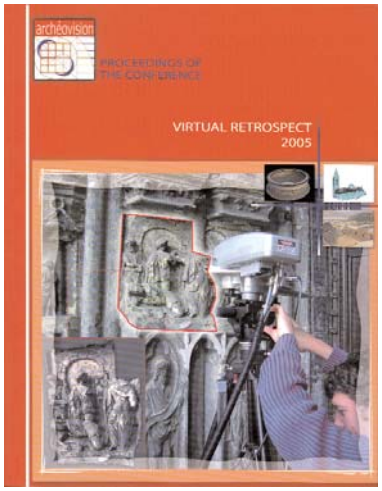
Modélisation hiérarchique : visite virtuelle de la crypte de la cathédrale de Bruxelles

Gauthier Hélin, Dragomir Milojevic, Nadine Warzée



Vergnieux R. et Delevoie C., éd. (2006), Actes du Colloque Virtual Retrospect 2005, *Archéovision 2*, Editions Ausonius, Bordeaux

Tiré-à-part des Actes du colloque
Virtual Retrospect 2005

Biarritz (France) 8, 9 et 10 novembre 2005

MODÉLISATION HIÉRARCHIQUE : VISITE VIRTUELLE DE LA CRYPTE DE LA CATHÉDRALE DE BRUXELLES

Gauthier Hélin, Dragomir Milojevic, Nadine Warzée
Service des Systèmes Logiques et Numériques - SLN, Université Libre de Bruxelles CP 165/57
50, avenue F. Roosevelt, B-1050-Bruxelles, Belgique
gahelin@ulb.ac.be,
dmilojev@ulb.ac.be
nadine.warzee@ulb.ac.be

Abstract : A Roman crypt, located under the cathedral of Brussels, has been modelled in 3D for the virtual visits purposes. In order to increase the level of perceived realism, the fine engravings on different walls of the crypt have been digitized. The obtained models being very complex, in this article we describe methods implemented in the dedicated 3D visualization engine allowing almost real time rendering of the scene.

Keywords : 3D acquisition — 3D modelling — lightning, — levels of detail — 3D visualisation engines.

1 Introduction

Des fouilles archéologiques effectuées dans les années 1990 sous la cathédrale de Bruxelles ont abouti à la découverte des restes d'une crypte romane [1]. Les murs qui ont été dégagés sont recouverts par endroits d'enduits comportant des graffiti finement gravés. Les voûtes de la crypte d'origine n'existent plus. Avant de pouvoir permettre au public de la visiter, des travaux importants de consolidation doivent être terminés. C'est pourquoi on envisage entre temps de proposer des visites virtuelles. Celles-ci nécessitent de réaliser une modélisation en trois dimensions de la crypte et de l'environnement dans lequel elle se situe. Un logiciel commercial a été utilisé afin d'obtenir un premier modèle. Dans le but de rendre la visite la plus réaliste possible, les modèles des murs ont été acquis grâce à un système d'acquisition 3D. La précision très élevée d'un tel dispositif permet d'obtenir les gravures, mais il en résulte des modèles trop complexes à afficher, même pour des cartes graphiques de la dernière génération. Des méthodes ont été mises en œuvre afin d'arriver à des conditions d'affichage proches du temps réel, même pour des modèles de plus de 700.000 polygones.

2 Modélisation

Afin de reproduire l'architecture de la crypte, nous avons d'abord réalisé une modélisation sommaire de celle-ci. En effet, seule une partie de la crypte est visible car le chœur de la cathédrale des Saints Michel-et-Gudule se situe à environ 2m au-dessus du sol de la crypte. Cette première modélisation a été réalisée sur base des plans disponibles et de relevés complémentaires effectués sur place. Des propositions de restitution des voûtes ont été réalisées en collaboration avec les archéologues. Elles s'inspirent également de voûtes, toujours présentes, d'une autre église comportant une crypte semblable. Les modèles géométriques obtenus ont été complétés par des textures provenant de photos numériques. Cette première modélisation a été effectuée à l'aide du logiciel Autodesk 3DSMax. L'intérêt de reproduire la crypte avec beaucoup de détails vient du fait que les murs comportent des gravures (fig. 1) qui peuvent être extrêmement fines (i.e. de l'ordre du millimètre) [2]. Afin de pouvoir représenter ces gravures avec un maximum de détails, un système de numérisation 3D a été utilisé. Celui-ci a permis d'obtenir un modèle informatique de la géométrie des murs.

