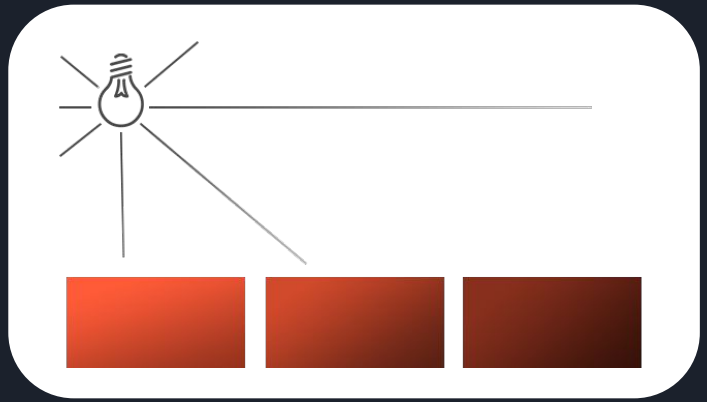
CG 2021 HW2

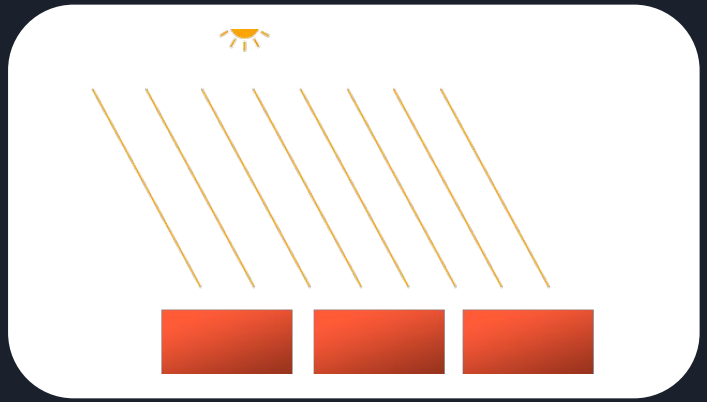
Lighting & Shading

- 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

Lighting



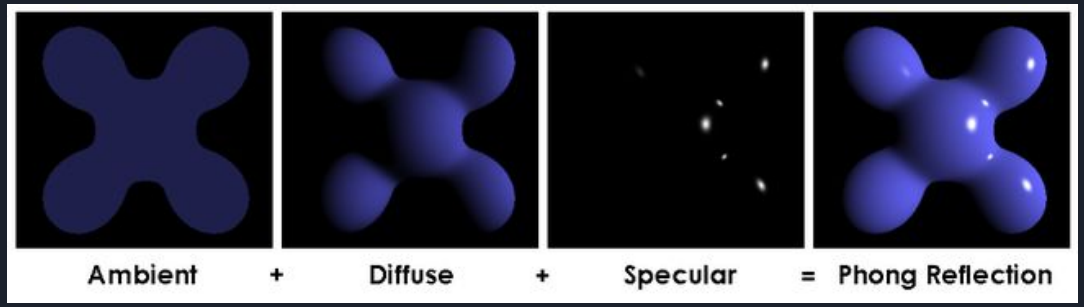
point light

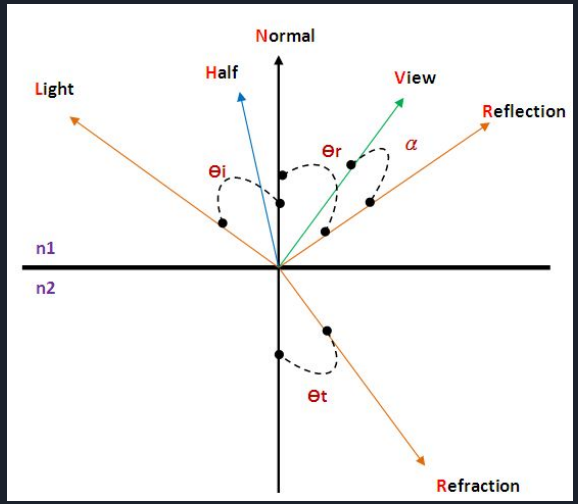


directional light

Lighting

- N: surface normal
- L: light direction
- H: half-way direction
- V: View direction
- R: reflection
- n: shininess



