

Reconstruction de Surfaces à partir de nuages de points

Frédéric CERDAT
fredocom@hotmail.com

31 mars 2002

Table des matières

1	Introduction	2
2	Algorithme α-shape basique	2
2.1	Idée	2
2.2	Calcul du rayon du cercle circonscrit	2
2.3	Réalisation	3
2.4	Interface	3
2.5	Résultats	4
2.6	Critique et améliorations possibles	5
3	Méthode des Métaballes	6
3.1	Interpolation de surfaces implicites	6
3.2	Technique des Métaballes 3D	7
3.3	L'Algorithme Marching-Cubes (AMC)	7
3.4	Détermination des normales pour les calculs d'éclairage	9
3.5	Résultats de la méthode	9
3.6	Résultats	11

1 Introduction

Cet article a pour objectif de développer deux méthodes de reconstruction de formes à partir de nuages de points. Il est issu d'un rapport que j'ai réalisé avec un ami dans le cadre d'un DEA d'imagerie. Je mets également à disposition le code source C/OpenGL implémentant ces méthodes disponible sur le site <http://progzone.free.fr>. Je tiens à remercier Nicolas Hautière pour sa participation à part égale sur ce travail ainsi que Gabriel Peyré alias Nikopol0 [4] pour ses supers articles. Pour toutes informations ou remarques relatives à cet article, prière de se référer en priorité au forum de la progzone.

2 Algorithme α -shape basique

2.1 Idée

L'idée de base du programme est de faire rouler une sphère de rayon R que l'on fait varier, à l'image d'un potentiomètre, sur la surface de l'objet et de ne représenter que les triangles qui répondent au critère suivant : le rayon du cercle circonscrit à la facette est inférieur à R . On comprend bien que l'on génère une surface «trouée» et facettisée.

2.2 Calcul du rayon du cercle circonscrit

Soient $P_1(x_1, y_1, z_1)$, $P_2(x_2, y_2, z_2)$, $P_3(x_3, y_3, z_3)$ et , les points constituant une facette. Le centre du cercle circonscrit est caractérisé par :

$$\begin{aligned}\forall(i, j) \in \{1, 3\}^2, \overrightarrow{P_i P_j} \cdot \overrightarrow{M_{ij} C} &= 0 \\ \overrightarrow{P_1 C} \cdot (\overrightarrow{P_1 P_2} \wedge \overrightarrow{P_1 P_3}) &= 0\end{aligned}$$

D'où le système d'équations :

$$\begin{bmatrix} 2x_2 - 2x_1 & 2y_2 - 2y_1 & 2z_2 - 2z_1 \\ 2x_3 - 2x_1 & 2y_3 - 2y_1 & 2z_3 - 2z_1 \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = \begin{bmatrix} x_2^2 - x_1^2 + y_2^2 - y_1^2 + z_2^2 - z_1^2 \\ x_3^2 - x_1^2 + y_3^2 - y_1^2 + z_3^2 - z_1^2 \\ x_1 \times A_{31} + y_1 \times A_{32} + z_1 \times A_{33} \end{bmatrix}$$

Avec :

$$\begin{aligned}A_{31} &= (y_2 - y_1)(z_3 - z_1) - (z_2 - z_1)(y_3 - y_1) \\ A_{32} &= (z_2 - z_1)(x_3 - x_1) - (x_2 - x_1)(z_3 - z_1) \\ A_{33} &= (x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1)\end{aligned}$$

De là, le rayon de la sphère circonscrite est :

$$r = \sqrt{(x_1 - x_c)^2 + (y_1 - y_c)^2 + (z_1 - z_c)^2}$$

2.3 Réalisation

Le programme (hors interface) est écrit en C++ (projet **alpha shape**) et est essentiellement construit autour d'une Classe Triangle dont le constructeur implémente directement le calcul du rayon de la sphère circonscrite. L'initialisation du programme consiste à charger les coordonnées 3D des points ainsi que les coordonnées des différentes facettes contenus dans des fichiers textes externes et à construire, via la Classe Triangle, un tableau de Triangles, contenant les coordonnées des trois sommets et le rayon de la sphère circonscrite. Ainsi en faisant varier le «potentiomètre» (cf. interface), il suffit de trier les facettes contenues dans le tableau et à afficher les facettes répondant au critère. La rapidité du programme vient de là.

