CS100433
Introduction to Modern OpenGL

Junqiao Zhao 赵君峤

Department of Computer Science and Technology

College of Electronics and Information Engineering

Tongji University

What is OpenGL?

- **Open G**raphics **L**ibrary that creates an interface to graphics hardware
  - https://www.khronos.org/opengl/
- Hardware and OS independent
  - Windows/Mac OS/Linux/FreeBSD C/C++/Java/python
- APIs for rendering 2D and 3D scene
- Low-level interface
- CAD, virtual reality, Games

SGI and GL

- Silicon Graphics (SGI) revolutionized the graphics workstation by implementing the pipeline in hardware (1982)
- To access the system, application programmers used a library called GL
- With GL, it was relatively simple to program three dimensional interactive applications

OpenGL

The success of GL lead to OpenGL (1992), a platform-independent API that was

- Easy to use
- Close enough to the hardware to get excellent performance
- Focus on rendering
- Omitted windowing and input to avoid window system dependencies

- At present OpenGL 4.6
  - https://www.khronos.org/registry/OpenGL/specs/gl/glspec46.core.pdf

OpenGL vs Others

- OpenGL vs DirectX
- OpenGL vs Vulkan
- OpenGL vs OpenCL

OpenGL Installation

- How to Install OpenGL
  - Proprietary: Nvidia, AMD, Intel etc.
  - Opensource: Mesa3D etc.
- Where is the OpenGL lib?
  - libGL.so
  - OpenGL32.lib
- Check your OpenGL version
  - glxinfo | grep “version”
  - E.g. OpenGL core profile version string: 3.3

OpenGL Installation

- Links with window system
  - GLX for X window systems
  - WGL for Windows
  - AGL for Macintosh
- Utilities
  - GLFW
    - sudo apt-get install glfw3-dev
  - GLEW/GLAD
    - sudo apt-get install libglew-dev libglew1.13
  - GLM
    - sudo apt-get libglm-dev

Compile setting with CMAKE

- Directories
  - src
  - build
  - include
  - shader
  - CMakeLists.txt

```
cmake_minimum_required (VERSION 3.0)
project(sample)
add_subdirectory(src)
```

Compile setting with CMAKE

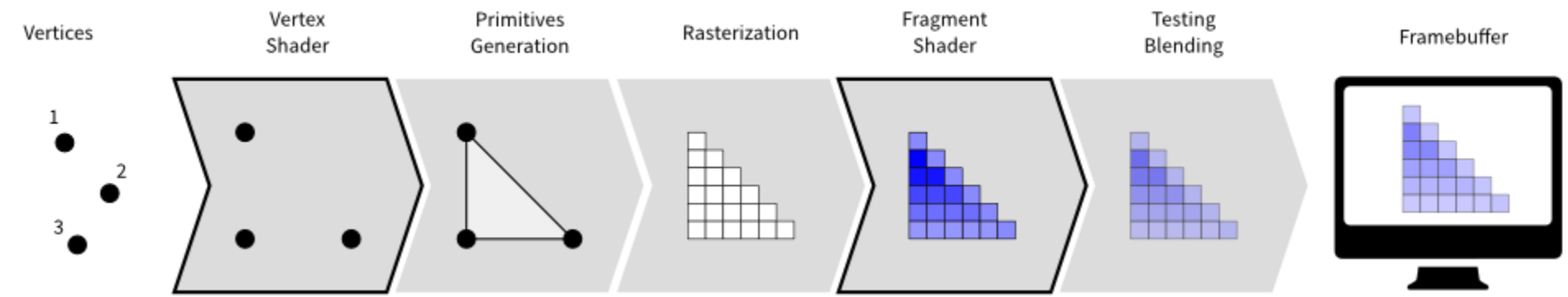
- Directories
  - src
    - CMakeLists.txt

```
cmake_minimum_required(VERSION 3.0)
project(sample)
find_package (OpenGL REQUIRED)
find_package (glfw3 REQUIRED)
find_package (GLEW REQUIRED STATIC)
include_directories(${CMAKE_SOURCE_DIR}/incl
ude}
add_library(libname libfile)
set(SOURCE_FILES main.cpp
__add_other_cpp_files_here__)
add_executable(sample ${SOURCE_FILES})
 target_link_libraries (sample libname
${OPENGL_LIBRARIES} ${GLFW3_LIBRARY}
${GLEW_LIBRARY} )
```

