OpenGL Tutorial

CMPT-361

Luwei Yang
Email: luweiy@sfu.ca

Before we start

SFU

OpenGL versions we are going to use

- New OpenGL: 3.x- 4.x
- Require programs to use Shaders.
- Almost all data is GPU-resident.

Programming Assignments

- Language: C++
- Basic code will be provided.
- The standard pipeline has been written for you.

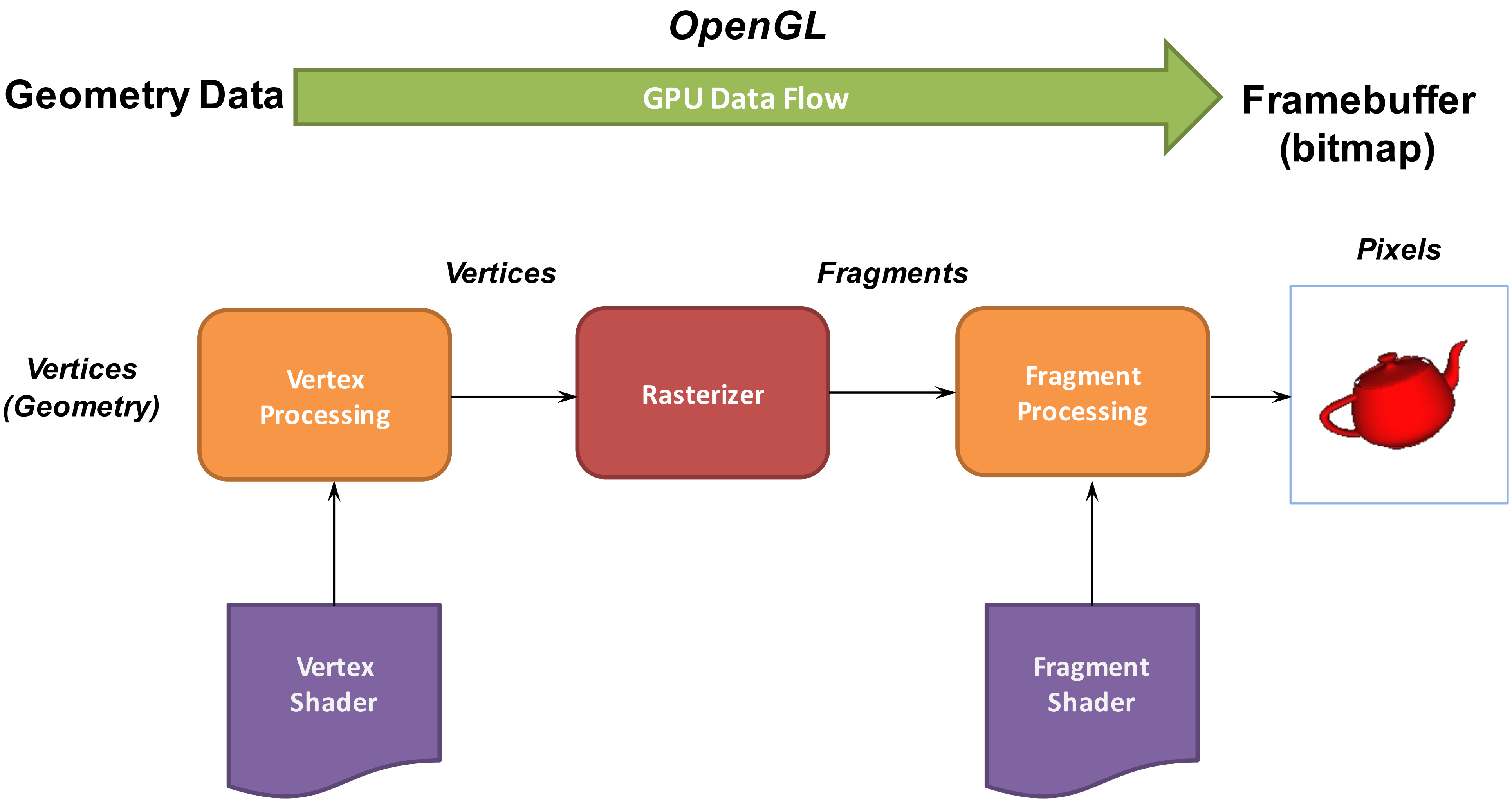
Useful Resource

- SIGGRAPH University : “An Introduction to OpenGL Programming” (https://www.youtube.com/watch?v=6-9XFm7XAT8)

Plan for Today

- Overview on modern OpenGL pipeline – 20 mins
- Walkthrough of the demo code – 30 mins

A Simplified Pipeline Model



OpenGL Programming in a Nutshell

- **Modern OpenGL programs essentially do the following steps:**

1. **Create Shader programs**

   Tell the GPU what to do with the data

2. **Create buffer objects and load data into them**

   Pass the data to GPU

3. **"Connect" data locations with Shader variables**

   Tell the GPU how to find the data in Shader programs.

