

Master 1 SIS IN, IM9 - Modélisation géométrique, discret ↔ continu

Romain Raffin

Master 1 SIS - Image
romain.raffin[at]univ-amu.fr

v2016

CM/TP disponibles sur ametice.univ-amu.fr

Sommaire

- 1** Introduction
- 2** Courbes
 - Paramétrisation
 - Courbes de Bézier
 - Polynômes de Bernstein
 - Interpolation de données
 - (re-)Paramétrisation
- 3** Courbes de Subdivision
 - Chaikin & Co
- 4** Surfaces de Subdivision
 - Introduction
 - Exemples de surfaces
 - Des outils pour les subdivisions
 - Subdivision adaptative
 - Subdivision Inverse
- 5** Maillages
 - Remeshing
 - Conversion
 - Des critères sur les maillages
- 6** OpenCL avancé

Sommaire

1 Introduction

2 Courbes

3 Courbes de Subdivision

4 Surfaces de Subdivision

5 Maillages

6 OpenGL avancé

1 Introduction

Planning

- Cours1 / TP1
- Cours2 / TP2
- Cours3 / TP3
- Cours4 / TP4
- Cours5 / TP5, contrôle théorique (sur feuille) le matin (cours, culture et cas pratique), ramassage du projet l'après-midi

Sommaire

1 Introduction

2 Courbes

3 Courbes de Subdivision

4 Surfaces de Subdivision

5 Maillages

6 OpenGL avancé

2 Courbes

- Paramétrisation
- Courbes de Bézier
- Polynômes de Bernstein
- Interpolation de données
- (re-)Paramétrisation

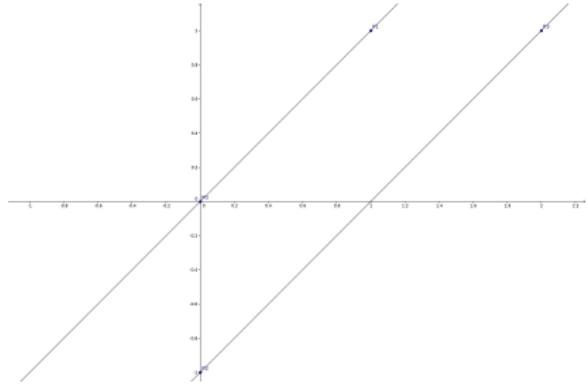
La droite ...

Comment décrire la droite passant par P_0 et P_1 ?

- 1^{ère} expression : $\mathcal{D}(t) = P_0 + t\overrightarrow{P_0P_1}$
- 2e expression (équivalente) : $\mathcal{D}(t) = (1 - t)P_0 + tP_1$

À faire évoluer entre $t = -3$ et $t = 3$ pour comparer la facilité de compréhension des 2 écritures ...

Comment les points sur la droite évoluent-ils ? A priori à vitesse constante, mais pourquoi ?



1^{ère} droite : $d_0(t)$

Points $P_0(0, 0)$ et $P_1(1, 1)$

$$d_0(t=0) = P_0, \quad d_0(t=1) = P_1$$

$$d(P_0, P_1) = \|\overrightarrow{P_0P_1}\| = \sqrt{2}$$

2^e droite : $d_1(t)$

Points $P_2(0, -1)$ et $P_3(2, 1)$

$$d_1(t=0) = P_2, \quad d_1(t=1) = P_3$$

$$d(P_2, P_3) = \|P_2 P_3'\| = 2\sqrt{2}$$

On voit donc que pour la même droite, la vitesse de parcours n'est pas forcément la même selon la construction

Des souvenirs ...

On peut réutiliser les notions vues en lycée :

$$vitesse = \frac{distance}{temps}$$

Pour nous, si t est le temps, un segment est l'élément de base qui a un intervalle de longueur 1. On note aussi dans la formule que pour un temps identique si la distance augmente, la vitesse augmente (et vice-versa).

2 constructions d'une même droite

Comment dans ce cas avoir 2 droites \mathcal{D}_0 , passant par les points P_0 et P_1 et \mathcal{D}_1 , passant par les points P_2 et P_3 , telles que leur $|P_0P_1| = 2|P_2P_3|$ qui évoluent avec la même vitesse ?

On peut choisir sa paramétrisation

La vitesse peut donc ne pas être constante, elle peut varier selon les résultats d'une fonction. Mais elle oblige la modification de t par exemple :

- $k(t) : t \in [0, \frac{1}{2}] : v_0$ et $t \in [\frac{1}{2}, 1] : v_1 = 2v_0$,
- $k(t) = a_0 t^2 + b_0$ et la vitesse varie quadratiquement,
- $k(t) = \cos(2\pi t)$ et la vitesse accélère et décélère périodiquement,
- jusqu'à l'amortissement ou le recul¹

Note : la dérivée seconde (l'accélération) nous donne la variation de vitesse.

1. testez le slider de votre smartphone...

Paramétrisation d'un cercle

Il reste que le choix du système de coordonnées pour la paramétrisation est important. Par exemple, pour un cercle on a :

$$x^2 + y^2 = R^2 \quad \forall x, y \in \mathbb{R} \times \mathbb{R}$$

ou :

$$x(t) = R \frac{1 - t^2}{1 + t^2} \text{ et } y(t) = R \frac{2t}{1 + t^2} \text{ avec } t \in \mathbb{R}$$

(par changement de variable²), ou encore :

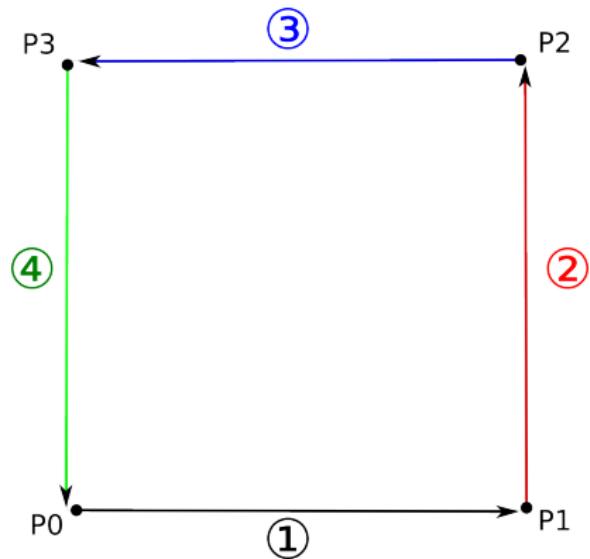
$$\begin{cases} x(t) = R \cos(t * \theta) \\ y(t) = R \sin(t * \theta) \\ \text{avec } \theta = 2\pi \text{ et } t \in [0, 1] \end{cases}$$

2. pour revenir à une expression angulaire, $\theta = 2 \arctan t$

Paramétrisation plus dure ...

Et encore, comment faire pour un carré ?

- suite de segment
- polyligne
- cordonnées circulaires ?



On s'approche des courbes

Essayons maintenant avec des courbes de Bézier ...

Rappel

la formule générale d'une courbe de Bézier s'écrit :

$$C(t) = \sum_{i=0}^n \mathcal{B}_{i,n}(t) P_i \text{ avec } t \in [0, 1] \quad (1)$$

Dans cette formule :

- $C(t)$ est la courbe paramétrique de Bézier,
- n est le *degré* de la courbe,
- $\mathcal{B}_{i,n}(t)$ sont des *polynômes de Bernstein*,
- P_i sont les *points de contrôle*,
- t est le *paramètre* de la courbe.

Le degré n de la courbe est égal au nombre de points de contrôle moins 1.

Polynômes de Bernstein

Les $\mathcal{B}_{i,n}(t)$ sont des polynômes de Bernstein. Ils permettent de calculer le poids de chaque point de contrôle, ils se calculent par :

$$\mathcal{B}_{(i,n)} = \frac{n!}{i!(n-i)!} t^i (1-t)^{n-i}, \text{ pour } t \in [0, 1] \quad (2)$$

Exemple de courbe de Bézier

Pour le degré 3, on utilise 4 points de contrôle (P_0 , P_1 , P_2 et P_3). La courbe $C(t)$ s'écrit alors :

$$C(t) = (1 - t)^3 P_0 + 3t(1 - t)^2 P_1 + 3t^2(1 - t)P_2 + t^3 P_3 \text{ avec } t \in [0, 1]$$

Exemple de courbe de Bézier

Utiliser les points $P_0(0, 5)$, $P_1(10, 5)$, $P_2(1, 4)$, $P_3(5, 0)$ **dans cet ordre** comme polygone de contrôle et :

- donner l'expression paramétrique de la courbe $C(t)$ associée,
- calculer les dérivées des fonctions de coordonnées $x(t)$ et $y(t)$,
- utiliser la dérivée des polynômes de Bernstein pour la question précédente,
- calculer vitesse et accélération de la courbe

Rappels divers

- dérivée d'un polynôme de Bernstein :

$$\mathcal{B}'_{i,n}(t) = n(\mathcal{B}_{i-1,n-1}(t) - \mathcal{B}_{i,n-1}(t))$$

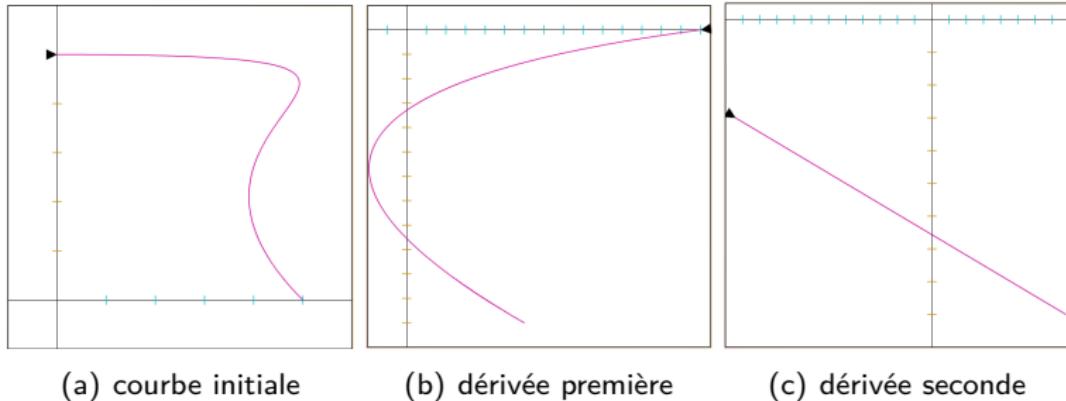
- Binôme de Newton $(x+y)^n = \sum_{k=0}^n \binom{n}{k} x^{n-k} y^k$, avec

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

- Dérivées usuelles :

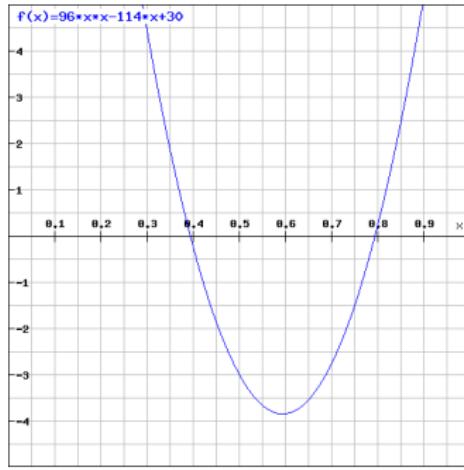
- $(u+v)' = u'v + uv'$
- $(u^n)' = nu^{n-1}u'$

Résultats1

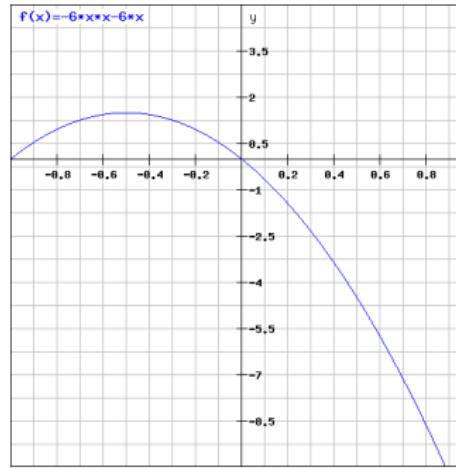


Résultats2

Attention, les coordonnées n'évoluent pas de la même manière :



(d) courbe de la vitesse en X, $x'(t)$



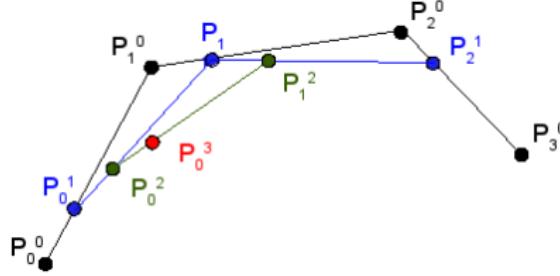
(e) courbe de la vitesse en Y, $y'(t)$

Algorithme de Casteljau

Permet de définir une courbe de Bézier par raffinements successifs. On va chercher l'évaluation pour un paramètre t_0 , en combinant des interpolations linéaires. La formule générale est :

$$\begin{cases} P_i^0 = P_i \\ P_i^k = P_i^{k-1}(1 - t_0) + P_{i+1}^{k-1}(t_0) \\ \quad i = 0 \dots n \end{cases}$$

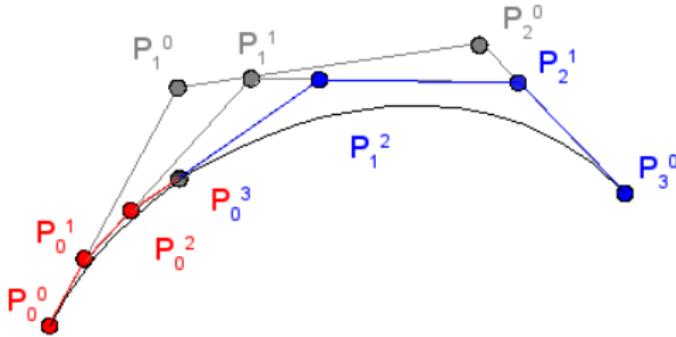
et finalement $C(t_0) = P_0^n$.



Conséquences de De Casteljau

On peut subdiviser la courbe de Bézier en t_0 par 2 sous-courbes constituées des polygones de contrôles :

$$\begin{cases} P_0^0 P_0^1 & \dots P_0^n \\ P_0^n P_1^{n-1} & \dots P_n^0 \end{cases}$$



Remarque : on voit apparaître le passage discret-continu.

Et l'élévation de degré ?

Une courbe de Bézier peut facilement être élevée à un degré supérieur.

En partant de :

$$C(t) = \sum_{i=0}^n B_{i,n}(t) P_i$$

On peut déduire une courbe :

$$D(t) = \sum_{i=0}^{n+1} B_{i,n+1}(t) Q_i$$

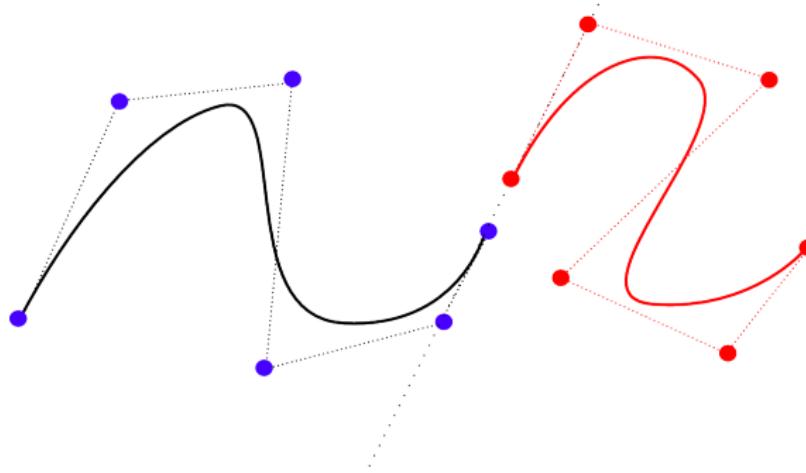
avec :

$$Q_i = \frac{i}{n+1} P_{i-1} + \left(1 - \frac{i}{n+1}\right) P_i$$

C'est pratique pour l'interpolation de données. Le contraire est plus délicat ...

Et le raccord de courbes ?

- gestion de la continuité
 - gestion des intervalles paramétriques
 - il peut y avoir un paramètre unique, pas une courbe de mêmes points de contrôles !



