Ray Tracing

Rasterization vs. Ray

Rasterization

for each primitive

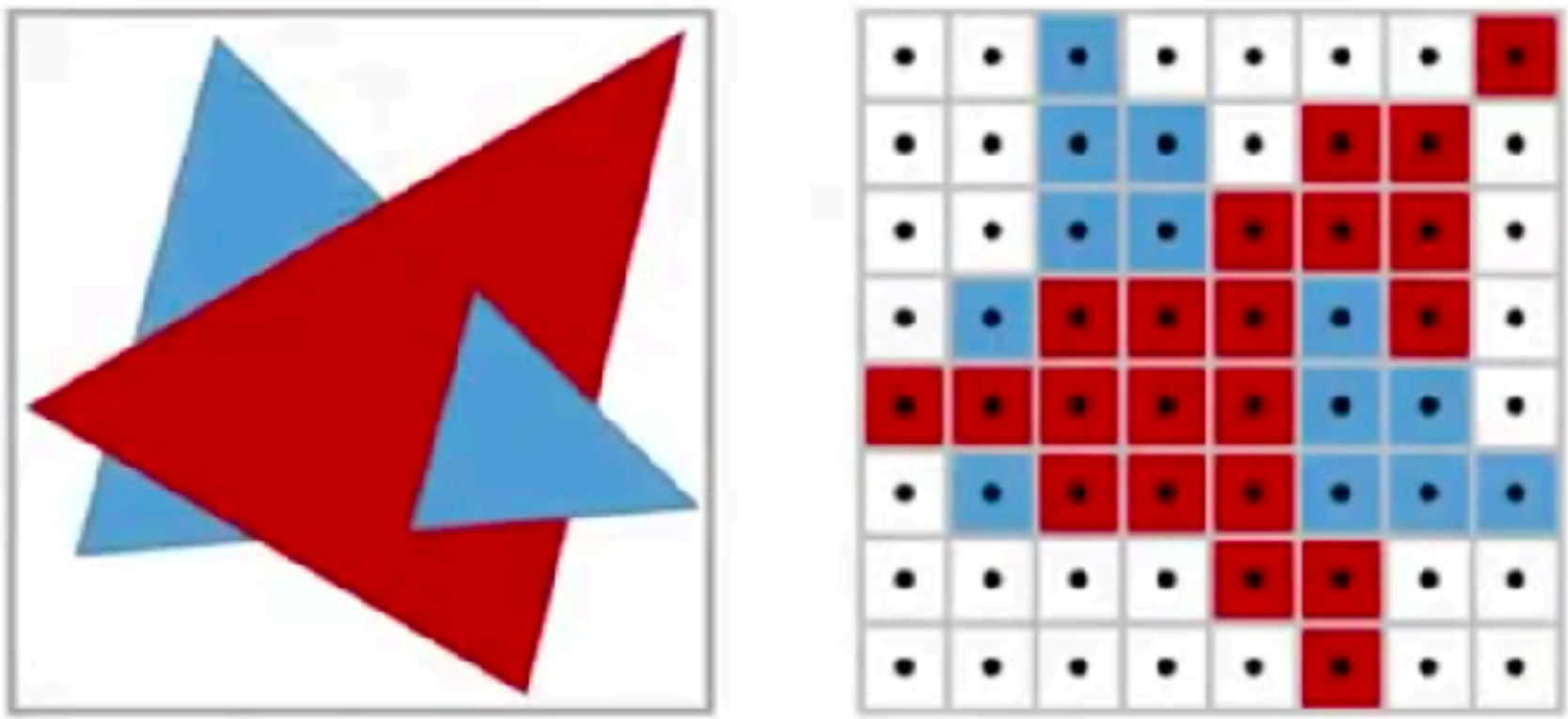
find pixel samples

Ray Tracing

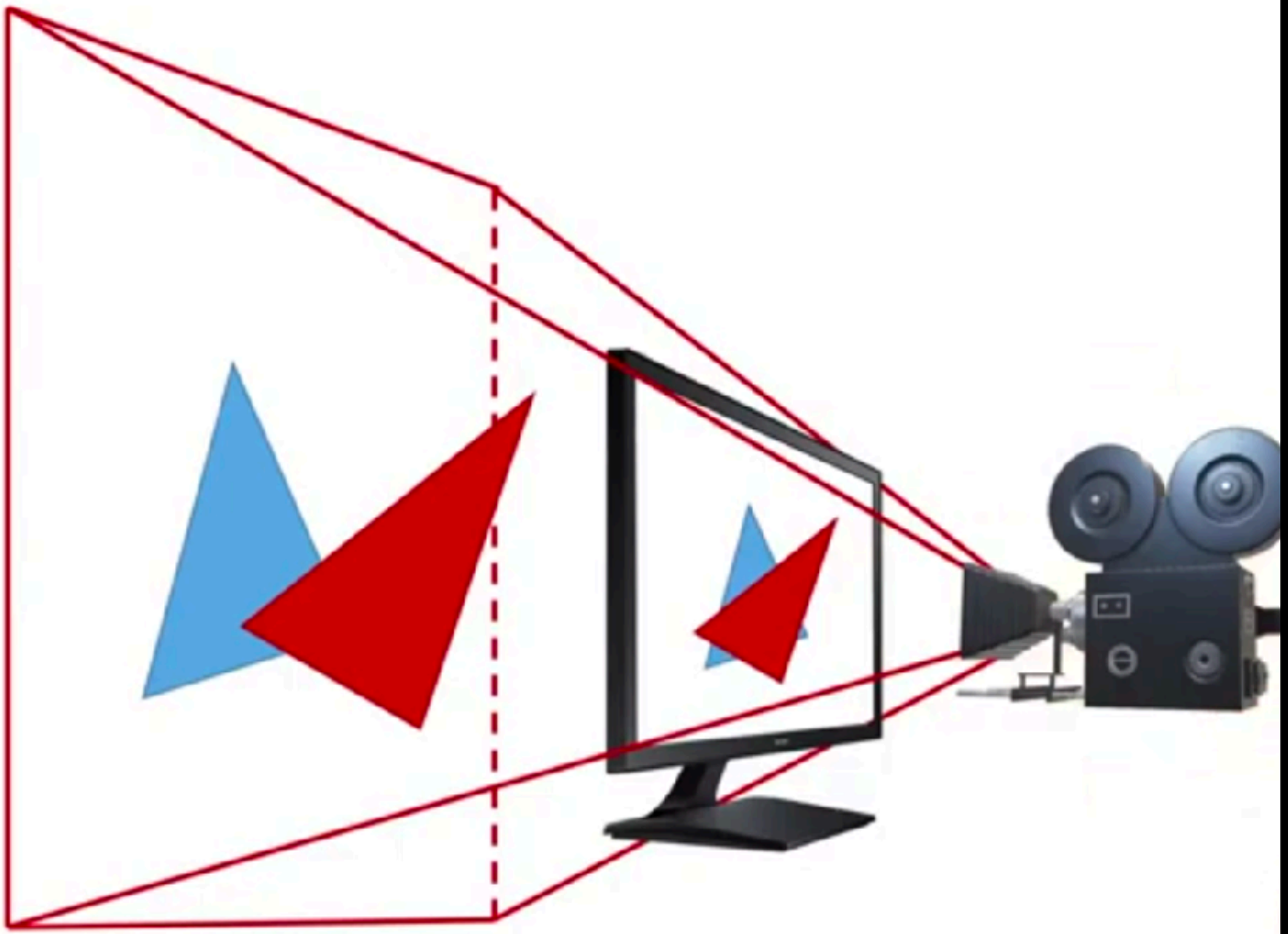
for each pixel sample

find the *closest* primitive

