

## Sujets de TP - automne 2020

PATINO Nicolas  
11926177

# Contenu

1. Manipulation des formes de base.....	5
1) Définir un cube avec des GL_TRIANGLE_STRIP.....	5
a. avec 6 quadrilatères.....	5
.....	5
b. avec une structure indexée.....	6
2) Définir un cylindre et un cône avec des GL_TRIANGLE_STRIP.....	7
3) Définir une sphère avec des GL_TRIANGLE_STRIP.....	9
4) Ajouter les normales et les coordonnées textures à ces formes de base	
.....	10
2. Affichage à l'aide de transformations géométriques.....	14
3. Terrain, texture, billboard (arbre) et cubemap.....	15
1) A partir d'une image interprétée comme une carte de hauteur, définir les sommets du terrain correspondant.....	15
2) Définir les triangles formant le terrain.....	15
3) Ajouter le calcul de normal pour chaque sommet du maillage.....	15
4) Ajouter les coordonnées textures.....	16
5) Afficher un arbre représenté par un billboard.....	17
6) Afficher un ensemble d'arbres sur le terrain en utilisant la carte de hauteur pour les positionner.....	18
7) Afficher un cube texturé autour de votre scène.....	19
4. Modélisation d'un objet par extrusion.....	20
5. Texture animée.....	22
SCÈNE FINAL.....	24

```

1  #ifndef VIEWER_ETUDIANT_H
2  #define VIEWER_ETUDIANT_H
3
4  #include "Viewer.h"
5
6
7
8
9  class ViewerEtudiant : public Viewer
10 {
11 public:
12     ViewerEtudiant();
13
14     int init();
15     int render();
16     int update( const float time, const float delta );
17
18 protected:
19
20     /// Declaration des Mesh
21     //Mesh des formes de bases
22     Mesh m_cubo_q;
23     Mesh m_cubo_s;
24     Mesh m_cylinder;
25     Mesh m_cone;
26     Mesh m_sphere;
27     Mesh m_disque;
28     //Mesh du terrain
29     Mesh m_terrain;
30     //Mesh de l'extraction
31     Mesh m_extru;
32     //Mesh pour les textures animee
33     Mesh m_iman;
34     Mesh m_iman2;
35     //Mesh pour le cube-map
36     Mesh m_cubo_map;
37     //Mesh pour les objets importer
38     Mesh obj1;
39     Mesh obj2;
40     Mesh obj3;
41
42     /// Declaration des Textures
43     //Texture des formes de base
44     GLuint m_cubo_texture;
45     GLuint m_globe_texture;
46     //Textures pour le terrain
47     Image m_terrainAlti;
48     GLuint m_terrain_texture;
49     //Textures pour le billboard
50     GLuint m_arbre_texture;
51     GLuint m_palmera_texture;
52     //Textures pour le cube-map
53     GLuint m_map_texture;
54     //Textures pour l'extraction
55     GLuint m_tent_texture;
56     //Texture pour animation (requin)
57     GLuint m_iman_texture1;
58     GLuint m_iman_texture2;
59     GLuint m_iman_texture3;
60     GLuint m_iman_texture4;
61     GLuint m_iman_texture5;
62     GLuint m_iman_texture6;
63     GLuint m_iman_texture7;
64     GLuint m_iman_texture8;
65     GLuint m_iman_texture9;
66     GLuint m_iman_texture10;
67     //Textures pour animation (cascade)
68     GLuint m_iman2_texture1;
69     GLuint m_iman2_texture2;
70     GLuint m_iman2_texture3;
71     GLuint m_iman2_texture4;
72     GLuint m_iman2_texture5;
73     GLuint m_iman2_texture6;
74     GLuint m_iman2_texture7;
75     GLuint m_iman2_texture8;
76     GLuint m_iman2_texture9;
77     //Textures pour les objets
78     GLuint m_ele_texture;
79     GLuint m_turtle_texture;
80     GLuint m_ufo_texture;
81

```

## ViewerEtudiant.h

```

82   /// Declaration des fonction de creation de Mesh du type init_votreObjet()
83   //init formes de bases
84   void init_cubo_q();
85   void init_cubo_s();
86   void init_cylinder();
87   void init_cone();
88   void init_sphere();
89   void init_disque();
90   //init terrain
91   void init_terrain(Mesh& m_terrain, const Image& im);
92   //init billboard
93   void init_billboard();
94   //init cube-map
95   void init_map();
96   //init objet par extraction
97   void init_obj_extru();
98   void creation_forme(); //creation de la silhouette 2D
99   //init des textures animees, requin et cascade
100  void init_iman();
101  void init_iman2();
102
103  ///Declaration des variables et tableaux
104  int compteur=0; //Compteur pour les textures animees
105  vec2 tab[200]; //Tableau pour les coordones des arbres
106  vec2 tab2[200]; //Tableau pour les coordones des palmiers
107  int NBPT; //Declaration des nombres de points pour la sihouette 2D
108  Point objt_v[100][100]; //Tableau pour les vertex de l'objet cree par extraction
109  Vector objt_vn[100][100]; //Tableau pour les normales de l'objet cree par extraction
110  int Narbre=200; //Nombre d'arbres a positionner
111  int Npalmera=100; //Nombre de palmiers a positionne
112
113  /// Declaration des fonctions draw_votreObjet(const Transform& T)
114  //Draw formes de base
115  void draw_cubo_q(const Transform& T);
116  void draw_cubo_s(const Transform& T);
117  void draw_cylinder(const Transform& T);
118  void draw_cone(const Transform& T);
119  void draw_sphere(const Transform& T);
120  //Draw bombe
121  void dessineObjet(const Transform& T);
122  //Draw terrain
123  void draw_terrain(const Transform& T);
124  //Draw billboards, arbre et palmier
125  void draw_billboard(const Transform& T);
126  void draw_billboard2(const Transform& T);
127  //Draw cube-map
128  void draw_map(const Transform& T);
129  //Draw objet par extraction
130  void draw_obj_extru(const Transform& T);
131  //Draw texture animee, requin et cascade
132  void draw_iman(const Transform& T);
133  void draw_iman2(const Transform& T);
134  //Draw objet importer, UFO et elephant sur tortue
135  void draw_obj1(const Transform& T);
136  void draw_obj2(const Transform& T);
137 };
138
139
140
141 #endif
142

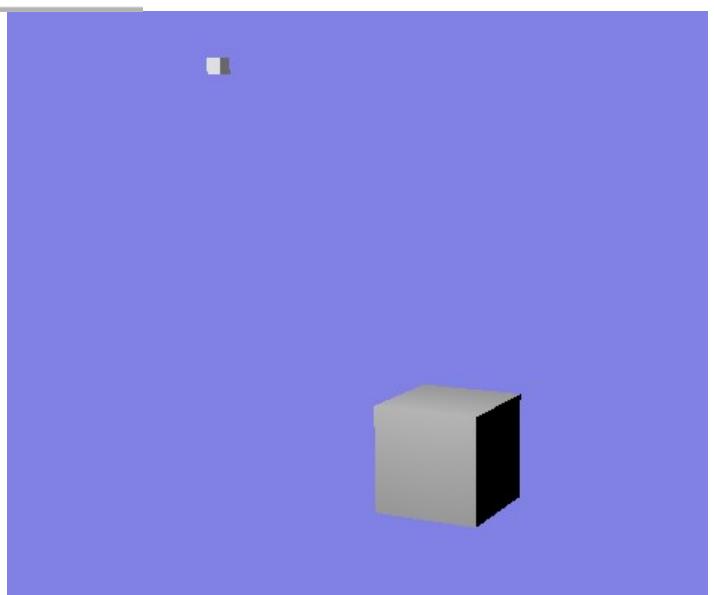
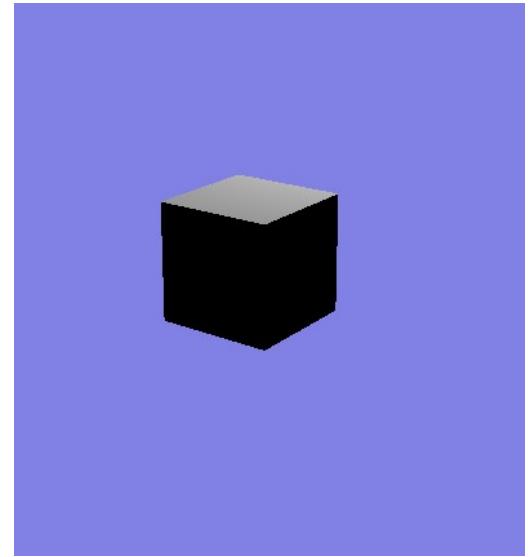
```

