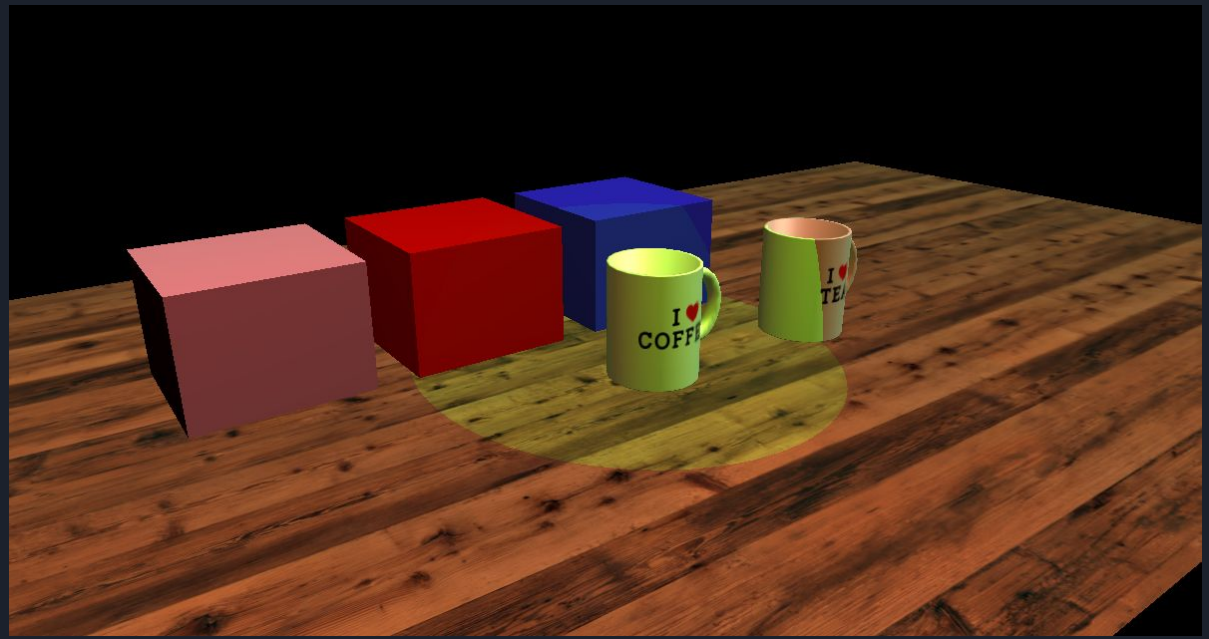
Basic shader

Homework 2 - 2022 Computer Graphics

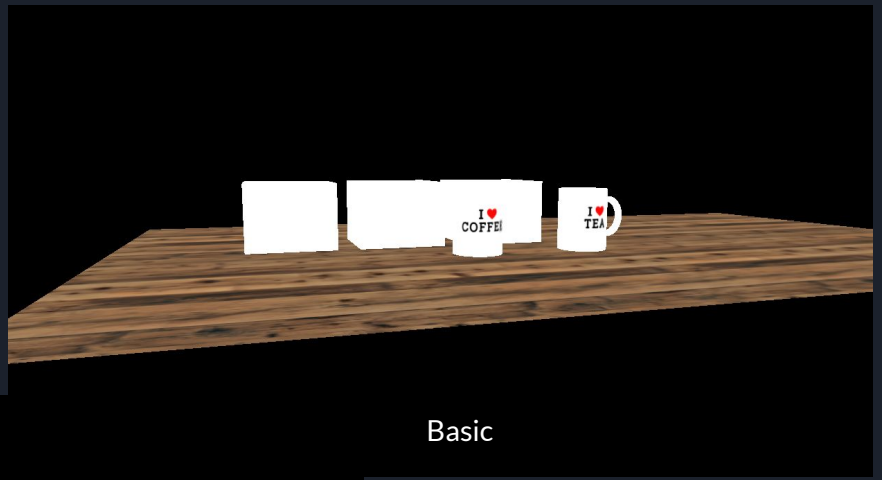
Lighting & Shading

- In this assignment, you are going to write a program based on the provided template that implementes several shader effect on different texture with GLSL



