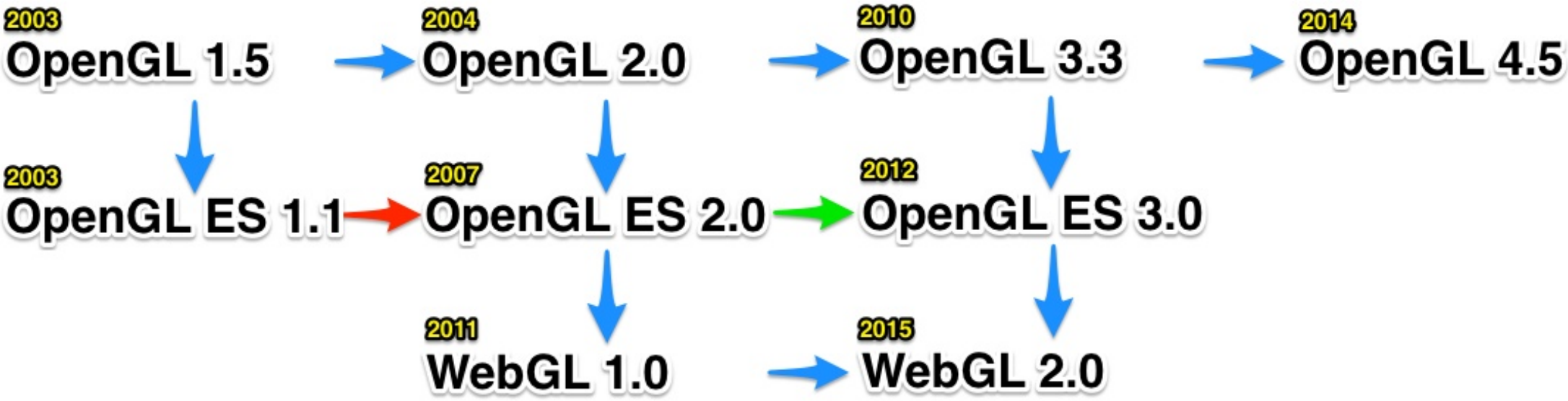
- I am no expert. We are all learning.
- Slide/Code/Resources not mine, mostly copied from different resources.
- Code shared may have bugs, modify as needed, write your own.
- Start early. Assignments are incremental. Focus now, will be helpful later.
- Collaborate, take help, but DO NOT copy code.

- Ask questions.
- Use slack for faster response. Describe issue with images/video.

WebGL Versions



Why WebGL ?

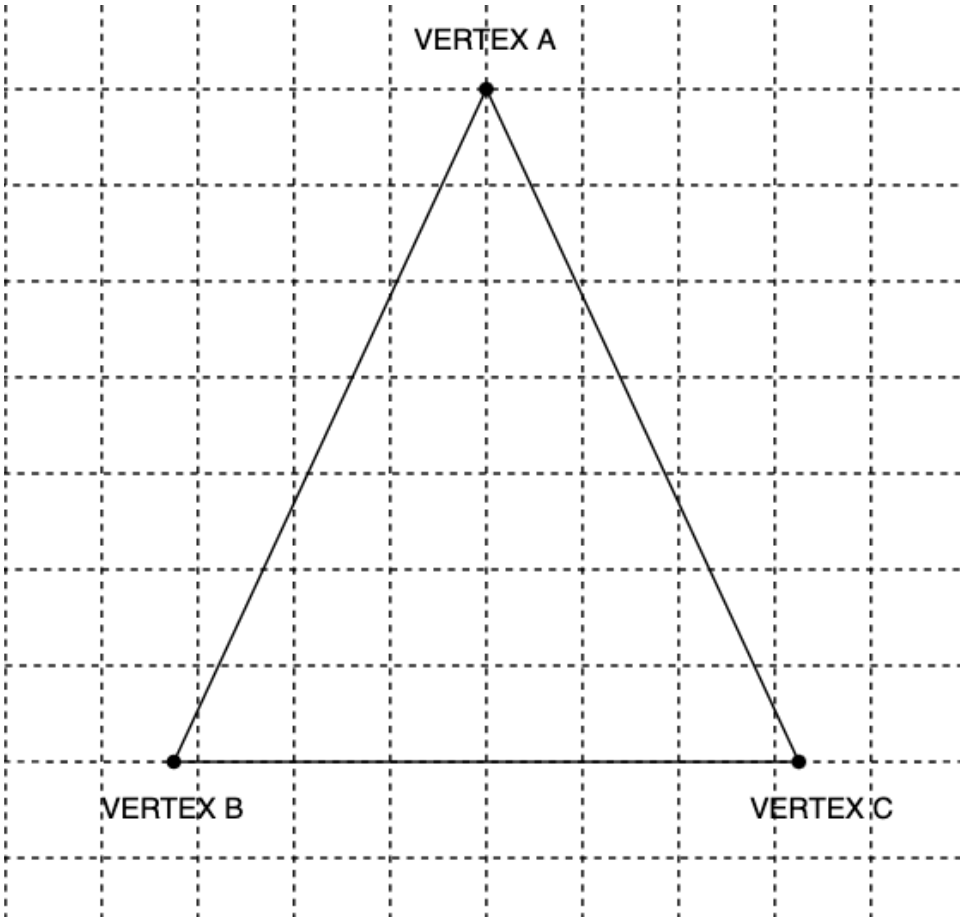
- I just need to make games, why not just learn Unity/Unreal/ThreeJS ?
- Creating a library/engine vs using one.

What will we learn?

- Drawing triangles, at the correction position in space, with right attributes.
- This is usually the most difficult step to understand, everything else just follows.

