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
C³ Survey

- What is your computing environment
  a) Linux, with lab machines
  b) Linux, personal
  c) Mac OSX
  d) Windows
  e) Something else
C³ 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

```c
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)

```c
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;
}
What is the mandatory output in a vertex shader?

a) The clip coordinates (gl\_Position)
b) The color of each vertex (e.g. frag\_Color 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

\[ L(\vec{v} + \vec{u}) = L(\vec{v}) + L(\vec{u}) \]

\[ L(a\vec{v}) = a L(\vec{v}) \]

Example: Scale
notations

In a basis \( \vec{b} \)

\[ \vec{v} = \sum_i v_i \vec{b}_i \]

\[ L(\vec{v}) = \sum_i v_i L(\vec{b}_i) \]

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

\[ = \begin{bmatrix} \vec{b} \vec{M}_1 & \vec{b} \vec{M}_2 & \vec{b} \vec{M}_3 \end{bmatrix} \vec{v} \]

\[ L(\vec{v}) : \vec{b} \vec{v} \rightarrow \vec{b} \vec{M} \vec{v} \]
Holds for any basis $\overrightarrow{b}$

If we agree upon a fixed basis, $\overrightarrow{v} \rightarrow \overrightarrow{Mv}$