

SURFACES NURBS PERFOREES EN TEMPS REEL ---

INTRODUCTION AUX COURBES ET AUX SURFACES	2
INTRODUCTION AUX B-SPLINES	2
<i>Introduction</i>	2
<i>Différents types de B-Splines</i>	4
SURFACES ET COURBES DE TYPE NURBS	5
<i>Approche mathématique générale</i>	5
<i>Dérivée en tout point de la courbe</i>	6
RENDU EN TEMPS REEL DES SURFACES NURBS	7
<i>Généralités</i>	7
<i>Tessellation de la surface</i>	8
<i>Conclusion</i>	9
PERFORATION DES SURFACES	9
<i>Perforer la surface</i>	9
<i>Affiner le contour des trous</i>	10
<i>Effets indésirables et autres choix écartés</i>	11
RENDU EN TEMPS REEL DES SURFACES NURBS PERFOREES	12
<i>Etapas critiques</i>	12
<i>Eliminer les deux passes</i>	12
CONCLUSION.....	12
BIBLIOGRAPHIE	13
FICHIERS ACCOMPAGNANT L'ARTICLE, REMERCIEMENTS, ME CONTACTER	13

Introduction aux courbes et aux surfaces

Toute courbe ou surface peut être décrite par un ensemble de fonctions paramétriques. Par exemple, les coordonnées (x,y,z) des points d'une courbe peuvent être données par :

$$x = X(t), y = Y(t), z = Z(t)$$

t étant le paramètre variant sur la courbe et X, Y, Z étant des fonctions polynomiales en t par exemple.

Si X, Y et Z sont des fonctions polynomiales du 1^{er} degré, un segment de droite sera décrit. Dans ce cas, seulement deux données (i.e. deux points) seront nécessaires pour décrire cette courbe. Si ce sont des fonctions polynomiales du 2nd degré, un segment de parabole sera décrit et trois données seront nécessaires pour le décrire (i.e. trois points ou deux points et une tangente). Pour des polynômes de degré plus élevé, plus de données seront nécessaires pour décrire la courbe. Ce nombre de données est ce qu'on appelle l'ordre de la courbe et il est toujours égal au degré du polynôme plus un.

Le plus souvent, les courbes sont représentées par des polynômes de degré 3. En effet, des polynômes de degré supérieur introduisent des données supplémentaires, ce qui rend difficile la création de telles courbes. A contrario, des polynômes de degré inférieur sont trop restrictifs dans les courbes qu'ils décrivent puisque celles-ci seront soit des droites soit des paraboles et seront de toutes façons planaires.

Différentes approches ont été considérées par les mathématiciens et les informaticiens, parmi les plus connues, nous pouvons citer les courbes de Bézier, de Hermite, les splines de Catmull-Rom et les B-Splines que nous étudierons plus particulièrement. Pour plus de détails sur ces différents types de courbe, se référer à [CGPP].

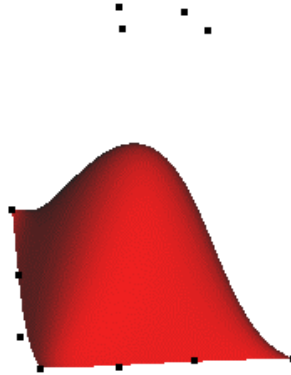
Introduction aux B-Splines¹

Introduction

Le terme spline renvoie aux longues bandes de métal utilisées pour modeler les surfaces composant les avions, les carrosseries de voiture ou les bateaux. Des poids étaient utilisés pour tirer les splines dans diverses directions et leur donner la forme voulue. Par analogie, un modèle mathématique a été élaboré et permet de définir une courbe interpolant des points de contrôle, définie par des polynômes. Ces courbes ont été appelées splines naturelles.

¹ Ce texte reste une introduction sommaire ; les problèmes de modélisation n'entrant pas dans le cadre de cette étude limitée aux problèmes d'affichage en temps réel, certains points tels que la continuité ne sont pas abordés. Pour une théorie plus complète sur les B-Splines, les problèmes de continuité entre courbes et surface et l'approche mathématique, se référer à [CGPP].

Le principal désavantage que possède une spline naturelle réside dans le fait qu'une action sur un des points de contrôle influence toute la courbe. En effet, les coefficients des polynômes dépendent de chacun des points de contrôle.



