OpenGL Tutorial 2

CMPSC 458
Fall 2022

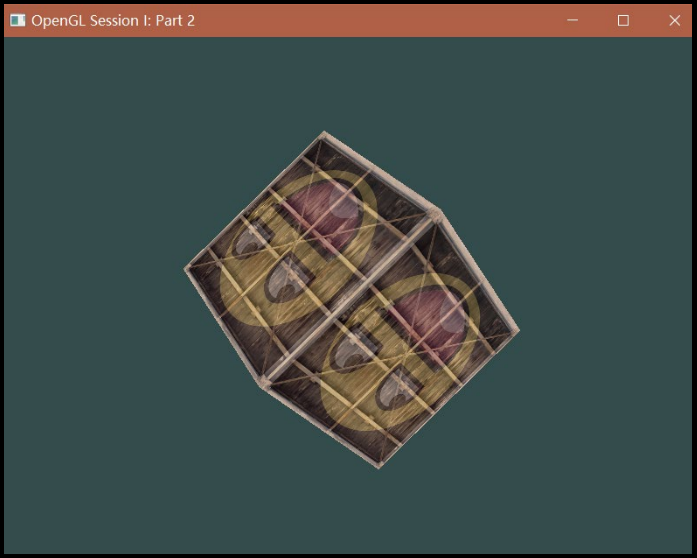
TA: Addison Petro

Tutorial 2 Overview

- How to perform transformations
- How to transform from 3D scene to 2D image
- How to create vertex and fragment shaders
- How to create textures

Tutorial 2 Setup

- Set OpenGL_tutorial_II as StartUp Project
  - Visual Studio only
  - For macOS/Linux, make and execute this project from terminal



Run the project and you should see a cube with continuous rotation

Part 1:
Math & Transformations

GLM (OpenGL Mathematics)

- Simple and lightweight mathematics library
- Fairly easy to use
- Header only library (no need to compile .lib)
- Most functions can be found:

```
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>
```

- Example transformation:

```
4x4 Matrix = glm::translate(4x4 Matrix, 3x1 vector)
```

Basic GLM – Math

- Defining vectors and matrices

```
glm::vec3(1.0f, 2.0f, 0.5f)
glm::vec4(1.0f, 0.0f, 0.3f, 1.5f)
glm::mat4(1.0f) // identity matrix
glm::mat3(1.0f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f, 1.0f)
```

- Vectors and matrices are column-major
  - The mat3 defined above would be $\begin{bmatrix} \boxed{\begin{matrix}1\\0\\0\end{matrix}} & \boxed{\begin{matrix}1\\0\\0\end{matrix}} & \boxed{\begin{matrix}0\\0\\1\end{matrix}} \end{bmatrix}$
  - Use `glm::to_string()` to format a matrix as a string
    - Helpful for debugging with print statements

Basic GLM – Math

- Add and subtract vectors or matrices:

```
glm::vec3(…) + glm::vec3(…);
glm::mat4(…) - glm::mat4(…);
```

- Multiply matrices and vectors

```
glm::vec4 x = glm::mat4(…) * glm::vec4(…);
glm::mat4 y = glm::mat4(…) * glm::mat4(…);
```

  - Multiplication from **right to left**
  - Will not compile if algebra not possible

```
glm::vec3(…) * glm::mat4(…);
```

Basic GLM – Translation

```
glm::translate(Matrix, Vector)
```

- Matrix: mat4 to be translated
- Vector: vec3 indicating translation
- Return: Matrix after translation by Vector

- Translation example

```
glm::mat4 M = glm::mat4(1.0f);
glm::vec3 V = glm::vec3(1.0f, 2.0f, 3.0f);
M = glm::translate(M, V);
```

$$\mathbf{M} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} \quad \mathbf{V} = \begin{bmatrix} \mathbf{1} \\ \mathbf{2} \\ \mathbf{3} \end{bmatrix} \quad \Longrightarrow \quad \mathbf{M} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{1} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{2} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{3} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}$$

Basic GLM – Rotation

```
glm::rotate(Matrix, Angle, Rot_Axis)
```

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Matrix: mat4 to be rotated
- Angle: float in <u>radians</u>, magnitude of rotation
- Rot_Axis: vec3 indicating rotation axis
- Return: Matrix after rotating by Angle around Rot_Axis

$$\boldsymbol{RA} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

- Rotation example

```
glm::mat4 M = glm::mat4(1.0f);
glm::vec3 RA = glm::vec3(0.0f, 0.0f, 1.0f);
M = glm::rotate(M, glm::radians(90.0f), RA);
```

$$\mathbf{M} = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Basic GLM – Scaling

```
glm::scale(Matrix, Vector)
```

