Chapter 3:
OpenGL Programming

OpenGL Graphics Library

Machines exist; let us then exploit them to create beauty, a modern beauty, while we are about it. For we live in the twentieth century.

-Aldous Huxley

Computer Graphics..

- Like many disciplines, computer graphics is mastered most quickly by doing it:
  - by writing and testing programs that produce a variety of pictures.
  - start with simple tasks, when these are mastered.. try different variations, see what happens and move towards drawing more complex scenes.
- To get started you need:
  1. **Hardware** to display pictures.
  2. Library of **software tools** that your programs can use to perform actual drawing.

- Every graphics program begins with:
  - some initialization to establish the desired **display mode**.
  - set up a **coordinate system** for specifying points, lines, etc.

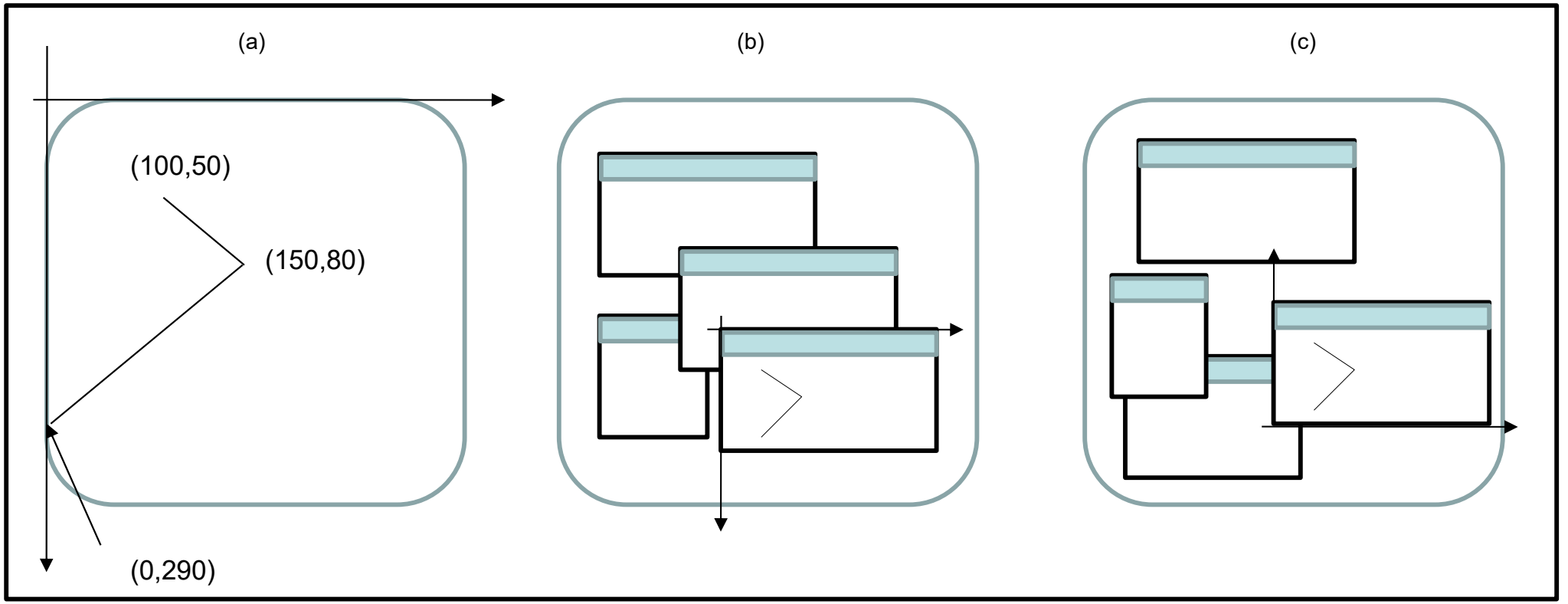


Figure 3.1 Some common varieties of display layouts

- Fig 3.1 a) : the entire screen is used for drawing.
  - the display is initialized into graphics mode
  - the coordinate system is established as shown (with x and y measured in pixel)
- Fig 3.1 b): A “window-based” system.
  - supports a number of different rectangular windows on the display screen at one time
  - initialization involves creating and “opening” a new window for graphics
  - graphics commands use a coordinate system that is attached to the window
- Fig 3.1 c): same as Fig 3.1 b) but with y increasing upwards.