Fig. 1 : Exemple de gravures.

3 Acquisition

L'acquisition a été faite à l'aide du système Fastscan de Polhemus, fonctionnant sur le principe de la triangulation laser. Ce dispositif est composé de deux caméras analysant la zone à acquérir sur laquelle est projetée une ligne laser. Ce système permet d'obtenir une précision inférieure au millimètre, ce qui correspond bien à la finesse des gravures que l'on veut modéliser (fig. 2). Le problème de cette méthode d'acquisition est que la distribution des triangles est uniforme sur la surface scannée, ce qui donne un modèle comportant plus d'un million de triangles pour une surface scannée de 1 m^2 [3]. Il est donc impossible d'exploiter directement ce type de données dans un moteur 3D classique, même en utilisant une carte graphique de dernière génération.

Fig. 2 : Acquisition 3D des gravures.

4 Modélisation hiérarchique

Afin de résoudre ce problème, une modélisation hiérarchique a été mise en œuvre. Ce type de modélisation consiste à stocker plusieurs versions de la même géométrie, mais avec des niveaux de détails différents. Une fois ces versions importées dans le moteur 3D, il ne reste qu'à sélectionner celle qui correspond le mieux à ce qui est vu par l'utilisateur. Ce type d'approche se justifie par le fait que lorsqu'un objet visualisé est situé à une grande distance de l'observateur, il se projettera sur une petite partie de l'écran seulement : seuls quelques pixels sont utilisés, ce qui implique que peu de détails pourront être rendus. Il est donc intéressant d'utiliser une version simplifiée de la géométrie des objets lointains car l'observateur ne pourra pas faire la distinction entre celle-ci et la version complexe de la géométrie.
La création de cette hiérarchie peut se faire de différentes manières.

4.1 Hiérarchie discrète

Ce type de hiérarchie peut être obtenu de deux manières différentes: par une création manuelle ou automatique. Les modèles créés manuellement par l'utilisateur correspondent aux différents niveaux de détails désirés (fig. 3). Ces modèles sont alors stockés, par ordre croissant de complexité. A chaque modèle est associée une erreur acceptable entre le modèle original et le modèle simplifié. Elle peut être calculée de différentes manières, comme nous le verrons au paragraphe 5.1. Une telle hiérarchie peut aussi être créée via un algorithme de simplification. Ceux-ci sont décrits dans [4]. La hiérarchie ainsi obtenue peut être stockée dans un fichier afin de ne pas avoir à la recréer à chaque chargement du modèle.
Le problème principal de ce type de simplification hiérarchique vient du fait que lors de la création des différents modèles, la position du point de vue n'est pas connue. Une simplification uniforme sur toute la surface est donc réalisée. Afin d'éviter ce problème nous pouvons avoir recours à une hiérarchie de type continue (fig. 4).

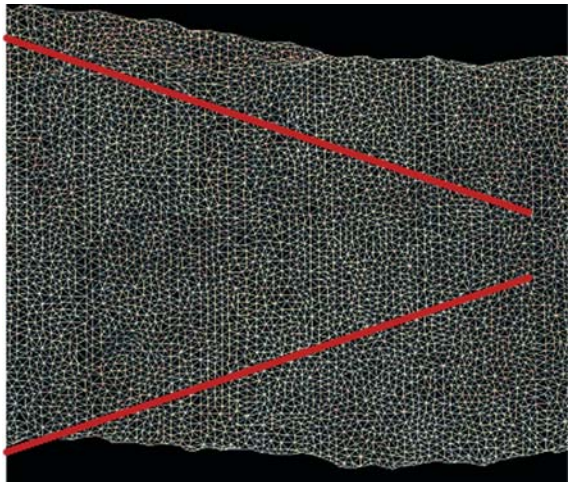


Fig. 3 : Simplification par hiérarchie discrète.

4.2 Hiérarchie continue

Dans les hiérarchies continues, les modèles ne sont plus stockés à des niveaux de détails différents, ce sont des combinaisons d'opérateurs de simplifications qui sont mémorisées. La création de cette hiérarchie se fait durant une phase de prétraitement de la géométrie. Une fois créée, il est possible de stocker la hiérarchie dans un fichier afin de ne pas avoir à la recalculer à chaque chargement du modèle. Les opérateurs de simplification utilisés sont généralement du type "Full Edge Collapse" ou "Half Edge Collapse" [4,5].