# 1. Manipulation des formes de base

## 1) Définir un cube avec des GL\_TRIANGLE\_STRIP

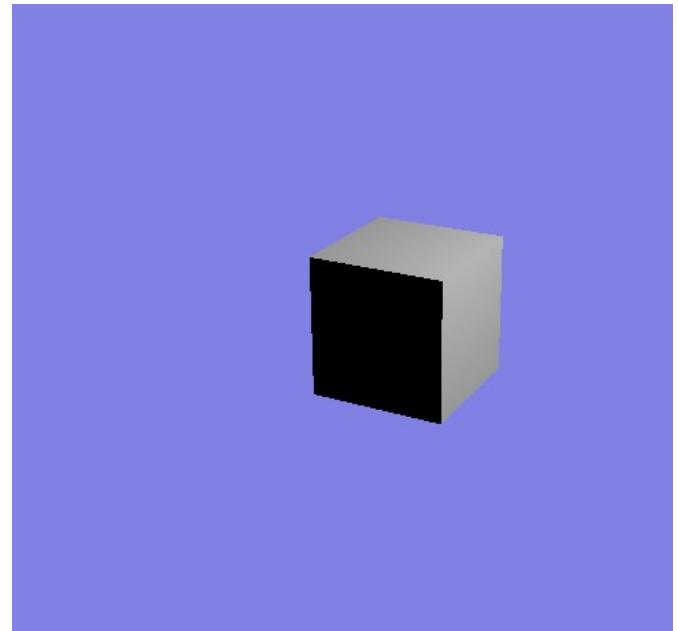
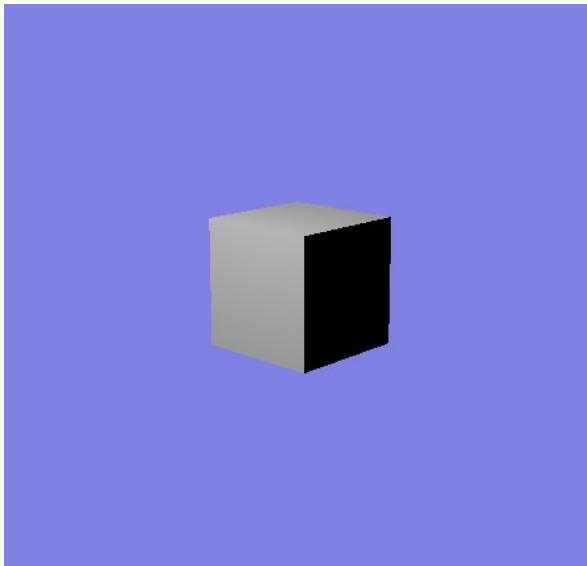
### a. avec 6 quadrillatères

```
15 void ViewerEtudiant::init_cubo_q()
16 {
17     //On choisit les primitives
18     m_cubo_q=Mesh(GL_TRIANGLE_STRIP);
19     //On cree chaqu'une des differentes faces
20     //1 face
21     //normale face
22     m_cubo_q.normal(0,-1,0);
23     //sommets de la face
24     m_cubo_q.vertex(-1,-1,-1);
25     m_cubo_q.vertex(1,-1,-1);
26     m_cubo_q.vertex(-1,-1,1);
27     m_cubo_q.vertex(1,-1,1);
28     //On demande un nouveau strip
29     m_cubo_q.restart_strip();
30     //2 face
31     m_cubo_q.normal(0,1,0);
32
33     m_cubo_q.vertex(1,1,-1);
34     m_cubo_q.vertex(-1,1,-1);
35     m_cubo_q.vertex(1,1,1);
36     m_cubo_q.vertex(-1,1,1);
37
38     m_cubo_q.restart_strip();
39
40     //3 face
41     m_cubo_q.normal(1,0,0);
42
43     m_cubo_q.vertex(1,-1,1);
44     m_cubo_q.vertex(1,-1,-1);
45     m_cubo_q.vertex(1,1,1);
46     m_cubo_q.vertex(1,1,-1);
47
48     m_cubo_q.restart_strip();
49
50     //4 face
51     m_cubo_q.normal(-1,0,0);
52
53     m_cubo_q.vertex(-1,-1,-1);
54     m_cubo_q.vertex(-1,-1,1);
55     m_cubo_q.vertex(-1,1,-1);
56     m_cubo_q.vertex(-1,1,1);
57
58     m_cubo_q.restart_strip();
59
60     //5 face
61     m_cubo_q.normal(0,0,1);
62
63     m_cubo_q.vertex(-1,-1,1);
64     m_cubo_q.vertex(1,-1,1);
65     m_cubo_q.vertex(-1,1,1);
66     m_cubo_q.vertex(1,1,1);
67
68     m_cubo_q.restart_strip();
69
70     //6 face
71     m_cubo_q.normal(0,0,-1);
72
73     m_cubo_q.vertex(1,-1,-1);
74     m_cubo_q.vertex(-1,-1,-1);
75     m_cubo_q.vertex(1,1,-1);
76     m_cubo_q.vertex(-1,1,-1);
77
78     m_cubo_q.restart_strip();
79 }
```



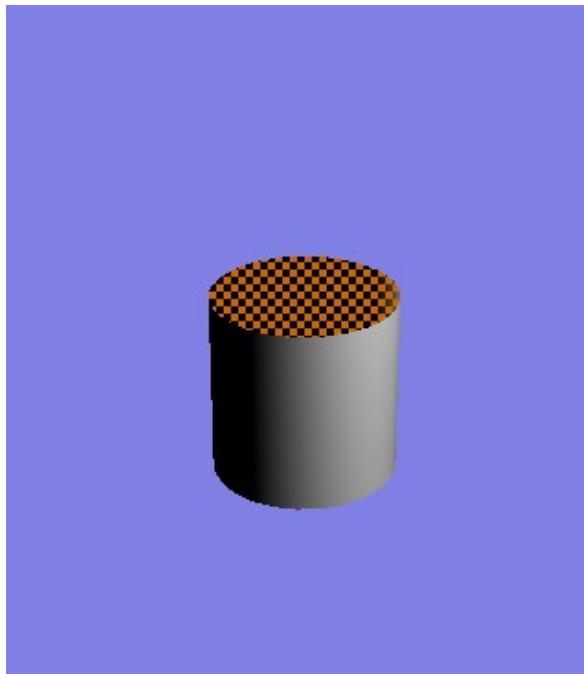
## b. avec une structure indexée

```
80
81 //Procédure pour initialiser un cube avec une structure indexée
82 void ViewerEtudiant::init_cubo_s()
83 {
84     //On choisit les primitives
85     m_cubo_s=Mesh(GL_TRIANGLE_STRIP);
86     //Sommets du cube
87     static float point[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}};
88     //Faces du cube
89     static int face[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}};
90     //Normales des faces
91     static float normal[6][3] = {{0,-1,0}, {0,1,0}, {1,0,0}, {-1,0,0}, {0,0,1}, {0,0,-1}};
92
93     //Maintenant on va placer la normale associé à chaque une des faces et les faces
94     for (int i=0; i<6; i++)
95     {
96         //Normal de la face i
97         m_cubo_s.normal(normal[i][0], normal[i][1], normal[i][2]);
98         //Sommets de la face i
99         m_cubo_s.texcoord(0,0);
100        m_cubo_s.vertex(point[face[i][0]][0], point[face[i][0]][1], point[face[i][0]][2]);
101        m_cubo_s.texcoord(1,0);
102        m_cubo_s.vertex(point[face[i][1]][0], point[face[i][1]][1], point[face[i][1]][2]);
103        m_cubo_s.texcoord(0,1);
104        m_cubo_s.vertex(point[face[i][3]][0], point[face[i][3]][1], point[face[i][3]][2]);
105        m_cubo_s.texcoord(1,1);
106        m_cubo_s.vertex(point[face[i][2]][0], point[face[i][2]][1], point[face[i][2]][2]);
107        m_cubo_s.restart_strip();
108    }
109 }
```



