Objectives

- Refine the first program
  - Alter the default values
  - Introduce a standard program structure
- Simple viewing
  - Two-dimensional viewing as a special case of three-dimensional viewing
- Fundamental OpenGL primitives
- Attributes
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
A Simple Program

Generate a square on a solid background

`simple.cpp`
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
In particular, we set

- Colors
- Viewing conditions
- Window properties
```c
#include <GL/glut.h>

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(0,0);
    glutCreateWindow("simple");
    glutDisplayFunc(mydisplay);
    init();
    glutMainLoop();
}
```
GLUT functions

- **glutInit** allows application to get command line arguments and initializes system
- **gluInitDisplayMode** requests properties for the window (the *rendering context*)
  - RGB color
  - Single buffering
  - Properties logically ORed together
- **glutWindowSize** in pixels
- **glutWindowPosition** from top-left corner of display
- **glutCreateWindow** create window with title “simple”
- **glutDisplayFunc** display callback
- **glutMainLoop** enter infinite event loop
```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);
}
```
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();
}
OpenGL Primitives

- **GL_POINTS**
- **GL_LINES**
- **GL_LINE_STRIP**
- **GL_LINE_LOOP**
- **GL_TRIANGLES**
- **GL_TRIANGLE_STRIP**
- **GL_TRIANGLE_FAN**
- **GL_POLYGON**

**GL_TRIANGLE_STRIP**
New point gives new triangle

**GL_TRIANGLE_FAN**
First point in all triangles

**GL_QUAD_STRIP**
Each pair of points adds a quad
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 can check if above true
  - OpenGL will produce output if these conditions are violated but it may not be what is desired
- Triangles satisfy all conditions

nonConvex Polygon
	nonsimple polygon
Attributes

Attributes are part of the OpenGL state 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 (default)
- Display edges
- Display vertices

- Only one set - cannot fill and display edges
void mydisplay()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
    glBegin(GL_POLYGON);
        glColor3f(1.0, 0.0, 0.0);
        glVertex2f(-0.5, -0.5);
        glColor3f(0.0, 1.0, 0.0);
        glVertex2f(-0.5, 0.5);
        glColor3f(0.0, 0.0, 1.0);
        glVertex2f(0.5, 0.5);
        glColor3f(1.0, 1.0, 0.0);
        glVertex2f(0.5, -0.5);
    glEnd();
    glFlush();
}
void mydisplay()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
    glBegin(GL_POLYGON);
        glColor3f(1.0, 0.0, 0.0);
        glVertex2f(-0.5, 0.5);
        glColor3f(0.0, 1.0, 0.0);
        glVertex2f(-0.5, 0.5);
        glColor3f(0.0, 0.0, 1.0);
        glVertex2f(0.5, 0.5);
        glColor3f(1.0, 1.0, 0.0);
        glVertex2f(0.5, -0.5);
    glEnd();
    glFlush();
}
void mydisplay()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glPolygonMode(GL_FRONT_AND_BACK, GL_POINT);
    glBegin(GL_POLYGON);
        glColor3f(1.0, 0.0, 0.0);
        glVertex2f(-0.5, -0.5);
        glColor3f(0.0, 1.0, 0.0);
        glVertex2f(-0.5, 0.5);
        glColor3f(0.0, 0.0, 1.0);
        glVertex2f(0.5, 0.5);
        glColor3f(1.0, 1.0, 0.0);
        glVertex2f(0.5, -0.5);
    glEnd();
    glFlush();
}
RGB color

- Each color component is stored separately in the frame buffer
- Usually 8 bits per component in buffer (256 values)
- Note in `glColor3f` the color values range from 0.0 (none) to 1.0 (all), whereas in `glColor3ub` the values range from 0 to 255
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

Diagram:
- Frame buffer
- Color lookup table
- Red
- Green
- Blue
- Monitor
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
  ```
void mydisplay()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_POLYGON);
    glColor3f(1.0, 0.0, 0.0);
    glVertex2f(-0.5, -0.5);
    glColor3f(0.0, 1.0, 0.0);
    glVertex2f(-0.5, 0.5);
    glColor3f(0.0, 0.0, 1.0);
    glVertex2f(0.5, 0.5);
    glColor3f(1.0, 1.0, 0.0);
    glVertex2f(0.5, -0.5);
    glEnd();
    glFlush();
}
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`
void mydisplay()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glShadeModel(GL_FLAT);
    glBegin(GL_POLYGON);
    glColor3f(1.0, 0.0, 0.0);
    glVertex2f(-0.5, -0.5);
    glColor3f(0.0, 1.0, 0.0);
    glVertex2f(-0.5, 0.5);
    glColor3f(0.0, 0.0, 1.0);
    glVertex2f(0.5, 0.5);
    glColor3f(1.0, 1.0, 0.0);
    glVertex2f(0.5, -0.5);
    glEnd();
    glFlush();
}
Viewports

- Do not have to use the entire window for the image: `glViewport(x, y, w, h)`
- Values in pixels (screen coordinates)
void mydisplay()
{
    glViewport(250, 250, 250, 250);
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_POLYGON);
        glColor3f(1.0, 0.0, 0.0);
        glVertex2f(-0.5, -0.5);
        glColor3f(0.0, 1.0, 0.0);
        glVertex2f(-0.5, 0.5);
        glColor3f(0.0, 0.0, 1.0);
        glVertex2f(0.5, 0.5);
        glColor3f(1.0, 1.0, 0.0);
        glVertex2f(0.5, -0.5);
    glEnd();
    glFlush();
}