

Textures

Sébastien Beugnon

24 novembre 2022

Sébastien Beugnon

R&D Researcher

mail : sebastien.beugnon@emersya.com

Github : @sbeugnon



Sommaire

Rappels

VBO

Shader

Texture Mapping

Problématique

OpenGL

Filtrage

Bouclage

Bump Mapping

Environment Mapping

TP

OpenGL : Vertex Buffer Object

► Création (Attributs de sommet)

```
GLuint vboId;  
float* vertices = new float[vCount*3]; // create vertex array  
GLuint dataSize = sizeof(float) * vCount * 3;  
glGenBuffers(1, &vboId);  
glBindBuffer(GL_ARRAY_BUFFER, vboId);  
glBufferData(GL_ARRAY_BUFFER, dataSize, vertices, GL_STATIC_DRAW);  
  
// it is safe to delete after copying data to VBO  
delete[] vertices;
```

► Création (Indices)

```
GLuint vboId2;  
int* indices = new int[vCount*3]; // create vertex array  
// ... populates array  
GLuint dataSize = sizeof(int) * vCount * 3;  
glGenBuffers(1, &vboId2);  
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, vboId2);  
glBufferData(GL_ELEMENT_ARRAY_BUFFER, dataSize, indices, GL_STATIC_DRAW);
```

OpenGL : Vertex Buffer Object

► Utilisation

```
// bind VBOs for vertex array and index array
glBindBuffer(GL_ARRAY_BUFFER, vboId1);           // for vertex coordinates
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, vboId2);    // for indices
// OpenGL 2.0 with custom shader
glEnableVertexAttribArray(attribVertex);          // activate vertex position
//      array (explanation of attribVertex later)
// set vertex arrays with generic API
glVertexAttribPointer(attribVertex, 3, GL_FLOAT, false, 0, 0);
// draw 6 faces using offset of index array
glDrawElements(GL_TRIANGLES, 36, GL_UNSIGNED_INT, 0);
// bind with 0, so, switch back to normal pointer operation
glBindBuffer(GL_ARRAY_BUFFER, 0);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, 0);
```

Exemple de shader GLSL

Vertex shader

```
// Vertex attributes
attribute vec3 a_position;
attribute vec3 a_normal; // (unused)
// in vec3 a_position;
// in vec3 a_normal;
// Uniforms
// Matrices, Material, etc
uniform mat4 u_projection, u_modelview;
// Textures
uniform sampler2D texture1;
// Interpolated values to transmit to fragment
  shaders
varying vec3 v_worldVertexPosition;
// Main program
void main(){
    vec4 vertPos4 = u_modelview * vec4(a_position,
        1.0);
    v_worldVertexPosition = vec3(vertPos4) /
        vertPos4.w;
//    vec3 rgb = texture(ourTexture, vertPos4.xy)
//    ;
    gl_Position = u_Projection * vertPos4;
}
```

Fragment shader

```
precision mediump float;
// Recover interpolated values
varying vec3 v_worldVertexPosition;
// Uniforms
uniform float u_intensity;

void main() {
    gl_FragColor = u_Intensity * vec4(
        v_worldVertexPosition, 1.0);
}
```

OpenGL : Shader

► Création

```
GLuint fragmentshader = glCreateShader(GL_FRAGMENT_SHADER);

glShaderSource(fragmentshader, 1, (const GLchar**)&fragmentsource, 0);

glCompileShader(fragmentshader);

glGetShaderiv(fragmentshader, GL_COMPILE_STATUS, &IsCompiled_FS);
if (IsCompiled_FS == FALSE) {
    glGetShaderiv(fragmentshader, GL_INFO_LOG_LENGTH, &maxLength);

    /* The maxLength includes the NULL character */
    fragmentInfoLog = (char *)malloc(maxLength);

    glGetShaderInfoLog(fragmentshader, maxLength, &maxLength, fragmentInfoLog);

    /* Handle the error in an appropriate way such as displaying a message or
       writing to a log file. */
    /* In this simple program, we'll just leave */
    free(fragmentInfoLog);
    return;
}

/* If we reached this point it means the vertex and fragment shaders compiled and
   are syntax error free. */
```