2.4 Interface

Le programme fournit une interface de visualisation OpenGL des formes reconstituées. Les touches x, y et z (resp. X,Y et Z) entraînent la rotation dans le sens direct resp. indirect) de la forme selon les axes principaux du même nom. Les touches i ou I provoquent le retour de la forme dans sa position initiale. Enfin, les touches + et -, permettent d'augmenter le rayon R de la sphère de référence. Ces deux dernières touches jouent le rôle d'un potentiomètre. Les flèches directionnelles permettent de translater la forme dans le plan (x, z) .

Pour faire tourner un programme utilisant les librairies OpenGL et GLUT sous Windows, il est nécessaire d'avoir les dll suivantes :

- opengl.dll (ou opengl32.dll sous Windows) : librairie principale d'OpenGL.
- glu.dll (ou glu32.dll sous Windows) :librairie de fonctions annexes.
- glut.dll (ou glut32.dll sous Windows) : librairie du GLUT (pour le fenêtrage).

Pour programmer, il faut les fichiers de librairies et d'en têtes suivants :

- Librairies : opengl.lib, glu.lib, glut.lib, opengl32.lib, glu32.lib, glut32.lib
- En-têtes : gl.h, glut.h, glu.h

Ces fichiers sont à placer :

- Pour les librairies : Dans le répertoire lib du compilateur
- Pour les en-têtes : Dans le répertoire include du compilateur
- Pour les dll : Dans le répertoire system de windows

Les fichiers glut sont fournis sur le cédérom dans le fichier **glut 3.7.zip**.

Pour compiler un projet Visual C++, il faut rajouter dans le menu Project → Settings → Link dans le champ object/library modules, les librairies suivantes `opengl32.lib glu32.lib glut32.lib`

2.5 Résultats



FIG. 1 – Lapin reconstitué : R petit



FIG. 2 – Lapin reconstitué : R medium



FIG. 3 – Lapin reconstitué : R grand

2.6 Critique et améliorations possibles

Le programme fonctionne bien et permet de comprendre de manière simplifiée les α -shapes (2-simplexes). Il a donc un intérêt pédagogique certain. Cependant ce programme suppose de connaître une triangulation de la surface. L'intérêt du programme, en tant que technique de reconstruction se trouve donc diminué.

Pour répondre à cette critique, une amélioration possible du programme serait de considérer une triangulation 3D de Delaunay de la forme, comme cela est décrit dans la partie précédente et d'appliquer le programme aux triangles constituant les facettes des tétraèdres. Étant donné que les points sont sur la surface, les triangles 2D ayant les rayons de leur cercle circonscrit les plus petits sont situés sur la surface. Ainsi, en choisissant bien le rayon R de la sphère de référence, on ne devrait afficher que les facettes reconstituant la surface. Par manque de temps, cette idée n'a pas pu être explorée.

Il est également possible de considérer tous les triplets de points possibles. On se heurte dans ce cas à une explosion combinatoire de nombre de triangles à considérer. Pour 1500 points, le nombre de triangles est supérieur à un demi-milliard. L'idée n'est donc pas raisonnable.

Il est également facilement envisageable de créer une deuxième sphère de référence de rayon R' , avec $R' < R$ et de n'afficher que les triangles dont le rayon de la sphère circonscrite est compris entre R et R' . On aurait ainsi à une topologie de la triangulation de la surface de la forme envisagée.