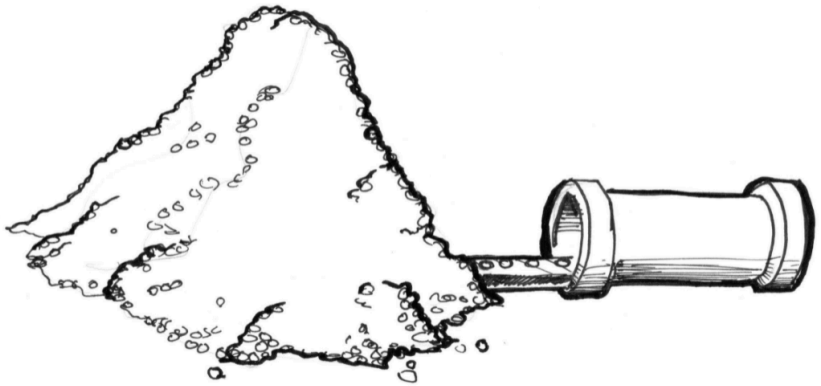
https://sketchfab.com/3d-models/pubg-male-model-d45092e3358943c29c7d7f79ceffb0d0

Why GPU ?

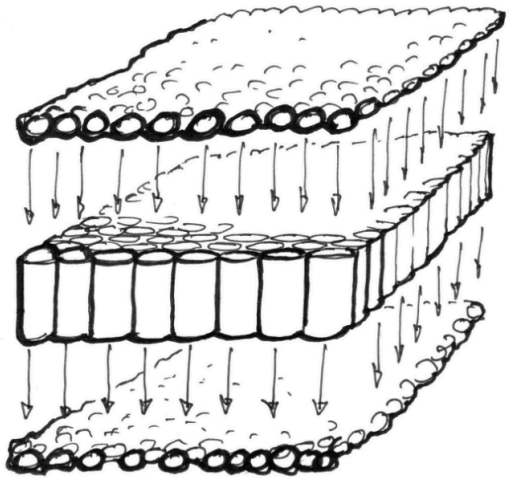


2880x1800 retina display running at 60 frames per sec

311,040,000 calculations per second



This is when parallel processing becomes a good solution. Instead of having a couple of big and powerful microprocessors, or *pipes*, it is smarter to have lots of tiny microprocessors running in parallel at the same time. That's what a Graphic Processor Unit (GPU) is.



In order to run in parallel every pipe, or thread, has to be independent from every other thread. We say the threads are *blind* to what the rest of the threads are doing. This restriction implies that all data must flow in the same direction. So it's impossible to check the result of another thread, modify the input data, or pass the outcome of a thread into another thread. Allowing thread-to-thread communications puts the integrity of the data at risk.

Also the GPU keeps the parallel micro-processor (the pipes) constantly busy; as soon as they get free they receive new information to process. It's impossible for a thread to know what it was doing in the previous moment. It could be drawing a button from the UI of the operating system, then rendering a portion of sky in a game, then displaying the text of an email. Each thread is not just **blind** but also **memoryless**. Besides the abstraction required to code a general function that changes the result pixel by pixel depending on its position, the blind and memoryless constraints make shaders not very popular among beginning programmers.

Overall requirements

- Generate some data on CPU
- Create some variables on GPU to store data
- Transfer data from CPU to GPU memory for parallel processing
- Create some special program which will run on GPU
- Compile-Link these program and provide to GPU
- Render

