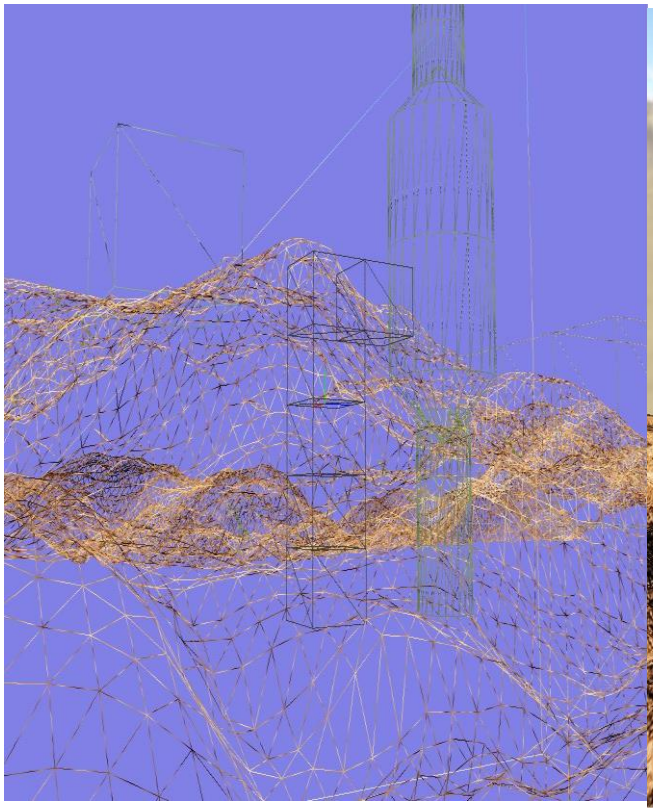
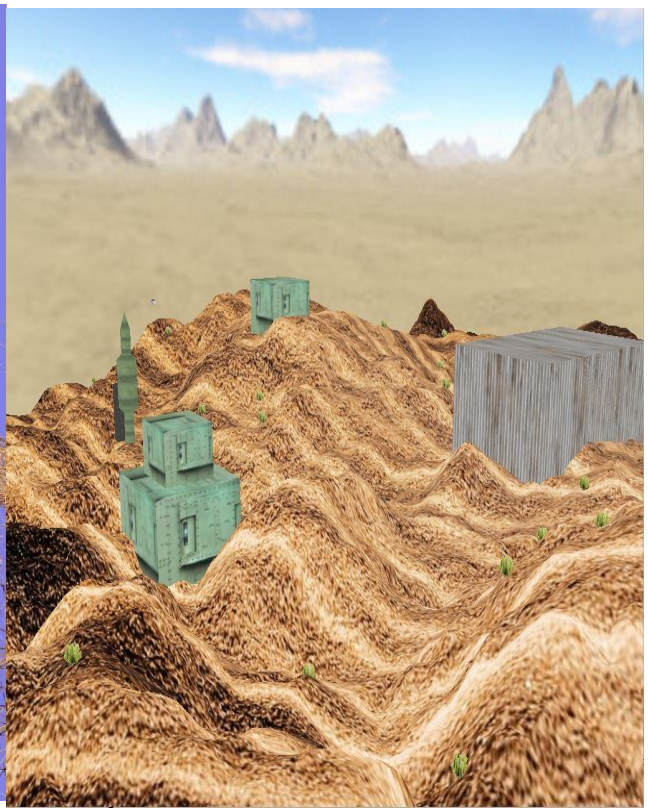
RAPPORT LIFGRAPHIQUE
MAP
VAGANAY AXEL

