OpenGL Tutorial

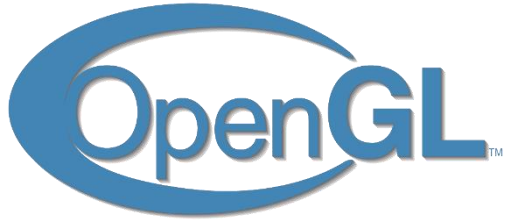
CS380 Computer Graphics

KAIST School of Computing

Computer Graphics & Visualization Lab.

OpenGL

- OpenGL(Open Graphics Library)
  - A cross-language, cross-platform standardized API for real-time computer graphics
  - The API is used to interact with a GPU
  - Purely concerned with **rendering**



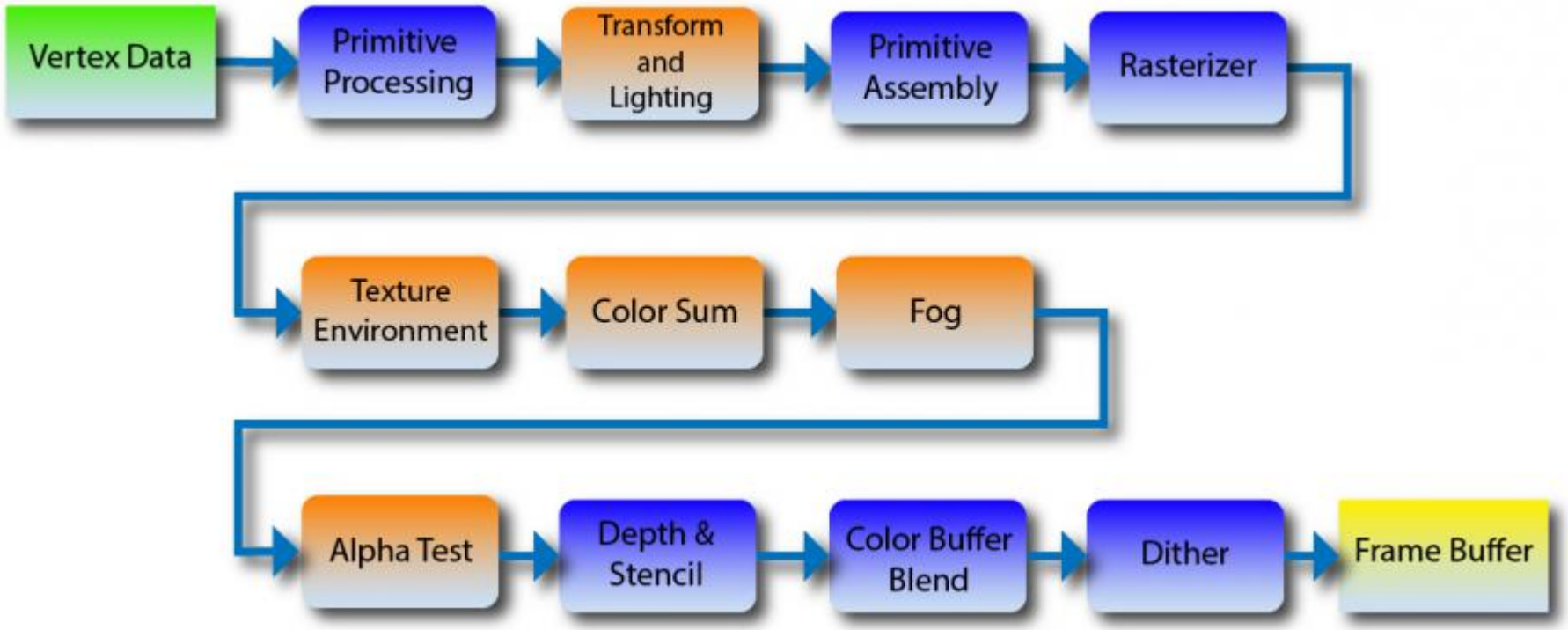
- http://www.opengl.org
  - Since 1992
  - Current version: 4.5