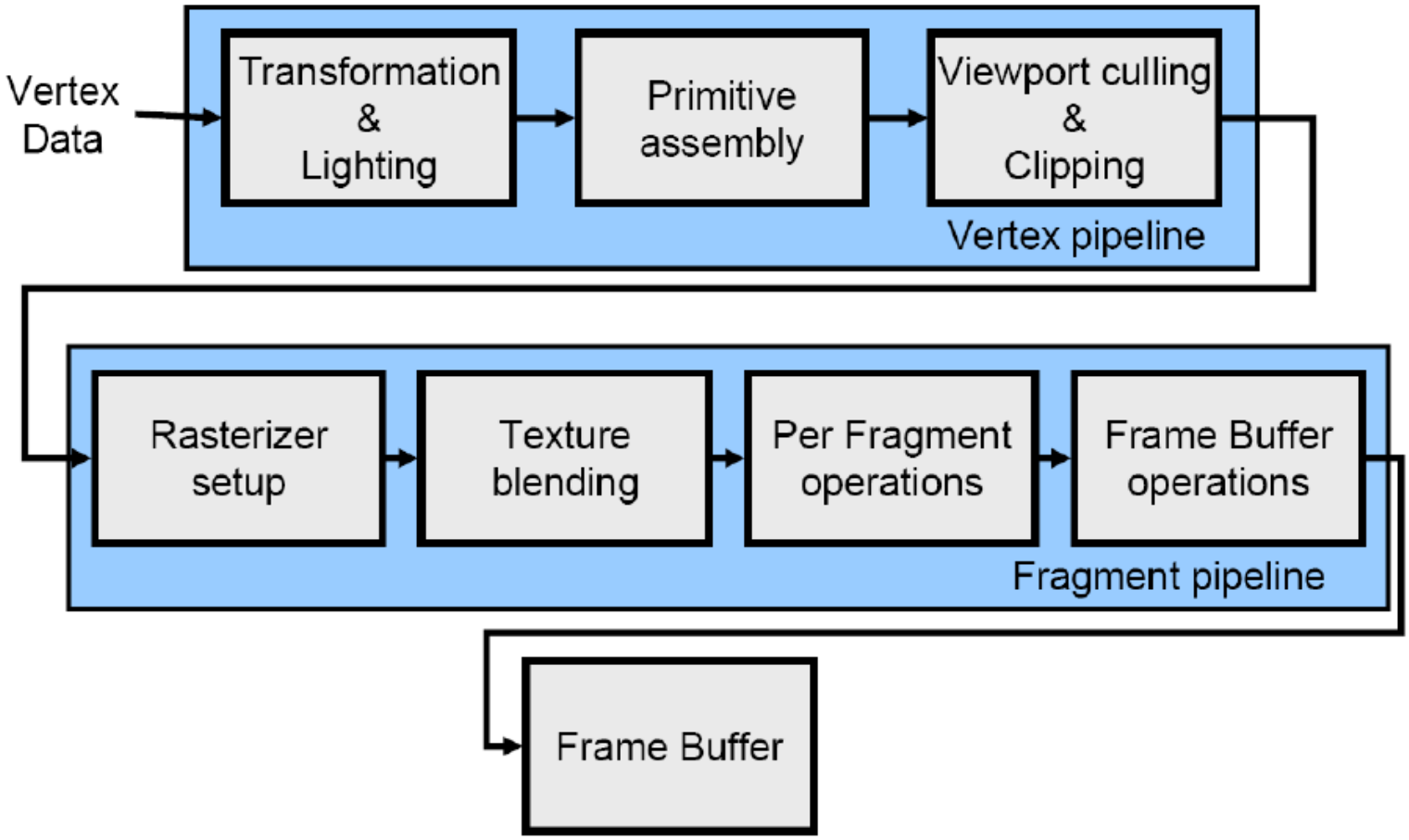
- Questions?