Résultats

1. Manipulation des formes de base

Cylindre

Pour le cylindre, il y a 2 init: 1 pour le disque inférieur et supérieur (init_disque) et 1 pour le rectangle qui entoure le cylindre.
Ajout des texcoord seulement pour le init_cylindre car les disque ne sont jamais visible sur mon objet complexe donc ils n'ont pas besoin de charger de texture inutilement.

```
void ViewerEtudiant::init_disque()
{
    // Variation de l'angle de 0 à 2!
    const int div = 25;
    float alpha;
    float step = 2.0 * M_PI / (div);
    // Choix primitive OpenGL
    m_disque = Mesh( GL_TRIANGLE_FAN );
    m_disque.normal( Vector(0,1,0) ); // Normale à la surface
    m_disque.vertex( Point(0,0,0) ); // Point du centre du disque
    // Variation de l'angle de 0 à 2!
    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)) );
    }
}

void ViewerEtudiant::init_cylinder()
{
    // Variation de l'angle de 0 à 2!
    int i;
    const int div = 25;
    float alpha;
    float step = 2.0 * M_PI / (div);
    // Choix primitive OpenGL
    m_cylindre = Mesh(GL_TRIANGLE_STRIP);

    for (int i=0; i<=div; ++i)
    {
        // Variation de l'angle de 0 à 2!
        alpha = i * step;

        m_cylindre.texcoord(alpha/float(2.0 * M_PI),0);
        m_cylindre.normal( Vector(cos(alpha), 0, sin(alpha)) );
        m_cylindre.vertex( Point(cos(alpha), -1, sin(alpha)) );


        m_cylindre.texcoord(alpha/float(2.0 * M_PI),1);
        m_cylindre.normal( Vector(cos(alpha), 0, sin(alpha)) );
        m_cylindre.vertex( Point(cos(alpha), 1, sin(alpha)) );
    }
}
```

```
void ViewerEtudiant::draw_disque(const Transform& T)
{
    // gl.model(T*Scale(0.5,0.5,0.5));
    gl.model(T);

    //gl.texture(tex);
    //gl.lighting(false);
    gl.draw(m_disque);
    //gl.lighting(true);
}

void ViewerEtudiant::draw_cylinder(const Transform& T, unsigned int tex)
{
    // gl.model(T*Scale(0.5,0.5,0.5));
    Transform Tcylindre = T * Scale(1.5);    //taille xl.5
    gl.model(Tcylindre);
    gl.texture(tex);
    //gl.lighting(false);
    gl.draw(m_cylindre);

    Transform Tdisquehaut = Tcylindre * Translation( 0, 1, 0) * Rotation(Vector(1,0,0), 180);   //le disque du haut suit le cylindre et est retourner pour avoir la face visible en haut
    gl.model(Tdisquehaut);  //disque du haut
    gl.draw(m_disque);

    Transform Tdisquebas = Tcylindre * Translation( 0, -1, 0);       //le disque du bas suit le cylindre
    gl.model(Tdisquebas);     //disque du bas
    gl.draw(m_disque);
    //gl.lighting(true);
}
```

Cône

A noter que le draw_cone appel le même draw_disque que le cylindre.
Ce init_cone a des texcoord car il est utilisé sur mon objet complexe.

```
void ViewerEtudiant::init_cone()
{
    // Variation de l'angle de 0 à 2!
    const int div = 25;
    float alpha;
    float step = 2.0 * M_PI / (div);
    // Choix de la primitive OpenGL
    m_cone = Mesh(GL_TRIANGLE_STRIP);
    for (int i=0;i<=div;++i)
    {
        alpha = i * step; // Angle varie de 0 à 2!
        m_cone.texcoord(alpha/float(2.0 * M_PI),0);
        m_cone.normal(Vector( cos(alpha)/sqrtf(2.f),1.f/sqrtf(2.f),sin(alpha)/sqrtf(2.f) ));
        m_cone.vertex( Point( cos(alpha), 0, sin(alpha) ));

        m_cone.texcoord(0.5,1);
        m_cone.normal(Vector( cos(alpha)/sqrtf(2.f),1.f/sqrtf(2.f),sin(alpha)/sqrtf(2.f) ));
        m_cone.vertex( Point(0, 1, 0) );
    }
}
```

```
void ViewerEtudiant::draw_cone(const Transform& T, unsigned int tex)
{
    // gl.model(T*Scale(0.5,0.5,0.5));
    Transform Tcone = T * Scale(2);
    gl.model(Tcone);
    gl.texture(tex);
    //gl.texture(tex);
    //gl.lighting(false);
    gl.draw(m_cone);

    Transform Tdisquebas = Tcone * Translation( 0, 0, 0);    //le disque du bas suit le cone
    gl.model(Tdisquebas);  //disque du bas
    gl.draw(m_disque);
}
```

