

## Objectives

- OpenGL Architecture
  - OpenGL as a state machine
- Functions
  - Types
  - Formats
- Simple program and refinements
- Fundamental OpenGL primitives
- Attributes
- Simple viewing



Slides by Edward Angel © 2002

## Early History of APIs

- IFIPS (1973) formed two committees to come up with a standard graphics API
  - Graphical Kernel System (GKS)
    - 2D but contained good workstation model
  - Core
    - Both 2D and 3D
  - GKS adopted as ISO and later ANSI standard (1980s)
- GKS not easily extended to 3D (GKS-3D)
- Far behind hardware development



Slides by Edward Angel © 2002

## PHIGS and X

- Programmers Hierarchical Graphics System (PHIGS)
  - Arose from CAD community
  - Database model with retained graphics (structures)
- X Window System
  - DEC/MIT effort
  - Client-server architecture with graphics
- PEX combined the two
  - Not easy to use (all the defects of each)



Slides by Edward Angel © 2002

## SGI and GL

- Silicon Graphics (SGI) revolutionized the graphics workstation by implementing the pipeline in hardware (1982)
- To use the system, application programmers used a library called GL
- With GL, it was relatively simple to program three dimensional interactive applications



Slides by Edward Angel © 2002

## OpenGL

- The success of GL lead to OpenGL (1992), a platform-independent API that was
  - Easy to use
  - Close enough to the hardware to get excellent performance
  - Focus on rendering
  - Omitted windowing and input to avoid window system dependencies



Slides by Edward Angel © 2002

## OpenGL Evolution

- Controlled by an Architectural Review Board (ARB)
  - Members include SGI, Microsoft, Nvidia, HP, 3DLabs, IBM, .....
  - Relatively stable (present version 1.4)
    - Evolution reflects new hardware capabilities
      - 3D texture mapping and texture objects
      - Vertex programs
  - Allows for platform specific features through extensions



Slides by Edward Angel © 2002



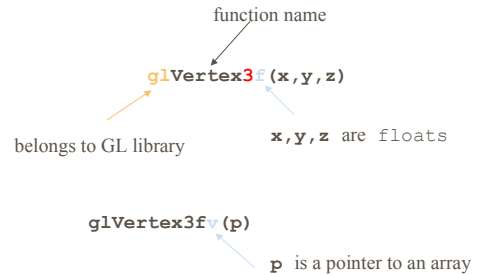
## Lack of Object Orientation

- OpenGL is not object oriented so that there are multiple functions for a given logical function, e.g. `glVertex3f`, `glVertex2i`, `glVertex3dv`,.....
- Underlying storage mode is the same
- Easy to create overloaded functions in C++ but issue is efficiency



Slides by Edward Angel © 2002

## OpenGL function format



Slides by Edward Angel © 2002

## OpenGL #defines

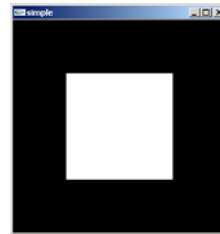
- Most constants are defined in the include files `gl.h`, `glu.h` and `glut.h`
  - Note `#include <glut.h>` should automatically include the others
  - Examples
    - `glBegin(GL_PLYGON)`
    - `glClear(GL_COLOR_BUFFER_BIT)`
- include files also define OpenGL data types: `Gfloat`, `Gdouble`,.....



Slides by Edward Angel © 2002

## A Simple Program

Generate a square on a solid background



Slides by Edward Angel © 2002

## simple.c

```
#include <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();
}
```



Slides by Edward Angel © 2002

## 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



Slides by Edward Angel © 2002

## 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



Slides by Edward Angel © 2002

## Notes on compilation

- See website and ftp for examples
- Unix/linux
  - Include files usually in `../include/GL`
  - Compile with `-lglut -lglu -lgl` loader flags
  - May have to add `-L` flag for X libraries
  - Mesa implementation included with most linux distributions
  - Check web for latest versions of Mesa and glut



