CS475/CS675
Computer Graphics

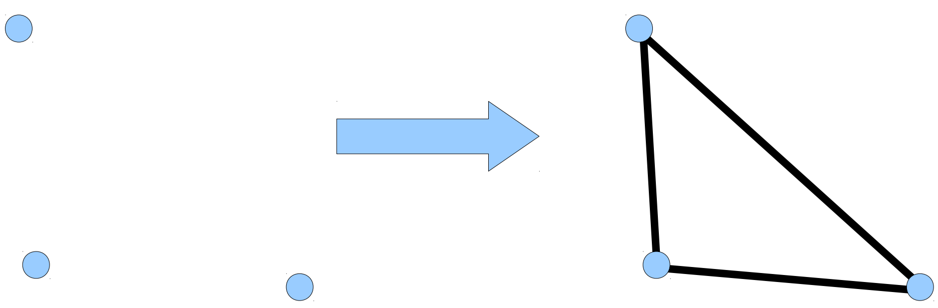
OpenGL Drawing

What is OpenGL?

- **O**pen **G**raphics **L**ibrary
- *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)

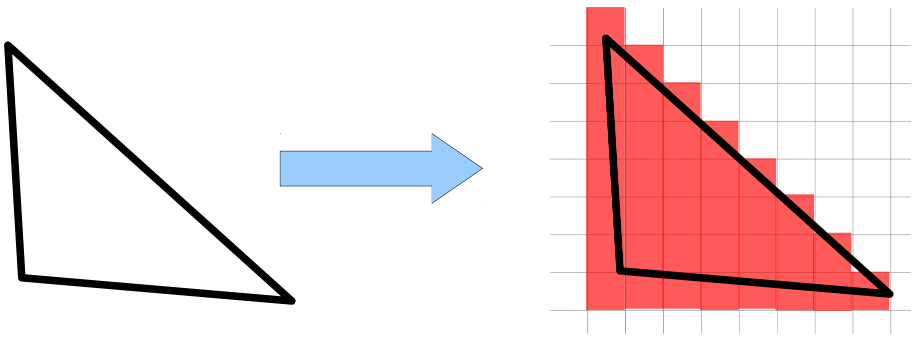
OpenGL Vertices

- 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

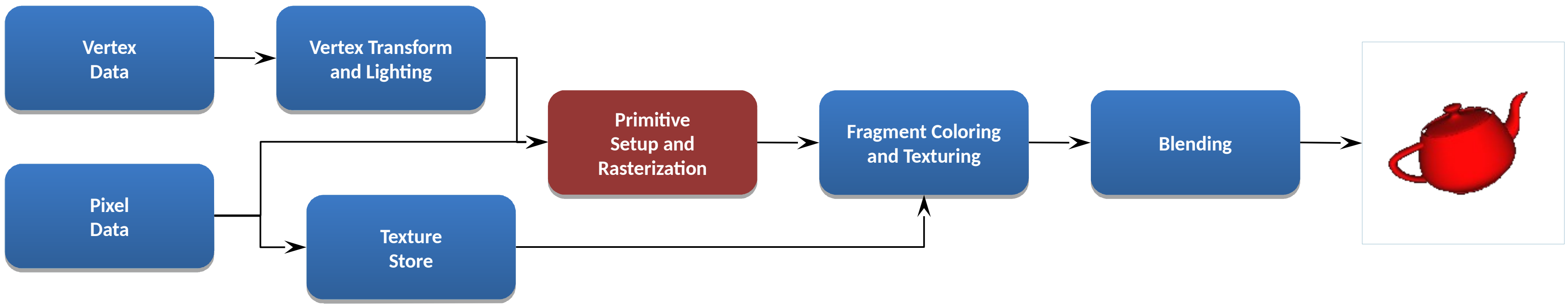
- A ***fragment*** is a pixel with a lot of other information:
  - Location
  - Color
  - Normal
  - Depth
  - Opacity
  - ….



OpenGL rasterizes primitive shapes and outputs fragments.

OpenGL 1 (1991)

- Entirely fixed-function

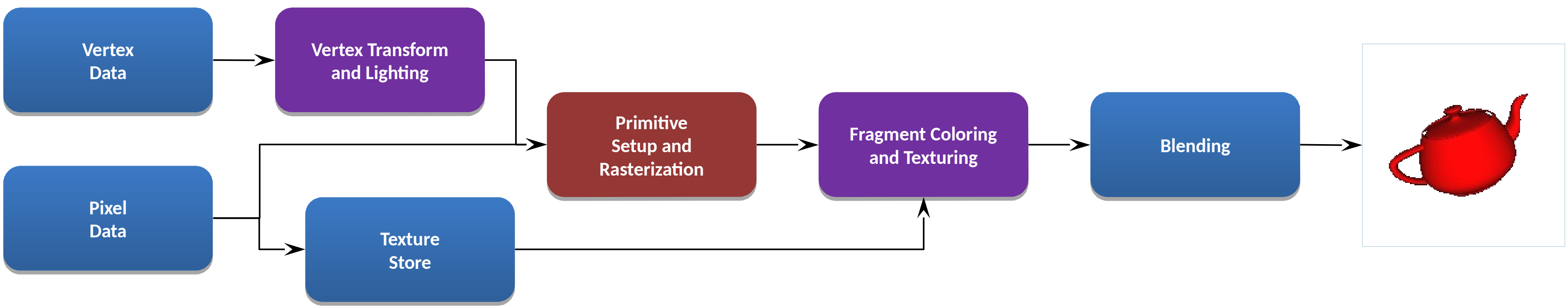


An Introduction to OpenGL Programming, Edward Angel and Dave Schreiner, SIGGRAPH 2013 Course.

- The pipeline evolved but remained based on fixed-function operation through OpenGL versions 1.1 through 2.0.

OpenGL 2 (1994)

- Introduced programmable shaders

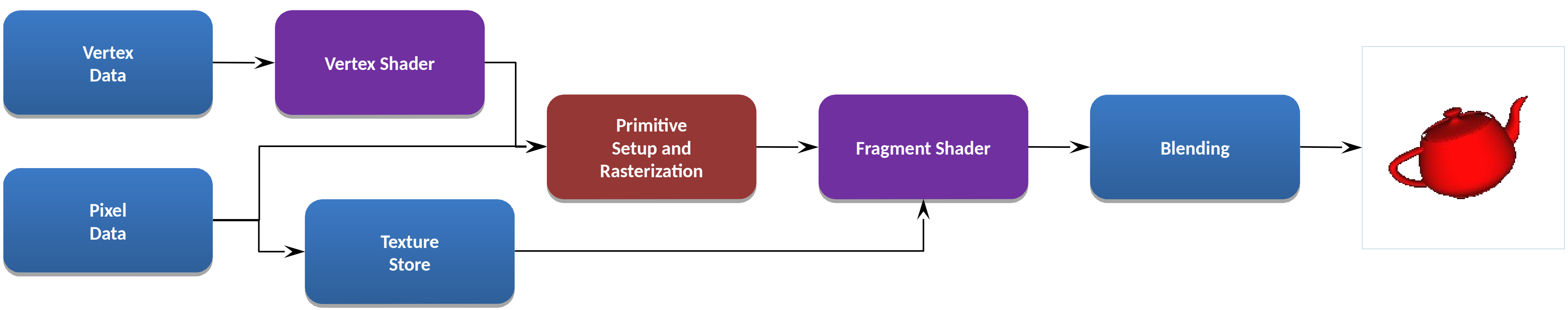


An Introduction to OpenGL Programming, Edward Angel and Dave Schreiner, SIGGRAPH 2013 Course.

