CIS 441/541: Introduction to Computer Graphics
Lecture 10: texture review, display lists, geometry specification, lighting and materials, project 2A

May 8th, 2013
Hank Childs, University of Oregon
Outline

- Misc.
- Debugging tips
- Texture Review
- Display Lists
- Specifying more than triangles in OpenGL
- Project 2A
- Lighting in OpenGL
- Shading in OpenGL
- Materials in OpenGL
Misc.

Debugging tips

Texture Review

Display Lists

Specifying more than triangles in OpenGL

Project 2A

Lighting in OpenGL

Shading in OpenGL

Materials in OpenGL
Regular OH

- Tues 10-12 and Weds 2-4
  - Surge OH on Fri’s, esp when Tues OH are cancelled
  - Tues OH cancelled on Tues, May 14
- Friday lecture in GSH 117 (no room change)
  - Still want to see research lecture?
- Acceleration: next week
Project 1F

- Clear what to do?
- Clear what to hand in?
- Questions?
- (don’t send me your movies)
Outline

- Misc.
- Debugging tips
- Texture Review
- Display Lists
- Specifying more than triangles in OpenGL
- Project 2A
- Lighting in OpenGL
- Shading in OpenGL
- Materials in OpenGL
“My specular component gets smaller and smaller…”

What’s wrong with this code?:

double *foo()
{
    double RV[16] = { ... };
    ...
    return rv;
}

My code runs for ever…
How ceil441 and floor441 started messing things up…
Outline

- Misc.
- Debugging tips
- Texture Review
- Display Lists
- Specifying more than triangles in OpenGL
- Project 2A
- Lighting in OpenGL
- Shading in OpenGL
- Materials in OpenGL
Visualization use case

Why is there purple in this picture?
class vtk441PolyDataMapper : public vtkOpenGLPolyDataMapper
{
    public:
        static vtk441PolyDataMapper *New();
    virtual void RenderPiece(vtkRenderer *ren, vtkActor *act)
    {
        glEnable(GL_COLOR_MATERIAL);
        float ambient[3] = { 1, 1, 1 };  
        glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, ambient);
        glBegin(GL_TRIANGLES);
        glColor3ub(0, 0, 255);
        glVertex3f(0,0,0);
        glColor3ub(0, 255, 0);
        glVertex3f(0,1,0);
        glColor3ub(255, 0, 0);
        glVertex3f(1,1,0);
        glEnd();
    }
};
Textures: a better way to specify a color map

There is no purple when we use textures
1D textures: basic idea

- Store color map on GPU as a texture

- Old color interpolation of fragment on a scanline:
  - For (int j = 0 ; j < 3 ; j++)
    - RGB[j] = leftRGB[j] + proportion*(rightRGB[j]-leftRGB[j])

- New color interpolation of fragment on a scanline:
  - textureVal = leftTextureVal
    + proportion*(rightTextureVal-leftTextureVal)
  - RGB ← textureLookup[textureVal]
Triangle with vertices with scalar values 2.9, 3.3, and 3.1.

- $T$ for 2.9 = $\frac{(2.9-3.0)}{(3.25-3)} = -0.4$
- $T$ for 3.1 = $\frac{(3.1-3.0)}{(3.25-3)} = 0.4$
- $T$ for 3.3 = $\frac{(3.3-3.0)}{(3.25-3)} = 1.2$

Fragment colors come from interpolating texture coordinates and applying texture.
class vtk441PolyDataMapper : public vtkOpenGLPolyDataMapper
{
 public:
  static vtk441PolyDataMapper *New();

  virtual void RenderPiece(vtkRenderer *ren, vtkActor *act)
  {
    GLubyte Texture3[9] = {
      0, 0, 255, // blue
      255, 255, 255, // white
      255, 0, 0, // red
    };
    glTexImage1D(GL_TEXTURE_1D, 0, GL_RGB, 3, 0, GL_RGB,
                 GL_UNSIGNED_BYTE, Texture3);
    glEnable(GL_COLOR_MATERIAL);
    glTexParameterf(GL_TEXTURE_1D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
    glTexParameterf(GL_TEXTURE_1D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    glEnable(GL_TEXTURE_1D);
    float ambient[3] = { 1, 1, 1};
    glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, ambient);
    glBegin(GL_TRIANGLES);
    glTexCoord1f(0);
    glVertex3f(0,0,0);
    glTexCoord1f(0.0);
    glVertex3f(0,1,0);
    glTexCoord1f(1.);
    glVertex3f(1,1,0);
    glEnd();
  }
};
What do we expect the output to be?

