OpenGL Syntax

- Functions have prefix **gl** and initial capital letters for each word

  glClearColor(), glEnable(), glPushMatrix() …

- **glu** for **GLU** functions

  gluLookAt(), gluPerspective() …

- constants begin with **GL_**, use all capital letters

  GL_COLOR_BUFFER_BIT, GL_PROJECTION, GL_MODELVIEW …

- Extra letters in some commands indicate the number and type of variables

  glColor3f(), glVertex3f() …

- OpenGL data types

  GLfloat, GLdouble, GLint, GLenum, …

OpenGL function format

function name

dimensions

`glVertex3f(x,y,z)`

belongs to GL library

`x,y,z` are `floats`

`glVertex3fv(p)`

`p` is a pointer to an array

OpenGL Syntax Examples

Setting the current color using **`glColor`**

- Colors may have 3 components (RGB) or 4 components (RGBA). Think of A (or alpha) as opacity.
- Floating point - color component values range from 0 to 1

```
glColor3f(0.0, 0.5, 1.0);
```

This is 0% Red, 50% Green, 100% Blue;

```
glColor4f(0.0, 0.5, 1.0, 0.3);
```

This is 0% Red, 50% Green, 100% Blue, 30% Opacity

```
GLfloat color[4] = { 0.0, 0.5, 1.0, 0.3 };
  glColor4fv(color);
```

0% Red, 50% Green, 100% Blue, 30% Opacity

OpenGL Syntax Examples

- Unsigned byte – color component values range from 0 to 255 (same as C’s unsigned char).

```
glColor3ub (0, 127, 255);
```

This is: 0% Red, 50% Green, 100% Blue

```
glColor4ub (0, 127, 255, 76);
```

This is 0% Red, 50% Green, 100% Blue, 30% Opacity

Windowing with OpenGL

- OpenGL is independent of any specific window system
- GLUT provide a portable API for creating window and interacting with I/O devices

GLUT

Developed by Mark Kilgard

- Hides the complexities of differing window system APIs
  - Default user interface for class projects
- Glut routines have prefix **`glut`**
  - **`glutCreateWindow() …`**
- Has very limited GUI interface
- **Glui** is the C++ extension of glut that provides buttons, checkboxes, radio buttons, etc.

Glut Routines

- **Initialization**: `glutInit()` processes (and removes) command line arguments that may be of interest to glut and the window system and does general initialization of Glut and OpenGL
  - Must be called before any other glut routines
- **Display Mode**: The next procedure, `glutInitDisplayMode()`, performs initializations informing OpenGL how to set up the frame buffer.

Display Mode	Meaning
GLUT_RGB	Use RGB colors
GLUT_RGBA	Use RGB plus alpha (for transparency)
GLUT_DOUBLE	Use double buffering (recommended)
GLUT_SINGLE	Use single buffering (not recommended)
GLUT_DEPTH	Use depth buffer (for hidden surface removal.)

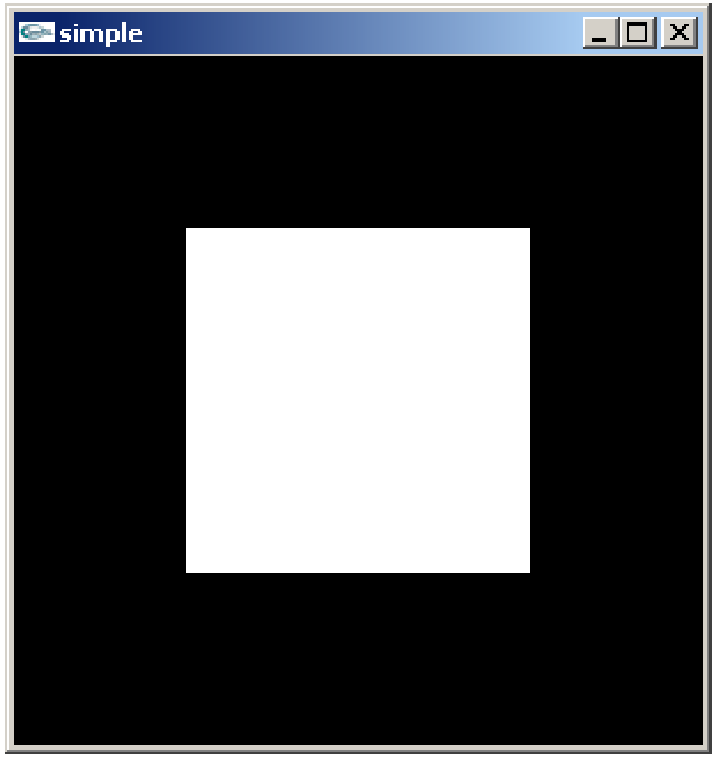
Glut Routines

- Window Setup

```
glutInitWindowSize(int width,int height)
glutInitWindowPosition(int x, int y)
glutCreateWindow(char* title)
```