Sphère

La sphère n'as pas de texcoord car je ne l'utilise pas sur ma map.

```
void ViewerEtudiant::init_sphere()
{
    // Variation des angles alpha et beta
    const int divBeta = 16;
    const int divAlpha = divBeta/2;
    int i,j;
    float beta, alpha, alpha2;
    // Choix de la primitive OpenGL
    m_sphere = Mesh(GL_TRIANGLE_STRIP);
    /*
     * Fonction dans laquelle les initialisations sont faites.
     */
    // Variation des angles alpha et beta
    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.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.vertex( Point(cos(alpha2)*cos(beta), sin(alpha2), cos(alpha2)*sin(beta)) );
        } // boucle sur les j, angle beta, dessin des sommets d'un cercle
            m_sphere.restart_strip(); // Demande un nouveau strip
    } // boucle sur les i, angle alpha, sphère = superposition de cercles
}
```

```
void ViewerEtudiant::draw_sphere(const Transform& T)
{
    // gl.model(T*Scale(0.5,0.5,0.5));
    Transform Tsphere = T * Scale(2);  //taille x2
    gl.model(Tsphere);
    //gl.texture(tex);
    //gl.lighting(false);
    gl.draw(m_sphere);
}
```

2. Affichage à l'aide de transformations géométriques

Objet complexe fusée

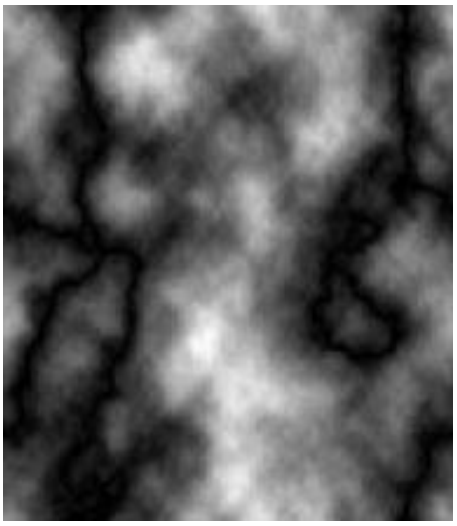
La fusée n'as pas de init_fusee car c'est simplement un assemblage des formes simples vu précédemment.