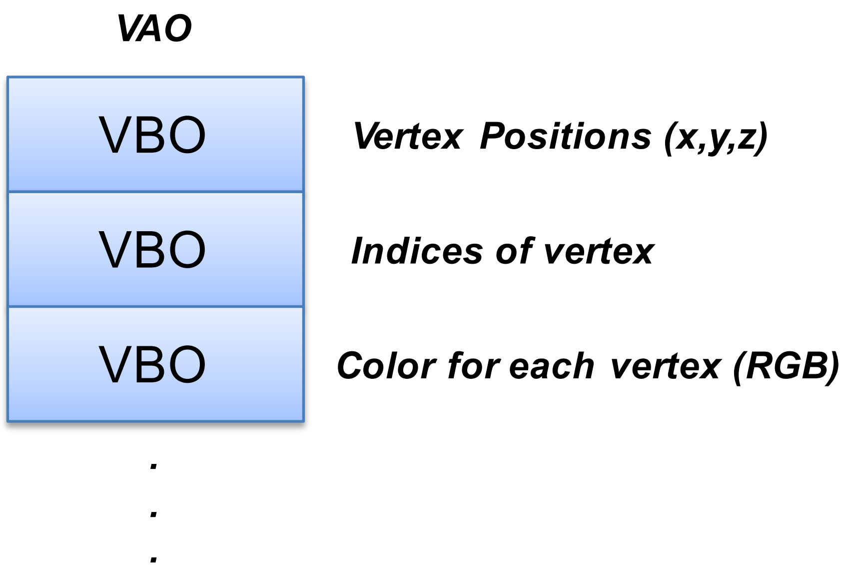
4. **Setup Transformation**
5. **Render!**

Vertex Buffer Objects (VBOs)

- **Vertex data must be stored in a VBO**
- **The code-flow:**
  - generate VBO by calling `glGenBuffers()`
  - bind a specific VBO for initialization by calling `glBindBuffer( GL_ARRAY_BUFFER, … )`
  - load data into VBO using `glBufferData( GL_ARRAY_BUFFER, … )`

Vertex Array Objects (VAOs)

- **VAOs store the data of geometric objects / attributes**



- **Steps in using a VAO**
  - generate a VAO object by calling `glGenVertexArrays()`
  - bind a specific VAO for initialization by calling `glBindVertexArray()`

Connecting Vertex Shaders with Geometric Data

- Vertex data enters the OpenGL pipeline through the vertex shader.
- Need to connect vertex data to Shader variables
- Tell the Shader the attribute location by calling: `glGetVertexAttribLocation()`

Link Shader programs

- **We've created a routine for this course to make it easier to load your shaders**

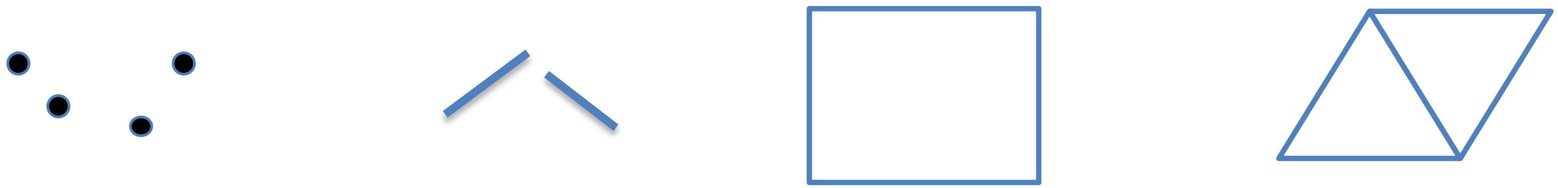
```
GLuint InitShaders(const char* vFile,
                   const char* fFile );
```

- **`InitShaders` takes two filenames**
  - `vFile` path to the vertex shader file
  - `fFile` for the fragment shader file

