OpenGL Drawing
What is OpenGL?

- **Open Graphics Library**
- *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 Variants

• OpenGL ES
  - For hand-held devices and embedded environments

• WebGL
  - Javascript implementation of ES
  - Runs on most recent browsers
Shading Languages

- Shading Languages
  - OpenGL Shading Language (GLSL)
  - High Level Shading Language (HLSL, DirectX)
  - CUDA
  - OpenCL (Open Compute Language)
  - CUDA and OpenCL are/have become more than shading languages and are now used for general purpose compute on the GPU.
Bleeding Edge Graphics Programming

- Vulkan (cross platform)
- Metal (apple only)
- DirectX DXR (microsoft only)

https://devblogs.nvidia.com/vulkan-raytracing/
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 rasterizes primitive shapes and outputs fragments.
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

An Introduction to OpenGL Programming, Edward Angel and Dave Schreiner, SIGGRAPH 2013 Course.
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 Pipeline Diagram](image-url)

An Introduction to OpenGL Programming, Edward Angel and Dave Schreiner, SIGGRAPH 2013 Course.
OpenGL 4.0 – 4.6 (current day)

- Geometry & Tessellation Shader – Geometry generated on the GPU
Simplified Pipeline

An Introduction to OpenGL Programming, Edward Angel and Dave Schreiner, SIGGRAPH 2013 Course.
Simplified Pipeline

Vertices → Vertex Processing → Rasterizer → Fragments → Fragment Processing → Pixels

An Introduction to OpenGL Programming, Edward Angel and Dave Schreiner, SIGGRAPH 2013 Course.
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

GL_POINTS
GL_LINES
GL_LINE_STRIP
GL_LINE_LOOP
GL_TRIANGLES
GL_TRIANGLE_STRIP
GL_TRIANGLE_FAN
GL_QUADS
GL_QUAD_STRIP
GL_POLYGON

Deprecated
Example: ColorCube

- A cube with different colors at each vertex

```cpp
//6 faces, 2 triangles/face, 3 vertices/triangles
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),
    ...
};

//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),
    ...
};
```
Example: ColorCube

- A cube with different colors at each vertex

```c
// quad defines two triangles for each face and assigns colors to the vertices
void quad(int a, int b, int c, int d)
{
    v_colors[tri_idx] = colors[a]; v_positions[tri_idx] = positions[a]; tri_idx++;
    v_colors[tri_idx] = colors[b]; v_positions[tri_idx] = positions[b]; tri_idx++;
    v_colors[tri_idx] = colors[c]; v_positions[tri_idx] = positions[c]; tri_idx++;
    v_colors[tri_idx] = colors[a]; v_positions[tri_idx] = positions[a]; tri_idx++;
    v_colors[tri_idx] = colors[c]; v_positions[tri_idx] = positions[c]; tri_idx++;
    v_colors[tri_idx] = colors[d]; v_positions[tri_idx] = positions[d]; tri_idx++;
}

// define 12 triangles: 36 vertices and 36 colors
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 );
}
```

● A cube with different colors at each vertex
Example: ColorCube

- A cube with different colors at each vertex

- VertexArrayObjects (VAOs)

- VertexBufferObjects (VBOs)

- Generate and Bind
Example: ColorCube

- 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( GL_ARRAY_BUFFER, ... )
    ```
  - Load data into VBO using
    ```
    glBufferData( GL_ARRAY_BUFFER, ... )
    glBindBufferSubData( GL_ARRAY_BUFFER, ..., )
    ```
Example: ColorCube

- Create a Shader Program

  - Create Program
  - Create Shader
  - Load Shader Source
  - Compile Shader
  - Attach Shader to Program
  - Link Program
  - Use Program

  ```
  glCreateProgram()
  glCreateShader()
  glShaderSource()
  glCompileShader()
  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.
- OpenGL relates shader variables to indices for the application to set
- Have to find variable/index association
  - Before linkage
  - After linkage
Example: ColorCube

• Connect application program data to shader variable.

• Assumes shader variable names are known

    GLint loc_idx = glGetAttribLocation(program_id, "variable_name");

    GLint loc_idx = glGetUniformLocation(program_id, "variable_name");

    glEnableVertexAttribArray(loc_idx);

    glVertexAttribPointer(loc_idx, 4, GL_FLOAT, GL_FALSE, 0, pointer_to_data_in_buffer);
Example: ColorCube

- Draw
  
  ```
  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.
Example: ColorCube

- GLFW

```c
GLFWwindow* window;
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

// Turn this on to get Shader based OpenGL
GLEWExperimental = GL_TRUE;
GLenum err = glewInit();
if (GLEW_OK != err)
{
    // Problem: glewInit failed, something is seriously wrong.
    std::cerr<<"GLEW Init Failed : %s"<<std::endl;
}

/Initialize GLEW
Example: ColorCube

• GLFW

```cpp
glfwSetKeyCallback(window, csX75::key_callback);
glfwSetFramebufferSizeCallback(window, csX75::framebuffer_size_callback);
glfwSetInputMode(window, GLFW_STICKY_KEYS, GL_TRUE);

// Initialize GL state
csX75::initGL(); initBuffersGL();

// Loop until the user closes the window
while (!glfwWindowShouldClose(window))
{
    renderGL();
    glfwSwapBuffers(window);
    glfwPollEvents();
}

glfwTerminate();
return 0;
```
Example: ColorCube
OpenGL Rasterization

X

Y

(0, 0) (0, 1)

(1, 0) (1, 1)
OpenGL Rasterization

X

Y

(0.0)   (1.0)
(0.5,0.5)   (1.5,0.5)
(0.5,1.5)   (1.5,1.5)

(0.1)   (1.1)
OpenGL Line Rasterization

X

Y

(0.0, 0.0)  (1.0, 0.0)
(0.0, 1.0)  (1.0, 1.0)
(0.5, 0.5)  (1.5, 0.5)
(0.5, 1.0)  (1.5, 1.0)
OpenGL Line Rasterization
OpenGL Line Rasterization

(0,0) (1,0) (0,1) (1,1)
OpenGL Polygon Rasterization
OpenGL Polygon Rasterization
OpenGL Polygon Rasterization

Y

X
OpenGL Polygon Rasterization