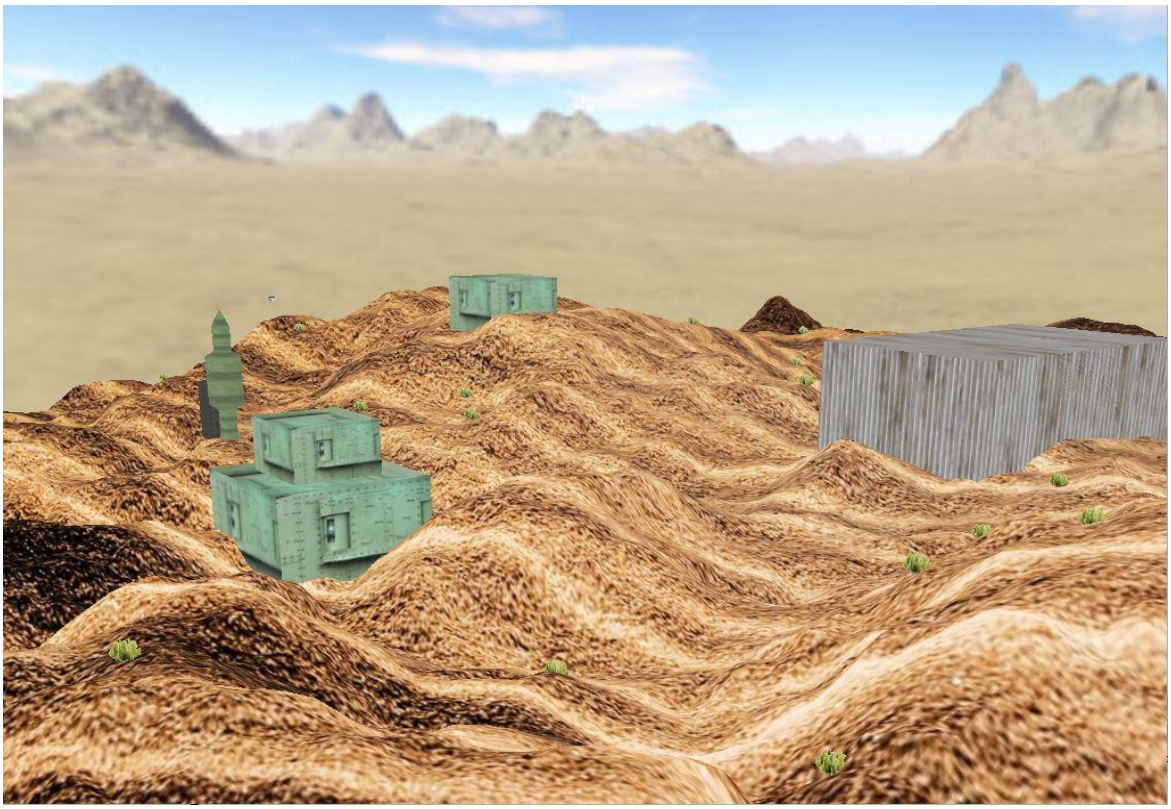
```
void ViewerEtudiant::draw_fusee(const Transform& T, GLuint t_missile)
{
    Transform Tdisquebas = T * Translation( 0, -1, 0);          //disque bas
    gl.model(Tdisquebas);
    gl.draw(m_disque);
    Transform Tcylindre1 = T;                                   //cyl 1
    gl.texture(t_missile);
    gl.model(Tcylindre1);
    gl.draw(m_cylindre);
    Transform Tcylindre2 = Tcylindre1 * Translation(0,2,0);     //cyl 2
    gl.texture(t_missile);
    gl.model(Tcylindre2);
    gl.draw(m_cylindre);
    Transform Tcylindre3 = Tcylindre2 * Translation(0,2,0);     //cyl 3
    gl.texture(t_missile);
    gl.model(Tcylindre3);
    gl.draw(m_cylindre);
    Transform Tcone1 = Tcylindre2 * Scale(2) * Translation(0,2,0) * Rotation(Vector(1,0,0),180) ;   //cone 1
    gl.model(Tcone1);
    gl.draw(m_cone);
    Transform Tcylindre4 = Tcylindre3 * Translation(0,4,0) * Scale(2);     //cyl 4
    gl.model(Tcylindre4);
    gl.draw(m_cylindre);
    Transform Tcylindre5 = Tcylindre4 * Translation(0,2,0);     //cyl 5
    gl.model(Tcylindre5);
    gl.draw(m_cylindre);
    Transform Tcone2 = Tcylindre4 * Translation(0,3,0) ;   //cone 2
    gl.model(Tcone2);
    gl.draw(m_cone);
    Transform Tcylindre6 = Tcylindre5 * Translation(0,2,0) * Scale(0.5);     //cyl 6
    gl.model(Tcylindre6);
    gl.draw(m_cylindre);
    Transform Tcylindre7 = Tcylindre6 * Translation(0,2,0);     //cyl 7
    gl.model(Tcylindre7);
    gl.draw(m_cylindre);
    Transform Tcone3 = Tcylindre6 * Translation(0,3,0) * Scale(1,3,1);     //cone 3   le scale allonge la  pointe x3
    gl.model(Tcone3);
    gl.draw(m_cone);
}
```

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

Terrain

Le terrain est un grand rectangle qui a des hauteurs de sols qui varient en fonction de la carte des hauteurs ci-dessous et auquel j'ai posé une texture de sable.