- Vertex shading, Fragment shading

OpenGL 3, 3.1, 3.2 (~2008)

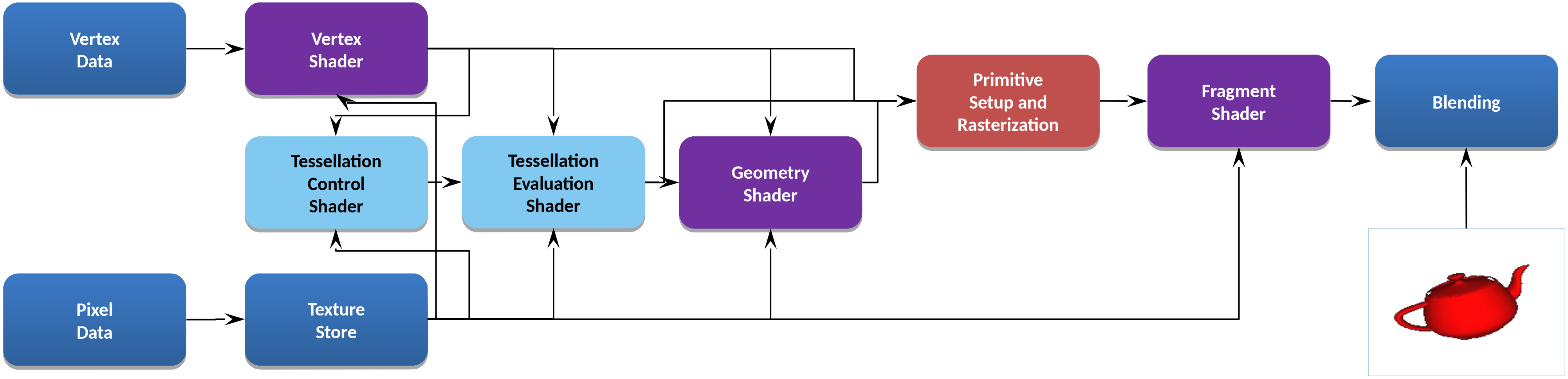
- Introduced the deprecation model
  - Removed the fixed function pipeline.
- Large chunks of data fed to the pipeline instead of small chunks.
- OpenGL catching up with GPU architecture



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

OpenGL 4.0 – 4.5 (current day)

- Geometry & Tesselation Shader – Geometry generated on the GPU

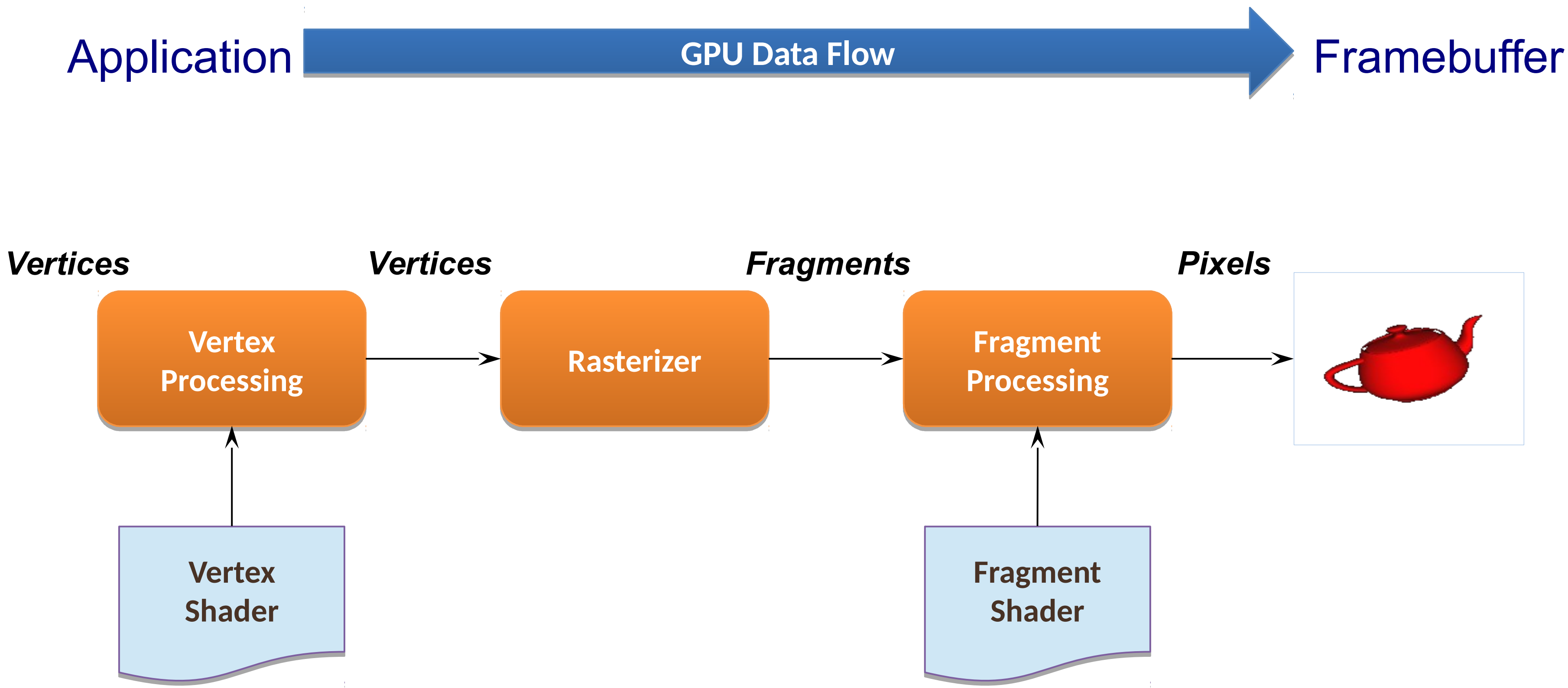


An Introduction to OpenGL Programming, Edward Angel and Dave Schreiner, SIGGRAPH 2013 Course.

