Computer Graphics (Spring 2008)

COMS 4160, Lecture 9: OpenGL 1

http://www.cs.columbia.edu/~cs4160

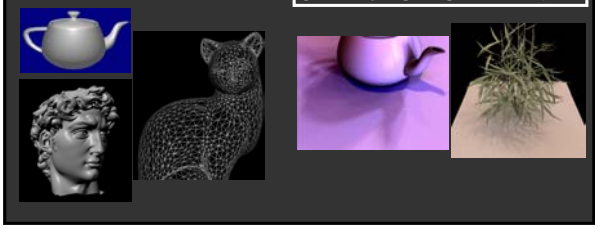
To Do

- Start thinking (now) about HW 3. Milestones are due soon.

Course Outline

- 3D Graphics Pipeline

Modeling
(Creating 3D Geometry) → **Rendering**
(Creating, shading images from geometry, lighting, materials)



Course Outline

- 3D Graphics Pipeline

Modeling
(Creating 3D Geometry) → **Rendering**
(Creating, shading images from geometry, lighting, materials)

Unit 1: Transformations
Weeks 1,2. Ass 1 due Feb 14

Unit 2: Spline Curves
Weeks 3,4. Ass 2 due Feb 26

Unit 3: OpenGL
Weeks 5-7.
Ass 3 due ??

Midterm on units 1-3: Mar 10

Demo: Surreal (HW 3)

Methodology for Lecture

- This unit different from others in course
  - Other units stress mathematical understanding
  - This stresses implementation details and programming
- I am going to show (maybe write) actual code
  - Same code (with comments) available online to help you understand how to implement basic concepts
  - I hope the online code helps you understand HW 3 better
  - ASK QUESTIONS if confused!!
- Simple demo 4160-opengl\opengl1\opengl1-orig.exe
  - This lecture deals with very basic OpenGL setup. Next 2 lectures will likely be more interesting

Outline

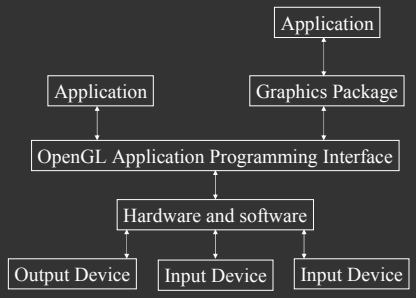
- *Basic idea about OpenGL*
- Basic setup and buffers
- Matrix modes
- Window system interaction and callbacks
- Drawing basic OpenGL primitives

Best source for OpenGL is the redbook. Of course, this is more a reference manual than a textbook, and you are better off implementing rather reading end to end. Though if you do have time, the book is actually quite readable

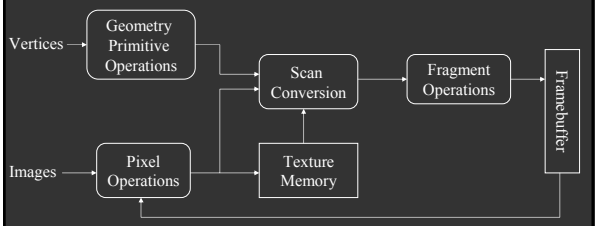
Introduction to OpenGL

- OpenGL is a graphics *API*
  - Software library
  - Layer between programmer and graphics hardware (and software)
- OpenGL can fit in many places
  - Between application and graphics system
  - Between higher level API and graphics system

Programmer's View



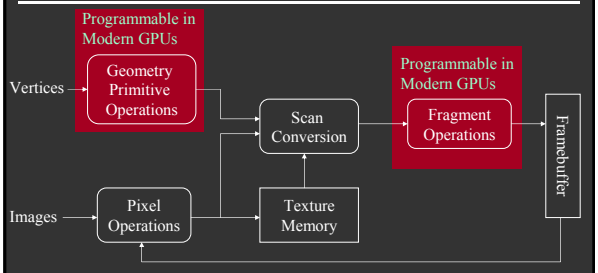
OpenGL Rendering Pipeline



Many operations controlled by state (projection matrix, transformation matrix, color etc.)

OpenGL is a large state machine

OpenGL Rendering Pipeline



Traditional Approach: Fixed function pipeline (state machine)
New Development (2003-): Programmable pipeline

GPUs and Programmability

- Since 2003, can write vertex/pixel shaders
- Fixed function pipeline special type of shader
- Like writing C programs (see back of OpenGL book)
- Performance >> CPU (even used for non-graphics)

GPUs and Programmability

- Since 2003, can write vertex/pixel shaders
- Fixed function pipeline special type of shader
- Like writing C programs (see back of OpenGL book)
- Performance >> CPU (even used for non-graphics)
- But parallel paradigm
  - All pixels/vertices operate in parallel
  - Severe performance overheads for control flow, loops (limitations beginning to be relaxed in modern releases)
- Not directly covered in COMS 4160
  - But you can make use of in assignments for extra credit

Why OpenGL?