class vtk441PolyDataMapper : public vtkOpenGLPolyDataMapper
{
public:
    static vtk441PolyDataMapper */New();

    virtual void RenderPiece(vtkRenderer *ren, vtkActor *act) {
        vtkJPEGReader *rdr = vtkJPEGReader::New();
        rdr->SetFileName("HankChilds_345.jpg");
        rdr->Update();
        vtkImageData *img = rdr->GetOutput();
        int dims[3];
        img->GetDimensions(dims);
        unsigned char *buffer = (unsigned char *) img->GetScalarPointer(0,0,0);
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, dims[0], dims[1], 0, GL_RGB,
                     GL_UNSIGNED_BYTE, buffer);
        glEnable(GL_COLOR_MATERIAL);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
        glEnable(GL_TEXTURE_2D);
        float ambient[3] = { 1, 1, 1 };  
        glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, ambient);
        glBegin(GL_TRIANGLES);
        glTexCoord2f(0,0);
        glVertex3f(0,0,0);
        glVertex3f(1,0);
        glVertex3f(0,1,0);
        glVertex3f(1,1,0);
        glEnd();
    }
};
Outline

- Misc.
- Debugging tips
- Texture Review
- Display Lists
- Specifying more than triangles in OpenGL
- Project 2A
- Lighting in OpenGL
- Shading in OpenGL
- Materials in OpenGL
CPU and GPU

- Most common configuration has CPU and GPU on separate dies
- I.e., plug GPU in CPU

CPU
(typically 4-12 cores, ~10GFLOPs)

GPU
(typically 100-1000 cores, ~100GFLOPs-~1000GFLOPs)

Peripheral Component Interconnect Express

What are the performance ramifications of this architecture?
Display lists

- Idea:
  - send geometry and settings to GPU once, give it an identifier
  - GPU stores geometry and settings
  - Just pass the identifier for every subsequent render
Display lists

- Generate an identifier:
  ```c
  GLuint displayList = glGenLists(1);
  ```

- Tell GPU that all subsequent geometry is part of the list:
  ```c
  glNewList(displayList,GL_COMPILE);
  ```

- Specify geometry (i.e., glVertex, etc)

- Tell GPU we are done specifying geometry:
  ```c
  glEndList();
  ```

- Later on, tell GPU to render all the geometry and settings associated with our list:
  ```c
  glCallList(displayList);
  ```
for (int frame = 0; frame < nFrames; frame++)
{
    SetCamera(frame, nFrames);
    glBegin(GL_TRIANGLES);
    for (int i = 0; i < triangles.size(); i++)
    {
        for (int j = 0; j < 3; j++)
        {
            glColor3ubv(triangles[i].colors[j]);
            glColor3fv(triangles[i].vertices[j]);
        }
        glEnd();
    }
    glEndList();
}

GLUint displayList = glGenLists(1);
glNewList(displayList, GL_COMPILE);
glBegin(GL_TRIANGLES);
for (int i = 0; i < triangles.size(); i++)
{
    for (int j = 0; j < 3; j++)
    {
        glColor3ubv(triangles[i].colors[j]);
        glColor3fv(triangles[i].vertices[j]);
    }
    glEnd();
    glEndList();
}

for (int frame = 0; frame < nFrames; frame++)
{
    SetCamera(frame, nFrames);
    glCallList(displayList);
}
glNewList

- **GL_COMPILE**
  - Make the display list for later use.

- **GL_COMPILE_AND_EXECUTE**
  - Make the display list and also execute it as you go.
Outline

- Misc.
- Debugging tips
- Texture Review
- Display Lists
- Specifying more than triangles in OpenGL
- Project 2A
- Lighting in OpenGL
- Shading in OpenGL
- Materials in OpenGL
Geometry Specification: glBegin

Name

glBegin — delimit the vertices of a primitive or a group of like primitives

C Specification

void glBegin(GLenum mode);

Parameters

mode

Specifies the primitive or primitives that will be created from vertices presented between glBegin and the subsequent glEnd. Ten symbolic constants are accepted: GL_POINTS, GL_LINES, GL_LINE_STRIP, GL_LINE_LOOP, GL_TRIANGLES, GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN, GL_QUADS, GL_QUAD_STRIP, and GL_POLYGON.