OpenGL Rendering Pipeline

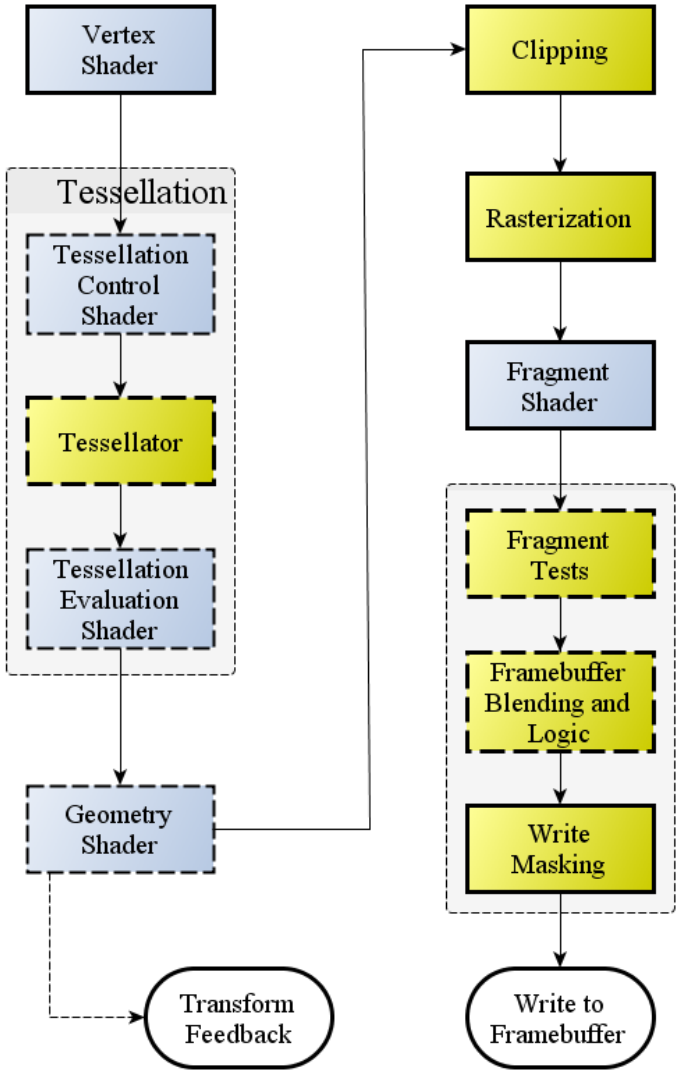
- Client-Server
- Immediate mode
- Core Profile
  - Vertex shader
  - Geometry shader
  - Fragment shader
  - Compute shader



OpenGL Rendering Pipeline (immediate mode)

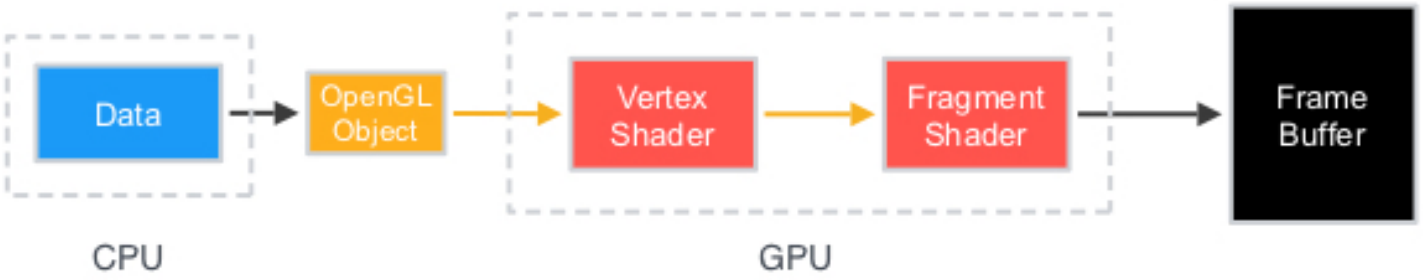


OpenGL Rendering Pipeline (Core profile)