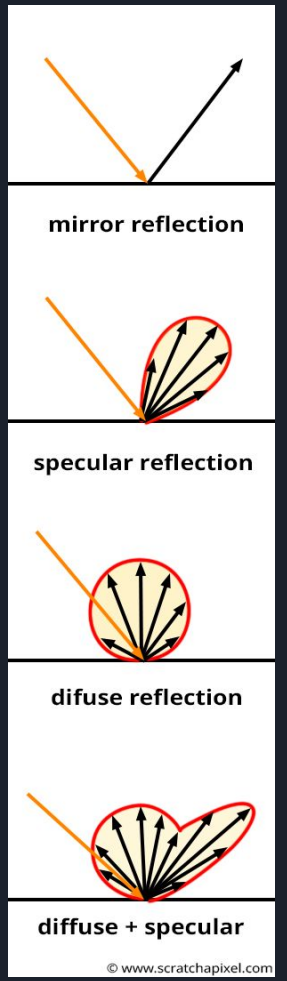


https://cg2010studio.files.wordpress.com/2011/03/20110327153245031.png

Lighting

- lighting = ambient + attenuation * shadow * (diffuse + specular)
- attenuation
  - for directional light, it is a constant
  - for point light, it is inversely proportional with the quadrant of the distance
- diffuse = Kd * (N · L)
- specular = Ks * (V · R) ~= Ks * (N · H) ^n

https://www.scratchapixel.com/lessons/3d-basic-rendering/phong-shader-BRDF



Spec

- Implementation (85%)
  - Camera 0%
  - Display 2 textures (10%)
    - Plane (5%)
      - texture: assets/texture/wood.jpg
    - Dice (5%)
      - 6 different faces
      - texture: assets/texture/[pos,neg][x,y,z].jpg

Spec

- Implementation (85%)
  - Shader switch with key-event (40%)
    - Gouraud shading (20%)
    - Blinn-Phong shading (20%)
  - Single light source switch with key-event (35%)
    - Directional light (11%)
    - Point light (11%)
    - Spotlight (13%)

Spec

- Implementation (85%)

Diffuse intensity(Kd)	0.75		
Specular intensity(Ks)	0.75	Shininess	8
Attenuation(for directional light just = 0.65)			
		Point light	Spotlight
constant		1.0	1.0
linear		0.027	0.014
quadratic		0.0028	0.007

Spec

- Report(15%)
  - Implementation(HOW & WHY)
  - Problems you encountered
  - Don’t paste code without any explaination
  - File name: report_<your student ID> .pdf
- Bonus(10%)
  - Ex: multiple light, shadow calculation
  - Other creativity

Hint

- Read the TODOs in the template
- Read comments to get more hints & ideas
- Before you ask question on E3, make sure you have Googled it
- If you have questions when you reading other part of the template code, you can ask it in forum too.
- Feel free to report bugs if you find one. :)

Notes

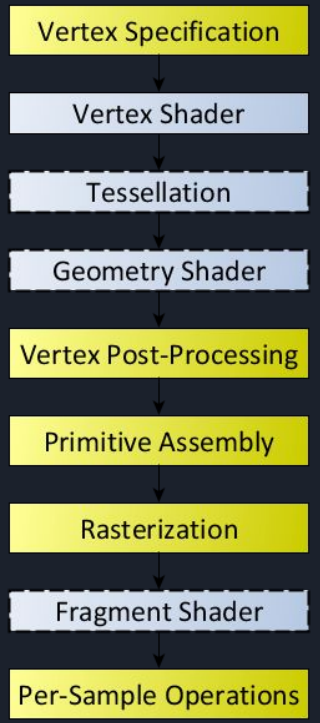
- Deadline: 11/15 23:59
  - You need to upload hw2_<your student ID>.zip and report_<your student ID> .pdf respectively
  - hw2_<your student ID>.zip (root)
    - src
    - include
  - You can use script/pack.ps1 (PowerShell) or script/pack.sh (Bash)
  - Incorrect submission will -5 points
- No plagiarism, -10 points per day after deadline
- No demo required this time
- HW 3 will be anounced at 11/16

