CPSC 314
Computer Graphics

Dinesh K. Pai

A first look at the Graphics Pipeline
and WebGL

Many slides courtesy of Min Hyuk Kim, KAIST and Steven Gortler, Harvard

Announcements

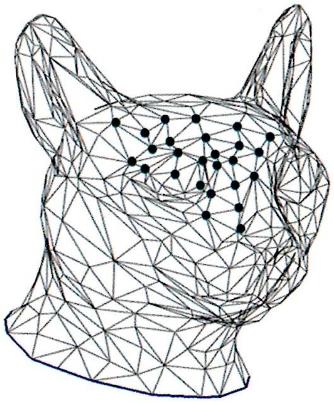
- Today:
  - Introduction to the OpenGL Graphics Pipeline
  - Intro to programming with GLSL, WebGL, Three.js (Assignment 1)
- Assignment 1 out very soon.
  - See <coursepage>/resources.html

What is OpenGL/WebGL?

- OpenGL = Open Graphics Library
  - An open industry-standard API for hardware accelerated graphics drawing
  - Implemented by graphics-card vendors
  - Maintained by the Khronos group
- OpenGL ES = Embedded Systems version of OpenGL with reduced functions
- WebGL 1.0 is based on OpenGL ES 2.0, accessible from JavaScript
- Same underlying graphics architecture

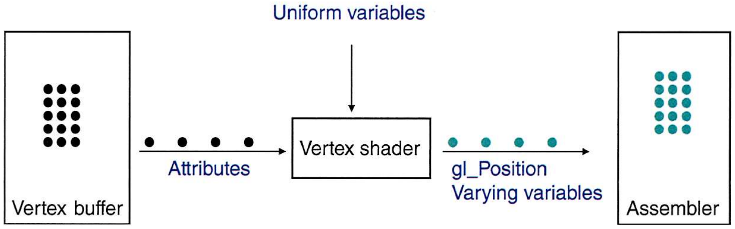
OpenGL Pipeline

- Reference:
  Textbook Chapter 1



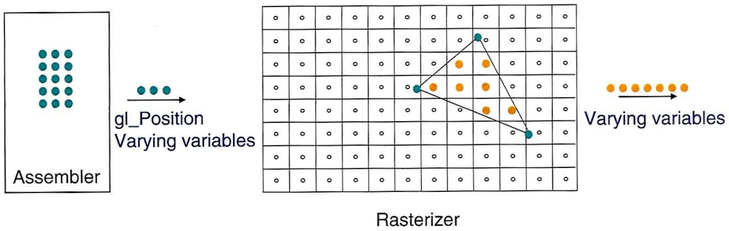
- Shapes are “discretized” into primitives:
  triangles, line segments, …
- We’ll focus on triangles most of the time

OpenGL Pipeline: Vertex Shader



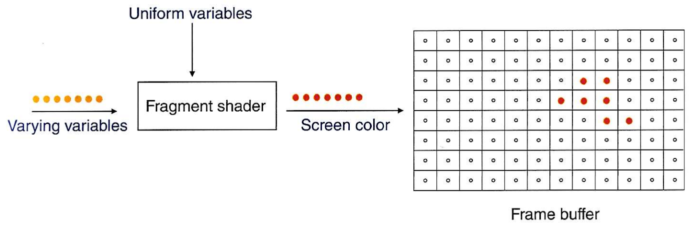
- Vertices are stored in a vertex buffer.
- When a draw call is issued, each of the vertices passes through the vertex shader
- On input to the vertex shader, each vertex (black) has associated attributes.
- On output, each vertex (cyan) has a value for gl_Position and for its varying variables.

OpenGL Pipeline: Rasterization



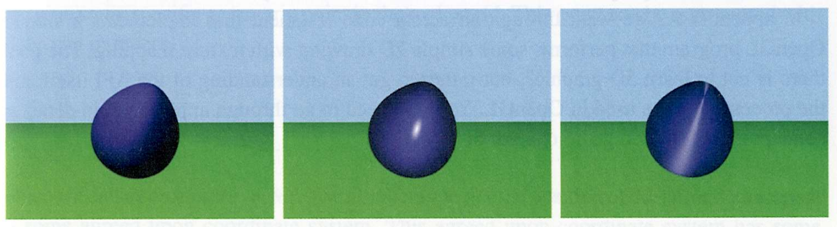
- The data in gl_Position are used to place the three vertices of the triangle on a virtual screen.
- The rasterizer figures out which pixels (orange) are inside the triangle and interpolates the varying variables from the vertices to each of these pixels.

