Implementation: Shaders

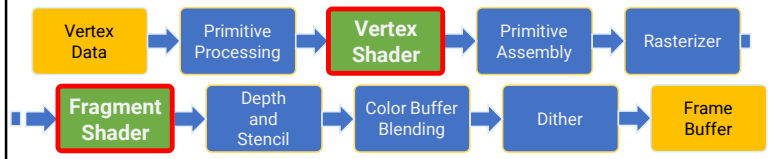
Introduction to Computer Graphics

Yu-Ting Wu

1

Overview

- The graphics pipeline in OpenGL 2.0
- Programmers need to provide the two shader programs
- Other stages maintain the same (set OpenGL states)



Important concepts

- The vertex shader runs **per vertex**
- The fragment shader runs **per (rasterized) fragment**

2

Goals

- Introduce how to create, load, and setup shaders in an OpenGL program
- Introduce the simplest vertex/fragment programs
- Introduce how to communicate between the CPU program and GPU shaders

3

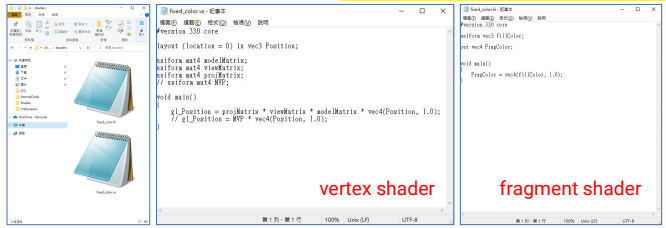
Programs

4

Prepare Shaders

- Shaders are just text files written in a special shader language, such as
  - OpenGL Shading Language (GLSL)
  - High-Level Shading Language (HLSL) for DirectX
  - Nvidia Cg (used by Unity)

the file extension does not matter!



5

Load and Create an OpenGL Shader

```
// Shader.
GLuint shaderProgId;
GLint locM, locV, locP, locMVP;
GLint locFillColor;

void CreateShader(const std::string vsFilePath, const std::string fsFilePath)
{
    // Create OpenGL shader program.
    shaderProgId = glCreateProgram();
    if (shaderProgId == 0) {
        std::cerr << "[ERROR] Failed to create shader program" << std::endl;
        exit(1);
    }

    // Load the vertex shader from a source file and attach it to the shader program.
    std::string vs, fs;
    if (!LoadShaderTextFromFile(vsFilePath, vs)) {
        std::cerr << "[ERROR] Failed to load vertex shader source: " << vsFilePath << std::endl;
        exit(1);
    }
    GLuint vsId = AddShader(vs, GL_VERTEX_SHADER);

    // Load the fragment shader from a source file and attach it to the shader program.
    if (!LoadShaderTextFromFile(fsFilePath, fs)) {
        std::cerr << "[ERROR] Failed to load vertex shader source: " << fsFilePath << std::endl;
        exit(1);
    };
    GLuint fsId = AddShader(fs, GL_FRAGMENT_SHADER);
```

Create OpenGL shader program (ID)

in our case, a shader program consists of a vertex shader and a fragment shader

Load vertex shader source

Create, compile the vertex shader and attach it to the shader program

Load fragment shader source

Create, compile the fragment shader and attach it to the shader program

6

Load and Create an OpenGL Shader (cont.)

```
// Link and compile shader programs.
GLint success = 0;
GLchar errorLog[MAX_BUFFER_SIZE] = { 0 };
glLinkProgram(shaderProgId);
glGetProgramiv(shaderProgId, GL_LINK_STATUS, &success);
if (success == 0) {
    glGetProgramInfoLog(shaderProgId, sizeof(errorLog), NULL, errorLog);
    std::cerr << "[ERROR] Failed to link shader program: " <<  errorLog << std::endl;
    exit(1);
}

// Now the program already has all stage information, we can delete the shaders now.
glDeleteShader(vsId);
glDeleteShader(fsId);

// Validate program.
glValidateProgram(shaderProgId);
glGetProgramiv(shaderProgId, GL_VALIDATE_STATUS, &success);
if (!success) {
    glGetProgramInfoLog(shaderProgId, sizeof(errorLog), NULL, errorLog);
    std::cerr << "[ERROR] Invalid shader program: " << errorLog << std::endl;
    exit(1);
}

// Get the location of uniform variables.
// Discuss later
}
```