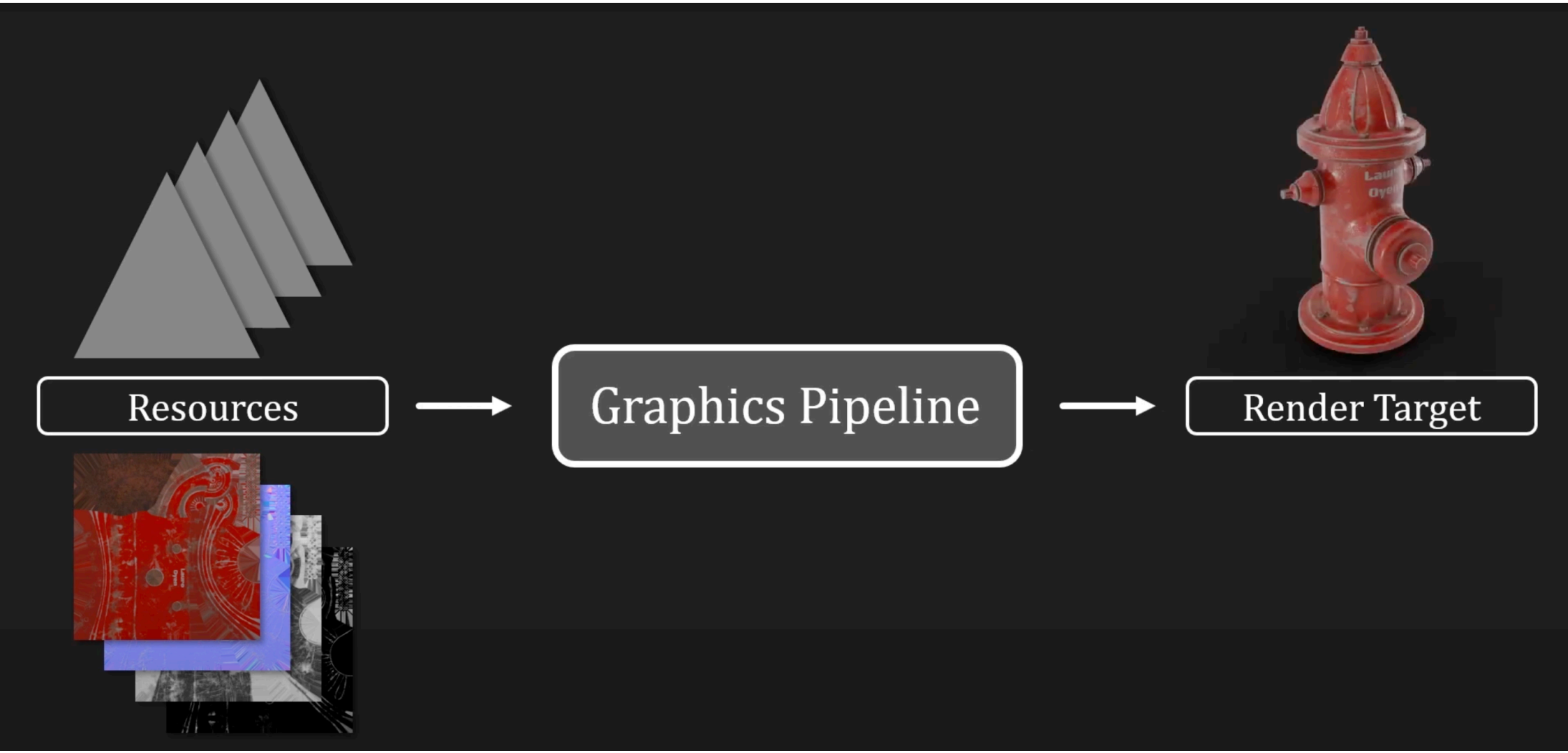
WebGL Programming in a Nutshell

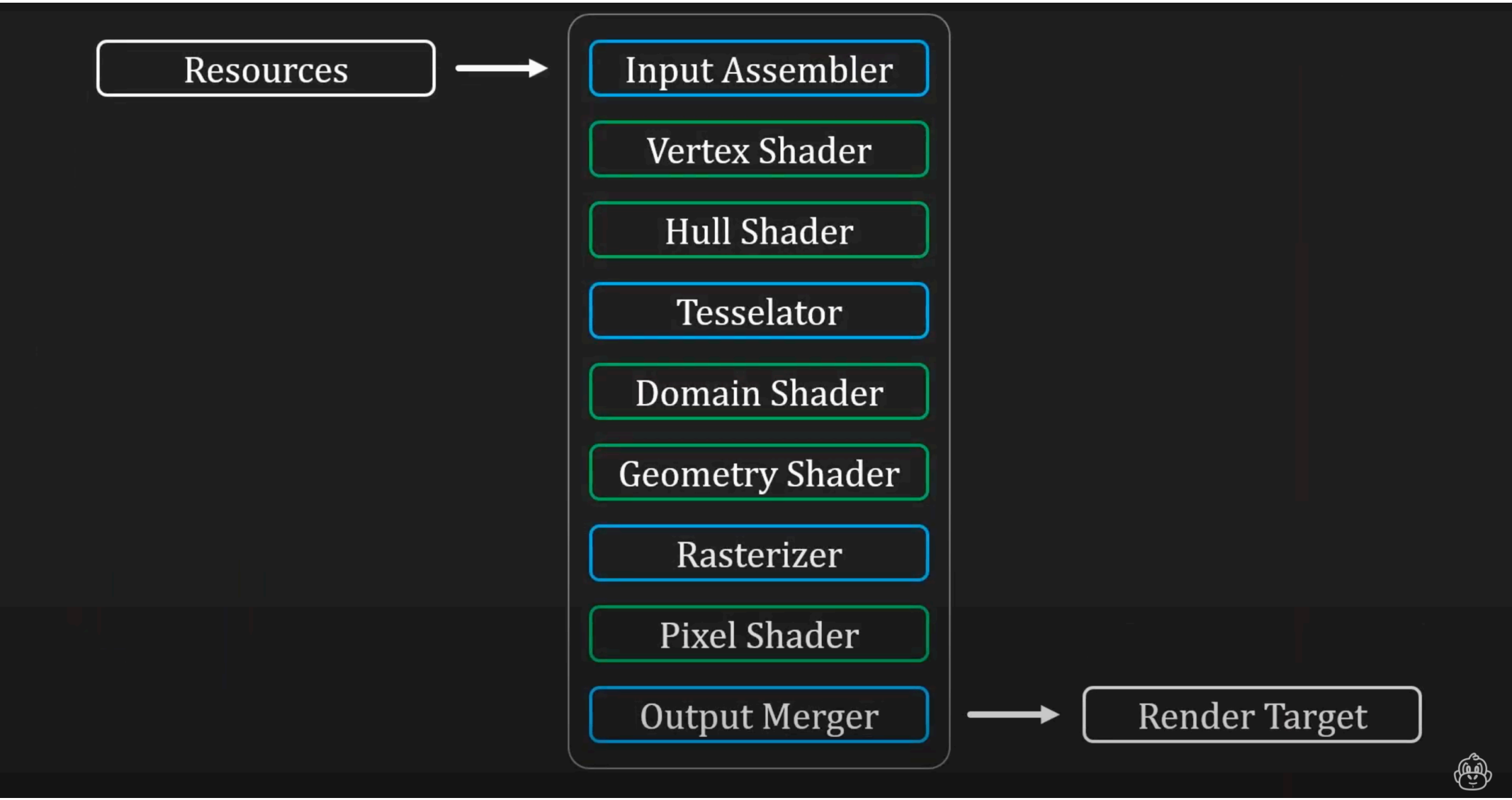
- All WebGL programs must do the following:
  - Set up canvas to render onto
  - Generate data in application
  - Create shader programs
  - Create buffer objects and load data into them
  - “Connect” data locations with shader variables
  - Render

Minimal Sample

https://github.com/Amit-Tomar/T2-21-CS-606/blob/main/src/example1/index.html

How rendering works
(https://www.youtube.com/watch?v=brDJVEPOeY8)


