Rapport TP Automne : LIFGRAPHIQUE

Réalisé par :
DE CLERCQ Allan
p2103328

Encadré par :
ZARA Florence

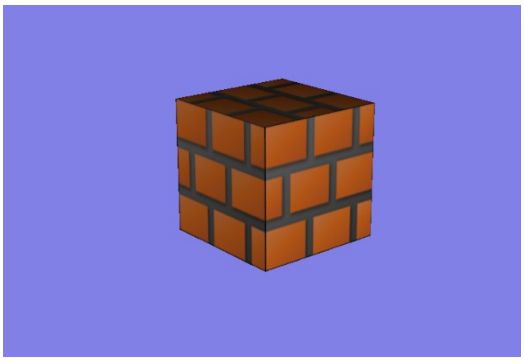
I. Manipulation des Formes de Bases

1. Définir un cube avec des GL TRIANGLE_STRIP

```
void ViewerEtudiant::init_cube()
{
   int i;
   static float pt[8][3]= { {-1,-1,-1}, {1,-1,-1}, {1,-1,1}, {-1,-1,1}, {-1,1,-1}, {1,1,-1}, {1,1,1}, {-1,1,1}
};
   static int f[6][4]= { {0,1,2,3}, {5,4,7,6}, {2,1,5,6}, {0,3,7,4}, {3,2,6,7}, {1,0,4,5} };
   static float n[6][3]= { {0,-1,0}, {0,1,0}, {1,0,0}, {-1,0,0}, {0,0,1}, {0,0,-1} };

   m_cube = Mesh(GL_TRIANGLE_STRIP);

   for (i=0;i<6;i++)
   {
      m_cube.normal(n[i][0], n[i][1], n[i][2]);

      m_cube.texcoord(0,0);
      m_cube.vertex( pt[ f[i][0] ][0], pt[ f[i][0] ][1], pt[ f[i][0] ][2] );

      m_cube.texcoord(1,0);
      m_cube.vertex( pt[ f[i][1] ][0], pt[ f[i][1] ][1], pt[ f[i][1] ]

      m_cube.texcoord(0,1);
      m_cube.vertex(pt[ f[i][3] ][0], pt[ f[i][3] ][1], pt[ f[i][3] ][

      m_cube.texcoord(1,1);
      m_cube.vertex( pt[ f[i][2] ][0], pt[ f[i][2] ][1], pt[ f[i][2] ]

      m_cube.restart_strip();
   }
}

int ViewerEtudiant::init()
{
      ...
      init_cube();
      tex_cube = read_texture(0, "data/mur.png" );
      ...

}

void ViewerEtudiant::draw_cube(const Transform& T, unsigned int tex)
{
   gl.model(T);
   gl.texture(tex);
   gl.draw(m_cube);

}

int ViewerEtudiant::render()
{
       ...
      draw_cube(Translation(0,0,0)*Scale(1,1,1),tex_cube);
      ...
}
```

Dans Viewer_Etudiant.h

```
class ViewerEtudiant : public Viewer
{
   ...
   protected:
        Mesh m_cube;

        GLuint tex_cube;

        void init_cube();

        void draw_cube(const
        Transform& T, unsigned int
        tex);
   ...
};
```



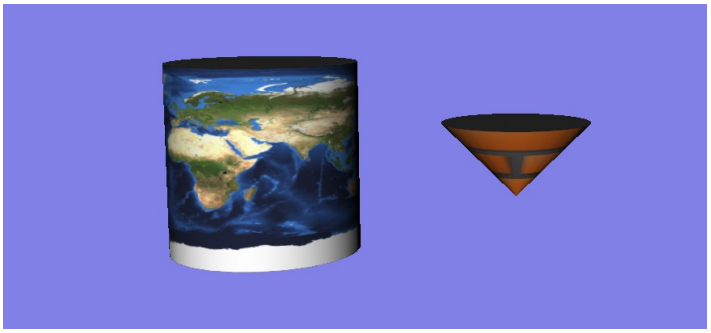
2. Définir un cylindre et un cône avec des GL_TRIANGLE_STRIP (+ Ajouter les normales et les coordonnées textures à ces formes de base)

