CPSC 314
Computer Graphics

Dinesh K. Pai

Nuts and bolts of

OpenGL programming, Part 2

Vector Spaces

Announcements

- Midterm exams now scheduled:
  - First midterm Friday Feb 7, in class
  - Second midterm Friday March 21, in class
- Assignment 1
  - Please use our README in A1.zip, not textbook's.
  - Mac issues still persist… please be patient. Setting up the environment is main work of this assignment
- Today:
  - Wrap up last class on practical aspects of programming with OpenGL and vertex shaders
  - Continue with graphics math review

C^3 Survey

- What is your computing environment
  a) Linux, with lab machines
  b) Linux, personal
  c) Mac OSX
  d) Windows
  e) Something else

C^3 Survey

- How far along are you with Assignment 1
  a) Not started
  b) Can run template code
  c) Finished at least one required part
  d) Finished all required parts (1,2,3)
  e) Finished everything

Recap

What you need to get started..

- GLUT and freeGLUT
- GLEW
- GLM
- GLSL

GLalphabet soup

- GLUT and freeGLUT
- GLEW
- GLM
- 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>, smoothstep, reflect,…
  - Used for both vertex shaders and fragment shaders, with small differences

Pattern of an OpenGL program

```
int main(int argc, char **argv) {
        initGlutState(argc,argv);
        glewInit(); // load the OpenGL extensions

        initGLState();
        initShaders();
        initBuffers();
        …
        glutMainLoop();
        return 0;
 }
```

Call back function “display"

- Registered with GLUT using glutDisplayFunc(display)

```
static void display(void) {
        glUseProgram(h_program)
        glClear(GL_COLOR_BUFFER_BIT |
                GL_DEPTH_BUFFER_BIT);
        drawObj();
        glutSwapBuffers();
}
```

Vertex Shader from textbook's hw2d example

```
#version 130

uniform float uVertexScale;

in vec2 aPosition;
in vec3 aColor;
in vec2 aTexCoord0, aTexCoord1;

out vec3 vColor;
out vec2 vTexCoord0, vTexCoord1;

void main() {
  gl_Position = vec4(aPosition.x * uVertexScale, aPosition.y, 0,1);
  vColor = aColor;
  vTexCoord0 = aTexCoord0;
  vTexCoord1 = aTexCoord1;
}
```

C^3: GLSL

- What is the mandatory output in a vertex shader?
  a) The clip coordinates (gl_Position)
  b) The color of each vertex (e.g. fragColor in the textbook example)
  c) The texture coordinates
  d) All of the above

OpenGL as a client-server system

- Server is a drawing machine, with state
  - includes data “Objects” and “Context”
- Context is all the state that can be drawn or manipulated by the client
- OpenGL API provides functions for client to change or read the state of the server
  - Create Objects on the server
  - Bind data buffers to targets in the Context
  - glDraw* initiates drawing
- Important things to create on server
  - Data: Vertex Buffer Objects (VBOs), Texture Objects, …
  - Programs: Shader programs

