Foundations of Computer Graphics
Online Lecture 8: OpenGL 2
Basic Geometry Setup

Ravi Ramamoorthi

Methodology for Lecture
- Make mytest1 more ambitious
- Sequence of steps
- Demo

Review of Last Demo
- Demo 0 [set DEMO to 4 all features]
- Changed floor to all white, added global for teapot and teapotloc, moved geometry to new header file
- Changed global for teapotloc
- Moved geometry to new header file
- Demo 0 [set DEMO to 4 all features]

#include <GL/glut.h> // also <GL/glew.h>, <GLUT/glut.h> for Mac OS
#include "shaders.h"
#include "geometry.h"

int mouseoldx, mouseoldy ; // For mouse motion
GLfloat eyevec = 2.0 ; // Where to look from. Initially 0 -2, 2
GLfloat teapotloc = -0.5 ; // ** NEW ** where the teapot is located
GLfloat animate = 0 ; // ** NEW ** whether to animate or not
GLint vertextexture, fragmenttexture, shaderprogram ; // shaders

#include <GL/glut.h> // also <GL/glew.h>, <GLUT/glut.h> for Mac OS
#include "shaders.h"
#include "geometry.h"

Review of demo from last lecture
- Basic geometry setup for cubes (pillars), colors
- Single geometric object, but multiple colors for pillars
- Matrix Stacks and Transforms (draw 4 pillars)
- Depth testing (Z-buffering)
- Animation (moving teapot)
- Texture Mapping (wooden floor)

Outline
- Review of demo from last lecture
- Basic geometry setup for cubes (pillars), colors
- Single geometric object, but multiple colors for pillars
- Matrix Stacks and Transforms (draw 4 pillars)
- Depth testing (Z-buffering)
- Animation (moving teapot)
- Texture Mapping (wooden floor)

Geometry Basic Setup 1
const int numobjects = 2 ; // number of objects for buffer
const int numperobj = 3 ;
const int ncolors = 4 ;
GLint VAOs[numobjects+ncolors], teapotVAO; // VAO for each object
GLint buffers[numperobj*numobjects+ncolors], teapotbuffers[3] ;
GLint objects[numobjects] ; // ** NEW ** For each object
GLenum PrimType[numobjects] ;
GLsizei NumElems[numobjects] ; // ** NEW ** For each object
GLenum PrimitiveType(numobjects); //
Glslaii Numslicks(numobjects);
std::vector <glm::vec3> teapotVertices; // For geometry of the teapot
std::vector <glm::vec3> teapotNormals;
std::vector <unsigned int> teapotIndices;
// To be used as a stack for the modelview.
std::vector <glm::mat4> modelviewStack;

Geometry Basic Setup 2
// ** NEW ** Floor geometry is specified with a vertex array
// ** NEW ** Same for other Geometry
enum (Vertices, Colors, Elements) ; // For arrays for object
enum (FLOOR, CUBE) ; // For objects, for the floor
const GLfloat floorverts[4][3] = {
(0.5,0.5,0.0), (-0.5,0.5,0.0), (-0.5,-0.5,0.0), (0.5,-0.5,0.0) ) ;
const GLfloat floorcol[4][3] = {
(1.0, 1.0, 1.0), (0.5, 1.0, 1.0), (1.0, 0.5, 1.0), (1.0, 0.0, 1.0) } ;
const GLbyte floorinds[1][6] = {{0, 1, 2, 0, 2, 3} } ;
const GLfloat floorvertex[4][2] = {
(1.0, 1.0), (0.0, 1.0), (0.0, 0.0), (1.0, 0.0) } ;
Cube geometry (for pillars)
const GLfloat wd = 0.1; const GLfloat ht = 0.5;
const GLfloat _cubecol[4][3] = {
    {1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {0.0, 0.0, 1.0}, {1.0, 1.0, 0.0}
};
const GLfloat cubeverts[8][3] = {
    {-wd, -wd, 0.0}, {-wd, wd, 0.0}, {wd, wd, 0.0}, {wd, -wd, 0.0},
    {-wd, -wd, ht}, {wd, -wd, ht}, {wd, wd, ht}, {-wd, wd, ht}
};

