Image Rendering

Lecture 7

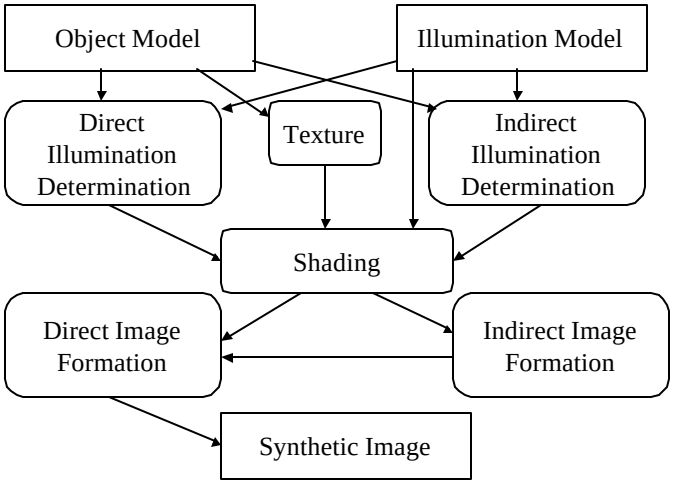
Rendering can be divided into three sub-problems

- Image Formation
  - "The hidden surface problem"
  - visibility determination steps
- Illumination
  - Direct illumination
  - Indirect illumination
- Shading
  - Process of determining the distribution of light leaving a surface

1

Image Rendering

Lecture 7



2

Visibility Determination

Lecture 7

Given.

- A set of surfaces in three dimensions
- A viewpoint
- An oriented image plane
- A field of view

For each point on the image plane within the field of view

Determine which point on the surface is closest to the viewpoint along a line form the viewpoint through the point in the image plane

3

Visibility Determination

Lecture 7

There are two basic classes of algorithms to solve this problem:

- Continues algorithms

  Performing visibility determination over continuous areas covering the entire image plane. Every visible piece of each surface will be detected regardless of location or size.

- Point sampling algorithms

  Forms an approximate solution to the visible surface problem. These algorithms determine visibility only at a finite number of sample points and make assumptions about the visibility of objects between sample points.

4

Visibility Determination

Lecture 7

Continuous Algorithms:
- Area/volume subdivision
- scan-plane

Point Sampling Algorithms:
- Z-buffer
- Ray Tracing
- Painter's
- Scan-line

Other Algorithms:
- wavefront propagation

5

Point Sampling Algorithms

Lecture 7

Scan-Line
```
sort objects by y, for all y {
    sort objects by x, for all x {
        compare z
    }
}
```

Painter's
```
sort objects by z, for all objects {
    for all covered pixels(x,y) {
        paint
    }
}
```

Z-Buffer
```
for all objects {
    for all covered pixels {
        compare z
    }
}
```

Ray Tracing
```
for all pixels (x,y) {
    for all objects {
        compare z
    }
}
```

6

OpenGL

Lecture 7

A Visual Introduction to OpenGL Programming

Siggraph 1998 - Course 7

Mason Woo
Dave Shreiner

What is OpenGL?

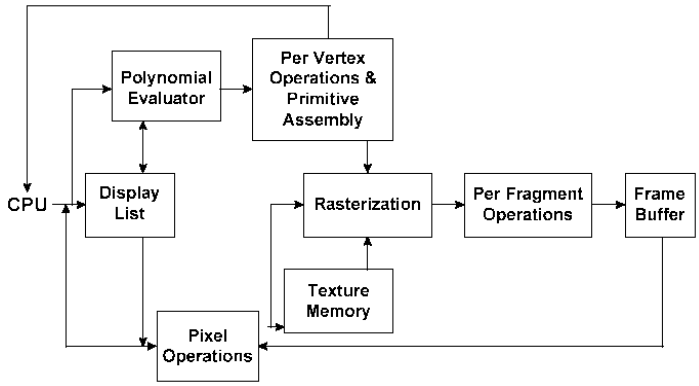
Lecture 7

A graphics rendering library API to produce high-quality, color images from geometric and raster primitives Window System and Operating System
independent.

