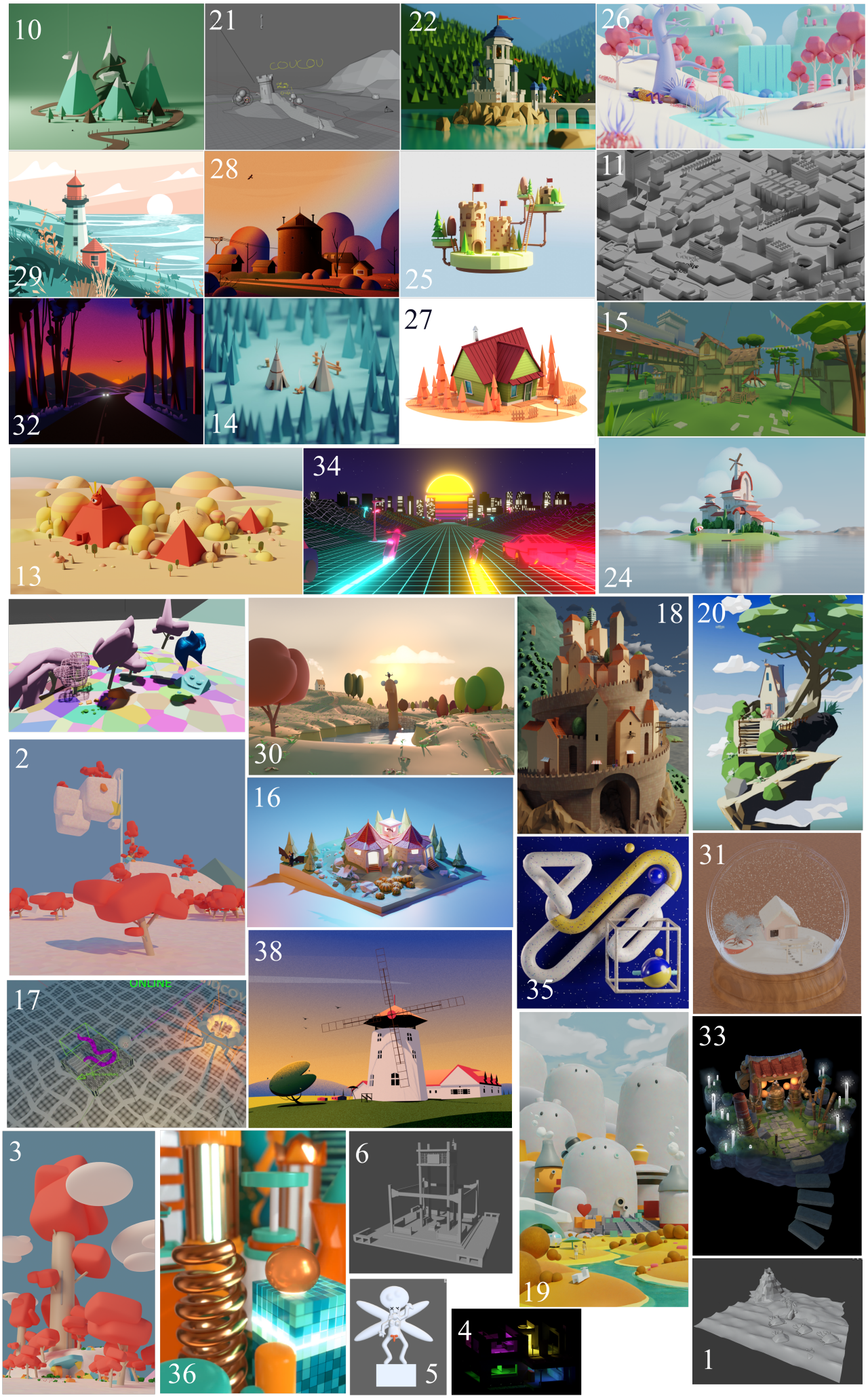
Le Tableau E.1 résume les projets collaboratifs menés sur le *Multiuser* entre août 2019 et septembre 2021. Certains *timelapses* de ces projets sont accessibles librement à l'adresse suivante (à moyen terme, l'intégralité des données liées à ces sessions sera disponible à la même adresse) :

https://doi.org/10.5281/zenodo.5549504

Id	Nom	Type	Ctx	Autheur de la référence	Sessions	Ts	Participants
1	Volcano beach	cours	aca	-	20190716	8	Ponnara, Céline, Laeticia, Julien, Tsov, Sonja, Guofan, Swann
2	Tangra concept 1	bg concept	ind	-	20191003	3	Clémence, Tanguy, Tristan
3	Tangra concept 2	bg concept	ind	-	20191017	2	Clémence, Tanguy
4	Lighing course	cours	aca	-	20200218	5	Ponnara, Ana, Tsov, Laeticia, Swann
5	Metaball exercise	cours	aca	-	20200218	5	Ponnara, Ana, Tsov, Laeticia, Swann
6	Asian temple	création libre	pub	-	20200717, 20200724, 20200728, 20200803	3	Ultr-X, Wuaieyo, Swann
-	Road	recréation	pub	Konstantin Gorshkov	20200807	3	Softyoda, Wuaieyo, Swann
-	Monument valley	recréation	pub	Ustwo Games	20200809	6	Ultr-X, Laurent, Softyoda, Yannick, Wuaieyo, Swann
-	City	recréation	pub	Konstantin Gorshkov	20200828	5	Softyoda, Laurent, Wuaieyo, Ikxi, Swann
10	Paper Summit	recréation	pub	Thomas Gugel	20200904	5	Softyoda, Laurent, Ultr-X, Ikxi, Swann
11	Silicon Valley	recréation	pub	HBO	20200915, 20200918, 20200922, 20200925, 20200929	4	Florian, Softyoda, Wuaieyo, Swann
-	All seing monolith test	recréation	pub	Stuart Wade	20201013	2	Wuaieyo, Swann
13	All seing monolith	recréation	pub	Stuart Wade	20201016	4	Softyoda, Wuaieyo, Staz, Swann
14	Campsite	recréation	pub	Mohamed Chahin	20201020	4	NotFood, Staz, Wuaieyo, Swann
15	Z ! Low Poly Medieval	recréation	pub	Shi Lei	20201027	5	Staz, Wuaieyo, LtTim, NotFood, Swann