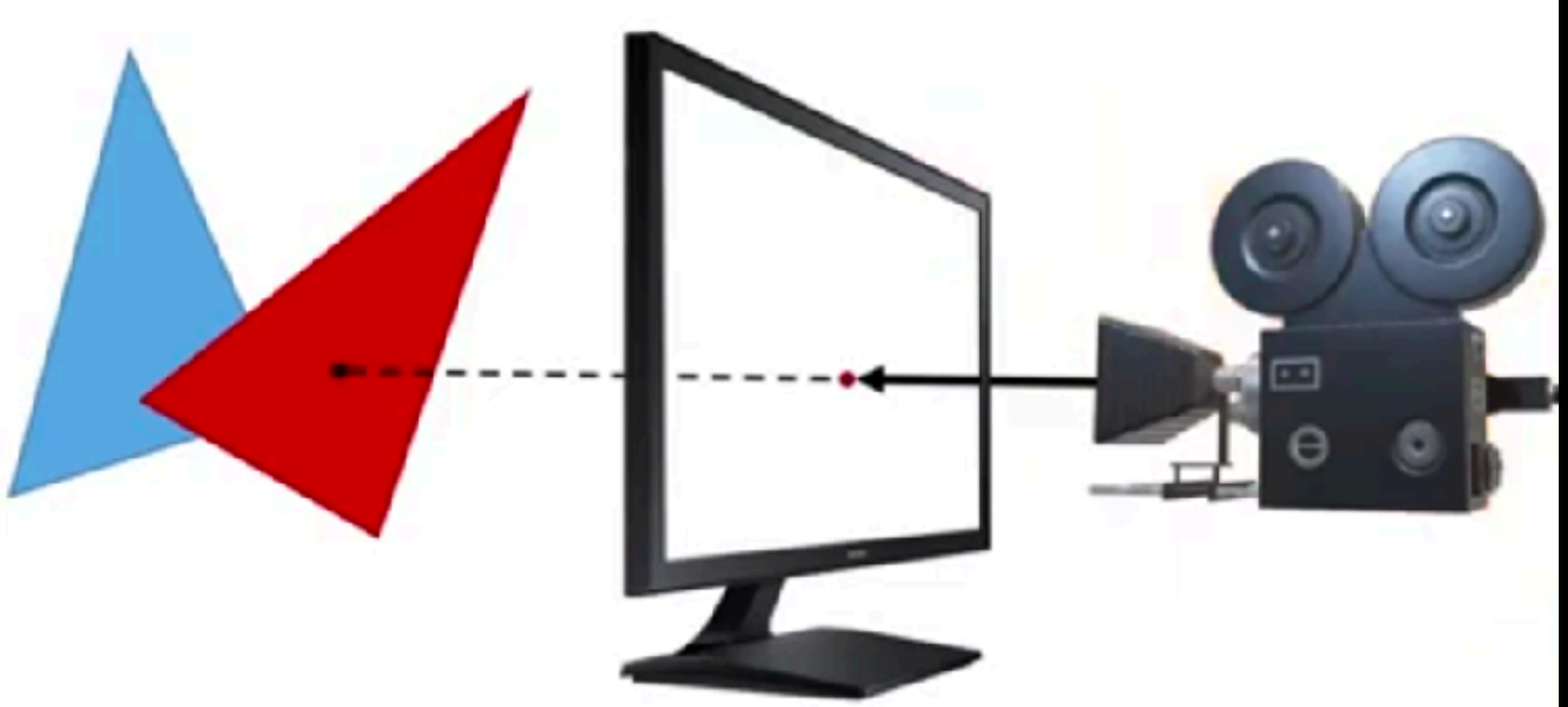
Rasterization vs. Ray Tracing



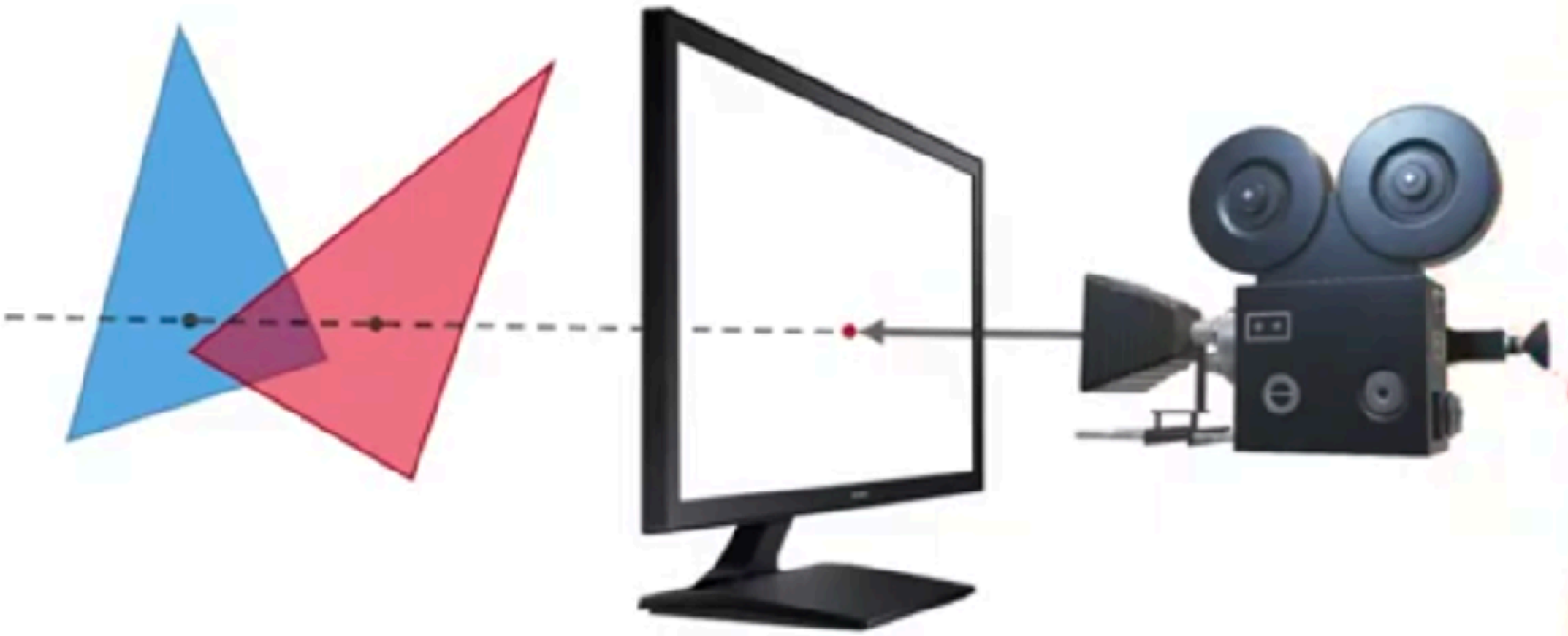
Rasterization



Ray Tracing



Ray Tracing

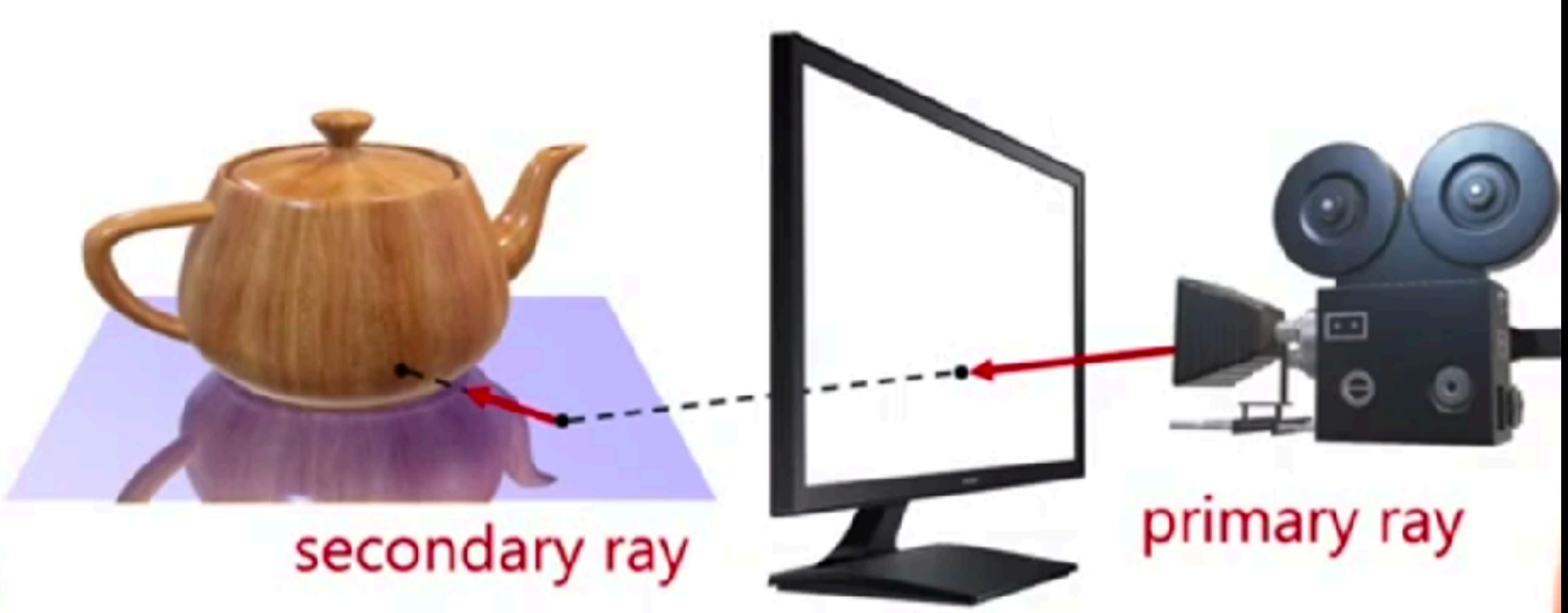


Ray Tracing

Ray Tracing

Secondary Rays:

