

CSCI 480 Computer Graphics
Lecture 2

Basic Graphics Programming

Graphics Pipeline
OpenGL API
Primitives: Lines, Polygons
Attributes: Color
Example

January 12, 2011

Jernej Barbic

University of Southern California

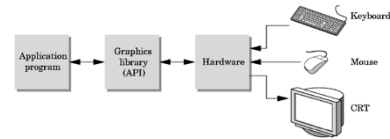
[Angel Ch. 2]

<http://www-bcf.usc.edu/~jbarbic/cs480-s11/>

1

What is OpenGL

- A low-level graphics API for 2D and 3D interactive graphics.
- Descendent of GL (from SGI)



2

OpenGL is cross-platform

- Same code works with little/no modifications
- Implementations:
Windows, Mac: ships with the OS
Linux: Mesa, a freeware implementation.

```

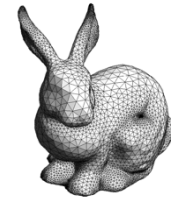
#if defined(WIN32) || defined(linux)
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>
#elif defined(__APPLE__)
#include <OpenGL/gl.h>
#include <OpenGL/glu.h>
#include <GLUT/glut.h>
#endif
  
```

3

How does OpenGL work

From the programmer's point of view:

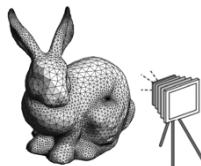
1. Specify geometric objects
2. Describe object properties
 - Color
 - How objects reflect light



4

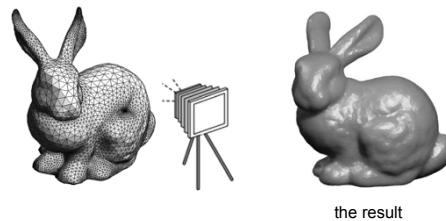
How does OpenGL work (continued)

3. Define how objects should be viewed
 - where is the camera
 - what type of camera
4. Move camera or objects around for animation



5

The result



6

OpenGL is a state machine

State variables: color, current viewing position, line width, material properties...

These variables (the *state*) then apply to every subsequent drawing command

They persist until set to new values by the programmer

7

OpenGL Library Organization

- GL (Graphics Library): core graphics capabilities
- GLU (OpenGL Utility Library): utilities on top of GL
- GLUT (OpenGL Utility Toolkit): input and windowing

```

    graph LR
      A[OpenGL application program] --> B[GLU]
      A --> C[GL]
      A --> D[GLUT]
      A --> E[GLX]
      B --> C
      C --> F[Frame buffer]
      D --> G[Xlib, Xtk]
      E --> G
      G --> F
    
```

8

Graphics Pipeline

Vertices	Transform	Clipper	Projector	Rasterizer	Pixels
Primitives+ material properties	Translate Rotate Scale	Is it visible on screen?	3D to 2D	Convert to pixels	Shown on the screen (framebuffer)

9

Immediate-mode rendering

- Application generates stream of geometric primitives (polygons, lines)
- System draws each one into the framebuffer
- Entire scene redrawn anew every frame
- Contrast: off-line rendering (e.g., Pixar Renderman)

10

The pipeline is implemented by OpenGL, graphics driver and the graphics hardware

OpenGL programmer does not need to implement the pipeline.

However, pipeline is reconfigurable if needed
→ "shaders"

11

Graphics Pipeline

- Efficiently implementable in hardware (but not in software)
- Each stage can employ multiple specialized processors, working in parallel, busses between stages
- #processors per stage, bus bandwidths are fully tuned for typical graphics use
- Latency vs throughput

12

Vertices

```

    Vertices → Transformer → Clipper → Projector → Rasterizer → Pixels
    
```

- Vertices in world coordinates
- void glVertex3f(GLfloat x, GLfloat y, GLfloat z)
 - Vertex (x, y, z) sent down the pipeline
 - Function call returns
- Use GL for portability and consistency
- glVertex{234}{sfd}[v](coords)

13

Transformer

```

    Vertices → Transformer → Clipper → Projector → Rasterizer → Pixels
    
```

- Transformer in world coordinates
- Must be set before object is drawn!


```

                glRotatef(45.0, 0.0, 0.0, -1.0);
                glVertex2f(1.0, 0.0);
            
```
- Complex [Angel Ch. 4]

14

Clipper

```

    Vertices → Transformer → Clipper → Projector → Rasterizer → Pixels
    
```

- Mostly automatic (must set viewport)

(a) (b)

15

Projector

```

    Vertices → Transformer → Clipper → Projector → Rasterizer → Pixels
    
```

- Complex transformation [Angel Ch. 5]

Orthographic

Perspective

16

Rasterizer

```

    Vertices → Transformer → Clipper → Projector → Rasterizer → Pixels
    
```

- Interesting algorithms [Angel Ch. 7]
- To window coordinates
- Antialiasing

17

Primitives

- Specified via vertices
- General schema


```

                glBegin(type);
                glVertex3f(x1, y1, z1);
                ...
                glVertex3f(xN, yN, zN);
                glEnd();
            
```
- *type* determines interpretation of vertices
- Can use glVertex2f(x,y) in 2D

18

Example: Draw Square Outline

- Type = GL_LINE_LOOP

```

glBegin(GL_LINE_LOOP);
glVertex3f(0.0, 0.0, 0.0);
glVertex3f(1.0, 0.0, 0.0);
glVertex3f(1.0, 1.0, 0.0);
glVertex3f(0.0, 1.0, 0.0);
glEnd();
    
```

GL_LINE_LOOP

19

Points and Line Segments

```

glBegin (GL_POINTS);
glVertex3f(...);
...
glVertex3f(...);
glEnd();
    
```

Draw points

20

Polygons

- Polygons enclose an area

GL_POINTS GL_POLYGON GL_QUADS GL_TRIANGLES

- Rendering of area (fill) depends on attributes
- All vertices must be in one plane in 3D

21

Polygon Restrictions

- OpenGL Polygons must be simple
- OpenGL Polygons must be convex

(a) simple, but not convex

(b) non-simple

(c) convex

22

Why Polygon Restrictions?