Dans Viewer_Etudiant.h

```
class ViewerEtudiant : public Viewer
{
   ...
   protected:
        Mesh m_cone;
        Mesh m_cylindre;
        Mesh m_disque;

        GLuint tex_cone;
        GLuint tex_cylindre;
        GLuint tex_disque;

        void init_cone();
        void init_cylindre();
        void init_disque();


        void draw_cone(const
        Transform& T, unsigned int
        tex);
        void draw_cylindre(const
        Transform& T, unsigned int
        tex);

   ...
};
```

```
void ViewerEtudiant::init_cylindre()
{
   int i;
   const int div = 25;
   float alpha;
   float step= 2.0 * M_PI / (div);
   m_cylindre = Mesh(GL_TRIANGLE_STRIP);

   for(int i=0; i<=div; ++i)
   {
      alpha = i * step;
      m_cylindre.normal( Vector(cos(alpha),0, sin(alpha)) );
      m_cylindre.texcoord((float)i/div,1);
      m_cylindre.vertex( Point(cos(alpha),-1, sin(alpha)) );
      m_cylindre.normal( Vector(cos(alpha),0, sin(alpha)) );
      m_cylindre.texcoord((float)i/div,0);
      m_cylindre.vertex( Point(cos(alpha), 1, sin(alpha)) );
   }
}

void ViewerEtudiant::init_cone()
{
   const int div = 25;
   float alpha;
   float step= 2.0 * M_PI / (div);

   m_cone= Mesh(GL_TRIANGLE_STRIP);
   for(int i=0;i<=div;++i)
   {
      alpha = i * step;
      m_cone.normal(Vector( cos(alpha)/sqrtf(2.f),1.f/sqrtf(2.f), sin(alpha)/sqrtf(2.f) ));
      m_cone.texcoord((float)i/div,1);
      m_cone.vertex( Point( cos(alpha),0, sin(alpha) ));
      m_cone.normal(Vector( cos(alpha)/sqrtf(2.f),1.f/sqrtf(2.f), sin(alpha)/sqrtf(2.f) ));
      m_cone.texcoord((float)i/div,0);
      m_cone.vertex( Point(0, 1, 0) );

   }
}

void ViewerEtudiant::init_disque()
{
   const int div = 25;
   float alpha;
   float step= 2.0 * M_PI / (div);
   m_disque = Mesh( GL_TRIANGLE_FAN );
   m_disque.normal( Vector(0,-1,0) );
   m_disque.vertex( Point(0,0,0)  );
   for(int i=0; i<=div; ++i)
      {
         alpha = i * step;
         m_disque.normal( Vector(0,-1,0) );
         m_disque.vertex( Point(cos(alpha),0, sin(alpha)) );
      }
}
```

```
int ViewerEtudiant::init()
{
      ...
      init_cone();
      init_cylindre();
      init_disque();

      tex_cone = read_texture(0, "data/mur.png" );
      tex_cylindre = read_texture(0, "data/monde.jpg" );
      ...
}

void ViewerEtudiant::draw_cone(const Transform& T, unsigned int tex)
{
   gl.model(T);
   gl.texture(tex);
   gl.draw(m_cone);

   Transform Tch = T * Translation(0,0,0);
   gl.model(Tch);
   gl.draw(m_disque);
}

void ViewerEtudiant::draw_cylindre(const Transform& T, unsigned int tex)
{
   gl.model(T);
   gl.texture(tex);
   gl.draw( m_cylindre );

   // Disque du bas
   draw_disque(T * Translation( 0, -1, 0),tex);

   // Disque du haut
   draw_disque(T * Translation( 0, 1, 0)* Rotation( Vector(1,0,0), 180),tex);
}

int ViewerEtudiant::render()
{
      ...
      draw_cone(Translation(0,0,0)*Scale(1,1,1),tex_cone);
      draw_cylindre(Translation(0,0,0)*Scale(1,1,1) *Rotation(Vector(1,0,0),180),tex_cylindre);
      ...
}
```



3. Définir une sphère avec des GL_TRIANGLE_STRIP (+ Ajouter les normales et les coordonnées textures à ces formes de base)

```
void ViewerEtudiant::init_sphere()
{
   const int divBeta= 16;
```

```
    const int divAlpha= divBeta/2;
    int i,j;
    float beta, alpha, alpha2;
    m_sphere= Mesh(GL_TRIANGLE_STRIP);
    for(int i=0; i<divAlpha; ++i)
    {
        alpha= -0.5f * M_PI + float(i) * M_PI / divAlpha;
        alpha2= -0.5f * M_PI + float(i+1) * M_PI / divAlpha;
        for(int j=0; j<=divBeta; ++j)
            {
                beta= float(j) * 2.f * M_PI / (divBeta);
                m_sphere.normal( Vector(cos(alpha)*cos(beta),sin(alpha), cos(alpha)*sin(beta)) );
                m_sphere.texcoord(beta/(2.f * M_PI),0.5-alpha/M_PI);
                m_sphere.vertex( Point(cos(alpha)*cos(beta),sin(alpha), cos(alpha)*sin(beta)) );
                m_sphere.normal( Vector(cos(alpha2)*cos(beta), sin(alpha2), cos(alpha2)*sin(beta)) );
                m_sphere.texcoord(beta/(2.f * M_PI),0.5-alpha2/M_PI);
                m_sphere.vertex( Point(cos(alpha2)*cos(beta),sin(alpha2), cos(alpha2)*sin(beta)) );
            }
                m_sphere.restart_strip();
    }
```

