University of Colorado
Boulder

Introduction to Virtual Reality

The Graphics Pipeline

Professor **Dan Szafir**

Computer Science & ATLAS Institute
University of Colorado Boulder

Recap of Wednesday

Using Three.js to create 3D graphics

Any questions on that?

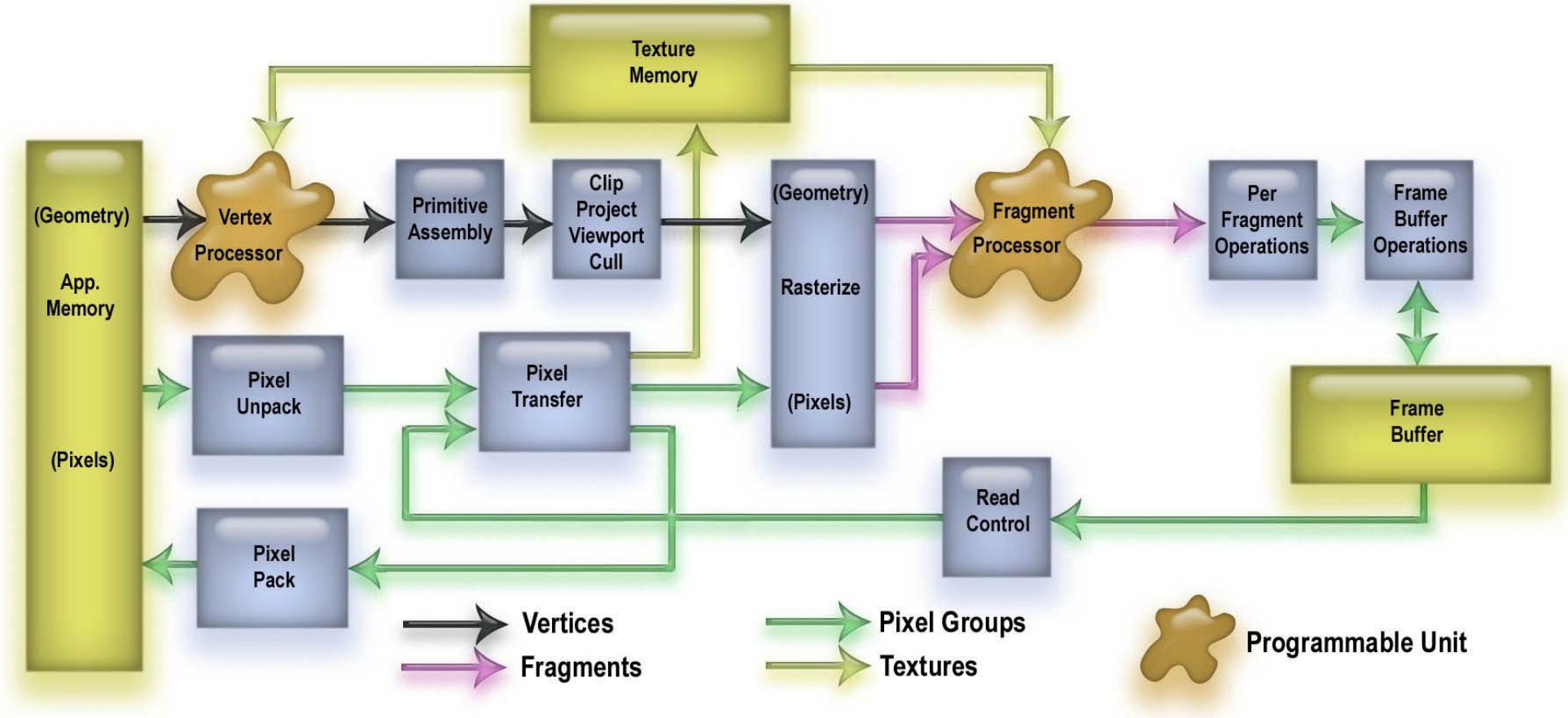
Let’s add animation

- Use the game/simulation loop!
- WebGL Exercises on Moodle
  - Exercise 2: Animating a Cube

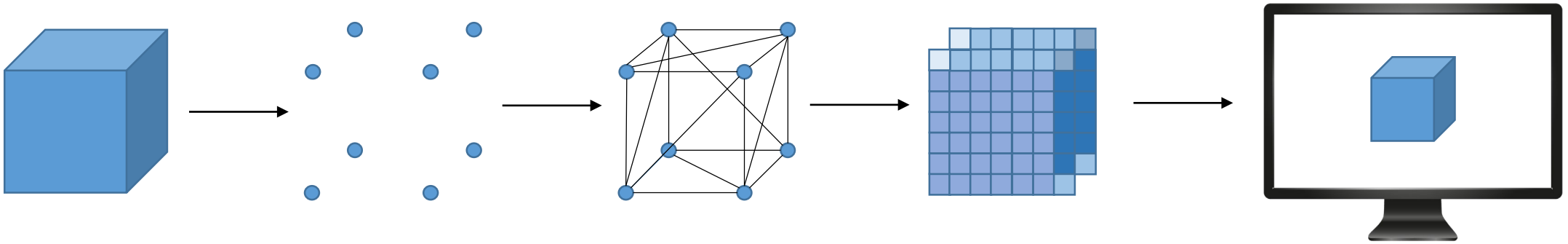
How does 3D content go from the application to the display?

The Graphics Pipeline

The Overly Complicated Version



The Pragmatic Version



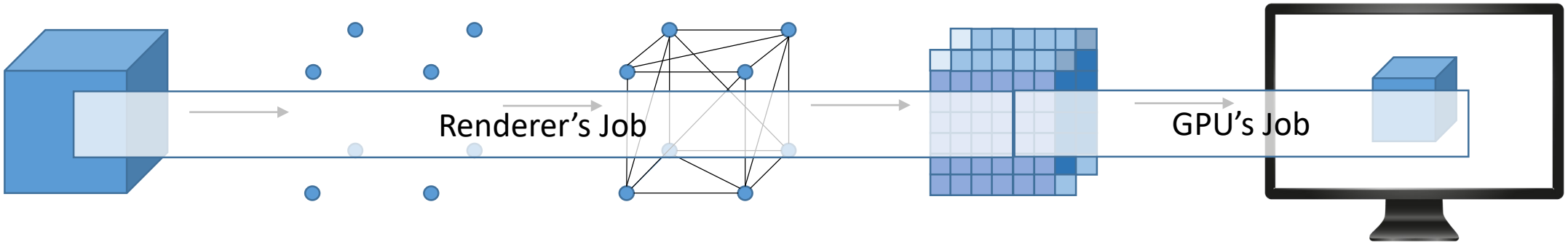
3D Scene Vertices Triangles Pixels Display

Today!

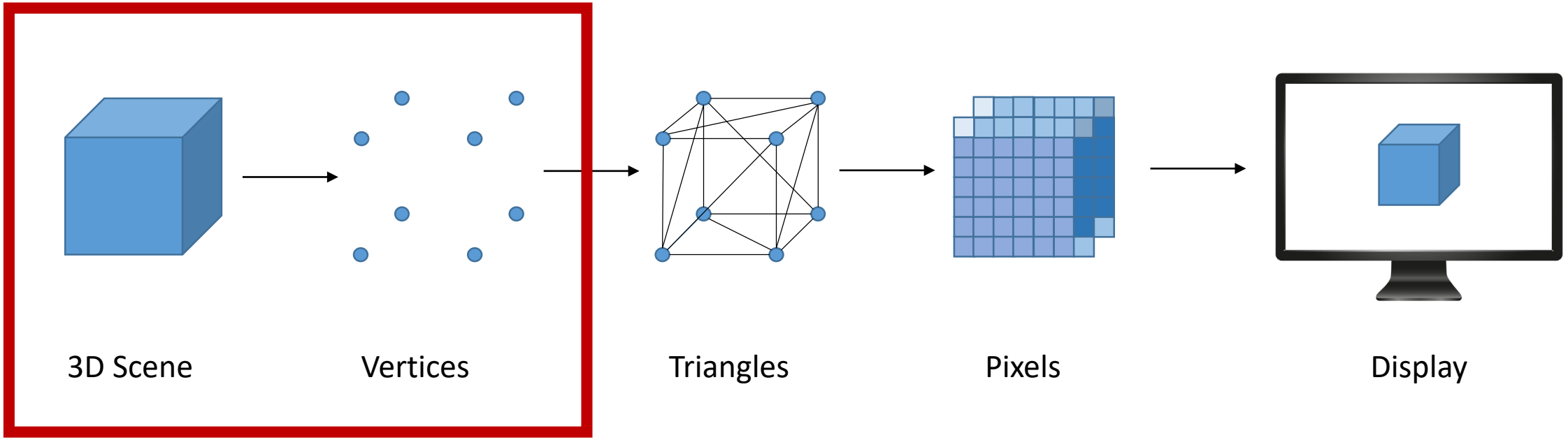
Renderers

Pull out the important information about a graphics application and translate it to something that the GPU can understand

Why use the GPU?