- Every graphics package has some elementary drawing functions that help user to get started.
- The most basic function has a name like setPixel(x,y,color)
  - sets the individual pixel at location (x,y) using the specified color
  - sometimes this function goes by different name such as putPixel(..), Setpixel(..) or drawPoint(..)
- Usually there are also function for drawing lines such as line(x1,y1,x2,y2),
  - that draws line between (x1,y1) and (x2,y2)
  - Sometimes this function is called as drawLine(..) or Line(..)
- Thus the commands

  Line(100,50,150,80); Line(150,80,0,290);

  will draw the lines shown in Fig 3.1 a)

Graphics functions

- A general purpose **graphics package** provides users with a variety of functions for creating and manipulating pictures.
- These functions can be classified according to whether they deal with graphics
  - **output** (drawing lines, points, curved lines etc)
  - **input** (input functions are used to control and process data flow from various interactive devices)
  - **attributes** (color specifications, line styles, text styles, and area-filling patterns)
  - **transformations** (change size, position, or orientation)
  - **viewing** (select view of scene, type of projection to use, location on video monitor where view is to be displayed)
  - **subdividing pictures** (dividing pictures into named components)
  - **general control** (housekeeping tasks, such as clearing a screen display etc.)

Device-independent programming

- An **uniform approach** to writing graphics applications
  - Same program can be compiled and run on a **variety** of graphics environments
  - With **guarantee** it will produce nearly identical graphical output on each display
  - This is known as **device-independent** graphics programming

OpenGL

- *OpenGL* is a hardware independent graphics library
  - **Porting** a graphics program only requires that you install the appropriate OpenGL libraries on the new machine
  - The application itself **doesn't** require any changes (i.e. it calls the same function with same parameters)
  - OpenGL has been adopted by many companies and OpenGL libraries **exists** for all the popular graphics environments.

…OpenGL

- OpenGL can be used to draw complicated shapes such as *automobiles*, *parts of the human body*, *airplanes* or *molecules*.

- To build these complicated shapes, a small set of **geometric primitives** such as:

  – points,
  – lines, and
  – polygons

  are used.

- Functions in **OpenGL can be used to specify** graphics primitives, attributes, geometric transformations, viewing transformations and other operations.

- OpenGL is designed to be **hardware independent**
  – Thus, operations such as input and output routines are **not** included in OpenGL.
  – However, input, output and other additional functions are **available in auxiliary** (supplementary) libraries that have been developed for OpenGL programs.

Basic OpenGL Syntax

- Function names in the OpenGL basic library are prefixed with **gl**, and each component word within a function name (usually it reflects the functionality of the function) has its first letter capitalized.

  glClear, glCopyPixels, glPointSize, glLoadIdentity

- **Symbolic constants** are sometimes used in functions, for instance as a *parameter name*, *a value for a parameter*, or *a particular mode*.
  - These constants begin with uppercase letters **GL**.
  - Component word within the function name are written in capital letters and an underscore "_" between component words in the name.

    GL_2D, GL_RGB, GL_CCW, GL_POLYGON, GL_AMBIENT_AND_DIFFUSE

OpenGL Data Types

- OpenGL functions also expects specific **data types**.
- An OpenGL function might expect a value specified as 32-bit integer, but the size of an integer specification can be different on different machines.
- To indicate a special data types, OpenGL uses special built-in, data type names, such as

GLbyte, GLshort, GLint, GLfloat

- Each data type begins with the capital letters **GL** the remainder is a standard data-type designation, written in lower-case.

..Open GL Data Types

Suffix	Data type	Typical C or C++ type	OpenGL type name
b	8-bit integer	signed char	GLbyte
s	16-bit integer	short	GLshort
i	32-bit integer	int or long	Glint, GLsizei
f	32-bit floating point	float	GLfloat, GLclampf
d	64-bit floating point	double	GLdouble, GLclampd
ub	8-bit unsigned number	unsigned char	GLubyte, GLboolean
us	16-bit unsigned number	unsigned short	GLshort
ui	32-bit unsigned number	unsigned int or unsigned long	GLuint, GLenum, GLbitfield

