TP Synthèse d'images: Courbes et surfaces paramétriques
CPE

durée - 4h

4ETI IMI, 2014

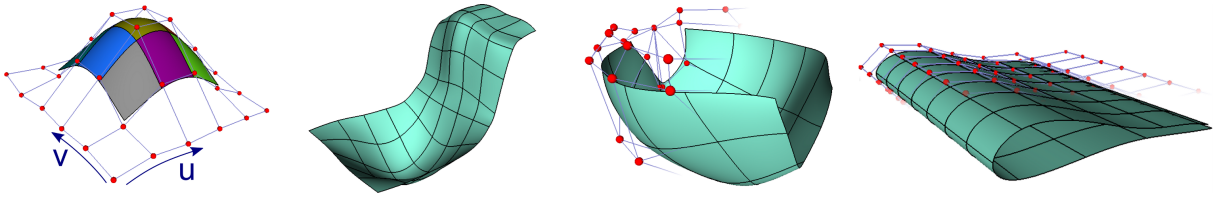

FIGURE 1 – Exemples de résultats de modélisation pour fonctions paramétriques de type Spline.

1 But du TP

Le courbes et surfaces paramétriques permettent de modéliser des objets lisses C^1, ou C^2. Les surfaces obtenues par raccords de *carreaux* (ou *patchs*) sont à la bases des logiciels de CAO et de modélisation d'objets manufacturés (voir fig 1).

Le but de ce TP est de se familiariser avec la modélisation de surfaces paramétriques. On s'interessera en particulier aux carreaux de Bézier ainsi qu'aux carreaux BSplines et à leurs raccordements.

2 Prise en main de l'environnement

Utilisez le 1er programme fourni.

Question 1 *Compilez et executez le programme en suivant les instructions du fichier* README.

Une documentation *Doxygen* du code est disponible dans le repertoire *doc/html/index.html*.

Note. Vous n'avez pas besoin de comprendre l'ensemble de ce code pour ce module. Vous pouvez cependant prendre connaissance d'un code d'affichage utilisant la librairie OpenGL pour la partie 3D, et Qt pour la partie fenêtre.

Vous disposez de classes de vecteurs de $\mathbb{R}^3$ et $\mathbb{R}^4$ (`vec3` et `vec4`). Ainsi que de matrices 3×3 et 4×4 (`matrix3`, et `matrix4`) qui possèdent leurs différents opérateurs redéfinis. Les classes de matrices 4×1 et 1×4 sont également définies et permettent de modéliser des vecteurs lignes/colonnes.

Le fichier `scene` est à compléter. Il est formé de deux parties. La première partie `load model` permet de charger la scène 3D avant d'afficher celle-ci. La seconde partie `draw_scene` contient les commandes d'affichages des objets visibles à l'écran.

3 Évaluateur

On s'interesse à des surfaces paramétriques, c'est à dire que l'on travail sur des fonctions dépendantes de 2 paramètres (u, v). De manière générale, on pourra écrire les coordonnées de la surface comme des

fonctions :

$$\begin{cases} x = f_1(u,v); \\ y = f_2(u,v); \\ z = f_3(u,v); \end{cases}$$

Le code vous propose une structure d'évaluation de fonctions (ici Bezier ou Spline) par le biais de la classe `evaluator_spline`. Cette classe redéfinie l'opérateur () en prenant deux arguments : les paramètres u et v de la surface, et retourne la valeur correspondante de l'interpolation.

Note : On appelle ce type de classe *foncteur* qui a pour rôle de remplacer l'utilisation de pointeurs de fonctions.

On pourra alors écrire ce type de code illustré en fig. 2. Notez que l'interpolation se réalise de manière $1D$. Dans le cas d'une surface possèdant 3 composantes, on devra donc appeler cette fonction 3 fois (une fois pour les composantes x, une fois pour y, et une fois pour z).

```
//la matrice 4x4 des donnees
matrix4 matrice_de_donnee(0,1,1,0,
                          1,2,2,1,
                          1,2,2,1,
                          0,1,1,0);

//la classe d'évaluation
evaluator_spline evaluateur(matrice_de_donnee);

//parametres dans [0,1]
double u=0.1;
double v=0.1;

//evaluation de la Bezier aux coordonnées paramétriques (u,v)
double f=evaluateur(u,v);
```

FIGURE 2 – Exemple d'utilisation de la classe d'évaluation.