A Simple Program

Generate a square on a solid background



simple.c

```c
#include <GL/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)
{
        glutInit(&argc, argv);
        glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
        glutInitWindowSize(500,500);
        glutInitWindowPosition(0,0);
        glutCreateWindow("simple");
        glutDisplayFunc(mydisplay);
        init();
        glutMainLoop();
}
```

The function main()

```
#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();
}
```

includes **`gl.h`** and **`glu.h`**

define window properties

display callback

set OpenGL state

enter event loop

The program goes into a infinite loop waiting for events

The function init()

black clear color

opaque window

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

   glColor3f(1.0, 1.0, 1.0);

   glMatrixMode (GL_PROJECTION);
   glLoadIdentity ();
   gluOrtho2D(-1.0, 1.0, -1.0, 1.0);
}
```

fill/draw with white

viewport

Callback functions

- Most of window-based programs are event-driven
  - Even driven means do nothing until an event happens, and then execute some pre-defined functions

- Events – key press, mouse button press and release, window resize, etc.

Callbacks

- Virtually all interactive graphics programs are event driven
- Glut uses callbacks to handle events
  - Windows system invokes a particular procedure when an event of particular type occurs.
  - MOST IMPORTANT: display event
    - Signaled when window first displays and whenever portions of the window reveals from blocking window
    - **`glutDisplayFunc(void (*func)(void))`** registers the display callback function
- Running the program: **`glutMainLoop()`**
  - Main event loop. Never exit()

More Callbacks

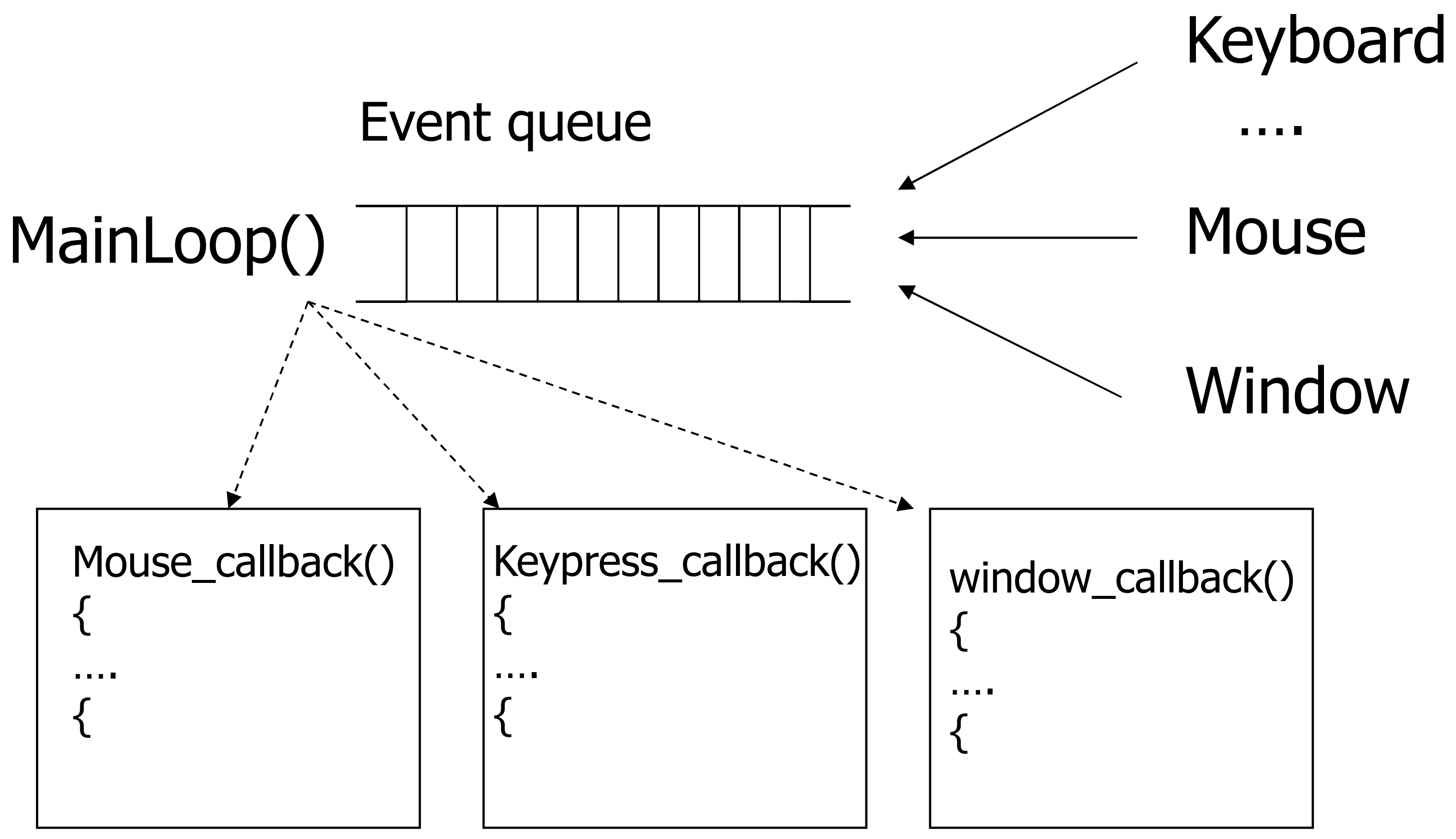
- **`glutReshapeFunc(void (*func)(int w, int h))`** indicates what action should be taken when the window is resized.
- **`glutKeyboardFunc(void (*func)(unsigned char key, int x, int y))`** and **`glutMouseFunc(void (*func)(int button, int state, int x, int y))`** allow you to link a keyboard key or a mouse button with a routine that's invoked when the key or mouse button is pressed or released.
- **`glutMotionFunc(void (*func)(int x, int y))`** registers a routine to call back when the mouse is moved while a mouse button is also pressed.
- **`glutMouseFunc(void (*func)(int button, int state, int x, int y))`** registers a function that's to be executed if a mouse button event occurs. The argument button can be GLUT_LEFT_BUTTON or GLUT_RIGHT_BUTTON. The argument state can be GLUT_UP or GLUT_DOWN. The arguments x and y indicated the mouse cursor position when the button was clicked.

glutDisplayFunc(void (*func)(void))