- Fast
- Simple
- Window system independent
- Supports some high-end graphics features
- Geometric *and* pixel processing
- Standard, available on many platforms

Outline

- Basic idea about OpenGL
- *Basic setup and buffers*
- Matrix modes
- Window system interaction and callbacks
- Drawing basic OpenGL primitives

Buffers and Window Interactions

- Buffers: Color (front, back, left, right), depth (z), accumulation, stencil. When you draw, you write to some buffer (most simply, front and depth)
- No window system interactions (for portability)
  - But can use GLUT (or Motif, GLX, Tcl/Tk)
  - Callbacks to implement mouse, keyboard interaction

Basic setup code (you will likely copy)

```
int main(int argc, char** argv)
{
   glutInit(&argc, argv);

   // Requests the type of buffers (Single, RGB).
   // Think about what buffers you would need...
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);

   glutInitWindowSize (500, 500);
   glutInitWindowPosition (100, 100);
   glutCreateWindow ("Simple Demo");
   init (); // Always initialize first

   // Now, we define callbacks and functions for various tasks.
   glutDisplayFunc(display);
   glutReshapeFunc(reshape) ;
   glutKeyboardFunc(keyboard);
   glutMouseFunc(mouse) ;
   glutMotionFunc(mousedrag) ;

   glutMainLoop(); // Start the main code
   return 0;   /* ANSI C requires main to return int. */
}
```

Outline

- Basic idea about OpenGL
- Basic setup and buffers
- *Matrix modes*
- Window system interaction and callbacks
- Drawing basic OpenGL primitives

Viewing in OpenGL

- Viewing consists of two parts
  - Object positioning: *model view* transformation matrix
  - View projection: *projection* transformation matrix
- OpenGL supports both perspective and orthographic viewing transformations
- OpenGL's camera is always at the origin, pointing in the –*z* direction
- Transformations move objects relative to the camera
- Matrices right-multiply top of stack.
  (Last transform in code is first actually applied)

Basic initialization code

```
#include <GL/glut.h>
#include <stdlib.h>

int mouseoldx, mouseoldy ; // For mouse motion
GLdouble eyeloc = 2.0 ; // Where to look from; initially 0 -2, 2


void init (void)
{
/* select clearing color      */
   glClearColor (0.0, 0.0, 0.0, 0.0);

/* initialize viewing values  */
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();

   // Think about this.  Why is the up vector not normalized?
   glMatrixMode(GL_MODELVIEW) ;
   glLoadIdentity() ;
   gluLookAt(0,-eyeloc,eyeloc,0,0,0,0,1,1) ;
}
```

Outline

- Basic idea about OpenGL
- Basic setup and buffers
- Matrix modes
- *Window system interaction and callbacks*
- Drawing basic OpenGL primitives

Window System Interaction

- Not part of OpenGL
- Toolkits (GLUT) available
- Callback functions for events
  - Keyboard, Mouse, etc.
  - Open, initialize, resize window
  - Similar to other systems (X, Java, etc.)
- Our main func included

```
glutDisplayFunc(display);
glutReshapeFunc(reshape) ;
glutKeyboardFunc(keyboard);
glutMouseFunc(mouse) ;
glutMotionFunc(mousedrag) ;
```

Basic window interaction code

```
/* Defines what to do when various keys are pressed */
void keyboard (unsigned char key, int x, int y)
{
  switch (key) {
  case 27:  // Escape to quit
    exit(0) ;
    break ;
  default:
    break ;
  }
}

/* Reshapes the window appropriately */
void reshape(int w, int h)
{
   glViewport (0, 0, (GLsizei) w, (GLsizei) h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();

   gluPerspective(30.0, (GLdouble)w/(GLdouble)h, 1.0, 10.0) ;
}
```

Mouse motion (demo 4160-opengl1\opengl1-orig.exe)

```
/* Defines a Mouse callback to zoom in and out */
/* This is done by modifying gluLookAt         */
/* The actual motion is in mousedrag           */
/* mouse simply sets state for mousedrag       */
void mouse(int button, int state, int x, int y)
{
  if (button == GLUT_LEFT_BUTTON) {
    if (state == GLUT_UP) {
      // Do Nothing ;
    }
    else if (state == GLUT_DOWN) {
      mouseoldx = x ; mouseoldy = y ; // so we can move wrt x , y
    }
  }
  else if (button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
    { // Reset gluLookAt
      eyeloc = 2.0 ;
      glMatrixMode(GL_MODELVIEW) ;
      glLoadIdentity() ;
      gluLookAt(0,-eyeloc,eyeloc,0,0,0,0,1,1) ;
      glutPostRedisplay() ;
    }
}
```