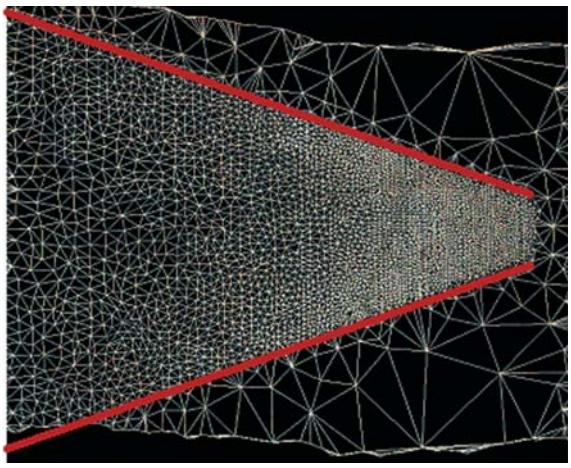


Fig. 4 : Simplification par hiérarchie continue.

5 Gestion des niveaux de détails

La gestion des niveaux de détails consiste en la création d'une hiérarchie – continue ou discrète – à partir d'un modèle de base et en la sélection du niveau de détails minimisant l'erreur entre le modèle simplifié et le modèle original, tout en limitant le nombre de triangles afin de ne pas surcharger les calculs à effectuer par la carte graphique. La sélection du niveau adéquat se fait sur base de l'erreur calculée entre le modèle original et le modèle qui sera affiché.

5.1 Métriques d'erreur

Deux métriques différentes peuvent être envisagées, suivant l'espace dans lequel l'erreur est calculée.

5.1.1 Espace objet

Cette métrique est la plus facile à utiliser car elle consiste à calculer la distance, euclidienne par exemple, séparant la surface des deux objets considérés. Ce type de méthode est couramment utilisée lorsque l'on souhaite exporter un objet afin de réaliser d'autres traitements (calculs de vu/caché, de lancer de rayon, …).

5.1.2 Espace écran

Contrairement au calcul de l'erreur dans l'espace objet, on s'intéresse ici à l'erreur produite après projection sur le plan de vue. En effet, on souhaite que l'image résultante à l'écran soit la plus fidèle possible, et ce même si l'objet 3D original est fortement déformé à cause des simplifications qui lui ont été appliquées. C'est donc l'image de l'objet qui est importante et non plus l'objet lui-même. En analysant la projection d'un objet sur le plan image (fig. 5) on peut en déduire la relation suivante par simples considérations de triangles semblables.

$$\frac{\varepsilon}{w} = \frac{p}{x} \qquad (1)$$

Cette équation lie l'erreur dans l'espace objet à l'erreur dans l'espace écran exprimée en nombre de pixels.
En isolant p on obtient :

$$p = \frac{\varepsilon x}{w} = \frac{\varepsilon x}{2d \tan\frac{\theta}{2}} \qquad (2)$$

A partir de cette équation, on peut calculer la limite p de la déviation dans l'espace écran pour un niveau de détails et une distance donnés. Nous calculons d comme étant :

$$d=(c\text{-}eye).v\text{-}r \qquad (3)$$

où *v* est le vecteur unitaire de la direction de vue. Il ne reste donc qu'une multiplication et une division pour calculer p à partir des constantes connues de l'équation (2).

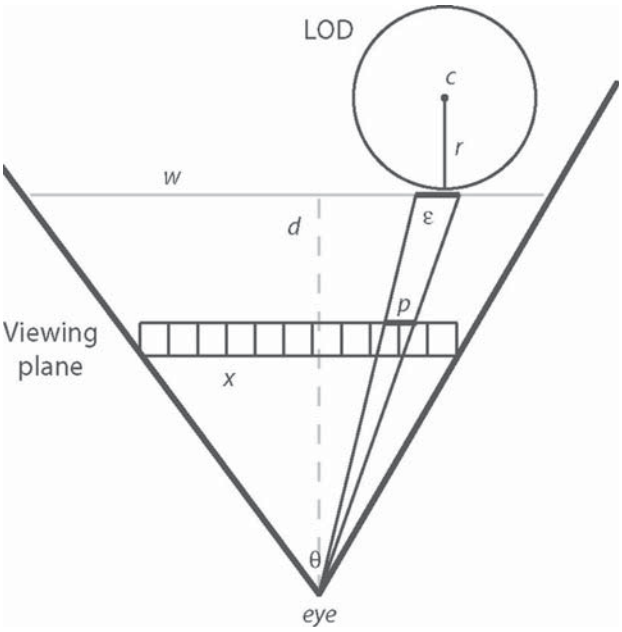


Fig. 5 : Mesure de l'erreur dans l'espace écran [4].

