CIS 441/541: Introduction to Computer Graphics
Lecture 16: textures
Announcements

- Midterm on November 18th

- Old:
  - 441 students must do self-defined final project

- New:
  - 441 students do self-defined final project
  - -- or --
  - 441 students do additional projects, as defined by 541 students
Any questions on Project 1?
(who made a movie?)
Review
OpenGL Architecture

Immediate Mode

Polynomial Evaluator

Per Vertex Operations & Primitive Assembly

Display List

Rasterization

Per Fragment Operations

CPU

Pixel Operations

Texture Memory

Frame Buffer

geometry pipeline
OpenGL Functions

- Primitives
  - Points
  - Line Segments
  - Polygons
- Attributes
- Transformations
  - Viewing
  - Modeling
- Control (GLUT)
- Input (GLUT)
- Query

VTK
OpenGL State

• OpenGL is a state machine

• OpenGL functions are of two types
  - Primitive generating
    • Can cause output if primitive is visible
    • How vertices are processed and appearance of primitive are controlled by the state
  - State changing
    • Transformation functions
    • Attribute functions
OpenGL function format

`glVertex3f(x, y, z)`

- function name
- `x, y, z` are floats
- belongs to GL library

`glVertex3fv(p)`

- `p` is a pointer to an array
OpenGL #defines

• Most constants are defined in the include files gl.h, glu.h and glut.h
  - Note #include <GL/glut.h> should automatically include the others
  - Examples
    – glBegin(GL_POLYGON)
    – glClear(GL_COLOR_BUFFER_BIT)

• include files also define OpenGL data types: GLfloat, GLdouble, …
A Simple Program

Generate a square on a solid background
#include <GL/glut.h>

void mydisplay()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_POLYGON);
        glVertex2f(-0.5, -0.5);
        glVertex2f(-0.5, 0.5);
        glVertex2f(0.5, 0.5);
        glVertex2f(0.5, -0.5);
    glEnd();
    glFlush();
}

int main(int argc, char** argv)
{
    glutCreateWindow("simple");
    glutDisplayFunc(mydisplay);
    glutMainLoop();
}

Angel: Interactive Computer Graphics 5E © Addison-Wesley 2009
Event Loop

• Note that the program defines a **display callback** function named **mydisplay**
  
  - Every glut program must have a display callback
  
  - The display callback is executed whenever OpenGL decides the display must be refreshed, for example when the window is opened
  
  - The **main** function ends with the program entering an event loop
Defaults

• `simple.c` is too simple
• Makes heavy use of state variable default values for
  - Viewing
  - Colors
  - Window parameters
• Next version will make the defaults more explicit
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!
Windows and Events

• Creating windows and dealing with events varies from platform to platform.
“Hello World” with X-Windows.

Compile with:

- gcc -L/usr/X11R6/lib -lX11 hello-x.c -o hello-x

```c
#include <X11/Xlib.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(void) {
    Display *d;
    Window w;
    XEvent e;
    char *msg = "Hello, World!";
    int s;

    d = XOpenDisplay(NULL);
    if (d == NULL) {
        fprintf(stderr, "Cannot open display\n");
        exit(1);
    }

    s = DefaultScreen(d);
    w = XCreateSimpleWindow(d, RootWindow(d, s), 10, 10, 100, 100, 1,
                            BlackPixel(d, s), WhitePixel(d, s));
    XSelectInput(d, w, ExposureMask | KeyPressMask);
    XMapWindow(d, w);

    while (1) {
        XNextEvent(d, &e);
        if (e.type == Expose) {
            XFillRectangle(d, w, DefaultGC(d, s), 20, 20, 10, 10);
            XDrawString(d, w, DefaultGC(d, s), 10, 50, msg, strlen(msg));
        }
        if (e.type == KeyPress)
            break;
    }

    XCloseDisplay(d);
    return 0;
}
```
Windows and Events

• Creating windows and dealing with events varies from platform to platform.

• Some packages provide implementations for key platforms (Windows, Unix, Mac) and abstractions for dealing with windows and events.

• GLUT: library for cross-platform windowing & events.
  - My experiments: doesn’t work as well as it used to.

• VTK: library for visualization
  - But also contains cross-platform windowing & events.
Visualization with VTK

Content from: Erik Vidholm, Univ of Uppsula, Sweden
David Gobbi, Robarts Research Institute, London, Ontario, Canada
VTK – The Visualization ToolKit

• Open source, freely available software for 3D computer graphics, image processing, and visualization
• Managed by Kitware Inc.
• Use C++, Tcl/Tk, Python, Java
The visualization pipeline

DATA → FILTER → MAPPING → DISPLAY

Visualization algorithms

Interactive feedback
We will replace these and write our own GL calls.

We will re-use these.

diagram:

```
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()
```
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!