Initialize Geometry Function
// This function takes in a vertex, color, index and type array
void initobject(GLuint object, GLfloat * vert, GLint sizevert, GLfloat * col, GLint sizecol, GLubyte * inds, GLint sizeind, GLenum type) {
    // ...}
    // Use layout location 0 for the vertices
    glBufferSubData(GL_ELEMENT_ARRAY_BUFFER, vertices * sizeof(GLfloat), vertexindent, sizevert);
    // Use layout location 1 for the colors
    glBufferSubData(GL_ELEMENT_ARRAY_BUFFER, buffercolors * sizeof(GLfloat), colorindex, sizecol);
    PrimType[object] = type;
    NumElems[object] = sizeind;
    // Prevent further modification of this VAO by unbinding it
    glBindVertexArray(0);
}

Initialize Cubes with Colors 1
void initcubes(GLuint object, GLfloat * vert, GLint sizevert, GLubyte * inds, GLint sizeind, GLenum type) {
    // ...
    // Use layout location 0 for the vertices
    glBufferSubData(GL_ELEMENT_ARRAY_BUFFER, vertext * sizeof(GLfloat), vertexindent, sizevert);
    // Use layout location 1 for the colors
    glBufferSubData(GL_ELEMENT_ARRAY_BUFFER, buffercolors * sizeof(GLfloat), colorindex, sizecol);
    // Use layout location 1 for the colors
    glBufferSubData(GL_ELEMENT_ARRAY_BUFFER, buffercolors * sizeof(GLfloat), colorindex, sizecol);
    PrimType[object] = type;
    NumElems[object] = sizeind;
    // Prevent further modification of this VAO by unbinding it
    glBindVertexArray(0);
}

Initialize Cubes with Colors 2
void initcubes(GLuint object, GLfloat * vert, GLint sizevert, GLubyte * inds, GLint sizeind, GLenum type) {
    // ...
    // Use layout location 0 for the vertices
    glBufferSubData(GL_ELEMENT_ARRAY_BUFFER, buffercolors * sizeof(GLfloat), colorindex, sizecol);
    PrimType[object] = type;
    NumElems[object] = sizeind;
    // Prevent further modification of this VAO by unbinding it
    glBindVertexArray(0);
}

Initialize Cubes with Colors 3
void initcubes(GLuint object, GLfloat * vert, GLint sizevert, GLubyte * inds, GLint sizeind, GLenum type) {
    // ...
    // Use layout location 0 for the vertices
    glBufferSubData(GL_ELEMENT_ARRAY_BUFFER, buffercolors * sizeof(GLfloat), colorindex, sizecol);
    PrimType[object] = type;
    NumElems[object] = sizeind;
    // Prevent further modification of this VAO by unbinding it
    glBindVertexArray(0);
}
**Drawing with/without Colors**

// Add a function to draw with these, similar to drawobject but with color
void drawcolor(GLuint object, GLuint color) {
  glBindVertexArray(VAOs[object + color]);
  glDrawElements(PrimType[object], NumElems[object], GL_UNSIGNED_BYTE, 0);
  glkindVertExArray[0];
}

void drawobject(GLuint object) {
  glBindVertexArray(VAOs[object]);
  glDrawElements(PrimType[object], NumElems[object], GL_UNSIGNED_BYTE, 0);
  glBindVertexArray(0);
}

void loadteapot() // See source code for details if interested

---

**Foundations of Computer Graphics**

Online Lecture 8: OpenGL 2

*Matrix Stacks and Transforms (Draw 4 Pillars)*

Ravi Ramamoorthy

---

**Outline**

- Review of demo from last lecture
- Basic geometry setup for cubes (pillars), colors
  - Single geometric object, but multiple colors for pillars
- **Matrix Stacks and Transforms (draw 4 pillars)**
- Depth testing (Z-buffering)
- Animation (moving teapot)
- Texture Mapping (wooden floor)

