CS 418: Interactive Computer Graphics

Introduction to WebGL

Eric Shaffer

Slides adapted from
Professor John Hart's CS 418 Slides

Some slides adapted from
Angel and Shreiner: Interactive Computer Graphics 7E

Image Formation Revisited

- Most low-level graphics libraries use a synthetic camera model
- Application Programmer Interface (API)
  - Requires user to specify
    - Objects in the scene
    - Materials the objects are made of
    - Viewer (position, view direction, field of view,…)
    - Lights - what parameters do you think typically are used?

Traditional Rasterization-based Pipeline

- Process objects one at a time in the order they are generated by the application
  - Can consider only local (direct) lighting
- Pipeline architecture

application program

Slide adapted from
Angel and Shreiner: Interactive Computer Graphics 7E © Addison-Wesley 2015

display

- All steps can be implemented in hardware on the graphics card

Vertex Shader

- Vertex processor (shader) typically converts vertex locations from one coordinate system to another
  - Object coordinates
  - Camera (eye) coordinates
  - Screen coordinates

- Change of coordinates equivalent to a matrix transformation
- Vertex processor often also computes vertex colors

Slide adapted from
Angel and Shreiner: Interactive Computer Graphics
7E © Addison-Wesley 2015

Projection

- *Projection* is the process that generates a 2D image of 3D geometry
  - Perspective projections: all projectors meet at the center of projection
    - Requires 3D viewer position with the 3D object position
  - Parallel projection: projectors are parallel, center of projection is replaced by a direction of projection

Slide adapted from
Angel and Shreiner: Interactive Computer Graphics
7E © Addison-Wesley 2015

Primitive Assembly

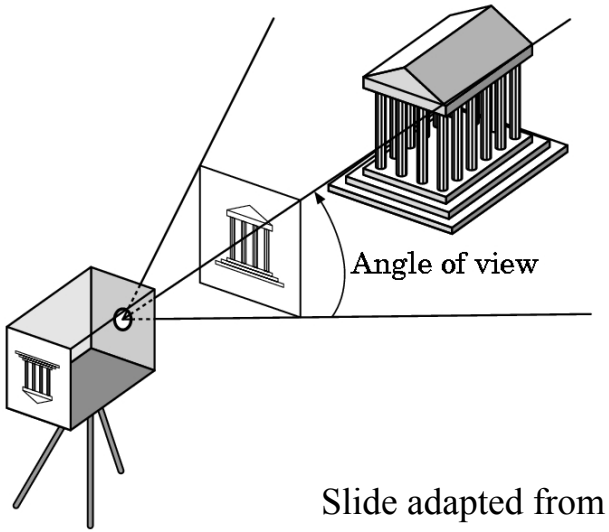
Vertices must be collected into geometric objects before clipping and rasterization can take place

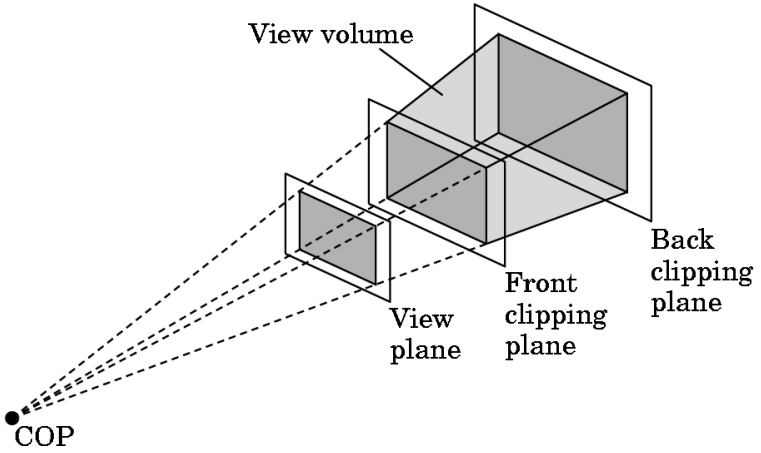
- Line segments
- Polygons
- Curves and surfaces

Slide adapted from
Angel and Shreiner: Interactive Computer Graphics
7E © Addison-Wesley 2015

Clipping

- Just as a real camera cannot "see" the whole world, the virtual camera can only see part of the world or object space
  - Objects not within this volume are said to be *clipped* out of the scene





Slide adapted from
Angel and Shreiner: Interactive Computer Graphics
7E © Addison-Wesley 2015

Rasterization

- If an object is not clipped out, the appropriate pixels in the frame buffer must be assigned colors
- Rasterizer produces a set of fragments for each object
- Fragments are "potential pixels"
  - Have a location in frame bufffer
  - Color and depth attributes
- Vertex attributes are interpolated over objects by the rasterizer

Slide adapted from
Angel and Shreiner: Interactive Computer Graphics
7E © Addison-Wesley 2015

Fragment Processing

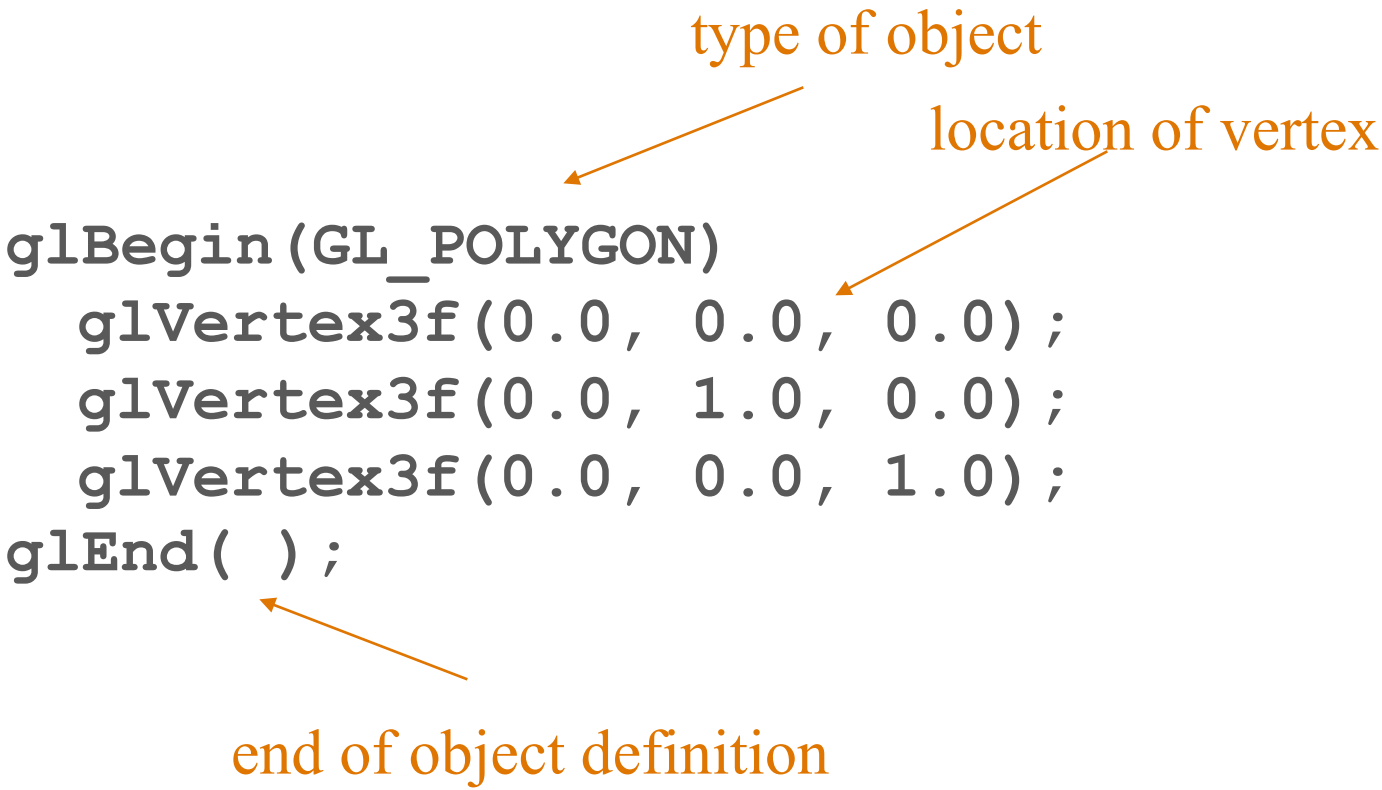
- Fragments are processed to determine the color of the corresponding pixel in the frame buffer
- Colors determined by texture mapping or interpolation of vertex colors
- Fragments may be blocked by other fragments closer to the camera
  - Hidden-surface removal