```
int main(int argc, char** argv)
{
        …
    glutDisplayFunc(mydisplay);
    …
}
```

void display() – the function you provide. It contains all the OpenGL drawing function calls and will be called when pixels in the window need to be refreshed.

Event Queue


Keyboard
....
Mouse
Window
Event queue
MainLoop()
Mouse_callback()
{
....
{
Keypress_callback()
{
....
{
window_callback()
{
....
{

glut Functions

- glutKeyboardFunc() – register the callback that will be called when a key is pressed
- glutMouseFunc() – register the callback that will be called when a mouse button is pressed
- glutMotionFunc() – register the callback that will be called when the mouse is in motion while a buton is pressed
- glutIdleFunc() – register the callback that will be called when nothing is going on (no event)

OpenGL Drawing

Steps in the display function

- Clear the window
- Set drawing attributes
- Send drawing commands
- Flush the buffer

Clear the Window

- `glClear(GL_COLOR_BUFFER_BIT)`
  - Clears the frame buffer by overwriting it with the background color.
- Background color is a state set by

`glClearColor(GLfloat r, GLfloat g, GLfloat b, GLfloat a)` in the `init()`.

Drawing Attributes: Color

- `glColor3f(GLfloat r, GLfloat g, GLfloat b)` sets the drawing color

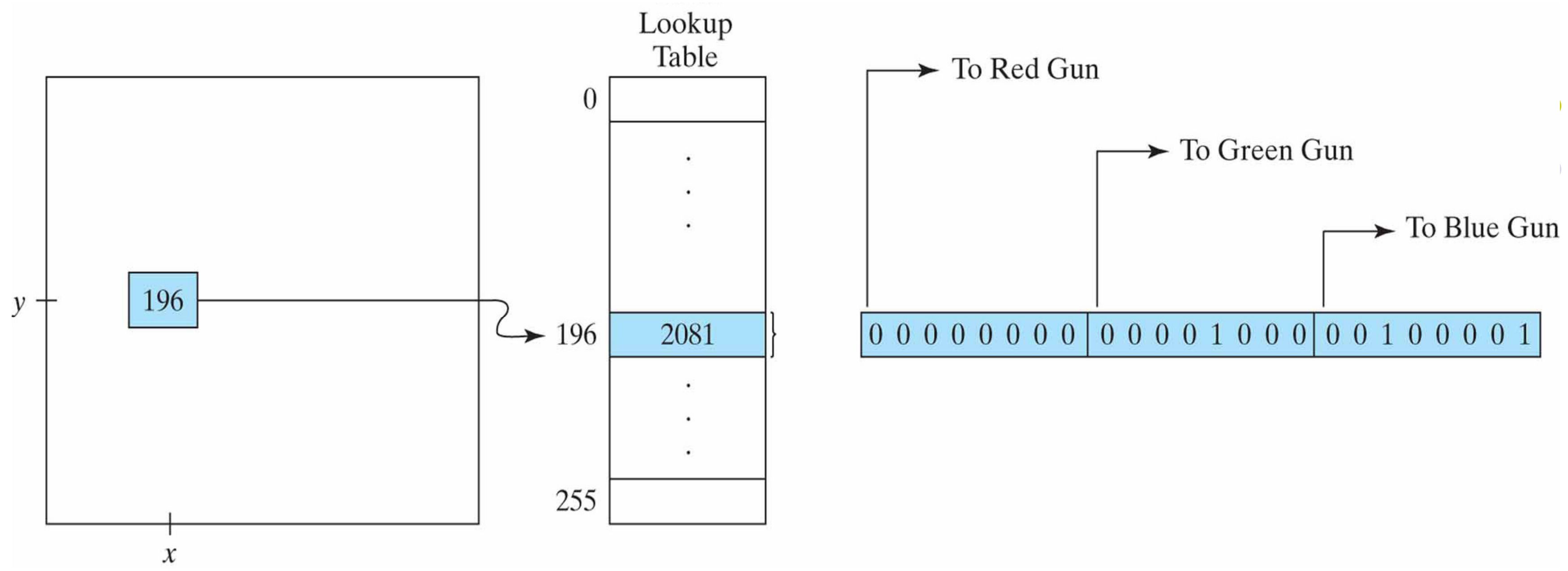
  `glColor3d()`, `glColor3ui()` can also be used
- OpenGL is a state machine
  - Once set, the attribute applies to all subsequent defined objects until it is set to some other value