Dans Viewer_Etudiant.h

```
class ViewerEtudiant : public Viewer
{
    …
    protected:
        Mesh m_sphere;

        GLuint tex_ sphere;

        void init_ sphere ();

        void draw_ sphere (const
        Transform& T, unsigned int
        tex);
    …
};
```

```
int ViewerEtudiant::init()
{
        …
        init_sphere();
        tex_sphere = read_texture(0, "data/monde.jpg" );
        …

}

void ViewerEtudiant::draw_sphere(const Transform& T, unsig
{
    gl.model(T);
    gl.texture(tex);
    gl.draw(m_sphere);


}

int ViewerEtudiant::render()
{
         …
        draw_shpere(Translation(0,0,0)*Scale(1,1,1) *Rotation(Vector(1,0,0),180),tex_sphere);
        …
}
```

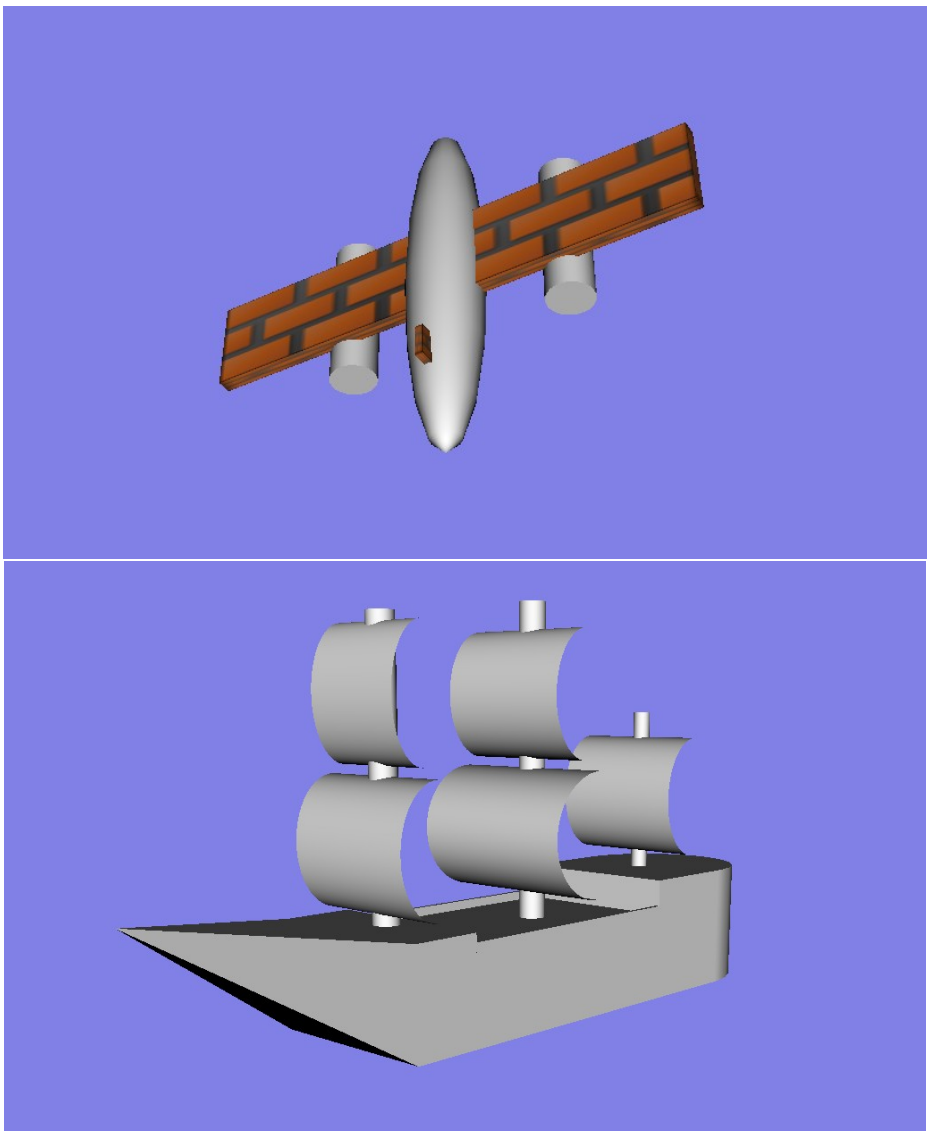
II. Affichage à l'aide de Transformations Géométriques

```
void ViewerEtudiant::draw_avion(const Transform& T)
{
    draw_sphere(T* Translation(0,0,0)*Scale(0.5,2.5,0.5),0);

    draw_cube(T* Translation(1.8,0.3,0)*Scale(1.5,0.6,0.1),tex_
```

