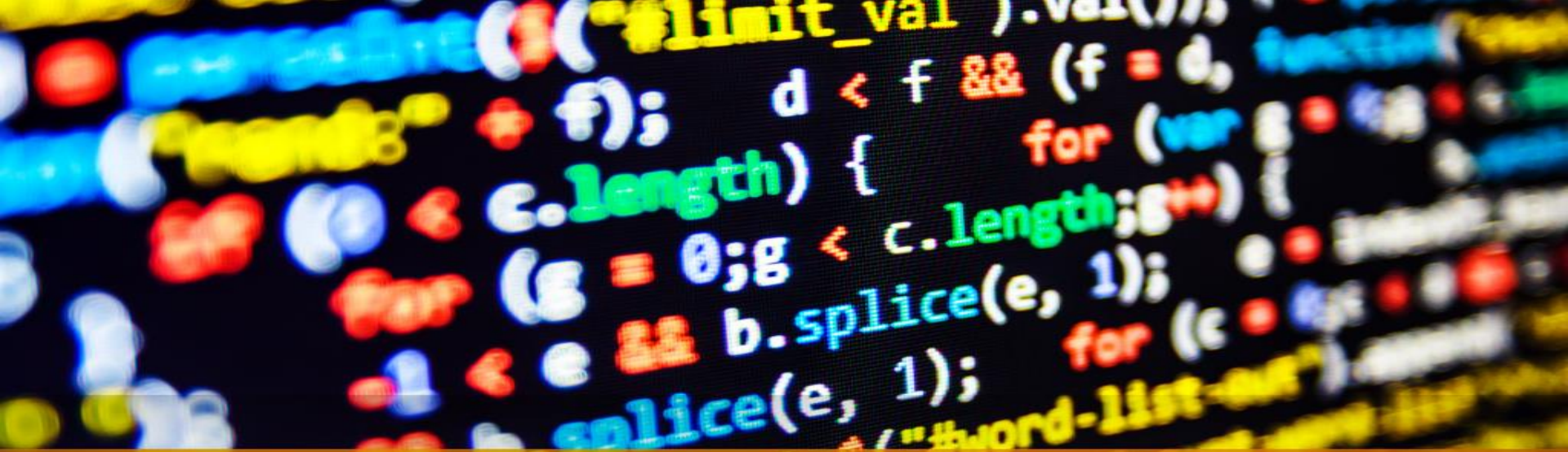


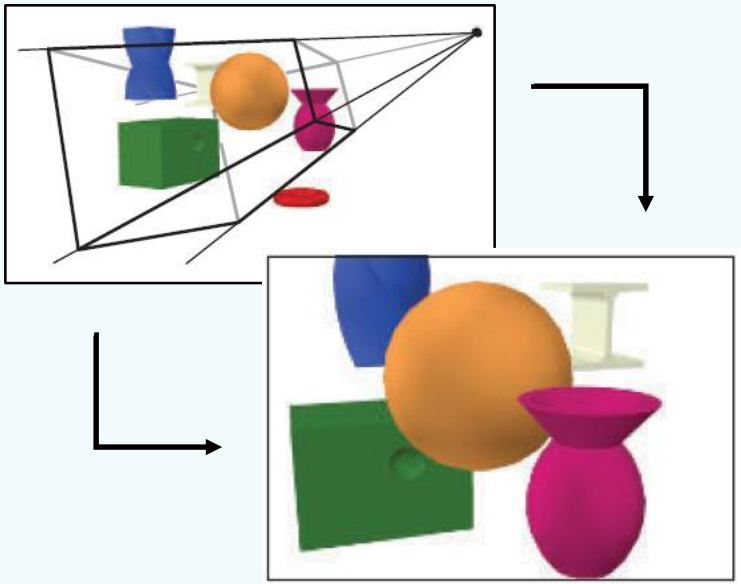
Lecture 02

# The Graphics Pipeline

Prepared by Ban Kar Weng (William)