Other notice

- Make sure you have find your teammates for Final Presentation
  - Form will be closed on 11/08 23:59
- Warning:
  - This homework is much more difficult than HW1

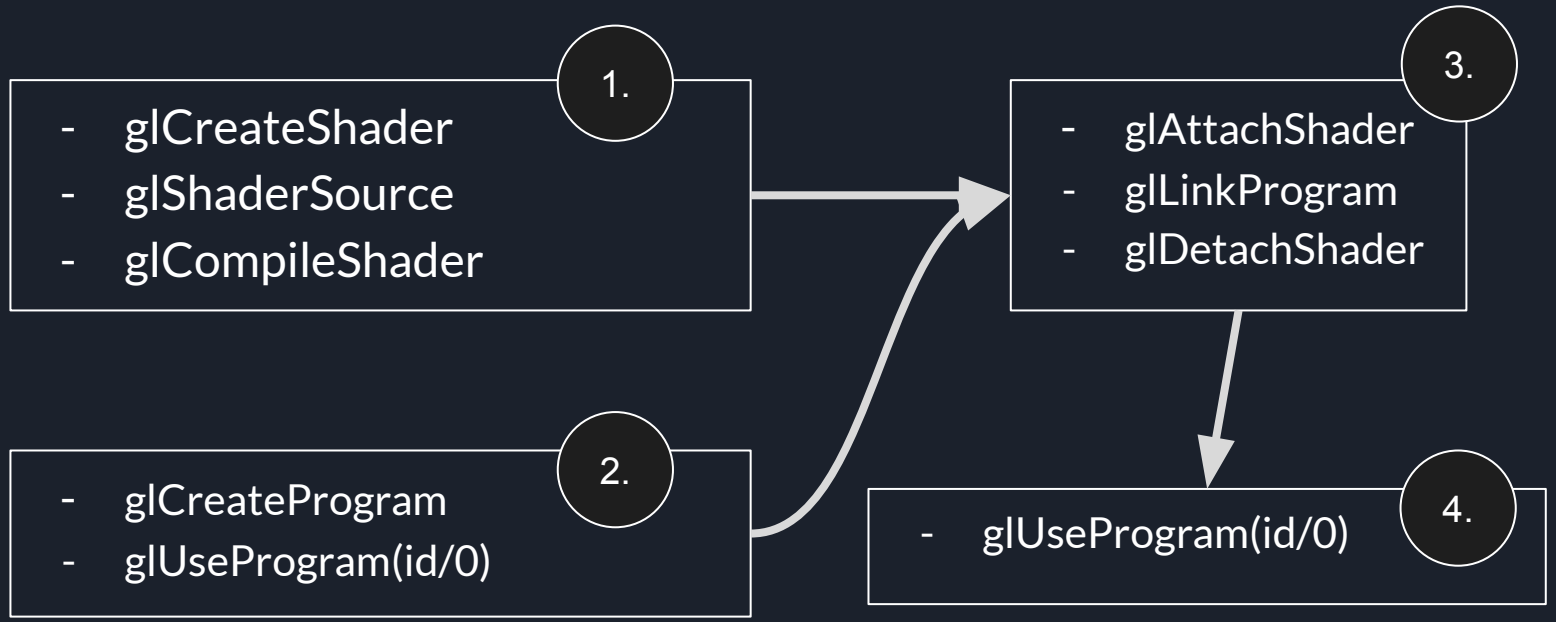
Appendix: Shader Programming in OpenGL

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

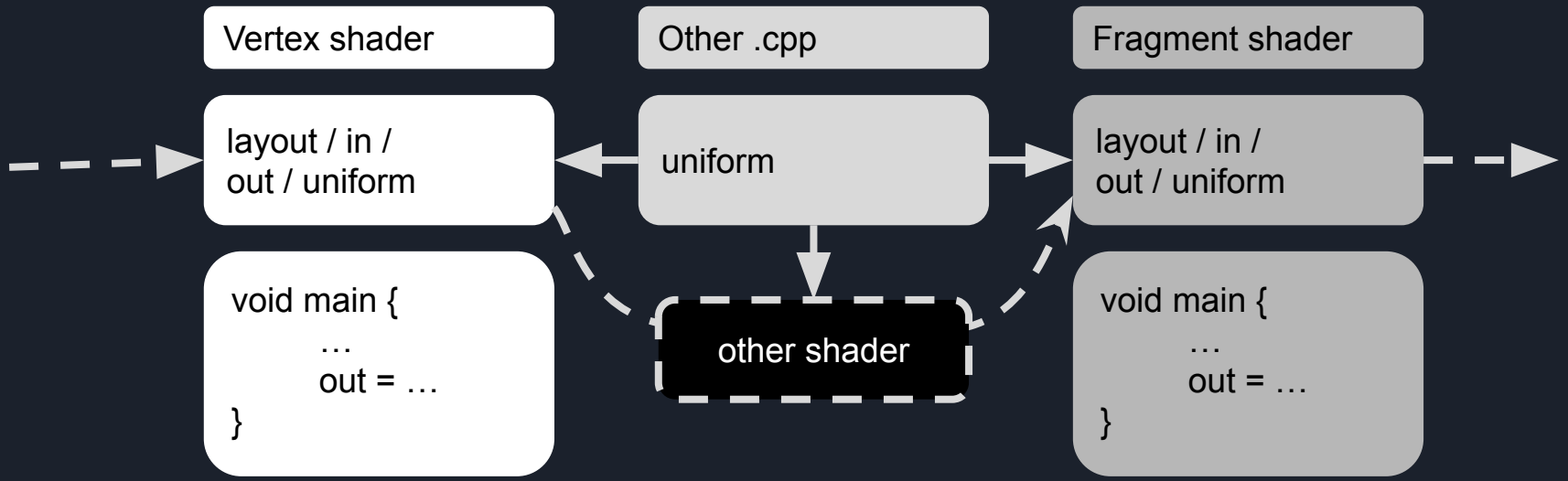


https://www.khronos.org/opengl/wiki/Rendering_Pipeline_Overview

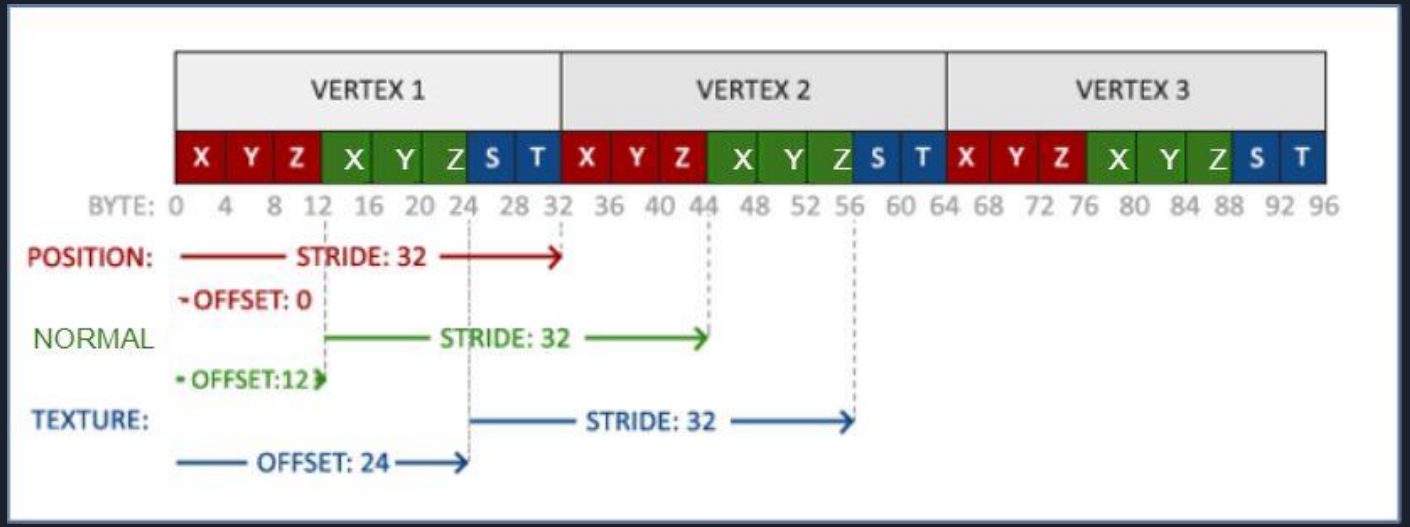
Appendix: Shader Programming in OpenGL