OpenGL Variants

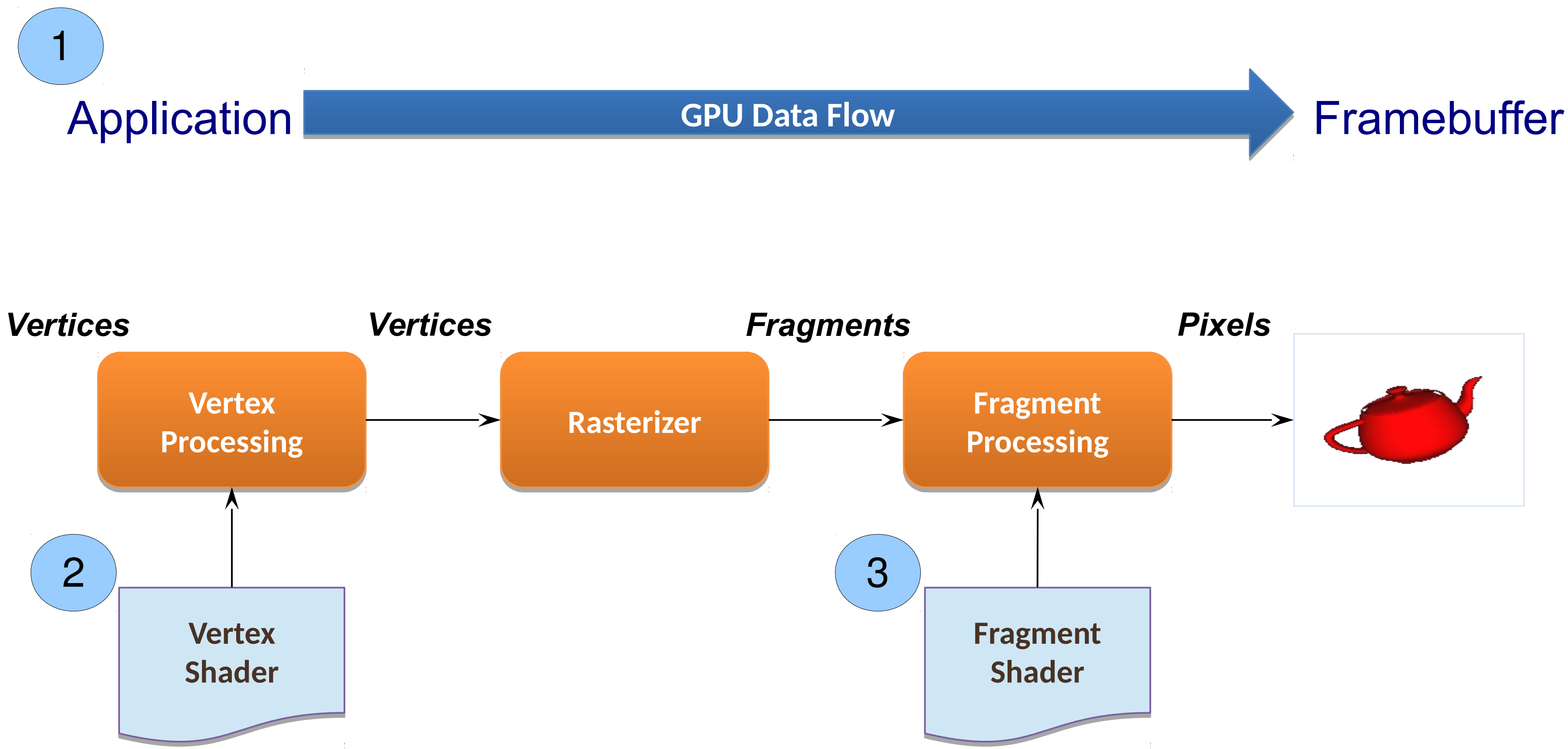
- OpenGL ES
  - For hand-held devices and embedded environments
- WebGL
  - Javascript implementation of ES
  - Runs on most recent browsers

Simplified Pipeline



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

Simplified Pipeline

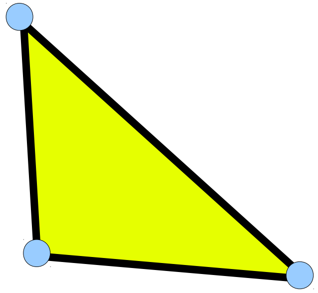


An Introduction to OpenGL Programming, Edward Angel and Dave Schreiner, SIGGRAPH 2013 Course.

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

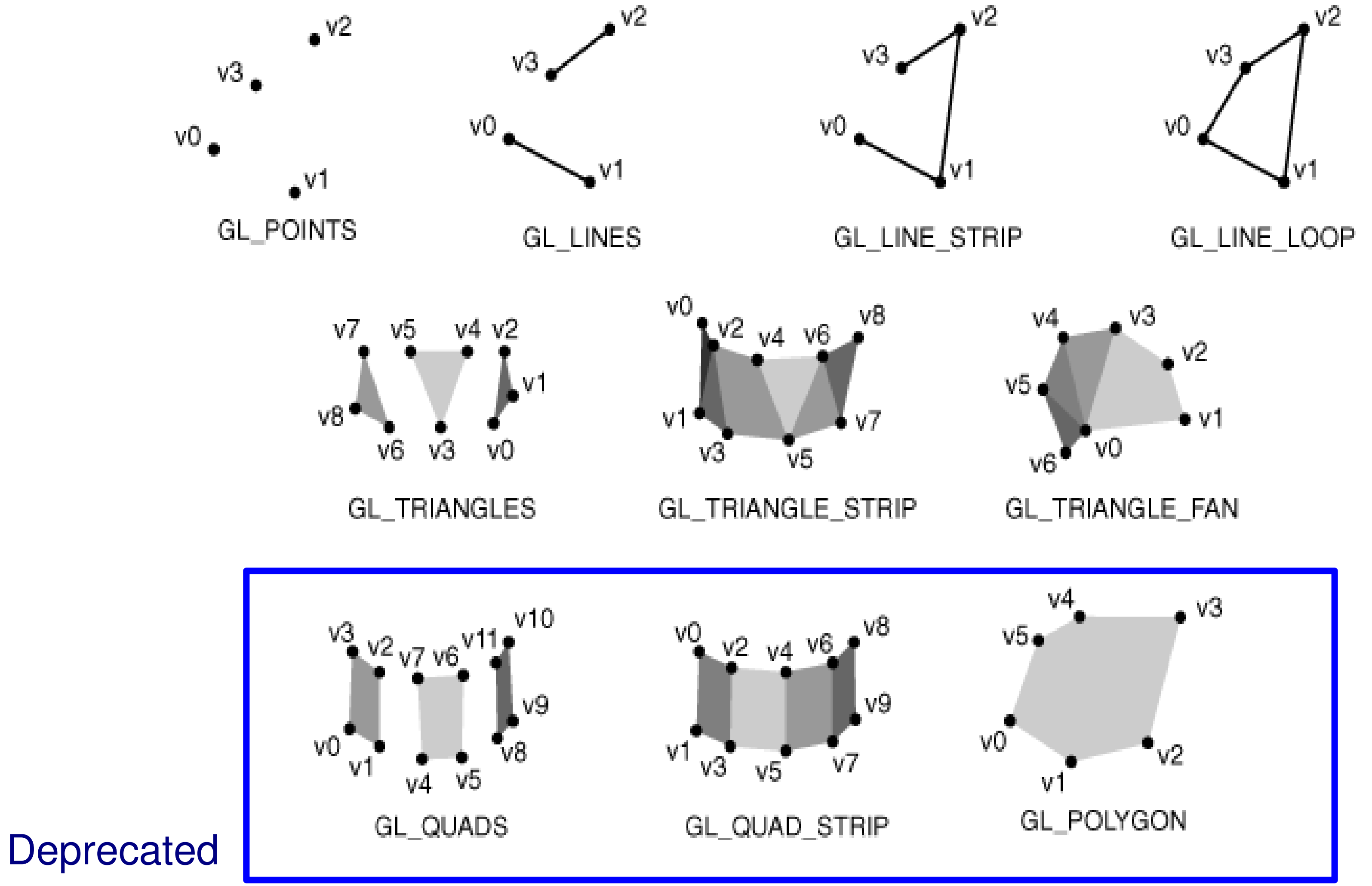
Geometric Objects in OpenGL



- 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)
- VBOs must be stored in vertex array objects (VAOs)

