HW2

Shader

- A program designed by user

- Runs in GPU

| Vertex | → | Vertex Shader | → | Fragment Shader | → | Frame Buffer |

Input: A vertex
Output: A vertex

Input: A pixel
Output: Pixel value
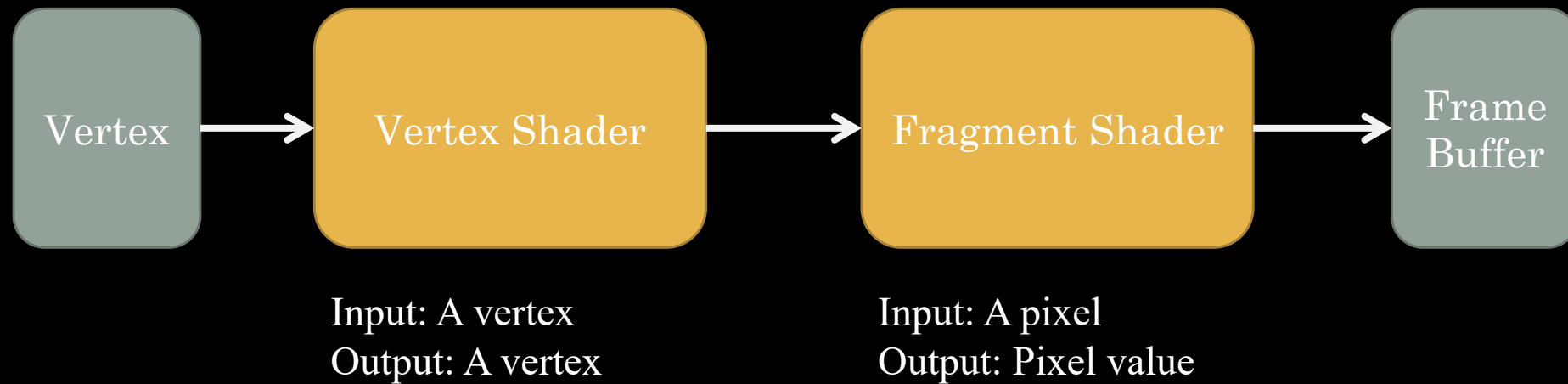
Use Shader

- First, create shader
  - GLint glCreateShader(GLenum shaderType)
    - Create a shader object with specified shader type
    - shaderType : GL_VERTEX_SHADER, GL_FRAGMENT_SHADER in this homework
  - void glShaderSource(GLint shader, GLsizei count, const GLchar **string, const GLint *length)
    - Set the source code if shader to string
  - void glCompileShader(GLint shader)
    - Compile the shader

Use Shader

- Then, create shader program
  - GLuint glCreateProgram()
    - Create a program object
  - void glAttachShader(GLuint program, GLuint shader)
    - Attach shader to program
  - void glLinkProgram(GLuint program)
    - Link the program
  - void glDeleteShader(GLunit shader)
    - Delete the shader after link

Use Shader

```
while (!glfwWindowShouldClose(window))
{
        glUseProgram(program);

        // Pass parameters to shader program

        // Draw with the shader program

        glUseProgram(0);

        // Stop using program

        glUseProgram(another_program);

        // …
}
```

VBO (Vertex Buffer Object)

- Passing data from CPU to GPU is relatively slow

- We can send a large amount of data to GPU at one time to speed up execution

- 1. void glGenBuffer(GLsizei n, GLuint *buffers)
  - Generate n buffer(s), stored in buffers

```
unsigned int VBO;
glGenBuffers(1, &VBO);
```

- 2. void glBindBuffer(GLenum target, GLuint buffer)
  - Bind buffer to target, which is GL_ARRAY_BUFFER in this homework

```
glBindBuffer(GL_ARRAY_BUFFER, VBO);
```