Dans Viewer_Etudiant.h

```
class ViewerEtudiant : public Viewer
{
    …
    protected:
        void draw_ avion (const
        Transform& T);
    …
};
```

```
    draw_cube(T* Translation(-1.8,0.3,0)*Scale(1.5,0.6,0.1),tex_cube);

    draw_cylindre(T* Translation(1.5,0.3,-0.3)*Scale(0.3,1,0.3),0);
    draw_cylindre(T* Translation(-1.5,0.3,-0.3)*Scale(0.3,1,0.3),0);

    draw_cube(T* Translation(0,-1.3,0.3)*Scale(0.05,0.2,0.3),tex_cube);

}

int ViewerEtudiant::render()
{
        ...
       draw_avion(Translation(0,0,0)*Scale(1,1,1));
       ...
}
```



III. Terrain, texture, billboard (arbre) et cubemap

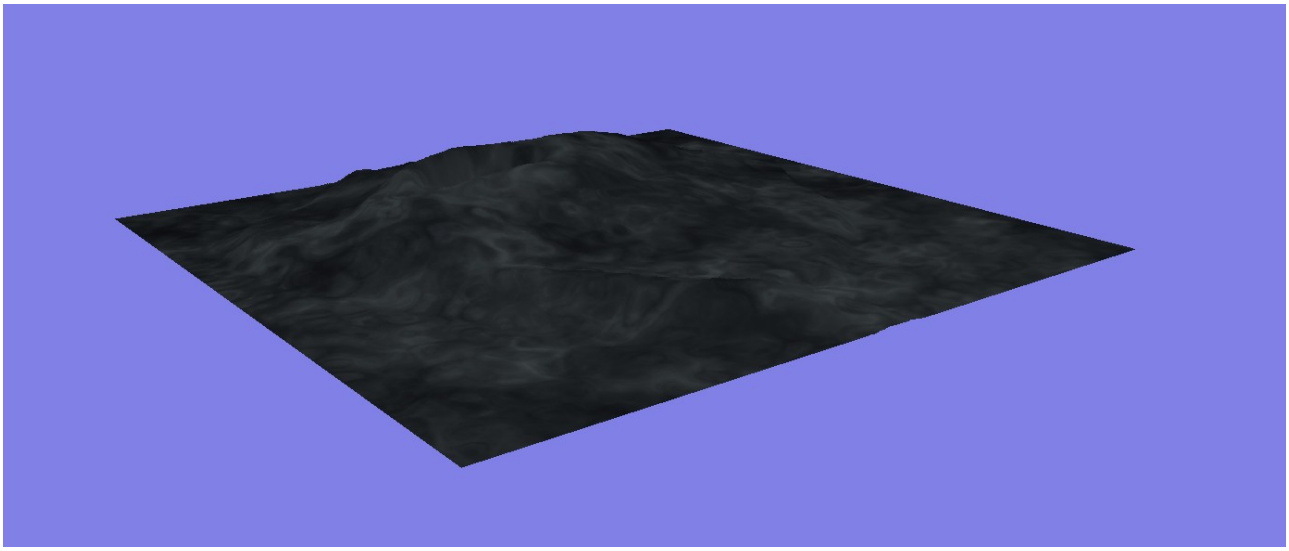
1. Terrain

```
Vector ViewerEtudiant::terrainNormal(const Image& im, const int i, const int j){
       // Calcul de la normale au point (i,j) de l'image
       int ip = i-1;
       int in = i+1;
```

```
        int jp = j-1;
        int jn = j+1;
        Vector a( ip, im(ip, j).r, j );
        Vector b( in, im(in, j).r, j );
        Vector c( i, im(i, jp).r, jp );
        Vector d( i, im(i, jn).r, jn );
        Vector ab = normalize(b - a);
        Vector cd = normalize(d - c);
        Vector n = cross(cd,ab);
        return n;
    }

void ViewerEtudiant::init_terrain(Mesh& m_terrain, const Image& im)
{
    m_terrain = Mesh(GL_TRIANGLE_STRIP);

    for(int i=1;i<im.width()-2;++i){ // Boucle sur les i
        for(int j=1;j<im.height()-1;++j){ // Boucle sur les j
            m_terrain.normal( terrainNormal(im, i+1, j) );
            m_terrain.texcoord(float(i+1)/im.width(),float(j)/ir
            m_terrain.vertex( Point(i+1, 100.f*im(i+1, j).r, j) )

            m_terrain.normal( terrainNormal(im, i, j) );
            m_terrain.texcoord(float(i+1)/im.width(),float(j)/ir
            m_terrain.vertex( Point(i, 100.f*im(i, j).r, j) );
        }
        m_terrain.restart_strip();
    }
}

int ViewerEtudiant::init()
{
    ...
    m_terrainAlti = read_image("data/terrain/island.jpg");
    init_terrain(m_terrain, m_terrainAlti);

    tex_terrain = read_texture(0,"data/terrain/Lava.jpg");
    ...
}

void ViewerEtudiant::draw_terrain(const Transform& T, unsigned int tex){
    gl.model( T );
    gl.texture(tex);
    gl.draw( m_terrain );
}

int ViewerEtudiant::render()
{
     ...
    draw_terrain(Translation(-50,-0.2,-50)*Scale(0.1,0.1,0.1),tex_terrain);
    ...
}
```

