Modélisation géométrique

Dmitry Sokolov

UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Modélisation géométrique

THÈSE

soutenue le 10 juin 2016

pour l'obtention d'une

Habilitation de l'Université de Lorraine

(mention informatique)

par

Dmitry SOKOLOV

Composition du jury

Président :	Isabelle DEBLED-RENNESSON, Professeure en Informatique à Université de Lorraine
Rapporteurs :	Dominique BECHMANN, Professeure en Informatique à l'Université de Strasbourg Alexander BELYAEV, Associate Professor in School of Engineering & Physical Sciences, UK Pascal FREY, Professeur en Mathématiques appliquées à l'Universite Pierre et Marie Curie
Examinateur :	Bruno LÉVY, Directeur de Recherche à l'Inria Nancy-Grand Est

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Contents

I Procedural modeling **7**

1 Boundary Controlled Iterated Function Systems **9**
1.1 Background . . . 9
1.1.1 Iterated Function Systems . . . 9
1.1.2 Controlled / Language Restricted IFS . . . 11
1.2 Boundary Controlled IFS . . . 14
1.2.1 Specifying the topology . . . 14
1.2.2 Controlling the geometric texture . . . 19
1.2.3 Example . . . 20
1.3 Subdivision curves and surfaces as BC-IFS . . . 22
1.3.1 Regular patch . . . 22
1.3.2 Extraordinary vertices . . . 22
1.3.3 Non-uniform B-spline curves . . . 26

2 Tangent spaces for self-similar shapes **32**
2.1 Related works . . . 34
2.2 Analysis of IFS attractors . . . 38
2.2.1 Complete set of eigenvectors . . . 39
2.2.2 Incomplete set of eigenvectors . . . 42
2.2.3 Conclusion . . . 45

II Meshing **48**

3 Polycube maps **50**
3.1 Normal constraints for polycube maps . . . 50
3.1.1 Problem statement . . . 53
3.1.2 Formalization . . . 54
3.1.3 Algorithm overview . . . 56
3.1.4 Results and discussion . . . 64
3.2 Robust tracing of streamlines on triangulated surfaces . . . 65
3.2.1 Field representation . . . 67
3.2.2 Stream-mesh . . . 68
3.2.3 Pairing intervals . . . 71
3.2.4 Crossing triangle with arbitrary precision . . . 75
3.2.5 Discussion . . . 75

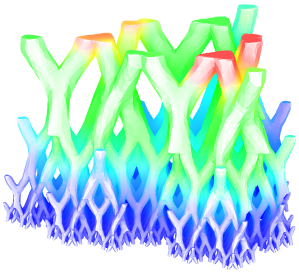
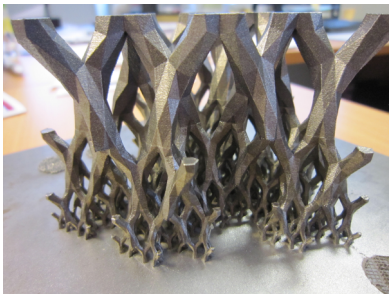
4 Hexahedral-dominant meshing **83**
4.1 On smooth frame field design . . . 86
4.1.1 In search of elusive ground truth, or $d \approx 0.85$ . . . 86
4.1.2 Functional representation of frames in $2D$ . . . 93
4.1.3 Optimization of $3D$ frame fields . . . 98
4.2 Integrating a 3D frame field: periodic global parameterization . . . 113
4.2.1 Problem statement . . . 114
4.2.2 Optimization . . . 116
4.2.3 Extracting gridpoints . . . 118
4.2.4 Optional pre-processing step: curl correction . . . 118

4.3 Generating the hexahedral-dominant mesh 120
4.3.1 Re-meshing the border of the domain 120
4.3.2 Recombining tetrahedra into hexahedra 120
4.4 Results and discussion 128
4.4.1 Hexahedra proportion and quality 128
4.4.2 Influence of the parameters 129
4.4.3 Robustness 131
4.5 Enumerating the decompositions of a hexahedron 135
4.5.1 Decomposition of a hexahedron into 5 or 6 tetrahedra 135
4.5.2 Decomposition of a hexahedron into 7 to 13 tetrahedra 138

Preface

This manuscript shows our first steps towards fundamental and applied aspects of geometry modeling and geometry processing. This research domain deals with acquiring, analyzing and optimizing digital representations of 3D objects. Our ultimate ambition is to develop all the tool chain, from modeling and simulation to physical validation.

The following figure shows a structure we modeled that allows for high vertical loads while minimizing heat transfer:

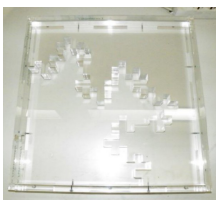



Additive fabrication (3d printing) makes it possible for the first time to create such complicated objects, even in metal (here with a high-end EOS M270 laser sintering printer). This type of technology will have a high societal and economical impact, by creating better systems (engines, cars, planes ...), designed and numerically tailored for optimal functionality thus consuming less raw materials for their fabrication and less energy when they are used.

However, current state computer aided design is not well suited for generation of such types of objects. For centuries, for millennia mankind produced goods with axes, files and CNC mills, removing unnecessary parts from a chunk of wood or plastic. Nowhere in this process we needed abrupt stops of the cutting tool, hence we have excellent background in modelling of smooth objects. That's why we had to wait for the XX century to have necessary mathematical background in order to model rough surfaces or porous structures: we were simply unable to produce them before. First part of this manuscript gives our approach of shape modeling, it is a conjoint work with Pr. Christian Gentil (Université de Bourgogne). This work encompasses previous approaches like free-form surfaces and subdivisions.

Applications domain is vast, it can be design of "fractal" micro-strip antennas, meta-materials and so on. Here is another example of a study performed by the laboratory of vibration and acoustics in Lyon, it shows a simulation and physical validation of normal modes for a domain with "fractal" boundary.

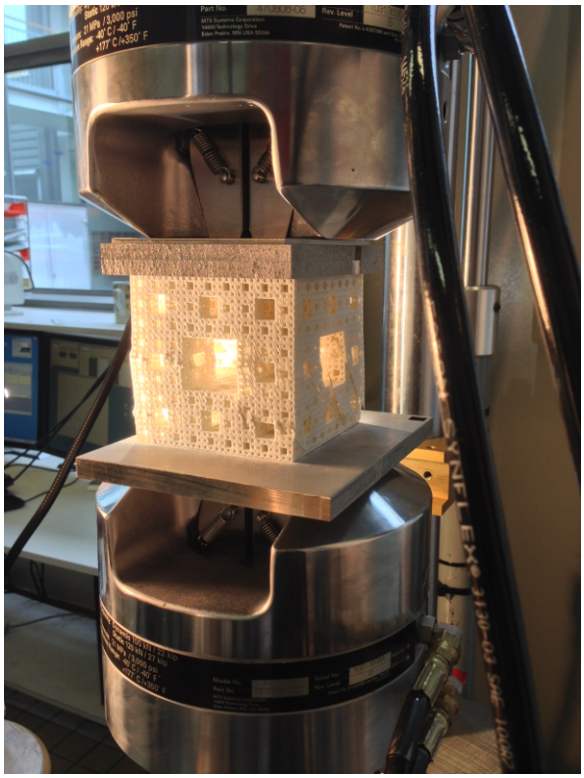


In these new design processes, numerical methods play a central role. Besides some "proofs of concepts", these methods remain to be developed in order to be general and robust enough to be integrated in existing industrial processes. Practitioners often consider the 4/20-node tet FEM to be the holy grail because of its simplicity, however using hexahedra as the discretization element of choice offers some important advantages over tetrahedral meshes. First of all, it has significantly better accuracy, especially in the presence of non-uniform scaling. Hexahedral meshes are much less complex and require much less memory.

Second part of this manuscript presents our work on geometry processing, namely on generation

of hexahedral meshes. It is a result of our current work with Dr. Nicolas Ray and Dr. habil. Bruno Lévy (INRIA Nancy-Grand Est), and I thank them for fructuous and creative team play! Currently we are able to (automatically) produce hexahedral-dominant meshes of acceptable quality and we start to solve our first partial derivative equations on the meshes. Just for a foretaste, here is an example of the heat transfer equation computed on a hexahedral-dominant mesh of a champagne cork:



Part I

Procedural modeling

"Je me détourne avec effroi et horreur de cette plaie lamentable des fonctions continues qui n'ont pas de dérivées."

— Charles Hermite

lettre à Thomas Stieltjes,
20 mai 1893

Objects modeled through Computer Aided Geometric Design (CAGD) systems are often inspired by standard (subtractive) machining processes. However, other types of objects, such as objects with a porous structure or with a rough surface, may be interesting to create: porous structures can be used for their lighter weight while maintaining satisfactory mechanical properties, rough surfaces can be used for acoustic absorption.

Fractal geometry is a relatively new branch of mathematics that studies complex objects of non-integer dimensions. Because of their specific physical properties, fractal-like structure is a center of interest in numerous areas such as architecture [RS14b], jewellery [SYC06], heat and mass transport [Pen10] or antennas [PRP+96, Coh97].

The emergence of techniques such as 3D printers allow for new possibilities that are as yet unused and even unexplored. Different mathematical models and algorithms have been developed to generate fractals. We can roughly categorize them into three families. The first gathers algorithms computing the basins of attraction of a given function. Julia sets and the Mandelbrot set [PR86] or Mandelbulb [Aro09] are some examples. The second is based on simulation of phenomena such as percolation or diffusion [Fal90]. The last one corresponds to deterministic or probabilistic algorithms or models based on the self-similarity property associated fractals like the terrain generator [ZSTR07], Iterated Function System [BHS08], L-system [PL90]. Shapes are generated from rewriting rules providing control of the geometry. Nonetheless, most of these models were developed for image synthesis without consideration of fabricability or were developed for very specific applications like Wood modeling [TGM+09].

Some studies address this aspect for specific applications for 3D printers [SYC06]. In [BV13] Barnsley defines fractal homeomorphisms from $[0,1]^2$ onto the modeling space $[0,1]^2$. The same approach is used in 3D to build 3D fractals. A 3D standard object is embedded in the domain space $[0,1]^3$ and then transformed into a 3D fractal object. This approach preserves the topology of the initial object which is an important point for fabricability.

The **main challenge** with traditional fractals **is the control of the resulting geometry**. For example, it is quite challenging to get the desired shape using the system of fractal homeomorphisms proposed by Barnsley. We elaborate here a new type of modeling system, using the facilities of existing CAGD software, while extending their capabilities and their application areas. This new type of modeling system will offer designers (engineers in industry) and creators (visual artists, stylists, designers, architects, etc.) new opportunities to design and produce a quick mock-up, a prototype or a single object. Our approach is to expand the possibilities of a standard CAD system by including fractal shapes while preserving ease of use for end users.

We propose to use **standard Iterated Function Systems enriched with boundary representation concepts**. It allows to dissociate the topology of final shapes from the geometric texture, greatly simplifying the design process. This approach is powerful, it generalizes standard (linear, stationary) subdivision curves and surfaces, allowing for additional control. For example, with the dissociation of topology from the geometry we were able to fill gaps between a primal and a dual subdivision surface [PGSL14], a challenging topic for the standard subdivision approach.

This part of the thesis consists of two chapters:

- Chapter 1 explains our approach of modelling objects. First section of the chapter describes general settings, namely how to introduce a B-rep structure on a fractal shape (the work was published in [SGGM15]), while the second section shows how standard subdivision curves and surfaces fit with this approach.

- Chapter 2 explains our way to analyze smoothness of produced shapes.

Chapter 1

Boundary Controlled Iterated Function Systems

1.1 Background

1.1.1 Iterated Function Systems

Iterated Function Systems (IFS) were introduced by Hutchinson [Hut81] and further developed and popularized by Barnsley [Bar88]. More research has followed on from these seminal studies [Gen92, Mas97, BHS08, Tos06]. IFS are based on the self-similarity property. A modeled object is made up of the union of several copies of itself; each copy is transformed by a function. These functions are usually contractive, that is to say they bring points closer together and make shapes smaller. Hence, the modeled object, called the *attractor*, is made up of several possibly overlapping smaller copies of itself, each copy also made up of copies of itself, ad infinitum.

Definition. Given a complete metric space $(\mathbb{X}, d)$ with the associated metric d, an IFS is defined by a finite set of continuous transformations $T = \{T_i\}_{i=0}^{N-1}$ in the space $\mathbb{X}$. Let $\Sigma = \{0, \ldots, N-1\}$ be the set of IFS transformation indices, thus $|\Sigma| = N$. The IFS is then denoted by $\{\mathbb{X}; T_i \mid i \in \Sigma\}$.

We are substantially interested in so-called *hyperbolic* IFS, whose transformations T_i are all contractive.

Definition. A transformation $T : \mathbb{X} \to \mathbb{X}$ is called *contractive* if and only if there exists a real s, $0 \leqslant s < 1$ such that $d(T(x), T(y)) < s \cdot d(x, y)$ for all $x, y \in \mathbb{X}$.

Definition. For the set of non-empty compacts of $\mathbb{X}$, denoted $\mathcal{H}(\mathbb{X})$, we define the Hausdorff distance $d_{\mathbb{X}}$ induced by the metric d:

$$d_{\mathbb{X}}(A, B) = \max\{\sup_{a \in A} \inf_{b \in B} d(a, b), \sup_{b \in B} \inf_{a \in A} d(a, b)\}.$$

Since the metric space $(\mathbb{X}, d)$ is complete, the set $(\mathcal{H}(\mathbb{X}), d_{\mathbb{X}})$ is also complete.

In the early 80's, Hutchinson [Hut81] used the Banach fixed point theorem to deduce the existence and the uniqueness of an attractor for a hyperbolic IFS, i.e. the fixed point of the associated contractive map. He defined an operator $\mathbb{T} : \mathcal{H}(\mathbb{X}) \to \mathcal{H}(\mathbb{X})$, now called Hutchinson operator [Bar88], as the union of the IFS transformations T_i:

$$\mathbb{T}(K) = \bigcup_{i=0}^{N-1} T_i(K).$$

If IFS is hyperbolic then $\mathbb{T}$ is also contractive in the complete metric space $(\mathcal{H}(\mathbb{X}), d_{\mathbb{X}})$. According to the Banach fixed point theorem [Bar88], $\mathbb{T}$ has a unique fixed point $\mathcal{A}$. This fixed point is named the attractor of the IFS:

$$\mathcal{A} = \mathbb{T}(\mathcal{A}) = \bigcup_{i=0}^{N-1} T_i(\mathcal{A}). \tag{1.1}$$

The theorem allows to calculate attractors in a very straightforward manner. Let us consider an example, where the IFS consists of three affine transformations on $\mathbb{R}^2$ (figure 1.1 shows an illustration):

$$T_0\begin{pmatrix}x\\y\end{pmatrix} = \begin{bmatrix}1/2 & 0\\ 0 & 1/2\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix}$$
$$T_1\begin{pmatrix}x\\y\end{pmatrix} = \begin{bmatrix}1/2 & 0\\ 0 & 1/2\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix} + \begin{bmatrix}1/2\\ 0\end{bmatrix}$$
$$T_2\begin{pmatrix}x\\y\end{pmatrix} = \begin{bmatrix}1/2 & 0\\ 0 & 1/2\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix} + \begin{bmatrix}0\\ 1/2\end{bmatrix}$$

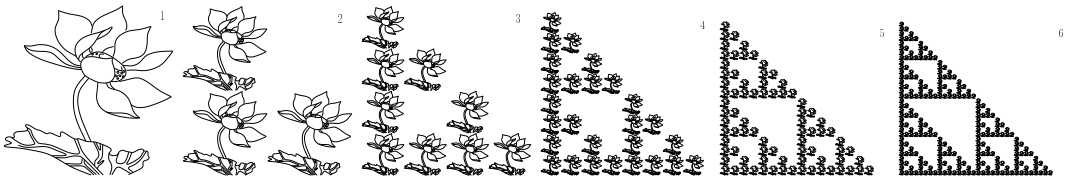


Figure 1.1: An illustration for the deterministic algorithm. Starting from any non-empty compact subset (the flower, for example), the sequence converges to the Sierpiński's Gasket.

The first step is to choose an arbitrary non-empty compact subset of $K \in \mathcal{H}(\mathbb{R}^2)$. The second step is to apply T_0, T_1 and T_2 to the current set, and then to take the union of the resulting sets. The transformation T_0 scales the input image size down, T_1 scales down and translates along x-axis, and T_2 scales down plus translates along y-axis. After merging three images $T_0(K)$, $T_1(K)$ and $T_2(K)$ we obtain second image of figure 1.1.

Therefore, the iteration process consists of applying each transformation T_0, T_1 and T_2 to the current set, and then taking the union of the resulting sets. Then T_0, T_1 and T_2 are to be applied to the union and so on. The limit of this process gives the attractor of the IFS, as shown above. As one can see, in few steps only the shape of the flower disappears, the attractor (the Sierpiński's Gasket in this case) does not depend on the initial K.

More formally, since the Hutchinson operator is contractive, the attractor of an IFS can be evaluated recursively. That is, the attractor can be approximated by a sequence $\{K_n\}_{n\in\mathbb{N}}$ converging to $\mathcal{A}$. The initial element in the sequence is defined by means of a primitive $K \in \mathcal{H}(\mathbb{X})$. The following elements are defined recursively:

$$K_0 = K$$
$$K_{n+1} = \bigcup_{i\in\Sigma} T_i(K_n).$$

The elements K_n are images of composite functions applied to K.

Each element in the sequence represents an approximation of the IFS attractor. Each term K_n is composed of N^n images of K by a composite of n functions. For example, a sequence of the attractor approximations for an IFS $\{\mathbb{X}; T_0, T_1\}$ is presented here:

$$\begin{aligned}
K_0 &= K,\\
K_1 &= \mathbb{T}(K_0) &&= T_0(K)\cup T_1(K),\\
K_2 &= \mathbb{T}(K_1) &&= T_0T_0(K)\cup T_0T_1(K)\cup T_1T_0(K)\cup T_1T_1(K),\\
K_3 &= \mathbb{T}(K_2) &&= T_0T_0T_0(K)\cup T_0T_0T_1(K)\cup T_0T_1T_0(K)\cup T_0T_1T_1(K)\cup\\
&&&\quad T_1T_0T_0(K)\cup T_1T_0T_1(K)\cup T_1T_1T_0(K)\cup T_1T_1T_1(K),\\
&\vdots &&\quad\vdots\\
K_n &= \mathbb{T}(K_{n-1}) &&= \bigcup_{\alpha_i\in\{0,1\}} T_{\alpha_1}\dots T_{\alpha_n}(K_n).
\end{aligned}$$

In this iterative algorithm a set of transformed primitives K is constructed recursively and calculations can be represented by an *evaluation tree*. Each node on the i-th level of the tree corresponds to the image of a composite of i IFS transformations. This tree is traversed up to a given depth n, where we display the image of K by the composite function associated with the current node, as shown in figure 1.2.

Note that these composite functions are calculated from left to right. The primitive K is transformed finally by a constructed composite function. In practice, the IFS transformations T_i are

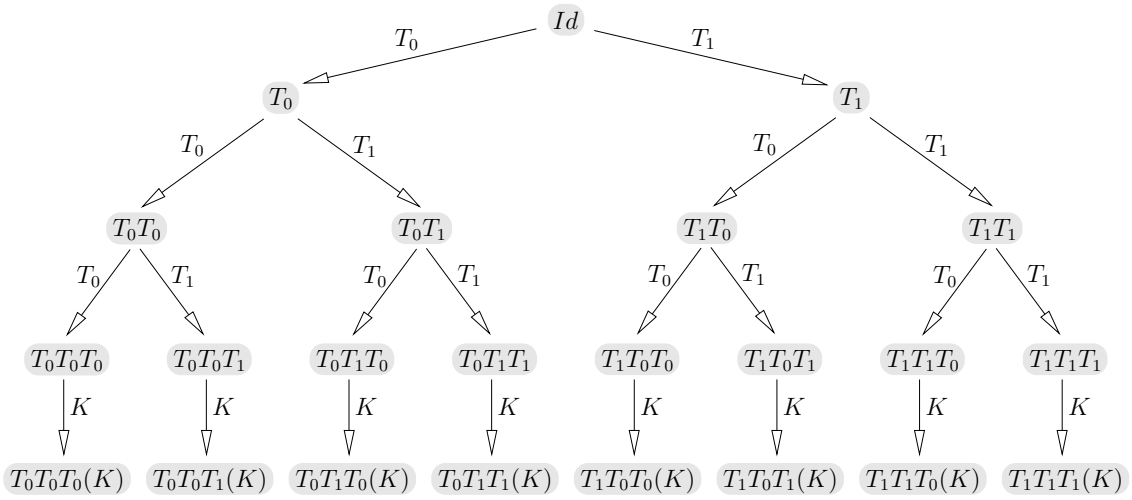


Figure 1.2: The IFS evaluation tree calculated to the third level. Internal nodes correspond to the calculation of a composite function. Leaves correspond to subsets of K_3^a to construct or to visualize.

affine operators and can therefore be represented by matrices. A composite affine transformation can thus be represented by a product of transformation matrices.

1.1.2 Controlled / Language Restricted IFS

In IFS all the transformations are applied on each iteration. It is possible to enrich this model by adding rules to control the iterations. This is the principle of a *CIFS* (Controlled IFS). CIFS are more general systems allowing us to control certain parts of the IFS attractor. A CIFS denotes an IFS with restrictions on transformation sequences imposed by a control graph. This system is similar to "Recurrent IFS" (RIFS) [Bar88], and is also described [PH94, TT93] by means of formal languages, called LRIFS (Language-Restricted Iterated Function System). CIFS defines objects whose geometry can be complex. However CIFS attractors are more convenient and controllable for manufacturability purposes than IFS attractors.

The attractor of a CIFS can be evaluated by an automaton [MW88] defined on the control graph. Each validated word of the automaton corresponds to an authorized composition of transformations. Each state of the automaton corresponds to different parts of the modeled object. States are associated with construction spaces. Transitions between states indicate that one subpart is contained in another one. It is then possible to control the attractor more precisely.

Definition. A CIFS is given by an automaton, where each state q is associated with an attractor $\mathcal{A}^q \in \mathbb{X}^q$, and each transition from q to w is associated with an operator $\mathbb{X}^w \to \mathbb{X}^q$. The following is a list of parameters describing the CIFS:

- An automaton (Σ, Q, δ), where Σ is an alphabet, Q is a set of states and δ is a transition function $\delta : Q \times \Sigma \to Q$;
- A set of complete metric spaces associated with the automaton states $\{\mathbb{X}^q\}_{q \in Q}$;
- An operator associated with each transition $T_i^q : \mathbb{X}^{\delta(q,i)} \to \mathbb{X}^q$;
- A compact set $K^q \in \mathcal{H}(\mathbb{X}^q)$, called a *primitive*, associated with each state $q \in Q$. Primitives are not used to define the attractor, but only to approximate it;
- Finally, the automaton is provided by an initial state, noted by $\natural$, and all states are final states.

In the following, we denote by Σ^q the restriction of Σ by outgoing transitions from the state q, i.e.:

$$\Sigma^q = \{i \in \Sigma,\ \delta(q, i) \in Q\}$$

CIFS defines a family of attractors associated with the states: $\{\mathcal{A}^q\}_{q\in Q}$, where $\mathcal{A}^q \in \mathcal{H}(\mathbb{X}^q)$. The attractors $\mathcal{A}^q$ are mutually defined recursively:

$$\mathcal{A}^q = \bigcup_{i\in\Sigma^q} T_i^q(\mathcal{A}^{\delta(q,i)}) \tag{1.2}$$

As for IFS, each CIFS attractor can be approximated by a sequence $\{K_n^q\}_{n\in\mathbb{N}}$ converging to $\mathcal{A}^q$. Each state $q \in Q$ is associated with a primitive $K^q \in \mathcal{H}(\mathbb{X}^q)$, which defines the initial element in the sequence. The following elements are mutually defined recursively:

$$\begin{array}{lcl} K_0^q & = & K^q \\ K_{n+1}^q & = & \bigcup_{i\in\Sigma^q} T_i^q(K_n^{\delta(q,i)}) \end{array}$$

In this iterative algorithm, a set of transformed primitives K^q is recursively constructed and the calculations can also be represented by an *evaluation tree*. Each node on the i-th level of the tree corresponds to the image of a composition of i CIFS transformations, i.e. a path of length i in the automaton. This tree is traversed up to a given depth n, where we display the image of K^q by the composite function associated with the current node.

Example 1

Consider an example of the CIFS attractor, illustrated in the right-hand image in figure 1.3. This was introduced by Bandt and Gummelt [BG97]. The system is described by an automaton with two states: a and b. There are two subdividing operators from state a as well as from b. The left panel in figure 1.3 shows the automaton of this CIFS.

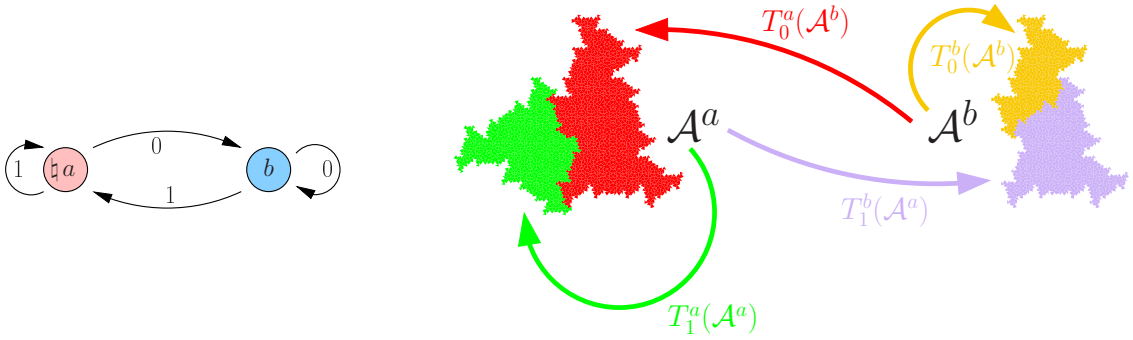


Figure 1.3: Fractal kite and dart for Penrose tilings. The CIFS automaton is shown on the left, the attractors with the transformations are shown on the right.

The automaton transition functions are the following:

$$\begin{array}{ll} \delta(a,0) = b & \delta(b,0) = b \\ \delta(a,1) = a & \delta(b,1) = a \end{array}$$

Each transition $\delta(q,i) = w$ of the automaton is associated with an operator $T_i^q : \mathbb{X}^w \to \mathbb{X}^q$. **N.B.:** *$T$ and δ act in opposite directions!*

In this example, $\mathbb{X}^a$ and $\mathbb{X}^b$ are both in the same Euclidean affine plane. Let us define the mappings as follows:

$$\begin{aligned} T_0^a\left(\begin{bmatrix}x\\y\end{bmatrix}\right) &= \begin{bmatrix}x\\y\end{bmatrix} \\ T_1^a\left(\begin{bmatrix}x\\y\end{bmatrix}\right) &= \frac{2}{1+\sqrt{5}}\begin{bmatrix}\cos(3/5\pi) & -\sin(3/5\pi)\\ \sin(3/5\pi) & \cos(3/5\pi)\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix} + \begin{bmatrix}0\\1\end{bmatrix} \\ T_0^b\left(\begin{bmatrix}x\\y\end{bmatrix}\right) &= \frac{2}{1+\sqrt{5}}\begin{bmatrix}\cos(4/5\pi) & -\sin(4/5\pi)\\ \sin(4/5\pi) & \cos(4/5\pi)\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix} + \begin{bmatrix}\frac{1+\sqrt{5}}{2}\sin(4/5\pi)\\ 1+\frac{1+\sqrt{5}}{2}\cos(4/5\pi)\end{bmatrix} \\ T_1^b\left(\begin{bmatrix}x\\y\end{bmatrix}\right) &= \frac{2}{1+\sqrt{5}}\begin{bmatrix}\cos(7/5\pi) & -\sin(7/5\pi)\\ \sin(7/5\pi) & \cos(7/5\pi)\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix} + \begin{bmatrix}0\\1\end{bmatrix} \end{aligned}$$

Thus, the attractors $\mathcal{A}^a$ and $\mathcal{A}^b$ satisfy the following equations:

$$\begin{aligned}\mathcal{A}^a &= \bigcup_{i\in\Sigma^a} T_i^a(\mathcal{A}^{\delta(a,i)}) &= T_1^a(\mathcal{A}^a) \cup T_0^a(\mathcal{A}^b)\\ \mathcal{A}^b &= \bigcup_{i\in\Sigma^b} T_i^b(\mathcal{A}^{\delta(b,i)}) &= T_0^b(\mathcal{A}^b) \ \cup \ T_1^b(\mathcal{A}^a)\end{aligned}$$

Figure 1.4 shows the CIFS evaluation tree calculated to the third level. Internal nodes correspond to the calculation of a composite function, while the leaves correspond to subsets of K_3^a to construct or to visualize. The pink and blue highlights help distinguish between current state (space) a and b, respectively.

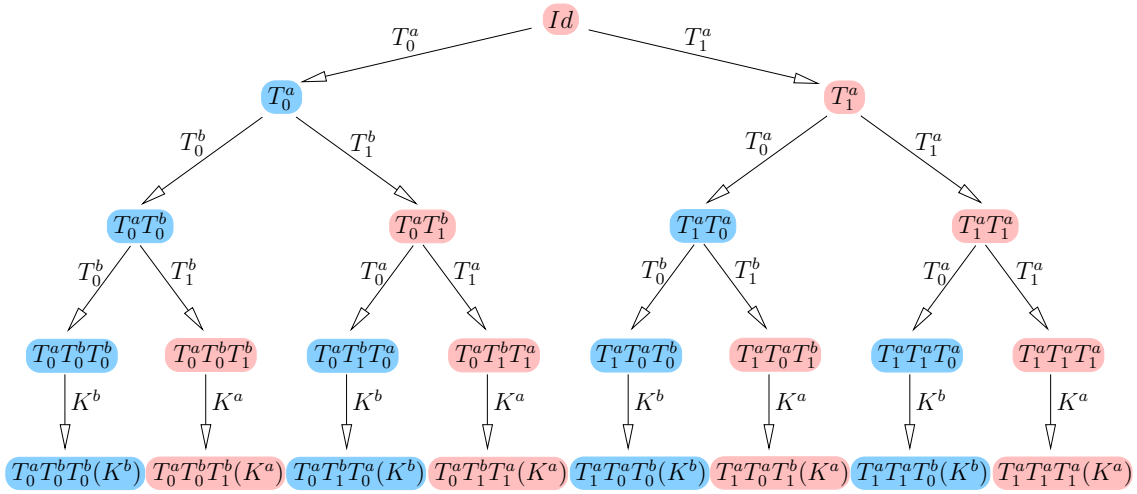


Figure 1.4: CIFS evaluation tree calculated to the third level. Internal nodes correspond to the calculation of a composite function. Leaves correspond to subsets of K_3 to construct or to visualize.

Example 2

Our second CIFS example is a simple uniform quadratic B-spline curve with 4 control points. Figure 1.5 gives the automaton (left) and the attractors with corresponding transformations (right). This is a special kind of CIFS, sometimes called Projected IFS in the literature [ZT96, GTB00].

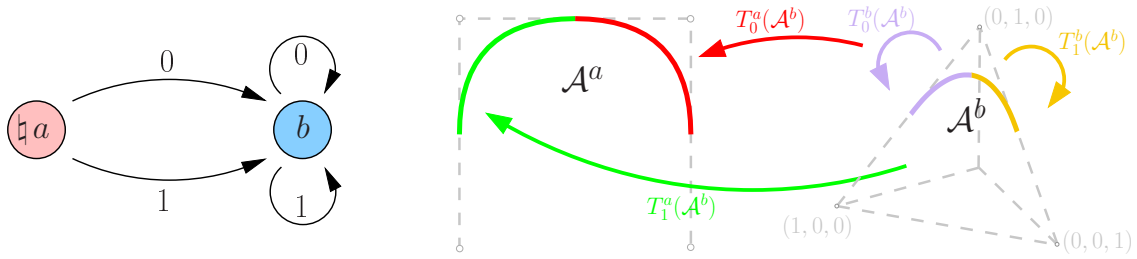


Figure 1.5: The $2D$ uniform quadratic B-spline curve can be defined as a projection of an attractor from the three-dimensional barycentric space.

The automaton transition functions are the following:

$$\begin{aligned}\delta(a,0) &= b & \qquad \delta(b,0) &= b\\ \delta(a,1) &= b & \qquad \delta(b,1) &= b\end{aligned}$$

Now the space associated with state a is still the Euclidean plane $\mathbb{R}^2$, while a three-dimensional barycentric space is associated with state b. Given the coordinates of the 4 control points P_0, P_1, P_2 and P_3, we can express the transformations as follows:

$$T_0^a\left(\begin{bmatrix}x\\y\\z\end{bmatrix}\right) = \begin{bmatrix}P_0^x & P_1^x & P_2^x\\ P_0^y & P_1^y & P_2^y\end{bmatrix}\begin{bmatrix}x\\y\\z\end{bmatrix} \qquad T_1^a\left(\begin{bmatrix}x\\y\\z\end{bmatrix}\right) = \begin{bmatrix}P_1^x & P_2^x & P_3^x\\ P_1^y & P_2^y & P_3^y\end{bmatrix}\begin{bmatrix}x\\y\\z\end{bmatrix}$$

$$T_0^b\left(\begin{bmatrix}x\\y\\z\end{bmatrix}\right)=\begin{bmatrix}3/4 & 1/4 & 0\\ 1/4 & 3/4 & 3/4\\ 0 & 0 & 1/4\end{bmatrix}\begin{bmatrix}x\\y\\z\end{bmatrix} \qquad T_1^b\left(\begin{bmatrix}x\\y\\z\end{bmatrix}\right)=\begin{bmatrix}1/4 & 0 & 0\\ 3/4 & 3/4 & 1/4\\ 0 & 1/4 & 3/4\end{bmatrix}\begin{bmatrix}x\\y\\z\end{bmatrix}$$

The transformations T_0^a and T_1^b are two projections of the same attractor representing the basis functions of the uniform quadratic B-spline illustrated in figure 1.5 (on the left side). Figure 1.6 gives the evaluation tree for the approximation K_3^a. Note that besides the 1st level of subdivision this is an ordinary IFS (all nodes are blue). The attractors of a CIFS are uniquely defined if operators associated with each cycle in the control graph are contractive. Hence, we do not have any constraints on the coordinates of the control points, as T_i^a do not appear in any cycle.

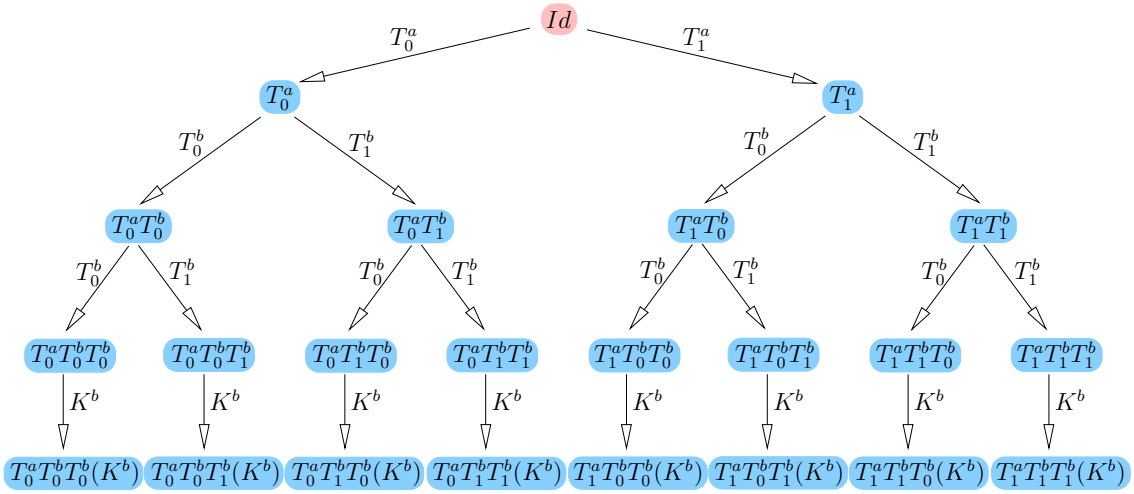


Figure 1.6: CIFS evaluation tree calculated to the third level. Note that the pattern of pink/blue nodes has changed completely from the previous example.

1.2 Boundary Controlled IFS

IFS and CIFS can model complex shapes; it is, however, difficult to control their topological properties. These shapes are determined by a set of geometry operators. Modifying these operators leads to both global and local changes in the shape and affects not only geometry but also topology. In order to control the topological structure of the modeled shape, we enrich CIFS by integrating a topological model to obtain *BCIFS* (Boundary Controlled Iterated Function System).

In standard CAD systems, topology and geometric properties of shapes are separated. The topological structure is encoded by a set of topological cells (faces, edges, vertices) interconnected by a set of incidence and adjacency relations. The incidence relations are based on the nesting of cells: each face is bounded by a set of edges, and each edge is bounded by two vertices. The adjacency relations are based on sharing cells: two adjacent faces share a common edge, and two adjacent edges are bounded by a common vertex.

Inspired by this approach, we propose to extend the CIFS model by integrating B-rep relations. BCIFS is thus an extension of a CIFS enriched by a description of topology. Sub-parts of the attractor are identified as topological cells by specifying incidence constraints. These cells are assembled during the subdivision process by adjacency constraints. These constraints induce constraints on the subdivision operators of the CIFS.

Our B-rep structure is more general than the standard one. A topological cell may be fractal. For example, a face can be the Sierpinski triangle or an edge can be the Cantor set, but the topological structure remains consistent. Each topological cell corresponds to an attractor in a certain space.

1.2.1 Specifying the topology

There are two types of transitions in the BCIFS automaton:

- transitions subdividing a topological cell;

- transitions embedding a topological cell in another one.

The alphabet Σ is also divided into:

- symbols of subdivision $\Sigma_{\div} = \{\div_i \mid i = 0, \cdots, n_{\div}\}$;
- symbols of incidence $\Sigma_{\partial} = \{\partial_i \mid i = 0, \cdots, n_{\partial}\}$.

Each subdividing transition $\delta(q, \div_i) = w$ is associated with a subdividing operator $T_i^q : \mathbb{X}^w \to \mathbb{X}^q$, where $q, w \in Q$ and $\div_i \in \Sigma_{\div}$. Similarly, each embedding transition $\delta(q, \partial_i) = w$ is associated with an embedding operator $B_i^q : \mathbb{X}^w \to \mathbb{X}^q$, where $q, w \in Q$ and $\partial_i \in \Sigma_{\partial}$.

Example

Let us illustrate the idea with an example. In this section we generate a continuous patch of a free-form surface with 9 control points. This patch will be defined as the attractor $\mathcal{A}^f$ of the IFS $(\mathbb{X}^f; T_0^f, T_1^f, T_2^f, T_3^f)$, let us call it "facet". We construct it as a B-rep structure with "edges" corresponding to the attractor $\mathcal{A}^e$ of the IFS $(\mathbb{X}^e; T_0^e, T_1^e)$ and "vertices" corresponding to the attractor $\mathcal{A}^v$ of the IFS $(\mathbb{X}^v; T_0^v)$.

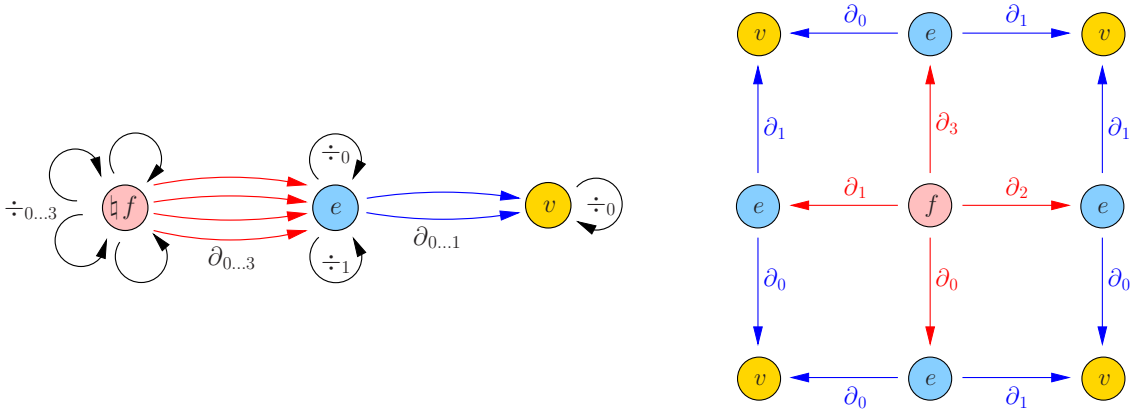


Figure 1.7: Left image: automaton representing a quad patch defined by its boundaries. Right image: expanded incidence relations of the automaton .

Figure 1.7 gives the corresponding automaton. Note that the automaton has a hierarchical structure: there are three separate IFS linked by incidence operators. We omit the evident step of projecting the attractors into the modeling space. Refer to figure 1.5 to see how the projection is carried out in general.

Incidence constraints

We start with the definition of the vertex attractor $\mathcal{A}^v$. To keep the example simple, we choose $\mathbb{X}^v$ to be a 1-dimensional barycentric space and T_0^v is simply a 1×1 identity matrix. The edge attractor will be defined in a 3-dimensional barycentric space.

We choose the inclusion of $\mathcal{A}^v$ (recall that it is just a point) inside the attractor $\mathcal{A}^e$, it defines boundaries of the edge $\mathcal{A}^e$. Let us say we want the vertex to be included twice at the coordinates $(1, 0, 0)$ and $(0, 0, 1)$. That is to say, we need certain constraints on the matrices T_0^e and T_1^e to force points $(1, 0, 0)$ and $(0, 0, 1)$ to belong to the attractor $\mathcal{A}^e$.

Let us assume $B_0^e = \begin{bmatrix}1 & 0 & 0\end{bmatrix}^\top$ and $B_1^e = \begin{bmatrix}0 & 0 & 1\end{bmatrix}^\top$. Now we can express first incidence constraints as

$$B_0^e T_0^v = T_0^e B_0^e$$
$$B_1^e T_0^v = T_1^e B_0^e$$

These constraints impose a structure of the matrices T_0^e and T_1^e:

$$T_0^e = \begin{bmatrix}1 & \cdot & \cdot \\ 0 & \cdot & \cdot \\ 0 & \cdot & \cdot\end{bmatrix} \qquad T_1^e = \begin{bmatrix}\cdot & \cdot & 0 \\ \cdot & \cdot & 0 \\ \cdot & \cdot & 1\end{bmatrix},$$

where dots stand for arbitrarily chosen reals.

The subset of $\mathcal{A}^e$ defined as an attractor of the IFS $(\mathbb{X}^e; T_0^e)$ is equal to the $B_0^e\mathcal{A}^v$, the attractor $\mathcal{A}^v$ embedded by the action of B_0^e. In the same manner, $(0,0,1)$ is the fixed point of T_1^e and thus contained in the $\mathcal{A}^e$. Note that the first constraint implies that T_0^e must have all eigenvectors of T_0^v transformed by the action of the embedding operator B_0^e.

Property. More generally, let us show that the incidence constraints force the inclusion of the boundary CIFS attractors into the corresponding cell. If a cell $\mathcal{A}^Y$ has a number of boundaries defined by attractors $\mathcal{A}^{X_i}$, then we want to show that each $\mathcal{A}^{X_i}$ (when embedded in the space associated with state Y) is a sub part of $\mathcal{A}^Y$: in other words, we want to show that the inclusion $B_i^Y\mathcal{A}^{X_i} \subset \mathcal{A}^Y$ holds.

The incidence constraints have the following expression:

$$B_i^Y T_j^{X_i} = T_{f(i,j)}^Y B_{g(i,j)}^Y,$$

with $i \in \Sigma_\partial^Y$ and $j \in \Sigma_\div^{X_i}$. The functions f and g are simply the corresponding ordering of the boundaries and subdivisions. For example, for a square patch with four boundaries, each subdivided by two operators, we have $|\Sigma_\partial^Y| \times |\Sigma_\div^{X_i}| = 4 \times 2$ constraints.

For each boundary embedding we can write the following:

$$B_i^Y\mathcal{A}^{X_i} = B_i^Y \bigcup_{j\in\Sigma_\div^{X_i}} T_j^{X_i}\mathcal{A}^{X_i} = \bigcup_{j\in\Sigma_\div^{X_i}} B_i^Y T_j^{X_i}\mathcal{A}^{X_i} = \bigcup_{j\in\Sigma_\div^{X_i}} T_{f(i,j)}^Y B_{g(i,j)}^Y\mathcal{A}^{X_i}.$$

This means that the boundary $B_i^Y\mathcal{A}^{X_i}$ can be obtained as a union of other boundaries $B_{g(i,j)}^Y\mathcal{A}^{X_i}$ under the action of the subdivision operators. We can repeat this process ad infinitum. This means that by restricting the generated language, every boundary $B_i^Y\mathcal{A}^{X_i}$ can be generated solely by operators T^Y and therefore the inclusion $B_i^Y\mathcal{A}^{X_i} \subset \mathcal{A}^Y$ holds.

We can choose any real values for the dots in the expression of T_0^e and T_1^e, the attractor $\mathcal{A}^e$ will include two points $(1,0,0)$ and $(0,0,1)$. Note that at this point the attractor $\mathcal{A}^e$ can be a disjoint set of points. In the following subsection we add an adjacency constraint that will enable the attractor $\mathcal{A}^e$ to be a continuous curve. Figure 1.8 provides an example of the attractor $\mathcal{A}^e$ with the subdivision operators chosen as follows:

$$T_0^e = \begin{bmatrix} 1 & 1/2 & 1/4 \\ 0 & 1/2 & 1/2 \\ 0 & 0 & 1/4 \end{bmatrix} \qquad T_1^e = \begin{bmatrix} 1/2 & 0 & 0 \\ 1/4 & 1/2 & 0 \\ 1/4 & 1/2 & 1 \end{bmatrix},$$

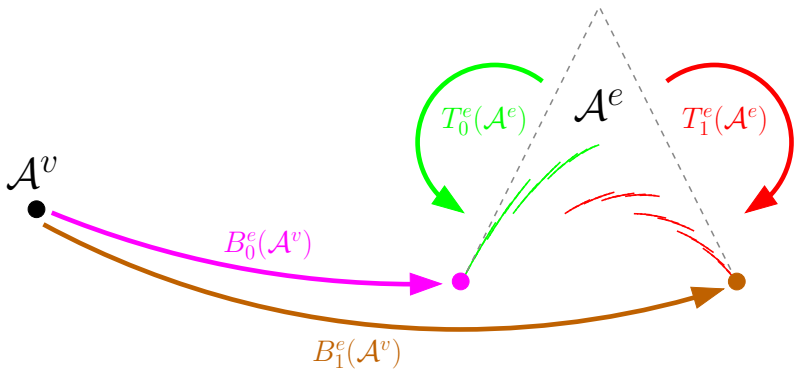


Figure 1.8: Incidence constraints ensure the inclusion $B_0^e\mathcal{A}^v \subset \mathcal{A}^e$ and $B_1^e\mathcal{A}^v \subset \mathcal{A}^e$, however they do not guarantee the connectivity of the attractor $\mathcal{A}^e$.

Let us define edge-to-facet embedding operators:

$$B_0^f = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^\top \qquad B_1^f = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}^\top$$

$$B_2^f = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}^\top \qquad B_3^f = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}^\top$$

and the corresponding incidence constraints:

$$\begin{aligned} B_0^f T_0^e &= T_0^f B_0^f & B_0^f T_1^e &= T_1^f B_0^f \\ B_3^f T_0^e &= T_2^f B_3^f & B_3^f T_1^e &= T_3^f B_3^f \\ B_1^f T_0^e &= T_0^f B_1^f & B_1^f T_1^e &= T_2^f B_1^f \\ B_2^f T_0^e &= T_1^f B_2^f & B_2^f T_1^e &= T_3^f B_2^f. \end{aligned}$$

This particular form of B_0^f simply signifies that the corresponding edge depends on the first three control points (out of nine total). If we take the first pair of constraints only, it ensures that the attractor of the IFS $(\mathbb{X}^f; T_0^f, T_1^f)$ (recall that it is a sub-attractor of $\mathcal{A}^f$) is an image of the edge $\mathcal{A}^e$ embedded by the action of B_0^f. In the same manner, three other pairs of constraints ensure that $\mathcal{A}^f$ contains edges $B_1^f\mathcal{A}^e$, $B_2^f\mathcal{A}^e$ and $B_3^f\mathcal{A}^e$. Figure 1.9 shows an example of $\mathcal{A}^f$ with randomly fixed degrees of freedom.

Adjacency constraints

Here we add the adjacency constraints that enforce connection of corresponding attractors. Recall that our attractors are self-similar, so after a subdivision one smaller copy of the attractor will be adjacent to another smaller copy.

Figure 1.10 illustrates the idea. Attractor $\mathcal{A}^e$ is defined as a union of its subdivisions $\mathcal{A}^e = T_0^e(\mathcal{A}^e) \cup T_1^e(\mathcal{A}^e)$. Let us apply the following constraint:

$$T_0^e B_1^e = T_1^e B_0^e.$$

This signifies that $T_0^e(\mathcal{A}^e)$ and $T_1^e(\mathcal{A}^e)$ must share a common vertex thus producing a connected attractor $\mathcal{A}^e$. Let us express the corresponding matrices explicitly:

$$T_0^e = \begin{bmatrix} 1 & a_0 & b_0 \\ 0 & a_1 & b_1 \\ 0 & 1-a_0-a_1 & 1-b_0-b_1 \end{bmatrix} \qquad T_1^e = \begin{bmatrix} b_0 & c_0 & 0 \\ b_1 & c_1 & 0 \\ 1-b_0-b_1 & 1-c_0-c_1 & 1 \end{bmatrix},$$

We have 6 degrees of freedom left in the matrices; any choice of the coefficients ensures the connectivity of the attractor of the IFS $(\mathbb{X}^e; T_0^e, T_1^e)$.

In exactly the same manner, we apply the adjacency constraints for the facet subdivision operators:

$$\begin{aligned} T_0^f B_2^f &= T_1^f B_1^f \\ T_2^f B_2^f &= T_3^f B_1^f \\ T_0^f B_3^f &= T_2^f B_0^f \\ T_1^f B_3^f &= T_3^f B_0^f \end{aligned}$$

Figure 1.11 provides an illustration. We omit an explicit expression of $T_{0\ldots3}^f$ here, since it is cumbersome but straightforward to obtain.

At this point (after applying incidence and adjacency constraints) the attractor of the IFS $(\mathbb{X}^f; T_0^f, T_1^f, T_2^f, T_3^f)$ is guaranteed to have the topology of a quad patch. The degrees of freedom left in the operators can only affect the geometric texture.

For instance, we can fix the coefficients of $T_{0\ldots3}^f$ to produce a bi-quadratic Bézier patch (left image in figure 1.12). But even randomly chosen coefficients produce a continuous surface (right image in figure 1.12).

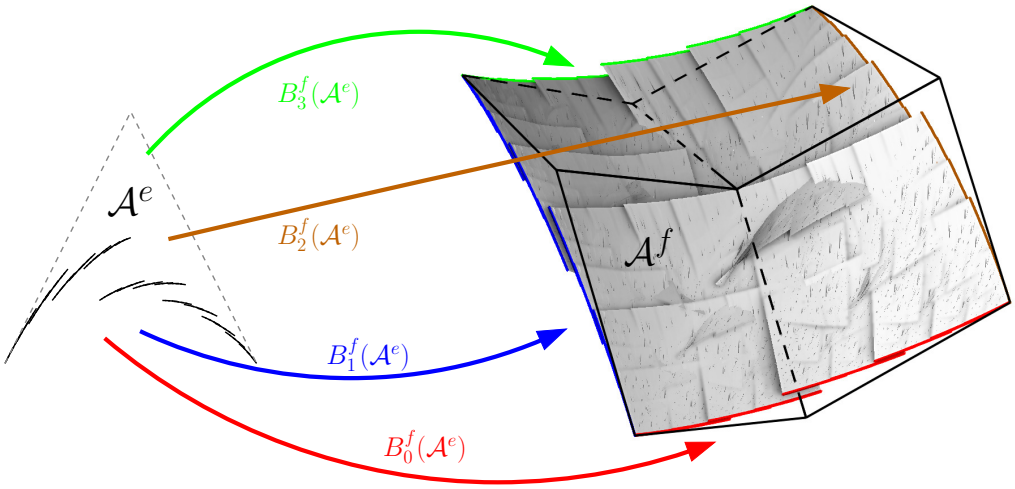


Figure 1.9: As for the edge, facet incidence constraints ensure the inclusion $B_0^f\mathcal{A}^e \subset \mathcal{A}^f$, $B_1^f\mathcal{A}^e \subset \mathcal{A}^f$, $B_2^f\mathcal{A}^e \subset \mathcal{A}^f$ and $B_3^f\mathcal{A}^e \subset \mathcal{A}^f$.

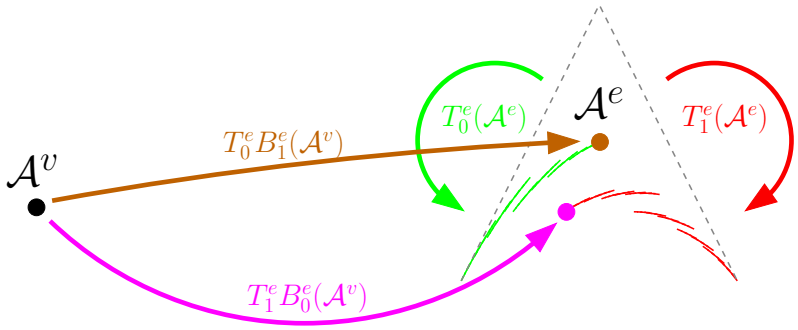


Figure 1.10: In order to obtain a continuous curve, it suffices to apply the constraint $T_0^eB_1^e = T_1^eB_0^e$.

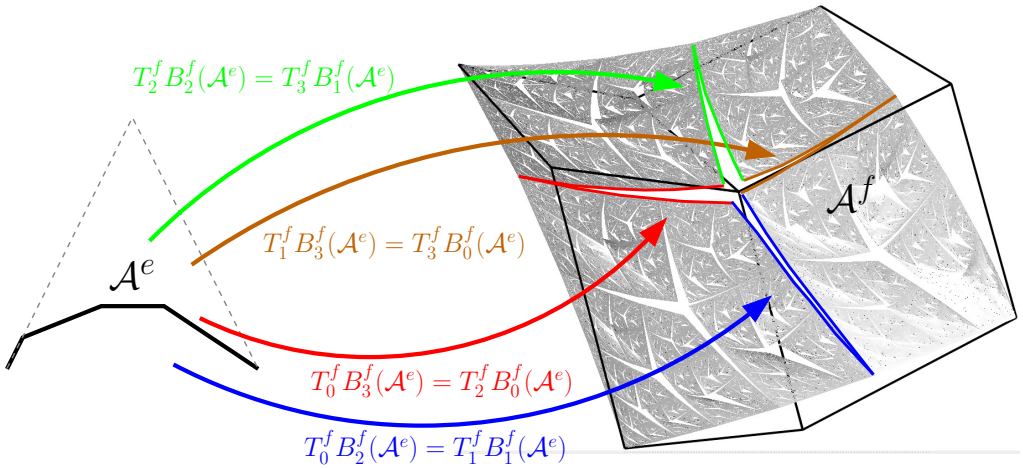


Figure 1.11: Non-respect of the adjacency constraints leads to a disconnected patch.

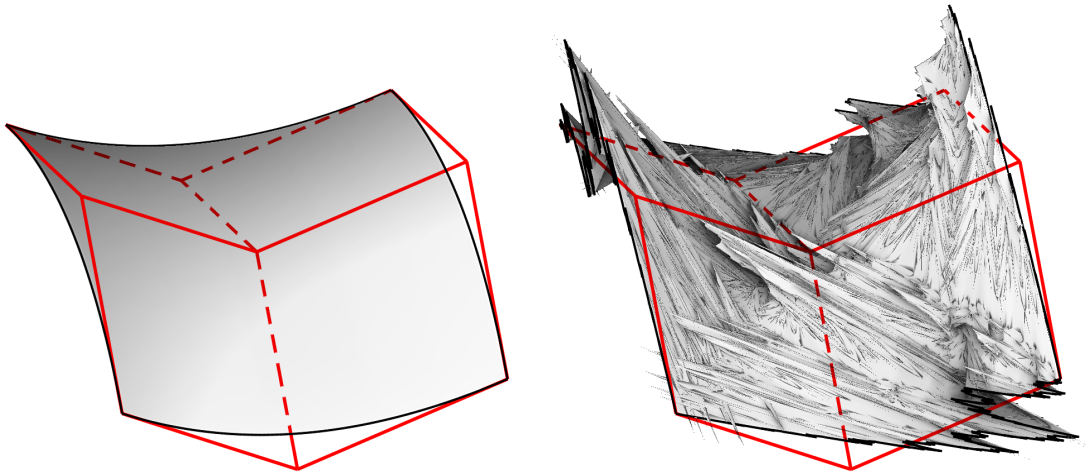


Figure 1.12: After applying incidence and adjacency constraints, the attractor of the IFS $(\mathbb{X}^f; T_0^f, T_1^f, T_2^f, T_3^f)$ is guaranteed to have the topology of a quad patch. The degrees of freedom left in the operators can only affect the geometric texture. Left image: the degrees of freedom were fixed to produce a bi-quadratic Bézier patch; right image: even randomly chosen coefficients produce a continuous patch.

1.2.2 Controlling the geometric texture

Full analysis of the differential behavior of produced shapes is beyond the scope of this section, but in this section we try to illustrate how the BCIFS model can be used to control geometry (in addition to topology). Refer to chapter 2 for more details on the subject.

Let us construct an edge bounded by two vertices, each depending on one control point. Each vertex can be represented by the same state v, the only one possible for a one dimensional space with its trivial subdivision operator : $T_0^v = \begin{bmatrix}1\end{bmatrix}$. Figure 1.13 provides the automaton.

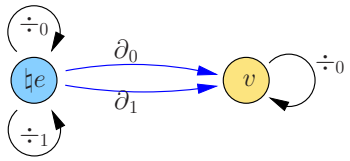


Figure 1.13: In this example the edge $\mathcal{A}^e$ is bounded by two different vertices $\mathcal{A}^{v_0}$ and $\mathcal{A}^{v_1}$.

Let us assume $B_0^e = \begin{bmatrix}1 & 0 & 0\end{bmatrix}^\top$ and $B_1^e = \begin{bmatrix}0 & 0 & 1\end{bmatrix}^\top$ and the usual incidence and adjacency constraints:

$$\begin{aligned} B_0^e T_0^{v_0} &= T_0^e B_0^e \\ B_1^e T_0^{v_1} &= T_1^e B_1^e \\ T_0^e B_1^e &= T_1^e B_0^e. \end{aligned}$$

Solving for the incidence and adjacency constraints we obtain:

$$T_0^e = \begin{bmatrix}1 & \cdot & a\\ 0 & \cdot & b\\ 0 & \cdot & c\end{bmatrix} \quad T_1^e = \begin{bmatrix}a & \cdot & 0\\ b & \cdot & 0\\ c & \cdot & 1\end{bmatrix} \text{ with } a+b+c=1$$

In order to control the differential behavior at each vertex, we define two additional states, denoted by d_0 and d_1, with a two dimensional barycentric space associated with each one, and two embedding operators, $B_0^{e\prime} = \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\end{bmatrix}^\top$ and $B_1^{e\prime} = \begin{bmatrix}0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}^\top$, specifying which control points are implied for each differential behavior. Each state has its own subdivision operator, respectively $T_0^{d_0}$ and $T_0^{d_1}$.

As for C_0 continuity, we use incidence constraints for the differential continuity.

$$B_0^{e\prime} T_0^{d_0} = T_0^e B_0^{e\prime}$$
$$B_1^{e\prime} T_0^{d_1} = T_1^e B_1^{e\prime}.$$

The consequence is:

$$T_0^{d_0} = \begin{bmatrix} 1 & 1-\lambda \\ 0 & \lambda \end{bmatrix} \qquad T_0^{d_1} = \begin{bmatrix} \mu & 0 \\ 1-\mu & 1 \end{bmatrix}$$
$$T_0^e = \begin{bmatrix} 1 & 1-\lambda & a \\ 0 & \lambda & b \\ 0 & 0 & c \end{bmatrix} \qquad T_1^e = \begin{bmatrix} a & 0 & 0 \\ b & 1-\mu & 0 \\ c & \mu & 1 \end{bmatrix}$$

Attractors of $\mathcal{A}^{d_0}$ and $\mathcal{A}^{d_1}$ are points with coordinates given by the dominant eigenvectors of $T_0^{d_0}$ and $T_0^{d_1}$. Hence we know that $\mathcal{A}^{d_0}$ is a point with coordinates $(1,0)$ and $\mathcal{A}^{d_1}$ is a point with coordinates $(0,1)$ in corresponding barycentric spaces.

Recall that incidence constraints embed all eigenvectors (and eigenvalues) of $T_0^{d_0}$ and $T_0^{d_1}$ into T_0^e and T_1^e. Therefore, if λ is a sub-dominant eigenvalue of T_0^e, then the half-tangent at the "left" endpoint of the curve $\mathcal{A}^e$ is the vector $(-1,1,0)$ (the first edge of the control polygon). In the same manner, if μ is sub-dominant in T_1^e, then the "right" half-tangent is the vector $(0,-1,1)$, the 2nd edge of the control polygon.

As for the C^0 adjacency constraint, we can apply the same constraint for the half-tangents: [1]

$$T_0^e B_1^{e\prime} \begin{bmatrix} -1 \\ 1 \end{bmatrix} = T_1^e B_0^{e\prime} \begin{bmatrix} -1 \\ 1 \end{bmatrix} \Rightarrow T_0^e \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix} = T_1^e \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix}.$$

Solving the incidence and adjacency constraints we obtain the following expression for the subdivision operators:

$$T_0^e = \begin{bmatrix} 1 & 1-\lambda & \frac{1-\lambda}{2} \\ 0 & \lambda & \frac{\lambda+\mu}{2} \\ 0 & 0 & \frac{1-\mu}{2} \end{bmatrix} \qquad T_1^e = \begin{bmatrix} \frac{1-\lambda}{2} & 0 & 0 \\ \frac{\lambda+\mu}{2} & \mu & 0 \\ \frac{1-\mu}{2} & 1-\mu & 1 \end{bmatrix}.$$

Provided that λ and μ are positive sub-dominant eigenvalues ($\lambda \geq \frac{1-\mu}{2}$ and $\mu \geq \frac{1-\lambda}{2}$), the attractor $\mathcal{A}^e$ is guaranteed to be a C^1 curve.

1.2.3 Example

Given a set of control points and a subdivision method, construction of a CIFS whose attractor is exactly the limit subdivision surface is straightforward. In this example, we push the concept a bit further. We want to construct a solid arborescent structure, whose boundary is a subdivision surface.

Figure 1.14 shows the core of the subdivision process. The final arborescent structure $\mathcal{A}^a$ is defined as an iterative condensation of the attractor $\mathcal{A}^b$. Here $\mathcal{A}^b$ is a limit subdivision surface for a given mesh. The idea is simple: $\mathcal{A}^a$ is defined layer-by-layer. $\mathcal{A}^b$ covers the top of the shape, then the second layer is defined by four smaller copies of $\mathcal{A}^b$, the third layer has 16 copies of $\mathcal{A}^b$ and so forth.

Both $\mathcal{A}^a$ and $\mathcal{A}^b$ have six facets; figure 1.15 shows the constraints we obtain on facets from the nature of the subdivision process. We are free to choose two facets (one upper and one lateral) for the attractor $\mathcal{A}^b$, the other four are fixed automatically by our choice.

Finally, figure 1.16 gives the final shape of the attractor $\mathcal{A}^a$.

[1] Strictly speaking, we do not need the equality of the half-tangents, collinearity suffices, so the adjacency constraint can be written as $T_0^e B_1^{e\prime} \begin{bmatrix} -\alpha \\ \alpha \end{bmatrix} = T_1^e B_0^{e\prime} \begin{bmatrix} -1 \\ 1 \end{bmatrix}$ for some $\alpha \neq 0$.

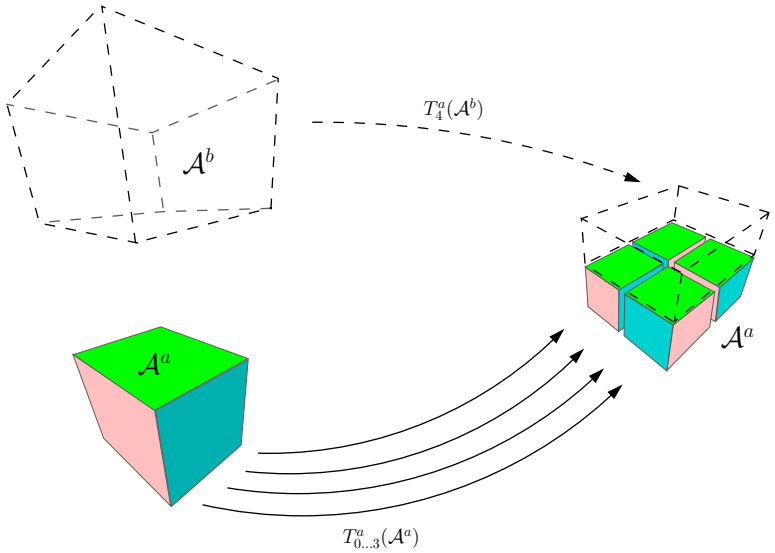


Figure 1.14: Arborescent structure of $\mathcal{A}^a$ is defined by iterative condensation of the attractor $\mathcal{A}^b$.

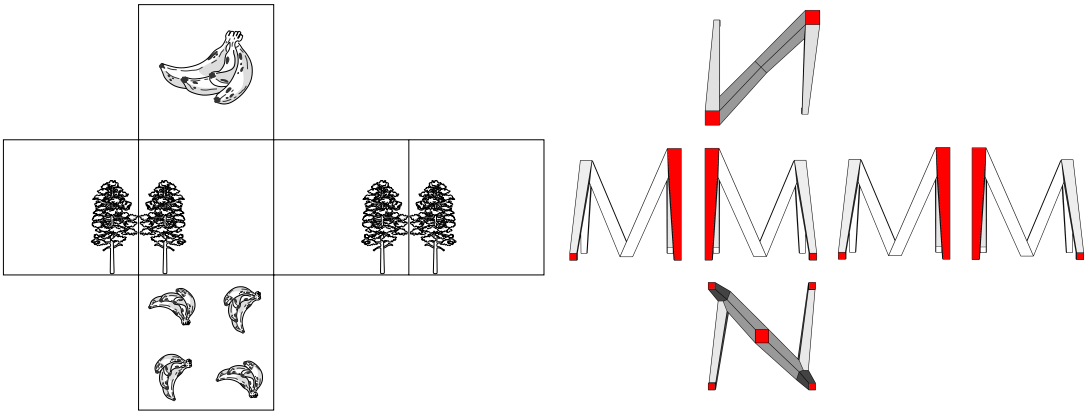


Figure 1.15: Left: unfolding of the six facets of $\mathcal{A}^b$. We are free to choose two of them; the other four are fixed automatically by our choice. Right: our choice.

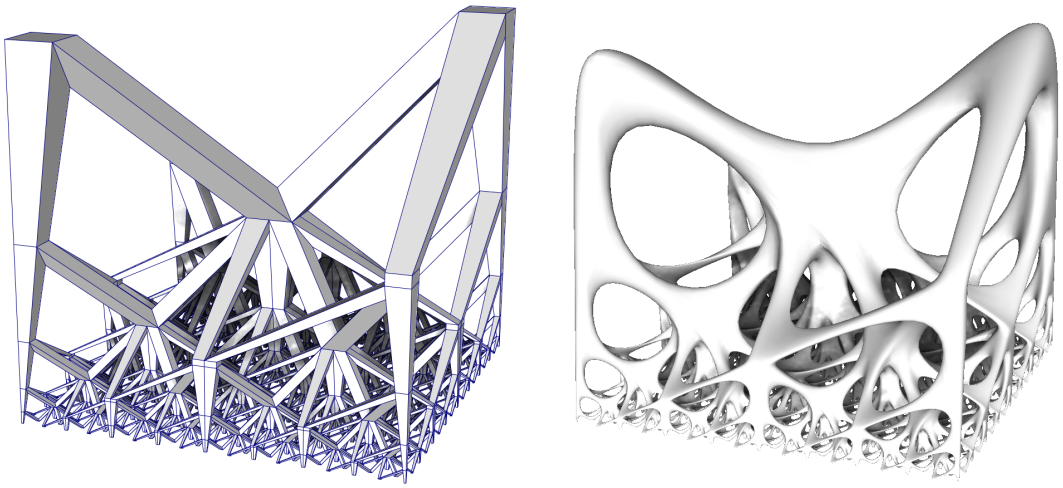


Figure 1.16: Left: mesh of control points for the $\mathcal{A}^a$, right: the final shape of $\mathcal{A}^a$.