## 2) Définir un cylindre et un cône avec des GL\_TRIANGLE\_STRIP

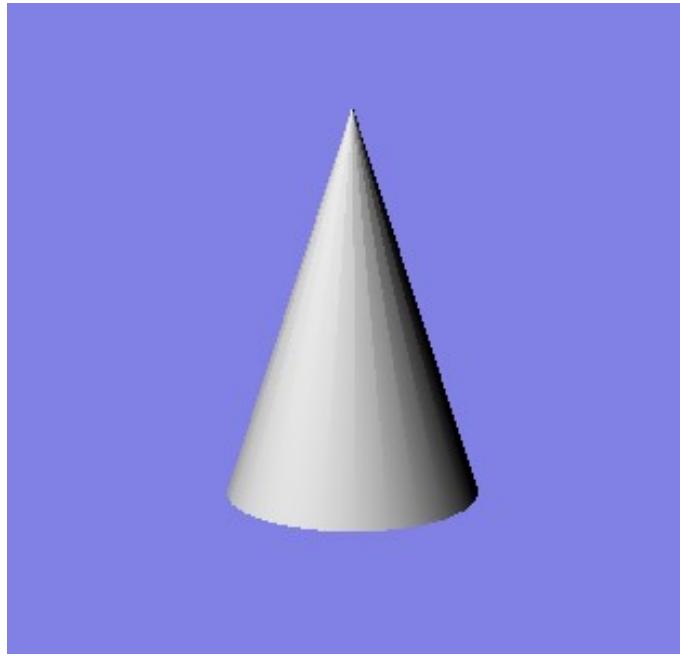
```
110 //Procedure pour initialiser un cylindre
111 void ViewerEtudiant::init_cylinder()
112 {
113     //Choix des primitives
114     m_cylinder=Mesh(GL_TRIANGLE_STRIP);
115
116     const int div = 25.0;    //Nombre de fois que le cylindre va etre coupe
117     float alpha;           //Angle qu'on va faire varie
118     float step = 2.0*M_PI/div;//Le pas qu'on va effectuer pour la creation des differents points
119
120     ///Maintenant on va place les normales et les points
121     for (int i=0;i<=div;i++)
122     {
123         //Variation de l'angle alpha de 0 a 2PI
124         alpha = i * step;
125         //Disque du bas
126         m_cylinder.normal(Vector(cos(alpha), 0, sin(alpha)));
127         m_cylinder.texcoord(float(i)/div,0);
128         m_cylinder.vertex(Point(cos(alpha), -1,sin(alpha)));
129         //Disque du haut
130         m_cylinder.normal(Vector(cos(alpha), 0, sin(alpha)));
131         m_cylinder.texcoord(float(i)/div,1);
132         m_cylinder.vertex(Point(cos(alpha),1,sin(alpha)));
133     }
134 }
135
136 }
```



```

137  ///Procedure pour initialiser un cone
138  void ViewerEtudiant::init_cone()
139  {
140      //Choix des primitives
141      m_cone=Mesh(GL_TRIANGLE_STRIP);
142
143      const int div = 25;      //Nombre de fois que le cone va etre coupe
144      float alpha;           //Angle qu'on va faire varier
145      float step = 2.0 * M_PI / (div); //Le pas qu'on va faire entre chaqu'un des points
146
147      //Maintenant on va placer les normales et les points
148      for (int i=0;i<=div;i++)
149      {
150          //Variation de l'angle alpha de 0 a 2PI
151          alpha = i * step;
152          //Cercle du bas
153          m_cone.normal(Vector( cos(alpha)/sqrtf(2.f),1.f/sqrtf(2.f),sin(alpha)/sqrtf(2.f) ));
154          m_cone.texcoord(float(i)/div,0);
155          m_cone.vertex( Point( cos(alpha), 0 ,sin(alpha)) );
156          //Pointe du cone
157          m_cone.normal(Vector( cos(alpha)/sqrtf(2.f),1.f/sqrtf(2.f),sin(alpha)/sqrtf(2.f) ));
158          m_cone.texcoord(float(i)/div,1);
159          m_cone.vertex( Point(0, 1, 0) );
160      }
161  }

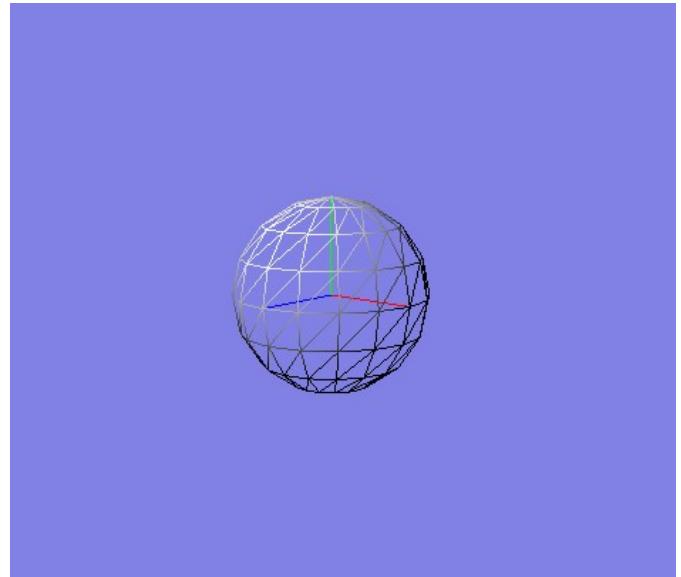
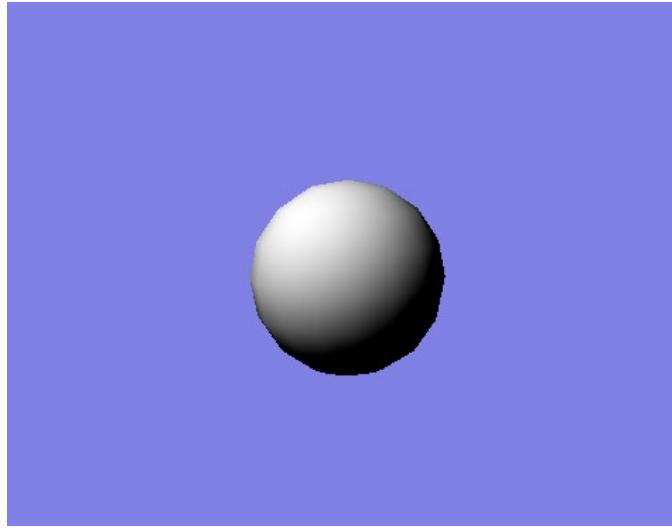
```



PATINO Nicolas  
11926177

### 3) Définir une sphère avec des GL\_TRIANGLE\_STRIP

```
189
190     //Procédure pour initialiser une sphère
191     void ViewerEtudiant::init_sphere()
192     {
193         //Choix des primitives
194         m_sphere = Mesh(GL_TRIANGLE_STRIP);
195
196         float beta, alphal, alpha2;      //Angles qu'on va faire varier
197         const int divBeta = 16;          //Nombre de fois qu'on va couper Beta
198         const int divAlpha = divBeta/2; //Nombre de fois qu'on va couper Alpha
199
200         //On fait varier alphal et alpha2, on fait la superposition des cercles
201         for(int i=0; i<divAlpha; i++)
202         {
203             alphal = -0.5f * M_PI + float(i) * M_PI / divAlpha;
204             alpha2 = -0.5f * M_PI + float(i+1) * M_PI / divAlpha;
205             //Variation de beta, on fait le dessin des sommets d'un cercle
206             for(int j=0; j<=divBeta; j++)
207             {
208                 beta = float(j) * 2.f * M_PI / (divBeta);
209                 m_sphere.normal(Vector(cos(alphal)*cos(beta), sin(alphal), cos(alphal)*sin(beta)));
210                 m_sphere.texcoord(beta/(2.0*M_PI), 0.5 + alphal/M_PI);
211                 m_sphere.vertex(Point(cos(alphal)*cos(beta), sin(alphal), cos(alphal)*sin(beta)));
212
213                 m_sphere.normal(Vector(cos(alpha2)*cos(beta), sin(alpha2), cos(alpha2)*sin(beta)));
214                 m_sphere.texcoord(beta/(2.0*M_PI), 0.5 + alpha2/M_PI);
215                 m_sphere.vertex(Point(cos(alpha2)*cos(beta), sin(alpha2), cos(alpha2)*sin(beta)));
216             }
217             //A chaque fois qu'on finit un cercle on demande un nouveau strip
218             m_sphere.restart_strip();
219         }
220     }
```



## 4) Ajouter les normales et les coordonnées textures à ces formes de base

```
80 //Procédure pour initialiser un cube avec une structure indexée
81 void ViewerEtudiant::init_cubo_s()
82 {
83     //On choisit les primitives
84     m_cubo_s=Mesh(GL_TRIANGLE_STRIP);
85     //Sommets du cube      1      2      3      4      5      6      7      8
86     static float point[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}};
87     //Faces du cube
88     static int face[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}};
89     //Normales des faces
90     static float normal[6][3] = {{0,-1,0}, {0,1,0}, {1,0,0}, {-1,0,0}, {0,0,1}, {0,0,-1}};
91
92     //Maintenant on va placer la normale associé a chaqu'une des faces et les faces
93     for (int i=0; i<6; i++)
94     {
95         //Normal de la face i
96         m_cubo_s.normal(normal[i][0], normal[i][1], normal[i][2]);
97         //Sommets de la face i
98         m_cubo_s.texcoord(0,0);
99         m_cubo_s.vertex(point[face[i][0]][0], point[face[i][0]][1], point[face[i][0]][2]);
100        m_cubo_s.texcoord(1,0);
101        m_cubo_s.vertex(point[face[i][1]][0], point[face[i][1]][1], point[face[i][1]][2]);
102        m_cubo_s.texcoord(0,1);
103        m_cubo_s.vertex(point[face[i][3]][0], point[face[i][3]][1], point[face[i][3]][2]);
104        m_cubo_s.texcoord(1,1);
105        m_cubo_s.vertex(point[face[i][2]][0], point[face[i][2]][1], point[face[i][2]][2]);
106        m_cubo_s.restart_strip();
107    }
108
109 }
```

```
603 //Procédure pour dessiner un cube avec structure indexée
604 void ViewerEtudiant::draw_cubo_s(const Transform& T)
605 {
606     gl.alpha(0.5f);
607     //Appel au texture
608     gl.texture(m_cubo_texture);
609     //Appel a la transformation
610     gl.model( T );
611     //Appel au mesh
612     gl.draw( m_cubo_s );
613 }
614
615 }
```



```

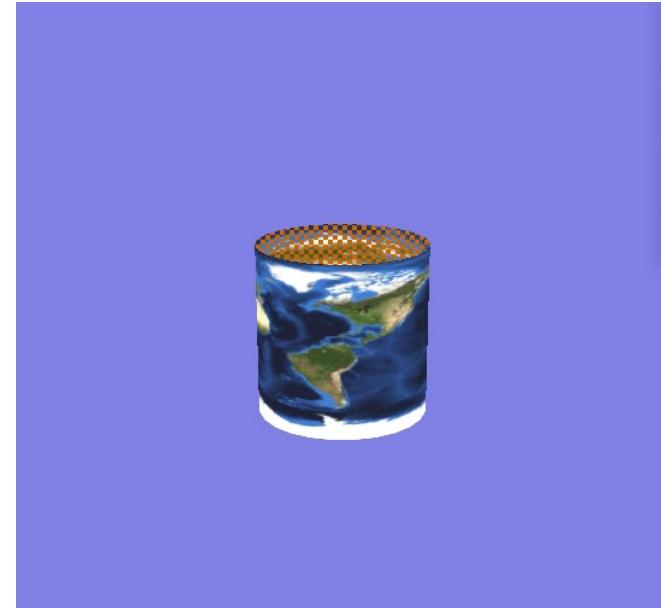
110
111     ///Procedure pour initialiser un cylindre
112     void ViewerEtudiant::init_cylinder()
113     {
114         //Choix des primitives
115         m_cylinder=Mesh(GL_TRIANGLE_STRIP);
116
117         const int div = 25.0;    //Nombre de fois que le cylindre va etre coupe
118         float alpha;           //Angle qu'on va faire varier
119         float step = 2.0*M_PI/div;//Le pas qu'on va effectuer pour la creation des differents points
120
121         ///Maintenant on va placer les normales et les points
122         for (int i=0;i<=div;i++)
123         {
124             //Variation de l'angle alpha de 0 a 2PI
125             alpha = i * step;
126             //Disque du bas
127             m_cylinder.normal(Vector(cos(alpha), 0, sin(alpha)));
128             m_cylinder.texcoord(float(i)/div,0);
129             m_cylinder.vertex(Point(cos(alpha), -1,sin(alpha)));
130             //Disque du haut
131             m_cylinder.normal(Vector(cos(alpha), 0, sin(alpha)));
132             m_cylinder.texcoord(float(i)/div,1);
133             m_cylinder.vertex(Point(cos(alpha),1,sin(alpha)));
134         }
135     }