Graphics Pipeline

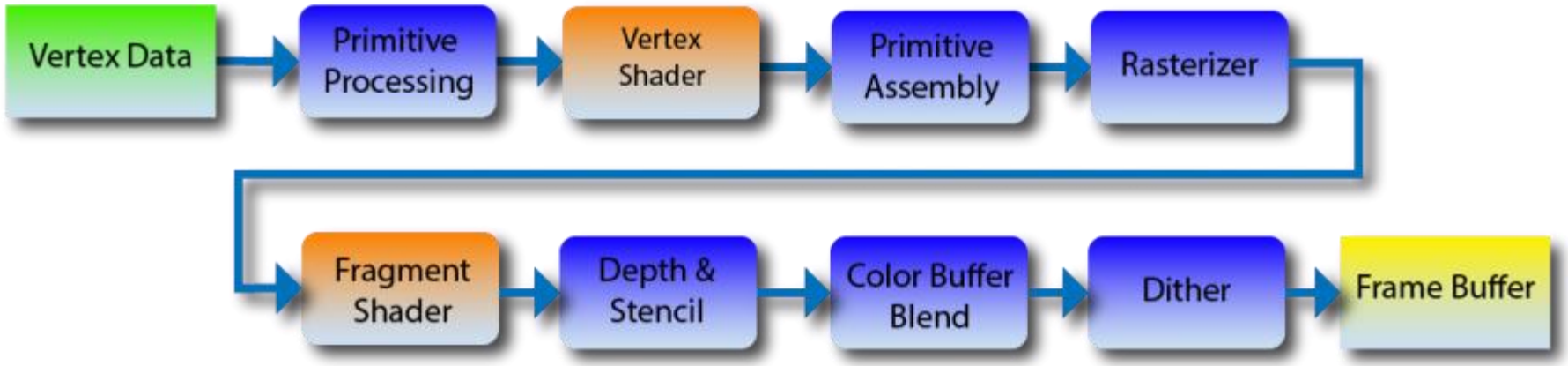
- A sequence of steps to generate a 2D raster representation of a 3D scene
  - Draw objects in the virtual world(3D) to your display(2D)

- Evolved from fixed pipeline to programmable pipeline
  - Becoming more flexible (as GPU evolved)
  - General concept remains same

OpenGL 1.0

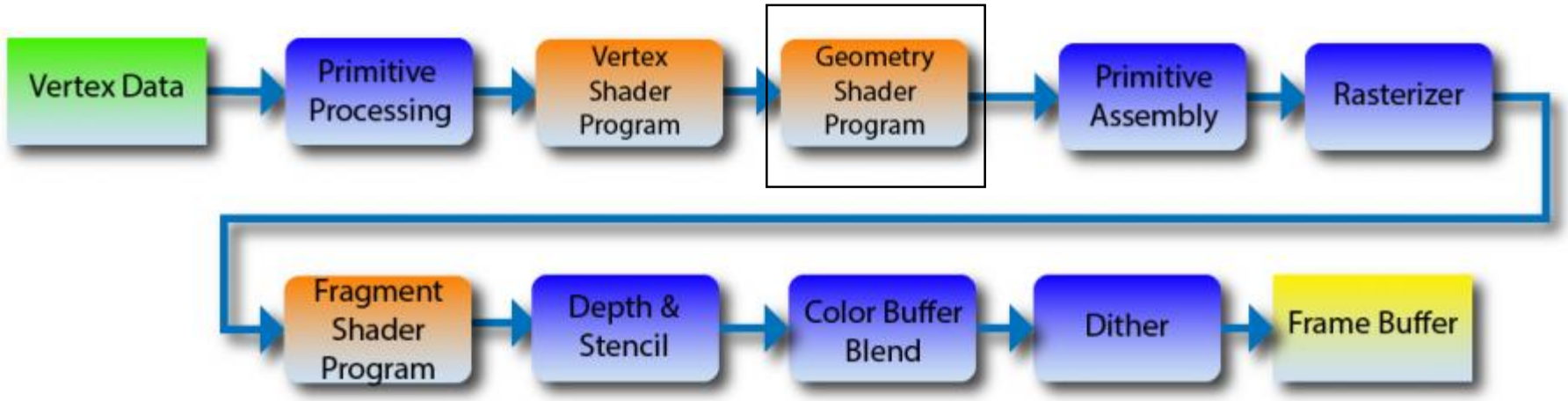

Vertex Data
Primitive Processing
Transform and Lighting
Primitive Assembly
Rasterizer
Texture Environment
Color Sum
Fog
Alpha Test
Depth & Stencil
Color Buffer Blend
Dither
Frame Buffer