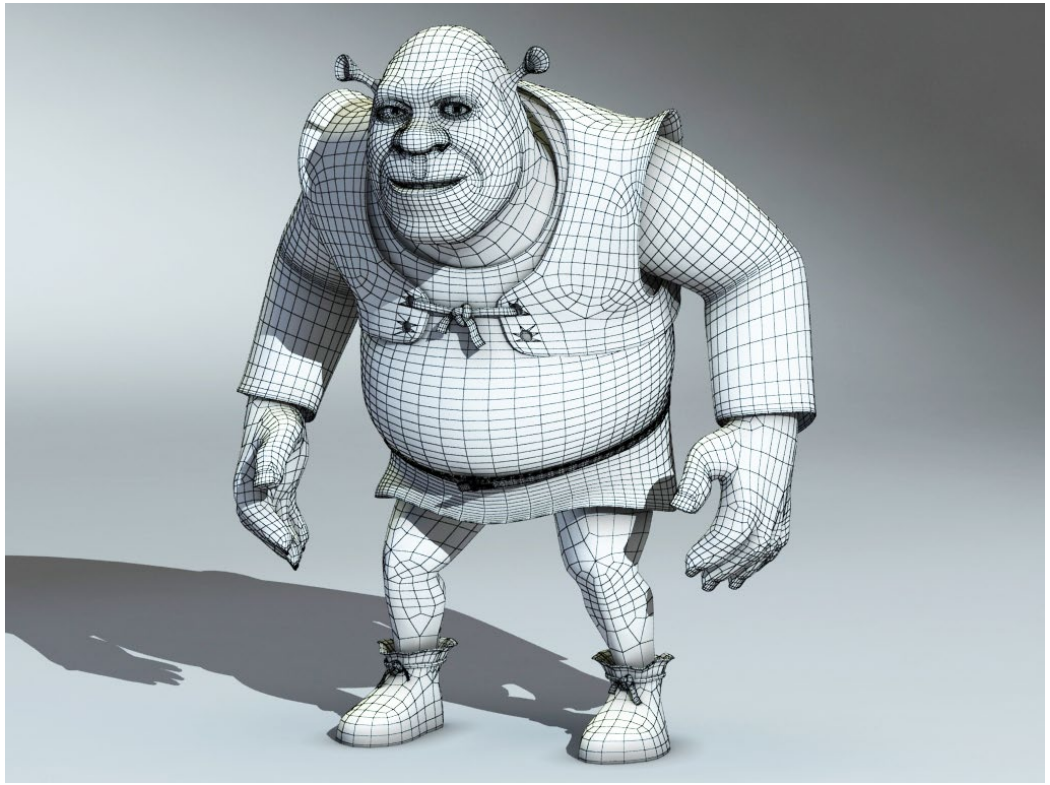


The Pragmatic Version



Meshes Revisited

Things have a structure (mesh) and an appearance (material)



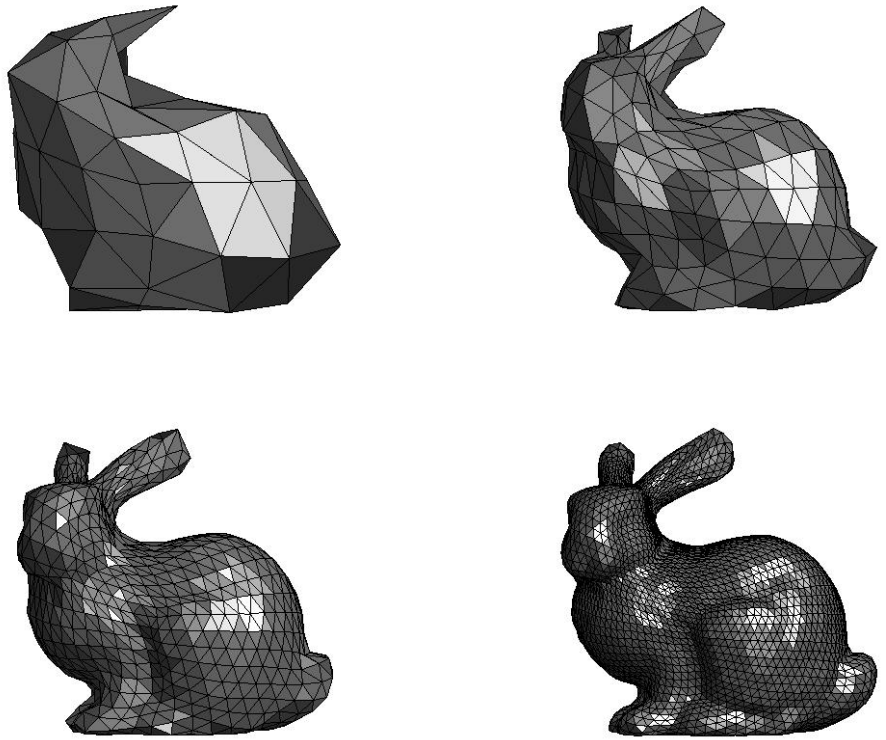
Meshes Revisited

Define the shape of an object

Generally triangles

Vertices: Points in a mesh

Edges: Lines connecting those points



Identify Vertices

Renderer picks out the position of all of the vertices (points) in the scene

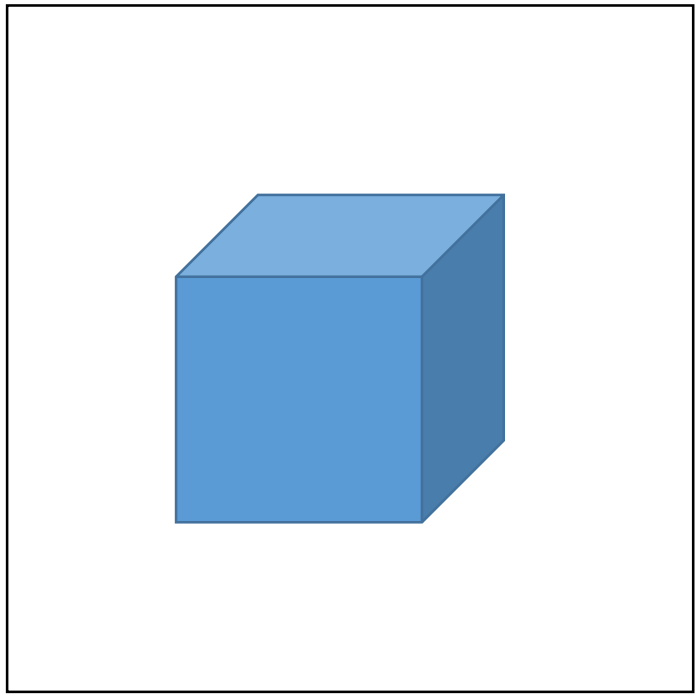
Projections

Vertices are positioned in **Model Space** (relative to other points in the mesh)

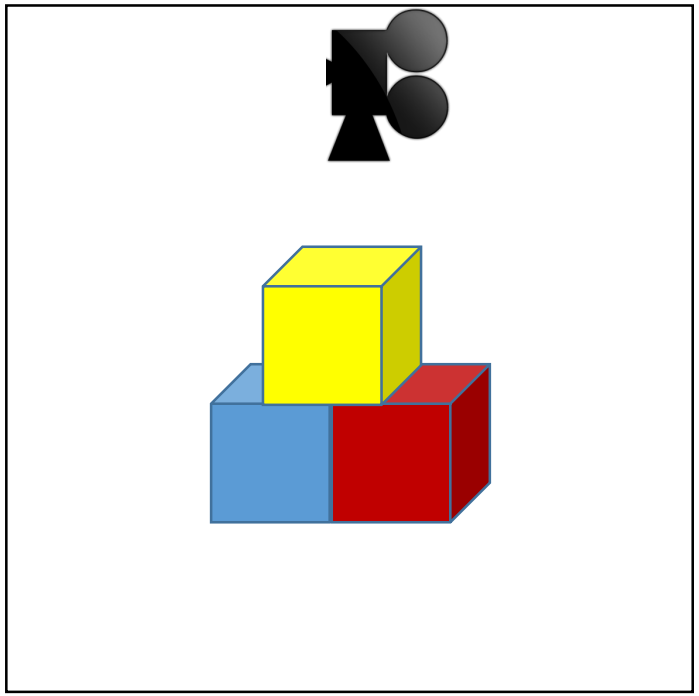
The renderer projects them into **World Space** (relative to everything in the scene)

The renderer then projects all of the vertices into **Camera Space** (relative to the camera)