```

```

613
614     ///Procedure pour dessiner un cylindre
615     void ViewerEtudiant::draw_cylinder(const Transform& T)
616     {
617         gl.texture(m_cubo_texture);
618         gl.model( T );
619         gl.draw( m_cylinder );
620     }
621
622
623

```



PATINO Nicolas  
11926177

```

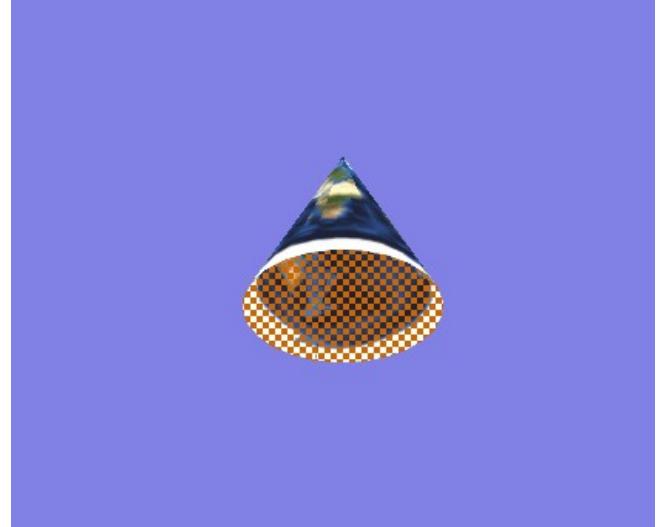
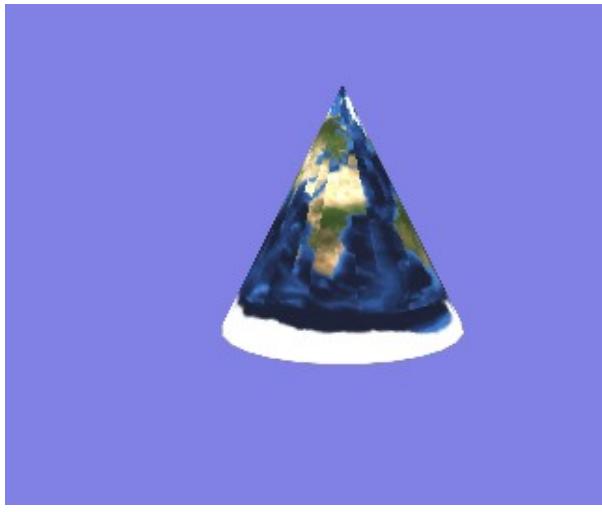
137  ///Procedure pour initialiser un cone
138  void ViewerEtudiant::init_cone()
139  {
140      //Choix des primitives
141      m_cone=Mesh(GL_TRIANGLE_STRIP);
142
143      const int div = 25;      //Nombre de fois que le cone va etre coupe
144      float alpha;           //Angle qu'on va faire varier
145      float step = 2.0 * M_PI / (div); //Le pas qu'on va faire entre chaque'un des points
146
147      //Maintenant on va placer les normales et les points
148      for (int i=0;i<=div;i++)
149      {
150          //Variation de l'angle alpha de 0 a 2PI
151          alpha = i * step;
152          //Cercle du bas
153          m_cone.normal(Vector( cos(alpha)/sqrtf(2.f),1.f/sqrtf(2.f),sin(alpha)/sqrtf(2.f) ));
154          m_cone.texcoord(float(i)/div,0);
155          m_cone.vertex( Point( cos(alpha), 0 ,sin(alpha)) );
156          //Pointe du cone
157          m_cone.normal(Vector( cos(alpha)/sqrtf(2.f),1.f/sqrtf(2.f),sin(alpha)/sqrtf(2.f) ));
158          m_cone.texcoord(float(i)/div,1);
159          m_cone.vertex( Point(0, 1, 0 ) );
160      }
161  }

```

```

623  ///Procedure pour dessiner un cone
624  void ViewerEtudiant::draw_cone(const Transform& T)
625  {
626      gl.texture(m_cubo_texture);
627      gl.modell( T );
628      gl.draw( m_cone );
629  }
630
631