3 Méthode des Métaballes

3.1 Interpolation de surfaces implicites

L'étape préliminaire à la reconstitution de formes par métaballes est la détermination de l'équation implicite de la surface formée par le nuage de points dont on dispose. Morse, Yoo, Rheingans, Chen et Subramanian décrivent dans [2] une méthode astucieuse de création de surfaces implicites en utilisant des combinaisons linéaires de fonctions de base. Les fonctions de base à support compact suivantes possèdent des propriétés intéressantes :

$$\Phi(r) = \begin{cases} (R - r)^2 & \text{si } r < R \\ 0 & \text{sinon} \end{cases}$$

De par le caractère borné de leur support, ces fonctions permettent notamment un gain de temps dans l'exécution de la méthode. On définit l'équation implicite représentant notre surface comme une combinaison linéaire de ces fonctions de bases par la formule suivante :

$$F(x) = \sum_{i=1}^n d_i \times \Phi(\| \bar{x} - \bar{x}_i \|) = 1$$

Où les \bar{x}_i sont les vecteurs position des points connus et d_i des coefficients à déterminer. Le but est de trouver les d_i tels que pour tout point \bar{x}_i $F(\bar{x}_i) = 1$. Nous déterminons les coefficients d_i en résolvant le système suivant :

$$\forall i \in (1, n), F(\bar{x}_i) = 1$$

En notant $\Phi_{ij} = \Phi(\| \bar{x}_i - \bar{x}_j \|)$, le système précédent équivaut à :

$$\begin{bmatrix} \Phi_{11} & \Phi_{12} & \dots & \Phi_{1n} \\ \Phi_{21} & \Phi_{22} & \dots & \Phi_{2n} \\ \vdots & & & \vdots \\ \Phi_{n1} & \Phi_{n2} & \dots & \Phi_{nn} \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}$$

Dans notre implémentation, nous avons choisi la résolution du système linéaire la méthode de résolution itérative de Gauss Seidel. Le programme **Gauss_Seidel** effectue la résolution du système d'équations précédent et écrit la liste des coefficients d_i dans un fichier. Avec l'obtention de ces coefficients, nous disposons d'une équation implicite définissant notre surface à reconstruire. Nous pouvons maintenant appliquer l'algorithme de reconstruction proprement dit.

3.2 Technique des Métaballes 3D

Les metaballes sont des iso surfaces, c'est à dire des surfaces de niveau générées à partir de valeurs contenues dans une grille. Elles permettent de modéliser des formes moins anguleuses que les classiques formes polygonales et notamment des formes «organiques». Les metaballes 3d peuvent être générées de différentes manières. Celle qui est le plus souvent utilisée fait intervenir l'algorithme des marching-cubes. Cet algorithme va créer le mesh des metaballes (la surface) à partir d'une grille à 3 dimensions qui contient les valeurs de la fonction implicite définissant notre forme.

Le reste de la méthode va consister à «relier» les points de la grille de valeur 1 (ces points vérifient l'équation $F(\vec{x}_i) = 1$ donc appartiennent à la surface). Pour cela, on applique l'algorithme des Marching-cubes.

3.3 L'Algorithme Marching-Cubes (AMC)

Cet algorithme va construire l'objet 3D à partir de la grille. Le principe est assez simple, la grille forme de petits cubes. Chaque cube contient 8 isovaleurs (une pour chaque sommet). Le cube est aussi formé de 12 arêtes sur lesquelles se trouveront les sommets du mesh formant la future surface. L'AMC va scanner chaque cube et va créer les sommets/triangles correspondants. Il y'a un maximum de 5 triangles par cube. Bien sûr, il peut également n'y en avoir aucun.

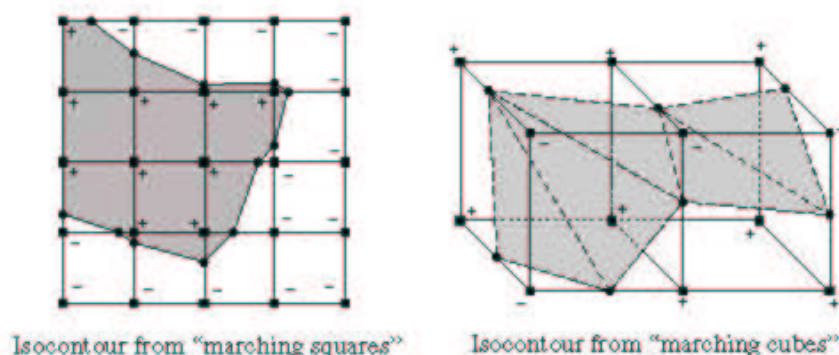


FIG. 4 – Algorithme AMC

Considérons un cube en particulier. La méthode employée par l'algorithme consiste à tracer un polygone qui est en fait l'intersection de notre cube et du plan qui modélise localement notre surface. Si le polygone formé

