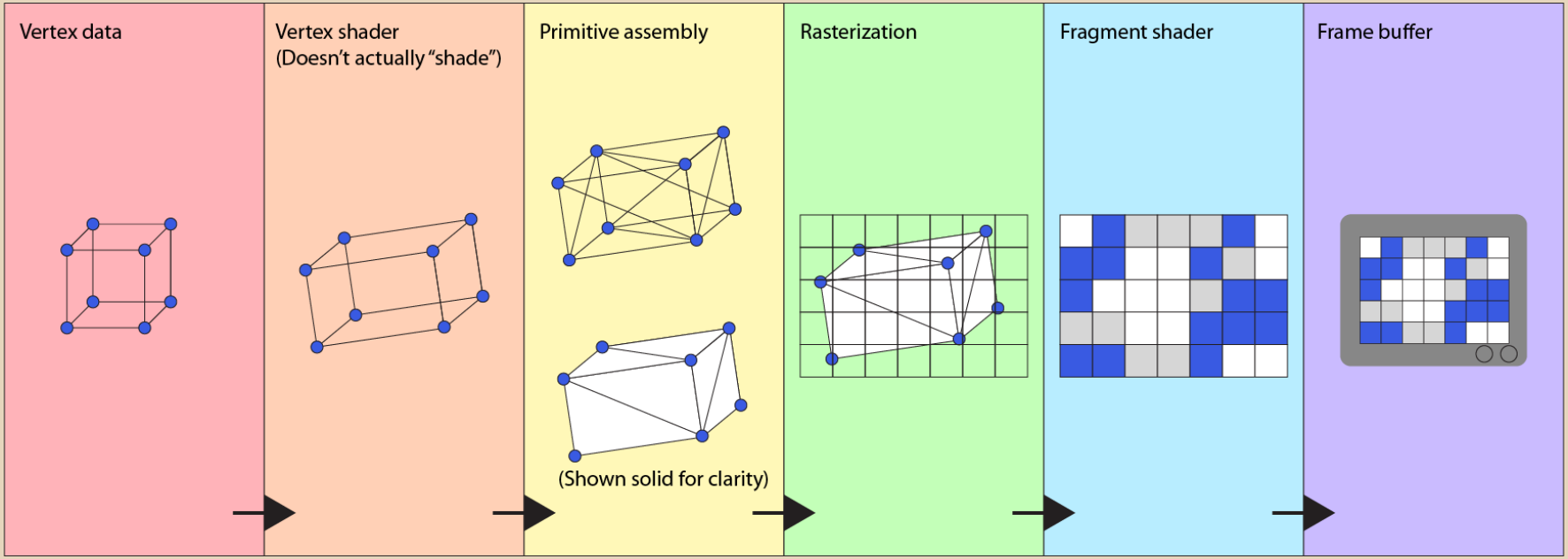
OpenGL

Adam Mally
CIS 560 Spring 2015
University of Pennsylvania

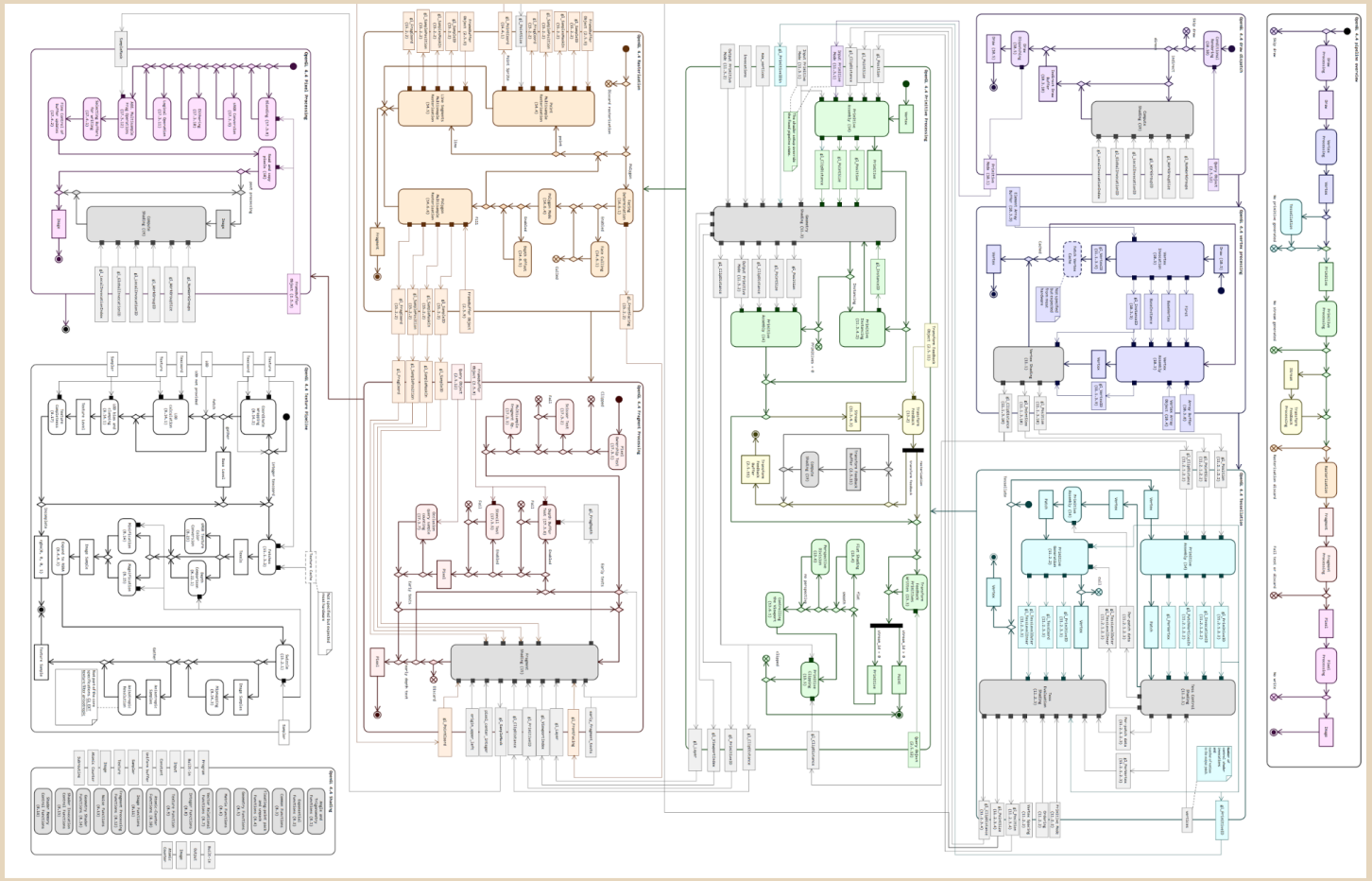
What is OpenGL?

- A multiplatform API for rendering 2D and 3D computer graphics
- In your computer, it's a collection of drivers, libraries, and header files
- Developed by Silicon Graphics Inc. in 1991
- Managed the Khronos Group
  - Regularly release updated specifications
- Why "Open" GL?
  - Open specification, not open source