On souhaite calculer la courbe $Q(t)$ qui passe par certains points M_0, \dots, M_k de l'espace. Comment faire ?

Essayons par exemple de faire passer une courbe de Bézier par 4 points M_0, M_1, M_2, M_3 . On sait que la courbe de Bézier interpole les extrémités du polygone de contrôle, donc si les points M_0 et M_3 sont les extrêmes (cela dépend d'un sens de parcours donné en hypothèse), alors ces 2 points sont les 2 points de contrôle extrêmes. On a donc, en faisant l'hypothèse d'une courbe de degré 3 ...

$$\begin{cases} P_0 = M_0 \\ P_3 = M_3 \end{cases}$$

Il nous reste donc les points P_1 et P_2 à calculer. C'est le cœur du problème car cela pose le problème de la paramétrisation. En effet, on ne peut deviner à quels valeurs de paramètres t_1 et t_2 correspondent les points M_1 et M_2 . Comment fait-on ? On « invente » tout simplement. On peut, par exemple, penser que le domaine paramétrique est divisé en 3 intervalles équilibrés ($t_1 = 1/3$ et $t_2 = 2/3$).

$$\begin{cases} Q(t_0 = 0) &= P_0 = M_0 \\ Q(t_1 = 1/3) &= M_1 \\ Q(t_2 = 2/3) &= M_2 \\ Q(t_3 = 1) &= P_3 = M_3 \end{cases}$$

Rappel :

$$C(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3 \text{ avec } t \in [0, 1]$$

Paramétrisation

Cet exemple repose donc le problème de la paramétrisation. Comment faire pour que les valeurs de paramètres choisis permettent d'avoir une vitesse constante, et/ou une « bonne » répartition des points de discrétisation sur la courbe ?

Quelques méthodes sont possibles :

- uniforme,
- par longueur de corde,
- par méthode centripète,
- par méthode logarithmique.

Paramétrisation uniforme

C'est la façon la plus simple de choisir les paramètres :

$$u_0 = 0$$

$$u_n = 1$$

$$u_k = \frac{k}{n} \text{ pour } k = 1 \dots n - 1$$

Les résultats sont assez mauvais, puisque les données ne sont pas généralement réparties de manière uniforme.

Paramétrisation par longueur de corde

On utilise la longueur de la courbe pour construire un ensemble de paramètres. Soit d la longueur de corde totale :

$$d = \sum_{i=1}^n |P_i - P_{i-1}|$$

La paramétrisation se fait sur cette longueur :

$$u_0 = 0$$

$$u_n = 1$$

$$u_k = u_{k-1} + \frac{|P_k - P_{k-1}|}{d} \text{ pour } k = 1 \dots n-1$$

la longueur de la courbe peut être calculée par une intégrale, ou par la longueur des segments de droite reliant les points de contrôle ou bien encore par paramétrisation uniforme (éventuellement en plusieurs passes).

Paramétrisation centripète

On utilise une racine carrée de la distance calculée précédemment, ce qui la rend moins sensible :

$$d = \sum_{i=1}^n \sqrt{|P_i - P_{i-1}|}$$

La paramétrisation se fait sur cette longueur :

$$u_0 = 0$$

$$u_n = 1$$

$$u_k = u_{k-1} + \frac{\sqrt{|P_k - P_{k-1}|}}{d} \text{ pour } k = 1 \dots n-1$$

Paramétrisation logarithmique

Dans le même ordre d'idée, on peut utiliser le logarithme népérien de l'expression de longueur :

$$d = \sum_{i=1}^n \log(1 + |P_i - P_{i-1}|)$$

La paramétrisation se fait sur cette somme :

$$u_0 = 0$$

$$u_n = 1$$

$$u_k = u_{k-1} + \frac{\log(1 + |P_k - P_{k-1}|)}{d} \text{ pour } k = 1 \dots n-1$$

Est-ce suffisant pour le passage discret - continu ?

La construction de De Casteljau permet de passer d'un polygone discret à une courbe mais comme cela s'appuie sur une courbe de Bézier le nombre de niveaux est fixe (= degré du polynôme).

On sait que le degré d'une courbe de type Bspline ne dépend plus du nombre de points de contrôle ...

Sommaire

1 Introduction

2 Courbes

3 Courbes de Subdivision

4 Surfaces de Subdivision

5 Maillages

6 OpenGL avancé

3 Courbes de Subdivision

■ Chaikin & Co

Algorithme de Chaikin

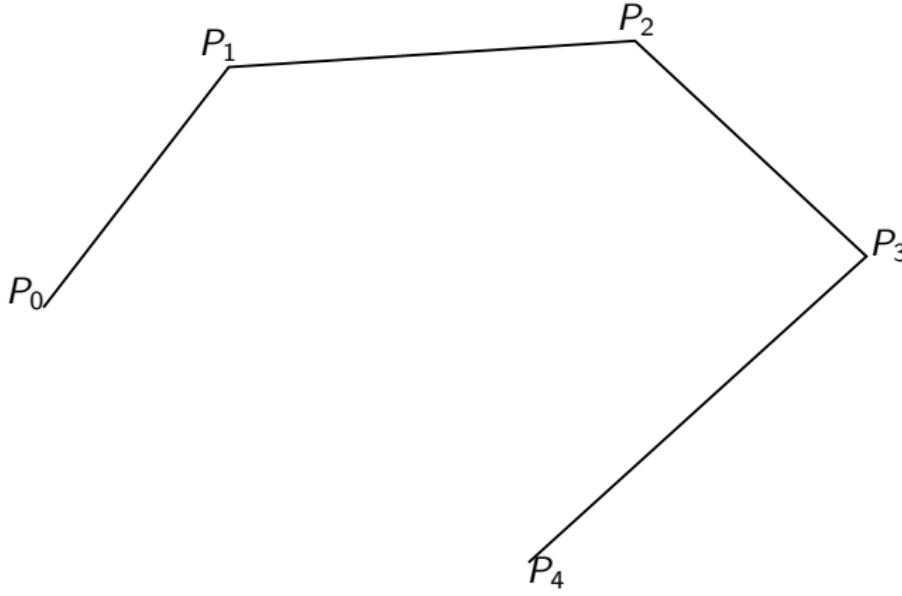
On cherche donc à créer une courbe paramétrique par morcelage successif de son polygone de contrôle. Chaikin et le « *Corner Cutting* » sont une première piste. La construction de la courbe est simple :

- 1** on prend le polygone de contrôle,
- 2** on découpe chaque arête en 3 parties (par insertion de sommets), qui créent des arêtes supplémentaires
- 3** on recommence ...

Il faut ensuite juste donner la façon de couper les 3 parties ...

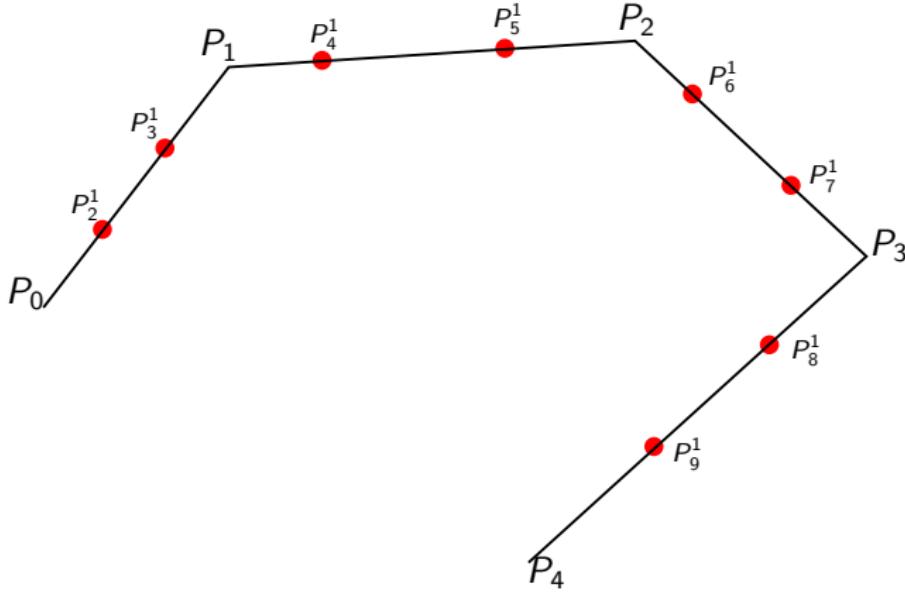
Construction

Chaikin (1974) propose de découper chaque arête en 3 par une moyenne géométrique des sommets avec les coefficients $\frac{1}{4}$ et $\frac{3}{4}$. Par exemple sur la figure :



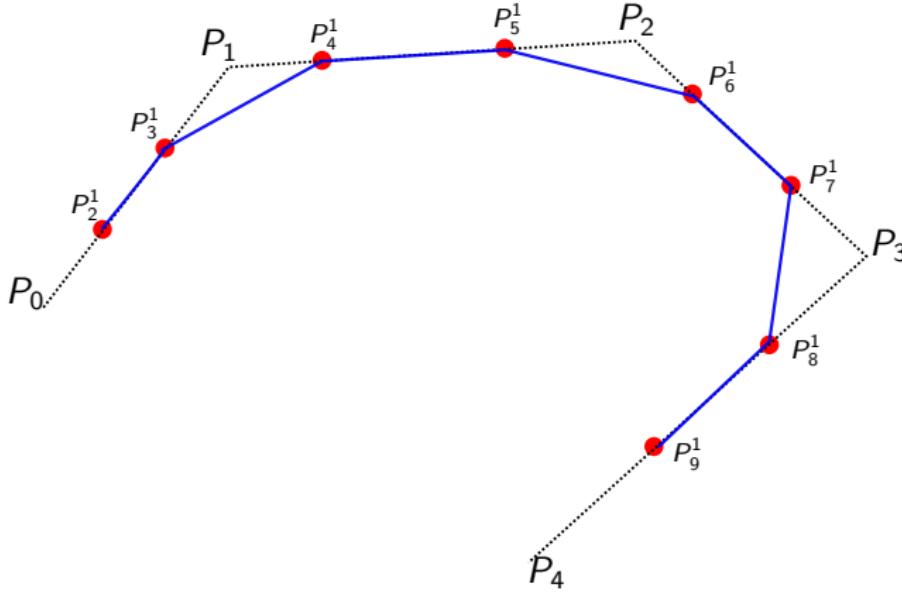
Construction

Chaikin (1974) propose de découper chaque arête en 3 par une moyenne géométrique des sommets avec les coefficients $\frac{1}{4}$ et $\frac{3}{4}$. Par exemple sur la figure :



Construction

Chaikin (1974) propose de découper chaque arête en 3 par une moyenne géométrique des sommets avec les coefficients $\frac{1}{4}$ et $\frac{3}{4}$. Par exemple sur la figure :



coefficients de Chaikin

On construit donc le polygone suivant par :

$$\begin{cases} P_{2i}^k = P_{i-1}^{k-1} + \frac{1}{4} \overrightarrow{P_{i-1}^{k-1} P_i^{k-1}} \\ P_{2i+1}^k = P_{i-1}^{k-1} + \frac{3}{4} \overrightarrow{P_{i-1}^{k-1} P_i^{k-1}} \end{cases}$$

Note : le nombre de points double à chaque subdivision.

coefficients de Chaikin

Ce qui revient à :

$$\begin{cases} P_{2i}^k = \frac{3}{4}P_{i-1}^{k-1} + \frac{1}{4}P_i^{k-1} \\ P_{2i+1}^k = \frac{1}{4}P_{i-1}^{k-1} + \frac{3}{4}P_i^{k-1} \end{cases}$$

Note : le nombre de points double à chaque subdivision.

Masques

On dit que $\begin{pmatrix} \frac{1}{4} & \frac{3}{4} \\ \frac{3}{4} & \frac{1}{4} \end{pmatrix}$ et $\begin{pmatrix} \frac{3}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{3}{4} \end{pmatrix}$ sont deux-sous masques de la subdivision de Chaikin, car :

$$\left\{ \begin{array}{l} P_{2i}^k = \frac{1}{4}P_i^{k-1} + \frac{3}{4}P_{i-1}^{k-1} \\ \text{et} \\ P_{2i+1}^k = \frac{3}{4}P_i^{k-1} + \frac{1}{4}P_{i-1}^{k-1} \end{array} \right.$$

Masques

Si on considère le point P_i^{k-1} il est utilisé par 2 masques. On peut les agréger en un seul pour plus de commodité :

Avec le schéma suivant :

$$P_i^k = \sum_{j \in \mathbb{Z}} a_{i-2j} P_j^{k-1}$$

et le masque (les a_{i-2j}) :

$$\begin{pmatrix} \frac{1}{4} & \frac{3}{4} & \frac{3}{4} & \frac{1}{4} \end{pmatrix}$$

Ça donne quoi ?

La subdivision de Chaikin crée une Bspline quadratique avec un vecteur de nœuds uniformes (démonstration Riesenfeld 1975³). La démonstration montre que le découpage d'une courbe Bspline en 2 sous-courbes recouvrantes utilise le motif de Chaikin. Évidemment si l'on veut une courbe continue il faut effectuer la subdivision de Chaikin un nombre infini de fois !

La Bspline quadratique est la **courbe limite** de la subdivision de Chaikin.

3. <http://www.idav.ucdavis.edu/education/CAGDNotes/Chaikins-Algorithm/Chaikins-Algorithm.html>

Générer d'autres courbes de subdivision

Si l'on veut créer une courbe BSpline C^2 -continue, avec une répartition uniforme des nœuds, le masque de subdivision sera :

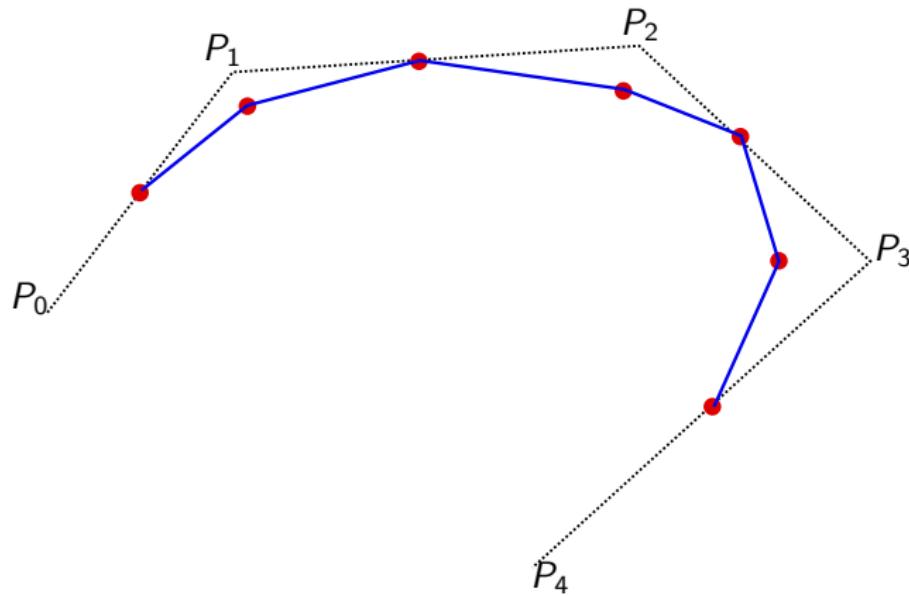
$$(a_0 \quad a_1 \quad a_2 \quad a_3 \quad a_4) = \frac{1}{8} (1 \quad 4 \quad 6 \quad 4 \quad 1)$$

Ce qui se sépare en 2 sous-masques, pour les indices pairs et impairs $(a_0 \quad a_2 \quad a_4)$ et $(a_1 \quad a_3)$:

$$\begin{cases} P_{2i}^k &= \frac{1}{8} (P_i^{k-1} + 6P_{i-1}^{k-1} + P_{i-2}^{k-1}) \\ P_{2i+1}^k &= \frac{1}{2} (P_i^{k-1} + P_{i-1}^{k-1}) \end{cases}$$

Degré 3

La subdivision obtenue sera :



De manière générale

Le masque d'une courbe de subdivision C^d -continue, de degré d , sera :

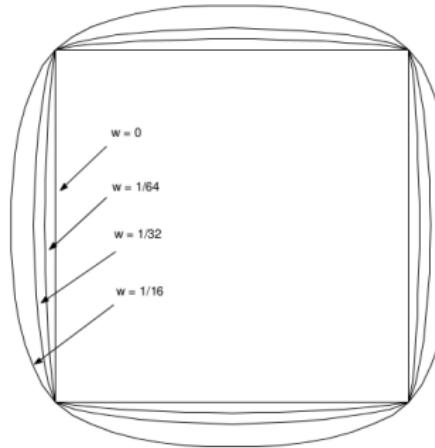
$$(a_0 \quad a_1 \quad \cdots \quad a_{d+1}) = \frac{1}{2^d} \left(\binom{d+1}{0} \quad \binom{d+1}{1} \quad \cdots \quad \binom{d+1}{d+1} \right)$$

Subdivision de Dyn

Dyn (1990) a proposé le schéma suivant :

$$\begin{cases} P_{2i}^k = P_i^{k-1} \\ P_{2i+1}^k = -w(P_{i+2}^{k-1} + P_{i-1}^{k-1}) + \left(\frac{1}{2} + w\right)(P_i^{k-1} + P_{i+1}^{k-1}) \end{cases}$$

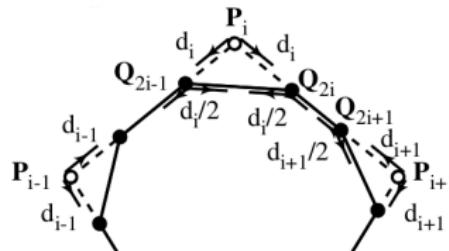
Cela permet une interpolation de points du polygone de contrôle.