Dans Viewer_Etudiant.h

```
class ViewerEtudiant : public Viewer
{
    ...
    protected:
        Mesh m_terrain;
        Image m_terrainAlti;
        Vector terrainNormal(const
        Image& im,const int i, const int
        j);

        GLuint tex_terrain;

        void init_terrain(Mesh&
        m_terrain, const Image& im);

        void draw_terrain(const
        Transform& T,unsigned int tex);
    ...
};
```



2. Billboard

```
void ViewerEtudiant::init_billboard(){

   m_quad = Mesh(GL_TRIANGLE_STRIP);

   m_quad.normal(0, 0, 1);

   m_quad.texcoord(0,0);
   m_quad.vertex(-1, -1, 0);

   m_quad.texcoord(0,1);
   m_quad.vertex(1, -1, 0);

   m_quad.texcoord(1,0);
   m_quad.vertex(-1, 1, 0);

   m_quad.texcoord(1,1);
   m_quad.vertex(1, 1, 0);
}

void ViewerEtudiant::init_height_tree(const Image& im)
{

   int k = im.width()-2;
   int j = im.height()-1;
   for(int i = 0 ; i < 100 ; i++  ){

      pos[i].x =(rand() % k + 1)*0.1;
      pos[i].z = (rand() % j + 1)*0.1 ;
      pos[i].y = (100.f*im(pos[i].x, pos[i].z).r )*0.1 ;
   }
}

int ViewerEtudiant::init()
{
   …
   init_billboard();
   init_height_tree(m_terrainAlti);
   tex_billboard = read_texture(0,"data/billboard/arbre.png" );
   …
```

Dans Viewer_Etudiant.h

```
#define MAXPTS 100
const int vase_NBPT = 10;
const int vase_NBROT = 20;

class ViewerEtudiant : public Viewer
{
   …
   protected:
        Mesh m_quad;
        Point pos[MAXPTS];
        GLuint tex_billboard;
        void init_billboard();
        … (void terrain)
        void init_height_tree(const
        Image& im);

        void draw_billboard(const
        Transform& T);
        void draw_quadcross(const
        Transform& T);
   …
};
```

```
}

void ViewerEtudiant::draw_billboard(const Transform& T){

    draw_quadcross(T* Translation(Vector(0, 0, 0)) * RotationY(90));
    draw_quadcross(T* Translation(Vector(0,0,0)));
}

void ViewerEtudiant::draw_quadcross(const Transform& T){

    gl.texture(tex_billboard);
    gl.alpha(0.3);
    gl.model(T * RotationZ(90));
    gl.draw(m_quad);
    gl.alpha();
    gl.alpha(0.3);
    gl.model(T * RotationZ(90) * RotationX(180));
    gl.draw(m_quad);
    gl.alpha();

}

for(int k = 0 ; k < 30 ; k++  ){
        if (pos[k].y > 0.4) {
            draw_billboard(Translation(pos[k].x-50 , pos[k].y , pos[k].z-50)*Scale(2,2,2));
        }
    }
```

3. CubeMap

```
void ViewerEtudiant::init_cubemap()
{
    int i;
    float w = 1.0/4.0;
    float h = 1.0/3.0;

    static float pt[8][3]= { {1,1,1}, {-1,1,1}, {-1,1,-1}, {1,1,-1}, {1,-1,1}, {-1,-1,1}, {-1,-1,-1},
{1,-1,-1} };
    static int f[6][4]= { {0,1,2,3}, {5,4,7,6}, {2,1,5,6}, {0,3,7,4}, {3,2,6,7}, {1,0,4,5} };
    static float n[6][3]= { {0,-1,0}, {0,1,0}, {1,0,0}, {-1,0,0}, {0,0,1}, {0,0,-1} };

    m_cubemap = Mesh(GL_TRIANGLE_STRIP);
```

