Methodology for Lecture

- Make mytest1 more ambitious
- Sequence of steps
- Demo

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

```
const int numobjects = 2; // number of objects for buffer
const int numperobj = 3;
const int ncolors = 4;
GLuint buffers[numobjects*numperobj+ncolors]; // ** NEW ** List of
    // Buffers for geometric data
GLuint objects[numobjects]; // For each object
GLenum PrimType[numobjects];

GLint numVertices(numobjects); // For arrays for object
GLint numColors(numobjects); // For arrays for object
GLint numElem(numobjects); // For arrays for object
```

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}};
```

Review of Last Demo

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

```c
#include <GL/glut.h>
#include "shaders.h"
#include "geometry.h"

int mouseoldx, mouseoldy; // For mouse motion
GLdouble eyeloc = 2.0; // Where to look from; initially 0 -2, 2
GLint teapotloc = -0.5; // ** NEW ** where the teapot is located
GLint animate = 0; // ** NEW ** whether to animate or not
GLuint vertexshader, fragmentshader, shaderprogram; // shaders
const int DEMO = 0; // ** NEW ** To turn on and off features
```
Cube Geometry (separate Color)

// Simple function to set the color separately. Takes out colors
void initobjectnocol(GLuint object, GLfloat * vert, GLint sizevert,
GLubyte * inds, GLint sizeind, GLenum type) {
    int offset = object * numperobj;
    glBindBuffer(GL_ARRAY_BUFFER, buffers[Vertices+offset]);
    glBufferData(GL_ARRAY_BUFFER, sizevert, vert, GL_STATIC_DRAW);
    glVertexPointer(3, GL_FLOAT, 0, BUFFER_OFFSET(0));
    glEnableClientState(GL_VERTEX_ARRAY);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, buffers[Elements+offset]);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeind, inds, GL_STATIC_DRAW);
    PrimType[object] = type;
    NumElems[object] = sizeind;
}

Cube Colors

// Simple function to init a bunch of color buffers for the cube
void initcolorscube(void) {
    int base = numobjects * numperobj;
    for (int i = 0; i < ncolors; i++){
        for (int j = 0; j < 8; j++)
            for (int k = 0; k < 3; k++)
                cubecol[j][k] = _cubecol[i][k];
        glBindBuffer(GL_ARRAY_BUFFER, buffers[base+i]);
        glBufferData(GL_ARRAY_BUFFER, sizeof(cubecol), cubecol, GL_STATIC_DRAW);
        glColorPointer(3, GL_FLOAT, 0, BUFFER_OFFSET(0));
        glEnableClientState(GL_COLOR_ARRAY);
    }
}

// in init
initobjectnocol(CUBE, (GLfloat *) cubeverts, sizeof(cubeverts), (GLubyte *)
cubeinds, sizeof(cubeinds), GL_QUADS);

Drawing with Cube Colors

// And a function to draw with them, similar to drawobject but with color
void drawcolor(GLuint object, GLuint color) {
    int offset = object * numperobj;
    int base = numobjects * numperobj;
    glBindBuffer(GL_ARRAY_BUFFER, buffers[Vertices+offset]);
    glVertexPointer(3, GL_FLOAT, 0, BUFFER_OFFSET(0));
    glEnableClientState(GL_VERTEX_ARRAY);
    glBindBuffer(GL_ARRAY_BUFFER, buffers[base+color]); // Set color
    glColorPointer(3, GL_FLOAT, 0, BUFFER_OFFSET(0));
    glEnableClientState(GL_COLOR_ARRAY);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, buffers[Elements+offset]);
    glDrawElements(PrimType[object], NumElems[object], GL_UNSIGNED_BYTE,
    BUFFER_OFFSET(0));
}

Foundations of Computer Graphics

Online Lecture 8: OpenGL 2

Matrix Stacks and Transforms (Draw 4 Pillars)

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)

Summary OpenGL Vertex Transforms

<table>
<thead>
<tr>
<th>Object coords (x y z w)</th>
<th>Clip coordinates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perspective Divide</td>
<td>(Dehomogenization)</td>
</tr>
<tr>
<td>Normalized Device</td>
<td>Coordinates</td>
</tr>
<tr>
<td>Viewport Transform</td>
<td>(glViewport)</td>
</tr>
</tbody>
</table>

| Mode/view matrix       | Eye coordinates   |
| Object Transforms      | (used for lighting) |
| glm::lookAt            |                   |

Projection matrix

[3D to 2D, usually glm::perspective]
Transformations

Matrix Stacks
- Useful for hierarchically defined figures, placing pillars
  - Old OpenGL: glPushMatrix, glPopMatrix, glLoad, glMultMatrixf
  - Mytest2 uses old-style stacks. Current recommendation is STL stacks managed yourself. (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)
  - gluLookAt just matrix like any other transform, affecting modelview
  - Must come before in code, after in action to other transforms
  - Why not usually an issue for gluPerspective?

Drawing Pillars 1 (in display)

```c
// 1st pillar
glPushMatrix();
glTranslatef(-0.4,-0.4,0.0);
drawcolor(CUBE, 0);
glPopMatrix();

// 2nd pillar
glPushMatrix();
glTranslatef(0.4,-0.4,0.0);
drawcolor(CUBE, 1);
glPopMatrix();
```

Drawing Pillars 2

```c
// 3rd pillar
glPushMatrix();
glTranslatef(0.4,0.4,0.0);
drawcolor(CUBE, 2);
glPopMatrix();

// 4th pillar
glPushMatrix();
glTranslatef(-0.4,0.4,0.0);
drawcolor(CUBE, 3);
glPopMatrix();
```

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?

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 (2-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:
  ```c
  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:
```c
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
  glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

In init function:
```c
  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?