- Matrix: mat4 to be scaled
- Vector: vec3 indicating scaling along each axis
- Return: Matrix after scaling by Vector

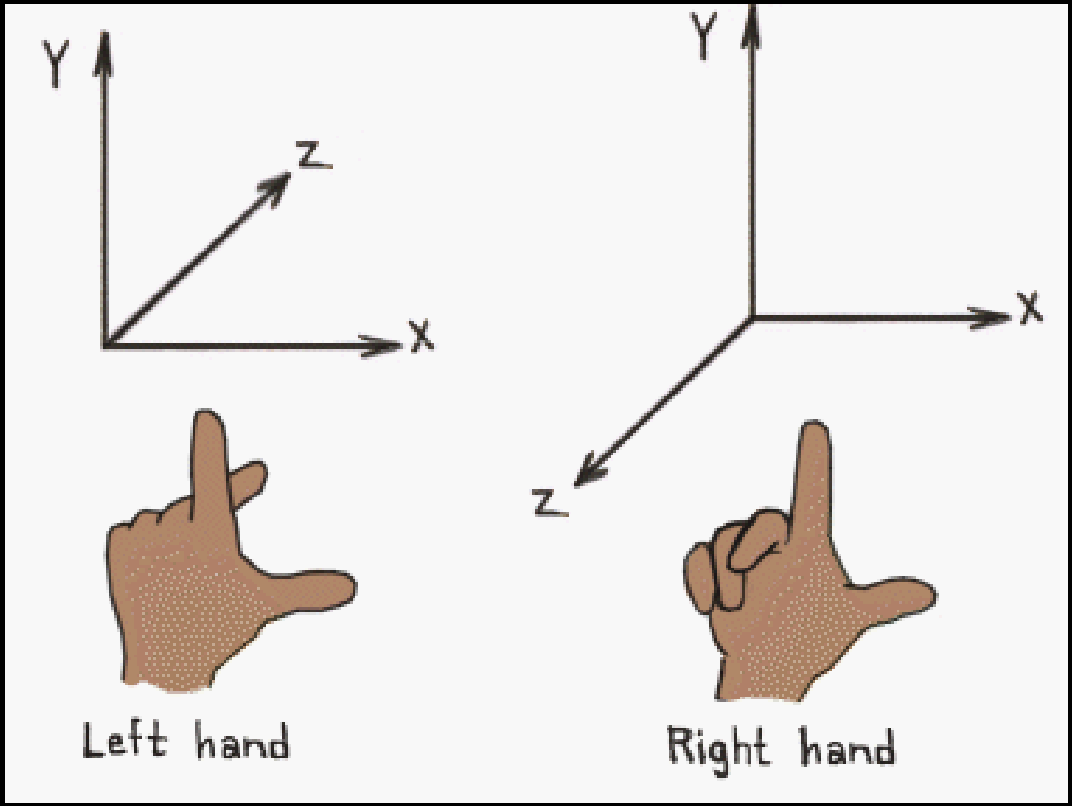
- Rotation example

```
glm::mat4 M = glm::mat4(1.0f);
glm::vec3 V = glm::vec3(1.0f, 2.0f, 3.0f);
M = glm::scale(M, V);
```

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{V} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \longrightarrow \mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Coordinate Systems