Surface basée sur des B-Splines

Les points affichés sont les points de contrôle (16 au total dont 5 invisibles ici)

Les B-Splines que nous allons étudier ici consistent en des segments de splines naturelles, chacun défini par un nombre réduit de points de contrôle. Ainsi, les coefficients des polynômes ne dépendent que des points de contrôle du segment courbe considéré. Nous parlons alors de contrôle local.

Les schémas ci-dessous² illustre la manière dont influent les points de contrôle sur la forme de la courbe.

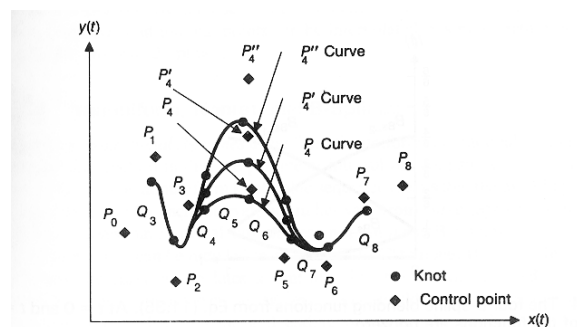
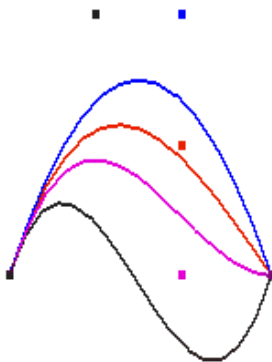


Fig. 11.23 A B-spline with control point P_4 in several different locations.

² Le schéma de droite est tiré de [CGPP].

Remarque : nous considérerons par la suite que la courbe est de degré n quelconque.

Différents types de B-Splines

Les B-Splines interpolent un ensemble de $p+1$ points de contrôle : $\{P_0, P_1, \dots, P_p\}$, $p \geq n$ et sont constituées de $p-(n-1)$ segments de courbe : $\{Q_n, Q_{n+1}, \dots, Q_p\}$. Nous pouvons de plus définir un paramètre t commun à chaque segment de courbe plutôt que de considérer t dans l'intervalle $[0, 1[$ pour chacun des segments. Ainsi, pour chaque segment de courbe Q_i , t sera défini sur un intervalle $[t_i, t_{i+1}[$, $n \leq i \leq p$. De plus, chaque segment Q_i sera influencé par les points de contrôle P_{i-n} à P_i .

Pour chaque $i \geq n$, il y a un noeud entre Q_i et Q_{i+1} pour la valeur t_i du paramètre t . Au total, il y a $p-n-2$ noeuds pour la B-Spline. C'est ici que nous pouvons introduire la notion d'uniformité : si les noeuds sont uniformément répartis sur l'intervalle $[0, 1[$ (i.e. $\forall i \in [n, p], t_{i+1} - t_i = t_{i+2} - t_{i+1}$), la B-Spline sera dite uniforme. Dans le cas contraire, on utilisera le terme non-uniforme. Nous pouvons noter au passage que cette définition des noeuds implique qu'ils soient rangés en ordre croissant (i.e. $\forall i \in [n, p], t_i \leq t_{i+1}$).

Si nous considérons maintenant que les coordonnées (x, y, z) d'un point de la courbe ne sont pas données par une fonction polynomiale mais par une fonction quotient de deux polynômes :

$$x = \frac{X(t)}{W(t)}, y = \frac{Y(t)}{W(t)}, z = \frac{Z(t)}{W(t)}$$

La B-Spline sera dite rationnelle si (x, y, z) sont fonctions d'un quotient de polynômes, non-rationnelle dans le cas contraire.

Pour résumer, nous avons abouti à quatre grands types de B-Spline :

- Uniformes Non-rationnelles
- Non-uniformes Non-rationnelles
- Uniformes Rationnelles
- Non-uniformes Rationnelles³

Seul le dernier type de courbe nous intéressera dans cette étude, celui-ci englobant les quatres autres cas.

³ Ce type de B-Spline est souvent appelé NURBS (**N**on-**U**niform **R**ational **B**-**S**pline). Notons que tout type de B-Spline est un cas particulier de NURBS : $W(t) = 1$ pour les non-rationnelles et le cas uniforme est un cas particulier du cas non-uniforme.