OpenGL : Shader Program

► Création

```
// Create a shader program
GLuint shaderprogram = glCreateProgram();
/* Attach our shaders to our program */
glAttachShader(shaderprogram, vertexshader);
glAttachShader(shaderprogram, fragmentshader);
// Link programs (should show error of shaders)
glLinkProgram(shaderprogram);
// Show link status
glGetProgramiv(shaderprogram, GL_LINK_STATUS, (int *)&IsLinked);
if (IsLinked == FALSE) {
    GLint maxLength = 0;
    glGetProgramiv(shaderprogram, GL_INFO_LOG_LENGTH, &maxLength);
    char * spInfoLog = new char[maxLength];
    glGetProgramInfoLog(shaderprogram, maxLength, &maxLength, spInfoLog);
    delete[] shaderProgramInfoLog;
    return;
}
// Now those shaders can be deleted after linking
glDeleteShader(vertexshader);
glDeleteShader(fragmentshader);
```

OpenGL : Shader Program

► Utilisation

```
// Usage of program
glUseProgram(shaderprogram);
// .. use program to render something
glUseProgram(0); // Reset to classic opengl
```

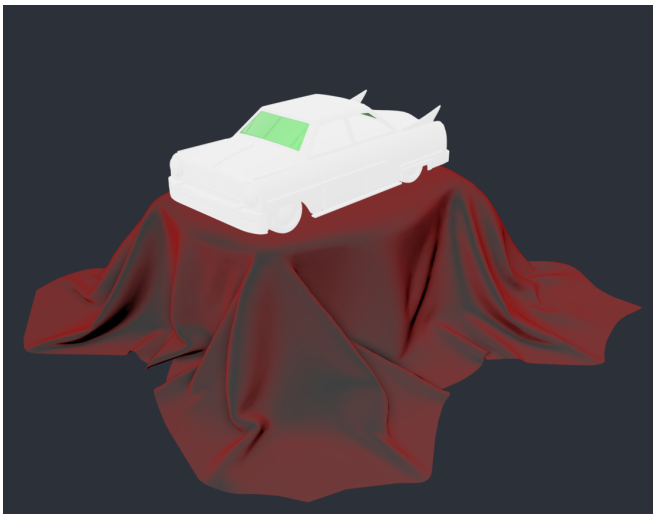
► Attributes

```
// get location of attribute "a_Position in shader program"
GLuint vertexAttrib = glGetAttribLocation(shaderProgram, "a_position");
if (vertexAttrib != -1) {
    glBindBuffer(GL_ARRAY_BUFFER, &vboId1);
    // activate vertex position array
    glEnableVertexAttribArray(vertexAttrib);
    // bind attribute with vbo
    glVertexAttribPointer(vertexAttrib, 3, GL_FLOAT, false, 0, 0);
}
```

► Uniformes

```
// Get location of attribute "a_Position in shader program"
GLuint modelViewLocation = glGetUniformLocation(shaderProgram, "u_modelView");
if (modelViewLocation != -1) {
    // ... float modelViewMatrix[16];
    glUniformMatrix4fv(modelViewLocation, 1, GL_FALSE, modelViewMatrix);
}
glUniform1f(glGetUniformLocation(shaderProgram, "u_intensity"), 1.0f);
```