La classe d'évaluation est actuellement complétée par une interpolation bilinéaire entre les 4 valeurs extrémales de la matrice de données (variable `P` de la classe d'évaluation).

Question 2 *Observez la classe d'évaluation* `evaluator_spline` *et visualisez son utilisation dans la méthode* `scene::load_model` *pour afficher une surface paramétrique.*

Question 3 *Quelle est l'expression de la surface (interpolation bilinéaire) en fonction des points de contrôle* P_{ij}*.*

Question 4 *Modifiez les coordonnées* z *de la variable* `patch_z` *dans la fonction* `scene::load_model()`*. Observez le résultats ? Quelles valeurs sont effectivement utilisées dans cette matrice. Faites de même pour les coordonnées* x *et* y *(*`patch_x` *et* `patch_y`*).*

4 Interpolation d'un carreau de Bézier

On rappelle que l'interpolation bi-dimensionnelle aux coordonnées (u, v) par des polynomes de degrés 3 peut se mettre sous la forme

$$f(u,v) = \mathbf{u}\,\mathrm{M}\,\mathrm{P}\,\mathrm{M}^T\,\mathbf{v}^T\,, \tag{1}$$

avec $\mathbf{u}$ le vecteur ligne $(u^3\ u^2\ u\ 1)$, $\mathbf{v}$ le vecteur ligne $(v^3\ v^2\ v\ 1)$, P la matrice 4×4 des données (=les coordonnées du polygone de contrôle), et M la matrice 4×4 de pondération fonction de la base choisie.

Question 5 *Modifiez la classe d'évaluation (opérateur parenthèse* `evaluator_spline::operator()`*) pour que celle-ci calcule la valeur d'un carreau (ou patch) de Bézier, et non l'interpolation linéaire.*

Notez que vous avez à disposition des classes de matrices 4×4, et des vecteurs lignes et colonnes (`matrix1x4` et `matrix4x1`). La matrice `P` des points de contrôle est un attribut de la classe d'évaluation, et il est possible de construire le vecteur/matrice transposée à l'aide de la fonction membre `transposed()`.

Question 6 *Quels propriétés géométriques sont vérifiées par un carreau de Bézier (par rapport à la position des points de contrôles). Vérifiez que ces propriétés sont bien vérifiées de manière graphique.*

Une fois l'opération réalisée, on peut noter que l'illumination du carreau de Bézier n'est pas correcte. En effet, pour permettre à OpenGL de réaliser l'illumination de Phong, il est nécessaire de fournir les normales de la surface aux sommets calculés. Or celles-ci sont toujours calculées comme étant celles de l'interpolation bilinéaire.

Question 7 *Soit f la fonction correspondant à la surface 3D. Donnez la relation entre la normale $n(u,v)$ d'un sommet aux coordonnées locales (u,v), et les dérivées partielles $\frac{\partial f}{\partial u}(u,v)$ et $\frac{\partial f}{\partial v}(u,v)$ des coordonnées 3D par rapport à u et v.*

Les deux méthodes `diff_u(u,v)` et `diff_v(u,v)` renvoient les dérivées partielles mentionnées. Notez leur utilisation dans le calcul de la normale dans le fichier `scene`.

Question 8 *Exprimez les dérivées partielles de la fonction f en fonction de u, M, P, et v. Notez que l'on pourra se servir de la relation matricielle.*

Question 9 *Mettez à jour les deux méthodes `diff_u(u,v)` et `diff_v(u,v)` avec le calcul correct des dérivées partielles, et observez que l'illumination de la surface est corrigée.*

Vérifiez que votre programme fourni un résultat visuel comparable à celui de la figure 3.

