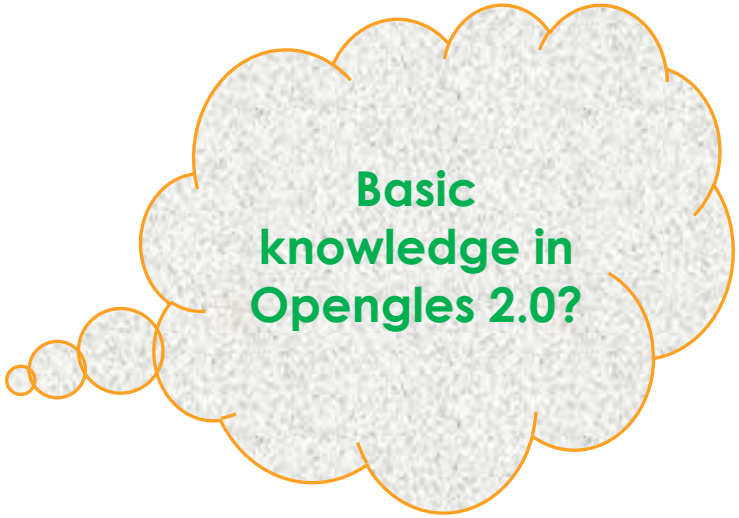
Gameloft
6/6/2011

3D Basic & OpenGLES 2.0

thuy.vuthiminh@gameloft.com
phong.caothai@gameloft.com
khiem.tranthien@gameloft.com
tam.la@gameloft.com

Content

- Introduction
- Rendering pipeline
- Shader
- Basic GLSL-ES
- Basic Math
- MVP matrices
- Textures
- Obj model
- Shader effect: Skydome using cube mapping

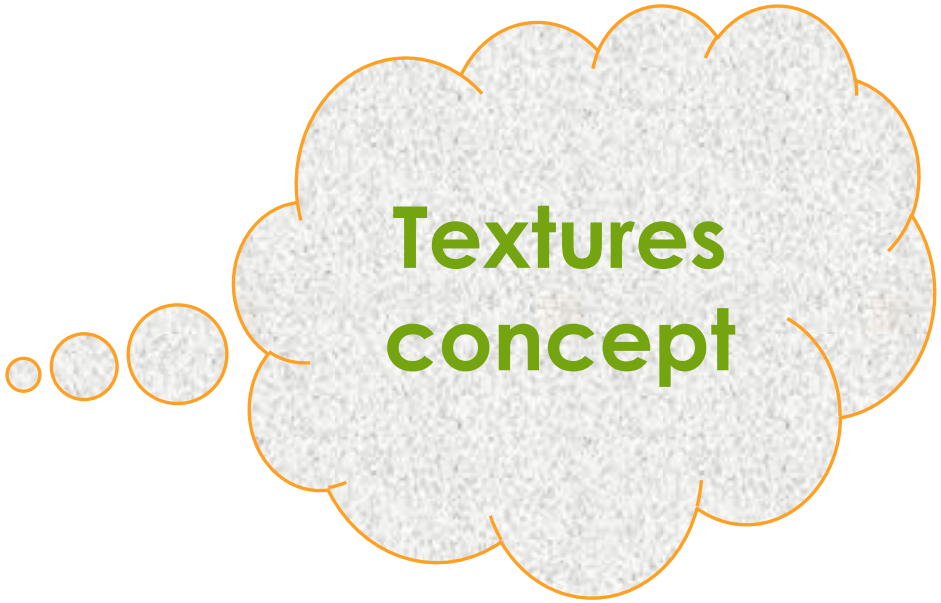


Content

- Introduction
- Rendering pipeline
- Shader
- Basic GLSL-ES
- Basic Math
- MVP matrices
- **Textures**
- Obj model
- Shader effect: Skydome using cube mapping

Basic Math

- What is Texture?
- Size of Texture
- Text coordinate and Texel
- Wrapping Modes
- Filters
- Mipmap
- Coding



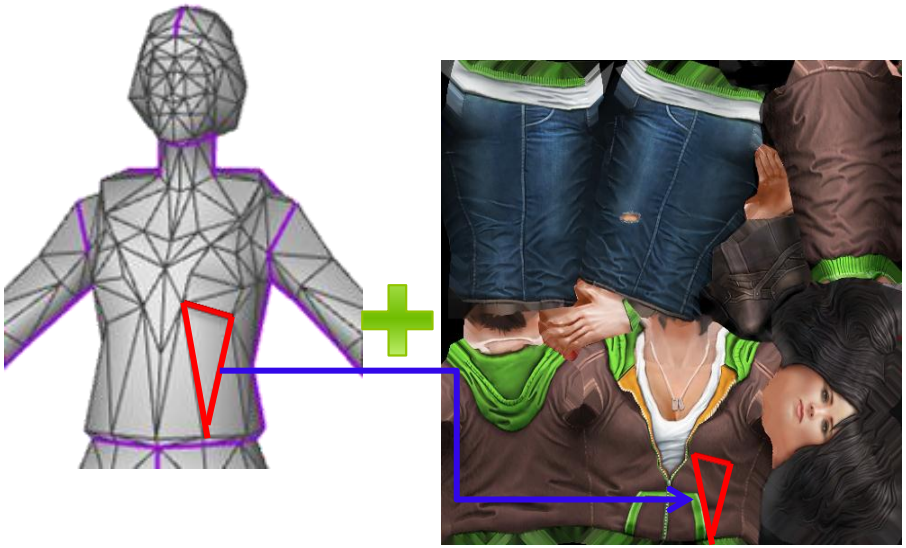
Textures

- Texture is 2D Image applied on a 3D Object.