OpenGL pipeline

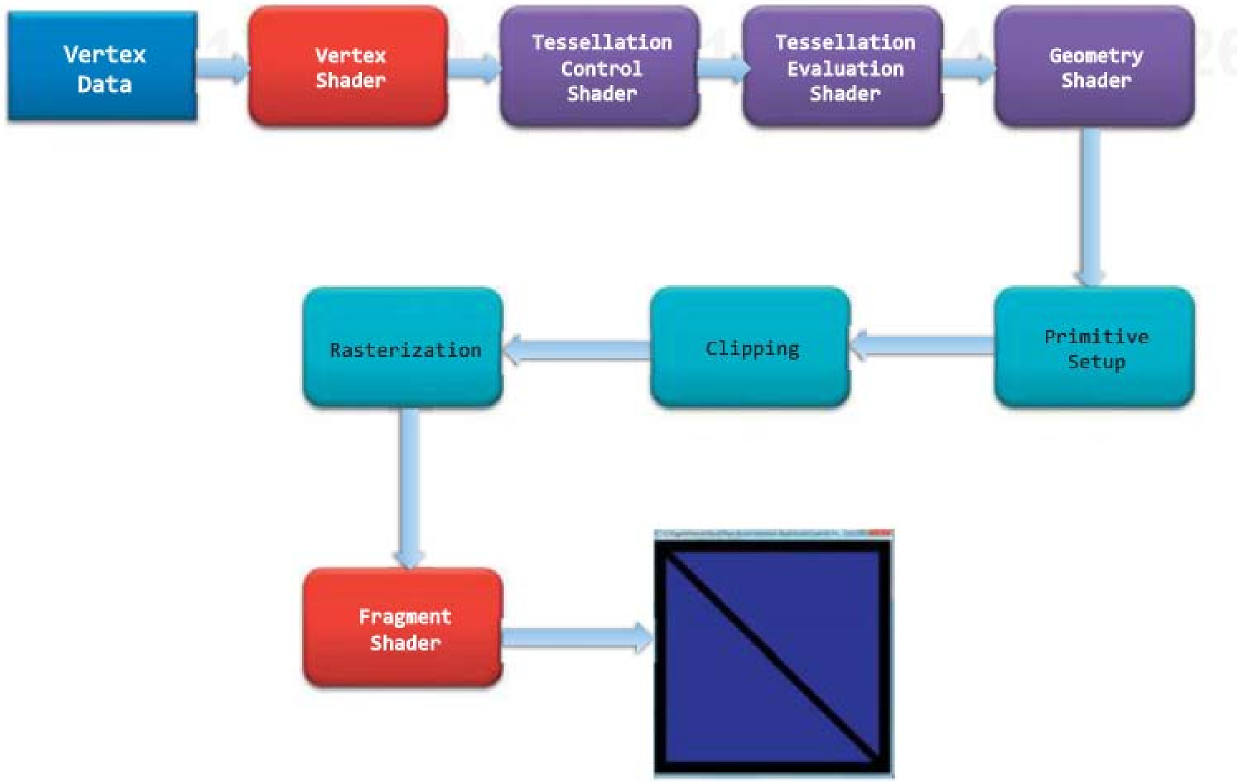

Figure 1.2 The OpenGL pipeline

Source: OpenGL programming guide, 8th edition

Summary of Key GLSL Concepts (1)

- ‘uniform’ type qualifier
  - Same for all vertices
- “in” and “out” type qualifiers configure data flow in the 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

Summary of Key GLSL Concepts (2)

- ‘layout’ qualifier
  - specify the attribute index explicitly
  - Note: each “attribute” is a vec4. So we can store up to 4 floats per attribute.
- Support for vector and matrix arithmetic
- Compiled by the OpenGL application, at runtime

Back to theory

Switch to tablet

Next class

- Representation of points AND vectors
  - Read Chapter 3 up to 3.5.

§ Linear transformations

$$\mathcal{L}(\vec{v} + \vec{u}) = \mathcal{L}(\vec{v}) + \mathcal{L}(\vec{u})$$

$$\mathcal{L}(a\vec{v}) = a\,\mathcal{L}(\vec{v})$$

Examples: scale
rotations

In a basis $\underline{\vec{b}}$

$$\vec{v} = \sum_i v_i \vec{b}_i$$

$$\mathcal{L}(\vec{v}) = \sum_i v_i\, \mathcal{L}(\vec{b}_i)$$

vectors! Can be represented in $\underline{\vec{b}}$

$$= \begin{bmatrix} \mathcal{L}(\vec{b}_1) & \mathcal{L}(\vec{b}_2) & \mathcal{L}(\vec{b}_3) \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

$$= \begin{bmatrix} \underline{\vec{b}}\,\overline{M}_1 & \underline{\vec{b}}\,\overline{M}_2 & \underline{\vec{b}}\,\overline{M}_3 \end{bmatrix} \overline{V}$$

$$= \underline{\vec{b}} \underbrace{\begin{bmatrix} \overline{M}_1 & \overline{M}_2 & \overline{M}_3 \end{bmatrix}}_{\text{a matrix}} \overline{V}$$

$$\begin{bmatrix} M_{11} & M_{21} & \\ M_{12} & M_{22} & \\ M_{13} & M_{23} & \end{bmatrix}$$

$$\mathcal{L}(\vec{v}):\ \underline{\vec{b}}\,\overline{v} \rightarrow \underline{\vec{b}}\ \underline{\overline{M}}\ \overline{v}$$

Holds for <u>any</u> basis $\underline{\vec{b}}$

If we agree upon a fixed basis,

$$\bar{v} \rightarrow \underline{\overline{M}}\,\bar{v}$$