- **Fails if shaders don't compile, or program doesn't link**

Drawing

- Draw your geometric objects `glDrawArrays(GL_TRIANGLES, 0, NumVertices);`

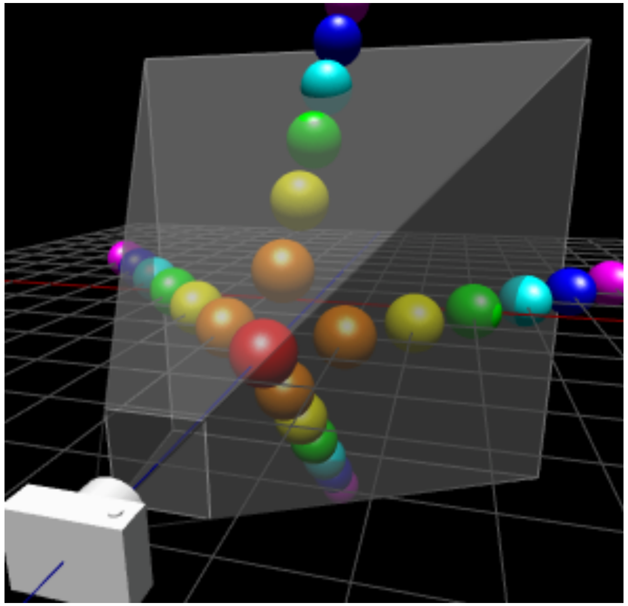


GL_POINTS GL_LINES GL_LINE_LOOP GL_TRANGLES

. . .

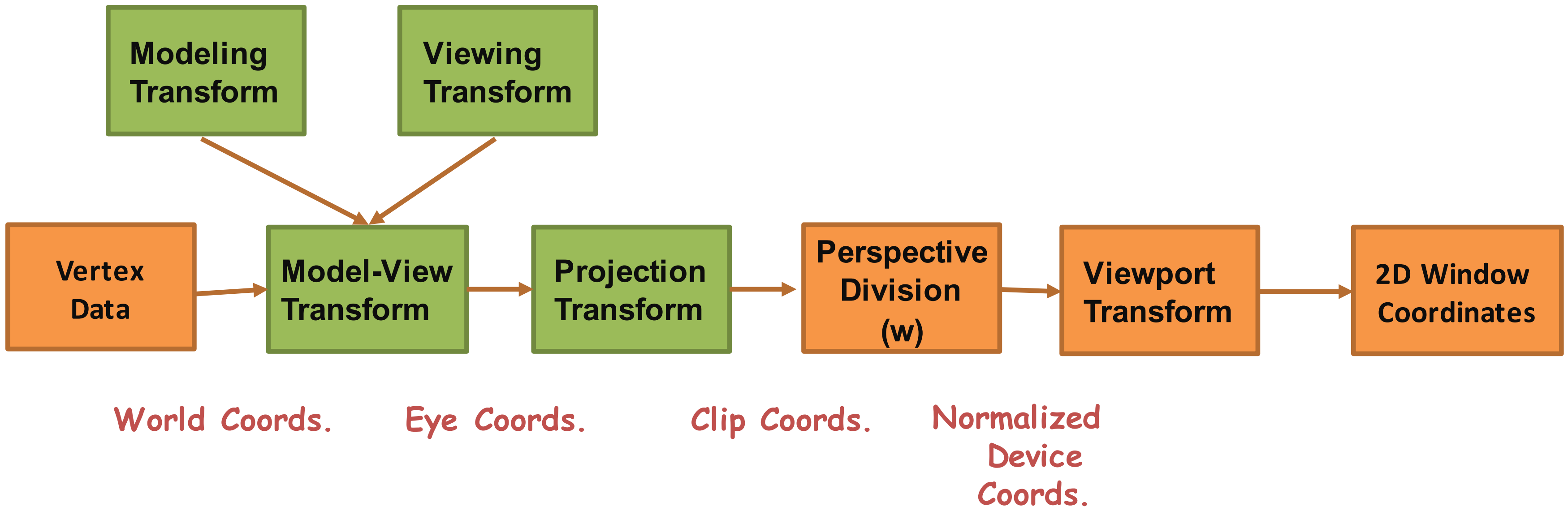
Transformations

- **3D is just like taking a photograph**



Transformations

- Transformations take us from one “space” to another
- All transforms are 4×4 matrices