Subdivision de Sederberg

À l'instar des courbes de Dyn, les courbes de Sederberg permettent de créer des courbes de subdivision Bspline non-uniforme. Le travail ne se fait pas seulement sur les points de contrôle mais aussi sur les vecteurs de nœuds. On utilise l'intervalle entre 2 nœuds comme paramètre de la subdivision.

$$\begin{cases} Q_{2i} = \frac{(d_i + 2d_{i+1})P_i + d_i P_{i+1}}{2(d_i + d_{i+1})} \\ Q_{2i+1} = \frac{d_{i+1}P_i + (2d_i + d_{i+1})P_{i+1}}{2(d_i + d_{i+1})} \end{cases}$$



Sommaire

1 Introduction

2 Courbes

3 Courbes de Subdivision

4 Surfaces de Subdivision

5 Maillages

6 OpenGL avancé

4 Surfaces de Subdivision

- Introduction
- Exemples de surfaces
- Des outils pour les subdivisions
- Subdivision adaptative
- Subdivision Inverse

Introduction

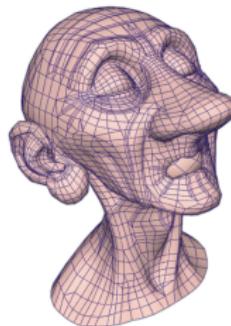
Les courbes de subdivision sont assez peu utilisées, le voisinage étant de même nature, on ne peut faire varier que les coefficients du masque pour créer des courbes de nature différente.

Les surfaces de subdivision sont construites comme des surfaces paramétriques classiques, par produit tensoriel de 2 courbes de subdivision.

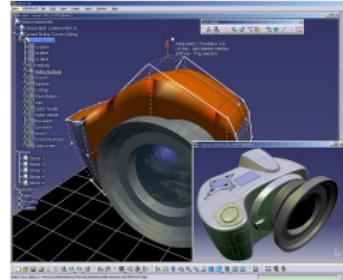
Les exemples de surfaces de subdivision sont nombreux :



(a) Pixar



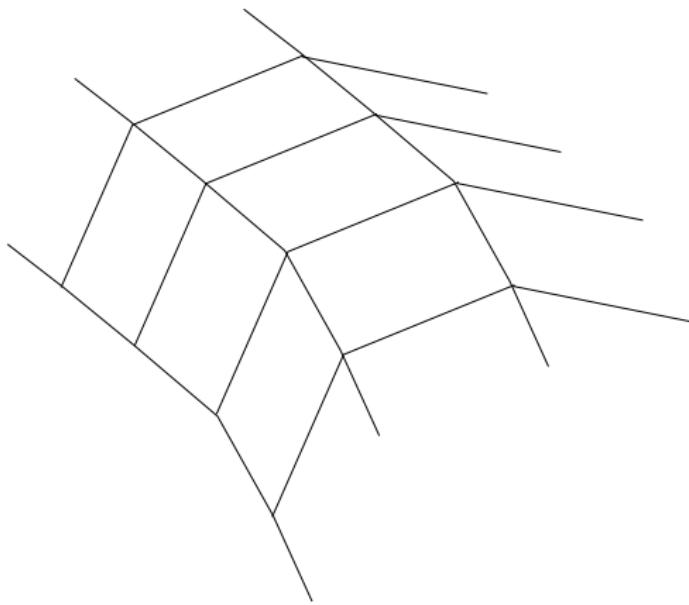
(d) Nvidia, MsDirectX11



(e) 3DS, Catia, Imagine and Shape2

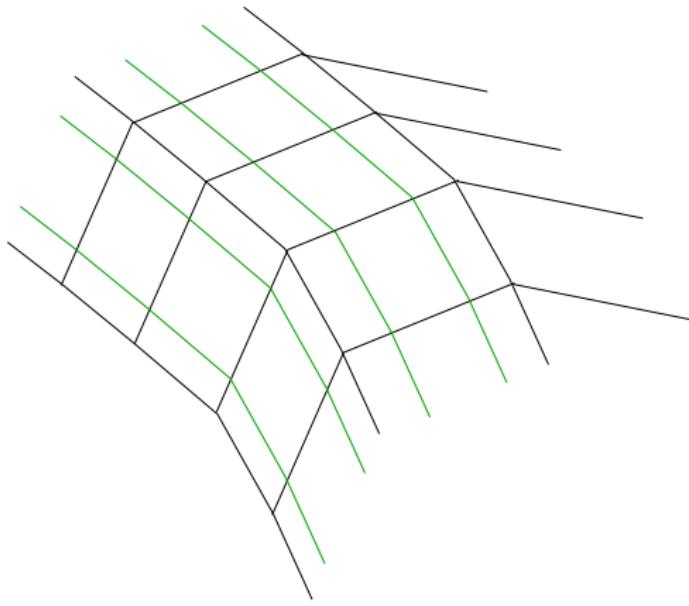
Surface de Chaikin

On peut construire une surface de subdivision de Chaikin en appliquant l'algorithme des courbes sur 2 courbes principales.



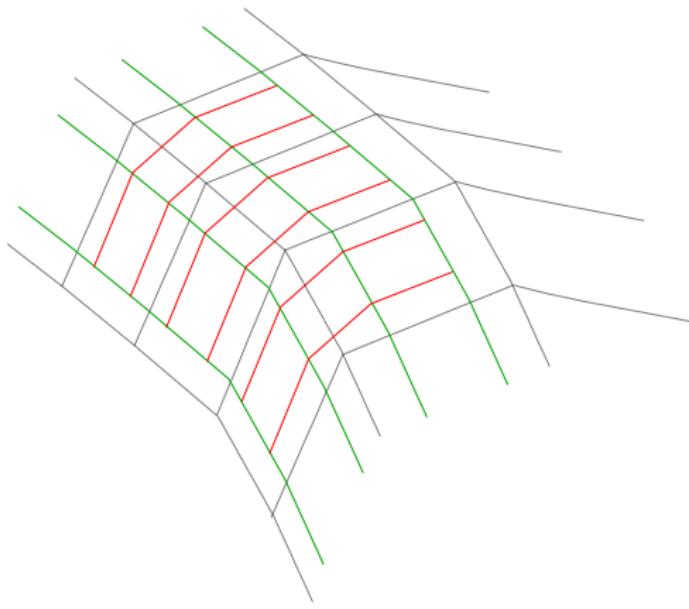
Surface de Chaikin

On peut construire une surface de subdivision de Chaikin en appliquant l'algorithme des courbes sur 2 courbes principales.



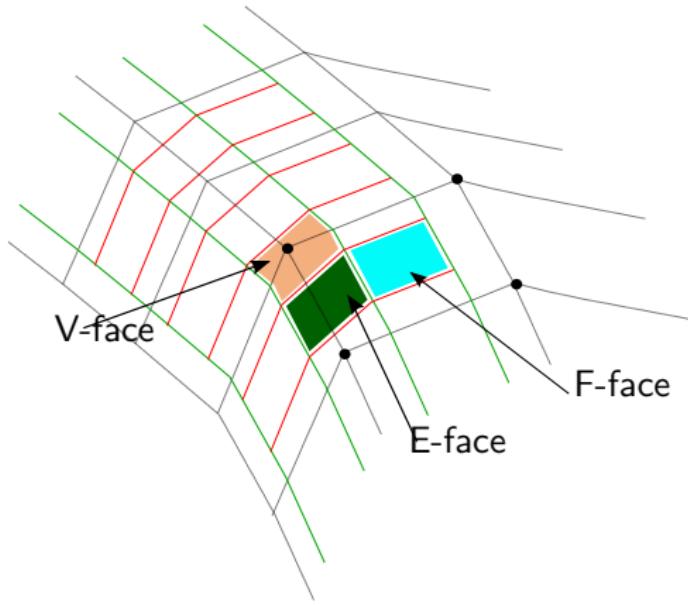
Surface de Chaikin

On peut construire une surface de subdivision de Chaikin en appliquant l'algorithme des courbes sur 2 courbes principales.



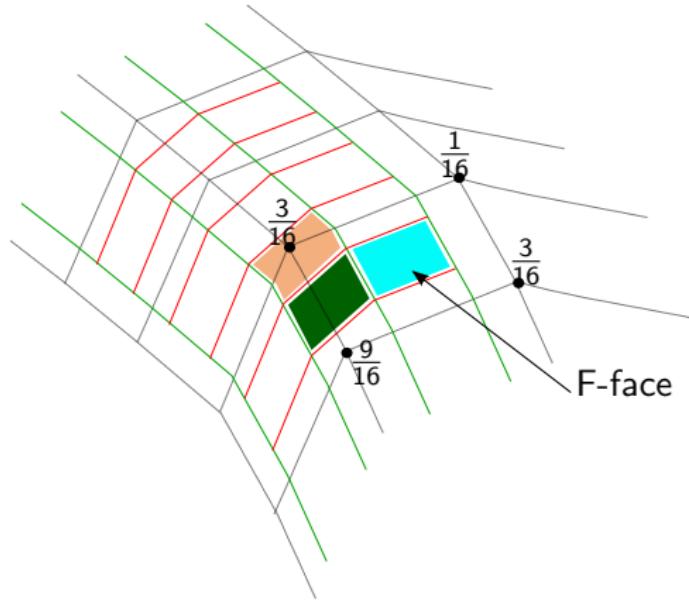
Surface de Chaikin

On crée des faces qui viennent de faces, d'arêtes ou de sommets du maillage au niveau précédent.

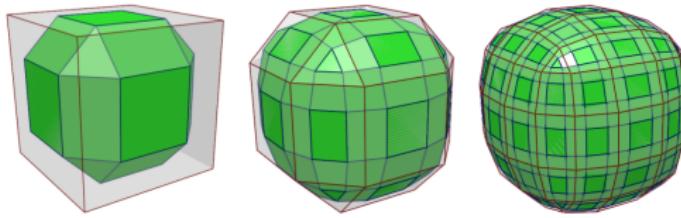


Surface de Chaikin

On peut déduire des masques de subdivision de cette construction.



Exemple de Corner Cutting sur un maillage



Terminologie des surfaces de subdivision

Approximation / Interpolation

- Approximation : les sommets du maillage initial ne sont pas sur la surface limite. D'un niveau à l'autre les sommets sont déplacés ou supprimés. C'est un lissage de la surface mais le résultat est difficile à prévoir.
- Interpolation : les sommets initiaux sont sur la surface limite. Celle-ci est plus souple, souvent trop.

Terminologie des surfaces de subdivision

Uniforme / Stationnaire

- Uniforme : les mêmes règles sont appliquées sur tout le réseau (quelle que soit la valence ou la configuration),
- Stationnaire : les règles de subdivision sont identiques à chaque itération

Terminologie des surfaces de subdivision

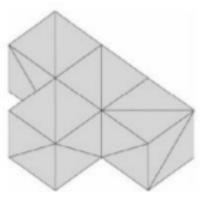
Primal / Dual

- Primal : le schéma de subdivision repose sur le partage des faces,
- Dual : le schéma de subdivision repose sur le partage des sommets

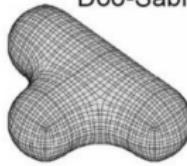
Les nouveaux sommets insérés sont dits *impairs*, ceux conservés sont dit *pairs*.

Quelques autres schémas

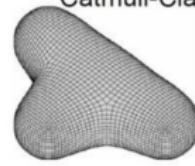
	Primal		Dual
	Triangles	Rectangles	
Approximating	Loop	Catmull-Clark	Doo-Sabin Midedge
Interpolating	Butterfly	Kobbelt	



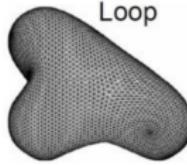
Doo-Sabin



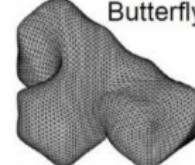
Catmull-Clark



Loop



Butterfly



Catmull-Clark⁴

Le processus de la subdivision de Catmull-Clark est un schéma Primal et crée des sommets. La subdivision s'effectue en 3 étapes, pour créer des sommets de face, d'arête, de sommets :

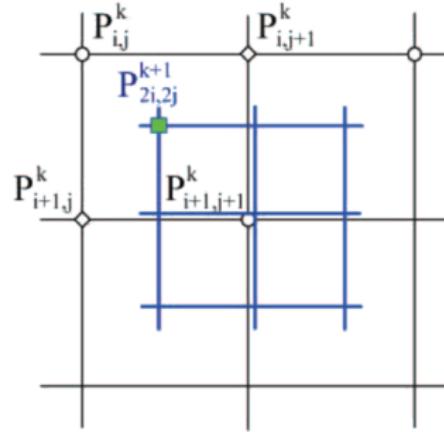
- un sommet de face est le **barycentre** de l'ancienne face
- un sommet d'arête est la **moyenne** entre le milieu des anciens sommets et le milieu des faces partageant l'arête,
- la nouvelle position du sommet est calculée par $Q + 2R + S(n - 3)$ où Q la moyenne des sommets de face adjacents, R la moyenne des points milieux des arêtes incidentes, S les anciennes coordonnées et n la valence en ce point.

4. Catmull, E. and J. Clark (1978), "Recursively generated B-spline surfaces on arbitrary topological meshes.", Computer Aided Design 9(6), p. 350-355.