- The OpenGL API defines functions that your graphics card must support in order to render images with it

The OpenGL Pipeline

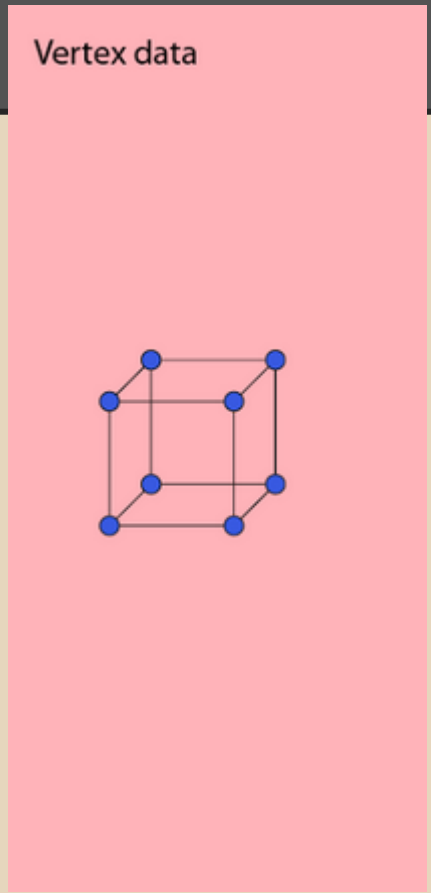


The Real OpenGL Pipeline



Pipeline: Vertex Data

- We begin by sending our vertex data to our graphics card
  - Vertex position, vertex normal, vertex color, etc.
- We use VBOS (vertex buffer objects) to pass the data
- Use calls to glVertexAttribPointer function
  - Called inside ShaderProgram::Draw()