par l'intersection ne coupe que trois côtés, on a un triangle. Mais s'il en coupe 4 ou 5 ? On comprend intuitivement qu'il va nous falloir trouver une méthode pour déterminer précisément ce polygone.

Fort heureusement, tous les cas d'intersection (soit 256 possibilités mais avec les symétries, on se ramène à 8 cas de base) ont été répertoriés par les concepteurs de l'algorithme qui les ont décrites dans deux tableaux. Le premier tableau, intitulé «EdgeArray» permet de déterminer quels côtés la surface traverse. Le second tableau, intitulé «FaceArray» permet de déterminer quels points il faut relier pour tracer le bon polygone. Encore faut-il savoir utiliser ces tableaux.

La démarche à suivre est très simple. A chaque côté de notre case et à chaque sommet est affecté un numéro :

Ensuite pour simplifier les calculs, on travaille en binaire. On considère le

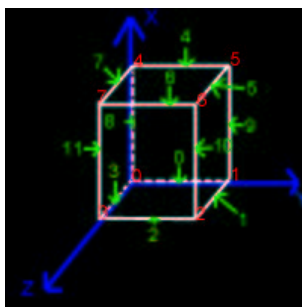


FIG. 5 – Intersection

sommet numéro i . Si sa valeur est supérieure à 1, on met le i -ème bit d'un index à 0, sinon on le met à 1. On obtient ainsi une valeur qui détermine la case de EdgeArray qu'il convient d'utiliser. Cette valeur nous donne de la même façon les côtés de la case qui nous intéressent (ceux que la surface intersecte). Si le bit 0 est à 1, c'est que le côté 0 nous intéresse, et ainsi de suite jusqu'au douzième côté (le numéro 11).

Pour déterminer quels triangles tracer (le polygone interceptant le cube et le plan sera la réunion de ces triangles), il suffit de se placer sur la case numéro $\langle \text{index} \rangle$ de FaceTab : elle donne une liste de vertex, qui doivent être groupés trois par trois, une valeur de -1 indiquant que l'on doit s'arrêter.

3.4 Détermination des normales pour les calculs d'éclairage

Cette partie n'est pas un élément indispensable dans la reconstruction d'une forme. Néanmoins, ce sont les différences d'éclairage qui donneront la sensation de relief 3D à la forme. D'où l'importance d'obtenir des valeurs de normales cohérentes en chaque point de la surface (en fait, à chaque sommet du polygone). Pour cela, nous allons utiliser une propriété fondamentale des champs scalaires différentiables : le vecteur gradient est orthogonal à la surface.

$$\overrightarrow{grad}(F)(x, y, z) = \left(\frac{\partial F}{\partial x}(x, y, z), \frac{\partial F}{\partial y}(x, y, z), \frac{\partial F}{\partial z}(x, y, z) \right)$$

Nous approximations ces dérivées partielles au niveau de chaque sommet du cube :

$$\begin{aligned} \frac{\partial F}{\partial x}(x, y, z) &= \frac{F(x+1, y, z) - F(x-1, y, z)}{2} \\ \frac{\partial F}{\partial y}(x, y, z) &= \frac{F(x, y+1, z) - F(x, y-1, z)}{2} \\ \frac{\partial F}{\partial z}(x, y, z) &= \frac{F(x, y, z+1) - F(x, y, z-1)}{2} \end{aligned}$$

Maintenant, pour un point quelconque de notre polygone, comme celui-ci se trouve sur une arête de la case, il ne reste plus qu'à interpoler linéairement la valeur des normales aux deux extrémités.