Surfaces et courbes de type NURBS

Approche mathématique générale

Une courbe non-uniforme rationnelle Q peut être défini par l'équation paramétrique eq. 1a. De la même façon, par l'équation eq. 1b, nous généralisons l'équation eq. 1a à un espace paramétrique à deux dimensions pour définir une surface non-uniforme rationnelle S .

$$Q(t) = \frac{\sum_{i=0}^p B_{i,n}(t) \cdot P_i \cdot w_i}{\sum_{i=0}^p B_{i,n}(t) \cdot w_i}$$

(eq. 1a)

$$S(u, v) = \frac{\sum_{i=0}^p \sum_{j=0}^q B_{i,m}(u) \cdot B_{j,n}(v) \cdot P_{i,j} \cdot w_{i,j}}{\sum_{i=0}^p \sum_{j=0}^q B_{i,m}(u) \cdot B_{j,n}(v) \cdot w_{i,j}}$$

(eq. 1b)

où P_i est un point de contrôle, w_i le poids lui étant associé et $B_{i,n}$ une fonction de base définie récursivement par :

$$B_{i,0}(t) = \begin{cases} 1 & \text{si } t_i \leq t < t_{i+1} \\ 0 & \text{dans les autres cas} \end{cases}$$

$$\forall k > 0, B_{i,k}(t) = \frac{t - t_i}{t_{i+k} - t_i} B_{i,k-1}(t) + \frac{t_{i+k+1} - t}{t_{i+k+1} - t_{i+1}} B_{i+1,k-1}(t)$$

(eq. 2)

Expliquons un peu plus en détail les différents termes des équations eq. 1a et eq. 1b. La présence d'un dénominateur découle du caractère rationnel de la courbe (resp. de la surface). Si la courbe avait été non

rationnelle, l'équation se serait réduite à $Q(t) = \sum_{i=0}^p B_{i,n}(t) \cdot P_i$. D'autre part, nous pouvons considérer les

fonctions $B_{i,n}$ comme des fonctions de pondération de l'influence des points de contrôle qui vont valoriser de manière plus importante les points de contrôle proches de la valeur de t considérée.

Dérivée en tout point de la courbe

Au final, l'équation eq. 1a nous fourni de quoi tracer une courbe de manière assez aisée. Cependant, que faire pour obtenir la valeur de la dérivée de $Q(t)$? En effet, savoir dériver l'équation eq. 1a revient à savoir dériver $B_{i,n}$, ce qui conduit à une impasse étant donné la forme de $B_{i,n}$. La même constatation peut être faite pour l'équation eq. 1b, l'obtention des dérivées partielles étant encore plus ardue. Nous allons donc nous souvenir que $B_{i,n}$ peut aussi s'écrire sous la forme d'un polynôme. La dérivation de $B_{i,n}$ sous sa forme polynomiale sera alors triviale.

$$B_{i,n}(t) = \sum_{k=0}^n C_{i,n,k}(t) t^k \quad (\text{eq. 3})$$

En combinant les équations 2 et 3, nous arrivons à obtenir une expression des coefficients du polynôme :

$$\begin{aligned} C_{i,0,0}(t) &= B_{i,0}(t) \\ C_{i,n,0}(t) &= \frac{t_{i+n+1}}{t_{i+n+1} - t_{i+1}} C_{i+1,n-1,0}(t) - \frac{t_i}{t_{i+n} - t_i} C_{i,n-1,0}(t) \\ C_{i,n,n}(t) &= \frac{1}{t_{i+n} - t_i} C_{i,n-1,n-1}(t) - \frac{1}{t_{i+n+1} - t_{i+1}} C_{i+1,n-1,n-1}(t) \\ \forall k \in \{1..n-1\}, C_{i,n,k}(t) &= \frac{C_{i,n-1,k-1}(t) - t_i \cdot C_{i,n-1,k}(t)}{t_{i+n} - t_i} - \frac{C_{i+1,n-1,k-1}(t) - t_{i+n+1} \cdot C_{i+1,n-1,k}(t)}{t_{i+n+1} - t_{i+1}} \end{aligned} \quad (\text{eq. 4})$$