5.2 Sélection du niveau de détails

Une fois l'erreur permise connue, il est possible de sélectionner le niveau de détails correspondant. Dans le cas d'une hiérarchie discrète, cette sélection est simple car il suffit de prendre le modèle correspondant à l'erreur que l'on s'est fixée. Dans le cas d'une hiérarchie de type continue, la sélection se fait par une coupe dans la hiérarchie comme indiqué à la figure 6. Comme nous pouvons le voir à la figure 4, ce type de coupe permet d'obtenir une simplification locale d'un modèle en fonction de la partie de ce modèle qui est vue par l'utilisateur.

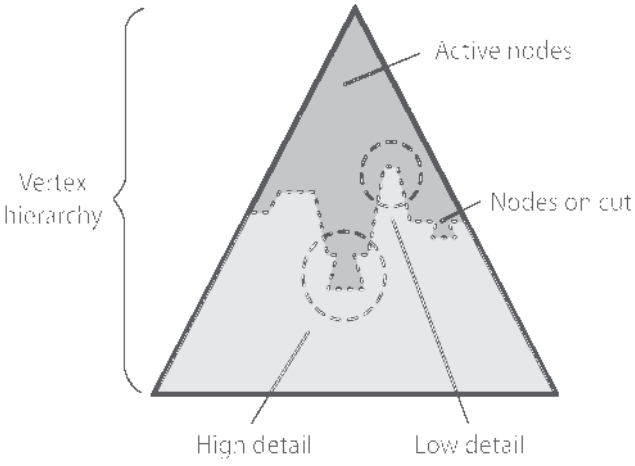


Fig. 6 : Sélection d'un niveau de détails dans une hiérarchie continue [4].

6 Implémentation

6.1 Moteur de rendu 3D

Le moteur 3D a été développé en C++ et en utilisant la librairie graphique OpenGL dans un souci de portabilité, le cas échéant, vers d'autres plates-formes.

Ce moteur supporte différents types de fichiers de données géométriques (comme 3DS et ASE) ainsi que différents types de textures (bmp, jpg et tga).

6.2 Gestion des niveaux de détails

La gestion des niveaux de détails a été réalisée à l'aide de la librairie GLOD développée par l'équipe de Jonathan Cohen (Johns Hopkins University) et David Luebke (University of Virginia). Cette librairie implémente une grande partie des algorithmes les plus couramment utilisés en gestion des niveaux de détails.

6.3 Optimisations

Malgré l'utilisation de la gestion des niveaux de détails, il n'est pas possible d'obtenir une fluidité de mouvement suffisante afin de fournir une visite virtuelle agréable pour le visiteur. Il est donc nécessaire d'avoir recours à des optimisations supplémentaires, comme la découpe par arbre octal [6,7]. Le principe de l'optimisation par arbre octal consiste tout d'abord à créer un cube englobant totalement la scène. Ce cube est alors subdivisé en 8 sous-cubes. Chacun d'entre eux est ensuite à son tour subdivisé en 8 sous-cubes pour autant qu'il contienne encore un nombre suffisant de triangles ou que le nombre de subdivisions maximales ne soit pas atteint.

L'avantage de cette découpe est qu'il est possible de réaliser les sélections de niveaux de détails uniquement dans les zones vues par l'utilisateur, c'est-à-dire situées à l'intersection entre le volume de vue et l'arbre octal (fig. 7).

6.4 Gestion des collisions

Une gestion des collisions a été implémentée afin de permettre une visite virtuelle la plus réaliste possible. Ceci implique un temps de calcul supplémentaire important. Il a donc été nécessaire d'optimiser à nouveau les méthodes utilisées [8].