Slides by Edward Angel © 2002

## Compilation on Windows

- Visual C++
  - Get `glut.h`, `glut32.lib` and `glut32.dll` from web
  - Create a console application
  - Add `opengl32.lib`, `glut32.lib`, `glut32.lib` to project settings (under link tab)
- Borland C similar
- Cygwin (linux under Windows)
  - Can use `gcc` and similar makefile to linux
  - Use `-lopengl32 -lglu32 -lglut32` flags



Slides by Edward Angel © 2002

## Program Structure

- Most OpenGL programs have a similar structure that consists of the following functions

- `main()`:
  - defines the callback functions
  - opens one or more windows with the required properties
  - enters event loop (last executable statement)
- `init()`: sets the state variables
  - viewing
  - Attributes
- callbacks
  - Display function
  - Input and window functions



Slides by Edward Angel © 2002

## Simple.c revisited

- In this version, we will see the same output but have defined all the relevant state values through function calls with the default values
- In particular, we set
  - Colors
  - Viewing conditions
  - Window properties



Slides by Edward Angel © 2002

## main.c

```
#include <GL/glut.h> // includes gl.h

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("simple"); // define window properties
    glutDisplayFunc(mydisplay); // display callback

    init(); // set OpenGL state

    glutMainLoop(); // enter event loop
}
```



Slides by Edward Angel © 2002

## GLUT functions

- `glutInit` allows application to get command line arguments and initializes system
- `glutInitDisplayMode` requests properties of the window (the *rendering context*)
  - RGB color, Single buffering, Properties logically OR'ed
- `glutWindowSize` in pixels
- `glutWindowPosition` from top-left corner of display
- `glutCreateWindow` create window with title "simple"
- `glutDisplayFunc` display callback
- `glutMainLoop` enter infinite event loop



Slides by Edward Angel © 2002

## init.c

```
void init()
{
    glClearColor (0.0, 0.0, 0.0, 1.0);
    glColor3f(1.0, 1.0, 1.0);

    glMatrixMode (GL_PROJECTION);
    glLoadIdentity ();
    glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
}
```

Annotations:  
- black clear color (points to 0.0, 0.0, 0.0)  
- opaque window (points to 1.0)  
- fill with white (points to 1.0, 1.0, 1.0)  
- viewing volume (points to glOrtho)



Slides by Edward Angel © 2002

## Coordinate Systems

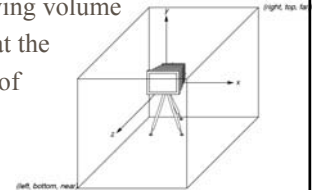
- The units of in `glVertex` are determined by the application and are called *world* or *problem coordinates*
- The viewing specifications are also in world coordinates and it is the size of the viewing volume that determines what will appear in the image
- Internally, OpenGL will convert to *camera coordinates* and later to *screen coordinates*



Slides by Edward Angel © 2002

## OpenGL Camera

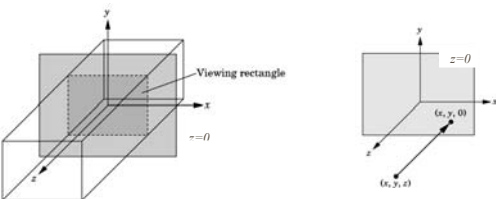
- OpenGL places a camera at the origin pointing in the negative  $z$  direction
- The default viewing volume is a box centered at the origin with a side of length 2



Slides by Edward Angel © 2002

## Orthographic Viewing

In the default orthographic view, points are projected forward along the  $z$  axis onto the plane  $z=0$



Slides by Edward Angel © 2002

## Transformations and Viewing

- In OpenGL, the projection is carried out by a projection matrix (transformation)
- There is only one set of transformation functions so we must set the matrix mode first  
`glMatrixMode (GL_PROJECTION)`
- Transformation functions are incremental so we start with an identity matrix and alter it with a projection matrix that gives the view volume  
`glLoadIdentity ();`  
`glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);`



