Computer Graphics Lab.

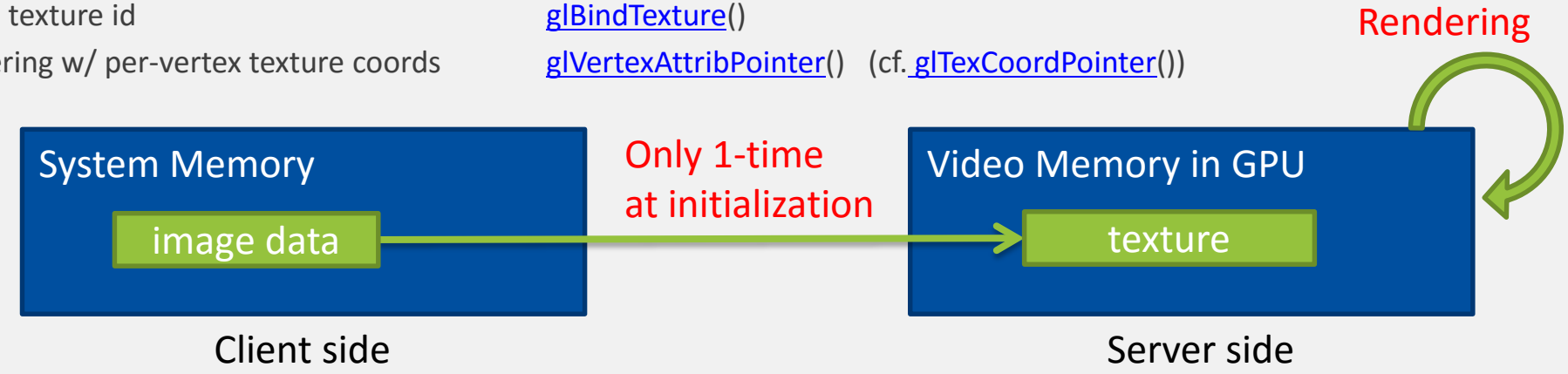
Texture Mapping in Modern OpenGL

Junho Kim

Visual Computing Lab.
Kookmin University

Using Texture Mapping in OpenGL ES 2.0

- Steps to use texture mapping
  1. Generate texture identifiers — glGenTextures()
  2. Binding a texture id — glBindTexture()
  3. Specify texture data
     - Load image from a file (or generate image)
     - Specify texture parameters — glTexParameter()
       - Wrapping mode, Filtering methods
     - Specify texture data — glTexImage2D()
  4. Rendering with texture mapping
     - Select active texture unit — glActiveTexture() (cf. glEnable(GL_TEXTURE_2D))
     - Bind a texture id — glBindTexture()
     - Rendering w/ per-vertex texture coords — glVertexAttribPointer() (cf. glTexCoordPointer())