Textures

- Each primitive on the 3D object will be map to a 2D Image

➔ **Texture Mapping**.

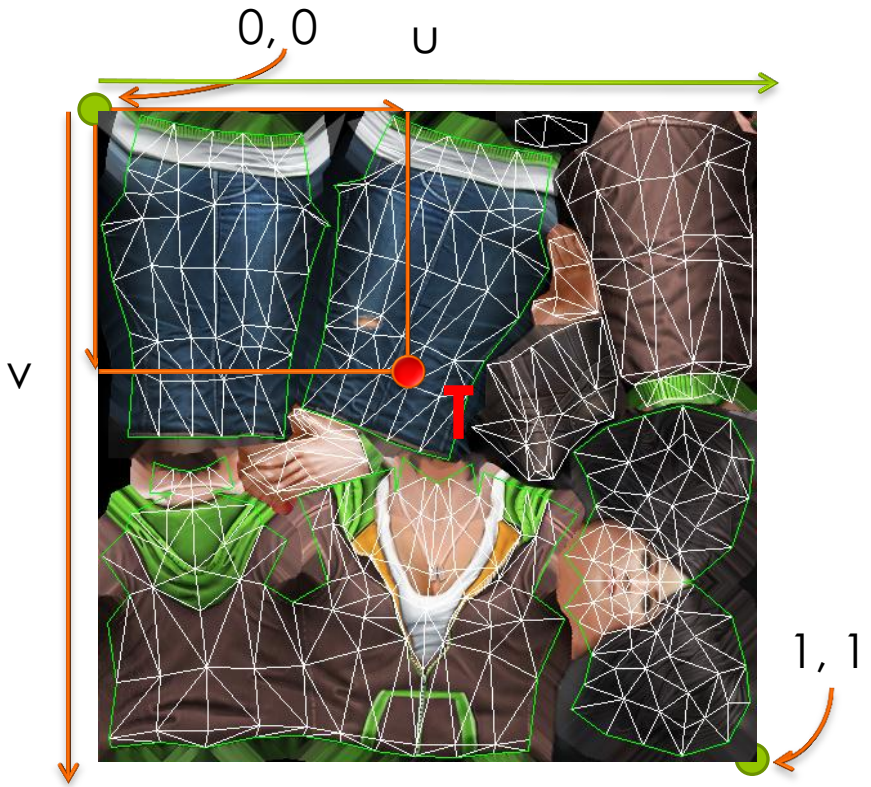


Size of Texture

- **Power of Two** (POT):
  - Size (width, height) of a texture must be a power-of-two number, that mean it should be 1, 2, 4, 8, 16, 32, 64,….
- **None POT texture:**
  - Some graphic cards support non-power-of-two textures.
  - For optimization and compatibility, we should use POT texture.

Text coordinate & Texel

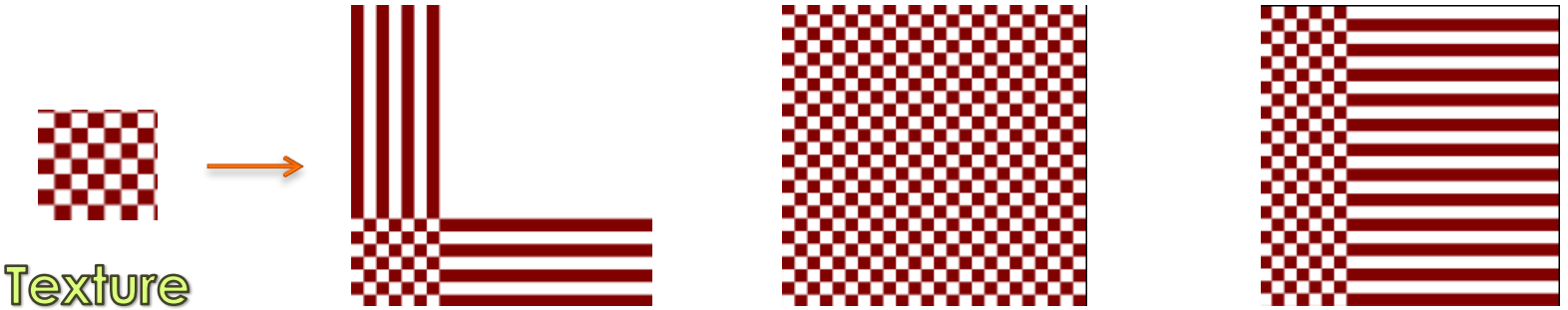
- UV or texture coordinate:
  - An attribute to describe the position of that vertex on the image
- Texel is a pixel on the texture.
- T is a texel:
  - The coordinate of T on the image
  - Defined by (u, v)
  - The u, v is in [0, 1] range



Wrapping Modes

- Process to receive color on the image is called sampling
- If pixel doesn't receive a color by sampling, filling or mirroring the image depends on setting.
- When UV is out of [0..1] range, to receive the color of that textel → options