1.3 Subdivision curves and surfaces as BC-IFS

While BC-IFS were motivated by a new universe of fractal shapes, it can perfectly reproduce standard subdivision techniques. This section shows how to model famous curves and surfaces with our B-rep Iterated Function Systems.

1.3.1 Regular patch

The quad patch example we have shown in figure 1.7 can be seen as a subdivision surface. Let us show a correspondence between, say, a Doo-Sabin regular patch and the example we have shown above.

A subdivision surface is a curved surface formed by recursively subdividing a regular grid of control points until it approximates a smooth surface. The subdivision process usually generates points on the basis of some spline function. A subdivision surface is thus like a spline patch, but has more flexibility due to the fact that the original grid need not be rectangular. An example of a subdivision scheme is the Doo-Sabin method. For the regular cases the method produces uniform bi-quadratic B-spline surfaces.

The procedure is similar to the subdivision of curves: having selected a 3×3 patch of 9 original control points one produces 16 new points, refer to figure 1.17 for an illustration. The original grid (without loss of generality we start with the smallest grid for a bi-quadratic B-spline surface) is drawn in red color while the refined one is drawn in green color. Then the process is restarted with each of 3×3 sub-patches independently, figure 1.18 shows the parallelized tasks in blue. Each of the refined grid patches (blue) in figure 1.18 can be seen as the original grid (red) under the action of (left-to-right) T_0^f, T_1^f, T_2^f and T_3^f.

Boundary operators select six control points out of nine total. Then the incidence constraint $B_1^f T_0^e = T_2^f B_1^f$ can be illustrated with the left half of figure 1.19, compare it with the figure 1.9. The adjacency constraint $T_0^f B_3^f = T_2^f B_0^f$ is illustrated in the right half of figure 1.19, compare it with the figure 1.11. These conditions impose the borders of the sub-patches to coincide. And, finally, we will "import" three conditions from the previous section to force the borders be curves and not, say, Cantor sets.

Imposing these constraints of the matrices T_i^f leave only three degrees of freedom. Figure 1.20 shows an illustration. The leftmost image is the control grid for the subdivision surface, the middle image is the surface with Doo-Sabin coefficients $\{.5625, .1875, .0625\}$ and the rightmost image is the surface with arbitrarily chosen coefficients $\{.09, .21, .49\}$. Note that the attractor of the IFS is always a surface, even if all three variables are taken randomly.

1.3.2 Extraordinary vertices

Of course, in real life it is impossible to use regular grids only, let us discuss the relationship between Iterated Function Systems and subdivision surfaces with extraordinary points. After one round of subdivision the Doo-Sabin scheme produces a mesh whose vertices all have the same degree 4, and most faces are rectangular, except for faces arising from original vertices of degree not equal to four and from non-rectangular faces. Figure 1.21 shows an example of the Doo-Sabin scheme on a "suitcase corner" mesh.

It is a very interesting fact that after one round of subdivision, the number of non-rectangular faces remains constant, and it turns out that these faces shrink and tend to a limit which is their common centroid. The centroid of each non-rectangular faces is referred to as an extraordinary point. Figure 1.22 illustrates the property. The original "suitcase corner" patch has 8 control points, whereas the regular patch has 9 points. The first step of the subdivision generates 15 new points. Then one selects three regular (with 9 points) patches and an extraordinary (with 8 points) patch and subdivides them independently. Therefore, the number of extraordinary points is preserved, since no extraordinary points could emerge from regular patches.

Let us build an IFS whose attractor is the limit surface of the Doo-Sabin subdivision scheme. To clarify the construction we start with extraordinary curves. The lower boundary of the "suitcase corner" depends on the control points shown in figure 1.23. Iterating on the control points, one gets a shrinking grid that tends to a curve. The original (red) grid has 5 vertices and the regular Doo-Sabin boundary grid must contain 6 vertices. At the second level of the process we have two different patches (in blue): an extraordinary and a regular one. Since they have different number of control points, the corresponding spaces differ in number of dimensions and we have to create

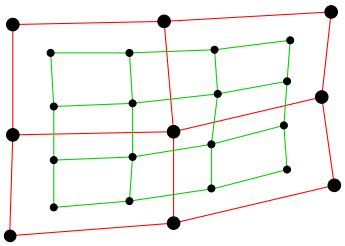


Figure 1.17: Doo-Sabin subdivision, the original grid and the refined one (after one subdivision step).

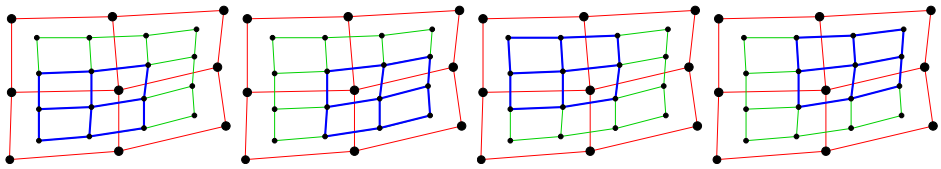


Figure 1.18: The second step of the Doo-Sabin subdivision process: one selects four 3×3 patches and refines them independently.

Figure 1.19: Two samples of connectivity constraints. On the left: incidence constraint $B_1^f T_0^e = T_2^f B_1^f$, boundary of the subdivision is the subdivision of the boundary. On the right: adjacency constraint $T_0^f B_3^f = T_2^f B_0^f$, or inner patch stitching.

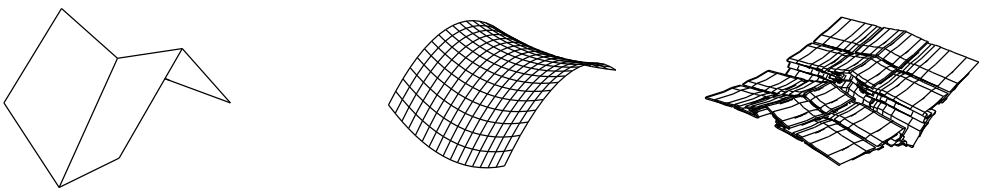


Figure 1.20: Doo-Sabin subdivision. At the left: the control grid. In the middle and at the right are the limit surfaces obtained with traditional coefficients and randomly chosen coefficients.

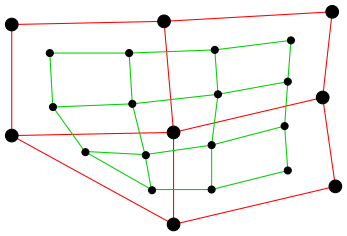


Figure 1.21: Doo-Sabin subdivision with extraordinary vertices, the original grid and the refined one (after one subdivision step).

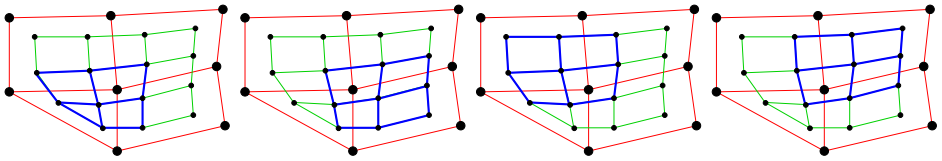


Figure 1.22: The second step of the Doo-Sabin subdivision process: one selects one extraordinary and three regular patches and refines them independently.

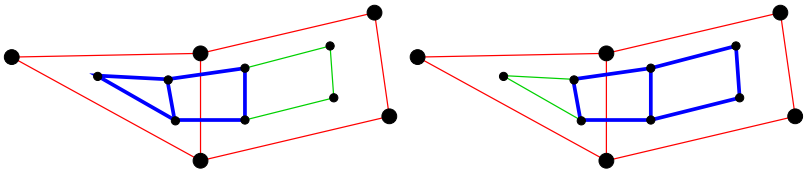


Figure 1.23: Doo-Sabin: subdivision of the boundary with an extraordinary point.

separate nodes in the control graph. However, the blue grids share four control points, therefore, there is a relation between the nodes.

Figure 1.24 presents the control graph for the subdivision of the boundary. The top row is the zero level of the subdivision: the left endpoint of the curve depends on 3 control points, whereas the right one depends on 4 vertices. So, an extraordinary *edge* is bordered with two types of *vertices*: a regular and an extraordinary one. Then, the bottom row shows the first level of the subdivision: the original grid (red, 5 points) is split into two blue sub-grids of 5 and 6 points each. One grid is again an extraordinary *edge*, and another is a perfectly regular *edge*. Figure 1.25 shows the compact version of the control graph.

We have determined the structure of the IFS, now it is time to impose conditions to insure the connectivity of the attractor. First joining conditions are straightforward, since there emerges a regular sub-curve. We simply import the conditions from the section 1.2.1:

$$
\begin{aligned}
B_0^e T_0^v &= T_0^e B_0^e \\
B_1^e T_0^v &= T_1^e B_0^e \\
T_0^e B_1^e &= T_1^e B_0^e.
\end{aligned}
$$

There are three other constraints easy to deduce, follow the arrows in the figure 1.24 to understand the logic. First of all, the boundary of the subdivision is the subdivision of the boundary. For the "left" endpoint: $B_0^{\hat{e}} T_0^{\hat{v}} = T_0^{\hat{e}} B_0^{\hat{e}}$ For the "right" one: $B_1^{\hat{e}} T_0^{\hat{v}} = T_1^{\hat{e}} B_1^e$. Finally, blue grids share control points, therefore: $T_0^{\hat{e}} B_1^{\hat{e}} = T_1^{\hat{e}} B_0^e$.

Once we have constructed an IFS for the boundary, the surfaces are to be dealt in the same way. Remember the figure 1.22. It shows that for an patch the subdivision process constructs three regular sub-patches and one extraordinary patch. The patch has two types of boundaries: the regular (top and right) and the extraordinary one (left and bottom). Then the structure of the IFS may be outlined as it is shown in figure 1.26. The joining conditions are constructed in the same manner. Note that for the sake of clarity we omit vertex nodes. In fact, we just "include" the graph shown in figure 1.25. Indeed, it is very convenient to add the constraints level-by-level (in B-rep sense) and not to obfuscate the highest-level representation with unnecessary details.

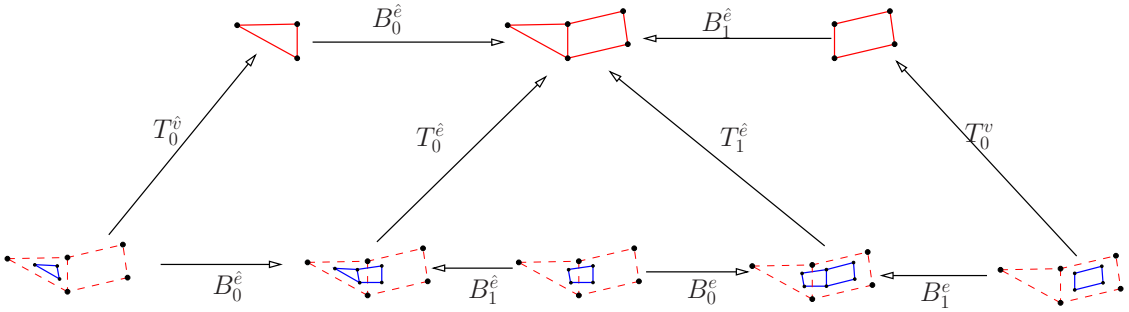


Figure 1.24: Iteration on a control net, refer to the figure 1.25 for the corresponding automaton.

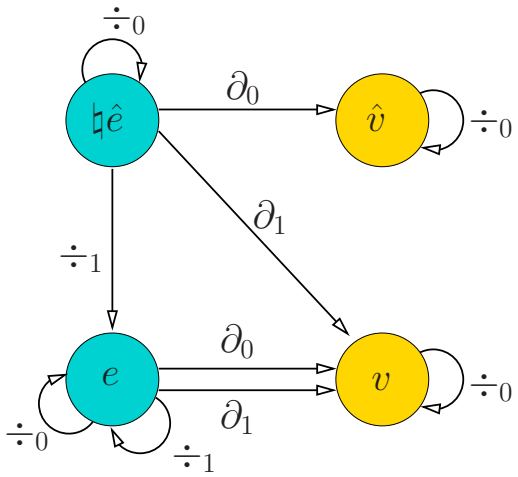


Figure 1.25: B-rep: the edge-vertex topological structure for an extraordinary curve.

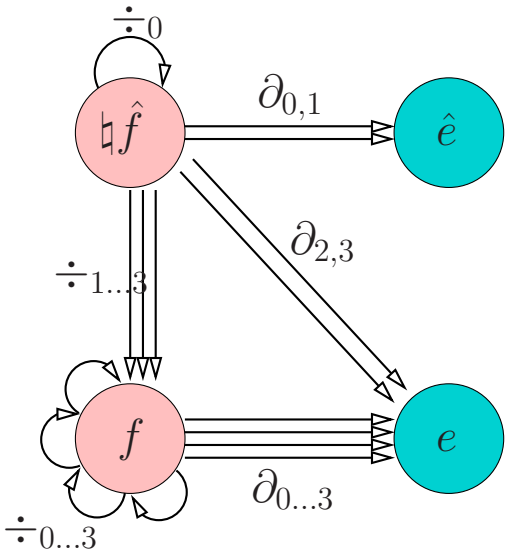


Figure 1.26: B-rep: the face-edge-vertex topological structure for an extraordinary patch.

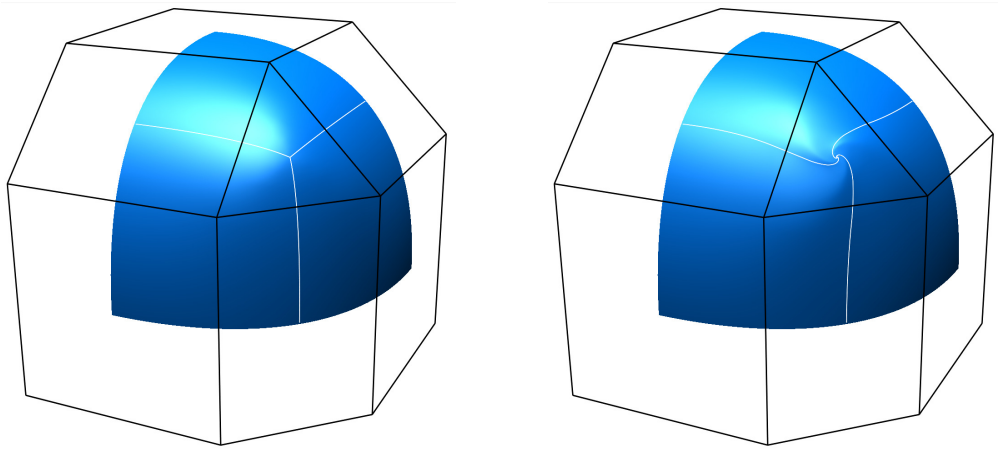


Figure 1.27: Left image: Doo-Sabin subdivision scheme applied on the suitcase corner mesh; right image: the subdivision with modified weights (refer to section 2.2.1 for more details), the studied curves (patch borders) are highlighted.

Therefore, in the case of Doo-Sabin "suitcase corner" patch there are two additional free variables in our IFS. Refer to figure 1.27 for an example. which affect surface features near the extraordinary point.

1.3.3 Non-uniform B-spline curves

A primer on free form curves

The general way to define a free form curve is to take a set of control points $(P_0, P_1, \ldots, P_n)$ in $\mathbb{R}^2$ or $\mathbb{R}^3$ with a set of blending functions $(f_0(t), f_1(t), \ldots, f_n(t))$, and to define a curve $C(t) = \sum_{i=0}^{n} f_i(t)P_i$.

There are may ways to define the blending functions, for example, for B-splines they can be expressed using Cox-de Boor recursion formula. Given a non-decreasing sequence $(t_0, t_1, \ldots, t_m)$, known as the knot vector, then a polynomial $f_{i,p}$ of degree p on the interval $t_i < t < t_{i+1}$ is given by the recurrence relations

$$f_{i,p}(t) = \frac{t - t_i}{t_{i+p} - t_i} f_{i,p-1}(t) + \frac{t_{i+p+1} - t}{t_{i+p+1} - t_{i+1}} f_{i+1,p-1}(t),$$

$f_{i,0} = 1$ if $t_i \leq t \leq t_{i+1}$ and 0 otherwise.

As can be seen from the above, to define a B-spline curve, one starts with $n+1$ control points $(P_0, P_1, \ldots, P_n)$ and $m+1$ knots $(t_0, t_1, \ldots, t_m)$. Then, having defined the degree of the spline $p \equiv m - n - 1$, the equation for p-degree B-spline with is $C(t) = \sum_{i=0}^{n} P_i f_{i,p}(t)$, $t_p \leq t \leq t_{n+1}$. Note that p must be at least 1 (linear) and can be not greater than $n+1$ (the number of control points).

This set of basis functions has the following properties:

- $f_{i,p}(t)$ is a degree p polynomial in t
- *Non-negativity:*
  For all i, p and t, basis function $f_{i,p}(t)$ is non-zero on $[t_i, t_{i+p+1})$. Or, equivalently, $f_{i,p}(t)$ is non-zero on $p+1$ knot spans $[t_i, t_{i+1})$, $[t_{i+1}, t_{i+2})$, $\ldots$, $[t_{i+p}, t_{i+p+1})$.
- On any span $[t_i, t_{i+1})$, **at most** $p+1$ degree p basis functions are non-zero, namely: $f_{i-p,p}(t)$, $f_{i-p+1,p}(t)$, $f_{i-p+2,p}(t)$, $\ldots$, $f_{i,p}(t)$. Therefore, B-spline depends on at most $p+1$ nearest control points at any point t.
- *Partition of Unity:*
  The sum of all non-zero degree p basis functions on span $[t_i, t_{i+1})$ is 1. The previous property shows that $f_{i-p,p}(t)$, $f_{i-p+1,p}(t)$, $f_{i-p+2,p}(t)$, $\ldots$, $f_{i,p}(t)$ are non-zero on $[t_i, t_{i+1})$. This one states that the sum of these $p+1$ basis functions is 1.

Curve knot doubling For the sake of clarity of presentation the following examples are provided for the quadratic case. Figure 1.28 shows an example of blending functions for a non-uniform quadratic B-spline with four control points. Thus, there are four blending functions, defined by a knot vector (t_i) and the degree $p \equiv 2$ of the spline. The curve is defined on the interval $[t_2, t_4]$.

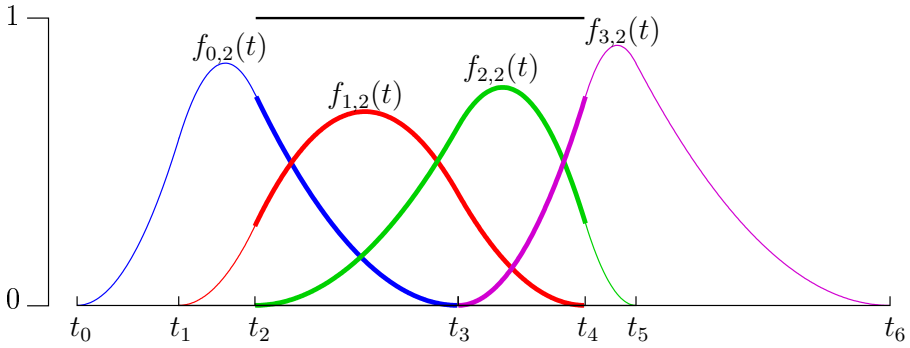


Figure 1.28: B-spline blending functions as like as Bézier ones are non-negative and have "partition of unity" property.

Cox-de Boor recursion formula yields that all blending functions are piecewise quadratic polynoms (where non-zero). Moreover, the shape of the polynoms depends on three adjacent knot intervals. Therefore it is easy to see that knots t_0 and t_6 do not affect the shape of the B-spline curve, because the curve is defined on the interval $[t_2, t_4]$. For our purposes it is convenient to use interval lengths instead of knots. Thus, (we discard two knots t_0 and t_6) for a set of four control points $\{P_0, P_1, P_2, P_3\}$ we have four knot intervals $\{u_0, u_1, u_2, u_3\}$, where $u_i = t_{i+2} - t_{i+1}$.

To draw the curve there are two options. Either we draw it directly with the definition $C(t) = \sum_{i=0}^{n} f_i(t)P_i$ or we can use an iterative approach. In the latter case, the idea is to double the number of control points (as well as the number of knots). In the knot insertion process, a knot is added to the knot vector of a given B-spline. This results in an additional control point and a modification of a few existing control points. The end result is a curve defined by a larger number of control points, but which defines exactly the same curve as before knot insertion. If a new knot is inserted at the midpoint of each current knot interval, the resulting control polygon has twice as many control points, and their coordinates Q_i are [SZSS98]:

$$Q_{2i} = \frac{(u_i + 2u_{i+1})P_i + u_i P_{i+1}}{2(u_i + u_{i+1})} \tag{1.3}$$

$$Q_{2i+1} = \frac{u_{i+1}P_i + (2u_i + u_{i+1})P_{i+1}}{2(u_i + u_{i+1})} \tag{1.4}$$

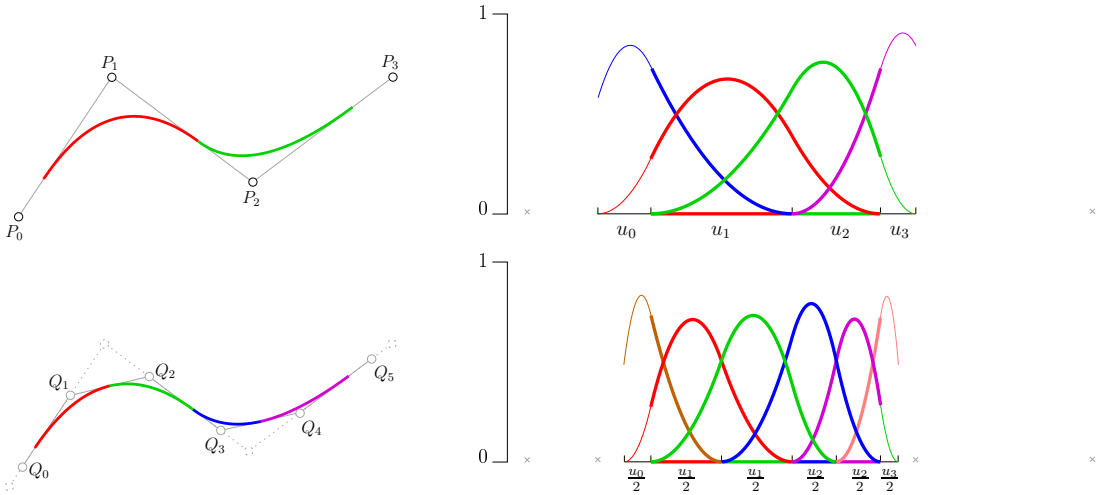


Figure 1.29: The set of control points $\{P_0, P_1, P_2, P_3\}$ and the knot vector $\{u_0, u_1, u_2, u_3\}$ define exactly the same B-spline curve as the control points $\{Q_0, \dots Q_5\}$ with the knot vector $\{\frac{u_0}{2}, \frac{u_1}{2}, \frac{u_1}{2}, \frac{u_2}{2}, \frac{u_2}{2}, \frac{u_3}{2}\}$.

Figure 1.29 illustrates both approaches. Starting with the control points $\{P_0, P_1, P_2, P_3\}$ and the knot vector $\{u_0, u_1, u_2, u_3\}$, it is possible to draw the curve by blending directly the functions $f_{0\dots3,2}(t)$. The top row of the figure gives an illustration.

The bottom row of the figure represents the second approach. Having inserted new knots at the midpoint of each knot interval, we get the control polygon $\{Q_0, \dots Q_5\}$. The new polygon along with the new knot vector $\{\frac{u_0}{2}, \frac{u_1}{2}, \frac{u_1}{2}, \frac{u_2}{2}, \frac{u_2}{2}, \frac{u_3}{2}\}$ defines exactly the same curve, however, points Q_i lie closer to the curve than the initial points P_i. We can get a set of control points that well approximates the curve just by repeating the process few times.

Note that it is possible to stretch uniformly the knot vector without affecting the curve, i.e., the curve from figure 1.29 may be equally defined by the control points $\{Q_0, \dots Q_5\}$ and the knot vector $\{u_0, u_1, u_1, u_2, u_2, u_3\}$.

Figure 1.30 shows one more iteration of the curve knot doubling process. Note that the simplest quadratic B-spline curve is defined with three control points, whereas our example has four control points. It is possible to split our example into two different curves. Refer to the top row of the figure 1.29. The first part of the curve (shown in red) is defined by the control points $\{P_0, P_1, P_2\}$ with the knot vector $\{u_0, u_1, u_2\}$, and the second (the green one) is given by the control points

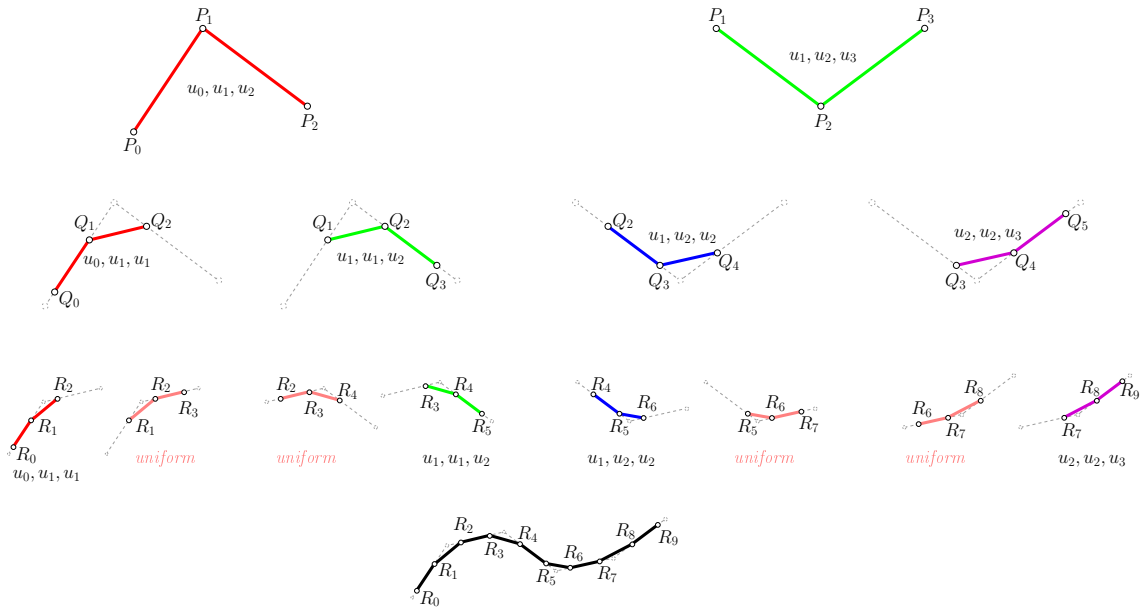


Figure 1.30: Curve knot doubling along with the splitting into atomic parts. Starting from the second iteration, uniform parts emerge.

$\{P_1, P_2, P_3\}$ with the knot vector $\{u_1, u_2, u_3\}$. The joint of the curves is assured by the overlap of the control sets.

In the same manner, the curve defined by $\{Q_0, \dots Q_5\}$ and $\{u_0, u_1, u_1, u_2, u_2, u_3\}$ can be split into four elementary curves (the second row of the figure 1.30). Going further, the curve with control points $\{R_0, \dots R_9\}$ is the union of eight elementary curves. The interesting point here is that the curve with control points $\{R_1, R_2, R_3\}$ has the knot vector $\{u_1, u_1, u_1\}$, and it means that it is a *uniform* B-spline curve. Moreover, starting with a uniform B-spline, no non-uniform curve can emerge. Thus, starting from the second iteration of the knot doubling process, the number of non-uniform elementary curves is *stationary*.

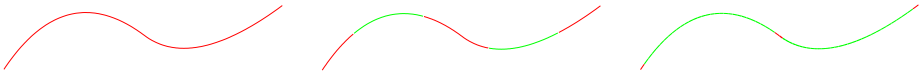


Figure 1.31: From left to right: the curve at iteration 0 (original control points), iteration 2 and iteration 6. Regular parts of the curve are shown in green, whereas non-uniform ones are in red.

Figure 1.31 illustrates the process. At the very beginning four control points define the non-uniform B-spline curve. After the second iteration the very same curve consists of eight elementary pieces (refer to the third row of the figure 1.30). There are four uniform pieces, the corresponding parts of the curve are shown in green. As the number of elementary pieces grows, each piece shrinks. It turns out that the non-uniform curve knot doubling is just a subdivision scheme with few singular points.

Subdivision masks

Figure 1.32 gives the schema of the process, read it along with the figure 1.30. At the very beginning we have three control points $P_{0,1,2}$ with three knot intervals. This phase is shown by the uvw node in the schema. Then the limit curve is split into two parts with control points $Q_{0,1,2}$ and $Q_{1,2,3}$, each of them depends on two original knot intervals. We may say that we apply two different transformations (shown by p_0 and p_1) to obtain the new control points, in other words:

$$\begin{bmatrix} Q_0 & Q_1 & Q_2 \end{bmatrix} = \begin{bmatrix} P_0 & P_1 & P_2 \end{bmatrix} \times p_0(u_0, u_1, u_2) \qquad : uvv \text{ node}$$
$$\begin{bmatrix} Q_1 & Q_2 & Q_3 \end{bmatrix} = \begin{bmatrix} P_0 & P_1 & P_2 \end{bmatrix} \times p_1(u_0, u_1, u_2) \qquad : vvw \text{ node}$$

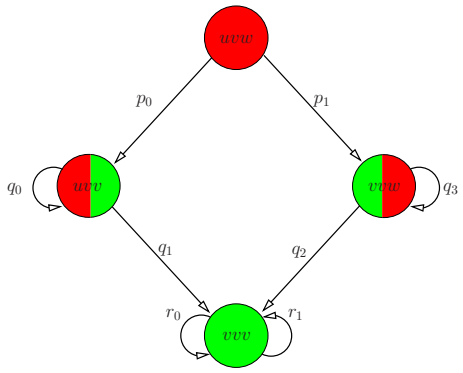


Figure 1.32: Schema of the knot insertion process.

The second iteration generates four groups of control points:

$$\begin{aligned}
\begin{bmatrix} R_0 & R_1 & R_2 \end{bmatrix} &= \begin{bmatrix} Q_0 & Q_1 & Q_2 \end{bmatrix} \times q_0(u_0, u_1) && : uvv \text{ node} \\
\begin{bmatrix} R_1 & R_2 & R_3 \end{bmatrix} &= \begin{bmatrix} Q_0 & Q_1 & Q_2 \end{bmatrix} \times q_1(u_0, u_1) && : vvv \text{ (uniform) node} \\
\begin{bmatrix} R_2 & R_3 & R_4 \end{bmatrix} &= \begin{bmatrix} Q_1 & Q_2 & Q_3 \end{bmatrix} \times q_2(u_1, u_2) && : vvv \text{ (uniform) node} \\
\begin{bmatrix} R_3 & R_4 & R_5 \end{bmatrix} &= \begin{bmatrix} Q_1 & Q_2 & Q_3 \end{bmatrix} \times q_3(u_1, u_2) && : vvw \text{ node}
\end{aligned}$$

Note that the transformations depend on the initial knot vector, however, they does not change over time, i.e., once the knot vector is given, the corresponding matrices are constant.

Let us determine the relationship between the transformations. We know that after the first iteration uvv and vvw nodes must share two control points Q_1 and Q_2. Here comes naturally the first constraint.

$$\begin{aligned}
\begin{bmatrix} Q_1 & Q_2 \end{bmatrix} = \begin{bmatrix} Q_0 & Q_1 & Q_2 \end{bmatrix} \times \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} &= \begin{bmatrix} P_0 & P_1 & P_2 \end{bmatrix} \times p_0(u_0, u_1, u_2) \times \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \\
\begin{bmatrix} Q_1 & Q_2 \end{bmatrix} = \begin{bmatrix} Q_1 & Q_2 & Q_3 \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} &= \begin{bmatrix} P_0 & P_1 & P_2 \end{bmatrix} \times p_1(u_0, u_1, u_2) \times \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}
\end{aligned}$$

Thus, $p_0(u_0, u_1, u_2) \times \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} = p_1(u_0, u_1, u_2) \times \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$. This equation states that two last columns of the matrix p_0 must be equal to the two first columns of the matrix p_1. Let us write down the matrices. Remember that the coefficients of the matrices are functions of three parameters (the knot vector).

$$p_0(u_0, u_1, u_2) = \begin{bmatrix} a_0 & b_0 & c_0 \\ a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{bmatrix} \qquad p_1(u_0, u_1, u_2) = \begin{bmatrix} b_0 & c_0 & d_0 \\ b_1 & c_1 & d_1 \\ b_2 & c_2 & d_2 \end{bmatrix}$$

Then the second constraint may be written as follows:

$$\begin{aligned}
\begin{bmatrix} Q_2 & Q_3 \end{bmatrix} = \begin{bmatrix} Q_1 & Q_2 & Q_3 \end{bmatrix} \times \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} &= \begin{bmatrix} P_0 & P_1 & P_2 \end{bmatrix} \times p_1(u_0, u_1, u_2) \times \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \\
\begin{bmatrix} Q_2 & Q_3 \end{bmatrix} = \begin{bmatrix} Q_2 & Q_3 & Q_4 \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} &= \begin{bmatrix} P_1 & P_2 & P_3 \end{bmatrix} \times p_0(u_1, u_2, u_3) \times \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}
\end{aligned}$$

This means that for all possible configurations of P_0, P_1, P_2, P_3 and u_0, u_1, u_2, u_3 the following equations must be held:

$$\begin{cases}
P_0 * c_0(u_0,u_1,u_2) + P_1 * c_1(u_0,u_1,u_2) + P_2 * c_2(u_0,u_1,u_2) = \\
\qquad\qquad P_1 * a_0(u_1,u_2,u_3) + P_2 * a_1(u_1,u_2,u_3) + P_3 * a_2(u_1,u_2,u_3) \\
P_0 * d_0(u_0,u_1,u_2) + P_1 * d_1(u_0,u_1,u_2) + P_2 * d_2(u_0,u_1,u_2) = \\
\qquad\qquad P_1 * b_0(u_1,u_2,u_3) + P_2 * b_1(u_1,u_2,u_3) + P_3 * b_2(u_1,u_2,u_3)
\end{cases}$$

The only case to make the system feasible is when $c_0 \equiv d_0 \equiv a_2 \equiv b_2 \equiv 0$ and the rest are functions depending on two parameters only, related as follows:

$$a_0(u_1,u_2,\cdot) \equiv c_1(\cdot,u_1,u_2) \qquad b_0(u_1,u_2,\cdot) \equiv d_1(\cdot,u_1,u_2)$$
$$a_1(u_1,u_2,\cdot) \equiv c_2(\cdot,u_1,u_2) \qquad b_1(u_1,u_2,\cdot) \equiv d_2(\cdot,u_1,u_2)$$

Now let us remember equations (1.3) and (1.4). The equations perfectly fit into the deduced constraints. Indeed, it is possible to rewrite the equations in the following manner:

$$p_0(u,v,w) = \begin{bmatrix} \frac{u+2v}{2(u+v)} & \frac{v}{2(u+v)} & 0 \\ \frac{u}{2(u+v)} & \frac{2u+v}{2(u+v)} & \frac{v+2w}{2(v+w)} \\ 0 & 0 & \frac{v}{2(v+w)} \end{bmatrix} \qquad p_1(u,v,w) = \begin{bmatrix} \frac{v}{2(u+v)} & 0 & 0 \\ \frac{2u+v}{2(u+v)} & \frac{v+2w}{2(v+w)} & \frac{w}{2(v+w)} \\ 0 & \frac{v}{2(v+w)} & \frac{2v+w}{2(v+w)} \end{bmatrix}$$

The relationship between the transformations $p_{0,1}$ and $q_{0,1,2,3}$, $r_{0,1}$ can be deduced in the very same manner. The transformations r_0 and r_1 correspond to uniform parts, therefore it is possible to take the coefficients from the subdivision masks directly (or from the equations (1.3) and (1.4) with equal knot intervals):

$$r_0 = \begin{bmatrix} 0.75 & 0.25 & 0 \\ 0.25 & 0.75 & 0.75 \\ 0 & 0 & 0.25 \end{bmatrix} \qquad r_1 = \begin{bmatrix} 0.25 & 0 & 0 \\ 0.75 & 0.75 & 0.25 \\ 0 & 0.25 & 0.75 \end{bmatrix}$$

$$q_0(u,v,\cdot) = \begin{bmatrix} \frac{u+2v}{2(u+v)} & \frac{v}{2(u+v)} & 0 \\ \frac{u}{2(u+v)} & \frac{2u+v}{2(u+v)} & 0.75 \\ 0 & 0 & 0.25 \end{bmatrix} \qquad q_1(u,v,\cdot) = \begin{bmatrix} \frac{v}{2(u+v)} & 0 & 0 \\ \frac{2u+v}{2(u+v)} & 0.75 & 0.25 \\ 0 & 0.25 & 0.75 \end{bmatrix}$$

$$q_2(\cdot,v,w) = \begin{bmatrix} 0.75 & 0.25 & 0 \\ 0.25 & 0.75 & \frac{v+2w}{2(v+w)} \\ 0 & 0 & \frac{v}{2(v+w)} \end{bmatrix} \qquad q_3(\cdot,v,w) = \begin{bmatrix} 0.25 & 0 & 0 \\ 0.75 & \frac{v+2w}{2(v+w)} & \frac{w}{2(v+w)} \\ 0 & \frac{v}{2(v+w)} & \frac{2v+w}{2(v+w)} \end{bmatrix}$$

Controlled IFS

In fact, we have just built a controlled iterated function system, whose attractor is exactly the non-uniform B-spline curve. To conclude, the control graph (refer to figure 1.32) and the above transformations $p_{0,1}$, $q_{0,1,2,3}$ and $r_{0,1}$ define a graph-directed IFS with the B-spline curve for its attractor. Thus, a non-uniform B-spline curve is a particular case of a self-similar (fractal) curve. In the previous section we have used the coefficients deduced by Sabin et al. What happens if we change the coefficients? The connectivity constraints we have imposed in the previous section force the attractor to be a connected (in general case non-differentiable) curve. Thus, the above constraints impose the following shape of the transformations:

$$p_0(u,v,w) = \begin{bmatrix} 1-a(u,v) & b(u,v) & 0 \\ a(u,v) & 1-b(u,v) & 1-a(v,w) \\ 0 & 0 & a(v,w) \end{bmatrix} \qquad r_0 = \begin{bmatrix} 1-c & d & 0 \\ c & 1-d & 1-c \\ 0 & 0 & c \end{bmatrix}$$

$$p_1(u,v,w) = \begin{bmatrix} b(u,v) & 0 & 0 \\ 1-b(u,v) & 1-a(v,w) & b(v,w) \\ 0 & a(v,w) & 1-b(v,w) \end{bmatrix} \qquad r_1 = \begin{bmatrix} d & 0 & 0 \\ 1-d & 1-c & d \\ 0 & c & 1-d \end{bmatrix}$$

$$q_0(u,v,\cdot) = \begin{bmatrix} 1-a(u,v) & b(u,v) & 0 \\ a(u,v) & 1-b(u,v) & 1-c \\ 0 & 0 & c \end{bmatrix} \qquad q_1(u,v,\cdot) = \begin{bmatrix} b(u,v) & 0 & 0 \\ 1-b(u,v) & 1-c & d \\ 0 & c & 1-d \end{bmatrix}$$

$$q_2(\cdot,v,w) = \begin{bmatrix} 1-c & d & 0 \\ c & 1-d & 1-a(v,w) \\ 0 & 0 & a(v,w) \end{bmatrix} \qquad q_3(\cdot,v,w) = \begin{bmatrix} d & 0 & 0 \\ 1-d & 1-a(v,w) & b(v,w) \\ 0 & a(v,w) & 1-b(v,w) \end{bmatrix}$$

The leftmost image of the figure 1.33 shows an attractor of the IFS with randomly chosen (with respect to the deduced shape) elements of the matrices.

A recent study of fractal curves [BGN08] introduces the notion of half-tangents in the endpoints of a curve. In fact, a large family of fractal curves possesses half-tangents in the endpoints (e.g. the leftmost image of figure 1.33). The paper states that the half-tangents depend on the eigenvector corresponding to the second largest eigenvalue of the transformation.

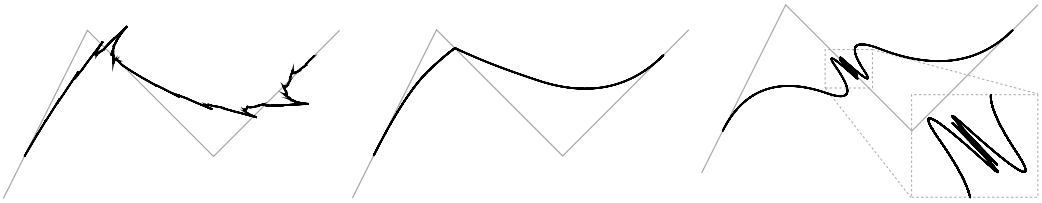


Figure 1.33: At the left: the curve has no derivatives at all (however, it does have half-tangents in the endpoints); in the middle and at the right: the curve has no derivative at the extraordinary point.

First of all, having defined half-tangents for the endpoints of a curve, we can define two half-tangents for the midpoint of the curve. That is so, one iteration of the subdivision process splits our curve into two independent ones and it is possible to define half-tangents for the endpoints of each one. If the half-tangents exist and coincide in the midpoint, we get C^1 continuity for the curve in the midpoint. If we continue the reasoning, we obtain half-tangents defined for a set of points dense in the curve.

There are four types of joints for our IFS curve: $uvv - vvv$, $vvv - vvv$, $vvv - vvw$, $vvw - uvv$ (refer to the third row of the figure 1.30). Let us consider the joint between vvw and uvv nodes. In our example there one point in the middle of the curve with this type of joint.

The transformation q_0 has three eigenvalues: $(1, c, 1-a-b)$ with the corresponding eigenvectors $\begin{pmatrix} b \\ a \\ 0 \end{pmatrix}, \begin{pmatrix} -b \\ 1-a-c \\ a+b+c-1 \end{pmatrix}, \begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix}$, as well as q_3 has $(1, d, 1-a-b)$ and $\begin{pmatrix} 0 \\ b \\ a \end{pmatrix}, \begin{pmatrix} a+b+d-1 \\ 1-b-d \\ -a \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ -1 \end{pmatrix}$. The eigenvectors corresponding to the eigenvalues 1 give fixed points of the transformations q_0 and q_3. The half-tangents depend of the eigenvector corresponding to the second largest eigenvalue, i.e. for q_0 there are two options: either $c > 1 - a - b$ and then the half-tangent depends on three control points, or $c \leq 1 - a - b$ and the half-tangent depends on two control points.

The C^1 continuity in this point may be obtained if the half-tangents coincide. We know that the vvw and uvv nodes share two control points. Thus, if the half-tangents depend on the position of the common control points only, the C^1 continuity will be achieved. To resume: the C^1 continuity in the joint $vvw - uvv$ is ensured by the conditions

$$\begin{cases} c \leq 1 - a - b \\ d \leq 1 - a - b \end{cases}$$

In the same way we can write down the conditions for the $vvv - vvv$ joint. The eigenvalues and eigenvectors for r_0 and r_1 are respectively $(1, 1-c-d, c)$, $\begin{pmatrix} d \\ c \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ -1 \\ 0 \end{pmatrix}, \begin{pmatrix} -d \\ 1-2c \\ 2c+2d-1 \end{pmatrix}$ and $(1, 1-c-d, d)$ $\begin{pmatrix} 0 \\ d \\ c \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ -1 \end{pmatrix}, \begin{pmatrix} c+2d-1 \\ 1-2d \\ -c \end{pmatrix}$. Therefore, the C^1 continuity in all joints of $vvv - vvv$ type is ensured by the following conditions:

$$\begin{cases} c \leq 1 - c - d \\ d \leq 1 - c - d \end{cases}$$

The continuity in $uvv - vvv$ and $vvv - vvw$ joints may be deduced in the same manner, but the conditions obtained are more feeble that the above ones. The leftmost image of the figure 1.33 shows a curve where $vvv - vvv$ joining conditions are not satisfied. In the middle and at the right of the figure there are curves that break the joining condition only in the "extraordinary" point.

This quick and incomplete study gives us a foretaste for the next chapter, where we will go through all aspects of differentiating fractal attractors given by affine IFS.

Chapter 2

Tangent spaces for self-similar shapes

This chapter presents a continuation of works realized by Bensoudane *et al* [Ben09] and Podkorytov *et al* [Pod13]. It is a first step towards full control of geometrical texture of objects we create with BC-IFS. This kind of analysis is useful when creating subdivision schemes to guarantee physical properties of modelled shapes.

We give a definition of a tangent subspace from a purely geometrical point of view. We want to avoid to define the tangent depending on a parameterization, because given the complex nature of self-similar shapes it may be difficult to find correct parameterizations. Here we focus on analyzing smoothness of attractors of *affine* Iterated Function Systems. So, first of all, what a tangent is?

Definition 1 (Tangent space)**.** *Let p be a point of some set $\mathcal{A} \subset \mathbb{R}^n$. An affine subspace E is said to be tangent to $\mathcal{A}$ in point p if:*

1. $p \in E$,
2. *for any $\varepsilon > 0$ there exists a ball $B(p, r)$ centered in p with radius r such that $\forall x \in \mathcal{A} \cap B(p, r) \Rightarrow \frac{\|x - x_{\perp E}\|}{\|x - p\|} < \varepsilon$, where $x_{\perp E}$ is the orthogonal projection of x onto E,*
3. *$dim(E) \geq 1$ and $dim(E)$ is a minimum of all possible subspaces E that satisfy above conditions.*

This is essentially the same definition Falconer [Fal90, p. 86] uses in his book, refer to figure 2.1 for an illustration. While including the standard notion of tangents, this definition allows to define tangents without introducing explicitly a parameterization. Moreover, it also includes other cases, figure 2.2 provides an illustration. For example, 3D parametric curve $(t \cos 1/t, t \sin 1/t, t^2)$ does not possess a tangent line at the origin, however the plane Oxy satisfies the definition of a tangent subspace. Another example is a 3D parametric surface $(x, y, (x^2 + y^2)^{1/4})$ that is tangent to the z-axis at the point $(x, y) = (0, 0)$.

An outline of our approach

In this chapter we propose a method to check differential behavior of an IFS attractor for a point with periodic address. Recall that any point of an IFS attractor possesses at least one address. An address is an infinte sequence of indices of IFS transformations: if we take an arbitrary (non-empty) compact subset and apply IFS (contractive) transformations corresponding to the address sequence, the compact converges to a point. Thus this sequence addresses the point. Currently we have a simple way to check differential behaviour only for points with **periodic** addresses (recall that they are dense in the attractor). Aperiodic points will be addressed in nearest future works.

If a point has multiple addresses, it suffices to verify that the differential behavior is the same for all of them, so **we focus here on checking a single address**. Moreover, without loss of generality **we can focus on the fixed point** of an IFS transformation: indeed, if we want to study a point with a periodic address other than a fixed point we can simply enrich our IFS with a composed transformation corresponding to the period. It does not alter the attractor, but provides us a convenient basis.

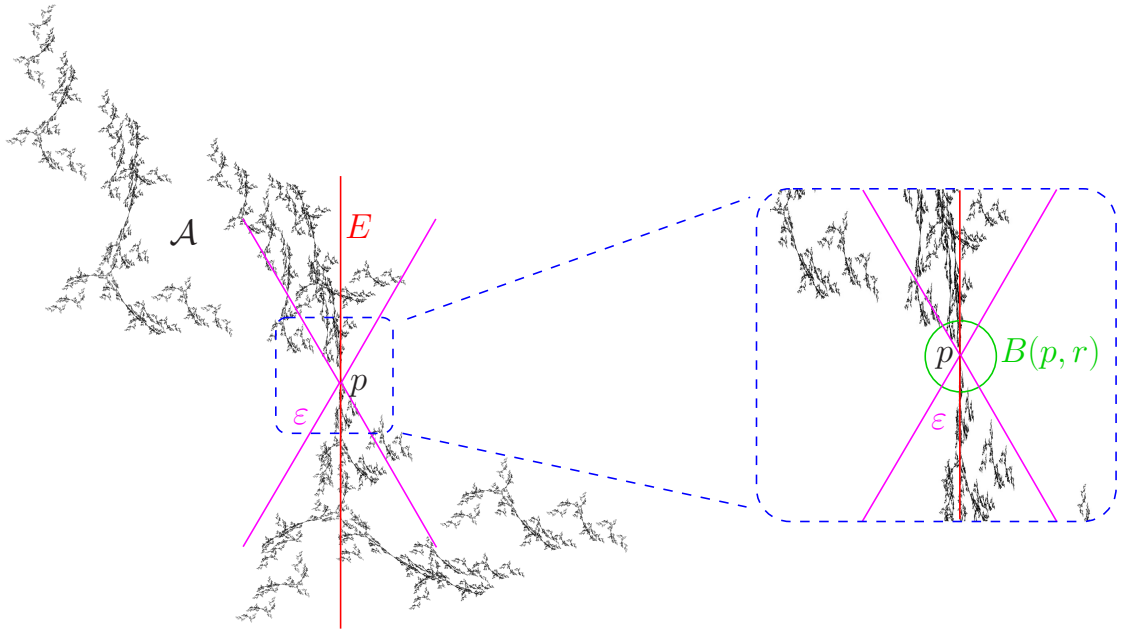


Figure 2.1: For $\mathcal{A}$ to have a tangent E at point p means that for any cone generated by E and ε there must be a ball $B(p,r)$ with radius r such that all points $\mathcal{A} \cap B(p,r)$ are in the given cone.

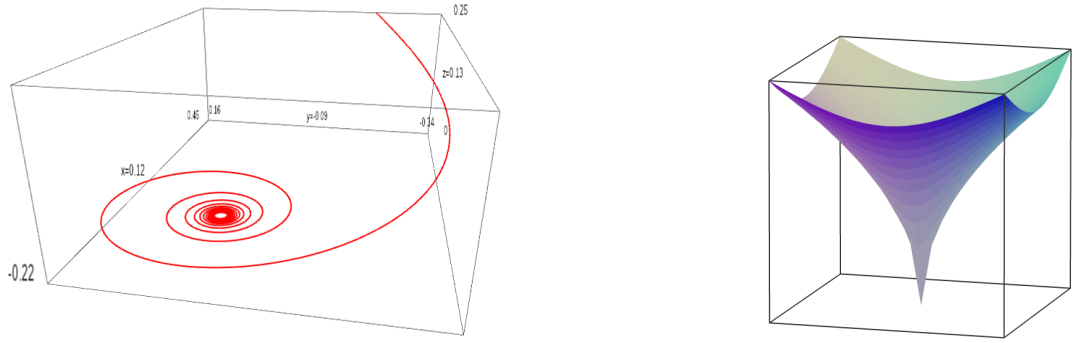


Figure 2.2: Left: parametric curve $(t\cos 1/t, t\sin 1/t, t^2)$ is tangent to the plane Oxy. Right: parametric surface $(x, y, (x^2+y^2)^{1/4})$ is tangent to the z axis.

Let us consider the IFS attractor we used to illustrate the tangent definition. The corresponding IFS consists of four transformations:

$$
\begin{aligned}
T_0\left(\begin{bmatrix}x\\y\end{bmatrix}\right) &= \begin{bmatrix}0.25 & 0\\ 0 & 0.5\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix}\\
T_1\left(\begin{bmatrix}x\\y\end{bmatrix}\right) &= \begin{bmatrix}0.45 & 0.15\\ -0.15 & 0.45\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix} + \begin{bmatrix}-0.8\\0.6\end{bmatrix}\\
T_2\left(\begin{bmatrix}x\\y\end{bmatrix}\right) &= \begin{bmatrix}0.3 & -0.03\\ 0.03 & 0.3\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix} + \begin{bmatrix}-0.1\\0.5\end{bmatrix}\\
T_3\left(\begin{bmatrix}x\\y\end{bmatrix}\right) &= \begin{bmatrix}-0.3 & 0.4\\ -0.4 & -0.3\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix} + \begin{bmatrix}0.2\\-0.7\end{bmatrix}
\end{aligned}
$$

We want to study the behavior in the vicinity of the origin, $p = 0$ (note that p is the fixed point of T_0). It is natural to suppose that the behavior around the fixed point is dictated by the eigenbasis of T_0. T_0 has two eigenvectors $\vec{v_1} = (0,1)$ and $\vec{v_2} = (1,0)$ with eigenvalues $\lambda_1 = 0.5$ and $\lambda_2 = 0.25$, correspondingly. T_0 contracts the space slower in the $\vec{v_1}$ direction, thus it is natural to expect that the tangent space $E = \text{span}(\vec{v_1})$. Figure 2.3 shows an illustration.

How do we prove that E is indeed tangent to $\mathcal{A}$ in point p?

- First of all, we define a covering of the attractor (in the same manner as Prautzsch [Pra98] does for subdivision surfaces). We define a set $R_1 = \overline{\mathcal{A} \setminus T_0(\mathcal{A})}$ and $R_i = T_0(R_{i-1})$. It defines a sequence of rings $\{R_i\}_{i=1}^{\infty}$, converging to the point of interest, whose (infinite) union is the attractor itself. Refer to figure 2.4 for an illustration.
- Next step is to check whether $R_1 \cap \operatorname{span}(\vec{v_2})$ is empty. In our example it means to verify if the red set intersects with the x-axis. Section 2.2.1 explains in detail why we need to check this.
- If the intersection is empty, then the tangent will be given by the dominant eigenvector(s).

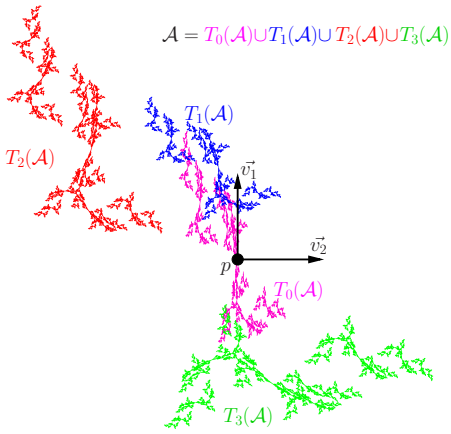


Figure 2.3: The attractor $\mathcal{A}$ consists of the union of four scaled copies of itself, we study the behavior in the point p, expecting span $\vec{v_1}$ to be tangent to $\mathcal{A}$.

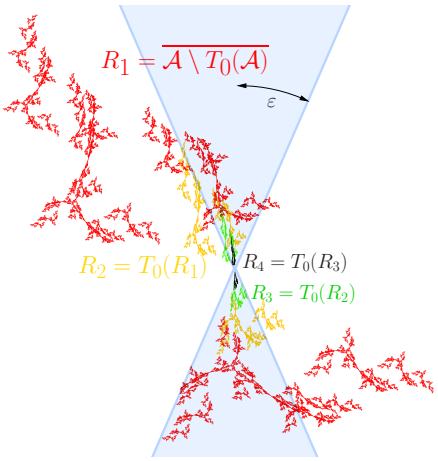


Figure 2.4: If for any choice of ε we can find a number of steps N such that $\forall i > N$ ring R_i lies in the cone generated by E and ε, E is tangent to $\mathcal{A}$. Here $N = 3$: for this choice of ε the ring R_4 (in black) lies in the cone.

In the following sections of this chapter we illustrate the related works and perform a more formal analysis of our approach.

2.1 Related works

Differential properties of subdivision surfaces are the subject of many studies. In [Pra98] Prautzsch studies the differentiability of subdivision schemes at the extraordinary vertices. His study exploits the following idea: for every extraordinary vertex at every step of the subdivision its neighboring control points define a ring of m irregular patches, where m is the valence of that vertex. After one step of the subdivision, m irregular patches are subdivided into m smaller irregular patches and $3m$ regular patches. These $3m$ regular patches form a ring on a surface that can be parameterized over $[0,1]^2 \times \{1, 2, \ldots, 3m\}$. Next subdivision step produces a smaller ring that also consists of $3m$ regular patches. A certain parameterization used by Prautzsch is derived from the notion of characteristic map introduced by Reif and Peters in [Rei95]. Our study will follow a similar idea, but in a more general way. Let us first show what a characteristic map is.

A primer on eigenvectors

Consider a $n \times n$ diagonalizable matrix T whose eigenvalues are all distinct and satisfy $|\lambda_0| > |\lambda_1| > |\lambda_i|,\ i \geq 2$. Define right eigenvectors $T\vec{u}_i = \lambda_i \vec{u}_i$ and left eigenvectors $\vec{v}_i T = \lambda_i \vec{v}_i$. Here $\vec{u}_i$ are column vectors and $\vec{v}_i$ are row vectors. The purpose of this section is to describe the role the dominant and sub-dominant eigenvectors play in the action of the repeated transformation T^k as k grows large.

Property 1. $i \neq j \Rightarrow \vec{u}_i \perp \vec{v}_j$

Proof. It suffices to compute $\vec{v}_j T \vec{u}_i$ in two different ways:

$$(\vec{v}_j T)\vec{u}_i = \lambda_j \vec{v}_j \vec{u}_i$$
$$\vec{v}_j (T\vec{u}_i) = \lambda_i \vec{v}_j \vec{u}_i$$

We supposed that all eigenvalues are distinct, therefore $\lambda_i \neq \lambda_j$. Subtracting two above equations we get $(\lambda_i - \lambda_j)\vec{u}_i\vec{v}_j = 0$ and thus $\vec{u}_i\vec{v}_j = 0$. □

Without loss of generality we can suppose that $\vec{u}_i$ and $\vec{v}_j$ are scaled so that the matrices built from the $\vec{u}_i$ and $\vec{v}_j$ are inverses of each other:

$$\begin{pmatrix} \vec{v}_0 \\ \vec{v}_1 \\ \vdots \\ \vec{v}_{n-1} \end{pmatrix} \begin{pmatrix} \vec{u}_0 & \vec{u}_1 & \dots & \vec{u}_{n-1} \end{pmatrix} = I$$

Property 2. $T = \sum_{i=0}^{n-1} \lambda_i \vec{u}_i \vec{v}_i$.

Proof. By our assumptions above, we have

$$\begin{pmatrix} \vec{v}_0 \\ \vec{v}_1 \\ \vdots \\ \vec{v}_{n-1} \end{pmatrix} T = \begin{pmatrix} \lambda_0 & 0 & \dots & 0 \\ 0 & \lambda_1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_{n-1} \end{pmatrix} \begin{pmatrix} \vec{v}_0 \\ \vec{v}_1 \\ \vdots \\ \vec{v}_{n-1} \end{pmatrix}$$

Let us left post multiply each member of this expression by the matrix $(\vec{u}_i) = (\vec{v}_i)^{-1}$:

$$T = \begin{pmatrix} \vec{u}_0 & \vec{u}_1 & \dots & \vec{u}_{n-1} \end{pmatrix} \begin{pmatrix} \lambda_0 & 0 & \dots & 0 \\ 0 & \lambda_1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_{n-1} \end{pmatrix} \begin{pmatrix} \vec{v}_0 \\ \vec{v}_1 \\ \vdots \\ \vec{v}_{n-1} \end{pmatrix}$$

This equation gives us an eigendecomposition of the matrix:

$$T = (\vec{u}_i)\Lambda(\vec{v}_i) = \sum_{i=0}^{n-1} \lambda_i \vec{u}_i \vec{v}_i. \tag{2.1}$$

Here Λ represents the diagonal matrix with eigenvalues at the main diagonal. □

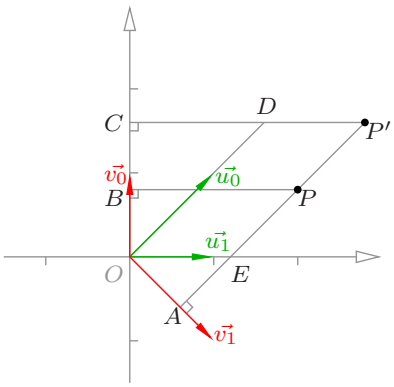


Figure 2.5: Action of a linear transformation on two eigenbases, the right eigenvectors are shown in green and left eigenvectors are red.

This property is known under the name of eigendecomposition of a matrix, it tells us that any matrix can be represented as a sum of rank-1 matrices. Let us illustrate the geometric meaning of the formula with $T = \begin{pmatrix} 1 & 1 \\ 0 & 2 \end{pmatrix}$. The matrix T has eigenvalues $\lambda_0 = 2$ and $\lambda_1 = 1$, right eigenvectors $\vec{u}_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$, $\vec{u}_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ and left eigenvectors $\vec{v}_0 = \begin{pmatrix} 1 & -1 \end{pmatrix}$, $\vec{v}_1 = \begin{pmatrix} 0 & 1 \end{pmatrix}$. Equation (2.1) can be understood more easily if applied to a point P (refer to figure 2.5 for an illustration). Basically it says that to find the transformed point $P' = T \times P$ from the original point P, we have two options: either decompose the vector $\overrightarrow{OP}$ in the basis $\vec{u}_0, \vec{u}_1$, scale the decomposition by corresponding eigenvalues and re-compose the transformed point. Or we can use the left eigenvectors $\vec{v}_0, \vec{v}_1$: first we (orthogonally) project the vector $\overrightarrow{OP}$, rescale vectors and (orthogonally) unproject to get the desired point P'. More strictly,

$$\overrightarrow{OP'} = \begin{pmatrix} P'_x \\ P'_y \end{pmatrix} = T \begin{pmatrix} P_x \\ P_y \end{pmatrix} = \vec{u}_0 \lambda_0 \left(\vec{v}_0 \begin{pmatrix} P_x \\ P_y \end{pmatrix} \right) + \vec{u}_1 \lambda_1 \left(\vec{v}_1 \begin{pmatrix} P_x \\ P_y \end{pmatrix} \right).$$

The part $\begin{pmatrix} \vec{v}_{ix} & \vec{v}_{iy} \end{pmatrix} \begin{pmatrix} P_x \\ P_y \end{pmatrix}$ is the dot product between vectors $\vec{v}_i$ and $\overrightarrow{OP}$. In this way we can find the orthogonal projection B of the point P onto the axis $\vec{v}_0$: $|\overrightarrow{OB}| = (\vec{v}_0 \cdot \overrightarrow{OP})/|\vec{v}_0|$. The transformed point P' can be found through $\overrightarrow{OP'} = \overrightarrow{OD} + \overrightarrow{OE}$. Vector $\overrightarrow{OD}$ can be found by re-scaling vector $\vec{u}_0$:

$$\overrightarrow{OD} = \frac{\vec{u}_0}{|\vec{u}_0|}|\overrightarrow{OD}| = \frac{\vec{u}_0}{|\vec{u}_0|} \frac{|\overrightarrow{OC}||\vec{u}_0||\vec{v}_0|}{\vec{u}_0 \cdot \vec{v}_0} = \vec{u}_0 \frac{|\overrightarrow{OC}||\vec{v}_0|}{\vec{u}_0 \cdot \vec{v}_0} = \vec{u}_0 \lambda_0 \frac{|\overrightarrow{OB}||\vec{v}_0|}{\vec{u}_0 \cdot \vec{v}_0} = \vec{u}_0 \lambda_0 \frac{\vec{v}_0 \cdot \overrightarrow{OP}}{\vec{u}_0 \cdot \vec{v}_0} = \vec{u}_0 \lambda_0 \vec{v}_0 \begin{pmatrix} P_x \\ P_y \end{pmatrix}$$

The last transition is valid due to our assumption on the eigenvectors scaling: $\vec{u}_i \cdot \vec{v}_i = 1$. In the same way it possible to show $\overrightarrow{OE} = \vec{u}_1 \lambda_1 \vec{v}_1 \begin{pmatrix} P_x \\ P_y \end{pmatrix}$.

Property 3. *Repeated action of T can be described as $T^k = \sum_{i=0}^{n-1} \lambda_i^k \vec{u}_i \vec{v}_i$.*

Proof. It is easy to see that $T \times T \times \cdots \times T = (\vec{u}_i)\Lambda(\vec{v}_i) \times \cdots \times (\vec{u}_i)\Lambda(\vec{v}_i)$. Since $(\vec{u}_i) \times (\vec{v}_i) = I$, we obtain $T^k = (\vec{u}_i)\Lambda^k(\vec{v}_i) = \sum_{i=0}^{n-1} \lambda_i^k \vec{u}_i \vec{v}_i$. □

As an immediate consequence we get that the second term decays exponentially quickly with respect to the first, and the third and higher terms decay exponentially quickly with respect to the second:

Corollary 1. $T^k = \lambda_0^k \vec{u}_0 \vec{v}_0 + \lambda_1^k \vec{u}_1 \vec{v}_1 + o(\lambda_1^k)$.

Characteristic map illustrated

In the barycentric space of control points the subdivision matrices look as follows (note that I follow IFS habits and the matrices are transposed with respect to the subdivision surfaces tradition):

$$T_0 = \begin{pmatrix} 3/4 & 1/4 & 0 \\ 1/4 & 3/4 & 3/4 \\ 0 & 0 & 1/4 \end{pmatrix} \qquad T_1 = \begin{pmatrix} 1/4 & 0 & 0 \\ 3/4 & 3/4 & 1/4 \\ 0 & 1/4 & 3/4 \end{pmatrix}$$

Figure 2.6 illustrates the action of the transformations T_0 and T_1 on the curve. Right eigenvectors of T_0:

$$\vec{u}_0 = \begin{pmatrix} 1/2 \\ 1/2 \\ 0 \end{pmatrix} \qquad \vec{u}_1 = \begin{pmatrix} -1 \\ 1 \\ 0 \end{pmatrix} \qquad \vec{u}_2 = \begin{pmatrix} 1 \\ -2 \\ 1 \end{pmatrix}$$

Left eigenvectors of T_0:

$$\vec{v}_0 = \begin{pmatrix} 1 & 1 & 1 \end{pmatrix} \qquad \vec{v}_1 = \begin{pmatrix} -1/2 & 1/2 & 3/2 \end{pmatrix} \qquad \vec{v}_2 = \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}$$

As in the background section, our eigenvectors are scaled in the way $\vec{u}_i \cdot \vec{v}_i = 1$.

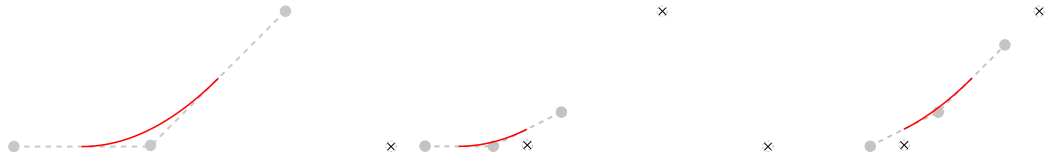


Figure 2.6: Left: uniform quadratic B-spline curve with its control points. Middle and right: image of the curve under subdivisions T_0 and T_1, respectively

Reif [Rei93, Rei95] defined the characteristic map for subdivision surfaces as a 2D spline function. For the case of a quadratic B-spline curve with 3 control points the domain of the map is the segment $[0, 1]$ and the image is the curve projected to $\mathbb{R}$ with control points $\begin{pmatrix} P_0 & P_1 & P_2 \end{pmatrix} = \vec{v}_1 = \begin{pmatrix} -1/2 & 1/2 & 3/2 \end{pmatrix}$. Figure 2.7 shows the image of the characteristic map. In order to show

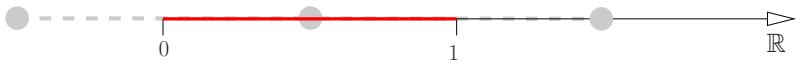


Figure 2.7: The image of the characteristic map for a quadratic B-spline is shown in red.

that quadratic B-splines are smooth it suffices to prove that the characteristic map is regular and injective.

Let us illustrate the motivations behind this definition of characteristic map. According to corollary 1 the repeated action of T_0 can be expressed as $T_0^k = \vec{u}_0\vec{v}_0 + 1/2^k\vec{u}_1\vec{v}_1 + o(1/2^k)$. In few iterations the action of T_0 on the subspace spanned by $\vec{u}_2$ becomes negligible, so let us discard this action and consider modified version of T_0 defined as follows:

$$T = \vec{u}_0\vec{v}_0 + \vec{u}_1\vec{v}_1 = \begin{pmatrix} 1/2 \\ 1/2 \\ 0 \end{pmatrix} \begin{pmatrix} 1 & 1 & 1 \end{pmatrix} + \begin{pmatrix} -1 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} -1/2 & 1/2 & 3/2 \end{pmatrix} =$$
$$= \begin{pmatrix} 1/2 & 1/2 & 1/2 \\ 1/2 & 1/2 & 1/2 \\ 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 1/2 & -1/2 & -3/2 \\ -1/2 & 1/2 & 3/2 \\ 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & 2 \\ 0 & 0 & 0 \end{pmatrix}$$

So we discarded the action of T_0 onto $\text{span}\{\vec{u}_2\}$ and removed the contraction factor $1/2$.