```

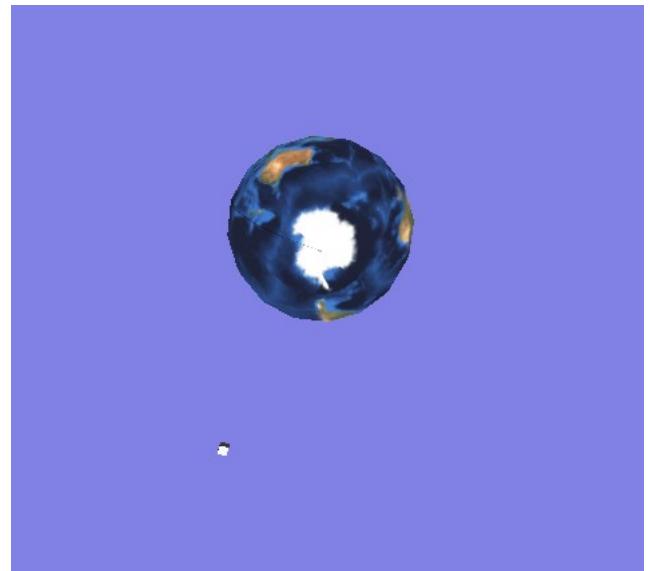


PATINO Nicolas  
11926177

```

189
190     ///Procedure pour initialiser une sphere
191     void ViewerEtudiant::init_sphere()
192     {
193         //Choix des primitives
194         m_sphere = Mesh(GL_TRIANGLE_STRIP);
195
196         float beta, alpha1, alpha2;      //Angles qu'on va faire varier
197         const int divBeta = 16;          //Nombre de fois qu'on va couper Beta
198         const int divAlpha = divBeta/2; //Nombre de fois qu'on va couper Alpha
199
200         //On fait varier alpha1 et alpha2, on fait la superposition des cercles
201         for(int i=0; i<divAlpha; i++)
202         {
203             alpha1 = -0.5f * M_PI + float(i) * M_PI / divAlpha;
204             alpha2 = -0.5f * M_PI + float(i+1) * M_PI / divAlpha;
205             //Variation de beta, on fait le dessin des sommets d'un cercle
206             for(int j=0; j<=divBeta; j++)
207             {
208                 beta = float(j) * 2.f * M_PI / (divBeta);
209                 m_sphere.normal(Vector(cos(alpha1)*cos(beta), sin(alpha1), cos(alpha1)*sin(beta)));
210                 m_sphere.texcoord(beta/(2.0*M_PI), 0.5 + alpha1/M_PI);
211                 m_sphere.vertex(Point(cos(alpha1)*cos(beta), sin(alpha1), cos(alpha1)*sin(beta)));
212
213                 m_sphere.normal(Vector(cos(alpha2)*cos(beta), sin(alpha2), cos(alpha2)*sin(beta)));
214                 m_sphere.texcoord(beta/(2.0*M_PI), 0.5 + alpha2/M_PI);
215                 m_sphere.vertex(Point(cos(alpha2)*cos(beta), sin(alpha2), cos(alpha2)*sin(beta)));
216             }
217             //A chaque fois qu'on finit un cercle on demande de nouveau un strip
218             m_sphere.restart_strip();
219         }
220     }
221
222
223
224     ///Procedure pour dessiner une sphere
225     void ViewerEtudiant::draw_sphere(const Transform& T)
226     {
227         gl.texture(m_globe_texture);
228         gl.model( T );
229         gl.draw( m_sphere );
230     }

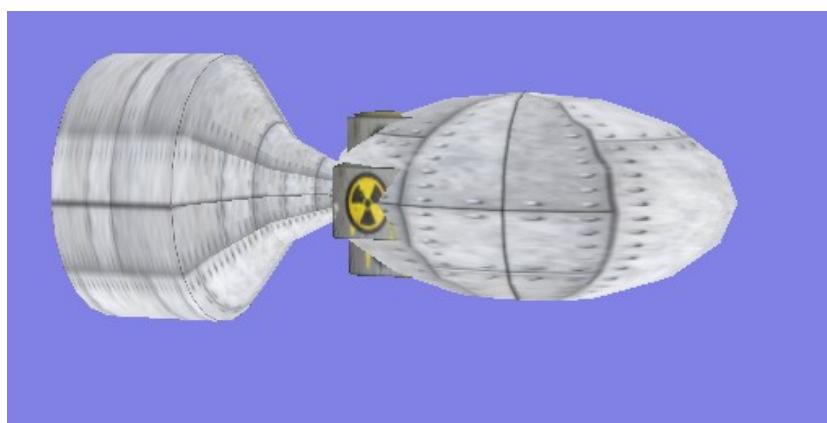
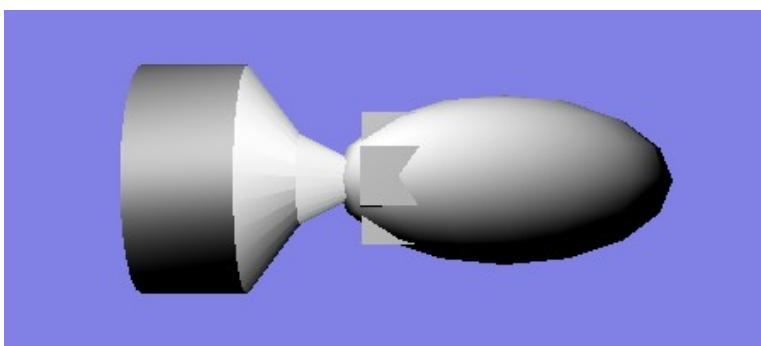
```



PATINO Nicolas  
11926177

## 2. Affichage à l'aide de transformations géométriques

```
639 //Procedure pour dessiner une Bombe
640 void ViewerEtudiant::dessineObjet(const Transform& T)
641 {
642     gl.alpha(0.5f);
643
644     gl.texture(m_globo_texture);
645     gl.model(T*Scale(3,3,6));
646     gl.draw( m_sphere);
647
648     gl.texture(m_cubo_texture);
649     gl.model(T*Scale(1,2,3,1)*Translation(0,0,4));
650     gl.draw(m_cube);
651
652     gl.model(T*Scale(2.3,1,1)*Translation(0,0,4));
653     gl.draw(m_cube);
654
655     gl.texture(m_globo_texture);
656     gl.model(T*Scale(2,2,4)*Translation(0,0,2)*RotationX(270));
657     gl.draw(m_cone);
658
659     gl.model(T*Scale(4,4,3)*Translation(0,0,3)*RotationX(270));
660     gl.draw(m_cone);
661
662     gl.model(T*Scale(4,4,1.8)*Translation(0,0,6)*RotationX(270));
663     gl.draw(m_cylinder);
664
665     gl.model(T*Scale(4,4,4)*Translation(0,0,3.1)*RotationX(270));
666     gl.draw(m_disque);
667
668
669 }
```



### 3. Terrain, texture, billboard (arbre) et cubemap

1) A partir d'une image interprétée comme une carte de hauteur, définir les sommets du terrain correspondant

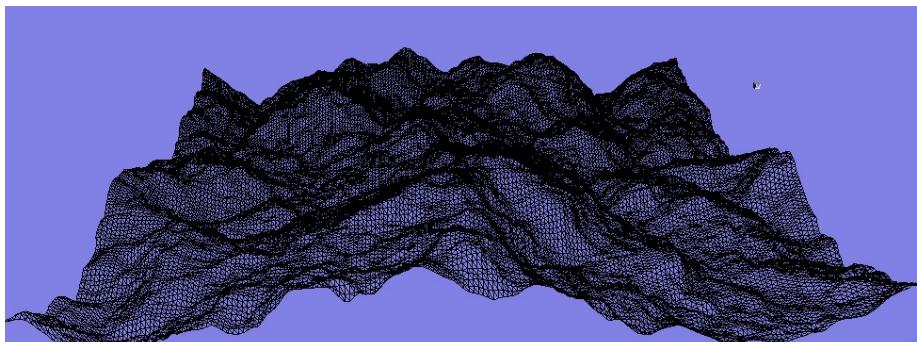
2) Définir les triangles formant le terrain

3) Ajouter le calcul de normal pour chaque sommet du maillage

```
220     }
221
222     ///Fonction qui retourne la normale au point (i,j) d'une image
223     Vector terNormal(const Image& im, const int i, const int j)
224     {
225         int ip = i-1;
226         int in = i+1;
227         int jp = j-1;
228         int jn = j+1;
229
230         //On calcule les points autour de (i,j)
231         Vector a( ip, im(ip, j).r, j );
232         Vector b( in, im(in, j).r, j );
233         Vector c( i, im(i, jp).r, jp );
234         Vector d( i, im(i, jn).r, jn );
235
236         //On calcule les vecteurs unitaires
237         Vector ab = normalize(b - a);
238         Vector cd = normalize(d - c);
239
240         //Puis on renvoi le produit de ces deux vecteurs qui va etre la normal au point (i,j)
241         Vector n = cross(ab,cd);
242
243         return n;
244     }
```

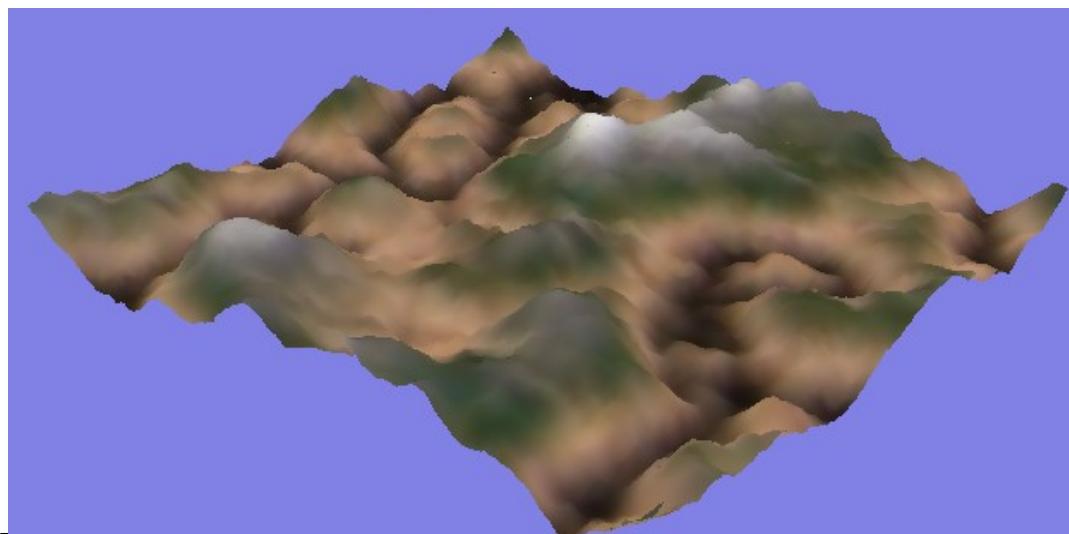
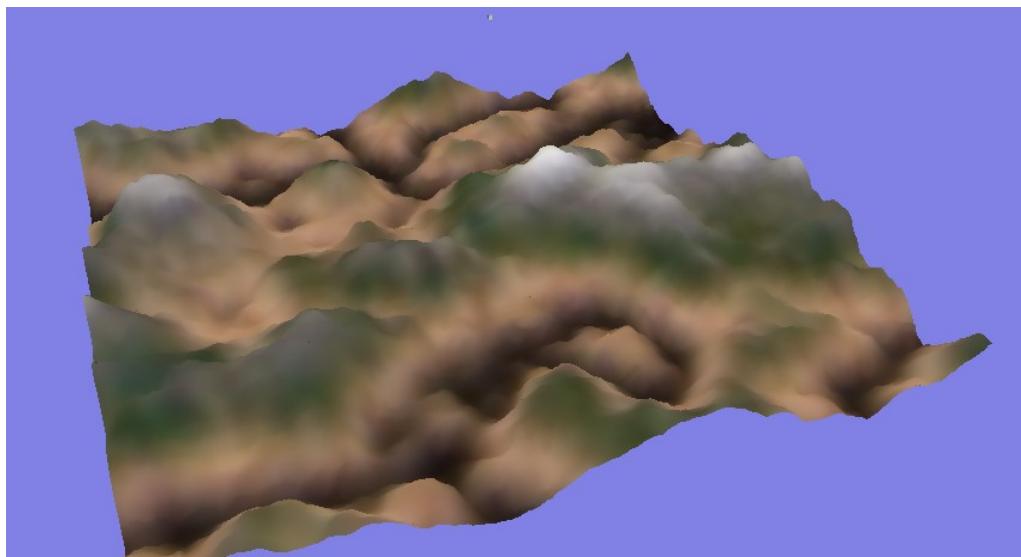


```
241
242     ///Procedure qui initialise un terrain a partir d'un height map en niveau de gris
243     void ViewerEtudiant::init_terrain(Mesh& m_terrain, const Image& im)
244     {
245         //Choix des primitives
246         m_terrain = Mesh(GL_TRIANGLE_STRIP);
247
248         //On parcourt la largeur et longeur de l'image pixel par pixel pour place les normales et les points
249         for(int i=1;i<im.width()-2;i++)
250         {
251             for(int j=1;j<im.height()-1;j++)
252             {
253                 //Calcule de la normale a (i+1,j)
254                 m_terrain.normal(terNormal(im,i+1,j));
255                 m_terrain.texcoord(float(i+1)/im.width(),float(j)/im.height());
256
257                 //On definit y grace a la couleur de l'image
258                 m_terrain.vertex(Point(i+1, 25.f*im(i+1,j).r,j));
259
260                 //Calcule de la normale a (i,j)
261                 m_terrain.normal(terNormal(im,i,j));
262                 m_terrain.texcoord(float(i)/im.width(),float(j)/im.height());
263
264                 //Meme chose ici
265                 m_terrain.vertex(Point(i, 25.f*im(i,j).r,j));
266
267             }
268
269         }
270
271         //On demande un nouveau strip au final de chaque iteration
272         m_terrain.restart_strip();
273
274     }
```