Slides by Edward Angel © 2002

## Two- and three-dimensional viewing

- In `glOrtho(left, right, bottom, top, near, far)` the near and far distances are measured from the camera
- Two-dimensional vertex commands place all vertices in the plane  $z=0$
- If the application is in two dimensions, we can use the function `gluOrtho2D(left, right, bottom, top)`
- In two dimensions, the view or clipping volume becomes a *clipping window*



Slides by Edward Angel © 2002

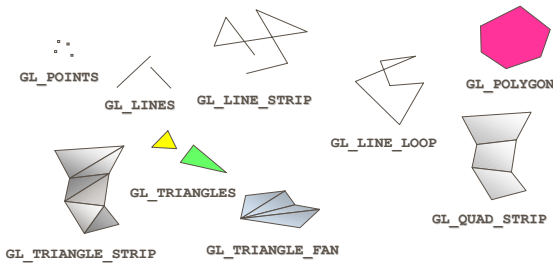
## mydisplay.c

```
void mydisplay()
{
    glClearColor(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();
}
```



Slides by Edward Angel © 2002

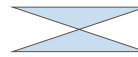
## OpenGL Primitives



Slides by Edward Angel © 2002

## Polygon Issues

- OpenGL will only display polygons correctly that are
  - Simple: edges cannot cross
  - Convex: All points on line segment between two points in a polygon are also in the polygon
  - Flat: all vertices are in the same plane
- User program must check if above true
- Triangles satisfy all conditions



nonsimple polygon



nonconvex polygon



Slides by Edward Angel © 2002

## Attributes

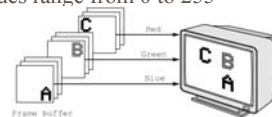
- Attributes are part of the OpenGL and determine the appearance of objects
  - Color (points, lines, polygons)
  - Size and width (points, lines)
  - Stipple pattern (lines, polygons)
  - Polygon mode
    - Display as filled: solid color or stipple pattern
    - Display edges



Slides by Edward Angel © 2002

## RGB color

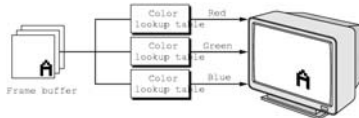
- Each color component stored separately in the frame buffer
- Usually 8 bits per component in buffer
- Note in `glColor3f` the color values range from 0.0 (none) to 1.0 (all), while in `glColor3ub` the values range from 0 to 255



Slides by Edward Angel © 2002

## Indexed Color

- Colors are indices into tables of RGB values
- Requires less memory
  - indices usually 8 bits
  - not as important now
    - Memory inexpensive
    - Need more colors for shading



Slides by Edward Angel © 2002

## Color and State

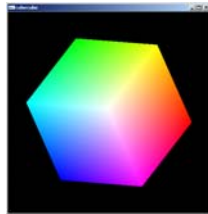
- The color as set by `glColor` becomes part of the state and will be used until changed
  - Colors and other attributes are not part of the object but are assigned when the object is rendered
- We can create conceptual *vertex colors* by code such as

```
glColor  
glVertex  
glColor  
glVertex
```

Slides by Edward Angel © 2002

## Smooth Color

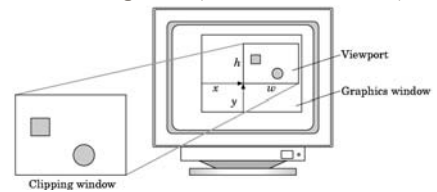
- Default is *smooth* shading
  - OpenGL interpolates vertex colors across visible polygons
- Alternative is *flat shading*
  - Color of first vertex determines fill color
- `glShadeModel`  
(`GL_SMOOTH`)  
OR `GL_FLAT`



Slides by Edward Angel © 2002

## Viewports

- Do not have use the entire window for the image: `glViewport(x, y, w, h)`
- Values in pixels (screen coordinates)



Slides by Edward Angel © 2002