Figure 2.8 shows the action of the transformation T on the control polygon of the B-spline curve. Note that the control points of the curve were chosen to have $\vec{u}_1$ and $\vec{u}_2$ as an orthonormal basis of $\mathbb{R}^2$ once projected to plane:

$$P_0 = \begin{pmatrix} -1/2 \\ 0 \end{pmatrix} \qquad P_1 = \begin{pmatrix} 1/2 \\ 0 \end{pmatrix} \qquad P_2 = \begin{pmatrix} 3/2 \\ 1 \end{pmatrix}$$

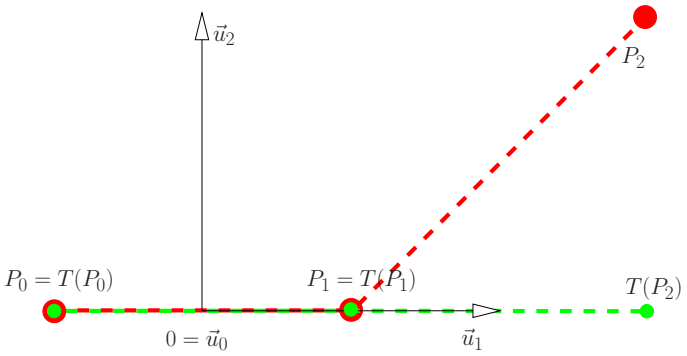


Figure 2.8: Control polygon of the quadratic B-spline curve (red) and its image under the transformation T (green). The corresponding curves are not shown in order not to occlude the image. In the modeling space T is simply the orthogonal projection to the x-axis.

The transformation T is simply a projection onto the subspace spanned by $\vec{u}_0$ and $\vec{u}_1$. In our example we can restate the condition by Reif: if the curve defined by the red control polygon in

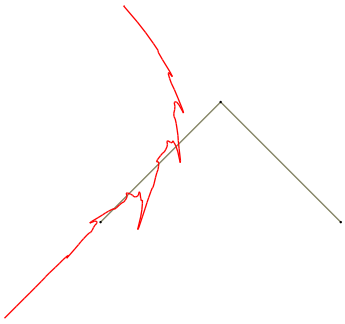


Figure 2.9: A curve with half-tangent at the endpoints and a non-injective characteristic map.

Figure 2.8 (compare the figure to figure 2.7!) is projected to the curve defined by the green control polygon without foldovers, then the original curve is smooth. Note that x-coordinates of green points are given by the vector $\vec{v}_1$, namely -1/2, 1/2, 3/2.

To conclude this illustration, the characteristic map is nothing else but restriction of the subdivision matrix to its sub-dominant eigenspace. Reif and Peters [Rei95] proved that if the characteristic map is regular and invertible then the resulting surface is C^1-continuous. Prautzsch shows that under certain conditions on eigenvalues and eigenvectors of the subdivision matrices the subdivision schemes with a regular and invertible characteristic map produce G^k-continuous surface at the extraordinary points.

However the characteristic map (in general) is not injective for fractal shapes. As an example we consider the curve presented in figure 2.9. The end points of curve are obtained with the following subdivision mask:

$$S = \begin{pmatrix} \frac{5}{4} & -\frac{1}{4} & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & \frac{5}{4} & -\frac{1}{4} \end{pmatrix}.$$

As any other curve obtained with subdivision, such curve is self-similar. According to [BGN08] the half-tangents exist at the endpoints, however as we are going to show the characteristic map is not injective.

Let us calculate the eigenvectors of S and associated eigenvalues:

$$\lambda_0 = 1, \lambda_1 = 3/4, \lambda_2 = -1/4,$$

$$\vec{v}_0 = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, \ \vec{v}_1 = \begin{pmatrix} 1 \\ 2 \\ 5/2 \end{pmatrix}, \ \vec{v}_2 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}.$$

Roughly speaking, the characteristic map is a projection of this curve onto left black segment. This projection has foldovers due to the negative eigenvalue $-1/4$ and therefore the characteristic map is not injective. Thus results by Reif *et al.* do not apply even if the curve has tangents at the endpoints. Moreover, it is quite difficult to check the injectivity of characteristic maps in general. The rest of this chapter describes our approach on testing whether tangents exist or not.

2.2 Analysis of IFS attractors

In this section we study behavior of IFS attractors for the fixed point of some transformation T. We give a series of lemmas, that describe this behavior depending on the eigenvalues and eigenvectors of T. We give our results for transformations expressed in the barycentric coordinates BI^m. First of all, it allows us to express translations as linear transformations, thus simplifying the treatment; second (in general settings) smoothness of, say, spline surfaces depend on the basis functions and not on particular projections to the modelling space.

Thus, we have a linear transformation T and we want to study the behavior of IFS attractor having T as one of its transformations; the point of interest is the fixed point of $p = T(p)$. There are two major cases to consider: when T has a full set of eigenvectors (section 2.2.1) and when it does not (section 2.2.2).

2.2.1 Complete set of eigenvectors

Let us start with the simplest case: when transformation T possesses a complete set of **real** eigenvectors. It is quite obvious that in general the differential behavior is given by the direction of slowest contraction of T. Dominant eigenvector for transformations in barycentric coordinates gives the fixed point (dominant eigenvalue is always 1), thus sub-dominant eigenvector gives us the direction with slowest rate of contraction. There is one more condition to verify, though, namely empty intersection of the attractor with the subspace dual to the subdominant eigenvector, the following section provides an illustration.

Real eigenvalues

Lemma 1. *Let T be an operator on BI^m with a complete set of real eigenvectors $\vec{v}_0, \vec{v}_1, \ldots, \vec{v}_{m-1}$ along with their respective eigenvalues $\lambda_0 = 1 > |\lambda_1| > |\lambda_2| \geq \ldots \geq |\lambda_{m-1}|$, $\forall i\ \lambda_i \in \mathbb{R}$. Let $p = \vec{v}_0$ be the fixed point of T and $E = p + \mathrm{span}\{\vec{v}_1\}$ be the affine subspace generated by point p and vector $\vec{v}_1$. Given a point $x \in BI^m : \vec{px} \notin \mathrm{span}\{\vec{v}_2, \ldots, \vec{v}_{m-1}\}$, we want to show that*

$$\lim_{n\to\infty} \frac{\|T^n(x) - (T^n(x))_{\perp E}\|}{\|T^n(x) - p\|} = 0,$$

where $(T^n(x))_{\perp E}$ is the orthogonal projection of $T^n(x)$ onto E.

Proof. Let us express $T^n(x)$ in the eigenbasis of T: $T^n(x) = p + \sum_{i=1}^{m-1} \lambda_i^n x^{(i)} \vec{v}_i$. Then the ratio we are interested in can be written as follows:

$$\frac{\|T^n(x) - (T^n(x))_{\perp E}\|}{\|T^n(x) - p\|} = \frac{\|p + \sum_{i=1}^{m-1} \lambda_i^n x^{(i)} \vec{v}_i - p - \lambda_1^n x^{(1)} \vec{v}_1\|}{\|p + \sum_{i=1}^{m-1} \lambda_i^n x^{(i)} \vec{v}_i - p\|} = \frac{\|\sum_{i=2}^{m-1} \lambda_i^n x^{(i)} \vec{v}_i\|}{\|\sum_{i=1}^{m-1} \lambda_i^n x^{(i)} \vec{v}_i\|}.$$

For any choice $\varepsilon > 0$ it is possible to find N such that for all $n > N$ the ratio is inferior to ε. Indeed, if we divide both the numerator and denominator by $|\lambda_1|$, there will be non-vanishing term in the denominator, while the numerator approaches zero exponentially fast. □

Corollary 2. Any IFS having T as one of its transformations, has its attractor $\mathcal{A}$ tangent to the affine subspace E if the ring $R_1 = \overline{\mathcal{A} \setminus T(\mathcal{A})}$, does not intersect $p + \mathrm{span}\{\vec{v}_2, \ldots, \vec{v}_{m-1}\}$.

Proof. The proof is obvious as long as we know the following fact: R_1 is compact not intersecting $p + \mathrm{span}\{\vec{v}_2, \ldots, \vec{v}_{m-1}\}$. Therefore for any choice of ε it is possible to find $N(\varepsilon)$ such that for any $n > N(\varepsilon)$ ring R_n lie in the cone defined by E and ε. □

Let us illustrate why it is important for the ring R_1 not to intersect the horizontal line on the figure 2.4. Let us modify the IFS by adding a fifth transformation:

$$T_4\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \begin{bmatrix} 0.75 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} .25 \\ 0 \end{bmatrix}.$$

The corresponding attractor is shown in figure 2.10 (compare it to the figure 2.3). Since it intersects the horizontal axis $p + \mathrm{span}\{\vec{v}_2\}$, any point on this axis will remain on it under repeated action of T_0, leading to the absence of non-trivial tangent subspace.

Another more simple example would be the IFS consisting of $\{T_0, T_4\}$. Its attractor is the horizontal segment between points $(0, 0)$ and $(1, 0)$. This segment lies completely inside the horizontal axis $p + \mathrm{span}\{\vec{v}_2\}$, therefore the tangent subspace is given by $\vec{v}_2$ and not $\vec{v}_1$!

Complex sub-dominant eigenvalues

Now we are going to study the case where a pair of complex sub-dominant eigenvalues is present. The goal of this section is to show that for if $\vec{v}$ is an eigenvector corresponding to the sub-dominant eigenvalue $\lambda \in \mathbb{C}$, then $\mathrm{span}\{\mathrm{Re}\,\vec{v}, \mathrm{Im}\,\vec{v}\}$ is a good candidate to be a tangent.

Lemma 2. *Let T be an operator on BI^m with a complete set of eigenvectors $\vec{v}_0, \vec{v}_1, \ldots, \vec{v}_{m-1}$ along with their respective eigenvalues $\lambda_0 = 1 > |\lambda_1| = |\lambda_2| > |\lambda_3| \geq \ldots \geq |\lambda_{m-1}|$, where $\lambda_1 = \bar{\lambda}_2$, $\lambda_1, \lambda_2 \in \mathbb{C}$ and all other eigenvalues are real. Let $p = \vec{v}_0$ be the fixed point of T and*

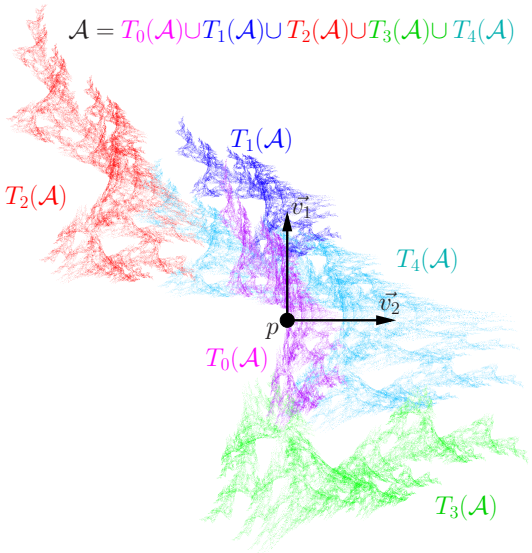


Figure 2.10: This attractor intersects the horizontal axis $p + \text{span}\{\vec{v}_2\}$, therefore $p + \text{span}\{\vec{v}_1\}$ is not tangent to the attractor in point p.

$E = p + \text{span}\{\text{Re}\,\vec{v}_1, \text{Im}\,\vec{v}_1\}$. *Given a point* $x \in BI^m : p\vec{x} \notin \text{span}\{\vec{v}_3, \ldots, \vec{v}_{m-1}\}$, *we want to show that*

$$\lim_{n\to\infty} \frac{\|T^n(x) - (T^n(x))_{\perp E}\|}{\|T^n(x) - p\|} = 0,$$

where $(T^n(x))_{\perp E}$ *is the orthogonal projection of* $T^n(x)$ *onto* E.

Proof. The idea of the proof is exactly the same as in lemma 1. First of all, we need to choose a basis of our space. It is easy to see that if a matrix has a complex eigenvalue λ, then its conjugate $\bar{\lambda}$ is also an eigenvalue. The same goes for eigenvectors: if $\vec{v}$ is an eigenvector corresponding to the eigenvalue $\lambda \in \mathbb{C}$ then $\bar{\vec{v}}$ is an eigenvector that corresponds to the eigenvalue $\bar{\lambda}$. Also for every pair of complex eigenvalues λ, $\bar{\lambda}$ there exists a subspace invariant to T. Indeed, let us consider a complex eigenvalue $\lambda = a + bi$ and the corresponding eigenvector $\vec{v} = \vec{x} + \vec{y}i$, where $a, b \in \mathbb{R}$ and $\vec{x}, \vec{y} \in \mathbb{R}^m$:

$$T(\vec{v}) = T(\vec{x}) + iT(\vec{y}) = (a + bi)(\vec{x} + \vec{y}i) = a\vec{x} - b\vec{y} + i(a\vec{y} + b\vec{x}).$$

So if we consider $T(\alpha\vec{x} + \beta\vec{y})$ we obtain:

$$T(\alpha\vec{x} + \beta\vec{y}) = \alpha(a\vec{x} - b\vec{y}) + \beta(a\vec{y} + b\vec{x}) = (a\alpha + b\beta)\vec{x} + (a\beta - b\alpha)\vec{y}.$$

Thus we choose to complete $\vec{v}_0, \vec{v}_3, \ldots, \vec{v}_{m-1}$ with $\text{Re}\,\vec{v}_1$ and $\text{Im}\,\vec{v}_1$ to form a basis of our space. Then we can express $T^n(x)$ in this basis: $T^n(x) = p + x^{(1)}\,\text{Re}\,\lambda_1^n\vec{v}_1 + x^{(2)}\,\text{Im}\,\lambda_1^n\vec{v}_1 + \sum_{i=3}^{m-1} \lambda_i^n x^{(i)}\vec{v}_i$. As in previous lemma, it is easy to see that the ratio $\frac{\|T^n(x)-(T^n(x))_{\perp E}\|}{\|T^n(x)-p\|}$ has non-vanishing denominator as n goes to infinity, while the numerator vanishes. □

It is straightforward to extend this reasoning to allow considered operator T to have complex eigenvalues other that the sub-dominant ones. Things become a little bit more difficult if the operator does not possess a complete set of eigenvectors as explained in the following section, but now let us provide an example of application of the above lemma.

An example: a spiral touching a tangent plane In this paragraph we study a curve which does not possess a tangent line but a tangent plane. The curve will be obtained by a modified Doo-Sabin scheme, we have already met it in the previous chapter, refer to figure 1.25 for the control graph. Figure 2.11 illustrates the subdivision process. The control mesh consists of two faces: a triangle and a quad; subdivided mesh converges to a limit curve.

The barycentric space associated with the state $\hat{e}$ has 5 dimensions, whereas e has 6 dimensions. The matrices corresponding to the standard Doo-Sabin subdivision scheme can be expressed as

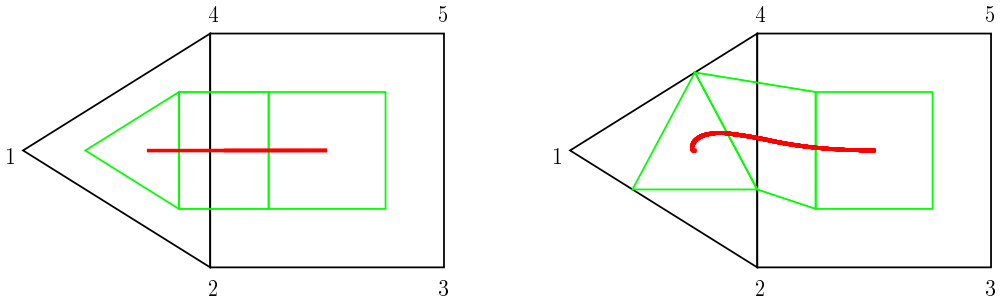


Figure 2.11: Left image shows the Doo-Sabin subdivision scheme applied on the control mesh shown in black, the green mesh is obtained by one iteration and the red curve is the limit curve. Right image shows the subdivision with modified weights.

follows:

$$T_0^{\hat{e}} = \begin{pmatrix} 2/3 & 1/6 & 0 & 1/6 & 0 \\ 1/6 & 2/3 & 9/16 & 1/6 & 3/16 \\ 0 & 0 & 3/16 & 0 & 1/16 \\ 1/6 & 1/6 & 3/16 & 2/3 & 9/16 \\ 0 & 0 & 1/16 & 0 & 3/16 \end{pmatrix}, \; T_1^{\hat{e}} = \begin{pmatrix} 1/6 & 0 & 0 & 1/6 & 0 & 0 \\ 2/3 & 9/16 & 3/16 & 1/6 & 3/16 & 1/16 \\ 0 & 3/16 & 9/16 & 0 & 1/16 & 3/16 \\ 1/6 & 3/16 & 1/16 & 2/3 & 9/16 & 3/16 \\ 0 & 1/16 & 3/16 & 0 & 3/16 & 9/16 \end{pmatrix}$$

The matrices T_0^e and T_1^e are obtained as a tensor product of quadratic B-spline and quadratic B-spline vertex. Left image of figure 2.11 shows one subdivision. For example, the green triangle is obtained from the black one with the weights defined in 1st, 2nd and 4th columns of the matrix $T_0^{\hat{e}}$.

For this section we modify the scheme, namely the green triangle is slightly rotated (refer to the right image of the figure 2.11). The modified weights are following:

$$T_0^{\hat{e}} = \begin{pmatrix} 2/3 & 0 & 0 & 1/3 & 0 \\ 1/3 & 2/3 & 9/16 & 0 & 3/16 \\ 0 & 0 & 3/16 & 0 & 1/16 \\ 0 & 1/3 & 3/16 & 2/3 & 9/16 \\ 0 & 0 & 1/16 & 0 & 3/16 \end{pmatrix}, \; T_1^{\hat{e}} = \begin{pmatrix} 0 & 0 & 0 & 1/3 & 0 & 0 \\ 2/3 & 9/16 & 3/16 & 0 & 3/16 & 1/16 \\ 0 & 3/16 & 9/16 & 0 & 1/16 & 3/16 \\ 1/3 & 3/16 & 1/16 & 2/3 & 9/16 & 3/16 \\ 0 & 1/16 & 3/16 & 0 & 3/16 & 9/16 \end{pmatrix}$$

Let us study the differential behavior of the curve at the fixed point of $T_0^{\hat{e}}$. The operator $T_0^{\hat{e}}$ has 5 eigenvalues:

$$\lambda_0 = 1, \;\; \lambda_1 = 1/2 - i\sqrt{3}/6, \;\; \lambda_2 = 1/2 + i\sqrt{3}/6, \;\; \lambda_3 = 1/4, \;\; \lambda_4 = 1/8$$

and the corresponding eigenmatrix is

$$(\vec{v}_0, \vec{v}_1, \vec{v}_2, \vec{v}_3, \vec{v}_4) = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & -1/2 + i\sqrt{3}/2 & -1/2 - i\sqrt{3}/2 & -41/4 & 5/8 \\ 0 & 0 & 0 & 21/4 & -43/24 \\ 1 & -1/2 - i\sqrt{3}/2 & -1/2 + i\sqrt{3}/2 & -5/4 & -13/8 \\ 0 & 0 & 0 & 21/4 & 43/24 \end{pmatrix}$$

Lemma 2 tells us that the plane spanned on $(\operatorname{Re}\vec{v}_1, \operatorname{Im}\vec{v}_1)$ is the tangent plane to our curve if there is no point in the ring R_0 with zero coordinates along the vectors $\operatorname{Re}\vec{v}_1, \operatorname{Im}\vec{v}_1$ in the basis

$$(\vec{v}_0, \operatorname{Re}\vec{v}_1, \operatorname{Im}\vec{v}_1, \vec{v}_3, \vec{v}_4) = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & -1/2 & -\sqrt{3}/2 & -41/4 & 5/8 \\ 0 & 0 & 0 & 21/4 & -43/24 \\ 1 & -1/2 & \sqrt{3}/2 & -5/4 & -13/8 \\ 0 & 0 & 0 & 21/4 & 43/24 \end{pmatrix}$$

How do we check that R_0 does not intersect $p + \operatorname{span}\{\vec{v}_3, \vec{v}_4\}$? First of all, our subdivision matrices have non-negative entries, therefore we know that the limit curve belongs to the convex hull of its control points (black mesh in the figure 2.11). The coordinates of the control points in

the standard basis are given by a 5×5 identity matrix $I_{5\times5}$. Therefore the ring R_0 belongs to the convex hull of 6 control points with coordinates $I_{5\times5} \times T_1^{\hat{e}}$ in the standard basis (two green quads in the figure 2.11). The coordinates in the eigenbasis can be obtained as follows:

$$(\vec{v_0}, \operatorname{Re}\vec{v}_1, \operatorname{Im}\vec{v}_1, \vec{v}_3, \vec{v}_4)^{-1} \times I_{5\times5} \times T_r = \begin{pmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ -\frac{1}{3} & -\frac{97}{301} & -\frac{271}{903} & 0 & -\frac{118}{301} & -\frac{460}{903} \\ -\frac{\sqrt{3}}{9} & -\frac{67\sqrt{3}}{301} & -\frac{302\sqrt{3}}{903} & \frac{2\sqrt{3}}{9} & \frac{24\sqrt{3}}{301} & -\frac{85\sqrt{3}}{903} \\ 0 & \frac{1}{42} & \frac{1}{14} & 0 & \frac{1}{42} & \frac{1}{14} \\ 0 & -\frac{3}{86} & -\frac{9}{86} & 0 & \frac{3}{86} & \frac{9}{86} \end{pmatrix}$$

Second and third rows give coordinates of 6 control points along the vectors $(\operatorname{Re}\vec{v}_1, \operatorname{Im}\vec{v}_1)$. We need to check if there is a point with coordinates $(\cdot, 0, 0, \cdot, \cdot)$ in the convex hull. Any point of the convex hull can be expressed as a weighted sum of hull vertices. Thus our problem can be restated as follows: find weights $(\alpha_0, \alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5)$ and coordinates $(\beta_0, \beta_3, \beta_4)$ subject to constraints $\sum_{i=0}^{5} \alpha_i = 1$, $\alpha_i \geq 0$ such that

$$\begin{pmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ -\frac{1}{3} & -\frac{97}{301} & -\frac{271}{903} & 0 & -\frac{118}{301} & -\frac{460}{903} \\ -\frac{\sqrt{3}}{9} & -\frac{67\sqrt{3}}{301} & -\frac{302\sqrt{3}}{903} & \frac{2\sqrt{3}}{9} & \frac{24\sqrt{3}}{301} & -\frac{85\sqrt{3}}{903} \\ 0 & \frac{1}{42} & \frac{1}{14} & 0 & \frac{1}{42} & \frac{1}{14} \\ 0 & -\frac{3}{86} & -\frac{9}{86} & 0 & \frac{3}{86} & \frac{9}{86} \end{pmatrix} \times \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \end{pmatrix} = \begin{pmatrix} \beta_0 \\ 0 \\ 0 \\ \beta_3 \\ \beta_4 \end{pmatrix}.$$

The second row of the linear system implies $\alpha_0 = \alpha_1 = \alpha_2 = \alpha_4 = \alpha_5 = 0$ and therefore from the third row we have $\alpha_3 \frac{2\sqrt{3}}{9} = 0 \Rightarrow \alpha_3 = 0$. This is incompatible with the constraint $\sum_{i=0}^{5} \alpha_i = 1$ and thus the convex hull does not include points with coordinates $(\cdot, 0, 0, \cdot, \cdot)$.

So, the conditions of the lemma 2 are verified and the plane spanned by the vectors $(\operatorname{Re}\vec{v}_1, \operatorname{Im}\vec{v}_1)$ is indeed a tangent plane. To better illustrate the existence of such a plane, right image of figure 1.27 shows the suitcase corner subdivision. The studied curve (present 3 times) is highlighted in the limit surface.

2.2.2 Incomplete set of eigenvectors

Previous section shows how to analyze differential behavior for a linear transformation with a complete eigenbasis, unfortunately in practice this condition is seldom achieved. In this section we use generalized eigenvectors to complete the eigenbasis and we show how to use it to express powers of a matrix.

A primer on eigenvectors

This section reminds necessary definitions of generalized eigenvectors and some of their properties. For a linear operator A acting on a vector space $\mathbb{R}^n$ an eigenvalue λ is a non-zero number for which a vector $\vec{v}$ exists such that $A\vec{v} = \lambda\vec{v}$. All eigenvectors corresponding to certain eigenvalue λ lie within the kernel of the linear map $A - \lambda I$, where I is a identity map over $\mathbb{R}^n$. Similarly the generalized eigenvectors of order k lie in the kernel of $(A - \lambda I)^k$ where k is a positive integer. Note that for $k = 1$ we obtain the standard eigenvectors. Also, since generalized eigenvectors of order k are solution of the linear equation

$$(A - \lambda I)^k \vec{x} = 0,$$

they are also related with eigenvectors of order $k - 1$ with the following relation:

$$(A - \lambda I)\vec{v}^k = \vec{v}^{k-1},$$

where $\vec{v}^k$ is a generalized eigenvector of order k.

Eigenvalues and multiplicities Consider a matrix $A \in \mathbb{R}^{n\times n}$. The characteristic polynomial $P_A(\lambda) = \det(A - \lambda I)$, where I is an identity matrix, has m roots:

$$\lambda_0, \lambda_1, \ldots, \lambda_{m-1},$$

where $m \leq n$. The values λ_i are called eigenvalues of the matrix A. The characteristic polynomial can be decomposed as: $P_A(\lambda) = \prod_{i=0}^{m-1}(\lambda - \lambda_i)^{r_i}$. The power r_i is called the algebraic multiplicity of the eigenvalue λ_i. The sum of all algebraic multiplicities is equal to the degree of the characteristic polynomial: $\sum_{i=0}^{m-1} r_i = n$.

Let $E_i = \{\vec{v} \in \mathbb{R}^n \text{ such that } A\vec{v} = \lambda_i\vec{v}\}$. E_i is called an eigenspace associated with an eigenvalue λ_i. The dimension of E_i (denoted as $dim(E_i)$) is always less or equal to the algebraic multiplicity of λ_i. Dimension of E_i is also referred to as geometric multiplicity of λ_i.

If $dim(E_i) = r_i$ for all $i = 0, \ldots, n-1$ then a linearly independent set of n eigenvectors can be found. This set forms a basis of $\mathbb{R}^n$.

Chains of generalized eigenvectors When there is not enough linearly independent eigenvectors to compose a basis of space, there is always enough linearly independent generalized eigenvectors that can be used to complete the basis. Given an eigenvalue λ, associated generalized eigenvectors $\vec{v}^1, \ldots, \vec{v}^k$ of corresponding order $1, \ldots, k$ are said to form a chain of generalized eigenvectors of length k. Let us consider a linear operator A acting on $\mathbb{R}^n$ with m distinct eigenvalues $\lambda_0, \lambda_1, \ldots, \lambda_{m-1}$. Then for each of λ_i there exists a chain of generalized eigenvectors, possibly limited to only one eigenvector:

$$\begin{array}{cccc} \vec{v}_0^1, & \vec{v}_1^1, & \ldots, & \vec{v}_{m-1}^1 \\ \vec{v}_0^2 & \vec{v}_1^2 & \ldots & \vec{v}_{m-1}^2 \\ \vdots & \vdots & & \vdots \\ \vdots & \vec{v}_1^{r_1} & & \vdots \\ \vec{v}_0^{r_0} & & & \vec{v}_{m-1}^{r_{m-1}} \end{array}$$

Vectors $\vec{v}_j^i$ are linearly independent and $\sum_{i=0}^{m-1} r_i = n$. So for all $\vec{a} \in \mathbb{R}^n$ there exists $a_{i,j} \in \mathbb{R}$ such that

$$\vec{a} = \sum_{\substack{0 \leq i \leq m-1 \\ 0 \leq j \leq r_i}} a_{i,j}\vec{v}_i^j.$$

Powers of a matrix in a generalized eigenbasis

An action of a linear operator can be expressed in a simple and elegant way though its eigenbasis. Unfortunately, the action in a basis of generalized eigenvectors is a little bit more cumbersome. Let us consider a linear operator T on $\mathbb{R}^k$. Let T have an eigenvalue λ with geometric multiplicity 1 and algebraic multiplicity k. We will denote one of its eigenvectors as $\vec{v}_0$:

$$T\vec{v}_0 = \lambda\vec{v}_0.$$

Let us complete the eigenbasis of $\mathbb{R}^k$ with $k-1$ generalized eigenvectors $\vec{v}_1, \vec{v}_2, \ldots \vec{v}_{k-1}$ defined as follows:

$$T\vec{v}_i = \lambda\vec{v}_i + \vec{v}_{i-1},\ 1 \leq i \leq k-1,$$

and vectors $\vec{v}_0, \vec{v}_1, \ldots, \vec{v}_{k-1}$ are linearly independent and therefore form a basis of $\mathbb{R}^k$. Any vector $\vec{a} \in \mathbb{R}^k$ can be decomposed in the basis: there exist $\{a_i\}_{i=0}^{k-1}$ such that $\vec{a} = \sum_{i=0}^{k-1} a_i\vec{v}_i$.

Lemma 3. *The action of T on an arbitrary vector $\vec{a}$ can be written in the above basis as follows:*

$$T^n\vec{a} = \sum_{j=0}^{k-1}\sum_{i=0}^{k-j-1} C_n^i a_{i+j}\lambda^{n-i}\vec{v}_j.$$

Proof. We will prove this assertion by induction. For the base of our induction ($n = 1$) we get:

$$T\vec{a} = \sum_{j=0}^{k-1} a_j T\vec{v}_j = a_0\lambda\vec{v}_0 + \sum_{j=1}^{k-1} a_j(\lambda\vec{v}_j + \vec{v}_{j-1}) =$$

$$= \lambda a_{k-1}\vec{v}_{k-1} + \sum_{j=0}^{k-2}(\lambda a_j + a_{j+1})\vec{v}_j.$$

It is immediate to verify that our assertion holds true for the base of induction. Now we are going to show that if our hypothesis is true for some n than is it also true for $n+1$:

$$
\begin{aligned}
T^{n+1}(\vec{a}) = T(T^n(\vec{a})) &= T\left(\sum_{j=0}^{k-1}\left(\sum_{i=0}^{k-j-1} C_n^i a_{i+j}\lambda^{n-i}\right)\vec{v}_j\right) = \\
&= \sum_{i=0}^{k-1} C_n^i a_i \lambda^{n-i+1}\vec{v}_0 + \sum_{j=1}^{k-1}\left(\sum_{i=0}^{k-j-1} C_n^i a_{i+j}\lambda^{n-i}\right)(\lambda\vec{v}_j + \vec{v}_{j-1}) = \\
&= \vec{v}_{k-1}\lambda^{n+1}a_{k-1} + \sum_{j=0}^{k-2}\vec{v}_j\left(\sum_{i=0}^{k-j-1} C_n^i a_{i+j}\lambda^{n-i+1} + \sum_{i=0}^{k-j-2} C_n^i a_{i+j+1}\lambda^{n-i}\right) = \\
&= \vec{v}_{k-1}\lambda^{n+1}a_{k-1} + \sum_{j=0}^{k-2}\vec{v}_j\left(\sum_{i=0}^{k-j-1} C_n^i a_{i+j}\lambda^{n-i+1} + \sum_{i=1}^{k-j-1} C_n^{i-1} a_{i+j}\lambda^{n-i+1}\right) = \\
&= \vec{v}_{k-1}\lambda^{n+1}a_{k-1} + \sum_{j=0}^{k-2}\vec{v}_j\left(C_n^0 a_j\lambda^{n+1} + \sum_{i=1}^{k-j-1}(C_n^i + C_n^{i-1})a_{i+j}\lambda^{n-i+1}\right) = \\
&= \vec{v}_{k-1}\lambda^{n+1}a_{k-1} + \sum_{j=0}^{k-2}\vec{v}_j\left(\sum_{i=0}^{k-j-1} C_{n+1}^i a_{i+j}\lambda^{n-i+1}\right) = \\
&= \sum_{j=0}^{k-1}\vec{v}_j\left(\sum_{i=0}^{k-j-1} C_{n+1}^i a_{i+j}\lambda^{n-i+1}\right)
\end{aligned}
$$

Thus the inductive step is performed and this finishes the proof. □

Corollary 3. $\lim\limits_{n\to\infty}\frac{T^n(\vec{a})}{C_n^{k-1}\lambda^n} = a_{k-1}\lambda^{1-k}\vec{v}_0$.

Proof. $\lim\limits_{n\to\infty}\frac{T^n(\vec{a})}{C_n^{k-1}\lambda^n} = \lim\limits_{n\to\infty}\sum\limits_{j=0}^{k-1}\sum\limits_{i-0}^{k-j-1}\frac{C_n^i}{C_n^{k-1}}a_{i+j}\lambda^{-i}\vec{v}_j = a_{k-1}\lambda^{1-k}\vec{v}_0$. The last transition is due to the fact that the only term depending on n is the ratio $\frac{C_n^i}{C_n^{k-1}}$ and it vanishes if $i \neq k-1$ when n grows large. □

From this observation we can make an obvious conclusion:

Corollary 4. If an IFS attractor has T (as above) as one of its transformations, and the fixed point p of T is a single-address point, then $E = p + \operatorname{span}\{\vec{v}_0\}$ is tangent to the attractor in the point p if the attractor does not intersect affine subspace $p + \operatorname{span}\{\vec{v}_1, \ldots, \vec{v}_{k-1}\}$.

In this section we considered a very special case when the transformation T has an eigenvalue with geometric multiplicity 1 and algebraic multiplicity k, however it is straightforward to extend the reasoning to other cases:

- under power iteration of matrix T on a vector a the component corresponding to the eigenvector of order 1 has the slowest rate of contraction within the corresponding chain
- if we compare two components corresponding to two different chains with the same eigenvalue, the one with the longer chain has the slowest rate of contraction.

An example: the Takagi curve

In this paragraph we are going to show how our theoretical results apply to such a well known attractor as the Takagi curve. One example of an IFS with an operator that does not has a complete set of eigenvectors is a Takagi curve (see figure 2.12). For the Takagi curve we chose to use 4 control points (one for each vertex of a rectangle it is projected into), see figure 2.12. In BI^4 the Takagi curve is an attractor of the following IFS:

$$
T_1 = \begin{pmatrix} 1 & 1/2 & 0 & 1/4 \\ 0 & 1/2 & 1/2 & 1/4 \\ 0 & 0 & 1/2 & 1/4 \\ 0 & 0 & 0 & 1/4 \end{pmatrix},\ T_2 = \begin{pmatrix} 1/4 & 0 & 0 & 0 \\ 1/4 & 1/2 & 0 & 0 \\ 1/4 & 1/2 & 1/2 & 0 \\ 1/4 & 0 & 1/2 & 1 \end{pmatrix}.
$$

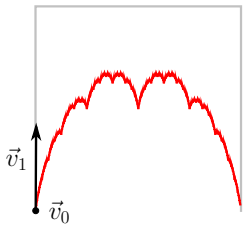


Figure 2.12: The Takagi curve is an attractor of an IFS without a complete set of eigenvectors.

Let us study the tangent at the fixed point of T_1. Transformation T_1 has three different eigenvalues $1, 1/2, 1/4$ with $1/2$ being double eigenvalue. Here are the corresponding eigenvectors:

$$\vec{v}_0 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \vec{v}_1 = \begin{pmatrix} 1/2 \\ -1/2 \\ 0 \\ 0 \end{pmatrix}, \vec{v}_2 = \begin{pmatrix} 1 \\ -1 \\ 1 \\ -1 \end{pmatrix}.$$

We need one generalized eigenvector to complete our eigenbasis:

$$\vec{v}_1^1 = \begin{pmatrix} 1 \\ 0 \\ -1 \\ 0 \end{pmatrix}.$$

To show that sub-dominant eigenvector gives the tangent we need to verify that the ring R_0 for the fixed point of T_1 does not have any point with zero components corresponding to the generalized eigenvector $\vec{v}_1^1$. To verify this we are going to calculate the control polygon for R_0. Then as we did in the example of the spiral curve, we can say that the ring R_0 lies within the convex hull of the control polygon. If the convex hull does not intersect affine subspace $\vec{v}_0 + \text{span}\{\vec{v}_1, \vec{v}_2\}$, then the ring R_0 does not intersect it either.

$$(\vec{v}_0, \vec{v}_1, \vec{v}_1^1, \vec{v}_2) = \begin{pmatrix} 1 & 1/2 & 1 & 1 \\ 0 & -1/2 & 0 & -1 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & -1 \end{pmatrix}, \qquad (\vec{v}_0, \vec{v}_1, \vec{v}_1^1, \vec{v}_2)^{-1} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & -2 & 0 & 2 \\ 0 & 0 & -1 & -1 \\ 0 & 0 & 0 & -1 \end{pmatrix}.$$

So the vertices of the control polygon for R_0 in the basis of generalized eigenvectors are as follows:

$$(\vec{v}_0, \vec{v}_1, \vec{v}_1^1, \vec{v}_2)^{-1} \times T_2 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & -1 & 1 & 2 \\ -1/2 & -1/2 & -1 & -1 \\ -1/4 & 0 & -1/2 & -1 \end{pmatrix}$$

Note that the third component for all four vertices is negative. Therefore there is no point in R_0 with a zero component corresponding to the generalized eigenvector $\vec{v}_1^1$. So the tangent at the fixed point of T_1 is given by the sub-dominant eigenvector $\vec{v}_1$, thus a tangent that passes through first and second control points. If we project the attractor from BI^4 into the unit square using the vertices of the square as control points we obtain a vertical tangent (see figure 2.12).

2.2.3 Conclusion

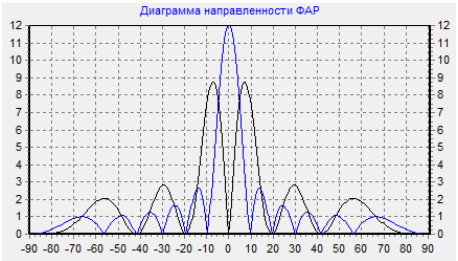
This chapter provides necessary foundations to analyze differential properties of affine IFS attractors. While it focuses solely on points with single periodic address, it is straightforward to extend the analysis to points with multiple addresses. For example, in the above example of Takagi curve mid-point has two addresses: let p be the fixed point of T_1 and q be the fixed point of T_2. Then the midpoint can be obtained in two different ways: $T_1(q) = T_2(p)$. Thus it suffices to verify the behavior in p, in q and to check if under action of T_2 and T_1, correspondingly, the tangent spaces coincide. For the Takagi curve this is the case, thus the midpoint possesses a vertical straight line as a tangent.

There is lots of work left though: currently we do not have a simple way to verify the behaviour for points with aperiodic addresses. For example, the Takagi curve posseses a dense set of points tangent to vertical lines, but we can not elegantly express condition for the rest of the curve. We think that en excellent work by Ingrid Daubechies and Jeffrey Lagarias [DL92] is a good starting point, since reasonably all our IFS matrices form RCP sets.

Next point we plan to work on is differential operators of BC-IFS. Currently BC-IFS attractors are guaranteed to have a given topology, since B-rep structure imposes a certain way of connectivity through its boundary operators. It is far from being obvious, but we work hard on trying to add one more level to the B-rep and to introduce differential boundary operators, thus effectively guaranteeing necessary smoothnes of the attractor.

Summary

Defining shapes by iterative processes allows us to generate new structures with specific properties (roughness, lacunarity), which can be achieved by classic modelling with great difficulty. Our goal is to create an iterative modeller to design fractals described by a BC-IFS; we continue our efforts to develop a set of tools and algorithms that allows to evaluate, to characterize and to analyse different geometric properties (localisation, convex hull, volume, fractal dimension) of fractal shapes. Anton Mishkinis encountered (and partly solved) these difficulties in his PhD thesis [Mis13], there are numerous questions yet to be answered. A simple boolean operation on two fractals is far from being obvious to compute with a given precision and this kind of things is crucial to create a user-friendly geometric modeller.



However our system is is mature enough to start exploration of its real life applications. We plan to study creation of fractal antennas and meta-materials with predetermined electrodynamic properties. This kind of antennas is very promising due to their bandwidth and reduced volume. We start a cooperation with a team of Federal University of Kazan who developed an expertise in this domain. They developed a software to model microstrip antennas and to simulate their electrodynamic characteristics. Thus our first goal is to explore the potential of our geometric model with this simulator to ease the design process. We do hope to hit a large variety of shapes yet undiscovered today thanks to the dissociation of topology and geometry in our model.

Part II

Meshing

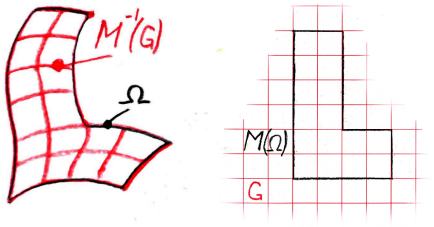
"Nothing clears up a case so much as stating it to another person."

— Sherlock Holmes

Silver Blaze

The state of the art in tetrahedral meshing has now reached a maturity that makes it reasonably easy to mesh arbitrary shapes using existing software [GHS90], [Si15], CGAL ... For hexahedral meshing, the situation is different, and despite important advances, the state of the art is still far away from a general and robust fully automatic solution.

Quite a common technique to re-mesh an object Ω is to find a deformation M that "straightens" Ω, and then to compute the preimage $M^{-1}(G)$ of a regular grid G inside the "straightened" version:



This part of the manuscript has two chapters: chapter 3 proposes a method of generating polycube maps, or, in other words, hexahedral meshes without any singularity inside the mesh. While being quite robust, this method has disadvantages and is not very well suited for re-meshing objects with lots of hard edges, producing distorted hexahedral elements.

At the moment there is no method capable of generating acceptable full-hexahedral meshes automatically. The number of failure cases remains important, even for simple objects that can exhibit some difficult combinatorial aspects of the problem. Despite an important amount of research efforts to solve these issues, designing a complete hexahedral re-meshing algorithm requires to solve many open problems [SJ08]. For this reason, hexahedral-dominant meshing may be an option worth investigating: by relaxing the problem, it still generates a valid result in complicated case where full-hexahedral methods generally fail, at the expense of introducing non-hexahedral elements such as tetrahedra, pyramids and prisms. Thus, we extend the polycubes approach and propose to use additional degrees of freedom: chapter 4 discusses a method of generating hexahedral-dominant meshes.

A major source of frustration we have are papers (often found in SIGGRAPH community and, admittedly, with excellent ideas, but) lacking thorough analysis of limitations and failure cases. We are inspired by excellent papers like the work on quadrangulation by Myles et al [MPZ14] and upset by some others. You will meet the word robustness at least twenty times throughout this part of the manuscript.

Chapter 3

Polycube maps

3.1 Normal constraints for polycube maps

A polycube is an abstract representation of a volume that is very efficient for tasks such as texturing, deformation or re-meshing. To convert a triangulated surface into a polycube, it is required to deform the surface such that its normal becomes always aligned with coordinate axes. A natural choice of the axis to align the normal with is the closest axis to the original surface normal. However, such a deformation satisfying these constraints on the surface normal may not exist. We present an algorithm able to detect such situations and repair the normal constraints. Figure 3.1 presents some cases where the normal equality constraints need to be edited to allow for a valid deformation.

This objective is motivated by the hexahedral re-meshing application proposed by Gregson *et al.* [GSZ11, LVS+13], where invalid axis assignments were not always fixed. Their pipeline (Figure 3.3-up) starts with a tetrahedral mesh and can be summarized as follows:

1. it applies a soft, rotation-based, deformation to roughly align the surface normals with the coordinate axes,

2. it determines which coordinate axis has to be aligned with the normal on each point of the surface,

3. it deforms the mesh to respect these constraints,

4. the geometry is then filled by a Cartesian grid that is mapped to the original mesh position to provide the all-hex re-meshing. Standard post-processing removes too distorted hexahedral elements close to the volume boundary.

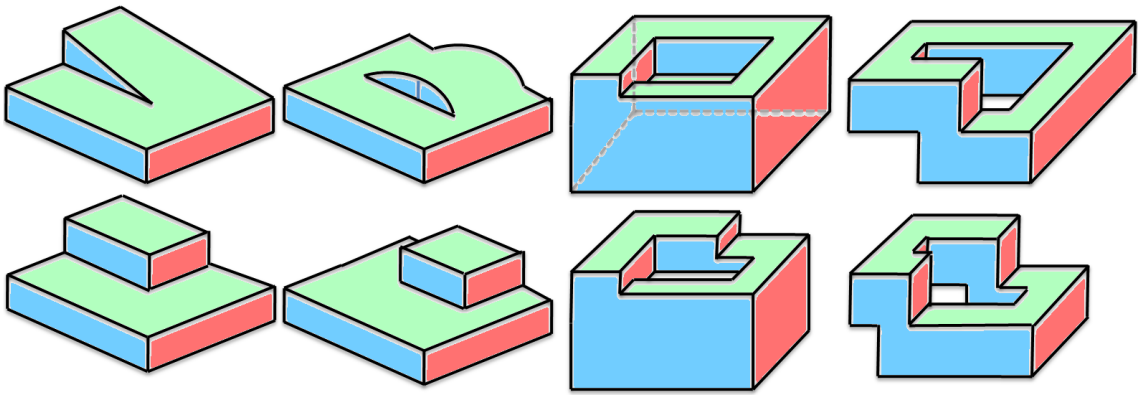


Figure 3.1: **Upper row:** no deformation can align all charts with coordinate axes without squeezing the surface. **Bottom row:** editing the normal constraints by adding "steps" makes the deformation possible.

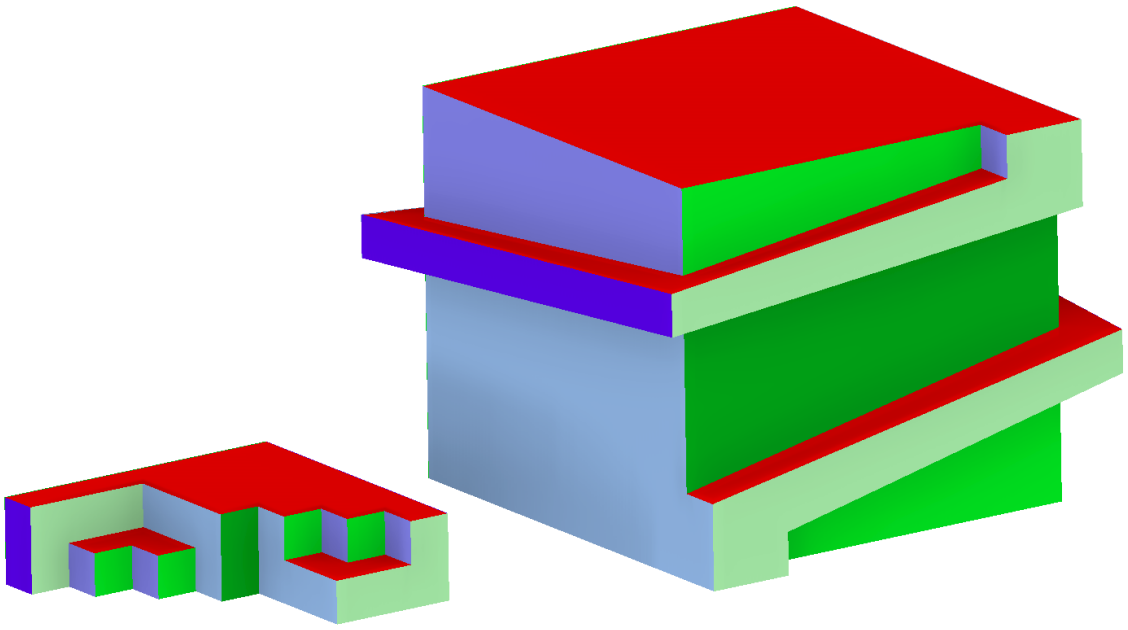


Figure 3.2: Meet the family: a L^∞-screw and its little brother, screw-junior. Both models have the same graph of charts, but clearly one is not a polycube we want for the other one. Slightly different hues are used to simplify the identification of charts.

A major drawback of this method is that during the construction of the polycube, it is not always possible to deform the object subject to the surface orientation constraints (Step 3). Gregson *et al.* repair *some* cases, however these configurations are far from being exhaustive.

To extract polycube structure, Gregson *et al.* label surface triangles according to the closest axis to smoothed triangle normals and group similarly labeled triangles into charts. Then they filter the decomposition by removing small, spurious charts. Since the required properties of orthogonal polyhedra are not fully known, Gregson *et al.* use a set of local heuristics inspired by the results of of Eppstein and Mumford [EM10]. Namely, they look for planar 3-regular 3-connected chart connectivity graphs. Unfortunately, these conditions were expressed for objects of zero genus. Moreover, Figure 3.2 gives a typical example where the method proposed by Gregson *et al.* will inevitably fail. The "L^∞-screw" has a planar 3-regular 3-connected chart graph, but it is impossible to transform it to an orthogonal polyhedron with a continuous transformation. Left image of figure 3.2 shows a xyz polyhedron with *the same* connectivity graph. No local conditions on the chart connectivity graph can guarantee existence of the corresponding polycube. Figure 3.15 shows the polycube for the screw constructed by our algorithm.

Our contribution is an algorithm that edits the normal constraints defined in Step 2, to ensure that there exists a deformation respecting these constraints. As illustrated in Figure 3.3-Middle, the coordinates axis assignment of the object boundary is represented by a new mesh (referred to as meta-mesh and similar to the one introduced in [TACSD06]) that is easier to manipulate than a direct per-point axis assignment. This meta-mesh is embedded in the original surface, so local editing operations of the meta-mesh are equivalent to directly editing the axis assignment on the original surface.

Our method defines a deformation of the object boundary by affecting a geometry to the meta-mesh (Figure 3.3-bottom). For each dimension e_i, each face of the meta-mesh is decomposed into quads, and the i^{th} coordinate of each meta-edge is determined by a constrained optimization algorithm. It ensures that each point of the surface has the prescribed normal when possible. If it is not possible, extra variables and constraints are added to determine where steps have to be created. A step creation is the basic operation that splits a meta-face's quad on the meta-mesh (Figure 3.3-bottom, dimension e_3), and introduces a new chart of axis alignment on the original surface. Our contributions are an analysis of the possibility to deform an object with constrained normal, an algorithm that detects failure cases, and a solution to edit the normal constraints such that the deformation is made possible.

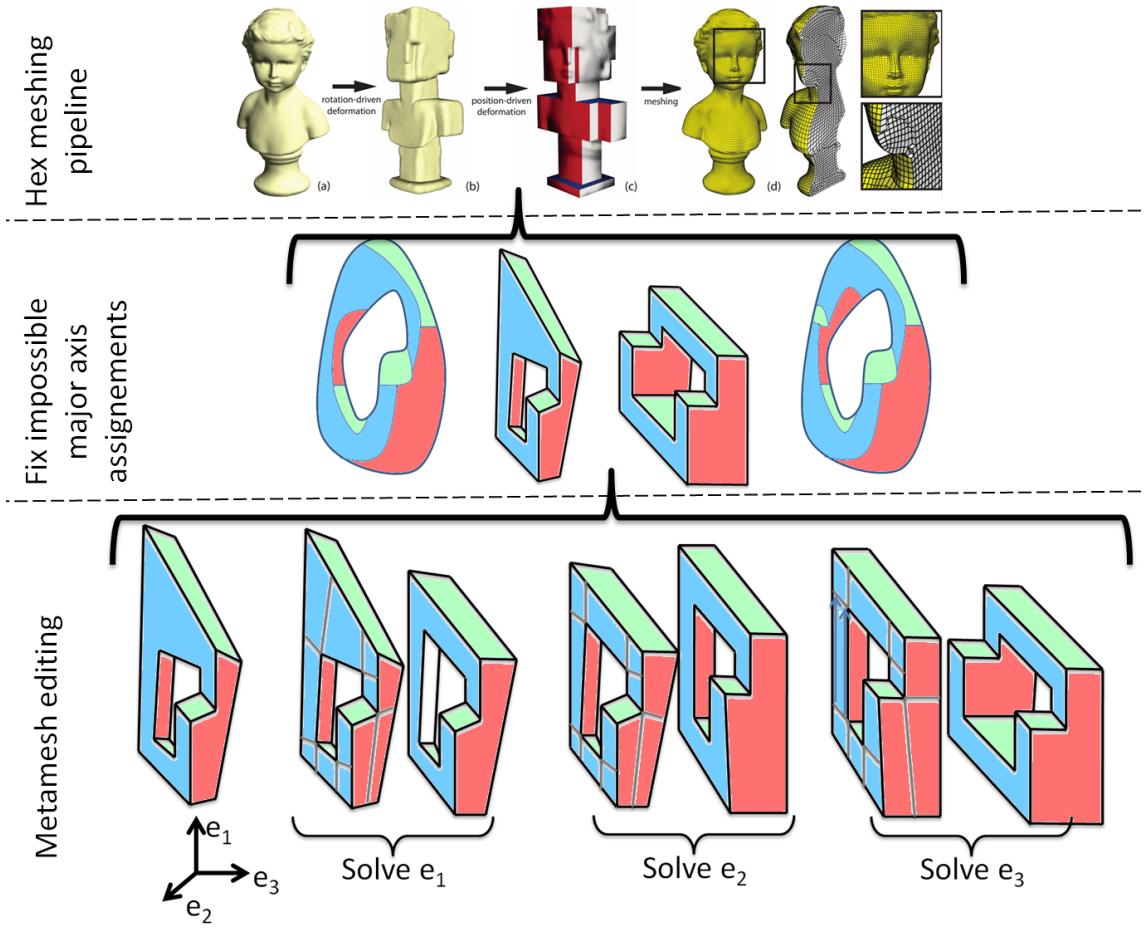


Figure 3.3: **Upper row:** overview of the all-hex meshing pipeline. The volume is deformed to better align its boundary with coordinates axis, this alignment is enforced as a constraint, the deformed mesh is voxelized, and mapped back to the object original shape. **Middle row:** fixing impossible coordinate axis assignments. The axis assignment is edited with aid of a meta-mesh, that is easier to manipulate than the axis assignment. **Bottom row:** meta-mesh editing, for each dimension e_i, each face of the meta mesh is decomposed into quads, and the geometry is resolved for the current dimension. Here, a solution was directly found for e_1 and e_2, but e_3 required to edit the meta-mesh topology.

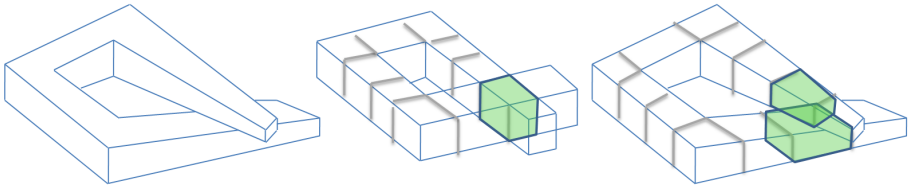


Figure 3.4: **Global overlaps:** given a volume (left image), aligning its boundary with coordinate axis (middle image) can create global overlaps (green voxel), but re-projecting it to the original volume provides a valid hexahedral mesh.

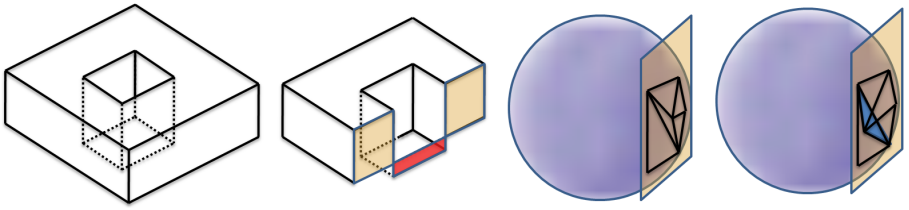


Figure 3.5: **Volume foldovers:** the volume is not defined if the deformed boundary surface has self-intersections, or when the surface has foldovers.

Related works

Polycubes were introduced in computer graphics by Tarini *et al.* [THCM04] for seamless texturing surfaces. They were later used for hexahedral re-meshing of shell meshes [HXH10], then extended to volume re-meshing [GSZ11]. In these applications, the construction of polycubes is not discussed besides in the framework of Gregson *et al.* [GSZ11] we are improving here.

Juncong *et al.* [LJFW08] proposed an algorithm to automatically construct polycubes. It is based on locally match simple polycube primitives, detect features such as protrusion, and merge them all together to produce a polycube. It is able to produce very convincing result for some classic computer graphics meshes, but it seems difficult to handle meshes from CAD/CAM where the features are not well characterized by their proxy.

Ying *et al.* [HWFQ09] have presented an alternative solution based on cutting the volume into slices and iteratively add slices to the polycube. This solution produces fair results for meshes reasonably aligned with coordinate axis.

Another contribution [WYZ+11] in this domain is an optimization of the mapping between a polycube and a surface. It requires a valid polycube to start with, but it can improve any others results by adjusting the mapping.

3.1.1 Problem statement

Let us consider a solid object Ω and a coordinate axis $N(P)$ associated to each point P of the boundary of Ω. Our objective is to determine how to modify $N(P)$ to make it possible to define a deformation D such that for each point P of Ω's boundary, the normal $n(D(P))$ of the deformed volume at point $D(P)$ is equal to $N(P)$.

A natural deformation D has to be one-to-one. However, in our context global overlaps do not impact the final hexahedral mesh (Figure 3.4) because the pre-image of each voxel in the voxelization of the deformed mesh (Step 4 of Gregson *et al.* pipeline) can be a set of hexes in the re-meshed model.

Therefore, the constraints we need are rather local than global i.e. ensuring that D has no foldovers is a sufficient property for our application. As a consequence, we want to produce a locally one-to-one map, so D has to preserve the orientation of the volume i.e. the determinant of its Jacobian matrix $\det(J(D))$ must be strictly positive.

There are two situations (Figure 3.5) that can enforce a negative determinant of the Jacobian matrix of D: if the deformed boundary surface self-intersects (it is not a boundary anymore), or if there are some problems on the surface i.e. $\det(J(D)) \leq 0$ on the surface. This work deals only with surface issues, and a possible solution to prevent surface self-intersections is discussed in Section 3.1.4.

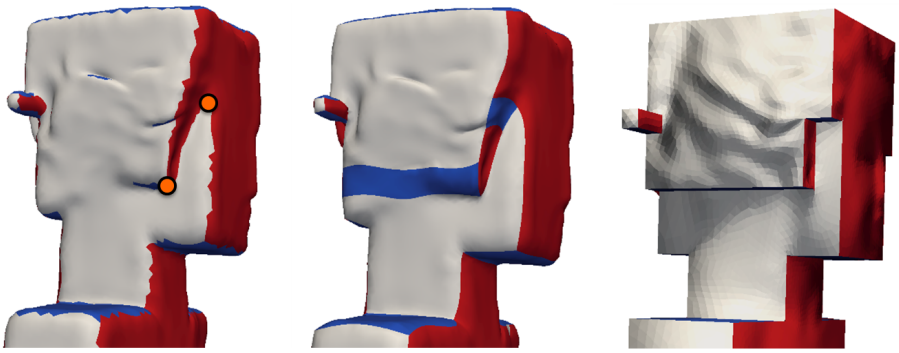


Figure 3.6: **Coordinate axis assignment correction:** colors represent axes that the deformed surface has to be orthogonal to. The volume can not be directly deformed to align its boundary normals with their associated coordinate axis (left). The desired coordinate axis can be modified (middle) to make it possible (right).

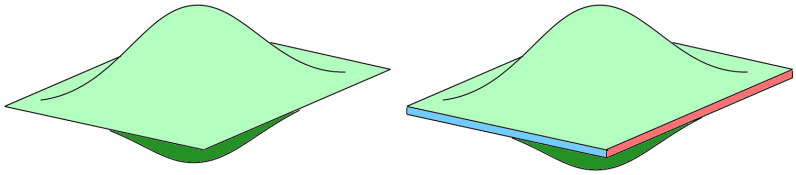


Figure 3.7: Opposite axes can not be assigned to neighboring points (left). A tiny band is introduced to avoid this situation (right).

3.1.2 Formalization

The object boundary is represented by a triangulated surface with an associated coordinate axis $N(f) \in \{e_i, -e_i\}_{i=1}^{i\leq 3}$ associated to each triangle f. We wish to edit N to ensure that the volume can be deformed without foldovers to make its boundary normal $n(f)$ equal to $N(f)$ (Figure 3.6 for an example).

Axis assignment assumption

Our objective is to preserve as much as possible the original axis assignment. Therefore, we limit the possible modification of N to a local operation creating a step. As a consequence, it is impossible for our algorithm to deal with completely random axis assignments. For example, adding steps does not allow to make valid an object where all points are assigned to the same axis. More generally, the genus of the object strongly constraints the set of potentially valid axis assignments and creating steps does not allow to change it.

The intuition is based on the fact that for a smooth object the degree of its Gaussian map is closely related to the genus of the object $deg(N) = 1 - g$, here $deg(N)$ is the degree of the Gauss map and g is the genus (Gauss-Bonnet theorem [Gau00, Bon48]). Given a Gauss map, no local operation (creation of steps in our case) changes the degree. We conjecture that it is possible with a series of local operations it is possible to converge to an object covering the normal sphere exactly $1 - g$ times.

Guided by this intuition, we assume that the initial flagging is given by the closest axis to the surface normal, thus the flagging corresponds to an object of the same genus. Obviously, in Gregson *et al.* pipeline, it is the closest axis to the deformed surface normal.

Unfortunately, degree of a map is defined for continuous maps, and for triangulated surfaces the Gauss map is not continuous. We repair evident cases of under-sampling: hard edges may generate adjacent triangles associated to opposite axes. It corresponds to a degenerated volume that can be directly handled by our algorithm if a new axis is assigned in a tiny band placed on the edge adjacent to conflicting triangles, as illustrated in Figure 3.7. This operation can be interpreted geometrically as hard edges smoothing.

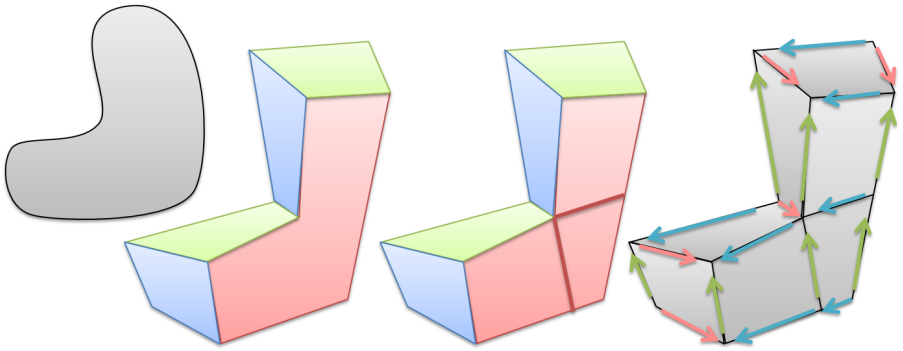


Figure 3.8: **Meta mesh:** the surface can be decomposed into charts such that each chart is associated to a single preferred coordinate axis. The meta mesh has a facet for each such chart (middle left), and is decomposed into quads (middle right). Each edge is then affected to its preferred coordinate axis.

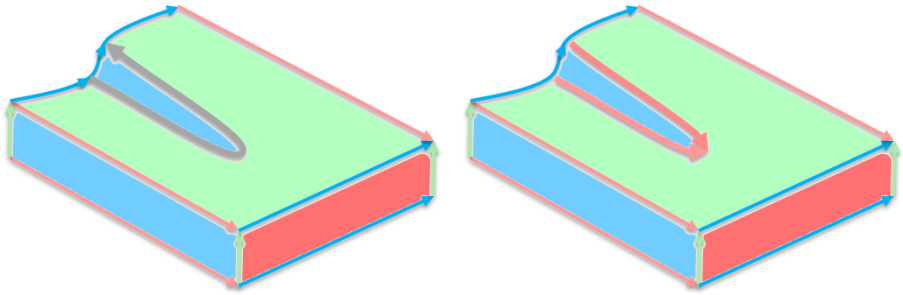


Figure 3.9: **Meta-edge axis assignment:** the coordinate axis assignment on meta-edges can not be deduced from the axis of meta-faces. Here, the gray oriented edge (left) goes in direction e_1 at the beginning and $-e_1$ at the end. A coherent axis assignment requires to split this meta-edge (right).

Meta-mesh

The coordinate axis assignment is difficult to manipulate directly on the object boundary. It is easier to manipulate a $2D$ mesh embedded in the original volume boundary referred to as meta-mesh. By embedded, we mean that the meta-edge geometry is given as paths defined on the original surface as described by Li *et* al. [LLP05a]. The meta-mesh is obtained from the original surface by merging all adjacent triangles f having the same $N(f)$ into a meta-face F. As a consequence, each meta-face F is associated to a coordinate axis $N(F)$, refer to figure 3.8 for an illustration.

To manipulate the meta-mesh geometry, it is also required to define the coordinate axis $N(E)$ associated to each oriented meta-edge E. The coordinate axis $N(E)$ must be orthogonal to its adjacent meta-faces, but this is not sufficient to set its orientation (e_i or $-e_i$). We rely on an heuristic to resolve the ambiguity: if the projection of the edges on e_i is positive, e_i is affected, else $-e_i$ is affected. As a consequence, a meta-edge may be split into several parts during this process if the coordinate axis associated to its original edges is not always the same like in Figure 3.9.

As the meta-mesh is embedded in the original surface, there exists a one-to-one mapping between both. Therefore, if we determine a $3D$ geometry of the meta-mesh that is a boundary (no self intersections) and has no local shrinks ($\det(J(D)) = 0$) nor foldovers ($\det(J(D)) < 0$), then the original surface can be mapped to it, and the deformation inside the volume can be interpolated by mean value coordinates [JSW05, HF06].

As stated in the previous section, we focus here on the local shrinks and fold-over problems (self intersections are discussed in section 3.1.4). To do so, the geometry of the meta-mesh must ensure that the normal $n(F)$ of each meta-face F is constant (flat meta-face) and defined everywhere (no shrink).

3.1.3 Algorithm overview

Having all meta-faces normals equal to their assigned axis can be ensured if all oriented meta-edge E are in the direction to their corresponding coordinate axis $N(E)$, and if the meta-face boundary do not self intersect. Both conditions are enforced by decomposing each meta-face into quads, and solving the quad edges geometry such that their normal equals their assigned axis. The quad decomposition makes it impossible for meta-face boundaries to self-intersect because it would require some quad edges to be oriented in the wrong direction.

When determining the geometry of oriented meta-edges, the e_0, e_1, e_2 coordinates do not interact with each other, making it possible to deal with one dimension at a time. Therefore, algorithm 1 performs on each dimension a decomposition of the meta faces into quads, determines the geometry of the quads edges with respect to their associated axis, possibly determine where to edit the meta-mesh by creation of steps, then create the steps and resolves again the geometry (Algorithm 1).

Algorithm 1: Algorithm overview

```
Data: Mesh m of the object boundary
Data: coordinate axis assignment N
for dim ← 1 to 3 do
    create meta-mesh M;
    decompose each meta-face of M into quads;
    resolve the e_dim coordinate of M's geometry;
    if no valid solution is found then
        add extra degrees of freedom corresponding to step creation;
        resolve the e_dim coordinate of M's geometry;
        create the corresponding steps;
        introduce the steps vertices variables to the system;
        resolve the e_dim coordinate of M's geometry;
    end
end
```

To improve the clarity of the explanations, we will assume from now that the current dimension to be solved is the vertical dimension e_1. Therefore, meta-faces orthogonal to e_1, and meta-edges in direction e_2 and e_3 will be characterized as **horizontal**, and other will be characterized as **vertical**.

The rest of the section presents how the meta-faces are decomposed into quads, an algorithm that determines if it is possible to directly construct an axis aligned meta-mesh, an algorithm that determines where new steps should be created, and the creation of a step in the meta-mesh corresponding to edit the axis assignment on the original surface. Results are then presented and the benefits and drawbacks of the method are discussed.

Meta-face decomposition

Solving the meta-edges geometry enforces meta-faces to be flat. However, it is not sufficient to ensure that the surface normal is defined everywhere, because the boundary may self-intersect. It is prevented by decomposing the meta-faces into quads, and solving the geometry of quad's edges instead of meta-edges.