OpenGL Objects

- Objects are containers for state (and data)
- Objects are responsible for transmitting state (and data) between CPU and GPU
- When they are bound to the context, the state that they contain is mapped into the context's state
  - **OpenGL is a state machine!**



OpenGL Objects

- Create and use an object have the same signature:
  - void **glGen***(GLsizei *n*, GLuint **objects*)
  - void **glBind***(GLenum *target*, GLuint *object*);
  - The name of an object is a Glunit
  - The target is the binding point, allows objects to be used for different purposes.
- Delete an object has the same signature:
  - void glDelete*(GLsizei *n*, const GLuint **objects*);
- Object types
  - Regular objects – contain data
    - E.g. Buffer Objects (VBO, EBO etc.), Texture Objects
  - Container objects – container for regular objects
    - E.g. Vertex Array Objects, Framebuffer Objects

```
float vertices[] = {// position color
        0.5f, -0.5f, 0.0f,  1.0f, 0.0f, 0.0f,  // bottom right
        -0.5f, -0.5f, 0.0f,  0.0f, 1.0f, 0.0f,  // bottom left
         0.0f,  0.5f, 0.0f,  0.0f, 0.0f, 1.0f   // top  };
unsigned int VBO;
glGenBuffers(1, &VBO); // Generate 1 vertex buffer object
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);

unsigned int VAO;
glGenVertexArrays(1, &VAO); // Generate 1 vertex array object
glBindVertexArray(VAO);
glBindBuffer(GL_ARRAY_BUFFER, VBO);
// position attribute
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)0);
// color attribute
glEnableVertexAttribArray(1);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)(3 *
sizeof(float)));
```

- Let’s move to the GPU part
- The Shader and Program Object

Shader program

- A small C-like program run on the GPU in parallel
  - Vertex shader controls vertex transformation
  - Fragment shader controls fragment shading
  - Geometry shader generate new primitives (Optional)
  - Tessellation shader subdivide meshes (Optional)

```
#version 330
layout(location = 0) in vec3 vp;
layout(location = 1) in vec3 vc;
out vec3 colour;
void main(){
    colour = vc;
    gl_Position = vec4(vp, 1.0);
}
```

A vertex shader

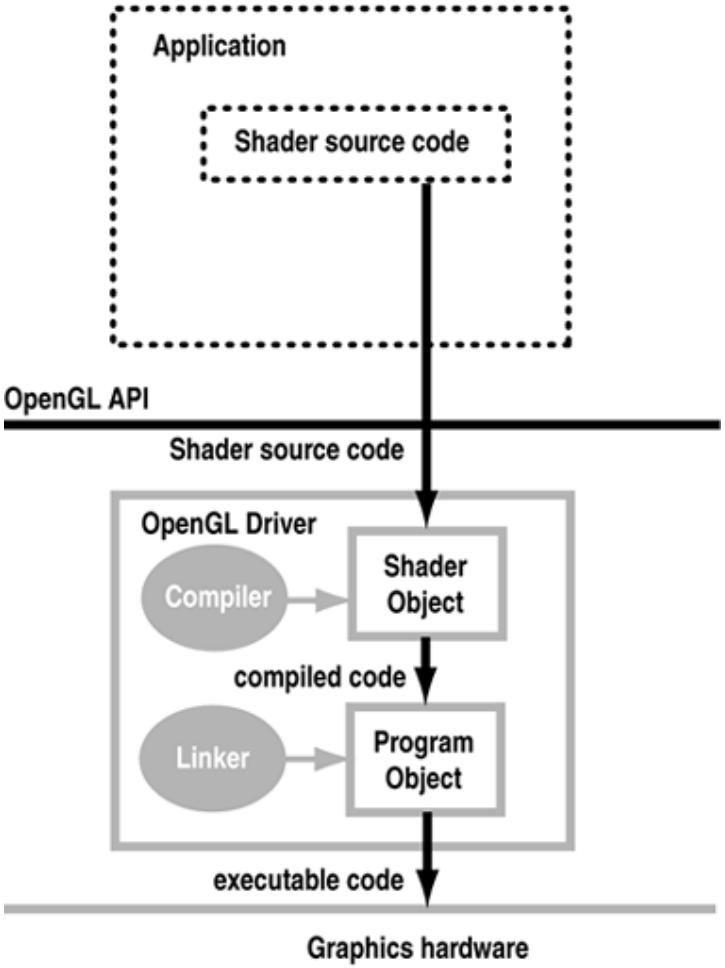
```
#version 330
in vec3 colour;
void main(){
    gl_FragColor = vec4(colour, 1.0);
}
```