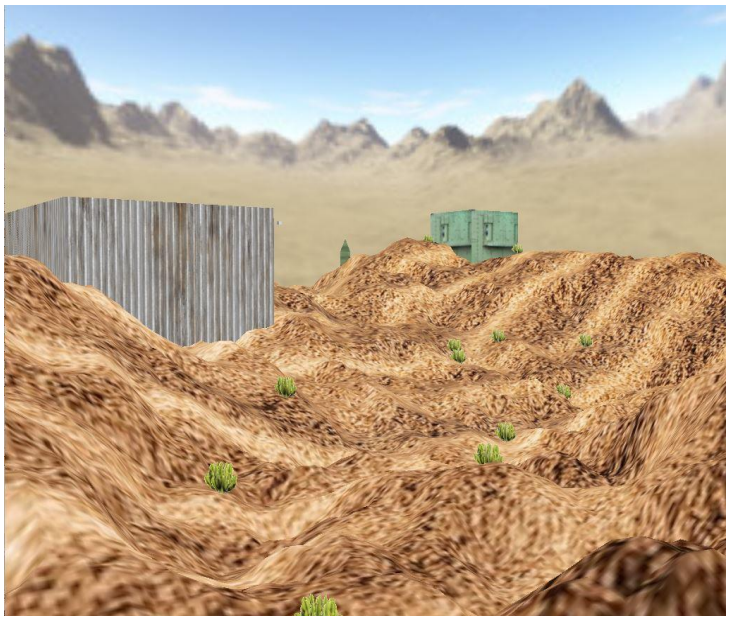


```
Vector terrainNormal(const Image& im, const int i, const int j)
{
    // Calcul de la normale au point (i,j) de l'image
    int ip = i-1;
    int in = i+1;
    int jp = j-1;
    int jn = j+1;
    Vector a( ip, im(ip, j).r, j );
    Vector b( in, im(in, j).r, j );
    Vector c( i, im(i, jp).r, jp );
    Vector d( i, im(i, jn).r, jn );
    Vector ab = normalize(b - a);
    Vector cd = normalize(d - c);
    Vector n = cross(ab,cd);
    return -n;                                  //METTRE UN - SINON TERRAIN NOIR
}

void ViewerEtudiant::init_terrain(Mesh& m_terrain, const Image& im)
{
    m_terrain = Mesh(GL_TRIANGLE_STRIP); // Choix primitive OpenGL
    for(int i=1;i<im.width()-2;++i)
    {
        for(int j=1;j<im.height()-1;++j)
        {
            m_terrain.texcoord((i+1)/float(im.width()),j/float(im.height()));
            m_terrain.normal( terrainNormal(im, i+1, j) );
            m_terrain.vertex( Point(i+1, 25.f*im(i+1, j).r, j) );

            m_terrain.texcoord(i/float(im.width()),j/float(im.height()));
            m_terrain.normal( terrainNormal(im, i, j) );
            m_terrain.vertex( Point(i, 25.f*im(i, j).r, j) );
        }
        m_terrain.restart_strip(); // Affichage en triangle_strip par bande
    }
}
```

```
void ViewerEtudiant::draw_terrain(const Transform &T, GLuint t_terrain)
{
    gl.texture(t_terrain);
    gl.model(T);
    gl.draw(m_terrain);
}
```

Arbres/Cactus

Pour les cactus, j'en ai généré 70 avec des positions aléatoire sur la map mais avec une hauteur qui est bien en adéquation avec la carte des hauteur pour que tout les cactus soient bien sur le sol.
Chaque cactus est un assemblage de 9 images de cactus entrecroisées pour donner une impression de 3D.