OpenGL “doesn’t do windows”

OpenGL Architecture

Lecture 7



9

OpenGL as a Renderer

Lecture 7

Renders simple geometric primitives
- points, lines, polygons

Renders images and bitmaps
- separate pipelines for geometry and pixels, linked through texture mapping

Rendering depends on state
- colors, light sources, materials
- surface normals, texture

10

OpenGL and Related APIs

Lecture 7

GLU (OpenGL Utility Library)

- guaranteed to be available
- tesselators, quadrics, NURBs, etc.
- some surprisingly common operations, such as projection transformations (such as gluPerspective)

GLX or WGL

- bridge between window system and OpenGL

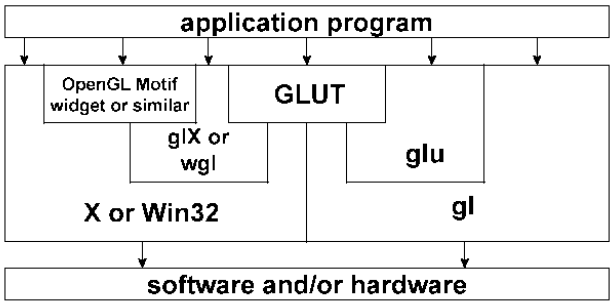
GLUT

- portable bridge between window system and OpenGL
- not “standard”, but informal popularity

11

OpenGL and Related APIs

Lecture 7



12

Program Structure

Lecture 7

Initialize visual & open window
Initialize states & display lists
Register display callback function
- clear screen, change states, render graphics, swap buffers...

Register reshape callback function
- modify clipping, viewing

Register input device (mouse, keybd) callback functions
Register idle callback function (the keep busy operation)
Enter main loop
- if contents need to be redrawn, display callback called
- if window resized, reshape callback called
- if input event, appropriate input callback function called
- if nothing happening, idle callback function called

13

Preliminaries

Lecture 7

Header files

```
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h> /* see note! */
```

Link with graphics libraries

```
cc prog.c -lglut -lGLU -lGL -lX11 -lXmu -o prog
cl proc.c glut32.lib glu32.lib opengl32.lib \
gdi32.lib user32.lib
```

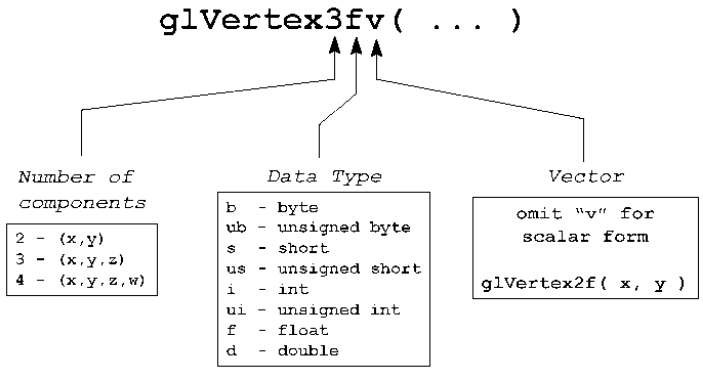
GL enumerated types
- for platform independence

GLbyte, GLshort, GLushort, GLint, GLuint, GLsizei, GLfloat, GLdouble, GLclampf, GLclampd, GLubyte, GLboolean, GLenum, GLbitfield

14

OpenGL Command Syntax

Lecture 7



A Simple Example Program

Lecture 7

```
#include <GL/glut.h>
void display(void) {
      glClear(GL_COLOR_BUFFER_BIT);
      glColor3f(0.0, 1.0, 1.0); /* cyan */
      glBegin(GL_QUADS);
      glVertex2i(100,100);
      glVertex2i(200, 100);
      glVertex2i(200,300);
      glVertex2i(100, 300);
      glEnd();
      glFlush();
}
void gfxinit(void) {
      glClearColor(0.0, 0.0, 0.0, 0.0);
}
```

A Simple Example Program


```
void reshape(int width, int height) {
      glViewport(0, 0, width, height)
      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      gluOrtho2D(0.0, (GLdouble) width, 0.0,
      (GLdouble) height);
}
void main(int argc, char **argv){
      glutInit(&argc, argv);
      glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
      win = glutCreateWindow("rect");
      glutReshapeFunc(reshape);
      glutDisplayFunc(display);
      gfxinit();
      glutMainLoop();
}
```


Error Handling

GLenum glGetError(void)

• Have an error handling routine

• Call it every time in ***display()***

```
GLenum errCode;
const GLubyte *errString;
if ((errCode = glGetError()) != GL_NO_ERROR) {
     errString = gluErrorString(errCode);
     fprintf (stderr, “OpenGL Error: %s\n”, errString);
}
```

• If ***GL_NO_ERROR*** returned, great!