Catmull-Clark : points de face

Pour chaque face du maillage M^k on produit un point de face $P_{2i,2j}^{k+1}$ du nouveau maillage de subdivision M^{k+1} :

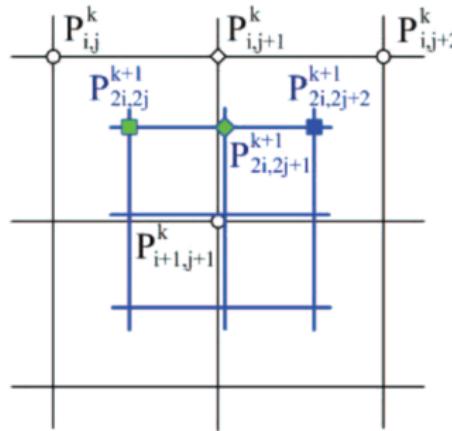
$$P_{2i,2j}^{k+1} = (P_{i,j}^k + P_{i,j+1}^k + P_{i+1,j+1}^k + P_{i+1,j}^k) / 4 \text{ avec } i = 0 \dots m-2, j = 0 \dots n-2$$



Catmull-Clark : points d'arête

Pour chaque arête du maillage M^k est créé un point d'arête $P_{2i,2j+1}^{k+1}$. Ce point est obtenu par la moyenne des sommets $P_{i,j+1}^k$, $P_{i+1,j+1}^k$ de cette arête, et la moyenne des points de face $P_{2i,2j+2}^{k+1}$, $P_{2i,2j}^{k+1}$ issus des deux faces adjacentes à cette arête :

$$P_{2i,2j+1}^{k+1} = (P_{i,j+1}^k + P_{i+1,j+1}^k + P_{2i,2j+2}^{k+1} + P_{2i,2j}^{k+1}) / 4 \text{ avec } i = 0 \dots m-2, j = 0 \dots$$

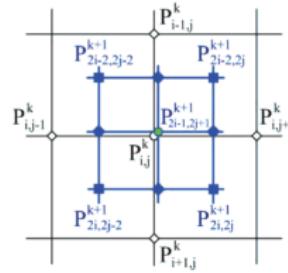


Catmull-Clark : points de sommet

Pour chaque sommet du maillage M^k est créé un nouveau point de sommet $P_{2i-1,2j-1}^{k+1}$. Ce point est calculé en se basant sur le point de sommet original $P_{i,j}^k$, les nouveaux points de faces du maillage M^{k+1} et les points d'arête voisins de ce point :

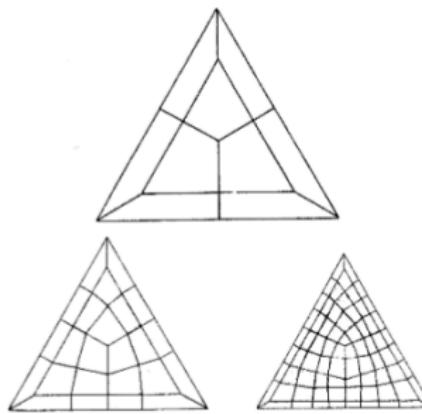
$$\begin{aligned} P_{2i-1,2j-1}^{k+1} = & P_{i,j}^k / 2 \\ & + (P_{2i-2,2j-2}^{k+1} + P_{2i-2,2j}^{k+1} + P_{2i,2j}^{k+1} + P_{2i,2j-2}^{k+1}) / 16 \\ & + (P_{i,j-1}^k + P_{i,j+1}^k + P_{i-1,j}^k + P_{i+1,j}^k) / 16 \end{aligned}$$

avec $i = 0 \dots m - 2, j = 0 \dots n - 2$



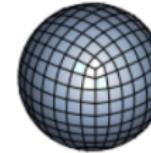
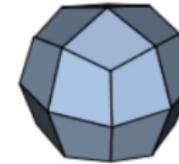
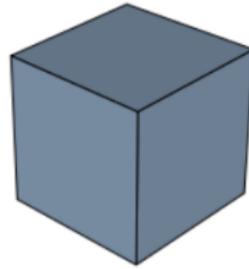
Quelques exemples

On peut partir d'une topologie quelconque on retombe sur un maillage quadrangulaire (+ des points extraordinaires)



Quelques exemples

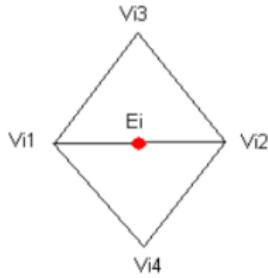
Si on l'applique sur une forme simple, on voit que le schéma est approximant. Il est aussi C^2 partout (surface Bspline bi-cubique) sauf aux points extraordinaires.



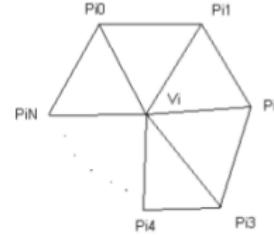
Subdivision de Loop⁵

Ne s'applique que sur un maillage triangulaire. Primal, il insère des points et déplace les sommets pairs.

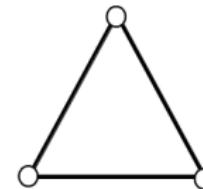
- sommet d'arête : $E^{i+1} = \frac{3}{8}(V_1 + V_2) + \frac{1}{8}(V_3 + V_4)$
- déplacement du sommet pair : $V^{i+1} = (1 - n\alpha)V^i + \alpha \sum_{k=0}^n P_k$,
avec $\alpha = \frac{1}{n} \left(\frac{5}{8} - \left(\frac{3}{8} - \frac{1}{4} \cos \left(\frac{2\pi}{n} \right) \right)^2 \right)$



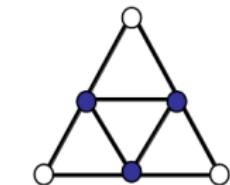
(f) sommet d'arête



(g) déplacement du sommet



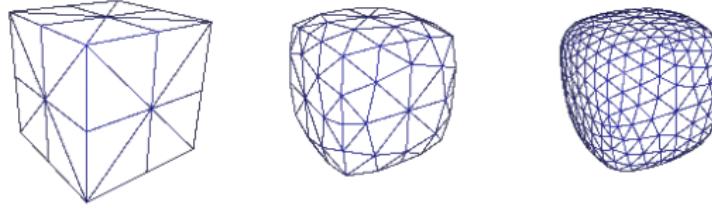
(h) résultat sur un triangle



5. Loop, C. (1987), "Smooth Subdivision Surfaces Based on Triangles.", Master's thesis, University of Utah, Department of Mathematics.

Des résultats de la subdivision de Loop

Sur un cube ...



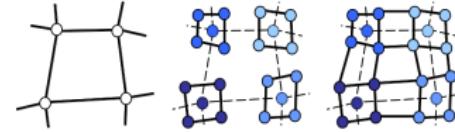
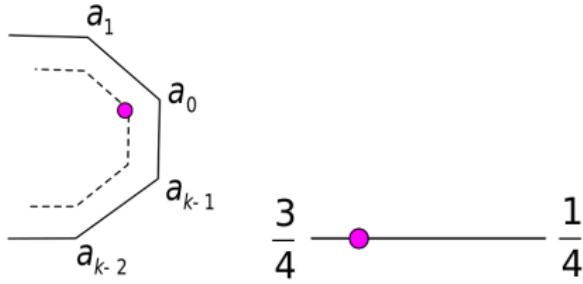
Attention la triangulation initiale modifie le voisinage, donc l'application de la subdivision. Il y a des règles spécifiques pour un sommet de valence 3, un sommet de bord ou une arête de bord.

Doo-Sabin

Doo-Sabin est un schéma Dual, qui subdivise les sommets. Dans une configuration générale, le point rose de la figure donne 4 points créés par :

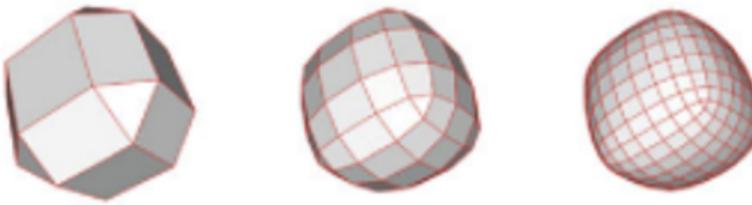
$$a_0 = \frac{1}{4} + \frac{5}{4k}$$

$$a_i = \frac{\left(3 + 2 \cos\left(\frac{2ip}{k}\right)\right)}{4k}, \text{ avec } i = 1, \dots, k-1$$



Exemple de Doo-Sabin sur un cube

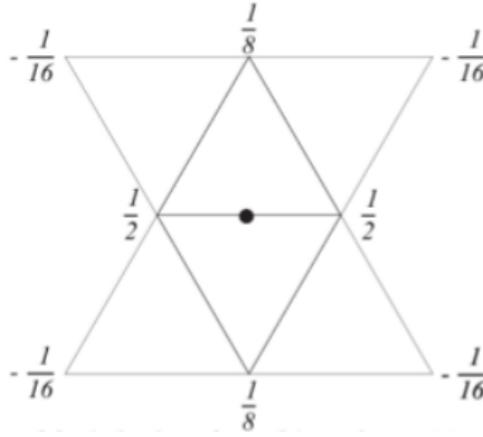
Cela ressemble à l'image du Chaikin ...



Butterfly

Butterfly est un schéma C^1 , interpolant s'appliquant sur des maillages triangulaires.

- si le maillage est régulier (valence 6), on divise chaque arête :



Butterfly

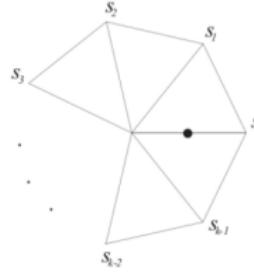
Butterfly est un schéma C^1 , interpolant s'appliquant sur des maillages triangulaires.

- si le sommet n'est pas régulier, l'insertion se fait suivant la valence :

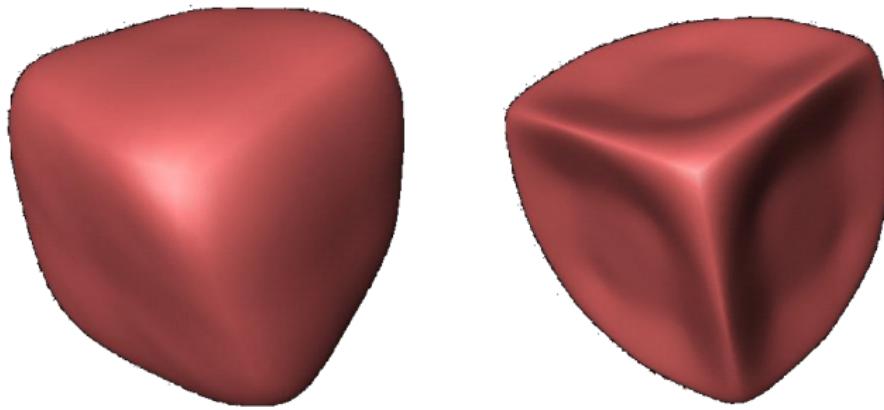
$$\text{si } k = 3, \alpha_0 = \frac{5}{12}, \alpha_{1,2} = -\frac{1}{12}$$

$$\text{si } k = 4, \alpha_0 = \frac{3}{8}, \alpha_2 = -\frac{1}{8}, \alpha_{1,3} = 0$$

$$\text{si } k \geq 5, \alpha_i = \frac{1}{k} \left(\frac{1}{4} + \cos \left(\frac{2\pi i}{k} \right) + \frac{1}{2} \cos \left(\frac{4\pi i}{k} \right) \right)$$



Exemple de Butterfly sur un cube



Des outils pour la subdivision

- image d'un point du maillage de contrôle sur la surface limite,
- image d'un point quelconque du maillage sur la surface limite,
- image d'un polygone plaqué sur le maillage sur la surface limite

Pour les courbes

Cela permet par exemple de calculer une carte ou une moyenne des déplacements, une mesure de la contraction. Cela dépend évidemment de la subdivision ...

Pour les courbes, avec une courbe B-spline quadratique⁶ (Chaikin), l'image du point P_k^0 du maillage initial, de voisins P_{k-1}^0 et P_{k+1}^0 est :

$$P_k^\infty = \frac{1}{8}P_{k-1}^0 + \frac{3}{4}P_k^0 + \frac{1}{8}P_{k+1}^0$$

6. Mueller, H. and R. Jaeschke (1998). "Adaptive Subdivision Curves and Surfaces.", Proceedings of Computer Graphics International 98, p. 48-58.

Projection d'un point initial - Catmull-Clark⁷

Pour les surfaces de Catmull-Clark, pour le point P^0 :

$$P^\infty = \frac{n^2 P^1 + 4 \sum_j e_j^1 + \sum_j f_j^1}{n(n+5)}$$

e les arêtes du voisinage et f les faces.

7. Jos Stam (1999), "Exact Evaluation of Catmull-Clark Subdivision Surfaces at Arbitrary Parameter Values", SIGGRAPH'99 Course Notes.

Projection d'un point initial - Surface de Loop⁸

Pour les surfaces de Loop, pour le point P^0 :

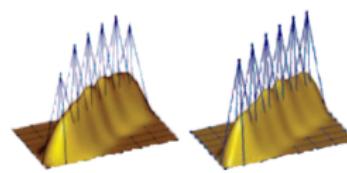
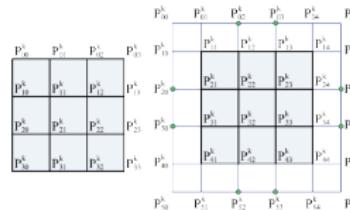
$$P_0^\infty = \frac{3}{8n\beta + 3} P_0^0 + \frac{8\beta}{8n\beta + 3} \sum_{i=1}^n P_i^0$$

Avec β le coefficient calculé à la construction de la surface (dépend de la valence de chaque sommet)

8. Jos Stam (1999), "Evaluation of Loop Subdivision Surfaces", SIGGRAPH'99 Course Notes.

Application du calcul du point sur la surface⁹

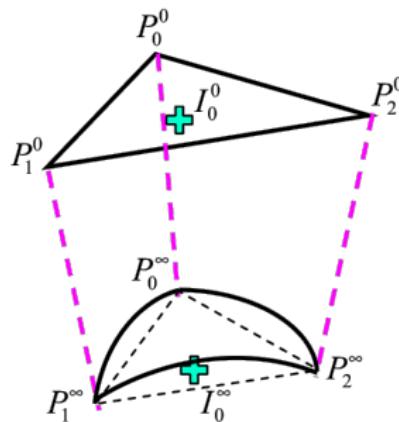
Si on utilise un schéma approximant, on contracte la surface et on perd les contraintes C^0 sur les bords. L'idée : calculer la contraction du polygone de contrôle à chaque subdivision (ou jusqu'à la surface) pour le contrebalancer —→ apparition de points « fantômes » sur le maillage initial.



9. K. Nguyen Tan (2010), "Surfaces polyédriques et surfaces paramétriques : une reconstruction par approximation via les surfaces de subdivision.", PhD thesis, LSIS, Aix-Marseille Université.

L'image d'un point quelconque : c'est plus dur ...

En effet, l'image d'un point quelconque du maillage initial n'est pas sur la surface finale (que le schéma soit approximant ou interpolant), et la surface finale n'est pas constituée de triangles (plutôt de triangles de Bézier, non planaires)



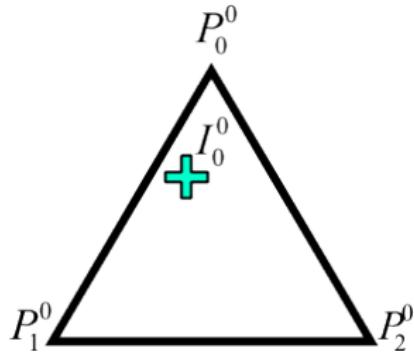
$$I_0^0 = \alpha_0 P_0^0 + \alpha_1 P_1^0 + \alpha_2 P_2^0, \text{ avec } \alpha_0 + \alpha_1 + \alpha_2 = 1 \text{ n'implique pas :}$$

$$I_0^\infty = \alpha_0 P_0^\infty + \alpha_1 P_1^\infty + \alpha_2 P_2^\infty.$$

Comment faire ?

Si on sait calculer l'image d'un point d'un niveau à l'autre on pourra peut-être appliquer une récurrence ...

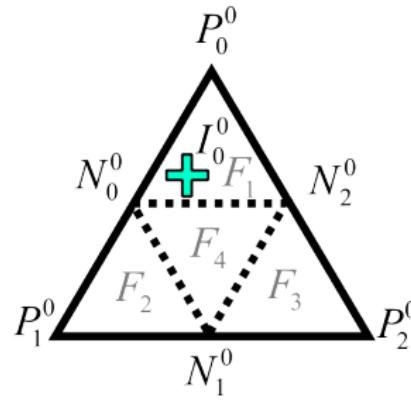
On cherche pour le point I_0^0 la face à laquelle il appartient sur le maillage initial,



Comment faire ?

Si on sait calculer l'image d'un point d'un niveau à l'autre on pourra peut-être appliquer une récurrence ...

On calcule ensuite le triangle subdivisé dans lequel sera I_0^0 ,



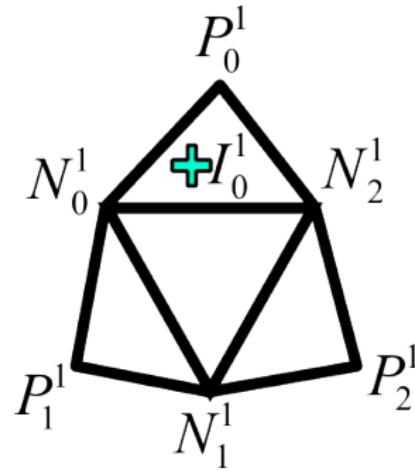
On calcule les coordonnées barycentriques de I_0^0 dans ce triangle (F_1 ici) :

$$I_0^0 = \alpha P_0^0 + \beta N_0^0 + \gamma N_2^0 \quad \alpha, \beta, \gamma \text{ inconnues}$$

Comment faire ?