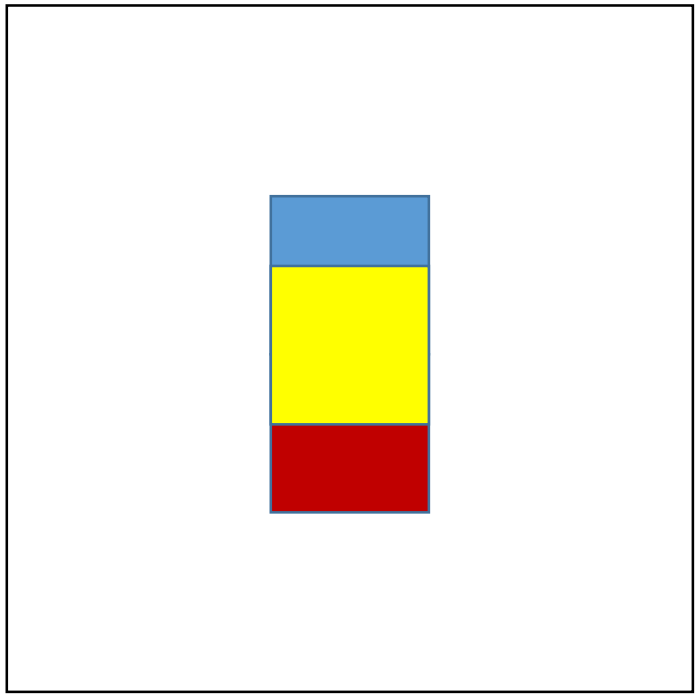
Projections



Model Space
(Relative to the Object)



World Space
(Relative to Center of the World)



Camera Space
(What is Rendered)

Color the Vertices

Renderer then computes how the lighting affects each vertex

Vertex Shader: Do something with every vertex

Why shaders?

http://threejs.org/examples/#webgl_shader

Color the Vertices

WebGL Exercises on Moodle

Exercise 3: Intro to Shaders #1

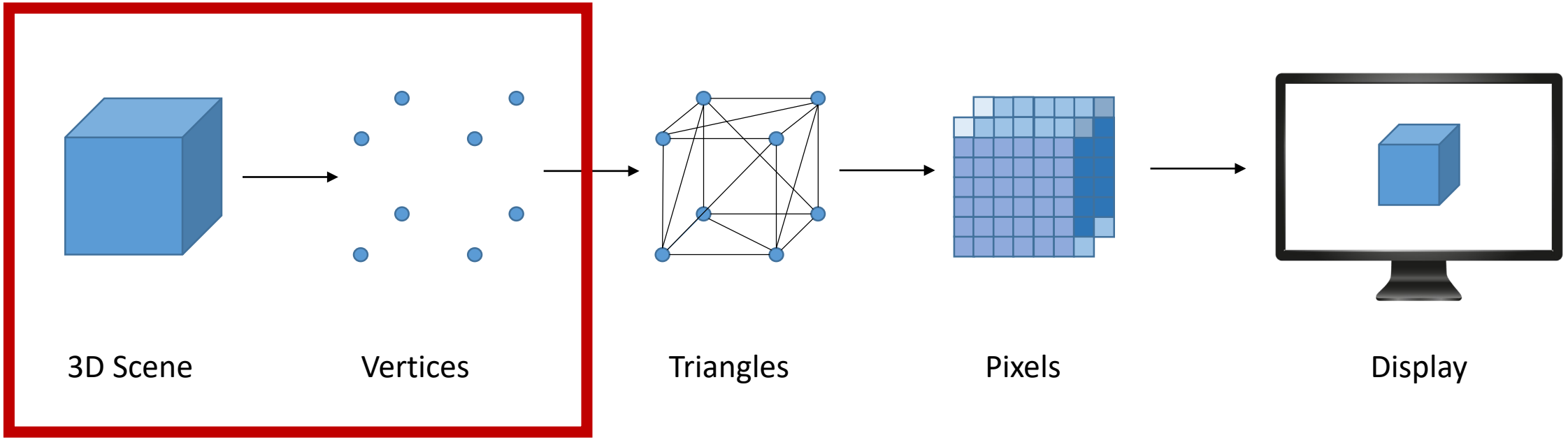
Vertex Shaders can also manipulate position

https://aerotwist.com/static/tutorials/an-introduction-to-shaders-part-2/demo/demo-6.html

WebGL Exercises on Moodle

Exercise 6: Vertex displacement with shaders

The Pragmatic Version



Assemble the Edges

Geometric Primitives

Generally triangles (occasionally rectangles or hexagons)

What not to draw

Lots of operations that must be done, so eliminate as much unnecessary work as possible

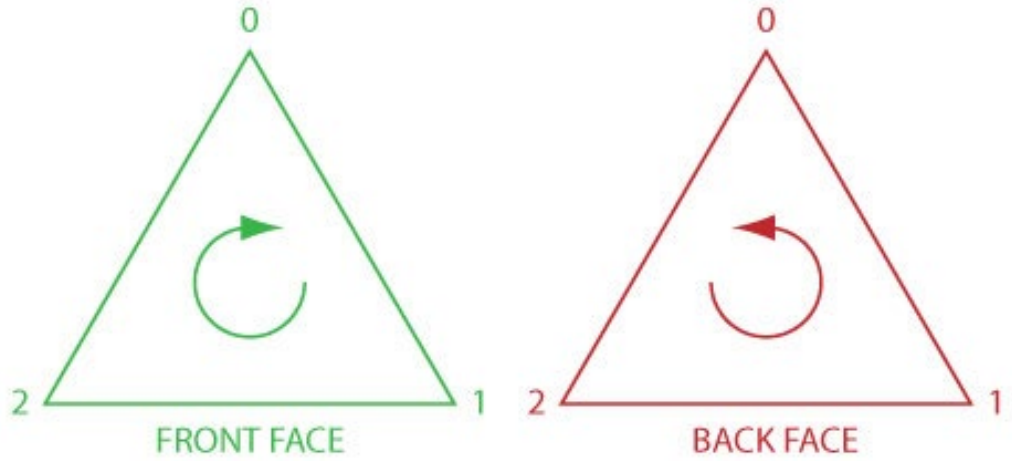
Meshes are hollow, but the objects they represent might not be

How can we save work when building a mesh?

Backface Culling

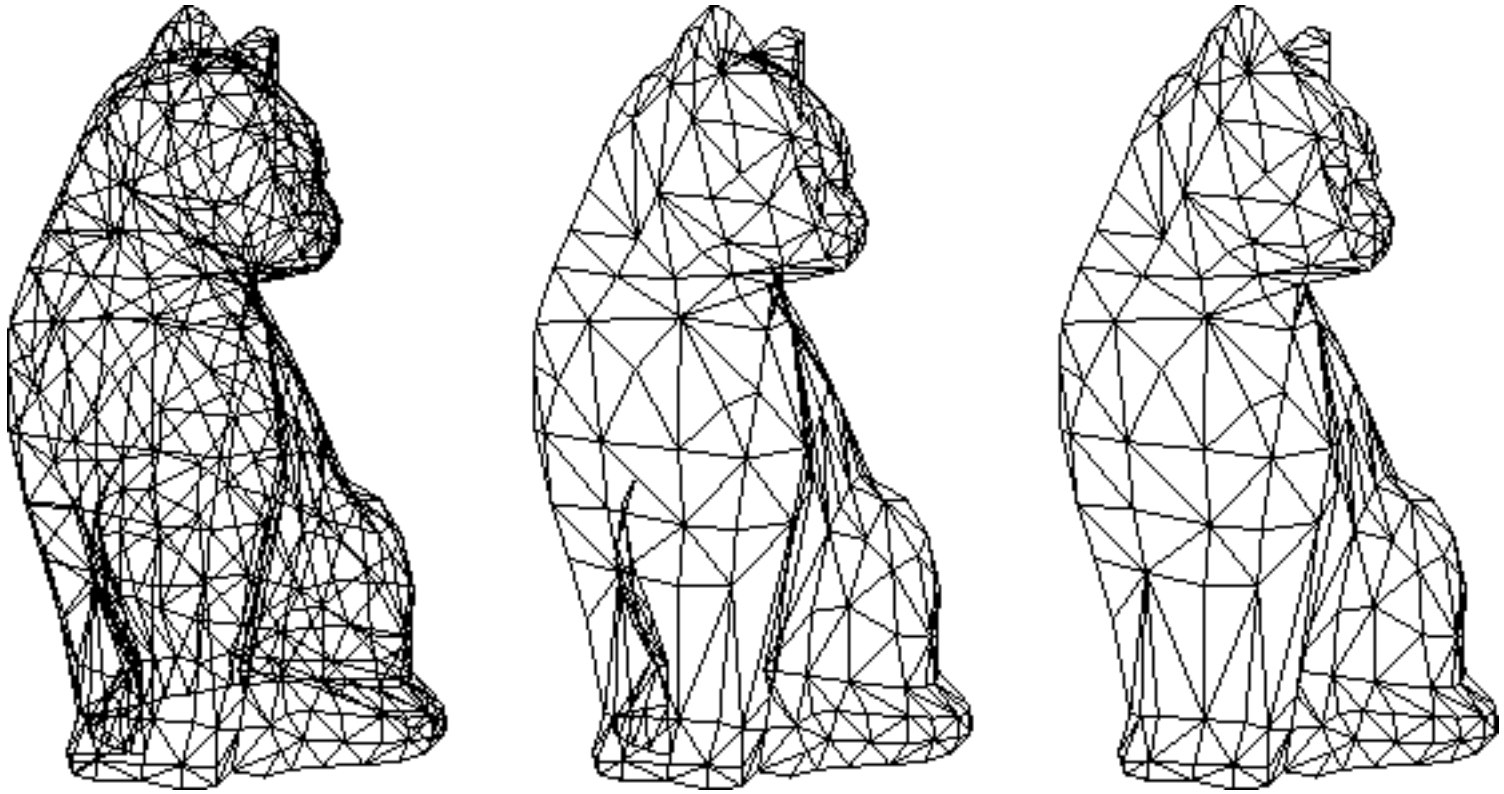
Answer: Don’t render anything where the normal is pointing away from the camera