OpenGL 2.0

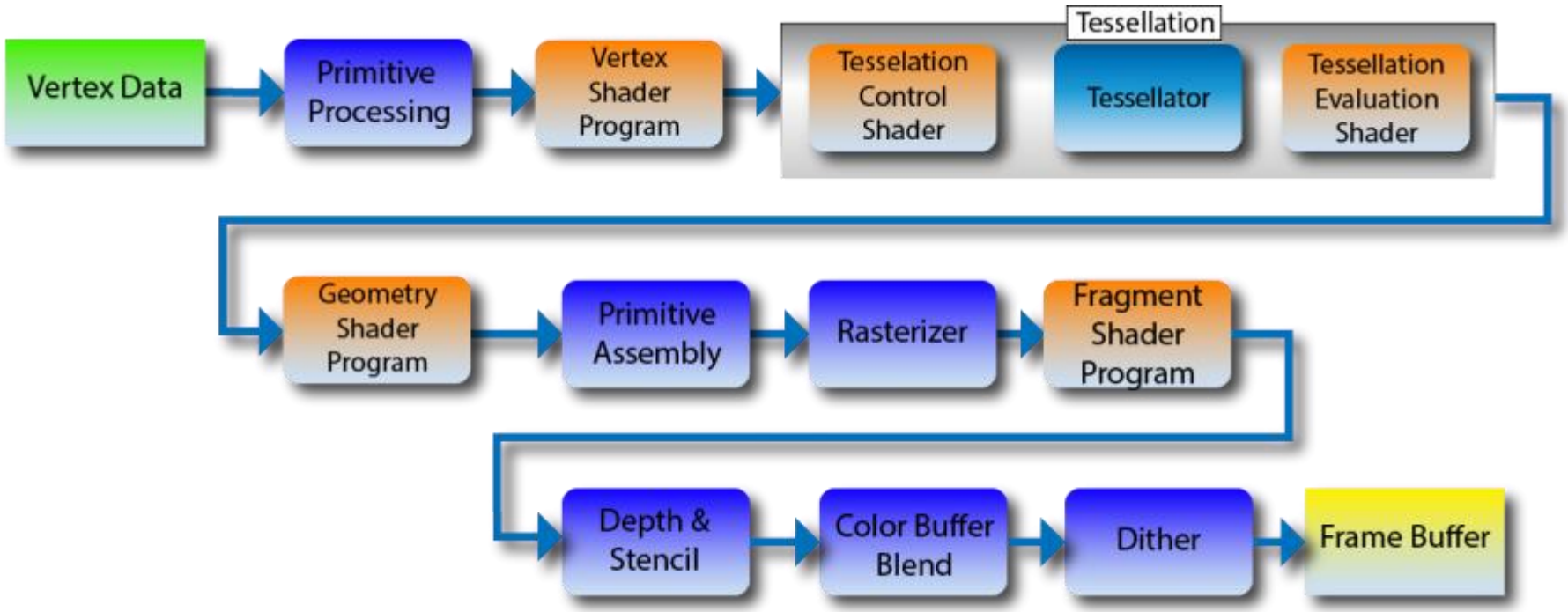


OpenGL 3.2

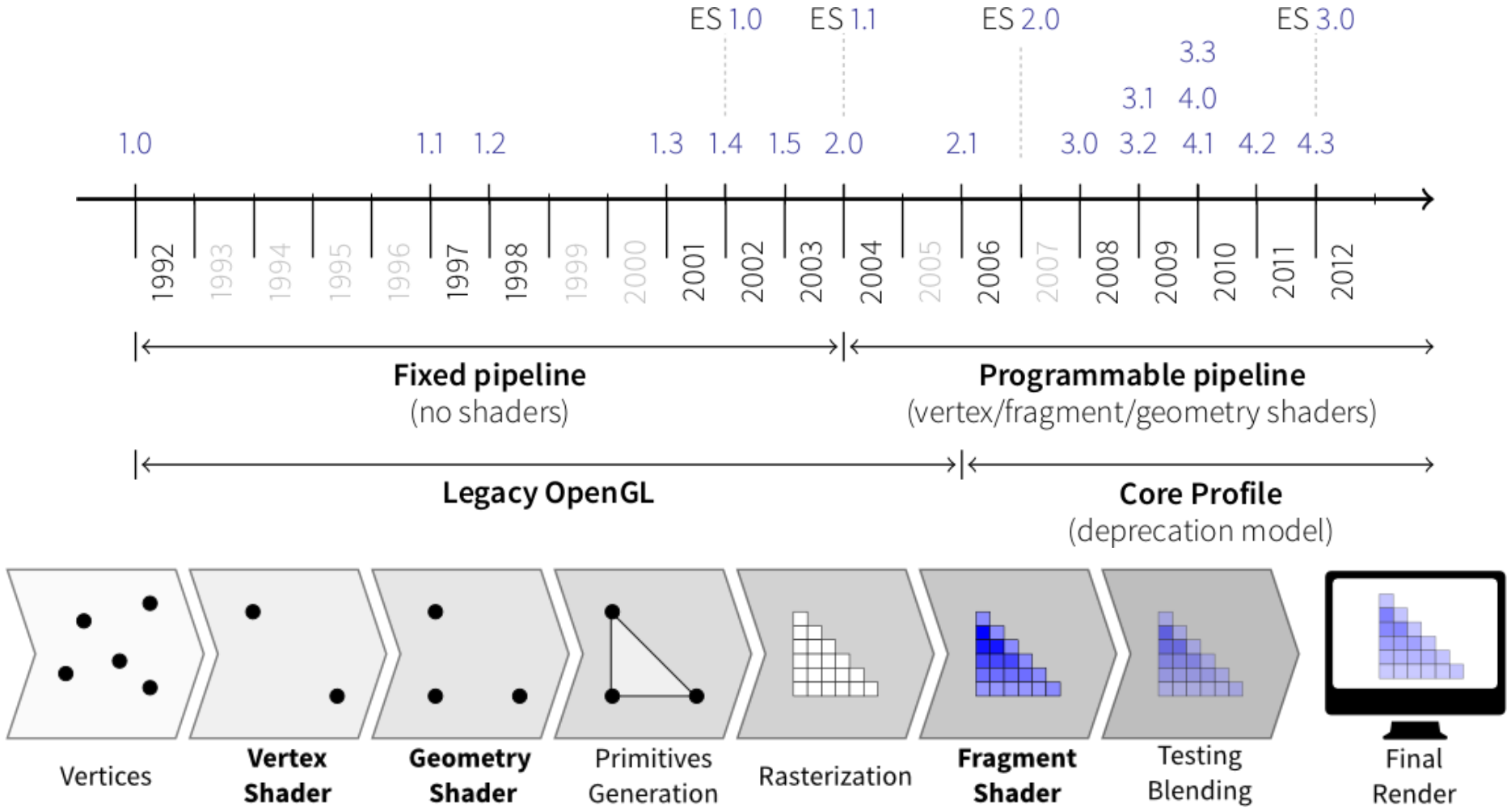
We do not cover geometry shader, you can use it if you want to



OpenGL 4.0 (Recent version)



Summary



KAIST School of Computing

Computer Graphics & Visualization Lab.

Shader

- A user-defined program that tells a computer how to draw something in specific and unique way

- Vertex Shader
  - Per-vertex operation which manipulates properties (color, position, textures, etc.)
- Fragment Shader
  - Per-fragment operation which handles properties (color, depth, sample mask)

GLSL

- OpenGL shader language
  - A C/C++ similar high level programming language
  - **A lot of built-in variables**
    - **Ex) gl_Position, gl_Color etc.**
  - No auto-completion support by IDEs.

- Version
  - Each version of OpenGL is required to support specific versions of GLSL
  - Ex) GL 3.0 – GLSL 1.30, GL 3.2 – GLSL 1.50
  - For all versions of OpenGL 3.3 and above, the corresponding GLSL version matches the OpenGL version.

GLEW & GLM

- GLEW
  - OpenGL Extension Wrangler
  - Provide efficient run-time mechanisms for determining which OpenGL extensions are supported on the target platform