VBO

- 3. Set up the data

- 4. void glBufferData(GLenum target, GLsizeptr size,
                    const GLvoid *data, GLenum usage)
  - Copy the data into buffer

> GL_STATIC_DRAW:
> Data is set only once
> and is used many times

```
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
```

Vertex Attribute Pointer

- We now need to specify how openGL interprets the data

- void glVertexAttribPointer(GLuint index, GLint size,
  GLenum type, GLboolean normalized, GLsizei stride,
  const GLvoid *pointer)
  - Index: sets the location of the vertex attribute
  - Size: the size of vertex attribute every vertex
  - Stride: the spacing between consecutive vertex attributes
  - Pointer: the offset where the data begins in the buffer

- void glEnableVertexAttribArray(GLuint index)
  - Enable the vertex attribute

Vertex Attribute Pointer

- void glVertexAttribPointer(GLuint index, GLint size, GLenum type, GLboolean normalized, GLsizei stride, const GLvoid *pointer)
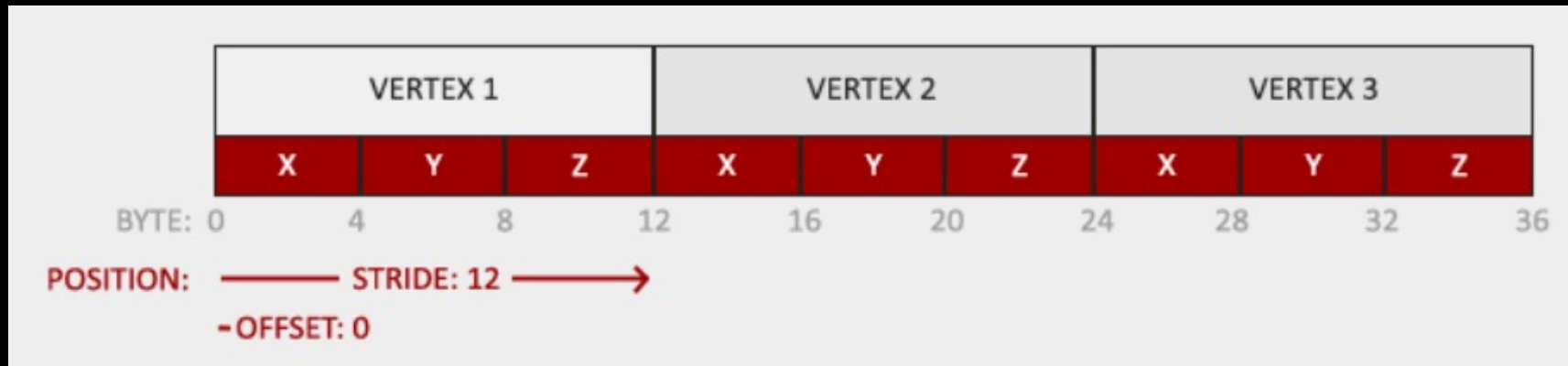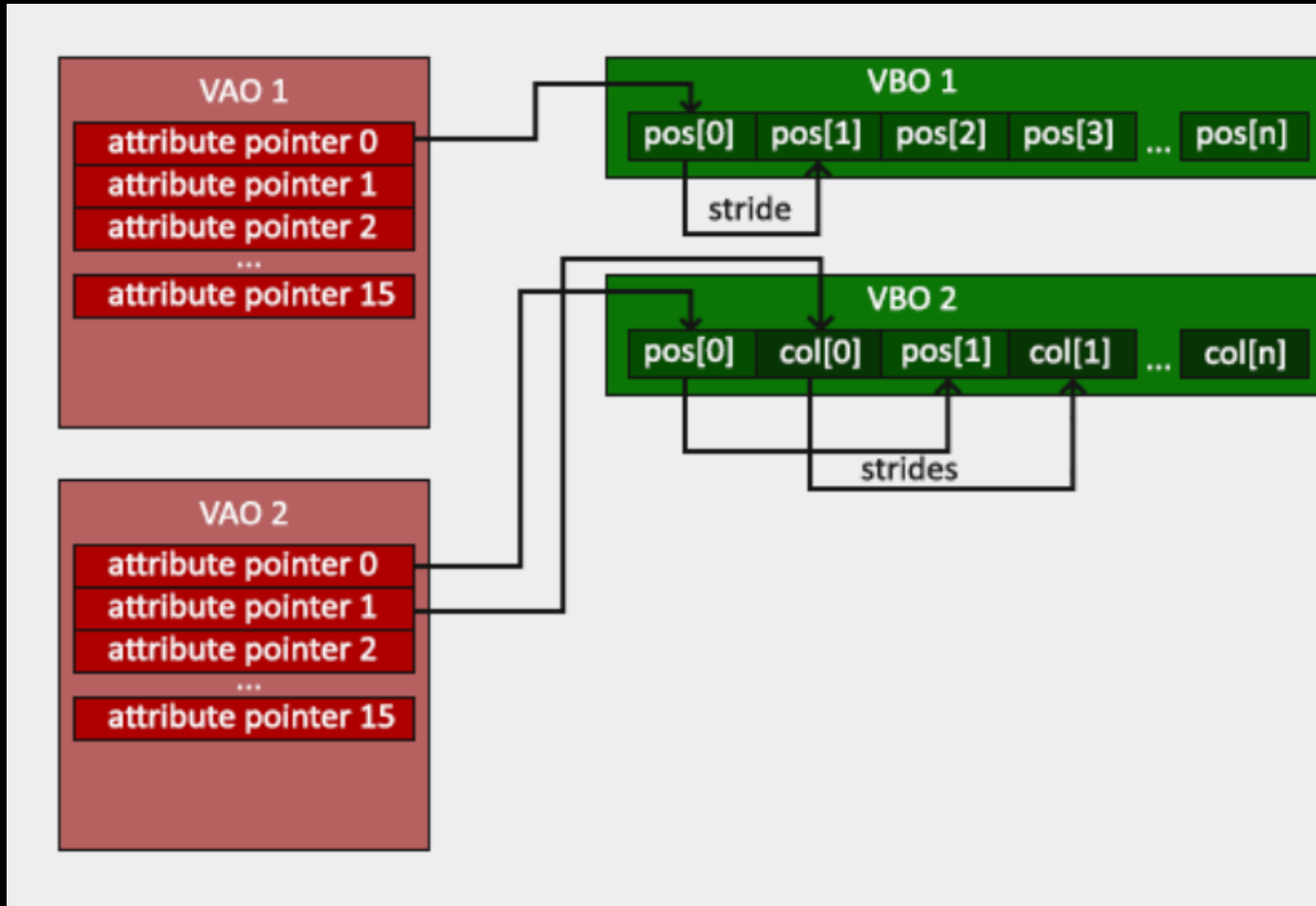


```
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void*)0);
glEnableVertexAttribArray(0);
```

