CPSC 314
Computer Graphics

Dinesh K. Pai

Pipeline wrap up
Nuts and bolts of graphics programming

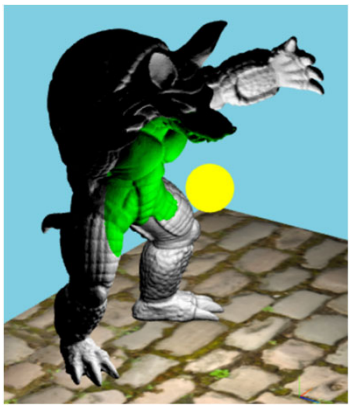
Announcements

- UBC decision: classes online till at least Feb 7
- Labs start next week
  - Attendance is optional. It's an opportunity to meet with a TA in a small group setting and get clarifications and help with course content
  - Will be on Zoom for now
- Today:
  - Assignment 1 will be available tonight
  - Wrap up OpenGL pipeline
  - Programming with THREE.js, WebGL and GLSL

2

Assignment 1 preview

- This term: Hello Armadillo!
  [demo]
- You will fill in parts of a program template provided to you, to create a scene like this.



3

Assignment 1

- First thing: download template from repository and get it running locally on your computer.
- There are lots of details in the template that you can ignore till later in the course. Skim the general structure. Look for comments “HINT” or “YOUR WORK”
- Make small modifications (a few of lines of code) to the shaders, and understand how to pass information from a JavaScript program to the shaders

4

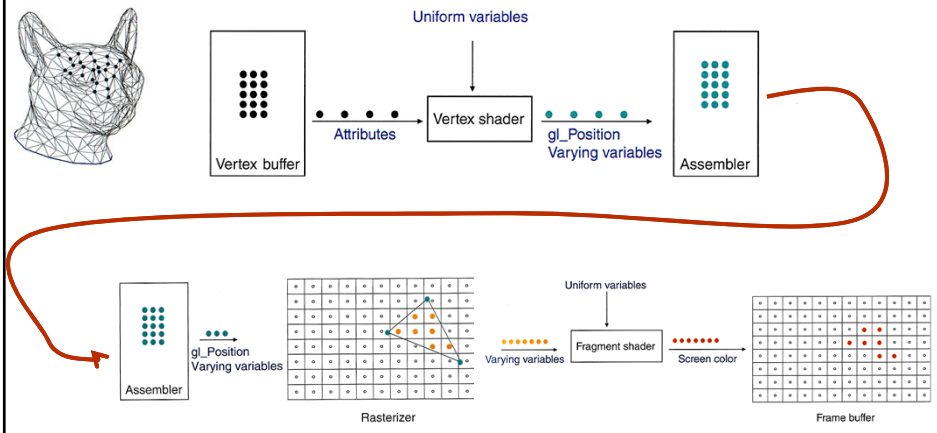
Anatomy of a basic Three.js program

Review helloWorld

5

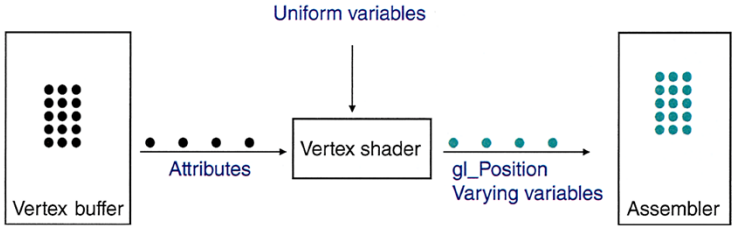
OpenGL pipeline

- Read Textbook Chapter 1



6

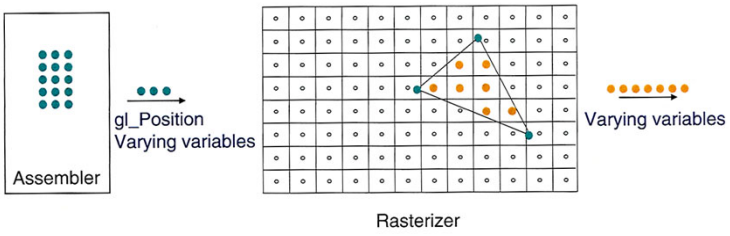
OpenGL Pipeline: Vertex Shader



- 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 (in WebGL 2, called “out/in”).