OpenGL Pipeline: Fragment Shader

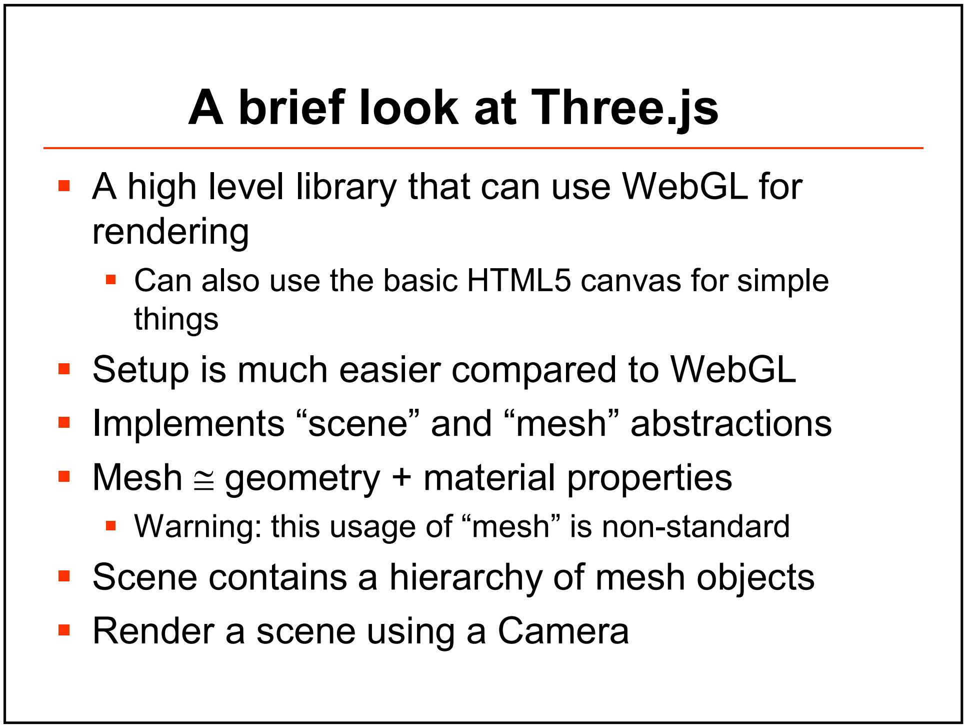


- Each pixel (orange) is passed through the fragment shader, which computes the final color of the pixel (pink).
- The pixel is then placed in the framebuffer for display.

OpenGL Pipeline: Fragment Shader



- By changing the fragment shader, we can simulate light reflecting off of different kinds of materials.



A brief look at Three.js

- A high level library that can use WebGL for rendering
  - Can also use the basic HTML5 canvas for simple things
- Setup is much easier compared to WebGL
- Implements “scene” and “mesh” abstractions
- Mesh ≅ geometry + material properties
  - Warning: this usage of “mesh” is non-standard
- Scene contains a hierarchy of mesh objects
- Render a scene using a Camera

Demo

http://mrdoob.com/projects/htmleditor/

Summary

- What is OpenGL/WebGL?
  - A software interface that allows a programmer to communicate with the graphics hardware
  - A programming interface for rendering 2D and 3D graphics
  - A cross-language multi-platform API for computer graphics
- What is Three.js
  - A high level JavaScript library that provides easy setup and access to WebGL

Important Point!

- In this course we will use WebGL and Three.js to understand the principles of 3D computer graphics
- This is NOT a course about programming with WebGL and Three.js
- Our primary focus will be on writing small shaders in GLSL to implement the key concepts of a computer graphics application

Introduction to Assignment 1

- Switch to demo

How to get started..

- First download assignment template and ensure that it runs in your preferred browser. See