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, wrap_s_option);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, wrap_t_option);
```

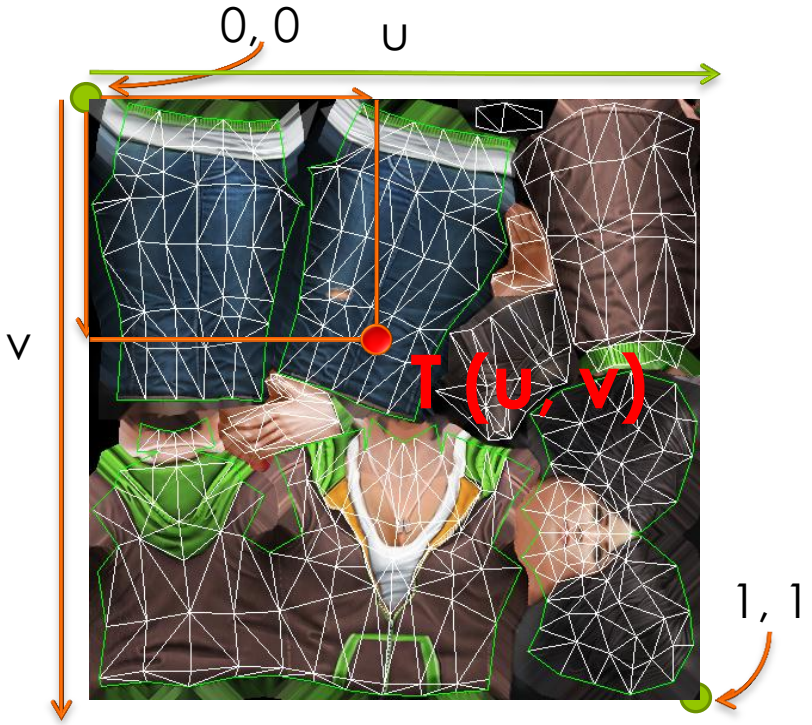


Filters

- u, v are float values
- it is not always stay on a pixel in the image

→ we need a way to received the color from pixels on the images ?

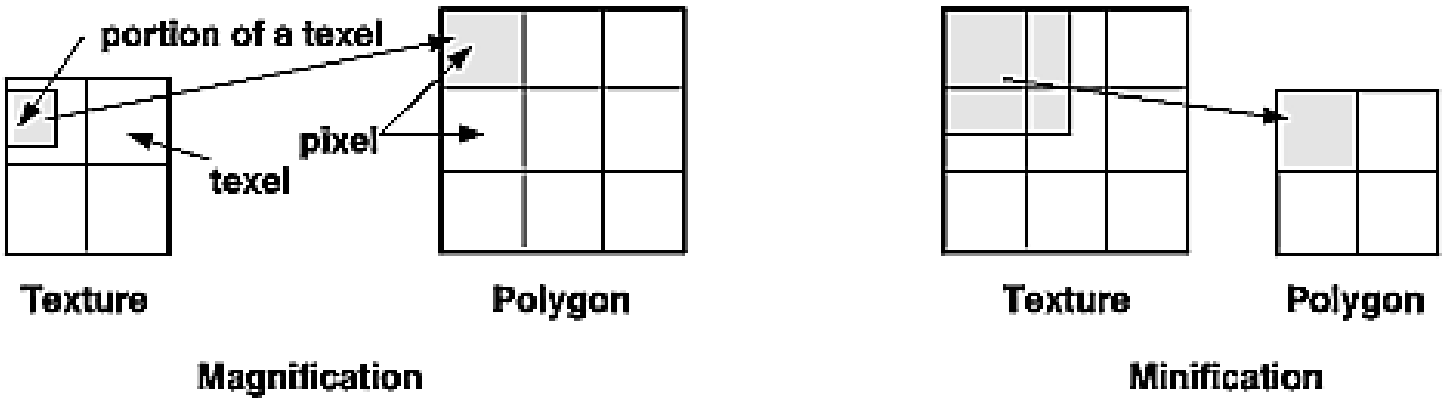
→ Filters will solve this problems.



Filters: Minification and Magnification

- Minification
  - ✓ More than one texel can cover a pixel
- Magnification
  - ✓ More than one pixel can cover a texel

➔ Like when you zoom in/out the image, there must be a method to make the result.



Filters

- Method to pick color from the texture with input u,v:
- To set filtering options we use:

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

- 6 options:
  - GL_NEAREST: Nearest neighbor
  - GL_NEAREST_MIPMAP_NEAREST: Nearest neighbor with mipmapping
  - GL_LINEAR: Bilinear
  - GL_NEAREST_MIPMAP_LINEAR
  - GL_LINEAR_MIPMAP_NEAREST
  - *GL_LINEAR_MIPMAP_LINEAR*

Filters

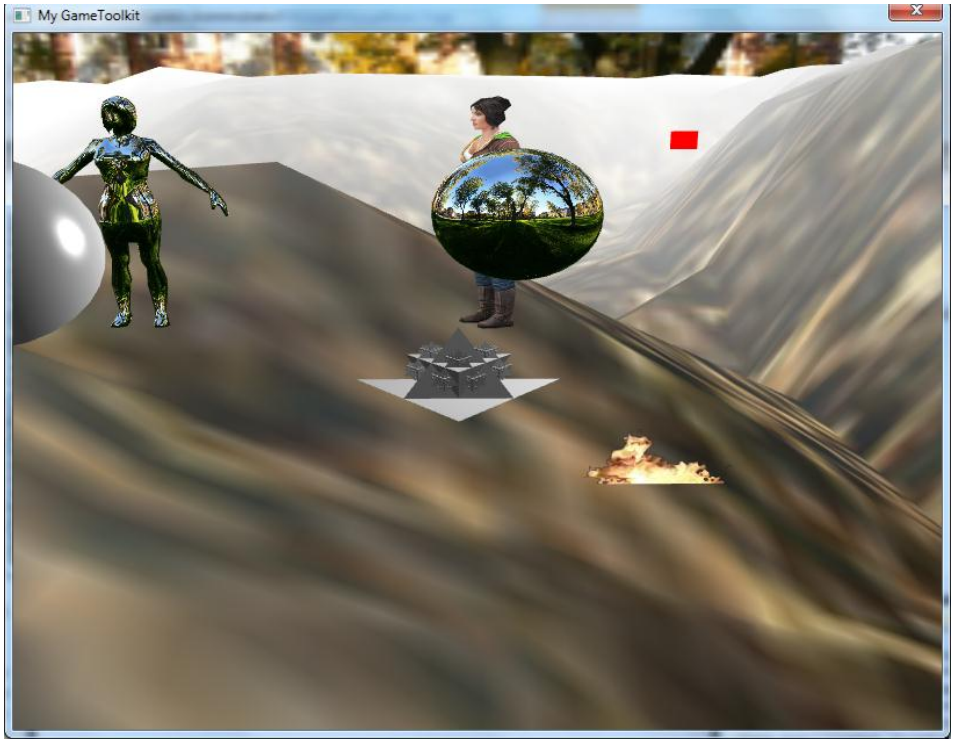
- **GL_NEAREST - Nearest neighbor:**
  Pick the color of the nearest pixel on the texture image.
  - ✓ *Fast method.*
  - ✓ *Generate a large amount of artifacts.*