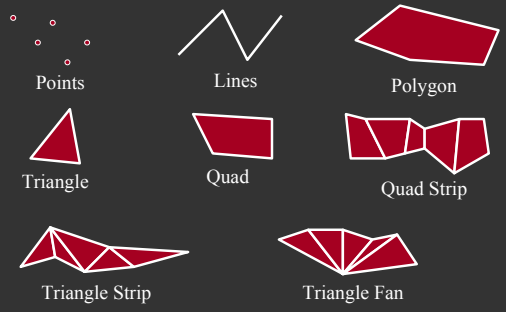
Mouse drag (demo 4160-opengl1\opengl1-orig.exe)

```
void mousedrag(int x, int y) {
  int yloc = y - mouseoldy   ;    // We will use the y coord
to zoom in/out
  eyeloc  += 0.005*yloc ;         // Where do we look from
  if (eyeloc < 0) eyeloc = 0.0 ;
  mouseoldy = y ;

  /* Set the eye location */
  glMatrixMode(GL_MODELVIEW) ;
  glLoadIdentity() ;
  gluLookAt(0,-eyeloc,eyeloc,0,0,0,0,1,1) ;

  glutPostRedisplay() ;
}
```

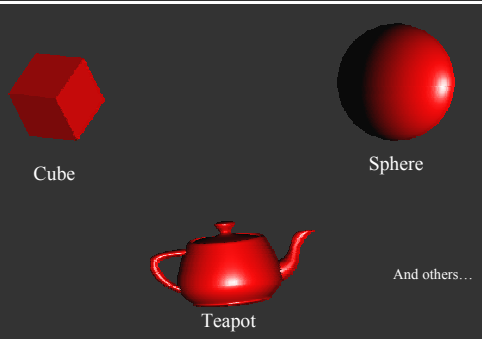
Outline

- Basic idea about OpenGL
- Basic setup and buffers
- Matrix modes
- Window system interaction and callbacks
- *Drawing basic OpenGL primitives*

OpenGL Primitives



GLUT 3D Primitives



Drawing idea

- Enclose vertices between glBegin() … glEnd() pair
  - Can include normal C code and attributes like the colors of points, but not other OpenGL commands
  - Inside are commands like glVertex3f, glColor3f
  - Attributes must be set ***before*** the vertex
- Assembly line model (pass vertices, transform, clip, shade)
- Client-Server model (client generates vertices, server draws) even if on same machine
  - glFlush() forces client to send network packet
  - glFinish() waits for ack, sparingly use synchronization

Geometry

- Points (GL_POINTS)
  Stored in Homogeneous coordinates
- Line segments (GL_LINES)
- Polygons
  - Simple, convex (take your chances with concave)
  - Tessellate, GLU for complex shapes
  - Rectangles: glRect
- Special cases (strips, loops, triangles, fans, quads)
- More complex primitives (GLUT): Sphere, teapot, cube,…

Specifying Geometry

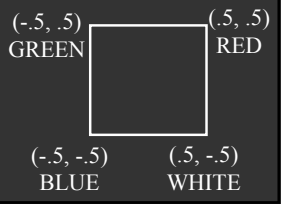
glBegin(GL_POLYGON) ; // Chapter 2 but I do Counter Clock W

- glVertex2f (4.0, 0.0) ;
- glVertex2f (6.0, 1.5) ;
- glVertex2f (4.0, 3.0) ;
- glVertex2f (0.0, 3.0) ;
- glVertex2f (0.0, 0.0) ;
- // glColor, glIndex, glNormal, glTexCoord, … (pp 47)
- // glMaterial, glArrayElement, glEvalCoord, … (pp 48)
- // Other GL commands invalid between begin and end
- // Can write normal C code…

glEnd() ;

Drawing in Display Routine

```
void display(void)
{
   glClear (GL_COLOR_BUFFER_BIT);

   // draw polygon (square) of unit length centered at the origin
   // This code draws each vertex in a different color.
   // The hardware will blend between them.
   // This is a useful debugging trick.  I make sure each vertex
   // appears exactly where I expect it to appear.

   glBegin(GL_POLYGON);
      glColor3f (1.0, 0.0, 0.0);
      glVertex3f (0.5, 0.5, 0.0);
      glColor3f (0.0, 1.0, 0.0);
      glVertex3f (-0.5, 0.5, 0.0);
      glColor3f (0.0, 0.0, 1.0);
      glVertex3f (-0.5, -0.5, 0.0);
      glColor3f (1.0, 1.0, 1.0);
      glVertex3f (0.5, -0.5, 0.0);
   glEnd();
   glFlush () ;
}
```



Demo (change colors)