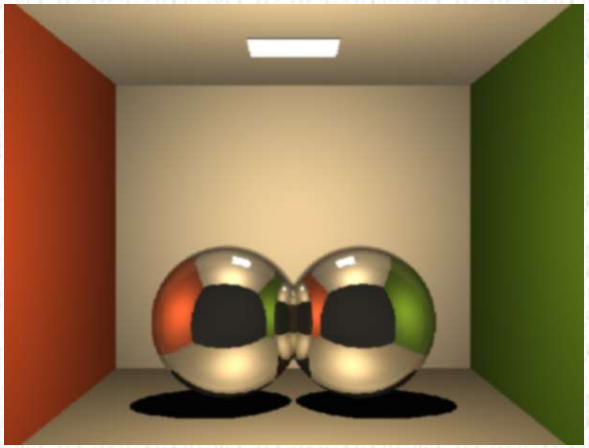
Introduction to OpenGL

Kostas Vardis

http://graphics.cs.aueb.gr/graphics/people.html

Athens University of Economics and Business

Computer Graphics BSc



Graphics API's

- Need interface to render elements on the screen

Most common are:

- Direct3D (part of DirectX) used mainly in games
- OpenGL (controlled by the Khronos Group) used in games, CAD, visualization, etc.

What is OpenGL??

- A software interface to graphics hardware
- Implemented in device driver
- Cross-platform (Windows, Mac OS, Linux, Mobile)
- Hardware-independent (Nvidia, ATI)
- Exposed to many languages (C is the most common)
- Can be used with other languages through bindings, e.g. Java through bindings (JOGL, Java3D)

What OpenGL does not do

- OpenGL is strictly for rendering!
- Does not create models
- Does not create windows
- Does not handle input
- Does not handled audio

FREEGLUT

- Handles windows, input, menus
- Uses callback functions
- Good for basic OpenGL programs

FREEGLUT

- glutInit/ glutInitWindowPosition/glutInitWindowSize/glutInitDisplayMode
- glutDisplayFunc for drawing
- glutPostRedisplay for redrawing
- glutReshapeFunc for resizing
- glutKeyboardFunc for key input
- glutMouseFunc for mouse input
- glutMainLoop to start!
- glutSwapBuffers for double buffering

OpenGL- How does it work?

- OpenGL is a state machine!!

```
// Enable depth test
glEnable(GL_DEPTH_TEST);
// Disable depth test (if this is not called, depth testing will remain enabled!)
glDisable(GL_DEPTH_TEST);
```

- Contains more than 300 function calls
- Function Syntax: glUniform3fv

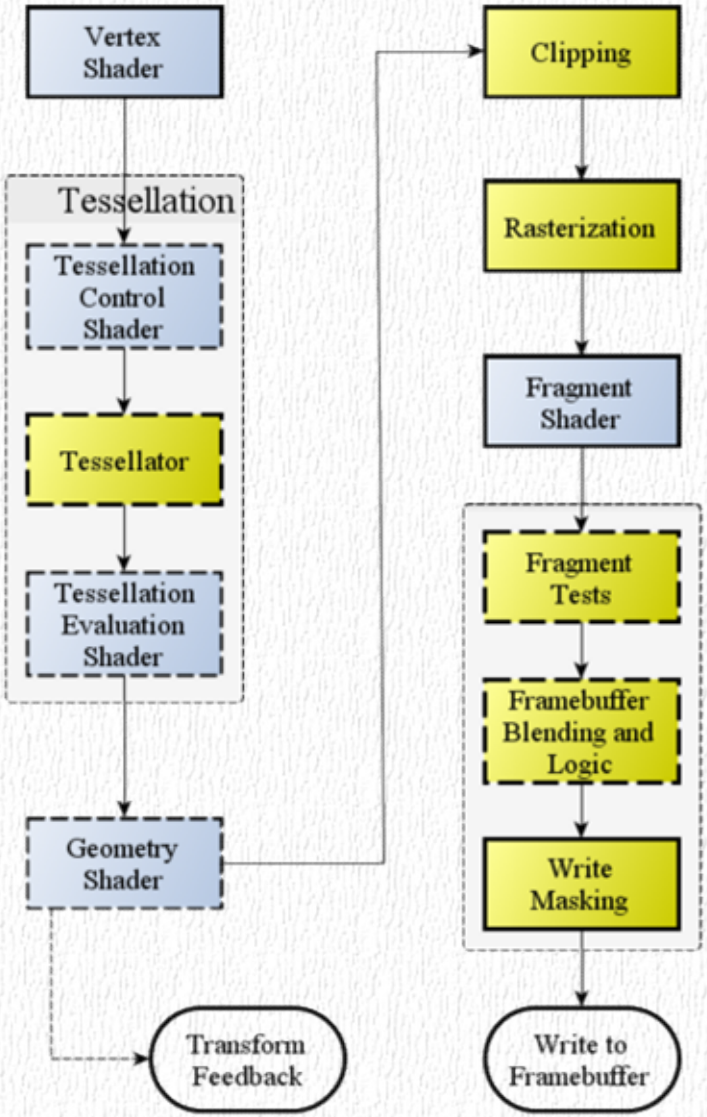
Example:

```
// pass to shader uniform variable 0, the vec3 (0.0, 0.0, 1.0)
glUniform3f(0, 0.0f, 0.0f, 1.0f);
// can also pass as a pointer to a vector or array
GLfloat color_data[] = {0.0f, 0.0f, 1.0f };
glUniform3fv(0, color_data);
```