OpenGL Primitives



Example: ColorCube

- A cube with different colors at each vertex

```
//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

```
// 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 );
}
```

Example: ColorCube

- A cube with different colors at each vertex
- VertexArrayObjects (VAOs)
- VertexBufferObjects(VBOs)
- Generate and Bind

Example: ColorCube

- Vertex Array Object

```
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, ...)

  glBufferSubData(GL_ARRAY_BUFFER, …)

Example: ColorCube

- Create a Shader Program

Create Program	glCreateProgram()
Create Shader	glCreateShader()
Load Shader Source	glShaderSource()
Compile Shader	glCompileShader()
Attach Shader to Program	glAttachShader()
Link Program	glLinkProgram()
Use Program	glUseProgram()

These steps need to be repeated for each type of shader in the shader program

Example: ColorCube

- GLSL Vertex Shader

```
#version 430

in vec4 vPosition;
in vec4 vColor;

out vec4 color;

void main ()
{
  gl_Position = vPosition;
  color = vColor;
}
```

Example: ColorCube

- GLSL Fragment Shader

```
#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

```
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

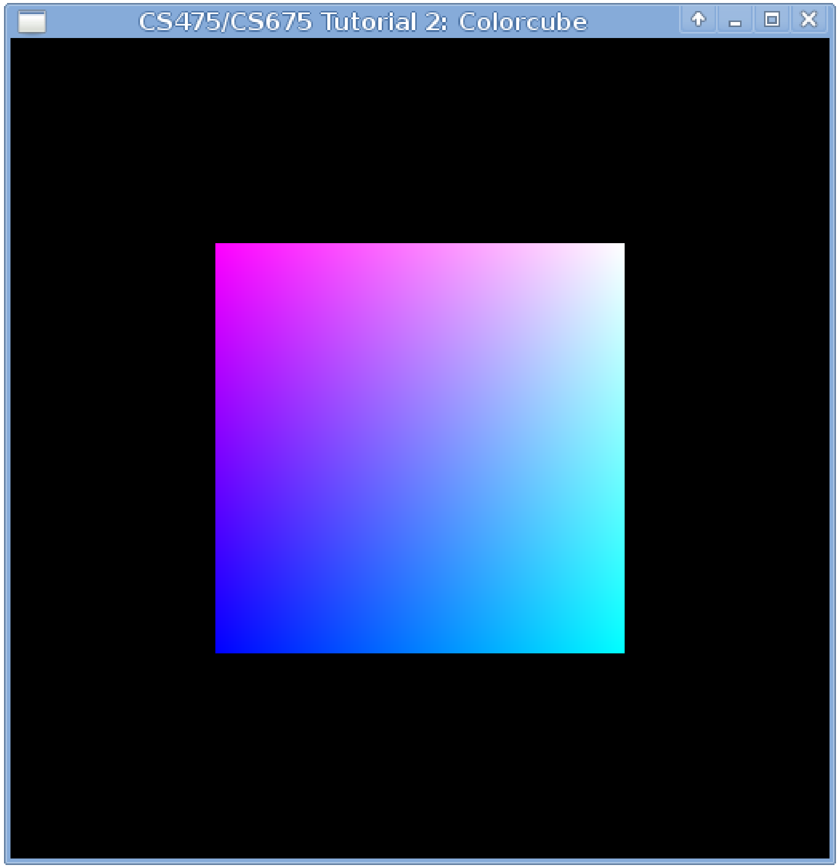
```
/Initialize GLEW
//Turn this on to get Shader based OpenGL
glewExperimental = GL_TRUE;
GLenum err = glewInit();
if (GLEW_OK != err)
  {
    //Problem: glewInit failed, something is seriously wrong.
    std::cerr<<"GLEW Init Failed : %s"<<std::endl;
  }
```

Example: ColorCube

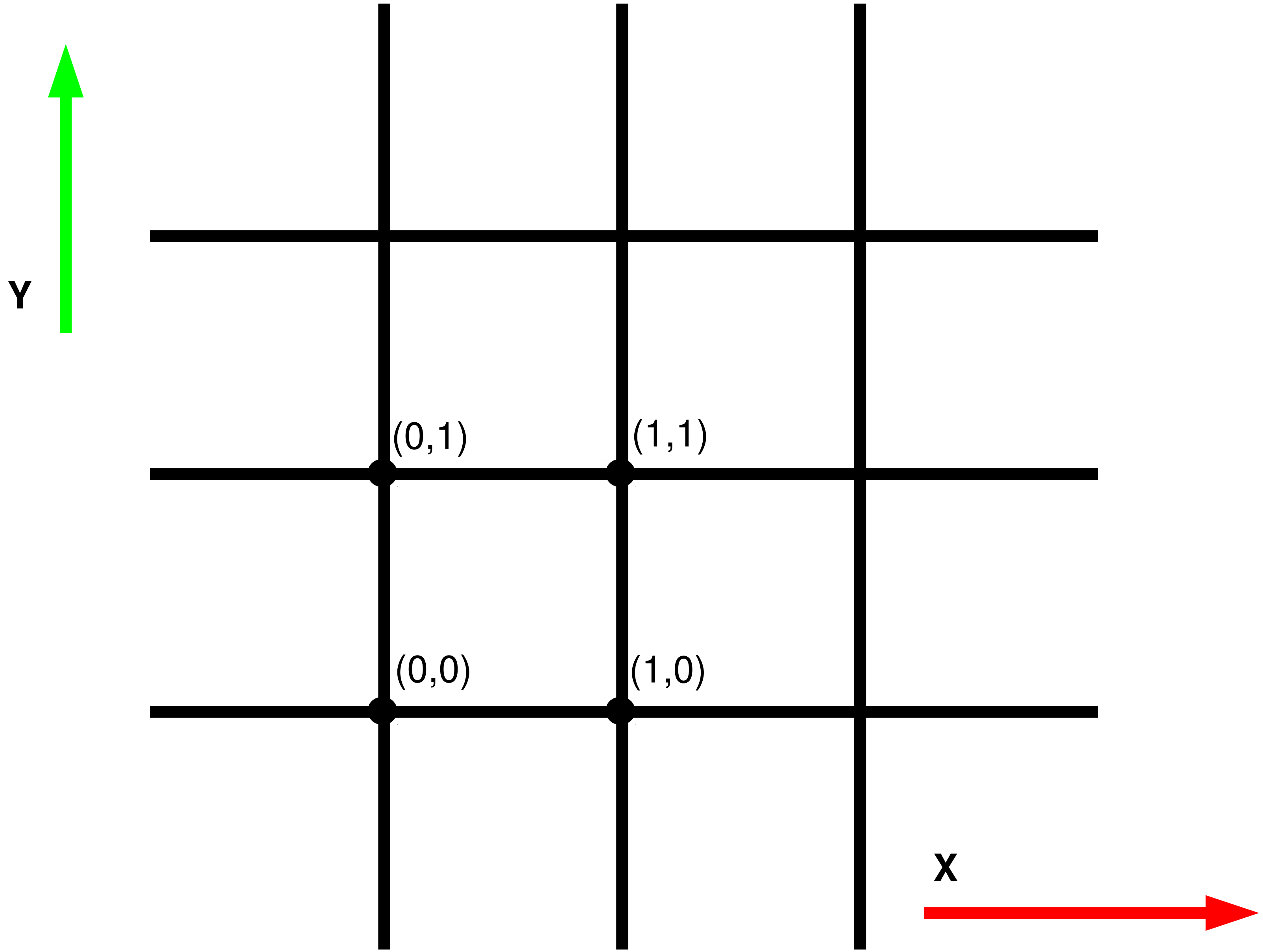
- GLFW

```
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) == 0)
  {
    renderGL();
    glfwSwapBuffers(window);
    glfwPollEvents();
  }

glfwTerminate();
return 0;
}
```