# The Graphics Pipeline

# The Graphics Pipeline



**Input:**

A virtual camera

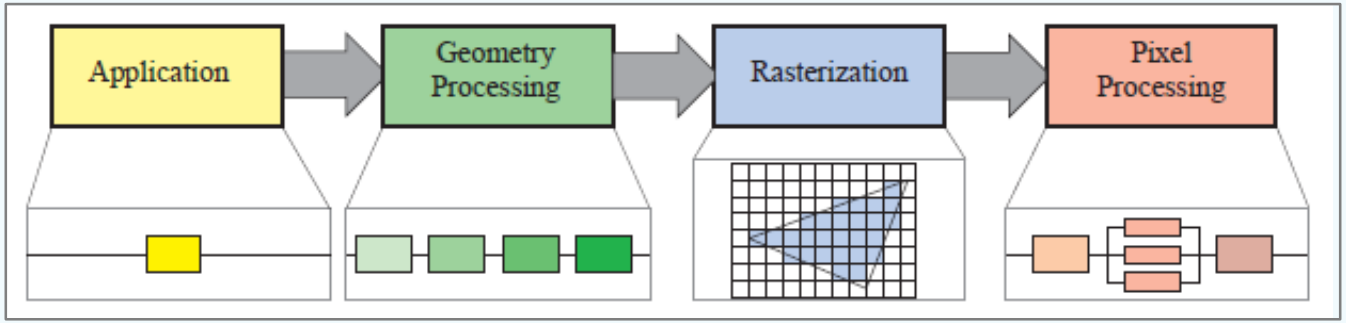
- Three dimensional (3D) objects
- Light sources
- *and more ……*

**Output:**

- A two dimensional (2D) image

Image taken from Figure 2.1 – Real-time Rendering (4th Edition)

# The Graphics Pipeline

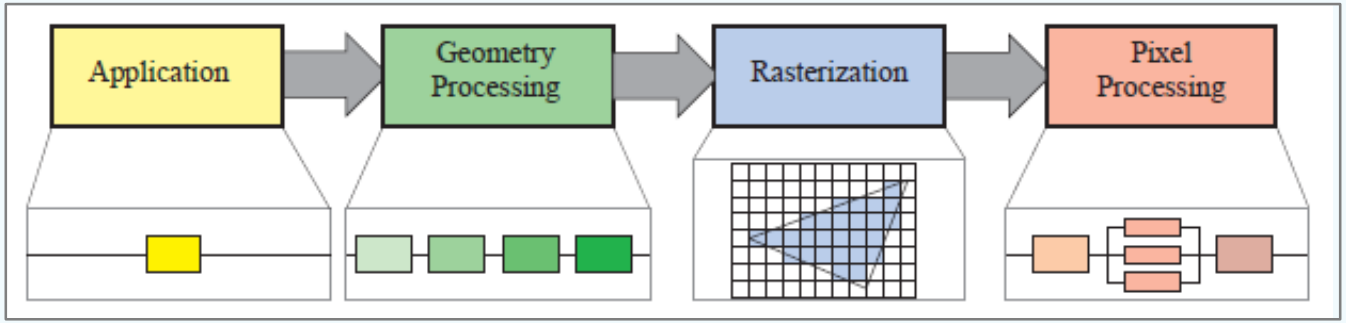



**Architecture (Part 1):**

- The pipeline consists of many stages, each of which performs part of a largest task.
- The stages executes in parallel, with each stage dependent upon the result of the previous stages. Such parallelism leads to increase in performance.

Image taken from Figure 2.2 – Real-time Rendering (4th Edition)

# The Graphics Pipeline

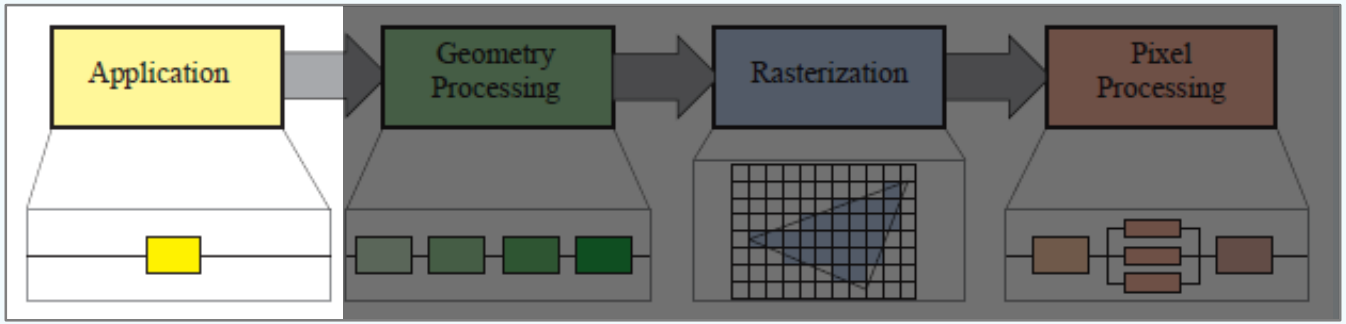



**Architecture (Part 2):**

- Above figure shows a coarse division of pipeline in 4 main stages:
  1. **Application** – Software running on CPU
  2. **Geometry Processing** – All types of geometry handling in GPU
  3. **Rasterization** – Generates pixels from primitive in GPU
  4. **Pixel Processing** – Performs per-pixel operations and filtering in GPU.