The Pipeline

Basic OpenGL 4 pipeline:

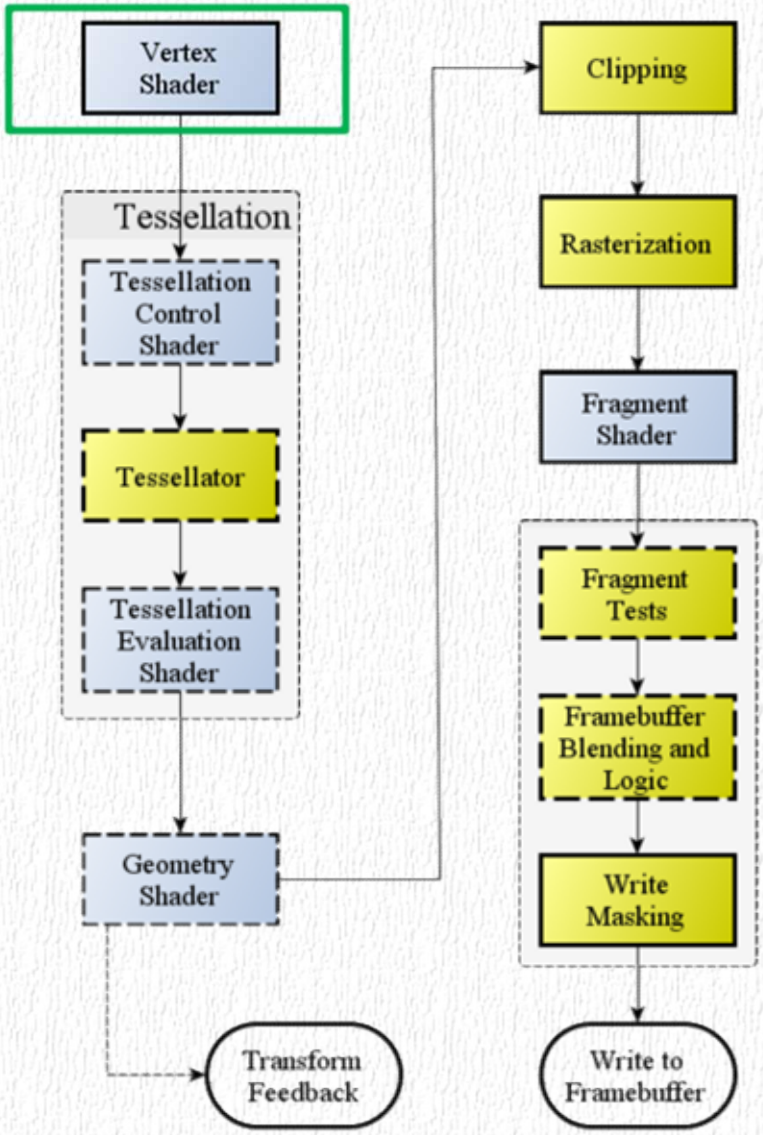
- Per-vertex processing
- Vertex shader
- Primitive Assembly
- Optional primitive tessellation
- Optional geometry shader processing
- Clipping
- Rasterization
- Fragment shader
- Post-fragment operations



The Pipeline

Vertex shader:

- Transformations at the vertex level
- Positions, Normals, Texture coordinates
- Per-Vertex Lighting



The Pipeline

Primitive Assembly:

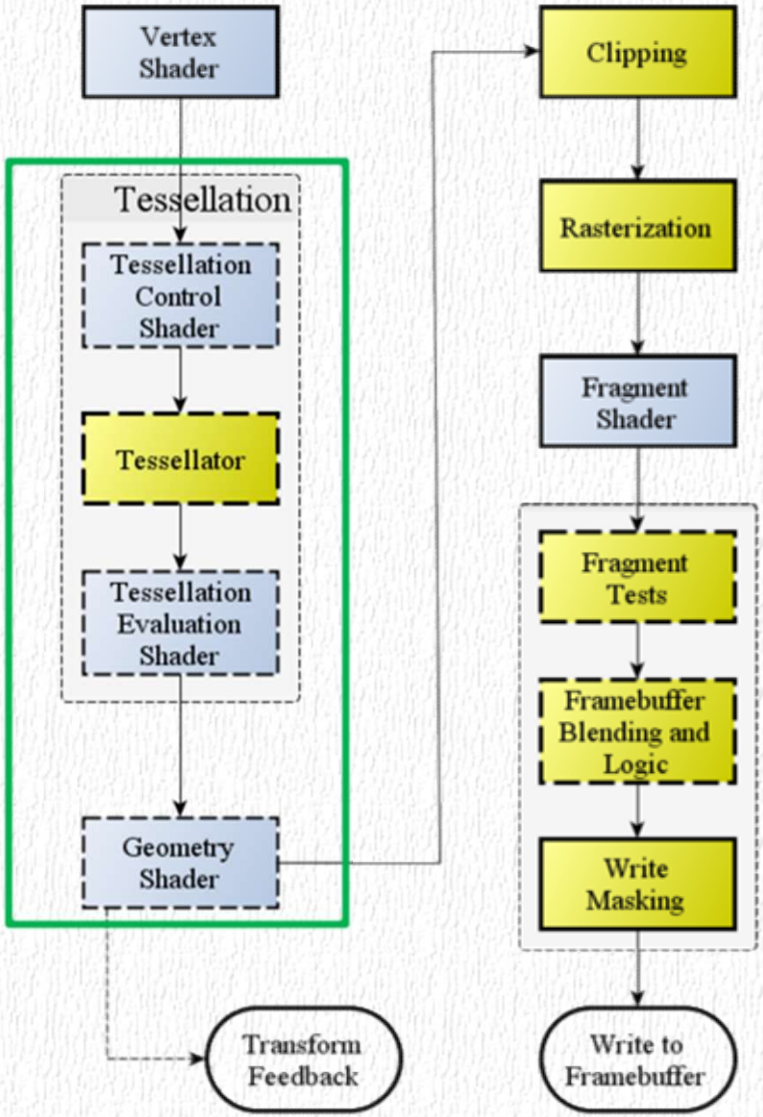
- Groups vertices into primitives

Tesselation shader:

- Optional
- Receives patches and subdivides them

Geometry shader:

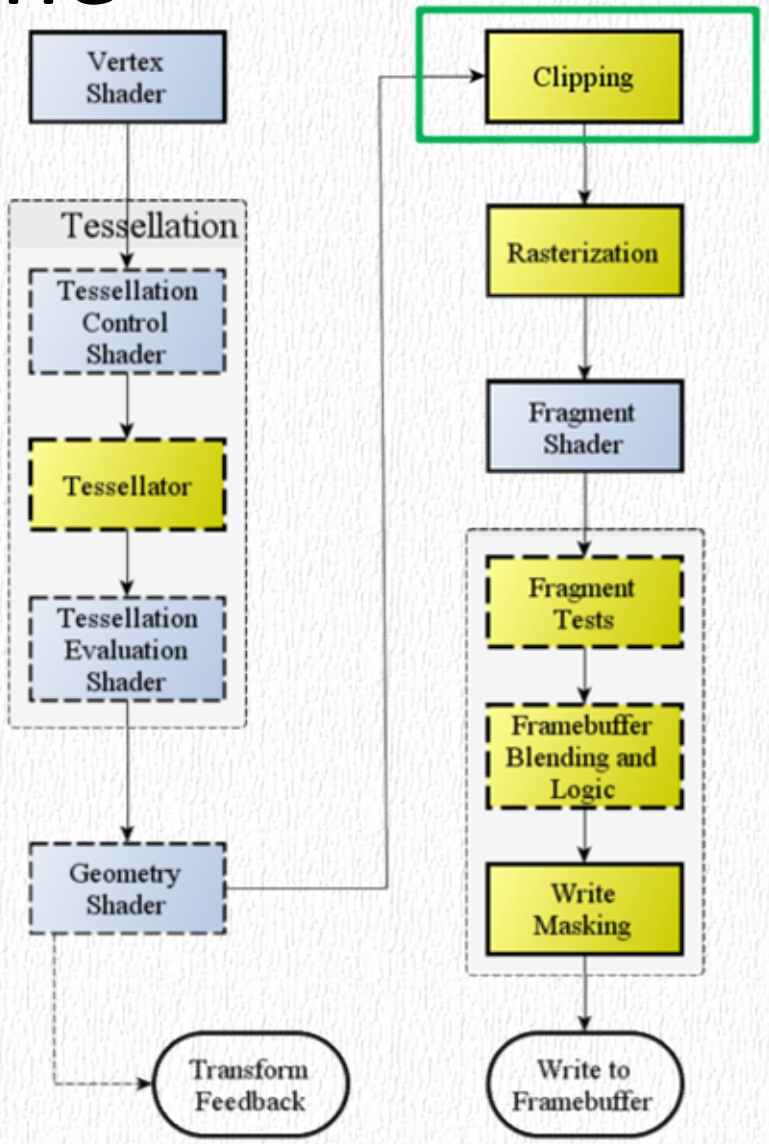
- Optional
- Receives primitives
- Outputs zero or more primitives