A fragment shader

Shader program

- A collection of shaders that run together
  - At least one vertex shader or one fragment shader.
  - Should have both
- Need to be Compiled at run-time and Attached and Linked to a Program Object
- At any time, the GPU runs only one program
  - Must specify program to use before drawing
- Entry point = “void main( )”
  - Two main functions when writing a vertex shader and a fragment shader together

Shaders Execution Model



http://cse.csusb.edu/tongyu/courses/cs520/notes/glsl.php

Shading Language

- GLSL: OpenGL Shading Language
  - First introduced in OpenGL 2.0 (2004)
  - C-like
  - No recursion, No pointers
- Other shading languages
  - HLSL: High-Level Shading Language

GLSL

- Data types
  - C-data types
  - bvec*n,* vec*n ,* ivec*n ,* dvec*n*
  - mat*n*, mat*n*x*m*
  - sampler1D 2D cube
- Swizzling
- Structs
- Operations
  - C-like, dot, cross
- Built-in functions
  - math functions, graphics specific

Qualifiers

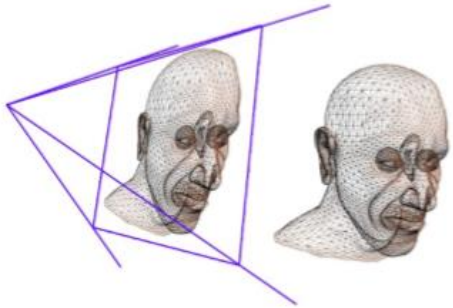
- GLSL has many of the same qualifiers such as **`const`** as C/C++
- Built in (OpenGL state variables)
  - **gl_***
- User defined (in application program)
  - **in/out**
    - Variables that are used to pass information between shaders
    - Can not be accessed in application
  - **uniform**
    - Variables that are constant for an entire primitive
    - Can be changed in application
    - Cannot be changed in shader
    - Used to pass information to shader
- Vertex attributes are interpolated by the rasterizer into fragment attributes

Example: Vertex Shader

```
#version 330
const vec4 red = vec4(1.0, 0.0, 0.0, 1.0);
in vec3 in_vertex
out vec3 color_out;
void main(void)
{
  gl_Position =
  gl_ProjectionMatrix*gl_ModelViewMartrix*gl_Vertex *vec4(in_vertex,
    1.0);
  color_out = red;
}
```