  - Existing extensions are listed in https://www.opengl.org/registry/
- GLM
  - OpenGL mathematics
  - C++ mathematics library for graphics software based on the GLSL specification

GLFW

- Multi-platform library for OpenGL, OpenGL ES, and Vulkan application development

- Platform-independent API for creating
  - Windows
  - Contexts and surfaces
  - Reading input
  - Handling events etc.

Make a OpenGL Program from Scratch

Lab session 1

KAIST School of Computing
Computer Graphics & Visualization Lab.

Download Lab1.zip from KLMS

- In the Lab1.zip,
  - Lab1/main.cpp
  - Lab1/VertexShader.glsl
  - Lab1/FragmentShader.glsl

} GLSL shaders

Add Project and Resources to CMakeLists.txt

```
# Skeleton Project
add_executable(SkeletonProject
  Skeleton/main.cpp
  common/shader.cpp
  common/shader.hpp

  Skeleton/TransformVertexShader.glsl
  Skeleton/ColorFragmentShader.glsl
)

target_link_libraries(SkeletonProject
  ${ALL_LIBS}
)

# Xcode and Visual Studio working directories
set_target_properties(SkeletonProject PROPERTIES XCODE_ATTRIBUTE_CONFIGURATION_BUILD_DIR "${CMAKE_CURRENT_SOURCE_DIR}/Skeleton/")
create_target_launcher(SkeletonProject WORKING_DIRECTORY "${CMAKE_CURRENT_SOURCE_DIR}/Skeleton/")
```