The Pipeline

Primitive Operations:

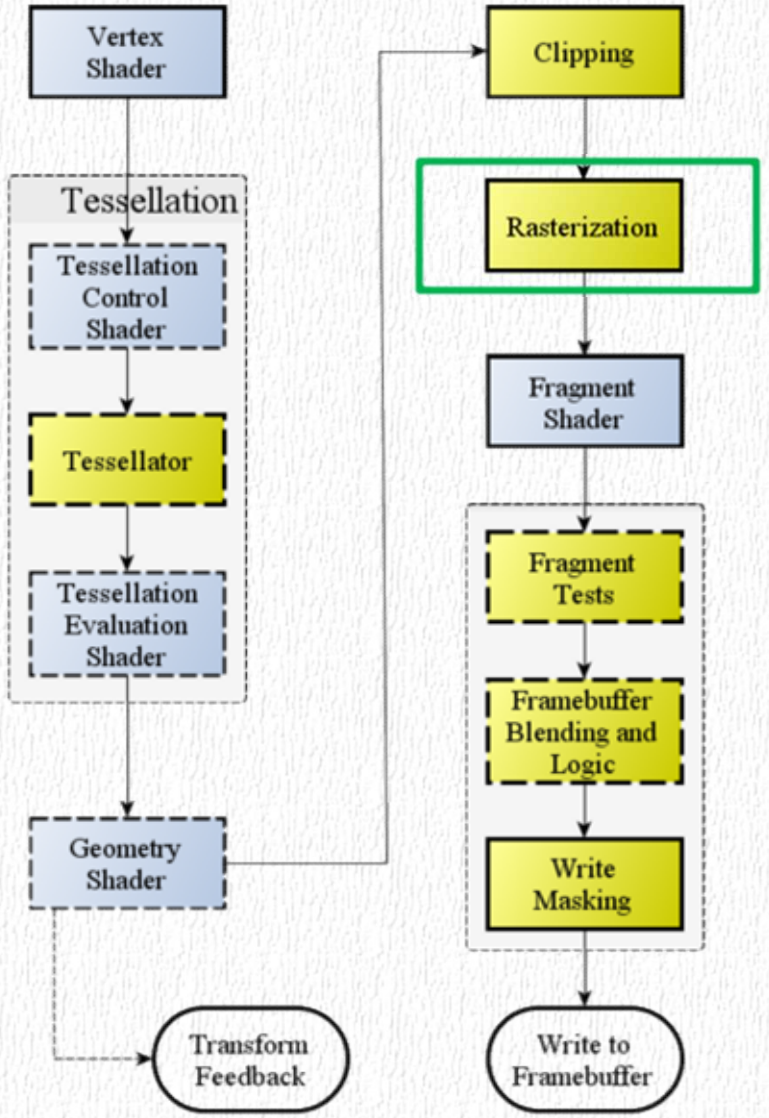
- Clipping
- Perspective Division
- Viewport transformations
- Face culling