Problématique



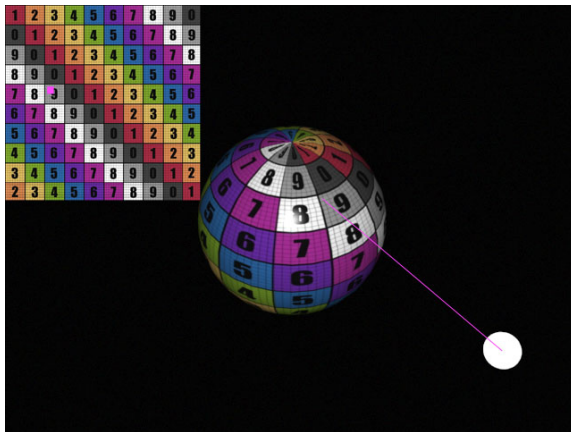
Problématique



Placage de texture

Définition

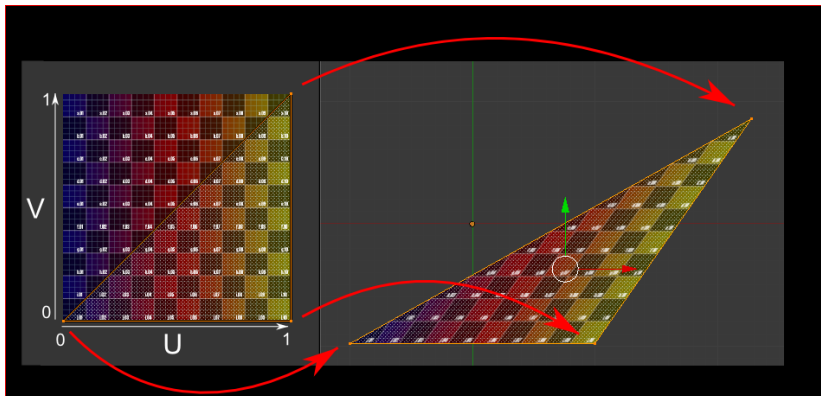
Application d'une image 1D, 2D, 3D sur des primitives géométriques
Un pixel d'une texture est appelé *texel*



Coordonnées de texture

Nouvel attribut de sommet 3D

- ▶ Texture 2D : UV (ou ST)
- ▶ Texture 3D : UVW (ou STR)



Création d'une texture

- ▶ Assignment des coordonnées de texture aux points 3D
- ▶ Spécification d'une texture
 - ▶ Créer une unité de texture en GPU
 - ▶ Charger une image sur le GPU
 - ▶ Activer le plaquage
 - ▶ Paramétrer la texture (wrapping, filtering)

Création d'une texture

- ▶ Assignment des coordonnées de texture aux points 3D
- ▶ Spécification d'une texture
 - ▶ Créer une unité de texture en GPU
 - ▶ Charger une image sur le GPU
 - ▶ Activer le plaquage
 - ▶ Paramétrer la texture (wrapping, filtering)

Règles à savoir (en OpenGL)

- ▶ Les images en puissance de 2 (Po2) sont à privilégier
- ▶ Les dimensions des textures sont limitées (par le hardware)
- ▶ Le nombre de textures actives en parallèle est limité par OpenGL, le hardware ou le moteur

OpenGL : Assignment des coordonnées de texture

► Legacy OpenGL

```
// Create triangle
glBegin(GL_TRIANGLES);
glTexCoord2f(0.0f,0.0f);
glVertex3f(4.0f, 5.0f, 0.0f);
glTexCoord2f(1.0f,0.0f);
glVertex3f(10.0f, 5.0f, 0.0f);
glTexCoord2f(0.0f,1.0f);
glVertex3f(4.0f, 12.0f, 0.0f);
glEnd();
```

Ou bien

```
// Create buffer
float vertices = new float[vCount*3];
float* uvs = new float[vCount*2]; // create vertex array (set values)
glEnableClientState(GL_VERTEX_ARRAY); // activate texture coord array
glEnableClientState(GL_TEXTURE_COORD_ARRAY); // activate texture coord array
// ...
glTexCoordPointer(2/*numberOfCoordinates*/, GL_FLOAT, 0/*stride*/, uvs);
glVertexPointer(3, GL_FLOAT, 0, glVertexPointer);
glDrawArrays(GL_TRIANGLES, 0, vCount);
glDisableClientState(GL_VERTEX_ARRAY);
glDisableClientState(GL_TEXTURE_COORD_ARRAY_EXT);
```