Pixel becomes big blocks in this scene

- **GL_NEAREST_MIPMAP_NEAREST - Nearest neighbor with mipmapping**:
  Pick the color of the nearest pixel on the nearest mipmap:
  - ✓ *Same to nearest neighbor but use mipmap.*
  - ✓ *Solve the antialiasing problem but we will still have blocks on our scene.*

Filters (conts)

- **GL_LINEAR - Bilinear**: *Take four adjacent pixels to the texel to calculate the average color ➔ result color.*

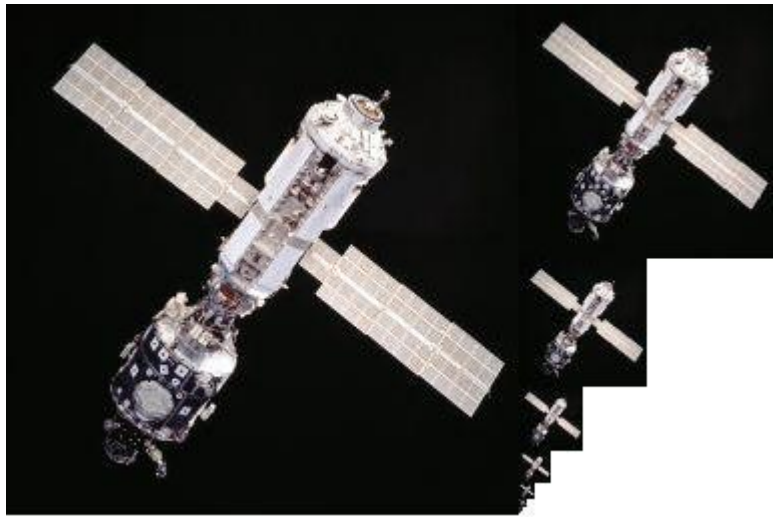


Filter (conts)

- **GL_NEAREST_MIPMAP_LINEAR**:
  - ✓ Take two adjacent texels on two nearest mipmaps by using GL_NEAREST, calculate the average value of the picked color → result color.
- **GL_LINEAR_MIPMAP_NEAREST**:
  - ✓ Chose the nearest mipmap and use GL_LINEAR operation.
- ***GL_LINEAR_MIPMAP_LINEAR:***
  - ✓ Chose the two nearnest mipmaps to the texcel and pick the two texel using GL_LINEAR → calculate the average of those two values to get the final color.

Mipmap

- Mipmaps are pre-calcuated, optimized collection of image of a main texture
- Advantages:
  - ✓ Increase rendering speed
  - ✓ Reduce anti-alias artifacts.
  - ✓ Lessens interpolation
- Disadvantages:
  - ✓ Use much memory



Coding: How to use texture?

1. **Generate the texture:**