The Pipeline

Rasterization

- Input -> Transformed and projected primitives
- Output -> Fragments



The Pipeline

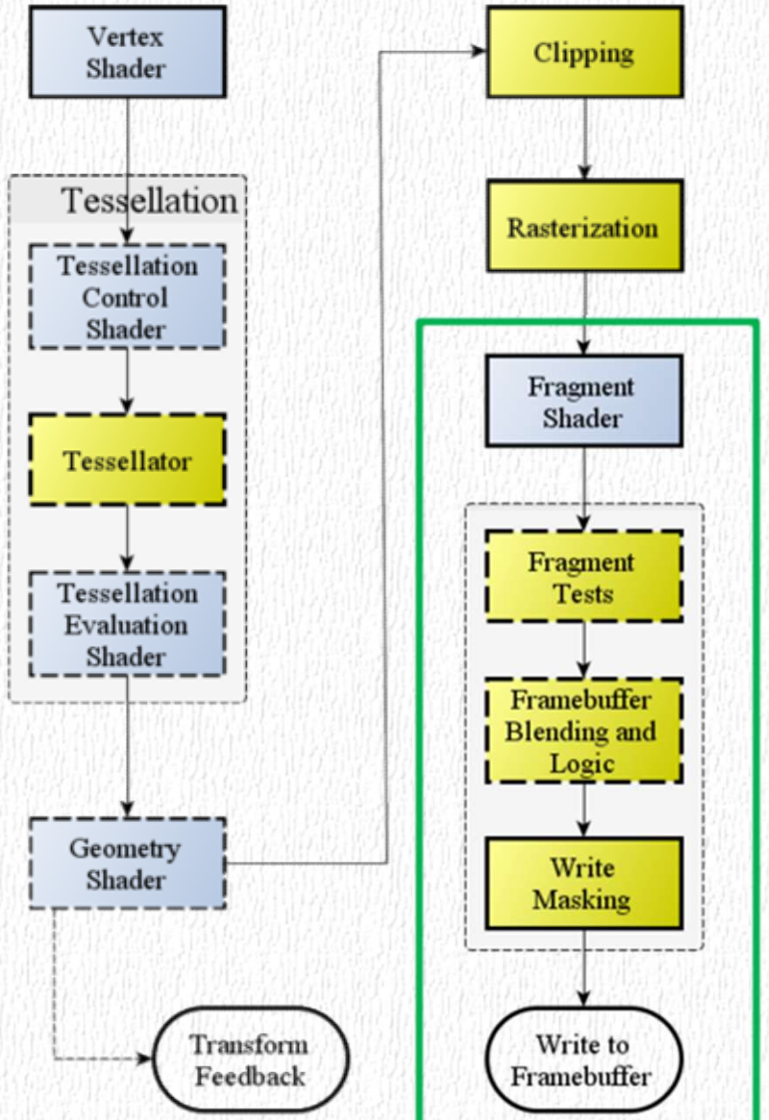
Fragment shader

- Processes data for each fragment
- Writes colors, depth, etc.

Per-Sample operations

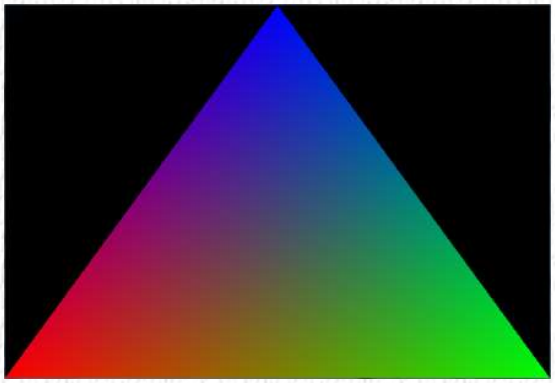
- Stencil test
- Depth test
- Blending

Final color output



Rendering of a triangle

```
// First this, we clear our depth and color buffers.
// We can clear both using an | operand to create the buffer-clear mask.
glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
glClearDepth(1.0f);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
// Set the basic matrices
glm::mat4x4 mvp = glm::ortho<GLfloat>(-1, 1, -1, 1, -1, 1);
// Set the shader active
glUseProgram(shader_obj);
// Draw a triangle
glBindVertexArray(vao_triangle);
glUniformMatrix4fv(uniform_mvp, 1, false,
&model_view_projection[0][0]);
glDrawArrays(GL_TRIANGLES, 0, 3);
```



Done!

- Check lab1 project

- Check pdf for more information and online content