Resources
Graphics Pipeline
Render Target


Resources
Input Assembler
Vertex Shader
Hull Shader
Tesselator
Domain Shader
Geometry Shader
Rasterizer
Pixel Shader
Output Merger
Render Target

4.55,4.99,5.03 0.09,0.56 0.50,0.73,0.21
4.78,4.27,2.53 0.76,0.48 0.42,0.76,0.26
7.69,3.84,2.03 0.44,0.65 0.81,0.82,0.41
4.56,4.56,5.14 0.38,0.67 0.03,0.57,0.34
2.64,5.26,3.68 0.64,0.48 0.86,0.84,0.27
4.22,3.41,5.73 0.48,0.31 0.70,0.56,0.08
0.98,3.33,6.41 0.34,0.03 0.64,0.13,0.57
6.53,7.09,4.02 0.93,0.26 0.31,0.61,0.58
7.95,7.88,3.59 0.61,0.32 0.24,0.27,0.54
2.45,3.21,1.19 0.29,0.54 0.46,0.33,0.15

(4.55,4.99,5.03), (0.09,0.56), (0.50,0.73,0.21),
(4.78,4.27,2.53), (0.76,0.48), (0.42,0.76,0.26),
(7.69,3.84,2.03), (0.44,0.65), (0.81,0.82,0.41),
(4.56,4.56,5.14), (0.38,0.67), (0.03,0.57,0.34),
(2.64,5.26,3.68), (0.64,0.48), (0.86,0.84,0.27),
(4.22,3.41,5.73), (0.48,0.31), (0.70,0.56,0.08),
(0.98,3.33,6.41), (0.34,0.03), (0.64,0.13,0.57),
(6.53,7.09,4.02), (0.93,0.26), (0.31,0.61,0.58),
(7.95,7.88,3.59), (0.61,0.32), (0.24,0.27,0.54),
(2.45,3.21,1.19), (0.29,0.54), (0.46,0.33,0.15),

Position	3	FLOAT 32
UV	2	FLOAT 32
Normal	3	FLOAT 32

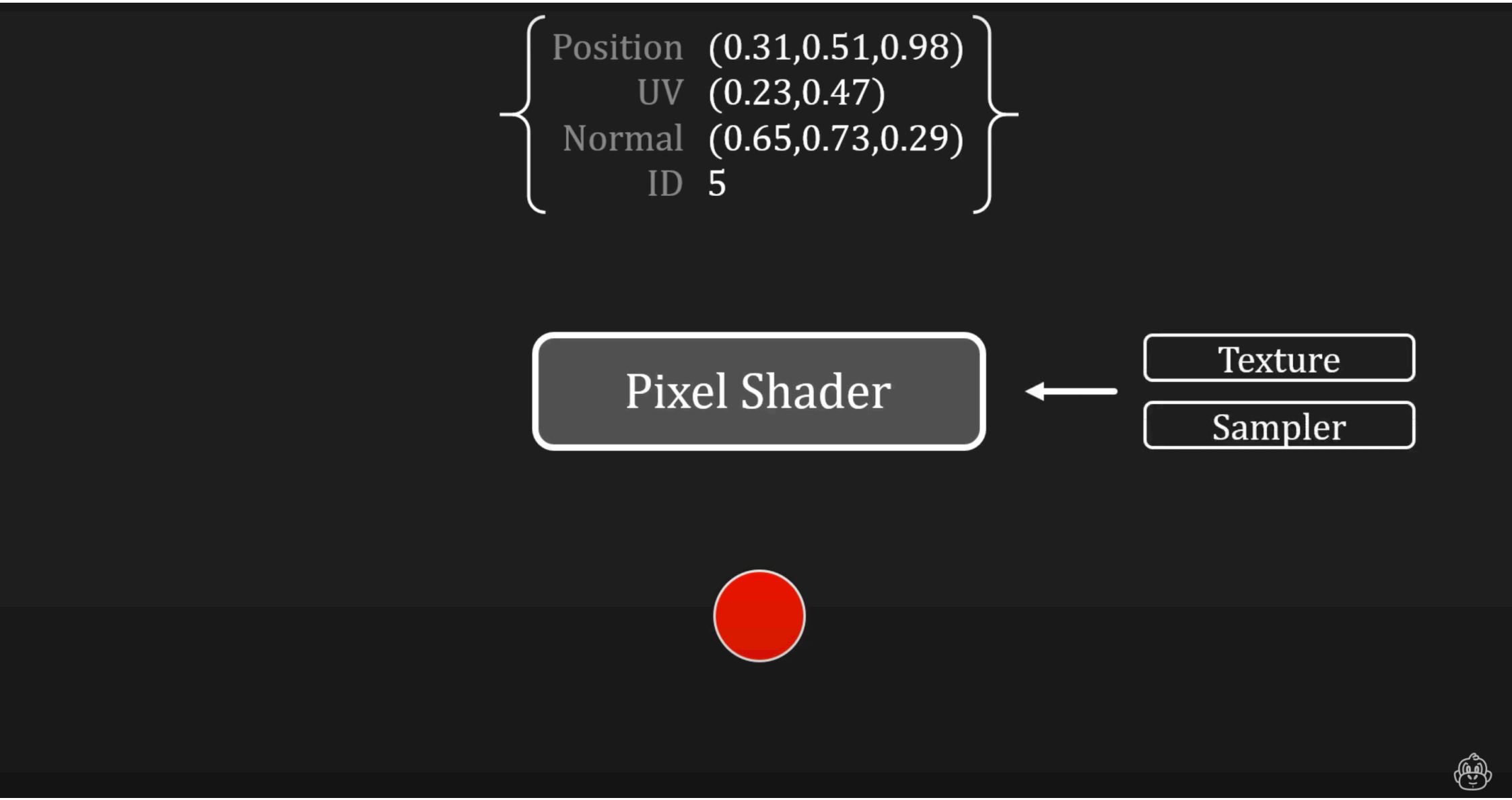
Position (4.22,3.41,5.73)
UV (0.48,0.31)
Normal (0.70,0.56,0.08)
ID 5