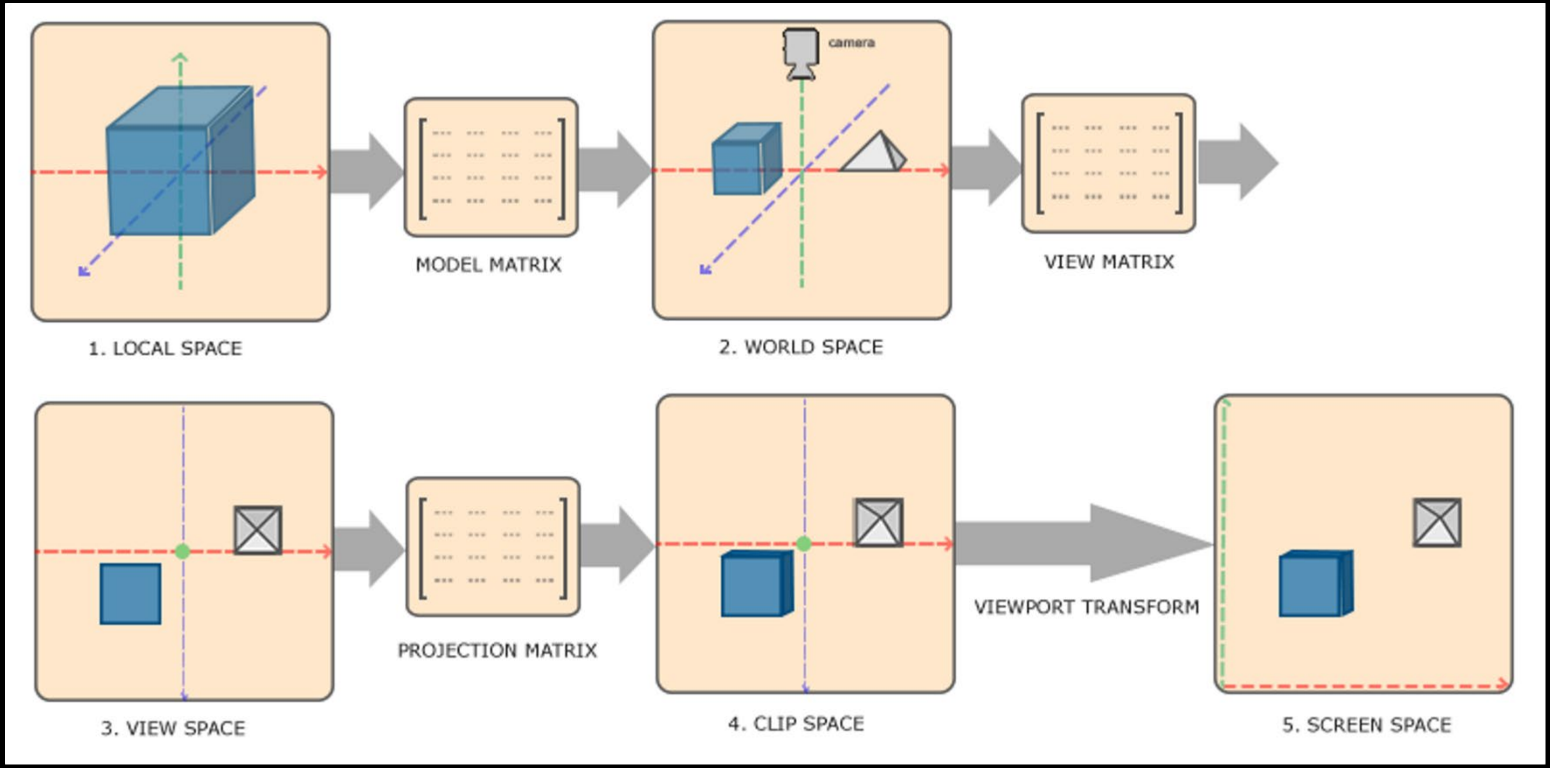
- OpenGL uses a right-handed coordinate system



Coordinate Systems

- Going from 3D to 2D

$$V_{clip} = M_{projection} * M_{view} * M_{model} * V_{local}$$



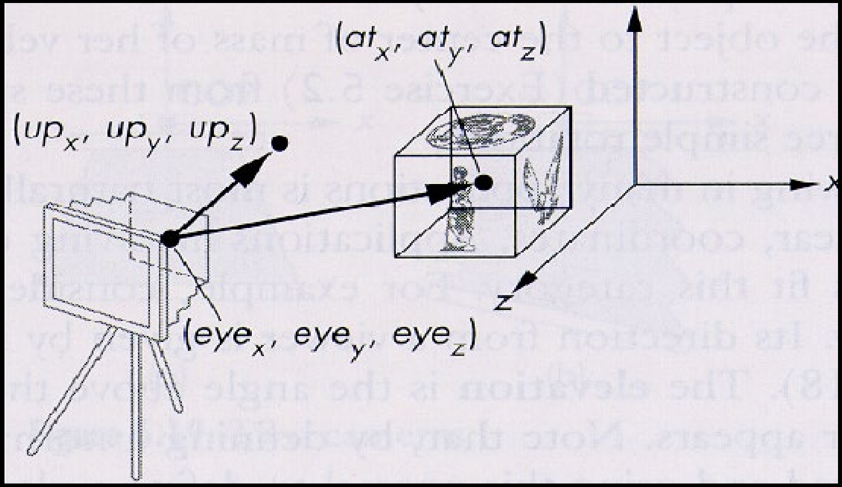
http://www.learnopengl.com/#!Getting-started/Coordinate-Systems