The decomposition is obtained by tracing curves on the original surface from the meta-vertices (see Figure 3.10). The curve directions are determined by two smooth tangent vector fields, each aligned with the subset of meta-edges that share a same associated axis. These smooth vector fields are computed by Ray *et al.*'s algorithm [RVLL06]. Note that tracing cross-free streamlines on arbitrary triangulated surface is not a trivial task. We need the computed streamlines to be cross-free to ensure the valid quad decomposition. Section 3.2 covers the methodology necessary for the task. The decomposition of a meta-face into quads depends on its orientation and the current axis to be solved.

Vertical meta-faces (See Figure 3.11-Middle) For vertical meta-faces, the quad decompositions are obtained by tracing the set of curves from their vertices and aligned with e_1.

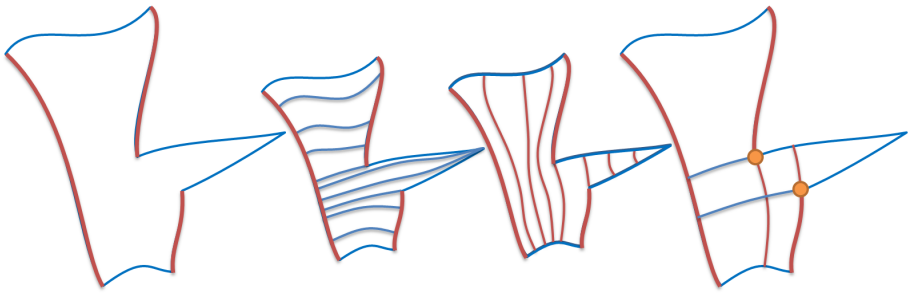


Figure 3.10: **Meta-face decomposition:** the axis associated to the meta-edges (left) allows to interpolate two tangent vector fields in the meta-face (middle). The quad decomposition is performed by following streamlines of these vector fields (right).

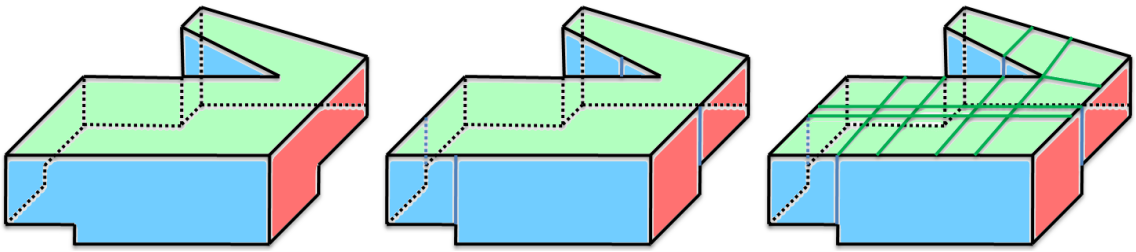


Figure 3.11: When solving in the vertical direction, vertical meta-faces are decomposed first by cutting them in the vertical direction (middle), then the horizontal meta-faces are decomposed by cutting them in the two horizontal directions from all their vertices (right).

When two adjacent meta-edges are associated to opposite coordinates axis, a degenerated quad is created. In this case, an extra edge (Fig 3.11) is generated from a point close to the degenerated edge, and lying in a quad edge adjacent to the degenerated edge. This will force the algorithm to generate a special step.

Horizontal meta-faces (See Figure 3.11-Right) For horizontal meta-faces, the quads decomposition is obtained by tracing curves in both directions from both the original meta-face vertices and the T-vertices introduced during the quad decomposition of other meta-faces.

Why doing an axis dependant quad decomposition? The decomposition of meta-faces into quads is axis dependant. It is motivated by two reasons:

- *It prevents T-junctions from generating impossible steps.* A T-junction splits a quad edge into two. If this quad edge lies on a horizontal meta-face, it will generate two "step" variables able to create different step positions. If the quad strip crossed by the step ends at both extremities with such T-junction, the step to create may have no valid realization (Figure 3.12—Left).
- *It prevents the generation of an over constrained system.* Any single T-junction could be avoided by iteratively splitting the quad adjacent to a single T-junction, but generally this process may not converge.

Our process cuts vertical faces only in the vertical direction to avoid T-junctions between vertical meta-faces, and propagates these T-junctions on horizontal meta-faces to prevent having edge split in these faces. The remaining T-junctions adds a vertex in the middle of an horizontal edge of a vertical quad. As illustrated in Figure 3.12—Right, these remaining T-junction can not produce impossible situations, and only requires to introduce a new slack variable to prevent foldovers.

Check assignment validity

A valid assignment of coordinate axis requires each oriented edge h_i to go in the direction of its associated coordinate axis $N(h_i)$. As meta faces are decomposed into quads, this will also ensure that each quad (and therefore all meta-faces) has the prescribed orientation.

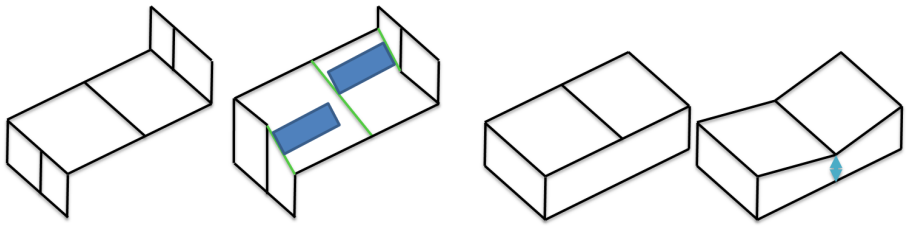


Figure 3.12: A T-junction having a double edge on an horizontal meta-face may generate a set of non horizontal edges that cannot be crossed in a single step (Left). However, if the double edge of the T-junction is associated to another meta-face, it is sufficient to add a slack variable to prevent foldovers in the vertical meta-face (Right).

Let x_i represent the first (vertical) coordinate of vertex i and Δx_{ij} denotes the first coordinate of edge joining vertices i and j.

A valid geometry requires that $\Delta x_{ij} = x_j - x_i$. Moreover, Δx_{ij} must be positive if the edge is associated to e_1, negative if the edge is associated to $-e_1$, and zero $\Delta x_{ij} = 0$ if the edge is horizontal. To find a unique solution, we minimize $\Sigma|\Delta x_{ij}|$ subject to the constraints $\Delta x_{ij} = 0$ if the edge is not associated to e_1 or $-e_1$. If the edge is associated with e_1, we require $\Delta x_{ij} \geq l_{ij}$, where l_{ij} is the geodesic length of the meta-edge h_{ij}. For oriented edges associated with $-e_1$, a constraint on opposing edges ensures that $\Delta x_{ji} > 0$ i.e. $\Delta x_{ij} < 0$.

This problem is solved by the revised simplex method. We use the implementation in the lp_solve library. If the problem has no solution, it means that there exists no deformation without fold-over that aligns the surface normal with the current flagging N. Therefore more degrees of freedom are required, corresponding to the creation of steps, as described in the next section.

Find where to create steps

From the point of view of the meta-mesh geometry, creating a step on a quad strip allows to lift all vertices of one side of the quad strip. This can be represented (before creating the steps) in the meta-mesh geometry by allowing for horizontal edges to have non null e_1 component. However, it is subject to the constraint that on each quad (of the meta-face quad decomposition), opposite edges have the same e_1 geometry.

Simply introducing new geometry variables x_i to horizontal edges and setting the constraint of equality on opposite quad edges would probably create many useless steps. Indeed, even on a simple cube, it is possible to have a double step on meta-faces associated with e_1 and $-e_1$.

Thus, we allow Δx_{ij} for horizontal edges to be non zero, and we include it into the objective function with a large penalty weight. The penalty weight tells us that we always prefer to change the geometry of vertical edges over horizontal edges that would lead to creation of new steps.

All horizontal edges are weighted by $10000 + \epsilon_{ij}$ in the objective function. The value of $\epsilon_{ij} \in [0..1000]$ is set to minimize the step length by setting it to be proportional to the corresponding quad strip length. Before introducing slack variables, the system minimizes:

$$\sum w_{ij}|x_j - x_i| \tag{3.1}$$

where $w_{ij} = 1$ if edge ij is vertical, and $w_{ij} = 10000 + \epsilon_{ij}$ if edge ij is horizontal. The system is subject to the constraints:

- $x_j > x_i$ (resp. $x_j < x_i$) if $N(h_{ij} = e_1)$ (resp. $N(h_{ij} = -e_1)$)
- $x_j - x_i = x'_j - x'_i$ if $h_{i'j'}$ and h_{ij} are opposite quad edges
- $x_j - x_i \geq l_{ij}$ (resp. $x_j - x_i \leq -l_{ij}$) where l_{ij} is the length (geodesic distance) of edge ij, if $N(h_{ij} = e_1)$ (resp. $N(h_{ij} = -e_1)$)

For horizontal edges we split Δx_{ij} into γ^+_{ij} and γ^-_{ij}: $\gamma^+_{ij} + \gamma^-_{ij} = \Delta x_{ij}$, $\gamma^+_{ij} \geq 0$. If in the solution γ^+_{ij} is positive, it means that we will create an ascending step, if γ^-_{ij} is negative, the step will be descending. Note that γ^+_{ij} and γ^-_{ij} can not be non-zero at the same time. Thus, after introduction

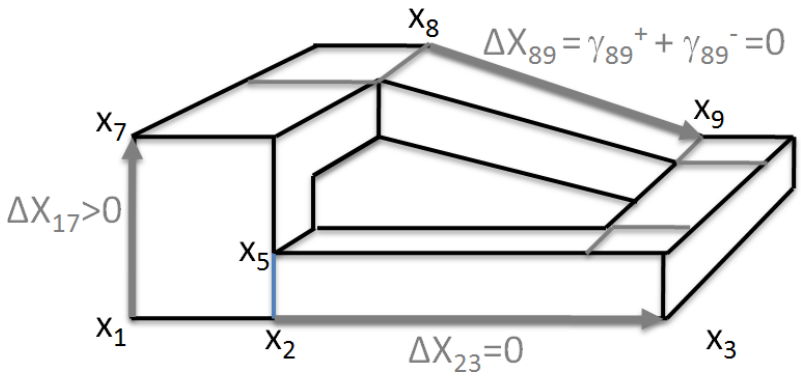


Figure 3.13: **Variables of the system:** x_i are the vertices height, Δx_{ij} are the height component of edges, γ_{ij}^+ and γ_{ij}^- are slack variables associated to horizontal edges.

of slack variables Δx_{ij}, γ_{ij}^+ and γ_{ij}^- (see Figure 3.13), the final system minimizes:

$$\sum_{ij \text{ is vertical}} |\Delta x_{ij}| + \sum_{ij \text{ is horizontal}} (10000 + \epsilon_{ij})(\gamma_{ij}^+ - \gamma_{ij}^-) \tag{3.2}$$

subject to the constraints:

- $\Delta x_{ij} = \Delta x_{i'j'}$ if $h_{i'j'}$ and h_{ij} are opposite quad edges
- $\Delta x_{ij} \geq l_{ij}$ (resp. $\Delta x_{ij} \leq -l_{ij}$) where l_{ij} is the length (geodesic distance) of edge ij, if edge ij is vertical
- $\Delta x_{ij} = \gamma_{ij}^+ + \gamma_{ij}^-$ if if edge ij is horizontal
- $\gamma_{ij}^+ \geq 0$
- $\gamma_{ij}^- \leq 0$

Solving this system provides the height of vertices x_i where all quads have the prescribed normal, or are inclined planes due to steps to be created.

Creation of a step

All modifications of the coordinate axis assignment N rely on a single editing operation on the meta-mesh: the creation of a step (Figure 3.14). This operation is local to a quad strip generated during the decomposition the meta-faces into quads. In the original mesh, it corresponds to define a band crossing all quads of the quad strip, and re-affecting to it another coordinate axis.

The creation of a step invalidates the quad decomposition of meta-faces adjacent to the created band. Figure 3.15 shows an illustration. The left topmost image is the deformed model with its axis flagging. Our linear program tells us that we need to create few steps, thus re-coloring few bands on the "thread" of the screw (middle image, top row). The final polycube is shown at the right, top row. Note that three pink quads on the front of the screw are no longer quads in the final polycube. Creation of steps invalidated the initial quad decomposition, therefore after re-coloring of the bands on the "thread" we re-quadify adjacent charts. New flagging of edges will produce new vector fields. The quadification is shown in the bottom image of figure 3.15. Note the close-up with a very thing quad, we had a lot of problems to decompose the charts into quads until we treated the problem of streamline tracing separately (refer to section 3.2 for a complete description).

Crossing steps: it is possible that two steps need to be created on the same quad: one in each direction. In this case, at the bands intersection, the flagging comes for the value of N for the the highest step, as illustrated in Figure 3.16.

Degenerated quad (jump ramps) require a special treatment because the band (where the coordinate axis must be redefined) must touch an edge of the quad. Indeed, if the band is defined as usual in the middle of the quad, a part of the adjacent meta-face will be shrunk. However, it is not sufficient to place it at the border as proposed by Gregson *et al.*, because another meta-face may have to shrink (Figure 3.18). To overcome these issues, we move the degenerated quad edge prior to placing the step (Figure 3.17).

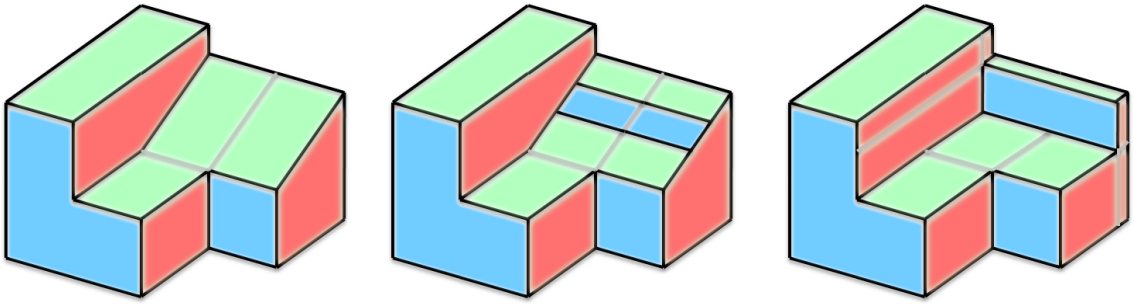


Figure 3.14: **Step creation:** assigning a band of a meta-face to another coordinate axis (middle) creates a step. This relaxes the planarity constraint between two sides of the quad-strip (right).

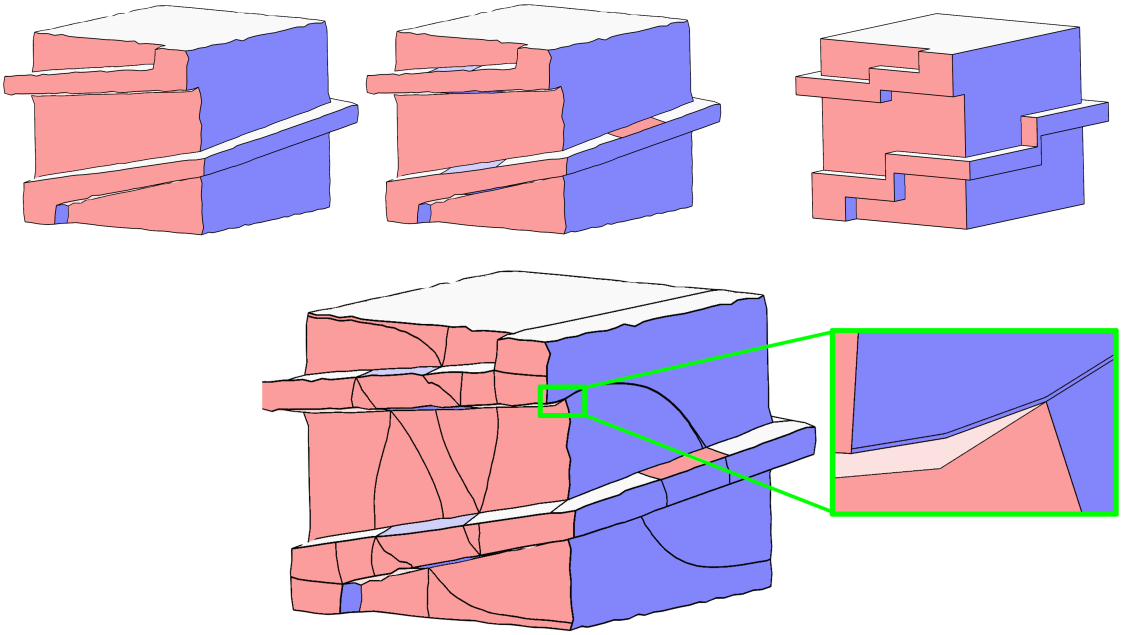


Figure 3.15: Upper row, left-to-right: input model after a deformation, steps to be created, final polycube. Note that creation of steps forces re-quadification of adjacent charts (bottom row).

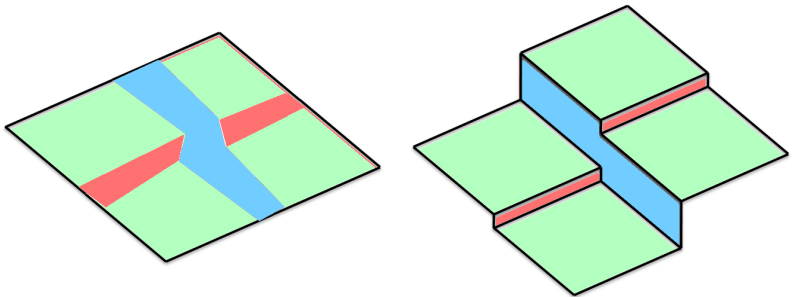


Figure 3.16: **Double step:** two orthogonal steps can be created on the same meta face. The intersection of two bands is assigned to the same flag as the highest of two steps.

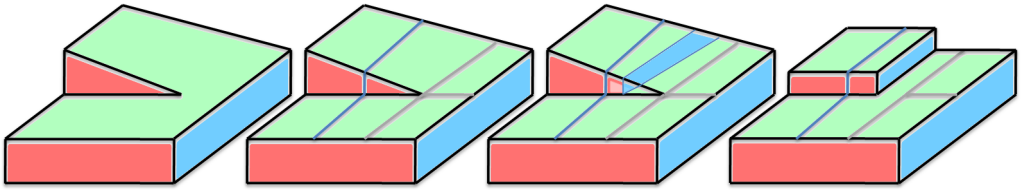


Figure 3.17: When two adjacent meta-edges are assigned to opposite axes, the quad decomposition isolates the degenerated quad (bolt cut). The axis assignment close to the degenerated edge is changed to the axis associated to the adjacent meta-face, and a step is created.

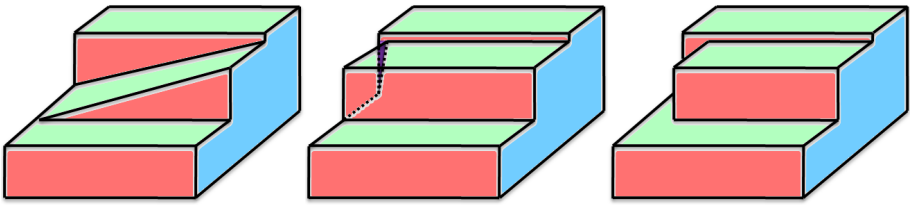


Figure 3.18: Degenerated quads require to place a step adjacent to the degenerated edge (left). It may be impossible to do so without squeezing a part of an adjacent meta-face (middle). Our solution first removes the extremity of the degenerated edge, leading to a valid polycube (right).

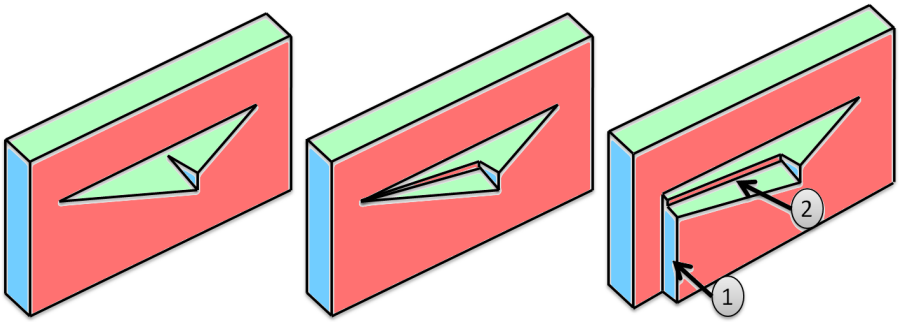


Figure 3.19: Dead lock for solving the first dimension (Left). Creating a step would create two new degenerated quads, including one in the current dimension to be solved (Middle). Solving the other dimension first (Right—1) makes it possible to solve the current dimension ((Right—2)).

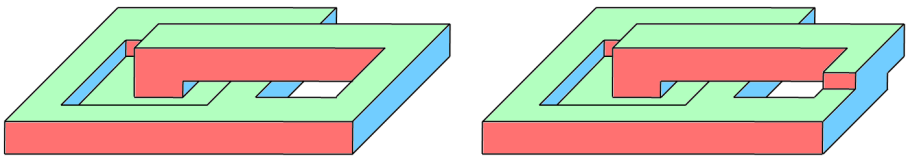


Figure 3.20: A deformation that will align the faces of the left object would push the hole under the opposite face, creating a volume fold-over. Preventing such situation would require to create a step (Right).

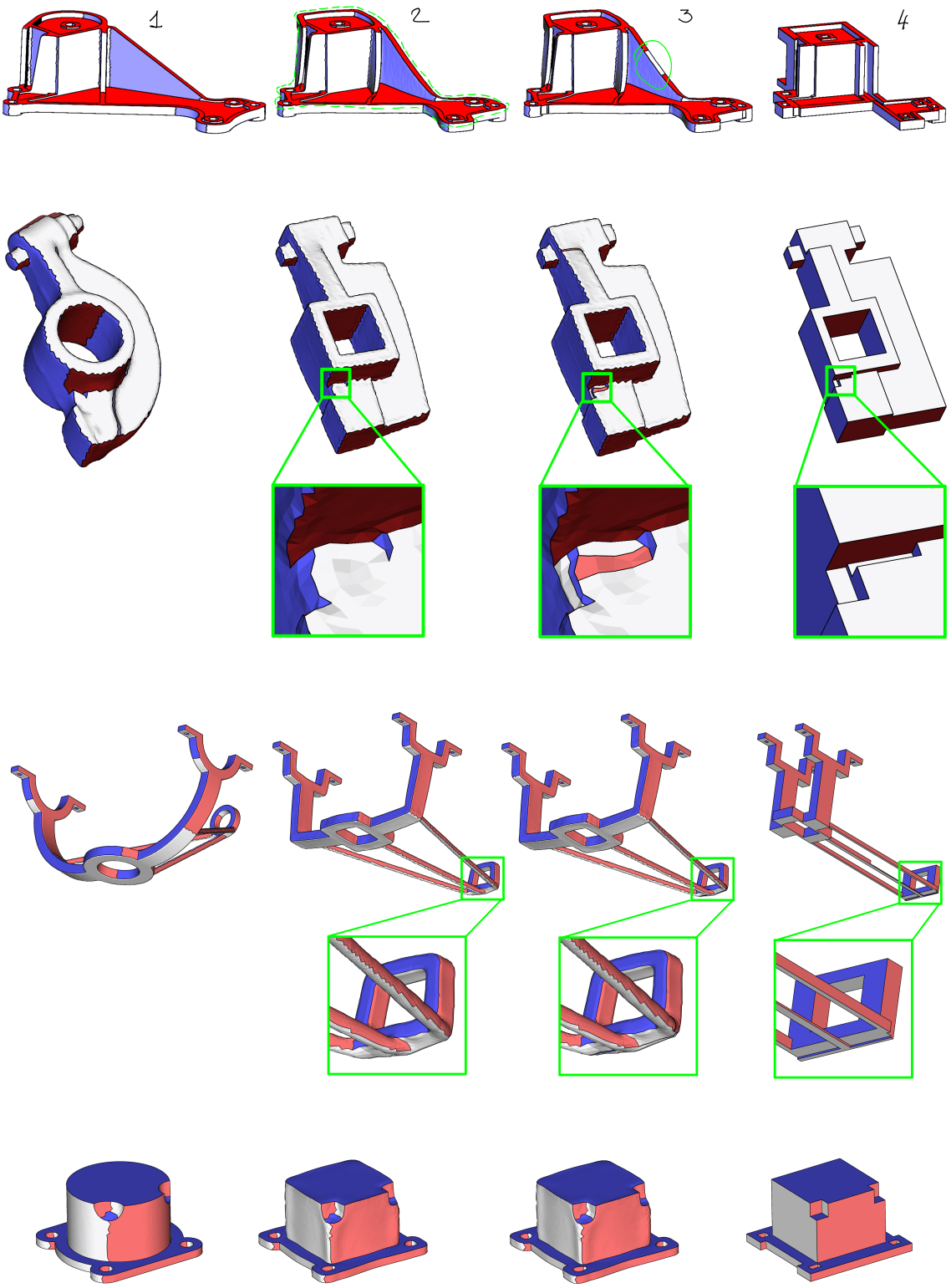


Figure 3.21: Results on CAD/CAM objects. **(1)** Input mesh with boundary faces flagged to the nearest axis. **(2)** A soft deformation is applied to improve the flagging. Note the dashed cycle in the upper row, it must be planar in the final polycube, it is a hard constraint squeezing the model. **(3)** We relax this constraint by re-flagging a part of the red chart. **(4)** Final polycube.

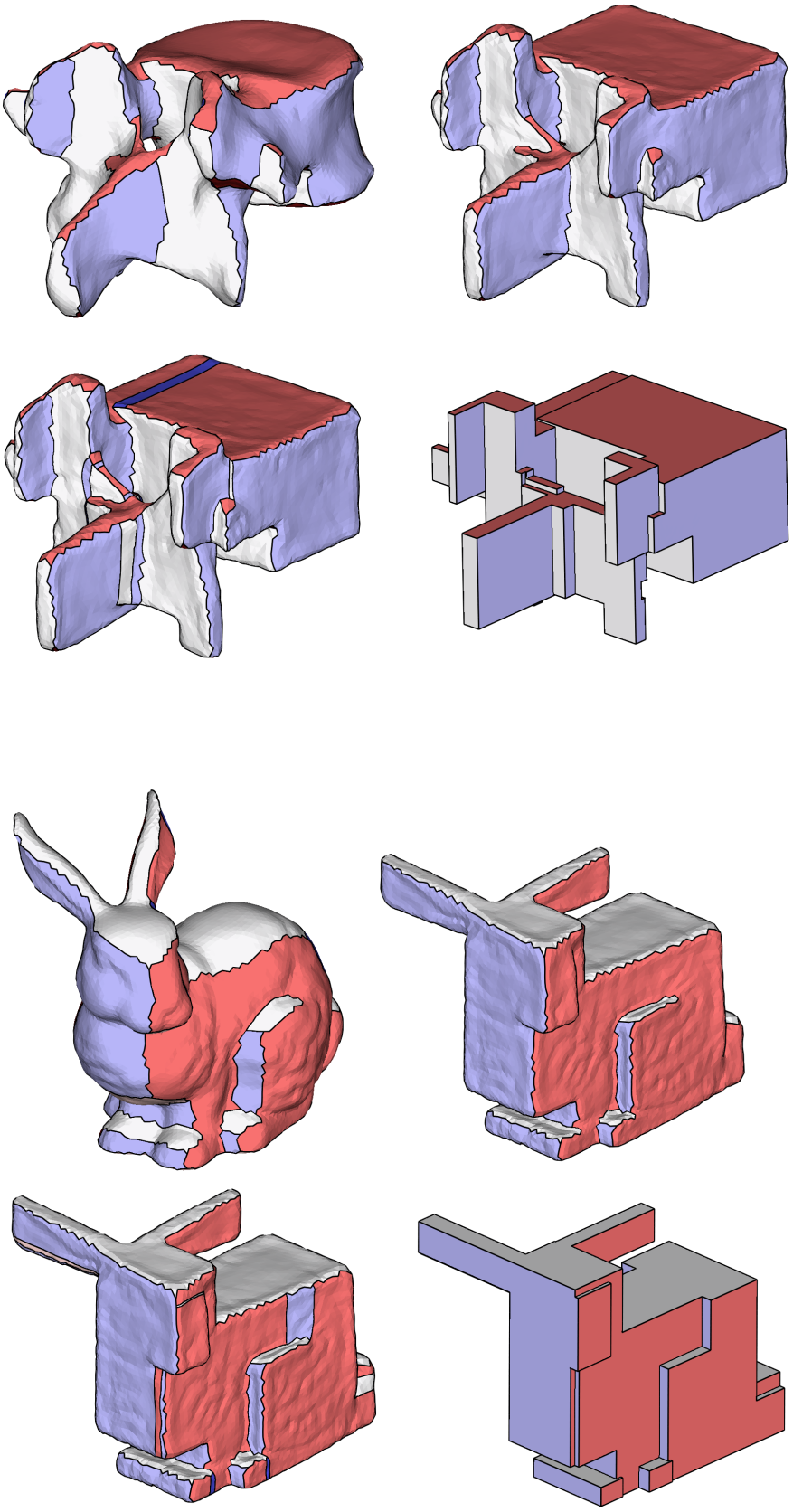


Figure 3.22: Results on smooth objects.

3.1.4 Results and discussion

Since we are working with meta-meshes, our algorithm running time is very low (under 10 seconds for all shown models). We have tested our algorithm on different type of objects, figure 3.21 provides results on CAD objects and figure 3.22 gives the behavior for smooth objects.

In rare cases editing of normal constraints is not necessary, a polycube is realizable directly from input flagging (bottom row of figure 3.21). Second row of figure 3.21 can be treated with the original repair procedures proposed by Gregson *et al.* [GSZ11]. However, the top row (and the 3rd one) of figure 3.21 shows a typical failure case of the method, it requires global constraints to be solved in order to create necessary steps.

For the sake of clarity, we did not give all details and justifications during the presentation of our algorithm. The following points are worth noting:

Characterization of target objects This approach creates hexahedral meshes with no singularities inside the volume, therefore it may be better suited to re-meshing of smooth objects like bones, petroleum reservoirs, etc. These objects have no or few hard edges, making it possible to change the mapping of the polycube edges on the surface. CAD/CAM objects can be re-meshed, but having a regular (distorted) grid inside the volume makes it hard to obtain nice results even on simple objects such as prism, pyramids, etc. Indeed, generally these objects are not nicely approximated by polycubes.

Geometric quality This work aims the robustness of the approach, produced steps are valid topologically, however the geometric quality is (almost) not taken into account.

Degenerate quads hard to fix When it is required to create a step on a quad strip that includes a degenerated quad, the situation becomes hard to solve (see Figure 3.19). We detect such cases and try to solve other dimensions first. We conjecture that there is no configuration where all dimensions are locked at the same time.

Prevent surface self intersections The current results ensure that the boundary surface can be deformed to satisfy the normal constraints. This does not prevent volume foldovers. It should be possible to detect the boundary intersections, and place slack variables to avoid such situations. In practice, such situations appear in some CAD/CAM models when the object thickness is very low. However, detection of conflicting meta-faces is beyond the scope of this project. It is interesting to notice that introducing such slack variables in our system should even be able to create steps if needed, as illustrated in Figure 3.20.

Conclusion

Automatic generation of polycubes is a challenging problem and affecting the desired polycube normal to the original surface prior to constructing the polycube is a promising idea. In this work, we have presented some limits of this approach as well as a solution to resolve the surfacic foldovers issues. It makes it possible to convert many challenging surfaces into polycubes such as Escher-style polycubes, screws, or situation involving partial shrink of the polycube face (degenerated quads). The polycube construction could be made more robust by dealing with the volume foldovers, and provide nicer results by both pre-processing (better smooth object deformation) and post-processing (optimizing the mapping between polycube and surface, use a better constrained deformation, apply suitable local hex mesh operations).

3.2 Robust tracing of streamlines on triangulated surfaces

This section explains how to trace streamlines on vector fields designed on triangulated surfaces. While it is an essential step to make the generation of polycubes robust (recall the close-up on the figure 3.15), we feel that it is useful for many other applications, so it is extracted to a separate section. This work was previously published at [RS14a].

Segmentations of triangulated surfaces that align chart boundaries with a direction field often exhibit useful properties for computer graphics applications. For instance, alignment with the main curvature directions is used for quad dominant remeshing [ACSD+03], following the gradient of a scalar field allows to compute pure quad decomposition using Morse-Smale complexes [DBG+06, SZ12], and streamlines of a cross field can decompose a mesh into quad shaped domains [KLF13]. However, it is required that cutting polylines do not merge to obtain a pure quad decomposition. This property is difficult to enforce in the presence of highly perturbed geometry, field singularities (Fig. 3.23), or limit cycles (Fig. 3.34).

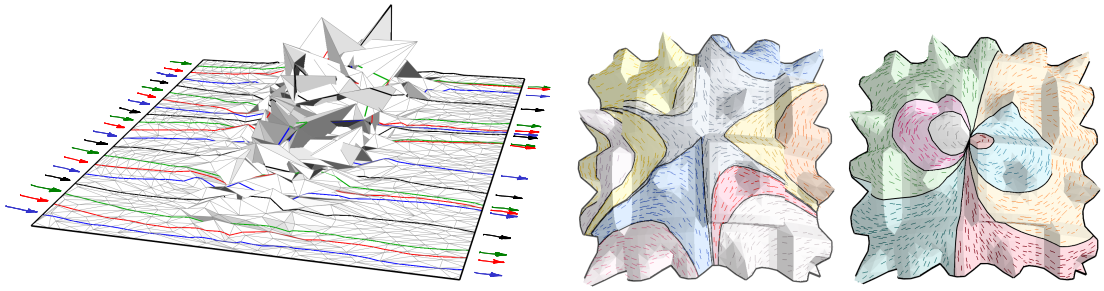
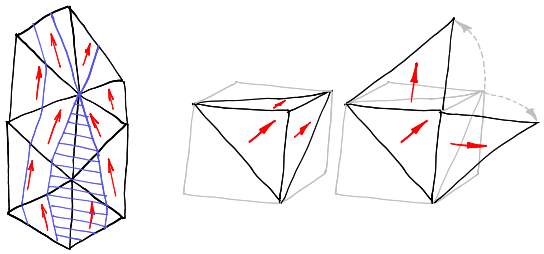


Figure 3.23: Our algorithm traces polylines on triangulated surfaces. Unlike previous algorithms our technique ensures that two polylines cannot cross each other. It works even with highly perturbed surfaces (top) and supports any type of vector field singularities (bottom).

All previous algorithms tend to generate polyline crossing or merging. Indeed, when tracing a polyline that converges to a limit cycle, the distance between the polyline and the limit cycle decreases at each loop, until the (floating point) number representation accuracy is reached and the polylines either merge or cross.

A more common source of merges comes from vector fields representations that are polynomial inside each triangle, leading to incompatible directions between pairs of adjacent triangles. As illustrated in Fig. 3.24, and observed in practice in Fig.3.35, these incompatibilities leads streamlines to converge to an edge. Zhang *et al* [ZMT06, Section 6.2] analyse this issue and provide an alternative vector field representation based on local flattening. Our representation differs from Zhang's one, but it also prevents merges or splits along streamlines.



(a) **(b)**

Figure 3.24: **(a)** all streamlines from the dashed area merge on an edge, and split on a vertex. **(b)** constant per triangle tangent vector fields have direction discontinuities along edges due to vertex angle defect.

Algorithm overview

Given a triangulated surface and a direction field in our representation (detailed in Section 3.2.1), our method traces cross-free and merge-free polylines that are oriented by the direction field.

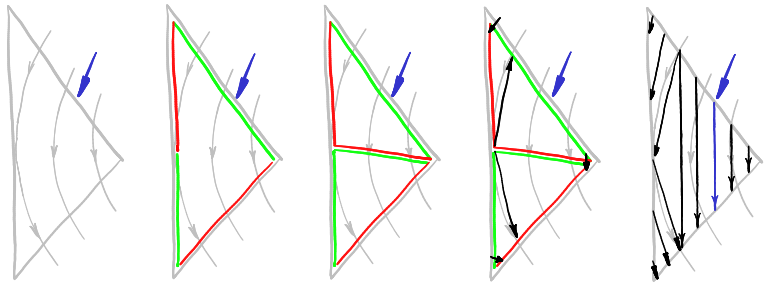
We start from a polyline extremity located at barycentric coordinates c (and $1-c$) on halfedge e. The algorithm crosses the triangle associated to e by finding the output point (e', c'), and continues on the next triangle. It stops when the streamline reaches the surface boundary, a sink of the field, or when the polyline's number of segments reaches a user given limit.

The main difficulty is to determine how each triangle is traversed by the polyline. Our approach (Fig. 3.25) is inspired by EdgeMaps [BJB+11]: we decompose the triangle into pairs of inflow/outflow interval, and define the triangle crossing function by a linear mapping between each inflow interval and its corresponding outflow interval.

The original EdgeMaps algorithm is not guaranteed to produce cross-free and merge-free streamlines because: its input is a linear vector field per triangle (subject to field discontinuities along edges), pairing of inflow/outflow interval requires to trace streamlines inside triangles (subject to numerical integration errors), and the linear maps between intervals is monotonic only up to numerical precision. We address these issues as follows:

- **input representation:** we introduce an explicit representation of the field direction on edges, making the field compatible on adjacent triangles.
- **pairing input/output interval:** the behavior of the field inside each triangle is described by a new structure called stream-mesh. The boundary of each (simple) stream-face can be decomposed into two regions: one where the field points inside the face and one where field points outside the face (Fig. 3.25–c, and Section 3.2.2). The problem of crossing a triangle can then be restated as crossing its stream-mesh (Section 3.2.3), by iteratively crossing its stream-faces. It guarantees the mapping to be monotonic, if it is performed with arbitrary precision numbers.
- **numerical precision:** we use arbitrary precision floating points to represent the barycentric coordinate c, and manipulate them by almost linear mapping functions (Section 3.2.4) that are guaranteed to be monotonic.

Practically, the first point prevents merges of the "real streamlines" of the input. The second point allows all non trivial configurations (field singularities, edge tangent to the field, high vertices angle defect) to be uniformly and correctly handled without any parameters inherent to numerical integration. The last point allows two polylines to become arbitrary close to each other, as it happens with limit cycles.



(a) **(b)** **(c)** **(d)** **(e)**

Figure 3.25: **(a)** a polyline (blue arrow) enters a triangle. **(b)** we construct a stream-mesh by a segmentation of the triangle boundary into inflow (green) and outflow (red) segments (Section 3.2.2). **(c)** the stream-mesh is split into simple stream-faces (Section 3.2.2). **(d)** we cross the triangle from stream-halfedges extremities (Section 3.2.3). **(e)** we map inflow intervals onto outflow intervals using an almost linear mapping with exact precision number representation (Section 3.2.4).

Previous Work

To the best of our knowledge, no prior work directly addresses our problem. However, it is interesting to review solutions developed for $2D$ streamline tracing, to notice similar issues occurring

for tracing other types of curves on surfaces, and to give an overview of the tangent vector field and, more generally, N-symmetry direction field design algorithms.

Streamline tracing

Tracing streamlines of $2D$ or $3D$ vector fields is a common task [SLCZ09, RT12] in visualization. In most cases, an order four Runge Kutta (RK4) integration scheme performs well. For piecewise linear vector field on a triangulation, Bhatia *et al.* propose EdgeMaps [BJB+11], a more robust solution that directly matches in/out flow intervals of the triangle border. Our method shares the idea of directly mapping in/out flow intervals, but is not limited by vertices angle defect or numerical precision issues.

Tracing curves on triangulated surfaces

Tracing curves on triangulated surfaces is a challenging task because the curve may cross triangles, follow edges, and pass through vertices [LLP05b]. All such configurations are naturally managed by our representation: a polyline passing through a vertex is considered as crossing a subset of its adjacent triangles, all polyline vertices being located on the vertex of the surface.

For computing optimal systems of loops [CdVL05], one needs to distinguish the order between curves following the same edge, leading to a complex data structure where all the curves following the same edge need to be ordered. Special efforts [MVC05, SSK+05, PS06] have also been devoted to tracing geodesics where the angle defect plays an important role, as in our case.

Recent works [SZ12] compute Morse decomposition of piecewise constant vector fields by converting them into a combinatorial structure. It results in a robust algorithm, but the streamlines traced from saddles to create edges of the Morse complex still merge of split due to the input field.

Direction field design

Many algorithms [ZMT06, WWT+06, FSDH07] allow to design tangent vector fields. The resulting field can be continuous enough to have (continuous) streamlines that do not cross one another [ZMT06], eventually at the expense of simultaneously refining the surface [WWT+06]. Albeit, there exist no solution to trace merge-free streamlines.

For mesh segmentation, it is more common to use N-symmetry direction fields than tangent vector fields, but a N-symmetry direction field is equivalent to a vector field on an N-covering of the surface [KNP07]. Such fields were used for quad remeshing based on global parameterization [RLL+06]. The lack of control over the topology of these direction fields was addressed later [PZ07, RVLL06]. A common representation [KNP07, RVLL06, RVAL09, BZK09] samples the direction on triangles, and makes explicit the field rotation between adjacent triangles.

3.2.1 Field representation

The continuity of tangent vector fields is naturally defined on smooth surfaces. It is possible to extend this notion of continuity on triangulated surfaces ([ZMT06], Section 6) by considering that, across an edge, the vector field should preserve its magnitude and the angle with respect to the edge. This is unfortunately impossible to achieve with most of existing vector field representations, and results in possible streamline merges. We see this issue in detail and introduce an alternative representation.

Most representations of tangent vector fields are polynomial on each triangle. These vector fields are differentiable everywhere on each triangle, so their direction expressed as an angle in a local basis of the triangle is also differentiable. This continuity of the field on triangles also involves discontinuities of the field direction on edges in the vicinity of vertices with non zero angle defect. Indeed, along an infinitesimal circle around the vertex, a unit regular vector field will undergo a rotation that is equal to the vertex angle defect. As the field is differentiable on triangles, the direction rotation accumulated along the cycle necessarily comes from direction discontinuities when crossing edges (Fig. 3.24b). Such discontinuities can lead to the merging streamlines on an edge where the flow leaves both adjacent triangles (Fig. 3.24a).

These issues were already addressed in section 6.2 of [ZMT06], where the field is defined on each vertex by a $2D$ vector in a local map of its one-ring neighborhood, and interpolated on each

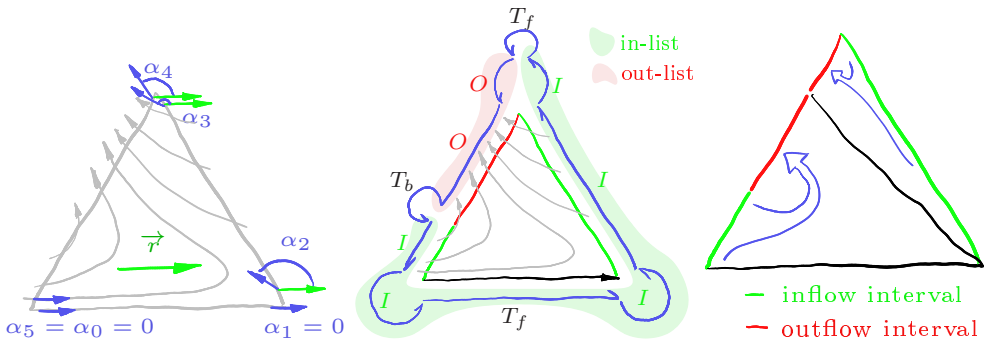


Figure 3.26: **Left:** the field is defined on edge extremities by angles α_i between the field direction and a reference vector $\vec{r}$. **Middle:** the field behavior with respect to the triangle boundary is explicitly represented by stream-halfedges (blue) where the field is either incoming, outgoing, or tangent (I, O, T_f or T_b). In this example, the stream-mesh is a simple stream-face: the field enters the triangle from a single triangle section (in-list) and leaves it from a single triangle section (out-list). **Right:** the black "streamline" is traced from the tangent of the in-list (lower edge). It defines two inflow/outflow pairs of intervals that define the final mappings to be performed with arbitrary precision floating point.

triangle. However, numerical approximations of this field's streamlines are not guaranteed not to cross each other.

To avoid the numerical integration of streamlines inside triangles, we rely only on the field direction along edges. Moreover, we prefer to interpolate the field in polar coordinates instead of Cartesian coordinates to allow for more general types of direction field and singularities. It also simplify the field representation by restricting singularities to be located on vertices.

We represent the input field on each triangle by sampling the field direction at each edge extremity: $\alpha_k, k \in [0 \dots 5]$ are the angles of the field, with respect to a reference vector $\vec{r}$ taken in the triangle plane (Fig. 3.26—Left). Note that due to angle defect and singularities on vertex, each triangle corner is associated to two angles: one for each incident edge.

To prevent crossings, we force the input field to be continuous across edges, i.e to have the same angle with respect to an edge on both adjacent triangles of this edge. We also constrain the angle discontinuity on triangle corners to be evenly distributed around each corner. This latter constraint is equivalent to the local flattening of one-ring neighborhood in [ZMT06] and allows to better manage singularities.

Connection with direction field

We have defined how to represent a vector field on each triangle. To handle N-symmetries, other directions are generated by applying a rotation of $2k\pi/N$ with $k \in 1..N-1$ to the vector field. The vector field across an edge requires the k's of each triangle to agree: their difference (referred to as layer shift in N-coverings) is uniquely defined due to the continuity (enforced in the previous paragraph) of the field across edges.

3.2.2 Stream-mesh

A stream-mesh is the combinatorial representation of the field behavior inside a triangle of the mesh. It is a halfedge data structure endowed with additional information that represents the field behavior with respect to the triangle boundary. The field direction is given at each stream-vertex by its angle α relative to the triangle reference vector $\vec{r}$. Along each stream-halfedge e, the field has a unique behavior that may be :

- incoming (I) if the field points inwards the stream-face,
- outgoing (O) if the field points outwards the stream-face,
- tangent in the forward direction (T_f) if the field has the stream-halfedge direction,
- or tangent in the backward direction (T_b) if the field direction is opposite to the stream-halfedge direction.

As illustrated in Fig. 3.26–Middle, stream-halfedges represent the behavior of the field with respect to the triangle boundary. They can correspond to a portion of an edge, or be limited to a single point where the field becomes tangent to an edge, or define the field behavior on a triangle corner. In the latter case (Fig. 3.27), multiple stream-halfedges are used to describe the field behavior on a single point (triangle corner).

In this representation, we can define:

- An **in-list** as a list of stream-halfedges that contains at least one incoming stream-halfedge, and no outgoing stream-halfedge.
- An **out-list** as a list of stream-halfedges that contains at least one outgoing stream-halfedge, and no incoming stream-halfedge.
- A **simple stream-face** as a stream-face having a border that can be decomposed into an in-list, followed by a forward tangent stream-halfedge, followed by an out-list, and followed by a backward tangent stream-halfedge (Fig. 3.28–right).

The stream-mesh is initialized as a single stream-face by decomposing the triangle border according to the field behavior (Section 3.2.2). The main stream-face is then decomposed into simple stream-faces by a strategy inspired from the ear clipping algorithm [Ebe98]: simple stream-faces are iteratively removed from the main stream-face until the main stream-face becomes simple (Section 3.2.2).

Main stream-face initialization

The initialization of the main stream-face of a triangle is performed independently between each pair of field samples. Each such pair corresponds either to a triangle edge, or to a corner of the triangle between an edge and the next edge around the triangle.

For the k^{th} edge E_k of the triangle, the angle of the field with respect to the edge is given by a linear interpolation between $\alpha_{2k} - \angle(\overrightarrow{r}, \overrightarrow{E_k})$ and $\alpha_{2k+1} - \angle(\overrightarrow{r}, \overrightarrow{E_k})$.

- When this angle equals $0 \mod 2\pi$ it is a forward tangent,
- when it equals $\pi \mod 2\pi$ it is a backward tangent,
- when it is strictly between 0 and $\pi \mod 2\pi$, it is incoming,
- and outgoing otherwise.

A stream-halfedge is generated for every interval with constant type of behavior, including zero length intervals when the field is tangent at a single point. These tangent directions must be explicitly represented as illustrated in the first row of Fig. 3.27, where columns 2 and 3 differ only by their opposite tangent directions.

On the triangle corner between k^{th} edge E_k and j^{th} edge E_j (with $j-k = 1 \mod 3$), α_{2k+1} and α_{2j} may be different due to vertex angle defect or field singularities. Consequently, it is possible for a vertex to contain important topologic information about the field. As illustrated in Fig. 3.27, the field behavior on a vertex (second row) is similar to its behavior along an edge (first row), and can be characterized in the same way. The segmentation is performed with the algorithm described for edges, except that angles are linearly interpolated between $\alpha_{2k+1} - \angle(\overrightarrow{r}, \overrightarrow{E_k})$ and $\alpha_{2j} - (\angle(\overrightarrow{r}, \overrightarrow{E_k}) + \angle(\overrightarrow{E_k}, \overrightarrow{E_j}))$. One can notice that using $\angle(\overrightarrow{r}, \overrightarrow{E_k}) + \angle(\overrightarrow{E_k}, \overrightarrow{E_j})$ instead of $\angle(\overrightarrow{r}, \overrightarrow{E_j})$ allows to consider that the triangle border rotation on the corner is in $]0, \pi[$ (not modulo 2π).

A possible geometric interpretation of stream-halfedges generated on triangle corners could be to consider the triangle as a rounded triangle having its corner radius tending to 0. It makes the field and the triangle border rotating along the arc of circle instead of a single point.

Split the stream-mesh into simple stream-faces

The stream-mesh is now initialized by a main stream-face. The decomposition iteratively removes simple stream-faces from the main stream-face until the main stream-face becomes simple (Fig. 3.28).

To remove a simple stream-face (Fig. 3.29), we search in the stream-halfedges list of the main stream-face a sequence of halfedges that can be decomposed into : a T_f stream-halfedge, followed

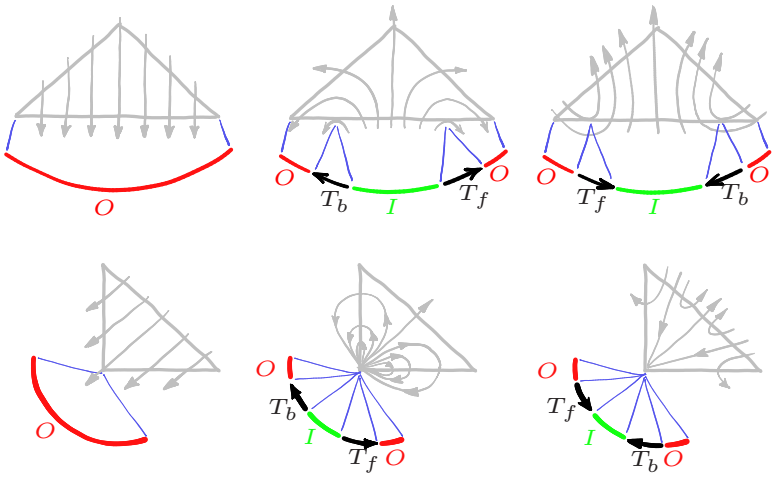


Figure 3.27: Combinatorial representation of the flow behavior. The first row shows the decomposition of an edge into incoming (green), outgoing (red), tangent forward and backward (black arrows) stream-halfedges, for three different fields. The second row shows that similar situations can occur on a triangle corner and can be characterized the same way. The field behavior is the same on both rows, but in the second row, the field rotation is performed on a single point instead of a triangle edge. The only difference between columns 2 and 3 is the tangent direction.

by an out-list, followed by a T_b stream-halfedge, followed by an in-list, and followed by a T_b stream-halfedge. We split the first T_f and last T_b stream-halfedges of the sequence and introduce a new stream-edge linking the stream-vertices produced by the stream-edge split. The type of the generated stream-halfedges is set to *incoming* in the simple stream-face side, and *outgoing* in the main stream-face side.

As illustrated in Fig. 3.29, the type of the produced stream-halfedges is coherent with the flux that can be computed across the stream-halfedge. Indeed, the triangle border being convex, the field direction at the new stream-halfedge's extremities will always point to the same half-plane of the new stream-halfedge.

By symmetry, it is also possible to apply the same operation on the opposite field, i.e. replace both $T_f \Leftrightarrow T_b$ and in-list$\Leftrightarrow$ out-list in the pattern and in the result.

Next section shows that recursively applying the split operation converges to a decomposition into simple stream-faces.

Correctness of the decomposition

Convergence Given a stream-face with n in-lists and n out-lists, let us choose one out-list as a reference. Any two adjacent lists i and $i+1$ have a tangent between them, let us define a sequence of labels $\{t_i\}_{i=0}^{\infty}$ as the label of tangent stream-halfedge incident to both lists i and $i+1$. Then we define a sequence of integers $\{a_i\}_{i=0}^{+\infty}$ as follows:

$$a_0 = 0$$

$$a_{2i+1} = \begin{cases} a_{2i} + 1 & \text{if } t_{2i+1} = T_f, \\ a_{2i} - 1 & \text{otherwise.} \end{cases}$$

$$a_{2i+2} = \begin{cases} a_{2i+1} + 1 & \text{if } t_{2i+2} = T_b, \\ a_{2i+1} - 1 & \text{otherwise.} \end{cases}$$

The defined sequence $\{a_i\}$ is arithmetic quasiperiodic: $a_{i+2n} = a_i - 2$ and is continuous in the sense that $|a_{i+1} - a_i| = 1$. A stream-face is simple if and only if the corresponding sequence $\{a_i\}$ is decreasing. The splitting rule described in section 3.2.2 searches for a pattern (per period $2n$) $(2i+1, 2i, 2i-1, 2i)$ in the sequence $\{a_i\}$ and replaces it with a new one $(2i+1, 2i)$. In other words, the splitting rule removes one (per period) local minimum of the sequence $\{a_i\}$. The symmetric rule replaces $(2i+2, 2i+1, 2i, 2i+1)$ with $(2i+2, 2i+1)$, again removing a local minimum. If a

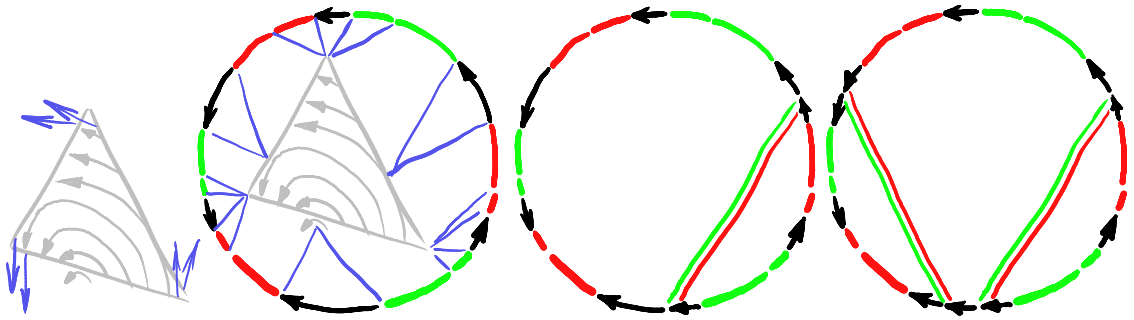


Figure 3.28: The field is converted into a stream-mesh, then a simple stream-face is removed at each step until the main stream-face becomes simple.

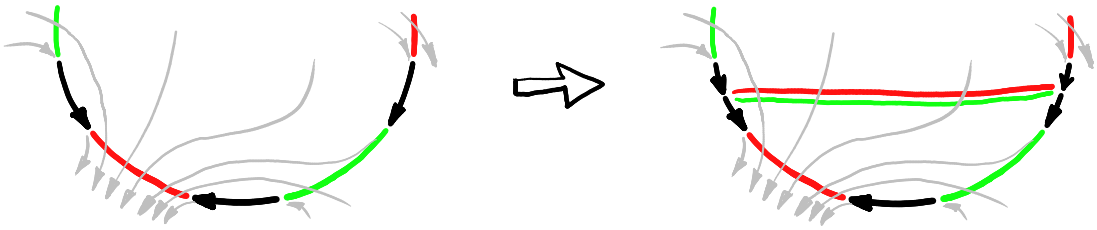


Figure 3.29: Splitting the main stream-face (left) by our rule produces a simple stream-face and removes a pair of in-list/out-list of the main stream-face border.

stream face is not simple, the corresponding sequence has at least one local minima, moreover, the sequence decreases by 2 with each period and therefore it is possible to apply one of the splitting rules. Both rules keep the continuity of the sequence, and the period is reduced by 2 with each iteration, leading to a final decomposition of the initial stream-face into a set of simple stream-faces.

Non-nullity of flux through simple faces We show that each simple stream-face is traversed by some flux. To do so we demonstrate that the out-list (as well as in-list) of a simple stream-face have non-zero associated flux.

First of all, let us note that all stream-halfedges created by splitting rules have non-zero flux. Indeed, their length is not zero: it is easy to see that due to the linear interpolation between angle samples, the sequence $\{a_i\}$ is monotonic inside triangle corners; however the splitting rule searches for a local minimum of the sequence. Therefore, it is not possible to create a simple face entirely contained in a triangle corner.

Now let us show that all simple faces have non-zero flux through them. Let us suppose that the out-list of a simple stream-face has a zero flux. All outflow stream-halfedges on triangle edges as well as outflow stream-halfedges corresponding to splits have non-zero flux, since their length is greater then zero. The only option for an out-list to have a zero flux is to be contained in a triangle corner and to have T_b, O, T_f structure, as defined in section 3.2.3. However it means that the corresponding sequence $\{a_i\}$ is increasing on this out-list, and that contradicts the monotonicity of the sequence $\{a_i\}$ for simple faces. Therefore, there is no out-list in a simple face that does not have a flux through it. The same argument shows by symmetry that there is no in-list without flux through it.

3.2.3 Pairing intervals

Pairing inflow/outflow intervals using the original EdgeMaps algorithm [BJB+11] requires to trace a set of "streamlines" inside each triangle. However, the numerical imprecision involved by the streamline integration inside triangles may produce an invalid decomposition. Typical failure cases include high field rotation close to field singularities, or fields that are almost tangent to an edge. To prevent such failure cases, we replace the numerical streamline integration by a traversal of the stream-mesh.

A robust method crosses each simple stream-face by almost linear mappings (Section 3.2.4) between their in-list and out-list. This solution guarantees the generation of intersection-free

streamlines, but the polyline orientation may not closely match the direction field geometry.

Alternatively, a more geometric method crosses each simple stream-face by using a heuristic to take the field geometry into account. This method may fail due to numerical approximations, but better fits the field geometry.

Both solutions only differs by the estimation of the direction field flux Φ across the stream-halfedges. We start with the geometric heuristic and switch to the robust version if needed.

Remark 1: At this point, we have two solutions to cross a triangle, but we don't use them directly for tracing the polyline because: the robust version has a poor geometry with respect to the direction field, and the other one may result in crossing streamlines. Instead we use this method only to decompose the triangle boundary into inflow/outflow intervals (Fig. 3.26—Right), then use EdgeMaps with arbitrary precision to perform the final mapping.

Remark 2: The geometric heuristic with fixed precision is usually good enough for the decomposition, but not for directly tracing streamline. On one hand, the decomposition requires to trace only few "streamlines", and it is possible to check the validity independently in each triangle. On the other hand, tracing a polyline is much more difficult because each new segment must be guaranteed not to cross all previous and future segments that may cross this triangle.

Crossing a simple stream-face

Crossing a simple stream-face requires to define how points in the in-list are mapped to points in the out-list. Any such mapping that does not cross streamlines will produce globally cross-free streamlines. However, it is better to choose a mapping that preserves as much as possible the field geometry. Our mapping is defined such that any evenly distributed set of streamlines that enters a triangle will leave it with an even distribution, except if field sinks or streamlines that are tangent to the boundary prevent it. It can be restated as follows: for a unit norm field, if the stream-face is split by a streamline, both parts should have the same ratio between the incoming flux and the outgoing flux. Here, we call by flux the amount of streamlines outgoing from a portion of the out-list (and symmetrically for the in-list). However, it can be considered as an abuse of terminology because we explicitly set a non zero flux for sink/source vertices to allow for an infinite set of streamlines to pass through them (Section 3.2.3), whereas computing the flux of the unit vector field would give zero. This heuristic perfectly respects the field when it is constant inside the triangle, and is evaluated in in more difficult situations (figures 3.23 and 3.36).

As illustrated in Fig. 3.30, we call f (resp. b) the stream-halfedge of type T_f (resp. T_b) that comes before the out-list (resp. in-list).

We denote by $\Phi(e, c)$ the flux crossing the in-list (resp. out) of stream-halfedges up to the point located at the $(c, 1-c)$ barycentric coordinate on the stream-halfedge e. It is recursively defined by $\Phi(e, c) = \Phi(prev(e), 1) + \phi_e(c)$ where $\Phi(f, 1) = 0$,$\Phi(b, 1) = 0$, and $\phi_e(c)$ is the flux crossing the stream-halfedge e up to the point of barycentric coordinates $c, 1-c$.

Using these notations (Fig. 3.30), the condition for a streamline to split the simple stream-face into two stream-faces having the same ratio between inflow and outflow writes:

$$\frac{\Phi(e_{in}, c_{in})}{\Phi(prev(f), 1)} = 1 - \frac{\Phi(e_{out}, c_{out})}{\Phi(prev(b), 1)}$$

where the input point is e_{in}, c_{in} and the output point is e_{out}, c_{out}. As a consequence, the output point is given by :

$$(e_{out}, c_{out}) = \Phi^{-1}\left(\Phi(prev(b), 1)\left(1 - \frac{\Phi(e_{in}, c_{in})}{\Phi(prev(f), 1)}\right)\right)$$

To compute the output position (e_{out}, c_{out}) of a streamline, we need to evaluate the functions Φ, and Φ^{-1}. The function Φ can be evaluated from $\phi_e(c)$ using its recursive definition. The function $\Phi^{-1}(x)$ requires to take the stream-halfedge e such that $\Phi(e, 0) \leq x \leq \Phi(e, 1)$ and $\phi_e(1) \neq 0$, and to define its barycentric coordinate $c = \phi_e^{-1}(x - \Phi(e, 0))$.

As a consequence, we only need to be able to evaluate $\phi_e(c)$ and its inverse $\phi_e^{-1}(x)$ to cross a simple stream-face. For the robust version, it is sufficient to set $\phi_e(c) = \phi_e^{-1}(c) = c$, and for the heuristic version it is described in Section 3.2.3 for $\phi_e(c)$ and Section 3.2.3 for $\phi_e^{-1}(c)$

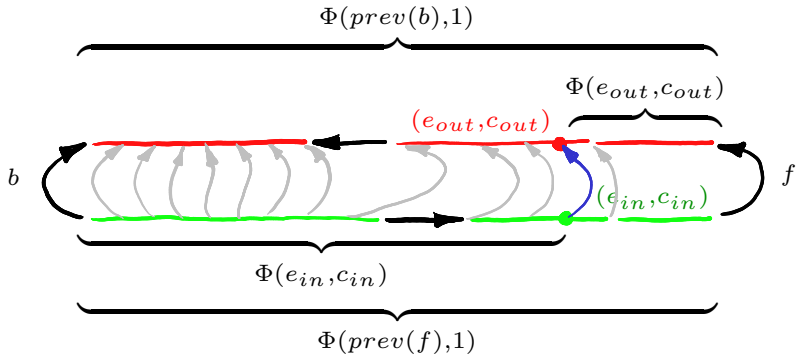


Figure 3.30: Flow notations used to cross a simple stream-face.

Computing $\phi_e(c)$

Flux on edges To estimate a flux across edges, the vector field orientation (direction field) is not sufficient, therefore we also assume that its magnitude is equal to one. On edges, we set $\phi_e(c)$ to be the flux of this vector field across the stream-halfedge e given by:

$$\begin{aligned}\phi_e(c) &= |\overrightarrow{e}| \int_0^c -\sin(\alpha_o + t(\alpha_d - \alpha_o) - \angle(\overrightarrow{e}, \overrightarrow{r}))dt \\ &= |\overrightarrow{e}| \frac{\cos(\alpha_o + t(\alpha_d - \alpha_o) - \angle(\overrightarrow{e}, \overrightarrow{r}))}{\alpha_d - \alpha_o} \Big|_0^c\end{aligned}$$

where α_o and α_d are the field directions located at the vertex pointed by the stream halfedges $prev(e)$ and e, and expressed by their angle relative to $\overrightarrow{r}$.

Flux on vertices N-symmetry direction fields may have singularities that can be characterized by their index. The index is well defined for smooth manifolds [Mro95], and has been extended to triangulated surfaces [RVAL09]. In our case, we assume that singularities can only appear on vertices, leading to the following characterization of indices:

$$Index(A) = \sum \frac{\Delta\alpha_e}{2\pi} + \frac{2\pi - \sum \beta_e}{2\pi} \tag{3.3}$$

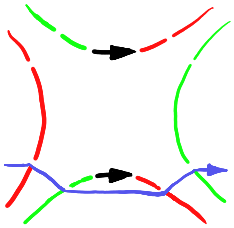
where the sums are performed on all triangle corners referred by their halfedge e incident to A, $Index(A)$ is the index of vertex A, $\Delta\alpha_e$ is the angle discontinuity on the triangle corner, β_e is the triangle corner angle. The first sum is the total amount of field rotation around A, and the rest is the angle defect of A divided by 2π.

Examples of singular vertices are given in Fig. 3.31. One can notice that an infinite number of streamlines can reach the vertex only for strictly positive indices, leading to two different behaviors of our algorithm as detailed bellow.

On corners, we can generally say that there is no flux that leaves the triangle i.e. $\phi_e(c) = 0$. However, for singularities with positive index such as source and sinks, there is an infinity of streamlines that reach or start from the corner (Fig. 3.31). If an outflow stream-halfedge e is defined in a triangle corner, in a sequence T_f, O, T_b, then we set $\phi_e(c) = c$. By symmetry, if an inflow stream-halfedge e is defined in a triangle corner, in a sequence T_b, I, T_f, then we set $\phi_e = -c$. This strategy provides a field behavior coherent with the continuous behavior of streamlines on field singularities as explained in the following section.

Geometric vertex crossing

The default behavior of our algorithm is when there is not an infinity of streamlines having the vertex as one of its extremities. In this case, when a streamline leaves a triangle on a vertex location, the output of the triangle crossing algorithm is an adjacent edge, with a barycentric coordinate being either 0 or 1 to fit the vertex location. The streamline then continues on



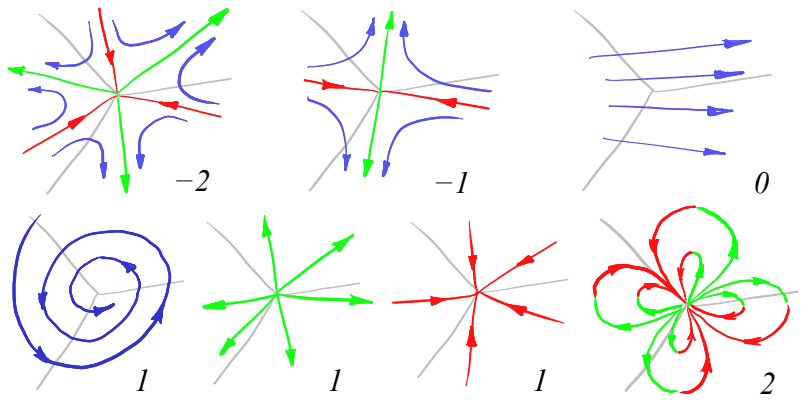


Figure 3.31: Singularities classified by index. On negative indices, there exist a finite number of streamlines (red and green) having the vertex as extremity. On regular vertices (index is zero), at most one streamline can cross the vertex. On positive index singularities, there exist an infinity of streamlines having the vertex as extremity, expect for the vortex case (lower-left).

the next triangle until it ends in the vertex or leaves the vertex location as illustrated in the inset figure.

Our algorithm has this behavior because the flux on a stream-halfedge defined on a triangle corner is generally zero, and the constraint that the simple stream-face crossing algorithm is not allowed to generate outputs on a stream-halfedge without flux.

Streamline extremity on a vertex Streamlines may also have one of its extremities located on a vertex, but this occurs only for vertices with strictly positive index, as illustrated in Fig. 3.31 (we consider that if a unique streamline reaches the vertex it will cross it with the previous behavior). We explain here why our way to determine the flux on stream-halfedges inside triangle corners (Section 3.2.3) gives non zero flux only for vertices with strictly positive index.

As the rotation speed of the field around the vertex A is constant, the difference of angle $\Delta\alpha_e$ is equal to the sum of such rotations around the vertex A times the ratio of β_e over the sum of triangle corner angles around A. Putting it together with equation (3.3), with summation over all halfedges e' pointing to A gives:

$$\Delta\alpha_e = \frac{\beta_e}{\sum \beta_{e'}} \left(2\pi(Index(A) - 1) + \sum \beta_{e'}\right)$$

so the variation of angle with respect to halfedges pointing to A is

$$\Delta\alpha_e - \beta_e = \frac{2\pi\beta_e}{\sum \beta_{e'}}(Index(A) - 1)$$

As a consequence, if $\Delta\alpha_e - \beta_e$ is strictly positive, the vertex index is greater or equal to 1. Otherwise, the index is strictly less than 1. Note that for direction fields with rational indices, we are still able to distinguish between singularities with and without flux.