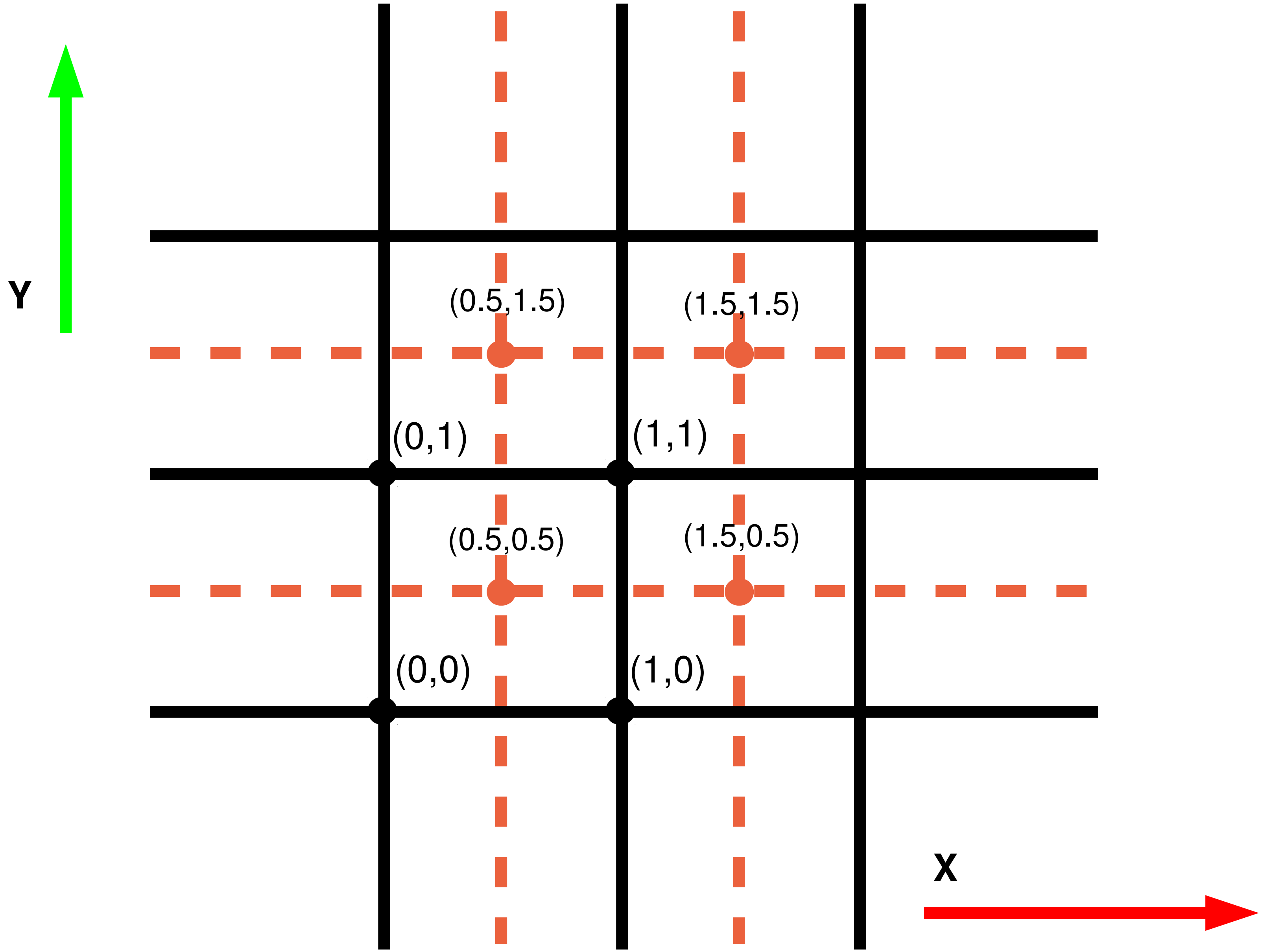
Example: ColorCube



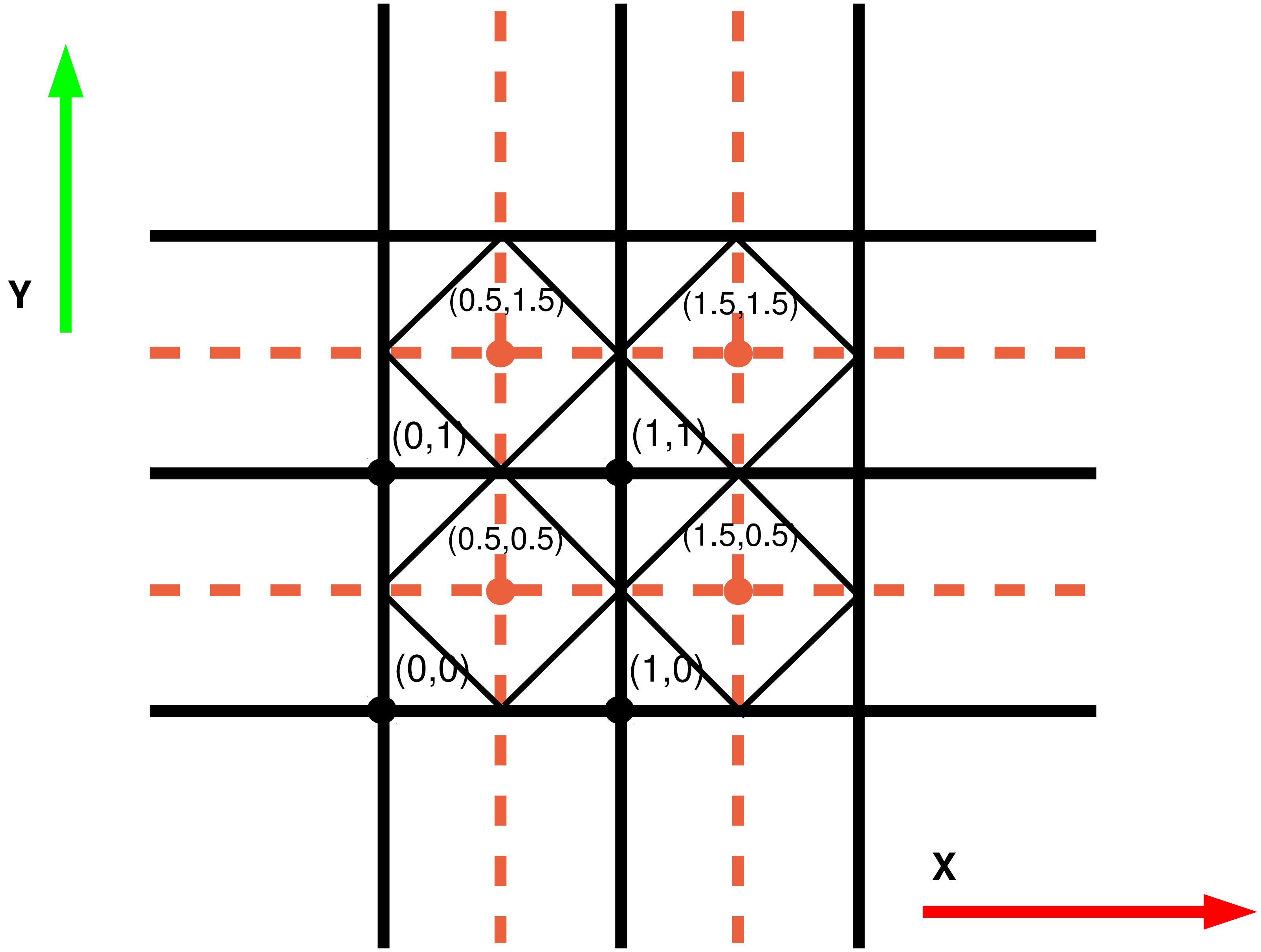
OpenGL Rasterization