Cull the back of any triangles

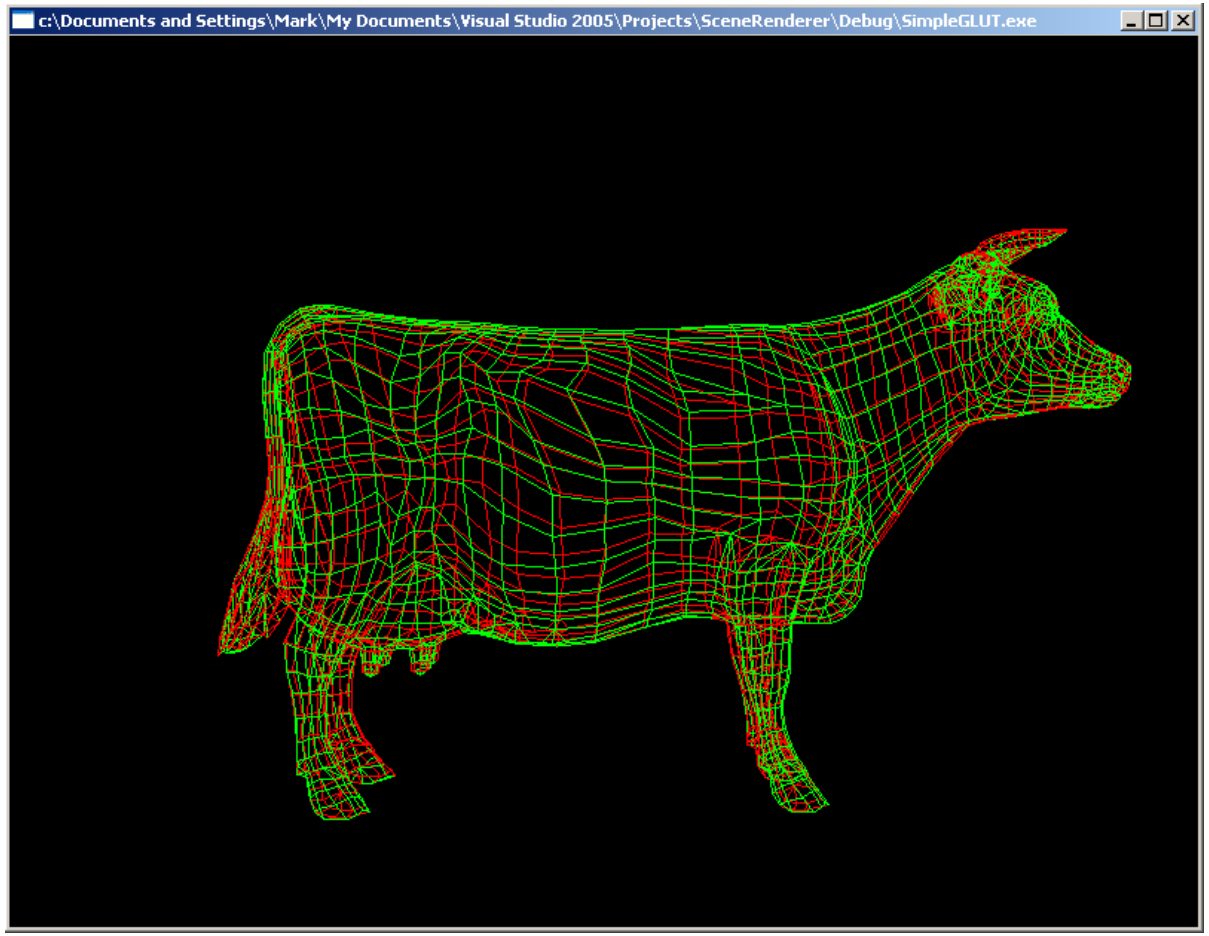


Normals: A vector perpendicular to the front of a primitive

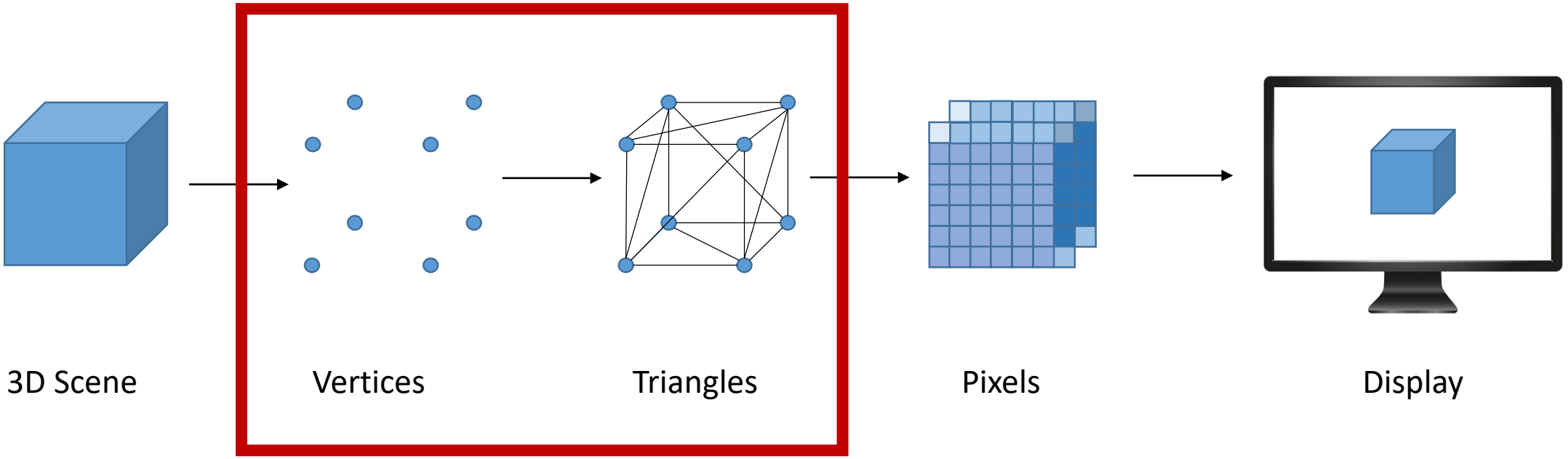
Backface Culling



Backface Culling

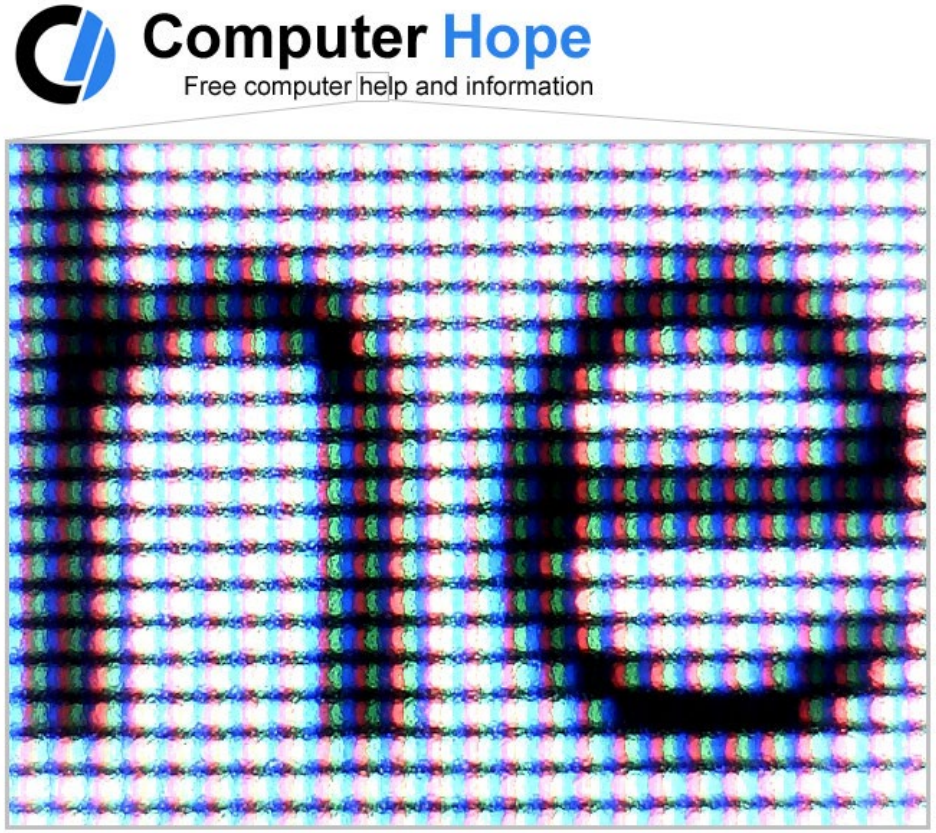