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);
```

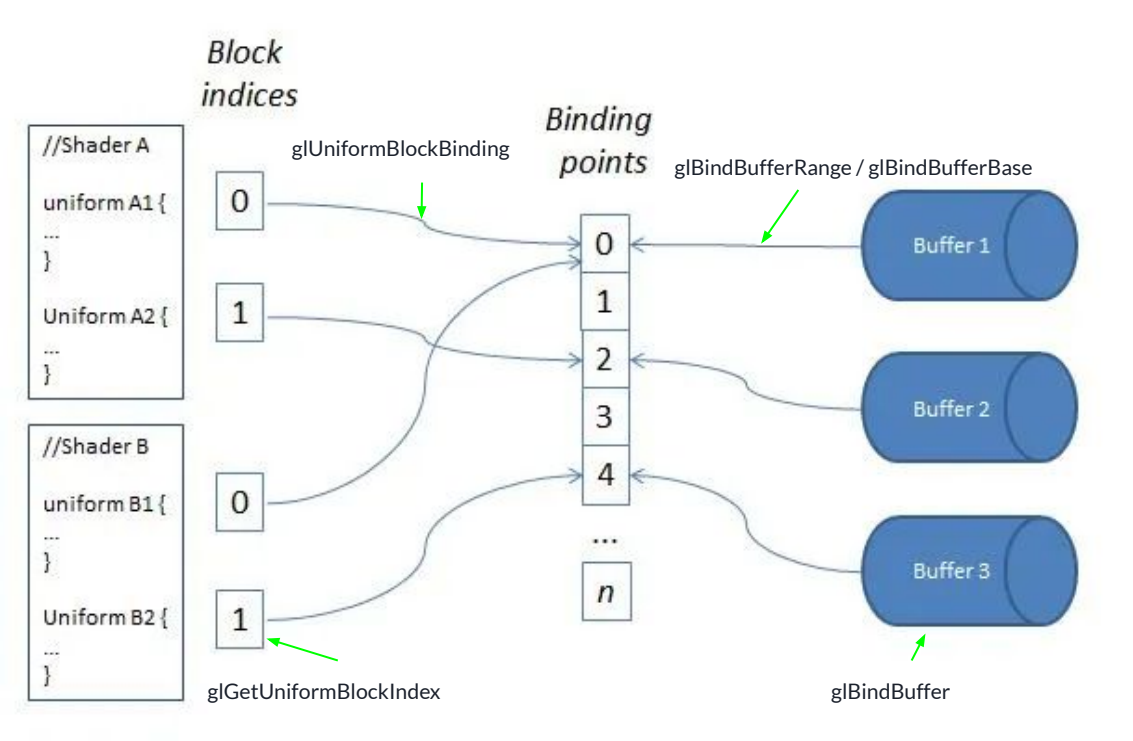
Appendix: GLSL - Uniforms

- 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.

Appendix: GLSL - Uniform Buffers

- glBufferData(allocate + store data), glBufferSubData(modify data)
  - Pass data from RAM(CPU) to buffer (RAM or VRAM)
  - Good for large data. (Need alignment, share between shaders)
- glBindBufferRange / glBindBufferBase
  - Bind [range / all] of the buffer to specific binding point
- glBindBuffer
  - Bind buffer to OpenGL context
- glUniformBlockBinding
  - Bind a shader uniform block index to specific binding point
- glGetUniformBlockIndex
  - Find a shader uniform block index

Appendix: GLSL - Uniform Buffers



https://www.lighthouse3d.com/tutorials/glsl-tutorial/uniform-blocks/

Reference

- E3
  - shading.ppt
  - textureMapping.ppt and textureMapping2.ppt
- https://learnopengl.com/Lighting/Light-casters
- https://www.khronos.org/opengl/wiki/Rendering_Pipeline_Overview
- https://learnopengl.com/Getting-started/Shaders