VAO (Vertex Array Object)

- A model may have multiple vertex attribute pointer

- We can use a VAO to store all vertex attribute pointer

- First, create a VAO and set up all its VBO and vertex attribute pointer. Then, simply bind the VAO to render the model.

VAO (Vertex Array Object)

VAO setup

- 1. void glGenVertexArrays(GLsizei n, GLuint *array)
  - Create VAO

- 2. void glBindVertexArray(GLuint array)
  - Bind a VAO

- 3. Set VBO and vertex attribute pointer up

- 4. Unbind VAO by glBindVertexArray(0)

VAO setup

```
unsigned int VAO, VBO;
glGenVertexArrays(1, &VAO);
glBindVertexArray(VAO);
glGenBuffers(1, &VBO);

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

glBindVertexArray(0);
```

VAO during rendering

```
while (!glfwWindowShouldClose(window))

{

        // Bind the VAO of desired model

        glBindVertexArray(VAO);

        // Draw the model

        glDrawArrays(GL_TRIANGLES, 0, vertexCounts);

        // Unbind VAO after use

        glBindVertexArray(0);

}
```

Send Data to shader - Uniform

In openGL:

glm::mat4 matrix(1.0f);
unsigned int loc = glGetUniformLocation(program, "matrix");

glUseProgram(program);
glUniformMatrix4fv(loc, 1, GL_FALSE, glm::value_ptr(matrix));

In GLSL:

uniform mat4 matrix;

https://registry.khronos.org/OpenGL-Refpages/gl4/html/glUniform.xhtml

GLSL

- C-like language for shaders

Version number

Input from vertex attribute pointer

Input from uniform

Output to fragment shader

Must assign gl_Position of type vec4 in vertex shader

```
// example vertex shader
#version 330 core

layout (location = 0) in vec3 position;

uniform vec4 color;

out vec4 vertexColor;

void main()
{
        gl_Position = vec4(position, 1.0f);
        vertexColor = color;

}
```

GLSL

Interpolated input from vertex shader

Color output

Must output a vec4 as color in fragment shader

```glsl
// example fragment shader
#version 330 core

in vec4 vertexColor;

out vec4 fragColor;

void main()
{
    fragColor = vertexColor;
}
```
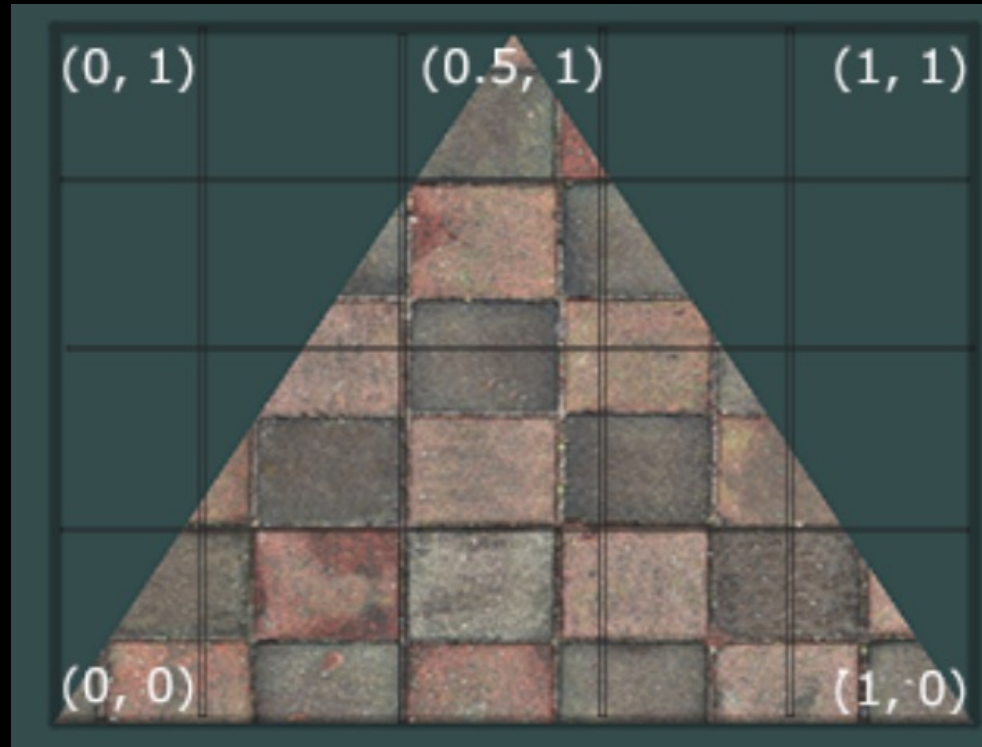
GLSL

- Variable type
  - vec2, vec3, vec4, …
  - int, float, bool, …
  - mat2, mat3, mat4, …
  - …

