What is OpenGL?
- **OpenGL** is a software interface to graphics hardware.
- **API** to specify geometric objects in 2D/3D and to control how they are rendered into the framebuffer.
- A software interface to graphics hardware.
- Cross-language, cross-platform, open source.
- Alternatives – Direct3D (Microsoft)

OpenGL Vertices
- A **vertex** is a point coordinate for OpenGL – it is internally represented as a four vector (more on why later).
- Vertices are assembled into **primitives** – Points, Lines, Triangles

OpenGL Fragments
- A **fragment** is a pixel with a lot of other information:
  - Location
  - Color
  - Normal
  - Depth
  - Opacity

OpenGL 1 (1991)
- Entirely fixed-function
- The pipeline evolved but remained based on fixed-function operation through OpenGL versions 1.1 through 2.0.

OpenGL 2 (1994)
- Introduced programmable shaders
- Vertex shading, Fragment shading
OpenGL 3, 3.1, 3.2 (~2008)

- Introduced the deprecation model
  - Removed the fixed function pipeline.
- Large chunks of data fed to the pipeline instead of small chunks.
- OpenGL catching up with GPU architecture

OpenGL 4.0 – 4.5 (current day)

- Geometry & Tessellation Shader - Geometry generated on the GPU

OpenGL Variants

- OpenGL ES
  - For hand-held devices and embedded environments
- WebGL
  - Javascript implementation of ES
  - Runs on most recent browsers

OpenGL Application Program

- OpenGL Programming
  - Create Shader Program
  - Create Buffer Objects and load data into them.
  - "Connect" data locations with shader variables
  - Render
- Windowing System Interface: GLFW
  - Opening windows, handling input
- Version, Context and profiles: GLEW
- OpenGL Math Library (only headers): GLM
Geometric Objects in OpenGL

- **A vertex** is a location in space.
  - Attributes: Position Coordinates
  - Colors, Texture Coordinates, Other Data

- Vertex data must be stored in vertex buffer objects (VBOs)
- Vertex attribute information must be stored in vertex array objects (VAOs)

OpenGL Primitives

Example: ColorCube

- A cube with different colors at each vertex

```c
const int num_vertices = 36;

// Eight vertices in homogenous coordinates
glm::vec4 positions[8] = {
  glm::vec4(-0.5, -0.5, 0.5, 1.0),
  glm::vec4(-0.5, 0.5, 0.5, 1.0),
  glm::vec4(0.5, -0.5, 0.5, 1.0),
  glm::vec4(0.5, 0.5, 0.5, 1.0),
  glm::vec4(-0.5, -0.5, -0.5, 1.0),
  glm::vec4(-0.5, 0.5, -0.5, 1.0),
  glm::vec4(0.5, -0.5, -0.5, 1.0),
  glm::vec4(0.5, 0.5, -0.5, 1.0)
};

// RGBA colors
glm::vec4 colors[8] = {
  glm::vec4(0.0, 0.0, 0.0, 1.0),
  glm::vec4(1.0, 0.0, 0.0, 1.0),
  glm::vec4(1.0, 1.0, 0.0, 1.0),
  glm::vec4(1.0, 0.0, 1.0, 1.0),
  glm::vec4(0.0, 1.0, 0.0, 1.0),
  glm::vec4(0.0, 0.0, 1.0, 1.0),
  glm::vec4(1.0, 1.0, 1.0, 1.0),
  glm::vec4(0.0, 1.0, 1.0, 1.0)
};

void colorcube(void)
{
  quad( 1, 0, 3, 2 );    quad( 2, 3, 7, 6 );
  quad( 3, 0, 4, 7 );    quad( 6, 5, 1, 2 );
  quad( 4, 5, 6, 7 );    quad( 5, 4, 0, 1 );
}
```

- Vertex Array Object
  - Stores attribute data corresponding to the VBOs
    ```c
    GLuint vao;
    glGenVertexArrays( 1, &vao );
    glBindVertexArray( vao );
    ```
Example: ColorCube

- Vertex Buffer Object
  - Vertex data must be stored in a VBO, and associated with a VAO
  - The code-flow is similar to configuring a VAO
    - generate VBO names by calling `glGenBuffers`
    - Bind a specific VBO for initialization by calling `glBindBuffer`
    - Load data into VBO using `glBufferData` or `glBufferSubData`

Example: ColorCube

- Create a Shader Program
  - `glCreateProgram`
  - `glCreateShader`
  - `glShaderSource`
  - `glCompileShader`
  - `glCreateShader`
  - `glAttachShader`
  - `glLinkProgram`
  - `glUseProgram`

  These steps need to be repeated for each type of shader in the shader program.

Example: ColorCube

- GLSL Vertex Shader
  ```glsl
  #version 430
  in vec4 vPosition;
  in vec4 vColor;
  out vec4 color;
  void main()
  {
      gl_Position = vPosition;
      color = vColor;
  }
  ```

Example: ColorCube

- GLSL Fragment Shader
  ```glsl
  #version 430
  in vec4 color;
  out vec4 frag_color;
  void main()
  {
      frag_color = color;
  }
  ```

Example: ColorCube

- Connect application program data to shader variable.
  - Assumes shader variable names are known
    ```c
    GLint loc_idx = glGetUniformLocation(program_id, "variable_name");
    ```
    ```c
    glVertexAttribPointer(loc_idx, 4, GL_FLOAT, GL_FALSE, 0, pointer_to_data_in_buffer);
    ```

Example: ColorCube

- Connect application program data to shader variable.
  - Have to find variable/index association
    - `glGetAttribLocation` before linkage
    - `glVertexAttribPointer` after linkage
Example: ColorCube

- **GLFW**
  - `glDrawArrays(GL_TRIANGLES, 0, num_vertices);`
  - Calls a vertex shader for each vertex.
  - Assembled into triangles and rasterized to fragments.
  - Calls a fragment shader for each fragment.

```
glfwSetErrorCallback(csX75::error_callback);
if (!glfwInit()) return -1;

// We want OpenGL 4.0
glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 4);
glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 1);

// We don't want the old OpenGL
glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);

// Create a windowed mode window and its OpenGL context
window = glfwCreateWindow(512, 512, "CS475/CS675 Tutorial 2: Colorcube", NULL, NULL);
if (!window) {  glfwTerminate(); return -1; }

// Make the window's context current
glfwMakeContextCurrent(window);
```

Example: ColorCube

- **GLEW**
  - `/Initialize GLEW`
  - `glewExperimental = GL_TRUE;`
  - `GLenum err = glewInit();`
OpenGL Rasterization

OpenGL Line Rasterization

OpenGL Polygon Rasterization
OpenGL Polygon Rasterization

X

Y

OpenGL Polygon Rasterization

X

Y