reflections, refractions, shadows,
realistic illumination, …



Rasterization vs. Ray Tracing

Rasterization

for each primitive

find pixel samples

linear memory access

fast

linear complexity

Ray Tracing

for each pixel sample

find the *closest* primitive

random memory access

slow

logarithmic complexity

Ray Tracing

Independence Day 2 – ©2016 20th Century Fox by Scanline VFX

Rasterization + Ray

Rasterization

for primary visibility

Ray Tracing

for secondary effects:

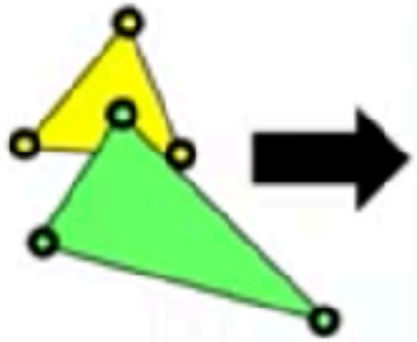
- reflections/refractions
- shadows
- realistic illumination

...

Introduction to Modern OpenGL

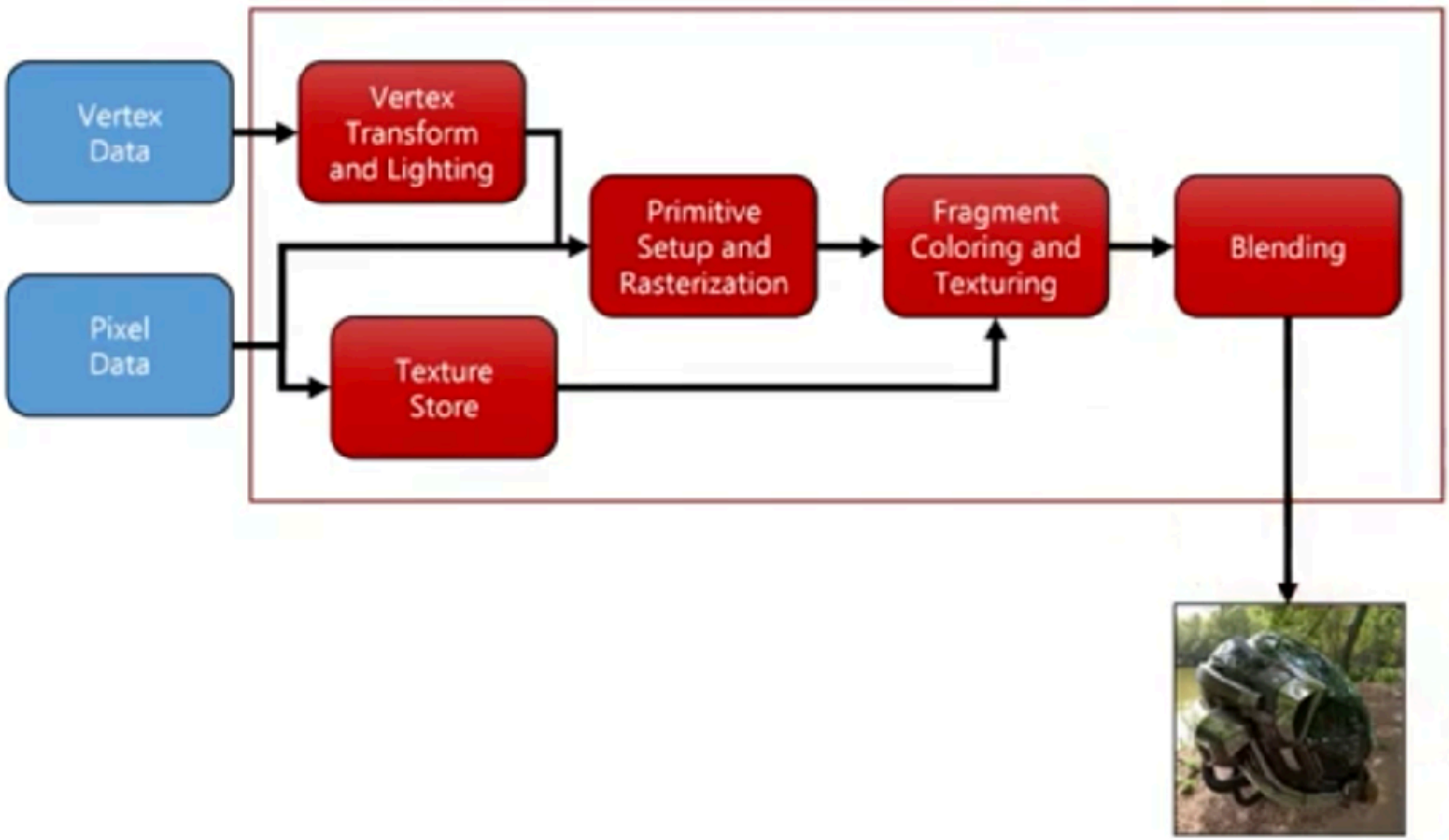
GPU Pipeline

GPU Pipeline

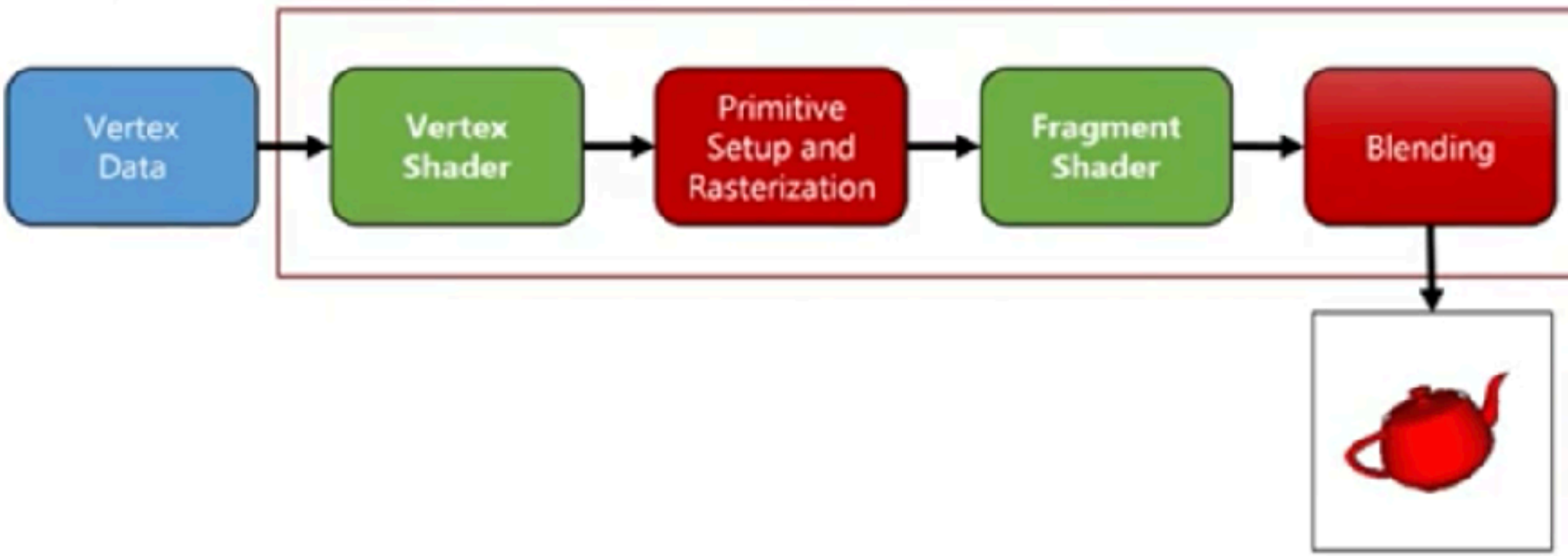


GPU

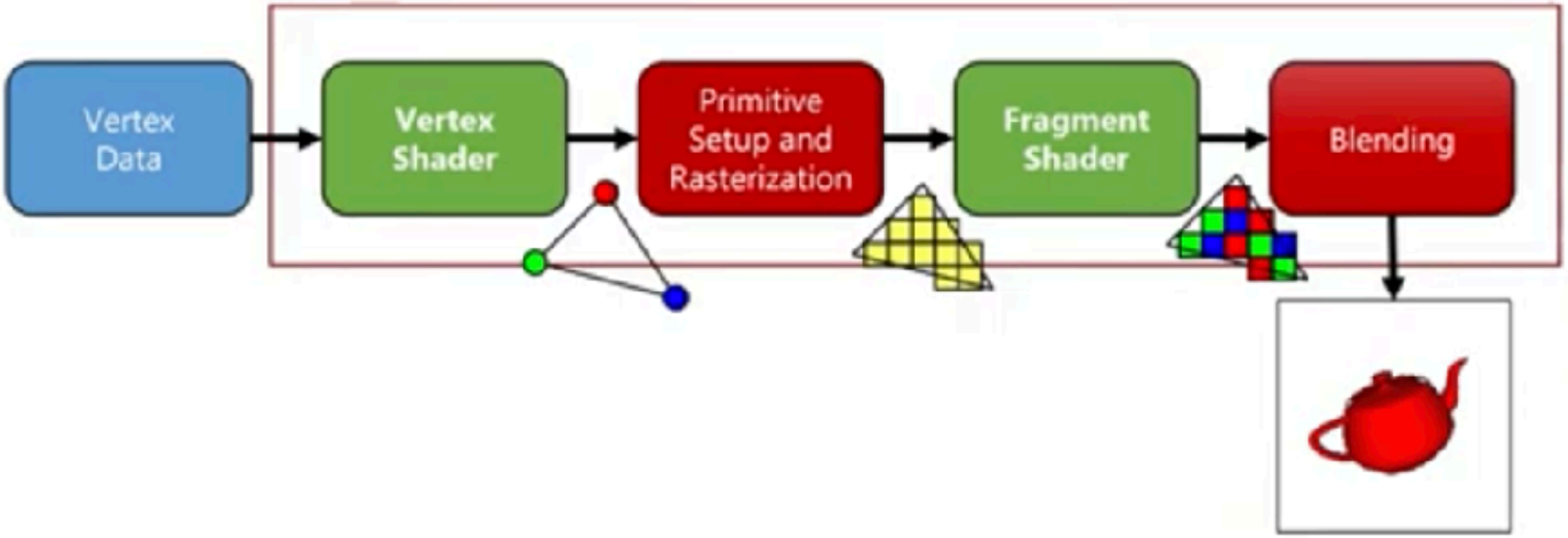
GPU Pipeline (Fixed Function)