Si on sait calculer l'image d'un point d'un niveau à l'autre on pourra peut-être appliquer une récurrence ...

On réutilise les coordonnées barycentriques de I_0^0 pour obtenir I_0^1

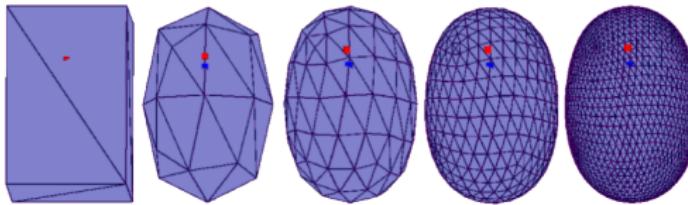


$$I_0 = \alpha P_0^1 + \beta N_0^1 + \gamma N_2^1$$

Comment faire ?

Si on sait calculer l'image d'un point d'un niveau à l'autre on pourra peut-être appliquer une récurrence ...

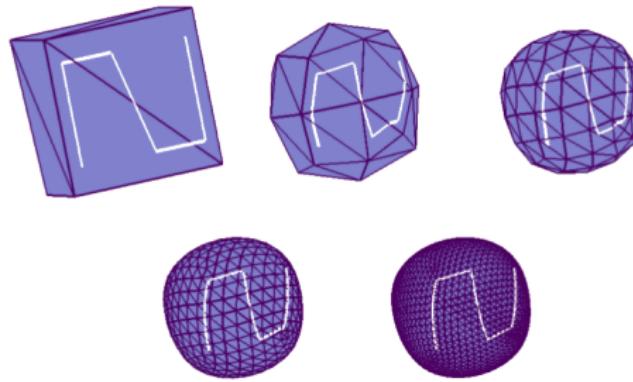
On obtiendra finalement le suivi du point I_0^0 de niveau en niveau de subdivision par itération successive.



Remarque1 : ce n'est pas la projection orthogonale du point sur le maillage, on a calculé une distance maximale à la surface. Remarque2 : cela peut être un moyen de calculer une profondeur de subdivision selon un critère d'erreur à la surface limite.

Et plus ?

Pendant qu'on y est on peut également suivre une courbe polygonale de niveau en niveau¹⁰

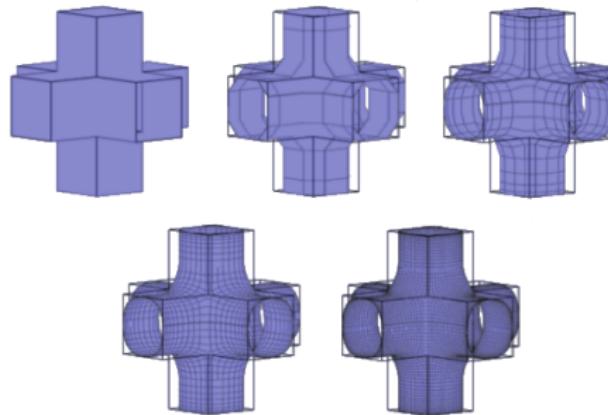


!! Attention, il faut calculer les points intermédiaires, sur les arêtes, qui sont obtenus par intersection des segments du polygone avec les arêtes des faces subdivisées !!

10. S. Lanquetin, (2004), "Les surfaces de subdivision : intersection, précision et profondeur de subdivision.", PhD Thesis, LE2I, Dijon.

Application

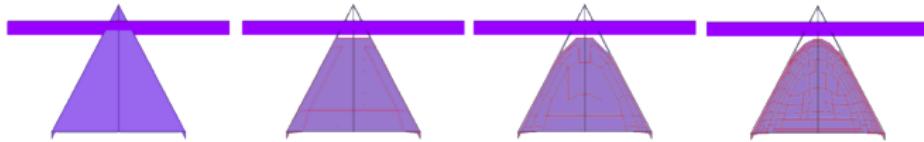
Le calcul d'intersection de surfaces de subdivision produit des polylinéaires (intersection de faces polygonales), on peut donc utiliser le résultat précédent pour suivre les intersections sur la surface. Par exemple, comment construire cet objet en assurant la continuité des bords des cylindres initiaux ?



Il faut manipuler les points de contrôle de la surface et calculer la surface limite. Le calcul d'intersection pourrait peut-être faciliter ce cas. Il peut

Attention aux « faux-amis » !

Le calcul d'intersection n'est pas forcément vrai pour tous les niveaux :



Il faut tester pour chaque niveau si l'intersection existe encore et éventuellement remonter l'information sur les niveaux précédents.

Application

On peut même faire de la CSG¹¹ !

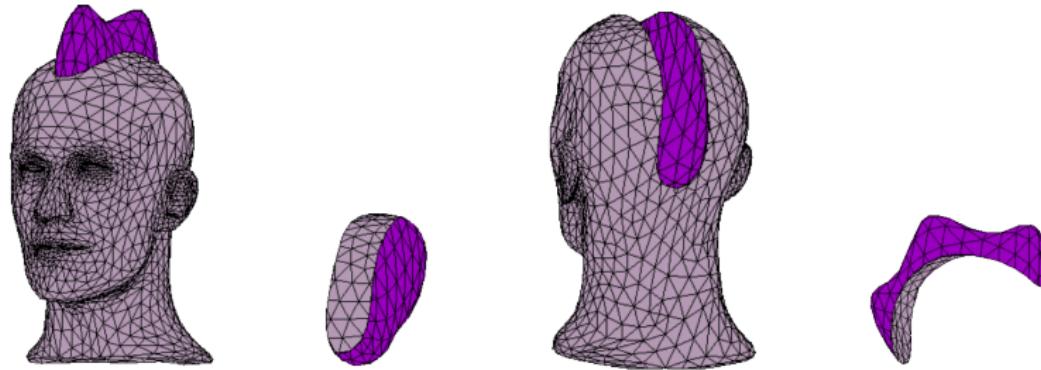


FIGURE: *

Union, intersection et différences

11. S. Lanquetin, (2004), "Les surfaces de subdivision : intersection, précision et profondeur de subdivision.", PhD Thesis, LE2I, Dijon.

Subdivision adaptative : besoins

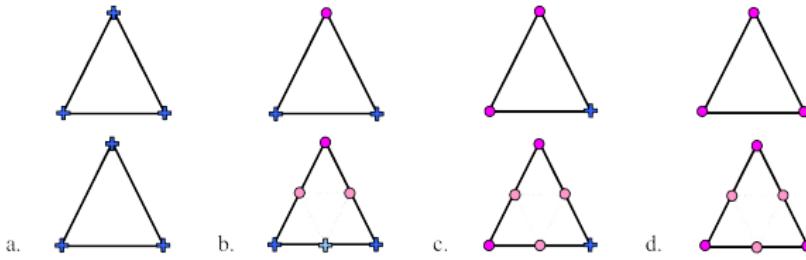
La subdivision adaptative est adapté à la visualisation ou à la diminution de l'empreinte mémoire. Il faut un moyen 1) de déterminer quelles faces seront subdivisées localement, 2) de savoir combien de niveaux sont nécessaires pour un critère donné, 3) de ne pas introduire de trous (« *cracks* ») entre les faces.



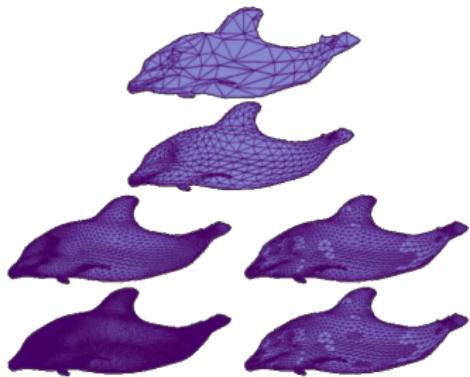
Cela nécessite une adaptation des algorithmes, pas des schémas de subdivision.

Subdivision adaptative : principes

Les faces sont classées selon qu'elles contiennent un sommet fixe, 2 (une arête), 3 (une face) pour une subdivision de Loop :



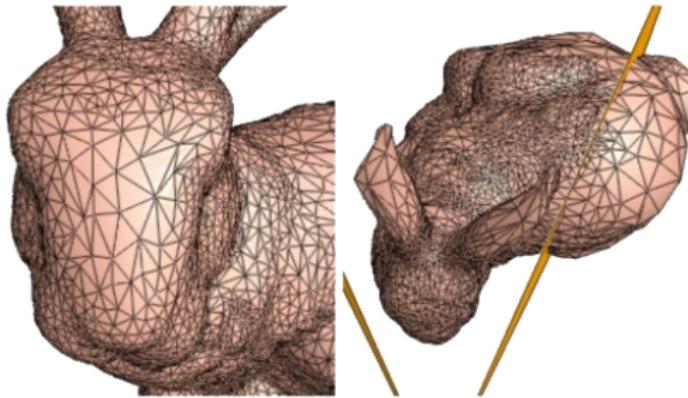
Un critère d'adaptation du nombre de niveaux de subdivision locale peut être la distance à la surface limite (calculée avant). Cela permet de représenter un objet « au mieux ».



Niveau de subdivision	Subdivision complète			Subdivision adaptative			Distance maximum
	Nombre de faces	Distance moyenne	Temps de subdivision	Nombre de faces	Distance moyenne	Temps de subdivision	
0	468	0.904		468	0.904		3.711
1	1872	0.176	1.793	1872	0.176	1.692	0.928
2	7488	0.043	43.552	5022	0.065	16.484	0.247
3	29952	0.011	897.371	5133	0.063	17.465	0.097

On peut également pré-calculer des « patchs » où la contraction est la plus forte, pour adapter la subdivision.

En cas de visualisation, si le niveau initial est très grossier, la différence avec la subdivision suivante sera visuellement choquante. Une solution ? une interpolation linéaire entre niveaux (en fait entre raffinement et subdivision), à la Hoppe¹².

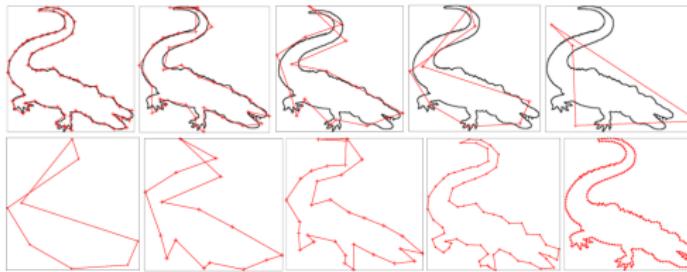


¹² Hugues Hoppe (1997), "View-dependent refinement of progressive meshes.", ACM SIGGRAPH 1997 Proceedings, pp. 189-198.

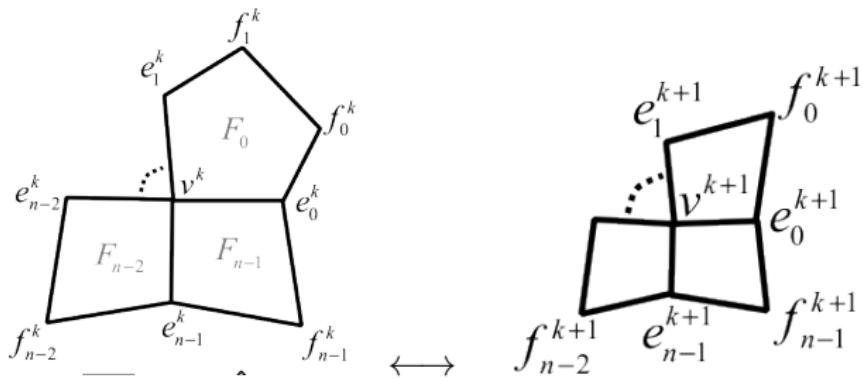
Chaïkin inverse¹³

À partir de la formule de subdivision vue précédemment (diapo 43) on peut déduire les points antécédents de la subdivision. Cela est facilité par le voisinage constant (2), la simplicité de la formule et le degré (B-spline quadratique). Il faut juste faire attention aux bords.

$$P_k^n = \frac{1}{4} (-P_{n+1}^{2k-2} + 3P_{n+1}^{2k-1} + 3P_{n+1}^{2k} - P_{n+1}^{2k+1})$$



13. M. F. Hassan (2005), "Reverse Subdivision", Advances in Multiresolution for Geometric Modelling, pp. 271-283

Catmull-Clark¹⁴

14. S. Lanquetin (2006), "Reverse Catmull-Clark Subdivision", Visualization and Computer Vision'2006 (WSCG), pp. 319-326.

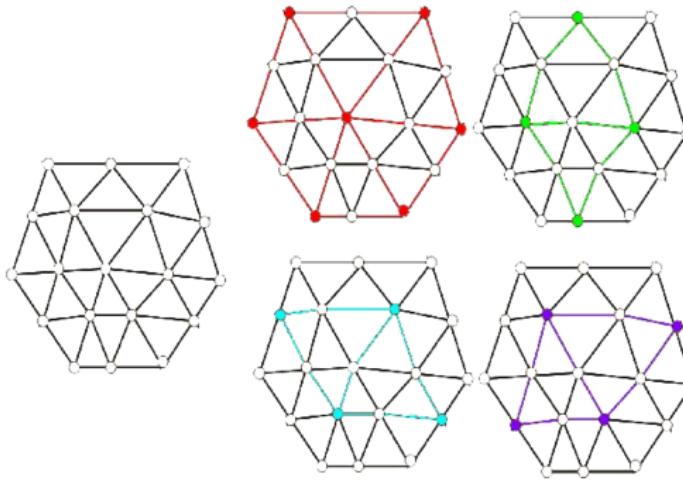
Le calcul des positions antécéduentes v^0 aux v^1 se font grâce à :

$$v^k = \frac{n}{n-3} v^{k+1} + \frac{-4}{n(n-3)} \sum_{j=0}^{n-1} e_j^{k+1} + \frac{1}{n(n-3)} \sum_{j=0}^{n-1} f_j^{k+1}$$

Là encore, l'inversion est facilitée par la valence constante d'un sommet (maillage quadrilatéral). Si ce n'est pas le cas (valence = 3 par ex.), il faut un traitement particulier.

Loop

La difficulté avec un maillage issu de Loop et qu'il faut retrouver le motif qui a créé l'assemblage de 4 triangles :



Le problème est donc de remonter à l'objet initial exact. On peut cependant trouver des objets approximants.

Sommaire

1 Introduction

2 Courbes

3 Courbes de Subdivision

4 Surfaces de Subdivision

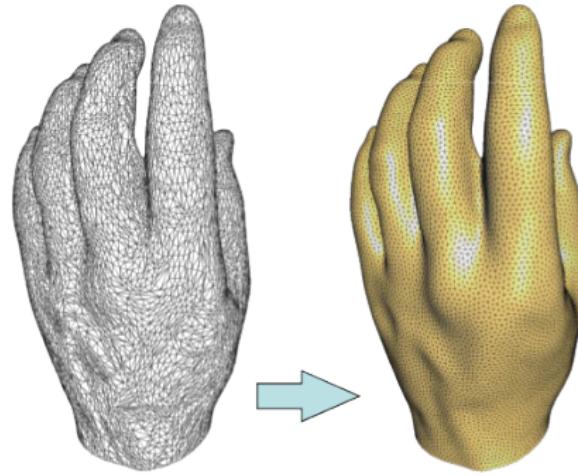
5 Maillages

6 OpenGL avancé

5 Maillages

- Remeshing
- Conversion
- Des critères sur les maillages

Si l'on doit passer d'un ensemble discret à un ensemble continu, on a vu qu'un point dur peut être la re-paramétrisation ou l'homogénéité de la surface.



Quel(s) paramètre(s) dois-je donner à ce point P_0 et quel(s) autre(s) à ce point P_1 (on ne sait pas s'ils sont voisins) ?

Quels triangles sont issus d'une subdivision de Loop ?