Link all **attached** shaders to the program

Delete (free memory) vertex/fragment shader object

Validate your shader program

7

Load and Create an OpenGL Shader

```
GLuint AddShader(const std::string& sourceText, GLenum shaderType)
{
    GLuint shaderObj = glCreateShader(shaderType);
    if (shaderObj == 0) {
        std::cerr << "[ERROR] Failed to create shader with type " << shaderType << std::endl;
        exit(0);
    }

    const GLchar* p[1];
    p[0] = sourceText.c_str();
    GLint lengths[1];
    lengths[0] = (GLint)(sourceText.length());
    glShaderSource(shaderObj, 1, p, lengths);
    glCompileShader(shaderObj);

    GLint success;
    glGetShaderiv(shaderObj, GL_COMPILE_STATUS, &success);
    if (!success) {
        GLchar infoLog[MAX_BUFFER_SIZE];
        glGetShaderInfoLog(shaderObj, MAX_BUFFER_SIZE, NULL, infoLog);
        std::cerr << "[ERROR] Failed to compile shader with type: " << shaderType << ". Info: " << infoLog << std::endl;
        exit(1);
    }

    glAttachShader(shaderProgId, shaderObj);

    return shaderObj;
}
```

Types:
GL_VERTEX_SHADER /
GL_FRAGMENT_SHADER
GL_GEOMETRY_SHADER
GL_TESS_CONTROL_SHADER,
GL_TESS_EVALUATION_SHADER,
GL_COMPUTE_SHADER

8

Vertex Shader

```
#version 330 core
layout (location = 0) in vec3 Position;
uniform mat4 modelMatrix;
uniform mat4 viewMatrix;
uniform mat4 projMatrix;

void main() {
    gl_Position = projMatrix * viewMatrix *
                  modelMatrix * vec4(Position, 1.0);
}
```

Vertex attribute
- **glEnableVertexAttribArray(0)**

uniform variables communicated with the CPU
- Get location by **glGetUniformLocation**
- Set value by **glUniformXXX**

the main program **executed per vertex**

Vertex Shader

```
#version 330 core
layout (location = 0) in vec3 Position;
uniform mat4 MVP;

void main() {
    gl_Position = MVP * vec4(Position, 1.0);
}
```

Input: vertex attribute
- **glEnableVertexAttribArray(0)**

uniform variables communicated with the CPU
- Get location by **glGetUniformLocation**
- Set value by **glUniformXXX**

the main program **executed per vertex**

Fragment Shader

```
#version 330 core
uniform vec3 fillColor;

out vec4 FragColor;

void main() {
    FragColor = vec4(fillColor, 1.0);
}
```

uniform variables communicated with the CPU
- Get location by **glGetUniformLocation**
- Set value by **glUniformXXX**

Output: fragment data

the main program **executed per fragment**

Connect the Program with Shaders

- Get the location of uniform variables in the shader

```
// Get the location of uniform variables.
locM = glGetUniformLocation(shaderProgId, "modelMatrix");
locV = glGetUniformLocation(shaderProgId, "viewMatrix");
locP = glGetUniformLocation(shaderProgId, "projMatrix");
locMVP = glGetUniformLocation(shaderProgId, "MVP");
locFillColor = glGetUniformLocation(shaderProgId, "fillColor");
```

- Assign values to the uniform variables in shaders

```
// Bind shader and set parameters.
glUseProgram(shaderProgId);
glUniformMatrix4fv(locM, 1, GL_FALSE, glm::value_ptr(M));
glUniformMatrix4fv(locV, 1, GL_FALSE, glm::value_ptr(camera->GetViewMatrix()));
glUniformMatrix4fv(locP, 1, GL_FALSE, glm::value_ptr(camera->GetProjMatrix()));
// glUniformMatrix4fv(locMVP, 1, GL_FALSE, glm::value_ptr(MVP));
glUniform3fv(locFillColor, 1, glm::value_ptr(fillColor));

// Render the mesh.
if (mesh != nullptr)
    mesh->Draw();

// Unbind shader.
glUseProgram(0);
```

bind (there might be several shaders in your program)

unbind

Connect the Program with Shaders (cont.)