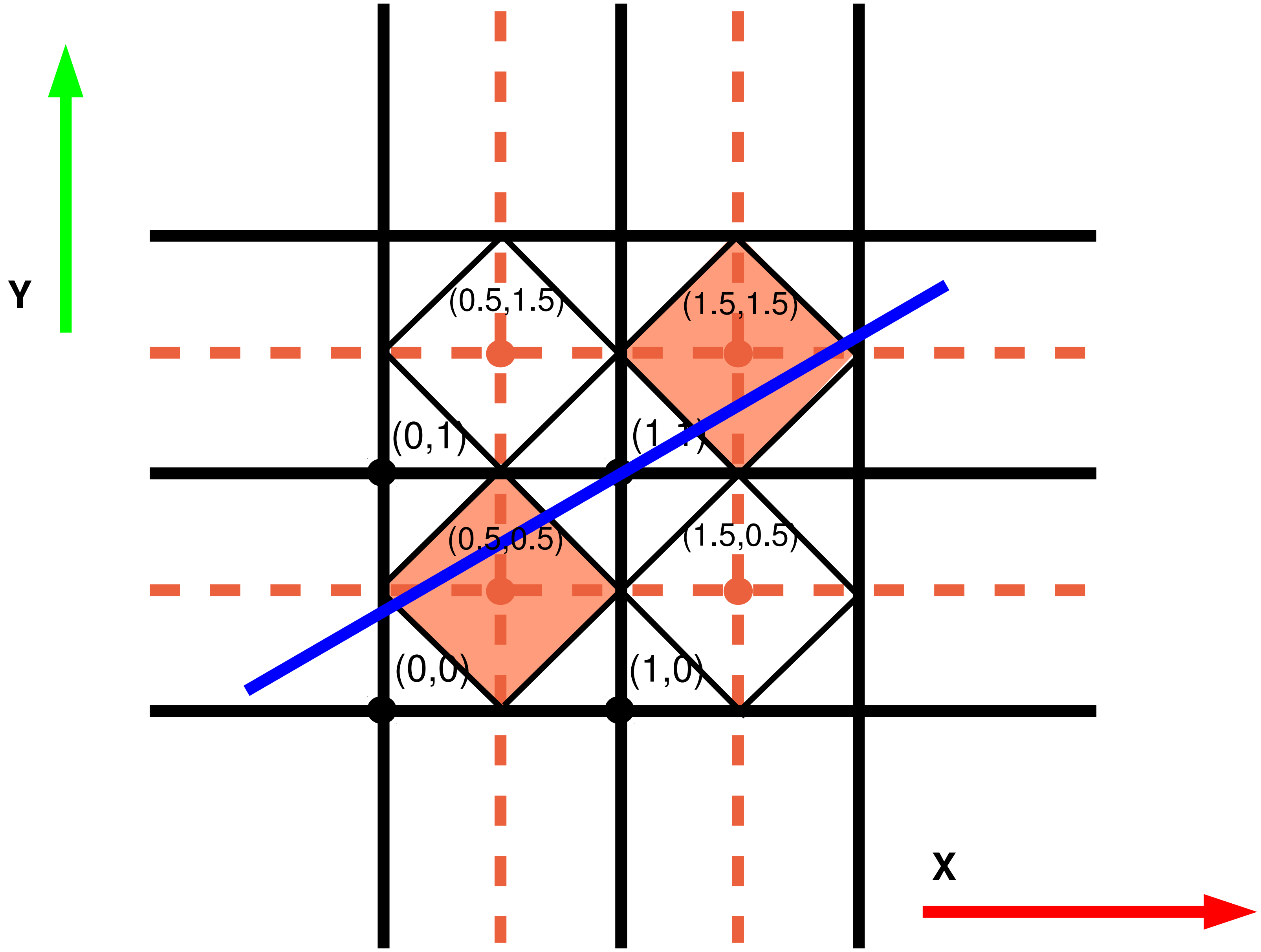
OpenGL Rasterization



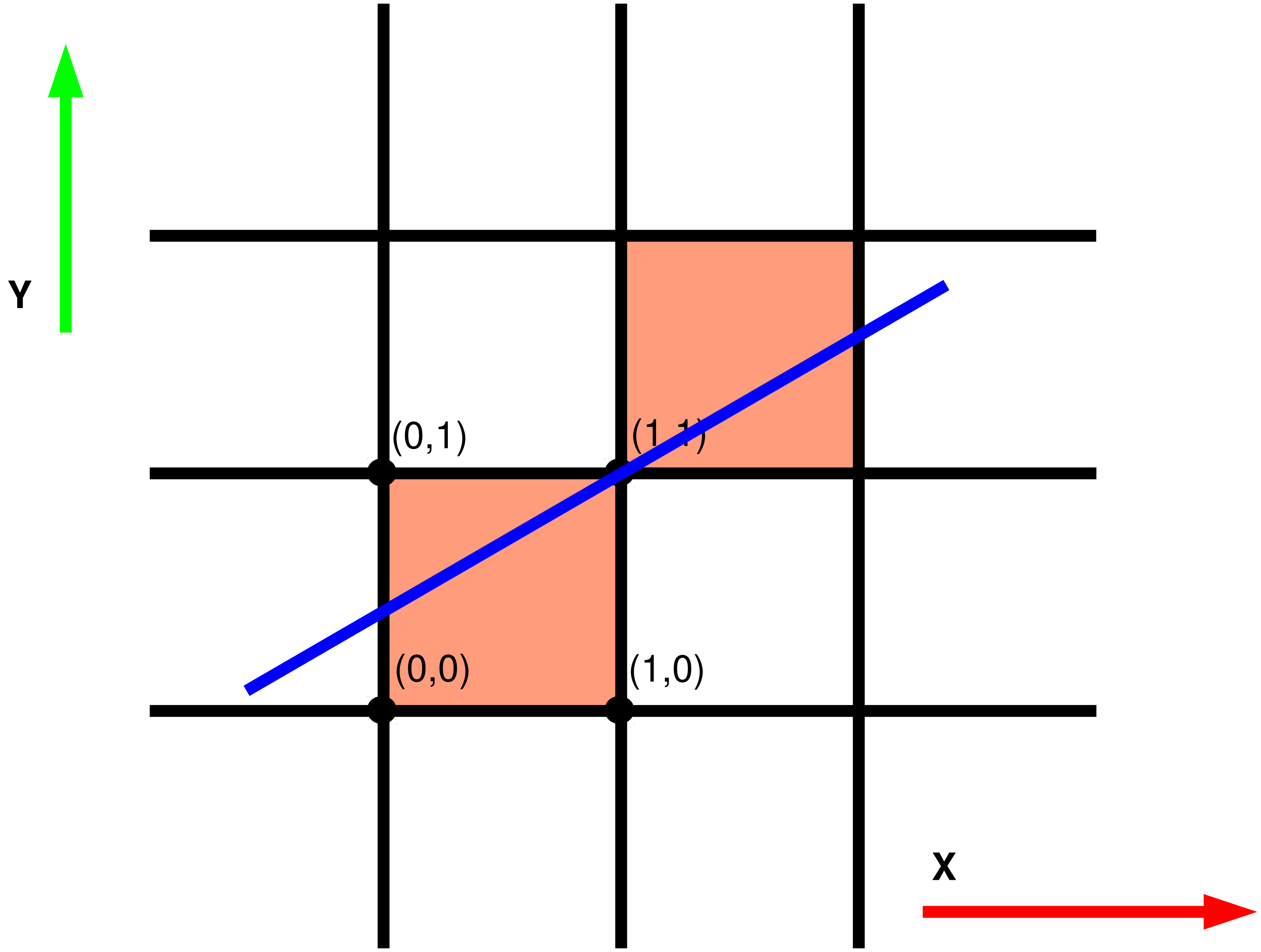
OpenGL Line Rasterization



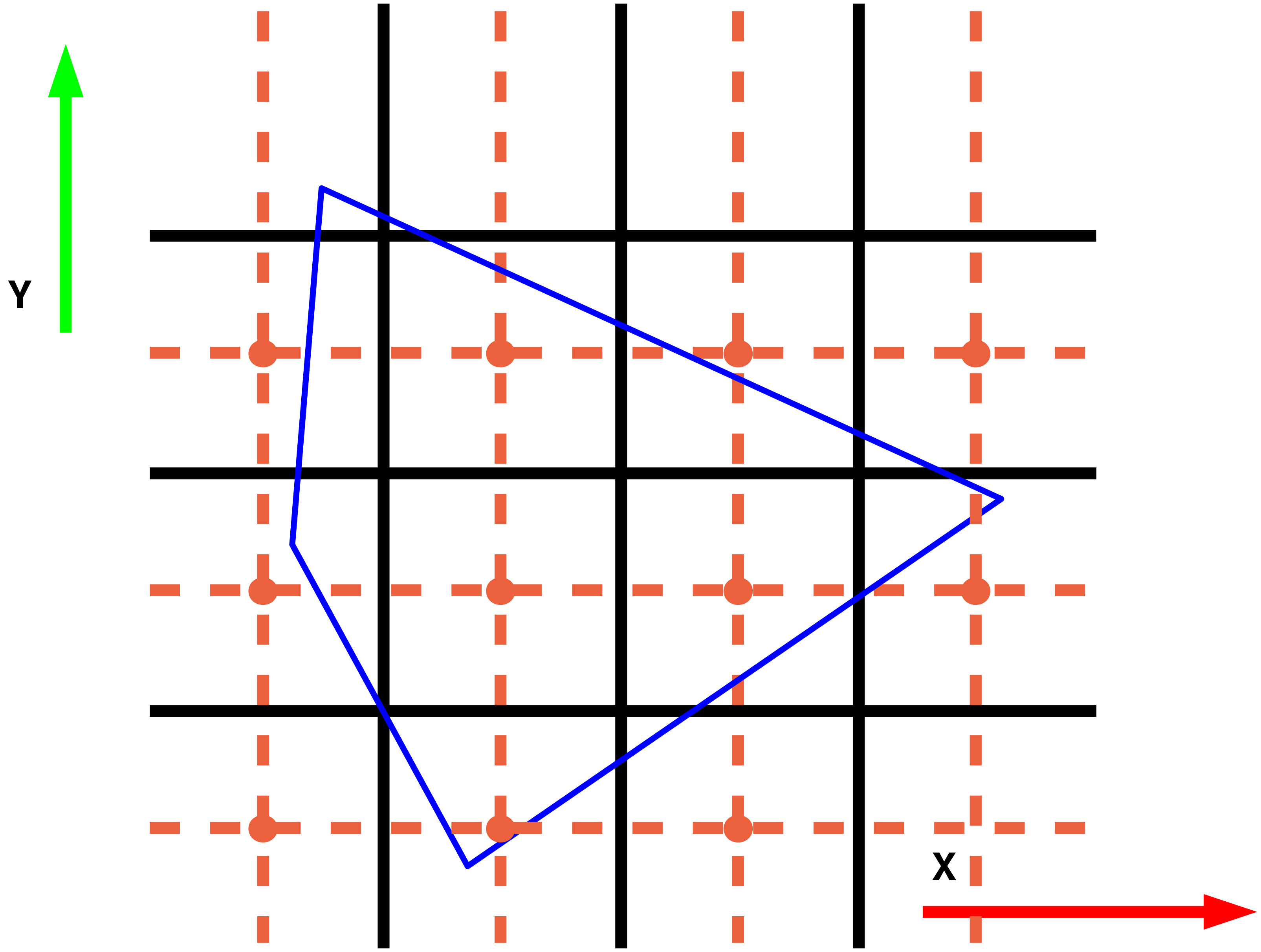