The Pragmatic Version



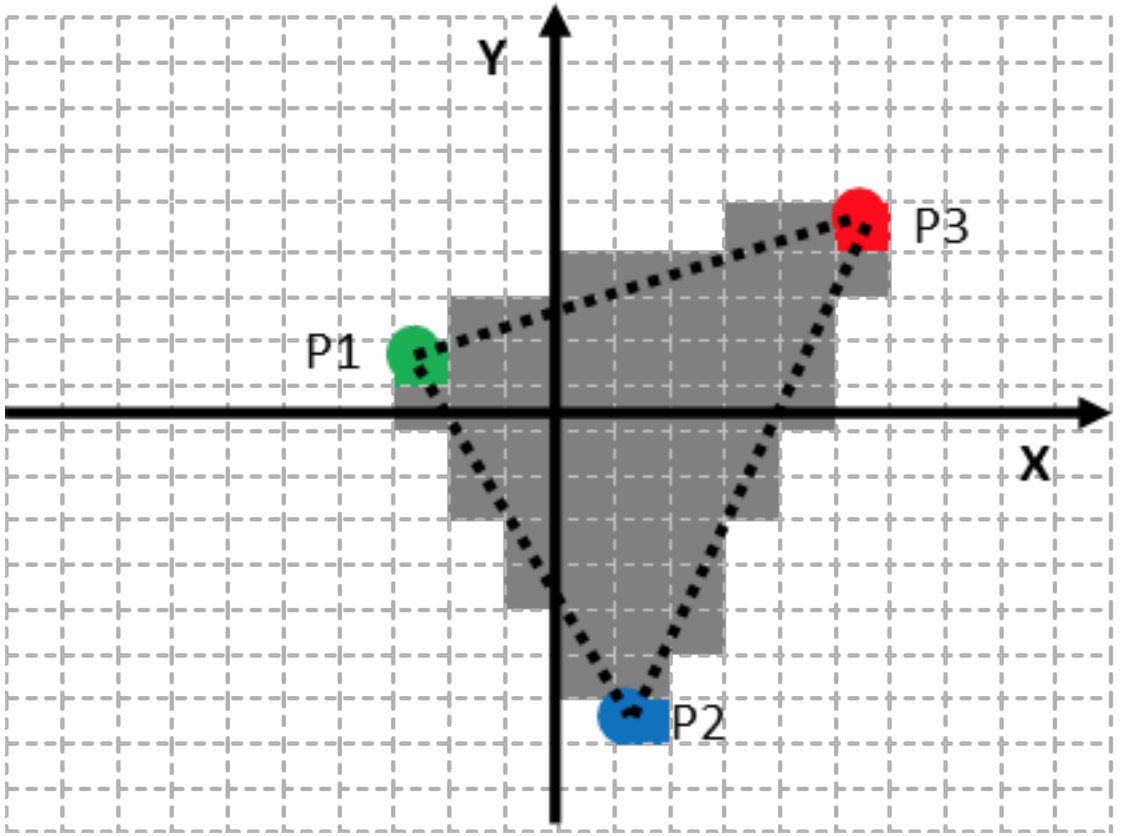
Why aren’t triangles good enough?

Monitors have pixels, not triangles

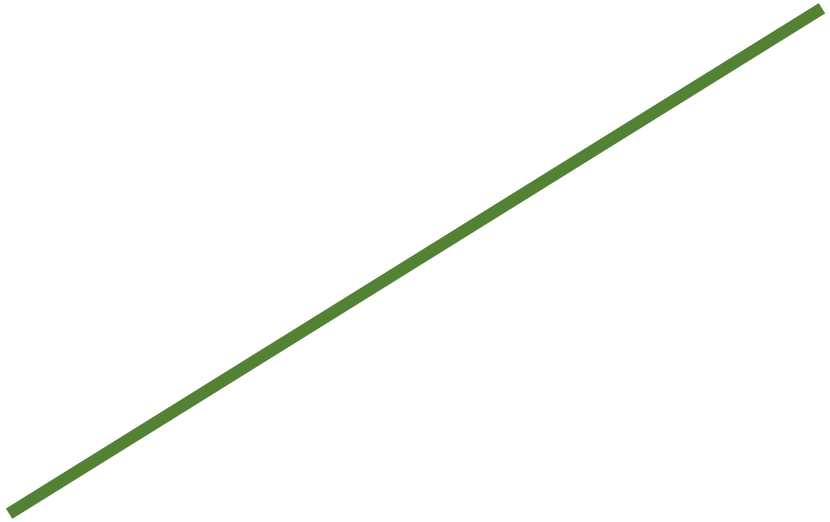


Rasterization

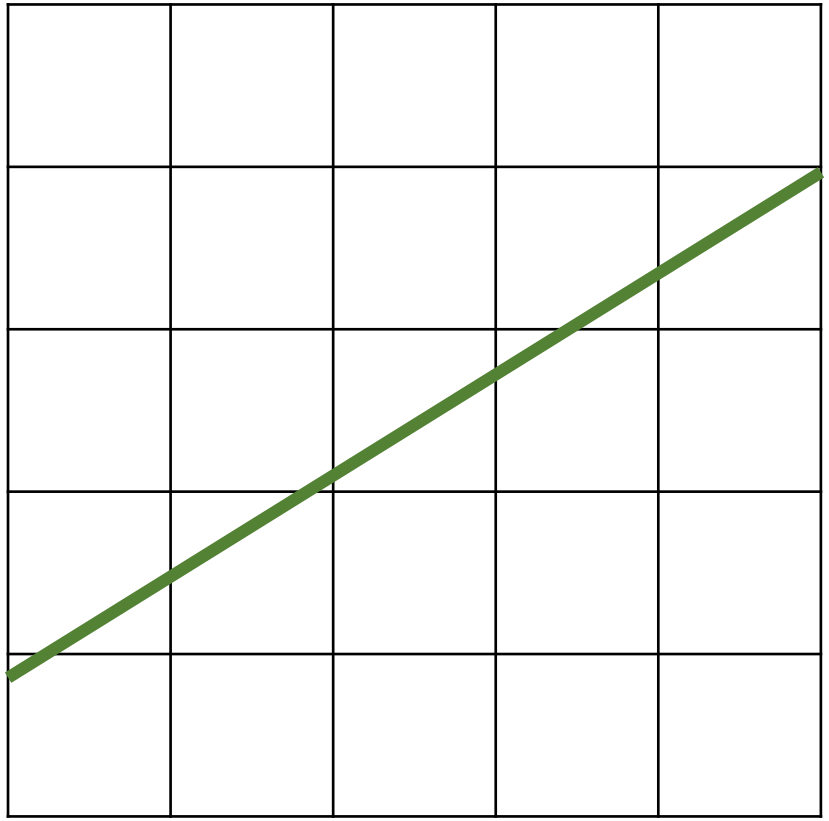
Map space covered by triangles to pixels