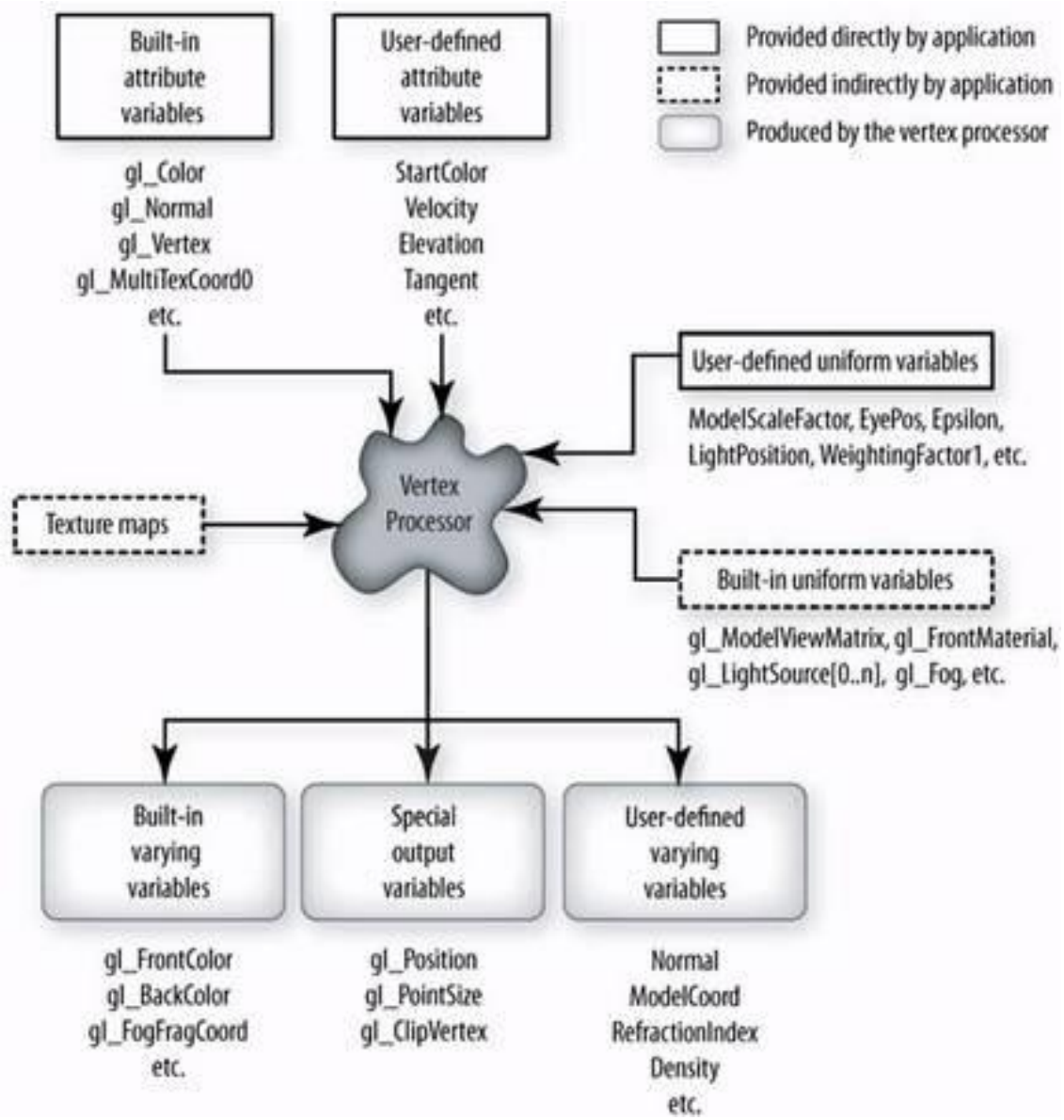
Vertex shader

- Transform vertices
  - Model, View and projection transformations
  - Custom transformation
    - Morphing
    - Wave motion
- Lighting
  - Color
  - Normal
  - Other per-vertex properties



Transformation to clipping space

Wave motion



http://cse.csusb.edu/tongyu/courses/cs520/notes/glsl.php

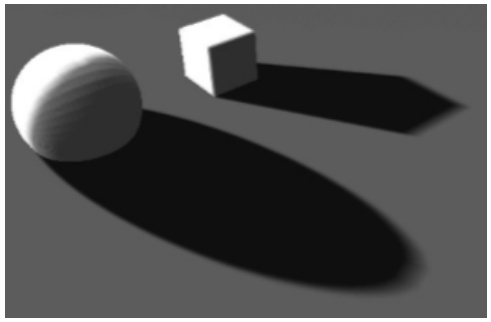
Required Fragment Shader

```
#version 330
in vec3 color_out;
void main(void)
{
  gl_FragColor = color_out;
}
```