Borrow  Build
OpenGL Functions

- Primitives
  - Points
  - Line Segments
  - Polygons
- Attributes
- Transformations
  - Viewing
  - Modeling
- Control (VTK)
- Input (VTK)
- Query

Today
later this week
First OpenGL programs

- Remember: none of these programs have windowing or events
- They contain just the code to put primitives on the screen, with lighting and colors.
First OpenGL programs

class vtk441PolyDataMapper : public vtkOpenGLPolyDataMapper
{
    public:
        static vtk441PolyDataMapper *New();
        virtual void RenderPiece(vtkRenderer *ren, vtkActor *act)
        {
            float ambient[3] = { 1, 1, 1 };
            glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, ambient);
            glBegin(GL_TRIANGLES);
            glVertex3f(0, 0, 0);
            glVertex3f(0, 1, 0);
            glVertex3f(1, 1, 0);
            glEnd();
        }
};
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);
      glVertex3f(0,1,0);
      glVertex3f(1,1,0);
      glEnd();
    }
};
Both glEnable and glDisable take a single argument, \textit{cap}, which can assume one of the following values:

- \texttt{GL_BLEND}
  
  If enabled, blend the fragment color values with the values in the color buffers. See \texttt{glBlendFunc}.

- \texttt{GL_CULL_FACE}
  
  If enabled, cull polygons based on their winding in window coordinates. See \texttt{glCullFace}.

- \texttt{GL_DEPTH_TEST}
  
  If enabled, do depth comparisons and update the depth buffer. Note that even if the depth buffer exists and the depth mask is non-zero, the depth buffer is not updated if the depth test is disabled. See \texttt{glDepthFunc} and \texttt{glDepthRange}.

- \texttt{GL_DITHER}
  
  If enabled, dither color components or indices before they are written to the color buffer.

- \texttt{GL_POLYGON_OFFSET_FILL}
  
  If enabled, an offset is added to depth values of a polygon's fragments produced by rasterization. See \texttt{glPolygonOffset}.

- \texttt{GL_SAMPLE_ALPHA_TO_COVERAGE}
  
  If enabled, compute a temporary coverage value where each bit is determined by the alpha value at the corresponding sample location. The temporary coverage value is then ANDed with the fragment coverage value.

- \texttt{GL_SAMPLE_COVERAGE}
  
  If enabled, the fragment's coverage is ANDed with the temporary coverage value. If \texttt{GL_SAMPLE_COVERAGE_INVERT} is set to \texttt{GL_TRUE}, invert the coverage value. See \texttt{glSampleCoverage}.

- \texttt{GL_SCISSOR_TEST}
  
  If enabled, discard fragments that are outside the scissor rectangle. See \texttt{glScissor}.

- \texttt{GL_STENCIL_TEST}
  
  If enabled, do stencil testing and update the stencil buffer. See \texttt{glStencilFunc} and \texttt{glStencilOp}.
First OpenGL programs

```cpp
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();
    }
};
```
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
Textures

- “Textures” are a mechanism for adding “texture” to surfaces.
  - Think of texture of a cloth being applied to a surface
  - Typically used in 2D form

- We will start with a 1D form, and work our way up to 2D later.
1D textures: basic idea

- Store color map on GPU as a texture
  - An array of colors

- Old color interpolation of fragment on a scanline:
  - For (int $j = 0 ; j < 3 ; j++$)
    $$\text{RGB}[j] = \text{leftRGB}[j] + \text{proportion} \times (\text{rightRGB}[j] - \text{leftRGB}[j])$$

- New color interpolation of fragment on a scanline:
  - $\text{textureVal} = \text{leftTextureVal}$
    $$+ \text{proportion} \times (\text{rightTextureVal} - \text{leftTextureVal})$$

  - $\text{RGB} \leftarrow \text{textureLookup}[\text{textureVal}]$
Example

- Triangle with vertices with scalar values 2.9, 3.3, and 3.1.
- $T$ for 2.9 = (2.9-3.0)/(3.25-3) = -0.4
- $T$ for 3.1 = (3.1-3.0)/(3.25-3) = 0.4
- $T$ for 3.3 = (3.3-3.0)/(3.25-3) = 1.2
- Fragment colors come from interpolating texture coordinates and applying texture
First OpenGL Texture Program

```cpp
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();
    }
};
```

(advanced texture features & 2D textures on Weds)
Project 2A
We will replace these and write our own GL calls.

We will re-use these.

```
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()
```
Project #2A (8%), Due Nov. 7th

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

- MAKE MANY BACKUPS OF YOUR PROGRAM

- If the program doesn’t run with VTK 7, use VTK 6

- If you are having issues on your laptop with a GL program, then use Room 100
  - (There’s only 2 of these projects)