Coordinate Systems

- Model Matrix ($\mathrm{M}_{\mathrm{model}}$)
  - Transform vertices in local space to world space
- View Matrix ($\mathrm{M}_{\mathrm{view}}$)

```
glm::mat4 view = glm::LookAt(eye, at, up)
```

  - eye: vec3 position
  - at: vec3 viewing direction
  - up: vec3 up direction



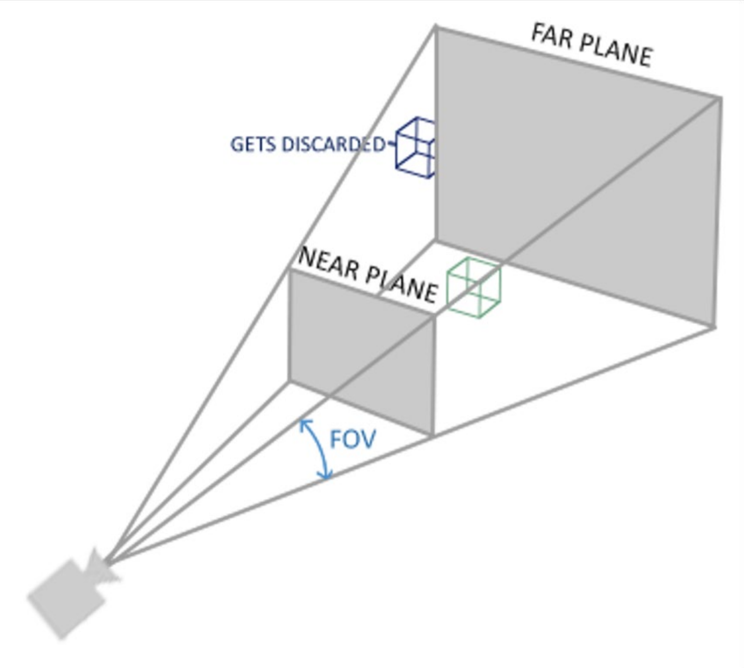
Coordinate Systems

- Projection Matrix ($\mathrm{M}_{projection}$)

  glm::mat4 proj = glm::perspective(fov, aspect, near, far)

  - fov: float field of view (radians)
  - ar: float aspect ratio
    - width / height
  - near: float near plane distance
  - far: float far plane distance

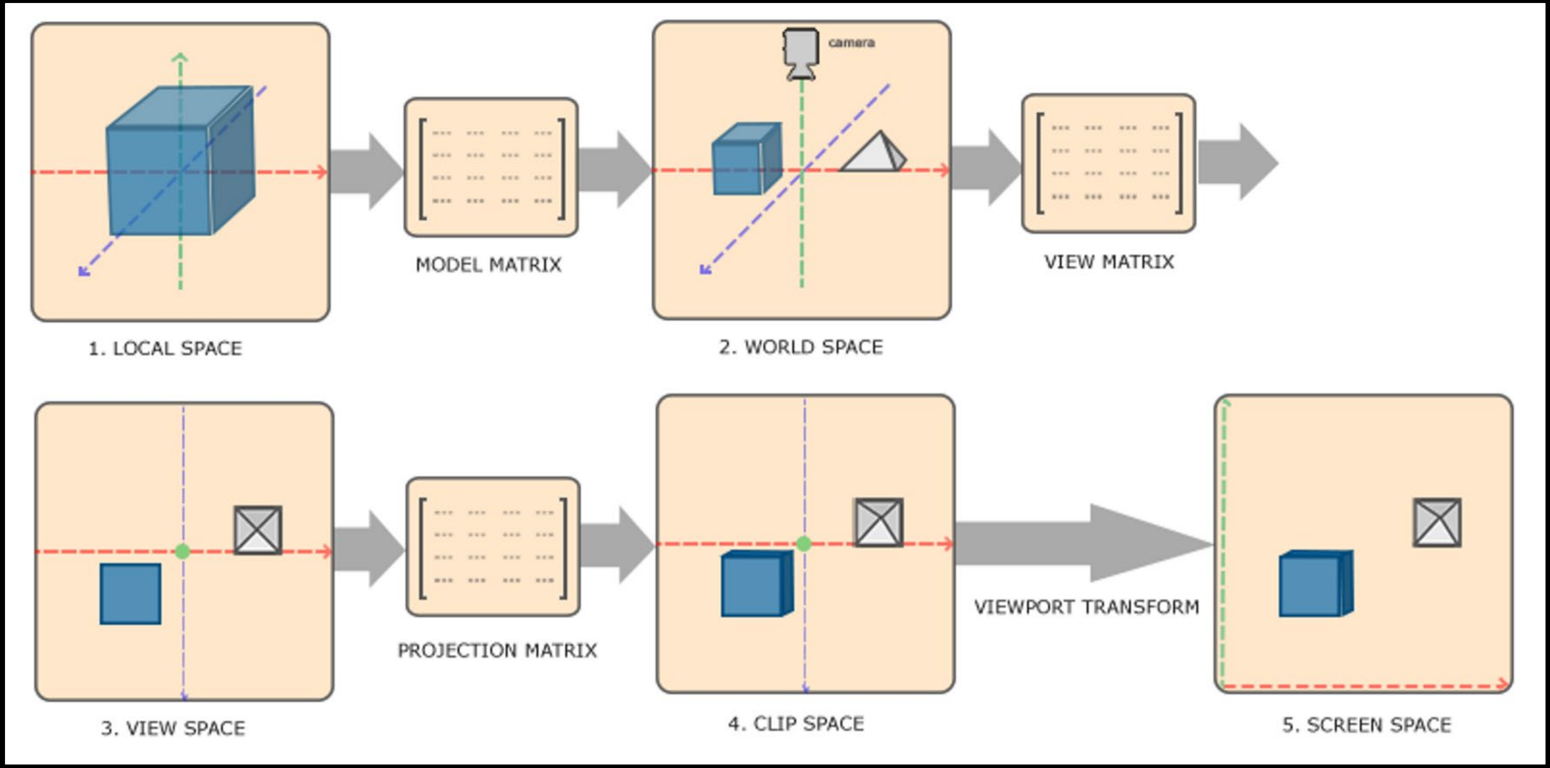
```
glm::perspective(
    glm::radians(45.0f),
    (float) w / (float) h,
    0.1f,
    100.0f);
```