Define a project with sources

Link target libraries

Xcode and MSVS launcher setting

Add Project and Resources to CMakeLists.txt

```
# Lab session 1
add_executable(Lab1
        Lab1/main.cpp
        common/shader.cpp
        common/shader.hpp

        Lab1/VertexShader.glsl
        Lab1/FragmentShader.glsl
)
target_link_libraries(Lab1
  ${ALL_LIBS}
)

# Xcode and Visual Studio working directories
set_target_properties(Lab1 PROPERTIES XCODE_ATTRIBUTE_CONFIGURATION_BUILD_DIR
 "${CMAKE_CURRENT_SOURCE_DIR}/Lab1/")
create_target_launcher(Lab1 WORKING_DIRECTORY "${CMAKE_CURRENT_SOURCE_DIR}/Lab1/")
```

Creating a Window and Context

```
- Initialize GLFW
if (!glfwInit())
{
	return -1;
}
glfwWindowHint(GLFW_SAMPLES, 4);
glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);

window = glfwCreateWindow(1024, 768, "Lab 1", NULL, NULL);
if (window == NULL)
{
	return -1;
}
glfwMakeContextCurrent(window);
glfwSetInputMode(window, GLFW_STICKY_KEYS, GL_TRUE);
```

Creating a Window and Context

```
- Initialize GLEW
glewExperimental = GL_TRUE;
if (glewInit() != GLEW_OK)
{
	return -1;
}
```

Initialize OpenGL and GLSL

```
glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
glEnable(GL_DEPTH_TEST);
glDepthFunc(GL_LESS);

int width, height;
glfwGetFramebufferSize(window, &width, &height);
glViewport(0, 0, width, height);

programID = LoadShaders("VertexShader.glsl","FragmentShader.glsl");
GLuint MatrixID = glGetUniformLocation(programID, "MVP");
```

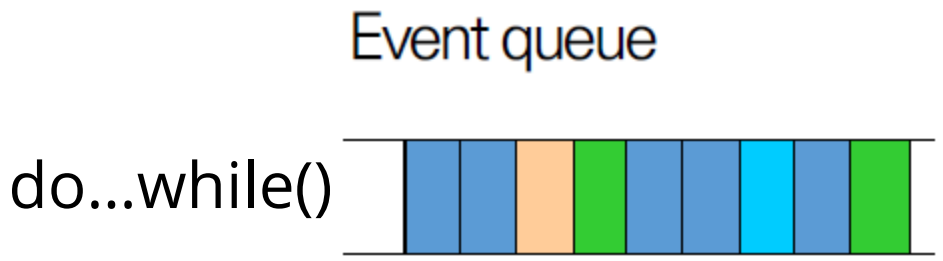
Initialize a Triangle

```
g_vertex_buffer_data = std::vector<glm::vec3>();
g_vertex_buffer_data.push_back(glm::vec3(-0.5f, -0.25f, 0.0f));
g_vertex_buffer_data.push_back(glm::vec3(0.0f, sqrt(0.75)-0.25f, 0.0f));
g_vertex_buffer_data.push_back(glm::vec3(0.5f, -0.25f, 0.0f));

// Generates Vertex Array Objects in the GPU's memory and passes back their identifiers
// Create a vertex array object that represents vertex attributes stored in a vertex buffer object.
glGenVertexArrays(1, &VAID);
glBindVertexArray(VAID);

// Create and initialize a buffer object, Generates our buffers in the GPU's memory
glGenBuffers(1, &VBID);
glBindBuffer(GL_ARRAY_BUFFER, VBID);
glBufferData(GL_ARRAY_BUFFER, sizeof(glm::vec3)*g_vertex_buffer_data.size(),
&g_vertex_buffer_data[0], GL_STATIC_DRAW);
```