OpenGL ES codes – Texture Mapping

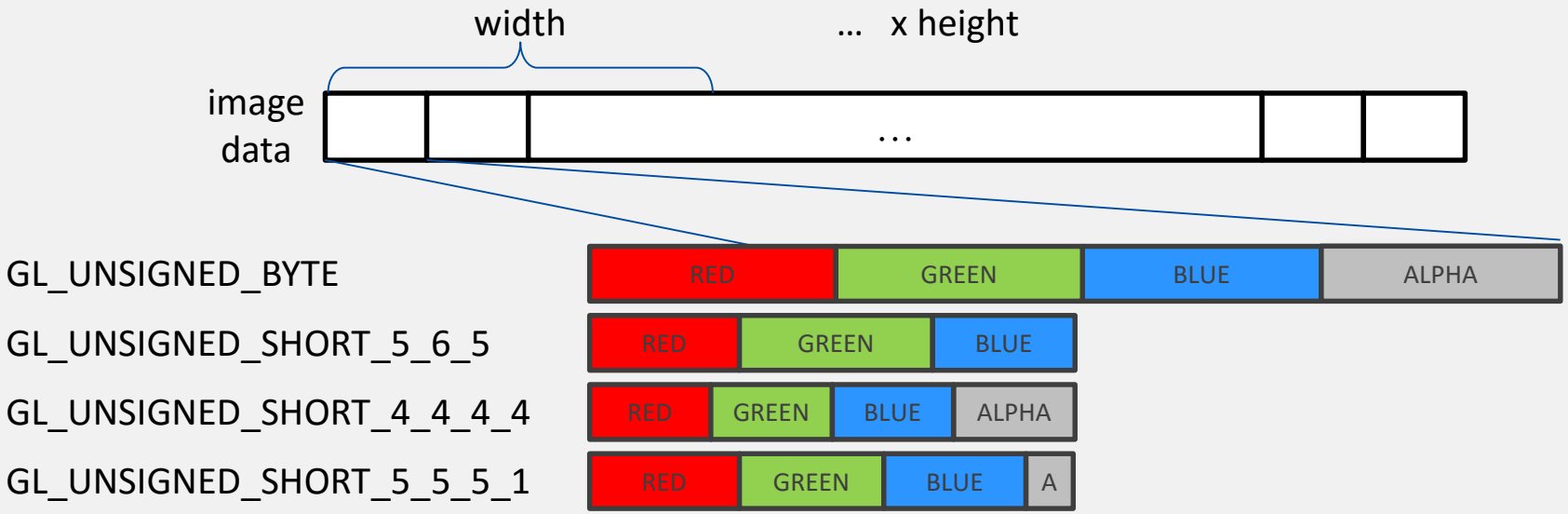
- Initialization
  1. Generate texture ids
  2. Binding a texture id
  3. Specify texture data
     - Load image from a file (or generate image)
     - Specify texture parameters
     - Wrapping mode, Filtering methods
     - Specify texture data
  4. Select active texture unit mapping
  5. Binding a texture id
  6. Rendering w/ texcoords

- OpenGL ES codes (C/C++)

```
// variables for texture mapping
GLuint      textureid;
int         width, height;
GLubyte*    image_data;

// Generate a texture
width  = 64;
height = 64;
image_data = new GLubyte[width*height*4];
// fill image_data anyhow

glGenTextures(1, &textureid);

// Bind a texture w/ the following OpenGL ES texture functions
glBindTexture(GL_TEXTURE_2D, textureid);

// Set texture parameters (wrapping modes, sampling methods)
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

// Transfer an image data in the client side to the server side
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0, GL_RGBA, GL_UNSIGNED_BYTE,
image_data);
```

Specifying a Texture Image

- Load an image from a file
  - There is no OpenGL function about it
    - You should use a platform-specific way
  - The image data should be stored in bitmap-like data structure
    - Data must be admitted by glTexImage2D()
    - In Android, you can use texImage2D() in android.opengl.GLUtils



OpenGL ES codes – Texture Mapping

- Rendering
  1. Generate texture ids
  2. Binding a texture id
  3. Specify texture data
     - Load image from a file (or generate image)
     - Specify texture parameters
     - Wrapping mode, Filtering methods
     - Specify texture data
  4. Select active texture unit
  5. Binding a texture id
  6. Rendering w/ texcoords

- OpenGL ES codes (C/C++)

```
// Enable texture mapping
glActiveTexture(GL_TEXTURE0);

// Bind a texture w/ the following OpenGL ES texture functions
glBindTexture(GL_TEXTURE_2D, textureid);

// Rendering w/ texcoords
glUniform1i(loc_u_texid, 0);

glVertexAttribPointer(loc_a_vertex, 3, GL_FLOAT, false, 0, vertices.data());
glVertexAttribPointer(loc_a_texcoord, 2, GL_FLOAT, false, 0, texcoords.data());

glEnableVertexAttribArray(loc_a_vertex);
glEnableVertexAttribArray(loc_a_texcoord);

glDrawArrays(GL_TRIANGLE_FAN, 0, 4);

glDisableVertexAttribArray(loc_a_vertex);
glDisableVertexAttribArray(loc_a_texcoord);
```