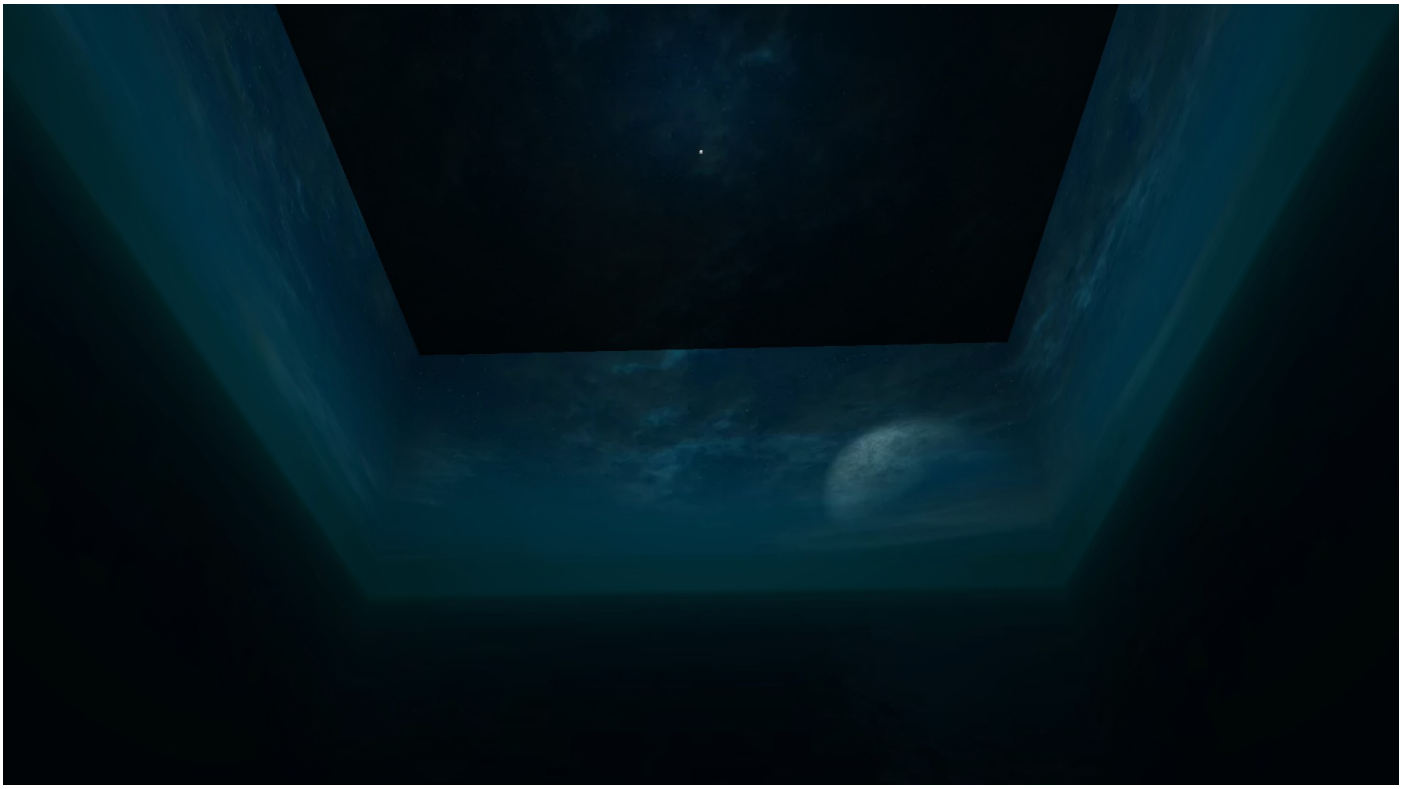
```
for (i=0;i<6;i++)
{
   switch(i) {
   case 0 :
      m_cubemap.normal(n[i][0], n[i][1], n[i][2]);
      m_cubemap.texcoord(2.0*w ,  1.0); // 0,0
      m_cubemap.vertex( pt[ f[i][0] ][0], pt[ f[i][0] ][1], pt[ f[i][0] ][2] );

      m_cubemap.texcoord(w , 1.0);  // 1,0
      m_cubemap.vertex( pt[ f[i][1] ][0], pt[ f[i][1] ][1], pt[ f[i][1] ][2] );

      m_cubemap.texcoord(2.0*w , 2.0*h);  // 0,1
      m_cubemap.vertex(pt[ f[i][3] ][0], pt[ f[i][3] ][1], pt[ f[i][3] ][2] );

      m_cubemap.texcoord(w , 2.0*h);  // 1,1
      m_cubemap.vertex( pt[ f[i][2] ][0], pt[ f[i][2] ][1], pt[ f[i][2] ][2] );

      m_cubemap.restart_strip();
   break;

   case 1 :
      m_cubemap.normal(n[i][0], n[i][1], n[i][2]);

      m_cubemap.texcoord(2.0*w , h); // 0,0
      m_cubemap.vertex( pt[ f[i][0] ][0], pt[ f[i][0] ][1], pt[ f[i][0] ][2] );

      m_cubemap.texcoord(w , h);  // 1,0
      m_cubemap.vertex( pt[ f[i][1] ][0], pt[ f[i][1] ][1], pt[ f[i][1] ][2] );

      m_cubemap.texcoord(2.0*w , 0);  // 0,1
      m_cubemap.vertex(pt[ f[i][3] ][0], pt[ f[i][3] ][1], pt[ f[i][3] ][2] );

      m_cubemap.texcoord(w ,  0);  // 1,1
      m_cubemap.vertex( pt[ f[i][2] ][0], pt[ f[i][2] ][1], pt[ f[i][2] ][2] );

      m_cubemap.restart_strip();
   break;

   case 2 :
      m_cubemap.normal(n[i][0], n[i][1], n[i][2]);

      m_cubemap.texcoord(w , 2.0*h); // 0,0
      m_cubemap.vertex( pt[ f[i][0] ][0], pt[ f[i][0] ][1], pt[ f[i][0] ][2] );

      m_cubemap.texcoord(0 , 2.0*h);  // 1,0
      m_cubemap.vertex( pt[ f[i][1] ][0], pt[ f[i][1] ][1], pt[ f[i][1] ][2] );

      m_cubemap.texcoord(w , h);  // 0,1
      m_cubemap.vertex(pt[ f[i][3] ][0], pt[ f[i][3] ][1], pt[ f[i][3] ][2] );

      m_cubemap.texcoord(0 ,  h);  // 1,1