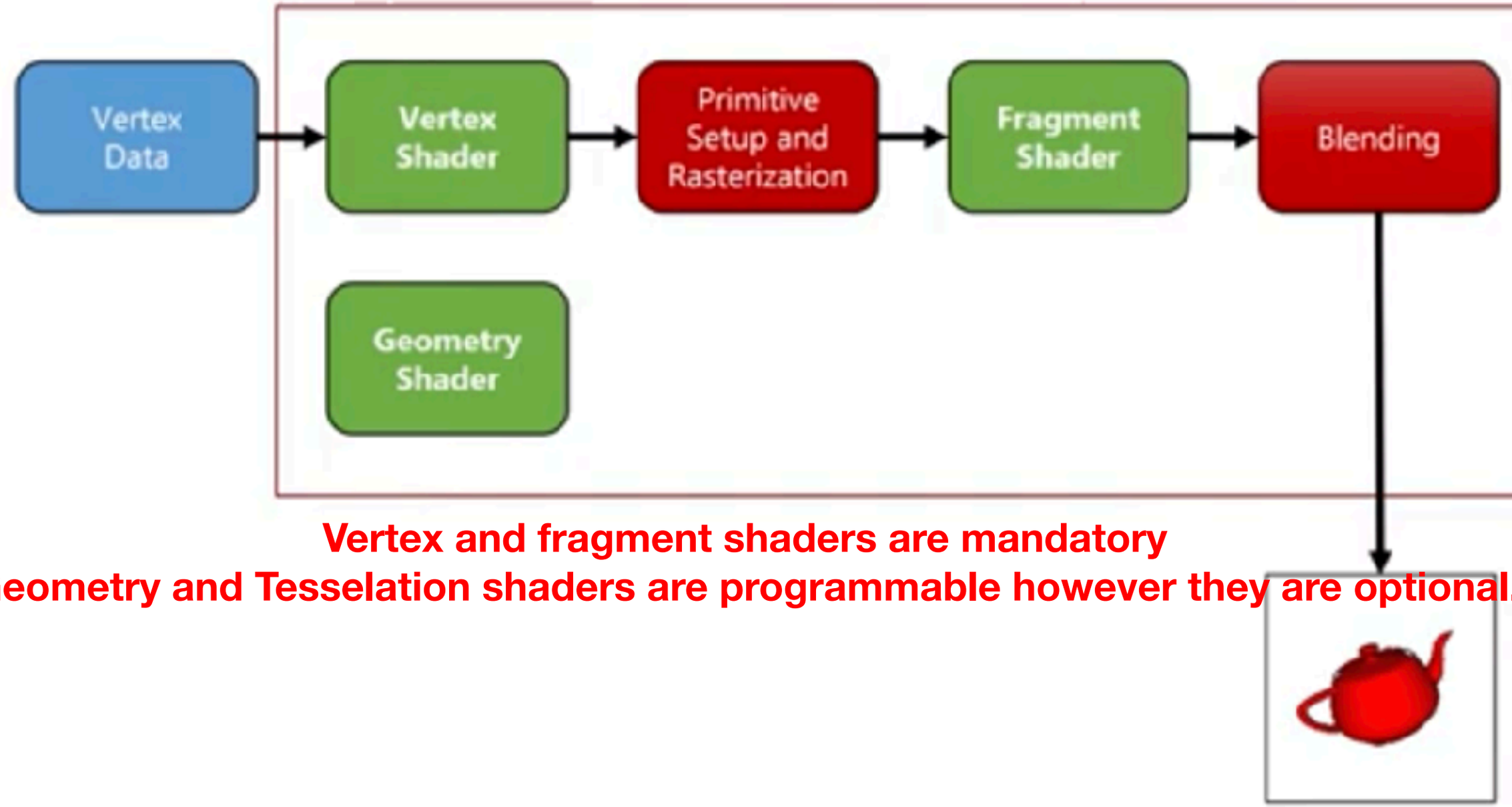
GPU Pipeline



GPU Pipeline



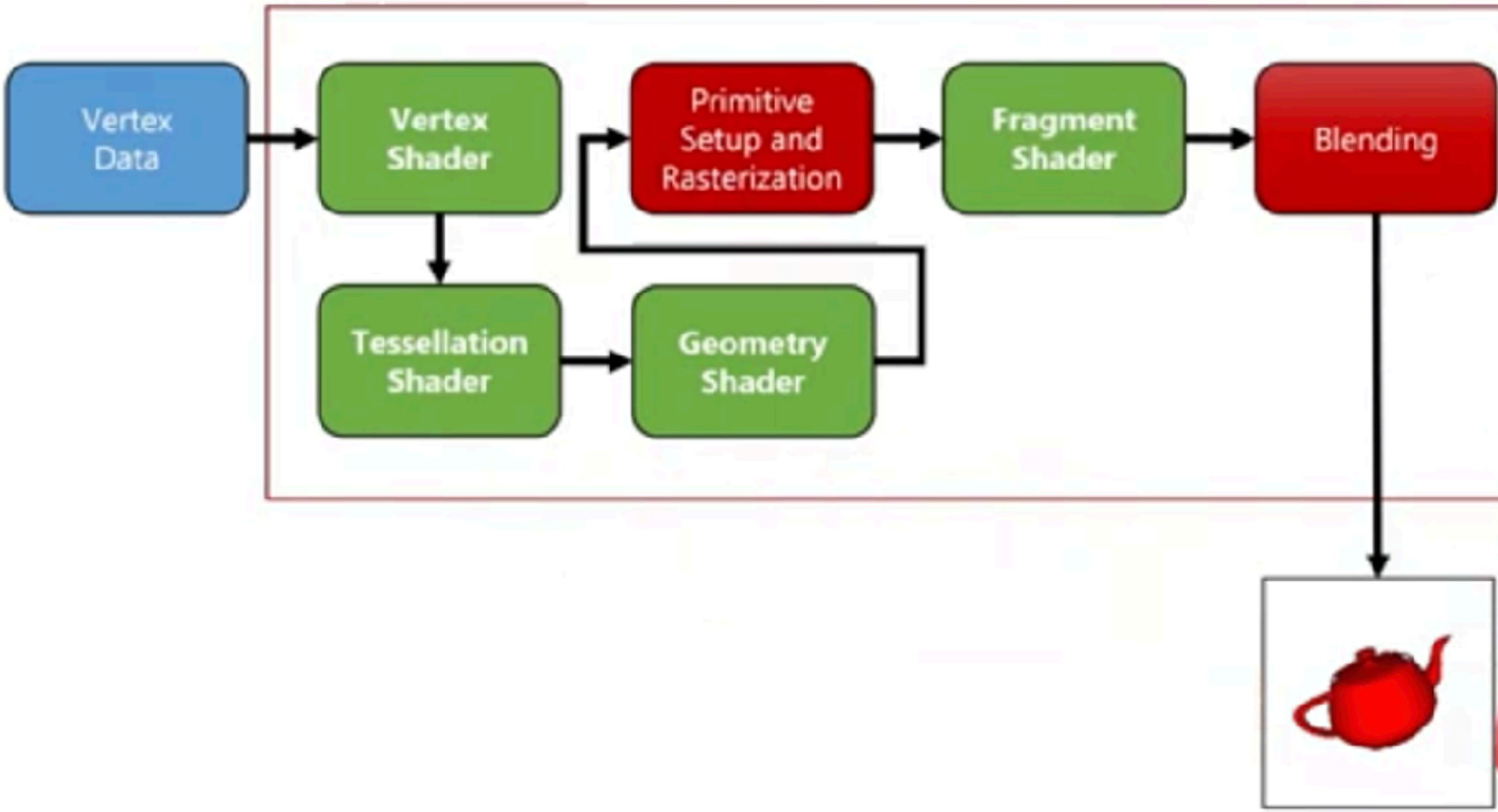
GPU Pipeline



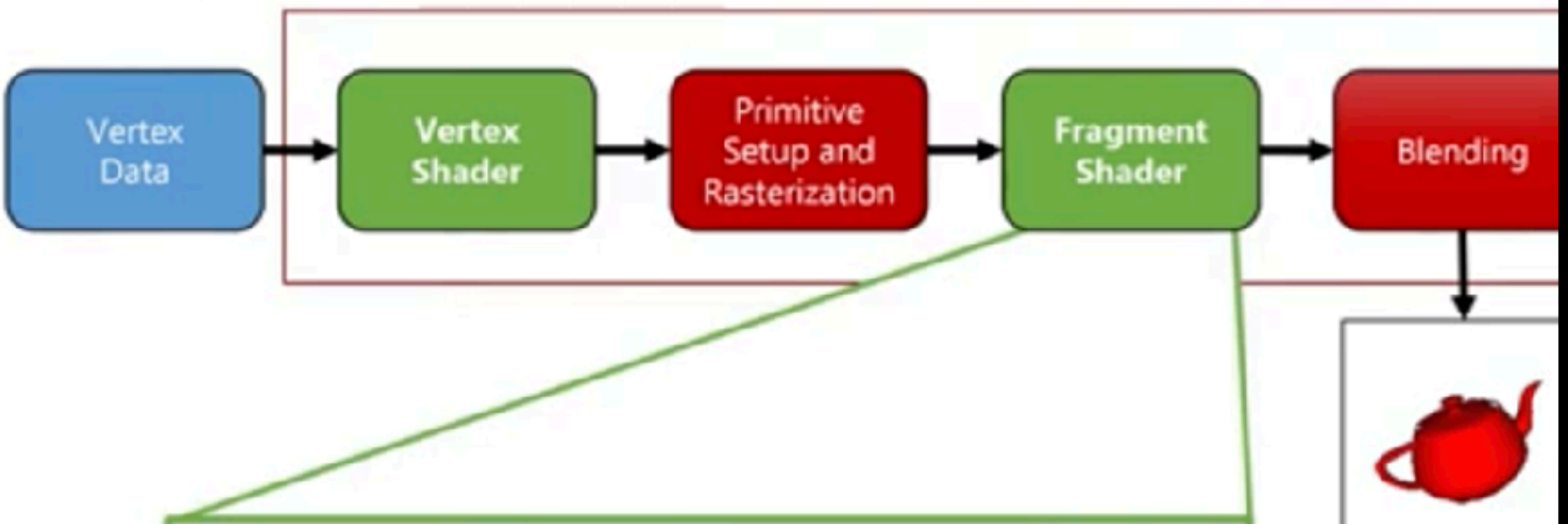
Vertex and fragment shaders are mandatory