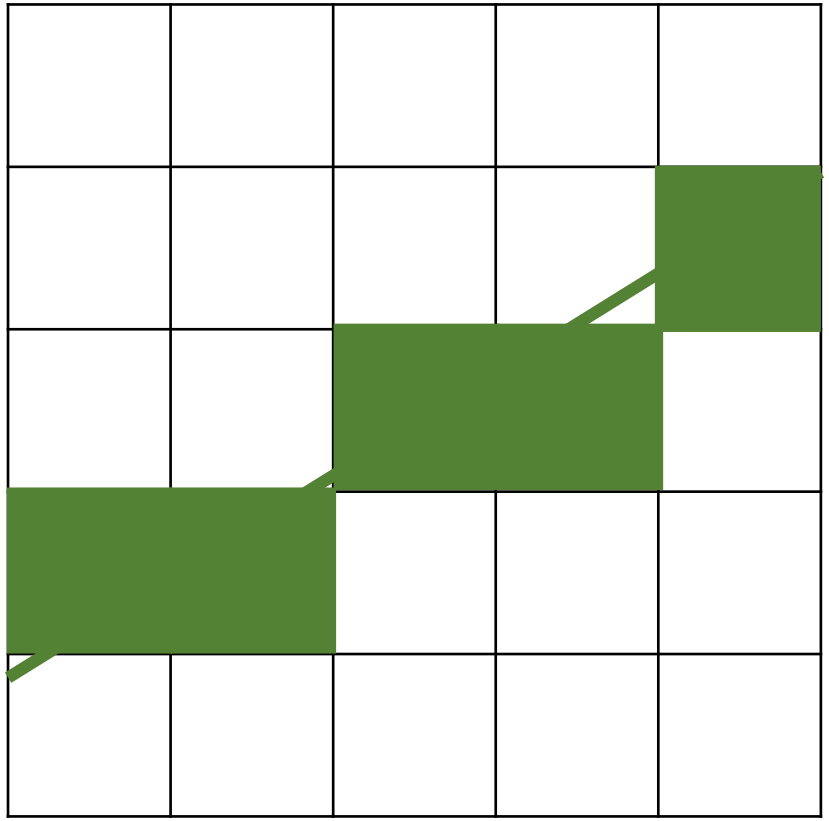
Ways of Drawing Lines



Ways of Drawing Lines



Ways of Drawing Lines



“Jaggies” (yes, this is a pseudotechnical term)

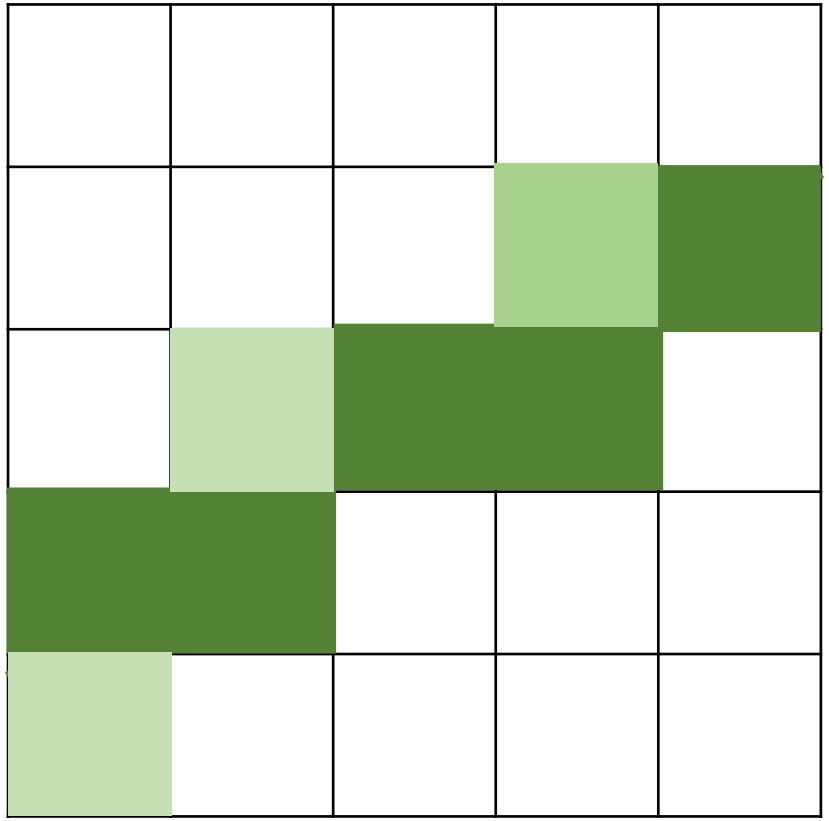
Aliasing occurs when there aren’t enough pixels to accurately rasterize a shape

Results in unpleasant jagged edges

How do we handle lines that don’t fill a pixel?

How do we handle lines that don’t fill a pixel?

Antialiasing



Visibility Testing

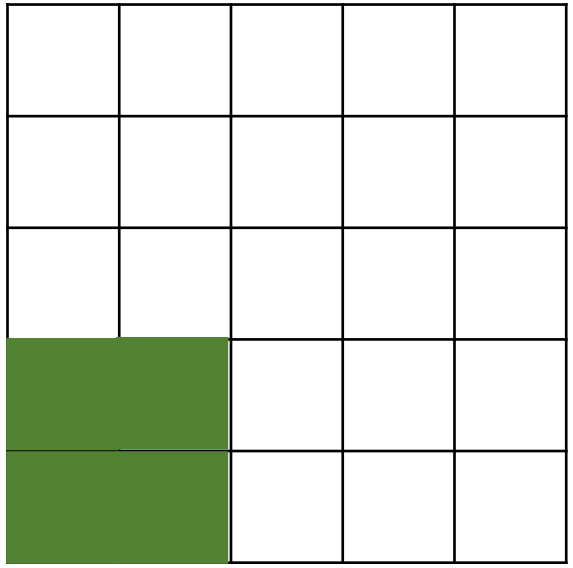
Another place to avoid computing what we can’t see

How might we test visibility?

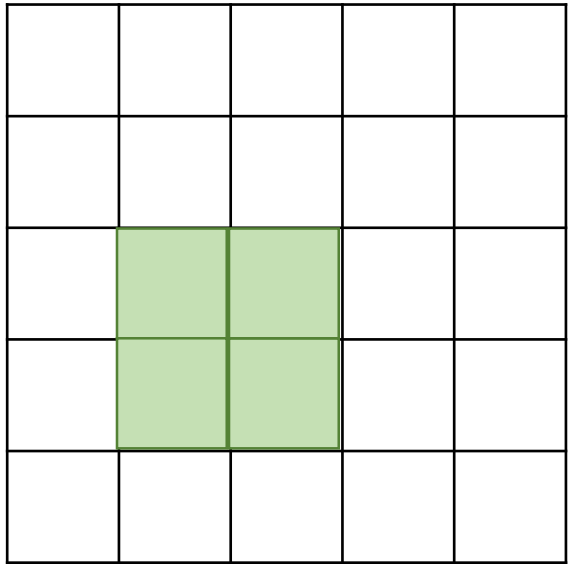
Depth Testing

Painter's Algorithm:

At each pixel, sort all objects intersecting that pixel by depth and draw the color of the pixel that is the closest



Close

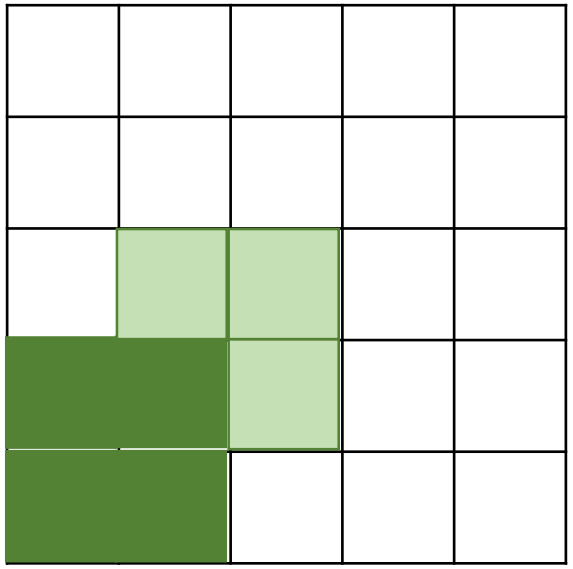


Far

Depth Testing

Painter's Algorithm:

At each pixel, sort all objects intersecting that pixel by depth and draw the color of the pixel that is the closest