Modeling transformation

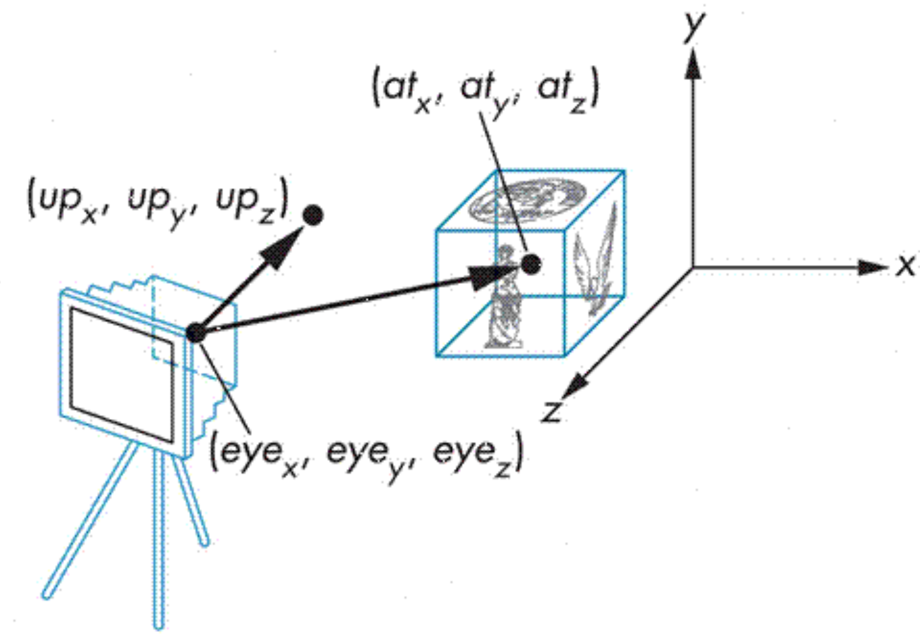
- **Translation, Rotation, Scale**

$$T(t_x, t_y, t_z) = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad S(s_x, s_y, s_z) = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

```
          ( cos q   sin q   0   0)
Rz (q) = (-sin q   cos q   0   0)
          ( 0           0   1   0)
          ( 0           0   0   1)
```

Viewing transfromation

- `LookAt( eye, target, up )`



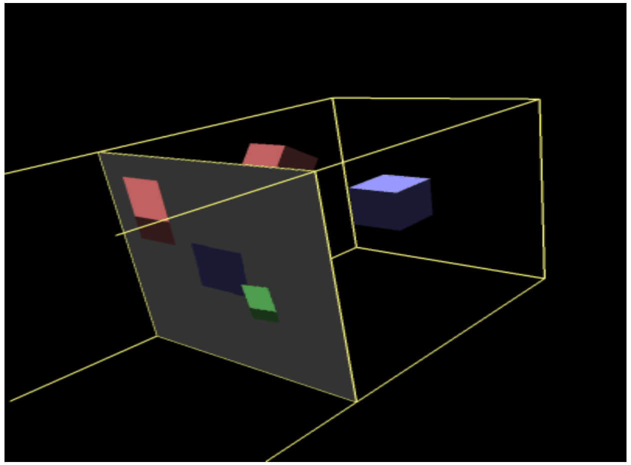
$$\hat{n} = \frac{\overrightarrow{\text{target}} - \overrightarrow{\text{eye}}}{\left\| \overrightarrow{\text{target}} - \overrightarrow{\text{eye}} \right\|}$$

$$\hat{u} = \frac{\hat{n} \times \overrightarrow{\text{up}}}{\left\| \hat{n} \times \overrightarrow{\text{up}} \right\|} \Rightarrow \begin{pmatrix} u_x & u_y & u_z & -(\text{eye} \cdot \vec{u}) \\ v_x & v_y & v_z & -(\text{eye} \cdot \vec{v}) \\ -n_x & -n_y & -n_z & -(\text{eye} \cdot \vec{n}) \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

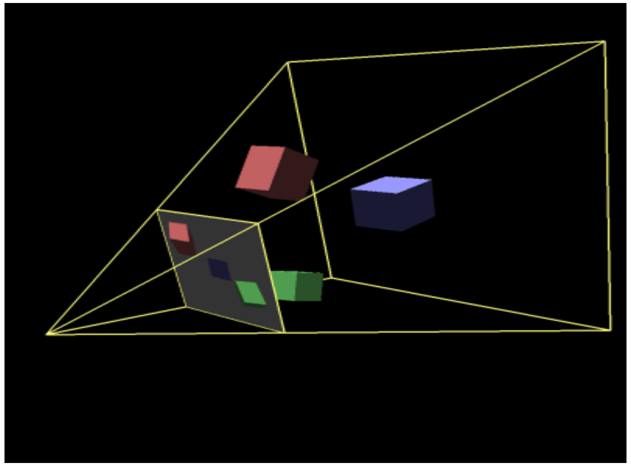
$$\hat{v} = \hat{u} \times \hat{n}$$

Projection

Orthographic View



Perspective View



$$O = \begin{pmatrix} \frac{2}{r-l} & 0 & 0 & \frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & \frac{t+b}{t-b} \\ 0 & 0 & \frac{-2}{f-n} & \frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$P = \begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Demo Code

Demo-1: Cube

- Compile/render your first 3D object
- Understand transformations

Demo-2: Square

Compile and run the demos on Linux

- Unzip the folder
- Change directory to folder cube
  `$: cd`
- Makefile
  `$: make`
- Run the executable
  `$: ./demo`