---

**Summary OpenGL Vertex Transforms**

- **Object coords**
  - (x y z w) vertex
- **Modelview matrix**
  - [Object Transforms and glm::lookAt]
- **Perspective Divide**
  - (Dehomogenization)
- **Clip coordinates**
- **Eye coordinates**
  - (used for lighting)
- **Normalized Device Coordinates**
- **Viewport Transform**
  - (glViewport)
- **Window Coords**

---

**Transformations**

- **Matrix Stacks**
  - Useful for hierarchically defined figures, placing pillars
  - Old OpenGL: glPushMatrix, glPopMatrix
  - Current recommendation is STL stacks managed yourself, which is done in mytest2. (You must manage the stack yourself for HW 2).
- **Transforms**
  - Write your own translate, scale, rotate for HW 1 and HW 2
  - Careful of OpenGL convention: In old-style, **Right-multiply** current matrix (last is first applied). glm operators follow this sometimes.
- Also gluLookAt (glm::lookAt), gluPerspective (glm::perspective)
  - Remember just matrix like any other transform, affecting modelview
  - See mytest for how to best implement these ideas

---

**Drawing Pillars 1 (in display)**

// 1st pillar: Right-multiply modelview as in old OpenGL
pushMatrix(modelview) ; // push/pop functions for stack
modelview = modelview * glm::translate(identity, glm::vec3(-0.4, -0.4, 0.0)) ; // build translation matrix
glUniformMatrix4fv(modelviewPos, 1, GL_FALSE, &(modelview)[0][0]);
drawcolor(CUBE, 0) ;
popMatrix(modelview) ;

// 2nd pillar
pushMatrix(modelview) ;
modelview = modelview * glm::translate(identity, glm::vec3(0.4, -0.4, 0.0)) ; // build translation matrix
glUniformMatrix4fv(modelviewPos, 1, GL_FALSE, &(modelview)[0][0]);
drawcolor(CUBE, 1) ;
popMatrix(modelview) ;
Drawing Pillars 2

```cpp
// 3rd pillar
pushMatrix(modelview);
modelview = modelview * glm::translate(identity,
glm::vec3(0.4, 0.4, 0.0));
glUniformMatrix4fv(modelviewPos, 1, GL_FALSE, &(modelview)[0][0]);
drawcolor(CUBE, 2);
popMatrix(modelview);

// 4th pillar
pushMatrix(modelview);
modelview = modelview * glm::translate(identity,
glm::vec3(-0.4, 0.4, 0.0));
glUniformMatrix4fv(modelviewPos, 1, GL_FALSE, &(modelview)[0][0]);
drawcolor(CUBE, 3);
popMatrix(modelview);
```

Push and Pop

```cpp
// Function pushes specified matrix onto the modelview stack
void pushMatrix(glm::mat4 mat) {
    modelviewStack.push_back(glm::mat4(mat));
}

// This function pops a matrix from the modelview stack
void popMatrix(glm::mat4& mat) {
    if (modelviewStack.size()) {
        mat = glm::mat4(modelviewStack.back());
        modelviewStack.pop_back();
    } else { // Just to prevent errors when popping from empty stack.
        mat = glm::mat4(1.0f);
    }
}
```

Demo

- Demo 1
  - Does order of drawing matter?
  - What if I move floor after pillars in code?
  - Is this desirable? If not, what can I do about it?