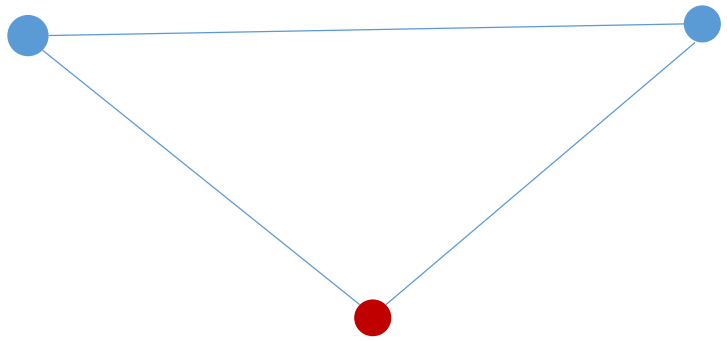


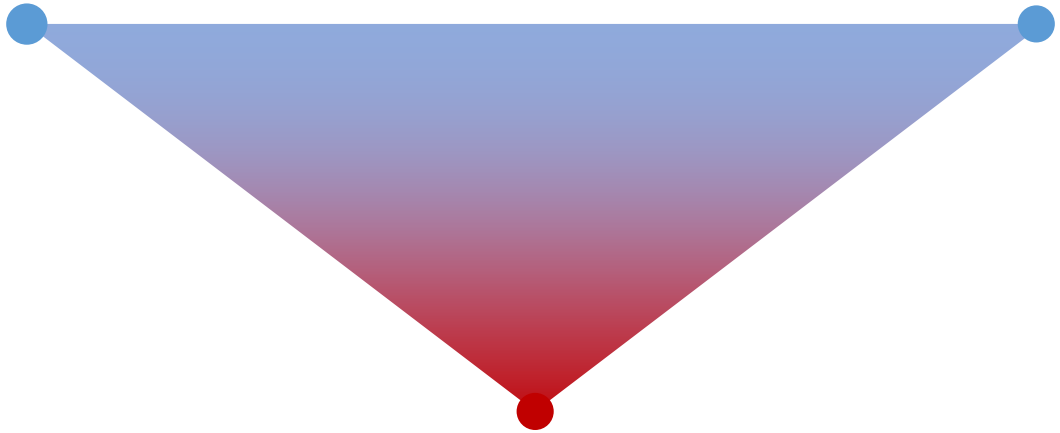
Shaders Revisited

Vertex Shader: Do something at each vertex

Fragment Shader: Do something at each pixel

What happens if there is no fragment shader? What will this triangle look like?





Fragment Shader

WebGL Exercises on Moodle

- Exercise 3: Intro to Shaders #2

GLSL Variable Qualifiers

- const – compile time constant
- attribute – global variables that may change per vertex; read-only, only used in vertex shader
- uniform – global variables that will be the same for both vertex and fragment shader
- varying – used to pass info from the vertex to the fragment shader (e.g., for interpolation); writable in vertex shader, read-only in fragment shader

Common GLSL Data Types:

- int
- float – must specify as float i.e., 1.0 or 0.0, not 1 or 0
- bool
- vec2, vec3, vec4
  - Can create via concatenation, e.g., vec4(someVec3, 4th value)
  - Can multiply by scalars e.g., 5 * someVec3
- mat2, mat3, mat4
- struct

Common GLSL Functions:

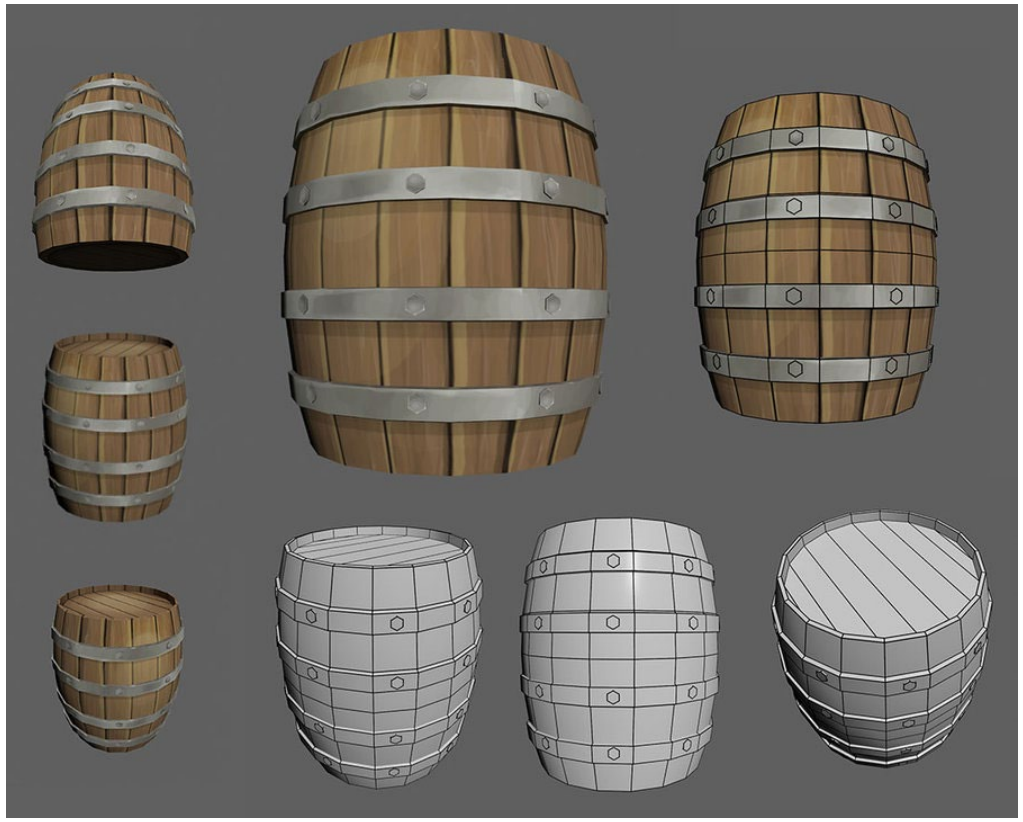
- sin, cos, tan, atan
- pow, exp, log, exp2, log2, sqrt, inversesqrt
- normalize, dot, length, distance, cross, reflect, refract
- mix, clamp

Built-in variables:

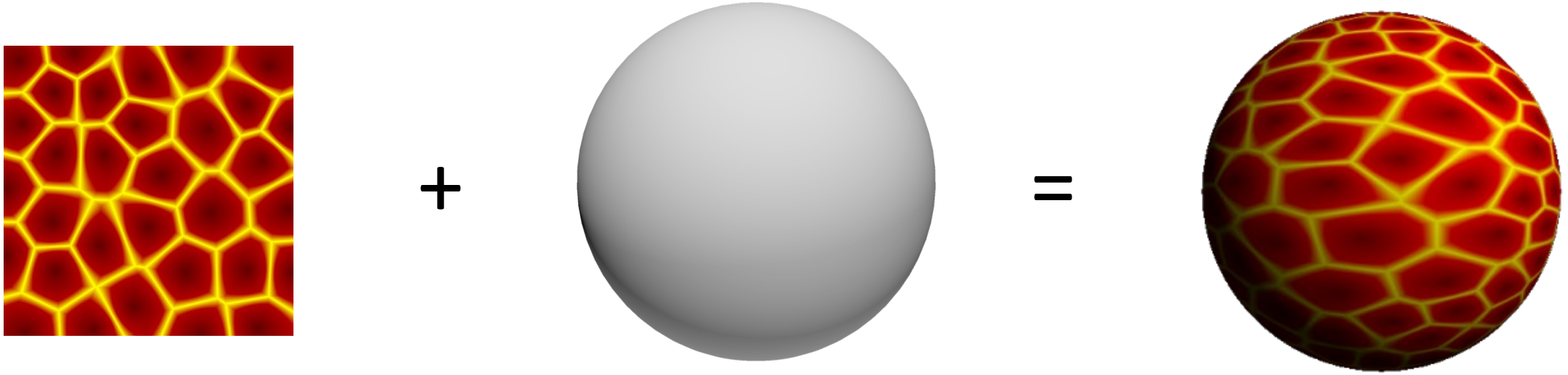
- normal, projectionMatrix, cameraPosition…
- https://threejs.org/docs/#api/en/renderers/webgl/WebGLProgram
- See also: https://stackoverflow.com/questions/15663859/threejs-predefined-shader-attributes-uniforms