Shaders

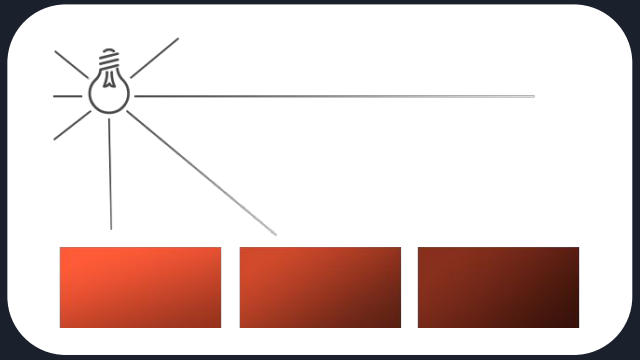
Example



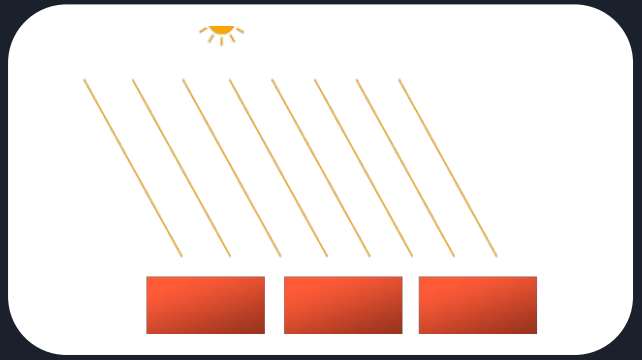
Basic

Light

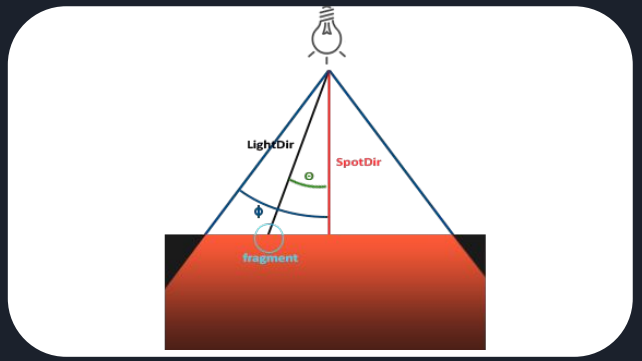
Lighting



point light

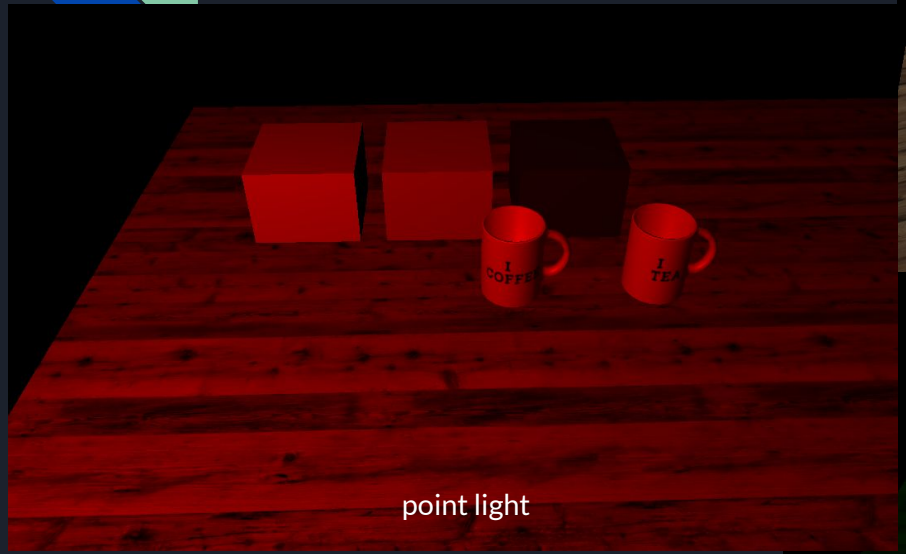


directional light

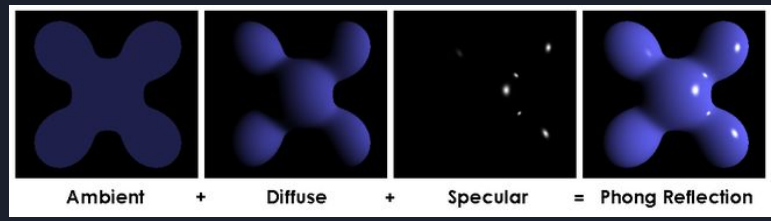


spot light

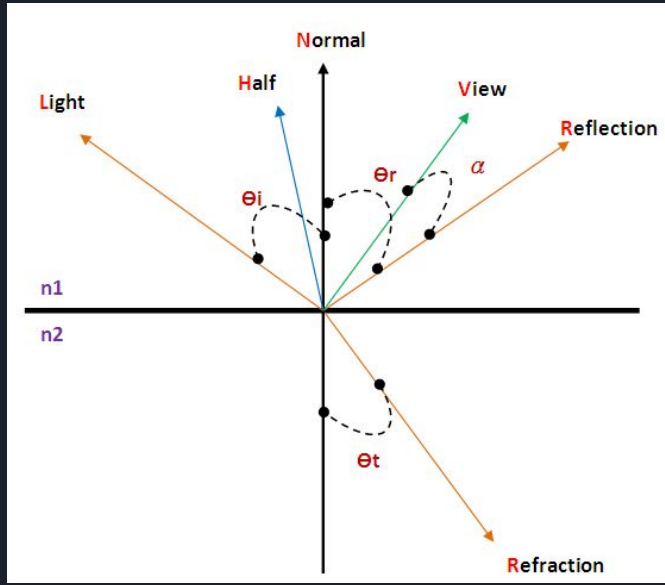
Lighting



Phong shading



- For direct light, lighting = ambient + diffuse + specular
- For point light & spot light, lighting = attenuation * (ambient + diffuse + specular)
- attenuation
  - No attenuation for directional light
  - for point light and spot light, follow distance attenuation formula 1/(a + bd +cd^2)
- Only show ambient color if the object is out of the spot light's cutoff angle
- ambient = La * Ka
- diffuse = Ld * Kd * (N · L)
- specular = Ls * Ks * (V · R) ~= Ks * (N · H) ^n
- La = Ld = Ls for each light
- Ka, Kd, Ks and n are defined in Material class for each scene object