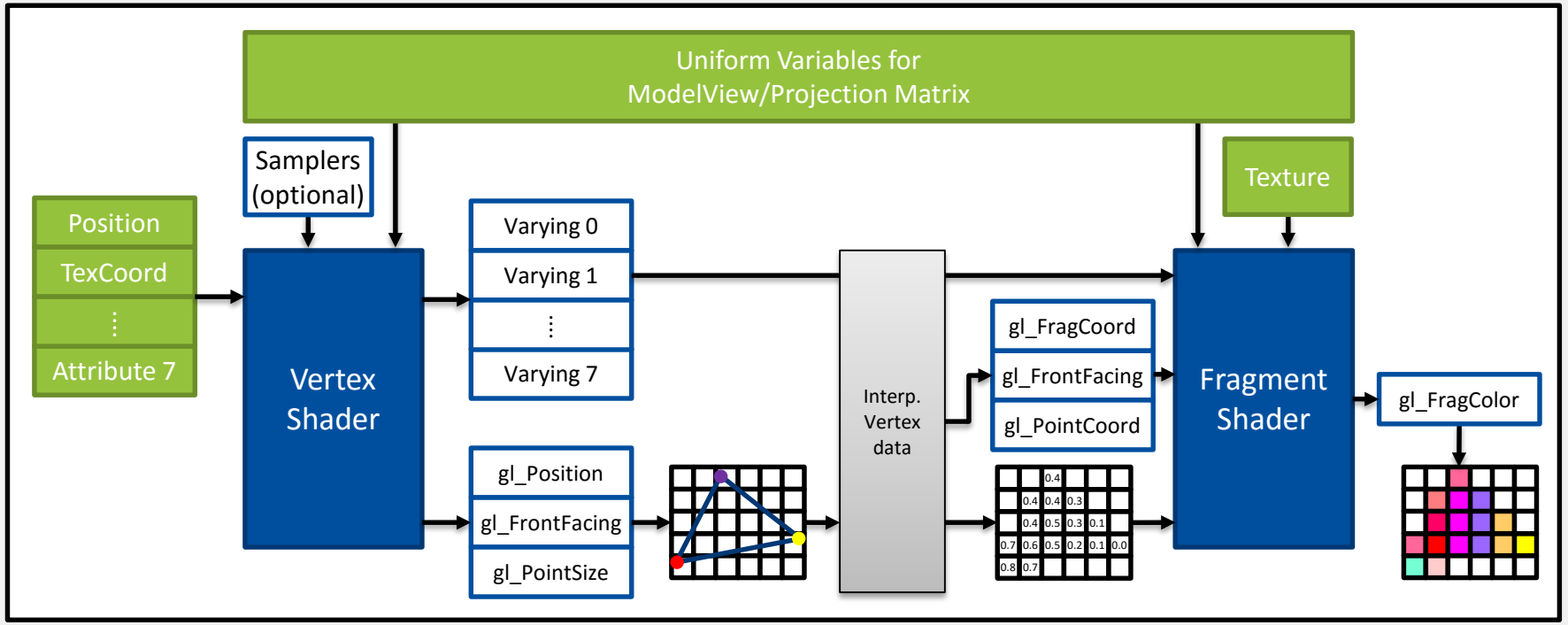
Sampler

- A specific type of uniforms that represent a single texture of a particular texture type
  - It should be initialized with the OpenGL ES API
    - It also can be declared as function parameters in Shaders
  - Used with the built-in texture lookup functions
    - texture2D(), texture2DProj(), textureCube()

```
// Use the texture coordinate coord to do a texture lookup in the 2D texture currently bound to sampler
// sampler              the sampler to which the texture from which texels will be retrieved in bound
// coord                the texture coordinates at which texture will be sampled
// bias                 an optional parameter that provides a mipmap bias to be applied during LOD computation
vec4 texture2D(sampler2D sampler, vec2 coord, [float bias]);

// coord                the texture coordinates at which texture will be sampled
                        The (x,y) arguments are divided by (z) such that the fetch occurs at (x/z, y/z)
vec4 textureProj(samplerCube sampler, vec3 coord, [float bias]);


// Use the texture coordinate coord to do a texture lookup in the cubemap texture currently bound to sampler
vec4 textureCube(samplerCube sampler, vec3 coord, [float bias]);
```

Texture Mapping with Sampler

- Strategy for programmable rendering pipeline
  - Vertex shader: typical vertex processing
    - Attributes: Position, TexCoord
    - Varying: Passing through TexCoord to the fragment shader as varying variables
  - Fragment shader: computing per-frgement color with sampler