- Bind and unbind to a shader program

the shader program you created

void glUseProgram(GLuint program);

```
glUseProgram(shaderProgId);
// set parameters
// render something
glUseProgram(0);
```


Connect the Program with Shaders (cont.)

- Get the location of uniform variables in the shader

GLint glGetUniformLocation(
GLuint program , the shader program you created
*const GLchar *name*
); the uniform variable in the shader

```
// Get the location of uniform variables.
locM = glGetUniformLocation(shaderProgId, "modelMatrix");
locV = glGetUniformLocation(shaderProgId, "viewMatrix");
locP = glGetUniformLocation(shaderProgId, "projMatrix");
locMVP = glGetUniformLocation(shaderProgId, "MVP");
locFillColor = glGetUniformLocation(shaderProgId, "fillColor");
```


Connect the Program with Shaders (cont.)

- Assign values to the uniform variables
- Lots of variants depending on the variable type, please refer to https://registry.khronos.org/OpenGL-Refpages/gl4/html/glUniform.xhtml

void glUniform3fv(
GLint location , the variable location get by glGetUniformLocation
GLsizei count , the number of the vectors (1 if not an array)
*const GLfloat *value* the values of the parameters
);

```
glm::vec3 fillColor = glm::vec3(1.0f, 1.0f, 0.0f);
glUniform3fv(locFillColor, 1, glm::value_ptr(fillColor));
```


Connect the Program with Shaders (cont.)

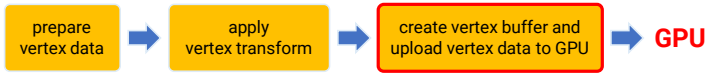
- Assign values to the uniform variables

void glUniformMatrix4fv(
GLint location ,
GLsizei count ,
GLboolean transpose , should the matrix be accessed in a transpose way **(since both OpenGL and GLM use column-major, we set it to FALSE)**
*const GLfloat *value*
);

```
glUniformMatrix4fv(locM, 1, GL_FALSE, glm::value_ptr(M));
glUniformMatrix4fv(locV, 1, GL_FALSE, glm::value_ptr(camera->GetViewMatrix()));
glUniformMatrix4fv(locP, 1, GL_FALSE, glm::value_ptr(camera->GetProjMatrix()));
```

Pitfalls

- In the previous sample code, we transform all vertices on the CPU and put their Clip-Space coordinates in the vertex buffer



- Now the vertex positions in the vertex buffer should be in Object Space

17

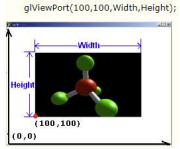
MISC

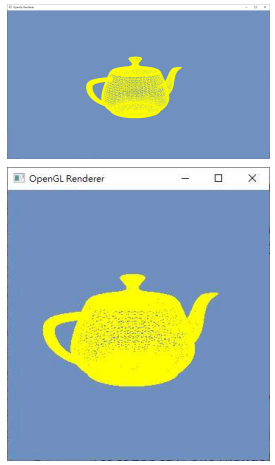
- Resize window

```
glutReshapeFunc(ReshapeCB);

void ReshapeCB(int w, int h)
{
    // Update viewport.
    screenWidth = w;
    screenHeight = h;
    glViewport(0, 0, screenWidth, screenHeight);
    // Adjust camera and projection.
    float aspectRatio = (float)screenWidth / (float)screenHeight;
    camera->UpdateProjection(fovy, aspectRatio, zNear, zFar);
    MVP = camera->GetProjMatrix() * camera->GetViewMatrix() * M;
}
```

remember to reset the range of rendering in an OpenGL window





18

Any Questions?

19