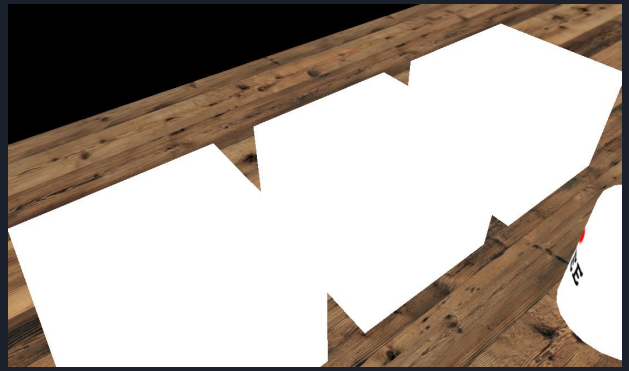


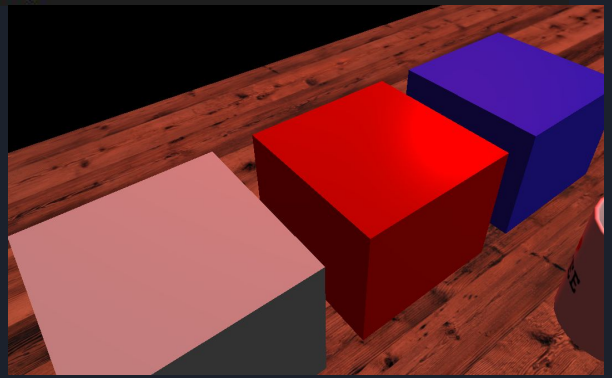
Material

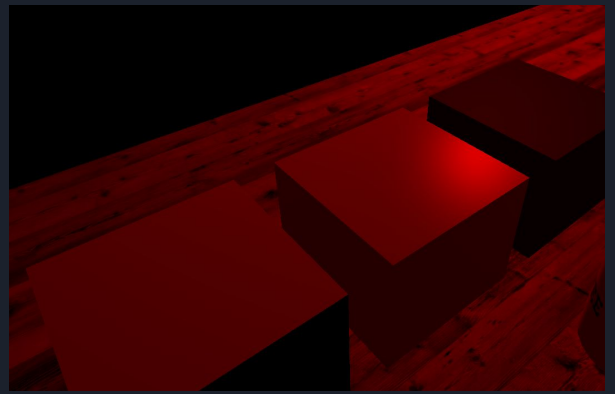
```
mFlatwhite.ambient = glm::vec3(0.0f, 0.0f, 0.0f);
mFlatwhite.diffuse = glm::vec3(1.0f, 1.0f, 1.0f);
mFlatwhite.specular = glm::vec3(0.0f, 0.0f, 0.0f);
mFlatwhite.shininess = 10;

mShinyred.ambient = glm::vec3(0.1985f, 0.0000f, 0.0000f);
mShinyred.diffuse = glm::vec3(0.5921f, 0.0167f, 0.0000f);
mShinyred.specular = glm::vec3(0.5973f, 0.2083f, 0.2083f);
mShinyred.shininess = 100.0f;

mClearblue.ambient = glm::vec3(0.0394f, 0.0394f, 0.3300f);
mClearblue.diffuse = glm::vec3(0.1420f, 0.1420f, 0.9500f);
mClearblue.specular = glm::vec3(0.1420f, 0.1420f, 0.9500f);
mClearblue.shininess = 10;
```







OBJ File

- OBJ file is a file format to store 3D model
  - Define a list of positions, normals and texture coordinates
  - List index of position, normal and texture coordinates for each vertex to define polygons (or faces)
- You need to implement OBJ file parser
  - No w parameter for v type element
  - Only v, vt, vn and f elements
  - All face are triangle (three vertex per face)
  - all face vertex have position, normal and texture coordinate index

Control

- All control are implemented in templates
  - 1: switch to example shader program
  - 2: switch to basic shader program
  - 3: switch to light shader program
  - 4: enable/disable direction light
  - 5: enable/disable point light
  - 6: enable/disable spot light
  - w/a/s/d, cursor: control camera position and rotation
  - k/l: change point light position
  - h/j: change point light color
  - i/o: change spot light position
  - y/u: change spot light color

Spec

- All light parameter are defined in context.h
- Model material are defined in model.h and main.cpp

```
ublic:
// Parameter of lights use in light shader (TODO#4)
int directionLightEnable = 1;
glm::vec3 directionLightDirection = glm::vec3(-0.3f, -0.5f, -0.2f);
glm::vec3 directionLightColor = glm::vec3(0.6f, 0.6f, 0.6f);

int pointLightEnable = 0;
glm::vec3 pointLightPosition = glm::vec3(6, 3, 0);
glm::vec3 pointLightColor = glm::vec3(1.0f, 0.0f, 0.0f);
float pointLightConstant = 0.9;
float pointLightLinear = 0.027;
float pointLightQuardratic = 0.0128;
float _pointLightPosisionDegree = 0.0f;

int spotLightEnable = 0;
glm::vec3 spotLightPosition = glm::vec3(0, 4, 5);
glm::vec3 spotLightDirection = glm::vec3(1.0f, 1.0f, 1.0f);
glm::vec3 spotLightColor = glm::vec3(0.0f, 0.8f, 0.0f);
float spotLightCutOff = glm::cos(glm::radians(12.5f));
float spotLightConstant = 0.9;
float spotLightLinear = 0.014;
float spotLightQuardratic = 0.007;
```

```
40    mShinyred.ambient = glm::vec3(0.1985f, 0.0000f, 0.0000f);
41    mShinyred.diffuse = glm::vec3(0.5921f, 0.0167f, 0.0000f);
42    mShinyred.specular = glm::vec3(0.5973f, 0.2083f, 0.2083f);
43    mShinyred.shininess = 100.0f;
```