    `glColor3fv()` takes a flat array as input

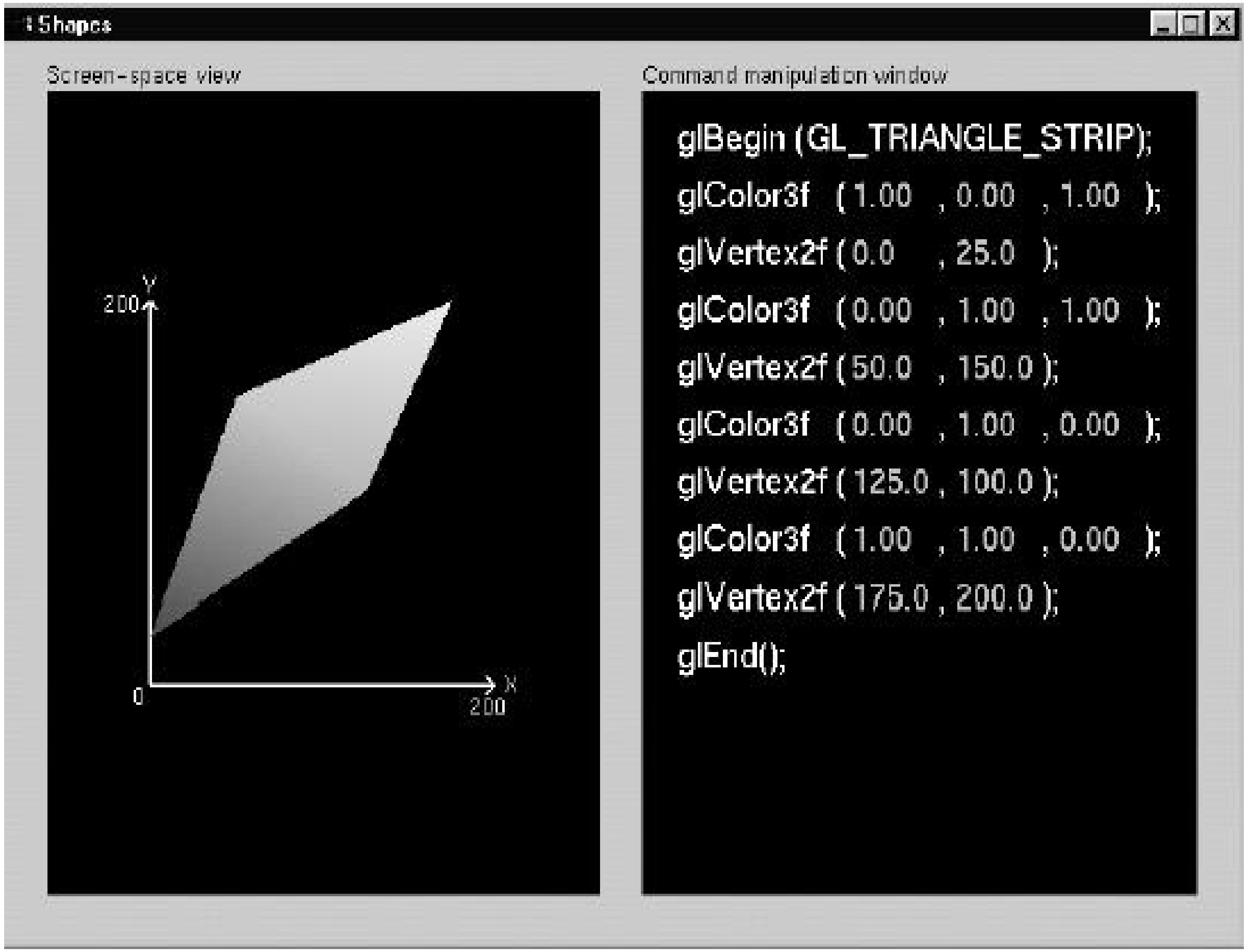
TABLE 5-1

The eight RGB color codes for a 3-bit-per-pixel frame buffer

	Stored Color Values in Frame Buffer			
Color Code	RED	GREEN	BLUE	Displayed Color
0	0	0	0	Black
1	0	0	1	Blue
2	0	1	0	Green
3	0	1	1	Cyan
4	1	0	0	Red
5	1	0	1	Magenta
6	1	1	0	Yellow
7	1	1	1	White



A color lookup table with 24 bits per entry that is accessed from a frame buffer with 8 bits per pixel.
A value of 196 stored at pixel position (*x*, *y*) references the location in this table containing the hexadecimal value 0x0821 (a decimal value of 2081).
Each 8-bit segment of this entry controls the intensity level of one of the three electron guns in an RGB monitor.

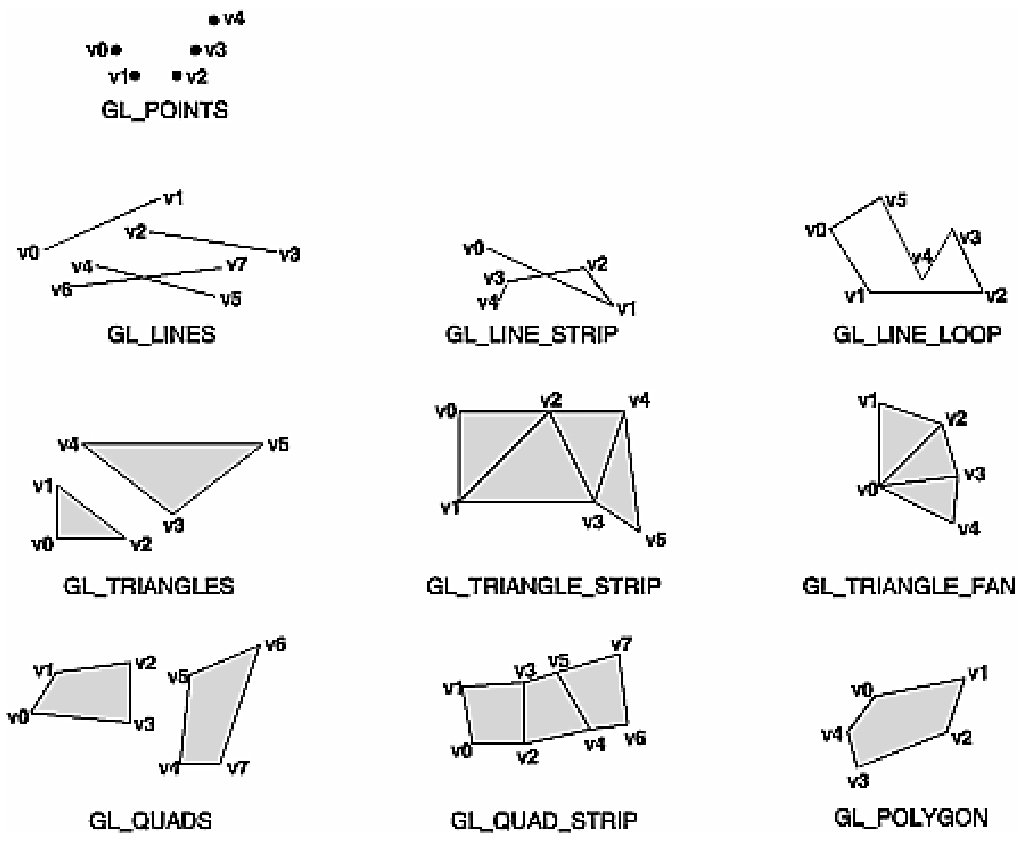