3.5 Résultats de la méthode

Cette méthode présente les avantages principaux d'être peu gourmande en mémoire (puisque la surface est définie essentiellement à l'aide d'une courbe mathématique) et d'être calculable à différents niveaux de détail (il suffit pour cela de diminuer le pas de la grille de calcul).

Nous avons appliqué cette méthode à la reconstruction d'un coeur humain et à celle du lapin Bunny de Stanford. Concernant le coeur humain, cette méthode fournit d'excellents résultats avec des temps d'exécution très raisonnables (de l'ordre de quelques minutes sur un PC standard). Concernant le lapin Bunny, les résultats sont plus mitigés dans la mesure où la surface reconstituée présente des trous. Ces trous sont dus à une trop faible densité des points échantillonnés comparativement à la surface étudiée. Une solution évidente consisterait à augmenter la densité des points représentant la surface.

Les projets **coeur** et **bunny** fournissent une interface de visualisation OpenGL des formes reconstituées. Les touches x, y et z (resp. X, Y et Z)

entraînent la rotation dans le sens direct resp. indirect) de la forme selon les axes principaux du même nom. Les touches i ou I provoquent le retour de la forme dans sa position initiale. Les flèches directionnelles permettent de translater la forme dans le plan (x, z) . Nous avons également placé un compteur fps (frames per second) qui donne une idée des ressources système utilisées. A noter que les calculs ne sont réellement exécutés qu'une seule fois à l'initialisation du programme. Une fois ces calculs effectués, le programme stocke les résultats dans une liste d'affichage donc sous forme pré compilée. Le compteur fps est un indicateur de la complexité du processus d'affichage de la forme mais pas de la complexité du processus de reconstruction proprement dit.

3.6 Résultats

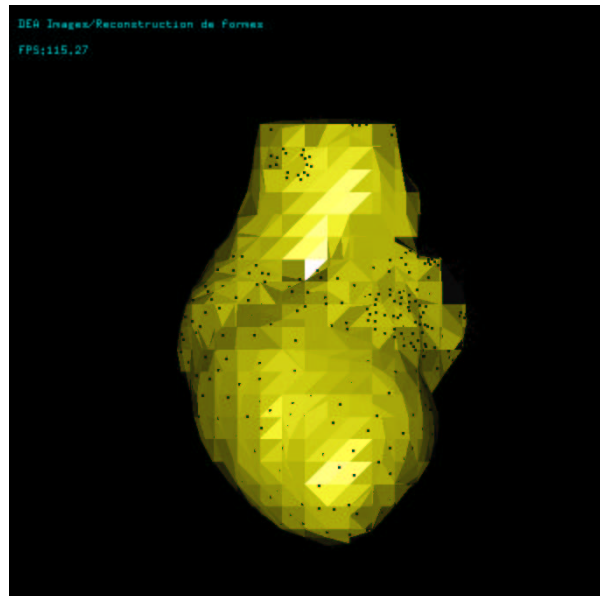


FIG. 6 – Coeur humain reconstitué par la méthode des metaballes : faible résolution de la grille

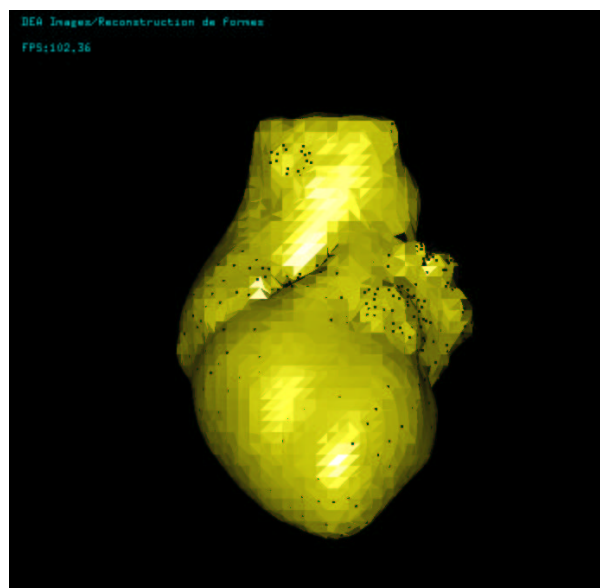


FIG. 7 – Coeur humain reconstitué par la méthode des metaballes : résolution médiane de la grille