Plane

- Size 8.192 * 5.12
- Center position (4.096, 0, 2.56)
- Normal (0, 1, 0)
- Note: texture is map to size 4.096 * 2.56
  - Extend texture by correctly set texcoords

Spec

- Implementation (85%)
  - Task#0 Change window title to “HW2 - `student id`” (0%)
    - -10% if title is wrong,
  - Task#1 OBJ file parser (10%)
  - Task#2 Basic shader program (20%)
    - Render objects with texture
  - Task#3 Display texture plane (10%)
    - Draw obj (5%) (positions, normals, texcoords…)
    - Texture (5%) (assets/models/Wood_maps/AT_Wood.jpg)
  - Task#4 Light shader program (three light source mixed)
    - Using phong shading
      - Directional light (15%)
      - Point light (15%)
      - Spotlight (15%)

Spec

- Report(15%)
  - Implementation(HOW & WHY)
  - Problems you encountered
  - Don’t paste code without any explaination
  - No format restriction
  - File name: report_<your student ID> .pdf
- Bonus(10%)
  - Ex: shadow calculation and other shader technique…
  - Other creativity
  - Enable bonus by click any keys
  - Mention what you did in report

Hint

- You can click ctrl+shift+F to search all TODOs in template
- Read the TODOs in the template and follow TODOs order
- Read notes to get more hints & ideas
- Before you ask question on E3, make sure you have Googled it
- Feel free to report bugs if you find one. :)

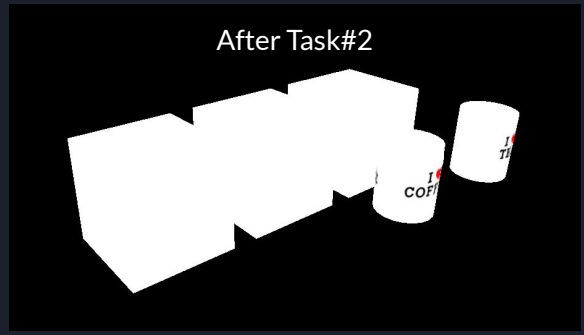
Files need implmenet

src/main.cpp
src/model.cpp
src/Programs/basic.cpp
src/Programs/light.cpp
assets/shaders/basic.vert
assets/shaders/basic.frag
assets/shaders/light.vert
assets/shaders/light.frag