In our algorithm, the condition to associate some flux to output stream-halfedges (defined on a triangle corner) is that the stream-halfedge must be contained in a sequence $T_f O T_b$. It means that the field angle with respect to the triangle border increases at least by π. Since the corner is convex, we have $\beta_e < \pi$. As a consequence, our algorithm gives some flux only for stream-halfedges in triangle corners corresponding to a vertex with strictly positive index. The same thing occurs for the sequence $T_b I T_f$.

Starting a streamline from a vertex For a vertex that is the origin of a finite number of streamlines (negative or null index), it is possible to generate all streamlines by simply starting a streamline for each inflow stream-halfedge on adjacent triangle corners. This is especially important for tracing streamlines from saddle points, as it is required for computing Morse-Smale complexes.

Computing $\phi_e^{-1}(x)$

Computing $\phi_e^{-1}(x)$ requires to invert Equation (3.3). As cosine is not a one to one function, determining $\phi_e^{-1}(x)$ requires to take into account that it is a barycentric coordinate in the halfedge e, and therefore $0 \leq \phi_e^{-1}(x) \leq 1$. This constraint fixes $s \in \{-1, 1\}$ and $k \in \mathbb{Z}$ in the formula:

$$\begin{aligned} \phi_e^{-1}(x) &= \frac{s \arccos(\cos(\alpha_o - \angle(\overrightarrow{e}, \overrightarrow{r}_T)) - x\frac{(\alpha_d - \alpha_o)}{|\overrightarrow{e}|})}{\alpha_d - \alpha_o} \\ &+ \frac{2k\pi - \alpha_o + \angle(\overrightarrow{e}, \overrightarrow{r}_T)}{\alpha_d - \alpha_o} \end{aligned}$$

3.2.4 Crossing triangle with arbitrary precision

When a polyline reaches a triangle edge e at barycentric position p (given in arbitrary precision floating point) on e, we are able (Section 3.2.3) to determine the corresponding inflow interval (barycentric coordinates $[a/2^i, b/2^i]$ on e) and outflow interval (barycentric coordinates $[c/2^j, d/2^j]$ on edge e'). The usage of dyadic rationals (denominator is a power of two) is motivated by the direct compatibility with floating points, and possible simplifications exploited in our almost linear mapping.

The objective is to determine the barycentric coordinate q on edge e'. A linear interpolation gives $q = c/2^j + (p - a/2^i)(d/2^j - c/2^j)/(b/2^i - a/2^i)$. However, doing so in exact arithmetic dramatically affects the performances: the memory required to represent q is approximately 100 bits larger (50 for the denominator, and 50 for the nominator) than for p. After crossing n triangles, the size of q is approximately $100n$ bits, which greatly reduces the performance and increases the memory required to store the polyline (Fig. 3.33).

Our solution, described in the next section, is an approximation of a linear mapping that reduces by two orders of magnitude the size of p and q (Fig.3.32). At coarse scale, it is linear up to approximations of 64 bits floating points, and at finer scale, it is linearly interpolated between the closest 64 bits floating points numbers.

Almost linear mapping

In this section we describe how to define a strictly monotonic mapping of all dyadic rationals of an origin interval $[a/2^i, b/2^i]$ to a destination interval $[c/2^j, d/2^j]$, where $a, b, c, d \in \mathbb{N}$ and $a < b, c < d$. Algorithm 2 gives an implementation, and Fig. 3.32 illustrates an example of execution.

Input/output interval boundaries define two grids of fractional numbers with given precision (resp. 2^i and 2^j). The idea is to refine the output grid until it becomes larger than the input one, and then to map the input grid onto the output grid by rounding the linear mapping (Fig. 3.32, middle). The final mapping is defined as a collection of linear transformations between pairs of grid segments (Fig. 3.32, right).

The section between lines 5 and 8 of the algorithm 2 is a loop that refines the input grid. While the loop is not mandatory to define a correct mapping, it is essential to save the memory. Indeed, the increase in the precision of the point $q/2^l$ with respect to the point $p/2^k$ is given by the relation $l - k = j - i$, thus we keep i as high as possible to avoid wasting memory. Fig. 3.33 gives a plot of occupied memory for a polyline in the limit cycle field (Fig.3.34). Note that the growth is not monotonic, this is due to two phenomena: either $i > j$ or a fraction that can be canonicalized.

3.2.5 Discussion

This section evaluates the performances of our algorithm on synthetic stress tests, proposes some applications where tracing robust polylines is required, compares with possible alternative algorithms and provides some details about local overlaps.

Synthetic tests

To evaluate the geometric quality of our polylines, we traced them on a circular vector field with different mesh quality (Fig. 3.36). It shows our polylines smoothness and accuracy with different triangle qualities (upper to lower) and different field rotation magnitude (border to center). In practice, computer graphic meshes are closer to the upper and middle images, and field design

Algorithm 2: Algorithm overview

Input: Origin interval boundaries $[a/2^i, b/2^i]$
Input: Point $p/2^k \in [a/2^i, b/2^i]$ with $k \geq i$
Input: Destination interval boundaries $[c/2^j, d/2^j]$

Output: Point $q/2^l \in [c/2^j, d/2^j]$

```
while b − a > d − c do
    (c, d) ← 2 · (c, d);
    j ← j + 1;
end
while 2 · (b − a) < d − c do
    (a, b, p) ← 2 · (a, b, p);
    (i, k) ← (i + 1, k + 1);
end
p′ ← ⌊p/2^(k−i)⌋;
p″ ← p′ + 1;
q′ ← ⌊(p′ − a) · (d − c)/(b − a)⌋ + c;
q″ ← ⌊(p″ − a) · (d − c)/(b − a)⌋ + c;
q ← (p − p′ · 2^(k−i)) · (q″ − q′)/(p″ − p′) + q′;
l ← j + k − i;
return q/2^l;
```

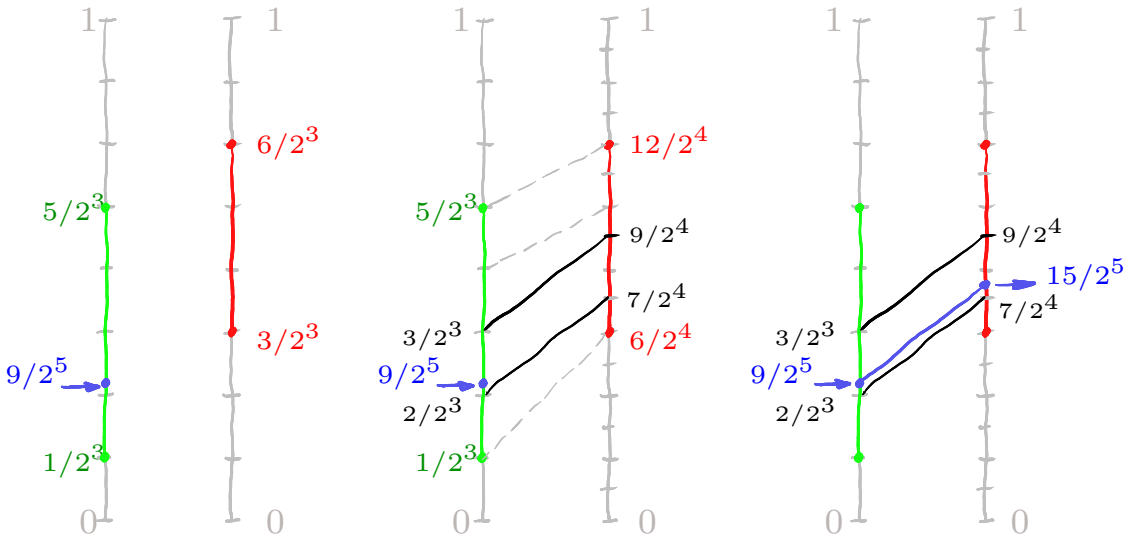


Figure 3.32: An example of execution of the algorithm 2. **(left)** Origin interval $[a/2^i, b/2^i] = [1/2^3, 5/2^3]$, destination interval $[c/2^j, d/2^j] = [3/2^3, 6/2^3]$ and an entry point $p = 9/2^5$. **(middle)** The destination interval is refined (c and d are doubled and j incremented) and we map points $p' = 2$ and $p'' = 3$ to points $q' = 7$ and $q'' = 9$, respectively. **(right)** The resulting point $q/2^l = 15/2^5$ is obtained by the linear mapping of the interval $[p'/2^i, p''/2^i]$ onto the interval $[q'/2^j, q''/2^j]$.

algorithms tends to produce as smooth as possible fields. It is interesting to notice that an important loss of accuracy only appears on very stretched triangles like one having a corner with a field singularity (see close-up).

The cross-free and merge-free properties are ensured by our approach. Fig. 3.23(top) and Fig. 3.34 show examples where these properties are hard to enforce due to noisy geometry and the very short distance between polylines.

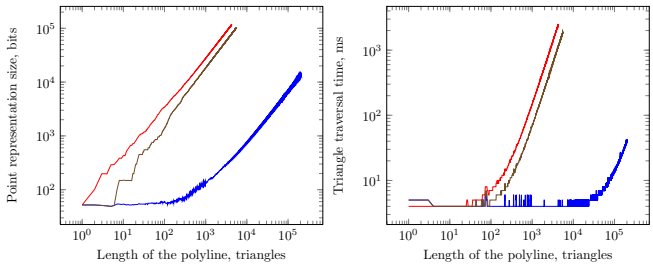


Figure 3.33: Results after one hour of computing: **(a)** length of a polyline (in triangles, x-axis) versus number of bits to represent the current polyline vertex (y-axis): linear mapping (red), without input grid refinement (brown), our algorithm (blue); **(b)** length of a polyline (in triangles, x-axis) versus time to cross one triangle (y-axis).

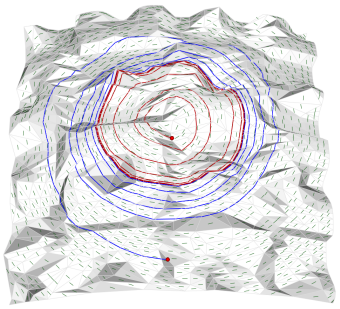


Figure 3.34: Robustness stress test: the polyline initialized at the red dot converges to a limit cycle. Our arbitrary precision representation of the polyline prevents crossings and merging whereas 64 bits precision leads to a merge after less than 10 loops. On the same data, we run with exact precision up to 900 loops for generating Figure 3.33.

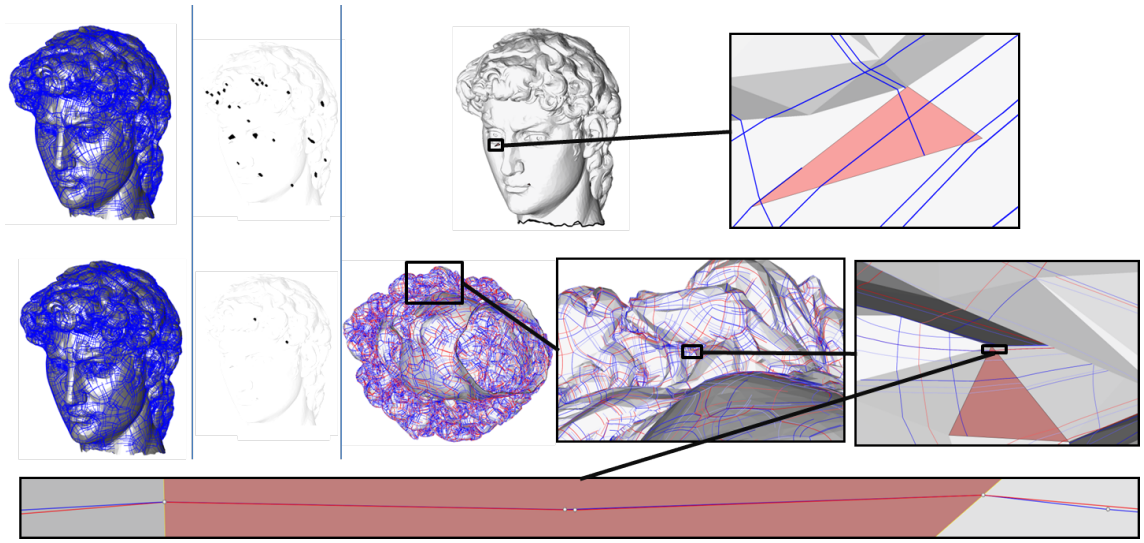


Figure 3.35: **Crosses and merges of streamlines observed with alternative algorithms** EdgeMaps (top row) create many streamlines (second column) that converge to an edge due to the piecewise linear representation of the vector field. Streamlines traced on a continuous field representation [ZMT06] with an order four Runge-Kutta (second row) lead to only two failures (second column) illustrated by the streamline switch observed in the close-up.

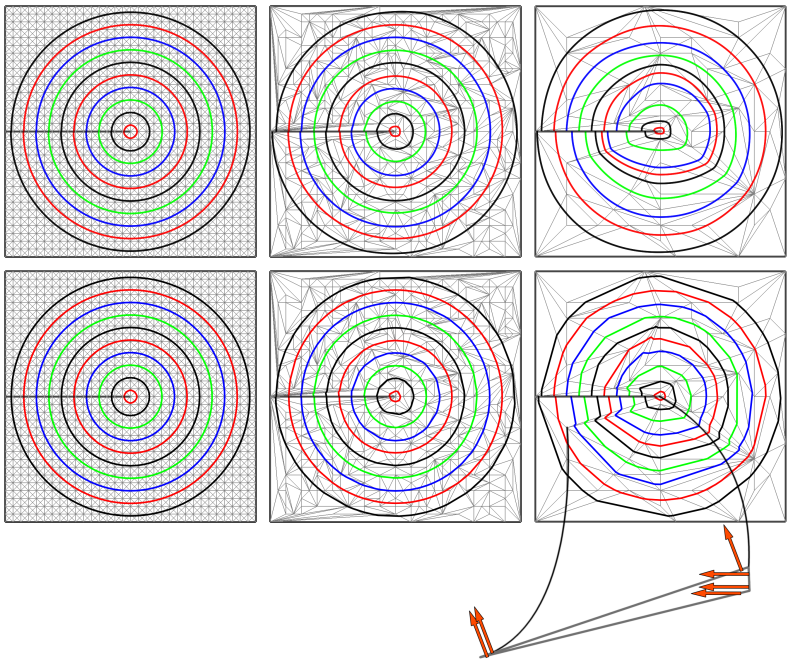


Figure 3.36: Our algorithm on mesh (bottom row) is compared with a numerical integration (RK4) on the same data (top row), with decreasing mesh quality from left to right.

Applications

We illustrate two possible applications of our method: computing quarangulations inspired by Morse-Smale complexes (Fig. 3.37), and splitting a mesh according to a direction field. Tracing streamlines of a N-symmetry direction field [KLF13] allows to partition $2D$ meshes. To illustrate a possible application of our method, we applied the same strategy on $3D$ surfaces, by growing all streamlines simultaneously, and stopping them when they reach a streamline defined on a perpendicular direction. As a result (Fig. 3.38) we obtain quadrangular charts with T-junctions everywhere except when a degeneracy is prescribed by feature curves as in the fandisk model. Such T-meshes could be useful after optimization, as proposed in [MPKZ10].

Alternative algorithms

We have proposed the first algorithm that guarantees non crossing (or merging) streamlines. However, in many applications, alternative algorithms can produce similar results. We review an existing algorithm [BJB+11], a fair solution obtained by combining order four Runge-Kutta with a continuous vector field representation [ZMT06], and our algorithm without adaptive numerical precision. The failure cases are illustrated in Fig. 3.35, and the number of failures on a set of models are given in Fig. 3.39.

- EdgeMaps

  EdgeMaps requires a linear vector field on each triangle. To produce it, we start from our smooth field, and set the vector on the i^{th} triangle corner to be equal to $(\alpha_{2i} + \alpha_{2i+1})/2$. This strategy is fair as it evenly distributes the angle defect of each vertex over all adjacent triangles (much better than a projection).

  On surfaces without angle defect, it offers the same guarantee than our algorithm without adaptive resolution. On other surfaces, streamlines can converge to an edge as in Fig. 3.24–left. In practice, one could expect the failure case to appear very rarely because the streamline must cross an edge with an angle lower than a portion of the angle defect of an incident vertex. However, our experiments illustrated in Fig. 3.39 demonstrate the opposite.

- RK4 on [ZMT06]

  The field introduced in [ZMT06] is sufficiently continuous to have non crossing streamlines. The problem is therefore to determine how often approximations of these streamlines (com-

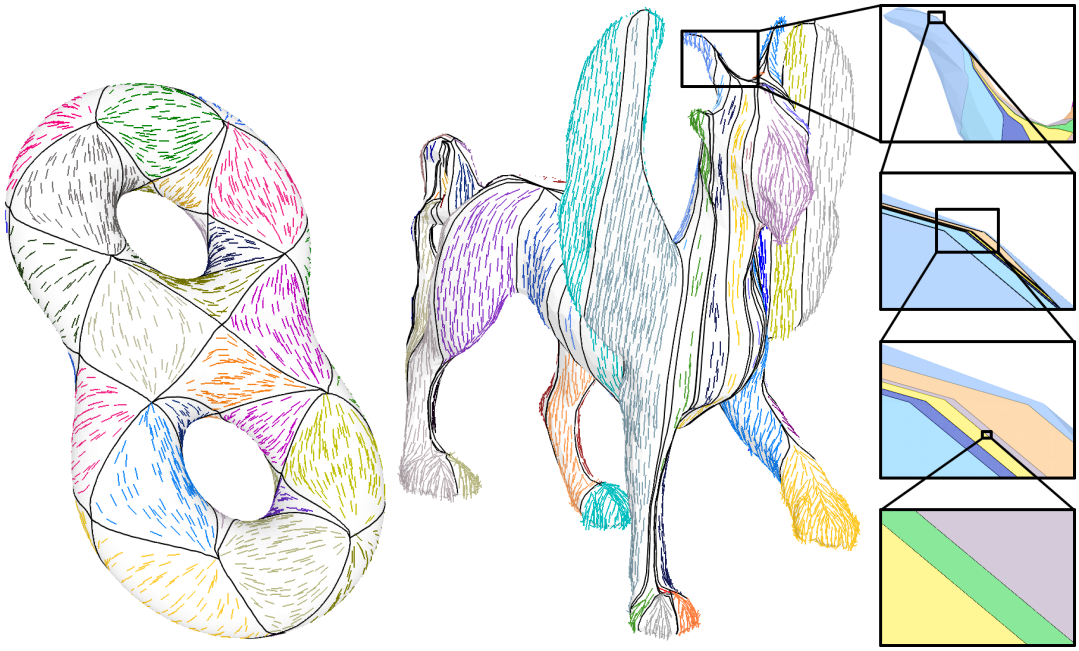


Figure 3.37: Morse-Smale complexes provide a quad shaped charts decomposition of a smooth manifold. Converting a scalar field gradient into our representation allows to have this property for triangulated surfaces. We used a Laplacian eigenfunction for the double torus and the z coordinate for the Feline. Close-ups allow to see that polylines can be very close to each other.

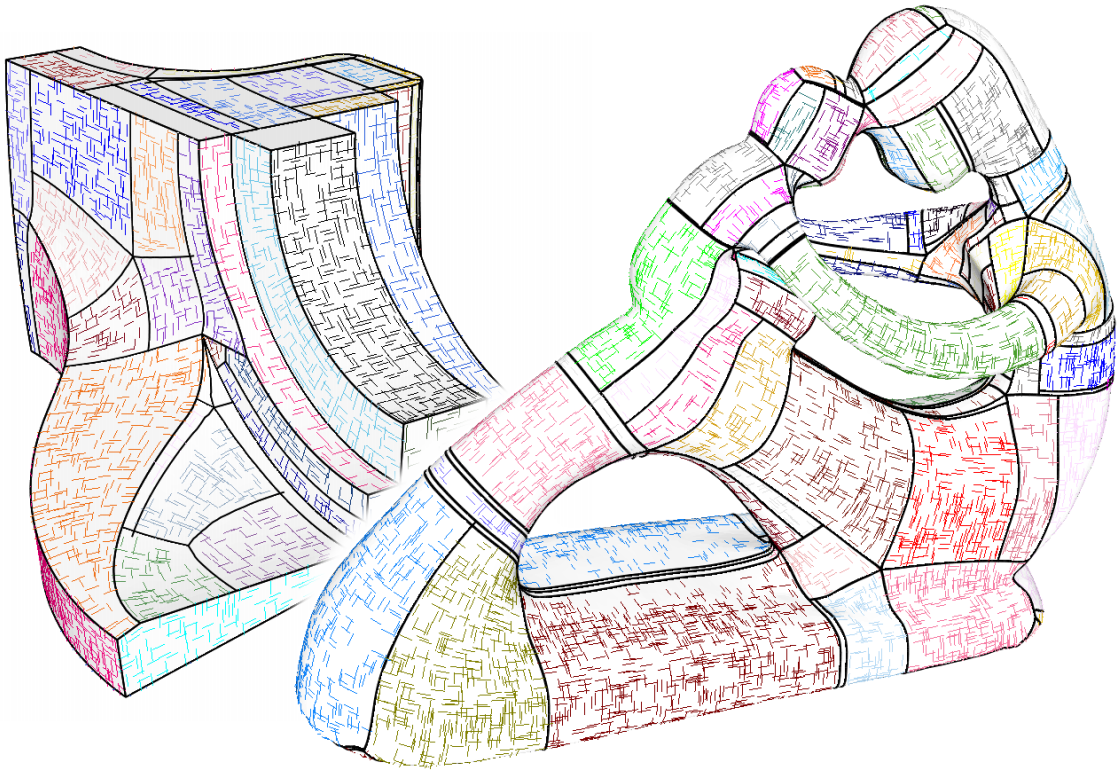


Figure 3.38: Tracing streamlines (black curves) from singularities of a cross field provides a decomposition of the surface

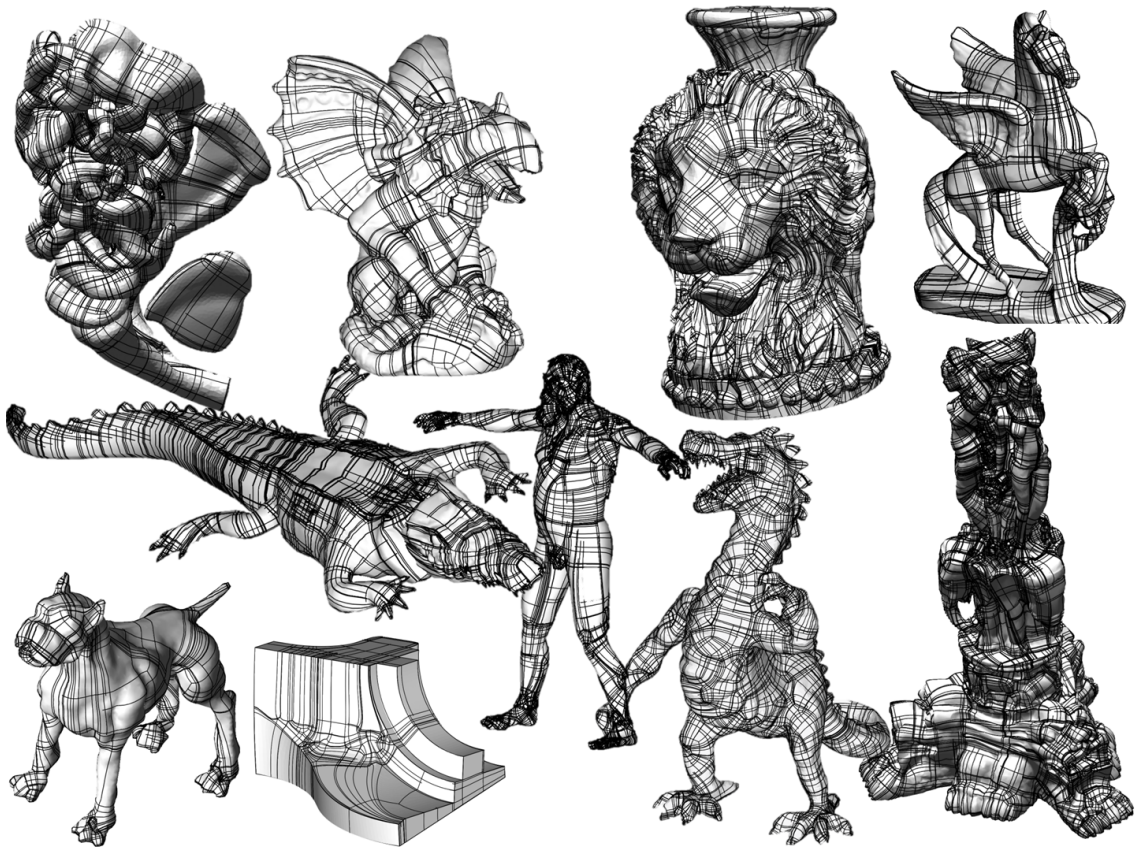


Figure 3.39: Number of failures using [EdgeMaps – RK4] : Jobard's plume 12–0, gargoyle 1–0, lionvase 12–0, pegasus 3–0, dog 0–0, crocodile 17–4, fandisk 0–0, monkey man 18–1, dragon 23–0, statue with elephant 66-6

puted by RK4) do cross each other. We don't exactly use their field representation: we perform the field interpolation on triangles in polar coordinates instead of Cartesian coordinates. This minor modification allows to work with direction fields, constraints singularities to be on vertices, allows to represent fields with high curvature, and is directly compatible with our field, resulting in more fair comparisons.

To trace streamlines on this field representation, we used an order four Runge-Kutta algorithm. This numerical integration scheme comes with the usual numerical imprecision, the difficulty to tune the time step parameter, and some thresholds required to manage singular points (reaching a sink, crossing a saddle vicinity, etc.). Moreover, to work on a triangulated surface, the algorithm must deal with high curvature due to the angle defect distributed on triangle corners, and numerical ambiguities (e.g. when the streamline have to follow sharp edges of a geometric feature). In our experiments, we use an average of 10 integration steps by triangle, and don't consider as errors crossings that occur in the one-ring of singular vertices (different strategies would have led to very different results).

In practice, with a smooth vector field, the numerical errors are similar to our algorithm without adaptive resolution ($\approx$ 10 loops on the spiral model). However, two proximal polylines traced with opposite directions are more likely to cross because RK4 evaluates the field at different positions (see Fig. 3.35).

With this approach, it is possible to decrease the time step to reduce the probability of crossings. However, it doesn't guarantee that no new crossings will appear (Fig. 3.40), and it requires to relaunch the crossing streamlines (and cancel all computations based on them).

- Our algorithm without adaptive resolution

  This solution is fair as long as we don't reach extreme cases as in the spiral test. In this configuration, we obtain a merge after ten loops, which is equivalent to the RK4 solution

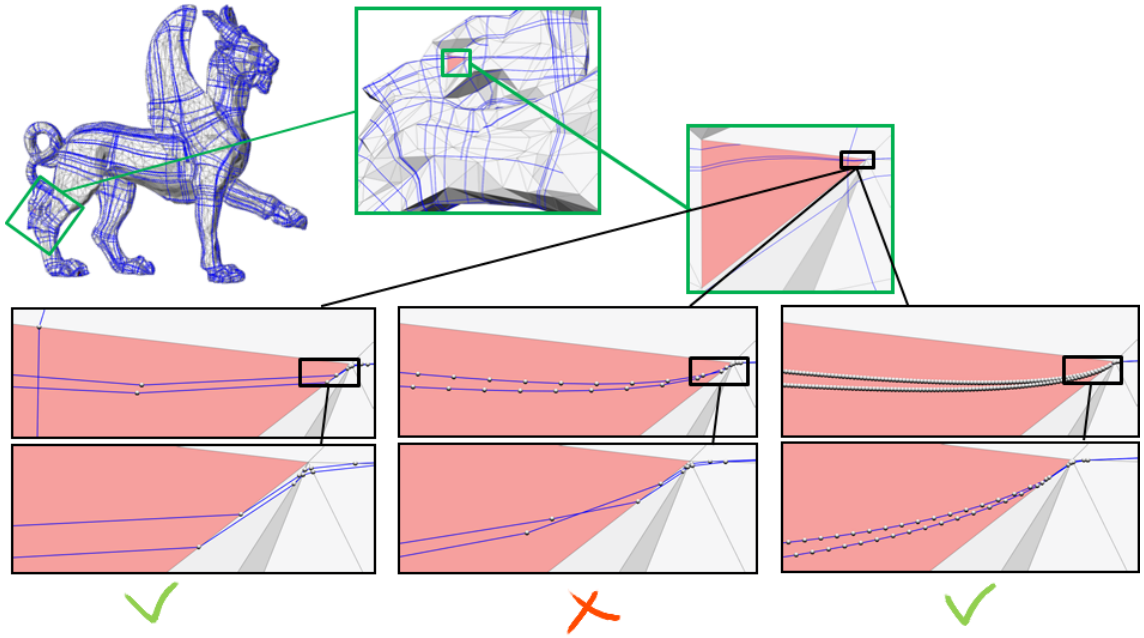


Figure 3.40: Impact of RK4 time step parameter. A triangle is crossed with 10, 100, and 1000 integration steps by triangle (in close-ups). A cross is detected with 100 integration steps.

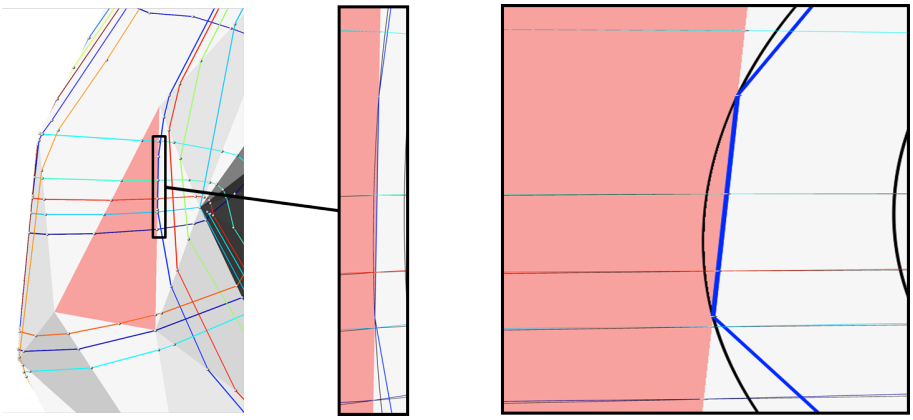


Figure 3.41: The blue streamline enters and leaves the pink triangle on the same edge. As illustrated in the stretched version of the close-up (Right), the continuous streamline (in black) is approximated by a segment (blue) contained in the edge.

(tested with 10, 100 or 1000 integration steps by triangle). In practice, it doesn't produce errors on all other tested examples. It also does not have any issues in the vicinity of singular vertices, and does not require any parameters tuning.

Our algorithm is the only one to ensure that polylines will never cross or merge. However, it could also work without adaptive resolution for common applications. A fair alternative solution would be to extend EdgeMaps to work with [ZMT06] field representation, and eventually our adaptive number representation. However, this latter solution would rely on RK4 to trace streamlines inside each triangle (to define the edge map), and it would be difficult to prove that no cross/merge could occur here.

Local overlaps

When a streamline enters and leaves a triangle on the same edge, the generated segment is localized on the edge (Fig. 3.41). If two such streamlines are traced, they may locally overlap on the triangle edge. However, if polylines are dedicated to cut the mesh into pieces, such overlaps will result in faces with degenerated geometry, but the desired topology. Another side effect of representing

streamlines by a single segment on each crossed triangle is that some points inside the triangle are not covered by polylines, as in the lower part of the triangle in Fig. 3.26—Right.

Conclusion

Tracing intersection-free polylines makes it easier to design new algorithms inspired by the continuous settings. Possible improvements of the method include using polycurves inside triangles, or finding a simpler way to cross each triangle. The question of the generalization to higher dimension arises naturally, but it is important to remember that the main issue (angle defect) requires that the metric is not induced by the object itself (for surfaces, it is induced by its embedding in $3D$ space, but volumes in $3D$ do not have this issue).

Chapter 4

Hexahedral-dominant meshing

In this chapter, we propose a new hexahedral-dominant meshing algorithm, that is to say an algorithm that creates from an input tetrahedral mesh a new mesh where most cells are hexahedral. We use the three steps pipeline (Fig. 4.1) introduced in [BRM+14]: (1) create a frame field to steer the placement and orientation of the cells; (2) generate a point set P that mostly corresponds to the vertices of a grid aligned with the frame field, (3) merge the tetrahedra in the Delaunay triangulation of P in order to create hexahedra. Our contribution is two-fold: the frame field and the point set are both generated by global optimization (instead of front propagation), and we provide a thorough analysis of the recombination problem (merging tetrahedra into hexahedra), leading to a both robust and efficient algorithm.

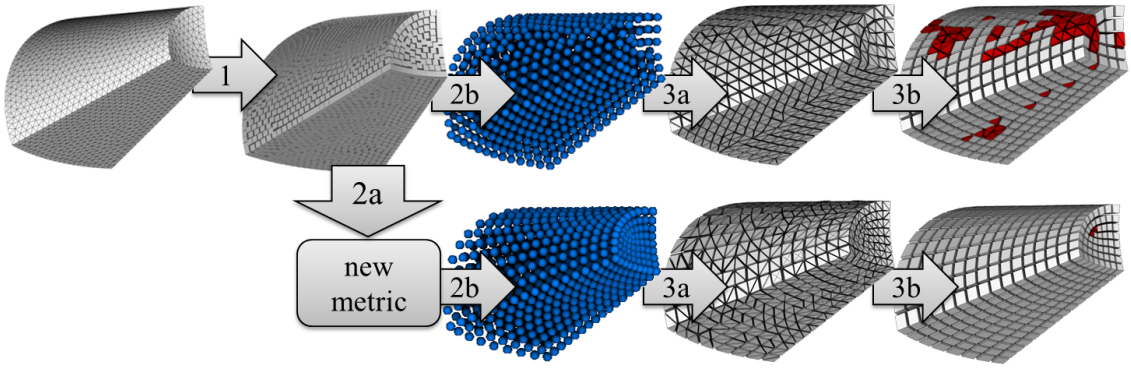


Figure 4.1: Starting from a tetrahedral mesh, we compute a frame field (1), optionally optimize the metric to increase the proportion of hexahedra (2a), compute a point set (2b), produce a new tetrahedral mesh with the point set as vertices and (3a) generate the hexahedral-dominant mesh by recombining the tetrahedra (3b).

Our two global optimizations used for generating the frame field and for the global parameterization are inspired from works in full hexahedral re-meshing [LLX+12, JHW+14]. We compute input frame fields using a variation of Huang's algorithm [HTWB11], the point set is defined by extracting the intersections of integer-valued iso-surfaces of a global parameterization. In our context, we relax the constraint induced by full hexahedral re-meshing (and thus we target *hexahedral-dominant* re-meshing instead). As a consequence, generating the smooth frame field does not require to predetermine its topology by optimizing combinatorial / integer variables, as done in previous work such as volumetric extensions of QuadCover [KNP07]. In our case, the global parameterization is computed by a volumetric extension of Periodic Global Parameterization [RLL+06] that can naturally make singularities emerge.

To extract the hexahedral-dominant mesh from the global parameterization, we first generate a point set P by mapping the points with integer-valued parameters. The points in P are then interconnected with tetrahedra using the Delaunay triangulation of P constrained to the boundary of the domain. We finally recombine the tetrahedra into hexahedra, with an algorithm that extends the analysis in [MT00]. We fill-in a gap in the original proof, extend the analysis to all the configurations with slivers, and identify some non-trivial forbidden configurations that are ruled-

out by our algorithm.

In most cases, our method outperforms the best hexahedral-dominant method [BRM+14] both in proportion and quality of hexahedral elements. This is mostly due to the front propagation approach they use, that creates discontinuities close to the medial axis. Compared with full hexahedral remeshing [LLX+12], the main advantage of our method is its ability of handling more industrial-size difficult cases (at the expense of introducing non hexahedral elements). The main drawback of our approach is that it may introduce non-hexahedral elements and/or non-natural branching structures, even in simple cases, where a full hexahedral mesher can avoid them and generate a more structured mesh, with for instance a constant number of layers of hexahedra in a thick surface.

Pipeline overview

Our method is summarized in Fig. 4.1. Starting from an input tetrahedral mesh, it consists in the following three steps:

1. **Generate the frame field that specifies the desired size and orientation of the elements (Section 4.1)**

   We set the desired edge size to be equal to the average edge size of the original mesh. It is also possible to explicitly specify the desired scale by an arbitrary function.

2. **Generate the point set P (Section 4.2)**

   - Modify the prescribed size and orientations in order to increase the proportion of hexahedra in the final mesh (§4.2.4), as shown in Fig. 4.1-bottom *(this sub-step is optional).*
   - Generate an atlas of grid-compatible maps (§4.2.2).
   - Extract the point set (§4.2.3).

3. **Generate the hexahedral dominant mesh (Section 4.3)**

   - Re-mesh the border of the domain using the points in P (§4.3.1);
   - generate a tetrahedral mesh using the Delaunay triangulation of the point set P constrained by the re-meshed border. *This sub-step uses state-of-the-art methods* [GHS90], [Si15].
   - Find all the candidate hexahedra that can be obtained by merging tetrahedra (§4.3.2).
   - Select among them the best mutually compatible set of hexahedra (§4.3.2).

Hexahedral-dominant meshing is often treated by adapting a hexahedral-only meshing method and completing the result with tetrahedra whenever the method gets stuck in configurations that cannot be meshed with hexahedra only. It is also recommended not to use it as a full automatic process [MTTT98]. Our strategy to tackle the problem is different: we first distribute points inside the volume to be meshed, then generate a tetrahedral mesh having these points as vertices, and finally form hexahedra by recombining tetrahedra from this mesh. This strategy decomposes the problem into two subproblems: **(1) Generating the points:** generate points that are likely to be the vertices of a nice hexahedral-dominant mesh and **(2) Recombining the tetrahedra into hexahedra:** finding the best hexahedra that can be recombined within a tetrahedral mesh.

1. **Generating the points:** In previous works, points were distributed by a centroidal Voronoi with a norm L_p in [LL10], or by a front propagation approach as used in [BRM+14]. We introduce another strategy inspired by re-meshing using a global parameterization.

   In $2D$, for quadrilateral re-meshing of triangulated surface, it is possible to define a cross field over the surface (4 orthogonal unit vectors of the tangent plane), then define a special atlas of the surface such that the pre-image of a unit grid defined in the maps gives a quadrilateral mesh of the surface. The first $2D$ method was introduced in [RLL+06], it produces quadrilateral meshes with very regular size, but not everywhere on the surface. These areas where the algorithm fails to produce quadrilaterals allow to introduce irregularities in the quad mesh. It corresponds either to singularities of the cross field, or to T-vertices that are

required to respect the desired scale of the quads. The following approaches [KNP07, BZK09] are able to generate a quadrilateral mesh on 100% of the surface, at the expand of creating more distorted quadrilaterals (they do not balance the field's curl by introducing T-vertices).

In $3D$, the algorithm in [NRP11] is a direct extension of the $2D$ [KNP07]. It can produce very nice results. However, its main restrictions is that it assumes to have as input a valid $3D$ frame field that corresponds to a hexahedral mesh, and it may significantly stretch the hexahedra in order to balance the frame field's curl. In many (simple) cases, a frame field can be initialized by [HTWB11], and locally optimized [LLX$^+$12, JHW$^+$14] to produce a field topology that corresponds to a hexahedral mesh. This solution is fast and provides excellent results provided that local modifications of the frame field are sufficient to make it compatible with hexahedral re-meshing, and that the stretch of hexahedra induced by the frame field's curl corresponds to the user-desired stretch. Another approach to ensure that the frame field's topology is compatible with re-meshing is to forbid having any singularity. This approach yields algorithms based on the generation of polycubes [GSZ11, LVS$^+$13, HJS$^+$14].

We instead choose to extend [RLL$^+$06] to the $3D$ case. As in the $2D$ case, it will not be able to produce hexahedra everywhere inside the volume, but it will produce hexahedra of desired stretch and orientation everywhere else in the volume, without any constraint on the frame field. The resulting mesh is hexahedral-dominant (rather than pure hexahedral), it has non-hexahedral elements where the Periodic Global Parameterization has singularities.

2. **Merging tetrahedra into hexahedra** Several algorithms were proposed to merge tetrahedra into hexahedra within a tetrahedral mesh [YS03], [MT00]. The latter reference provides an abstract formalization that can be used to systematically enumerate all the possible configurations. We give a complete explanation of this formalism, fill-in a gap in the original proof and then elaborate on it to obtain an exhaustive enumeration of all the decompositions of a hexahedron into tetrahedra, together with a complete understanding of the structure behind the set of all possible decompositions: any decomposition of the cube can be deduced by simple operations from six "atomic" configurations. This specific understanding of the problem leads to an algorithm that is both short, efficient and easy to implement.

4.1 On smooth frame field design

In computer graphics, a frame field can be defined on a surface ($2D$) or inside a volume ($3D$). For each point of the domain, it defines a set of 4 (resp. 6) unit vectors invariant by rotations of $\pi/2$ around the surface normal (resp. around any of its member vector). The main motivation to study these fields is to split the quad and hexahedral re-meshing problems into two steps: (1) the design of a smooth frame field, (2) and the partitioning of the domain by quads or hexes aligned with the frame field. Our objective is to unify the formulation of the $2D$ and $3D$ frame field design problems, and to use it to extend an efficient $2D$ algorithm to the $3D$ case.

In most cases, frame field design is formalized as the following optimization problem: find the smoothest frame field subject to some constraints. What makes them different from each other is obviously the dimension of the frames ($2D$ or $3D$), but also the definition of the field smoothness, the expression of the constraints, and the optimization method. Interestingly, the $2D$ case and the $3D$ case are addressed by very different strategies:

- In $2D$, the frame field design problem can be restated as a vector field design problem thanks to the introduction of the **"representation vector"**. In local polar coordinates, each vector of a frame has the same angle modulo $\pi/2$, if we multiply it by 4 we obtain a unique representation vector (modulo 2π). It is easy to derive optimization algorithms acting on the representation vectors. For simplicity reasons, we limit ourselves to planar frame fields and use the algorithm proposed by Kowalski *et al.* [KLF12] as reference.

- In $3D$, it is not possible to extend the idea of "representation vector". Instead, Huang *et al.* [HTWB11] propose to represent frames by **functions** defined on the sphere, refer to figure 4.2 for an illustration. A definition of the field smoothness is derived from this representation and optimized in a two step procedure: (1) initialization based on optimization of spherical harmonics coefficients in [HTWB11] or front propagation of boundary constraints in [LLX+12], followed by (2) smoothing iterations performed by L-BFGS on Euler angles representation of frames.

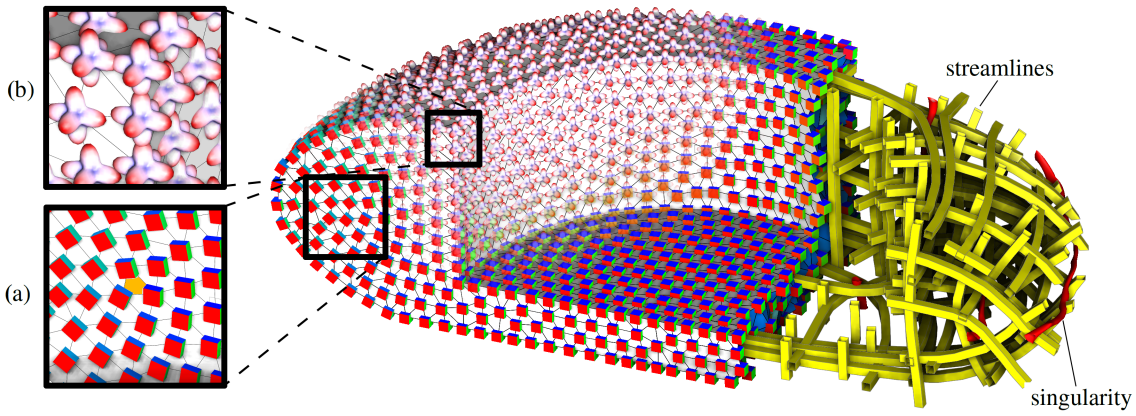


Figure 4.2: Orthonormal frames (close-up (a)) are represented by spherical harmonic functions (close-up (b)), attached to each vertex of a tetrahedral mesh. Streamlines and singularities of the field are shown in yellow and red, respectively.

Thus our goal is to better understand how $2D$ and $3D$ problems are related to each other and to extend [KLF12] to $3D$. We first show that [KLF12] can be derived with the formalization approach inherited from the $3D$ case, and then we extend it to $3D$ by the same logical flow. *Busy readers interested in only reproduction of the method can skip to implementation section §4.1.3, the only required tool is a linear solver, all calculations are given explicitly.*

4.1.1 In search of elusive ground truth, or $d \approx 0.85$

In this section we focus on the relation between the Dirichlet's energy and curvature of (discrete) vector fields. It is not a trivial problem, thus we focus on a $2D$ toy problem to gain some intuition about the relation. First of all, what is a direction field?

Direction (Frame) Field: a frame field can be seen as the orientation of the normal of the edges of a square as done in [RVLL08]. Hence a $2D$ frame field is a function that associates, to each point of a sub-domain of $\mathbb{R}^2$, a set of 4 unit vectors invariant under rotation by $\pi/2$.

Representation of a frame by an angle: a frame can be represented by an angle θ such that the set of vectors is given by complex numbers $e^{i(\theta+2k\pi/4)}$ with $k \in \{0,1,2,3\}$. In this representation, any $\theta = \theta_0 + 2k\pi/N$ represents the same frame. As a consequence, the continuity of direction fields is not equivalent to the continuity of their corresponding θ function.

Continuity: a frame field is continuous at P_0 if $\forall \varepsilon > 0\ \exists r$ s.t. $\forall P, |P - P_0| < r \Rightarrow \exists k \in \mathbb{N}$ s.t. $|\theta(P) - \theta(P_0) + 2k\pi/N| < \varepsilon$. Note that we took a standard definition of continuity, and added the integer k in it to account for the equivalence between frames represented by θ and $\theta + 2k\pi/N$.

Gradient: for a continuous frame field and its angle representation θ, for each disk inside the domain, there exists an integer valued function k such that $\theta' = \theta + k$ is a continuous real valued function. For ease of writing, we directly write $\nabla\theta$ for the gradient of θ'.

Singularity: most frame fields that are suitable for applications have their gradient $\nabla\theta$ defined everywhere on the domain except at a finite number of points. These points are called singularities, and can be classified by their index: the integral of $\nabla\theta/2\pi$ along an infinitesimal circle turning counter clockwise around the point.

Now we have all the vocabulary necessary to aboard the problem. Computer scientists always define smooth $2D$ frame fields in discrete setting: the field is defined on vertices (or facets) and the field smoothness is evaluated only between pairs of adjacent samples. The standard way to measure the smoothness of a field is through the field curvature. If the field is sampled on vertices, then the field curvature is defined as follows:

$$\sum_{ij} |\theta_i - \theta_j \mod \pi/2|^2,$$

where θ_i and θ_j are the field samples for adjacent vertices i and j, respectively.

In the literature the field curvature is often called Dirichlet's energy, however it is not appropriate, since Johann Peter Gustav Lejeune Dirichlet worked in continuous settings. Moreover, the problem to find a smooth (discrete) frame field is typically posed as a minimization of the field curvature. However, discrete field curvature is always finite, whereas the Dirichlet's integral diverges in the vicinity of singular points. Continuous counterpart of a 2D frame field singularity is the atan2 function. For example, if we take any neighborhood Ω of the point $(0,0)$ then the integral $\frac{1}{2}\int_\Omega \|\nabla \operatorname{atan2}(y,x)\|^2 dydx$ diverges.

This problem is often overlooked, the only (unsatisfactory) discussion we are aware of is the work by Knöppel *et al.* [KCPS13]. They rightfully mentioned that the frame field curvature is not a reliable measure of quality, since under refinement the energy contributed by a singularity grows without bound. They say that this energy is ill-defined, and then right in the next sentence they re-formulate it as an eigenvalue problem without suppressing unbounded potential [KCPS13, p. 8].

In this section we analyze a closed form solution for a $2D$ unit disc, comparing Dirichlet's energy with its discretization, the frame field curvature. The question we try to answer in this section is: "Does it make any sense to minimize for a divergent Dirichlet's energy?" The answer can not be directly projected to $3D$ frame fields or even to other $2D$ fields. The only thing we aim here for is the intuition about the problem.

Closed form solution for a disk

In this section we are looking for the minimal curvature frame field in a disk, with normal boundary constraints. With above definitions, the continuous formulation of the problem is very similar to the discrete formulation given in the core of the chapter.

The curvature is evaluated by the integral over the domain of the square of the magnitude of the gradient $\|\nabla\theta\|^2$, that is the Dirichlet energy of the function θ. The boundary constraint simply enforces (on the boundary) that $(\cos(\theta), \sin(\theta))$ is equal to the normal of the boundary of

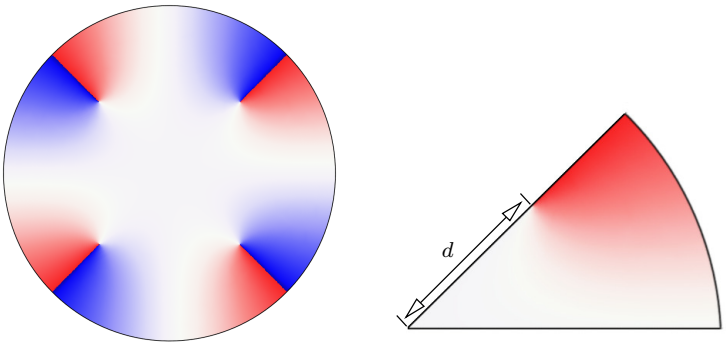


Figure 4.3: Angle function f_d plotted on the whole unit disk and the sector for which we actually solve the Laplace's equation $\Delta f_d(r,\theta) = 0$. Blue corresponds to $-\pi/4$ and red to $\pi/4$.

the domain. The minimization of the curvature is easy ($\Delta\theta = 0$) when a solution exists without singularities, but becomes much more difficult if position of singularities and index have to be automatically generated. Indeed, a singularity is a point where the gradient is undefined, so it must be removed from the curvature integration domain. Moreover, we will see that even when removing singularities, the integral diverges in the neighborhood of singularities.

From the symmetry of the problem we assume that the solution will be symmetric, i.e. there will be 4 index 1/4 singularities placed onto two perpendicular diameters of the disk, equidistant to the disk center. We denote this distance as d. An example plot is given on the left of the figure 4.3. The goal of this section is to answer the following question: *"Which value of d should we pick to get the smoothest field?"*

> **WARNING:** The frame angle θ will now be represented by the function f_d, and we re-use the notation θ to denote the angle of the points in polar coordinates.

More formally, we parameterize our disk by (r,θ) and we are looking for a function $f_d(r,\theta)$ that gives for any point of the disk the rotation angle (referred to as θ in the introduction) of the reference frame. Therefore the domain of values of the function f_d is $[-\pi/4, \pi/4]$. As we want to minimize the Dirichlet's energy, f_d must be harmonic. Due to the high symmetry of the problem, we restrain our domain to $\pi/4$-angle sector of the unit disk to simplify the notations.

Thus we are looking to solve the Laplace's equation $\Delta f_d(r,\theta) = 0$ under Dirichlet's boundary conditions:

$$\begin{cases} f_d(1,\theta) = \theta \\ f_d(0,\theta) = 0 \\ f_d(r,0) = 0 \\ f_d(r,\pi/4) = \begin{bmatrix} 0, r \leq d \\ \pi/4, r > d \end{bmatrix} \end{cases}$$

Given these boundary conditions the solution to the Dirichlet's problem exists and unique.

Divergent energy analysis Let us verify that the following function is the solution: [1]

$$f_d(r,\theta) = 1/4(\text{atan2}((rd)^4 \sin(4\theta), (rd)^4 \cos(4\theta) + 1) + \text{atan2}((r/d)^4 \sin(4\theta), (r/d)^4 \cos(4\theta) + 1))$$

[1] It is quite a detective story how we found this function. By the way, beware of mathematical software! We extensively used SageMath and Wolfram and found bugs in both. Just a simple screen-shot from https://cloud.sagemath.com:

```
version()
var('t,x')
f = sin(t)*atan2(2*sin(t),-2*cos(t)+1)
print f
print "numerical integral: ", numerical_integral(f,0,pi)[0]
print "symbolic integral: ", integrate(f,(t,0,pi))
```

```
'SageMath Version 6.7, Release Date: 2015-05-17'
(t, x)
arctan2(2*sin(t), -2*cos(t) + 1)*sin(t)
numerical integral:  2.35619449019
symbolic integral:  -5/4*pi
```

Test your software engineering intuition (without imagining the graph of the function): which answer is right?

Left image of the figure 4.3 gives a plot of the function $f_{0.6}([0,1],[0,2\pi])$. Right image of the figure 4.3 gives a plot of the function $f_{0.6}([0,1],[0,\pi/4])$.

The calculations are made with sage 5.11 (online version is available at cloud.sagemath.com). In this section for each computing stage we provide Sage code followed by a screen-shot of Sage answers. The following code verifies that the laplacian of the function f_d is indeed zero:

```
var('r,d')
var('t', latex_name=r"\theta")
f = 1/4*(atan2((r/d)^4*sin(4*t), (r/d)^4*cos(4*t)+1) +
         atan2(d^4*r^4*sin(4*t), d^4*r^4*cos(4*t)+1))
Lf = f.diff(r,r) + f.diff(r)/r + f.diff(t,t)/r^2
print "Lf =", Lf.full_simplify()
```

```
Lf = 0
```

Thus the laplacian is zero, it is straightforward to verify that the boundary conditions are satisfied. We are interested in minimizing the Dirichlet's energy $E(d)$ over $d \in [0,1]$, however the integral $E(d) = \int_0^1 \int_0^{\pi/4} ||\nabla f_d(r,\theta)||^2 r d\theta dr$ is divergent. Does it mean that any choice of d is equivalent and there is no way to pick the best one? No, it does not. Let us remove 2ε-width ring from our domain of integration (thus eliminating the singularities) and study the behavior of the rest of the integral:

$$E(d,\varepsilon) = \int_0^{d-\varepsilon} \int_0^{\pi/4} ||\nabla f_d(r,\theta)||^2 r d\theta dr + \int_{d+\varepsilon}^1 \int_0^{\pi/4} ||\nabla f_d(r,\theta)||^2 r d\theta dr.$$

$E(d,\varepsilon)$ is a convex function of d with a single minimum for any choice of ε. It means that if we exclude any neighborhood of the singularity point from the domain of integration, we can find the function with lowest variation.

Unfortunately we can not operate complex notations in the source code, so let us call by $g(r,\theta)$ the integrand $g(r,\theta) = ||\nabla f_d(r,\theta)||^2 r$ in the source code. Thus we are analyzing the function

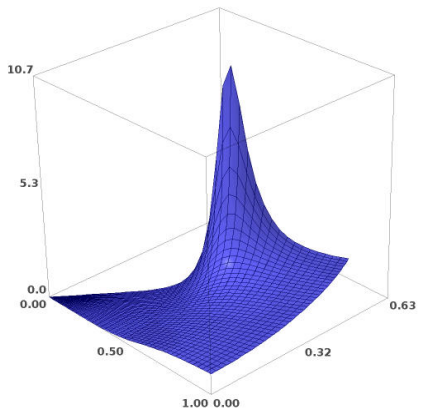
$$E(d,\varepsilon) = \int_0^{d-\varepsilon} \int_0^{\pi/4} g(r,\theta) d\theta dr + \int_{d+\varepsilon}^1 \int_0^{\pi/4} g(r,\theta) d\theta dr.$$

To calculate g we need to express the Cartesian gradient in polar coordinates, here we provide the expression for the integrand as well as its plot:

```
g = ((f.diff(r)*cos(t) - f.diff(t)*sin(t)/r)^2 +
     (f.diff(r)*sin(t) + f.diff(t)*cos(t)/r)^2)*r
g = g.simplify_full().trig_reduce()
show(g.factor())
parametric_plot3d([r*cos(t), r*sin(t), g.subs(d==.6)], (r,0,1), (t,0,pi/4-.1))
```

$$\frac{\left(4\,d^{12}r^4\cos(4\,\theta)+d^{16}+4\,d^8r^8+4\,d^4r^4\cos(4\,\theta)+2\,d^8+1\right)r^7}{2\,d^{12}r^{12}\cos(4\,\theta)+d^{16}r^8+d^8r^{16}+2\,d^8r^8\cos(8\,\theta)+2\,d^{12}r^4\cos(4\,\theta)+2\,d^4r^{12}\cos(4\,\theta)+2\,d^8r^8+2\,d^4r^4\cos(4\,\theta)+d^8+r^8}$$

Sleeping... Make Interactive



So we start with computing $\int_0^{\pi/4} g d\theta$. In the resulting integral we will get several expression involving atan2$(0,\dots)$, these are if-then-else depending on whether r belongs to $[0,d]$ or $[d,1]$ interval. To ease the job we calculate $\int_0^{\pi/4} g d\theta$ twice, once assuming $r<d$ and once $r>d$.

```
antiderivative = integrate(g, t)
assume(0<d,d<1,0<r,r<1,0<t,t<pi/4,r<d, (d*r)^4<1)
show(assumptions())
integral_a = (limit(antiderivative, t=pi/4, dir='-') -
              limit(antiderivative, t=0, dir='+')).full_simplify()
show(integral_a)
```

$$\left[0<d, d<1, 0<r, r<1, 0<\theta, \theta<\frac{1}{4}\,\pi, r<d, d^4r^4<1\right]$$

$$-\frac{4\,\pi d^8 r^{23}-\left(3\,\pi+3\,\pi d^{16}+2\,\pi d^8\right)r^{15}+\left(\pi+\pi d^{16}+2\,\pi d^8\right)r^7}{4\left(d^8r^{24}-\left(d^{16}+d^8+1\right)r^{16}+\left(d^{16}+d^8+1\right)r^8-d^8\right)}$$

```
forget(r<d)
assume(r>d)
show(assumptions())
integral_b = (limit(antiderivative, t=pi/4, dir='-') -
              limit(antiderivative, t=0, dir='+')).full_simplify()
forget(r>d)
show(integral_b)
```

$$\left[0<d, d<1, 0<r, r<1, 0<\theta, \theta<\frac{1}{4}\,\pi, d^4r^4<1, r>d\right]$$

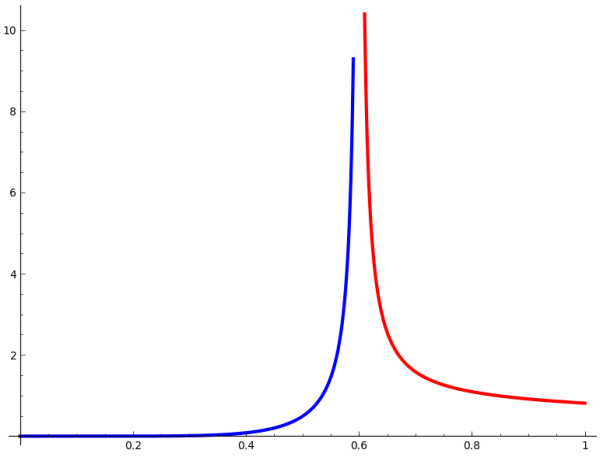
$$-\frac{\left(\pi-\pi d^{16}\right)r^7}{4\left(d^8r^{16}-\left(d^{16}+1\right)r^8+d^8\right)}$$

Now it is easy to assemble

$$E(d,\varepsilon)=\int_0^{d-\varepsilon}\text{integral\_a}\;dr+\int_{d+\varepsilon}^{1}\text{integral\_b}\;dr.$$

In the following figure the graph of integral_a is blue and integral_b is red. The plots are made for the value $d=0.6$.

```
plot(integral_a.subs(d==.6),(r,0,.6-.01), thickness=3, color='blue') + \
plot(integral_b.subs(d==.6),(r,.6+.01,1), thickness=3, color='red')
```

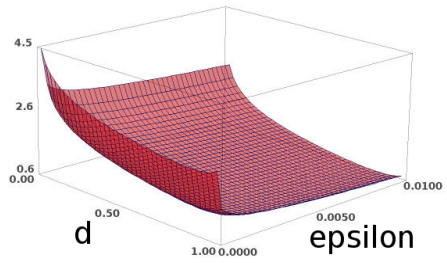


Now let us integrate over r to find the expression for $E(d,\varepsilon)$:

```
iia = integrate(integral_a, r).simplify_log()
iib = integrate(integral_b, r).simplify_log()
var('eps',latex_name=r"\varepsilon")
E = (limit(iia, r=d-eps, dir='-') - limit(iia, r=0, dir='+') +
     limit(iib, r=1, dir='-')     - limit(iib, r=d+eps, dir='+'))
E=E._maxima_().radcan().logcontract().sage()
E = E - I*pi^2/32 + SR(-1).log(hold=True)*pi/32;
E = E._maxima_().logcontract().sage()
show(E._maxima_().radcan().sage().factor())
plot3d(E,(d,0,1),(eps,1/10^6,1/10^2),color='red',opacity=.5)
```

$$\frac{1}{32}\pi\Big(\log\big(d^8+4d^7\varepsilon+6d^6\varepsilon^2+4d^5\varepsilon^3+d^4\varepsilon^4+1\big)-\log\big(-d^8+4d^7\varepsilon-6d^6\varepsilon^2+4d^5\varepsilon^3-d^4\varepsilon^4-1\big)-\log\big(2d^4+4d^3\varepsilon+6d^2\varepsilon^2+4d\varepsilon^3+\varepsilon^4\big)-\log\big(2d^4-4d^3\varepsilon$$

Sleeping... Make Interactive



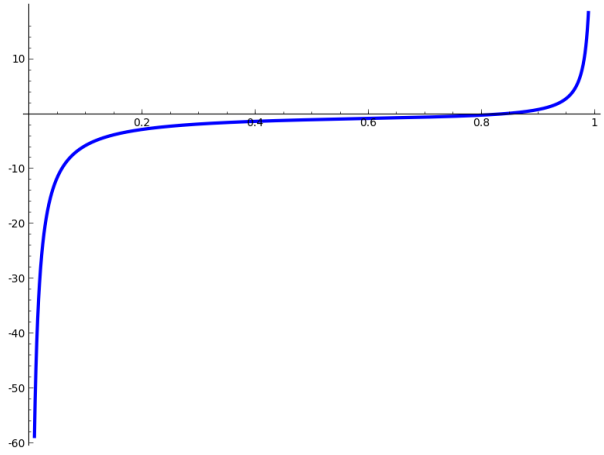
The expression for $E(d,\varepsilon)$ did not fit into the screen-shot, let us give it explicitly:

$$\begin{aligned}E(d,\varepsilon) = \frac{1}{32}\pi(&\log(d^8+4d^7\varepsilon+6d^6\varepsilon^2+4d^5\varepsilon^3+d^4\varepsilon^4+1)-\log(-d^8+4d^7\varepsilon-6d^6\varepsilon^2+4d^5\varepsilon^3-d^4\varepsilon^4-1)-\\&-\log(2d^4+4d^3\varepsilon+6d^2\varepsilon^2+4d\varepsilon^3+\varepsilon^4)-\log(2d^4-4d^3\varepsilon+6d^2\varepsilon^2-4d\varepsilon^3+\varepsilon^4)+\\&+\log(d^4+2d^3\varepsilon+d^2\varepsilon^2+1)-\log(d^4-2d^3\varepsilon+d^2\varepsilon^2+1)-2\log(d^4-4d^3\varepsilon+6d^2\varepsilon^2-4d\varepsilon^3+\varepsilon^4+1)-\\&-\log(2d^2+2d\varepsilon+\varepsilon^2)-\log(2d^2-2d\varepsilon+\varepsilon^2)+\log(d^2+d\varepsilon+1)+\log(d^2+d\varepsilon-1)-\\&-2\log(d^2-2d\varepsilon+\varepsilon^2+1)-\log(-d^2+d\varepsilon+1)-\log(-d^2+d\varepsilon-1)-\log(2d+\varepsilon)+\\&+8\log(d)-2\log(-d+\varepsilon+1)-2\log(-d+\varepsilon-1)-\log(-2d+\varepsilon)-2\log(\varepsilon)\Big)\end{aligned}$$

$E(d,\varepsilon)$ is a convex function of d with a single minimum for any choice of ε. Solving for $\lim_{\varepsilon\to 0}\frac{\partial E}{\partial d}=0$ gives $d=\sqrt[8]{3/11}$:

```
dEdd = E.diff(d).simplify_full().factor()
show(solve(limit(dEdd, eps=0, dir='+')==0, d)[6])
plot(limit(dEdd, eps=0, dir='+'), (d,.01,.99), thickness=3)
```

$$d=\frac{1}{11}\,11^{\frac{7}{8}}3^{\frac{1}{8}}$$



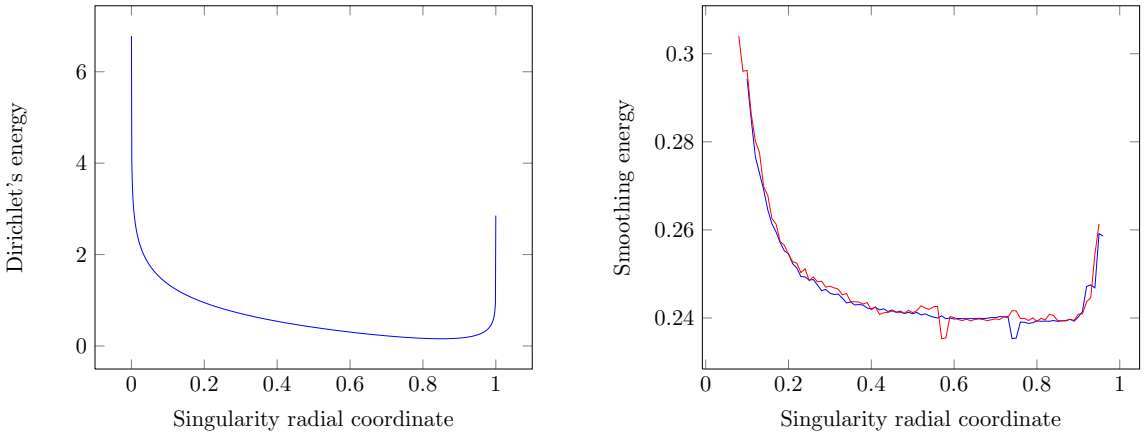


Figure 4.4: Left: Dirichlet's energy with a summand removed, right: the discretization for two different triangular meshes.

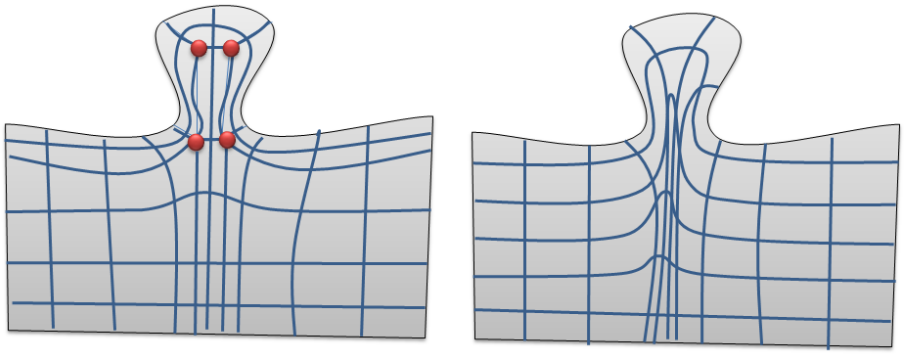


Figure 4.5: The discrete algorithm can generate singularities in order to minimize the field curvature elsewhere (left). The continuous settings have infinite penalty to singularities, so it will always produces a frame field without singularities when it is possible (right).

Comparison with the discrete problem We saw that in the continuous case Dirichlet's energy diverges in the vicinity of singular points, however the field curvature for the rest of the domain can be minimized. Left image of the figure 4.4 gives the plot of $\int \lim_{\varepsilon \to 0} \frac{\partial E}{\partial d} = 0$, this is the Dirichlet's energy up to a summation constant that we removed by derivating and by re-integrating back. The minimum is attained for $d = \sqrt[8]{3/11}$.

In the discrete case we place samples at triangular mesh vertices; then it is natural to define the field curvature as a sum over all edges of the squared angle difference of adjacent samples. This sum does not diverge, what is its behavior compared to the continuous case?