**Foundations of Computer Graphics**

Online Lecture 8: OpenGL 2

*Animation (Moving Teapot)*

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)

**Demo**

- Demo 3
- Notice how teapot cycles around
- And that I can pause and restart animation
- And do everything else (zoom etc.) while teapot moves in background
Drawing Teapot (in display)

```c
// ** NEW ** Put a teapot in the middle that animates
void glutDisplay() {
    glColor3f(0.0,1.0,1.0) ; // Deprecated command to set the color
    glPushMatrix() ;
    // I now transform by the teapot translation for animation */
    glutSolidTeapot(0.15) ;
    glutPostRedisplay() ;
    glutIdleFunc(animation) ;
}
```

Simple Animation routine

```c
// ** NEW ** in this assignment, is an animation of a teapot
void animation(void) {
    GLfloat teapotloc = teapotloc + 0.005 ;
    if (teapotloc > 0.5) teapotloc = -0.5 ;
    glutPostRedisplay() ;
}
```

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
        if (animate) glutIdleFunc(animation) ;
        else glutIdleFunc(NULL) ;
        break ;
    default:
        break ; }
}
```

Foundations of Computer Graphics
Online Lecture 8: OpenGL 2
Texture Mapping (Wooden Floor – mytest3)
Ravi Ramamoorthi

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
GLuint woodtexture[256][256][3] ; // texture (from givitese.com)
GLuint texName[1] ; // texture buffer
GLuint istex ; // blend parameter for texturing
GLuint islight ; // for lighting
GLuint texName[4] ; // for texturing floor
GLuint istex[4] ; // to turn on/off texturing
GLuint lighting[4] ; // to turn on/off lighting

glUniform1i(islight,0) ; // Turn off lighting (except on teapot, later)
glUniform1i(istex,0) ; // Texturing floor
drawtexture(FLOOR,texNames[0]) ; // Texturing floor
```

```c
GLuint istex[4] ; // Other items aren’t textured
```
Simple Toggles for Keyboard

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

case 's': // ** NEW ** to turn on/off shading (always smooth)
    lighting = !lighting;
    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

Polygonal model With surface texture

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);

glBindBuffer(GL_ARRAY_BUFFER, buffers[numobjects*numperobj+ncolors]);
glBufferData(GL_ARRAY_BUFFER, sizeof(floortex), floortex, GL_STATIC_DRAW);

glActiveTexture(GL_TEXTURE0);
if (texture) glEnable(GL_TEXTURE_2D);

glTexCoordPointer(2, GL_FLOAT, 0, BUFFER_OFFSET(0));
glEnableClientState(GL_TEXTURE_COORD_ARRAY);

glBindTexture(GL_TEXTURE_2D, texNames[0]);
```

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
- `border = 0` (usually)
- `format = GL_RGB` or `GL_RGBA` (usually)
- `type = GL_UNSIGNED_BYTE, GL_FLOAT, etc...`
Texture Image and Bind to Shader

```c
void drawtexture(GLuint object, GLuint texture) {
    // Textures
    glEnable(GL_TEXTURE_2D, textures[object]);
    glBindTexture(GL_TEXTURE_2D, texture);
    glTexCoordPointer(2, GL_FLOAT, 0, BUFFER_OFFSET(0));
    glDrawElements(PrimType[object], NumElems[object], GL_UNSIGNED_BYTE, BUFFER_OFFSET(0));
}
```

Drawing with Texture

```c
Texture Image and Bind to Shader

void drawtextured(GLuint object, GLuint texture) {
    // Textures
    glEnable(GL_TEXTURE_2D, textures[object]);
    glBindTexture(GL_TEXTURE_2D, texture);
    glTexCoordPointer(2, GL_FLOAT, 0, BUFFER_OFFSET(0));
    glDrawElements(PrimType[object], NumElems[object], GL_UNSIGNED_BYTE, BUFFER_OFFSET(0));
}
```

Final Steps for Drawing (+Demo)

- Vertex shader (just pass on texture coords)
  ```
  gl_TexCoord[0] = gl_MultiTexCoordCoord;
  ```
- Fragment shader (can be more complex blend)
  ```
  uniform sampler2D tex;
  uniform int istex;
  void main (void) {
      if (istex > 0) gl_FragColor = texture2D(tex, gl_TexCoord[0].st);
  }
  ```