Before all

After Task#1

After Task#2



After Task#3

After Task#4

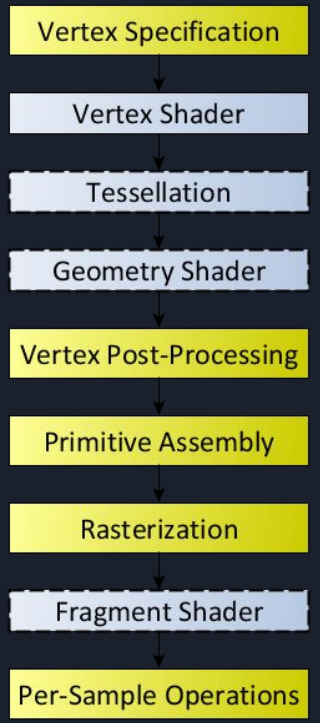
Notes

- Deadline: 11/14 23:59
  - You need to upload hw2_<your student ID>.zip and report_<your student ID> .pdf respectively
  - hw2_<your student ID>.zip (root)
    - src
    - include
    - assets/shaders
  - If you add or remove any files, you need to mention in report
  - You can use script/pack.ps1 (PowerShell) or script/pack.sh (Bash)
  - Incorrect submission will -5 points
- No plagiarism, -10 points per day after deadline
- No demo required this time
- No TA office hour
- HW 3 will be anounced at 11/15
- Ask homework problem in E3 forum
- For personal problem, you please email to all three TAs from E3

```
include/
include/camera.h
include/context.h
include/gl_helper.h
include/model.h
include/opengl_context.h
include/program.h
include/utils.h
src/
src/camera.cpp
src/CMakeLists.txt
src/gl_helper.cpp
src/main.cpp
src/model.cpp
src/opengl_context.cpp
src/Programs/
src/Programs/basic.cpp
src/Programs/example.cpp
src/Programs/light.cpp
assets/shaders/
assets/shaders/basic.frag
assets/shaders/basic.vert
assets/shaders/example.frag
assets/shaders/example.vert
assets/shaders/light.frag
assets/shaders/light.vert
```

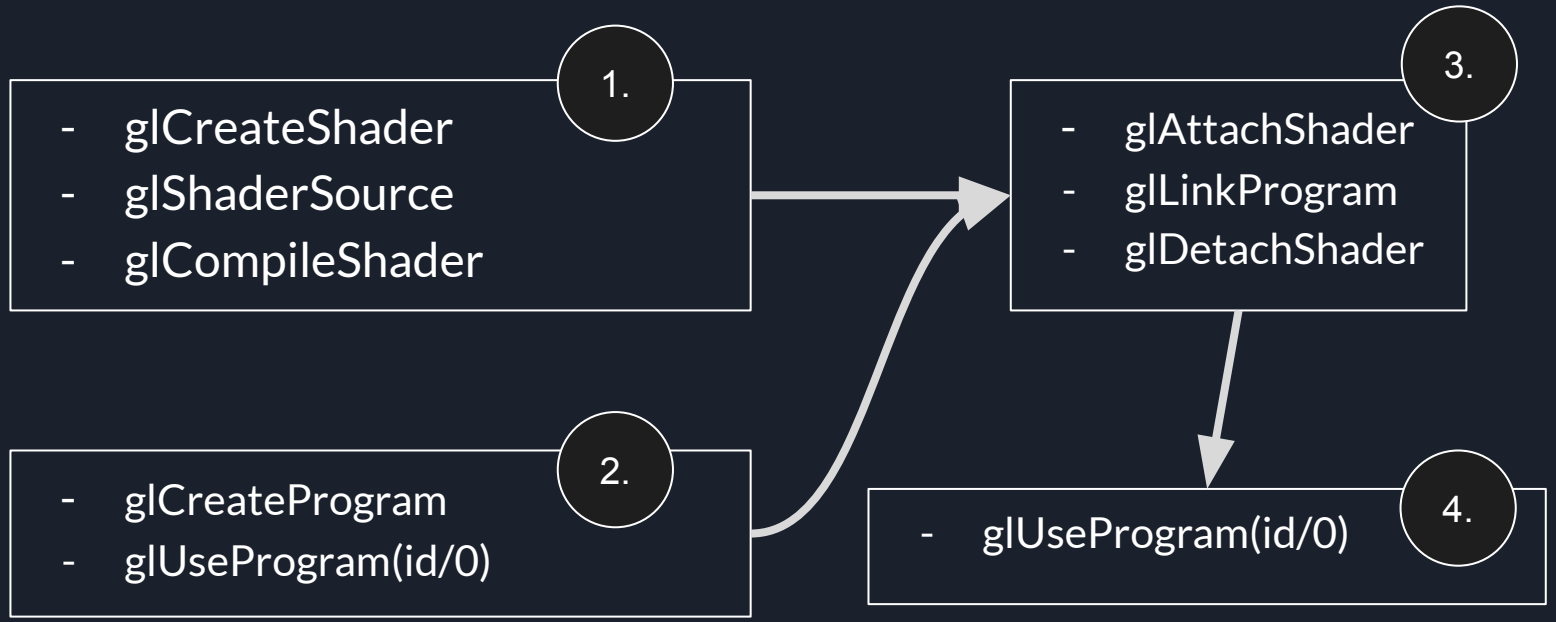
Appendix: Shader Programming in OpenGL

- C-like language
- Pure text
- How to connect shaders?
- How to pass parameters?
  - We will focus on vertex shader & fragment shader