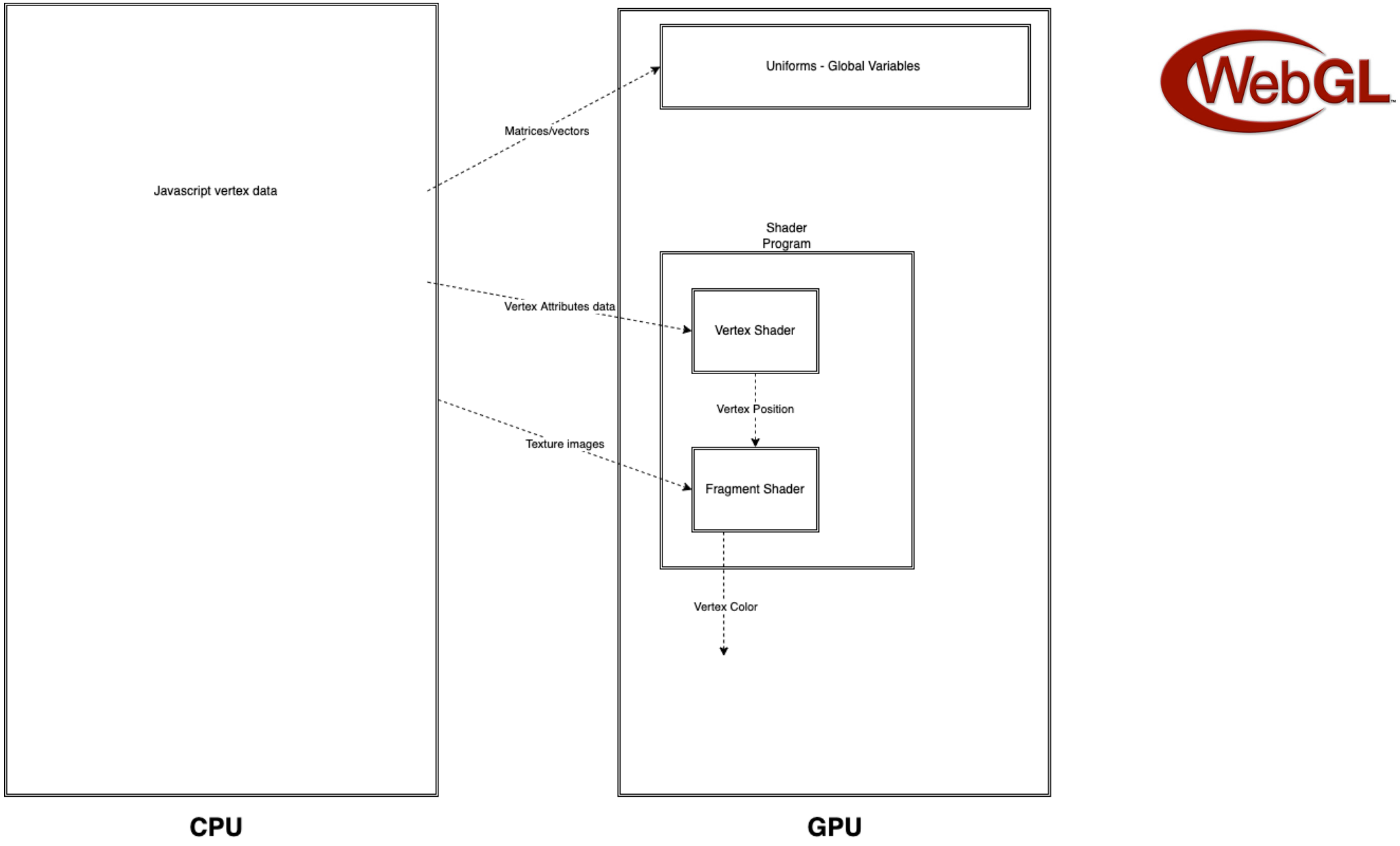
Vertex Shader

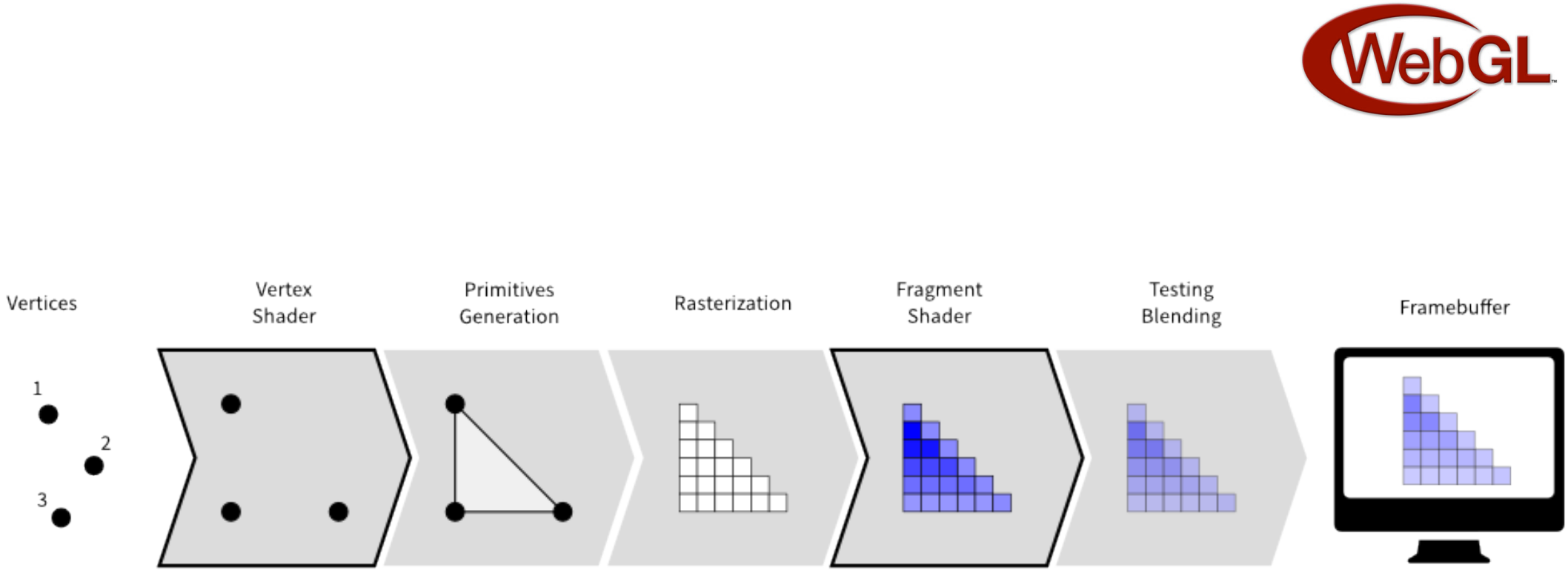
Matrix

Position (0.57,0.35,0.82)
UV (0.48,0.31)
Color (0.42,0.96,0.31)

$-1 \leq x \leq 1$
$-1 \leq y \leq 1$
$-1 \leq z \leq 1$

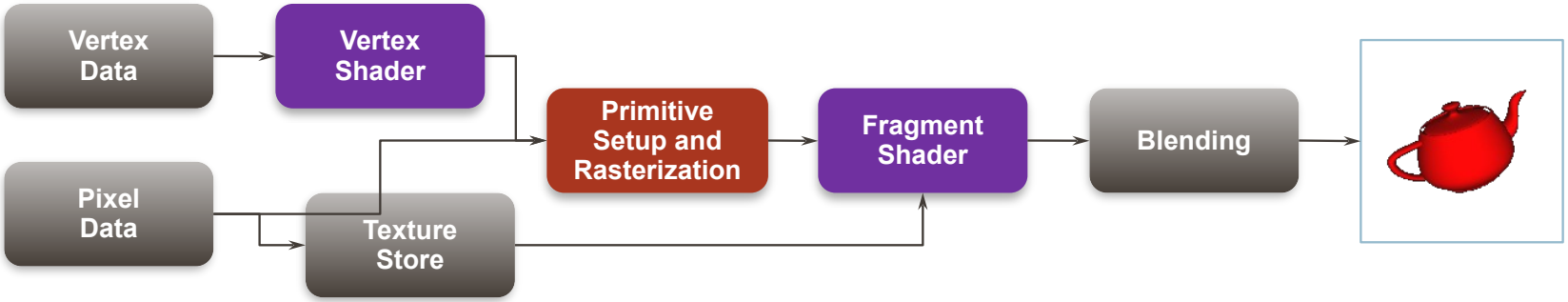

Position (0.31,0.51,0.98)
UV (0.23,0.47)
Normal (0.65,0.73,0.29)
ID 5
Pixel Shader
Texture
Sampler


Uniforms - Global Variables
Matrices/vectors
Javascript vertex data
Shader
Program
Vertex Attributes data
Vertex Shader
Vertex Position
Texture images
Fragment Shader
Vertex Color
CPU
GPU