```
GLuint textureID;
glGenTextures(1, &textureID);
```

➔ A texture will be generated inside GPU.

2. **Bind and load Texture data.**

```
glBindTexture(GL_TEXTURE_2D, textureID);
glTexImage2D(GL_TEXTURE_2D, 0, RGBA, iWidth, iHeight, 0,
RGBA, GL_UNSIGNED_BYTE, imageData);
```

➔ imageData is the image in RGBA format (4 byte / pixel).

Coding (conts)

3. **Setting texture parameters:**

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

- For un-mipmap texture

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

- For mipmap texture

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                                        GL_NEAREST_MIPMAP_NEAREST );
// no GL_TEXTURE_MAG_FILTER with mipmap
glGenerateMipmap(GL_TEXTURE_2D);
```

We will have a texture inside the GPU for later use.

Coding (conts)

4. **Writing the shader:**

+ Add uv attribute.

```
attribute vec2 a_uv;
```

+ Add uv varying to pass it to the fragment shaders:

```
varying vec2 v_uv;

....

v_uv = a_uv;
```

+ Add uniform of the texture as a sampler2D.

```
uniform sampler2D u_texture;
```

+ Read the color value:

```
gl_FragColor = texture2D(u_texture, v_uv);
```

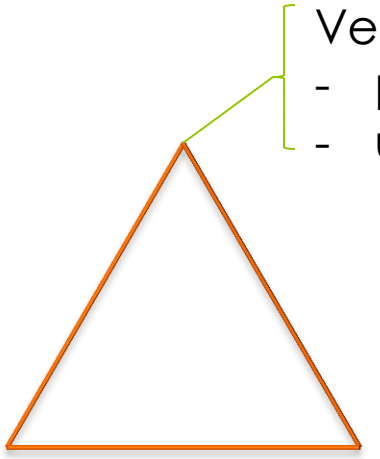
```
attribute vec4 a_position;
attribute vec2 a_uv;

varying vec2 v_uv;
void main()
{
    gl_Position = a_position;
    v_uv = a_uv;
}
```

```
uniform sampler2D u_texture;

varying vec2 v_uv;
void main()
{
    gl_FragColor = texture2D(u_texture,
v_uv);
}
```

Coding (conts)

5. **Define the Object and its attributes:**

```
float vertices_pos[] = { … };

float texcoords_pos[] = {…};
```



Vertex attributes:
- position.
- uv.

Coding (conts)

6. **Setting the texture uniform:**

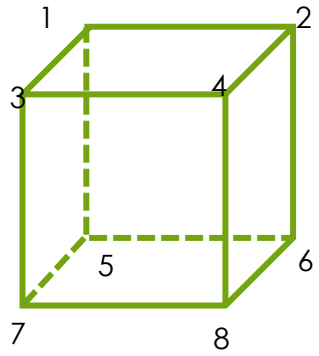
```
glBindTexture(GL_TEXTURE_2D, textureID);
glUniform1i(userData→u_texture, 0);
```

7. **Command the GPU to draw the object:**

```
glVertexAttribPointer(iVertexLoc, 3, GL_FLOAT, GL_FALSE, 0, vertices_pos);
glEnableVertexAttribArray(iVertexLoc);
glVertexAttribPointer(iTexcoordLoc, 2, GL_FLOAT, GL_FALSE, 0, texcoords_pos);
glEnableVertexAttribArray(iTexcoordLoc);
glDrawArrays(GL_TRIANGLES, 0, 3);
```

Texture: Practice

- Suppose that I have below cube and picture



- Write a code to map texuture on this cube
- Result will see in the right picture

Content

- Introduction
- Rendering pipeline
- Shader
- Basic GLSL-ES
- Basic Math
- MVP matrices
- Textures
- **Obj model**
- Shader effect: Skydome using cube mapping

Model

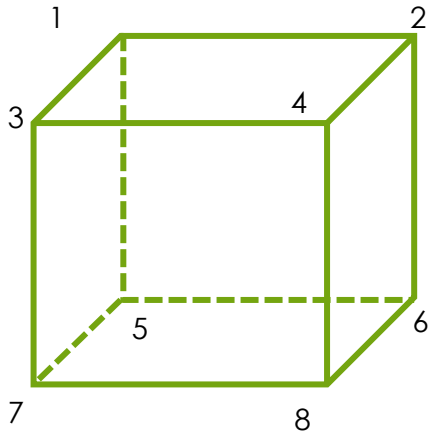
- 3D models/geometries are typically comprised of
  - Polygons
  - Verticies
  - Textures
  - Normal

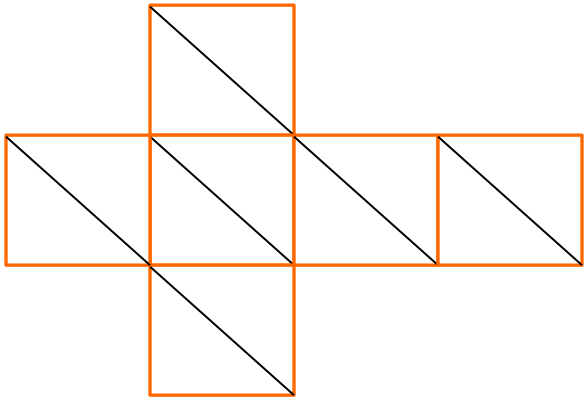
  which create the model's shape.
- Normally, using file format with *obj, md2, md5,…* file extension

Obj File Format

Ex: If we want to draw a cube

- 8 vertices
- 6 planes, 2 triangles/plane
- 12 triangles ↔12 faces





Obj File Format

- v: vertex (x, y, z)

  Ex: A(-0.4, 0.0, 0.4)

- vn: normal vector (x, y, z)

  Ex: $\vec{v}$ (0.0, -1.0, -0.0)

- vt: texture coordination (u, v, p)

  Ex: (1.0, 0.0)