Nous remarquons au passage que les coefficients ne dépendent pas de t directement mais simplement de l'intervalle dans lequel il se trouve, ce qui nous arrange grandement pour la suite puisque ces coefficients sont donc constants sur des intervalles de t (entre les noeuds). La dérivée de $B_{i,n}$ par rapport à t est donc maintenant connue :

$$\frac{dB_{i,n}(t)}{dt} = \sum_{k=1}^n k \cdot C_{i,n,k}(t) t^{k-1} \quad (\text{eq. 5})$$

Ce raisonnement sur des polynômes peut être tenu de la même manière dans un espace paramétrique à deux dimensions. Nous obtenons donc aisément les formules pour les surfaces :

$$\frac{dB_{i,n}(u)}{du} = \sum_{k=1}^n k \cdot C_{i,n,k}(u) u^{k-1}$$

(eq. 6a)

$$\frac{dB_{i,n}(v)}{dv} = \sum_{k=1}^n k.C_{i,n,k}(v).v^{k-1}$$

(eq. 6b)

Arrivés à ce stade, nous avons tous les outils nécessaires pour connaître les coordonnées d'un point donné sur la courbe ou la surface. Nous pouvons maintenant songer à effectuer tous ces calculs en temps réel et de manière optimale. Nous nous concentrerons sur les surfaces, les courbes se déduisant aisément de ce qui sera expliqué.

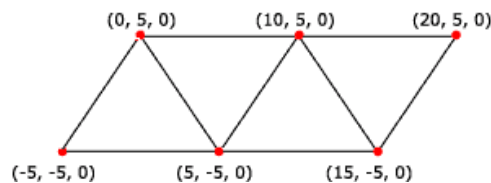
Rendu en temps réel des surfaces NURBS

Généralités

Le rendu d'une surface se fait en deux temps : tout d'abord, calculer les différents points qui vont constituer le maillage de la surface puis, demander l'affichage de ces points.

La première étape, que l'on appelle tessellation, correspond en fait au passage de la forme mathématique continue de la surface à sa forme informatique discrète. Nous choisirons pour des raisons de simplicité et de performances la tessellation uniforme⁴, c'est à dire que nous calculerons des points pour des valeurs régulièrement espacées des paramètres u et v. Cette étape est entièrement indépendante de l'API 3D utilisée étant donné qu'elle ne met en jeu aucun procédé d'affichage.

La seconde étape est dépendante de l'API 3D employée. La méthode la plus efficace pour demander l'affichage des points reste la production de bandes de triangle⁵. Cette interprétation d'un ensemble de points est celle qui permet une utilisation minimale de mémoire, contrairement à l'utilisation de facettes indépendantes.



⁴ D'autres techniques sont envisageables : une tessellation selon la courbure de la courbe permettrait d'ajouter du détail seulement dans les régions où cela est nécessaire. Cependant, une telle méthode est plus difficile à mettre en place et gourmande en ressources.

⁵ Ou "triangle strips" en anglais

La forme géométrique ci-dessus⁶ peut être représentée de deux manières :

- Par un ensemble de triangles (4 triangles définis chacun par 3 sommets, soit 12 triplets nécessaires).
- Par une bande de triangles (une bande de triangle définie par 6 sommets, soit 6 triplets nécessaires).

Dans les deux cas le résultat sera identique mais nous voyons clairement dans le premier cas que des points communs à plusieurs triangles sont stockés de manière redondante. Nous utiliserons dans notre étude OpenGL, mais il faut savoir que Direct 3D permet aussi l'interprétation de ces bandes de triangles. Concentrons nous maintenant sur la tessellation, l'affichage n'étant pas le point critique lors du calcul d'une surface.

Tessellation de la surface

Voyons tout d'abord comment précalculer les valeurs des fonctions de base. C'est ce procédé qui est coûteux lors de la tessellation d'une surface.

Nous avons vu que les coefficients de la forme polynômiale de $B_{i,n}$ sont constants entre deux noeuds. Donc, à moins de changer le vecteur des noeuds, nous n'avons pas à évaluer ces coefficients à chaque tessellation. Voici donc un ensemble de valeurs à calculer lors de la création de la surface, une fois pour toute.