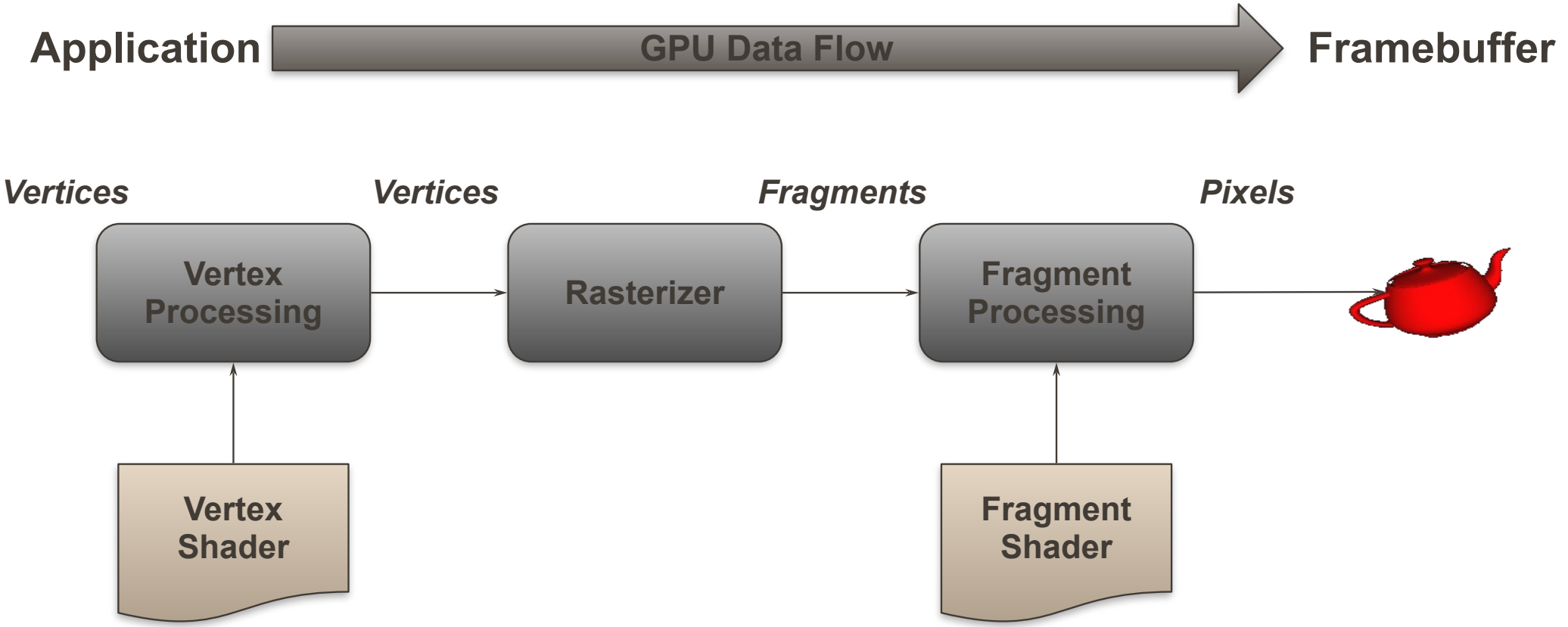

Vertices
Vertex Shader
Primitives Generation
Rasterization
Fragment Shader
Testing Blending
Framebuffer
1
2
3

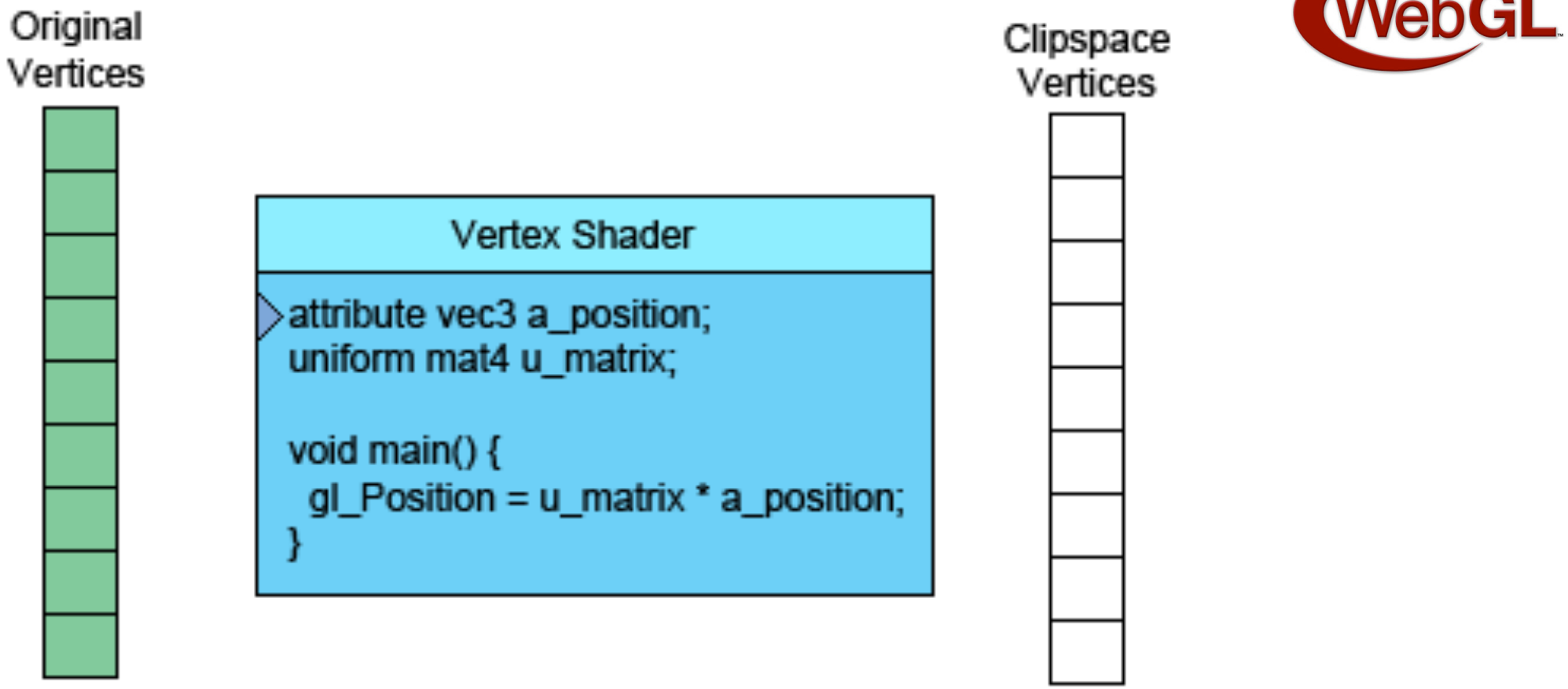
Programmable Pipeline

- Fixed-function vs Programmable shaders



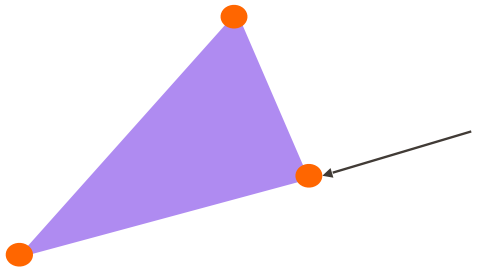
Simplified Pipeline Model




Original
Vertices
Vertex Shader
attribute vec3 a_position;
uniform mat4 u_matrix;
void main() {
gl_Position = u_matrix * a_position;
}
Clipspace
Vertices
WebGL

Representing Geometric Objects

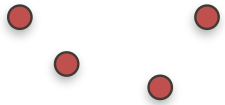
- Geometric objects are represented using *vertices*
- A vertex is a collection of generic attributes
  - positional coordinates
  - colors
  - texture coordinates
  - any other data associated with that point in space
- Position stored in 4 dimensional homogeneous coordinates
- Vertex data must be stored in vertex buffer objects (VBOs)