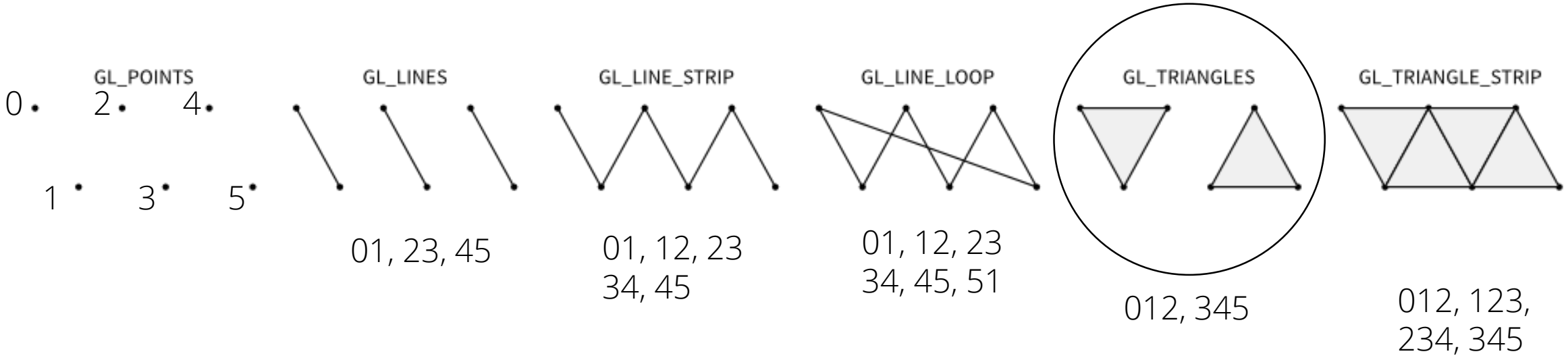
Main Event Loop

```
// Clear buffers
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

… Draw something

// Double buffering
glfwSwapBuffers(window);
// Process all pending events
glfwPollEvents( );
```

Events
- Keypress event
- Mouse event
- Joystick event
- …

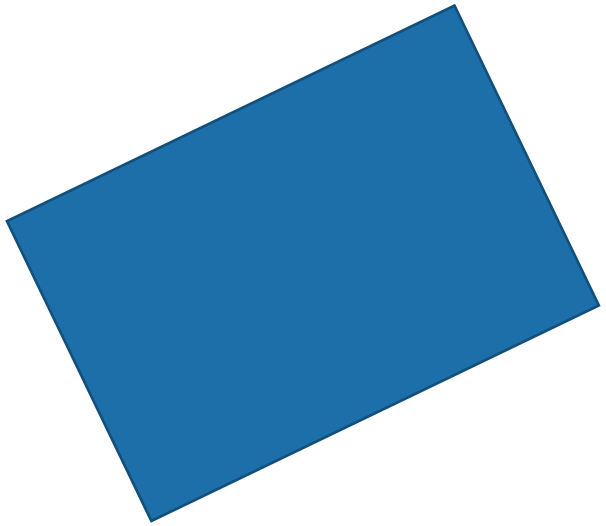


Choosing Primitives

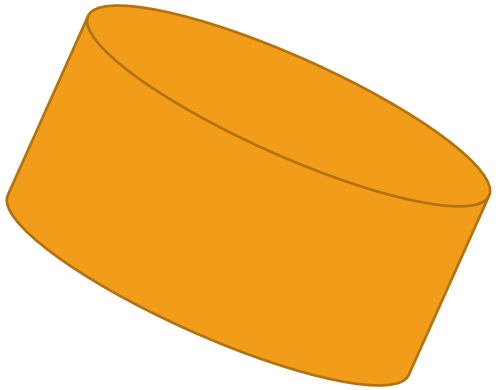


Drawing Basics

```
glUseProgram(program1);
glBindVertexArray(VAID1);
glBindBuffer(GL_ARRAY_BUFFER, VBID1);
//.. Setting Uniforms etc ..
glDrawArrays(GL_TRIANGLES, 0, M);

glUseProgram(program2);
glBindVertexArray(VAID2);
glBindBuffer(GL_ARRAY_BUFFER, VBID2);
//.. Setting Uniforms etc ..
glDrawArrays(GL_TRIANGLES, 0, N);
```