Fragment shader

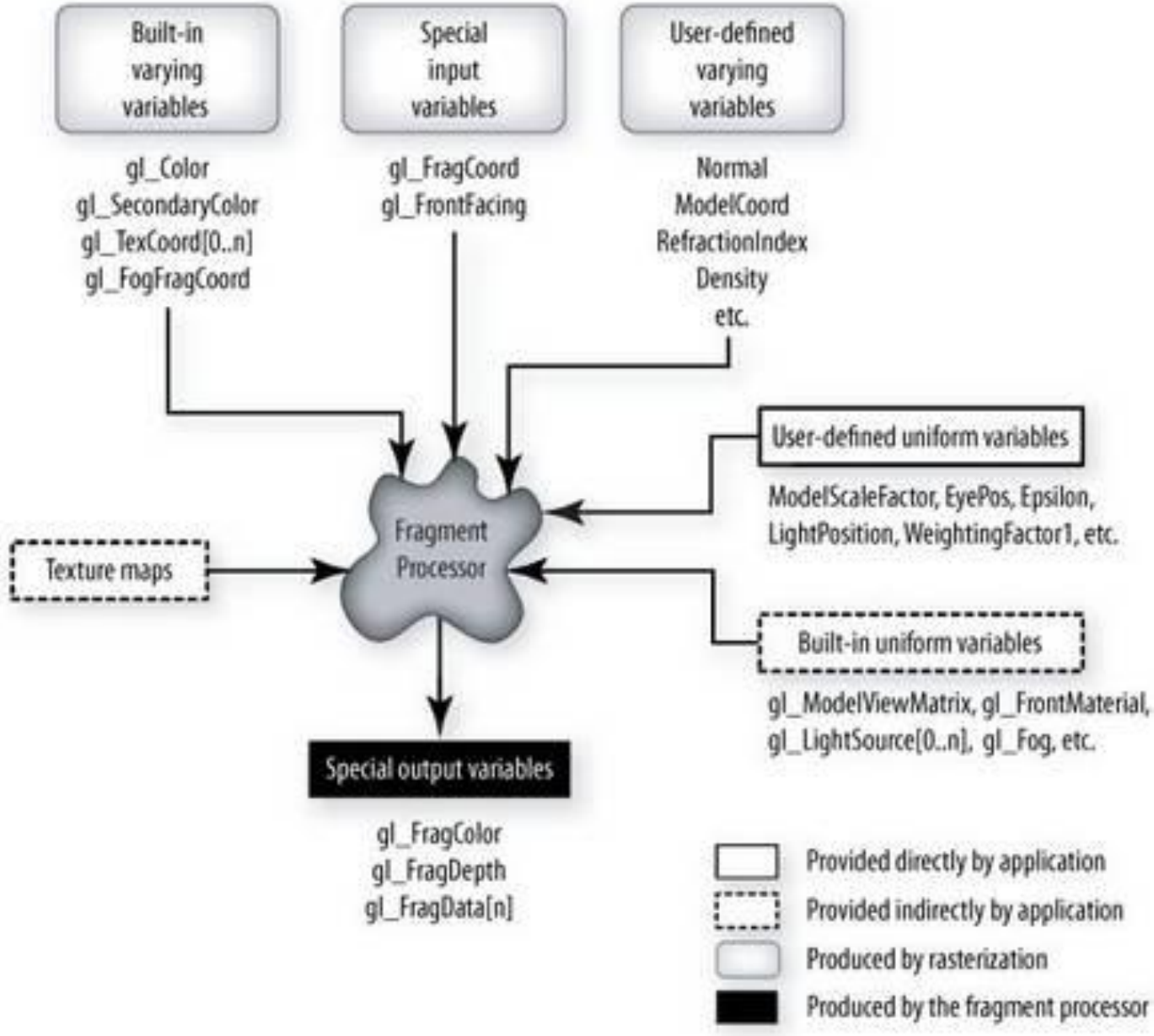
- Compute the color of a fragment/pixel
- The input data is from rasterization and textures and other values