## 4) Ajouter les coordonnées textures

```
241 //Procedure qui initialise un terrain a partir d'un height map en niveau de gris
242 void ViewerEtudiant::init_terrain(Mesh& m_terrain, const Image& im)
243 {
244     //Choix des primitives
245     m_terrain = Mesh(GL_TRIANGLE_STRIP);
246
247     ///On parcourt la largeur et longeur de l'image pixel par pixel pour place les normales et les points
248     for(int i=1;i<im.width()-2;i++)
249     {
250         for(int j=1;j<im.height()-1;j++)
251         {
252             //Calcule de la normale a (i+1,j)
253             m_terrain.normal(terNormal(im,i+1,j));
254             m_terrain.texcoord(float(i+1)/im.width(),float(j)/im.height());
255             //On definie y grace a la couleur de l'image
256             m_terrain.vertex(Point(i+1, 25.f*im(i+1,j).r,j));
257             //Calcule de la normale a (i,j)
258             m_terrain.normal(terNormal(im,i,j));
259             m_terrain.texcoord(float(i)/im.width(),float(j)/im.height());
260             //Meme chose ici
261             m_terrain.vertex(Point(i, 25.f*im(i,j).r,j));
262         }
263         //On demande un nouveau strip au final de chaque iteration
264         m_terrain.restart_strip();
265     }
266 }
267
268 }
```



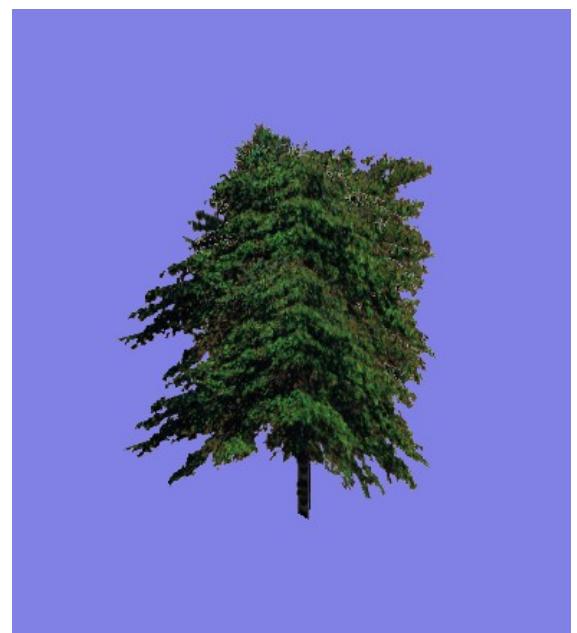
PATINO Nicolas  
11926177

## 5) Afficher un arbre représenté par un billboard

```
269 //Procedure qui initialise un billboard  
270 void ViewerEtudiant::init_billboard()  
271 {  
272     //Choix des primitives  
273     m_quad = Mesh(GL_TRIANGLE_STRIP);  
274     ///Construction des quad de tel facon que l'image soit toujours face a la camera  
275     //1 quad  
276     m_quad.texcoord(0,0);  
277     m_quad.vertex(-1,-1,0);  
278     m_quad.texcoord(0,1);  
279     m_quad.vertex(-1,1,0);  
280     m_quad.texcoord(1,0);  
281     m_quad.vertex(1,-1,0);  
282     m_quad.texcoord(1,1);  
283     m_quad.vertex(1,1,0);  
284     //On demande un nouveau strip apres la creation du premier quad  
285     m_quad.restart_strip();  
286     //2 quad  
287     m_quad.texcoord(0,0);  
288     m_quad.vertex(0,-1,-1);  
289     m_quad.texcoord(0,1);  
290     m_quad.vertex(0,1,-1);  
291     m_quad.texcoord(1,0);  
292     m_quad.vertex(0,-1,1);  
293     m_quad.texcoord(1,1);  
294     m_quad.vertex(0,1,1);  
295 }  
296  
297 }
```



```
679  
680     ///Procedure pour dessiner un arbre  
681     void ViewerEtudiant::draw_billboard(const Transform &T)  
682     {  
683         gl.alpha(0.5f);  
684         Transform tt=T;  
685         gl.model(tt );  
686         gl.texture(m_arbre_texture);  
687         gl.draw( m_quad);  
688  
689         gl.model(tt *RotationY(180));  
690         gl.draw( m_quad);  
691     }  
692  
693 }
```



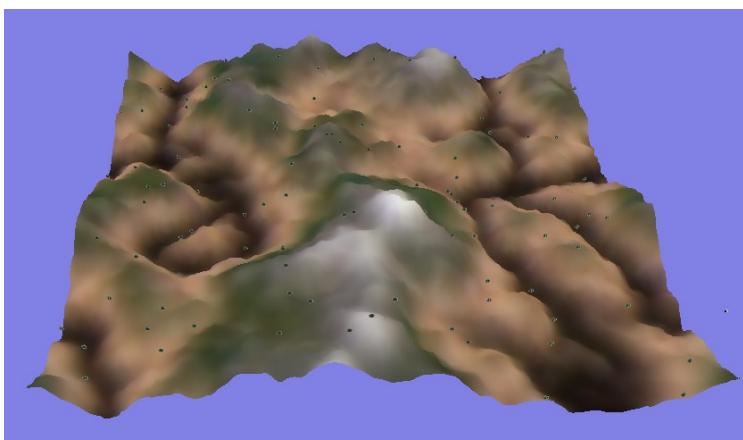
## 6) Afficher un ensemble d'arbres sur le terrain en utilisant la carte de hauteur pour les positionner.

```
int ViewerEtudiant::init()
```

```
559
560     //Initialisation des coordonnes des billboards
561     float i,j,x,y;
562     //Arbre
563     for (int k=0; k<Narbre; k++)
564     {
565         do{
566             //On prend des nombres au hasard entre 0 et la longeur et larguer de l'image
567             i =rand() % m_terrainAlti.width();
568
569             j = rand() % m_terrainAlti.height();
570             //Pour positioner les arbres seulement dans une zone, ou 25.f*m_terrainAlti(i, j).r+1 est la position Y des billboards
571             }while(i>192/4 || j>(-i+48) || 25.f*m_terrainAlti(i, j).r+1<=5);
572             //On garde les coordonees dans un tableau
573             tab[k]=vec2(i,j);
574
575     }
576     //Palmiers
```