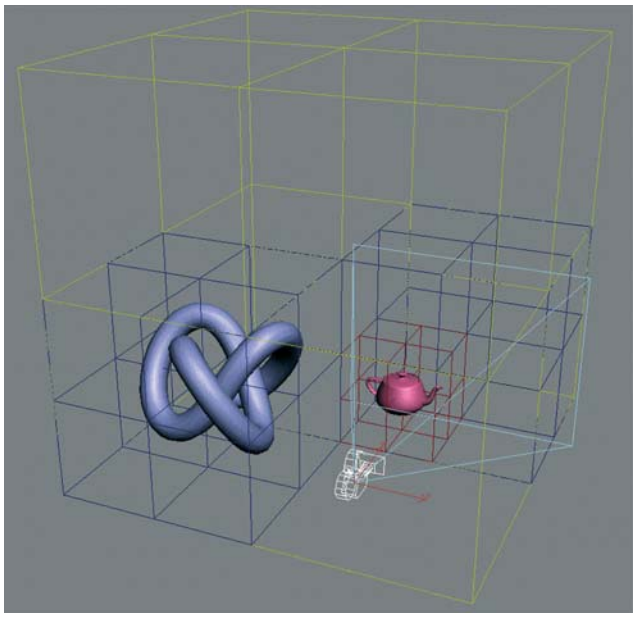


Fig. 7 : Découpe par arbre octal [8].

La solution retenue consiste, d'une part, à stocker une seconde hiérarchie (de type discrète) pour chaque modèle afin de l'utiliser pour la gestion des collisions. D'autre part, les détections de collisions ont été limitées aux zones proches de la position de l'observateur, c'est-à-dire que les collisions ne sont calculées que pour les éléments se trouvant dans le même cube que le point de vue.

6.5 Gestion des lumières

La crypte n'ayant pas de fenêtres, une attention toute particulière a également été apportée à l'éclairage de la scène afin d'essayer de recréer le mieux possible la lumière due vraisemblablement à l'époque à des bougies et des torches. Des lumières dynamiques ont été créées [9]. Pour ce faire nous avons utilisé la partie programmable des cartes graphiques actuelles : les "pixels shaders". Leur utilisation permet de réaliser des effets de lumières avancés afin de simuler le comportement d'une flamme (i.e. scintillement de la flamme, changement de couleurs,…). Un autre avantage de cette technique est son influence très faible sur le nombre d'images affichées puisque tous les calculs sont exécutés directement dans la carte graphique.

7 Résultats

La figure 9 montre le modèle 3D de la crypte chargé dans le moteur 3D développé.

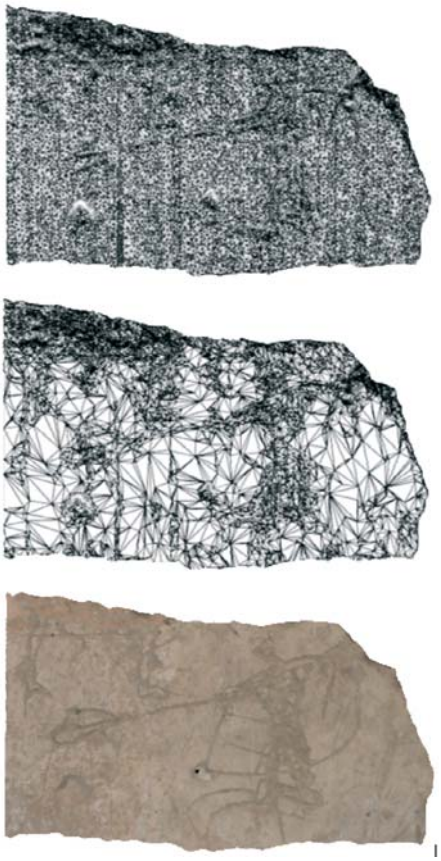


Fig. 8 : Simplification localisée de la géométrie [8].

Fig. 9 : Le modèle de la crypte affiché par le moteur 3D développé.

			GLOD			
	OpenGL		Très Proche		Très Loin	
	SC	AC	SC	AC	SC	AC
	c1	c2	c3	c4	c5	c6
m1	52	10	190	95	252	120
m2	14	3	97	41	94	40
m3	7	1	*150*	27	58	23
m4	4	1	40	12	28	11

Tab. 1 : Performance d'affichage des librairies OpenGl et GLOD.