Image taken from Figure 2.2 – Real-time Rendering (4th Edition)

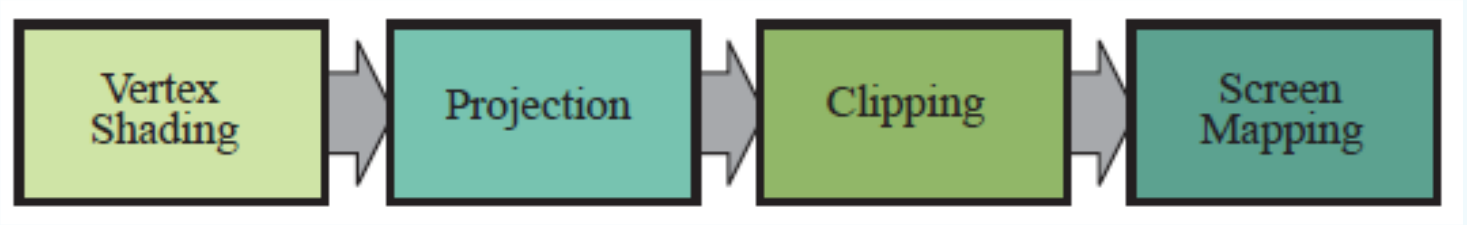
# Application Stage




- The developer has full control over the application implementation in this stage.
- The application determines what primitive(i.e. the most basic geometry) to be rendered and feeds them to the geometry processing stages.
- **Common tasks** : collision detection, acceleration algorithms, animations, physics simulation, etc.

Image taken from Figure 2.2 – Real-time Rendering

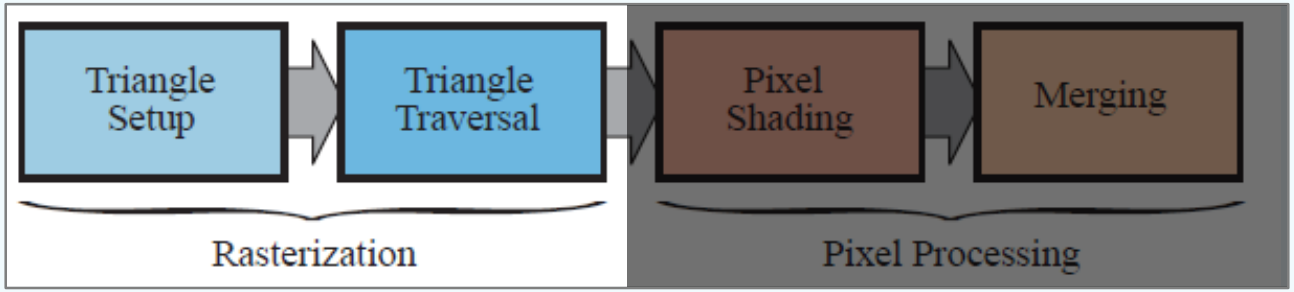
# Geometry Processing Stage

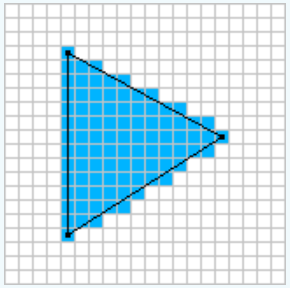



- Responsible for most per-triangle and per-vertex operations
- Further divided into 4 functional stages:
  1. Vertex Shading
  2. Projection
  3. Clipping
  4. Screen Mapping

Image taken from Figure 2.3 – Real-time Rendering (4th Edition)