Foundations of Computer Graphics

Online Lecture 8: OpenGL 2

Depth Testing (Z-Buffering)

Ravi Ramamoorthi

Outline

- Review of demo from last lecture
- Basic geometry setup for cubes (pillars), colors
  - Single geometric object, but multiple colors for pillars
- Matrix Stacks and Transforms (draw 4 pillars)
- Depth testing (Z-buffering)
- Animation (moving teapot)
- Texture Mapping (wooden floor)

Double Buffering

- New primitives draw over (replace) old objects
  - Can lead to jerky sensation
- Solution: double buffer. Render into back (off-screen) buffer. When finished, swap buffers to display entire image at once.
- Changes in main and display
  `glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);`
  `glutSwapBuffers();
glFlush();`
Turning on Depth test (Z-buffer)

OpenGL uses a Z-buffer for depth tests
- For each pixel, store nearest Z value (to camera) so far
- If new fragment is closer, it replaces old z, color
  ["less than" can be over-ridden in fragment program]
- Simple technique to get accurate visibility

Changes in main fn, display to Z-buffer

```
glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);  
glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);  
```

In init function

```
glEnable (GL_DEPTH_TEST) ;  
glDepthFunc (GL_LESS) ; // The default option
```

Demo

- Demo 2
- Does order of drawing matter any more?
- What if I change near plane to 0?
- Is this desirable? If not, what can I do about it?

Drawing Teapot (in display)

```
// ** NEW ** Put a teapot in the middle that animates
pushMatrix (modelview);  
modelview = modelview * glm::translate (identity, 
glm::vec3 (teapotloc, 0.0, 0.0));  
// The following two transforms set up and center the teapot 
// Transforms right-multiply the modelview matrix (top of the stack)
modelview = modelview * glm::translate (identity, glm::vec3 (0.0, 0.0, 0.1));  
modelview = modelview * glm::rotate (identity, glm::pi<float>() / 
2.0f, glm::vec3 (1.0, 0.0, 0.0));  
teatop size = 0.235f; // Teapot size
modelview = modelview * glm::scale (identity, glm::vec3 (size, size, size));  
glLoadIdentityMatrix4fv (modelviewPos, 1, GL_FALSE, & (modelview)[0][0]);  
drawteapot() ;  
popMatrix (modelview);  
```
Simple Animation routine

```c
// ** NEW ** in this assignment, is an animation of a teapot
// Hitting 'p' will pause this animation, see keyboard callback
void animation(void)
{
    teapotloc = teapotloc + 0.005 ;
    if (teapotloc > 0.5) teapotloc = -0.5 ;
    glutPostRedisplay() ;
}
```

drawteapot() function in geometry.h

```c
void drawteapot()
{
    glBindVertexArray(teapotVAO);
    glDrawElements(GL_TRIANGLES, teapotIndices.size(), GL_UNSIGNED_INT, 0);
    glBindVertexArray(0);
}
```

Keyboard callback (p to pause)

```c
GLint animate = 0 ; // ** NEW ** whether to animate or not
void keyboard (unsigned char key, int x, int y)
{
    switch (key) {
    case 27:  // Escape to quit
        exit(0) ;
        break ;
    case 'p': // ** NEW ** to pause/restart animation
        animate = !animate ;
        if (animate) glutIdleFunc(animation) ;
        else glutIdleFunc(NULL) ;
        break ;
    default:
        break ;
    }
}
```

Outline

- Review of demo from last lecture
- Display lists (extend init for pillars)
- Matrix stacks and transforms (draw 4 pillars)
- Depth testing or z-buffering
- Animation (moving teapot)
- Texture mapping (wooden floor) [mytest3]

New globals and basic setup

```c
// In mytest3.cpp
GLubyte woodtexture[256][256][3] ; // texture (from grsites.com)
GLuint texNames[1] ; // texture buffer
GLuint istex ; // blend parameter for texturing
GLuint ilight ; // for lighting
GLuint texturing = 1 ; // to turn on/off texturing
GLuint lighting = 1 ; // to turn on/off lighting
// In Display
glUniform1i(istex,0) ; // Turn off lighting (except on teapot, later)
gUniform1i(istex,texturing) ;
glUniform1i(istex,0) ; // Texturing floor // drawobject(FLOR) ;
gUniform1i(istex,0) ; // Other items aren't textured
```