```
glBegin (GL_TRIANGLE_STRIP);
glColor3f  ( 1.00  , 0.00  , 1.00  );
glVertex2f ( 0.0    , 25.0  );
glColor3f  ( 0.00  , 1.00  , 1.00  );
glVertex2f ( 50.0  , 150.0 );
glColor3f  ( 0.00  , 1.00  , 0.00  );
glVertex2f ( 125.0 , 100.0 );
glColor3f  ( 1.00  , 1.00  , 0.00  );
glVertex2f ( 175.0 , 200.0 );
glEnd();
```

Drawing Commands

- Simple Objects
  `glRectf()`
- Complex Objects
  - Use construct **`glBegin(mode)`** and **`glEnd()`** and a list of vertices in between

    -
      ```
      glBegin(mode)
            glVertex(v0);
            glVertex(v1);
            ...
      glEnd();
      ```

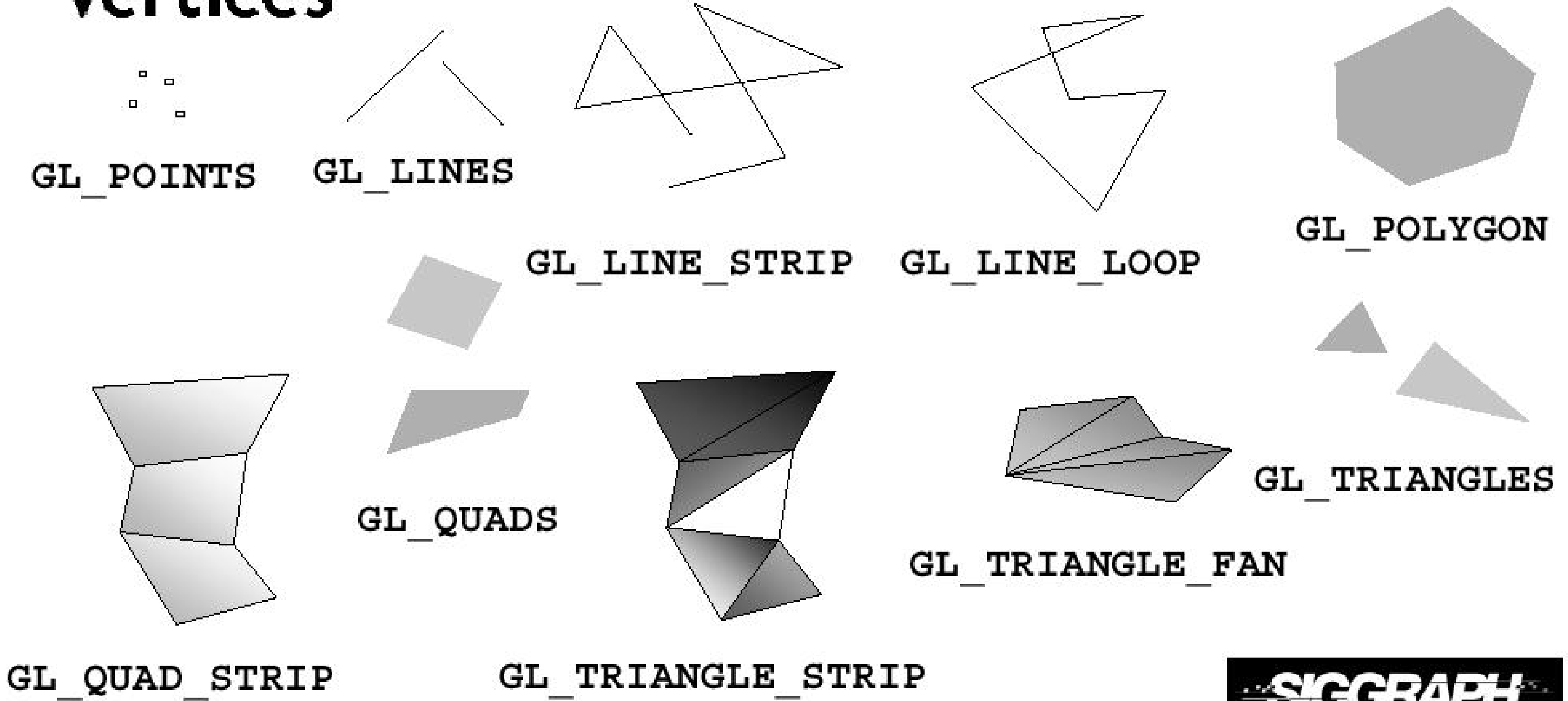


Drawing Attributes

- Besides `glVertex()` commands, other attributes commands can also be used between `glBegin()` and `glEnd()`, e.g. `glColor3f()`.