Phong shading (per fragment shading)



shadows

Reflection, refraction etc.



http://cse.csusb.edu/tongyu/courses/cs520/notes/glsl.php

The power of Shaders

- https://www.shadertoy.com/

GLSL with OpenGL

- We should tell OpenGL to use the shader program
- Program object
  - Container for multiple shaders

```
GLuint myProgObj;
myProgObj = glCreateProgram();

/*define shader objects here*/

glLinkProgram(myProgObj);
glUseProgram(myProgObj);
```

Read a shader

- Shader are added to the program object and compiled
- Usual method of passing a shader is as a null-terminated string using the function **`glShaderSource`**
- If the shader is in a file, we can write a reader to convert the file to a string

Adding a Vertex shader

```
GLunit myVertexObj;
GLchar vShaderfile[] = “my_vertex_shader”;
GLchar* vSource = readShaderSource(vShaderFile);
myVertexObj = glCreateShader(GL_VERTEX_SHADER);
glShaderSource(myVertexObj, 1, &vSource, NULL);
glCompileShader(myVertexObj);
glAttachObject(myProgObj, myVertexObj);
```

A Shader Class

https://learnopengl.com/Getting-started/Shaders

```
class Shader
{
public:
   unsigned int ID; // program object ID
   Shader(const char* vertPath, const char* fragPath);
   void use() {
      glUseProgram(ID);
   }
   // utility uniform functions
   void setInt(const std::string &name, int value) const{
      glUniform1i(glGetUniformLocation(ID, name.c_str()),
value);
   }
void setFloat(const std::string &name, float value) const{
      glUniform1f(glGetUniformLocation(ID, name.c_str()),
value);
   }
}
```

```
Shader::Shader (const char* vertPath, const char* fragPath)
{

      // 1. retrieve the vertex/fragment source code as const
char* vShaderCode, fShaderCode
      // 2. compile shaders
      unsigned int vertex, fragment;
      vertex = glCreateShader(GL_VERTEX_SHADER);
      glShaderSource(vertex, 1, &vShaderCode, NULL);
      glCompileShader(vertex);
      // fragment Shader
      fragment = glCreateShader(GL_FRAGMENT_SHADER);
      glShaderSource(fragment, 1, &fShaderCode, NULL);
      glCompileShader(fragment);
      // shader Program
      ID = glCreateProgram();
      glAttachShader(ID, vertex);
      glAttachShader(ID, fragment);
      glLinkProgram(ID);
      // delete the shaders as they're linked into our program now
and no longer necessery
      glDeleteShader(vertex);
      glDeleteShader(fragment);
}
```

A OpenGL program

```
//Init glfw and glew
//Create glfw window and context
//Prepare OpenGL objects
//Create Shader Program object
Shader myShader("cube.vert", "cube.frag");
while (!glfwWindowShouldClose(window)) {// render loop
        //Clear buffer
        glClearColor(0.f, 0.f, 0.f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT);
        //use program object
        ourShader.use();
        //bind vao
        glBindVertexArray(VAO);
        //draw
        glDrawArrays(GL_TRIANGLES, 0, 3);
        glfwSwapBuffers(window);
        glfwPollEvents();
}
```

- Let’s draw a Cube!

References

- https://www.khronos.org/opengl/wiki/OpenGL_Object
- https://www.haroldserrano.com/blog/understanding-opengl-objects
- http://cse.csusb.edu/tongyu/courses/cs520/notes/glsl.php
- https://learnopengl.com/Getting-started/Hello-Triangle
- Steve Marschner, CS4620/5620 Computer Graphics, Cornell
- Ed Angel, CS/EECE 433 Computer Graphics, University of New Mexico

- Questions?