Slide adapted from
Angel and Shreiner: Interactive Computer Graphics
7E © Addison-Wesley 2015

Geometric Modeling

- Most APIs support a limited set of primitives including
  - Points (0D object)
  - Line segments (1D objects)
  - Polygons (2D objects)
  - Some curves and surfaces
    - Quadrics
    - Parametric polynomials
- In WebGL, you get triangles and lines (and the lines aren't great)
- All are defined through locations in space or *vertices*

Example (old style OpenGL)



Slide adapted from
Angel and Shreiner: Interactive Computer Graphics
7E © Addison-Wesley 2015

Example (new-style OpenGL/WebGL)

- Put geometric data in an array

```
var triangleVertices = [
        0.0,  0.5,  0.0,
       -0.5, -0.5,  0.0,
        0.5, -0.5,  0.0
  ];
```

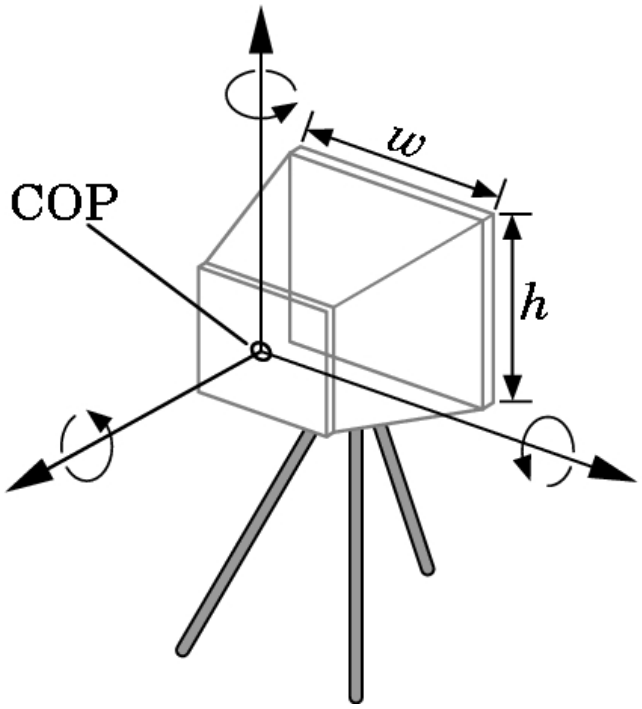
- Send array to GPU
- Tell GPU to render as triangle

Slide adapted from
Angel and Shreiner: Interactive Computer Graphics
7E © Addison-Wesley 2015

Generic Camera Parameters

- Six degrees of freedom
  - Position of center of lens
  - Orientation
- Lens – what are some possible types?
- Film size - what do you think this refers to
- Orientation of film plane

NOTE: Not every API exposes all of these parameters to the programmer



Lights and Materials

- Types of lights
  - Point sources vs distributed sources
  - Spot lights
  - Near and far sources – physically, what is the major difference?
  - Color properties
  - WebGL supports ambient, directional, and point lights
- Material properties
  - Absorption: color properties
  - Scattering
    - Diffuse
    - Specular

WebGL Application Structure

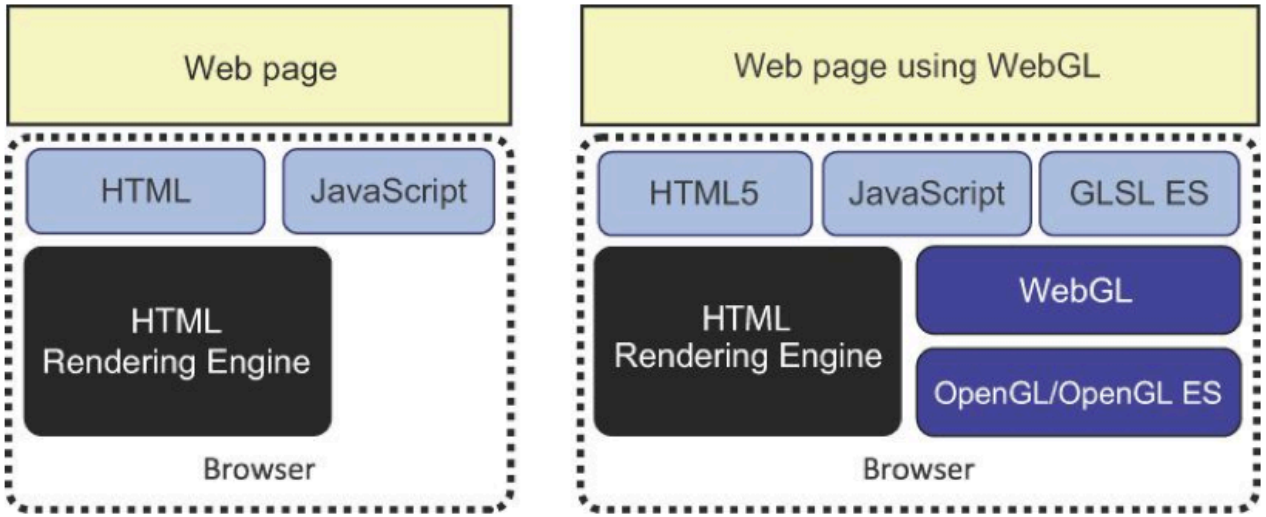


Figure from *WebGL Programming Guide: Interactive 3D Graphics Programming with WebGL* by Matsuda and Lea

Your application will generally just have HTML and JavaScript files

What you will learn…later today…or Friday

- Create a basic but complete WebGL application
- Create a WebGL context
- Write a simple vertex shader and a fragment shader
- Load your shader source code through the WebGL API
- Compile and link your shaders
- Load your vertex data into the WebGL buffers
- Use the buffers to draw your scene

“It used to be easy…” – Edward Angel

```
#include <GL/glut.h>
void mydisplay(){
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_QUAD;
        glVertex2f(-0.5, -0.5);
        glVertex2f(-0,5, 0,5);
        glVertex2f(0.5, 0.5);
        glVertex2f(0.5, -0.5);
    glEnd()
}
int main(int argc, char** argv){
    glutCreateWindow("simple");
    glutDisplayFunc(mydisplay);
    glutMainLoop();
}
```

What happened?