- There are more drawing attributes than color
  - Point size: `glPointSize()`
  - Line width: `glLinewidth()`
  - Dash or dotted line: `glLineStipple()`
  - Polygon pattern: `glPolygonStipple()`
  - …

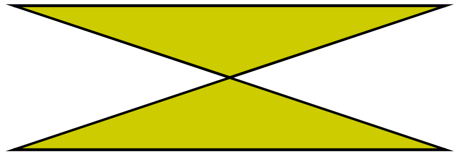
Primitive Types

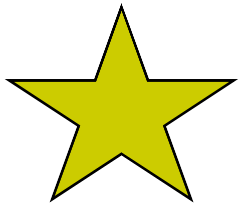
All geometric primitives are specified by vertices



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

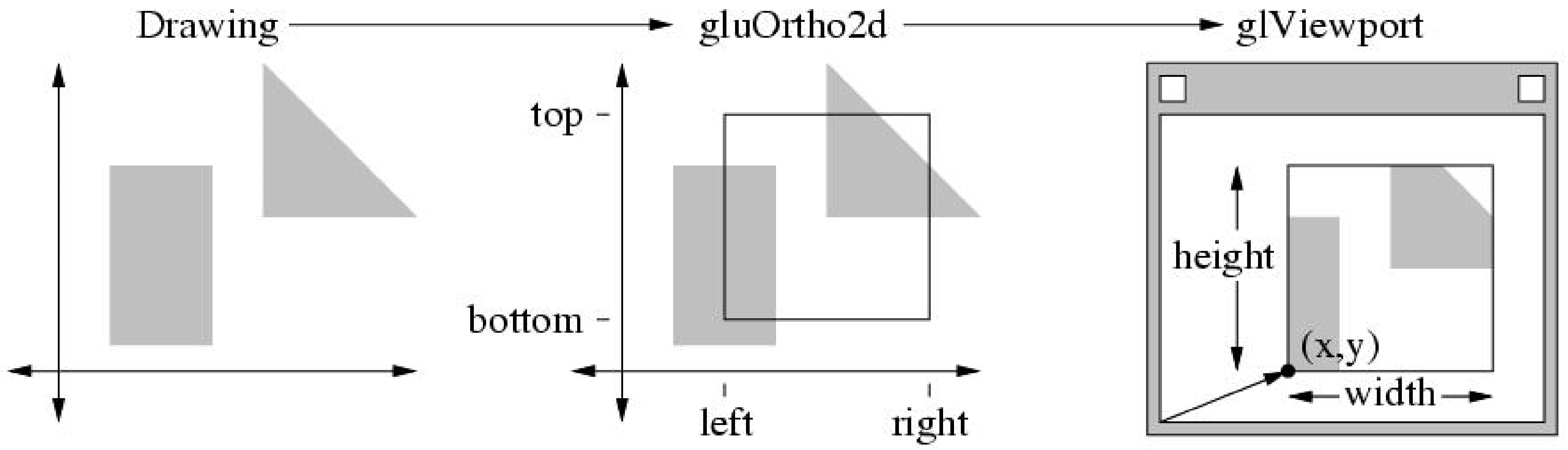

nonsimple polygon


nonconvex polygon

Display Callback

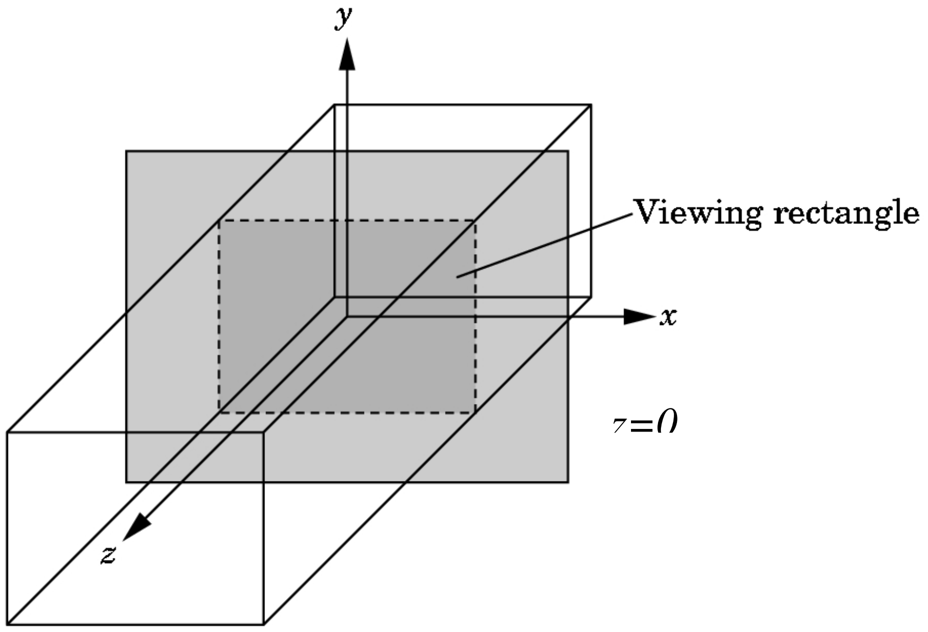
```
void mydisplay()
{
        glClear(GL_COLOR_BUFFER_BIT);
        glBegin(GL_POLYGON);
                glVertex2f(-0.5, -0.5);
                glVertex2f(-0.5, 0.5);
                glVertex2f(0.5, 0.5);