      m_cubemap.vertex( pt[ f[i][2] ][0], pt[ f[i][2] ][1], pt[ f[i][2] ][2] );
```

```
    m_cubemap.restart_strip();
break;

case 3 :
    m_cubemap.normal(n[i][0], n[i][1], n[i][2]);

    m_cubemap.texcoord(3.0*w ,  2.0*h); // 0,0
    m_cubemap.vertex( pt[ f[i][0] ][0], pt[ f[i][0] ][1], p

    m_cubemap.texcoord(2.0*w , 2.0*h);  // 1,0
    m_cubemap.vertex( pt[ f[i][1] ][0], pt[ f[i][1] ][1], p

    m_cubemap.texcoord(3.0*w , h);  // 0,1
    m_cubemap.vertex(pt[ f[i][3] ][0], pt[ f[i][3] ][1], pt

    m_cubemap.texcoord(2.0*w , h);  // 1,1
    m_cubemap.vertex( pt[ f[i][2] ][0], pt[ f[i][2] ][1], p

    m_cubemap.restart_strip();
break;

case 4 :
    m_cubemap.normal(n[i][0], n[i][1], n[i][2]);

    m_cubemap.texcoord(2.0*w ,  2.0*h); // 0,0
    m_cubemap.vertex( pt[ f[i][0] ][0], pt[ f[i][0] ][1], pt[ f[i][0] ][2] );

    m_cubemap.texcoord(w , 2.0*h);  // 1,0
    m_cubemap.vertex( pt[ f[i][1] ][0], pt[ f[i][1] ][1], pt[ f[i][1] ][2] );

    m_cubemap.texcoord(2.0*w , h);  // 0,1
    m_cubemap.vertex(pt[ f[i][3] ][0], pt[ f[i][3] ][1], pt[ f[i][3] ][2] );

    m_cubemap.texcoord(w , h);  // 1,1
    m_cubemap.vertex( pt[ f[i][2] ][0], pt[ f[i][2] ][1], pt[ f[i][2] ][2] );

    m_cubemap.restart_strip();
break;

case 5:
    m_cubemap.normal(n[i][0], n[i][1], n[i][2]);

    m_cubemap.texcoord(1.0 ,  2.0*h); // 0,0
    m_cubemap.vertex( pt[ f[i][0] ][0], pt[ f[i][0] ][1], pt[ f[i][0] ][2] );

    m_cubemap.texcoord(3.0*w , 2.0*h);  // 1,0
    m_cubemap.vertex( pt[ f[i][1] ][0], pt[ f[i][1] ][1], pt[ f[i][1] ][2] );

    m_cubemap.texcoord(1.0 , h);  // 0,1
    m_cubemap.vertex(pt[ f[i][3] ][0], pt[ f[i][3] ][1], pt[ f[i][3] ][2] );

    m_cubemap.texcoord(3.0*w , h);  // 1,1
    m_cubemap.vertex( pt[ f[i][2] ][0], pt[ f[i][2] ][1], pt[ f[i][2] ][2] );
```