Texture Mapping with Sampler

Vertex / Fragment Shaders

Vertex Shader

```
#version 120

// uniforms used by the vertex shader
uniform mat4   u_pvm_matrix;

// attributes input to the vertex shader
attribute vec3 a_vertex;
attribute vec2 a_texcoord;    // input vertex color

// varying variables - input to the fragment shader
varying vec2   v_texcoord;    // output vertex texCoord


void main()
{
  gl_Position = u_pvm_matrix * vec3(a_vertex, 1.0);
  v_texCoord  = a_texCoord;
}
```

Fragment Shader

```
#version 120

uniform sampler2D u_texid;
varying vec2      v_texcoord;

void main()
{
  gl_FragColor = texture2D(u_texid, v_texcoord);
}
```

OpenGL ES codes (C/C++)

```
String vertex_source, fragment_source;
// Texture object handle
int program;
int loc_u_pvm_matrix, loc_u_texid, loc_a_vertex, loc_a_texcoord;
GLuint textureid;

void init()
{
 // Compile sahders and Link them into the program
 program = Shader::create_program(vertex_source, fragment_source);

 loc_u_pvm_matrix = GLES20.glGetUniformLocation(program, " u_pvm_matrix");
 loc_u_texid = GLES20.glGetUniformLocation(program, "u_texid");
 loc_a_vertex = GLES20.glGetAttribLocation(program, "a_vertex");
 loc_a_texcoord = GLES20.glGetAttribLocation(program, "a_texcoord");
}
void set_texture2D()
{
 // Bind a texture w/ the following OpenGL ES texture functions
 glBindTexture(GL_TEXTURE_2D, textureid);

 // Set texture parameters (wrapping modes, sampling methods)
 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
}
void render()
{
 // Enable texture mapping
 glActiveTexture(GL_TEXTURE0);
 // Bind a texture w/ the following OpenGL ES texture functions
 glBindTexture(GL_TEXTURE_2D, textureid);
 glUniform1i(loc_u_texid, 0);
}
```

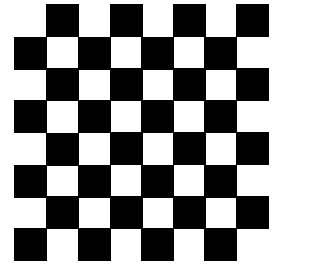
- **Chessboard Texture**
- Image Texture

Practice

Texture Mapping

Chessboard Texture – 1. Make Chessboard Bitmap

```
void load_texture()
{
  int width, height;
  GLubyte* image_data;

  // 1. generate image data as an array of rgba
  width  = 64;
  height = 64;
  image_data = new GLubyte[width*height*4];     // (width * height) * rgba
  unsigned int  cnt = 0;
  for (int i = 0; i < height; ++i)
  {
    for (int j = 0; j < width; ++j)
    {
      int c = ((((i & 0x8) == 0) ^ ((j & 0x8)) == 0)) * 255;
      image_data[cnt] = (GLubyte)c;     ++cnt;
      image_data[cnt] = (GLubyte)c;     ++cnt;
      image_data[cnt] = (GLubyte)c;     ++cnt;
      image_data[cnt] = (GLubyte)255;   ++cnt;
    }
  }
  // 2-5. DO something about handling textures in OpenGL

  // 6. destroy image data
  delete[] image_data;
}
```