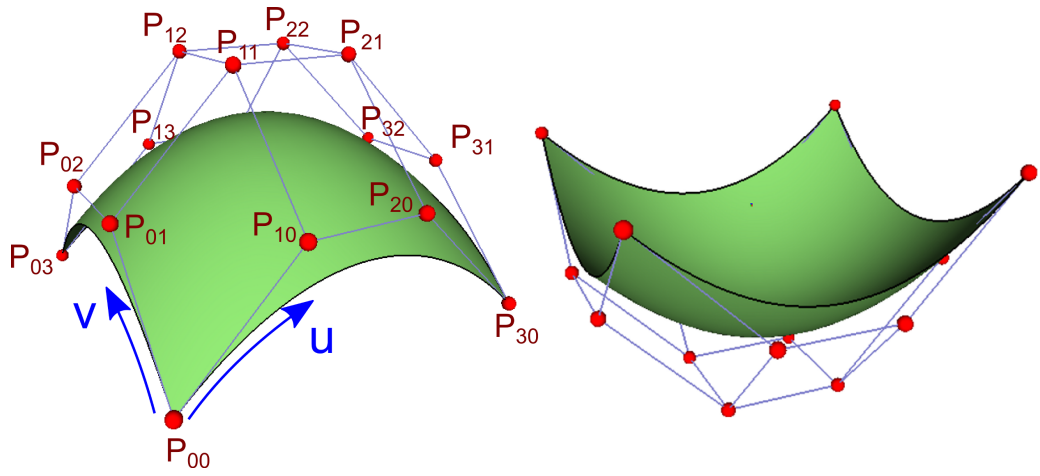

FIGURE 3 – Exemple de résultat de surface de Bézier.

Question 10 *Modifiez les coordonnées de vos points de contrôles et vérifiez le comportement de votre surface de Bézier.*

Soit la courbe $\mathcal{C}$ définie par la relation

$$\forall t \in [0,1]\,, \quad \begin{cases} x = f_1(0.5,t) \\ y = f_2(0.5,t) \\ z = f_3(0.5,t) \end{cases}$$

Question 11 *Calculez et stockez dans un vecteur (de type `std::vector<vec3>`) les coordonnées de la courbe $\mathcal{C}$. Affichez cette courbe en noire.*

Notez que l'appel `opengl_drawer::draw_curve()` permet d'afficher visuellement un vecteur de type `std::vector<vec3>`. Notez également que l'appel à `glColor3f(r,g,b)` est une commande OpenGL permettant d'affecter la couleur (r,g,b) à tous les affichages qui suivent cet appel.

Question 12 *Construisez un second carreau de coordonnées (3 nouvelles matrices 4×4). Construisez la surface associée et affichez la (vous définirez pour cela un second objet de type* surf*). Faites en sorte que la surface résultant des deux carreaux soit la plus continue possible. Quantifiez le* degré de continuité *que vous pouvez obtenir en fonction des contraintes que vous devez appliquer.*

5 Raccordement lisse et B-Splines

Prenez désormais le second programme qui vous est fourni. Celui-ci, plus complet, permet de gérer des grilles de tailles quelconques ainsi que de déplacer manuellement des sommets de la grille de controle.

Question 13 *Compilez et executez le second programme.*

Question 14 *Remplacer la classe* `evaluator_spline` *par celle que vous avez réalisé dans le programme précédent. Manipulez les sommets de contrôle à la main en les selectionnant à l'aide du* clic gauche+touche CTRL, *observez la déformation de la surface de Bézier.*

Pour obtenir des surfaces C^2 (deux fois dérivables, dont la dérivée seconde est continue) à partir d'une grille de contrôle quelconque on s'intéresse aux surfaces de type B-Spline (voir fig. 4).

Question 15 *Modifiez la matrice de pondération dans la fonction d'évaluation de la spline par celle permettant d'obtenir une fonction de type BSplines Naturelle Uniforme. Observez le résultat, que constatez vous.*

Question 16 *Ajoutez manuellement une ligne dans votre grille de contrôle (boutons de l'interface graphique). Observez le résultat obtenu, que pouvez vous dire ?*

Question 17 *Observez la manière dont sont construite les surfaces en calculant les sommets patchs à patchs (fonction* `build_spline_surface` *dans le fichier* `scene`*). Trouvez et comprenez la méthode de sélection des patchs les uns derrière les autres.*

Question 18 *Dupliquez les sommets du bord de votre polygone de contrôle (bouton de l'interface disponible). Observez le résultat obtenu. Dupliquez désormais 2x ces même sommets, qu'obtenez vous ?*

Question 19 *Quelles équations satisfont les courbes tracées en noires sur la surface ? Pourrait-on définir des courbes intermédiaires entre ces carreaux ? quels serait leurs équations ?*

La grille de contrôle possède la méthode `build_meshgrid()` qui construit une grille ordonnées suivant x et y de manière similaire à l'appel Matlab/Octave.