Le remaillage (« Remeshing ») veut, depuis un maillage brute, de topologie quelconque, fournir un maillage régulier, de triangles, ou de quadrilatères de structure régulière ou semi-régulière. Il permet également de passer d'une structure triangulaire à une structure quadrilatérale ou mixte. Son intérêt est évident pour la subdivision et la subdivision inverse, le maillage régulier est aussi important pour la compression, la simulation mécanique, le placage de texture ou le stockage.

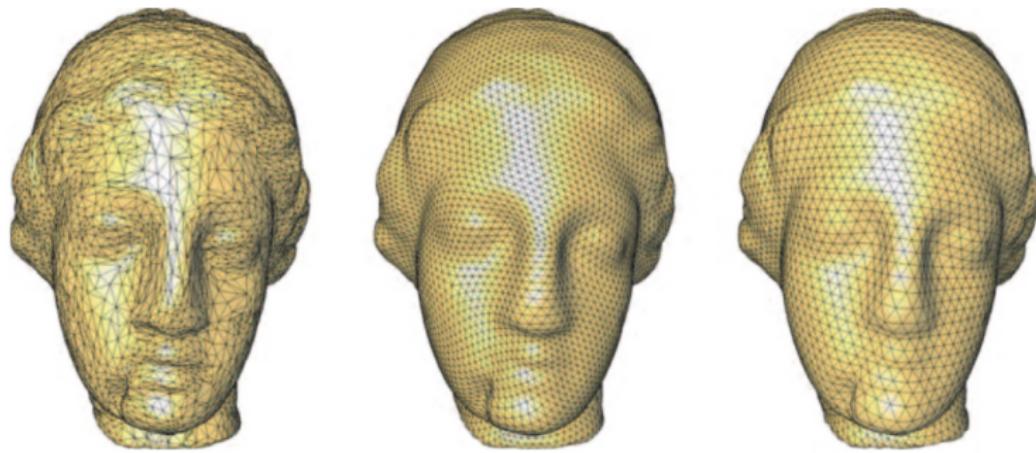
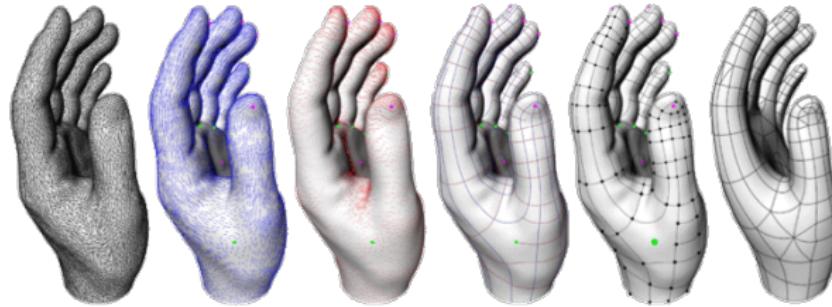


FIGURE: *

Remeshing par la courbure¹⁶

À partir d'un maillage fin et irrégulier d'une main (1), on extrait les directions principales (2-3, minimales en bleu, maximales en rouge). Puis on trace deux réseaux de lignes tangentes à ces champs de directions, en choisissant l'espacement entre les lignes en fonction de la courbure (4). À partir de ces réseaux, on déduit un nouveau maillage de la surface constitué de quadrilatères (5-6).



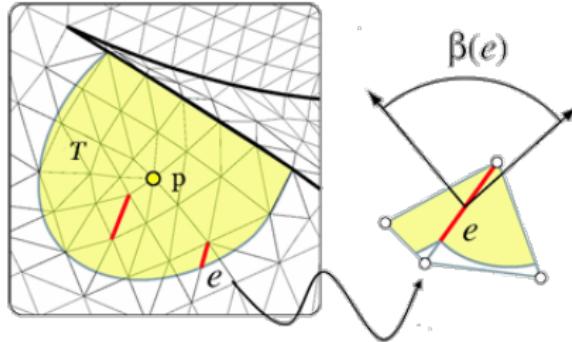
16. David Cohen-steiner (2005), http://interstices.info/jcms/c_15007/calculer-la-courbure-d-un-maillage

Tenseur de courbure

On cherche à exprimer en un point p une normale et les directions principales. Si on considère autour de p une portion de maillage T qui contient « un certain nombre » de faces. On teste pour chaque arête e l'angle dièdre $\beta(e)$ des 2 faces concourantes. Le tenseur de courbure est H_T :

$$H_T = \sum_{e \in T} \beta(e) \text{length}(e) E E^t$$

avec E le vecteur de l'arête e , E^t son transposé. L'angle $\beta(e)$ est positif ou négatif selon que le maillage local est convexe ou concave.



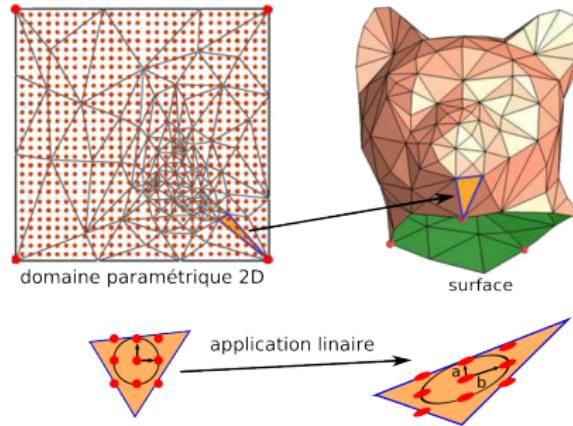
Plongement paramétrique

Dans le même esprit qu'aller d'un nuage de points à une courbe, on veut ici « déplier » le maillage discret sur un plan, qui constituera un espace paramétrique (discret ou non). On doit également créer une fonction qui fasse passer (comme les fonctions linéaires ou paramétriques) des couples (u, v) de l'espace paramétrique aux points $P(x, y, z)$ de l'espace \mathbb{R}^3 .

On parle de paramétrisation (« *mesh parameterization* ») ou de plongement de maillage (« *mesh embedding* »).

Plongement paramétrique

Dans le même esprit qu'aller d'un nuage de points à une courbe, on veut ici « déplier » le maillage discret sur un plan, qui constituera un espace paramétrique (discret ou non). On doit également créer une fonction qui fasse passer (comme les fonctions linéaires ou paramétriques) des couples (u, v) de l'espace paramétrique aux points $P(x, y, z)$ de l'espace \mathbb{R}^3 .

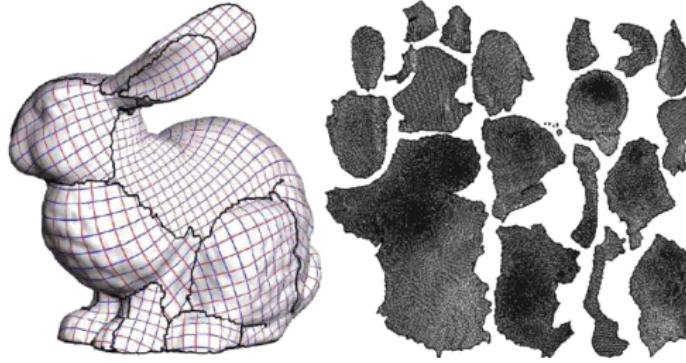


Quels intérêts et problèmes ?

- + on conserve le voisinage,
- + on peut utiliser ce domaine paramétrique pour le passer du réel au discret (approximation),
- + travailler dans ce plan est plus facile que dans l'espace \mathbb{R}^3 ,
- on perd l'accès direct aux caractéristiques locales (tangences, normales, courbures), même si la méthode de plongement peut en tenir compte,
- on n'est pas sûr que le dépliage à plat soit aisé, d'autres projections existent (sphère ou plus complexe) mais pas de méthode universelle.

par Atlas

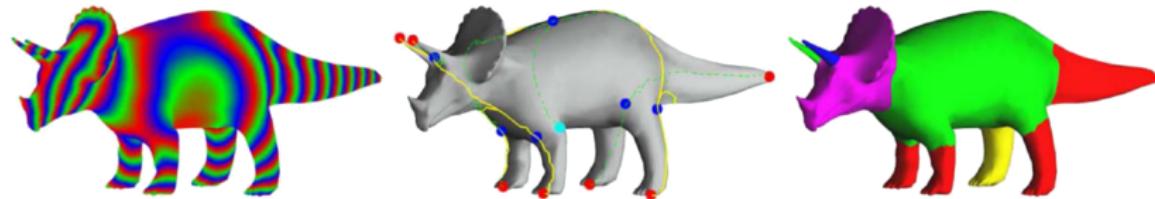
Dans la même veine que l'*uv-mapping* on retrouve les méthodes par atlas de faces. On découpe suivant un ou plusieurs critères (planéité, aire, distance, variation de courbures, *feature*).



Découpage de patchs¹⁷

Quelques méthodes de « *mesh unfolding* » dans des atlas de cartes, comme des textures (G. Turk, H. Hoppe, M. Levoy, ...).

Par découpage selon les distances géodésiques :



17. Zhang (2005), "Feature-Based Surface Parameterization and Texture Mapping", ACM Transactions on Graphics, Vol. 24, No. 1.

Découpage de patchs¹⁷

Quelques méthodes de « *mesh unfolding* » dans des atlas de cartes, comme des textures (G. Turk, H. Hoppe, M. Levoy, ...).

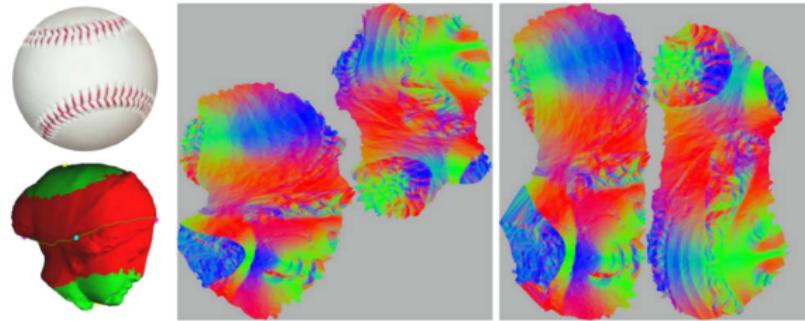
En faisant attention au genre de la surface (détection des bords et des cycles d'arêtes).



Découpage de patchs¹⁷

Quelques méthodes de « *mesh unfolding* » dans des atlas de cartes, comme des textures (G. Turk, H. Hoppe, M. Levoy, ...).

En projetant sur une surface découpée connue (homéomorphisme) :

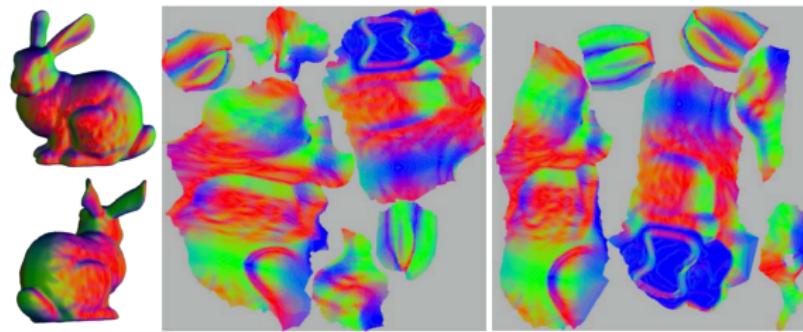


17. Zhang (2005), "Feature-Based Surface Parameterization and Texture Mapping", ACM Transactions on Graphics, Vol. 24, No. 1.

Découpage de patchs¹⁸

Quelques méthodes de « *mesh unfolding* » dans des atlas de cartes, comme des textures (G. Turk, H. Hoppe, M. Levoy, ...).

En essayant de modifier la paramétrisation initiale pour avoir une densité homogène¹⁷, ou une répartition selon les aires (Green-Lagrange Tensor) :

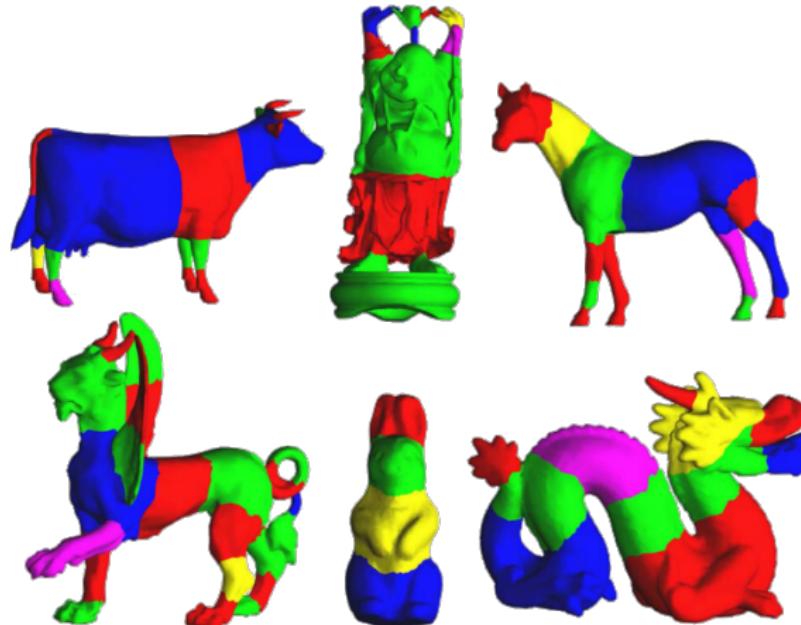


17. P.V. Sander (2001), "Texture mapping progressive meshes.", Computer Graphics Proceedings, SIGGRAPH 2001, pp. 409–416.

18. Zhang (2005), "Feature-Based Surface Parameterization and Texture Mapping", ACM Transactions on Graphics, Vol. 24, No. 1.

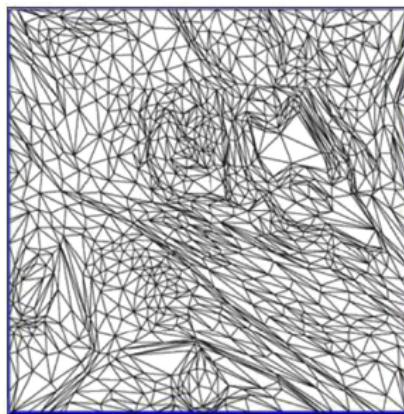
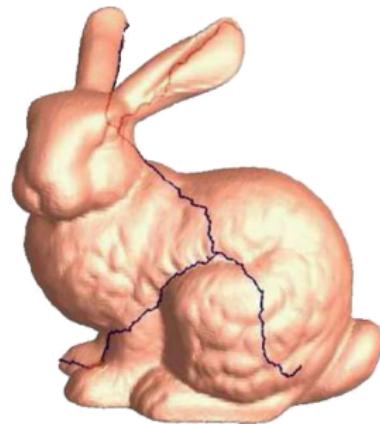
Découpage de patchs¹⁷

Quelques méthodes de « *mesh unfolding* » dans des atlas de cartes, comme des textures (G. Turk, H. Hoppe, M. Levoy, ...).



Espace paramétrique unitaire

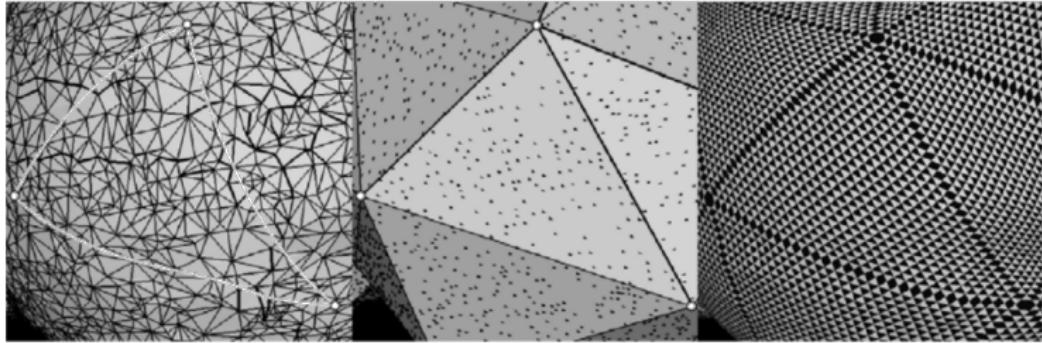
On peut aussi effectuer le découpage topologique de la surface pour un plongement dans un espace paramétrique unitaire¹⁸ :



18. B. Lévy (2002), "Least squares conformal maps for automatic texture atlas generation". In ACM SIGGRAPH'02, pp. 362-371.