Que peut-on faire au niveau des valeurs des fonctions $B_{i,n}$? Nous savons tout d'abord que, pour une valeur de (i, j) , seulement quelques-unes de ces fonctions sont non-nulles : ce nombre est l'ordre de la surface pour être précis. Pour chaque point qui va être calculé, nous pouvons donc stocker les valeurs de $B_{i,m}(u)$ et $B_{j,n}(v)$. Enfin, nous pouvons multiplier par avance les coordonnées (x, y, z) des points de contrôle par leur poids respectif. Il ne restera donc plus que le calcul de la somme de l'équation *eq. 1b* à effectuer.

Pour le calcul des normales aux points, nous devons calculer les tangentes à la courbe en chaque point selon les directions u et v . La normale est le produit vectoriel de ces deux tangentes. Nous pouvons faire ceci grâce aux équations *eq. 1a*, *eq. 6a* et *eq. 6b*. Nous précalculons de même les valeurs des dérivées partielles de $B_{i,n}$ et il ne restera plus qu'une somme à effectuer lors de la tessellation.

Enfin, nous allons pouvoir mettre en place le procédé de tessellation dynamique, c'est à dire le procédé permettant de changer le nombre de points d'une surface selon sa position par rapport au point de vue : si une surface est loin, elle n'a pas besoin d'être aussi détaillée qu'une surface très proche de l'observateur. Ce procédé est vital lors de l'affichage d'un grand nombre de surface pour ne pas saturer la carte graphique avec des surfaces inutilement trop détaillées.

⁶ Illustration tirée de la documentation de Direct X, (c) Microsoft.

Conclusion

Globalement, l'affichage d'une surface comprendra quatre étapes :

- Détermination du nombre de points à calculer.
- Si ce nombre a changé par rapport au calcul précédent, évaluer de nouveau les valeurs des fonctions $B_{i,n}$ et de leurs dérivées.
- Effectuer la tessellation de la surface.
- Envoyer les points à afficher à la carte graphique par bandes de triangles.

Nous n'avons pas parlé de l'application de textures à la surface. Nous avons ici employé un texturage plan uniforme. Cette méthode a l'avantage d'être simple et rapide. Cependant, elle montre ses limites dans certains cas de surfaces où des déformations peuvent apparaître. Elle reste néanmoins suffisante pour l'application qui nous concerne.

Nous avons aussi considéré que nous n'aurions pas à effectuer la tessellation à chaque calcul d'image, ce qui est valable pour des scènes où l'observateur se déplace peu ou lentement. Nous calculons donc chaque point puis nous les stockons dans un tableau. Si la tessellation devait avoir lieu à chaque image ou presque, dans le cas d'un jeu d'avion ou de voitures par exemple, il serait plus efficace d'envoyer à la carte les bandes de triangle au fur et à mesure du calcul des points, rassemblant du coup les étapes 3 et 4.

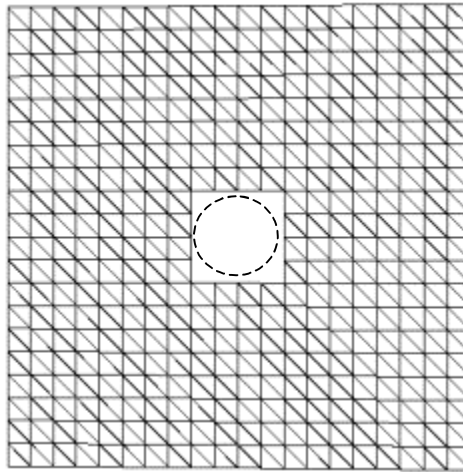
Perforation des surfaces

Deux raisonnements peuvent être tenus pour la perforation des surfaces : bâtir le maillage de triangles en tenant compte des trous par avance ou construire puis modifier le maillage de la surface non trouée pour la perforer par la suite. La première solution est souvent basée sur le calcul de l'intersection du contour du trou avec la surface. Ensuite, un nuage de point est calculé et le maillage est finalement construit. Il faut savoir que le calcul d'intersection entre une courbe et une surface NURBS n'est pas trivial et l'effectuer en temps réel fait encore l'objet de thèses de recherche qui m'ont été impossible de trouver. Il reste donc la seconde solution qui, bien que posant quelques problèmes dans certains cas particuliers, s'avère relativement aisée à mettre en place et extrêmement performante. C'est celle qui sera développée dans ce chapitre.