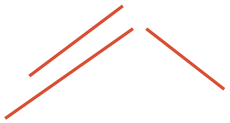
$$\mathbf{p} = \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

WebGL Geometric Primitives

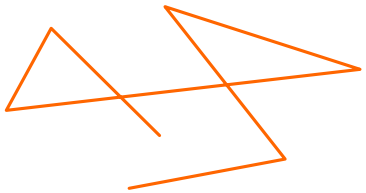
- All primitives are specified by vertices



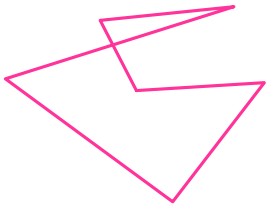
`gl.POINTS`



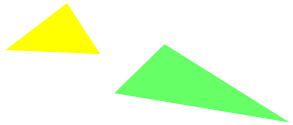
`gl.LINES`



`gl.LINE_STRIP`



`gl.LINE_LOOP`



`gl.TRIANGLES`

`gl.TRIANGLE_STRIP`

`gl.TRIANGLE_FAN`

Vertex Shaders

- A shader that's executed for each vertex
  - Each instantiation can generate one vertex
  - Outputs are passed on to the rasterizer where they are interpolated and available to fragment shaders
  - Position output in clip coordinates

- There are lots of effects we can do in vertex shaders
  - Changing coordinate systems
  - Moving vertices
  - Per vertex lighting: height fields

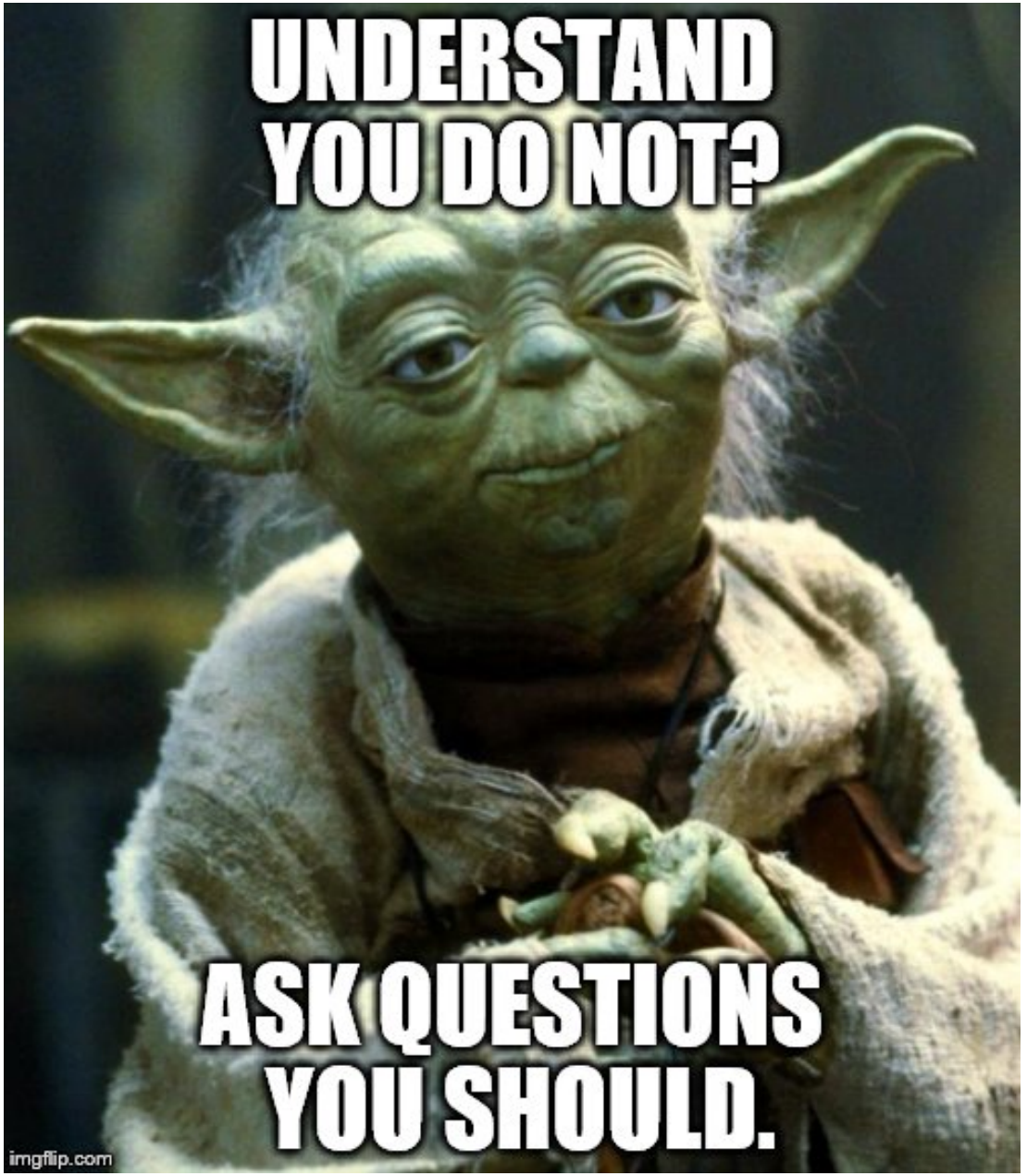
Fragment Shaders

- A shader that's executed for each "potential" pixel
  - fragments still need to pass several tests before making it to the framebuffer
- There are many effects we can implement in fragment shaders
  - Per-fragment lighting
  - Texture and bump Mapping
  - Environment (Reflection) Maps

How to run code samples ?

- https://github.com/Amit-Tomar/T2-21-CS-606

- Code will be updated frequently, please keep your forked branch in sync.



WebGL