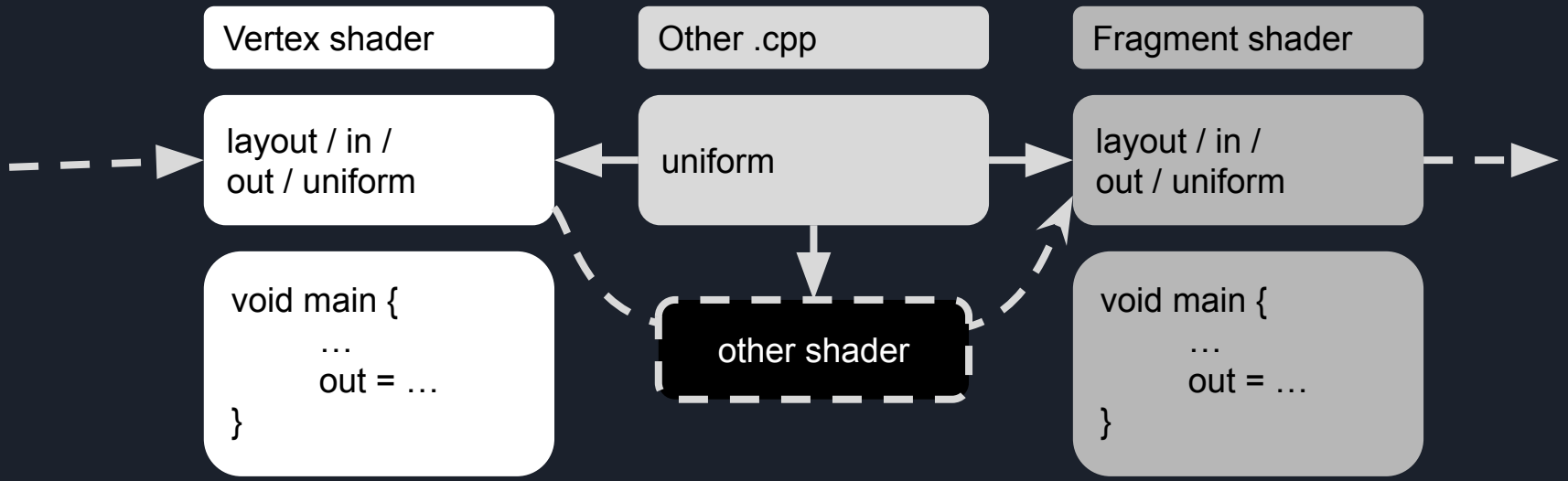


https://www.khronos.org/opengl/wiki/Rendering_Pipeline_Overview

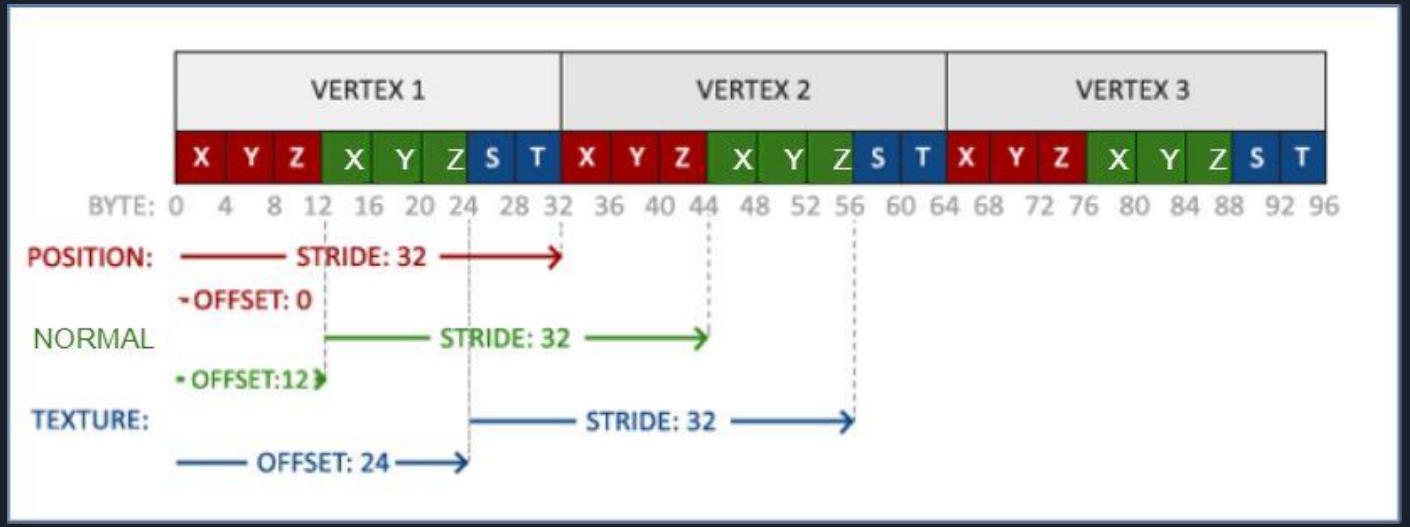
Appendix: Shader Programming in OpenGL



Appendix: Shader Programming in OpenGL



Appendix: Shader Programming in OpenGL



- We use (x, y, z, nx, ny, nz, u, v)

Appendix: Shader Programming in OpenGL

- Shader file
  - C-like → basic statements can be used
  - data type: int, float, vec[2,3,4], mat[2,3,4]fv, struct …
    - basic vector operators provided
    - vector access elements with ‘.’, ex: v.xyz → a vec3
  - useful function: normalize, length, dot, pow, min, max …
  - Similar to GLM’s API

Appendix: Shader

- Vertex shader
  - Input: per vertex data (anything, usually position, normal ...)
  - Output: vertex position(clip space)
- Fragment shader
  - Input: per pixel data (interpolated between vertices)
  - Output: pixel color

Appendix: Buffer

- Pass data to GPU
- Array buffer
  - Any data usually vertex data for vertex shader
- Element array buffer
  - Store rander order
- Uniform buffer
  - Store blocks of uniforms