Geometry and Tesselation shaders are programmable however they are optional.

GPU Pipeline



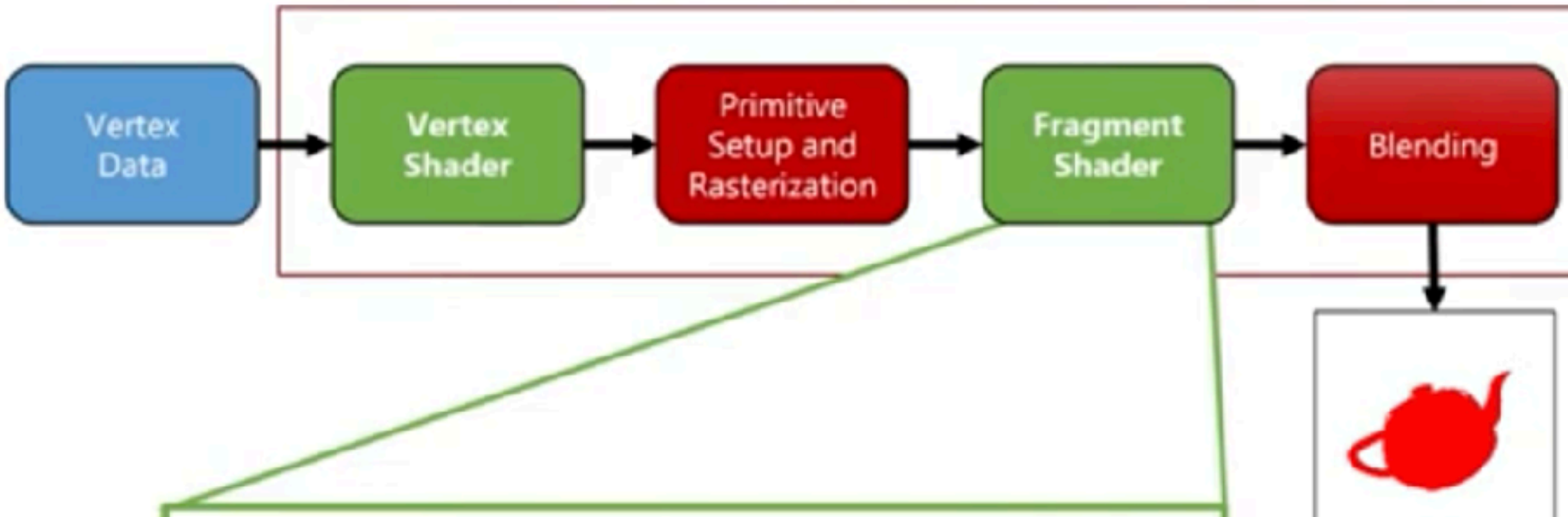
GPU Pipeline



GLSL

```
#version 330 core

layout(location = 0) out vec4 color;

void main()
{
    color = vec4(1,0,0,1);
}
```