# Rasterization Stage

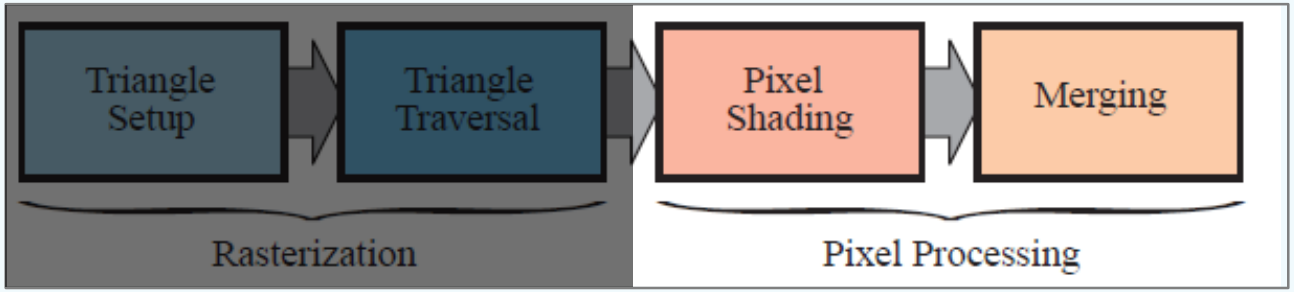


- Conversion from two-dimensional vertices in screen-space (z-value still maintained) into pixels on screen.
- Further divided into 2 functional stages:
  1. Triangle Setup / Primitive Assembly
  2. Triangle Traversal

Image taken from Figure 2.3 – Real-time Rendering (4th Edition)

# Pixel Processing Stage

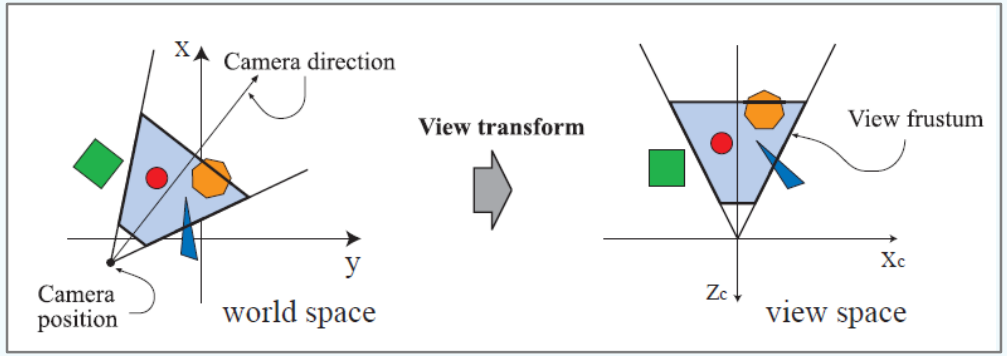



- Performs per-pixel or per-sample computations and operations are performed on pixels or samples that are inside a primitive.
- Further divided into 2 functional stages:
  1. Pixel Shading
  2. Merging

Image taken from Figure 2.3 – Real-time Rendering (4th Edition)

# Geometry Processing

# Geometry Processing | Vertex Shading



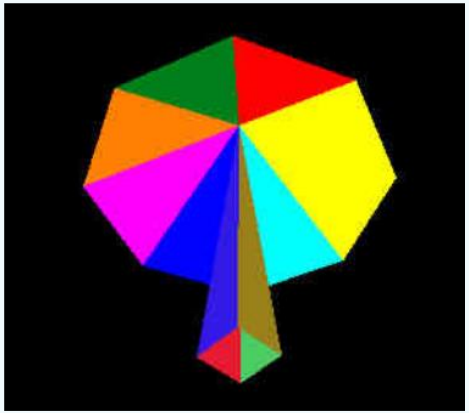

**Task #1: Compute vertex position**

- Vertex is defined as a point in space. Its position is described by a set of coordinates
- Vertex is transformed into several different spaces or coordinate systems:

*Model space -> World space -> View space*

Image taken from Figure 2.4 – Real-time Rendering (4th Edition)