- f: face, in order

  vertex / textcoord / normal

  Ex:
  f (-0.4, 0.0, 0.4 / 1.0, 0.0, 0.0 / 0.0, -1.0, -0.0
  -0.4, 0.0, -0.4 / 1.0, 1.0, 0.0 / 0.0, -1.0, -0.0
  0.4, 0.0, -0.4 / 0.0, 1.0, 0.0 / 0.0, -1.0, -0.0)

```
#
# object Box01
#
v  -0.40    0.00    0.40
v  -0.40    0.00    -0.40
v  0.40     0.00    -0.40
v  0.40     0.00     0.40
v  -0.40    0.40     0.40
v  0.40     0.40     0.40
…
# 8 vertices

vn 0.00    -1.00    -0.00
vn 0.00     1.00    -0.00
vn 0.00     0.00     1.00
…
# 6 vertex normals

vt 1.00     0.00     0.00
vt 1.00     1.00     0.00
vt 0.00     1.00     0.00
…
# 4 texture coords

g Box01
usemtl 01___Default
f 1/1/1 2/2/1 3/3/1
f 3/3/1 4/4/1 1/1/1
f 5/4/2 6/1/2 7/2/2
…
# 12 faces
```

Coding

- Read from *.obj below info:

vertices_index[num_vertices x 3]

→ return the array contains all vertices

Ex: vertices_index[24] = {

-0.4, 0.0, 0.4, //vertex 1

-0.4, 0.0, -0.4, //vertex 2

0.4, 0.0, -0.4,…} //vertex 8

normal_index[num_normals x 3]

→ return the array contains all normal

Ex: normals_index[18] = {

1.0, 0.0, 0.0, //normal1

1.0, 1.0, 0.0, //normal 2

0.0, 1.0, 0.0,…} //normal 6

```
#
# object Box01
#
v  -0.40    0.00    0.40
v  -0.40    0.00   -0.40
v   0.40    0.00   -0.40
v   0.40    0.00    0.40
v  -0.40    0.40    0.40
v   0.40    0.40    0.40
…
# 8 vertices

vn 0.00   -1.00   -0.00
vn 0.00    1.00   -0.00
vn 0.00    0.00    1.00
…
# 6 vertex normals

vt 1.00    0.00    0.00
vt 1.00    1.00    0.00
vt 0.00    1.00    0.00
…
# 4 texture coords

g Box01
usemtl 01___Default
f 1/1/1 2/2/1 3/3/1
f 3/3/1 4/4/1 1/1/1
f 5/4/2 6/1/2 7/2/2
…
# 12 faces
```

Coding

`texcoords_index[num_textcoord x 3]`

→ return the array contains all textcoord

Ex: texcoords_index[24] = {

1.0, 0.0, 0.0, //textcoord 1

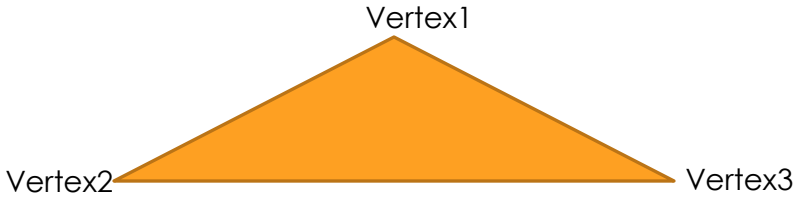
1.0, 1.0, 0.0, // textcoord 2

0.0, 1.0, 0.0,…} // textcoord 4

```
#
# object Box01
#
v  -0.40    0.00    0.40
v  -0.40    0.00   -0.40
v   0.40    0.00   -0.40
v   0.40    0.00    0.40
v  -0.40    0.40    0.40
v   0.40    0.40    0.40
…
# 8 vertices

vn 0.00   -1.00   -0.00
vn 0.00    1.00   -0.00
vn 0.00    0.00    1.00
…
# 6 vertex normals

vt 1.00    0.00    0.00
vt 1.00    1.00    0.00
vt 0.00    1.00    0.00
…
# 4 texture coords

g Box01
usemtl 01___Default
f 1/1/1 2/2/1 3/3/1
f 3/3/1 4/4/1 1/1/1
f 5/4/2 6/1/2 7/2/2
…
# 12 faces
```

Coding

- f 1/1/1 2/2/1 3/3/1
- f 3/3/1 4/4/1 1/1/1
- f 5/4/2 6/1/2 7/2/2
- …



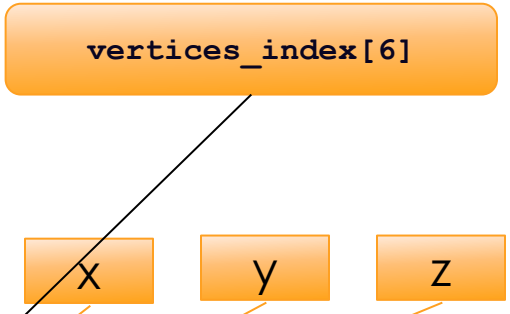
This means:

- f vertex 1 / textcoord 1 / normal 1 vertex 2 / textcoord 2 / normal 2 vertex 3 / textcoord 3 / normal 3
- f vertex 3 / textcoord 3 / normal 1 vertex 4 / textcoord 4 / normal 1 vertex 1 / textcoord 1 / normal 1
- f vertex 5 / textcoord 4 / normal 2 vertex 6 / textcoord 1 / normal 2 vertex 7 / textcoord 2 / normal 2