Coordinate Systems

- Putting it all together

$$V_{clip} = M_{projection} * M_{view} * M_{model} * V_{local}$$



http://www.learnopengl.com/#!Getting-started/Coordinate-Systems

Coordinate Systems

- Clip Space
  - OpenGL expects coordinates to be on range [-1.0, 1.0]
  - Any coordinates outside of range get clipped
  - OpenGL performs perspective division on clip space coordinates to transform to normalized device coordinates
- Viewport Transformation
  - Use `glViewport()` to map clip space to screen space
  - Recall this function from OpenGL Tutorial 1

Part 2: Shaders

GLSL (OpenGL Shading Language)

- Vectors in GLSL have size 1, 2, 3, or 4 elements
- Vectors can have different types
  - vecX: vector of X floats
  - bvecX: vector of X bools
  - ivecX: vector of X ints
  - uvecX: vector of X uints
  - dvecX: vector of X doubles

```
vec3 aPos;
vec2 aTexCoord;
```

- Matrices can be NxM or NxN where N,M ∈ {2, 3, 4}
  - matNxM: float matrix with N columns x M rows
    - Backward from math convention!
  - matN: float matrix with N columns and rows
    - Square matrices

```
mat4 model;
mat4 view;
mat4 projection;
```

GLSL (OpenGL Shading Language)

- Accessing components of vectors
  - Use .x, .y, .z, and .w to access 1st, 2nd, 3rd, and 4th components respectively

    ```
    TexCoord = vec2(aTexCoord.x, aTexCoord.y);
    ```

  - Flexible component selection (swizzling)

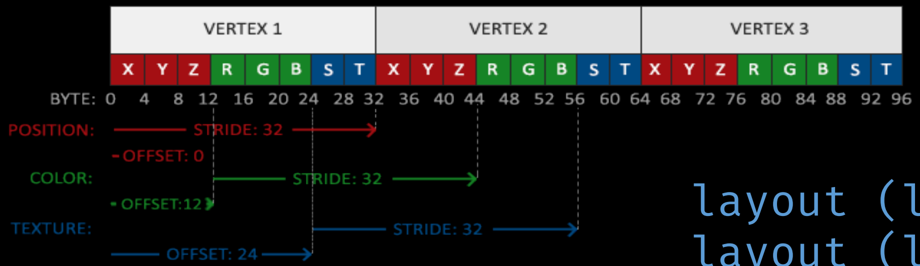
    ```
    vec2 someVec; // let it be [1, 2] for argument
    vec4 otherVec = someVec.yxxy; // [2, 1, 1, 2]
    vec3 nextVec = otherVec.wyz; // [2, 1, 1]
    vec2 lastVec = nextVec.zx + someVec.yy; // [3, 4]
    ```