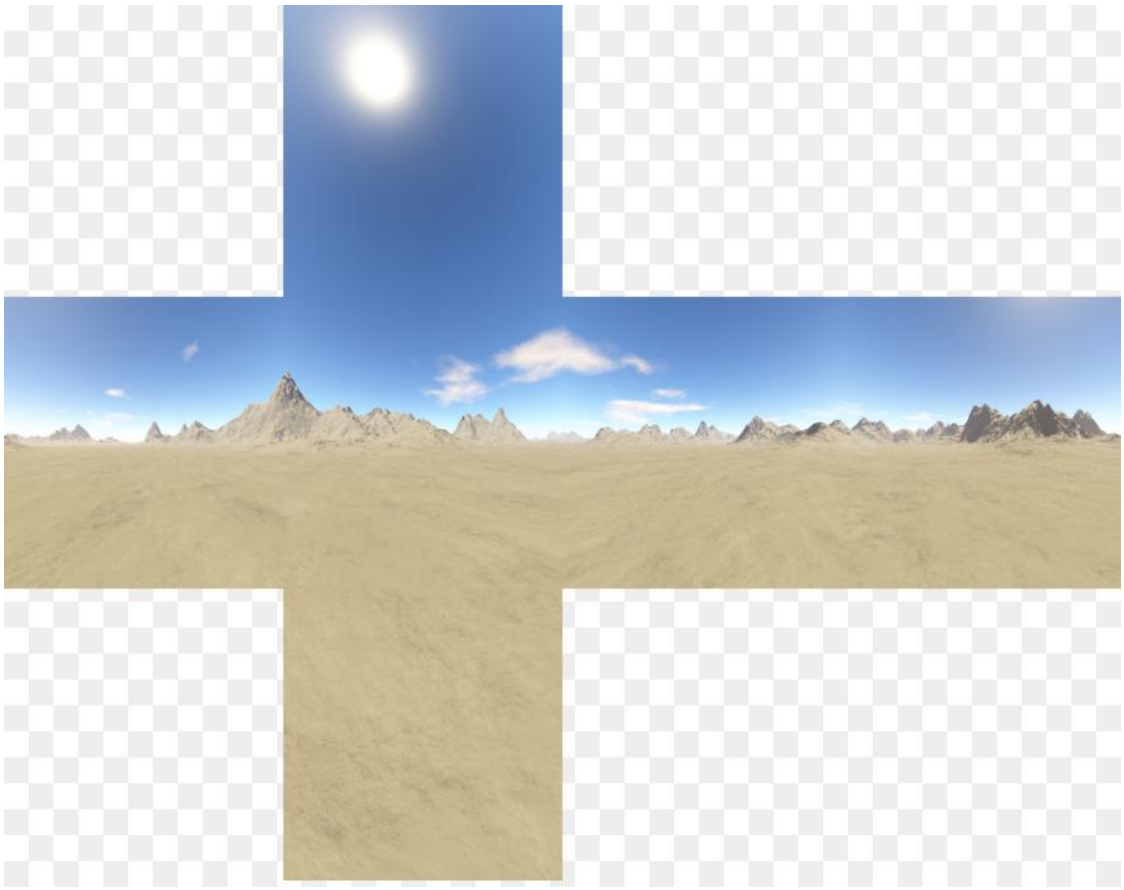


```
void ViewerEtudiant::init_tree()
{
    m_tree = Mesh(GL_TRIANGLE_STRIP);
    m_tree.color( Color(1, 1, 1));

    m_tree.normal(  0, 0, 1 );

    m_tree.texcoord(0,0 );                  //les 4 angles
    m_tree.vertex(-1, -1, 0 );
    m_tree.texcoord(1,0);
    m_tree.vertex(  1, -1, 0 );
    m_tree.texcoord(0,1);
    m_tree.vertex( -1, 1, 0 );
    m_tree.texcoord( 1,1);
    m_tree.vertex(  1,  1, 0 );
}

void ViewerEtudiant::init_alltree(const Image& im)
{
    for(int i=0; i < 70;i++)
    {
        int x = rand()%im.width();
        int z = rand()%im.height();
        float hauteur = im(x, z).r * 25.0 + 1;
        tabtree[i] = Point(x, hauteur, z);
    }
}
```

```
void ViewerEtudiant::draw_tree(Transform &T, GLuint t_tree)
{
    for(int i=0; i<9;i++)
    {
        T = T * Rotation(Vector(0,1,0),40);
        gl.alpha_texture(t_tree, 0.5);
        gl.model(T);
        gl.draw(m_tree);
    }
}

void ViewerEtudiant::draw_alltree(Transform &T, GLuint tree)    ////////DANS UN TAB PUIS CETTE FONCTION FAIS T=T*tab[i][j]
{
    for(int i=0; i<70;i++)
    {
        Transform Ttree= T * Translation(tabtree[i].x, tabtree[i].y, tabtree[i].z);
        gl.model(Ttree);
        draw_tree(Ttree, tree);
    }
}
```