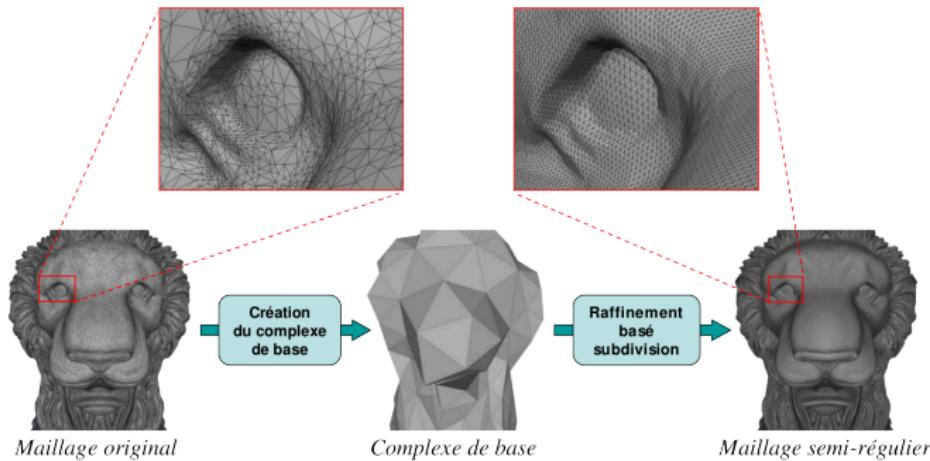
Les remailleurs semi-réguliers

Les remailleurs SR (semi-réguliers) s'appuient sur un découpage en patchs et y effectuent des subdivisions (Loop généralement).



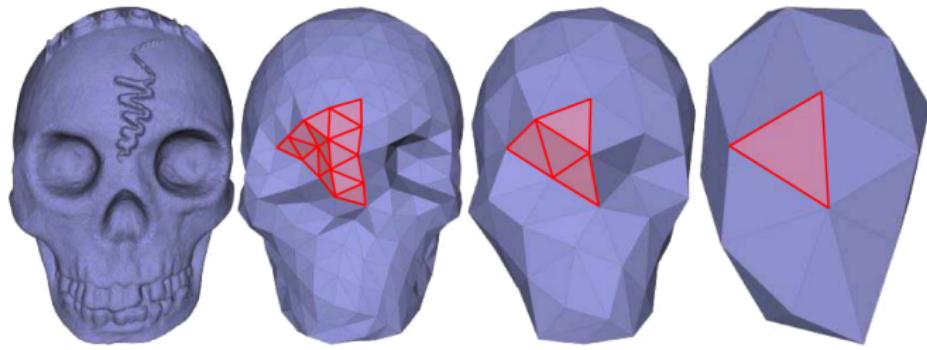
Remaillage semi-régulier¹⁹

Cette méthode passe d'un maillage quelconque à un maillage « de base » d'une subdivision, et ses propriétés de régularité intrinsèques pour obtenir un maillage remaillé régulier.



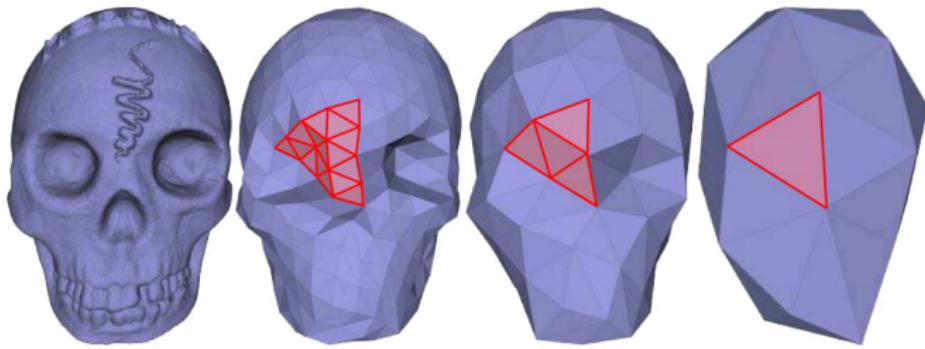
19. C. Roudet (2010), "Remaillage semi-régulier pour les maillages surfaciques triangulaires : un état de l'art", REFIG

La difficulté est donc de détecter dans une forme les zones (*patches*) pouvant participer à la description d'une partie de l'objet.



Il faut mettre en place ces descripteurs...

La difficulté est donc de détecter dans une forme les zones (*patches*) pouvant participer à la description d'une partie de l'objet.

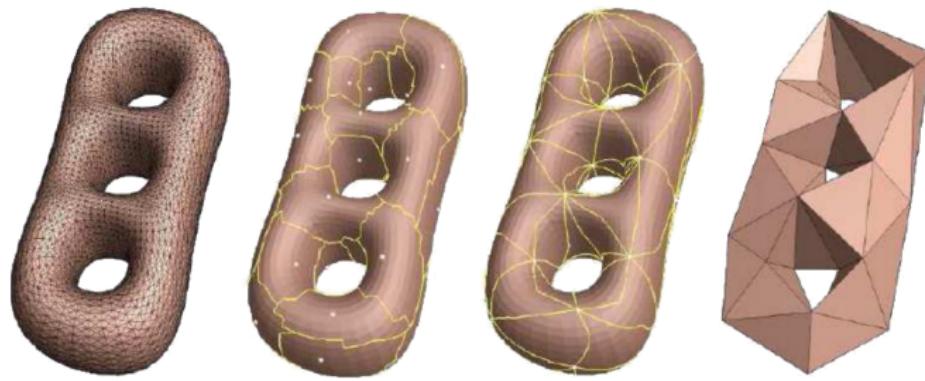


Il faut mettre en place ces descripteurs... ... qui peuvent être les mêmes que précédemment.

Note : le processus peut être itératif, l'interpolation des bords se faisant sur la surface, puis par paramétrisation du patch de subdivision.

Clustering par cellules de Voronoï

On peut définir un remaillage grossier en cherchant les cellules de Voronoï/sites de Delaunay²⁰



Le contrôle de qualité du maillage produit est ici difficile.

20. M. Eck (1995), "Multiresolution analysis of arbitrary meshes.", In ACM SIGGRAPH'95, pp. 173-182.

Décimation Edge/Vertex collapse

Le but de la méthode ici est de décimer itérativement l'objet (comme dans la méthode Progressive Mesh de Hoppe) pour obtenir un objet de base de la subdivision²¹.

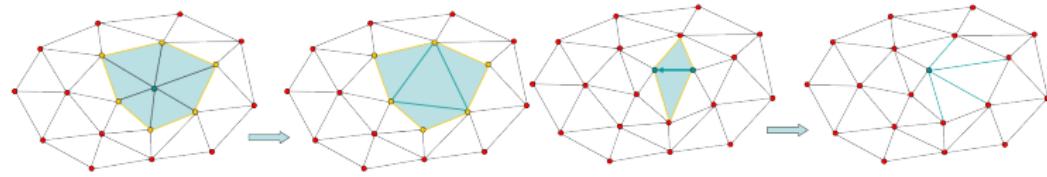


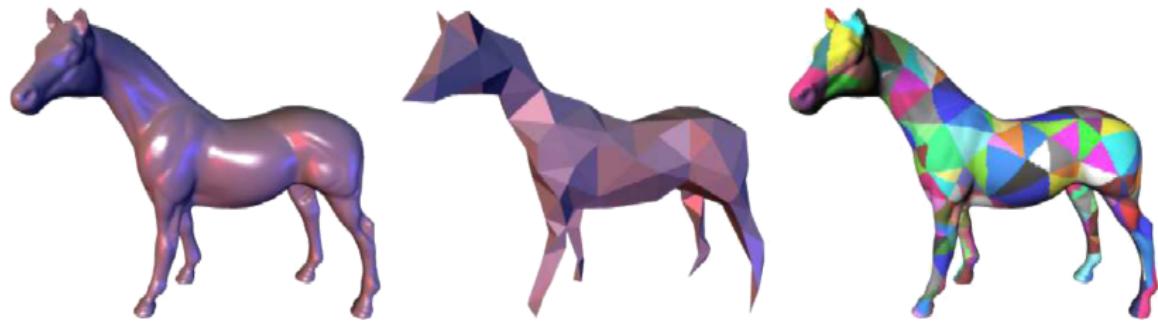
FIGURE: *

Vertex et Edge collapse

21. A.W. Lee(1998), "MAPS : multiresolution adaptive parameterization of surfaces.", In ACM SIGGRAPH'98, vol. 32, pp. 95-104.

Décimation Edge/Vertex collapse

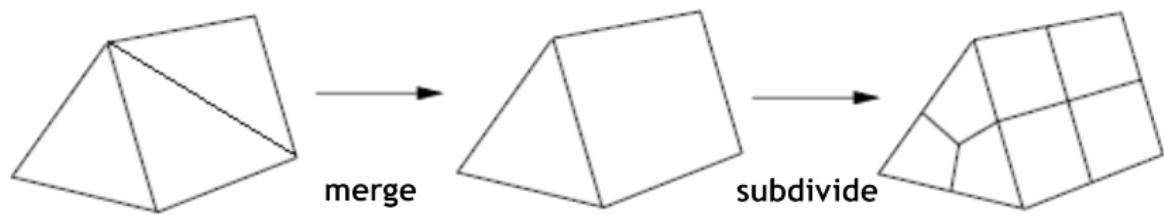
Le but de la méthode ici est de décimer itérativement l'objet (comme dans la méthode Progressive Mesh de Hoppe) pour obtenir un objet de base de la subdivision²¹.



21. A.W. Lee(1998), "MAPS : multiresolution adaptive parameterization of surfaces.", In ACM SIGGRAPH'98, vol. 32, pp. 95-104.

Conversion en quadrangles²²

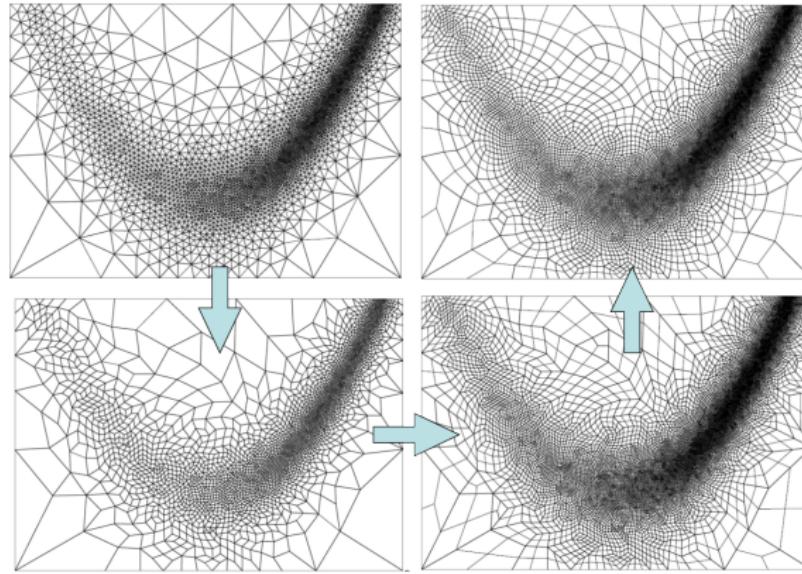
On passe de triangles à quadrangles par fusion successive de 2 triangles, sous contraintes d'aire, d'angle dièdre, ... Les triangles restant sont subdivisés.



22. Borouchaki (1996), "Adaptive Triangular-Quadrilateral Mesh Generation."

Conversion en quadrangles²²

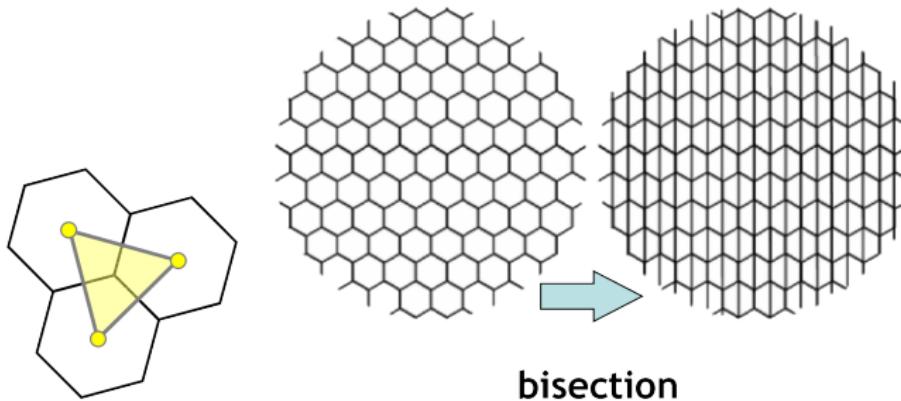
On passe de triangles à quadrangles par fusion successive de 2 triangles, sous contraintes d'aire, d'angle dièdre, ... Les triangles restant sont subdivisés.



22. Borouchaki (1996), "Adaptive Triangular-Quadrilateral Mesh Generation."

Conversion en quadrangles²³

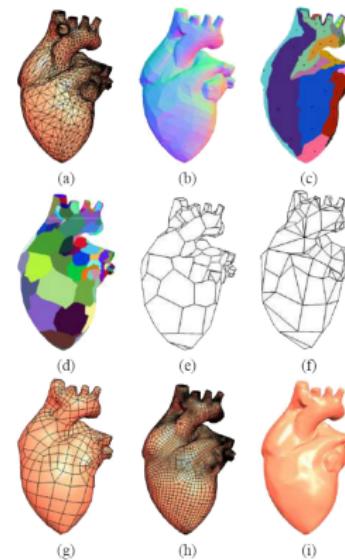
On passe de triangles à quadrangles en 2 phases : 1) calcul des complexes simpliciaux, 2) division.



23.

Conversion en quadrangles²³

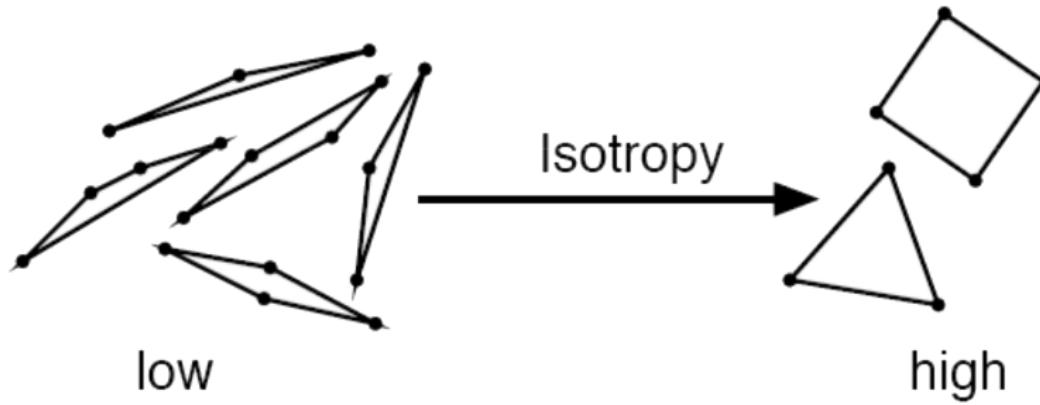
On passe de triangles à quadrangles en 2 phases : 1) calcul des complexes simpliciaux, 2) division.



23.