Coding

- f 1/1/1 2/2/1 3/3/1
- f 3/3/1 4/4/1 1/1/1
- f 5/4/2 6/1/2 7/2/2
- …

```
indices[12 x 3] = {
        1, 1, 3, // first triangle
        3, 4, 1, // second triangle
        5, 6, 7, // third triangle
…}
```




```
vertices_array[num_face x 3 x 3] = {
-0.4, 0.0, 0.4, -0.4, 0.0, -0.4, 0.4, 0.0, 0.4, // first triangle
0.4, 0.0, -0.4, 0.4, 0.0, 0.4, -0.4, 0.0, 0.4, // second triangle
-0.4, 0.4, 0.4, 0.4, 0.4, 0.4, x, y, z,        // third triangle
…}
```

```
#
# object Box01
#
v  -0.40   0.00   0.40
v  -0.40   0.00  -0.40
v   0.40   0.00  -0.40
v   0.40   0.00   0.40
v  -0.40   0.40   0.40
v   0.40   0.40   0.40
…
# 8 vertices

vn 0.00  -1.00  -0.00
vn 0.00   1.00  -0.00
vn 0.00   0.00   1.00
…
# 6 vertex normals

vt 1.00   0.00   0.00
vt 1.00   1.00   0.00
vt 0.00   1.00   0.00
…
# 4 texture coords

g Box01
usemtl 01___Default
f 1/1/1 2/2/1 3/3/1
f 3/3/1 4/4/1 1/1/1
f 5/4/2 6/1/2 7/2/2
…
# 12 faces
```

Coding

- `float vertices_array[] = {…} //array of vertices`
- `float texcoords_array[] = {…}`
- `float normals_array[] = {…}`
- `int iNumFace = … //num of face / triangle`

```
glVertexAttribPointer(iVertexLoc, 3, GL_FLOAT, GL_FALSE, 0, vertices_array);
glEnableVertexAttribArray(iVertexLoc);
glVertexAttribPointer(iTexco ordLoc, 2, GL_FLOAT, GL_FALSE, 0,
                                                    texcoords_array);
glEnableVertexAttribArray(iTexcoordLoc);
glDrawArrays(GL_TRIANGLES, 0, iNumFace * 3);
```

Content

- Introduction
- Rendering pipeline
- Shader
- Basic GLSL-ES
- Basic Math
- MVP matrices
- Textures
- Obj model
- **Shader effect: Skydome using cube mapping**

Skymapping: Sky dome

- Called Cube mapping

Sky mapping: Sky dome

- An effect known as environment mapping
- A cube (sky box) or a sphere (sky sphere) that encapsulate the whole scene composed of six 2D textures
- A camera is placed in the center of the scene

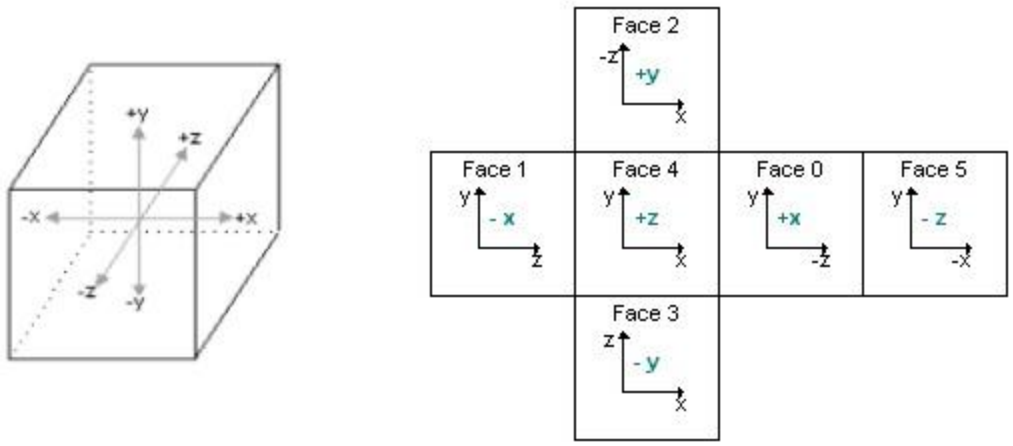
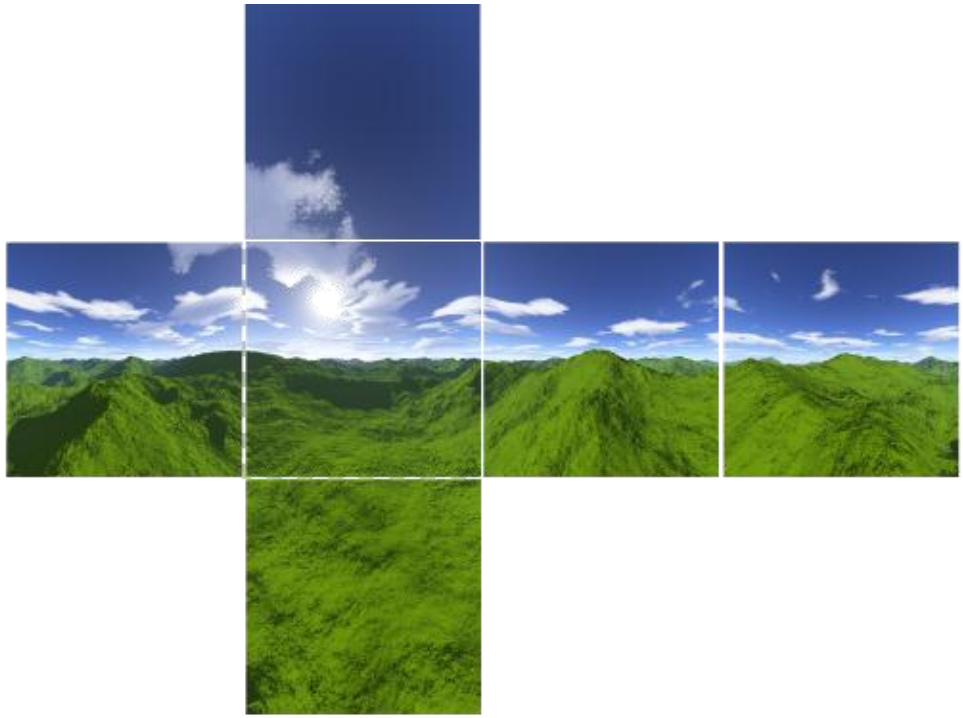