OpenGL : Assignment des coordonnées de texture

► Modern OpenGL

```
// Create data
float* uvs = new float[vCount*2]; // create uv array

// Create VBO
GLuint dataSize = sizeof(float) * vCount * 2;
glGenBuffers(1, &vboId);
glBindBuffer(GL_ARRAY_BUFFER, vboId);
glBufferData(GL_ARRAY_BUFFER, dataSize, uvs, GL_STATIC_DRAW);
delete[] uvs;
// ...
// Use VBO as shader attribute
GLuint shaderProgram; // shader index
GLuint vertexAttrib = glGetAttribLocation(shaderProgram, "aUV");// cache this!
glBindBuffer(GL_ARRAY_BUFFER, vboId);
glVertexAttribPointer(vertexAttrib, 2, GL_FLOAT, GL_FALSE, 0, 0);
```


OpenGL : Spécification d'une texture

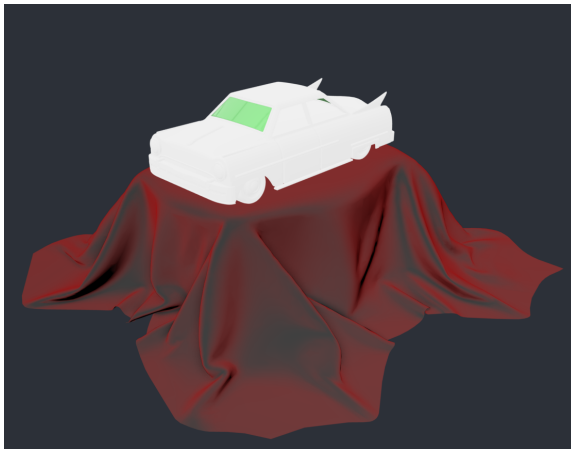
► Création & Chargement

```
unsigned int texture;
glGenTextures(1, &texture);
glBindTexture(GL_TEXTURE_2D, texture);
// load and generate the texture
int width, height, nrChannels;
unsigned char *data = stbi_load("container.jpg", &width, &height, &nrChannels, 0)
;
if (data) {
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB,
        GL_UNSIGNED_BYTE, data);
    glGenerateMipmap(GL_TEXTURE_2D);
} else {
    std::cout << "Failed to load texture" << std::endl;
}
stbi_image_free(data);
```

► Suppression

```
glDeleteTexture(1, &texture);
```

Votre réaction Je vois rien



Rien ne s'affiche ! Car nous n'avons pas **paramétriser** notre texture.

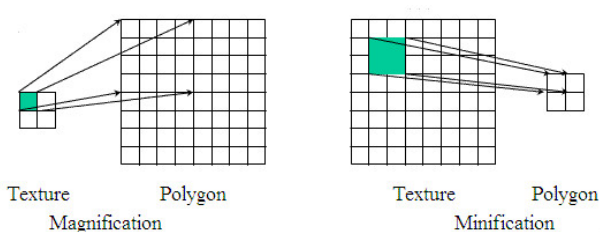
Paramétrisation d'une texture

- ▶ Mode de filtrage (Filtering)
 - ▶ Réduction, agrandissement
 - ▶ Mip-mapping
- ▶ Mode de bouclage (Wrapping)
 - ▶ Répéter, tronquer
- ▶ Fonctions de textures (Blending)
 - ▶ Only for good ol' OpenGL

Filtrage

Réduction/Agrandissement (Minify/Magnify)

Les textures et les objets texturés ont rarement la même taille (en pixel)



