OpenGL - T.P. 2

Franck Hétroy
d'après Sylvain Lefebvre

07 décembre 2005

1 Visualisation d'un diagramme de Voronoï avec OpenGL

Terminez le T.P. commencé lors de la séance précédente.

Rappel des instructions :

1. Créez, grâce à *libQGLViewer*, une fenêtre permettant l'affichage d'un ensemble de points aléatoirement répartis dans le carré de centre l'origine, de rayon 1 et orthogonal à l'axe z. Choisissez la taille des points pour qu'ils soient visibles à l'écran !
2. Ajoutez la visualisation de cônes d'origines ces points, d'axe z et d'angle $45°$.
3. Utilisez le Z-buffer pour afficher le diagramme de Voronoï des points (pensez à placer de manière optimale la caméra).
4. **Pour aller plus loin :**
   - vous pouvez essayer de laisser à l'utilisateur le choix du nombre de points, voire leur position (mais il faut contraindre celles-ci à être dans le carré) ;
   - vous pouvez aussi ajouter la possibilité de sélectionner un point, et d'afficher entièrement le cône dont il est le sommet ;
   - vous pouvez créer une interface multi-vue, avec par exemple une vue de face du carré et une vue en perspective pour visualiser les cônes "au-delà" du diagramme de Voronoï ;
   - les personnes très à l'aise avec OpenGL peuvent essayer de créer une animation balayant le carré par une droite $x = k$ et affichant le diagramme de Voronoï à gauche de cette droite (animation illustrant à peu près l'algorithme de S. Fortune).

2 Effets de diffusion

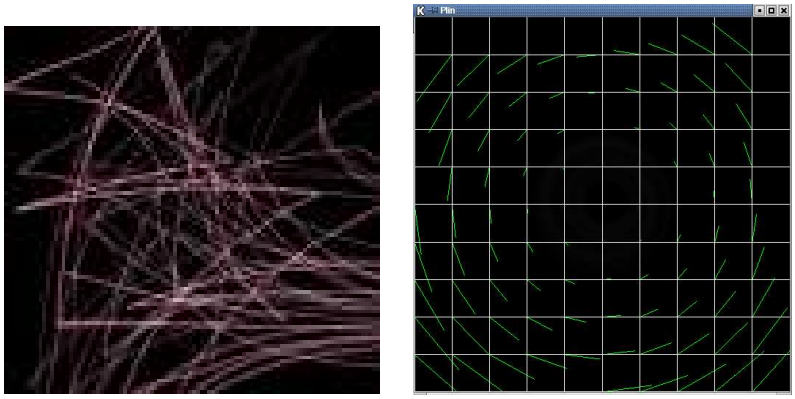
L'objectif de ce T.P. est de concevoir un algorithme de création d'*effets de diffusion*, dans le style des économiseurs d'écrans ou de certains plug-ins Winamp (voir la figure 1 (a) pour un exemple).

2.1 Aspect théorique

Ce genre d'effet fonctionne très simplement, en dessinant une texture déformée par un champ de vecteurs.

Exemple : supposons que la texture consiste en un O plus ou moins grand, et que la déformation décale la texture de la gauche vers la droite et la rétrécit. Une suite d'affichage donnerait un effet qui peut être schématisé de la sorte : $|O|||| \rightarrow ||o||| \rightarrow |||.|| \rightarrow ||||||$

L'affichage se fera sur une grille régulière. Chaque cellule de cette grille sera en fait un quadrilatère texturé (`GL_QUADS`), et le champ de vecteurs sera défini en chaque sommet de la grille (cf. figure 1 (b)). La déformation est obtenue en ajoutant le vecteur aux coordonnées de texture durant le plaquage de texture (elle sera interpolée automatiquement par OpenGL à l'intérieur des cellules, ce qui donne un effet de flou).



(a) (b)

Figure 1: (a) Un exemple d'effet de diffusion. (b) Grille et champ de vecteurs.

Attention : la grille et la texture ne sont calculées qu'*une fois*, au début. Le mouvement est généré simplement en *ré-appliquant le résultat d'une étape de déformation comme texture d'entrée de l'étape suivante.*

2.2 Aspect pratique

2.2.1 Instructions

Implémentez l'affichage d'un effet de diffusion à l'aide d'OpenGL.