7

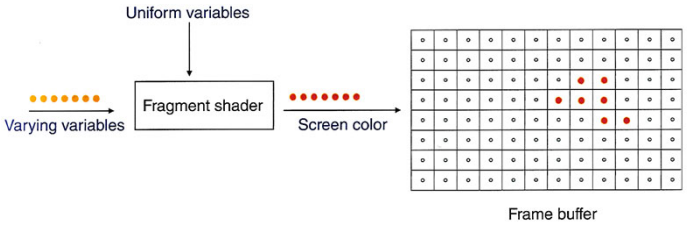
OpenGL Pipeline: Rasterization



- 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.

8

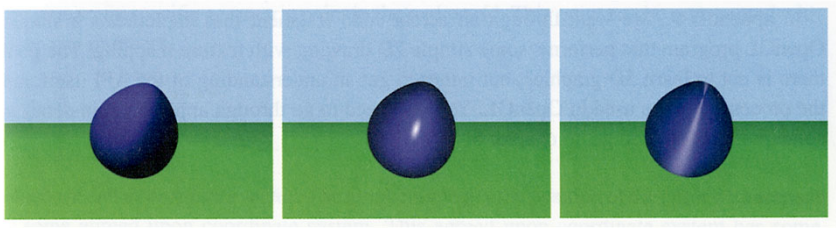
OpenGL Pipeline: Fragment Shader



- 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.

9

OpenGL Pipeline: Fragment Shader



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

10

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

A closer look at GLSL shaders

Handy reference:
https://www.khronos.org/files/webgl20-reference-guide.pdf

Pages 6-8 cover GLSL ES 300

12

GLSL

- OpenGL shading language
- C-like, w. data types and functions useful for graphics
  - vec3, vec4, dvec4, mat4, sampler2D …
    (OpenGL data are floats unless qualified)
  - <matrix-vector multiplication>, reflect, refract
- Used for both vertex shaders and fragment shaders, with small differences
- WebGL 1.0 uses GLSL version 100, compatible with Open GL ES 2.0
- WebGL 2.0 uses GLSL version 300 es
  We use this in our course

13

Summary of Key GLSL Concepts (1)

- 'uniform' type qualifier
  - Same for all vertices
- "in", "out" (WebGL 2) or "varying" (WebGL 1) type qualifiers configure data flow in pipeline
- "in" type qualifiers
  - Input from previous shader stage
  - For vertex shaders, these are per-vertex attributes
- "out" type qualifiers
  - Outputs to next stage
  - gl_position is built-in output variable that must be set before rasterization

14

WebGL 1.0 version

- ‘attribute’ type qualifier: per vertex data
- ‘varying’ type qualifiers: configure data flow in the pipeline.
  - Output of vertex shader, input to fragment shader (after interpolation)

15

Three.js support

- THREE.ShaderMaterial() lets you set shaders, uniforms
- Built-in uniforms and attributes. See https://threejs.org/docs/#api/renderers/webgl/WebGLProgram
- Some vertex attributes
  - position, normal, and uv
- Some uniforms
  - modelView matrix and cameraPosition

16

Hello Shader Material

Review updated helloWorld

17

ShaderMaterial Example

```
var material = new THREE.ShaderMaterial( {

        uniforms: {
                time: { type: "f", value: 1.0 },
                resolution: { type: "v2", value: new THREE.Vector2() }
        },
        attributes: {
                vertexOpacity: { type: 'f', value: [] }
        },
        vertexShader: document.getElementById( 'vertexShader' ).textContent,
        fragmentShader: document.getElementById( 'fragmentShader' ).textContent

} );
```

https://threejs.org/docs/#api/materials/ShaderMaterial

18

Animation (infinite) Loop

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

// Do this last
update();
```

19

Debugging your program

- Debugging GLSL programs can be challenging. Keep calm. Many problems are due to strict typing. E.g., float literals must use decimal point
- Good news: easy to run and see results. No compilation step. Test code as you write it.
- Browsers provide some tools for JavaScript debugging, but not for GLSL programs*
  - Toggle console with, e.g., <F12>
  - Reload page with CTRL-R
  - * The situation is improving rapidly. Firefox now has better GLSL debugging support.

20

Next Class

- Geometry 1: Points and Vectors
- Homework: read Textbook Chapter 2

21