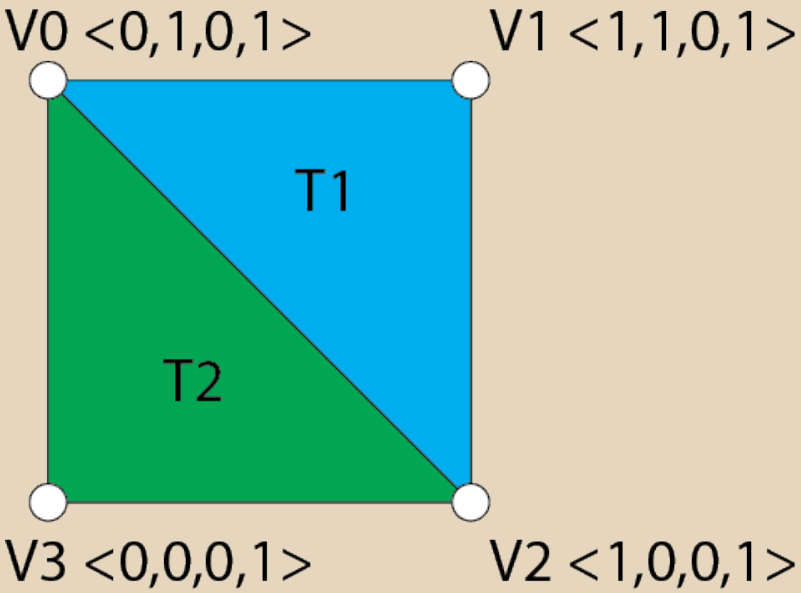


Vertex Buffer Objects (VBOs)

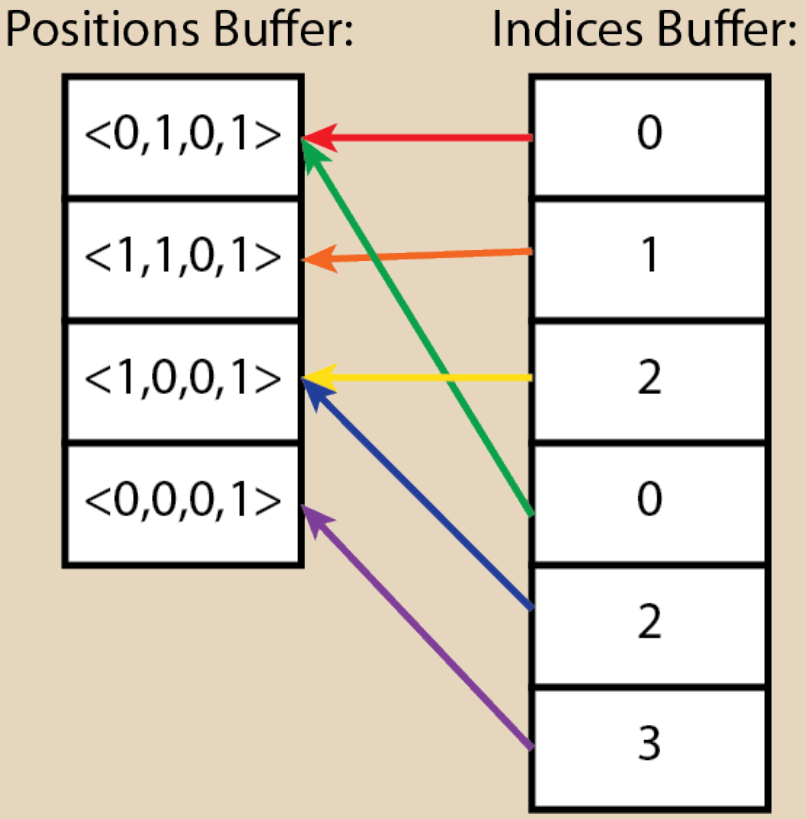
- Buffers stored on our GPU that hold vertex data such as position, normal, and color
- Once you've passed data to the VBO, you can't read it back
  - It's stored on your GPU, which is inaccessible to your CPU-executed program

