Computer Graphics HW2

Info

- Author: 0712238, Yan-Tong Lin
- Online Version: https://hackmd.io/3xeAp1AHTzGBpZawoP1kRg (https://hackmd.io/3xeAp1AHTzGBpZawoP1kRg)

GLSL for drawing Umbreon

Overview

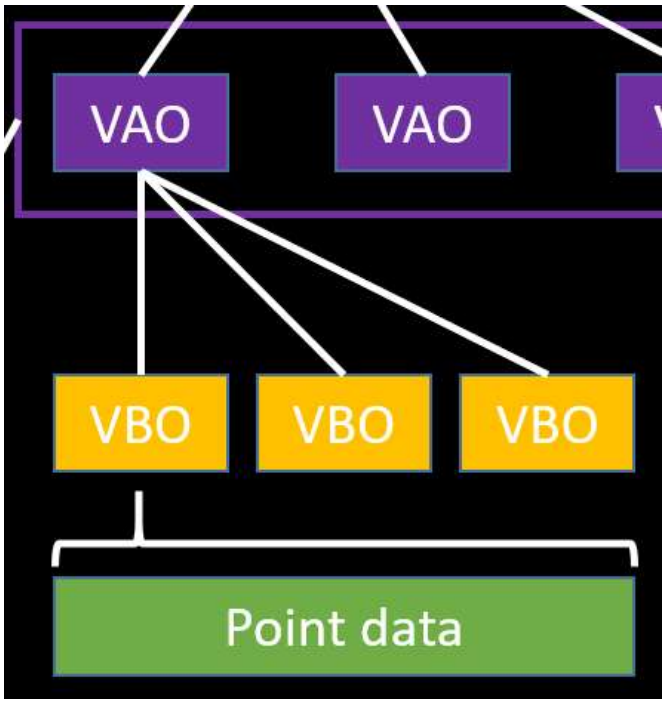
- Shader Codes $\xrightarrow{\text{bind}}$ `program`
- Vectors $\xrightarrow{\text{bind}}$ `VBO` $\xrightarrow{\text{bind}}$ `VAO`
- Texture Graphics $\xrightarrow{\text{load}}$ `obj_texture`
- Data Vectors/MVP matrices $\xrightarrow{\text{pass with binding}}$ Vertex Shader $\xrightarrow{\text{texture coordinates}}$ Fragment Shader(texture is passed to here) $\rightarrow$ Result

Program Linking

```
void shaderInit() {
    //// TODO: ////
    //
    // Hint:
    // 1. createShader
    // 2. createProgram
    GLuint vert = createShader("Shaders/vertexShader.vert", "vertex");
    GLuint frag = createShader("Shaders/fragmentShader.frag", "fragment");
    program = createProgram(vert, frag);
}
```

VAO and VBO Binding

- bind three VBOs to a VAO



- utilities for VBOs
  1. position vectors
  2. texture coordinates
  3. normal vectors

```
void bindbufferInit() {
    //// TODO: ////
    //
    //   Hint:
    // 1. Setup VAO
    // 2. Setup VBO of vertex positions, normals, and texcoords
    int verticeNumber = (model->fNum) * 4;
    glGenVertexArrays(1, &VAO);
    glBindVertexArray(VAO);
    glGenBuffers(3, VBO);

    glBindBuffer(GL_ARRAY_BUFFER, VBO[0]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(float) * model->positions.size(), &model->positions[0], GL_STATIC_DRAW);
    glEnableVertexAttribArray(0);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(float) * 3, 0);

    glBindBuffer(GL_ARRAY_BUFFER, VBO[1]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(float) * model->texcoords.size(), &model->texcoords[0], GL_STATIC_DRAW);
    glEnableVertexAttribArray(1);
    glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, sizeof(float) * 2, 0);

    glBindBuffer(GL_ARRAY_BUFFER, VBO[2]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(float) * model->normals.size(), &model->normals[0], GL_STATIC_DRAW);
    glEnableVertexAttribArray(2);
    glVertexAttribPointer(2, 3, GL_FLOAT, GL_FALSE, sizeof(float) * 3, 0);
    glBindBuffer(GL_ARRAY_BUFFER, 0);
    glBindVertexArray(0);
}
```