- ▶ `GL_TEXTURE_MAG_FILTER`
- ▶ `GL_TEXTURE_MIN_FILTER`

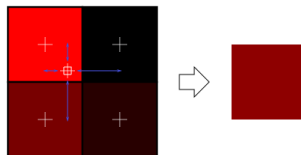
OpenGL : Paramétrisation du filtrage

► OpenGL

```
glBindTexture(GL_TEXTURE_2D, texture);  
glTexParameteri(target, type, mode);  
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);  
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
```



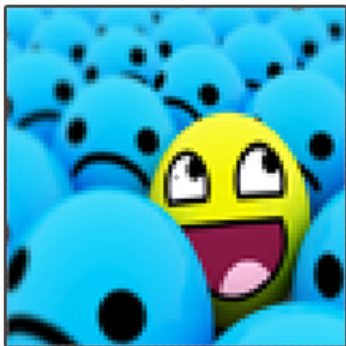
Au plus proche



Interpolation linéaire

- Il existe également le filtrage anisotrope

Filtrage : Résultats



GL_NEAREST



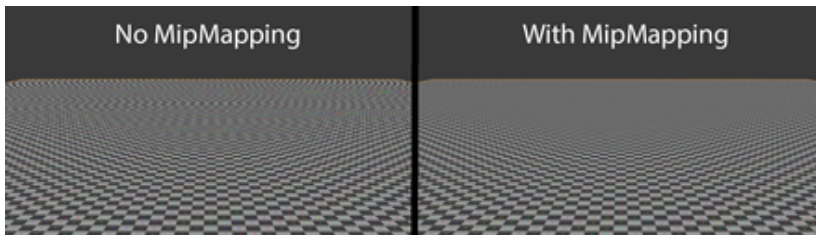
GL_LINEAR

(From learnopengl.com)

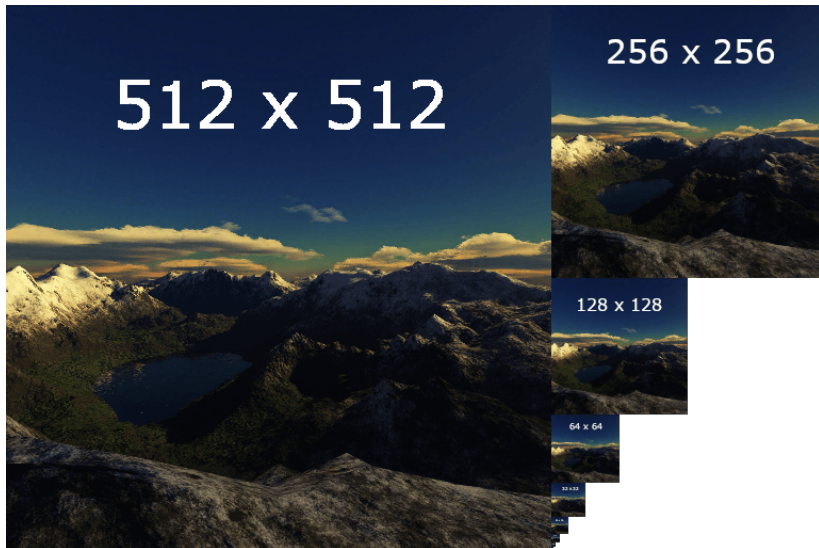
Filtrage

Mip-mapping

- ▶ Technique consistant à précalculer plusieurs versions réduite d'une même texture
- ▶ Adapte le niveau de détails en fonction de la distance



OpenGL : Mip-Mapping



OpenGL : Mip-Mapping

► Manuelle

```
glBindTexture(GL_TEXTURE_2D, texture);  
glTexImage2D(GL_TEXTURE_2D, 0/* level */,  
GL_RGB /* components*/, width, height, border, format, type, imgBase);  
glTexImage2D(GL_TEXTURE_2D, 1/* level */,  
GL_RGB /* components*/, width/2, height/2, border, format, type, imgDividedBy2);  
glTexImage2D(GL_TEXTURE_2D, 2/* level */,  
GL_RGB /* components*/, width/4, height/4, border, format, type, imgDividedBy4);
```