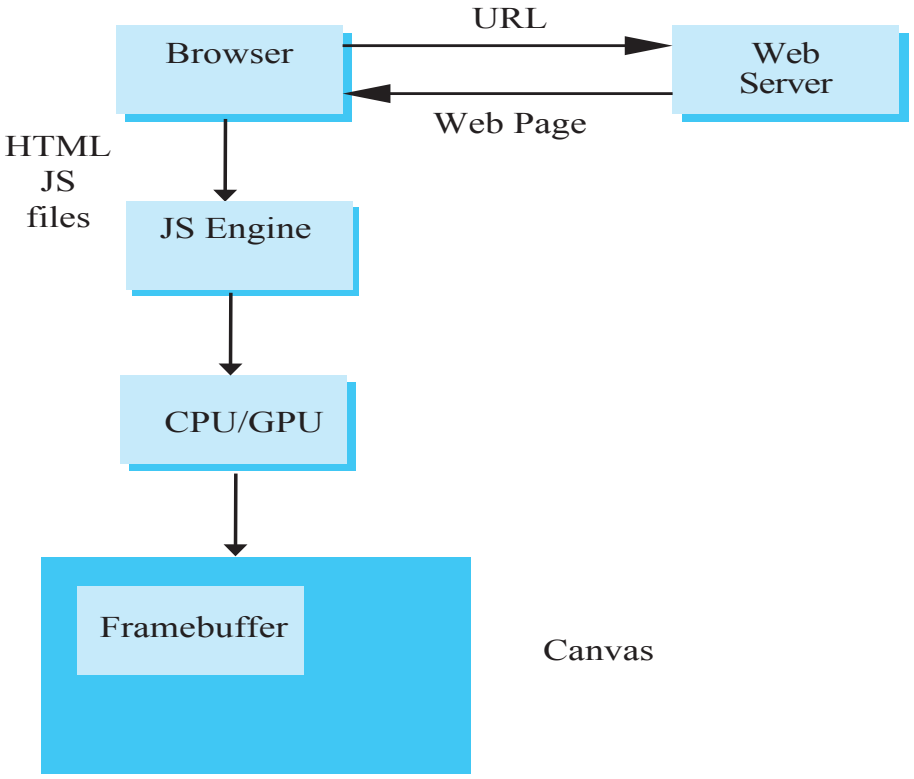
- Most OpenGL functions deprecated
  - immediate vs retained mode

Immediate mode API means the application must call all the rendering commands to draw the entire scene for every frame. For example, WebGL and HTML5 Canvas are immediate mode.

Retained mode API means the application only describes the scene objects but DOES NOT issue rendering requests. For example, SVG is retained mode.

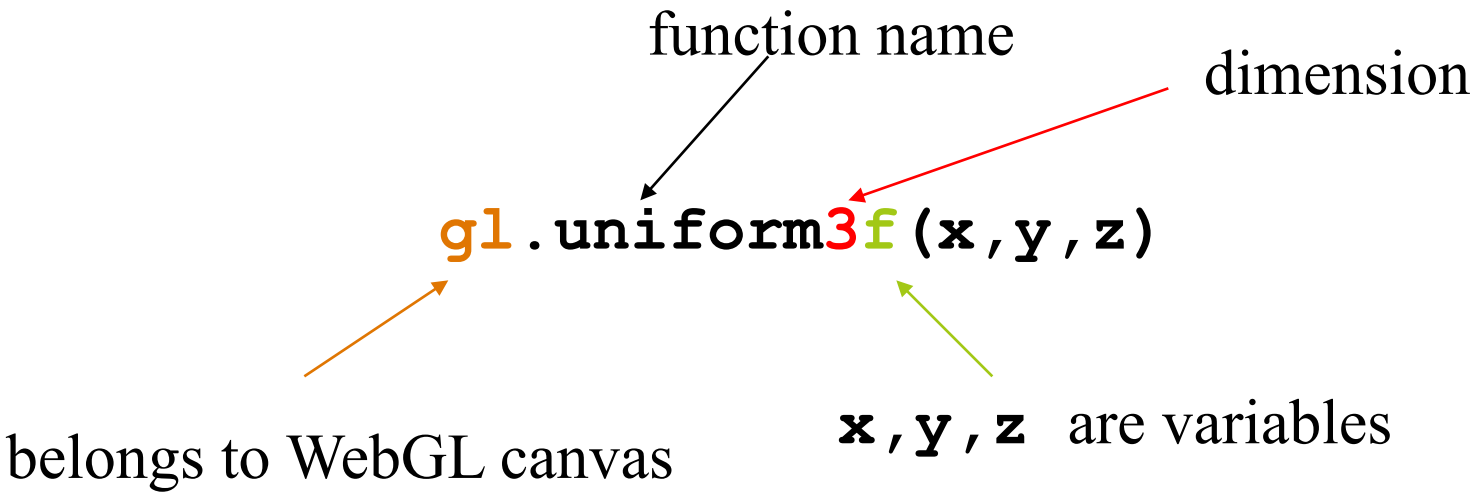
  - move the API closer to the GPU architecture for increased efficiency
  - shader programmability lets you do more
    - e.g. different shading models for non-photorealistic rendering
- The basic ideas behind rasterization-based rendering are the same

Execution in Browser



Slide adapted from
Angel and Shreiner: Interactive Computer Graphics
7E © Addison-Wesley 2015

WebGL function naming conventions




```
gl.uniform3fv(p)
```

p is an array

Slide adapted from
Angel and Shreiner: Interactive Computer Graphics
7E © Addison-Wesley 2015

WebGL constants

- Most constants are defined in the canvas object
  - In desktop OpenGL, they were in #include files such as **gl.h**
- Examples
  - desktop OpenGL
    - `glEnable(GL_DEPTH_TEST);`
  - WebGL
    - `gl.enable(gl.DEPTH_TEST)`
  - `gl.clear(gl.COLOR_BUFFER_BIT)`

WebGL and GLSL

- WebGL requires shaders
- Most state variables, attributes and related pre 3.1 OpenGL functions have been deprecated
- Action happens in shaders
- Job of application is to get data to GPU

GLSL

- OpenGL Shading Language
- C-like with
  - Matrix and vector types (2, 3, 4 dimensional)
  - Overloaded operators
  - C++ like constructors
- Similar to NVIDIA's Cg and Microsoft HLSL
- Code sent to shaders as source code
- WebGL functions compile, link and get information to shaders

Slide adapted from
Angel and Shreiner: Interactive Computer Graphics
7E © Addison-Wesley 2015

Shaders

- Shader source code will be in the HTML file or a JS file…usually
- At a minimum shaders must set the two required built-in variables
  - gl_Position
  - gl_FragColor
- Vertex Shaders generally move vertices around
  - Projection, animation, etc.
- Fragment Shaders generally determine a fragment color