Texture Mapping

Layering complex materials on a geometry using shaders

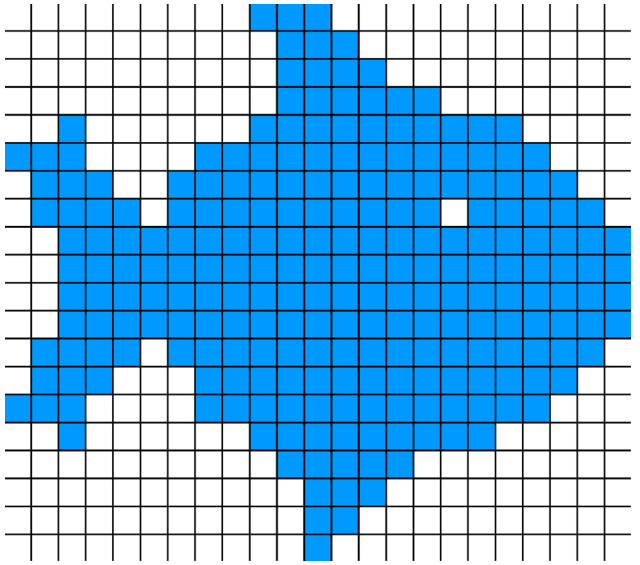


Texture Mapping

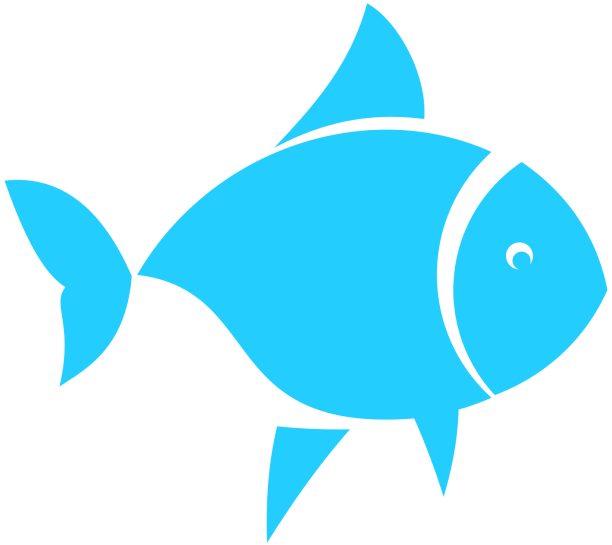


An Aside:
Raster v. Vector Images

Raster: Pixel-based



Vector: Math-based

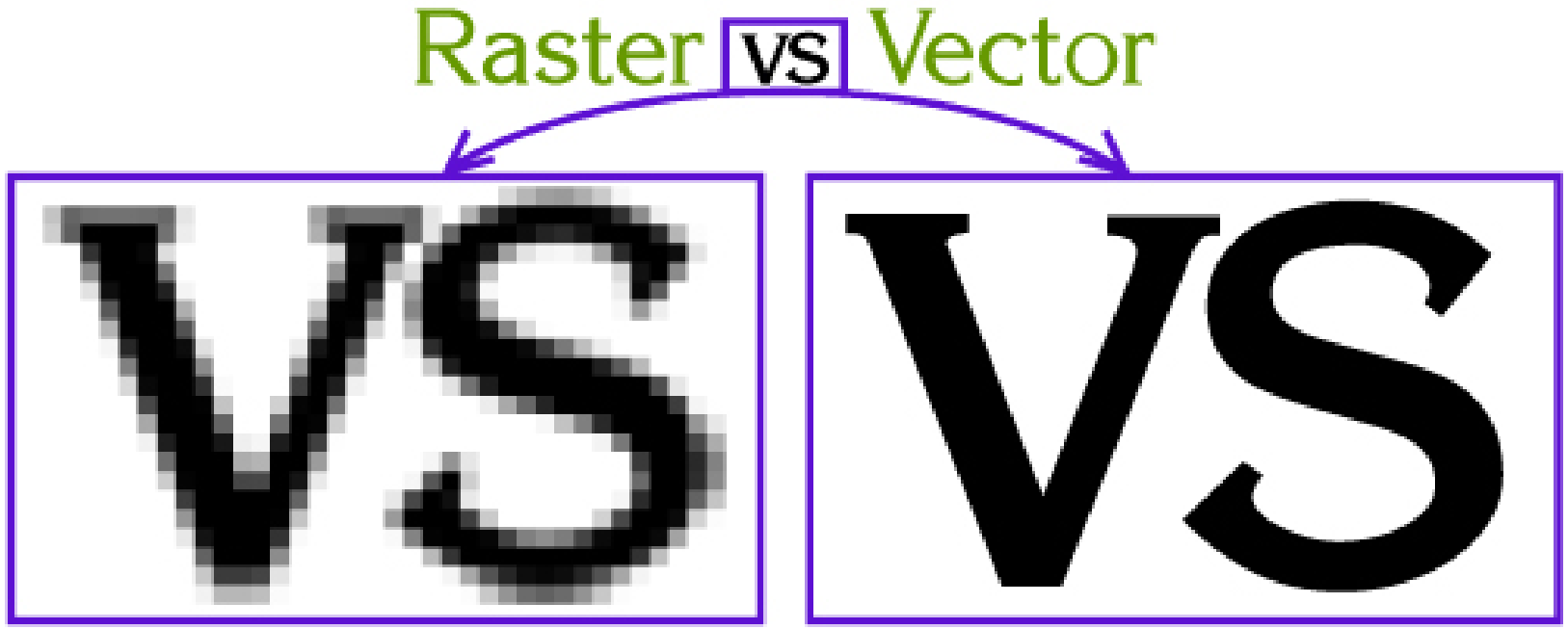


When might we want to use raster images?

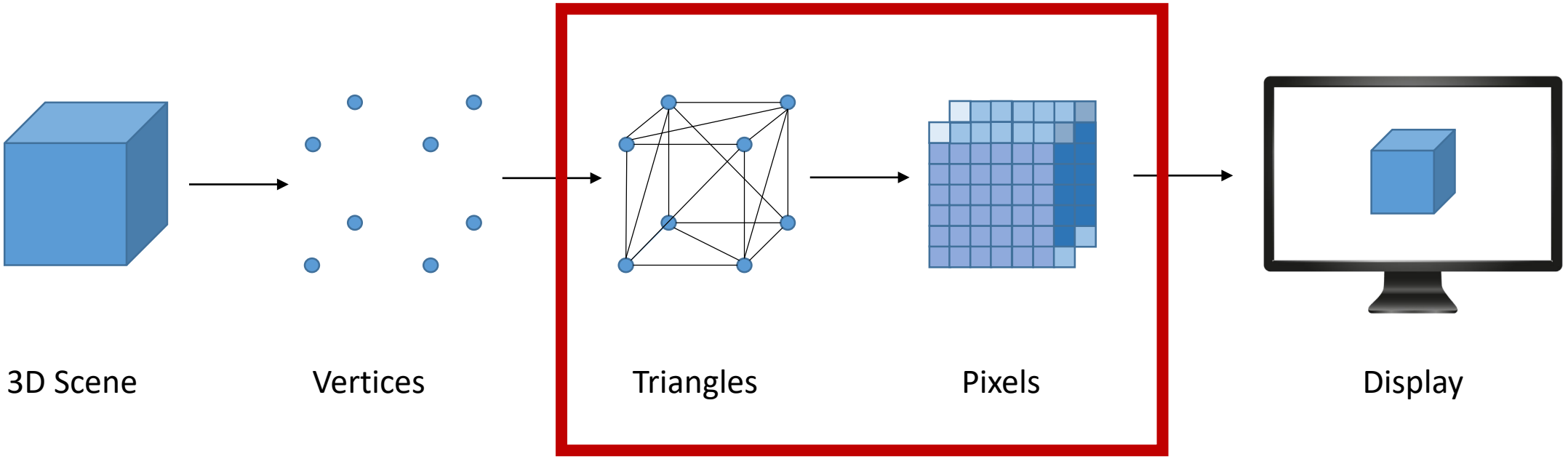
raster

vector

When might we want to use vector images?


Raster vs Vector
VS
VS

The Pragmatic Version



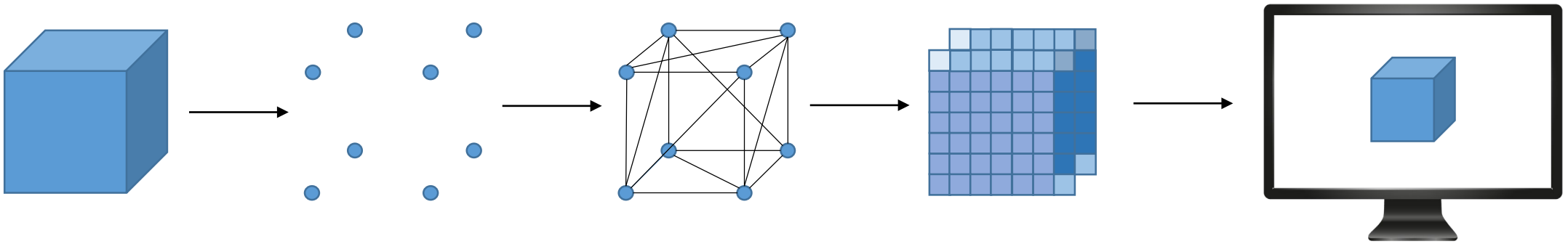
Why don’t we see pixels being drawn?

Two framebuffers:

Front Buffer holds what we see on the screen

Back Buffer is what is actively being rendered

The Pragmatic Version



3D Scene Vertices Triangles Pixels Display

Three.js and the Graphics Pipeline

The Renderer is a beautiful thing…

Three.js hides **as much** or **as little** of this process from you as you'd like

Shaders, lighting, normals, meshes, etc. are handled in many Three.js objects

Any Questions?

If you're interested in more of what we've talked about this week, check out CSCI 5229: Computer Graphics

Check out final WebGL Exercises on Moodle:

- Exercise 4: Animating a shader
- Exercise 5: Basic lighting calculations with shaders
- Exercise 6: Vertex displacement with shaders

We will come back to shaders and talk about how to implement them within Unity

University of Colorado
Boulder

THANKS!

Professor **Dan Szafir**

Computer Science & ATLAS Institute
University of Colorado Boulder