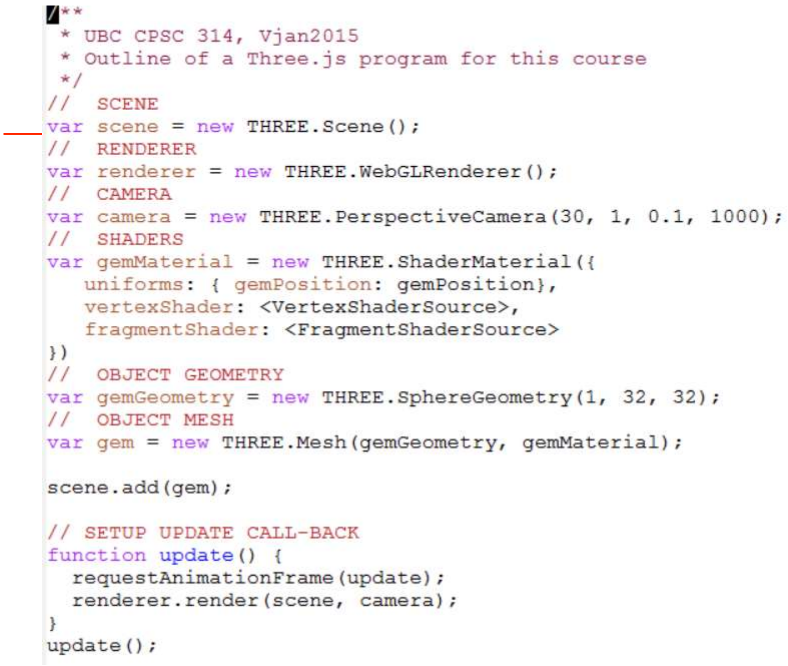
  https://threejs.org/docs/#manual/introduction/How-to-run-thing-locally

  DO THIS ASAP!

- Work on the different parts in sequence. Later parts will need material covered later this week.

The good news

- Even though there are lots of details and options, a few useful things go a long way.
- After initial setup, most of your effort will be on translating graphics concepts into code
- For Assignment 1, this is already setup for you. You mainly have to focus on the vertex shader.

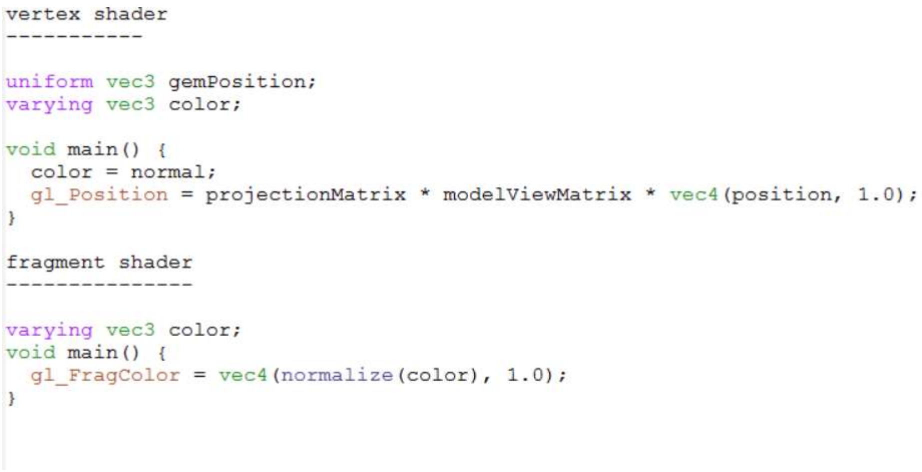


```
/**
 * UBC CPSC 314, Vjan2015
 * Outline of a Three.js program for this course
 */
//  SCENE
var scene = new THREE.Scene();
//  RENDERER
var renderer = new THREE.WebGLRenderer();
//  CAMERA
var camera = new THREE.PerspectiveCamera(30, 1, 0.1, 1000);
//  SHADERS
var gemMaterial = new THREE.ShaderMaterial({
   uniforms: { gemPosition: gemPosition},
   vertexShader: <VertexShaderSource>,
   fragmentShader: <FragmentShaderSource>
})
//  OBJECT GEOMETRY
var gemGeometry = new THREE.SphereGeometry(1, 32, 32);
//  OBJECT MESH
var gem = new THREE.Mesh(gemGeometry, gemMaterial);

scene.add(gem);

// SETUP UPDATE CALL-BACK
function update() {
  requestAnimationFrame(update);
  renderer.render(scene, camera);
}
update();
```

Minimalist shaders



```
vertex shader
-----------

uniform vec3 gemPosition;
varying vec3 color;

void main() {
  color = normal;
  gl_Position = projectionMatrix * modelViewMatrix * vec4(position, 1.0);
}

fragment shader
---------------

varying vec3 color;
void main() {
  gl_FragColor = vec4(normalize(color), 1.0);
}
```

Next class

- Wrap up introduction and Assignment 1 discussion
  - Make sure you’ve read Chapter 1 of textbook
- 3D Math for Graphics
  - Read Chapter 2, Chapter 3 up to 3.5.