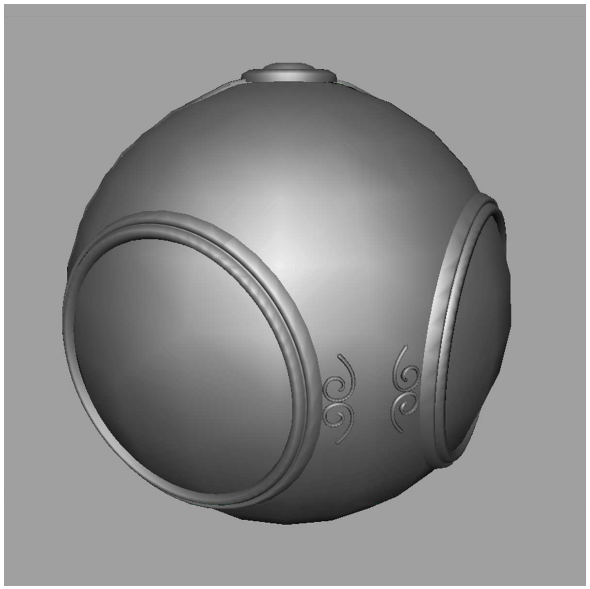
Things Fragment Shaders Can Do

Per fragment lighting calculations

per vertex lighting

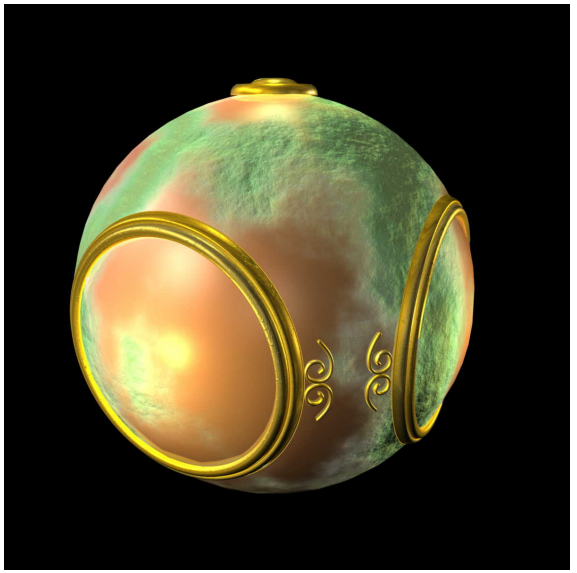
per fragment lighting

Things Fragment Shaders Can Do



smooth shading

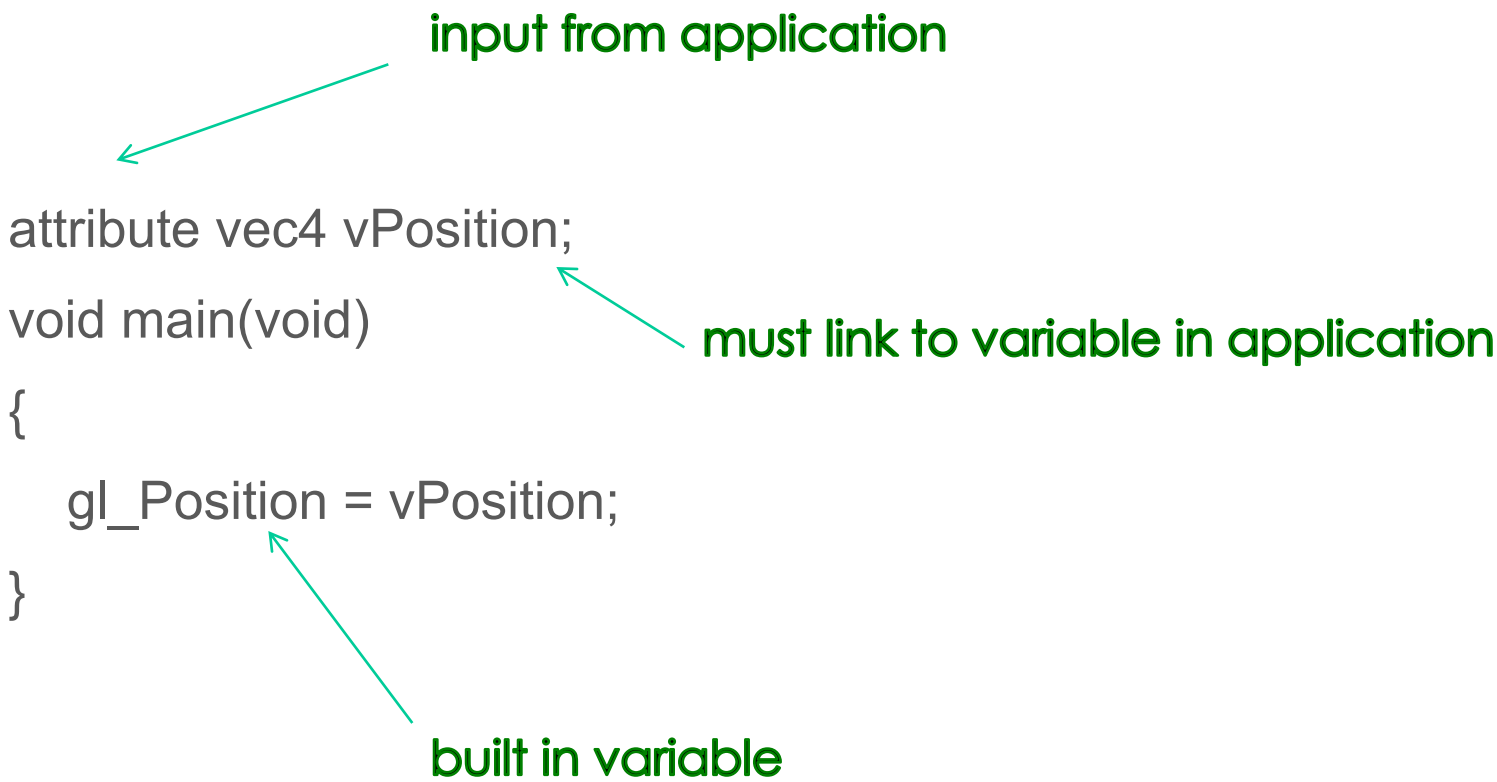
environment mapping



bump mapping

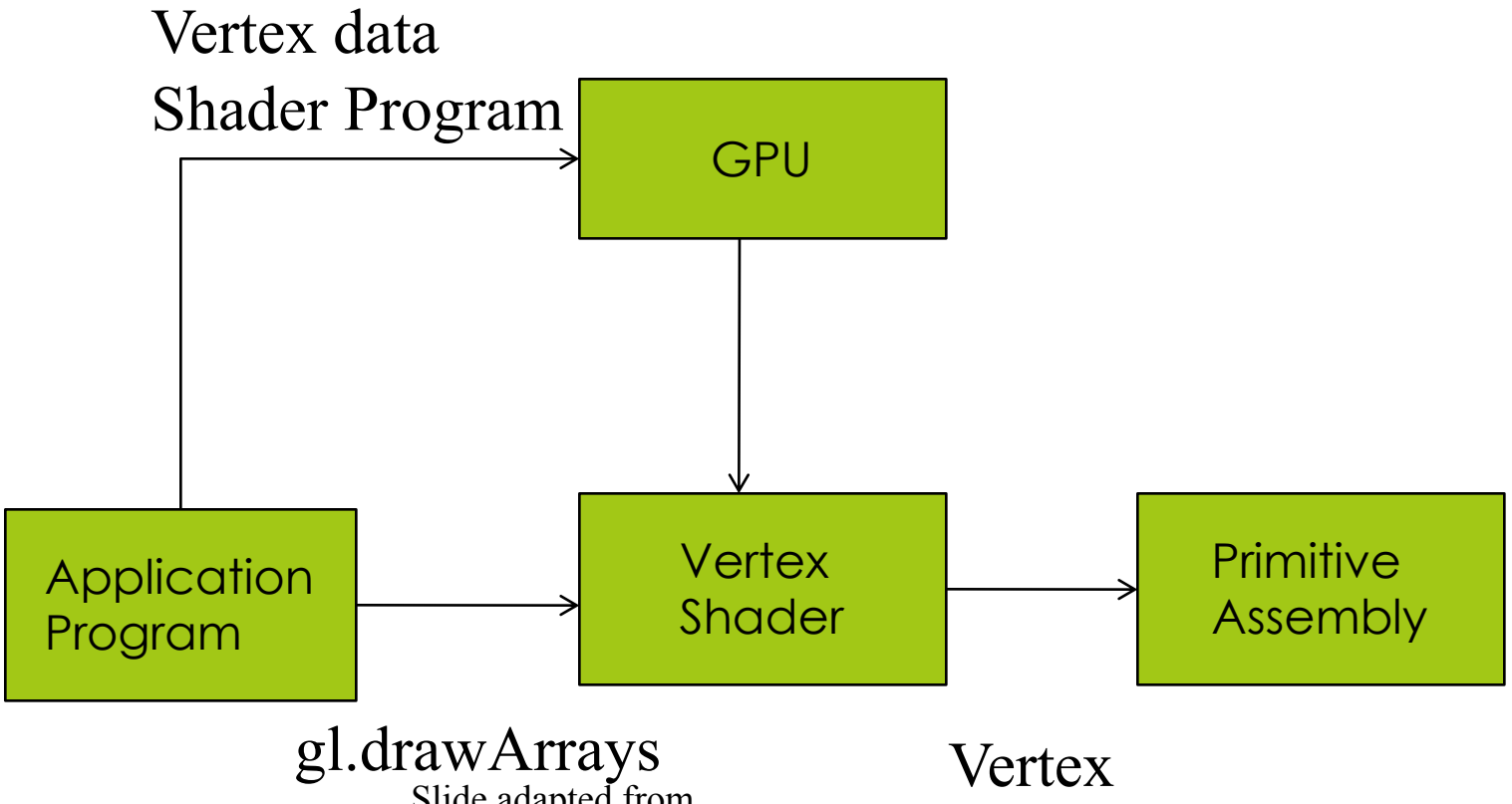