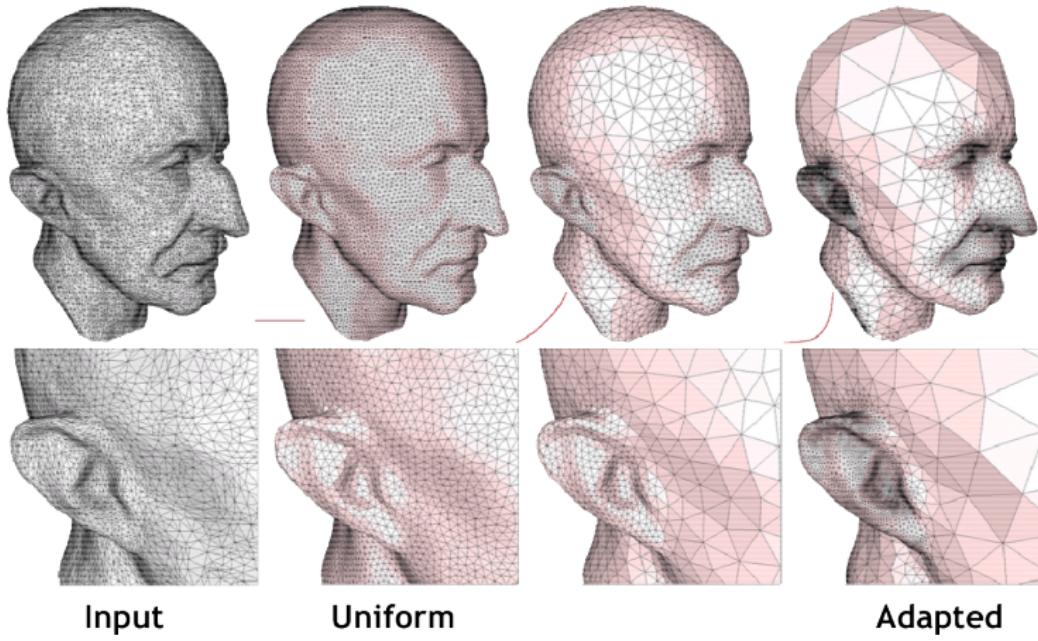
Entropie

Le calcul de l'entropie permet de régulariser la taille des éléments.



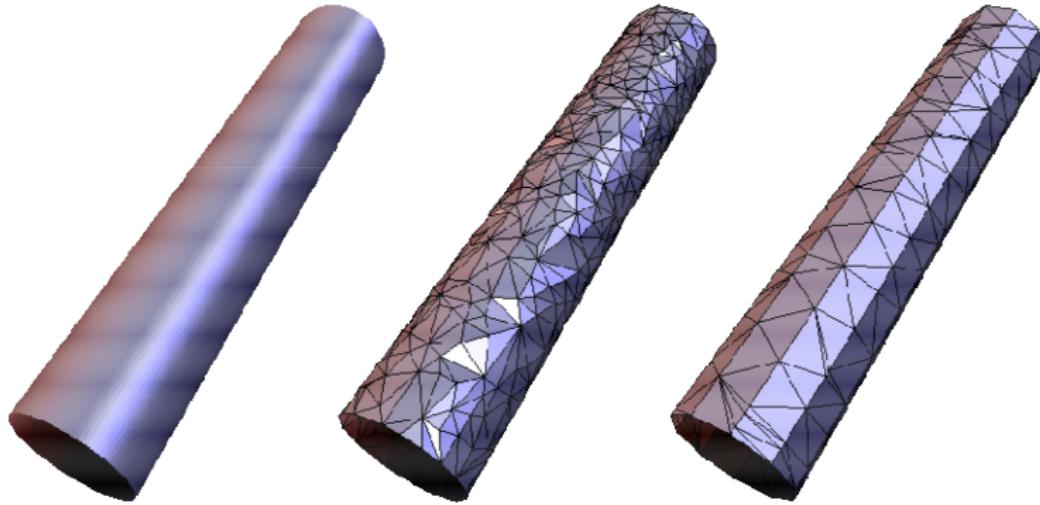
Densité

La densité ou la répartition des triangles peuvent être gouvernées par des lois (probabilités, mécanique, descripteurs locaux et avancée de front).



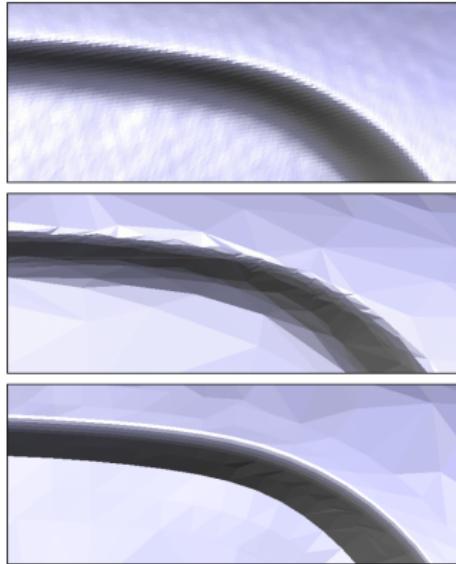
Orientations

L'orientation globale de l'objet peut guider le remaillage (calculée par ACP par ex.).



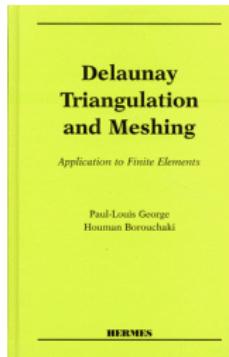
Orientations

L'orientation locale peut également être prise en compte, par des descripteurs locaux (courbure, tangence)



Un livre ?

Et bien d'autres comme les angles diédraux, la distance entre la pièce remaillée et l'originale, ... (cf. les critères de qualités de Borouchaki vus en S1).



Des liens

- la librairie CGAL et les docs associés peuvent aussi vous aider :
<http://www.cgal.org/Tutorials/>
- OpenNL <http://alice.loria.fr/index.php/software/4-library/23-opennl.html>
- Unfolding dans Blender
http://wiki.blender.org/index.php/Doc:FR/2.4/Manual/Textures/UV/Unwrapping_a_Mesh
- Polygonal Design / Unfold3D
<http://www.polygonal-design.fr/>

... Fin ...

Sommaire

1 Introduction

2 Courbes

3 Courbes de Subdivision

4 Surfaces de Subdivision

5 Maillages

6 OpenGL avancé

6 OpenGL avancé

- Avant c'était bien statique ...
- ... mais ça c'était avant
- Les VBO
- Les FBO

Mode direct

... Et coûteux en temps et en débit.

```
//Dans la boucle de rendu
glBegin(GL_TRIANGLES);
    glVertex3f(0.0f, 0.0f, 0.0f);
    glVertex3f(1.0f, 1.0f, 1.0f);
    glVertex3f(2.0f, 2.0f, 2.0f);
glEnd();
```

Cela implique qu'à chaque image (à chaque fois qu'OpenGL a besoin de retracer), on re-décrit toute la géométrie de la scène et qu'on la renvoie de la RAM du CPU vers la RAM du GPU. Si l'objet n'est pas modifié, c'est beaucoup de temps et d'énergie de perdue !

Avant c'était bien statique ...

DisplayLists

Vous avez donc vu les DisplayLists, qui permettent la pré-compilation des objets en RAM. La carte graphique arrange les données, les prépare pour leur utilisation et repère leurs emplacements grâce à un identifiant.

```
id = glGenLists(1);

glNewList(id, GL_COMPILE);

for (int i=0; i < nbfaces; i++)
    glBegin(GL_TRIANGLES);
        glVertex3f(...);
        glVertex3f(...);
        glVertex3f(...);
    glEnd();

glEndList();
```

On effectue donc un pré-travail pour la carte graphique, il n'y a plus qu'à appeler la liste (`glCall(id)`) pour l'envoyer au traitement du GPU.

Il reste 2 inconvénients : les transmissions CPU->GPU et l'impossibilité de modifier ces géométries sans recréer la liste.

Évidemment si l'objet est modifié, il faut détruire et refaire la DisplayList et la recompiler. Le 2ème inconvénient peut être résolu en implémentant un VertexShader qui peut modifier une géométrie dans une DisplayList. C'est évidemment plus rapide, pas la peine de recréer la géométrie à chaque fois mais on peut peut-être faire mieux...

Mode indirect

Les géométries, en mieux : VertexArray

Depuis la version 1.2 d'OpenGL, on a la possibilité de préparer un peu mieux les données pour la carte graphique. Les VertexArray permettent de stocker la géométrie d'un objet, mais aussi sa (ses) couleur(s), ses normales, ses coordonnées de texture... dans des tableaux. Lors du rendu, on envoie ces tableaux à la carte graphique qui n'a plus qu'à lire, les données sont déjà organisées.

Vertex Array

Comment fait-on ? On crée des tableaux qui vont contenir les données de l'objet (des tableaux de float), on met dedans toutes les données de l'objet chargé, et on les utilise plus tard.

```
float ColorArray[nbsommets*3];  
float NormalArray[nbsommets*3];  
float VertexArray[nbsommets*3];
```

Dans le rendu, on active le rendu depuis des buffers, pour chaque type de données à transmettre :

```
//attention, on va se servir de sommets  
glEnableClientState(GL_VERTEX_ARRAY);  
//attention, on va se servir de normales  
glEnableClientState(GL_NORMAL_ARRAY);  
//attention, on va se servir de couleurs  
glEnableClientState(GL_COLOR_ARRAY);  
  
//et voilà le buffer des couleurs, sur 3 float pour chaque couleur  
glColorPointer(3, GL_FLOAT, 0, ColorArray);  
  
//le buffer des normales, sur des floats (3 implicitement)  
glNormalPointer(GL_FLOAT, 0, NormalArray);  
  
//le buffer des sommets, 3 floats pour chaque coordonnées  
glVertexPointer(3, GL_FLOAT, 0, VertexArray);  
  
//et hop, on envoie au rendu de la CG  
glDrawArrays(GL_TRIANGLES, 0, nbfaces * 3);  
  
glDisableClientState(GL_COLOR_ARRAY);  
glDisableClientState(GL_NORMAL_ARRAY);  
glDisableClientState(GL_VERTEX_ARRAY);
```

Le Graal : les VBO

Les géométries, en encore encore mieux : `VertexBufferObject`

Depuis OpenGL1.5 on peut se servir des `VertexBufferObject`. L'intérêt de ces petites bêtes est que l'on peut stocker directement la géométrie sur la carte graphique. Comme pour les `VertexArrays`, on utilise des tableaux pour stocker la géométrie d'un objet, on lie ensuite ces tableaux avec des zones mémoires de la carte graphique, on copie les données, et hop, c'est prêt ! (du coup on peut désallouer la mémoire en RAM du CPU puisque la géométrie est sur la carte graphique). Il devient donc inutile de conserver la géométrie à la fois dans le CPU et le GPU, d'où un gain de mémoire en plus du gain de transfert.

Le mode des VBO

On peut gérer les VBO sous 3 modes :

- `GL_STREAM_DRAW` lorsque les informations sur les sommets peuvent être mises à jour entre chaque rendu = VertexArrays. On envoie les tableaux à chaque image.
- `GL_DYNAMIC_DRAW` lorsque les informations sur les sommets peuvent être mises à jour entre chaque frame. On utilise ce mode pour laisser la carte graphique gérer les emplacements mémoires des données, pour le rendu multi-passe notamment.
- `GL_STATIC_DRAW` lorsque les informations sur les sommets ne sont pas mises à jour - ce qui redonne le fonctionnement d'une `DisplayList`.
- Se rajoute à ces modes, des modes d'accès aux données : `READ`, `WRITE`, `COPY`, `READ_WRITE`. Ce qui permet également de copier des parties d'un objet dans un autre, et même de faire des interactions avec le CPU.

note : attention aux différents modes, cela ne fonctionne pas sur toutes les cartes graphiques.

Comment fait-on cela ?

Préparation des données :

- 1 on crée les tableaux de géométrie comme pour les VertexArray

```
float lcolors[nbsommets*3];  
float lnormales[nbsommets*3];  
float lsommets[nbsommets*3];
```

- 2 on crée les buffers GPU qui contiendront ces données
(glBindBuffer, glBufferData)

Création de VBO

```
//VBO pour les sommets
glGenBuffers((GLsizei) 1, &VBOSommets);
    glBindBuffer(GL_ARRAY_BUFFER, VBOSommets);
    glBufferData(GL_ARRAY_BUFFER, nbfaces * 9 * sizeof(float), lsommets, GL_STATIC_DRAW);

//VBO pour les couleurs
glGenBuffers((GLsizei) 1, &VBOCouleurs);
    glBindBuffer(GL_ARRAY_BUFFER, VBOCouleurs);
    glBufferData(GL_ARRAY_BUFFER, nbfaces * 9 * sizeof(float), lcolors, GL_STATIC_DRAW);

//VBO pour les normales
glGenBuffers((GLsizei) 1, &VBONormales);
    glBindBuffer(GL_ARRAY_BUFFER, VBONormales);
    glBufferData(GL_ARRAY_BUFFER, nbfaces * 9 * sizeof(float) , lnormales, GL_STATIC_DRAW);
```

Rendu de VBO

```
glEnableClientState(GL_VERTEX_ARRAY);
glEnableClientState(GL_COLOR_ARRAY);
glEnableClientState(GL_NORMAL_ARRAY);

glBindBuffer( GL_ARRAY_BUFFER, VBOSommets);
glVertexPointer( 3, GL_FLOAT, 0, (char *) NULL );

glBindBuffer( GL_ARRAY_BUFFER, VBOCouleurs);
glColorPointer( 3, GL_FLOAT, 0, (char *) NULL );

glBindBuffer( GL_ARRAY_BUFFER, VBONormales);
glNormalPointer(GL_FLOAT, 0, (char *) NULL );

glDrawArrays(GL_TRIANGLES, 0, monobjet.nbfaces * 3);

glDisableClientState(GL_NORMAL_ARRAY);
glDisableClientState(GL_COLOR_ARRAY);
glDisableClientState(GL_VERTEX_ARRAY);
```

Les VBO

Et pour les images : le FBO

Un Frame Buffer Object (FBO pour les intimes) est l'espace de stockage qu'utilise OpenGL pour mettre, au fur et à mesure de son avancement, toutes les données issues d'algorithmes de l'espace image (lié à la résolution du rendu, en pixels). Par exemple les pixels de la scène, mais aussi le Z-buffer. Dans un fonctionnement statique, vous devez utiliser des stencils ou des masques pour interagir avec le Z-buffer, et vous ne pouvez pas modifier les pixels de l'image 2D rendue.

À partir d'OpenGL 2, on peut adresser ces espaces de stockages comme des textures (Texture Object) ou des buffers (RenderBuffer Object). Pour schématiser, les premiers sont faciles d'accès (par le programme principal ou un shader) et les seconds sont plus rapides (stockés en mémoire vive de la carte graphique, pas dans l'espace des textures).

Création d'un FBO

Pour créer un FBO, c'est comme pour les textures et les VBO, il faut générer une référence sur la carte graphique :

- 1 obtenir un identifiant

```
void glGenFramebuffersEXT(GLsizei quantité, GLuint* idfbo)
```

Création d'un FBO

Pour créer un FBO, c'est comme pour les textures et les VBO, il faut générer une référence sur la carte graphique :

- 2 Pour attacher une texture à ce FBO, on écrit :

```
glFramebufferTexture2DEXT(GLenum target,  
                           GLenum attachmentPoint,  
                           GLenum textureTarget,  
                           GLuint textureId,  
                           GLint level)
```

avec :

- target = GL_FRAMEBUFFER_EXT,
- attachmentPoint dans GL_COLOR_ATTACHMENT0_EXT, ..., GL_COLOR_ATTACHMENTn_EXT, GL_DEPTH_ATTACHMENT_EXT, GL_STENCIL_ATTACHMENT_EXT selon le FBO à récupérer.
- textureTarget = GL_TEXTURE_2D,
- textureId = identifiant de votre texture
- level = ?, comme vous voulez, c'est le niveau de mipmapping de la texture finale.

Rendu d'un VBO

On active ensuite le FBO dans la boucle de rendu par :

```
void glBindFramebufferEXT(GLenum target, GLuint id)
```

avec target = `GL_FRAMEBUFFER_EXT`. Si on met target = 0, le FBO est désactivé.

Note : faire du rendu (`off-screen`) directement dans une image ou un fichier AVI -> c'est le pbuffer, et c'est différent.

Programmation OpenGL

Dépendances :

- pour la gestion des fichiers images : FreeImage, bibliothèque open-source qui manipule env. 30 types de fichiers,
<http://freeimage.sourceforge.net/>, paquets libfreeimage3 et libfreeimage-dev,
- pour la gestion des extensions OpenGL : GLEW, bibliothèque open-source qui permet de vérifier les capacités de la carte graphique et de l'implémentation OpenGL (ARB_...), <http://glew.sourceforge.net/>, paquets libglew1.6 et libglew1.6-dev,
- pour la gestion dans l'environnement fenêtré (fenêtre, interactions, contexte OpenGL) : GLUT,
<http://freeglut.sourceforge.net/>, paquets freeglut3 et freeglut3-dev .