Le tableau 1 indique la vitesse de calcul de l'application développée, exprimée en nombre d'images affichées par seconde, pour quatre modèles m1, m2, m3, m4, de complexité croissante de 48.000, 192.000, 384.000 et 768.000 polygones. Les colonnes c1 et c2 indiquent la performance de la librairie OpenGL (SC – sans gestion des collisions et AC – avec la gestion des collisions). Les colonnes c3, c4, c5 et c6 reprennent la performance de la librairie GLOD : c3 et c4 lorsque l'on se trouve très proche du modèle (la caméra est placée très près de la géométrie), sans et avec la gestion des collisions; et c5 et c6 lorsque l'on se trouve loin du modèle (la géométrie affichée occupe moins de 10% de l'écran).

Le tableau 2 indique la vitesse de calcul, dans les mêmes conditions, mais cette fois-ci pour des librairies OpenGL et GLOD couplées avec la méthode d'optimisation par arbre octal.

On constate que pour la librairie GLOD avec optimisation, le système parvient à afficher 20 images pour seconde (colonnes c11 à c14), même pour le modèle à haute résolution (modèle m4 : 768.000 polygones). Les valeurs en italique dans les tableaux 1 et 2 sont dues à la forme de la géométrie utilisée qui n'est pas la même que pour les autres modèles (i.e. une plus grande proportion de la géométrie qui se situe loin du volume de vue lorsque l'on fait un gros plan). Cette différence n'apparaît pas lorsque toute la géométrie est affichée.

	OpenGl + arbre octal				GLOD + arbre octal			
	Très Proche		Très Loin		Très Proche		Très Loin	
	c7	c8	c9	c10	c11	c12	c13	c14
	SC	AC	SC	AC	SC	AC	SC	AC
m1	199	198	64	64	185	184	250	250
m2	199	185	16	16	96	96	90	90
m3	199	183	8	8	*147*	*147*	55	55
m4	199	182	4	4	38	38	25	25

Tab. 2 : Performance d'affichage des librairies OpenGL et GLOD couplé avec l'optimisation basée sur les arbres octaux.

8 Conclusion

Une application de visualisation de modèles 3D complexes permettant des visites virtuelles en temps réel a été présentée. Grâce aux méthodes d'optimisation utilisées, le moteur 3D développé permet de visualiser des environnements très complexes, tout en respectant la contrainte d'un affichage d'au mois 20 images par seconde. Cette performance comprend la gestion des collisions ainsi que la possibilité d'ajouter des lumières dynamiques, pour un niveau de réalisme encore plus important.

Bibliographie

[1] Bonenfant, P. et M. Fourny (1992) : "Fouilles dans le chœur de la cathédrale Saint-Michel à Bruxelles", Archeologia Mediaevali, 15-47.

[2] Balzani,M., M. Callieri, M. Fasano, C. Montani, P. Pingi, N. Santopuoli, R. Scopigno, F. Uccelli and A. Varone (2004) : "Digital representation and multimodal presentation of archeological graffiti at Pompei", The 5th International Symposium on Virtual Reality, Archaeology and Cultural Heritage, 93-103.

[3] El-Hakim, S., G. Godin, V. Valzano, A. Bandiera, J.-A. Beraldin, and M. Picard (2003) : "Virtualizing a byzantine crypt : Challenge and impact", Videometrics VIII. Proceedings of the SPIE, 501, 148-159.

[4] Luebke, D., M. Reddy, J. D. Cohen, A. Varshney, B. Watson et R. Huebner (2003) : *Level of Detail for 3D Graphics*, Morgan Kaufmann Plublishers.

[5] Zhu, Y. (2005) : "Uniform Remeshing with an Adaptive Domain: A New Scheme for View-Dependent Level-of-Detail Rendering of Meshes", IEEE Transactions on Visualization and Computer Graphics, 11,3, 306.

[6] Luebke, D. et C. Erikson (1997) : "View-Dependent Simplification Of Arbitrary Polygonal Environments. Computer Graphics", SIGGRAPH '97 Proceedings, 31, 199-208.

[7] Shaffer, E. et M. Garland (2005) : "A Multiresolution Representation for Massive Meshes", IEEE Transactions on Visualization and Computer Graphics, 11,2, 139-148.

[8] Yoon, S.-E., B. Salomon, M. Lin et D. Manocha (2004) : "Fast collision detection between massive models using dynamic simplification", Eurographics Symposium on Geometry Processing, 139-149.

[9] Gillain, R. (2005) : *Reconstruction Virtuelle d'une Crypte*, Université Libre de Bruxelles.