Slide adapted from
Angel and Shreiner: Interactive Computer Graphics
7E © Addison-Wesley 2015

Simple Vertex Shader



Slide adapted from
Angel and Shreiner: Interactive Computer Graphics
7E © Addison-Wesley 2015

Execution Model

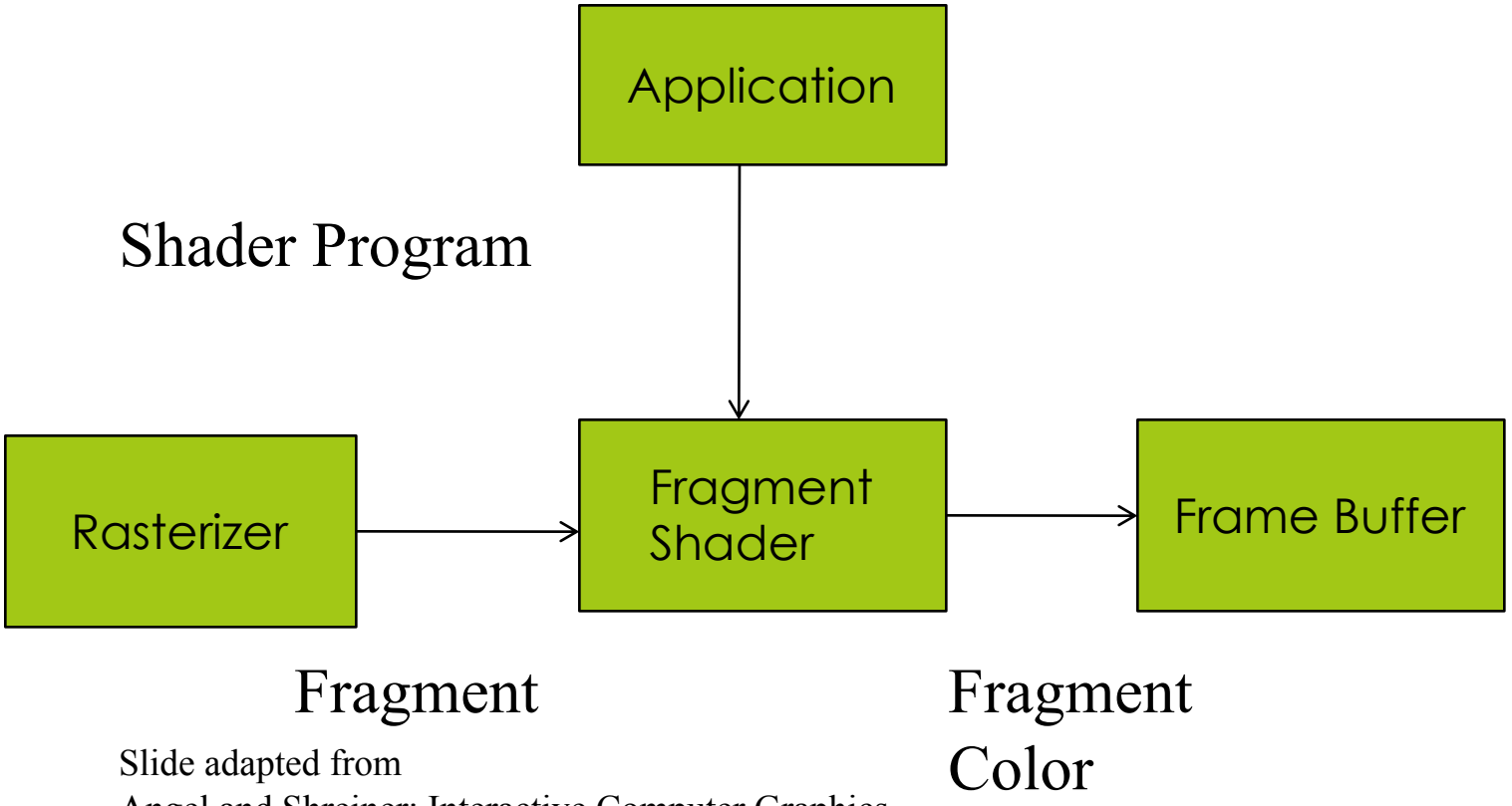


Slide adapted from
Angel and Shreiner: Interactive Computer Graphics
7E © Addison-Wesley 2015