- You can check comments in
  - src/buffer/buffer.cpp
  - include/buffer/buffer.h

Appendix: Buffer example

- A array buffer storing {-1, 1, 0, -1, -1, 0, 1, -1, 0, 1, 1, 0} = A square
- We can use glDrawElements with an element array buffer

{0, 1, 2, 2, 1, 0} using GL_TRIANGLES, it should be like:

```
<T> vtx =  {0, 1, 2, 2, 1, 0};
GLuint vboName;
glGenBuffer(GL_ARRAY_BUFFER, vboName);
glBindBuffer(GL_ARRAY_BUFFER, sizeof(<T>) * vtx.length, vtx, GL_STATIC_DRAW);
```

Appendix: GLSL - Uniforms

- glUniform[Matrix][1,2,3,4][i,f,v]
  - Pass data from RAM(CPU) to VRAM(GPU)
  - Good for small data. (A few bytes)
- glGetUniformLocation
  - How to find where is the uniform?
    - uniform int isCube; (In your shader)
    - GLint loc = glGetUniformLocation(<your shader handle>, isCube);
  - Then you can call glUniform with location = loc to set value.

Reference

- E3
  - shading
  - textureMapping
- https://learnopengl.com/Lighting/Light-casters
- https://www.khronos.org/opengl/wiki/Rendering_Pipeline_Overview
- https://learnopengl.com/Getting-started/Shaders