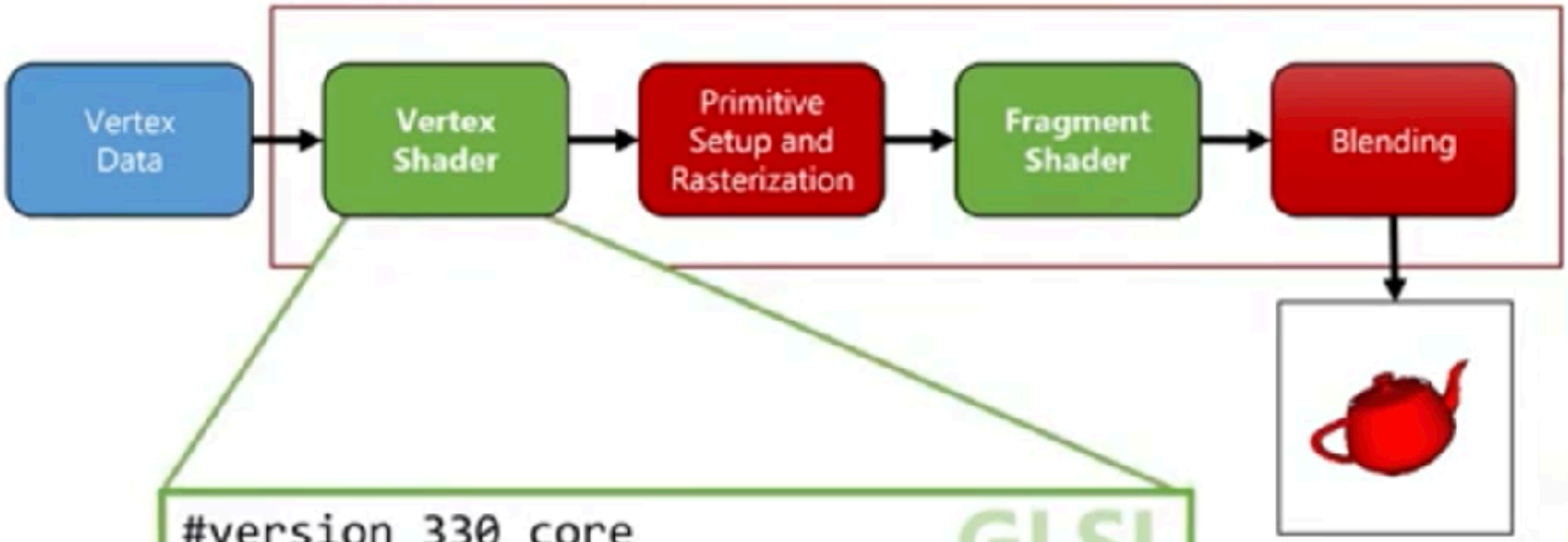
GPU Pipeline



GLSL

```
#version 330 core

layout(location = 0) out vec4 color;

void main()
{
    color = vec4(1,0,0,1);
}
```

GPU Pipeline



GLSL

```
#version 330 core

layout(location=0) in vec3 pos;

uniform mat4 mvp;

void main()
{
    gl_Position = mvp * vec4(pos,1);
}
```

Compile Shaders

```
GLuint progam = glCreateProgram();

// Compile vertex shader
GLuint vs;
...

// Compile fragment shader
GLuint fs;
...
```

Compile Shaders

```
GLuint progam = glCreateProgram();

// Compile vertex shader
char *vsSource = ReadFromFile("shader.vert");
GLuint vs = glCreateShader();
glShaderSource( vs, 1, vsSource, nullptr );
glCompileShader( vs );
glAttachShader( program, vs );
delete [] vsSource;
```

Compile Shaders

```
GLuint progam = glCreateProgram();

// Compile fragment shader
char *fsSource = ReadFromFile("shader.frag");
GLuint fs = glCreateShader();
glShaderSource( fs, 1, fsSource, nullptr );
glCompileShader( fs );
glAttachShader( program, fs );
delete [] fsSource;
```

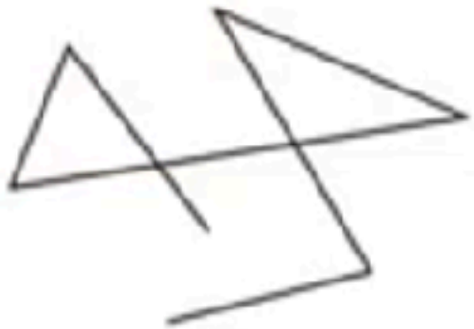
Compile Shaders

```
GLuint progam = glCreateProgram();

// Compile vertex shader
...

// Compile fragment shader
...

glLinkProgram( program );
```

Primitives

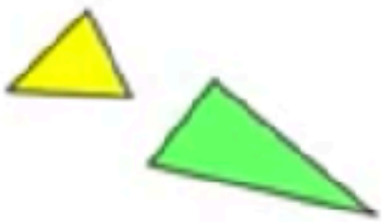
GL_POINTS

GL_LINES

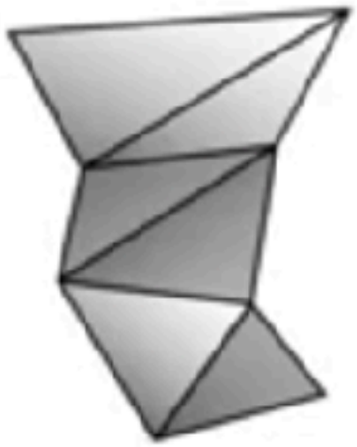


GL_LINE_STRIP

GL_LINE_LOOP



GL_TRIANGLES



GL_TRIANGLE_STRIP

GL_TRIANGLE_FAN

Vertex Buffer Object

Vertex Buffer Object

```
float positions[] = {
        -0.8f,  0.4f, 0.0f,
         0.8f,  0.4f, 0.0f,
         0.8f, -0.4f, 0.0f,
        -0.8f,  0.4f, 0.0f,
         0.8f, -0.4f, 0.0f,
        -0.8f, -0.4f, 0.0f
};
```

Vertex Buffer Object

```
GLuint buffer;
glGenBuffers( 1, &buffer );
glBindBuffer( GL_ARRAY_BUFFER, buffer );
glBufferData( GL_ARRAY_BUFFER,
              sizeof(positions),
              positions,
              GL_STATIC_DRAW );
```

Vertex Buffer Object

VBO
(vertex buffer object)

Vertex Shader

GLSL

```
#version 330 core

layout(location=0) in vec3 pos;
uniform mat4 mvp;
void main()
{
    gl_Position = mvp * vec4(pos,1);
}
```

Vertex Buffer Object

GLSL

```
#version 330 core

layout(location=0) in vec3 pos;
uniform mat4 mvp;
void main()
{
    gl_Position = mvp * vec4(pos,1);
}
```

```
GLuint pos = glGetAttribLocation(
    program, "pos" );
glEnableVertexAttribArray( pos );
glVertexAttribPointer(
    pos, 3, GL_FLOAT,
    GL_FALSE, 0, (GLvoid*)0 );
```

Rendering

```
glClear( GL_COLOR_BUFFER_BIT
       | GL_DEPTH_BUFFER_BIT );

glUseProgram( program );
glDrawArrays( GL_POINTS,
              0,
              num_vertices );

glutSwapBuffers();
```