Chessboard Texture – 2. Initialize Texture

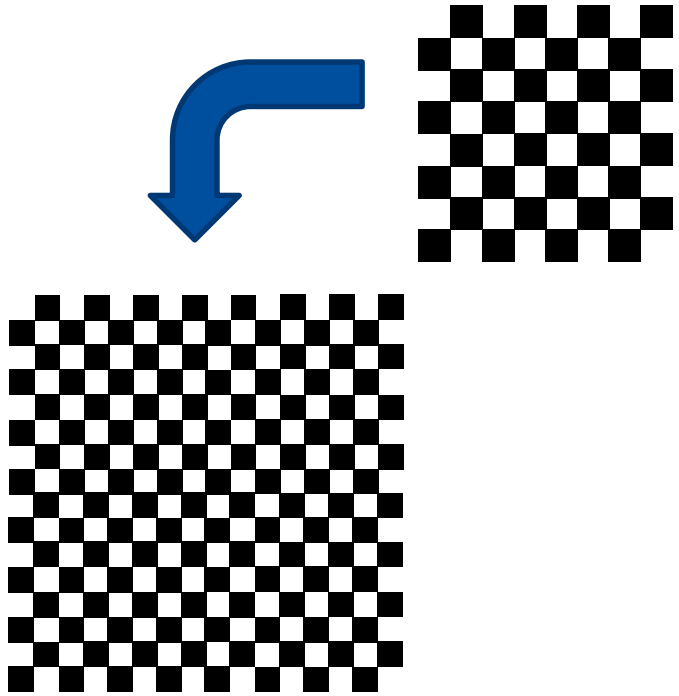
```
void load_texture()
{
  // 1. generate image data as an array of rgba
  // . . .

  // 2. create a texture object
  glGenTextures(1, &textureid);
  // 3. bind the texture object, so texture operations work for the binded texture object
  glBindTexture(GL_TEXTURE_2D, textureid);
  // 4. upload texture data to the GPU
  glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, width, height, 0, GL_RGBA, GL_UNSIGNED_BYTE, image_data);

  // 5. set texture parameters
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

  // 6. destroy image data
  // . . .
}
```

Chessboard Texture – 3. Bind Texture

```
void display()
{
  // . . .

  // 7. activate texture unit that number 0
  glActiveTexture(GL_TEXTURE0);
  // 8. bind the texture object
  //  so all texture operations as follows work for the binded texture object
  glBindTexture(GL_TEXTURE_2D, textureid);

  // 9. let shader know that we will use the texture unit 0
  glUniform1i(loc_u_texid, 0);

  // 10. send texcoord as per-vertex attribute data
  glVertexAttribPointer(loc_a_texcoord, 2, GL_FLOAT, false, 0, texcoords.data());

  // . . .
  glEnableVertexAttribArray(loc_a_texcoord);

  glDrawArrays(GL_TRIANGLE_FAN, 0, 4);

  glDisableVertexAttribArray(loc_a_texcoord);
  // . . .
}
```

Chessboard Texture – 4. Set texcoord values

```
// texcoord values
std::vector<glm::vec2> texcoords;

void init()
{
  // . . .

  // texture coordinates
  texcoords.push_back(glm::vec2(0.0f, 0.0f));
  texcoords.push_back(glm::vec2(2.0f, 0.0f));
  texcoords.push_back(glm::vec2(2.0f, 1.0f));
  texcoords.push_back(glm::vec2(0.0f, 2.0f));

  load_texture();
  // . . .
}
```



Chessboard Texture – 5. Draw

```
void display()
{
  // . . .

  glVertexAttribPointer(loc_a_vertex, 3, GL_FLOAT, false, 0, vertices.data());
  glVertexAttribPointer(loc_a_texcoord, 2, GL_FLOAT, false, 0, texcoords.data());

  glEnableVertexAttribArray(loc_a_vertex);
  glEnableVertexAttribArray(loc_a_texcoord);

  glDrawArrays(GL_TRIANGLE_FAN, 0, 4);

  glDisableVertexAttribArray(loc_a_vertex);
  glDisableVertexAttribArray(loc_a_texcoord);

  // . . .
}
```

Chessboard Texture – 6. Vertex & Fragment Shaders

Vertex Shader

```
#version 120

// uniforms used by the vertex shader
uniform mat4   u_pvm_matrix;

// attributes input to the vertex shader
attribute vec4 a_vertex;
attribute vec4 a_texcoord;    // input vertex color

// varying variables - input to the fragment shader
varying vec2   v_texcoord;    // output vertex texCoord


void main()
{
  gl_Position = u_pvm_matrix * a_vertex;
  v_texCoord  = a_texCoord;
}
```

Fragment Shader

```
#version 120

varying vec2 v_texcoord;
uniform sampler2D u_texid;

void main()
{
  gl_FragColor = texture2D(u_texid, v_texcoord);
}
```

Chessboard – Result