Question 20 *Testez et modifier la génération de la grille de contrôle de manière à générer une surface plane de* 10×10 *éléments dont vous viendrez en déformer certains manuellement.*

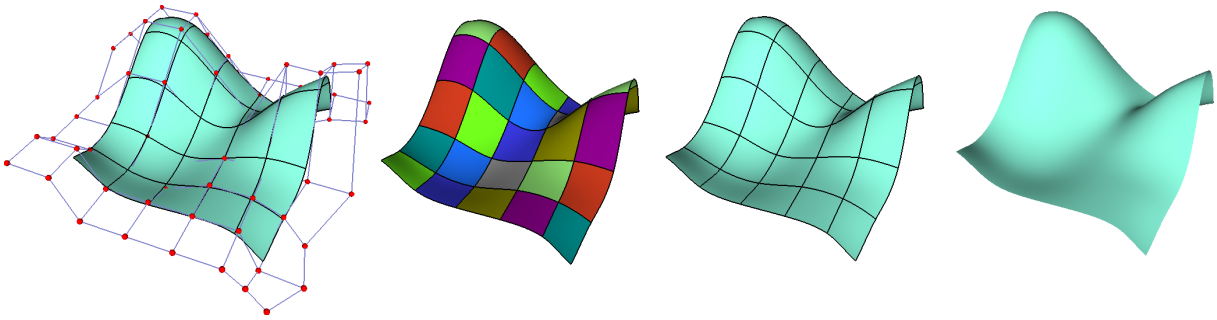

FIGURE 4 – Exemple de résultat de surface BSpline. De gauche à droite : Visualisation de la surface+grille de contrôle ; Visualisation des différents carreaux séparément ; Visualisation de la surface et des courbes de séparation des carreaux ; Visualisation de la surface uniquement.

Il est également possible de donner une structure tridimentionnelle à votre grille de contrôle pour modéliser des formes avancées.

Question 21 *Complétez la méthode* `grid::build_sphere()` *qui viens répartir uniformément vos points de contrôle sur une sphere (voir fig. 5). Testez le résultat.*

Question 22 *Modélisez une forme avancée lisse à l'aide de ce programme.* ex. *Véhicule, aile d'avion, etc. Notez que vous disposez d'une méthode de sauvegarde et de chargement de grille de contrôle.*

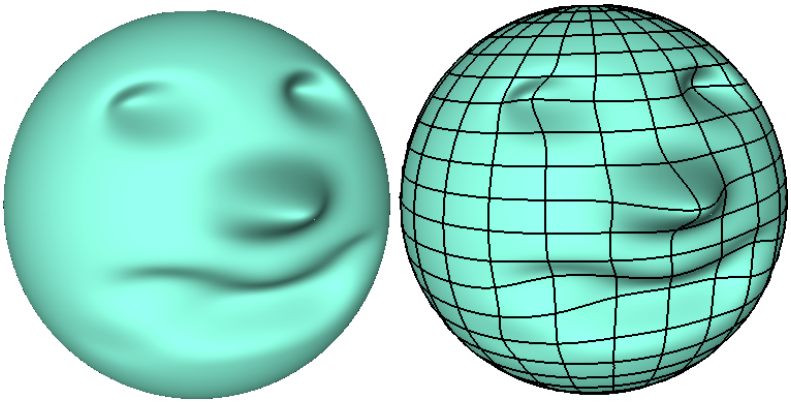

FIGURE 5 – Exemple de modélisation sur une grille de contrôle initialisée sur une sphère.

6 Pour aller plus loin

Question 23 *Optimisez l'efficacité du calcul de l'évaluateur. Pouvez vous factoriser certaines calculs ? Pouvez vous paralléliser des calculs* [1].

Question 24 *Proposez une modification sur le parcours de votre grille lors du calcul des carreaux de surface afin de prendre en compte des surface périodique suivant u, suivant v, ou suivant u et v en même temps. Appliquez cela à des formes de topologie cylindrique ou sphérique.*

Question 25 *Implémentez des surfaces animées (dont le polygone de contrôle varie au cours du temps).*

Question 26 *Généralisez votre surface aux cas de paramétrage non homogène (vecteur de noeuds), à l'élévation en degré, ou bien généralisez celle-ci aux cas des NURBS.*

1. Pensez aux threads, et/ou visitez le site d'OpenMP : http ://openmp.org/wp/