OpenGL Line Rasterization



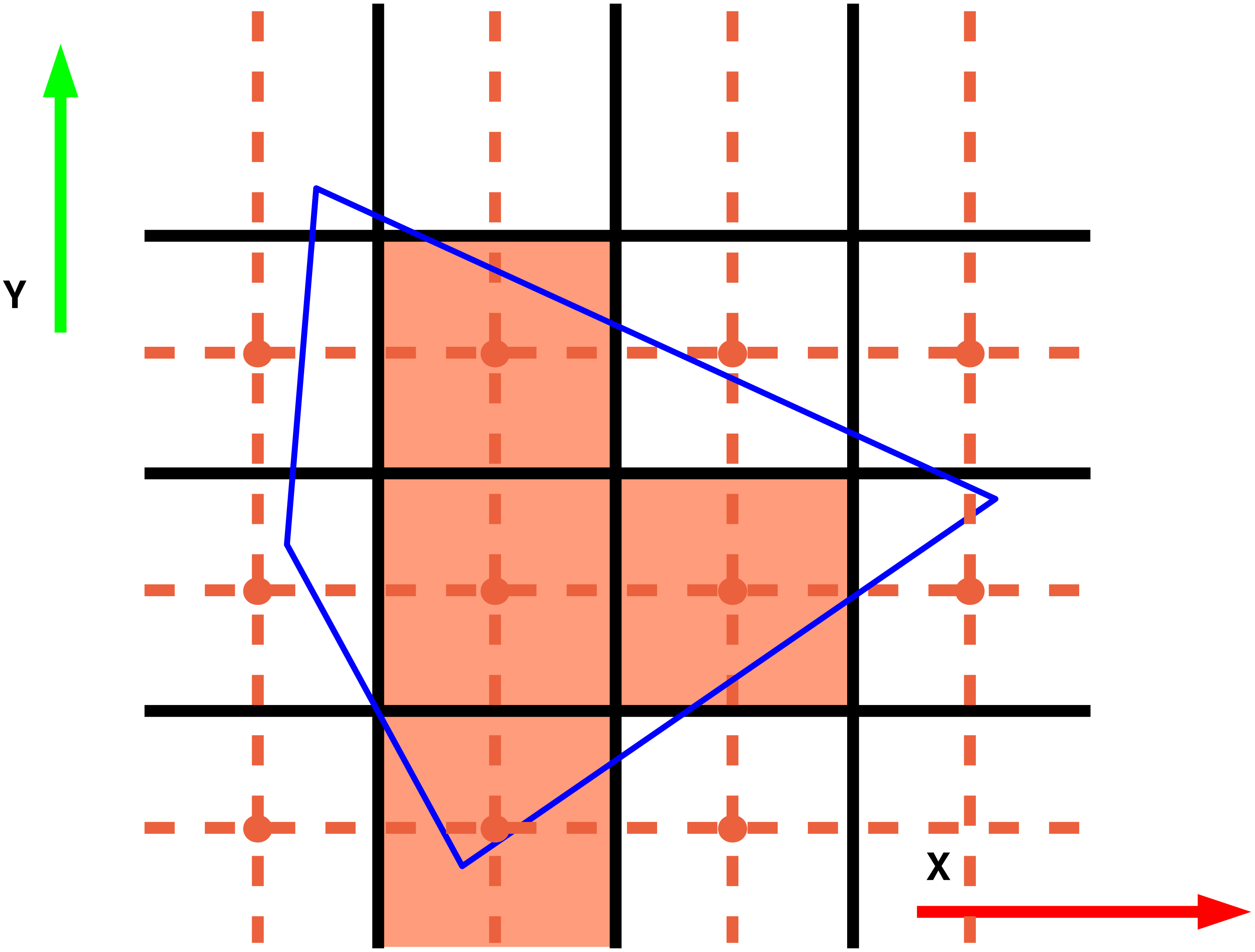
OpenGL Line Rasterization



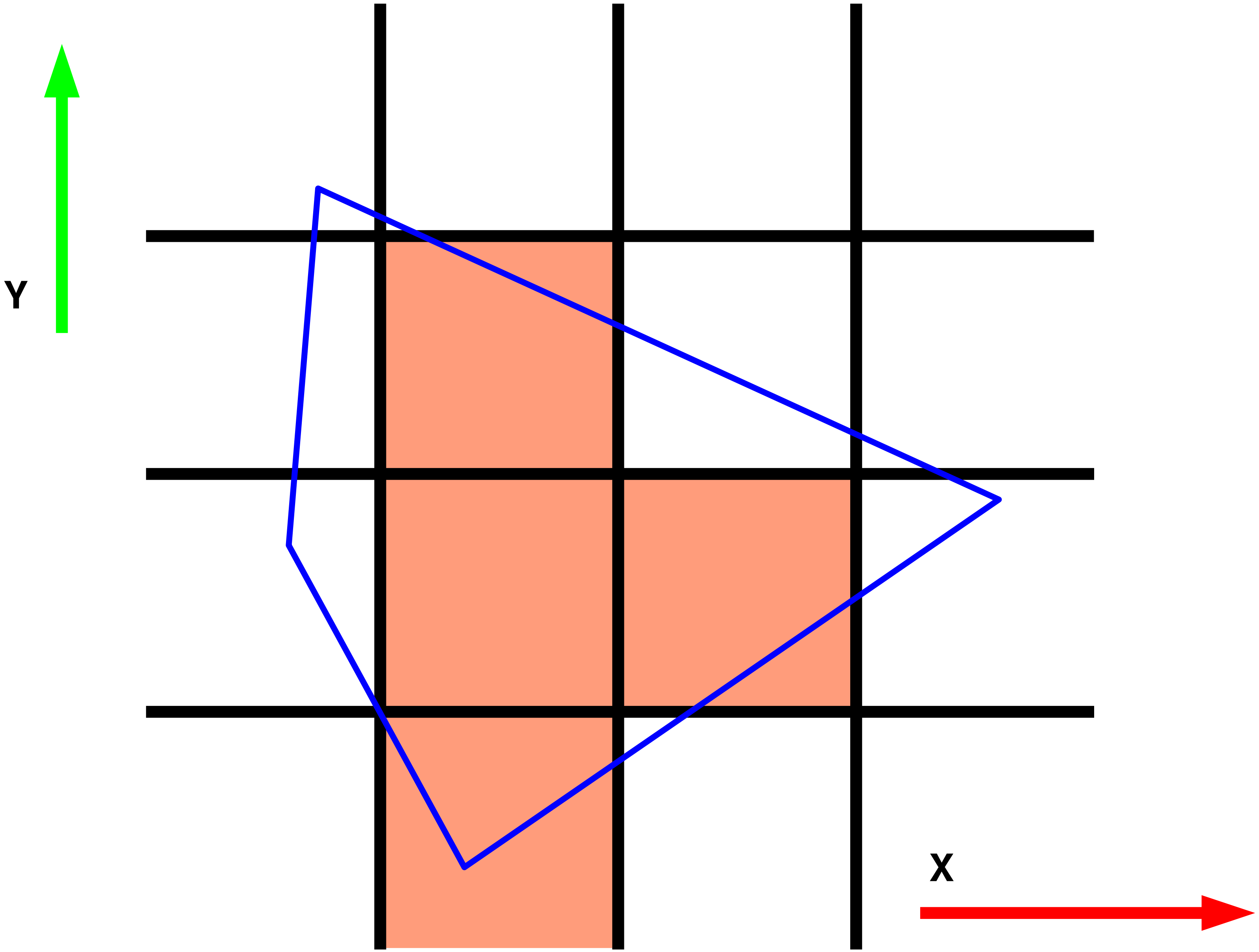
OpenGL Polygon Rasterization



OpenGL Polygon Rasterization



OpenGL Polygon Rasterization



OpenGL Polygon Rasterization