Data Types in C/C++

Name	Description	Size*	Range*
char	Character or small integer.	1byte	signed: -128 to 127 unsigned: 0 to 255
short int (short)	Short Integer.	2bytes	signed: -32768 to 32767 unsigned: 0 to 65535
int	Integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
long int (long)	Long integer.	4bytes	signed: -2147483648 to 2147483647 unsigned: 0 to 4294967295
bool	Boolean value. It can take one of two values: true or false.	1byte	true or false
float	Floating point number.	4bytes	3.4e +/- 38 (7 digits)
double	Double precision floating point number.	8bytes	1.7e +/- 308 (15 digits)
long double	Long double precision floating point number.	8bytes	1.7e +/- 308 (15 digits)
wchar_t	Wide character.	2bytes	1 wide character

Auxiliary Libraries for OpenGL

- These libraries are for handling special operations that is not provided by OpenGL.
- There are many different libraries such as

  – **OpenGL Utility (GLU)**, provides routines for setting up viewing and projection matrices, describing complex objects with line and polygon approximation and other complex tasks.
    - Every OpenGL implementation includes the **GLU** library
    - All GLU function names start with the prefix **glu**.

  – **Open inventor** (provides routines and predefined object shapes for interactive 3D applications).
    - It is an object oriented toolkit written in C++.

  – Apple systems can use **Apple GL** (**AGL**) interface for window-management operations.
    - Function names are prefixed with **agl**.

…Auxiliary libraries for OpenGL

- **Windows-to-OpenGL** (**WGL**) is an interface for Microsoft Windows systems.
  - The routines are prefixed with the letters **wgl**.

- **Presentation Manager to OpenGL (PGL)** an interface for IBM OS/2.
  - The routines are prefixed with **pgl**.

- **The OpenGL Utility toolkit (GLUT)** provides library for interacting with any screen-windowing system.
  - This routines are prefixed with **glut**.
  - It also contains methods for describing and rendering quadric curves and surfaces.

Header files

- In all of our graphics programs, we will need to include the header files for **OpenGL core library** and for most application we also need **GLU**.
- For Microsoft Windows, the header file that accesses **WGL** routines is "windows.h". So must include "windows.h".
  - This file must be listed before the **OpenGL** and **GLU** header files because it contains macros needed by the Microsoft Windows version of OpenGL libraries.
  - So the source file in this case would begin with,

```
#include <windows.h>
#include <GL/gl.h>
#include <GL/glu.h>
```

- However, if we use **GLUT** to handle the window-managing operations, we do not need to include "gl.h" and "glu.h" because GLUT ensures that this will be included correctly.
  - So we can replace header files for OpenGL and GLU with

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

“Window-based” programming

- The **first task** in making pictures is to open a screen window for drawing.

- Because OpenGL functions are device independent, they provide **no support** for controlling windows on specific systems.

- But **auxiliary tool** for OpenGL such as GLUT (OpenGL Utility Toolkit) provides library of functions for interacting with any screen-windowing system.

A simple program using OpenGL and GLUT to draw a line on a window

```
#include <GL/glut.h>      // (or others, depending on the system in use)

void init (void)
{
   glClearColor (1.0, 1.0, 1.0, 0.0);  // Set display-window color to white.
   glMatrixMode (GL_PROJECTION);       // Set projection parameters.
   gluOrtho2D (0.0, 200.0, 0.0, 150.0);
}

void lineSegment (void)
{
   glClear (GL_COLOR_BUFFER_BIT);  // Clear display window.

   glColor3f (1.0, 0.0, 0.0);      // Set line segment color to red.
   glBegin (GL_LINES);
      glVertex2i (180, 15);       // Specify line-segment geometry.
      glVertex2i (10, 145);
   glEnd ( );

   glFlush ( );     // Process all OpenGL routines as quickly as possible.
}

void main (int argc, char** argv)
{
   glutInit (&argc, argv);                         // Initialize GLUT.
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);   // Set display mode.
   glutInitWindowPosition (50, 100);   // Set top-left display-window position.
   glutInitWindowSize (400, 300);      // Set display-window width and height.
   glutCreateWindow ("An Example OpenGL Program"); // Create display window.

   init ( );                           // Execute initialization procedure.
   glutDisplayFunc (lineSegment);      // Send graphics to display window.
   glutMainLoop ( );                   // Display everything and wait.
}
```

Display-Window management using GLUT

1. First step is to **initialize GLUT**.
   - The initialization can also process any *command line arguments*.
   - GLUT initialization is performed with

     glutInit(&argc,argv);
2. Then we set options for display window such as **buffering** and a **choice of color modes**.
   - This is performed using *glutInitDisplayMode* function.
   - Arguments for this function are **symbolic GLUT** constants.
   - The following command specifies that a single refresh buffer is to be used for the display window (GLUT_SINGLE) and RGB color mode is used for selecting color (GLUT_RGB)

     glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);