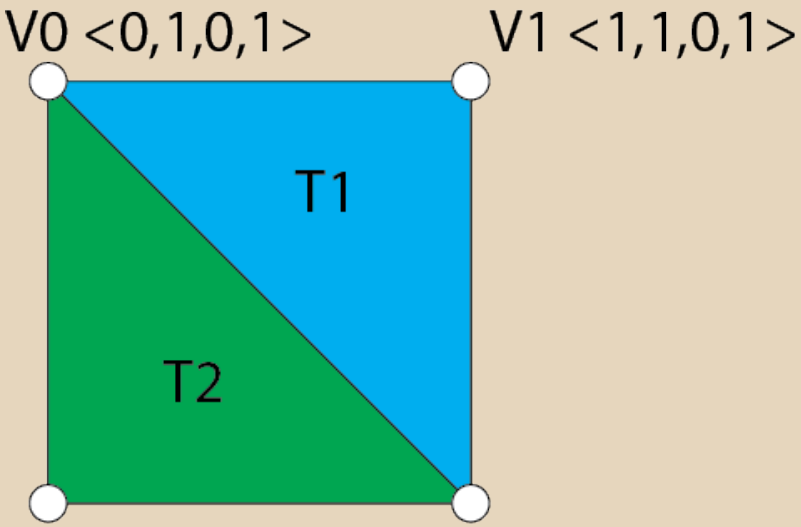
VBOs: A Visualization

- A square is comprised of four vertices
- We draw geometry in OpenGL by dividing it into triangles
  - Why?
- Need to read from the same vertex more than once



VBOs: A Visualization





T1 = [V0, V1, V2]
T2 = [V0, V2, V3]

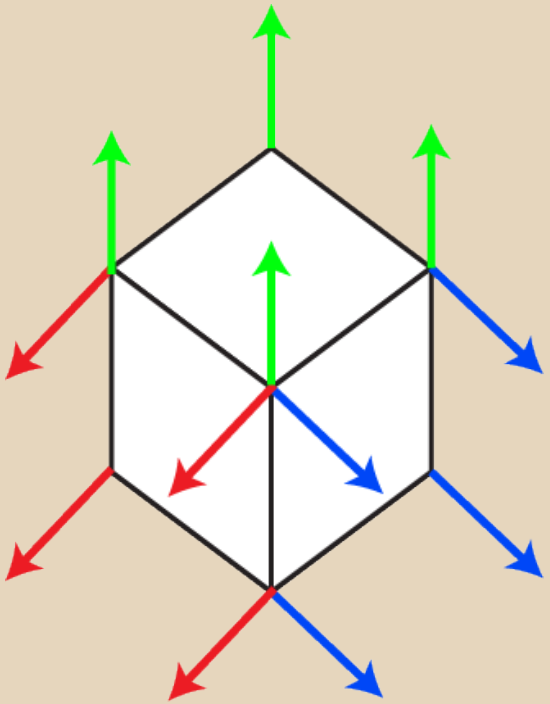
Lining up VBO data

- Vertex positions, normals, colors, and any other data must match up in a 1:1 ratio
  - i.e. if two faces share vertex positions but have different face normals, two vertex positions with the same value must be created in order to be associated with the two different vertex normals

VBOs: Concept test

How long will our position, normal, and index arrays have to be for this cube?

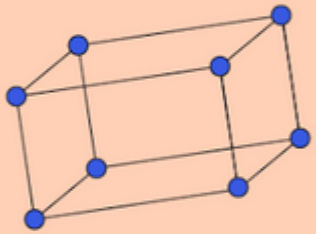
1. Pos: 8, Nor: 24, Idx: 24
2. Pos: 8, Nor: 8, Idx: 24
3. Pos: 8, Nor: 24, Idx: 36
4. Pos: 8, Nor: 36, Idx: 36
5. Pos: 24, Nor: 24, Idx: 24
6. Pos: 24, Nor: 36, Idx: 36
7. Pos: 24, Nor: 24, Idx: 36



Pipeline: Vertex Shader

- The vertex shader code gets executed on our graphics card
- GPUs have many execution threads with which to run shaders
  - Can execute the shader on multiple vertices at the same time
- Vertex shaders are used to transform the input vertex data
  - Calling it a “shader” is an archaism

Vertex shader
(Doesn’t actually “shade”)