program1, VAID1



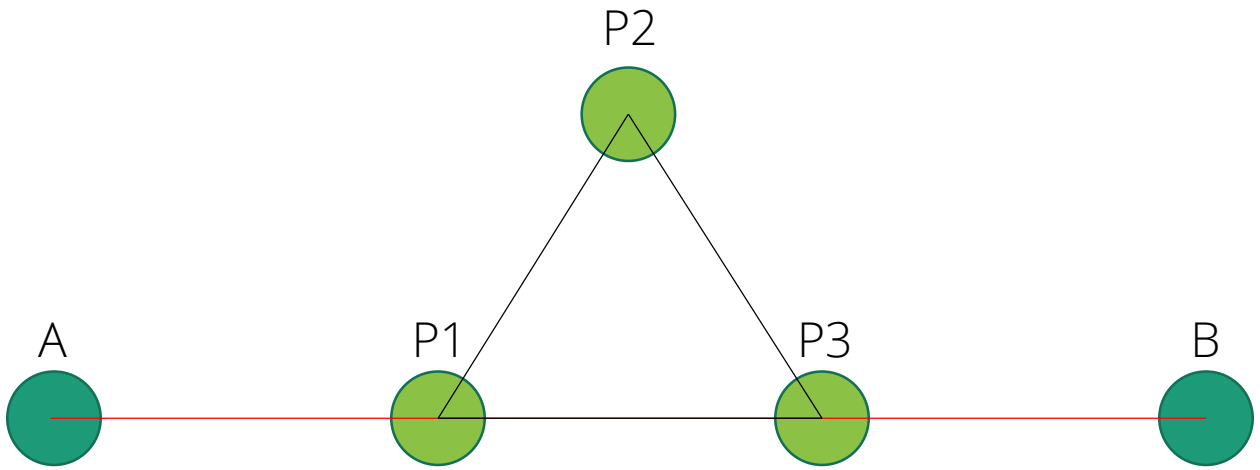
program2, VAID2

Draw Initial Triangle

```
glUseProgram(programID);
glBindVertexArray(VAID);
glEnableVertexAttribArray(0);
glBindBuffer(GL_ARRAY_BUFFER, VBID);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(glm::vec3), BUFFER_OFFSET(0));

glm::mat4 Model = glm::mat4(1.0f);
glm::mat4 MVP = Projection * View * Model;

GLuint MatrixID = glGetUniformLocation(programID, "MVP");
glUniformMatrix4fv(MatrixID, 1, GL_FALSE, &MVP[0][0]);

glDrawArrays(GL_TRIANGLES, 0, g_vertex_buffer_data.size());

glDisableVertexAttribArray(0);
```

Koch Snowflake

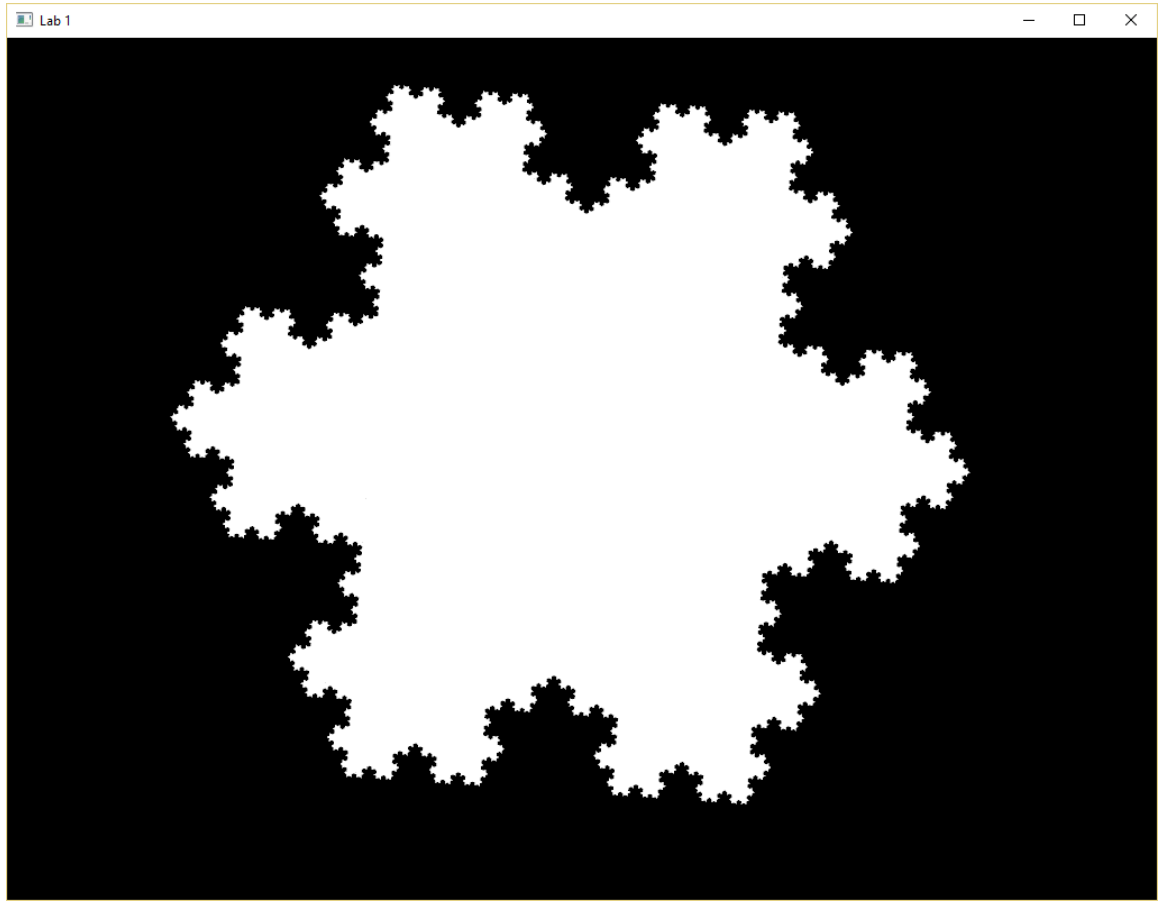
- The Koch snowflake is a mathematical curve and one of fractals
- In this lab session, you need to generate Koch snowflake with triangles