- Accessing matrix components

  ```
  someMatrix[col][row] // single element
  someMatrix[col] // entire column vector
  ```

Vertex Shader - Intro

- Input and output of shader
  - GLSL keywords `in` (input) and `out` (output)
  - Shader gets input directly from vertex data
  - Keyword `layout` to define vertex data organization




```
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aColor;
layout (location = 2) in vec3 aTexCoord;
```

http://www.learnopengl.com/#!Getting-started/Textures

Shaders – Uniforms

- Uniform
  - Another way to pass data to shaders
  - Global + unique
  - Used by any shader at any stage in shader program
  - Will keep values until reset or updated
  - Use `glGetUniformLocation()` to get uniform location
  - Use `glUniform()` to update value
- In practice, use a shader class for simplicity
  - See OpenGL_tutorial_II/Headers/shader.hpp

Vertex Shader

- Recall pipeline for coordinate transformation

$$\mathrm{V_{clip}} = \mathrm{M_{projection}} * \mathrm{M_{view}} * \mathrm{M_{model}} * \mathrm{V_{local}}$$

- Resulting vertices get assigned to `gl_Position`
- OpenGL automatically performs
  - Perspective division
  - Clipping

Vertex Shader

- OpenGL_tutorial_II/Shaders/shader.vert

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec2 aTexCoord;

out vec2 TexCoord;

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

void main()
{
    gl_Position = projection * view * model * vec4(aPos, 1.0);
    TexCoord = vec2(aTexCoord.x, aTexCoord.y);
}
```

Fragment Shader

- Communicate with vertex shader
  - Get `TexCoord` from vertex shader
- Define texture
  - Built-in type sampler2D/sampler3D to pass texture
  - Built-in: `FragColor = texture(texSampler, texCoord)`
  - Built-in: `mix(FragColor1, FragColor2, mixValue)`
- In source code, bind texture before using `glDrawElement()`

Fragment Shader

- OpenGL_tutorial_II/Shaders/shader.frag

```
#version 330 core
out vec4 FragColor;

in vec2 TexCoord;

// texture samplers
uniform sampler2D texture1;
uniform sampler2D texture2;

void main()
{
    // linearly interpolate between both textures (80% container, 20% awesomeface)
    FragColor = mix(texture(texture1, TexCoord), texture(texture2, TexCoord), 0.2);
}
```

Part 3: Textures

Texture Data

- Define vertex data
  - 3 position coordinates + 2 texture coordinates

```
float vertices[] = {
    -0.5f, -0.5f,  0.5f,  0.0f, 0.0f,
     0.5f, -0.5f,  0.5f,  1.0f, 0.0f,
     0.5f,  0.5f,  0.5f,  1.0f, 1.0f,
    …
}
```

- 2 attribute pointers to position + texture attributes

```
// position attribute
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 5 * sizeof(float), (void*)0);
glEnableVertexAttribArray(0);
// texture coord attribute
glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, 5 * sizeof(float), (void*)(3 * sizeof(float)));
glEnableVertexAttribArray(1);
```

Load + Create Texture

- Generate texture object

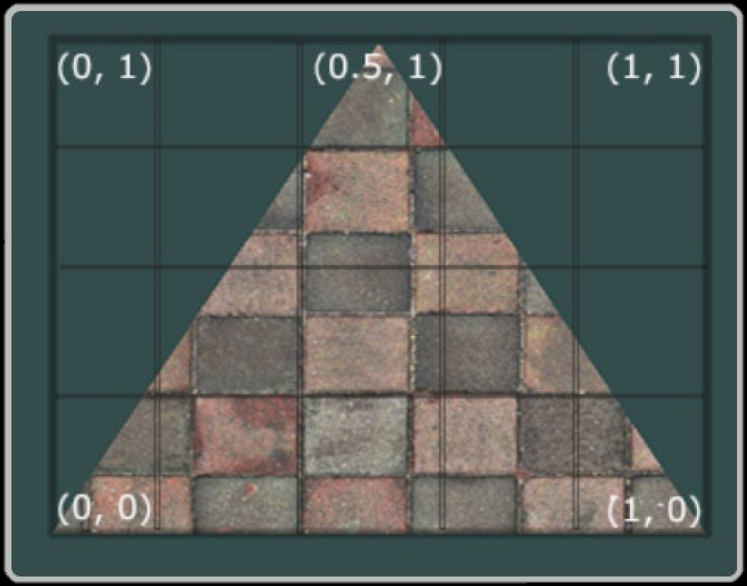
```
unsigned int textureID;
glGenTextures(1, &textureID);
glBindTexture(GL_TEXTURE_2D, textureID);
```

- Load image
  - Use stb_image.h to load images