Elementary Rendering

Lecture 7

Geometric Primitives

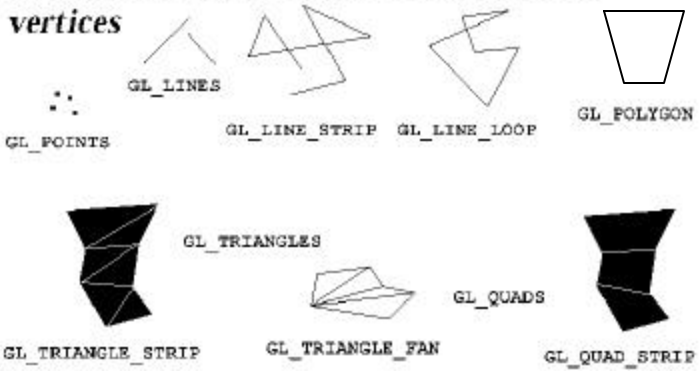
Managing OpenGL State

OpenGL Buffers

OpenGL Geometric Primitives

Lecture 7

All geometric primitives are specified by vertices



Specifying Geometric Primitives

Primitives are specified using

glBegin(*primType*);

glEnd();

• primType determines how vertices are combined

```
GLfloat r, g, b;
Glfloat coords[3];
glBegin(  primType );
  for ( i = 0; i < nVerts; ++i ) {
      glColor3f( red, green,blue);
      glVertex3fv( coords );
  }
glEnd();
```


Simple Example


```
void drawRhombus( GLfloat color[] )
{
    glColor3fv( color );
    glBegin( GL_QUADS );
        glVertex2f( 0.0, 0.0 );
        glVertex2f( 1.0, 0.0 );
        glVertex2f( 1.5, 1.118 );
        glVertex2f( 0.5, 1.118 );
    glEnd();
}
```

Advanced Primitives

Lecture 7

Vertex Arrays
Bernstein Polynomial Evaluators
- basis for GLU Nurbs

GLU Quadric Objects
- sphere
- cylinder
- disk

Vertex Arrays

Lecture 7

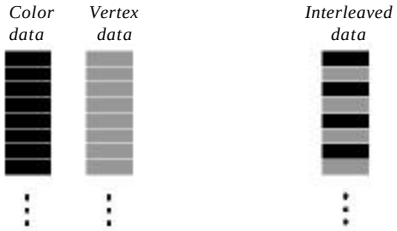
Pass arrays of vertices, colors, etc. to OpenGL in a large chunk

glVertexPointer*(3, GL_FLOAT, 0, coords)*
glColorPointer*(4, GL_FLOAT, 0, colors)*
glEnableClientState*(GL_VERTEX_ARRAY)*
glEnableClientState*(GL_COLOR_ARRAY)*

All active arrays are used in rendering

Combined Vertex Data

Combine all vertex data into a single array
glInterleavedArrays(GL_C3F_V3F, 0, data);



Rendering with Vertex Arrays

Render arrays sequentially

```
glDrawArrays( GL_TRIANGLE_STRIP, 0, numVerts );
```

Automatically index into arrays

```
Gluint indices[] = { 0, 2, 1, 3, 2, 3, 6, 5 };
glDrawElements( GL_QUADS, 2, GL_UNSIGNED_INT,
indices );
```

Manually index into arrays

```
glBegin( GL_LINES );
  for ( i = 0; i < numLines; ++i ) {
     glArrayElement( leftEnd[i] );
     glArrayElement( rightEnd[i] );
  }
glEnd();
```

OpenGL's State Machine

All rendering attributes are encapsulated in the OpenGL State

- rendering styles
- shading
- lighting
- texture mapping

Manipulating OpenGL State

Appearance is controlled by current state

```
foreach( primitive to render ) {
    update OpenGL state
    render primitive
}
```

Manipulating vertex attributes is most common way to manipulate state

```
glColor*() / glIndex*()
glNormal*()
glTexCoord*()
```

Controlling current state

Lecture 7

Setting State

```
glPointSize( size );
glLineStipple( repeat, pattern );
glMaterialfv( GL_FRONT, GL_DIFFUSE,
color );
```