Simple Fragment Program

```
precision mediump float;
void main(void)
{
   gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
}
```

Execution Model



Slide adapted from
Angel and Shreiner: Interactive Computer Graphics
7E © Addison-Wesley 2015

WebGL Coordinate Systems

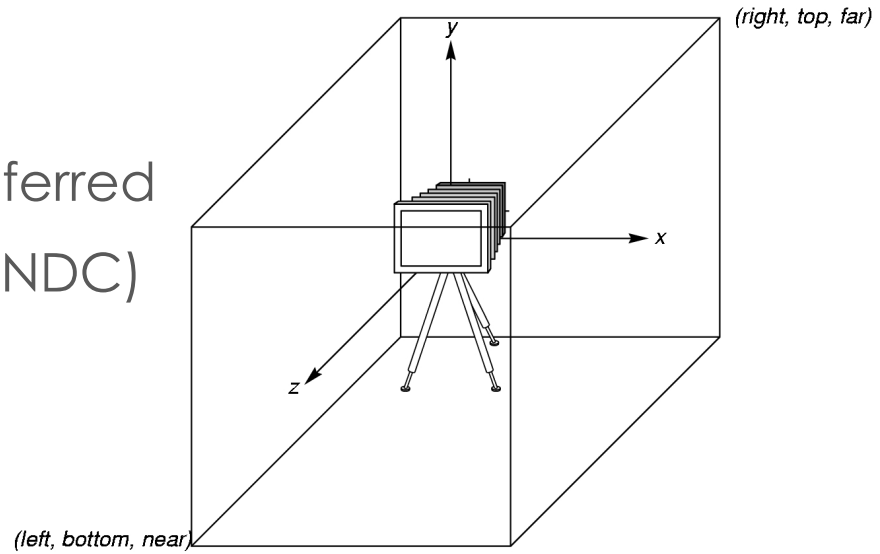
- The units in are determined by the application and are called *object* or *world coordinates*
  - Viewing specifications usually are also in world coordinates
- In WebGL, vertices leave the vertex shader in *clip coordinates*
- Eventually pixels will be produced in *window coordinates*

WebGL Camera

- WebGL places a camera at the origin in object space pointing in the negative z direction

- The default viewing volume is a box centered at the origin with sides of length 2

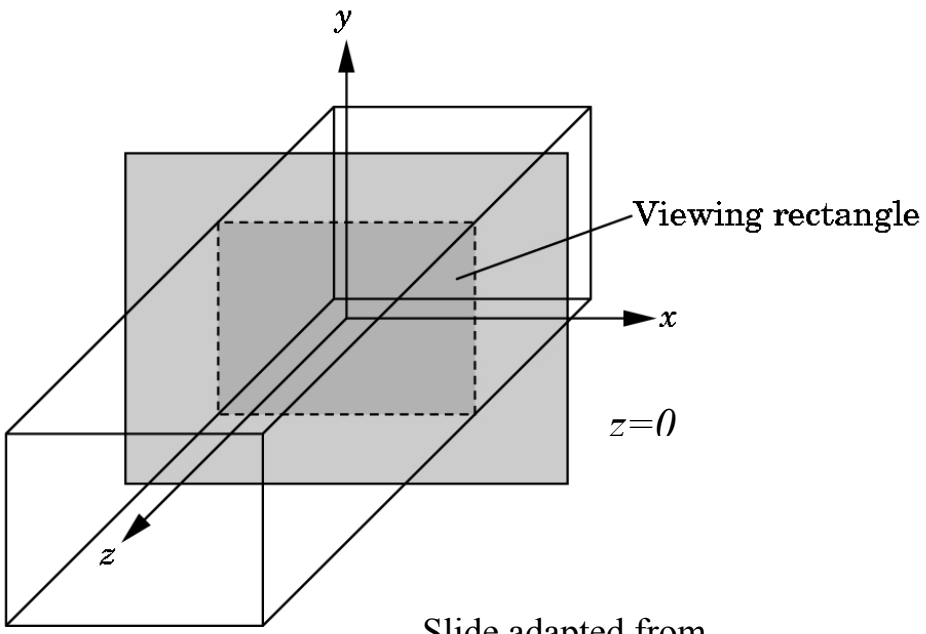
This coordinate system is sometimes referred to a Normalized Device Coordinates (NDC)

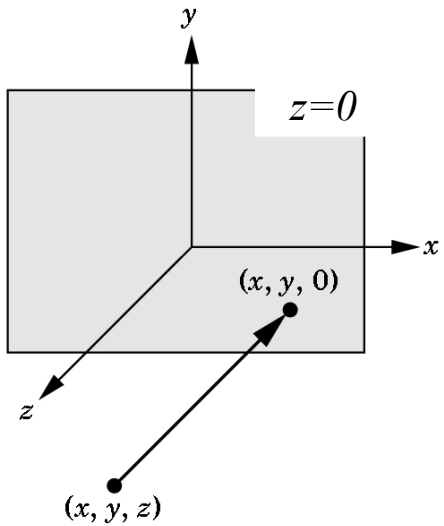


Slide adapted from
Angel and Shreiner: Interactive Computer Graphics
7E © Addison-Wesley 2015

Orthographic Projection

In the default orthographic view, points are projected forward along the z axis onto the plane *z=0*

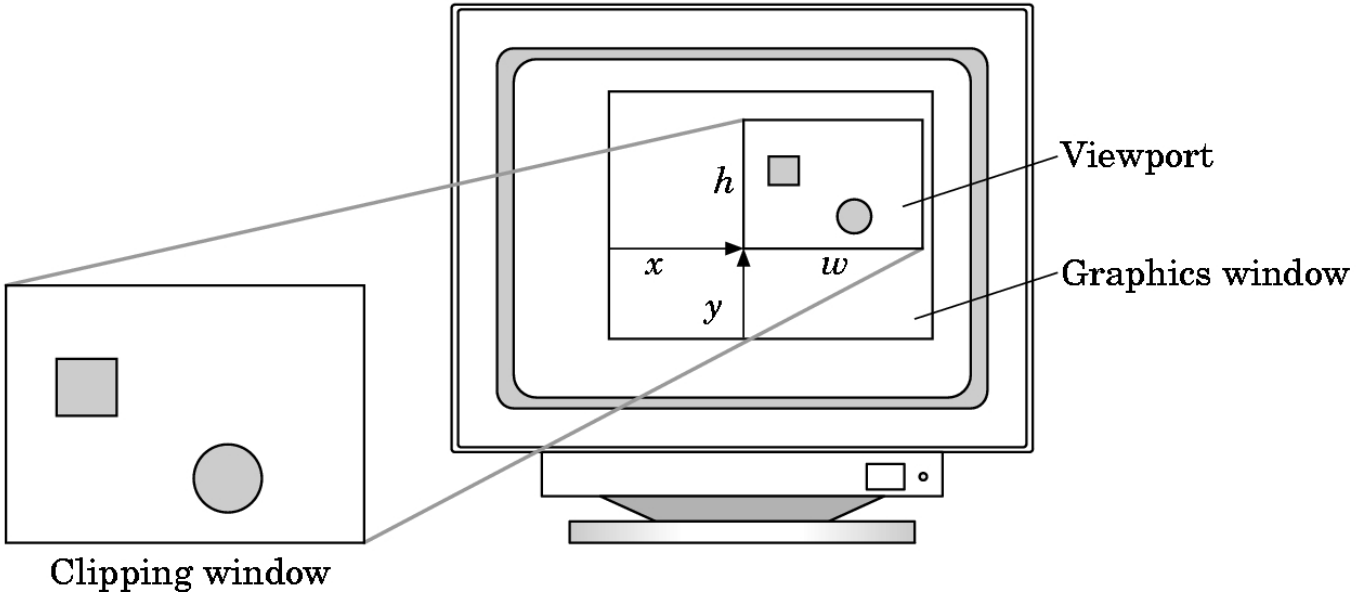




Slide adapted from
Angel and Shreiner: Interactive Computer Graphics
7E © Addison-Wesley 2015