                glVertex2f(0.5, -0.5);
        glEnd();
        glFlush();
}
```

Projection and Viewport



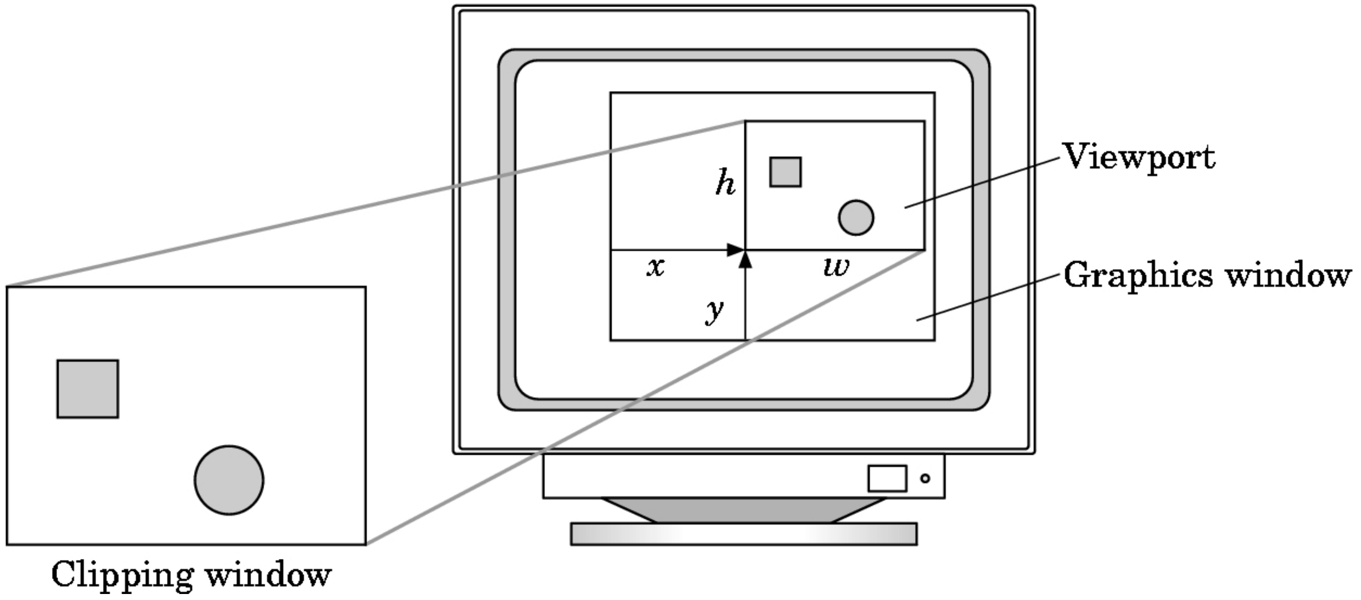
Orthographic projection

- Orthographic projection used for 2D drawing, Perspective project often used for 3D drawing
- 2D Viewing: Orthographic View
  - `gluOrtho2D(left, right, bottom, top)`
    - Specifies the coordinates of 2D region to be projected into the viewport.
    - Any drawing outside the region will be automatically clipped away.

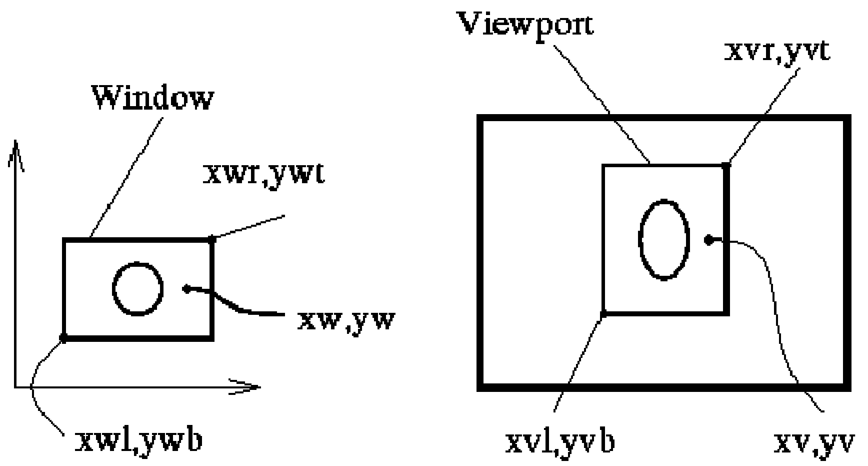


Viewports

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



Window to Viewport mapping



Aspect Ratio: Height/Width
If the aspect ratio of the window
Is different from that of the
viewport, the picture will be
distorted.