Dans Viewer_Etudiant.h

```
class ViewerEtudiant : public Viewer
{
   …
   protected:
        Mesh m_cubemap;
        GLuint tex_cubemap;
        void init_cubemap();
        void draw_cubemap(const
        Transform& T,unsigned int tex);
        …
};
```

```
            m_cubemap.restart_strip();
        break;
        }
    }
}

int ViewerEtudiant::init()
{
    ...
    init_cubemap();
    tex_cubemap =  read_texture(0,"data/cubemap/skybox.jpg");
    ...

}

void ViewerEtudiant::draw_cubemap(const Transform& T, unsigned int tex)
{
    gl.model(T);
    gl.texture(tex);
    gl.draw(m_cubemap);

}

int ViewerEtudiant::render()
{
    ...

    draw_cubemap(Translation(0,0,0)*Scale(100,100,100),tex_cubemap);
    ...

}
```



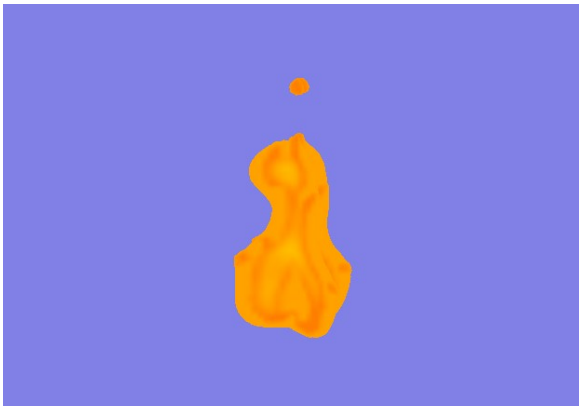
IV. Extrusion
V. Texture animée

```
void ViewerEtudiant::init_textureanimee() {

    // Choix de la primitive OpenGL
    m_ textureanimee = Mesh(GL_TRIANGLE_STRIP);

    // Vecteur normal a la face
    m_ textureanimee.normal(0, 0, 1);

    m_textureanimee.texcoord(0 ,0);
    m_ textureanimee.vertex(-1, -1, 0);

    m_ textureanimee.texcoord(1.0/9.0, 0);
    m_ textureanimee.vertex(1, -1, 0);

    m_textureanimee.texcoord(0, 1);
    m_textureanimee.vertex(-1, 1, 0);

    m_textureanimee.texcoord(1.0 / 9.0, 1);
    m_textureanimee.vertex(1, 1, 0);

 }

int ViewerEtudiant::init()
{
     ...
      init_textureanimee();
      tex_feu =  read_texture(0,"data/fire.png");
     ...
}
void ViewerEtudiant::draw_feu(const Transform& T){
```

Dans Viewer_Etudiant.h

```
class ViewerEtudiant : public Viewer
{
    ...
    protected:
        Mesh m_textureanimee;
        int compteurTps;
        GLuint tex_feu;
        void init_textanime();
        void draw_feu(const
Transform& T);
        void draw_quadfeu(const
        Transform& T);
        ...
};
```

```
    draw_quadfeu(T* Translation(Vector(0, 0, 0)) * RotationY(90));
    draw_quadfeu(T* Translation(Vector(0,0,0)));
}

void ViewerEtudiant::draw_quadfeu(const Transform& T){

    gl.alpha(0.3);
    gl.texture(tex_feu);
    gl.model(T);
    gl.draw(m_textureanimee);
    gl.model(T * RotationY(180));
    gl.draw(m_textureanimee);
    gl.alpha();

}

int ViewerEtudiant::render()
{
     ...
     draw_feu(Translation(-24,11,-5)*Scale(15,15,15));
     ...
}

int ViewerEtudiant::update( const float time, const float delta )
{
    // time est le temps ecoule depuis le demarrage de l'application, en millisecondes,
    // delta est le temps ecoule depuis l'affichage de la derniere image / le dernier appel a draw(), en
millisecondes.

    float temps1 = time / 150;
    int temps2 = int(temps1);


    compteurTps = temps2;

    int nt = compteurTps % 9;

    m_textureanimee.texcoord(0, nt * (1.0 / 9.0) , 0 );
    m_textureanimee.texcoord(1, nt * (1.0 / 9.0)+ (1.0/9.0),0);
    m_textureanimee.texcoord(2, nt * (1.0 / 9.0) , 1);
    m_textureanimee.texcoord(3, nt * (1.0 / 9.0) + (1.0 / 9.0), 1);

    return 0;
}
```



VI. Animation

```
int ViewerEtudiant::render()
{
   …
   draw_bateau(Transform_Bateau*Scale(1,1,1)*Rotation(Vector(0,1,0),180));
   …

}
int ViewerEtudiant::update( const float time, const float delta
{
   …
   float Temps2 = time/1000.0;
   int Temps1 = int(Temps2);

   int iTemps = Temps1 % m_animBateau.nb_points();
   int iTemps_suiv = (Temps1+1) % m_animBateau.nb_points(
   int iTemps_suiv_suiv = (iTemps_suiv + 1) % m_animBateau

   Point p0 = m_animBateau[iTemps];
   Point p1 = m_animBateau[iTemps_suiv];
   Point p2 = m_animBateau[iTemps_suiv_suiv];

   float poids = Temps2 - Temps1;

   Vector pos_suiv = Vector(p1) + (Vector(p2) - Vector(p1)) * poids;
   Vector pos = Vector(p0) + (Vector(p1) - Vector(p0)) * poids;

   Vector dir = normalize(pos_suiv - pos);
   Vector up(0,1,0);
   Vector codir = cross(dir, up);

   Transform_Bateau = Transform(dir, up, codir, pos);

   Transform_Bateau = Transform(dir.x, up.x, codir.x, pos.x,
                    dir.y, up.y, codir.y, pos.y,
                    dir.z, up.z, codir.z, pos.z,
                    0    , 0   , 0      , 1);
    …
}
```

Dans Viewer_Etudiant.h

```
class ViewerEtudiant : public Viewer
{
   …
   protected:

        void draw_bateau(const
        Transform& T);
        Transform Transform_Bateau;
        AnimationCurve
        m_animBateau;
        …
};
```