```
int w, h, chans;
unsigned char *data = stbi_load(“pic.jpg”, &w, &h, &chans, 0);
```

- Set wrapping + filtering options

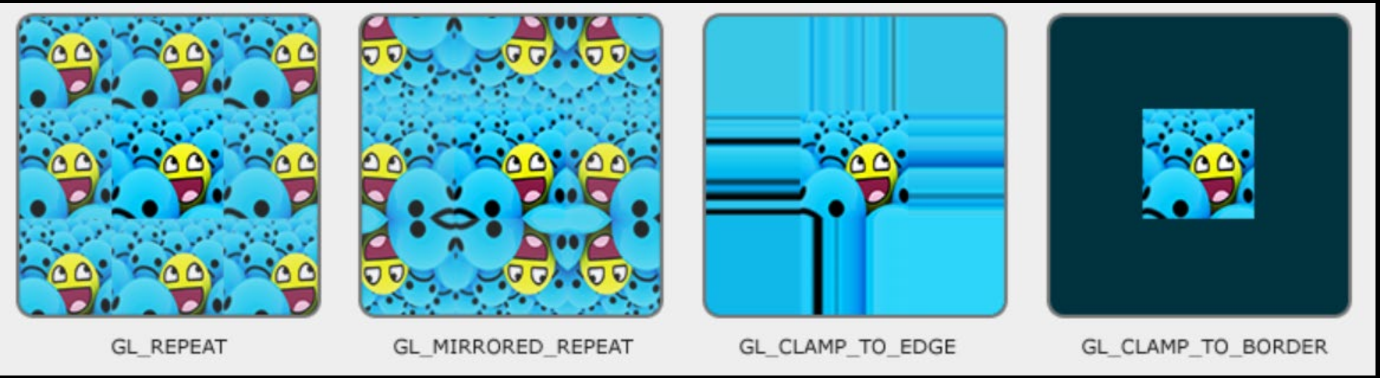
Texture Wrapping



- Texture coordinates
  - On range [0.0, 1.0]
  - Outside of range?
- Texture wrapping options:
  - `GL_REPEAT`: repeat texture image
  - `GL_MIRRORED_REPEAT`: repeat but mirror on repeat
  - `GL_CLAMP_TO_EDGE`: clamps coordinates to [0.0, 1.0]
  - `GL_CLAMP_TO_BORDER`: outside of range = solid color

Texture Wrapping

- Examples:



- Set texture wrapping option
  - Can be set per coordinate axis
  - (s, t, r) is texture equivalent to position (x, y, z)
  - Sometime (s, t) is referred to as (u, v)

Texture Wrapping

- Set texture parameters with glTexParameter*

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

  - i: stands for integer parameter
- If `GL_CLAMP_TO_BORDER`, need to specify border color

```
float borderColor[] = {0.8f, 0.5f, 1.0f, 1.0f};
glTexParameterfv(GL_TEXTURE_2D, GL_TEXTURE_BORDER_COLOR, borderColor);
```

  - fv: stands for float vector parameter
- For your skybox, try `GL_CLAMP_TO_EDGE`

Texture Filtering

- Figure out how to map texture pixels when given a floating-point value
- Two most important options:
  - `GL_NEAREST`: selects the texture pixel whose center is closest to the texture coordinate (default value)

  - `GL_LINEAR`: takes an interpolated value from the texture coordinate's neighboring texture pixel

Texture Filtering

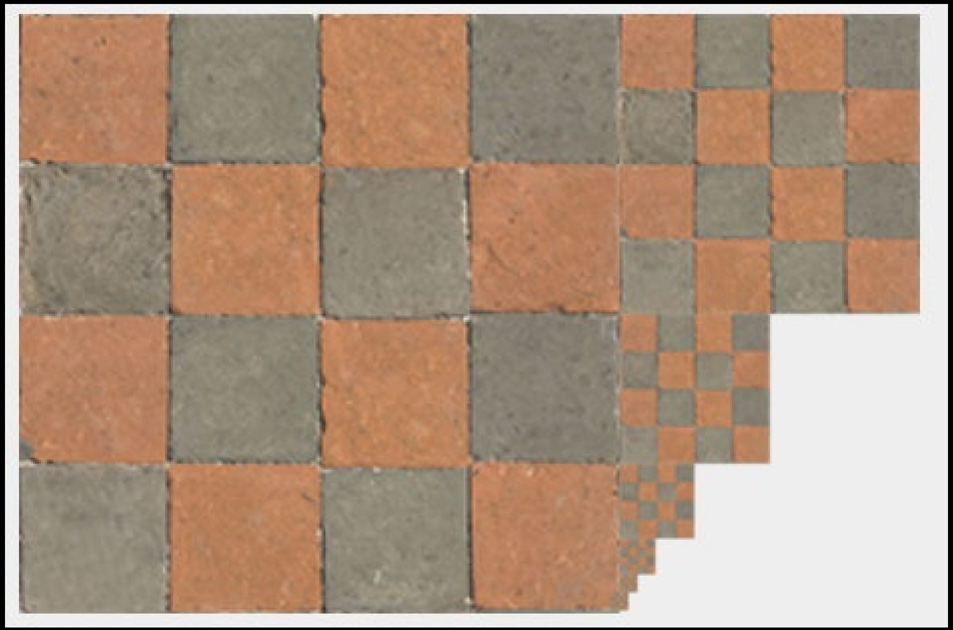
- Examples:

- Set texture filtering option
  - Options for both minifying or magnifying

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

Mipmaps

- Collection of texture images in different resolutions
- More efficient way of interpolating
- Solve issue of visible artifacts on small objects



Mipmaps

- Generate with OpenGL