Enabling Features

```
glEnable( GL_LIGHTING );
glDisable( GL_TEXTURE_2D );
```

29

Transformations

Lecture 7

Prior to rendering, view, locate, and orient:

- eye/camera position

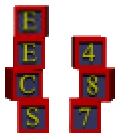
- 3D geometry

Manage the matrices

- including matrix stack

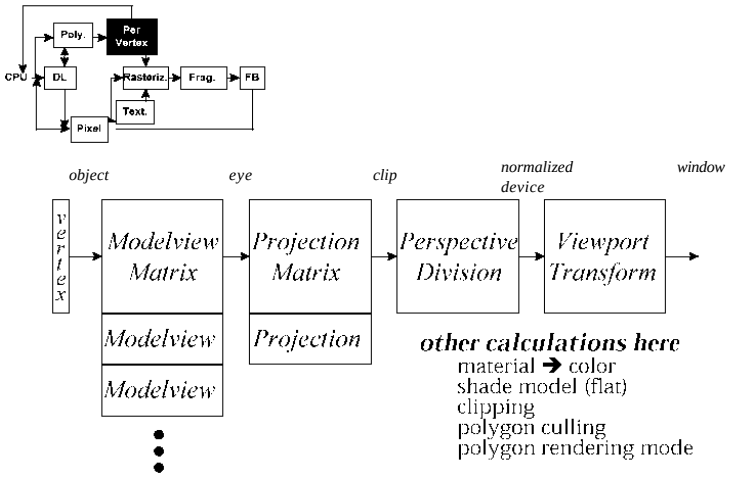
Combine (composite) transformations

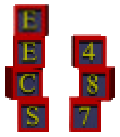
30



Transformation Pipeline

Lecture 7





Matrix Operations

Lecture 7

Specify Current Matrix Stack

glMatrixMode (GL_MODELVIEW or GL_PROJECTION)

Other Matrix or Stack Operations

glLoadIdentity()
glPushMatrix()
glPopMatrix()

Viewport

• usually same as window size
• viewport aspect ratio should be same as projection transformation or resulting image may be distorted

glViewport(x, y, width, height)

Projection Transformation

Lecture 7

•Shape of viewing frustum

•Perspective projection
gluPerspective (fovy, aspect, zNear, zFar)
glFrustum (left, right, bottom, top, zNear, zFar)

•Orthographic parallel projection
glOrtho (left, right, bottom, top, zNear, zFar)
gluOrtho2D (left, right, bottom, top)
– calls glOrtho with z values near zero

33

Viewing Transformations

Lecture 7

Position the camera/eye in the scene
• place the tripod down; aim camera

To "fly through" a scene
• change viewing transformation and redraw scene

gluLookAt(eyex, eyey, eyez, aimx, aimy, aimz, upx, upy, upz)
• up vector determines unique orientation
• careful of degenerate positions

34

Modeling Transformations

Lecture 7

Move object

glTranslate{fd}(x, y, z)

Rotate object around arbitrary axis (x, y, z)

glRotate{fd}(angle, x, y, z)

- angle is in degrees

Dilate (stretch or shrink) or mirror object

glScale{fd}(x, y, z)

35

Projection is left handed

Lecture 7

Projection transformations (*gluPerspective, glOrtho*) are left handed

- think of zNear and zFar as distance from view point

Everything else is right handed, including the vertexes to be rendered

36

Modeling in OpenGL

Lecture 7

- All Geometric Primitives are Eventually Described in Terms of Verticies
- All Verticies are 3-D and Described by 4 Coordinates (x, y, z, w)
  - If w is not specified, w=1
  - If w!= 0 ==> (x/w, y/w, z/w, 1)
- Restrictions on OpenGL Polygons
  - Polygons MUST be *Simple*
    - Edges intersect ONLY at verticies
    - Exactly two edges meet at ant vertex
  - Polygons MUST be *Convex*
    - Given any two interior points, the line segment joinging them is also interior

Modeling in OpenGL

Lecture 7

- Restrictions on OpenGL Polygons
  - Polygons may NOT have Holes
  - Polygons may have any number of edges
- Tesselation Polygons
- Restrictions Allow Fast Polygon Rendering in Hardware

Modeling in OpenGL

- After Transformations, Non-planer Polygons May No Longer Be Simple
  - Not unusual when curved surfaces are approximated with quadrilateral polygons
  - Avoid problem by using triangular facets