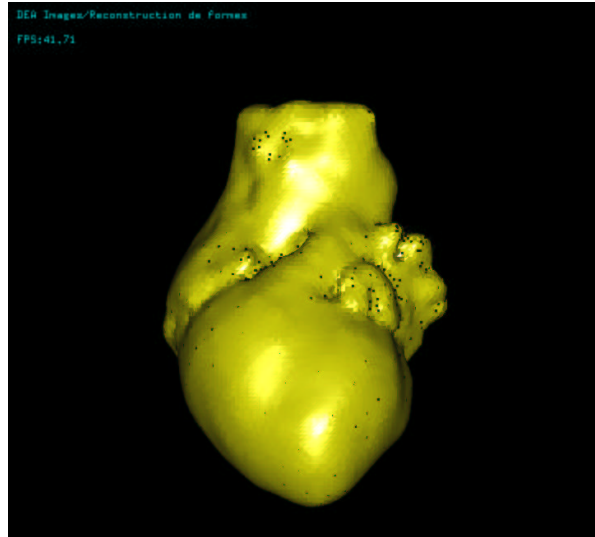


FIG. 8 – Coeur humain reconstitué par la méthode des métaballes : forte résolution de la grille

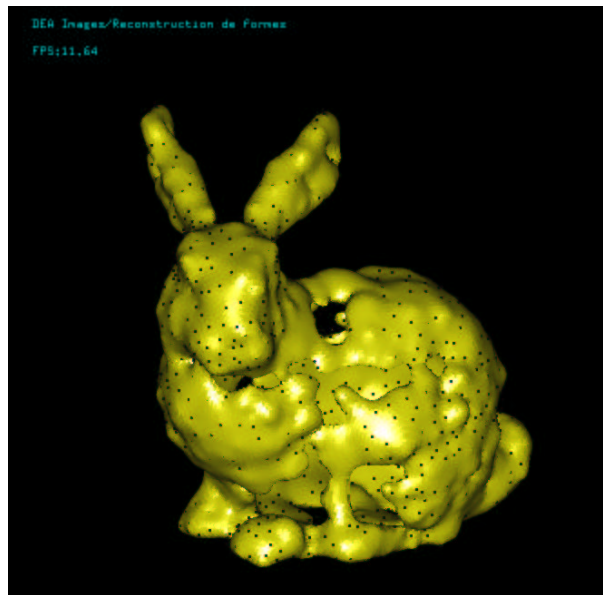


FIG. 9 – Lapin de Stanford reconstitué par la méthode des métaballes

Références

- [1] BLINN *A generalization of algebraic surface drawing* 1982
- [2] MORSE, YOO, RHEINGANS, CHEN, SUBRAMANIAN *Interpolating Implicit Surfaces From Scattered Surface Data Using Compactly Supported Radial Basis Functions* 2001
- [3] AMENTA, BERN, KAMVYSSELIS *A New Voronoi-Based Surface Reconstruction Algorithm* 1998
- [4] PEYRE Gabriel *Implementation de l'algorithme Marching Cubes*
[http ://www.orion3d.fr.st](http://www.orion3d.fr.st)
- [5] EDELSBRUNNER, MUCKE *Three-dimensional Alpha Shapes* 1994
- [6] HOPPE, DE ROSE, DUCHAMP, Mc DONALD, STUETZLE *Surface Reconstruction from Unorganized Points* 1994
- [7] LORENSEN, CLINE *Marching Cubes : a high resolution 3D surface reconstruction algorithm* 1992
- [8] MENON *An Introduction to Implicit Techniques* 1996
- [9] GUO, MENON, WILLETTE *Surface Reconstruction Using Alpha Shapes* 1996
- [10] BAJAJ, BERNARDINI, XU *Automatic Reconstruction of Surfaces and Scalar Fields from 3D Scans* 1995