Texture Loading

```
void textureInit() {
    // use LoadTexture function to init texture matrix
    LoadTexture(basistexture, "basis.jpg");
    LoadTexture(modeltexture, "Umbreon.jpg");
}
```

Code for Vertex Shader

- read in the parameters
- calculate `gl_Position`

- pass data required by the fragment shader (i.e. `texcoord_frag` and `normal`)

```
#version 430

//// TODO: ////
//
// Hint:
// 1. Receive position, normal, texCoord from bind buffer
// 2. Receive Model matrix, View matrix, and Projection matrix from uniform
// 3. Pass texcoord and Normal to fragment shader (normal will not use in this homework)
// 4. Calculate view space by gl_Position (must be vec4)

layout(location = 0) in vec3 position;
layout(location = 1) in vec2 texCoord;
layout(location = 2) in vec3 noral;

uniform mat4 Projection;
uniform mat4 M;
uniform mat4 View;

//pass to fragment shader
out vec2 texCoord_frag;
out vec3 noral_frag;

void main()
{
  gl_Position = Projection * View * M * vec4(position, 1.0);
  texCoord_frag = texCoord;
  noral_frag = noral;
}

```

Code for Fragment Shader

- use
  - texture object
  - texture coordinate (i.e. `texcoord_frag`) from vertex shader
- to calculate

- color for each fragment

main.cpp | fragmentShader.frag | vertexShader.vert

```
#version 430
//// TODO ////
//
// Hint:
// 1. Recieve texcoord and Normal from vertex shader
// 2. Calculate and return final color to opengl
//

layout(binding = 0) uniform sampler2D objectTexture;
in vec2 texcoord_frag;
out vec4 outColor;

void main()
{
    outColor = texture2D(objectTexture, texcoord_frag);
}

```

Drawing with Shaders

- the params passing part

```
void DrawUmbreon()
{
    glm::mat4 M(1.0f);
    M = glm::rotate(M, glm::radians(angle), glm::vec3(0, 1, 0));
    M = glm::translate(M, glm::vec3(0, 1.3, 0));
    GLuint ModelMatrixID = glGetUniformLocation(program, "M");

    //// TODO: ////
    // pass projection matrix, and view matrix and trigger by Uniform (use getP() amd getV())
    // also pass modeltexture to shader and trigger by Uniform
    glm::mat4 vmtx = getV();
    glm::mat4 pmtx = getP();
    GLint pmatLoc = glGetUniformLocation(program, "Projection");
    GLint vmatLoc = glGetUniformLocation(program, "View");
    GLint texLoc = glGetUniformLocation(program, "objectTexture");

    glUseProgram(program);
    glUniformMatrix4fv(ModelMatrixID, 1, GL_FALSE, &M[0][0]);
    glUniformMatrix4fv(pmatLoc, 1, GL_FALSE, &pmtx[0][0]);
    glUniformMatrix4fv(vmatLoc, 1, GL_FALSE, &vmtx[0][0]);

    glBindVertexArray(VAO);
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, modeltexture);
    glUniform1i(texLoc, 0);


    glDrawArrays(GL_QUADS, 0, 4 * model->fNum);
    glBindVertexArray(0);
    glBindTexture(GL_TEXTURE_2D, 0);

    glActiveTexture(0);
    glUseProgram(0);
}
```

Problems Encountered

1. `glUseprogram` has to be in front of all matrices binding (i.e. `glUniformMatrix4fv`) for them to be read in.
2. The order and the names of the variables passed to shaders (and from the vertex shader to the fragment shader) should be compatible.

Bonus

- None