..Display-Window management using GLUT

3. The function *glutInitWindowPosition* is used to give initial location for the top-left corner of the display window.
   - this position is specified in integer screen coordinate,
   - origin is **upper left** corner of the screen.
   - To place the display windows top-left corner at (50,100) the following command is issued

     glutInitWindowPosition (50, 100);

4. The *glutInitWindowSize* function is used to set initial pixel width and height of the display window.
   - The following command sets an initial width of 400 pixels and a height of 300 pixels.

     glutInitWindowSize (400, 300);

…Display-Window management using GLUT

5. Next we can state that a display window is to be created with a given caption for the title bar.

```
glutCreateWindow ("An Example OpenGL Program");
```

- These five **GLUT** functions initialize and display the screen window in which our program will produce graphics.

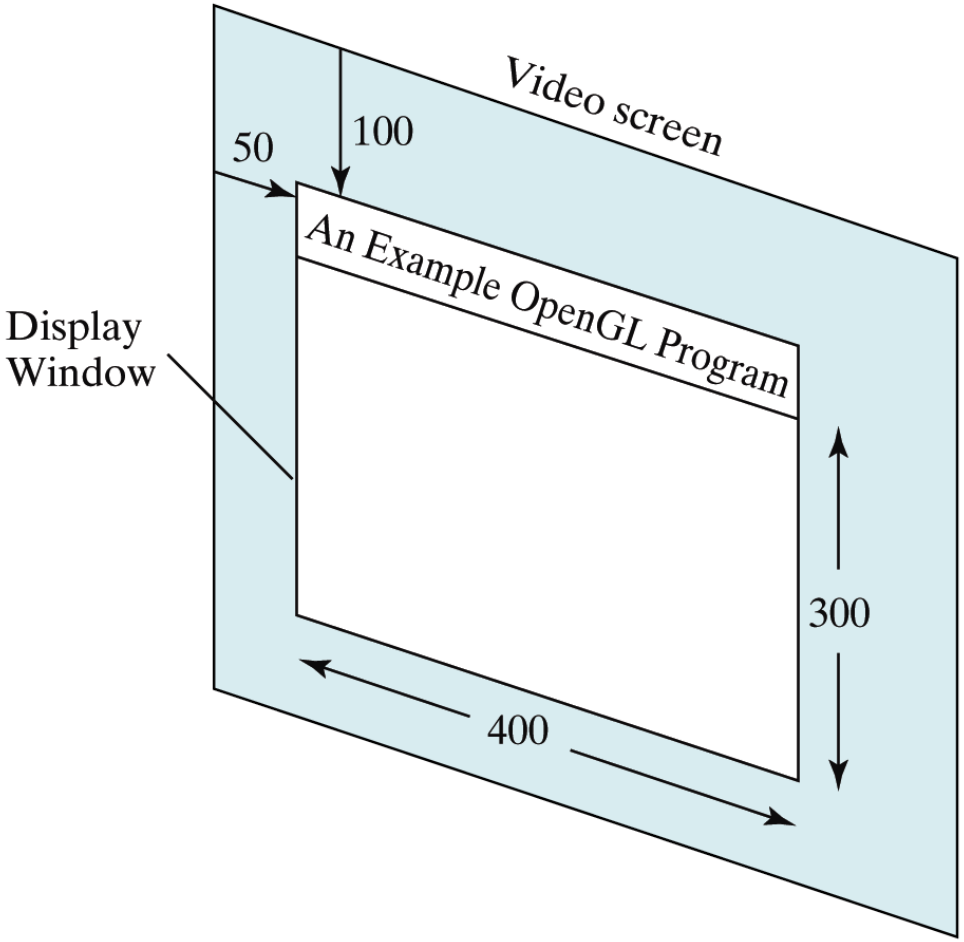


Figure 2-61

A 400 by 300 display window at position (50, 100) relative to the top-left corner of the video display.