- Non-convex and non-simple polygons are expensive to process and render
- Convexity and simplicity is expensive to test
- Behavior of OpenGL implementation on disallowed polygons is "undefined"
- Some tools in GLU for decomposing complex polygons (tessellation)
- Triangles are most efficient

23

Polygon Strips

- Efficiency in space and time
- Reduces visual artefacts

GL_POINTS GL_TRIANGLE_STRIP GL_QUAD_STRIP

- Polygons have a front and a back, possibly with different attributes!

24

Attributes:
color, shading and reflection properties

- Part of the OpenGL state
- Set before primitives are drawn
- Remain in effect until changed!

25

Physics of Color

- Electromagnetic radiation
- Can see only tiny piece of the spectrum

26

Color Filters

- Eye can perceive only 3 basic colors
- Computer screens designed accordingly

27

Color Spaces

- RGB (Red, Green, Blue)
 - Convenient for display
 - Can be unintuitive (3 floats in OpenGL)
- HSV (Hue, Saturation, Value)
 - Hue: what color
 - Saturation: how far away from gray
 - Value: how bright
- Other formats for movies and printing

28

RGB vs HSV

Gimp Color Picker

29

Example: Drawing a shaded polygon

- Initialization: the "main" function

```

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize (500, 500);
    glutInitWindowPosition (100, 100);
    glutCreateWindow (argv[0]);
    init ();
    ...
}
    
```

30

GLUT Callbacks

- Window system independent interaction
- glutMainLoop processes events

```

...
glutDisplayFunc(display);
glutReshapeFunc(reshape);
glutKeyboardFunc (keyboard);
glutMainLoop();
return 0;
}
    
```

31

Initializing Attributes

- Separate in "init" function

```

void init(void)
{
    glClearColor (0.0, 0.0, 0.0, 0.0);

    /* glShadeModel (GL_FLAT); */
    glShadeModel (GL_SMOOTH);
}
    
```

32

The Display Callback

- The routine where you render the object
- Install with glutDisplayFunc(display)

```

void display(void)
{
    glClear (GL_COLOR_BUFFER_BIT); /* clear buffer */
    setupCamera();                /* set up the camera */
    triangle ();                  /* draw triangle */
    glutSwapBuffers ();           /* force display */
}
    
```

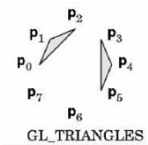
33

Drawing

- In world coordinates; remember state!

```

void triangle(void)
{
    glBegin (GL_TRIANGLES);
    glColor3f (1.0, 0.0, 0.0); /* red */
    glVertex2f (5.0, 5.0);
    glColor3f (0.0, 1.0, 0.0); /* green */
    glVertex2f (25.0, 5.0);
    glColor3f (0.0, 0.0, 1.0); /* blue */
    glVertex2f (5.0, 25.0);
    glEnd();
}
    
```



34

The Image

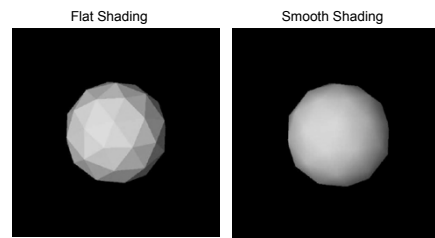
glShadeModel(GL_FLAT) glShadeModel(GL_SMOOTH)

color of last vertex each vertex separate color smoothly interpolated



35

Flat vs Smooth Shading



36

Projection

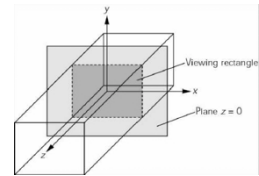
- Mapping world to screen coordinates

```
void reshape(int w, int h)
{
    glViewport (0, 0, (GLsizei) w, (GLsizei) h);
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity ();
    if (w <= h)
        gluOrtho2D (0.0, 30.0, 0.0, 30.0 * (GLfloat) h/(GLfloat) w);
    else
        gluOrtho2D (0.0, 30.0 * (GLfloat) w/(GLfloat) h, 0.0, 30.0);
    glMatrixMode(GL_MODELVIEW);
}
```

37

Orthographic Projection

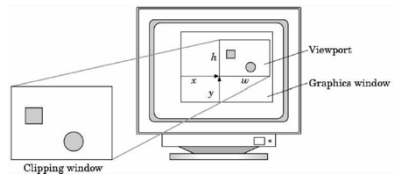
- 2D and 3D versions
- `glOrtho2D(left, right, bottom, top)`
- In world coordinates!



38

Viewport

- Determines clipping in window coordinates
- `glViewport(x, y, w, h)`



39

Summary

1. A Graphics Pipeline
2. The OpenGL API
3. Primitives: vertices, lines, polygons
4. Attributes: color
5. Example: drawing a shaded triangle



40