Rigid Body Transformation

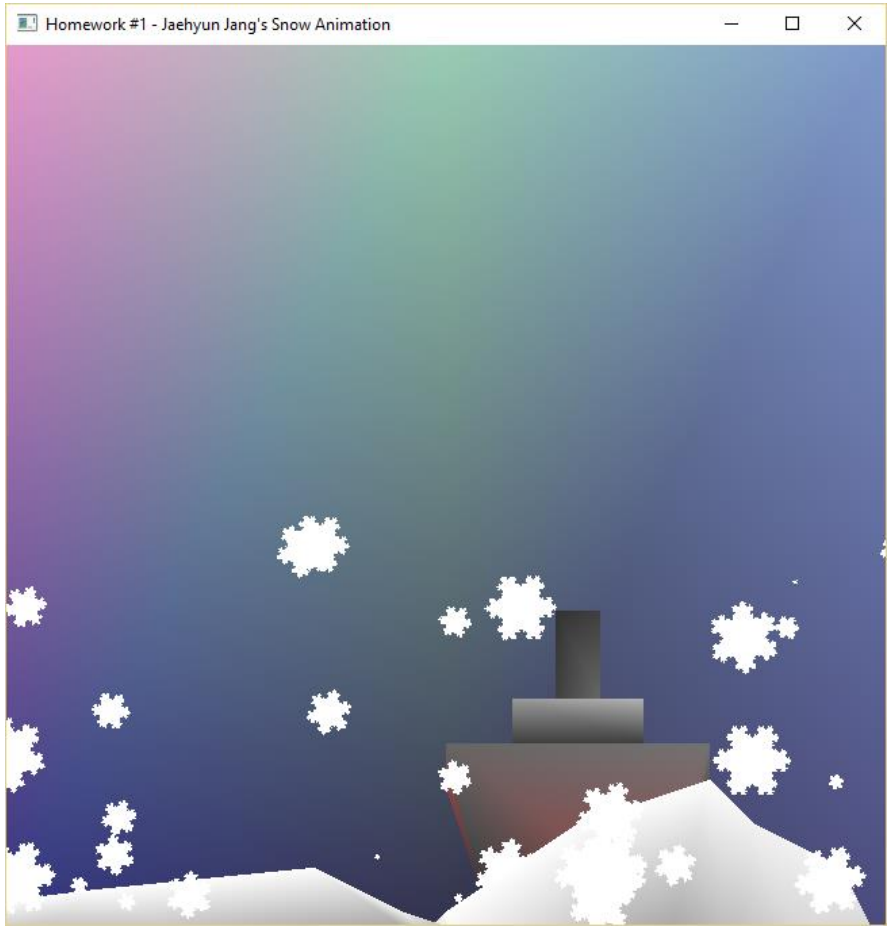
- A = TR (T: Translation, R: Rotation)
- Translation:
  - Translate with respect to x-y plane
  - glm::mat4 T
    = glm::translate(Some Matrix, glm::vec3(x, y, 0.0f));
- Rotation
  - Rotate with respect to z-axis
  - glm::mat4 R
    = glm::rotate(Degree, glm::vec3(0,0,1));
- Apply to MVP matrix
  - glm::mat4 MVP = Projection * View * T * R;

Lab 1 Result



Project 1 – Snow Animation

- Due: March 22



KAIST School of Computing

Computer Graphics & Visualization Lab.

References

- C++ Standard Template Library
  - http://www.sgi.com/tech/stl/
- OpenGL 3.3 Reference
  - http://www.opengl.org/sdk/docs/man3/
  - https://www.opengl.org/registry/doc/glspec33.core.20100311.pdf
- GLSL 3.3 Specification
  - https://www.opengl.org/registry/doc/GLSLangSpec.3.30.6.pdf
- GLFW 3.0.3 Reference
  - http://www.glfw.org/docs/3.0.3/modules.html
- GLM 0.9.4 Reference
  - http://glm.g-truc.net/0.9.4/api/index.html