Shaders

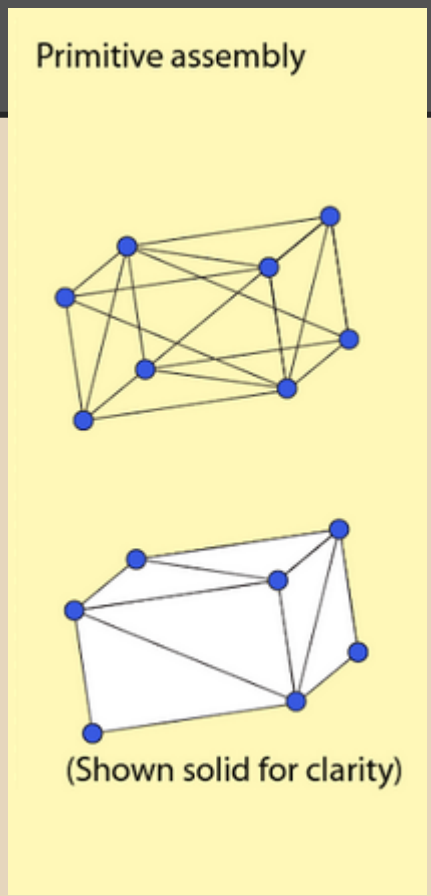
- Written in the OpenGL Shading Language (GLSL)
- Two main types of variables:
  - Uniforms are constant across all instances of a shader program
  - Ins/Outs differ across instances
    - In our case, they’re the per-vertex information
- Shaders all have a main function that is run when the shader program is run
- No GLSL headers

GLSL

- Like C++ but without:
  - Pointers
  - Dynamic memory allocation
  - Recursion
  - User-defined classes
- Has its own built-in classes
  - vec2, vec3, vec4, mat2, mat3, mat4, etc.
- Also has a built-in linear algebra library for common graphics functions
  - normalize, length, dot, cross, reflect, etc.
- We had you write your linear algebra library to follow the GLSL API
  - While the functions called in the shaders may look like the ones you wrote, they are different function calls!

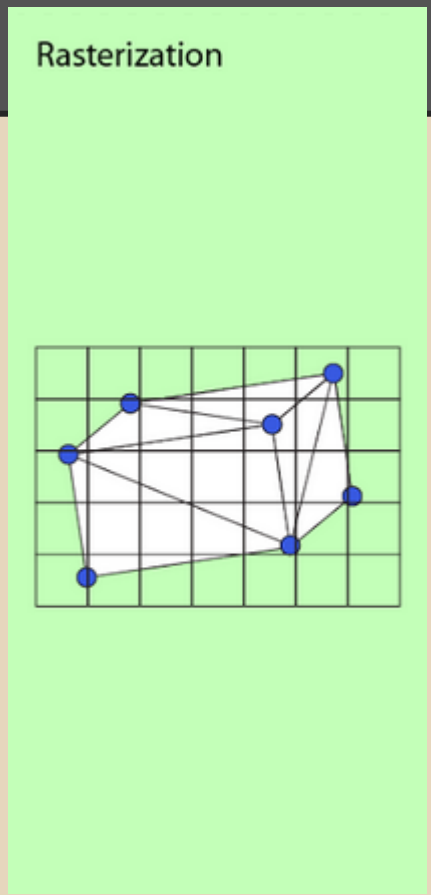
Pipeline: Primitive Assembly

- The transformed vertices are read in the order prescribed by the index buffer
- Triangles are created by this process to form solid geometry



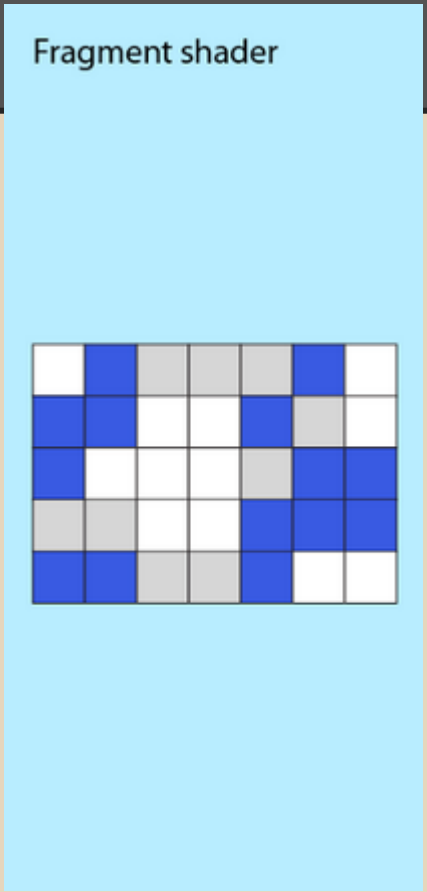
Pipeline: Rasterization

- The screen is divided into cells, with each cell corresponding to one pixel
- Every triangle overlapped by a pixel is divided into a **fragment**
- If multiple triangles are overlapped by a pixel, each one has a fragment made from it