We sampled the function $f_d(r, \theta)$ for different values of d and then computed the discrete curvature. Right image of the figure 4.4 gives the plot of the sum for two different meshes. In continuous settings, the energy is high when singularities are too close to the disk center or to its boundary, but has low variations in a large interval in the middle. The discrete version have a similar global shape (up to a scale factor), but is strongly perturbed by the mesh. Indeed, it is more influenced by how the field rotation induced by the singularity can be dissipated in the singularity neighborhood than by the singularities position.

To conclude, even if the Dirichlet's energy diverges in the vicinity of singular points, it does make sense to minimize it's discrete counterpart under the condition that the underlying mesh is uniform.

Does it extend to other models? The continuous settings will always place a minimal number of singularities (with minimal absolute value of its index). As shown in Fig. 4.5, it can produce fields that are too distorted for the applications (e.g. quad re-meshing).

The discrete version is not always a very good approximation of the continuous version, but it punishes less singularities, which is often better for the applications.

4.1.2 Functional representation of frames in $2D$

This section introduces how to optimize $2D$ frame fields using a functional representation for each frame. While we do not claim any technical contribution in this section, we think that it is important to reformulate existing concepts using the functional representation, because $3D$ case inherits exactly the same difficulties and the intuition we gained in $2D$ helps to motivate the choices made for $3D$ fields. We derive an energy and the boundary conditions from this new representation. The resulting optimization problem is exactly the one usually solved by direction field algorithms based on the standard "representation vector". We show on a toy and a real examples that this optimization problem is much easier than directly minimizing the frame rotation. We then present the algorithm [KLF12] that we extend to $3D$ in the following section.

Problem settings

Given a $2D$ shape, frame field design in $2D$ consists of finding a smooth frame field aligned with the boundary of the shape. We formulate it as minimizing the field curvature, based on the following definitions:

- A **frame** is a set of 4 unit vectors $f = \{f_i\}, i \in [0, 3]$ that is invariant by a rotation of $\pi/2$ (Figure 4.6). It can be represented by the angle θ such that $\forall i, f_i = (\cos(\theta + i\pi/2), \sin(\theta + i\pi/2))$.
- A **frame field** is a frame per vertex of a $2D$ shape triangulation.
- The **boundary constraint:** a frame located on a boundary vertex must have one of its member vectors equal to the normal on the boundary.
- The **rotation angle between two frames** is the angle $\Delta\theta$ of the rotation that transforms one frame into the other. This angle being defined modulo $\pi/2$, we choose the $\Delta\theta$ with minimum absolute value.
- The **curvature of a frame field** is the sum over each edge of the squared rotation angle between frames that are defined at the edge extremities.
- A triangle is said to be **singular** if the sum of the rotation angles over his boundary is not equal to 0.

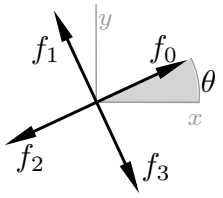


Figure 4.6: A $2D$ frame is a set f of 4 vectors f_0, f_1, f_2, f_3 invariant under rotation by $\pi/2$. Its angle representation is the rotation θ between the global axis x and f_0.

Representing frames by angles is simple, but it makes the field curvature hard to optimize [BZK09, RVLL06], and it does not extend nicely in $3D$. For these reasons, we propose an alternative representation based on functions, and use this curvature based formalization as a reference.

Functional approach: frame representation The frame aligned with coordinate axes is called the reference frame $\tilde{f} = \{(1, 0), (0, 1), (-1, 0), (0, -1)\}$. Instead of using the angle approach, we represent it by the function $\tilde{F}(\alpha) = \cos(4\alpha)$ with $\alpha \in [0, 2\pi[$ (Figure 4.7–left), that exhibits the same $\pi/2$ rotation invariance as the frame.

Any other frame f can be obtained by a rotation of $\tilde{f}$ by angle θ. The functional counterpart is to rotate the graph of the function $\tilde{F}$, namely any frame f can be represented by a function $F(\alpha) = \tilde{F}(\alpha - \theta) = \cos(4(\alpha - \theta))$ with $\alpha \in [0, 2\pi[$ (Figure 4.7–right).

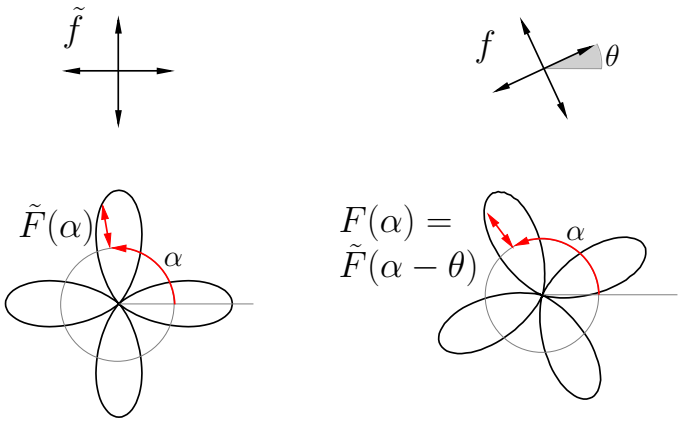


Figure 4.7: Parametric plot of the reference frame representation $\tilde{F}(\alpha)$ (left) and an arbitrary frame $F(\alpha)$ (right). The plot is made using $x(\alpha) = (1+F(\alpha))\cos(\alpha)$ and $y(\alpha) = (1+F(\alpha))\sin(\alpha)$ for $\alpha \in [0, 2\pi[$. It is easy to see that corresponding frames are aligned with maxima of the representation functions.

A compact representation of these functions is given by the trigonometric relation $f(\alpha) = \cos(4\alpha - 4\theta) = \cos(4\theta)\cos(4\alpha) + \sin(4\theta)\sin(4\alpha)$: we see that a frame function F can be represented by a $2D$ vector of coefficients $a = (a_0, a_1)^\top = (\cos(4\theta), \sin(4\theta))^\top$ in Fourier basis $B = (\cos(4\alpha), \sin(4\alpha))$ i.e. $F = Ba$.

A **coefficient vector** a **is feasible** if and only if there exists θ such that $a = (\cos(4\theta), \sin(4\theta))^\top$. Geometrically, a is constrained to live on a curve parameterized by θ. This curve represents, in coefficient space, all possible rotations of the reference frame. In $2D$, this curve is the unit circle, so the constraint on a is simply : $a^\top a = 1$.

At this point, we can observe that the coefficient vector a is exactly the representative vector used in the direction field literature. We can also notice that expressed in Cartesian coordinates, our reference frame function $\tilde{F}$ is the polynomial $4(x^4 + y^4) - 3$ restricted to the unit circle, thus it is also equivalent to the traceless symmetric 4^{th} order tensors manipulated in [PZ07].

Functional approach: objective function We have defined the field curvature as the sum over each edge of the squared difference between frames at the edges extremities. In our formalization, the difference between two frames (at vertices i and j) is no longer the rotation angle, but the L^2 norm of the difference between the corresponding functions : $\int_0^{2\pi}(F^j(\alpha) - F^i(\alpha))^2 d\alpha$. It leads to the new objective function:

$$\begin{aligned} E &= \sum_{ij} \int_0^{2\pi} (F^j(\alpha) - F^i(\alpha))^2 d\alpha \\ &= \sum_{ij} \int_0^{2\pi} (Ba^j - Ba^i)^2 d\alpha \\ &= \sum_{ij} (a^j - a^i)^\top \left(\int_0^{2\pi} B^\top B d\alpha \right) (a^j - a^i) \end{aligned}$$

As the function basis B is orthogonal, and all functions are of norm $\sqrt{\pi}$, so the expression simplifies to:

$$E = \pi \sum_{ij} \|a^j - a^i\|^2 \tag{4.1}$$

Functional approach: constraints As discussed in previous paragraph, the first constraint is clearly that the variables a^i must be feasible (i.e. there exists a frame represented by a^i).

The second constraint is that frames of boundary vertices i must have one member vector equals to the normal direction. If θ^i denotes the normal direction, the frame can be directly fixed

by satisfying two equations:

$$\begin{aligned} a_0^i &= \cos(4\theta^i) \\ a_1^i &= \sin(4\theta^i) \end{aligned} \tag{4.2}$$

However, as we already have the feasibility constraint ${a^i}^\top a^i = 1$, enforcing only one equation has the same effect:

$$a_0^i \cos(4\theta^i) + a_1^i \sin(4\theta^i) = 1. \tag{4.3}$$

Toy example

It is natural to ask the question: *"Does minimizing our energy minimizes the field curvature as well?"*

Two frames f^i and f^j are represented by a^i and a^j, both located on the unit circle. The field curvature measures the circle arc length between them, whereas our L^2 norm is the chord length between a^i and a^j.

From a practical point of view, we want to produce smooth fields, so most edges will have low curvature. In this case the objective function E is almost proportional to the field curvature. If, however, two adjacent frames are not similar (e.g. they are close to singularities), then the function E is smoother than the field curvature, making it easier for the optimization algorithm to move singularities.

Let us illustrate our intuition on a very simple interpolation example: a chain of four vertices having its extremities locked. The toy problem is therefore to find two frames interpolating the extremity frames.

If we represent two free frames by their angle θ_1 and θ_2, we can plot and compare the field curvature versus our objective function E (Figure 4.8). The field curvature is not smooth (it is piecewise quadratic) and we can observe that there are two local minima. Our objective function is smooth, and has exactly the same minima on this example. Note that it could also have a smaller number of minima e.g. if the constrained frames are more similar.

Figure 4.9 shows the frames corresponding to minima P_0 and P_1: they differ by the sense of rotation. The point P_0 minimizes objective function E and has better field curvature. In next two sections we expose current state of the art approach to minimization of the objective function.

Implementation

We have to minimize our objective function E (eq. (4.1)) with linear equality constraints on boundary vertices (eq. (4.2) or eq. (4.3)) and quadratic equality constraints ${a^i}^\top a^i = 1$ for each vertex i.

We use [KLF12]'s algorithm to solve this problem. It finds an initial solution by relaxing the quadratic feasibility constraints on a^i and finding the nearest feasible solution. Then it performs a number of smoothing iterations to ameliorate the solution. In order to respect the feasibility, the quadratic constraints are linearized at each smoothing step.

Initialization Here, we relax the feasibility constraints, so we need to minimize the quadratic function E subject to linear boundary constraints. To do it, we simply replace the linear constraints by a strong penalty term in the objective function, leading to a new quadratic function to optimize. This penalty method is very simple and sufficient in practice.

More precisely, the new quadratic function is expressed in the form $\|AX - b\|^2$ where A is a matrix, X our variable vector ($X_{2i} = a_0^i$ and $X_{2i+1} = a_1^i$) and b is a vector. The system of equations $AX - b = 0$ is constructed line-by-line:

- *initial objective function E:* for each edge ij, we add two equations (eq. (4.1)):

$$\begin{aligned} \sqrt{\pi}(X_{2i} - X_{2j}) &= 0 \\ \sqrt{\pi}(X_{2i+1} - X_{2j+1}) &= 0 \end{aligned}$$

- *boundary constraints:* for each boundary vertex i, we add two equations (eq. (4.2)):

$$\begin{aligned} \lambda X_{2i} &= \lambda \cos 4\theta^i \\ \lambda X_{2i+1} &= \lambda \sin 4\theta^i, \end{aligned}$$

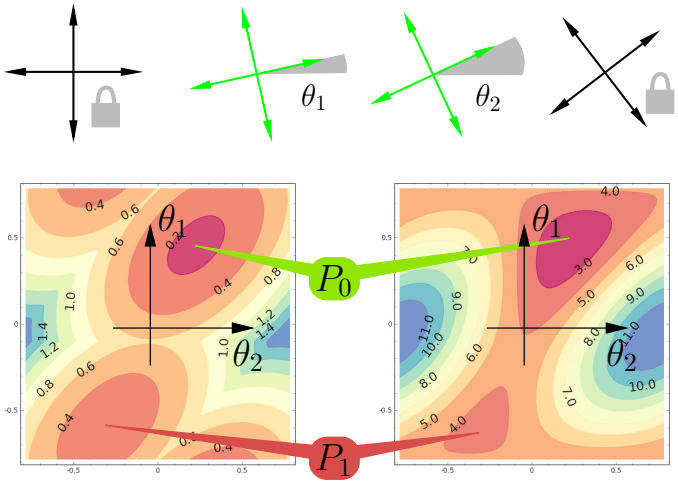


Figure 4.8: Top row: interpolation problem with two locked extremity frames. Bottom row, left: field curvature plot. Bottom row, right: our objective function E. The plots are made as functions of the rotations θ_1, θ_2 of interpolated frames. Both functions share same local minima P_0 and P_1.

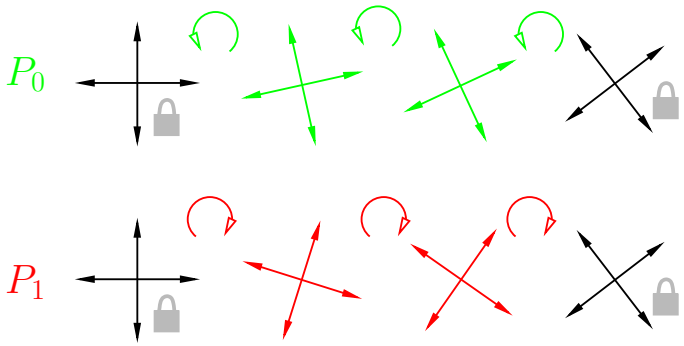


Figure 4.9: Two minima for the toy problem shown in Figure 4.8. P_0 turns the frames counter-clockwise while P_1 turns clockwise. P_0 minimizes energy E and has better field curvature.

where we set $\lambda = 100$ in our experiments.

From A and b, we just need to solve the linear system $A^\top AX = A^\top b$ to obtain a minimizer of $\|AX - b\|^2$. Then from X we can obtain an initialization of a^i by projecting corresponding vectors on the set of feasible coefficients:

$$a^i \leftarrow (X_{2i}, X_{2i+1})^\top / \|(X_{2i}, X_{2i+1})\|.$$

Smoothing iterations Each smoothing iteration is similar to the initialization problem, except that we add to the objective function a new quadratic penalty term that corresponds to a linear approximation of the feasibility constraint as done in [KLF12, p. 6]. As before it is expressed by a new set of linear equations when constructing A and b: for each vertex i, we add one equation $\lambda(X_{2i}a_0^i + X_{2i+1}a_1^i - 1) = 0$, where a^i denotes the solution obtained at the previous iteration.

To solve linear systems we use OpenNL library [Lé]: it automatically constructs $A^\top AX = A^\top b$ from the set of linear equations and then solves it by the conjugate gradient method.

Toy problem revisited

This section explains our optimization approach on the toy problem already presented above. As we mentioned before, at the initialization step we relax the constraints of feasibility of a^i. Unfortunately we can not plot the corresponding energy since without the constraints it becomes four-dimensional.

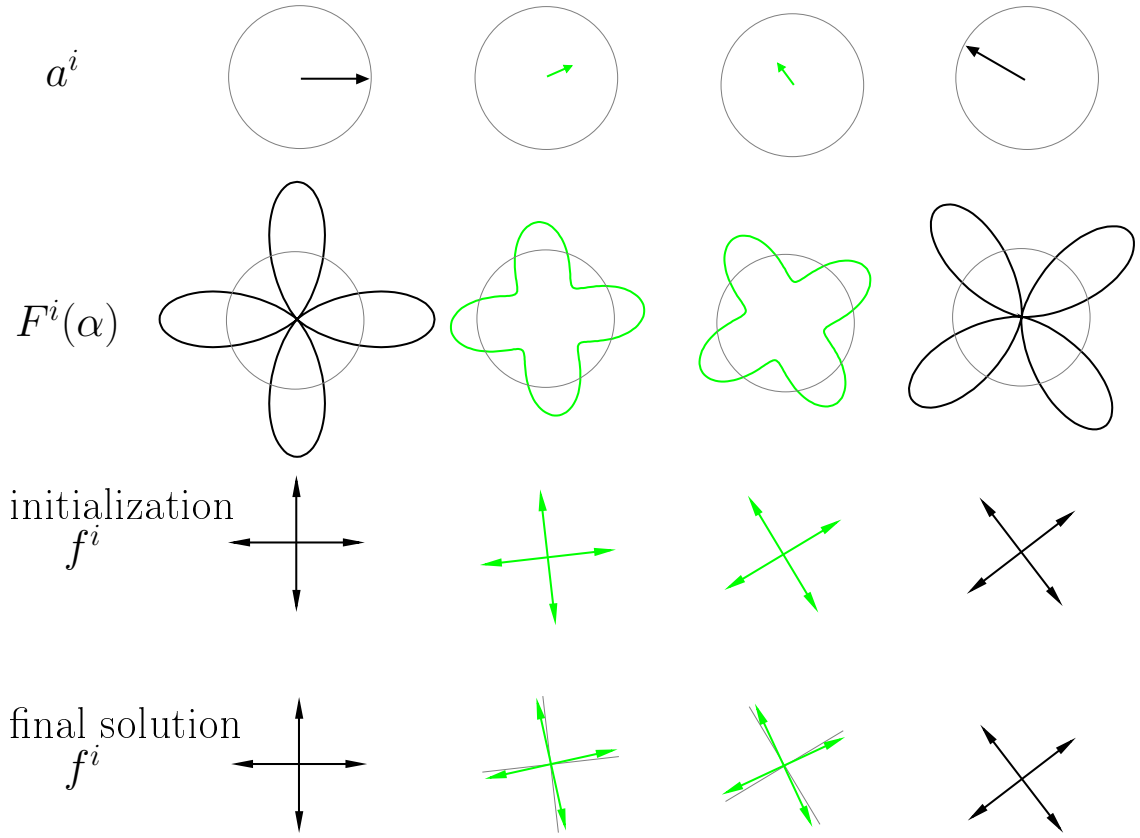


Figure 4.10: First row: initialization stage solution. Second row: corresponding functional interpretation. Third and fourth rows: frames obtained by the projection of initialization a^i and after smoothing iterations.

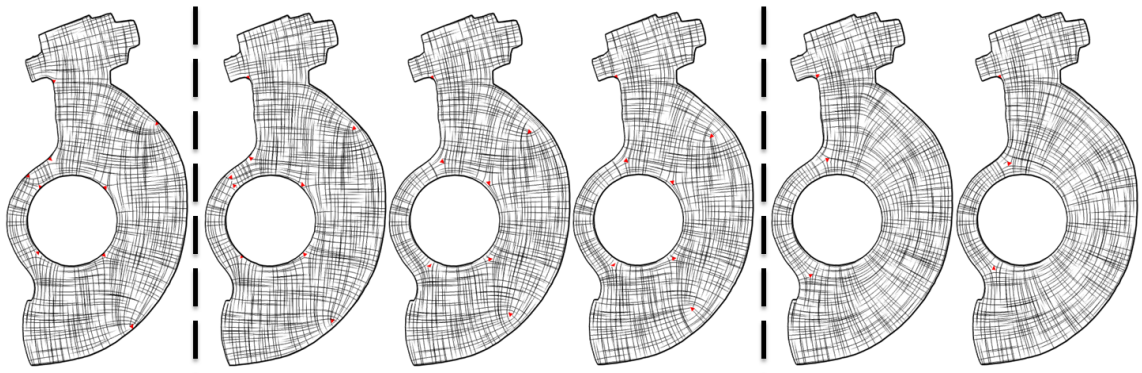


Figure 4.11: *Evaluation of 2D frame field optimization algorithms:* singular triangles are highlighted in red. Compared algorithms are: **(left)** steepest descent of the field curvature, initialized with an axis aligned frame field, **(middle)** smoothing iterations with our objective function E initialized with axis aligned frame field after 10^2, 10^3 and 10^6 iterations (from left to right), **(right)** initialization step alone (left) and the initialization step followed by 10^3 smoothing iterations (right).

Top row of Figure 4.10 shows the solution of the initialization stage. Intuitively, we allow the points a^i not to be on the unit circle. Hence a^1 and a^2 lie on the chord between locked points a^0 and a^3. Second row of Figure 4.10 shows the corresponding functions and the third row gives the frames obtained by projecting points a^i on the circle of constraints.

Note that the initialization stage produces the correct sense of rotation (Figure 4.9). However it does not directly produce the optimum point P_0. The reason being that the initialization stage produces points a^1 and a^2 (before normalization) in the way that all three chord segments are equal: $\|a_0 - a_1\| = \|a_1 - a_2\| = \|a_2 - a_3\|$. But after projecting the points onto the feasible circle (3rd row of Figure 4.10) the corresponding arc lengths are not equal. Therefore, we need a few smoothing iterations to reach the optimum (Figure 4.9—bottom row).

Results

Figure. 4.11–left shows the optimization of the field curvature by a gradient descent algorithm, initialized by an axis aligned frame field. We obtain a frame field that bends to match the boundary constraints, but its singularities remain on the boundary. The system is only able to reach the local minima corresponding to the initial field topology. The field curvature [2] is 34.21.

Figure 4.11–middle shows the optimization using only the smoothing iterations, initialized by an axis aligned frame field. We observe that smoothing iterations were able to slightly move the singularities away from the border and even to merge two singularities. It results in a better field curvature (equal to 31.41, 24.01 and 23.95 after 10^2, 10^3 and 10^6 iterations).

Figure 4.11–right shows that the initialization step alone finds a solution with a simple topology and a lower field curvature (20.84). Smoothing iterations further decrease the field curvature (to 17.91).

These observations suggest that our initialization step is mandatory, and smoothing iterations further improve the result. However, it is important to notice that each iteration takes approximately the same time as the initialization step.

Boundary constraints When working only with feasible solutions a single equation (equation (4.3)) is sufficient to enforce each boundary constraint. Using it for the initialization step is wrong: for example, a normal constraint of angle $\theta = 0$ forces $a_0 = 1$ but let a_1 free. As illustrated in Figure 4.12, it can produce very bad frames fields. Therefore we must use two separate equations (4.2).

There exists a similar issue in $3D$: Huang et al [HTWB11] use a $3D$ boundary condition that is not sufficient for the initialization step. It leads to a poor initialization for their smoothing algorithm, making it very slow, and getting possibly locked with a bad initial topology. This issue is discussed in details in the results section.

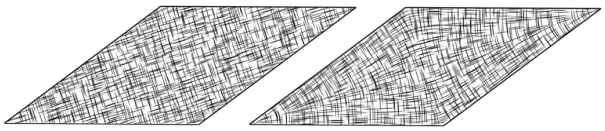


Figure 4.12: *Boundary constraints for global optimization:* if we only use one constraint (eq.4.3), the initialization step finds a constant frame field on a parallelogram (left). It is therefore mandatory to lock the frames to get the proper boundary constraints (right).

4.1.3 Optimization of $3D$ frame fields

Our objective is to extend to $3D$ the optimization algorithm presented in previous section. In $2D$, our framework allows to retrieve the representation vector that was the key to efficient optimization of direction fields. We now extend our framework to $3D$, find a generalization of this representation vector, express the boundary alignment condition with respect to this representation, and derive the optimization algorithm.

[2] The value of the field curvature is relevant only for comparing different fields on the same mesh.

Problem settings

The problem is to define, inside a $3D$ shape, a smooth frame field that is aligned with the boundary of the shape. We are working in discrete settings on a tet mesh. The problem to minimize the field curvature is defined as follows:

- The **reference frame** $\tilde{f}$ is the set of 6 unit vectors forming normals of a cube aligned with coordinate axes (Figure 4.13).
- A **frame** is the reference frame rotated by a 3×3 matrix R: $f = R\tilde{f}$. Note that multiplying a matrix by a set is a slight abuse of notation, it means that we obtain a new set where each vector is rotated by the given matrix.
- A **frame field** is the definition of a frame for each vertex of the tet mesh.
- The **boundary constraint**: The frame of a boundary vertex must have one of its member vectors equal to the normal of the boundary.
- The **rotation angle between two frames** is the minimal angle of rotation that brings one frame to the other.
- The **curvature of a frame field** is the sum, over each edge, of the squared rotation angle between adjacent frames.
- A tet is called **singular** if any of its triangles is singular.[3] The triangle ijk is singular if and only if $R^{ij} \times R^{jk} \times R^{ki} \neq Id$, where R^{ij} denotes the rotation matrix that brings the frame f^i to the frame f^j.

Frame representation

The reference frame $\tilde{f}$ is represented by the function $\tilde{F} = \sqrt{\frac{7}{12}}Y_{4,0} + \sqrt{\frac{5}{12}}Y_{4,4}$, where $Y_{l,m}$ is the real valued spherical harmonic of degree l and order m. These harmonics are sometimes called tesseral [Fer77, p. 74]. The list of harmonics of degree 4 can be found in [GWB96, p. 239]). Function $\tilde{F}$ is defined as $\mathbb{R}^3 \to \mathbb{R}$, but we are interested by its restriction to the unit sphere $\mathbb{S}^2 \to \mathbb{R}$.

This function is chosen because it is the lowest-frequency function exposing exactly the set of 24 rotational symmetries of the unit cube. To the best of our knowledge, it first appeared in works of Moakher [Moa09] under the name of cubic orientation distribution function, it was used to model distribution of fibers in physics. Huang *et al.* [HTWB11] were first to use this function for representing frame field samples.

Any other frame f can be obtained as a rotation of $\tilde{f}$ by a matrix R. It is represented by the function $F(P) = \tilde{F}(RP)$, where P is a point of the unit sphere (Figure 4.13).

$Y_{l,m}$ forms a functional basis over the unit sphere with an interesting property: applying a rotation to a spherical harmonic of degree l produces another harmonic of degree l. As a consequence, since we represent the reference frame by a sum of two spherical harmonics of degree 4, each frame function F can be represented in the basis $B = (Y_{4,-4}, Y_{4,-3}, \ldots, Y_{4,4})$. Using it, we can rewrite the expression for the reference frame function as $\tilde{F} = B\tilde{a}$ with $\tilde{a} = \left(0, 0, 0, 0, \sqrt{\frac{7}{12}}, 0, 0, 0, \sqrt{\frac{5}{12}}\right)^\top$. Any other frame $f = R\tilde{f}$ can be represented by $F = Ba$, where $a = R_B\tilde{a}$ with R_B being a 9×9 rotation matrix acting on coefficients space, defined as follows: let us denote by R^x, R^y and R^z 3×3 matrices of rotation around axes x, y and z respectively. Any frame f can be obtained as a composed rotation of the reference frame $\tilde{f}$, where the reference frame is the set of 6 unit vectors aligned with coordinate axes:

$$f = R^x(\alpha) \times R^y(\beta) \times R^z(\gamma) \times \tilde{f}.$$

If (α, β, γ) are Euler angles of rotation between a frame f and $\tilde{f}$, the representation vector a is calculated as $a = R^x_B(\alpha) \times R^y_B(\beta) \times R^z_B(\gamma) \times \tilde{a}$, where R^x_B, R^y_B and R^z_B are 9×9 matrices of

[3] We can define what a singular tet is, but we are not able to characterize them by an equivalent of the index in $2D$. This fact is discussed in the supplemental material.

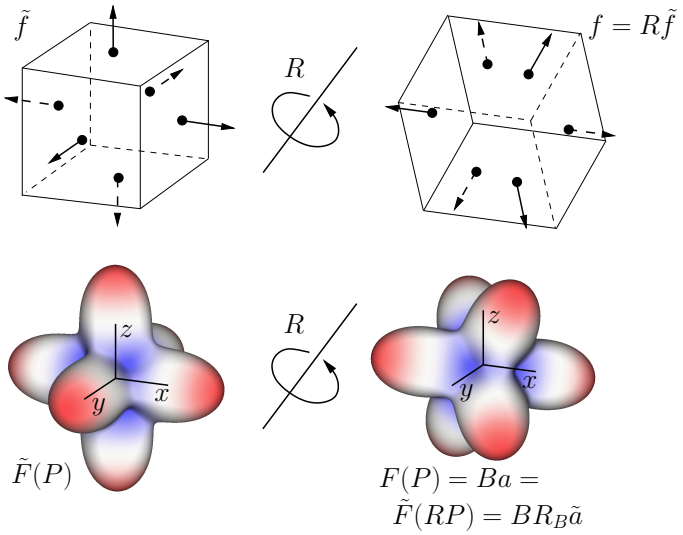


Figure 4.13: A frame f is the reference frame $\tilde{f}$ rotated by a 3×3 matrix R. The plots of the corresponding functions F and $\tilde{F}$ are also rotated by R, and their coefficients vectors verify $a = R_B\tilde{a}$ where R_B is a 9×9 rotation matrix.

rotation defined as follows:

$$R_B^z(\gamma) = \begin{pmatrix} \cos(4\gamma) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \sin(4\gamma) \\ 0 & \cos(3\gamma) & 0 & 0 & 0 & 0 & 0 & \sin(3\gamma) & 0 \\ 0 & 0 & \cos(2\gamma) & 0 & 0 & 0 & \sin(2\gamma) & 0 & 0 \\ 0 & 0 & 0 & \cos(\gamma) & 0 & \sin(\gamma) & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -\sin(\gamma) & 0 & \cos(\gamma) & 0 & 0 & 0 \\ 0 & 0 & -\sin(2\gamma) & 0 & 0 & 0 & \cos(2\gamma) & 0 & 0 \\ 0 & -\sin(3\gamma) & 0 & 0 & 0 & 0 & 0 & \cos(3\gamma) & 0 \\ -\sin(4\gamma) & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cos(4\gamma) \end{pmatrix}$$

$$R_B^x(\pi/2) = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & \sqrt{14}/4 & 0 & -\sqrt{2}/4 & 0 \\ 0 & -3/4 & 0 & \sqrt{7}/4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \sqrt{2}/4 & 0 & \sqrt{14}/4 & 0 \\ 0 & \sqrt{7}/4 & 0 & 3/4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3/8 & 0 & \sqrt{5}/4 & 0 & \sqrt{35}/8 \\ -\sqrt{14}/4 & 0 & -\sqrt{2}/4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sqrt{5}/4 & 0 & 1/2 & 0 & -\sqrt{7}/4 \\ \sqrt{2}/4 & 0 & -\sqrt{14}/4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sqrt{35}/8 & 0 & -\sqrt{7}/4 & 0 & 1/8 \end{pmatrix}$$

$$R_B^y(\beta) = R_B^x(\pi/2) \times R_B^z(\beta) \times R_B^x(\pi/2)^\top$$

$$R_B^x(\alpha) = R_B^y(\pi/2)^\top \times R_B^z(\alpha) \times R_B^y(\pi/2)$$

These matrices are called Wigner D-matrices and the literature on their construction is vast [CRL+89, BFB97, CIGR99, IR96]. However, as we are using degree 4 harmonics only, we performed the symbolic computation.

A **feasible coefficient vector** a is a vector that can be written as $a = R_B\tilde{a}$ where R_B is a $9D$ rotation matrix that can be derived from a $3D$ rotation. Geometrically, a is constrained to be on a manifold of dimension 3 embedded in the $9D$ coefficient space.

At this point we can consider the coefficient vector a as an extension of the representative vector used in the direction field literature. It is also the representation introduced in [HTWB11].

Other orientation fields In $2D$, our frames are 4-symmetry direction fields, and are very easy to extend to any integer number N of symmetries: it basically just replace all 4 by N in our equations.

For $3D$ frame fields, the representation by spherical harmonics was introduced in [Moa09, HTWB11] by converting from a quartic equation. These frames have 6 member vectors, and are invariant by rotation of $\pi/2$ around these vectors.

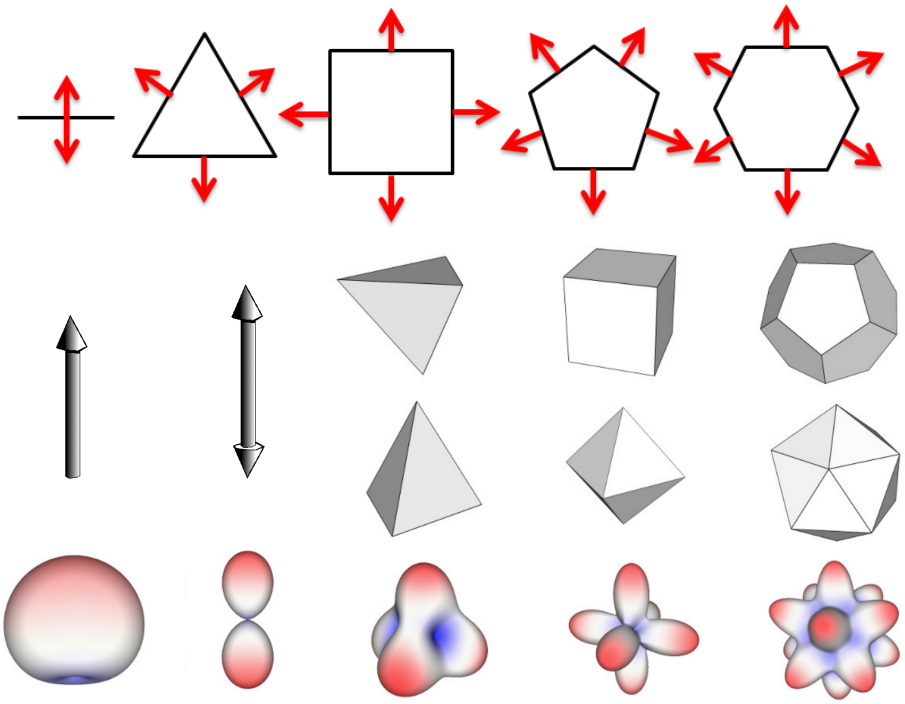


Figure 4.14: A N-symmetry direction field can be interpreted as the set of (edge) normal of regular polygons. This definition extends nicely to the $3D$ case where the set of facet normals of regular polyhedron is considered.

Frames in $3D$ can be extended to other type of orientation by considering that an orientation is the set of normal of a regular polyhedron. As a consequence, the only possible orientations have 4, 6, 8, 12 and 20 vector members. Moreover, we can notice that dual regular polyhedra are invariant by the same set of rotations, so there exist only 3 possible orientations with 4, 6 or 8, and 12 or 20 member vectors.

We propose a systematic way to construct spherical harmonic representations of these orientations. It also confirms that our frames have only one possible spherical harmonic representation.

A spherical harmonic able to represent a regular polyhedra must be invariant by exactly all rotations that transform the corresponding polyhedra (cube for a frame) into itself. We first compute the set of such rotations R^i: we combine all possible rotations around one face (4 for a frame), with all rotations that exchange this face with each faces (6 for a frame). It gives $4 \times 6 = 24$ rotations for a frame. Then, for each harmonic band, we compute the corresponding rotation matrices R^i_B, and we compute $\sum_i R^i_B a$ for a random non null coefficient vector a. This sum gives harmonics that have the desired rotation invariance. Indeed, applying a desired rotation to them is equivalent to re-arrange the term order in the sum.

The band 0 is always a solution $Y_{0,0}$, but it is not interesting as it is invariant by all rotations. The next bands typically gives only null vectors of coefficients because the band of harmonics is not sufficient to capture the rotations... until we reach a band with a non null coefficient vector. At this stage, we can check that the obtained vector is the only eigen vector of the $\sum_i R^i_B$ matrix. In practice, we did just verify that the coefficient vector is colinear to another invariant coefficient vector constructed from another random vector a. This was always true for the first successful band, and false for the next bands.

Using this technique, we produced SH representations of other types of orientation fields: vector fields $Y_{1,0}$, line fields $Y_{2,0}$, tetrahedron $Y_{3,2}$, cube or octahedron $\sqrt{7}Y_{4,0} + \sqrt{5}Y_{4,4}$ and dodecahedron or icosahedron $(\sqrt{154} + 14)Y_{6,-5} + (\sqrt{154} + 11)Y_{6,0}$. Figure 4.14 provides an illustration.

Note: we can also consider orientation with 1 or 2 member vectors. With one member vector, the coefficient vector equals the member vector. With two member vector, the coefficient vector is a basis of traceless symmetric order 2 tensor in $3D$.

Objective function

As in $2D$, the objective function is the sum, over each edge ij, of the squared difference between frames located at the edges extremities. In our formalism, the difference between two frames is not the rotation angle, but the L^2 norm of the difference between the corresponding functions: $\int_0^{2\pi}(F^j(\alpha) - F^i(\alpha))^2 d\alpha$. It gives the energy:

$$E = \sum_{ij} \int_0^{2\pi} (F^j(\alpha) - F^i(\alpha))^2 d\alpha$$

Here, the function basis B is orthonormal, so the expression simplifies to:

$$E = \sum_{ij} \|a^j - a^i\|^2 \tag{4.4}$$

Constraints

There are two types of constraints: each coefficient vector a^i must be feasible, and boundary frames must have one vector aligned with the normal of the volume boundary. The first constraint is presented in the frame representation section, and will be enforced by our optimizer in a dedicated "projection" step (the $3D$ counterpart of the normalization of a in the $2D$ case). Here we focus on the boundary constraint.

Smooth vertex: we assume first that there is only one normal associated with the vertex, it can be computed as the average normal vector of incident boundary triangles.

Case 1: the normal is equal to the z axis.

Let us first consider the case where the fixed vector (the surface normal) is the axis z. If we rotate $\tilde{F}$ around z by angle θ, we obtain $a = R_B\tilde{a}$ with R_B being a rotation around z. The simple structure of R_B together with the null coefficients of $\tilde{a}$ gives the equation:

$$a = \left(\sqrt{\frac{5}{12}}\sin 4\theta, 0, 0, 0, \sqrt{\frac{7}{12}}, 0, 0, 0, \sqrt{\frac{5}{12}}\cos 4\theta\right)^\top \tag{4.5}$$

As done in the construction of coefficient vectors in the $2D$ case, we can get rid of the angle θ by replacing it by a vector $c = (c_0, c_1) = \left(\sqrt{\frac{5}{12}}\cos 4\theta, \sqrt{\frac{5}{12}}\sin 4\theta\right)$.

$$a = \sqrt{\frac{7}{12}}(0,0,0,0,1,0,0,0,0)^\top + c_0(0,0,0,0,0,0,0,0,1)^\top + c_1(1,0,0,0,0,0,0,0,0)^\top$$

With this equation, all frames having a vector equal to z can be represented by the $2D$ vector c. As in the $2D$ case it comes with a norm constraint: $c_0^2 + c_1^2 = \frac{5}{12}$.

The variable c defines the rotation of the frame around the surface normal i.e. a $2D$ frame field. The optimization of this $2D$ frame field using c as variables is exactly what we did in $2D$ by introducing the coefficient vector a. Our $3D$ solution restricted to the object boundary is therefore almost [4] equivalent to our $2D$ solution (Figure 4.15).

Case 2: the normal is not equal to the z axis.

To handle this more general case, we rotate the constraint. If we want the vector $\vec{n}$ to be preserved, we first compute a rotation that brings z axis to $\vec{n}$. From this rotation, we compute the corresponding $9D$ rotation matrix R_B, and derive the constraints:

$$\begin{aligned} a &= \sqrt{\frac{7}{12}} R_B(0,0,0,0,1,0,0,0,0)^\top && (4.6)\\ &+ c_0 R_B(0,0,0,0,0,0,0,0,1)^\top && (4.7)\\ &+ c_1 R_B(1,0,0,0,0,0,0,0,0)^\top && (4.8) \end{aligned}$$

This expression of the normal constraint gives us a set of 9 linear equations per boundary vertex. It introduces two new variables c_0 and c_1, and a quadratic constraint $c^\top c = 5/12$.

Note As in the $2D$ case, the boundary constraint has a simpler expression [HTWB11] : $a^\top R_B(0,0,0,0,1,0,0,0,0)^\top = \sqrt{\frac{7}{12}}$ that is valid only if all a^i are feasible. Consequently, it cannot be used safely during the initialization step (see Figure 4.18).

[4] The boundary has curvature that was not assumed in our $2D$ frame fields.

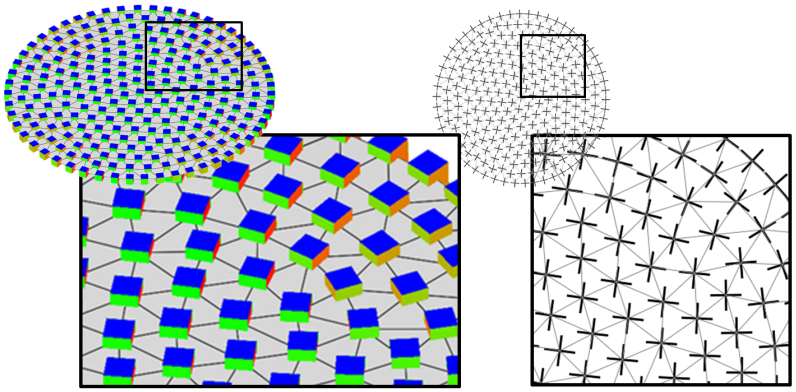


Figure 4.15: A $3D$ frame field produced on a ($2D$) disk (Left) produces the $2D$ frame field we could obtain from the $2D$ algorithm (Right).

Non smooth vertices: frames of vertices located on hard edges have to conform to more than one normal. These vertices have multiple normal constraints, we pick two normals that are almost orthogonal, perturb them (by rotations around their cross product vector) to make them orthogonal, and compute the rotation that brings x to the first normal, and y to the second normal. We compute the corresponding coefficient space rotation R_B and fix the frame coefficient vector a^i to $R_B\tilde{a}$.

Implementation

We have to minimize our objective function (eq. (4.4)) with linear equality constraints on boundary vertices eq. (4.6), quadratic equality constraints $c^i \cdot {c^i}^\top = 1$ on boundary vertices, and the constraint that each a^i is feasible.

As in the $2D$ case, our minimization algorithm (Algo. 3) is formulated as a series of least squares problems (minimize $\|AX - b\|^2$), where A and b are constructed without the feasibility constraint of a^i at the first iteration (initialization), and with a linear approximation of it in the subsequent iterations (smoothing iterations).

- **Initialization:** Our variable vector X must represent the representation vectors a^i but also the c^i variables introduced to express the boundary constraint. To do so, we first reorder vertices to have boundary vertices first [5]. We can then organize the variable vector X by blocks: $X[9i + d] = a^i_d$, and $X[9n_v + 2i + d] = c^i_d$ where n_v is the number of vertices.

  As in the $2D$ case, the matrix A and the vector b are constructed iteratively by adding new equations for the objective function (algorithm 5) and the boundary constraints (algorithm 4). In our approach, we do not explicitly enforce the feasibility of c^i ($c^i \cdot {c^i}^\top = 5/7$), but it will be indirectly respected by the feasibility of a^i.

  The projection of a^i on the set of feasible coefficient vectors is no longer a simple normalization. Instead we perform, for each vertex, a gradient descent (algorithm 7) initialized by $\tilde{a}$. More precisely, starting with $\tilde{a}$ we rotate our current frame gradually in order to minimize the distance between the current frame function and the function to be projected. The gradient is evaluated by calculating the variation of the L^2 norm induced by infinitesimal rotation matrices with Euler's angles.

- **Smoothing iteration:** For the linearized feasibility constraint of a^i, we must also add 3 extra variables per vertex to our system. These variables account for the position in a local basis of the tangent space of the $3D$ manifold of feasible a^i. The introduction of these constraints in the matrix A is detailed in algorithm 6.

  Note that this step requires to linearize 9×9 rotation matrix R_B. To do so, we define matrices E^x_B, E^y_B and E^z_B as follows:

[5] It is possible to increase the performances by $\approx 30\%$ by doing a Hilbert sort in the boundary vertices block, and another one in the free vertices block

$$E_B^x = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\sqrt{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\sqrt{7/2} & 0 & -\sqrt{2} \\ 0 & 0 & 0 & 0 & 0 & -3/\sqrt{2} & 0 & -\sqrt{7/2} & 0 \\ 0 & 0 & 0 & 0 & -\sqrt{10} & 0 & -3/\sqrt{2} & 0 & 0 \\ 0 & 0 & 0 & \sqrt{10} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3/\sqrt{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sqrt{7/2} & 0 & 3/\sqrt{2} & 0 & 0 & 0 & 0 & 0 \\ \sqrt{2} & 0 & \sqrt{7/2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sqrt{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$E_B^y = \begin{pmatrix} 0 & \sqrt{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -\sqrt{2} & 0 & \sqrt{7/2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -\sqrt{7/2} & 0 & 3/\sqrt{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -3/\sqrt{2} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -\sqrt{10} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sqrt{10} & 0 & -3/\sqrt{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3/\sqrt{2} & 0 & -\sqrt{7/2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \sqrt{7/2} & 0 & -\sqrt{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \sqrt{2} & 0 \end{pmatrix}$$

$$E_B^z = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

It is easy to verify that these matrices are chosen to verify the following equations for small rotations α, β, γ:

$$\begin{aligned} R_B^x(\alpha) &= I_{9\times 9} + \alpha E_B^x + o(|\alpha|) \\ R_B^y(\beta) &= I_{9\times 9} + \beta E_B^y + o(|\beta|) \\ R_B^z(\gamma) &= I_{9\times 9} + \gamma E_B^z + o(|\gamma|). \end{aligned}$$

Finally, for small rotations the multiplication is commutative:

$$\begin{aligned} R_B(\alpha,\beta,\gamma) &= R_B^x(\alpha) \times R_B^y(\beta) \times R_B^z(\gamma) = \\ &= I_{9\times 9} + \alpha E_B^x + \beta E_B^y + \gamma E_B^z + o(|\alpha| + |\beta| + |\gamma|). \end{aligned}$$

This frame field design algorithm can be implemented without being expert in spherical harmonics. We give explicit construction of matrices R_B, E^x, E^y, E^z. The system to solve $A^\top AX = A^\top b$ is simply a linear system with a positive definite matrix. We use the OpenNL library [Lé] because the system can be directly constructed from the equations (lines of A and elements of b).

In order to keep the algorithm easy to read, we did not detail how to lock frames for vertices with multiple normal constraints.

Results

It is impossible to compare frame field design algorithms only from the images and results presented in the state of the art papers. First of all, our implementation of [HTWB11] has significantly better performances compared to what was presented in the original paper. Next, Li *et al.* [LLX+12] did not present any frame field results, but only hex meshes that was the main focus of their work. Therefore, we implemented both methods; there are few points worth noting:

Sampling: In previous works the frame fields were sampled either on each tet face or on each tet. Instead we sample it on vertices, otherwise we would not be able to compare corresponding energies.

Gimbal Lock: Both Huang and Li use Euler angles as variables in their L-BFGS optimization, which have numerical issues when the angles are close to the gimbal lock. Note that each frame can be represented by 48 triplets of equivalent Euler angles. In our implementation we choose the triplet maximizing the distance to the nearest gimbal lock.

Rendering: For rendering purposes, we rely on a combination of techniques (Figure 4.17) to show the spherical harmonics field, the frame field (locally and globally) and the field topology.

Algorithm 3: Frame field optimization

Input:

- A tetrahedral mesh $\mathcal{M}$ with:
  - n_v vertices including n_l vertices with normal constraint
  - edges $\mathcal{E}$
- number of smoothing iterations N

Output: A frame f^i for each vertex i

```
sort(M, E); // vertex i is a boundary vertex ⟺ i < n_l
foreach I ∈ 0...N do
    // A and b will be constructed iteratively
    create matrix A with 0 rows and 9n_v + 2n_l + 3n_v columns;
    create vector b of size 0;
    add_smoothing_terms(M, A, b);
    add_normal_constraints(M, A, b);
    // add constraints only if we are in a smoothing iteration
    if I > 0 then
        add_local_optim_constraints({a^i}, M, A, b);
    end
    // solve A^⊤AX = A^⊤b
    X ← call_least_squares_solver(A, b);
    // find the frame for each vertex
    foreach i < n_v do
        a^i ← X[9i...9i + 8];
        (f^i, a^i) ← closest_frame(a^i);
    end
end
```

Algorithm 4: add_normal_constraints

Input: A tetrahedral mesh $\mathcal{M}$, matrix A, row b

Output: Modified matrix A and vector b

```
// enforcing normal constraints by quadratic penalty
foreach i < n_l do
    estimate normal n at vertex i;
    find Euler angles (α, β, γ) to rotate z-axis to n;
    find 9 × 9 rotation matrix R_B;
    h_0 ← R_B × (1,0,0,0,0,0,0,0,0)^⊤;
    h_4 ← R_B × (0,0,0,0,1,0,0,0,0)^⊤;
    h_8 ← R_B × (0,0,0,0,0,0,0,0,1)^⊤;
    λ ← 100; // quadratic penalty multiplier
    foreach d ∈ 0...8 do
        create vector row;
        row[9i + d] ← λ;
        row[9n_v + 2i + 0] ← λh_0[d];
        row[9n_v + 2i + 1] ← λh_8[d];
        A.add_row(row);
        b.push(λ√(7/12)h_4[d]);
    end
end
```

Algorithm 5: add_smoothing_terms

Input: A tetrahedral mesh $\mathcal{M}$, matrix A, row b
Output: Modified matrix A and vector b

foreach $ij \in \mathcal{E}$ **do**
  foreach $d \in 0 \dots 8$ **do**
    create vector row;
    $row[9i + d] \leftarrow 1$;
    $row[9j + d] \leftarrow -1$;
    A.add_row(row);
    b.push(0);
  end
end

Algorithm 6: add_local_optim_constraints

Input: Previous solution $\{a^i\}_{i<n_v}$, tetrahedral mesh $\mathcal{M}$, matrix A, row b
Output: Modified matrix A and vector b

foreach $i < n_v$ **do**
  $c_x \leftarrow E^x \times a^i$;
  $c_y \leftarrow E^y \times a^i$;
  $c_z \leftarrow E^z \times a^i$;
  $\lambda \leftarrow 100$; // *quadratic penalty multiplier*
  foreach $d \in 0 \dots 8$ **do**
    create vector row;
    $row[9i + d] \leftarrow \lambda$;
    $row[9n_v + 2n_l + 3i + 0] \leftarrow -\lambda c_x[d]$;
    $row[9n_v + 2n_l + 3i + 1] \leftarrow -\lambda c_y[d]$;
    $row[9n_v + 2n_l + 3i + 2] \leftarrow -\lambda c_z[d]$;
    A.add_row(row);
    b.push($\lambda a^i[d]$);
  end
end

Algorithm 7: closest_frame

Input: 9-component vector q
Output: A frame f and its representation vector a

$f \leftarrow \tilde{f}$;
$a \leftarrow \tilde{a}$;
$s \leftarrow 10^{-1}$; // *optimization step size*
$\varepsilon \leftarrow 10^{-4}$; // *step threshold*
$q \leftarrow q/|q|$;
while *True* **do**
  $g \leftarrow (q^\top E_B^x a, q^\top E_B^y a, q^\top E_B^z a)$; // *gradient in point a*
  if $\|g\| < \varepsilon$ **then**
    break;
  end
  $R_B \leftarrow R_B^x(s \cdot g[0]) \times R_B^y(s \cdot g[1]) \times R_B^z(s \cdot g[2])$;
  $R \leftarrow R^x(s \cdot g[0]) \times R^y(s \cdot g[1]) \times R^z(s \cdot g[2])$;
  $a \leftarrow R_B a$;
  $f \leftarrow Rf$;
end
return f, a;

Comparison with Huang's method Recall that Huang *et al.* proposed a method in two steps:

- find an initial frame field by solving a linear system and projecting the solution onto the manifold of feasible solutions
- represent each frame by a triplet of Euler angles and optimize the smoothness using an L-BFGS descent method.

Our implementation produces results very similar to those presented in [HTWB11], but with significantly better timings. For example, the rock-arm (Figure 4.16) with one million tetrahedra takes about 10 minutes on a single thread application on a Dell M6600 laptop compared to 155 minutes obtained by Huang *et al.* on a two-thread i7 processor.

Huang's initialization is very similar to ours, however their boundary condition is not sufficient in this case (it requires the SH coefficients to be feasible). Moreover, they enforce the boundary condition with a penalty term that is very light (10^{-2} weight). As a consequence, their initial fields are almost constant everywhere (it maximizes the smoothness), with a topology very far from being optimal. The smoothing iterations are performed with much higher weight (10^3) of the penalty term using L-BFGS.

After the initialization step we measured the deviation of the field from the given constraints on the rock-arm model. Note that the penalty term being the sum of deviations over all vertices, we can conclude that deviation at a given vertex belongs to $[0, 2\sqrt{7/12}]$. Thus on the rock-arm the initial frame field has the average deviation of 0.34, whereas the maximum frame deviation is 0.96.

If we use a much higher penalty weight to enforce the boundary constraint, we obtain an initial frame field with average deviation from constraints equal to 0.07 and maximum deviation equal to 0.75. The field has better topology and L-BFGS converges faster for this initialization. The initialization provided by our method has average deviation from constraints equal to 10^{-8} with maximum deviation of 10^{-7}.

Figure 4.16 gives an illustration, it compares three methods: Huang's algorithm (left image) Huang's algorithm with much higher penalty weight (middle image) and our initialization followed by Huang's smoothing iterations (right).

Huang's paper is the pioneer work and the main contribution in [HTWB11] was the introduction of the energy used in frame field optimization, however the initialization is not very good and smoothing stage was also outperformed by later works.

Comparison with Li's method Li's initialization computes a $2D$ frame field on the surface, then propagates it inside the volume by advancing front. As a consequence, the initial field perfectly matches boundary constraints, but is discontinuous across its medial axis.

The smoothing iterations are performed by L-BFGS. It acts on a new set of variables that characterizes, per vertex, the rotation that brings the reference frame to the current frame. For the frames located inside the object, variables are the Euler angles as in Huang's method. For frames located on the object boundary, the rotation is characterized by a single rotation angle around the normal vector.

The frame field results presented in their paper were limited to an ellipsoid and a sphere (frame field design was not the primary objective). We guess that most of presented results were not fully automatically generated frame fields, as they wrote: *"For instance, we use guiding boxes to modify the frames inside the narrow ears of Bunny (Figure 13-a) and the head of rock arm (Figure 13-b) to reduce singularities"*.

Moreover, before implementing the method, we thought that their algorithm was strongly limited by the original field topology from the sentence: *"However, our propagation-based frame field initialization likely generates singular edges around the medial axis of the volume, and most of them cannot be eliminated by frame optimization."* Surprisingly, our implementation of their method is able to generate smooth frame fields automatically in most situations, even when the topology of the initial field is complex close to the medial axis. Our implementation is slightly different:

- we initialize the $2D$ field by our $3D$ algorithm restricted to boundary vertices
- we sample the field on vertices
- and we prevent gimbal locks by a proper initialization of Euler angles.

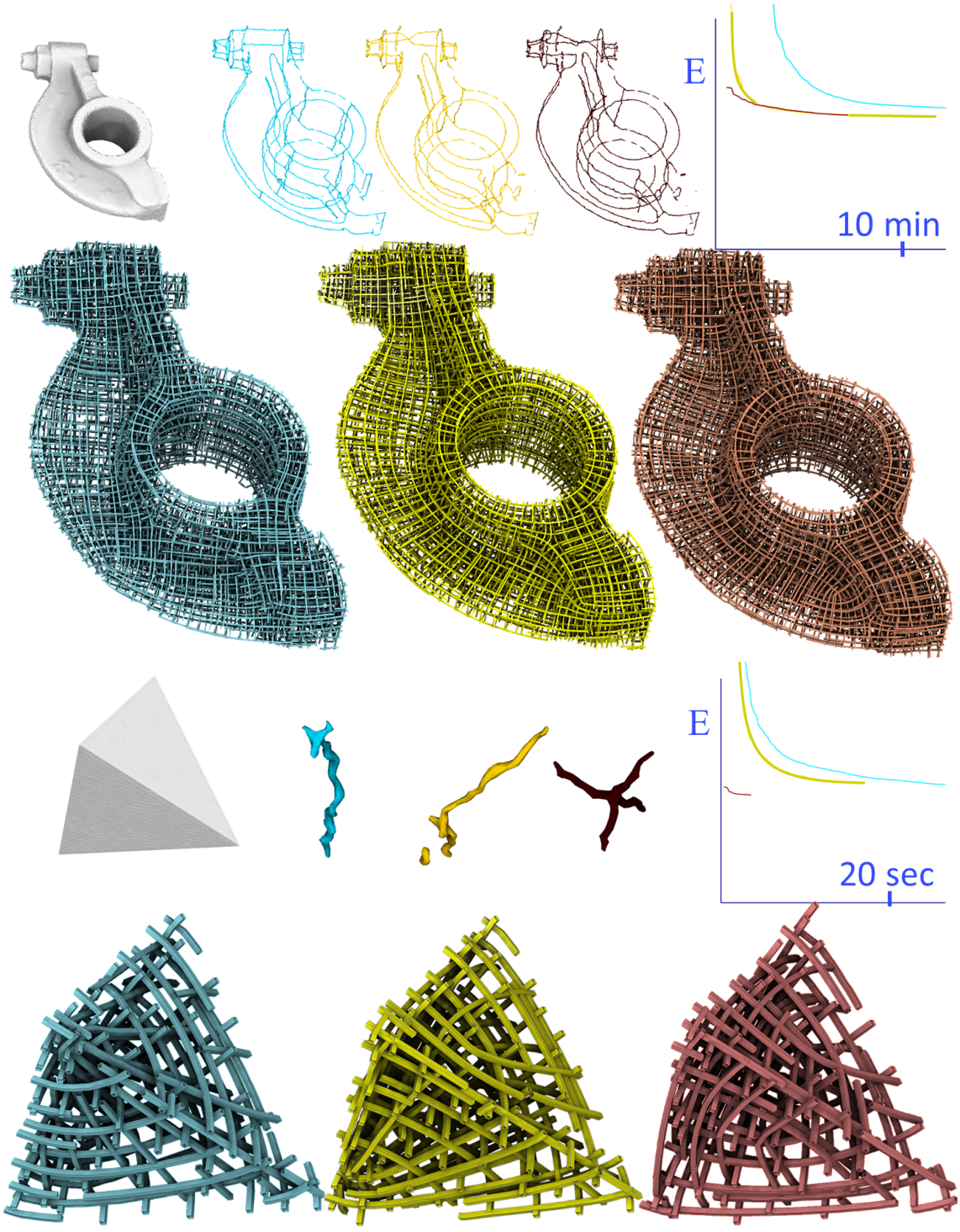


Figure 4.16: Comparison of initialization of Huang's algorithm with their weight of boundary penalty term (10^{-2}) (left/blue), with a much higher weight (10^2) (middle/yellow), and our initialization (right/red). All smoothing iterations are performed by Huang's algorithm. For the rocker model, the energies (we take the run with our initialization for 100%) are respectively $94,7\%$, $101,6\%$ and (obviously) 100%. For the tet model, we obtain 91%, $89,7\%$ and 100%.

The only real failure case we found using their method is due to the front propagation algorithm: when a boundary frame is copied to a large number on inner samples. In this case, the L-BFGS solver is sometimes locked with a bad topology (see Fig. 4.19).

On more complex examples, we have compared their algorithm against our initialization followed by their smoothing iterations. In most cases, we obtain an energy that is a bit better (Fig 4.20). We also observed that their topology often differs from ours (Fig 4.21), so we conjecture that our topology is somehow better. However, the quality of the field topology depends on the application, and is not well evaluated by the energy, even for topologies very far from being optimal (see e.g. Fig 4.16).

Comparison of smoothing iterations In the previous sections we have shown (Fig. 4.16 and 4.19) that our method provides the best initialization, however our smoothing iterations are not clearly better than others.

Figure 4.22 shows a comparison of three different smoothing strategies (our linearization, L-BFGS proposed by Huang *et al.* and L-BFGS proposed by Li *et al.*). In all three cases we use our method to initialize the field. L-BFGS smoothing proposed by Huang *et al.* is the slowest in all test cases. First of all, in our implementation the time to evaluate the energy and the gradient is four times slower for the method by Huang *et al.* than for the method by Li *et al.* Second, the crucial difference between these two methods is the way to enforce the boundary alignment: Huang *et al.* use a penalty term, whereas Li *et al.* use the set of variables directly satisfying the boundary constraints. In our test we noticed that usage of penalty terms increases the number of iterations to converge and interferes with topological choices to be made, leading to inferior final fields.

We also noticed that the behavior of linearization changes in function of how far the initialization is from the final solution.

- First row of figure 4.22 shows a simple case without topology changes, the smoothing iterations change the field geometry only. In this case the linearization of feasibility constraints works flawlessly, in two iterations the method converges, taking about the same time as the smoothing by Li *et al.*
- Middle row shows a second case, where few topology changes must be made. It slowes the linearization down, even if two iterations produce a reasonably good field.
- Finally, the bottom row shows the case where the initial topology is really bad. The linearization method fails on this model: two first iterations are still very far from the final solution and to reach the minimum it requires four times more time than the method by Li *et al.*

We conclude that the best solution is our initialization followed by the optimization of Li *et al.*. In practice, the implementation of our linearization smoothing iterations is almost free (incremental with respect to our initialization algorithm), whereas Li *et al.* smoothing algorithm is more difficult to implement. Moreover, in most cases few iterations of linearization steps suffice to obtain a fairly good field. As a consequence, we suggest starting with our smoothing iterations (almost free to implement), then possibly replace it with Li *et al.* smoothing algorithm if performances are not sufficient.

Conclusion

This section unifies the frame field design problem in $2D$ and $3D$. Both problems are formulated with a similar representation of frames, constraints and objective function. As a consequence, they can also be solved by similar algorithms.

From this analysis, we discovered that the best actual solution to produce smooth $3D$ frame fields is to initialize it by our proposed extension of [KLF12], followed by smoothing iterations of [LLX$^+$12]. The main drawback of this solution is requirement to implement two very different approaches (a sparse linear system solver and a L-BFGS descent). A fair alternative is to use our extension of [KLF12], it is simple to implement and requires a linear system solver only. In practice for our models we perform only two or three linearization iterations, however if the initialization is a bad guess (e.g. the sphere), it can be insufficient. With this approach we are able to generate (on a laptop) fields on the models up to few millions tetrahedra in less than 10 minutes, refer to Figure 4.23 for an illustration.

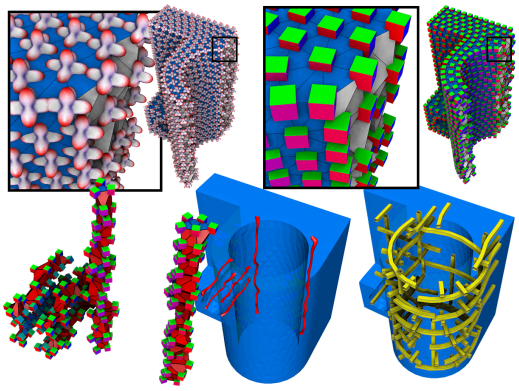


Figure 4.17: Our results are shown using combinations of the following rendering techniques. We can plot for each vertex i its F^i (upper left), or its frame as a cube(upper right). We can show the singular tets (lower left), or a smoothed and refined version (lower middle) to better see it in $3D$ (thanks to the lighting). The field inside the volume can be rendered by curved french fries.

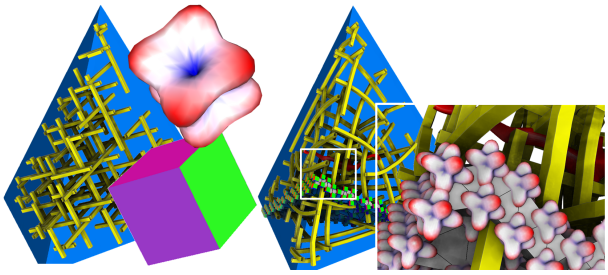


Figure 4.18: The initialization of [HTWB11] (left) is a constant frame field whereas ours (right) is aligned to the boundary. Their F^i are all equal, and very far from being feasible, making it possible to violate the boundary condition.

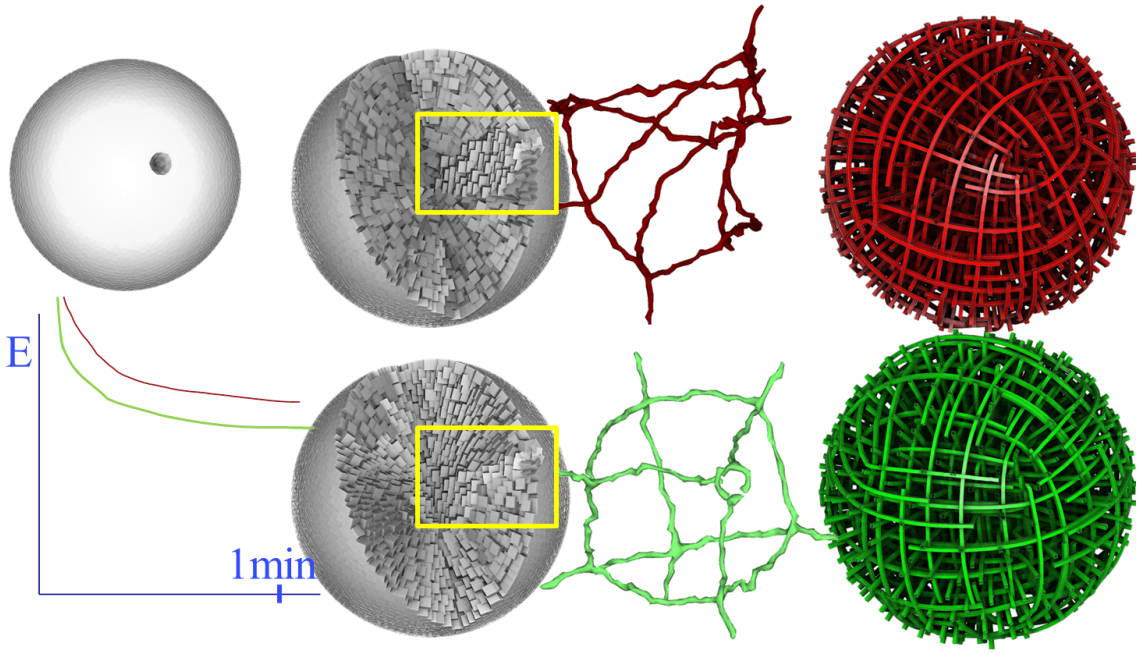


Figure 4.19: Li's algorithm (red) is compared to our algorithm (green) on a one-finger bowling ball. The hole has a huge impact on the initialization due to the advancing front approach (second column, inside the yellow box). As a result, their initialization provides a field with a poor topology and smoothing iterations are not sufficient to find the expected topology (like ours). Our final energy is 86% of theirs.

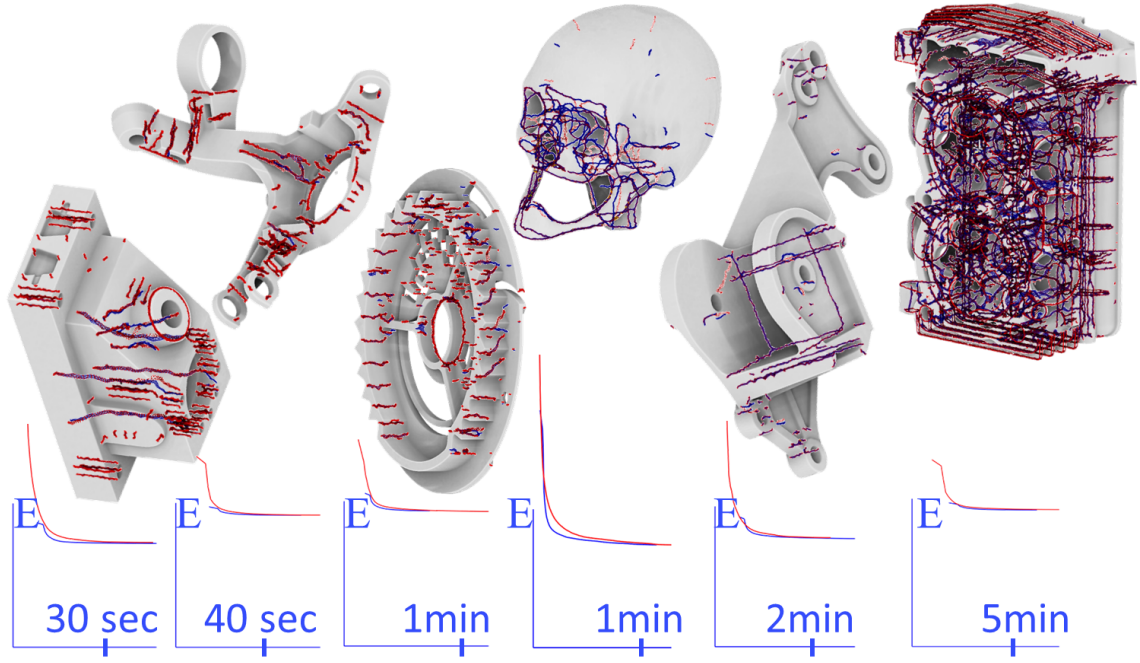


Figure 4.20: Comparison of two fields generated by Li *et al.* smoothing iterations: using their initialization (red) or ours (blue). Our energy in percent of theirs is 99%, 99.87%, 100.5%, 99.97%, 99%, 99.6%. The difference is always very low ($< 1\%$) but always in our advantage except for the third model.