Simple Toggles for Keyboard

```c
case 't': // ** NEW ** to turn on/off texturing :
         texturing = !texturing ;
         glutPostRedisplay() ;
         break ;
```
Adding Visual Detail

- Basic idea: use images instead of more polygons to represent fine scale color variation

Texture Mapping

- Important topic: nearly all objects textured
  - Wood grain, faces, bricks and so on
  - Adds visual detail to scenes
- Can be added in a fragment shader

Setting up texture

```c
inittexture("wood.ppm", shaderprogram); // in init()

// Very basic code to read a ppm file
// And then set up buffers for texture coordinates
void inittexture(const char * filename, GLenum program) {
    int i, j, k;
    FILE * fp;
    assert(fp = fopen(filename, "rb"));
    fscanf(fp, "%*s %*d %*d %*d%*c");
    for (i = 0; i < 256; i++)
        for (j = 0; j < 256; j++)
            for (k = 0; k < 3; k++)
                fscanf(fp, "%c", &woodtexture[i][j][k]);
    fclose(fp);
}
```

Texture Coordinates

- Each vertex must have a texture coordinate: pointer to texture. Interpolate for pixels (each fragment has st)

```c
// Set up Texture Coordinates
glGenTextures(1, texNames);
glBindVertexArray(VAOs[FLOOR]);
glBindBuffer(GL_ARRAY_BUFFER, buffers[numobjects*numperobj+ncolors]);
glBufferData(GL_ARRAY_BUFFER, sizeof(floortex), floortex, GL_STATIC_DRAW);
// Use layout location 2 for texcoords
glEnableVertexAttribArray(2);
gVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 2 * sizeof(GLfloat), 0);
GLint texsampler;
texsampler = glGetUniformLocation(program, "tex");
gUniform1i(texsampler, 0); // Could also be GL_TEXTURE0
istex = glGetUniformLocation(program, "istex");
```

Specifying the Texture Image

- `glTexImage2D(target, level, components, width, height, border, format, type, data)`
- `target` is GL_TEXTURE_2D
- `level` is (almost always) 0
- `components` = 3 or 4 (RGB/RGBA)
- `width/height` MUST be a power of 2
- `format` = GL_RGB or GL_RGBA (usually)
- `type` = GL_UNSIGNED_BYTE, GL_FLOAT, etc...

Texture Image and Bind to Shader

```c
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 256, 256, 0, GL_RGB, GL_UNSIGNED_BYTE, woodtexture);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);

GLint texsampler;
texsampler = glGetUniformLocation(program, "tex");
gUniform1i(texsampler, 0); // Could also be GL_TEXTURE0
istex = glGetUniformLocation(program, "istex");
```
Drawing with Texture

// And a function to draw with texture, similar to drawObject
void drawTexture(GLuint object, GLuint texture) {
    glBindTexture(GL_TEXTURE_2D, texture);
    glBindVertexArray(VAOs[object]);
    glDrawElements(PrimType[object], NumElems[object], GL_UNSIGNED_BYTE, 0);
    glBindVertexArray(0);
}

Final Steps for Drawing

- Vertex shader (just pass on texture coords)

```glsl
layout (location = 2) in vec2 texCoords;
out vec2 texcoord; // similar definitions for positions and normals
uniform int istex;
void main() {
    gl_Position = projection * modelview * vec4(position, 1.0f);
    mynormal = mat3(transpose(inverse(modelview))) * normal;
    myvertex = modelview * vec4(position, 1.0f);
    texcoord = vec2(0.0, 0.0); // Default value just to prevent errors
    if (istex != 0) { texcoord = texCoords; }
}
```

Final Steps for Drawing (+Demo)

- Fragment shader (can be more complex blend)

```glsl
uniform sampler2D tex;
uniform int istex;
void main (void) {
    if (istex > 0) fragColor = texture(tex, texCoord);
}
```