Vertex Attributes

VBO -> GPU memory buffer
Efficient
Array of vertices -> GPU

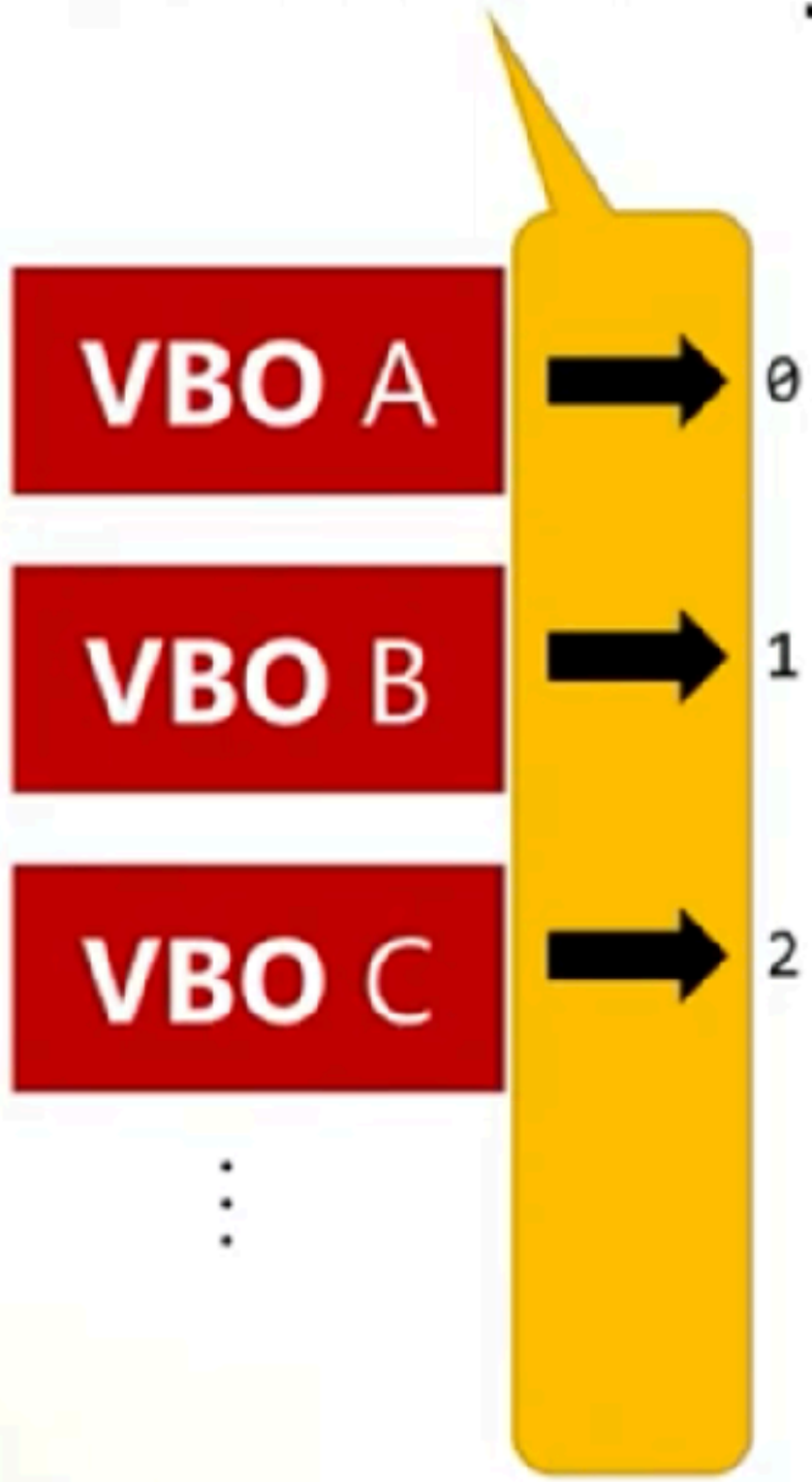
VBO A → `layout(location=0) in vec3 pos;`

VBO B → `layout(location=1) in vec3 clr;`

VBO C → `layout(location=2) in float t;`

⋮

Vertex Array Object



Vertex Array Object

```
Gluint vao;
glGenVertexArrays( 1, &vao );
glBindVertexArray( vao );
...
GLuint pos = glGetAttribLocation(
    program, "pos" );
glEnableVertexAttribArray( pos );
glVertexAttribPointer(
    pos, 3, GL_FLOAT,
    GL_FALSE, 0, (GLvoid*)0 );
```

Overview

- Initialization
  - Create VAO
  - Create VBO
  - Compile shaders
- Rendering
  - Assign VBO to vertex attribute
  - Call glDrawArrays

Overview

- Initialization
  - Create VAOs
  - Create VBOs
  - Compile shaders
  - For each VAO
    - Assign VBOs to vertex attributes
- Rendering
  - For each VAO
    - Call glDrawArrays

Vertex Shader

GLSL

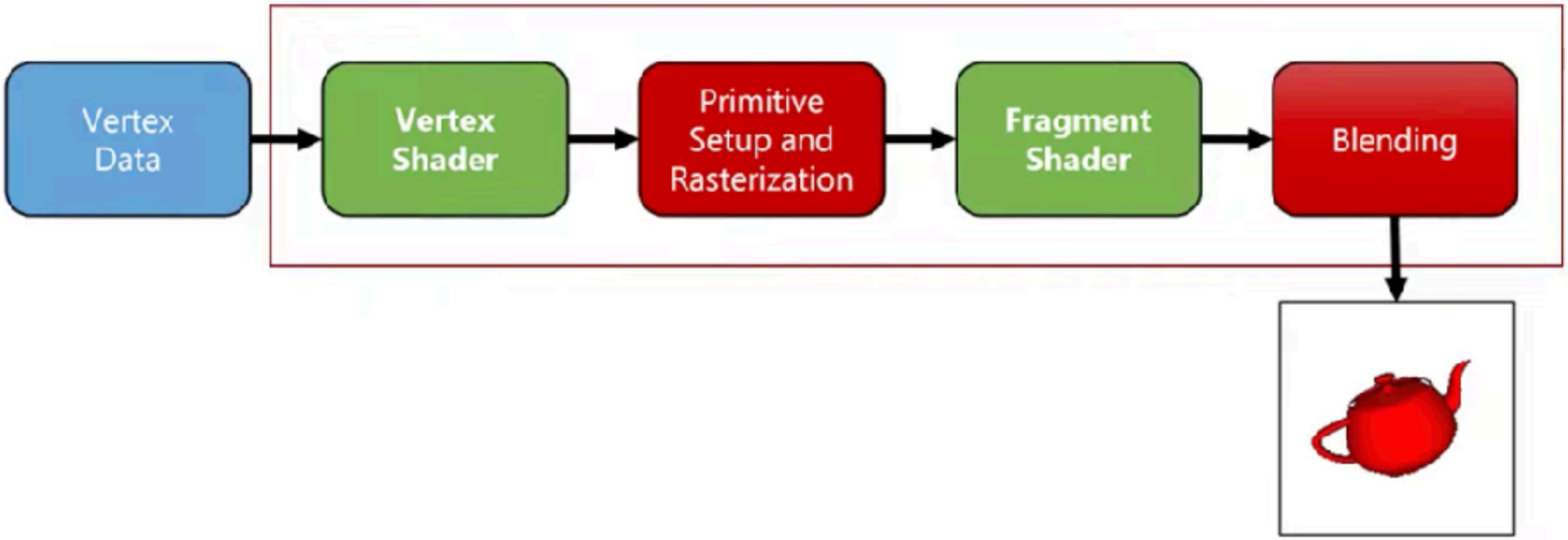
```
#version 330 core

layout(location=0) in vec3 pos;

uniform mat4 mvp;

void main()
{
    gl_Position = mvp * vec4( pos, 1 );
}
```

Fragment Shader

GLSL

```
#version 330 core

layout(location = 0) out vec4 color;

void main()
{
    color = vec4( 1, 0, 0, 1 );
}
```

GPU Pipeline



Vertex Shader

GLSL

```
#version 330 core

layout(location=0) in vec3 pos;
layout(location=1) in vec4 clr;

out vec3 vColor;

uniform mat4 mvp;

void main()
{
    gl_Position = mvp * vec4( pos, 1 );
    vColor = clr.xyz;
}
```

Fragment Shader

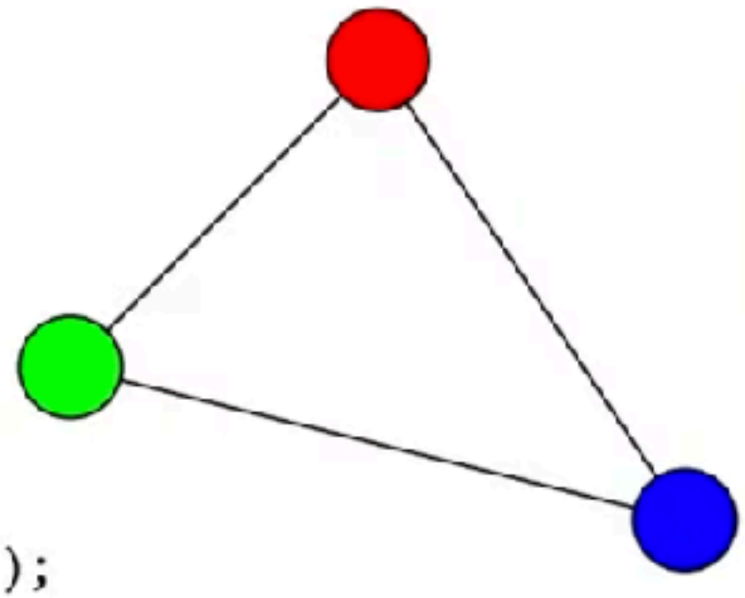
GLSL

```
#version 330 core

layout(location = 0) out vec4 color;

in vec3 vColor;

uniform float alpha;

void main()
{
    color = vec4( vColor, alpha );
}
```