# Geometry Processing | Vertex Shading

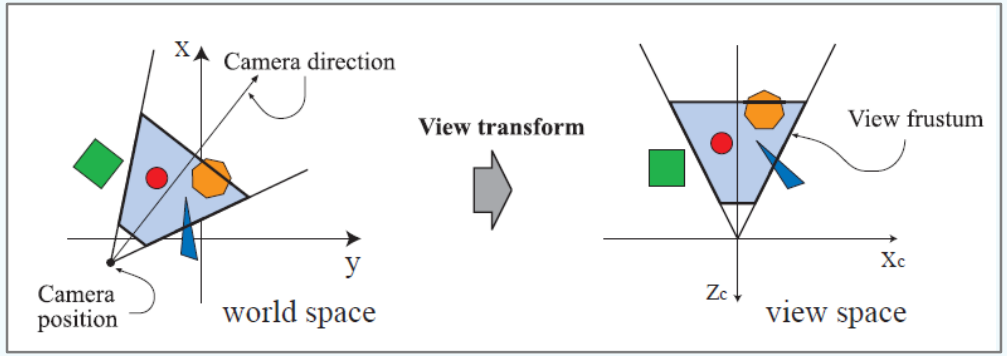


**Task #2: Compute vertex output data**

- Example: normal vector, texture coordinates.
- Encode the appearance information of an object at a vertex.
- With modern GPU, this stage has become more general. Programmers can set associate any data to each vertex.

Image taken from Lab02 of previous TGD2151 materials

# Geometry Processing | Projection




**Task: Projects 3D vertices from camera space to clip space.**

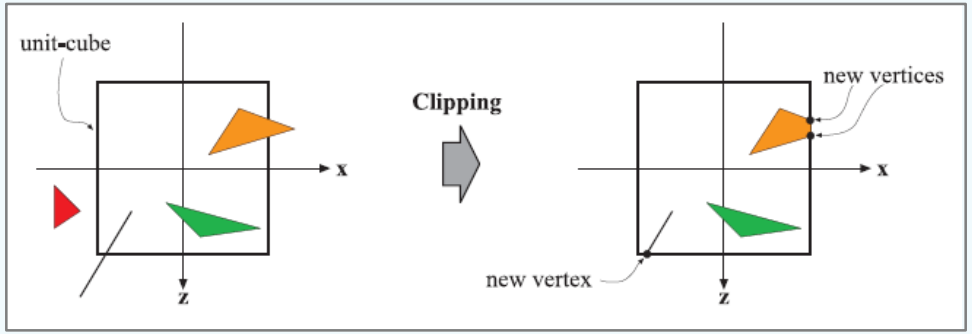
- Involves projective transformation:

  *Camera space -> Clip space*

- Example: Orthographic projection, perspective projection, etc.

Image taken from Figure 2.2 – Real-time Rendering (4th Edition)

# Geometry Processing | Clipping



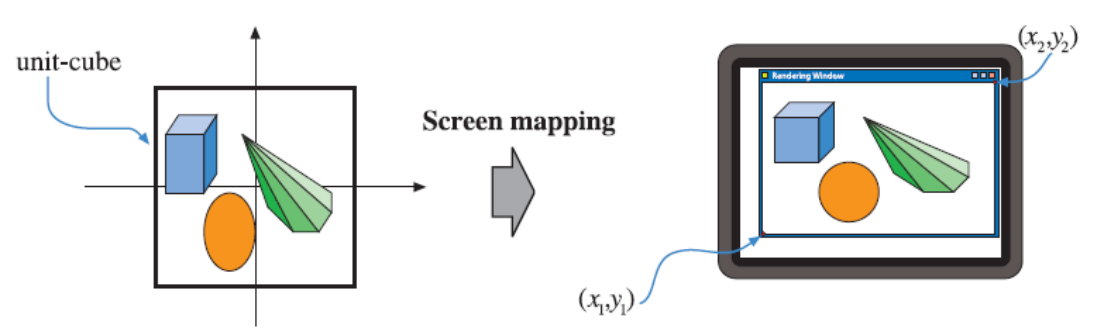

**Task: Primitive Filtering**

- Allows primitive wholly or partially inside the view volume to pass on to the next stage.

*Clip space -> Normalized device coordinate space*

Image taken from Figure 2.6 – Real-time Rendering (4th Edition)

# Geometry Processing | Screen Mapping




**Task: Maps primitives inside the view volume to screen coordinate**

- The corners of the view volume are mapped to the corners of a window.

*Normalized device coordinate space -> Screen coordinate*

- Z-coordinate is mapped to [0, 1].
- The coordinate values are still in floating point.

Image taken from Figure 2.7 – Real-time Rendering (4th Edition)

# Rasterization

# Rasterization

**Triangle Setup**

- Convert a vertex stream into a sequence of base primitives.
- Edge equations are also computed for triangle traversal and for interpolation.

**Triangle Traversal**

- Each pixel that has its centre covered by the primitive is checked.
- A ***fragment*** is generated for the part of the pixel that overlaps the triangle.
- Pixels inside a primitive are sent to the Pixel Processing stage.