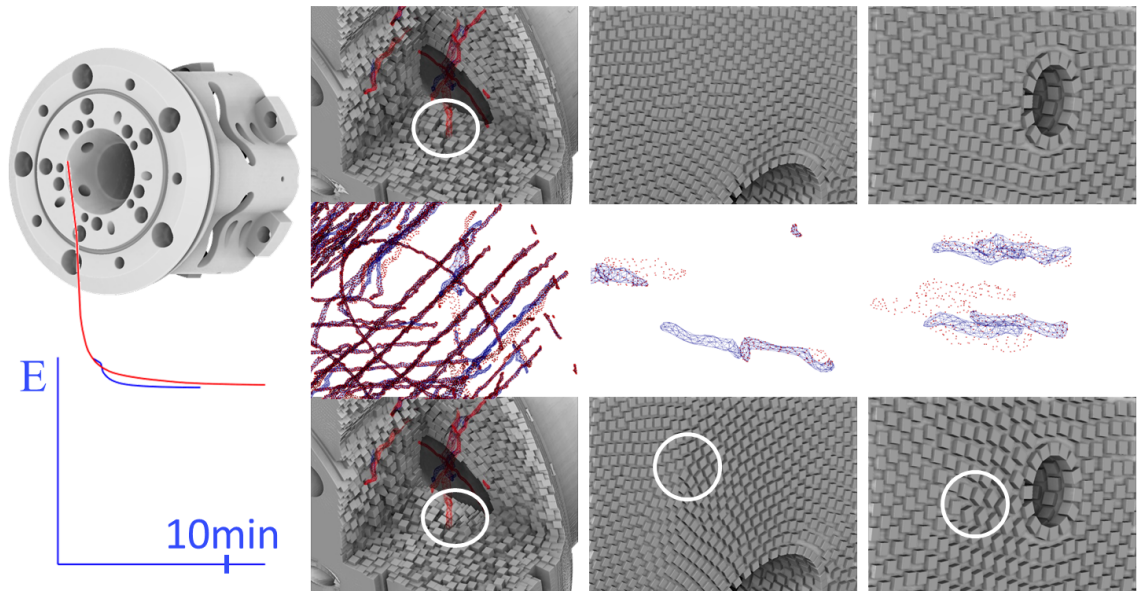


Figure 4.21: Comparison of two fields generated by Li's smoothing iterations: using their initialization (red) or ours (blue). Our energy in percent of theirs is 98.5%. Close-ups show the field where the singularity graphs diverge: one is inside the volume (leftmost) and others are on the object boundary. Our results are on the top row and theirs are on the bottom row with singularity encircled in white.

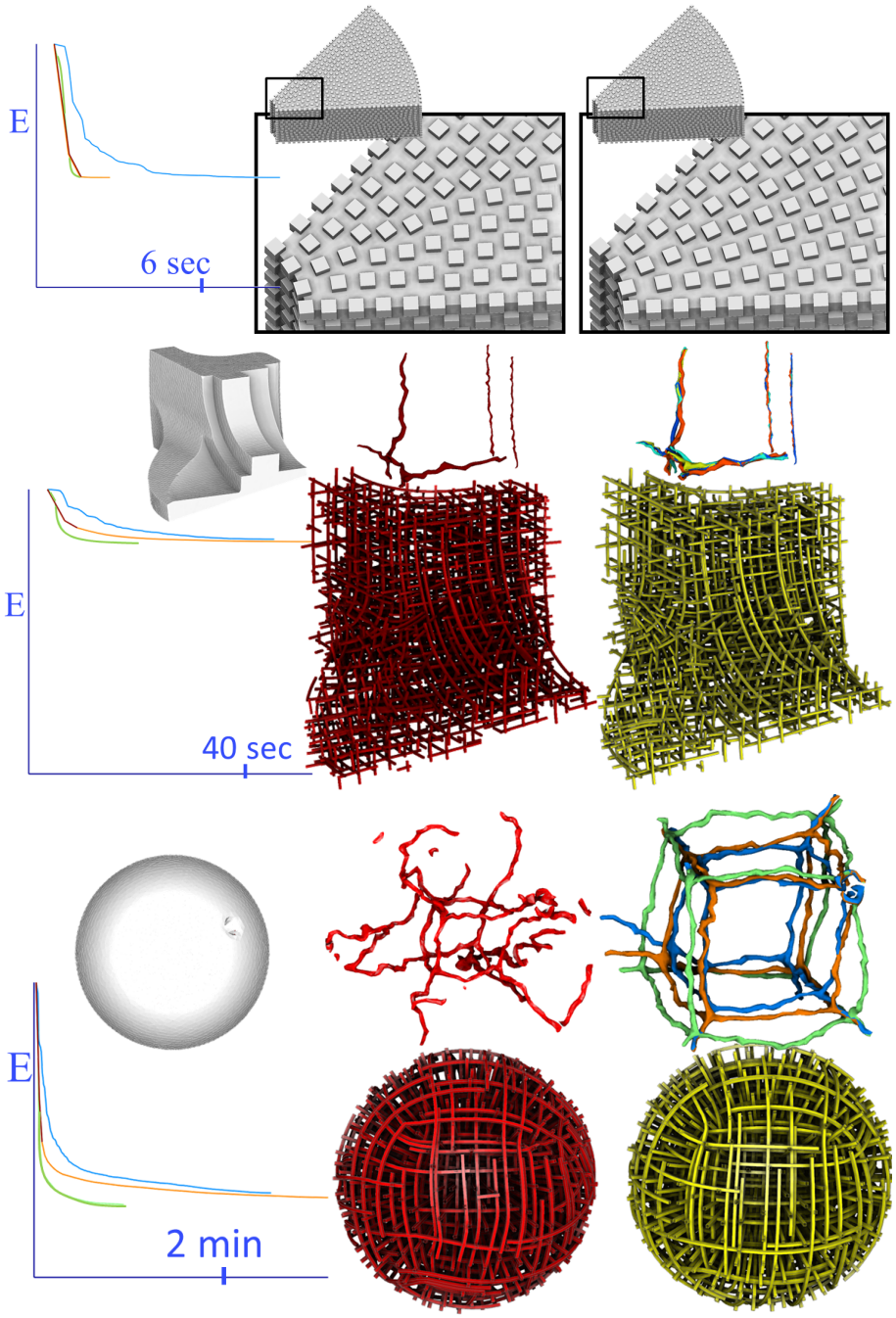


Figure 4.22: Comparison of smoothing iteration algorithms combined with our initialization algorithm. We compare Huang's method (blue), Li's method (green), our method (orange), and our method limited to two iterations (red). Top row compares the initialization (left) with the field after two iterations of our algorithm (right). Middle and bottom rows compare the field with two iterations of our smoothing algorithm with other smoothing strategies. Singularity graphs reflect nicely the convergence of thee algorithms. We obtain energy (we take Li's result for 100%) of resp. 99.98%, 100%, 99.98% and 100.5% on the sector, 101.5%, 100%, 100.4%, and 106.5% on the fan disk, and 116%, 100%, 109%, 185% on the one-finger bowling ball.

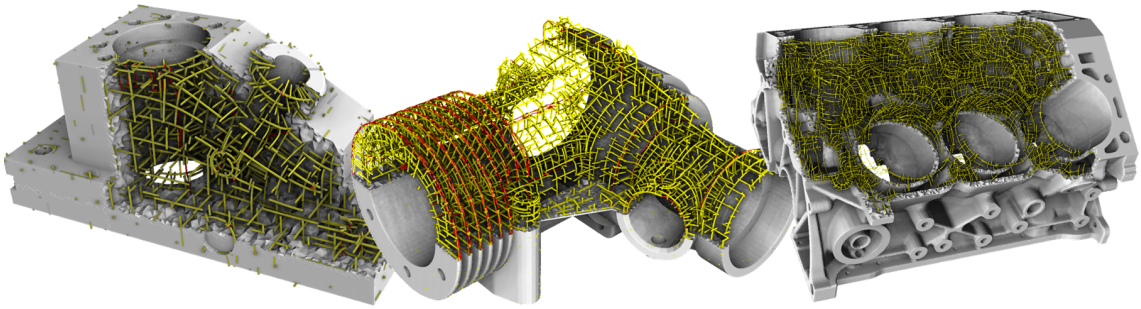


Figure 4.23: Results on CAD objects. Names are (from left to right): Anc, Crank, 40head.

4.2 Integrating a 3D frame field: periodic global parameterization

Notations: throughout this section we use bold font to denote vectors. Subscripts are used to index variables. For example, the vertex number i of a tetrahedral mesh will be denoted by $\mathbf{x_i}$. Square brackets are used to access to individual components of vectors or matrices. For instance, $\mathbf{x_i}[0]$ denotes the first component of a three-dimensional vector $\mathbf{x_i}$. Given a matrix R, its first row is denoted by $R[0,\cdot]$ and its first column is denoted by $R[\cdot,0]$.

- **The input of our algorithm** is a tetrahedral mesh T representing an object to be remeshed, as well as the desired sizes and orientations of the hexahedral elements. The desired size and orientation at each vertex i is represented by a matrix B_i such that its columns $B_i[\cdot,k]$, $k=0,1,2$ form an orthogonal basis that corresponds to the edges of the typical hexahedral elements to be created in the vicinity of vertex i (Fig. 4.24);
- **the output of our algorithm** is a set of points P meant to be the vertices of a hexahedral-dominant mesh where the orientation and size of hexahedra is as close as possible to the one defined by the matrices B_i.

Our algorithm consists in the following three steps (an optional step and two mandatory ones):

1. *Optional step:* optimize the size of the hexahedra to reduce the number of singularities (curl-correction). The curl-corrected frame field is obtained as the solution of a sparse linear system (§4.2.4);
2. **Parameterize the object** T by solving a sparse linear system (§4.2.2).
3. **Back-project points with integer coordinates** from the parameter space onto the object T (§4.2.3)

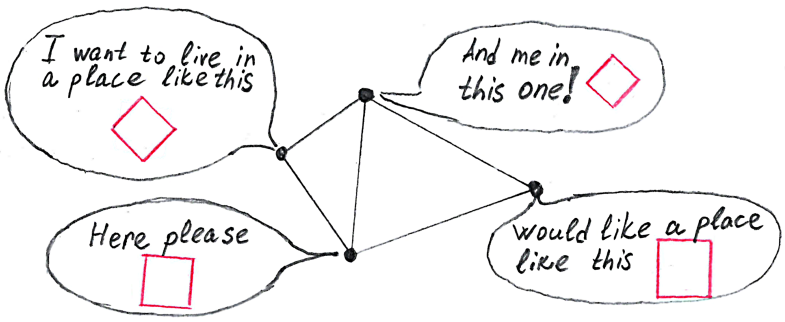


Figure 4.24: Each vertex of the input mesh knows the size and the orientation of the hexahedron to be created in his vicinity. We store this information as an orthogonal basis matrix B_i per vertex i.

The following subsection details the method that generates the volumetric parameterization. For the sake of clarity, the accompanying illustrations are in $2D$.

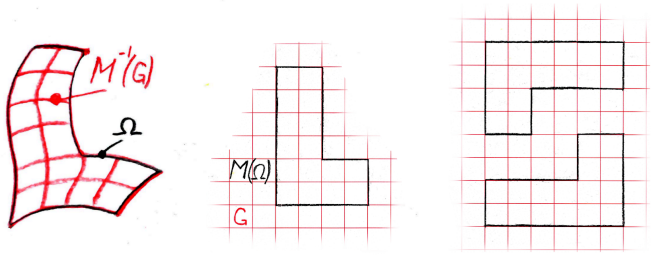


Figure 4.25: Considering a continuous map M from an object Ω (left) to $\mathbb{R}^3$ (center), it is possible to generate a hexahedral mesh of Ω by tracing the preimage of the regular grid from $\mathbb{R}^3$ (center) to the object. Note that a whole family of maps – that we call "grid-equivalent" – produces exactly the same hexahedral mesh (right).

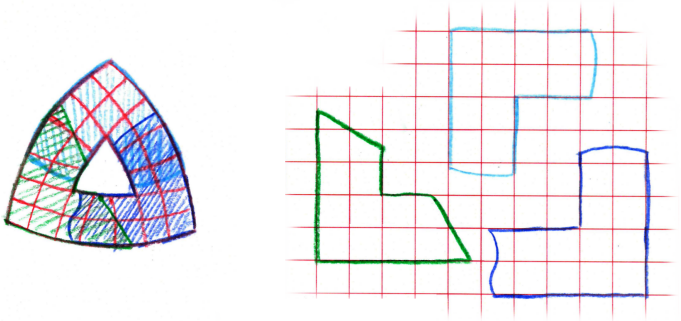


Figure 4.26: The object shown on the left can be remeshed with the atlas of grid-equivalent maps shown on the right. The preimage of G through the three maps is unique and defines a hexahedral mesh.

4.2.1 Problem statement

We suppose that we have a 3D domain Ω and a (global) continuous map $M : \Omega \rightarrow \mathbb{R}^3$. Let G denote a regular 3D grid: $G = \{(x, y, z) \in \mathbb{R}^3 : (x \mod 1 = 0)$ or $(y \mod 1 = 0)$ or $(z \mod 1 = 0)\}$. The pre-image $M^{-1}(G)$ of G defines the facets of a hexahedral mesh for Ω (see Fig. 4.25). In this configuration, $\mathbb{R}^3$ may be thought of as a "texture space", and then the hexahedral mesh of Ω is obtained by using a rectilinear grid as a "texture".

Note that for a given map M and its induced hexahedral mesh $M^{-1}(G)$, it is possible to find another map M' that generates exactly the same mesh (Fig. 4.25-right). We now formalize such pair of maps that produce the same grid :

Definition 2. *Two maps M and M' are said to be grid-equivalent if and only if there exists $R \in \mathbf{R}$ and $\mathbf{t} \in \mathbf{T}$ such that $M = RM' + \mathbf{t}$, where:*

- $\mathbf{R}$ *is the set of 24 rotation matrices that permute the normal vectors of the unit cube* $(1, 0, 0)$, $(0, 1, 0)$, $(0, 0, 1)$, $(-1, 0, 0)$, $(0, -1, 0)$ *and* $(0, 0, -1)$*;*
- $\mathbf{T}$ *is $\mathbb{Z}^3$, the set of vectors with integer coordinates.*

A regular grid G is invariant under the action of the 24 rotations and integer translations: $R \in \mathbf{R}, \mathbf{t} \in \mathbf{T} \Rightarrow G = RG + \mathbf{t}$. Therefore, the preimages of two grid-equivalent maps M and M' correspond to the same hexahedral mesh $M^{-1}(G) = M'^{-1}(G)$ (Fig. 4.25-right).

In general, a single global continuous map is not sufficient to create a boundary-aligned hexahedral mesh. For example, it is impossible to remesh in this way the object shown on the left of Fig. 4.26. Thus we will search instead for an *atlas* of local maps. If any pair of maps within the atlas is grid-equivalent on the intersection of their domain, then the final remesh is still unique (see Fig. 4.26).

In our setting, the domain Ω is the tetrahedral mesh T. We associate to each vertex i a local map M_i, defined on the tetrahedra incident to i, and linear in each tetrahedron.

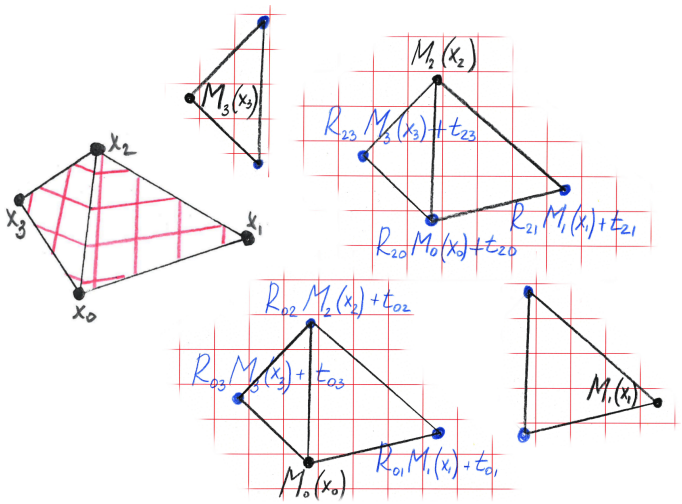


Figure 4.27: The maps M_i for the example shown in Fig. 4.24. Each map M_i is represented by the texture coordinates of the vertices connected to i with an edge. To ensure the grid compatibility of the maps, we constrain the position of the blue vertices: they are obtained from the black ones by an integer translation $\mathbf{t_{ij}}$ (and possibly a rotation).

We formulate our problem as an energy minimization (defined later in this section). We represent each map M_i by texture coordinates of the vertex star of the vertex i. The problem is a minimization under constraints of grid equivalence.

Recall that for two adjacent vertices i and j the maps M_i and M_j are grid-equivalent iff there exists $R \in \mathbf{R}$ and $\mathbf{t} \in \mathbf{T}$ such that $M_i = RM_j + \mathbf{t}$. As our maps are linear (per tetrahedron), it is sufficient to enforce this equality at 4 points on each tetrahedron: the maps M_i and M_j are grid-equivalent iff we can find $R_{ij} \in \mathbf{R}$ and $\mathbf{t_{ij}} \in \mathbf{T}$ such that on each tetrahedron (i, j, k, l) we have :

$$\begin{cases} M_i(\mathbf{x_i}) = R_{ij}M_j(\mathbf{x_i}) + \mathbf{t_{ij}} & (1) \\ M_i(\mathbf{x_j}) = R_{ij}M_j(\mathbf{x_j}) + \mathbf{t_{ij}} & (2) \\ M_i(\mathbf{x_k}) = R_{ij}M_j(\mathbf{x_k}) + \mathbf{t_{ij}} & (3) \\ M_i(\mathbf{x_l}) = R_{ij}M_j(\mathbf{x_l}) + \mathbf{t_{ij}} & (4) \end{cases} \tag{4.9}$$

Thus the problem consists in finding $M_i(\mathbf{x_i})$ and $M_i(\mathbf{x_j})$ for each oriented edge i, j, under the constraint of grid-equivalence Eqn (4.9). It is difficult to find an expression of the grid-equivalence constraint suitable to numerical optimization (i.e. existence of rotations and translations). Therefore, we prefer to introduce the rotations and translations $(R_{ij}, \mathbf{t}_i)$ into the set of variables, together with constraints that connect the texture coordinates on neighboring maps. However, if we directly introduce unknowns R_{ij} and $\mathbf{t_{ij}}$ for each oriented edge ij, it creates much redundancy in the variables. Indeed, from a texture coordinate $M_i(\mathbf{x_i})$, rotation R_{ij} and translation $\mathbf{t_{ij}}$ we can deduce the value of $M_j(\mathbf{x_i})$, because our system must satisfy the grid-equivalence constraint (4.9). Therefore, we can remove $M_j(\mathbf{x_i})$ (as well as $M_i(\mathbf{x_j})$) from the set of variables. Finally, each oriented edge ij involves $M_i(\mathbf{x_i})$, R_{ij} and $\mathbf{t_{ij}}$ as variables (refer to Fig. 4.27 for an illustration). Note that with this particular choice of variables, lines 1 and 2 of Equation 4.9 are naturally satisfied.

We formulate the problem as a least squares problem with a set of equations per oriented edge ij. These equations express the geometric objective, i.e. make the Jacobian matrix $J(M_i)$ as close as possible to ${B_i}^{-1}$. From now on we note $M_i(\mathbf{x_i})$ as $\mathbf{u_i}$. Thus, each oriented edge ij introduces a term $\mathbf{u_i} + {B_i}^{-1}(\mathbf{x_j} - \mathbf{x_i}) = R_{ij}\mathbf{u_j} + \mathbf{t_{ij}}$ in the objective function.

Remark 1. *This formulation does not ensure that lines 3 and 4 of Equation 4.9 are satisfied. A configuration where they are not satisfied corresponds to a singularity of the frame field or a singularity of the parameterization (more on this in the next section).*

Remark 2. *This formulation does not ensure the positivity of the Jacobian determinant* $\det(J(M_i)) > 0$*, so the trivariate functions M_i may not provide a valid mapping. Following our definition, two maps can be grid-equivalent even if they are not valid. However in practice, most M_i will be valid*

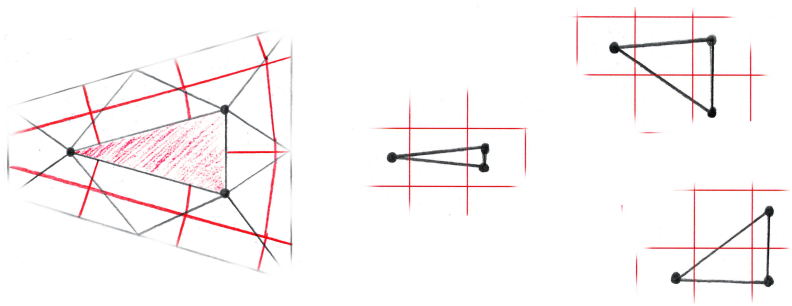


Figure 4.28: A PGP-singularity corresponds to a T-junction in the generated mesh. Left image: the T-junction. Right images: three (non grid-equivalent) maps for the (grayed-out) singular tetrahedron.

maps at the end. Invalid ones generate singular tetrahedra (see §4.2.2), treated as explained in §4.2.3.

4.2.2 Optimization

We first solve for the matrices R_{ij}

We have $J(M_i) = R_{ij}J(M_j)$ and we want $J(M_i) \approx B_i$, thus we define R_{ij} as :

$$R_{ij} = \underset{R \in \mathbf{R}}{\arg\min} \|R - {B_i}^{-1} B_j\|$$

.

We can now replace $\mathbf{t_{ij}}$ by $R_{ij}\mathbf{t_{ji}}$ for $i < j$

Let us regroup the equations: for each oriented edge $i < j$ we have two equations:

$$\begin{cases} \mathbf{u_i} + {B_i}^{-1}(\mathbf{x_j} - \mathbf{x_i}) = R_{ij}\mathbf{u_j} + \mathbf{t_{ij}} \\ \mathbf{u_j} + {B_j}^{-1}(\mathbf{x_i} - \mathbf{x_j}) = R_{ji}\mathbf{u_i} + \mathbf{t_{ji}} \end{cases}$$

Then we multiply the second equation by $-R_{ij}$ and regroup the terms:

$$\begin{cases} \mathbf{u_i} - R_{ij}\mathbf{u_j} - \mathbf{t_{ij}} & -{B_i}^{-1}(\mathbf{x_i} - \mathbf{x_j}) & = \mathbf{0} \\ \mathbf{u_i} - R_{ij}\mathbf{u_j} - \mathbf{t_{ij}} & -R_{ij}{B_j}^{-1}(\mathbf{x_i} - \mathbf{x_j}) & = \mathbf{0} \end{cases}$$

Note that if a linear system is composed of equations of type $x - a = 0$ and $x - b = 0$, then solving it *in the least squares sense* is equivalent to solving the system of equations $x-(a+b)/2 = 0$.

It leads to the new formulation

We now have a set of equations (one per oriented edge $i < j$) to be solved in the least squares sense:

$$\forall i < j, \quad \mathbf{u_i} - R_{ij}\mathbf{u_j} - \mathbf{t_{ij}} + \mathbf{g_{ij}} = \mathbf{0} \tag{4.10}$$

where the variables are $\mathbf{u_i}$, $\mathbf{u_j}$ and $\mathbf{t_{ij}}$ and where the input constants $\mathbf{g}_{ij}$ are given as follows:

$$\mathbf{g_{ij}} = ((B_i^{-1} + R_{ij}{B_j}^{-1})/2)(\mathbf{x_j} - \mathbf{x_i}) \tag{4.11}$$

$$R_{ij} = \underset{R \in \mathbf{R}}{\arg\min} \|R - {B_i}^{-1} B_j\| \tag{4.12}$$

As each $\mathbf{t_{ij}}$ appears exactly in one line of the system (4.10), it is sufficient to consider the *fractional* part of each equation:

$$\forall i < j, \quad \mathbf{u_i} + \mathbf{g_{ij}} = R_{ij}\mathbf{u_j} \mod 1$$

We split this vector equation into six scalar ones and use the periodicity of the cosine and sine functions to remove the modulo:

$$\begin{array}{ll} \forall i<j, & \left\{ \begin{array}{lcl} \cos(2\pi(\mathbf{u_i}+\mathbf{g_{ij}})[d]) & = & \cos(2\pi(R_{ij}\mathbf{u_j})[d]) \\ \sin(2\pi(\mathbf{u_i}+\mathbf{g_{ij}})[d]) & = & \sin(2\pi(R_{ij}\mathbf{u_j})[d]) \end{array} \right. \\ \forall d\in\{0,1,2\}, & \end{array}$$

By expanding the cosine and sine of a sum, one obtains :

$$\begin{array}{lll} \forall i<j, \forall d\in\{0,1,2\}, & & \\ \cos(2\pi\mathbf{u_i}[d])\cos(2\pi\mathbf{g_{ij}}[d]) & -\sin(2\pi\mathbf{u_i}[d])\sin(2\pi\mathbf{g_{ij}}[d]) & \\ & -\cos(2\pi(R_{ij}\mathbf{u_j})[d]) & =0 \\ \sin(2\pi\mathbf{u_i}[d])\cos(2\pi\mathbf{g_{ij}}[d]) & +\cos(2\pi\mathbf{u_i}[d])\sin(2\pi\mathbf{g_{ij}}[d]) & \\ & -\sin(2\pi(R_{ij}\mathbf{u_j})[d]) & =0 \end{array}$$

Then we perform a change of variables

Each scalar variable $\mathbf{u_i}[d]$ is replaced with two variables $\mathbf{a_i}[d]$ and $\mathbf{b_i}[d]$ that correspond to its cosine and sine, respectively:

$$\mathbf{a_i}[d] \stackrel{\text{def}}{=} \cos(2\pi\mathbf{u_i}[d]) \qquad \mathbf{b_i}[d] \stackrel{\text{def}}{=} \sin(2\pi\mathbf{u_i}[d]).$$

Note that as $R_{ij}\in\mathbf{R}$ is one of 24 rotation matrices, for any d the corresponding row $R_{ij}[d,\cdot]$ contains two zeroes and one 1 or -1. Let us define r_{ij}^d to be the index of the non-zero entry in the row $R_{ij}[d,\cdot]$, and s_{ij}^d its sign. We can write $(R_{ij}\mathbf{u_j})[d] = R_{ij}[d,\cdot]\mathbf{u_j} = s_{ij}^d\mathbf{u_j}[r_{ij}^d]$. Then we solve the following linear system in the least squares sense:

$$\begin{array}{lll} \mathbf{a_i}[d]\cos(2\pi\mathbf{g_{ij}}[d]) - \mathbf{b_i}[d]\sin(2\pi\mathbf{g_{ij}}[d]) & - & \mathbf{a_j}[r_{ij}^d] = 0 \\ \mathbf{b_i}[d]\cos(2\pi\mathbf{g_{ij}}[d]) + \mathbf{a_i}[d]\sin(2\pi\mathbf{g_{ij}}[d]) & -s_{ij}^d & \mathbf{b_j}[r_{ij}^d] = 0 \end{array} \tag{4.13}$$

To ensure that points are generated on the boundary of the volume, (at least) one component of the parameterization has to be zero there. Therefore, for each tetrahedron vertex on the boundary, we find the vector of the input frame field (column of B_i) aligned with the normal to the boundary and constrain the corresponding variables $\mathbf{a_i}[d]$ and $\mathbf{b_i}[d]$ to be equal to 1 and 0 respectively.

To solve the linear system, we use the implementation of the Jacobi preconditioned Conjugate Gradient algorithm [AMS90], available in [Lé]. We then retrieve the original variables as $\mathbf{u_i}[d] \leftarrow \text{atan2}(\mathbf{b_i}[d], \mathbf{a_i}[d])$. Once we have all $\mathbf{u_i}$, corresponding $\mathbf{t_{ij}}$ are straightforward to compute.

Singularities

As previously mentioned, our choice of variables satisfies the first two equations of the system (4.9). However, the last two equations can be violated. In this case, the tetrahedron is said to be *singular*. There are two types of singularities :

Definition 3. *FF-Singularity (Frame-Field):*

- *The triangle ijk is said to be FF-singular if $R_{ij}R_{jk}R_{ki} \neq Id_{3\times3}$;*
- *A tetrahedron is FF-singular if any of its faces is FF-singular.*

Definition 4. *PGP-Singularity (Periodic Global Param.):*

- *The triangle ijk is said to be PGP-singular if $\mathbf{t_{ij}} + R_{ij}\mathbf{t_{jk}} \neq \mathbf{t_{ik}}$;*
- *A tetrahedron is PGP-singular if any of its faces is PGP-singular.*

Note that the variables $\mathbf{u_i}$ do not appear in this criterion.

Frame-Field singularities create degenerate maps, since the FF-singularity condition enforces $\mathbf{t_{ij}} = \mathbf{0}$ and $\mathbf{u_i} = \mathbf{0}$. PGP singularities yield T-junctions in the generated hexahedral mesh (Fig. 4.28).

4.2.3 Extracting gridpoints

Once the vectors $\mathbf{u_i}$ and $\mathbf{t_{ij}}$ are computed, it is easy to extract the gridpoints, as follows : For each tetrahedron $ijkl$ we choose an arbitrary map among M_i, M_j, M_k, M_l and then we extract the pre-images of the points with integer texture coordinates inside the image of the tetrahedron. There are two cases: either the tetrahedron is non-singular and then the result does not depend on the choice of the map, or it is singular and then the result can depend on the map. If the tetrahedron is singular, to ensure a sufficient sampling density, we select among M_i, M_j, M_k, M_l the one that maximizes the volume of the mapped tetrahedron. In general, this generates points (and then Delaunay tetrahedra) that are not recombined into hexahedra in the subsequent step.

4.2.4 Optional pre-processing step: curl correction

Curl correction[6] is an optional step that pre-processes the "frame" field (B_i) that steers the optimization in Section 4.2.2. In a nutshell, it adds a corrective term $\mathbf{c}_{ij}$ to the input $\mathbf{g}_{ij}$ as $\mathbf{g_{ij}} \leftarrow \mathbf{g_{ij}} + \mathbf{c_{ij}}$, in such a way that our optimization algorithm produces a smaller number of singularities. It basically trades a higher proportion of grid-compatible maps for a larger distance to the geometric objective $J(M_i) = B_i^{-1}$.

We first define the constraints on $\mathbf{c_{ij}}$ that enforce grid-compatible maps §4.2.4, then we derive an algorithm to compute values of $\mathbf{c_{ij}}$ that prevent degeneracies and that limit the distortion §4.2.4.

Curl-correction constraints on $\mathbf{c_{ij}}$

Two maps M_i and M_j are grid compatible iff each tetrahedron incident to the vertices i and j satisfies Equation 4.9. This means that for each triangle ijk, we have $M_i(\mathbf{x_k}) = R_{ij}M_j(\mathbf{x_k}) + \mathbf{t_{ij}}$. This later expression can be expressed with our set of variables $\mathbf{u_i}$ as :

$$R_{ik}\mathbf{u_k} - R_{ij}R_{jk}\mathbf{u_k} = R_{ij}\mathbf{t_{jk}} + \mathbf{t_{ij}} - \mathbf{t_{ik}}$$

Assuming that the geometric objective is perfectly respected (Equation 4.10), we can write :

$$\begin{aligned} R_{ik}\mathbf{u_k} - R_{ij}R_{jk}\mathbf{u_k} &= R_{ij}(\mathbf{u_j} - R_{jk}\mathbf{u_k} + \mathbf{g_{jk}}) && (4.14)\\ &+ \mathbf{u_i} - R_{ij}\mathbf{u_j} + \mathbf{g_{ij}} && (4.15)\\ &- (\mathbf{u_i} - R_{jk}\mathbf{u_k} + \mathbf{g_{ik}}) && (4.16) \end{aligned}$$

If the triangle ijk is not a singularity of the frame field, we have $R_{ij}R_{jk}R_{ki} = Id_{3\times3}$ so the condition simplifies to: $R_{ij}\mathbf{g_{jk}} + \mathbf{g_{ij}} - \mathbf{g_{ik}} = \mathbf{0}$. As a consequence, we can assert the following:

Remark 3. *If the $\mathbf{c_{ij}}$'s are such that $R_{ij}(\mathbf{g_{jk}} + \mathbf{c_{jk}}) + (\mathbf{g_{ij}} + \mathbf{c_{ij}}) - (\mathbf{g_{ik}} + \mathbf{c_{ik}}) = \mathbf{0}$, then there is a trivial solution to our optimization process (without boundary condition), with zero energy, and that produces only grid compatible maps.*

It gives us a simple sufficient condition on the $\mathbf{c_{ij}}$'s to produce grid-equivalent maps.

To have hexahedra faces located on the surface boundary, the image of boundary faces and edges of T are constrained to live in an iso-value of a coordinate of the map. This boundary condition is expressed as the following set of constraints :

$$(B_i + R_{ij}B_j)(\mathbf{g_{ij}} + \mathbf{c_{ij}}) \cdot \mathbf{n} = 0$$

where $\mathbf{n}$ is the normal of the surface on edge ij located on the boundary of the surface.

Curl-Correction Algorithm

We first search for the corrective term $\mathbf{c_{ij}}$ of minimal norm that respects the constraints, in other words we solve the following constrained optimization problem:

[6] In DEC language, if $R_{ij} = Id_{3\times3}$, this step is a modification of the trivariate 1-form $\mathbf{g}$ to make it closed, i.e. canceling its curl.

$$\min \Sigma_{ij} \|\mathbf{c_{ij}}\|^2$$
$$\text{s.t.} \begin{cases} \text{For all non-singular triangles } ijk, \\ \quad R_{ij}(\mathbf{g_{jk}} + \mathbf{c_{jk}}) + (\mathbf{g_{ij}} + \mathbf{c_{ij}}) - (\mathbf{g_{ik}} + \mathbf{c_{ik}}) = \mathbf{0} \\ \text{For all edges on the border } ij, \\ \quad (B_i + R_{ij}B_j)(\mathbf{g_{ij}} + \mathbf{c_{ij}}) \cdot \mathbf{n} = 0 \end{cases}$$

The obtained $\mathbf{c_{ij}}$ ensures that Equation 4.9 is respected everywhere. However, it can produce highly distorted (and probably not injective) maps. To avoid this, we scale the result by :

$$\mathbf{c_{ij}}[d] \leftarrow \min\left(1, \frac{\gamma\|\mathbf{g_{ij}}\|}{\|\mathbf{c_{ij}}\|}\right)\mathbf{c_{ij}}[d]$$

The parameter γ sets the minimum ratio between the corrected desired scale and the original one. The impact of γ is discussed in §4.4.2, and all other results are obtained with the default setting $\gamma \leftarrow 0.35$.

4.3 Generating the hexahedral-dominant mesh

The algorithm described in the previous section §4.2.3 generates a set of points P. The points in P are well spaced and mostly organized on a warped regular grid (except on the singularities). In addition, P samples the border ∂T of the input tetrahedral mesh and fills its interior. Our method to generate a hexahedral-dominant mesh from P can be summarized as follows (more details further):

- **Step 1:** re-mesh of the border of the domain ∂T using as vertices the points of P that are located on the border of T (§4.3.1);
- **Step 2:** compute an intermediate tetrahedral mesh T' using the points P constrained by the re-meshed border. One can use an off-the-shelf constrained Delaunay implementation, such as MGTetra [GHS90] or tetgen [Si15];
- **Step 3:** recombine in T' the sets of tetrahedra that can be assembled to form a hexahedron. Optionally, if supported by the application that uses the mesh, prisms and pyramids can be generated as well. We use a refinement of the algorithm proposed by Meshkat and Talmor in [MT00] §4.3.2.

4.3.1 Re-meshing the border of the domain

This section explains how to re-mesh the border ∂T of the input tetrahedral mesh T to generate a new triangulated surface with its vertices in the point set P generated at the previous step (plus some additional vertices, as explained later). We start by computing $\text{Del}(P)|_{\partial T}$, the Delaunay triangulation of P restricted to the border of the domain ∂T. In other words, this means computing the intersection between the Voronoi diagram of P and ∂T (thin yellow lines in Fig. 4.29 top-left). Each time three Voronoi cells meet (i.e. when a Voronoi edge has a non-empty intersection with ∂T), then the corresponding three points are connected with a triangle (bottom-left). We use the algorithm described in [YLL$^+$09] implemented in [Lé15] with arithmetic filters [MP08], exact arithmetics using expansions [She97] and symbolic perturbations [EM90]. Such perturbations are useful to disambiguate triangle connections whenever four Voronoi cells meet (instead of three in the generic case). In our specific case, such degenerate configurations occur very often since the points that we generate are aligned on a regular grid.

Note that the sampling of ∂T realized by P is not always sufficiently dense to capture all the geometric features of ∂T (see Fig. 4.29 bottom-left). For this reason, we iteratively insert points in the regions of largest geometric error (red points on Fig. 4.29) until the Hausdorff distance between ∂T and the re-meshed border is smaller than a user-defined threshold ε. The refinement algorithm is detailed in Algorithm 8.

The algorithm iteratively adds batches of points B. To facilitate reproducing our results, we further detail some parts of the algorithm. Line 2: the 'sample_surface' function generates a regular sampling for each triangle of ∂T in such a way that the maximal distance between two samples is smaller than ε. Line 6: the function 'distance_to_surface' gives the minimal distance between a point and a triangulated surface, implemented using an AABB-tree. Line 11: the constant ϵ' specifies the minimal distance between points inserted in the same batch B (see condition Line 13). We use $\epsilon' = 2 \times \text{ave\_edge\_length}(T)$ where 'ave_edge_length' denotes the average edge length of the input mesh. We found this value empirically: a lower value generates useless points, and a higher value makes the algorithm slower by inserting smaller batches and thus requiring a larger number of outer iterations. Line 13: the function 'distance_to_pointset' gives the minimal distance between the input point and all the elements of the point set. The parameter ε ensures that all the points of ∂T are at most at a distance of ε of the re-meshed border. We use $\varepsilon \leftarrow 0.2 \times \text{ave\_edge\_length}(T)$. The effect of this parameter is discussed in §4.4.2.

4.3.2 Recombining tetrahedra into hexahedra

Once the border is re-meshed (by $\text{Del}(P)|_{\partial T}$), we can obtain an intermediate tetrahedral mesh T' by computing the Delaunay triangulation of the point set P constrained by the re-meshed border, using off-the-shelf software [GHS90, Si15]. In this section, we explain how to deduce from this intermediate tetrahedral mesh a hexahedral-dominant mesh by recombining the tetrahedra into

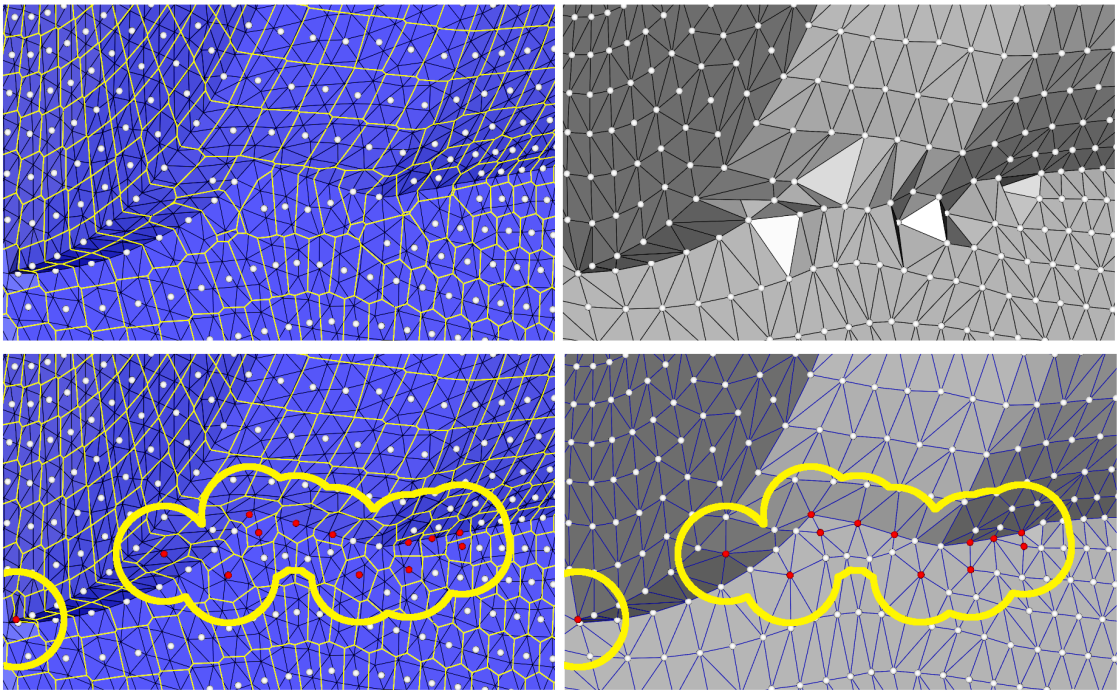


Figure 4.29: The border is re-meshed using the Delaunay triangulation of the generated points (white) restricted to the border of the domain. Additional points (red) are inserted until the geometric error is smaller than a user-defined threshold.

Algorithm 8: The mesh refinement algorithm

RefinePointSet(T,P,ε,ε'):

Data: T: the input tetrahedral mesh, and ∂T its boundary;
P: the point set to be refined; $\varepsilon > 0$: the maximum distance between the re-meshed boundary and the input boundary ∂T;
Result: The updated point set P with new inserted point to make the distance between Del(P)$|_{\partial T}$ and ∂T smaller than ϵ.

```
point set E ← sample_surface(∂T, ε)
point set B ← ∅
do
    for i := 1 to |E| do
        d[i] ← distance_to_surface(E[i], Del(P)|∂T)
    end
    sort (E, d) by decreasing d
    B ← ∅ ; i ← 0
    // Min. dist. between two points inserted in the same batch
    constant ε' ← 2 × avg_edge_length(T)
    while i < |E| and d[i] < ε/2 do
        if distance_to_pointset(E[i], B) > ε' then
            B ← B ∪ {E[i]}
        end
        i ← i + 1
    end
    P ← P ∪ B
while B ≠ ∅;
```

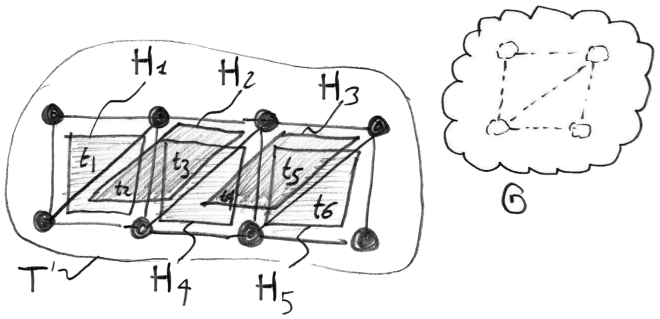


Figure 4.30: Recombination (in 2D): the template G describes the recombination of two triangles into a quad. Five instances $\mathbf{H}^* = \{H_1, H_2, H_3, H_4, H_5\}$ can be found in the intermediate mesh T'. Clearly, H_2 and H_3 are of lower quality (lower value of function Q). H_1 and H_2 are mutually incompatible ($C(H_1, H_2) = 1$) because they have t_2 in common. The other mutually incompatible pairs are (H_2, H_4), (H_4, H_3) and (H_3, H_5).

other primitives (hexahedra, and optionally prisms and pyramids). We first present the general problem statement:

Optimal Recombination: Problem statement

Given the following elements (see Fig. 4.30):

- an intermediate tetrahedral mesh $T' = \{t_i\}_{i=1}^{nt}$ where t_i denotes one of the tetrahedra and nt the number of tetrahedra;
- the set of "primitive templates" $\mathbf{G}$ to be recognized in T' (hexahedra, prisms, pyramids). Each template $G \in \mathbf{G}$ is a graph that encodes the combinatorial relations within a set of tetrahedra that corresponds to a primitive. This combinatorial representation comprises the adjacencies of each pair of tetrahedra along their common facets and additional information (more on this below). It is said that a primitive represented as a set of tetrahedra $H = (t_1, t_2, \dots t_k)$ *matches* a template G if the adjacencies between $(t_1, t_2, \dots t_k)$ correspond to the arcs of the graph G (a more formal definition will be given later).
- a criterion $Q(H) > 0$ that measures the geometric quality of a recognized primitive H. The criterion Q that we use is explicited further (see Section 4.3.2);
- a set of compatibility constraints $C(H_1, H_2)$. If both primitives of the pair can be constructed simultaneously in the resulting mesh, then they are said to be *mutually compatible* ($C(H_1, H_2) = 0$), otherwise they are *mutually incompatible* ($C(H_1, H_2) = 1$). Clearly, two primitives H_1 and H_2 that use the same tetrahedron are mutually incompatible ($H_1 \cap H_2 \neq \emptyset \Rightarrow C(H_1, H_2) = 1$). There are also less trivial compatibility constraints, described below (Section 4.3.2).

the optimal recombination problem can be stated as:

> Find the set of primitives $\mathbf{H}$ that maximizes the quality $Q(\mathbf{H}) = \sum_i Q(H_i)$, such that each primitive $P \in \mathbf{H}$ matches one of the templates $G \in \mathbf{G}$ and such that the mutual compatibility constraints are satisfied ($\forall H_1 \neq H_2 \in \mathbf{H}, C(H_1, H_2) = 0$).

Re-formulation / decomposition

This combinatorial optimization problem can be decomposed into two steps:

> *Step 1 - Template matching:*
> Find the set of all possible primitives $\mathbf{H}^*$ extracted from the intermediate tetrahedral mesh T' that match a template in $\mathbf{G}$.

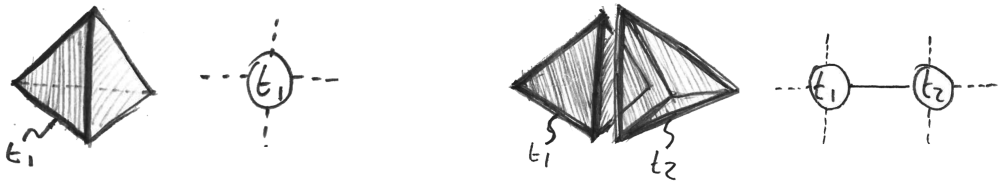
Step 2 - Constrained optimization:
$\text{Max}_{\mathbf{b}\in\{0,1\}^N}\left[\sum_{i=1}^{N} b_i Q(H_i)\right]$ where $N = |\mathbf{H}^*|$
s.t. $\forall i,j \in (1..N)^2, b_i b_j C(H_i, H_j) = 0$

Template matching (Step 1) is detailed further, in Section 4.3.2. In the constrained optimization problem of Step 2, one tries to find the *boolean vector* $\mathbf{b}$ that indicates for each potential primitive $H_i \in \mathbf{H}^*$ whether it will be used ($b_i = 1$) or not ($b_i = 0$) in the final result. The constraint indicates that whenever a primitive H_i is used, the primitives H_j that are incompatible with it cannot be used. Written in this form, this (combinatorial) constrained optimization problem can be recognized as the *maximum independent set problem*, classical in graph theory [BBPP99]. Since we will be able to find the set of all candidate primitives $\mathbf{H}^*$, this lets hope for an algorithm that provably finds the optimum recombination. Unfortunately, even for a few hundred elements, algorithms for the maximum independent set problem take a considerable amount of time, as confirmed by our experiments Therefore, for this step, we use a classical greedy heuristic (more on this in Section 4.3.2).

We now give more details about each phase of the algorithm, template matching (Section 4.3.2) and combinatorial optimization (Section 4.3.2).

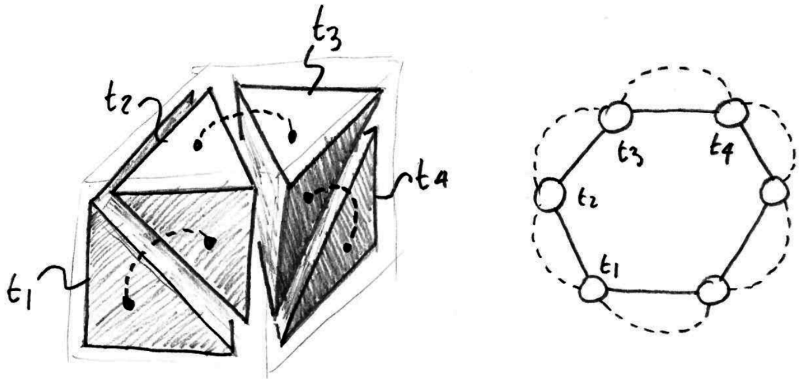
Template matching

Several algorithms were proposed for recombining tetrahedra into hexahedra [YS03, MT00]. We chose to elaborate on the approach proposed by Meshkat and Talmor [MT00], that provides a formalism that can be used to systematically analyze all the configurations. Their formalism represents a configuration as a graph, where each node corresponds to a tetrahedron:



Each (solid) arc connects two tetrahedra that share a facet. The dangling dashed arcs correspond to facets on the border, not connected to another tetrahedron.

In addition, to identify the quadrilateral facets in the decomposition of a hexahedron, they connect each pair of dangling dashed edges that correspond to the pair of triangular facets that form each quadrilateral facet:



In the example shown above, three dashed arcs are materialized on the left image (the three other hidden ones are in a similar configuration).

Meshkat and Talmor refer to such a graph (with both solid and dashed arcs) as the *augmented graph* of the decomposition. They enumerate the possible augmented graphs for decompositions with 5 and 6 tetrahedra. They mention (without describing it) a possible generalization in the presence of slivers. In Section 4.5, we further analyze their formalization, fill a gap in the proof, prove that a configuration cannot contain more than 13 tetrahedra, extend the analysis to all the possible configurations (from 5 to 13 tetrahedra), summarized in the following theorem:

Theorem 1. *A decomposition of a hexahedron into tetrahedra without any sliver on the border can have 5,6 or 7 tetrahedra. There is 1 configuration with 5 tetrahedra, 5 configurations with 6 tetrahedra and 4 configurations with 7 tetrahedra (see Fig. 4.31)*

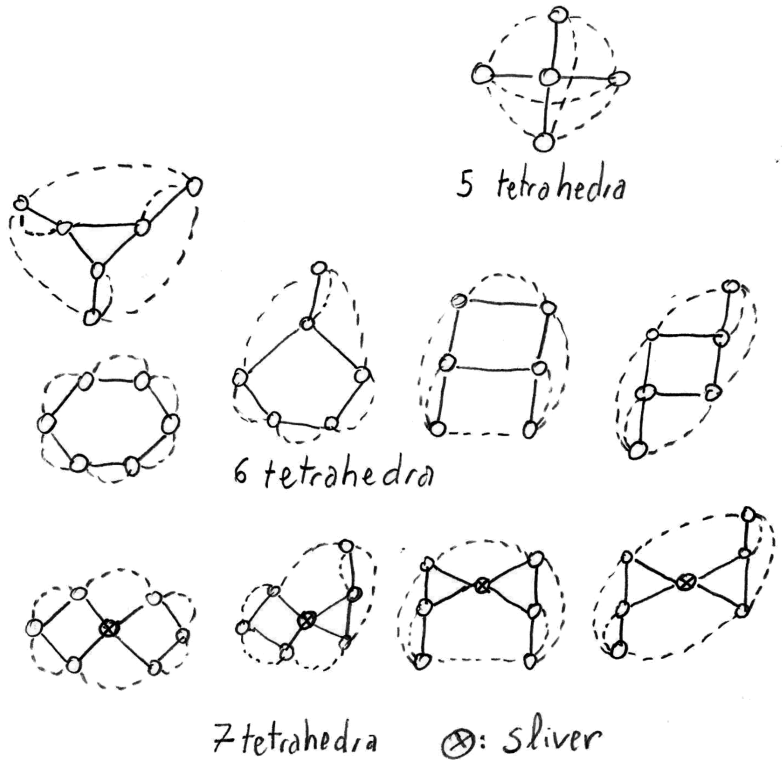
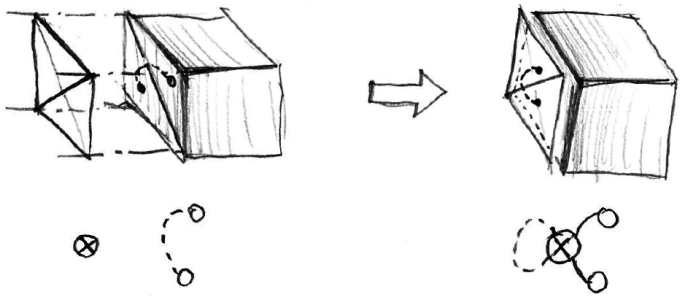


Figure 4.31: The augmented graphs of the ten decompositions of a hexahedron into tetrahedra.

Proof. See section 4.5 □

In addition, a sliver can be "glued" to each quadrilateral face of the hexahedron:



where the sliver is symbolized by a circled cross. The corresponding graph transform can be applied to each dashed arc, thus generating up to 2^6 graphs from each initial graph (modulo symmetries). Thus, a decomposition can have up to 13 tetrahedra (the maximum 13 is reached with one of the decompositions into 7 tetrahedra with 6 slivers glued on the quadrilateral facets).

Now that all the possible configurations of a hexahedron are known, we can proceed to describe the algorithm that recognizes them in the intermediate tetrahedral mesh T'. This is an instance of the subgraph isomorphism problem, known to be NP-complete [Coo71]. However, since the template graph to be recognized is small (no more than 7 nodes), systematic exploration with backtracking remains reasonably efficient. We follow the approach in [MT00], that first transforms each augmented graph into a "program", as exemplified in Fig. 4.32. Each node (tetrahedron) of the graph is numbered (left), then the graph is "linearized" (center). The "program-form" of the graph is a sequence of `LINK` and `QUAD` instructions that encode the solid and dashed arcs respectively.

At each step of the program, nodes touched by a previous `LINK` instruction are said to be *visited*. Node 0 is initially considered to be visited as well, before the program starts. To facilitate the design of the matching algorithm described below, the `LINK` and `QUAD` instructions are scheduled in the program in such a way that:

- a `QUAD` instruction always connects two already visited nodes;

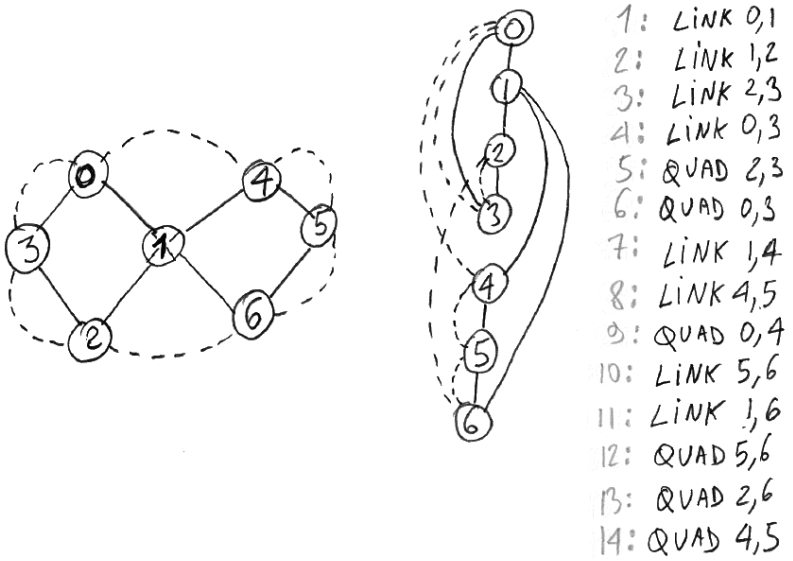


Figure 4.32: An augmented graph (left) is linearized (center) and transformed into a program (right).

- a `LINK` instruction can connect an already visited node to a new one (the new one is then visited), as in lines 1,2,7,8,10;
- or a `LINK` instruction can bridge two already visited nodes (lines 4 and 11).

Once each of the 10 augmented graphs of Fig. 4.31 is encoded as a program, all the possible primitives in the input tetrahedral mesh T' can be recognized by the following algorithm:

Algorithm 9 finds each mapping $H = [t_0, t_1, \dots t_7]$ that maps local node indices in the template graph to global tetrahedra indices in the intermediate mesh T'. When such a mapping H is complete (then referred to as a *matched primitive*), it is appended to the set $\mathbf{H}^*$ of recognized primitives. A mapping H that is incomplete is referred to as a *matching state*. The program `Prg` is a list of operations, each operation being one of $\texttt{LINK}_{i,j}$ or $\texttt{QUAD}_{i,j}$. The two operations are detailed in Algorithm 10 below. The function MergeSlivers detects all the slivers glued onto the 6 quadrilateral facets of the recognized hexahedron and merges them into the detected hexahedron. Such slivers are encountered whenever the pair of triangular facets that form a quad are connected to the same tetrahedron.

The $\texttt{QUAD}_{i,j}$ operation checks whether it is combinatorially possible to find a quadrilateral facet between tetrahedra t_i and t_j. The $\texttt{LINK}_{i,j}$ operation needs to explore all the possible assignments for tetrahedron t_j, resulting in a non-terminal recursion and requiring backtracking. In the algorithm, the facet f of a tetrahedron t_i is said to be *free* if tet_adjacent(T', t_i, f) is not in H $(\forall j \in [0..7], t_j \neq \text{tet\_adjacent}(T', t_i, f))$.

Implementation details / Optimizations: Since no inter-thread communication/synchronization is required, parallelization of the algorithm is very easy and directly gains a factor nearly linear in the number of cores. Besides this trivial optimization, we further optimized the algorithm, by early-discarding the matches that do not meet minimum quality requirements (for instance, a quadrilateral facet with angles that differ too much from 90 degrees). In addition, in the implementation of `LINK`, we avoid copying the matching state $(H' \leftarrow H)$ when there is only a single facet of t_i that can be linked (i.e., when the recursion is terminal). Finally, to avoid unnecessary traversals of both T' and H, we keep track of the tetrahedron facets that are free by using a bitfield.

Constrained optimization

Once the template matching algorithm of the previous section is executed, we obtain the set $\mathbf{H}^*$ of all possible primitives that can be recombined in the intermediate mesh T' by applying Algorithm 9 to the 10 programs that correspond to the 10 possible decompositions of a hexahedron (and optionally to the program that recognizes prisms and the one that recognizes pyramids).

Algorithm 9: The template-matching algorithm

FindAllMatches(`Prg`, T'):

Data: `Prg`: The program of an augmented graph; T': the intermediate tetrahedral mesh
Result: **H**: the set of all the primitives recognized by `Prg` in T'

H $\leftarrow \emptyset$
foreach $t \in T'$ **do**
  // H is a "matching state", i.e. a local-node-index to
  // global-tet-index mapping, initialized with $t_0 = t$
  H $\leftarrow [t, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset]$
  H $\leftarrow$ **H**$\cup$ FindMatchesRecursive(`Prg`, H, T')
end

FindMatchesRecursive(`Prg`, H, T'):

Data: `Prg`: Part of a program; H: a matching state; T': the intermediate tetrahedral mesh
Result: **H**: the set of all the instances recognized by `Prg` in T' from state H

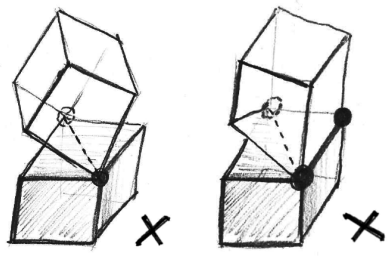
if `Prg` = `EMPTYLIST` **then**
  // If `Prg` *is empty, then all the instructions of the program*
  // where consumed (H is a matched primitive)
  MergeSlivers(H,T') ; **return** { H }
else
  // Consume the first operation in the program
  // `Op` is one of $\texttt{LINK}_{i,j}$ *or* $\texttt{QUAD}_{i,j}$ (see algo 10).
  `Op` $\leftarrow$ Head(`Prg`)
  `PrgRest` $\leftarrow$ Tail(`Prg`)
  // The rest of the program is passed to `Op`,
  // to allow recursion / backtracking.
  return `Op`*(H, T',* `PrgRest`*)*
end

The goal now is to find the subset $\mathbf{H} \subset \mathbf{H}^*$ that maximizes the quality criterion $Q(\mathbf{H}) = \sum_{H \in \mathbf{H}} Q(H)$ and that satisfies the compatibility constraints $\forall H_i, H_j \in \mathbf{H}, C(H_i, H_j) = 0$. We use the following quality criterion that favors flat quadrilateral facets with right angles:

$$Q(H) = \sum_{\text{q} \in \text{quads}(H)} \left(\widehat{n_1, n_2}^2 + \frac{1}{4} \sum_c \left(\hat{c} - \frac{\pi}{2} \right)^2 \right)$$

where $\widehat{n_1, n_2}$ denotes the angle between the two normals n_1, n_2 of the two triangular facets recombined in the quadrilateral facet q, and where $\hat{c}$ denotes the angle at the corner c of the quadrilateral facet q.

The compatibility criterion $C(H_1, H_2)$ is defined in function of the envisioned application for the generated hexahedral-dominant mesh. In our case, the Finite Element Modeling application that we target tolerates a single type of nonconformity in the mesh, that is a quadrilateral facet connected to two triangular facets. All the other nonconformities that involve the diagonal edges of the quadrilateral faces are forbidden. In other words, this means that the following two configurations are forbidden:



More formally, two primitives H_1 and H_2 are mutually incompatible ($C(H_1, H_2) = 1$) if one of the following conditions is met:

Algorithm 10: Algorithms for `QUAD` and `LINK`

`QUAD`$_{i,j}(H,T',$`PrgRest`$)$:
Data: H: a matching state; T': the intermediate tetrahedral mesh, `PrgRest`: The rest of the program.
Result: **H**: the set of recognized primitives
if t_i and t_j have two free facets with a common edge **then**
  return FindMatchesRecursive(`PrgRest`, H, T')
else
  return $\emptyset$
end

`LINK`$_{i,j}(H,T',$`PrgRest`$)$:
Data: H: a matching state; T': the intermediate tetrahedral mesh, `PrgRest`: The rest of the program.
Result: **H**: the set of recognized primitives
H $\leftarrow \emptyset$
for $f \in \{0,1,2,3\}$ **do**
  if CanLink$_{i,j,f}(H,T')$ **then**
    $H' \leftarrow H$ // *Copy the state, for backtracking*
    $t'_j \leftarrow$ adjacent_tet(T', t_i, f)
    H $\leftarrow$ **H**$\cup$ FindMatchesRecursive(`PrgRest`, H', T')
  end
end

CanLink$_{i,j,f}(H,T')$
Data: i,j: two local tet indices; f: a facet index, H: a matching state; T': the intermediate tetrahedral mesh
Result: `TRUE` if i and j can be linked, `FALSE` otherwise
if $t_j = \emptyset$ **then**
  return facet f of t_i is free
else
  return adjacent_tet$(T', t_i, f) = t_j$
end

1. H_1 and H_2 have one or more tetrahedra in common ($H_1 \cap H_2 \neq \emptyset$);

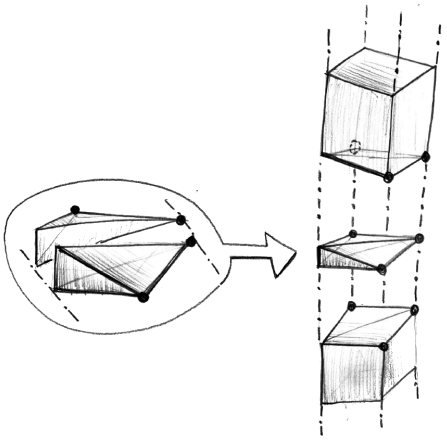
2. an edge of H_1 corresponds to a diagonal edge of H_2 or vice-versa (left figure);

3. two quadrilateral facets of H_1 and H_2 have three vertices in common (right figure).

Otherwise, H_1 and H_2 are mutually compatible ($C(H_1, H_2) = 0$).

Remark 4. [7] *At first sight, one may think that constraint (3) can be enforced by simply ensuring that the pair of tetrahedral facets adjacent to a quadrangular facet are connected to the same hexahedron. However, there exists a configuration where two hexahedra share three vertices without any direct adjacency between their tetrahedra:*

[7] We lost more than two days debugging and trying to figure out what was going on with this configuration, therefore we think its description may be useful for readers who want to reproduce our results.



The shown (invalid) configuration cannot be ruled-out by simply examining tetrahedron facet adjacencies. As a consequence, the algorithm needs to traverse the set of cells incident to each vertex.

Since there is no efficient algorithm that finds the exact set of primitives that maximizes Q while satisfying the compatibility constraints (maximum independent set problem), we use instead the heuristic outlined in Algorithm 11.

Algorithm 11: The constrained optimization algorithm

SelectPrimitives($\mathbf{H}^*, T'$):

Data: $\mathbf{H}^*$: the set of all the possible primitives recognized in T'
Result: $\mathbf{H}$: a set of mutually compatible primitives selected from $\mathbf{H}^*$

sort $\mathbf{H}^*$ by primitive type: hexahedron > prism > pyramid
sort primitives of same type by decreasing $Q(H)$
$\mathbf{H} \leftarrow \emptyset$
foreach $H_1 \in \mathbf{H}^*$ **do**
  if $(\forall H_2 \in \mathbf{H}, C(H_1, H_2) = 0)$ **then**
    $\mathbf{H} \leftarrow \mathbf{H} \cup \{H_1\}$
  end
end

Implementation details / Optimizations: To avoid the n^2 cost of testing mutual compatibility for all primitive in $\mathbf{H}^*$ against all the already recognized ones in $\mathbf{H}$, we observe that incompatibility between two primitives only occur when they share at least a vertex. Thus we restrict the compatibility test to the set of primitives that share a vertex with H_1. To quickly obtain this set, we chain in T' the tetrahedra incident to each vertex and we maintain an array that maps each tetrahedron of t to the primitive in $\mathbf{H}$ it belongs to.

4.4 Results and discussion

We evaluate the quality of our results by the proportion of hexahedra elements, and some measures of their geometric quality. Our algorithm was tested on a large set on models (available in supplemental material) and is discussed for a representative subset in this section.

We compare the classic hexahedral mesh quality measures with previous works in §4.4.1, we show how it behaves when the desired size for the hexahedra is not constant §4.4.2, and we discuss its strengths and weaknesses in term of robustness.

4.4.1 Hexahedra proportion and quality

We tested our method on various modeled shapes and CAD models with a 2.2Ghz Intel Core i7 CPU and 8GB of RAM laptop. Table 4.1 summarizes the resulting statistics and timings. We

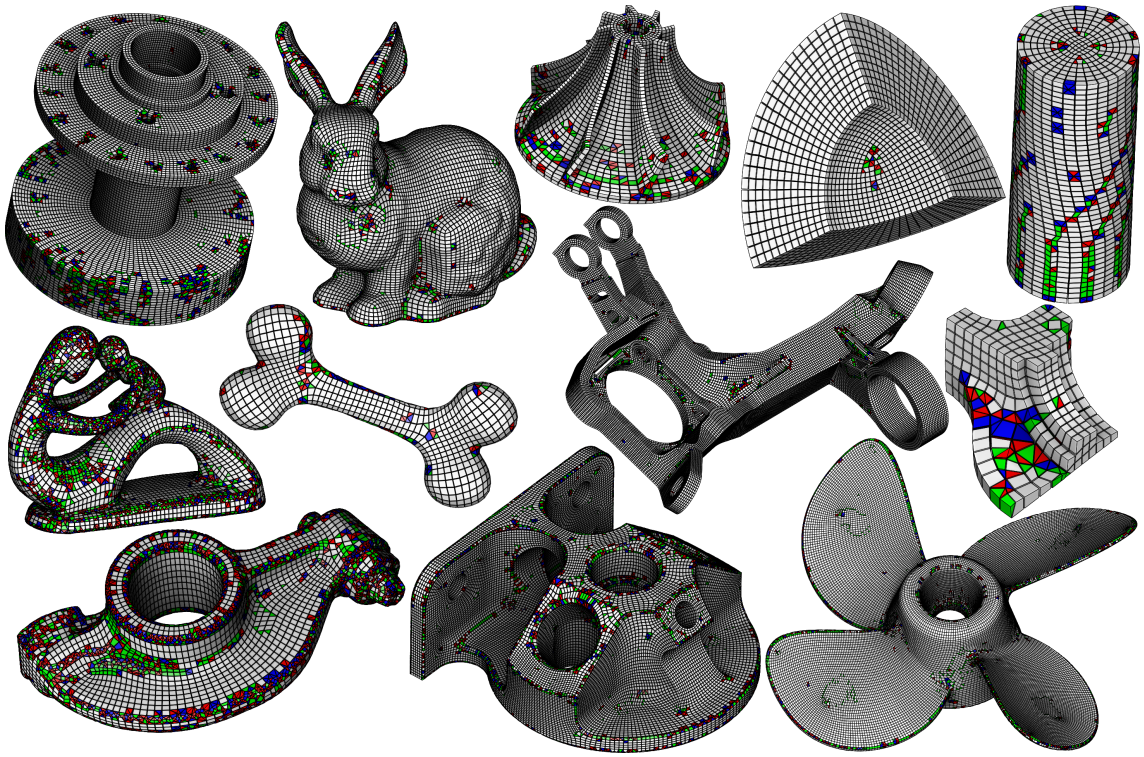


Figure 4.33: Some hexahedral-dominant meshes generated by our method, see statistics in Table 4.1 (first line: cubo, bunny, impeller, corner, cylinder; second line: fertility, bone, fusee, fandisk; third line: rocker arm, CV745, propeller)

copied the results reported in [BRM+14] in Table 4.1 (lines in gray) for comparison purposes. In all our experiments we made the number of vertices generated by our method as close as possible to the other method for comparison purposes.