Vertex Shader

GLSL

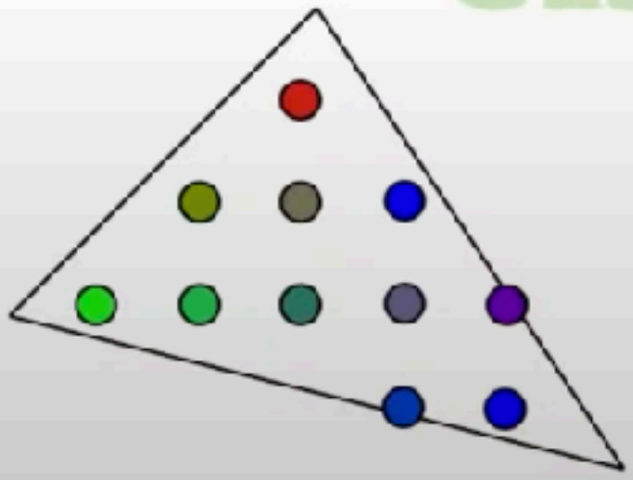
```
#version 330 core

layout(location=0) in vec3 pos;
layout(location=1) in vec4 clr;

out vec3 vColor;

uniform mat4 mvp;

void main()
{
    gl_Position = mvp * vec4( pos, 1 );
    vColor = clr.xyz;
}
```



Fragment Shader

GLSL

```
#version 330 core

layout(location = 0) out vec4 color;

in vec3 vColor;

uniform float intensity;

void main()
{
    color = vec4( vColor * intensity, 1 );
}
```



Vertex Shader

GLSL

```
#version 330 core

layout(location=0) in vec3 pos;
layout(location=1) in vec4 clr;

out vec3 vColor;

uniform mat4 mvp;

void main()
{
    gl_Position = mvp * vec4( pos, 1 );
    vColor = clr.rbg;
}
```

Fragment Shader

GLSL

```
#version 330 core

layout(location = 0) out vec4 color;

in vec3 vColor;

uniform float intensity;

void main()
{
    color = vec4( vColor * intensity, 1 );
}
```

Vertex Shader

GLSL

```
#version 330 core

layout(location=0) in vec3 pos;
layout(location=1) in vec3 clr;

out vec3 vColor;

uniform mat4 mvp;

void main()
{
    gl_Position = mvp * vec4( pos, 1 );
    vColor = clr;
}
```

Fragment Shader

GLSL

```
#version 330 core

layout(location = 0) out vec4 color;

in vec3 vColor;

uniform float intensity;

void main()
{
    color = vec4( vColor * intensity, 1 );
}
```

Vertex Shader

GLSL

```
#version 330 core

layout(location=0) in vec3 pos;
layout(location=1) in vec3 clr;

out vec3 vColor;

uniform mat4 mvp;

void main()
{
    gl_Position = mvp * vec4( pos, 1 );
    vColor = clr;
}
```

Fragment Shader

GLSL

```
#version 330 core

layout(location = 0) out vec4 color;

in vec3 vColor;

void main()
{
    color = vec4( vColor, 1 );
}
```

Vertex Shader

GLSL

```
#version 330 core

layout(location=0) in vec3 pos;

out vec3 vColor;

uniform mat4 mvp;

void main()
{
    gl_Position = mvp * vec4( pos, 1 );
    vColor = vec3( 1, 0, 0 );
}
```

Fragment Shader

GLSL

```
#version 330 core

layout(location = 0) out vec4 color;

in vec3 vColor;

void main()
{
    color = vec4( vColor, 1 );
}
```

Vertex Shader

GLSL

```
#version 330 core

layout(location=0) in vec3 pos;

uniform mat4 mvp;

void main()
{
    gl_Position = mvp * vec4( pos, 1 );
}
```

Fragment Shader

GLSL

```
#version 330 core

layout(location = 0) out vec4 color;

void main()
{
    color = vec4( 1, 0, 0, 1 );
}
```

Overview

- Initialization
  - Create OpenGL context/window
  - Create VAOs
  - Create VBOs
  - Compile shaders
  - For each VAO
    - Assign VBOs to vertex attributes
- Rendering
  - For each VAO
    - Call `glDrawArrays(...)`

OpenGL vs GLSL Versions

OpenGL	GLSL
2.0	1.10
2.1	1.20
3.0	1.30
3.1	1.40
3.2	1.50
3.3	**3.30**
⋮	⋮

OpenGL Extensions

- OpenGL version 1.1

```
#include <GH/gl.h>
```

- Newer OpenGL versions

```
#include <glew.h>
glewInit();
```

GLEW (OpenGL Extension Wrangler)

OpenGL Extensions

- OpenGL version 1.1

```
#include <GH/gl.h>
```

- Newer OpenGL versions

```
#include <glext.h>
#include <wingdi.h>
PFNGLGENVERTEXARRAYSPROC glGenVertexArrays =
   (PFNGLGENVERTEXARRAYSPROC)
   wglGetProcAddress("glGenVertexArrays");
```

OpenGL Versions & Extensions

```
glGenProgram(…)

glGenProgramARB(…)

glUniform1i64vNV(…)

glDrawBuffersATI(…)
```

Uniform Variables

function name

```
glUniform3f( location, x, y, z )
```

dimension

float

x, y, z are values

```
glUniform3fv( location, val )
```

val is an array of values

Uniform Variables

function name

```
glUniform3f( location, x, y, z )
```

dimension

float

x, y, z are values

Uniform Variables

```
GLint location = glGetUniformLocation(
                                    program,
                                    "alpha" );

glUniform1f( location, 0.5f );
```

Compile Shaders

```
GLuint progam = glCreateProgram();

// Compile vertex shader
GLuint vs = glCreateShader();
...

// Compile fragment shader
GLuint fs = glCreateShader();
...

glLinkProgram( program );
```

cyTriMesh.h

```
cy::TriMesh mesh;
bool success =
    mesh.LoadFromFileObj("model.obj");
```

```
glBufferData(
    GL_ARRAY_BUFFER,
    sizeof(cy::Vec3f)*mesh.NV(),
    &mesh.V(0),
    GL_STATIC_DRAW );
```

cyMatrix.h

```
cy::Matrix3f rotMatrix =
    cy::Matrix3f::RotationY(yRot);
cy::Matrix4f projMatrix =
    cy::Matrix4f::Perspective(
        DEG2RAD(40),
        float(width)/float(height),
        0.1f,
        1000.0f );

cy::Matrix4f m = projMatrix * rotMatrix
```

cyGL.h

```
cy::GLSLProgram prog;
prog.BuildFiles( "shader.vert",
                 "shader.frag" );

prog["mvp"] = myMatrix;

prog.Bind();
glDrawArrays(…);
```

cyGL.h

```
cy::GLSLProgram prog;
prog.BuildFiles( "shader.vert",
                 "shader.frag" );

prog.Bind();
glDrawArrays(…);
```

cyGL.h

```
glutCreateWindow("my window");

CY_GL_REGISTER_DEBUG_CALLBACK;
```

Make sure to generate a debug context using `glutInitContextFlags(GLUT_DEBUG);`

cyGL.h

```
glutCreateWindow("my window");

cy::GLDebugCallback callback(true);
callback.IgnoreNotifications();
```

Make sure to generate a debug context using `glutInitContextFlags(GLUT_DEBUG);`

cyGL.h

```
glutCreateWindow("my window");

cy::GLDebugCallback callback;
callback.Register();
callback.IgnoreNotifications();
```

Make sure to generate a debug context using
`glutInitContextFlags(GLUT_DEBUG);`

cyGL.h

```
glutCreateWindow("my window");

CY_GL_REGISTER_DEBUG_CALLBACK;
```

Make sure to generate a debug context using
`glutInitContextFlags(GLUT_DEBUG);`