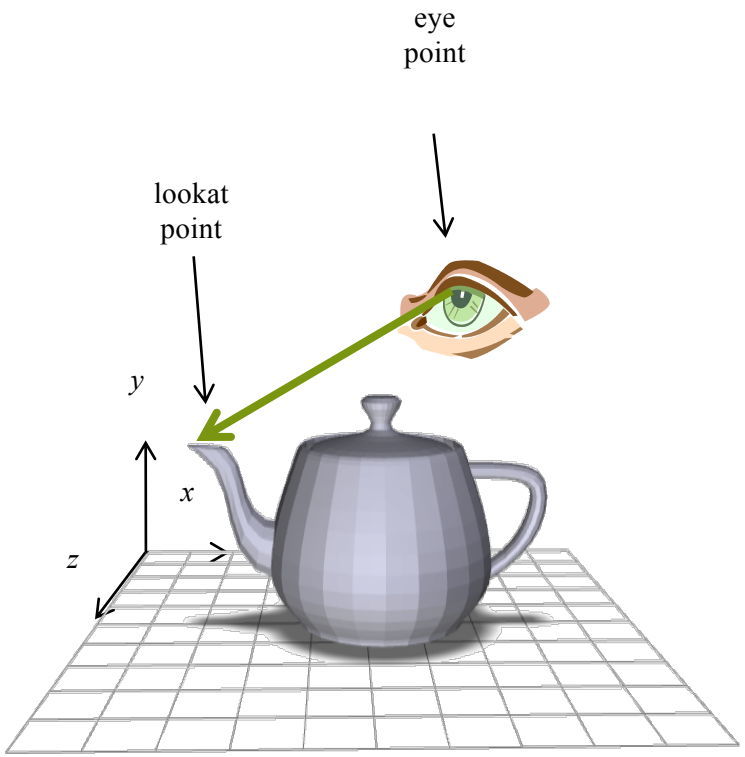
Viewports

- Do not have use the entire window for the image: **gl.viewport(x,y,w,h)**
- Values in pixels (window coordinates)



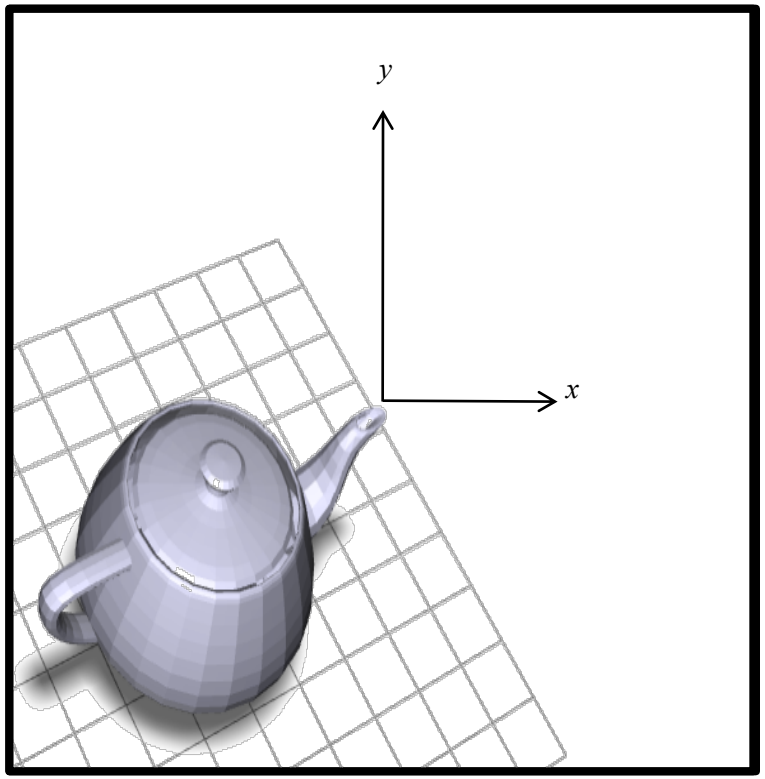
Viewing a 3-D Scene

World Coordinates



Clip Coordinates

(-1,1) (1,1)



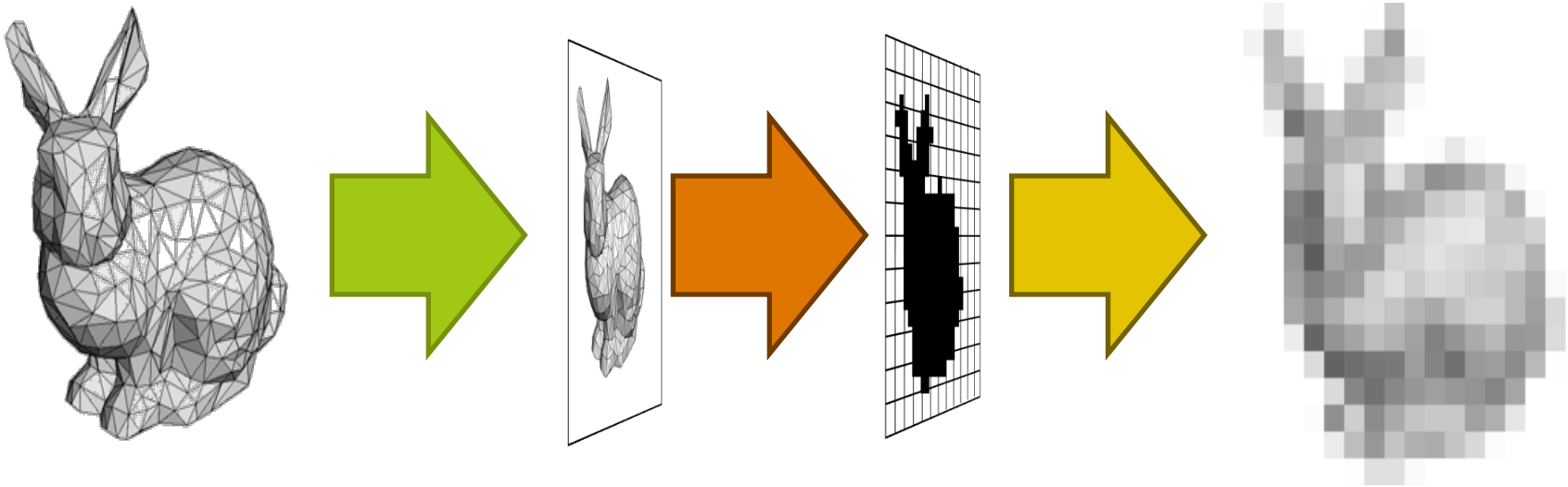
(-1,-1) (1,-1)