```
glGenerateMipmap(GL_TEXTURE_2D);
```

- 4 more options to specify filtering method between mipmap levels:

```
GL_NEAREST_MIPMAP_NEAREST
GL_LINEAR_MIPMAP_NEAREST
GL_NEAREST_MIPMAP_LINEAR
GL_LINEAR_MIPMAP_LINEAR
```

Mipmaps

- Generate texture with glTexImage2D()

```
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height,
             0, GL_RGB, GL_UNSIGNED_BYTE, data);
```

  - Target texture
  - Mipmap level to create
  - Format to store the texture
  - Width of the texture image
  - Height of the texture image
  - Border (value should be 0)
  - Format of the source image
  - Data type of source image
  - Source image data

loadTexture()

```
unsigned int loadTexture(char const * path)
{
    unsigned int textureID;
    glGenTextures(1, &textureID);

    int width, height, nrComponents;
    unsigned char *data = stbi_load(path, &width, &height, &nrComponents, 0);
    if (data)
    {
        GLenum format;
        if (nrComponents == 1)
            format = GL_RED;
        else if (nrComponents == 3)
            format = GL_RGB;
        else if (nrComponents == 4)
            format = GL_RGBA;

        glBindTexture(GL_TEXTURE_2D, textureID);
        glTexImage2D(GL_TEXTURE_2D, 0, format, width, height, 0, format, GL_UNSIGNED_BYTE, data);
        glGenerateMipmap(GL_TEXTURE_2D);

        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_MIPMAP_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

        stbi_image_free(data);
    }
    else
    {
        std::cout << "Texture failed to load at path: " << path << std::endl;
        stbi_image_free(data);
    }

    return textureID;
}
```

Setting up Textures in Code

- Set up the texture for fragment shader

```
ourShader.use();
ourShader.setInt("texture1", 0);
ourShader.setInt("texture2", 0);
```

- Activate texture unit for each drawing call

```
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, texture1);
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, texture2);
```

A Simple Texture Trick!

- OpenGL texture coordinates != image coordinates
- OpenGL expects 0.0 on y-axis to be bottom
- Images usually have 0.0 on y-axis at top
- stb_image.h can flip the y-axis during image load

```
stbi_set_flip_vertically_on_load(true);
```

Part 4: Wrapping Up

Near Future

- Heightmap intro/tutorial
  - Later this week, ideally Wednesday, Sept. 14
- Extra office hours
  - TBD, looking like Wednesday, Sept. 14

References

- Coordinate Systems
  - https://learnopengl.com/Getting-started/Transformations
  - https://learnopengl.com/Getting-started/Coordinate-Systems
  - https://learnopengl.com/Getting-started/Camera
- Shaders
  - https://learnopengl.com/Getting-started/Shaders
  - https://www.khronos.org/opengl/wiki/Vertex_Shader
  - https://www.khronos.org/opengl/wiki/Fragment_Shader
- Textures
  - https://learnopengl.com/Getting-started/Textures
  - https://www.learnopengles.com/tag/mipmap/

Acknowledgements

- Thanks to https://learnopengl.com/ for providing fantastic figures and tutorials for beginners
- Thanks to https://github.com/Polytonic/Glitter for providing the outline for the tutorial program
- Thanks to Shimian Zhang for making previous iterations of this presentation