► Automatique

```
// GLU dependencies  
glBindTexture(GL_TEXTURE_2D, texture);  
gluBuild2DMipmaps(GL_TEXTURE_2D, GL_RGB, width, height, format, type, imgBase);
```

► Utilisation du *Mip-Map*

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);  
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR_MIPMAP_LINEAR);
```

Mode de bouclage (Wrap)

Définition

Fournit un comportement lorsque les coordonnées de texture sortent de l'intervalle $[0, 1]$

- ▶ Répéter (Repeat)
- ▶ Tronquer (Clamp)

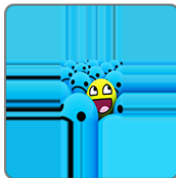
```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, mode);  
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, mode);
```



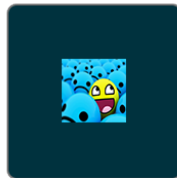
GL_REPEAT



GL_MIRRORED_REPEAT



GL_CLAMP_TO_EDGE



GL_CLAMP_TO_BORDER

OpenGL : Utilisation des textures dans les shaders

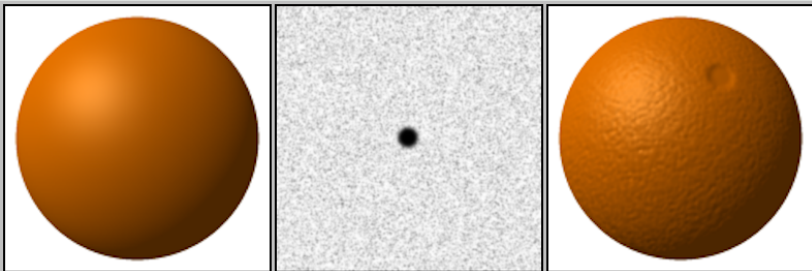
► Assigner une texture dans un shader