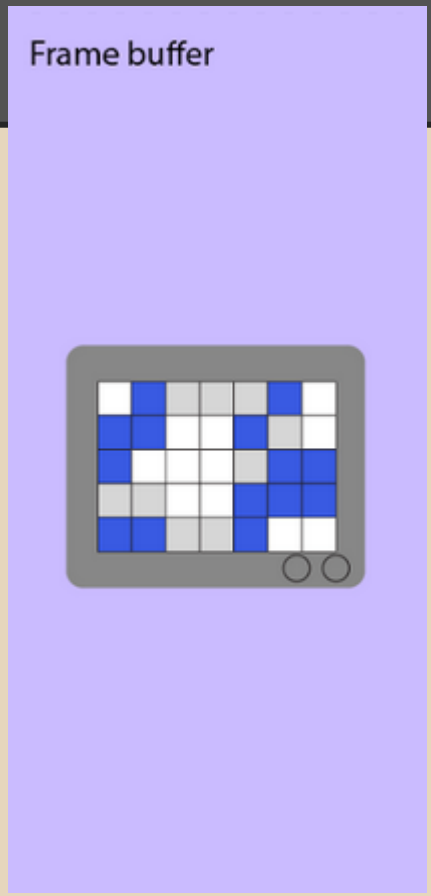
Pipeline: Fragment Shader

- The fragment shader is run on every fragment generated by the rasterization process
- If multiple fragments exist for one pixel, which fragment to actually render is decided after the shader has processed each fragment

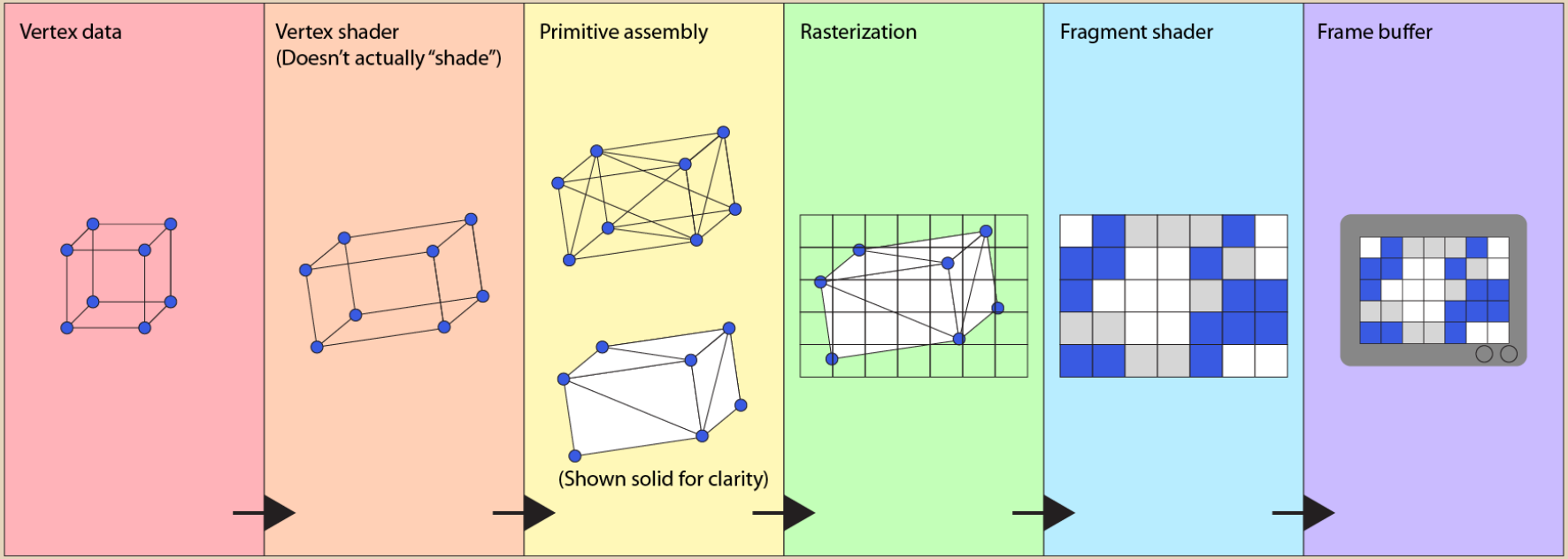


Pipeline: Frame Buffer

- The results of the fragment shader executions are sorted and sent to the frame buffer
- This array of pixels is sent to our screen, where we finally see our geometry transformed and colored



The OpenGL Pipeline



Fun with shaders

Now that we've gone over how shaders work, let's use our vertex and fragment shaders to make some cool-looking effects!