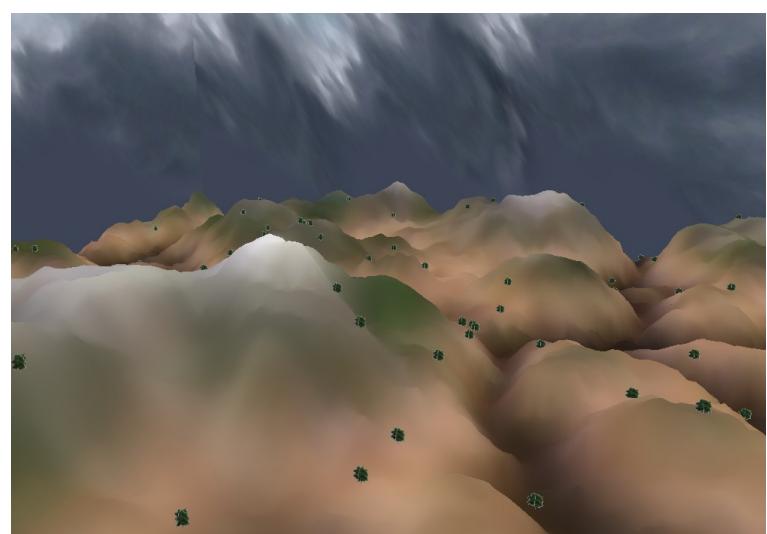
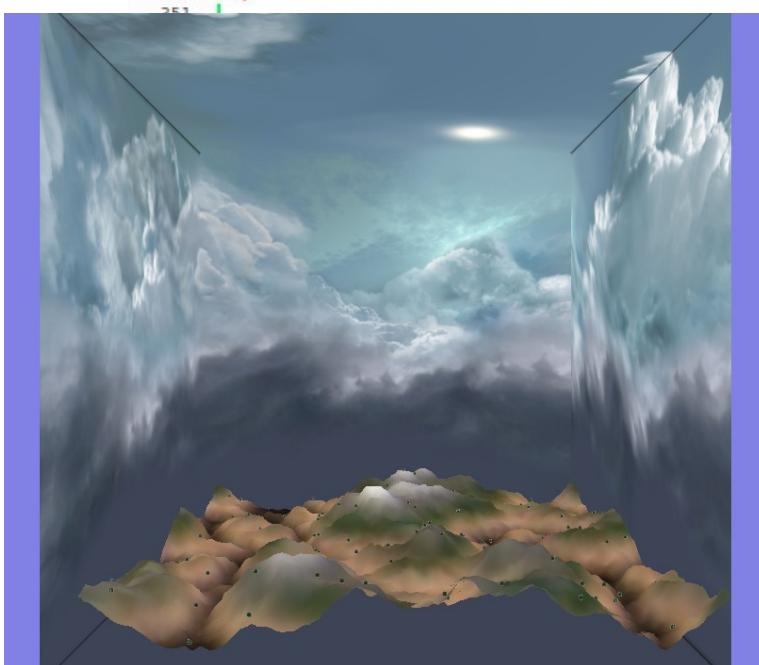
```
int ViewerEtudiant::render()
```

```
849
850     draw_terrain(T);
851     //On parcours le tableau avec les coordonees des arbres
852     for (int k=0; k<Narbre; k++)
853     {
854         //On applique une translation pour pouvoir place l'arbre a la bonne place par rapport au terrain
855         Transform Ttt =T*Translation(tab[k].x, 25.f*m_terrainAlti(tab[k].x, tab[k].y).r+1, tab[k].y);
856         draw_billboard(Ttt);
857     }
858     //De meme
```



## 7) Afficher un cube texturé autour de votre scène

```
297 //Procedure qui initialise un cubemap
298 void ViewerEtudiant::init_map()
299 {
300     //Choix des primitives
301     m_cubo_map=Mesh(GL_TRIANGLE_STRIP);
302     //Sommets du cube
303     static float point[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}};
304     //Faces du cube
305     static int face[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}};
306     //Normales des faces
307     static float normal[6][3] = {{0,-1,0}, {0,1,0}, {1,0,0}, {-1,0,0}, {0,0,1}, {0,0,-1}};
308     //Coordonnees des texcoord pour chaqu'une des faces
309     static vec2 txc[6][4]={
310         //Bas
311         {{1/4.f,0.f},{2/4.f,0.f},{2/4.f,1/3.f},{1/4.f,1/3.f}},
312         //Haut
313         {{2/4.f,1.f},{1/4.f,1.f},{1/4.f,2/3.f},{2/4.f,2/3.f}},
314         //Right
315         {{1/4.f,1/3.f},{3/4.f,1/3.f},{3/4.f,2/3.f},{2/4.f,2/3.f}},
316         //Left
317         {{0.f,1/3.f},{1/4.f,1/3.f},{1/4.f,2/3.f},{0.f,2/3.f}},
318         //Front
319         {{1/4.f,1/3.f},{2/4.f,1/3.f},{2/4.f,2/3.f},{1/4.f,2/3.f}},
320         //Back
321         {{1/4.f,1/3.f},{1.f,1/3.f},{1.f,2/3.f},{3/4.f,2/3.f}}
322     };
323     //Maintenant on va placer la normale associé a chaqu'une des faces
324     for (int i=0; i<6; i++)
325     {
326         //Normal de la face i
327         m_cubo_map.normal(normal[i][0],-1*normal[i][1], normal[i][2]);
328         //Sommets de la face i
329         m_cubo_map.texcoord(txc[i][0].x,txc[i][0].y);
330         //Ici on inverse la coordonnee z pour qu'on voit l'interieur du cube
331         m_cubo_map.vertex(point[face[i][0]][0], point[face[i][0]][1], -point[face[i][0]][2]);
332         m_cubo_map.texcoord(txc[i][1].x,txc[i][1].y);
333         m_cubo_map.vertex(point[face[i][1]][0], point[face[i][1]][1], -point[face[i][1]][2]);
334         m_cubo_map.texcoord(txc[i][3].x,txc[i][3].y);
335         m_cubo_map.vertex(point[face[i][3]][0], point[face[i][3]][1], -point[face[i][3]][2]);
336         m_cubo_map.texcoord(txc[i][2].x,txc[i][2].y);
337         m_cubo_map.vertex(point[face[i][2]][0], point[face[i][2]][1], -point[face[i][2]][2]);
338         //On demande un nouveau strip
339         m_cubo_map.restart_strip();
340     }
341 }
```



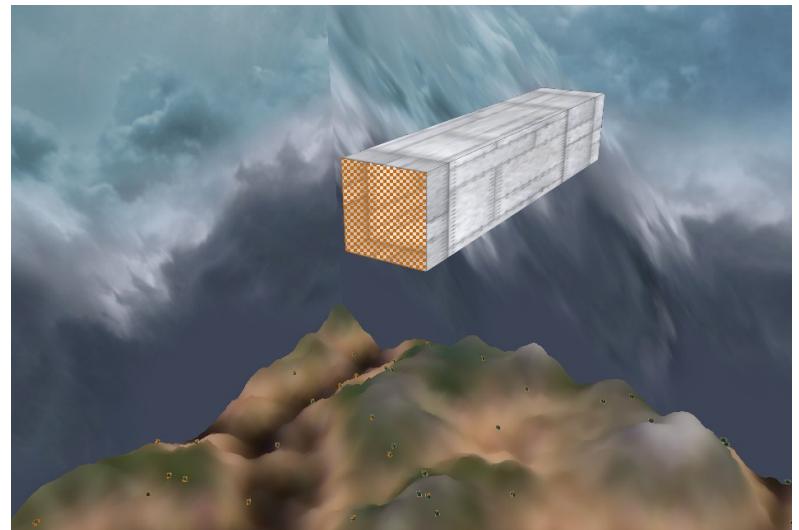
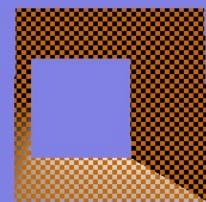
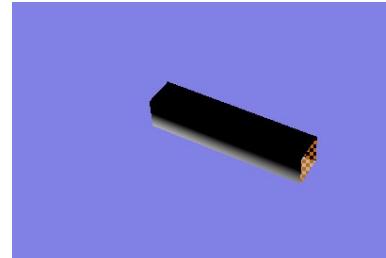
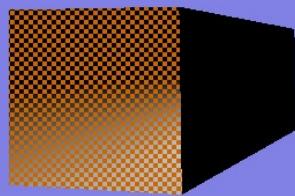
## 4. Modélisation d'un objet par extrusion

```
352 //Procédure qui crée la forme de la figure qui va subir l'extrusion
353 void ViewerEtudiant::creation_forme()
354 {
355     // Nombre de points de la silhouette 2D
356     NBPT = 5; // déclaré dans la class ViewerEtudiant
357
358     Point objt_p[NBPT];
359     /// Points de la silhouette 2D
360
361     objt_p[0] =Point(0.0,0.0,0.0);
362     objt_p[1] =Point(0.0,2.0,0.0);
363     objt_p[2] = Point(2.0,2.0,0.0);
364     objt_p[3] = Point(2.0, 0.0, 0.0);
365     objt_p[4]=Point(0.0,0.0,0.0);
366
367     float prof = 5.0;
368
369     //On applique la matrice de translation pour la coordonne z
370     for(int i=0; i < 2; i++){
371         for(int j=0; j < NBPT; j++){
372             objt_v[i][j].x =objt_p[j].x;
373             objt_v[i][j].y =objt_p[j].y;
374             objt_v[i][j].z =objt_p[j].z+i*prof;
375         }
376     }
377
378     //On initialise tout les normales nulles
379     for(int i=0; i<2; i++)
380         for(int j=0; j<NBPT; j++)
381             objt_vn[i][j] = Vector(0, 0, 0);
382
383     //On initialise les normales
384     for(int i=0; i<NBPT-1; i++){
385         Vector a, b, vntmp;
386         a = normalize(objt_v[0][i] - objt_v[1][i+1]);
387         b = normalize(objt_v[0][i] - objt_v[1][i]);
388         vntmp = cross(a, b);
389         //On transmet cette normal au quatres sommets de la face
390         objt_vn[0][i] = vntmp + objt_vn[0][i];
391         objt_vn[1][i] = vntmp + objt_vn[1][i];
392         objt_vn[1][i+1] = vntmp + objt_vn[1][i+1];
393         objt_vn[0][i+1] = vntmp + objt_vn[0][i+1];
394     }
395     //Normale a un sommet est la moyenne des 4 sommets de la face
396     for(int i=0; i<2; i++){
397         for(int j=0; j<NBPT; j++){
398             float q = 4.0f;
399             if (j == NBPT-1)
400                 q = 2.0f;
401             objt_vn[i][j] = objt_vn[i][j] / q;
402         }
403     }
```

```

484 //Procedure qui initialise l'objet
485 void ViewerEtudiant::init_obj_extru()
486 {
487     //Choix des primitives
488     m_extru = Mesh(GL_TRIANGLES);
489
490     //On parcours chaque point pour place la normale et les coordonnes des points
491     for(int j=0; j<NBPT-1; j++)
492     {
493
494         //1er triangle
495         m_extru.texcoord(0,0);
496         m_extru.normal(objt_vn[1][j]);
497         m_extru.vertex(objt_v[1][j].x, objt_v[1][j].y, objt_v[1][j].z);
498
499         m_extru.texcoord(1,1);
500         m_extru.normal(objt_vn[0][j+1]);
501         m_extru.vertex(objt_v[0][j+1].x, objt_v[0][j+1].y, objt_v[0][j+1].z);
502
503         m_extru.texcoord(1,0);
504         m_extru.normal(objt_vn[0][j]);
505         m_extru.vertex(objt_v[0][j].x, objt_v[0][j].y, objt_v[0][j].z);
506
507
508         //2eme triangle
509
510         m_extru.texcoord(0,0);
511         m_extru.normal(objt_vn[1][j]);
512         m_extru.vertex(objt_v[1][j].x, objt_v[1][j].y, objt_v[1][j].z);
513
514         m_extru.texcoord(0,1);
515         m_extru.normal(objt_vn[1][j+1]);
516         m_extru.vertex(objt_v[1][j+1].x, objt_v[1][j+1].y, objt_v[1][j+1].z);
517
518         m_extru.texcoord(1,1);
519         m_extru.normal(objt_vn[0][j+1]);
520         m_extru.vertex(objt_v[0][j+1].x, objt_v[0][j+1].y, objt_v[0][j+1].z);
521
522     }
523 }