- Basic functions
  - min, max, sin, cos, pow, log, …
  - dot, normalize, …
  - transpose, reverse, …

Texture

- A vertex is assigned with a texture coordinate

- Color is sampled with the texture coordinate

Load Texture

- void glEnable(GL_TEXTURE_2D)
  - Enable 2D texture

- void glGenTexture(GLsizei n, GLuint *texture)
  - Generate a texture

- void glBindTexture(GLenum target, GLunit texture)
  - Bind texture so following call affect to bound texture

- void glTexImage2D(GLenum target, GLint level, GLint internalformat, GLsizei width, GLsizei height, GLint border, GLenum format, GLenum type, const void *data)
  - Generate 2D texture image with given image data

Load Texture

- void glTexParameteri(GLenum target, GLenum pname, GLint param)

- Texture coordinates usually range from (0, 0) to (1, 1). We can define how to deal with coordinates outside [0, 1]. (default to GL_REPEAT)
  - glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT)
  - glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT)

- Texture coordinates are float point value regardless of resolution. We need to decide which texture pixel to map the texture coordinate to
  - glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEARSET)
  - glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_ MAG_FILTER , GL_LINEAR)

Use Texture

// tell openGL which texture unit a sampler belong to

// texture unit 0 to ourTexture sampler

glUniform1i(glGetUniformLocation(, "ourTexture"), 0);

while (!glfwWindowShouldClose(window))

{

       glAvtiveTexture(GL_TEXTURE0);

       glBindTexture(GL_TEXTURE_2D, texture);

       // Draw with shader

}

You can pass more then one texture to shader.
In that case, you need to tell openGL which texture unit a texture sampler uses

Use Texture in GLSL

```glsl
// example texture fragment shader
#version 330 core

in vec4 vertexColor;
in vec2 texCoord;

uniform sampler2D ourTexture;

out vec4 fragColor;

void main()
{
    fragColor = texture(ourTexture, texCoord);
}
```

HW2

- Goal: use GLSL to draw a cat and a box and apply some effect with shader

- Some parameters you may need:
  - model->positions
  - model->normals
  - model->texcoords

```
vector<float> positions;
vector<float> normals;
vector<float> texcoords;
```

- Box: scale (0.0625, 0.05, 0.05)

- Cat: rotate 90 degrees around +y axis

- Both rotate 90 degrees/second
  - Hint: glfwGetTime() return the time in second from initialization

HW2 – special effect

- Effect 1: model deformation
  - Create deformation effect by changing vertices positions
  - Flattening, squeezing, …, you can do any effect you want

- Effect 2: change color
  - Change PART of the pixels color
  - For instance, you can use a threshold to find the strips of the cat and darken them
  - You can do any effect you want

- Both effects are initially off and are triggered by key press

- Tell us what you did and how to activate them in report

HW2

HW2 - score

- Basic shader program(10%)

- Set up VAO for cat and box (10% each)

- Set up texture for cat and box (10% each)

- Pass perspective, view, transform matrices, textures through uniform (10%)
  - Hint: getPerspective(), getView()

- Effects (10% each)

- Bonus (10%) – do anything you want without breaking the spec
  - Please have your bonus activated <u>after</u> pressing a key, so that we can judge your program without the bonus
  - Amazing deformation/color effect will also get bonus point!

HW2 – report (20%)

- Include your name and student ID in the report

- Tell us how you do your homework

- Don't paste your code without explanation

- Describe problems you met and how you solved them

- Explain your bonus and how to activate (optional)

- File name: studentID_report.pdf

HW2 - submission

- Deadline: 2022/11/22 23:59:59

- Pack your project and report in a zip file. File name should be studentID_hw2.zip

- 10% penalty for each week late

HW2 – reference

- https://learnopengl.com/

- https://www.glfw.org/docs/3.3/index.html