Cubemap

Le fond est juste un grand cube avec la texture à l'intérieur.
Pour récupérer les bon morceaux de la texture sur le patron en croix j'ai créé un tableau de tableau de tableau [6][4][2]. Pour les 6 faces du cube on a les 4 angles avec chacun 2 coordonnées.



```
void ViewerEtudiant ::init_cubemap()
{
    //       4---5
    //      /|  /|
    //     7---6 |
    //     | 0-|-1
    //     |/  |/
    //     3---2


    // Sommets                    0          1          2       3          4          5       6          7
    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} };
    // Faces                      0          1          2          3          4          5
    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} };
    // Normales
    static float n[6][3] = { {0,-1,0}, {0,1,0}, {1,0,0}, {-1,0,0}, {0,0,1}, {0,0,-1} };

    int i;
    m_cubemap = Mesh(GL_TRIANGLE_STRIP);
    m_cubemap.color( Color(1, 1, 1) );

    //tab coordonne          bas gauche    bas droit      haut gauche    haut droit
    float tab[6][4][2] = { {{1.0/4.0,0},{1.0/4.0,1.0/2.0},{1.0/2.0,0},{1.0/2.0,1.0/3.0}}
                         , {{1.0/4.0,2.0/3.0},{1.0/4.0,1},{1.0/2.0,2.0/3.0},{1.0/2.0,1}}
                         , {{1.0/4.0,1.0/3.0},{1.0/4.0,2.0/3.0},{1.0/2.0,1.0/3.0},{1.0/2.0,2.0/3.0}}
                         , {{3.0/4.0,1.0/3.0},{3.0/4.0,2.0/3.0},{1,1.0/3.0},{1,2.0/3.0}}
                         , {{0,1.0/3.0},{0,2.0/3.0},{1.0/4.0,1.0/3.0},{1.0/4.0,2.0/3.0}}
                         , {{1.0/2.0,1.0/3.0},{1.0/2.0,2.0/3.0},{3.0/4.0,1.0/3.0},{3.0/4.0,2.0/3.0}} };
    // Parcours des 6 faces
    for (i=0;i<6;i++)
    {
        m_cubemap.normal(n[i][0], n[i][1], n[i][2]);

        m_cubemap.texcoord(tab[i][0][0],tab[i][0][1]);                          //recupere la bonne partie de la texture en bas a gauche
        m_cubemap.vertex( pt[ f[i][0] ][0], pt[ f[i][0] ][1], pt[ f[i][0] ][2] );

        m_cubemap.texcoord(tab[i][1][0],tab[i][1][1]);                        //recupere la bonne partie de la texture en bas a droite
        m_cubemap.vertex( pt[ f[i][3] ][0], pt[ f[i][3] ][1], pt[ f[i][3] ][2] );

        m_cubemap.texcoord(tab[i][2][0],tab[i][2][1]);                        //recupere la bonne partie de la texture en haut a gauche
        m_cubemap.vertex( pt[ f[i][1] ][0], pt[ f[i][1] ][1], pt[ f[i][1] ][2] );


        m_cubemap.texcoord(tab[i][3][0],tab[i][3][1]);                    //recupere la bonne partie de la texture en haut a droite
        m_cubemap.vertex( pt[ f[i][2] ][0], pt[ f[i][2] ][1], pt[ f[i][2] ][2] );

        m_cubemap.restart_strip();
    }
}
```

```
void ViewerEtudiant::draw_cubemap(Transform &T, GLuint cubemap)
{
    Transform Tcubemap = T * Scale(100) * Translation(0,0.4,0);
    gl.alpha_texture(t_cubemap, 1);
    gl.model(Tcubemap);
    gl.draw(m_cubemap);
}
```