Perforer la surface

L'idée est la suivante : lors de la tessellation, nous allons mémoriser les points qui sont sur la surface et ceux qui n'y sont pas. Pour ce faire, il suffit de reprendre la tessellation effectuée dans le chapitre précédent et ajouter un test pour chaque point, vérifiant selon les valeurs des paramètres u et v si celui-ci est dans un des trous. Si tel est le cas, nous mémoriserons le trou dans lequel se trouve ce point. Dans le cas contraire, nous indiquons que ce point est bien sur la surface, en mémorisant -1 par exemple. Il faut maintenant aussi modifier la fonction envoyant les points à la carte graphique. L'idée est ici encore simplissime, je ne vais

donc pas l'expliquer ici, le code source se suffisant à lui même. A ce stade de la perforation, nous obtenons la surface suivante avec en pointillé, le trou tel qu'il devrait apparaître.



Le résultat n'est pas très satisfaisant mais nous avons une surface perforée. L'étape suivante va consister à affiner cette surface pour obtenir un trou un peu plus rond.

Note : vous aurez remarqué que les trous sont définis dans le plan paramétrique, et non dans l'espace des objets. Ceci permet de garder la description du trou indépendante de tout paramètre de la surface (points de contrôle, ordre).

Affiner le contour des trous

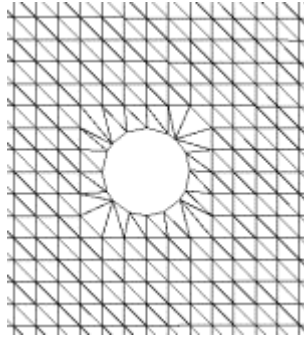
Le principe de l'affinage des trous est simple : il nous faut bouger les points qui sont sur un bord du trou et modifier leurs coordonnées de manière à approcher la forme du trou.

La première question à se poser est comment reconnaître si un point est sur le bord d'un trou et quels points modifier, les points sur la surface, les points du trou que l'on a éliminé ? Tout d'abord, nous pouvons constater qu'un point est au bord d'un trou si au moins un de ses 8 points voisins n'a pas le même état⁷ que lui. Cela suppose de connaître l'état de chacun de ses points, et donc que la perforation des surfaces n'est pas faisable en même temps que le calcul de la surface (ou nous connaissons l'état de 5 des voisins dans une version de base de l'algorithme). La deuxième partie de la réponse à la question est presque triviale : affiner les points frontière qui sont dans le trou conduit à une perte de précision dans le tracé du contour du trou puisque ceux-ci sont moins nombreux que les points frontières sur la surface.

Maintenant que nous savons quels points déplacer, comment allons-nous modifier leurs coordonnées pour approcher la forme du trou ? Nous allons tout simplement rechercher le point du trou qui se rapproche le plus du point à déplacer, tout en se souvenant que ceci se fait dans l'espace paramétrique. Nous aurons ici

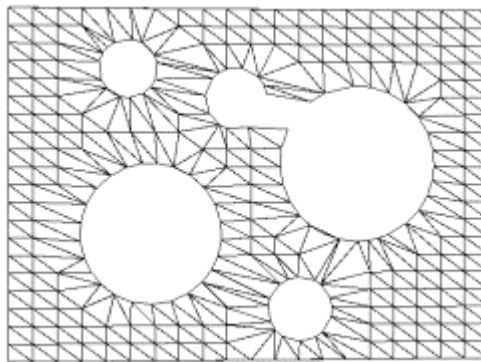
⁷ Rappelez-vous, nous avons mémorisé l'état d'un point auparavant (le trou dans lequel il est ou le cas échéant, -1).

besoin de l'équation *eq. 1b* qui permet de calculer les coordonnées d'un point quel que soient les valeurs des paramètres u et v . Le point obtenu sera alors vraiment sur un bord du trou et nous obtenons un contour fidèle à celui attendu.



Effets indésirables et autres choix écartés

Le premier effet indésirable se produit lorsque les trous d'une surface sont trop rapprochés. Ceci vient du fait qu'entre ces deux trous, aucun point ne permet d'affiner le contour du trou, le maillage étant trop grossier.