Fig. 4.33 illustrates the models produced with our method and reported in Table 4.1. The hexahedra are depicted in white, tetrahedra in red, prisms in green and pyramids in blue. Examples of extra points addition can be observed on fandisk, fertility, rocker arm and CV745 models. For these models the sampling of ∂T realized by P is not sufficient as measured by the Hausdorff distance and reported in the column $dist1$. The added points imply the creation of tetrahedra, prisms and pyramids that can be observed in several regions of these models.

We observe in Table 4.1 that the percentage of hexahedra is higher in number and volume with our method despite the addition of points (e.g. CV745 model). The quality Q of the produced hexahedra is also better with our method. The significant amount of time of the refinement step is due to the number of batches required to reach the user defined threshold. Indeed, each batch of points inserted in P implies to update $\mathrm{Del(P)}|_{\partial T}$, the Delaunay triangulation of P restricted to the border of the domain ∂T.

To experiment how the method scales up both in terms of input complexity and number of generated cells, we used it to generate an hexahedral-dominant mesh of a complete engine block (TRX engine from GRABCAD), shown in Fig. 4.43.

4.4.2 Influence of the parameters

All our results in the previous subsection were produced using the default parameters. We now discuss the influence of each parameter and its impact on the result quality.

Maximal proportion of curl correction

The parameter γ (introduced in § 4.2.4) that tunes how much curl correction is allowed really impacts complex models (Fig. 4.34). High values of γ (left) make the correction term meaningless, so the algorithm introduces many T-junctions to balance the curl of the frame field. When γ is low (right), the correction term makes the field locally integrable everywhere, exactly as in CubeCover

Table 4.1: Statistics and timings. #vert is the number of vertices of the generated model. The hexahedra proportion is measured by the percentage of hexahedra in number (H_{nbr}) and volume (H_{vol}). The hexahedra mesh quality Q is measured by the scaled Jacobian in the format of average | standard deviation. We measure the Hausdorff distance between our mesh and the reference input mesh in the format of $M \rightarrow \partial T | \partial T \rightarrow M$ with M the mesh before (dist1) and after (dist2) the remeshing step. We measure the timings of the tetrahedralization of the input surface, frame field generation (FF), curl correction (CC), global parameterization (PGP), point set extraction, refinement step, tetrahedra to hexahedra step and whole pipeline in seconds.

model	#vert	H_{nbr}	H_{vol}	Q	dist1	dist2	tet input	FF	CC	PGP	pointset	refine	tet2hex	total
fusee	131,508	81.22	94.62	0.98 | 0.04	0.387 | 0.252	0.070 | 0.060	52.41	104.52	195.87	58.76	29.28	110.82	116.42	728.32
fusee	126,922	62.91	85.70	0.95 n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	488
CV745	133,331	72.45	91.63	0.97 | 0.05	0.845 | 0.540	0.101 | 0.091	109.29	221.86	494.2	117.51	59.46	266.07	125.67	1496.42
CV745	133,436	n/a	89.74	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	247
propeller	132,469	71.27	91.10	0.97 | 0.05	0.355 | 0.327	0.035 | 0.062	47.95	127.08	191.78	51.24	28.99	119.82	123.28	748.57
propeller	133,678	n/a	83.65	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	268
cubo	109,182	72.87	89.01	0.98 | 0.04	0.782 | 0.671	0.705 | 0.257	7.45	13.22	19.5	5.14	7.74	69.54	98.01	247.17
cubo	102,946	n/a	78.55	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	225
cylinder	11,205	64.66	90.85	0.96 | 0.06	0.208 | 0.214	0.134 | 0.137	33.32	65.56	116.86	39.00	17.37	21.55	12.08	327.13
cylinder	11,648	58.16	82.68	0.94 n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	112
corner	6,538	95.46	99.55	0.99 | 0.02	0.095 | 0.102	0.095 | 0.102	2.73	5.59	6.05	2.37	1.66	0.83	6.04	27.98
corner	6,006	84.54	94.10	0.96 n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	80
fandisk	856	51.36	77.82	0.95 | 0.06	2.184 | 3.289	0.458 | 0.493	0.96	1.87	2.52	0.69	0.45	1.59	0.53	9.6
rockerarm	15,564	33.05	80.15	0.95 | 0.09	1.308 | 1.258	0.072 | 0.074	91.45	215.2	546.68	103.00	39.99	147.66	11.56	1209.32
impeller	13,896	53.31	81.11	0.95 | 0.06	1.408 | 0.848	0.333 | 0.219	4.51	8.02	11.28	3.16	2.69	11.9	10.7	59.37
bunny	116,149	60.57	88.61	0.95 | 0.06	0.791 | 0.779	0.123 | 0.129	23.74	56.1	127.88	29.62	19.66	42.6	136.43	468.89
bone	4,225	45.17	82.54	0.92 | 0.10	1.838 | 1.145	0.264 | 0.257	2.38	5.13	12.48	3.15	1.82	3.11	4.15	35.07
fertility	20,068	33.63	78.40	0.93 | 0.11	1.292 | 0.904	0.069 | 0.071	89.42	205.58	454.71	106.06	43.15	145.71	16.51	1120.7

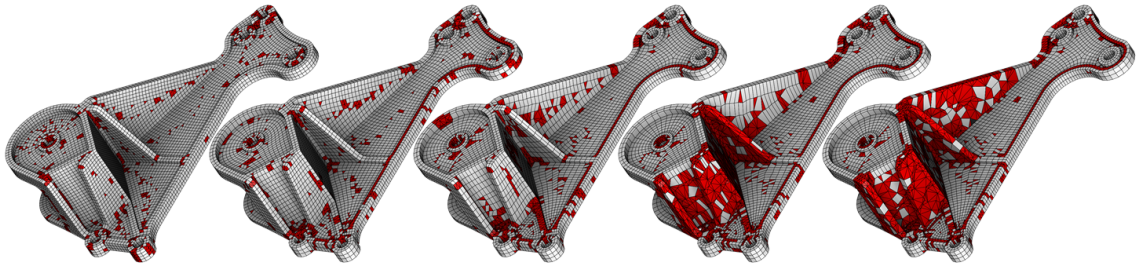


Figure 4.34: Influence of the maximum curl-correction parameter $\gamma \in \{1, 0.8, 0.6, 0.4, 0.2, 0\}$. A value of γ that is too low results in the same over-constrained problem as in CubeCover (right images).

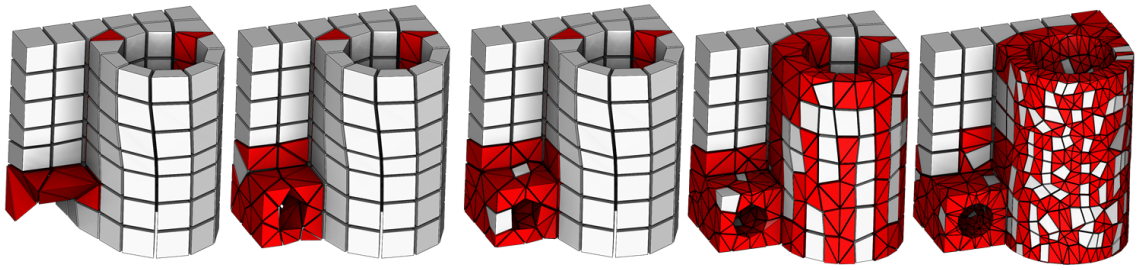


Figure 4.35: Influence of the maximal deviation parameter $\epsilon \in \{100\%, 2\%, 1\%, 0.5\%, 0.25\%\}$ of the average edge length of the input mesh. A high tolerance misses some features (left), whereas a too strict tolerance inserts too many points (right).

before introducing the constraint to have integer variables. As a consequence, when γ is too low, our algorithm have the same failure cases as CubeCover.

Extra points to fit the original model

Our algorithm adds extra points to the point set P to ensure that the hexahedral-dominant mesh is close enough to the input mesh §4.3.1. It stops when it does not find any point of the input mesh that is further than a given threshold to the current reconstructed tetrahedral mesh. The impact of this threshold is illustrated in Fig. 4.35: if the threshold is high, our algorithm fails to capture details of the model, but if it is too low, then it adds too many extra points to curved areas and thus prevents some hexahedra from being reconstructed.

Varying scale

In our method, the scale of the hexahedra is prescribed by the norm of the columns of B_i. We demonstrate a varying isotropic scaling on a real object Fig. 4.36–up, and an anisotropic scaling on a cube Fig. 4.36–bottom. Our method handles the scale variation by introducing new singularities that split a layer of cubes into two layers of cubes. Intuitively, this behavior may be considered as the $3D$ counterpart of the T-vertices used in quadrilateral surface re-meshing.

4.4.3 Robustness

Recent advances in pure hexahedral re-meshing [LLX+12] produce very good results, but are limited by global constraints due to the topology of the input frame field. We review the most common failure cases of recent full hexahedral re-meshing algorithms and show what we produce in these situations.

The most famous failure case is the jump ramp (Fig. 4.39) that make global parameterization fail with a field that has no singularity. It was observed (and partially fixed) in [GSZ11] where the global parameterization was a simple polycube map. In this example, our method adds non hexahedral elements to smoothly connect 0 to $5 - 10$ layers of hexes.

Failure cases can also come from constraints that are not local (as in the lower part of the jump ramp). Such non-local constraints are typically encountered in the square screw example

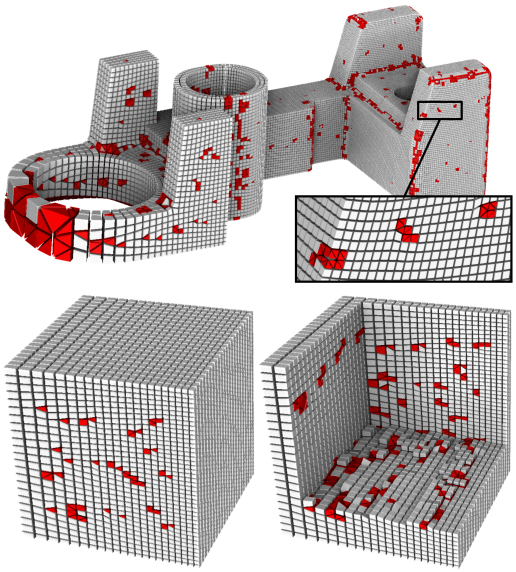


Figure 4.36: Varying scale: In the upper image, the desired hexahedra size varies linearly with the x coordinate. The bottom row shows a cube (clipped on the right image) with a varying anisotropic scale. Singularities are evenly distributed to absorb the variation of resolution.

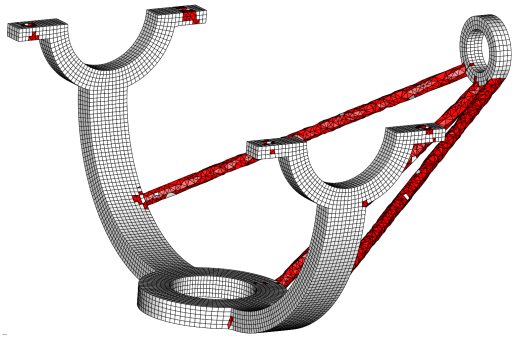


Figure 4.37: Our algorithm is not able to produce hexahedra for small parts of the object (red), where it generates tetrahedra.

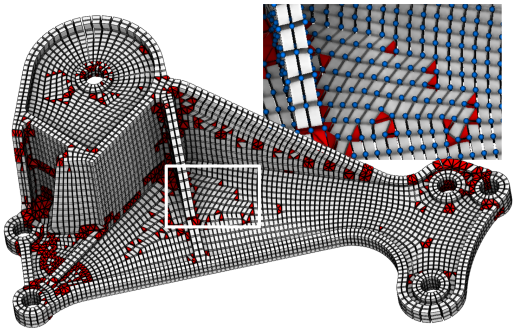


Figure 4.38: When the point set has too much shear, our recombination algorithm may find a (locally) better grid as shown in the close-up.

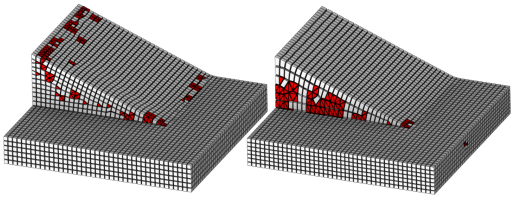


Figure 4.39: Jump Ramp (a classical failure case of global parameterization method), treated by our method without (left) or with (right) scale correction. With scale correction, artifacts similar to the ones obtain with global parameterization methods are encountered.

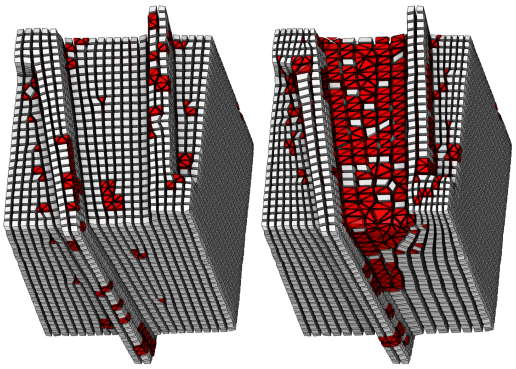


Figure 4.40: Our algorithm without scale correction produces a nice hexahedral-dominant mesh for the squared screw (left). If we try to use scale correction, we have the same over constrained problem as CubeCover and it does not improve the result.

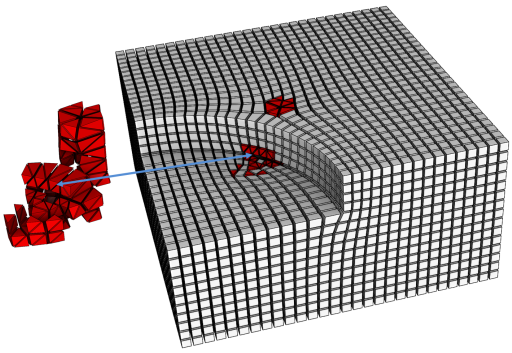


Figure 4.41: Even if the frame field does not correspond to a full hexahedral mesh, our algorithm is able to produce a hexahedral-dominant mesh. The produced non-hexahedral elements are shown on the left.

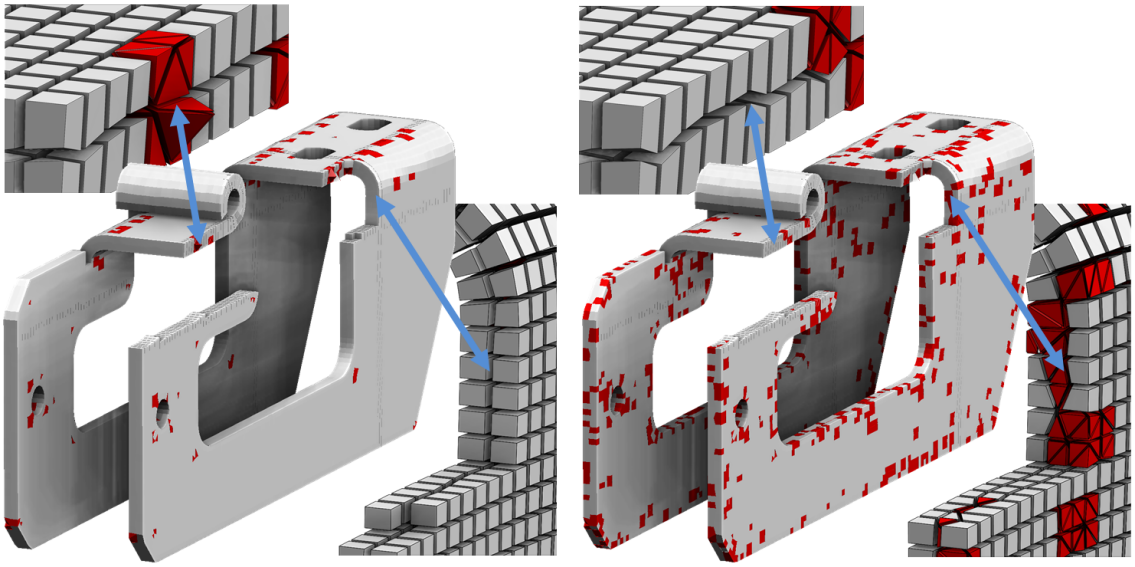


Figure 4.42: The number of layers of hexahedra depends on the prescribed edge length. For a plate, this number of layers is constant (left). In some limit cases (right), the algorithm is not able to find the same number of layers everywhere on the model and "dithers" between two integer values, thus producing many singularities.

(Fig. 4.40). Considering a frame field topology without any singularity, finding a global parameterization is equivalent to constructing a polycube map. If we try to align all the surface normals with their closest axis, it will necessarily squish the model. In this configuration, our algorithm relaxes the incompatible constraints by introducing non hexahedral elements.

Beside these global issues, the input frame field is not always compatible with CubeCover, that requires a nice behavior of the frame field around its singularity curves. Local editing [LLX+12] allows to filter-out noisy topology, but it is not always sufficient. For example, in Fig. 4.41 a singularity curve is doing half a turn inside the volume, that requires 3 different and incompatible orientation flags for the same curve. Our algorithm still works properly in these situations.

Our solution has its own drawbacks:

- When either the input mesh or the output mesh is not dense enough to represent the frame field singularity graph, the algorithm is not able to produce hexahedra (Fig. 4.37). This drawback also exists in other methods;

- for plates and thick surfaces, our algorithm produces a small number of layers of hexahedra according to the desired edge length. For some particular values of the prescribed edge length, the number of layers is undetermined, in the sense that the optimization step produces a number of layers that varies on the model. Intuitively, the algorithm "dithers" between two number of layers, and these variations induce many singularities (Fig. 4.42). We mention that this phenomenon is seldom encountered, and producing such failure cases is difficult: changing the prescribed edge length by more than 1% solves the problem. However, by changing the prescribed edge value by up to 10%, one may still observe this behavior over limited local zones of the model, due to the input mesh resolution, as in Fig. 4.42—Left where a small number of tetrahedra is still produced (in red);

- whenever the orientation followed by the point set is too much sheared, our recombination algorithm may produce hexahedra with a better geometry, but that are less coherent with their neighbors, as in the highlighted zone of Fig. 4.38.

The possibility of creating non-hexahedral elements allows our algorithm to escape from the failure cases encountered with pure hexahedral meshing methods. In all the examples that we tested, our method generated a valid hexahedral dominant mesh in a fully automatic manner. The main drawback is that, in the cases listed above, our algorithm introduces non-hexahedral elements

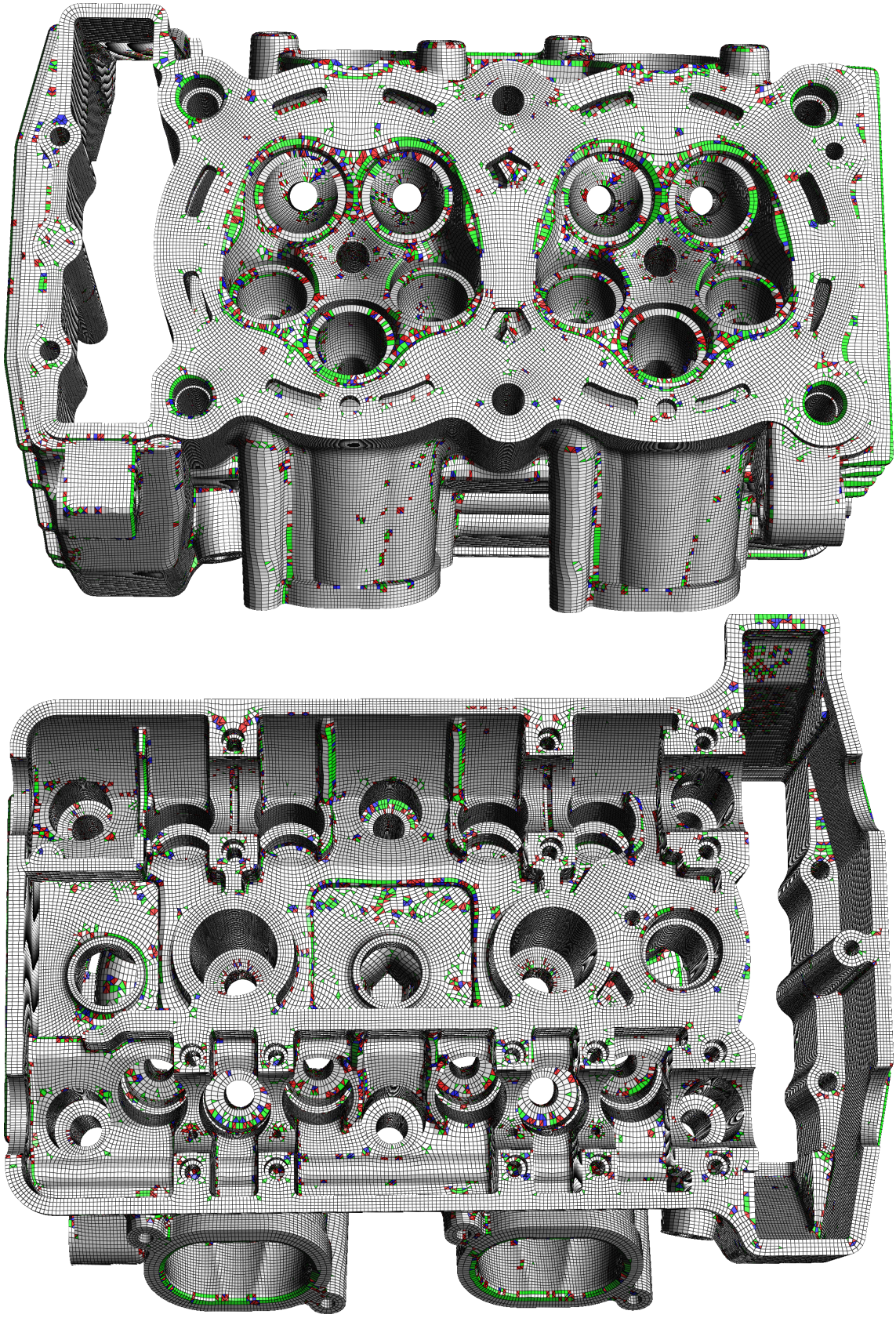


Figure 4.43: Hexahedral-dominant mesh of the TRX engine (from GRABCAD). The input tetrahedral mesh has 2,123,979 vertices and 10,192,895 tetrahedra. The hexahedral-dominant mesh was generated by our algorithm in 1170 seconds (on a laptop with an Intel core i7 and 16Gb RAM). The result has 1,337,083 vertices and 1,894,549 cells in which 1,006,899 are hexahedra.

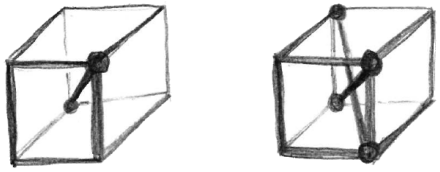
to avoid distortion of hexahedra, even in some cases where the distortion of a fully regular grid of hexahedra would remain acceptable.

4.5 Enumerating the decompositions of a hexahedron

The main result in [MT00] (theorem 2 below) enumerates all the possible decompositions of a hexahedron into 5 or 6 tetrahedra. We explain here the main arguments used in the proof, identify some configurations that were not analyzed in the initial article and provide the additional arguments (Lemma 9) to rule them out. Our conclusions are the same as in the initial article, but comes with a complete proof (section 4.5.1). Our additional analysis provides the basis for studying the configurations with slivers (section 4.5.2). Before studying the configurations, we first give a lemma that bounds the total number of tetrahedra in a decomposition.

Lemma 4. *The decomposition of a hexahedron into tetrahedra without any sliver glued on a quadrilateral face has at least 5 tetrahedra and at most 7 tetrahedra.*

Proof. The decomposition satisfies the Euler-Poincaré identity, i.e. $\chi = V - E + F - T = 1$, where $V = 8$ denotes the number of vertices of the cube, E the number of edges in the decomposition, F the number of faces and T the number of tetrahedra. There are 12 facets on the border (two per quadrilateral facet), and 18 edges on the border (12 + 6 diagonals). Let F_{int} and E_{int} denote the number of internal facets and the number of internal edges respectively. We have $F = 12 + F_{int}$ and $E = 18 + E_{int}$. Note also that $4T = 12 + 2F_{int}$, or $F_{int} = 2T - 6$. Injecting these identities into the Euler-Poincaré identity, we get $T = E_{int} + 5$.



An internal edge is a diagonal of the cube, as shown on the left figure. In a cube decomposition, there can be at most two internal edges, configured like on the right figure (other configurations generate intersecting tetrahedra), therefore we have $0 \leq E_{int} \leq 2$, and $5 \leq T \leq 7$. □

Note that we supposed that there was no sliver glued on a face of the hexahedron. If we include configurations with such slivers, we can find up to $7 + 6 = 13$ tetrahedra in a decomposition(more on this in section 4.5.2).

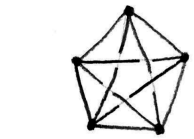
4.5.1 Decomposition of a hexahedron into 5 or 6 tetrahedra

We now enumerate all the graphs that correspond to the decomposition of a hexahedron into five or six tetrahedra (and later, decompositions with up to 13 tetrahedra). To analyze the different possible decompositions, we use a graph representation, where each node corresponds to a tetrahedron, each solid arc corresponds to a facet shared by two tetrahedra, and each dashed arc corresponds to a pair of facets on the border that form a quadrilateral facet (see illustrations in §4.3.2).

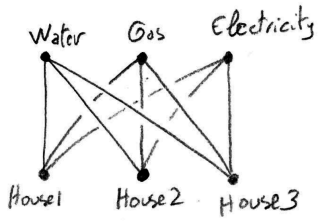
The following lemmas are useful, since they exhibit some constraints that significantly restricts the set of graphs to be analyzed.

Lemma 5. *The graph of the decomposition into five or six tetrahedra is planar.*

Proof. A non-planar graph contains either the complete graph or the utility graph as a subgraph or a minor [Liu68] (i.e. can be transformed into the complete graph or the utility graph by deleting edges and nodes).



COMPLETE GRAPH



UTILITY GRAPH

Recalling that $F_{int} = 2T - 6$ (see proof of Lemma 4), the decomposition into 5 tetrahedra has $e = F_{int} = 4$ arcs and the decomposition into 6 tetrahedra has 5 arcs, which is smaller than the number of arcs in the complete graph and in the utility graph, therefore they do not contain any of them as a subgraph or a minor. □

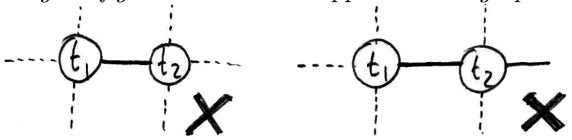
Lemma 6. *The graph of the decomposition into 5 tetrahedra is a tree.*

Proof. The 5 tetrahedra have 20 faces, including the 12 facets on the border of the hexahedra. The remaining 8 faces are internal. Since each solid arc corresponds to a pair of internal facets, there are 4 solid arcs in the graph. Since the graph is planar, the decomposition of the plane that it yields satisfies the Euler-Poincaré identity, i.e. $\chi = v - e + f = 2$, where v denotes the number of vertices in the graph ($v = T = 5$), e the number of arcs ($e = 4$) and f the number of faces (including the infinite face). In this case, we obtain $f = 1$, which means there is only the infinite face, thus the graph is a tree. □

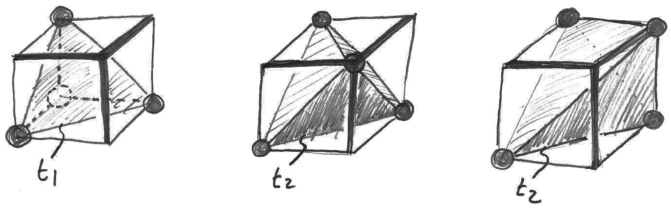
Lemma 7. *The graph of the decomposition into six tetrahedra has exactly one cycle.*

Proof. A derivation similar to the proof of Lemma 6 leads to $f = 2$, therefore the decomposition has the infinite face and another one, that corresponds to a cycle in the graph. □

Lemma 8. *In the decomposition of a hexahedron into tetrahedra, if a tetrahedron has 3 facets on the border, its neighboring tetrahedron has either 0 or 1 facet on the border.*
In other words, the following configurations cannot appear in the graph:

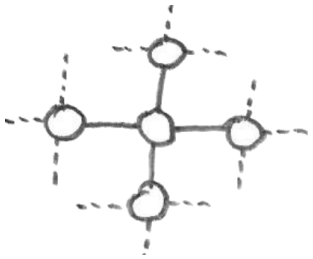


(where each dangling dashed edge corresponds to a tetrahedron facet on the border of the hexahedron).

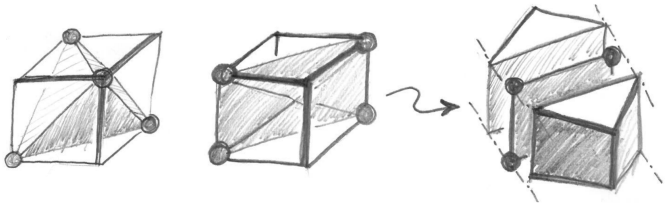


Proof. Let t_1 denote a tetrahedron with three facets on the border of the hexahedron (left figure). Let t_2 be a tetrahedron adjacent to t_1. There are only two possible configurations for t_2, with either no facet on the border (center figure) or one facet on the border (right figure). □

Lemma 9. *In the decomposition of a hexahedron into 5 or 6 tetrahedra, if a tetrahedron has no facet on the border, then each of its 4 neighbors has 3 facets on the border.*
In other words, within the decompositions into 5 or 6 tetrahedra, only the following configuration has a tetrahedron with no facet on the border:



Proof. By enumerating all the possible ways of choosing the 4 vertices of a tetrahedron from the 8 vertices of a hexahedron, one can see that the only two configurations where a tetrahedron does not have 3 vertices on the same facet of the hexahedron are as follows (left and center image):

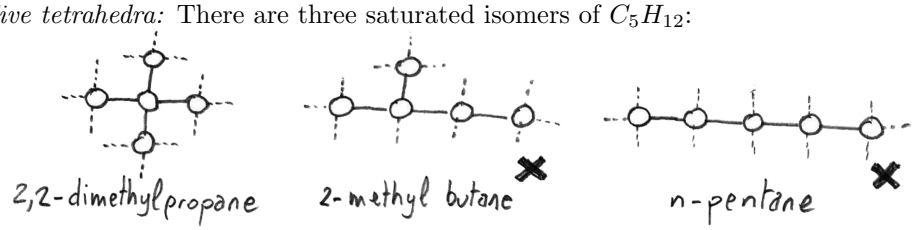


The configuration shown in the center image corresponds to a sliver that splits the hexahedron into two prisms (right image). Since the decomposition of each prism has at least 3 tetrahedra, this means that such a configuration can only appear in decompositions with at least 7 tetrahedra (more on this later). With 5 or 6 tetrahedra, only the configuration on the left can appear. Since there are exactly 12 triangular facets on the border of the hexahedron, each of the neighboring 4 tetrahedra has its remaining 3 facets on the border. □

Equipped with these lemmas, it is now simple to enumerate all the configurations with 5 and 6 tetrahedra. We find it worth mentioning that since all nodes are of degree 4, there is a natural correspondence between the set of admissible graphs and hydrocarbons. By looking-up all the isomers of C_6H_{12}, we found configurations that were overlooked in the initial article (in the end, the conclusion is the same, but this fills a hole in the proof). Each tetrahedron corresponds to a carbon atom, and facets on the border to hydrogen atoms. The configurations with 5 tetrahedra correspond to isomers of C_5H_{12}, and the configurations with 6 hexahedra to isomers of C_6H_{12} with one cycle:

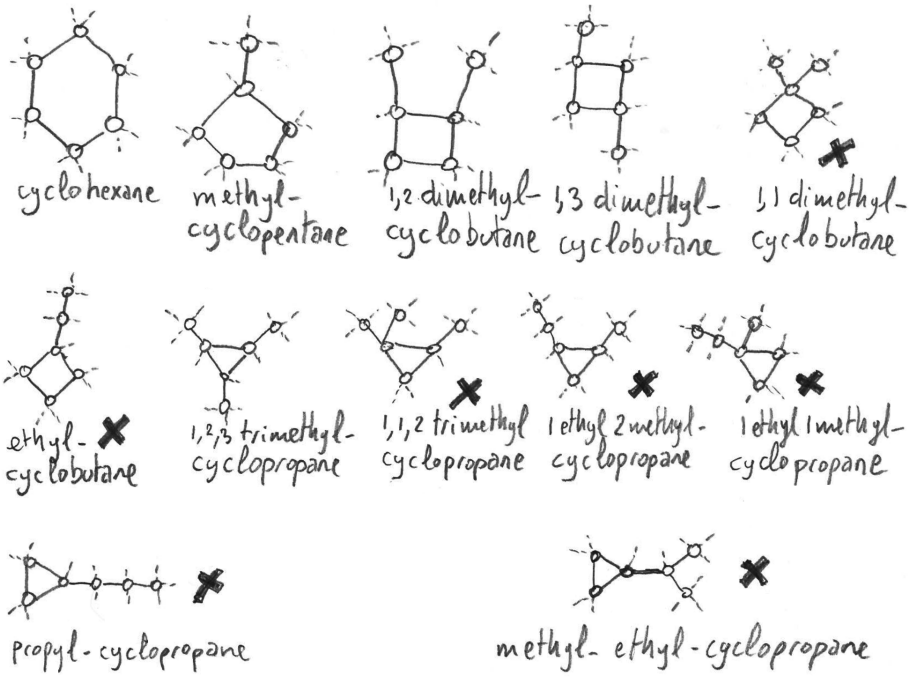
Theorem 2. *There is exactly one possible decomposition of a hexahedron into five tetrahedra and there are exactly five decompositions of a hexahedron into six tetrahedra.*

Proof. Five tetrahedra: There are three saturated isomers of C_5H_{12}:



Since they violate the condition in Lemma 8, the second one (2-methylbutane) and third one (n-pentane) do not correspond to the decomposition of a hexahedron.

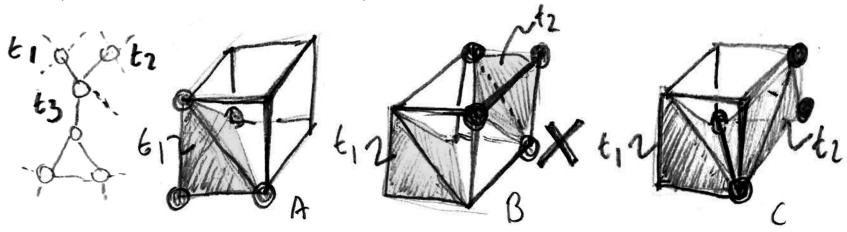
Six tetrahedra: There are twelve isomers of C_6H_{12} with one cycle:



Among these twelve isomers, only five of them correspond to the decomposition of a hexahedron:

- ethyl-cyclobutane, propyl-cyclopropane, 1-ethyl-2-methyl-cyclopropane and 1-ethyl-1-methyl-cyclopropane are ruled-out by Lemma 8;
- 1,1-dimethyl-cyclobutane and 1,1,2-trimethyl-cyclopropane are ruled-out by Lemma 9;

The last one (methylethyl-cyclopropane) requires a particularized analysis:



Tetrahedron t_1 has three facets on the border, and corresponds to a "chopped corner" of the hexahedron (A). For tetrahedron t_2, that also has three facets on the border, there are two possibilities (B,C). Configuration (B) is ruled out because there is no tetrahedron t_3 adjacent to both t_1 and t_2, therefore they must be in configuration (C). The only tetrahedron t_3 adjacent to both t_1 and t_2 has no facet on the border, which mismatches the "methylethyl-cyclopropane" graph on the left, where t_3 has exactly one facet on the border. Therefore there is no decomposition of an hexahedron into tetrahedra that matches this graph.

This completes the enumeration of all possible graphs and the discrimination of the ones that do not correspond to the decomposition of a hexahedron. □

4.5.2 Decomposition of a hexahedron into 7 to 13 tetrahedra

We now proceed to analyze the decomposition of an hexahedron into a larger number of tetrahedron, that include slivers, i.e. tetrahedra with nearly coplanar vertices. To introduce a sliver into the decomposition, there are two possibilities:

- append a sliver to a face of the hexahedron:

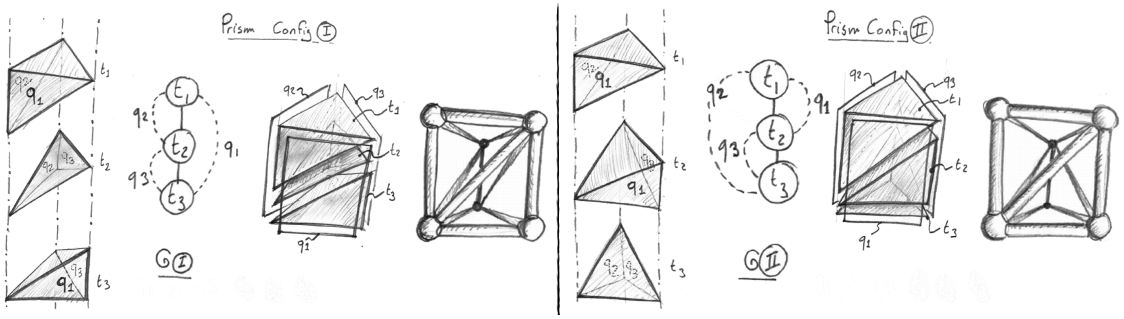
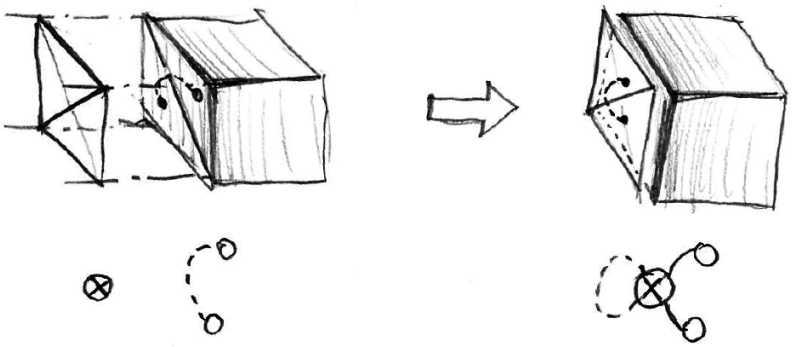


Figure 4.44: For a fixed front quadrilateral face (q_1), there are two decompositions of a prism into three tetrahedra, from left to right, in exploded view, graph representation, compact view and shaded wire-frame.



where the sliver is symbolized by a circled cross. The corresponding graph transform can be applied to each dashed arc, thus generating 2^6 graphs from each initial graph (modulo symmetries). We do not need enumerating all these graphs, it is easier to change the recognition algorithm as follows: once a hexahedron is recognized, we test whether a sliver can be merged to it for each of its faces;

- split the hexahedron into two prisms and connect them with a sliver (see the figure in the proof of Lemma 9).

This second way of introducing a sliver into the decomposition requires a finer analysis, since it more deeply changes the graph. Therefore, we need to enumerate all the new graphs that are generated by this operation. In other words, we need to identify within the five decompositions of a hexahedron into six tetrahedra which ones correspond to two prisms, and how the corresponding tetrahedra relate with the two prisms.

Without loss of generality, we consider the decomposition of a prism into three tetrahedra where the front quadrilateral facet q_1 and its diagonal are constrained, as shown in Fig. 4.44. In this setting, there are exactly two decompositions, G_I and G_{II}, that respect the constraint. Clearly the two graphs G_I and G_{II} are isomorphic (there is only one decomposition of a prism into three tetrahedra). What distinguishes G_I from G_{II} in this setting is which dashed arc corresponds to the constrained facet q_1.

At this point, one can notice that G_I is invariant by a 180 degrees rotation in the plane of the figure: the two vertices that are "far away" (black dots in the shaded wire-frame view) are both of degree 4. In contrast, G_{II} is not rotation invariant: the two vertices that are "far away" are of degree 5 (top one) and 3 (bottom one). Therefore, G_{II} comes in two different "flavors", that we will call G^+_{II} if the degree 5 vertex is at the top (like on the figure), and G^-_{II} if the degree 5 vertex is at the bottom. Again, this does not make any difference when considering a single isolated prism, but it will generate different graphs when gluing two prisms along q_1.

We can now enumerate the different ways of creating a hexahedron by assembling two prisms in configuration G_I, G^+_{II} or G^-_{II}. As shown in Fig. 4.45, there are four different configurations, $G_I - G_I$, $G_I - G^+_{II} (= G_I - G^-_{II})$, $G^+_{II} - G^+_{II} (= G^-_{II} - G^-_{II})$ and $G^+_{II} - G^-_{II} (= G^-_{II} - G^+_{II})$. They correspond to four of the five decompositions of an hexahedron into six tetrahedra listed in the previous subsection. Inserting a sliver results in the graphs shown on the right column. The fifth configuration with six tetrahedra (1,2,3 trimethyl-cyclopropane) cannot be split into two prisms

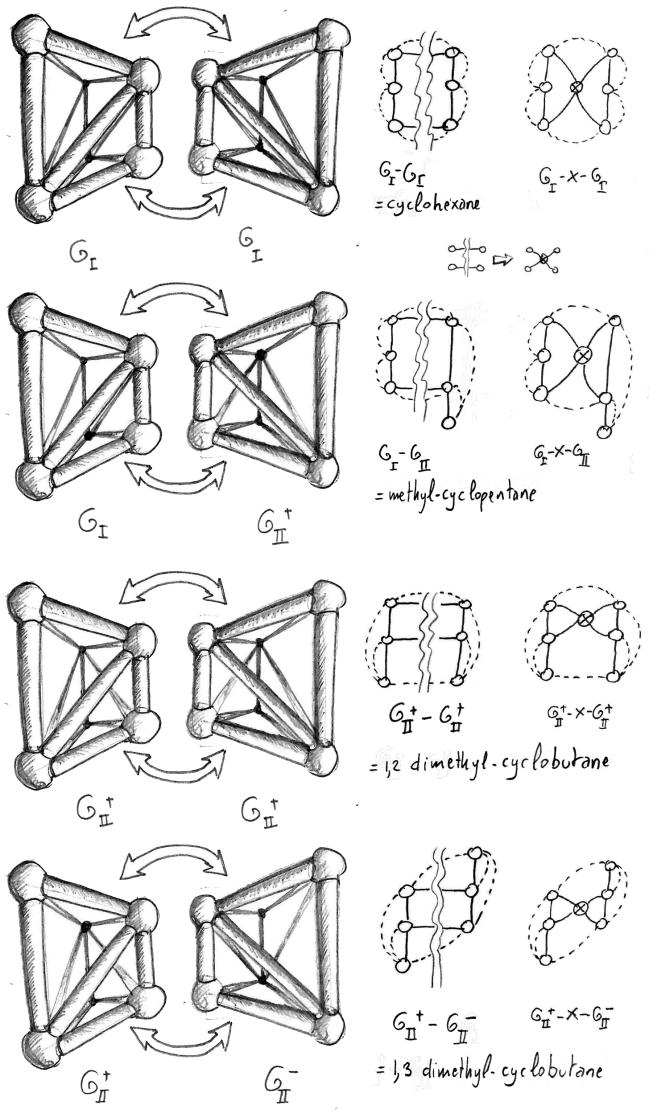


Figure 4.45: There are four possible ways of assembling prisms of type G_I, G_{II}^+ and G_{II}^-. For each configuration, a sliver can be inserted between the two prisms (graphs on the right).

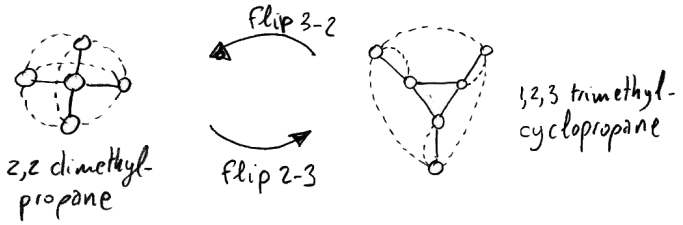


Figure 4.46: Among the six decompositions into 5 or 6 tetrahedra, two of them cannot be split into 2 prisms. The one with six tetrahedra (right) can be deduced from the one with five tetrahedra (left) by applying a flip-2-3 operation to any pair of tetrahedra that share a facet.

(and thus a sliver cannot be inserted into it).

This completes the enumeration of the decompositions of a hexahedron into 5 to 13 tetrahedra. To summarize, such a decomposition can be one of:

- One of the two "non-prismatic" decompositions into 5 and 6 tetrahedra (Fig. 4.46);
- one of the four "prismatic" decompositions (Fig. 4.45, left column);
- one of the four "prismatic" configurations with an internal sliver (Fig. 4.45, right column);
- a configurations obtained by appending 1 to 6 slivers to the quadrilateral faces of a configuration listed above.

Remark 5. *With the same argument as in Lemmas 6 and 7, one can see that the graph of a decomposition into 7 tetrahedra has exactly two cycles, which is the case of the four configurations that we found (prismatic configuration with an internal sliver, right column of Fig. 4.45).*

Bibliography

[ACSD+03] Pierre Alliez, David Cohen-Steiner, Olivier Devillers, Bruno Lévy, and Mathieu Desbrun. Anisotropic polygonal remeshing. *ACM Trans. Graph.*, 22(3):485–493, 2003.

[AMS90] Ashby, Manteuffel, and Saylor. A taxonomy for conjugate gradient methods. *J. Numer. Anal.*, 27:1542–1568, 1990.

[Aro09] Jacob Aron. The mandelbulb: first 'true' 3d image of famous fractal. *New Scientist*, 204(2736):54 –, 2009.

[Bar88] Michael Barnsley. *Fractals everywhere*. Academic Press Professional, Inc., San Diego, CA, USA, 1988.

[BBPP99] I. M. Bomze, M. Budinich, P. M. Pardalos, and M. Pelillo. The maximum clique problem. In *Handbook of Combinatorial Optimization 4*. Kluwer Academic, 1999.

[Ben09] Hicham Bensoudane. *Etude différentielle des formes fractales*. PhD thesis, 2009. Thèse de doctorat dirigée par Neveu, Marc Informatique Dijon 2009.

[BFB97] Miguel A. Blanco, M. Flórez, and M. Bermejo. Evaluation of the rotation matrices in the basis of real spherical harmonics. *Journal of Molecular Structure: {THEOCHEM}*, 419(1–3):19 – 27, 1997.

[BG97] C. Bandt and P. Gummelt. Fractal penrose tilings i. construction and matching rules. *aequationes mathematicae*, 53(1-2):295–307, 1997.

[BGN08] Hicham Bensoudane, Christian Gentil, and Marc Neveu. The local fractional derivative of fractal curves. *Signal-Image Technologies and Internet-Based System, International IEEE Conference on*, 0:422–429, 2008.

[BHS08] M. Barnsley, J. Hutchinson, and O. Stenflo. V-variable fractals: Fractals with partial self similarity. *Advances in Mathematics*, 218(6):2051 – 2088, 2008.

[BJB+11] Harsh Bhatia, Shreeraj Jadhav, Peer-Timo Bremer, Guoning Chen, Joshua A. Levine, Luis Gustavo Nonato, and Valerio Pascucci. Edge maps: Representing flow with bounded error. In *PacificVis*, pages 75–82. IEEE, 2011.

[Bon48] Pierre Ossian Bonnet. *Journ. École Polytechnique*, 19:1–146, 1848.

[BRM+14] TristanCarrier Baudouin, Jean-François Remacle, Emilie Marchandise, François Henrotte, and Christophe Geuzaine. A frontal approach to hex-dominant mesh generation. *Advanced Modeling and Simulation in Engineering Sciences*, 1(1), 2014.

[BV13] Michael Barnsley and Andrew Vince. Fractal homeomorphism for bi-affine iterated function systems. *Int. J. Applied Nonlinear Science*, 1(1):3–19, 2013.

[BZK09] David Bommes, Henrik Zimmer, and Leif Kobbelt. Mixed-integer quadrangulation. *ACM Trans. Graph.*, 28(3):77:1–77:10, July 2009.

[CdVL05] Éric Colin de Verdière and Francis Lazarus. Optimal system of loops on an orientable surface. *Discrete & Computational Geometry*, 33(3):507–534, March 2005.

[CIGR99] Cheol Ho Choi, Joseph Ivanic, Mark S. Gordon, and Klaus Ruedenberg. Rapid and stable determination of rotation matrices between spherical harmonics by direct recursion. *The Journal of Chemical Physics*, 111(19):8825–8831, 1999.

[Coh97] N. Cohen. Fractal antenna applications in wireless telecommunications. In *Electronics Industries Forum of New England, 1997. Professional Program Proceedings*, pages 43–49, May 1997.

[Coo71] S. A. Cook. The complexity of theorem-proving procedures. In *Proc. 3rd ACM Symposium on Theory of Computing*, pages 151–158, 1971.

[CRL+89] José Ramón Alvarez Collado, Jaime Fernández Rico, Rafael López, Miguel Paniagua, and Guillermo Ramírez. Rotation of real spherical harmonics. *Computer Physics Communications*, 52(3):323 – 331, 1989.

[DBG+06] Shen Dong, Peer-Timo Bremer, Michael Garland, Valerio Pascucci, and John C. Hart. Spectral surface quadrangulation. *ACM Trans. Graph.*, 25(3):1057–1066, 2006.

[DL92] Ingrid Daubechies and Jeffrey C. Lagarias. Sets of matrices all infinite products of which converge. *Linear Algebra and its Applications*, 161:227 – 263, 1992.

[Ebe98] David Eberly. Triangulation by ear clipping. *Geometric Tools*, 1998.

[EM90] Herbert Edelsbrunner and Ernst Peter Mücke. Simulation of simplicity. *ACM TRANS. GRAPH*, 9(1), 1990.

[EM10] David Eppstein and Elena Mumford. Steinitz theorems for orthogonal polyhedra. In *Proceedings of the Twenty-sixth Annual Symposium on Computational Geometry*, SoCG '10, pages 429–438, New York, NY, USA, 2010. ACM.

[Fal90] H.J. Falconer. *Fractal geometry : mathematical foundations ands applications*. Wiley, 1990. 2nd edition.

[Fer77] N.M. Ferrers. *An Elementary Treatise on Spherical Harmonics and Subjects Connected with Them*. Macmillan and Company, 1877.

[FSDH07] Matthew Fisher, Peter Schröder, Mathieu Desbrun, and Hugues Hoppe. Design of tangent vector fields. *ACM Trans. Graph*, 26:56, 2007.

[Gau00] Carl Friedrich Gauß. *Werke*, volume 8. Gesellschaft Wissenschaft, Göttingen, 1900.

[Gen92] C. Gentil. *Les fractales en synthése d'images: le modèle IFS*. PhD thesis, Université LYON I, March 1992. Jury: D. Vandorpe, P. Chenin, J. Mazoyer, J. P. Reveilles, J. Levy Vehel, M. Terrenoire, E. Tosan.

[GHS90] Paul-Louis George, Frédéric Hecht, and E. Saltel. Fully automatic mesh generator for 3d domains of any shape. *IMPACT Comput. Sci. Eng.*, 2(3):187–218, 1990.

[GSZ11] J. Gregson, A. Sheffer, and E. Zhang. All-hex mesh generation via volumetric polycube deformation. *Computer Graphics Forum (Special Issue of Symposium on Geometry Processing 2011)*, 30(5), 2011.

[GTB00] E. Guerin, E. Tosan, and A Baskurt. Fractal coding of shapes based on a projected ifs model. In *Image Processing, 2000. Proceedings. 2000 International Conference on*, volume 2, pages 203–206 vol.2, Sept 2000.

[GWB96] Christiane Görller-Walrand and Koen Binnemans. Rationalization of crystal-field parametrization. *status: published*, 1996.

[HF06] Kai Hormann and Michael S. Floater. Mean value coordinates for arbitrary planar polygons. *ACM Trans. Graph.*, 25(4):1424–1441, October 2006.

[HJS+14] Jin Huang, Tengfei Jiang, Zeyun Shi, Yiying Tong, Hujun Bao, and Mathieu Desbrun. L1 based construction of polycube maps from complex shapes. *ACM Trans. Graph.*, 33(3):25:1–25:11, June 2014.

[HTWB11] Jin Huang, Yiying Tong, Hongyu Wei, and Hujun Bao. Boundary aligned smooth 3d cross-frame field. *ACM Trans. Graph.*, 30(6):143:1–143:8, December 2011.

[Hut81] J. Hutchinson. Fractals and self-similarity. *Indiana University Journal of Mathematics*, 30:713 – 747, 1981.

[HWFQ09] Ying He, Hongyu Wang, Chi-Wing Fu, and Hong Qin. A divide-and-conquer approach for automatic polycube map construction. *Computers & Graphics*, 33(3):369 – 380, 2009. IEEE International Conference on Shape Modelling and Applications 2009.

[HXH10] Shuchu Han, Jiazhi Xia, and Ying He. Hexahedral shell mesh construction via volumetric polycube map. In *Proceedings of the 14th ACM Symposium on Solid and Physical Modeling*, SPM '10, pages 127–136, New York, NY, USA, 2010. ACM.

[IR96] Joseph Ivanic and Klaus Ruedenberg. Rotation matrices for real spherical harmonics. direct determination by recursion. *The Journal of Physical Chemistry*, 100(15):6342–6347, 1996.

[JHW+14] Tengfei Jiang, Jin Huang, Yuanzhen Wang, Yiying Tong, and Hujun Bao. Frame field singularity correctionfor automatic hexahedralization. *IEEE Transactions on Visualization and Computer Graphics*, 20(8):1189–1199, 2014.

[JSW05] Tao Ju, Scott Schaefer, and Joe Warren. Mean value coordinates for closed triangular meshes. *ACM Trans. Graph.*, 24(3):561–566, July 2005.

[KCPS13] Felix Knöppel, Keenan Crane, Ulrich Pinkall, and Peter Schröder. Globally optimal direction fields. *ACM Trans. Graph.*, 32(4), 2013.

[KLF12] Nicolas Kowalski, Franck Ledoux, and Pascal Frey. A PDE based approach to multi-domain partitioning and quadrilateral meshing. 2012.

[KLF13] Nicolas Kowalski, Franck Ledoux, and Pascal Frey. A pde based approach to multidomain partitioning and quadrilateral meshing. In *Proceedings of the 21st International Meshing Roundtable*, pages 137–154. Springer Berlin Heidelberg, 2013.

[KNP07] Felix Kälberer, Matthias Nieser, and Konrad Polthier. Quadcover - surface parameterization using branched coverings. *Comput. Graph. Forum*, 26(3):375–384, 2007.

[Liu68] C.L. Liu. *Introduction to Combinatorial Mathematics*. Mc-Graw Hill, 1968.

[LJFW08] Juncong Lin, Xiaogang Jin, Zhengwen Fan, and CharlieC.L. Wang. Automatic polycube-maps. In Falai Chen and Bert Jüttler, editors, *Advances in Geometric Modeling and Processing*, volume 4975 of *Lecture Notes in Computer Science*, pages 3–16. Springer Berlin Heidelberg, 2008.

[LL10] Bruno Lévy and Yang Liu. Lp centroidal voronoi tesselation and its applications. *ACM Transactions on Graphics (SIGGRAPH conference proceedings)*, 2010. patent pending - FR 10/02920 (filed 07/09/10).

[LLP05a] Bruno Lévy, Wan Chiu Li, and Jean-Claude Paul. Mesh Editing with an Embedded Network of Curves. In *IEEE International Conference on Shape Modeling and Applications - SMI 2005*, Shape Modeling and Applications, 2005 International Conference, pages 62–71, Cambridge, USA, June 2005. IEEE. http://ieeexplore.ieee.org.

[LLP05b] Wan Chiu Li, Bruno Lévy, and Jean-Claude Paul. Mesh editing with an embedded network of curves. In *IEEE International Conference on Shape Modeling and Applications*, pages 62–71, 2005.

[LLX+12] Yufei Li, Yang Liu, Weiwei Xu, Wenping Wang, and Baining Guo. All-hex meshing using singularity-restricted field. *ACM Trans. Graph.*, 31(6):177:1–177:11, November 2012.

[LVS+13] Marco Livesu, Nicholas Vining, Alla Sheffer, James Gregson, and Riccardo Scateni. Polycut: Monotone graph-cuts for polycube base-complex construction. *Transactions on Graphics (Proc. SIGGRAPH ASIA 2013)*, 32(6), 2013.

[Lé] Bruno Lévy. OpenNL, Open Numerical Library. http://alice.loria.fr/index.php/software/4-library/23-opennl.html.

[Lé15] Bruno Lévy. GEOGRAM, a programming library of geometric algorithms, 2015. http://alice.loria.fr/software/geogram/doc/html/index.html.

[Mas97] P. R. Massopust. Fractal functions and their applications. *Chaos, Solitons & Fractals*, 8(2):171 – 190, 1997.

[Mis13] Anton Mishkinis. *Extension des méthodes de géométrie algorithmique aux structures fractales*. PhD thesis, 2013. Thèse de doctorat dirigée par Gentil, Christian et Lanquetin, Sandrine Informatique Dijon 2013.

[Moa09] M. Moakher. *Visualization and Processing of Tensor Fields : Advances and Perspectives*. D. Laidlaw and J. Weickert, eds., Springer, 2009.

[MP08] Meyer and Pion. FPG: A code generator. In *Real Numbers and Computers*, pages 47–60, 2008.

[MPKZ10] Ashish Myles, Nico Pietroni, Denis Kovacs, and Denis Zorin. Feature-aligned t-meshes. In *ACM SIGGRAPH 2010*, pages 117:1–117:11, 2010.

[MPZ14] Ashish Myles, Nico Pietroni, and Denis Zorin. Robust field-aligned global parametrization. *ACM Trans. Graph.*, 33(4):135:1–135:14, July 2014.

[Mro95] Marian Mrozek. Conley index theory. In *Handbook of Dynamical Systems II, North-Holland*, pages 393–460. Elsevier, 1995.

[MT00] Sia Meshkat and Dafna Talmor. Generating a mixed mesh of hexahedra, pentahedra and tetrahedra from an underlying tetrahedral mesh. *International Journal for Numerical Methods in Engineering*, 49(1-2):17–30, 2000.

[MTTT98] Ray J. Meyers, Timothy J. Tautges, Philip M. Tuchinsky, and Dr. Philip M. Tuchinsky. The "hex-tet" hex-dominant meshing algorithm as implemented in cubit. In *in CUBIT; Proceedings, 7 th International Meshing Roundtable 98*, pages 151–158, 1998.

[MVC05] Dimas Martínez, Luiz Velho, and Paulo C. Carvalho. Computing geodesics on triangular meshes. *Comput. Graph.*, 29(5):667–675, October 2005.

[MW88] R. D. Mauldin and S. C. Williams. Hausdorff dimension in graph directed constructions. *Transactions of the American Mathematical Society*, 309(2):811 – 829, 1988.

[NRP11] Matthias Nieser, Ulrich Reitebuch, and Konrad Polthier. CubeCover - Parameterization of 3D Volumes. *Computer Graphics Forum*, 30(5):1397–1406, 2011.

[Pen10] Deborah Pence. The simplicity of fractal-like flow networks for effective heat and mass transport. *Experimental Thermal and Fluid Science*, 34(4):474 – 486, 2010. {ECI} International Conference on Heat Transfer and Fluid Flow in Microscale.

[PGSL14] Sergey Podkorytov, Christian Gentil, Dmitry Sokolov, and Sandrine Lanquetin. Joining primal/dual subdivision surfaces. In Michael Floater, Tom Lyche, Marie-Laurence Mazure, Knut Mørken, and LarryL. Schumaker, editors, *Mathematical Methods for Curves and Surfaces*, volume 8177 of *Lecture Notes in Computer Science*, pages 403–424. Springer Berlin Heidelberg, 2014.

[PH94] P. Prusinkiewicz and M. Hammel. Language-restricted iterated function systems, koch constructions, and l-systems. *SIGGRAPH'94 Course Notes*, 1994.

[PL90] P. Prusinkiewicz and Aristid Lindenmayer. *The Algorithmic Beauty of Plants*. Springer-Verlag New York, Inc., New York, NY, USA, 1990.

[Pod13] Sergey Podkorytov. *Espaces tangents pour les formes auto-similaires*. PhD thesis, 2013. Thèse de doctorat dirigée par Gentil, Christian et Sokolov, Dmitry Informatique Dijon 2013.

[PR86] Heinz-Otto Peitgen and Peter Richter. *The beauty of fractals : images of complex dynamical systems*. Springer-Verlag, Heidelberg, 1986.

[Pra98] Hartmut Prautzsch. Smoothness of subdivision surfaces at extraordinary points. *Advances in Computational Mathematics*, 9:377–389, 1998. 10.1023/A:1018945708536.

[PRP+96] C. Puente, J. Romeu, R. Pous, X. Garcia, and F. Benitez. Fractal multiband antenna based on the sierpinski gasket. *Electronics Letters*, 32(1):1–2, Jan 1996.

[PS06] Konrad Polthier and Markus Schmies. Straightest geodesics on polyhedral surfaces. In *ACM SIGGRAPH 2006 Courses*, SIGGRAPH '06, pages 30–38, New York, NY, USA, 2006. ACM.

[PZ07] Jonathan Palacios and Eugene Zhang. Rotational symmetry field design on surfaces. *ACM Trans. Graph.*, 26(3), July 2007.

[Rei93] Ulrich Reif. *Neue Aspekte in der Theorie der Freiformflachen beliebiger Topologie.* PhD thesis, Mathematisches Institut A der Universitat Stuttgart, 1993.

[Rei95] Ulrich Reif. A unified approach to subdivision algorithms near extraordinary vertices. *Computer Aided Geometric Design*, 12(2):153 – 174, 1995.

[RLL+06] Nicolas Ray, Wan Chiu Li, Bruno Lévy, Alla Sheffer, and Pierre Alliez. Periodic global parameterization. *ACM Trans. Graph.*, 25(4):1460–1485, October 2006.

[RS14a] Nicolas Ray and Dmitry Sokolov. Robust polylines tracing for n-symmetry direction field on triangulated surfaces. *ACM Trans. Graph.*, 33(3):30:1–30:11, June 2014.

[RS14b] Iasef Md Rian and Mario Sassone. Tree-inspired dendriforms and fractal-like branching structures in architecture: A brief historical overview. *Frontiers of Architectural Research*, 3(3):298 – 323, 2014.

[RT12] C. Rossl and H. Theisel. Streamline embedding for 3d vector field exploration. *IEEE Transactions on Visualization and Computer Graphics*, 18(3):407 –420, march 2012.

[RVAL09] Nicolas Ray, Bruno Vallet, Laurent Alonso, and Bruno Levy. Geometry-aware direction field processing. *ACM Trans. Graph.*, 29(1):1:1–1:11, December 2009.

[RVLL06] Nicolas Ray, Bruno Vallet, Wan-Chiu Li, and Bruno Lévy. N-symmetry direction fields on surfaces of arbitrary genus. Technical report, INRIA - ALICE, 2006.

[RVLL08] Nicolas Ray, Bruno Vallet, Wan Chiu Li, and Bruno Lévy. N-symmetry direction field design. *ACM Trans. Graph.*, 27(2):10:1–10:13, May 2008.

[SGGM15] Dmitry Sokolov, Gilles Gouaty, Christian Gentil, and Anton Mishkinis. Boundary controlled iterated function systems. In Jean-Daniel Boissonnat, Albert Cohen, Olivier Gibaru, Christian Gout, Tom Lyche, Marie-Laurence Mazure, and Larry L. Schumaker, editors, *Curves and Surfaces*, volume 9213 of *Lecture Notes in Computer Science*, pages 414–432. Springer International Publishing, 2015.

[She97] Shewchuk. Adaptive precision floating-point arithmetic. *Discrete & Computational Geometry*, 18(3), 1997.

[Si15] Hang Si. Tetgen, a delaunay-based quality tetrahedral mesh generator. *ACM Trans. Math. Softw.*, 41(2):11:1–11:36, February 2015.

[SJ08] Jason F. Shepherd and Chris R. Johnson. Hexahedral mesh generation constraints. *Eng. with Comput.*, 24(3):195–213, June 2008.

[SLCZ09] Benjamin Spencer, Robert S. Laramee, Guoning Chen, and Eugene Zhang. Evenly spaced streamlines for surfaces: An image-based approach. *Comput. Graph. Forum*, 28(6):1618–1631, 2009.

[SSK+05] Vitaly Surazhsky, Tatiana Surazhsky, Danil Kirsanov, Steven J. Gortler, and Hugues Hoppe. Fast exact and approximate geodesics on meshes. *ACM Trans. Graph.*, 24(3):553–560, July 2005.

[SYC06] S.C. Soo, K.M. Yu, and W.K. Chiu. Modeling and fabrication of artistic products based on {IFS} fractal representation. *Computer-Aided Design*, 38(7):755 – 769, 2006.

[SZ12] Andrzej Szymczak and Eugene Zhang. Robust morse decompositions of piecewise constant vector fields. *IEEE Transactions on Visualization and Computer Graphics*, 18:938–951, 2012.

[SZSS98] Thomas W. Sederberg, Jianmin Zheng, David Sewell, and Malcolm Sabin. Non-uniform recursive subdivision surfaces. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 387–394, New York, NY, USA, 1998. ACM.

[TACSD06] Y. Tong, P. Alliez, D. Cohen-Steiner, and M. Desbrun. Designing quadrangulations with discrete harmonic forms. In *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*, SGP '06, pages 201–210, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[TGM+09] Olivier Terraz, Guillaume Guimberteau, Stéphane Mérillou, Dimitri Plemenos, and Djamchid Ghazanfarpour. 3gmap l-systems: an application to the modelling of wood. *The Visual Computer*, 25(2):165–180, 2009.

[THCM04] Marco Tarini, Kai Hormann, Paolo Cignoni, and Claudio Montani. Polycube-maps. *ACM Trans. Graph.*, 23(3):853–860, August 2004.

[Tos06] E. Tosan. Surfaces fractales définies par leurs bords. Grenoble, 2006. Journées Courbes, surfaces et algorithmes.

[TT93] J. Thollot and E. Tosan. Construction of fractales using formal languages and matrices of attractors. In Harold P. Santos, editor, *Conference on Computational Graphics and Visualization Techniques, Compugraphics*, pages 74 – 78, Alvor, Portugal, 1993.

[WWT+06] Ke Wang, Weiwei, Yiying Tong, Mathieu Desbrun, and Peter Schröder. Edge subdivision schemes and the construction of smooth vector fields. *ACM Trans. Graph.*, 25(3):1041–1048, July 2006.

[WYZ+11] Shenghua Wan, Zhao Yin, Kang Zhang, Hongchao Zhang, and Xin Li. Smi 2011: Full paper: A topology-preserving optimization algorithm for polycube mapping. *Comput. Graph.*, 35(3):639–649, June 2011.

[YLL+09] Dongming Yan, Bruno Lévy, Yang Liu, Feng Sun, and Wenping Wang. Isotropic remeshing with fast and exact computation of restricted voronoi diagram. In *ACM/EG Symposium on Geometry Processing / Computer Graphics Forum*, 2009.

[YS03] Soji Yamakawa and Kenji Shimada. Fully-automated hex-dominant mesh generation with directionality control via packing rectangular solid cells. *International Journal for Numerical Methods in Engineering*, 57(15):2099–2129, 2003.

[ZMT06] Eugene Zhang, Konstantin Mischaikow, and Greg Turk. Vector field design on surfaces. *ACM Trans. Graph.*, 25(4):1294–1326, October 2006.

[ZSTR07] Howard Zhou, Jie Sun, Greg Turk, and James M. Rehg. Terrain synthesis from digital elevation models. *IEEE Transactions on Visualization and Computer Graphics*, 13(4):834–848, July/August 2007.

[ZT96] Chems Eddine Zair and Eric Tosan. Fractal modeling using free form techniques. *Computer Graphics Forum*, 15(3):269–278, 1996.