```
// activate usage of 2D textures
glEnable(GL_TEXTURE_2D);
// activate texture (limited to 32 or less depending on machine)
int indexActiveTexture = 0; // 0 to 31
// bind texture as Texture 0
glBindTexture(GL_TEXTURE_2D, texture);
glActiveTexture(GL_TEXTURE0 + indexActiveTexture++);
// set used active texture (Modern OpenGL)
glUniform1i(glGetUniformLocation(shaderProgram, "texture"), indexActiveTexture);
```

Bump Mapping

Définition

- ▶ Simule les détails dans la géométrie sans ajouter de données à cette dernière
- ▶ Interaction avec la lumière (rugosité)
- ▶ Perturbation de la normale pour le calcul des lumières

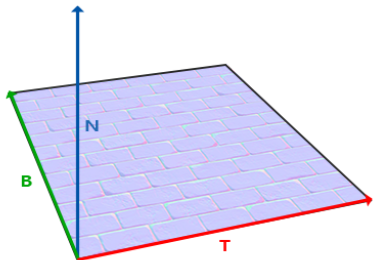


Bump Mapping : mathématiques

Théorie de la perturbation des normales

Modification de la position de la surface en ajoutant une petite perturbation (bump function) dans la direction de la normale :

$$P'(u, v) = P(u, v) + B(u, v) * N$$



- ▶ P point de la surface
- ▶ P_u tangente de P dans la direction u
- ▶ P_v tangente de P dans la direction v
- ▶ N Normal au point P
- ▶ $B(u, v)$ est un fonction

Théorie de la perturbation des normales

On ne modifie pas la position du sommet P , mais la normale interpolée par

► Pentes

► $d_u = \frac{\Delta z}{\Delta x} = \frac{s}{2}[h(i+1, j) - h(i-1, j)]$

► $d_v = \frac{\Delta z}{\Delta y} = \frac{s}{2}[h(i, j+1) - h(i, j-1)]$

où s est un facteur multiplicatif de la hauteur maximale en fonction de la taille d'un seul texel, $h(i, j)$ correspond aux texels de la texture.

► Calcul des directions dir_u , dir_v

► $dir_u = (1, 0, d_u)$ et $dir_v = (0, 1, d_v)$

► Calcul de la nouvelle normale

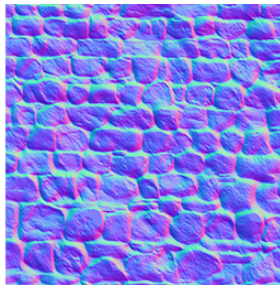
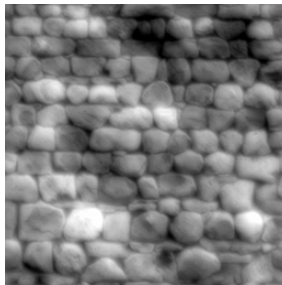
► $m = \text{normalize}(dir_u \times dir_v) = \frac{(-d_u, -d_v, 1)}{\sqrt{d_u^2 + d_v^2 + 1}}$

où m est une normale dans l'espace tangent.

Alternative : Normal Mapping

Nouvelle forme de bump mapping

- ▶ Les calculs sont stockés directement dans l'image (baking)
- ▶ Attention : stockée dans l'espace tangent et normalisée
- ▶ $RGB = 0.5 * m + 0.5$



Environment Mapping

Définition

Simulation de l'effet de réflexion d'un environnement sur un objet réfléchissant (brillant)

- ▶ Skybox, Panorama (Réflexion statique)
- ▶ La scène 3D (Réflexion dynamique)



Comment générer un environnement

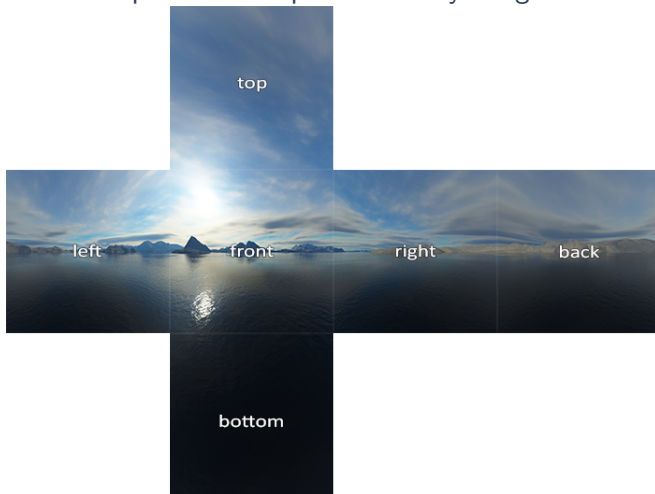
Différentes approches

- ▶ (pré-)Calculé (Google Map, 3D)
- ▶ Capturé (Photo, HDRI)



Skybox

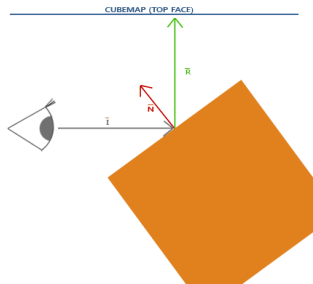
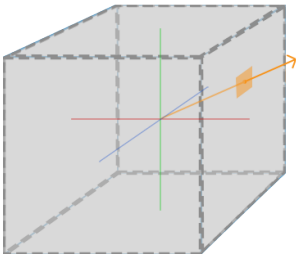
pre-rendered panoramic sky images



Skybox

Cubemap

- ▶ 6 faces utilisant 6 images
- ▶ Lancer de rayon



(from learnopengl.com)

OpenGL : Texture Cube

► Création et chargement (avec stb_image.h)