Setting background color

- To set the **background color** for the display window use

  glClearColor(red,green,blue,alpha);

  alpha specifies the degree of transparency.

  e.g

  for background to be white,

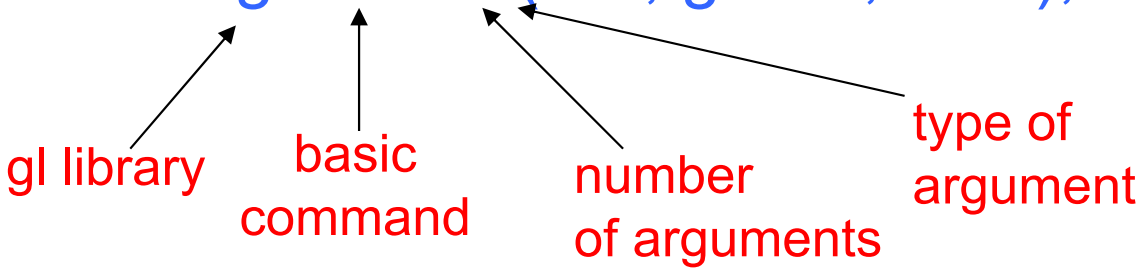
  glClearColor(1.0,1.0,1.0,0.0);

  – The glClearColor() command assigns a color to the display window, but **does not** put display window on the screen. Thus, we **need to invoke** glClear(GL_COLOR_BUFFER_BIT);

  – The symbolic constant GL_COLOR_BUFFER_BIT specifies that it is the bit values in the color buffer (refresh buffer) that are to be set to the values indicated in the glClearColor function .

- The color of a drawing can be specified using

```
glColor3f(red, green, blue);
```



- To display a 2D line segment,
  - Need to tell OpenGL how to "project" our picture onto the display window.
  - This is because generating 2D picture is treated to by OpenGL as a special case of 3D viewing operations.
  - We can use the following two functions to set the projection type (mode) and other viewing parameters

```
glMatrixMode(GL_PROJECTION);
gluOrtho2D(0.0, 200.0, 0.0,150.0);
```

– The gluOrtho2D command is a function we can use to set up any 2D Cartesian reference frame.
  - The arguments for this function are the 4 values defining the x and y coordinate limits for the picture we want to display.
  - In our example, the x-coordinate values within this rectangle range from 0.0 to 200.0 with y-coordinate values ranging from 0.0 to 150.0

– Next, the OpenGL routine to create the line segment.

```
glBegin(GL_LINES);
        glVertex2i(180,15);
        glVertex2i(10,145);
glEnd();
```

– The code defines a 2D, straight-line segment with integer, Cartesian endpoint coordinates (180,15) and (10,145).

- The geometric description of the “picture” we want to display is in function *lineSegment*.
  - This is the function that will be referenced by the **GLUT** function glutDisplayFunc.
  - This **GLUT** routine assigns our picture to the display window.

```
glutDisplayFunc(lineSegment);
```

- But the display window is not yet on the screen.
  - The **GLUT** glutMainLoop() is the function required to complete the window-processing operations.
  - After executing this function, all display windows that we have created, including their graphic content, are now activated.

– The function displays the initial graphics and puts the program into an infinite loop that checks for input from devices such as mouse or keyboard.

A simple program using OpenGL and GLUT to draw a line on a window

```
#include <GL/glut.h>      // (or others, depending on the system in use)

void init (void)
{
   glClearColor (1.0, 1.0, 1.0, 0.0);  // Set display-window color to white.
   glMatrixMode (GL_PROJECTION);       // Set projection parameters.
   gluOrtho2D (0.0, 200.0, 0.0, 150.0);
}

void lineSegment (void)
{
   glClear (GL_COLOR_BUFFER_BIT);  // Clear display window.

   glColor3f (1.0, 0.0, 0.0);      // Set line segment color to red.
   glBegin (GL_LINES);
      glVertex2i (180, 15);       // Specify line-segment geometry.
      glVertex2i (10, 145);
   glEnd ( );

   glFlush ( );     // Process all OpenGL routines as quickly as possible.
}

void main (int argc, char** argv)
{
   glutInit (&argc, argv);                         // Initialize GLUT.
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);   // Set display mode.
   glutInitWindowPosition (50, 100);   // Set top-left display-window position.
   glutInitWindowSize (400, 300);      // Set display-window width and height.
   glutCreateWindow ("An Example OpenGL Program"); // Create display window.

   init ( );                           // Execute initialization procedure.
   glutDisplayFunc (lineSegment);      // Send graphics to display window.
   glutMainLoop ( );                   // Display everything and wait.
}
```