- La résolution de la grille sera une constante : `#define GRID_RES 10`.
- Les coordonnées de texture seront lues dans un tableau de texture `double ***tbl` généré par une fonction `void genTexCoords(int resolution,double ***tbl)`.
- L'affichage de la grille texturée devra se faire en utilisant une *display list*.
- **Utilisation du clavier.**Vous devez laisser le choix à l'utilisateur :
  - d'afficher ou non la grille (avec la touche G) ;
  - d'afficher ou non le champ de vecteurs (avec la touche V) ;
  - de changer la déformation appliquée (avec la touche D) ;
  - d'effacer la texture affichée (avec la touche C).
- **Pour aller plus loin :**
  - Vous pouvez créer deux fonctions `draw()`, afin d'ajouter à la scène des objets n'interagissant pas avec l'effet de diffusion.
  - **Utilisation de la souris.** Vous pouvez ajouter des interactions entre la souris et l'effet affiché.

2.2.2 Indications

- Commencez en partant d'un exemple de *libQGLViewer* (par exemple *simpleViewer*).
- Fonctions à créer : `main(int argc, char **argv)`, `draw()`, `genTexCoords(int resolution, double ***tbl)`, `computeGrid(int resolution, double ***texCoords)`, `drawVectorField(int resolution, double*** texCoords)`, `mainRender()`, ...
- Rappel : la commande qui attache les coordonnées de texture à un sommet de la grille est `glTexCoord2d(...)` (à appeler *avant* `glVertex2d(...)`).
- **Création d'une déformation**. La déformation pourra être créée de manière (plus ou moins) aléatoire. Un exemple est donné en annexe. Vous pouvez vous en inspirer, mais n'hésitez pas à chercher d'autres déformations.
- Pour générer le mouvement, pensez à bien modifier la texture : *le résultat de l'affichage devient la nouvelle texture utilisée* (vous pouvez utiliser la commande `glCopyTexSubImage2D(...)`).

3 Si vous avez fini

... ou si ça ne vous plaît pas

Sujet libre :

- trouvez vous-même un problème qui vous intéresse et essayez de le résoudre avec OpenGL ;
- vous pouvez vous inspirer des exemples de libQGLViewer : `http://artis.imag.fr/Members/Gilles.Debunne/QGLViewer/examples/` ;
- vous pouvez vous inspirer de votre expérience personnelle ou de ce que vous avez étudié dans d'autres matières (notamment lors du stage DirectX) ;
- si vous manquez d'inspiration, voilà quelques pistes de sujets : `http://artis.imag.fr/Members/Gilles.Debunne/Enseignement/StageOpenGL/sujetsTPOpenGL.html`

Annexe : création d'une déformation (exemple)

```
void genTexCoords(int resolution,double ***tbl) {
  double x,y,c,s,d;
  int    i,j;

  for (i=0;i<resolution;i++) {
      for (j=0;j<resolution;j++) {
          tbl[i][j][0]=((double)i)/((double)resolution-1.0);
          tbl[i][j][1]=((double)j)/((double)resolution-1.0);
          switch (g_iDeform) {
            case 0:
              tbl[i][j][0]+=0.01;
              tbl[i][j][1]+=0.01;
              break;
            case 1:
              tbl[i][j][0]+=(i-resolution/2.0)*0.001;
              tbl[i][j][1]+=(j-resolution/2.0)*0.001;
              break;
            case 2:
              tbl[i][j][0]*=cos((i+j-resolution)/50.0*M_PI);
              tbl[i][j][1]*=cos((i-j)/100.0*M_PI);
              break;
            case 3:
              tbl[i][j][0]+=(i-resolution/2.0)*0.003;
              tbl[i][j][1]+=(j-resolution/2.0)*0.003;

              tbl[i][j][0]-=0.5;
              tbl[i][j][1]-=0.5;
              x=tbl[i][j][0];
              c=cos(M_PI/32.0+(i-resolution/2)*M_PI/32.0);
              s=sin(M_PI/32.0+(j-resolution/2)*M_PI/32.0);
              tbl[i][j][0]=c*x-s*tbl[i][j][1];
              tbl[i][j][1]=s*x+c*tbl[i][j][1];
              tbl[i][j][0]+=0.5;
              tbl[i][j][1]+=0.5;
              break;
            case 4:
              tbl[i][j][0]-=0.5;
              tbl[i][j][1]-=0.5;
              x=tbl[i][j][0];
             d=sqrt((i-resolution/2.0)*(i-resolution/2.0)+(j-resolution/2.0)*(j-resolution/2.0));
              c=cos((d)*M_PI/64.0);
              s=sin((d)*M_PI/64.0);
              tbl[i][j][0]=c*x-s*tbl[i][j][1];
              tbl[i][j][1]=s*x+c*tbl[i][j][1];
              tbl[i][j][0]+=0.5;
              tbl[i][j][1]+=0.5;
              break;
            default:
              break;
            }
        }
    }
}
```