# Pixel Processing

# Pixel Processing

**Pixel Shading**

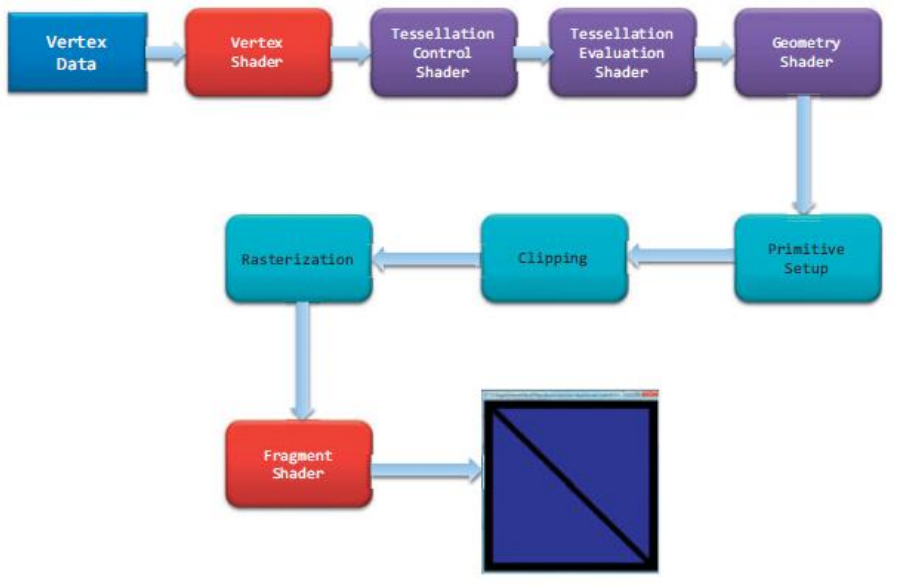
- Performs per-pixel data computation (e.g. texture mapping)
- Outputs one or more colours.

**Merging**

- Pixel information are stored in *colour buffer*.
- Combine colour of pixels from Pixel Shading with the corresponding pixels in the colour buffer.
- Z-buffering is performed to determine which pixel is actually visible (i.e. not occluded by pixels of other primitives).

# Modern OpenGL Graphics Pipeline

# Modern OpenGL Graphics Pipeline

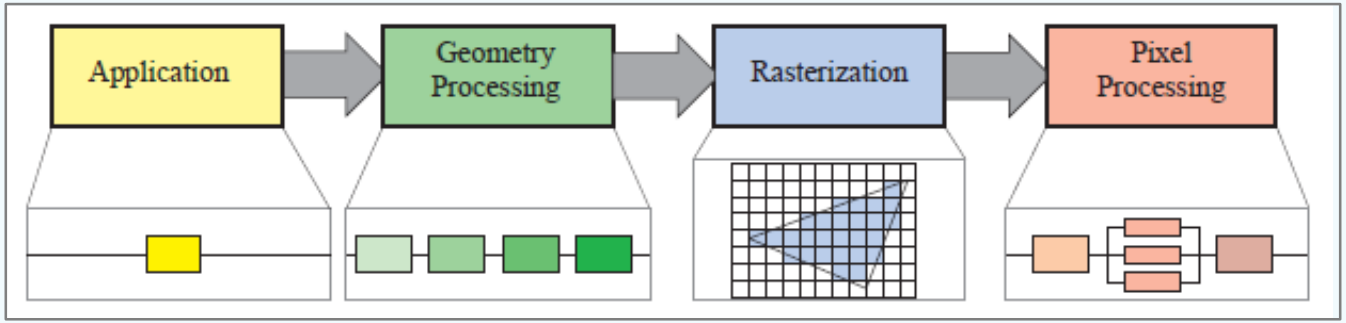



**Programmable Stages:**

1. Vertex Shader
2. Tessellation Control Shader *(optional)*
3. Tessellation Evaluation Shader *(optional)*
4. Geometry Shader *(optional)*
5. Fragment Shader

Image taken from Figure 1.2 – OpenGL Programming Guide (8th Edition) – The Official Guide to Learning OpenGL Version 4.3

# Modern OpenGL Graphics Pipeline




- Vertex Shader
- Tessellation Control Shader
- Tessellation Evaluation Shader
- Geometry Shader

- Fragment Shader

Image taken from Figure 2.2 – Real-time Rendering (4th Edition)

# Q & A

# Acknowledgement


- This presentation has been designed using resources from PoweredTemplate.com