Id	Nom	Type	Ctx	Autheur de la référence	Sessions	Ts	Participants
16	Halloween	recréation	pub	Roman Klčo	20201103, 20201106, 20201110	3	NotFood, Wuaieyo, Spacefarer, Swann
17	Freestyle	création libre	ind	-	20201112	4	Tanguy, Bruno, Adrien, Gorgio
18	Surreal Architecture	recréation	pub	Michiel Schrijver	20201113, 20201117, 20201120, 20201124, 20201127	10	Staz, NotFood, Softyoda, Wuaieyo, Oenvoyage, Spacefarer, Tianerad, Fabian, Yoyodespin, Swann
19	Abstract city	recréation	ind	Quan Pham Tung	20201119, 20201126, 20201204, 20201210, 20201216	5	Adrien, Tanguy, Bruno, Gorgio, Axel
20	Sky island	recréation	pub	Péng ái	20201201, 20201204, 20201211	5	Fabian, NotFood, Wuaieyo, Staz, Swann
21	Collaborative course	recréation	aca	-	20201215	5	Pierre, Laura, Anne-Laure, Swann
22	Xbox clubs image	recréation	pub	Alex Pushilin	20201215, 20201222, 20201229, 20210105	5	NotFood, Wuaieyo, Staz, Deajynn, Swann
-	Undo testing	création libre	pub	-	20210112	3	Wuaieyo, NotFood, Swann
24	A comfortable Island	recréation	ind	Jiangyuan Zhong	20210113, 20210218, 20210304, 20210318, 20210325	3	Tanguy, Giorgio, Adrien
25	Castles	recréation	pub	Timothy J. Reynolds	20210115, 20210120	5	Softyoda, Fabian, NotFood, Wuaieyo, Swann
26	Fairy tales	recréation	pub	UV	20210122, 20210126, 20210129, 20210202, 20210205	4	NotFood, Wuaieyo, Yannick, TheLegend5550, Swann
27	Random House #5	recréation	pub	Mohamed Chahin	20210209, 20210212, 20210216, 20210219	3	NotFood, Wuaieyo, Swann
28	Wind mill in Němčice	recréation	pub	Febin Raj	20210223, 20210226, 20210302, 20210305	3	Wuaieyo, NotFood, Swann
29	The Sea	recréation	pub	Sumi77	20210309, 20210312, 20210316	3	NotFood, Wuaieyo, Swann
30	Riverside Evenings	recréation	pub	Paul Corfield	20210319, 20210326, 20210330	5	NotFood, Wuaieyo, Staz, JungleKing, Swann
31	Snowball	création libre	pub	-	20210406, 20210409, 20210414	4	NotFood, Wuaieyo, Bigbossbro08, Swann
32	Travel	recréation	ind	Febin Raj	20210401, 20210408, 20210415	4	Adrien, Tanguy, Bruno, Gorgio
33	Ancient Gods Threshold II,	recréation	pub	Elodie Mondoloni	20210416, 20210420, 20210423, 20210428, 20210430, 20210504, 20210511, 20210517, 20210521, 20210601, 20210604, 20210608, 20210611, 20210615, 20210618	4	NotFood, Kysios, Wuaieyo, Swann
34	Outrun	freestyle	pub	-	20210622, 20210625, 20210629, 20210702, 20210706, 20210709, 20210713, 20210716, 20210720	5	NotFood, Fabian, Wuaieyo, Ultr-X, Swann
35	Mstrd II	recréation	ind	gifmk7	20210422, 20210506	3	Tanguy, Gorgio, Adrien
36	Elmo	freestyle	ind	ELMØ	20210521, 20210603, 20210610, 20210617, 20210624, 20210701	4	Gorgio, Fabian, Tanguy, Adrien
-	Brief Layout	freestyle	ind	-	20210707, 20210715	5	Tristan, Jerome, Tanguy, Fabian, Swann
38	Wind mill	recréation	pub	Febin Raj	20210723, 20210727, 20210730, 20210803, 20210806, 20210810, 20210817, 20210820	4	NotFood, Fabian, Ikxi, Swann

TABLE E.1 – Liste des projets réalisés avec le Multiuser lors des sessions expérimentales académiques (aca), industrielles (ind) avec le nombre de participant (Ts).



Projets achevés durant les sessions expérimentales qui prirent place entre juillet 2020 et août 2021.