```



PATINO Nicolas  
11926177

## 5. Texture animée

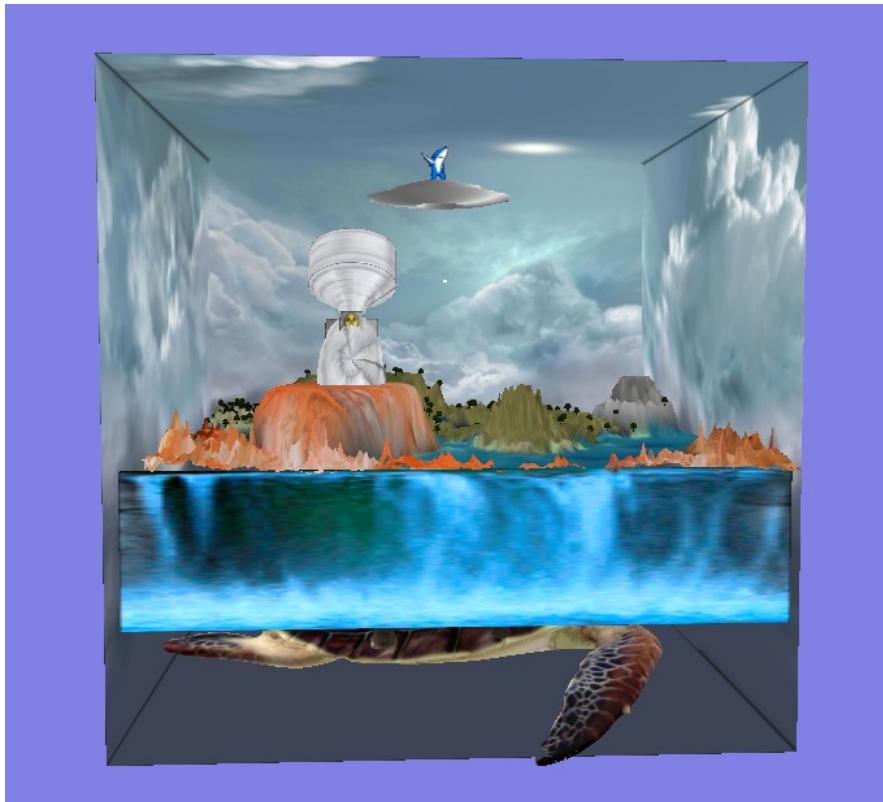
```
443 //Procedure qui initialise un quad ou va etre present une texture animee
444 void ViewerEtudiant::init_iman()
445 {
446     m_iman = Mesh(GL_TRIANGLE_STRIP);
447 
448     m_iman.normal( 0, 0, 1 );
449 
450     m_iman.texcoord(0,0);
451     m_iman.vertex(-1, -1, 0 );
452 
453     m_iman.texcoord(1,0);
454     m_iman.vertex( 1, -1, 0 );
455 
456     m_iman.texcoord(0,1);
457     m_iman.vertex( -1, 1, 0 );
458 
459     m_iman.texcoord( 1,1);
460     m_iman.vertex( 1, 1, 0 );
461 
462 }
463 
464 
465 //Procedure pour dessiner une texture animee (requin)
466 void ViewerEtudiant::draw_iman(const Transform& T)
467 {
468     //Nombre d'images
469     int x=10;
470     //Le compteur est une valeur entier qui va augmenter donc selon le resultat
471     //du compteur modulo x, les images vont varier et vont changer au cours du temps
472     if(compteur%x==0)gl.texture(m_iman_texture1);
473 
474     if(compteur%x==1)gl.texture(m_iman_texture2);
475 
476     if(compteur%x==2)gl.texture(m_iman_texture3);
477 
478     if(compteur%x==3)gl.texture(m_iman_texture4);
479 
480     if(compteur%x==4)gl.texture(m_iman_texture5);
481 
482     if(compteur%x==5)gl.texture(m_iman_texture6);
483 
484     if(compteur%x==6)gl.texture(m_iman_texture7);
485 
486     if(compteur%x==7)gl.texture(m_iman_texture8);
487 
488     if(compteur%x==8)gl.texture(m_iman_texture9);
489 
490     if(compteur%x==9)gl.texture(m_iman_texture10);
491 
492     gl.alpha(0.5f);
493 
494     gl.model(T);
495     gl.draw(m_iman);
496 
497     gl.model(T*RotationY(180));
498     gl.draw(m_iman);
499 
500 }
```

```
873
874  /*
875   * Fonction dans laquelle les mises à jours sont effectuées.
876   */
877 int ViewerEtudiant::update( const float time, const float delta )
878 {
879     // time est le temps écoulé depuis le démarrage de l'application, en millisecondes,
880     // delta est le temps écoulé depuis l'affichage de la dernière image / le dernier appel à draw(), en millisecondes.
881     float top=time/100;
882     int ta=int(top);
883     //Le compteur est un entier qui va augmenter par rapport au temps
884     compteur=ta;
885
886
887     return 0;
888 }
```

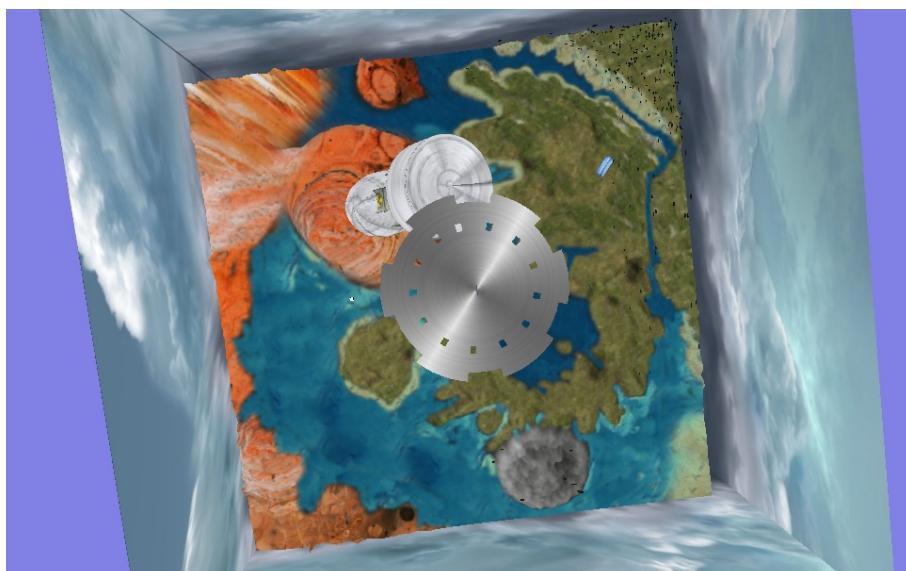


PATINO Nicolas  
11926177

# SCÈNE FINAL



PATINO Nicolas  
11926177



PATINO Nicolas  
11926177