Les différentes solutions à ce problème sont :

- Adopter dans ce cas la méthode consistant à affiner les points frontière du trou (la méthode qui a été écarté dans le paragraphe précédent)
- Augmenter le nombre de points de la surface pour que le maillage devienne assez fin entre les trous.
- Changer d'algorithme : vous avez sans doute eu l'idée de recalculer chaque point du trou puis de les relier à la surface. Ce choix a été écarté aux vues des problèmes que pose le fait de reconstituer le réseau de triangles entre les points du trou et les points frontière de la surface en temps réel. Cependant, c'est une méthode qui serait extrêmement efficace et précise, qui corrigerait les effets indésirables dus à la méthode qui a été adoptée au final.

Rendu en temps réel des surfaces NURBS perforées ---

Etapas critiques

Lors de la première version de ces surfaces perforées, deux principaux facteurs de ralentissement ont été identifiés (Au passage, j'en profite pour vous conseiller l'emploi de profilers pour identifier les étapes critiques de vos programmes) :

- Le test d'appartenance d'un point à un contour
- Le calcul des coordonnées du point affiné

En effet, le test d'appartenance d'un point à un contour était basé sur la méthode la plus intuitive : un point est à l'intérieur d'un polygone si la somme des angles formés avec les points du polygone est 2π rad. Une méthode plus efficace a donc été cherchée puis trouvée dans les FAQs du forum `comp.graphics.algorithms`. Ce point éliminé, il en restait un encore plus gourmand en ressources : le calcul des coordonnées du point affiné.

Cette partie de l'algorithme était inefficace car elle utilisait la formule récursive de l'équation *eq. 2*. Etant donné que les contours n'ont qu'un nombre limité de points, la méthode consistant à précalculer ces points est apparue tout naturellement.

Eliminer les deux passes

Il reste cependant un problème non encore résolu : comment affiner la surface en même temps que l'on calcule les points qui la compose ? En effet, nous effectuons pour le moment deux passes pour calculer et perforer la surface, ce qui est peu efficace. Hors, nous pouvons remarquer que, après la 3e colonne et la 3e ligne, nous connaissons les 8 voisins du point situé une colonne et une ligne avant le point courant. Nous allons donc pouvoir calculer et affiner la majorité des points en une passe. Nous affinons les points restants dans une seconde passe.

Conclusion ---

Nous voilà arrivé au bout de cet article sur les surfaces perforées. Je suis persuadé qu'il reste de nombreuses optimisations à faire tant au niveau algorithmique qu'au niveau programmation. Le code source est disponible ci-dessous, documenté avec [Doxygen](#) (que je vous encourage à utiliser dès que possible). Au passage, un article sur cet utilitaire est disponible [ici](#).

Je serais heureux de recevoir des suggestions, remarques ou un retour sur cet article.

Bibliographie

- [CGPP] Foley, VanDam, Feiner et Hughes, "*Computer Graphics: Principles & Practice - 2nd edition*", Addison-Wesley, 1996
- [VRML] Grahm, Volk et Wolters, "*NURBS in VRML*", Blaxxun Interactive
- [UNRT] Dean Macri, "*Using NURBS Surfaces In Real-time Applications*", www.gamasutra.com, Nov. 1999

Fichiers accompagnant l'article, remerciements, me contacter

Repertoire /prog : exemple d'implementation (en anglais, je ne sais pas programmer en français, désolé). Un projet Visual C++ 6 et un fichier makefile. Ce programme est normalement portable, du moins j'ai tout fait pour.

Repertoire /prog/bin : executables et dll de l'exemple (précompilés)

Repertoire /prog/doc : documentation du code source au format .chm

Je tiens à remercier allergy pour sa patience et pour le portage du programme vers linux. Visitez son site web pour plus de docs programmation en français : www.alrj.org

Pour me contacter, vprat@ifrance.com et vous pouvez d'ailleurs visiter mon modeste site vprat.ifrance.com

© 2001 Vincent PRAT.

Vous pouvez distribuer ce document librement SANS modifications de votre part.

Ne partagez pas le fichier zip, n'enlever aucun fichier, ne corrigez pas cet article sans ma permission.