C Specification

void glEnd(void);

Description

glBegin and glEnd delimit the vertices that define a primitive or a group of like primitives. glBegin accepts a single argument that specifies in which of ten ways the vertices are interpreted. Taking n as an integer count starting at one, and N as the total number of vertices specified, the interpretations are as follows:
Geometry Primitives

GL_POINTS

Treats each vertex as a single point. Vertex n defines point n. N points are drawn.

GL_LINES

Treats each pair of vertices as an independent line segment. Vertices 2n - 1 and 2n define line n. N 2 lines are drawn.

GL_LINE_STRIP

Draws a connected group of line segments from the first vertex to the last. Vertices n and n + 1 define line n. N - 1 lines are drawn.

GL_LINE_LOOP

Draws a connected group of line segments from the first vertex to the last, then back to the first. Vertices n and n + 1 define line n. The last line, however, is defined by vertices N and 1. N lines are drawn.
GL_TRIANGLES

Treats each triplet of vertices as an independent triangle. Vertices $3 \times n - 2$, $3 \times n - 1$, and $3 \times n$ define triangle $n$. $N$ 3 triangles are drawn.

GL_TRIANGLE_STRIP

Draws a connected group of triangles. One triangle is defined for each vertex presented after the first two vertices. For odd $n$, vertices $n$, $n + 1$, and $n + 2$ define triangle $n$. For even $n$, vertices $n + 1$, $n$, and $n + 2$ define triangle $n$. $N - 2$ triangles are drawn.

GL_TRIANGLE_FAN

Draws a connected group of triangles. One triangle is defined for each vertex presented after the first two vertices. Vertices $1$, $n + 1$, and $n + 2$ define triangle $n$. $N - 2$ triangles are drawn.
**GL_QUADS**

Treats each group of four vertices as an independent quadrilateral. Vertices $4 \cdot n - 3$, $4 \cdot n - 2$, $4 \cdot n - 1$, and $4 \cdot n$ define quadrilateral $n$. $N$ 4 quadrilaterals are drawn.

**GL_QUAD_STRIP**

Draws a connected group of quadrilaterals. One quadrilateral is defined for each pair of vertices presented after the first pair. Vertices $2 \cdot n - 1$, $2 \cdot n$, $2 \cdot n + 2$, and $2 \cdot n + 1$ define quadrilateral $n$. $N - 1$ quadrilaterals are drawn. Note that the order in which vertices are used to construct a quadrilateral from strip data is different from that used with independent data.

**GL_POLYGON**

Draws a single, convex polygon. Vertices 1 through $N$ define this polygon.
What can go inside a glBegin?

Only a subset of GL commands can be used between glBegin and glEnd. The commands are glVertex, glColor, glSecondaryColor, glIndex, glNormal, glFogCoord, glTexCoord, glMultiTexCoord, glVertexAttrib, glEvalCoord, glEvalPoint, glArrayElement, glMaterial, and glEdgeFlag. Also, it is acceptable to use glCallList or glCallLists to execute display lists that include only the preceding commands. If any other GL command is executed between glBegin and glEnd, the error flag is set and the command is ignored.
How to make a graphics program?

- Need to create a window
  - This window contains a “context” for OpenGL to render in.
- Need to be able to deal with events/interactions
- Need to render graphics primitives
  - OpenGL!
We will replace these and write our own GL calls.

from vtkpython import *
cone = vtkConeSource()
cone.SetResolution(10)
coneMapper = vtkPolyDataMapper()
coneMapper.SetInput(cone.GetOutput())
coneActor = vtkActor()
coneActor.SetMapper(coneMapper)
ren = vtkRenderer()
ren.AddActor(coneActor)
renWin = vtkRenderWindow()
renWin.SetWindowName("Cone")
renWin.SetSize(300,300)
renWin.AddRenderer(ren)
iren = vtkRenderWindowInteractor()
iren.SetRenderWindow(renWin)
iren.Initialize()
iren.Start()

We will re-use these.
Project #2A (8%), Due Thurs May 16th, 6am

- Goal: OpenGL program that does regular colors and textures
- New VTK-based project2A.cxx
- New CMakeLists.txt (but same as old ones)
Code exploration
Hints

- I recommend you “walk before you run” & “take small bites”. OpenGL can be very punishing. Get a picture up and then improve on it. Make sure you know how to retreat to your previously working version at every step.

- OpenGL “state thrashing” is common and tricky to debug.
  - Get one window working perfectly.
  - Then make the second one work perfectly.
  - Then try to get them to work together.
    - Things often go wrong, when one program leaves the OpenGL state in a way that doesn’t suit another renderer.