```
unsigned int loadCubemap(vector<std::string> facesPath) {
    unsigned int textureID;
    glGenTextures(1, &textureID);
    glBindTexture(GL_TEXTURE_CUBE_MAP, textureID);
    int width, height, nrChannels;
    for (unsigned int i = 0; i < faces.size(); i++) {
        unsigned char *data = stbi_load(faces[i].c_str(), &width, &height, &
            nrChannels, 0);
        if (data) {
            glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_X + i,
                0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE,
                data);
        } else {
            std::cout << "Cubemap tex failed: " << faces[i] << std::endl;
        }
        stbi_image_free(data);
    }
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_R, GL_CLAMP_TO_EDGE);
    return textureID;
}
```

OpenGL : Texture Cube dans le shader

► Vertex Shader

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;

out vec3 Normal;
out vec3 Position;

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

void main()
{
    mat3 normalMatrix = mat3(transpose(inverse(model)));
    Normal = normalMatrix * aNormal; // world normal
    Position = vec3(model * vec4(aPos, 1.0)); // world Position
    gl_Position = projection * view * vec4(Position, 1.0);
}
```

OpenGL : Texture Cube dans le shader

► Fragment shader

```
#version 330 core
out vec4 FragColor;

in vec3 Normal;
in vec3 Position;

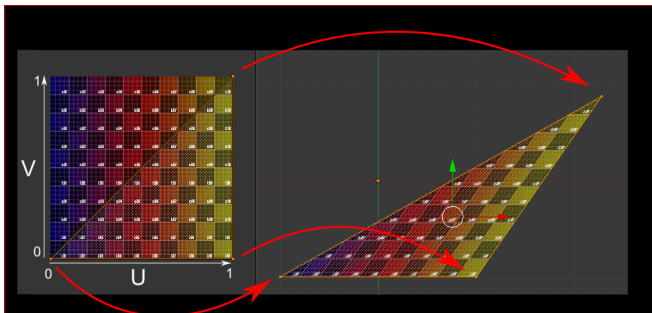
uniform vec3 cameraPos;
uniform samplerCube skybox;

void main()
{
    vec3 I = ...; // camera to position
    vec3 R = reflect(<vec3>, <vec3>);
    FragColor = vec4(texture(skybox, R).rgb, 1.0);
}
```

Travaux pratiques

Niveau 0 : Plaquage d'une texture 2D sur un plan 3D

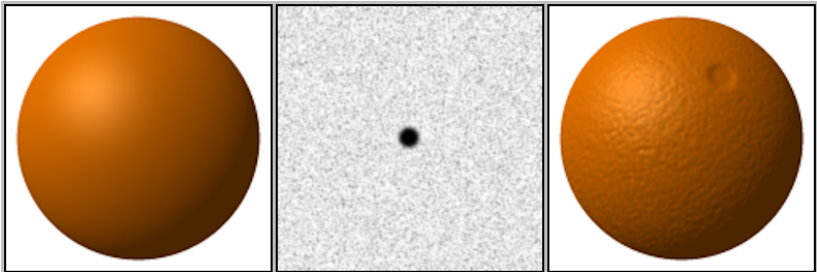
- ▶ Créer un plan 3D (avec des coordonnées 3D, UV et normales)
- ▶ Charger une texture 2D (stb_image.h)
- ▶ Plaquer la texture 2D sur le plan 3D
 - ▶ modifier le shader pour utiliser la couleur RGB de la texture



Travaux pratiques

Niveau 1 : Bump Mapping

- ▶ Charger une texture en niveau de gris
- ▶ Utiliser cette seconde texture pour modifier les normales de votre modèle
- ▶ Alternative : Normal Mapping



Travaux pratiques

Niveau 2 : Environnement Mapping

- ▶ Créer un cube 3D pour la skybox
- ▶ Charger une texture cube
- ▶ Créer un shader faisant le rendu de la sky
- ▶ Faire un premier rendu de la skybox
- ▶ Faire un rendu réfléchif d'un objet 3D