3-D Graphics Pipeline

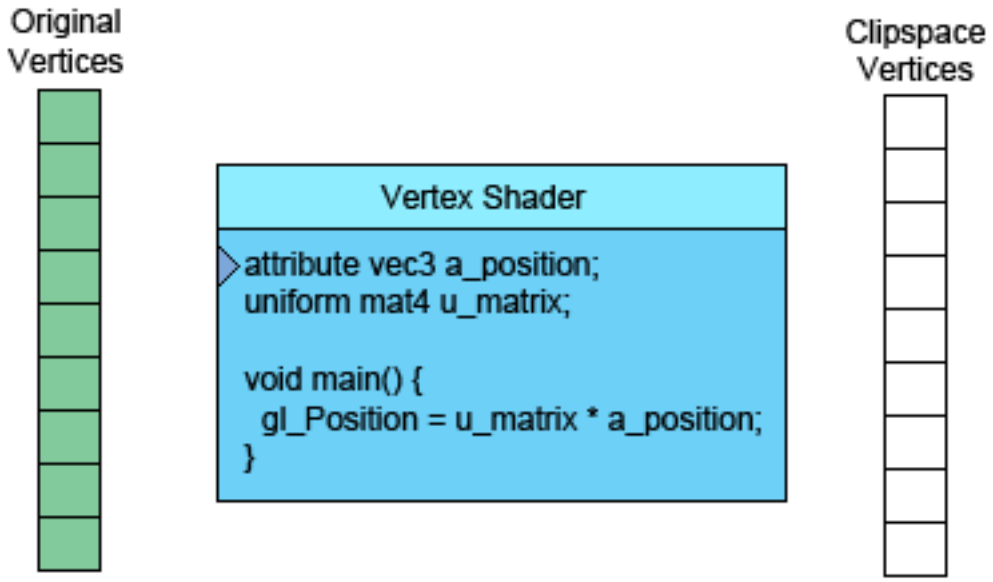
Vertex Processing

Rasterization

Pixel Processing



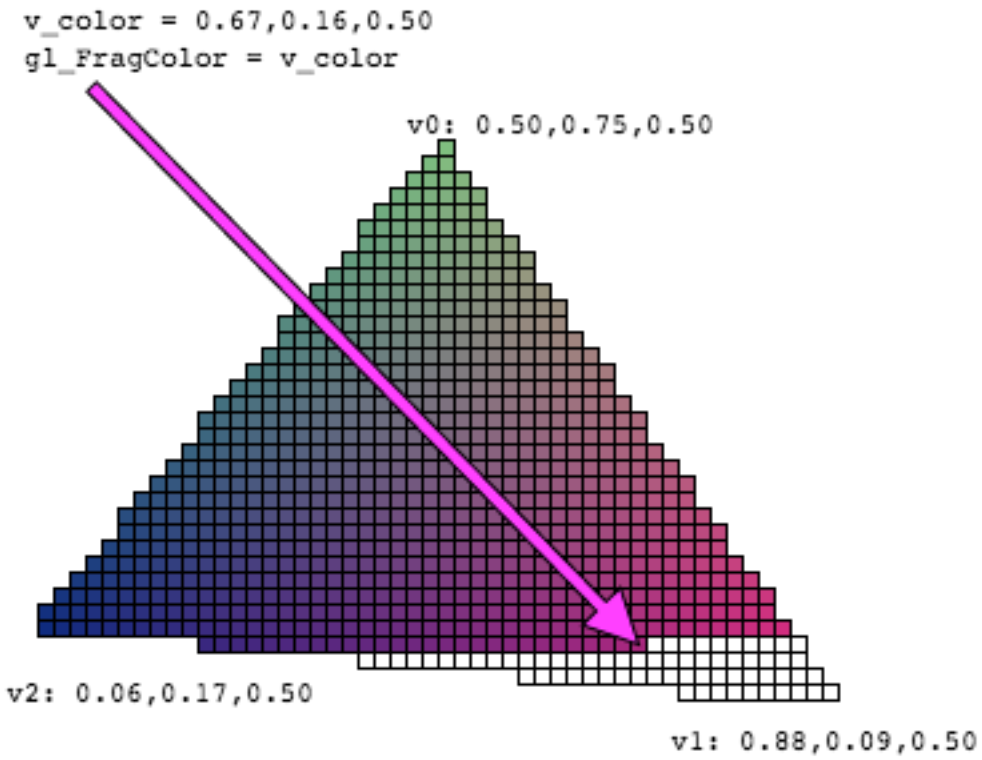
What a Vertex Shader Does…



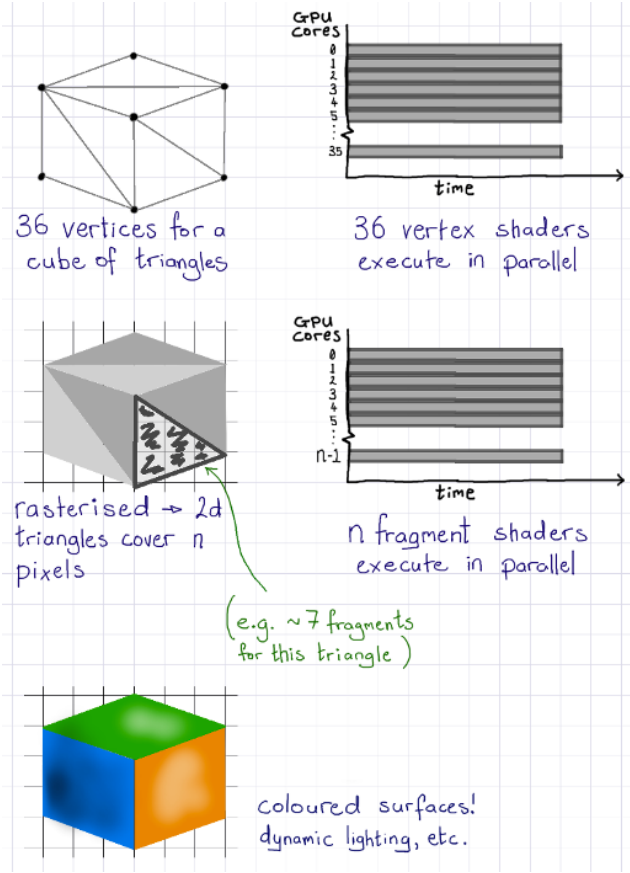
Taken from webglfundamentals.org

What is slightly incorrect about this animation?

What a Fragment Shader Does…



Processing on a GPU



The Graphics Processing Unit (GPU) will have a large number of cores.

This architecture supports a massively-threaded environment for processing vertices and fragments (think of fragments as pixels for now)

Image from http://antongerdelan.net/opengl/shaders.html