Image 26 : Cube Map Unwrap

Sky mapping: Sky dome

- A cubic texture included 6 sides of a cube
- An image of the scene is captured from each of the six axis directions (+X, –X, +Y, –Y, +Z, –Z) and stored in each cube face



GL_TEXTURE_CUBE_MAP_POSITIVE_X,

GL_TEXTURE_CUBE_MAP_NEGATIVE_X,

GL_TEXTURE_CUBE_MAP_POSITIVE_Y,

GL_TEXTURE_CUBE_MAP_NEGATIVE_Y,

GL_TEXTURE_CUBE_MAP_POSITIVE_Z,

GL_TEXTURE_CUBE_MAP_NEGATIVE_Z

Sky mapping: Coding

Vertex Cube shader

```
attribute vec4 a_CubeVertexPos;
uniform mat4 u_CubeMVPMatrix;
varying vec4 v_pos;

void main(void)
{
    gl_Position = u_CubeMVPMatrix *
                          a_CubeVertexPos;

    gl_Position.z = 1 - 0.0001;

    v_pos = a_CubeVertexPos;
}
```

Fragment Cube shader

```
precision mediump float;
uniform samplerCube u_samplerCubeMap;
varying vec4 v_pos;

void main(void)
{
   gl_FragColor =  textureCube(
              u_samplerCubeMap, v_pos.xyz);
}
```

Calculate the position of the vertex, set it's z position to be the on the farthest z possible.

Coding (conts)

Official way

```
// Generate a texture object
glGenTextures(1, &textureId);

// Bind the texture object
glBindTexture(GL_TEXTURE_CUBE_MAP, textureId);

// Load the cube face - Positive X
glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_X, 0, GL_RGB, 512,
512, 0, GL_RGB, GL_UNSIGNED_BYTE, &cubePixels[0]);

// Load the cube face - Negative X
glTexImage2D(GL_TEXTURE_CUBE_MAP_NEGATIVE_X, 0, GL_RGB,
512, 512, 0, GL_RGB, GL_UNSIGNED_BYTE, &cubePixels[1]);

// Load the cube face - Positive Y
glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_Y, 0, GL_RGB, 512,
512, 0, GL_RGB, GL_UNSIGNED_BYTE, &cubePixels[2]);

// Load the cube face - Negative Y
glTexImage2D(GL_TEXTURE_CUBE_MAP_NEGATIVE_Y, 0, GL_RGB,
512, 512, 0, GL_RGB, GL_UNSIGNED_BYTE, &cubePixels[3]);

// Load the cube face - Positive Z
glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_Z, 0, GL_RGB, 512,
512, 0, GL_RGB, GL_UNSIGNED_BYTE, &cubePixels[4]);

// Load the cube face - Negative Z
glTexImage2D(GL_TEXTURE_CUBE_MAP_NEGATIVE_Z, 0, GL_RGB, 512,
512, 0, GL_RGB, GL_UNSIGNED_BYTE, &cubePixels[4]);
```

Optimized way

```
// Generate a texture object
glGenTextures(1, &textureId);

glBindTexture(GL_TEXTURE_CUBE_MAP,
                                    textureId);

for (int i=0; i<6; i++)
{
   glTexImage2D (
      GL_TEXTURE_CUBE_MAP_POSITIVE_X+i,
      0,
      GL_RGB,
      512,
      512,
      0,
      GL_RGB,
      GL_UNSIGNED_BYTE,
      &cubePixels[i] );
}
```

Sky mapping: Coding (conts)

```
glBindTexture (GL_TEXTURE_CUBE_MAP, textureId);


glEnableVertexAttribArray ( iPosVertexLoc );
glVertexAttribPointer (iPosVertexLoc, 3, GL_FLOAT, GL_FALSE, 0,
                                                    m_aVerticesArray);


glDrawArrays (GL_TRIANGLES, 0, 36);
```

Practice

- Go to:

\\sai-data01\Documents\Specialized\Programming\Training\01. MegaTraining\Basic\3D & OpenGL\GLES 2.0 workshop\

File: OpenGL_Practice_gles_2.0.pdf

Do part 3.

Any question?