CHAPTER 1

INTRODUCTION

1.1 COMPUTER GRAPHICS

The term Computer Graphics has been used in a broad sense to describe "almost everything on computers that is not text or sound".Typically, the term computer graphics refers to several different things:

- the representation and manipulation of image data by a computer
- the various technologies used to create and manipulate images
- the images so produced, and
- the sub-field of computer science which studies methods for digitally synthesizing and manipulating visual content, see study of computer graphics

Today almost every computer can do some graphics, and people have even come to expect to control their computer through icons and pictures rather than just by typing.

Here in our lab at the Program of Computer Graphics, we think of computer graphics as drawing pictures on computers, also called rendering. The pictures can be photographs, drawings, movies, or simulations -- pictures of things which do not yet exist and maybe could never exist. Or they may be pictures from places we cannot see directly, such as medical images from inside your body.

We spend much of our time improving the way computer pictures can simulate real world scenes. We want images on computers to not just look more realistic, but also to BE more realistic in their colors, the way objects and rooms are lighted, and the way different materials appear. We call this work "realistic image synthesis", and the following series of pictures will show some of our techniques in stages from very simple pictures through realistic ones.

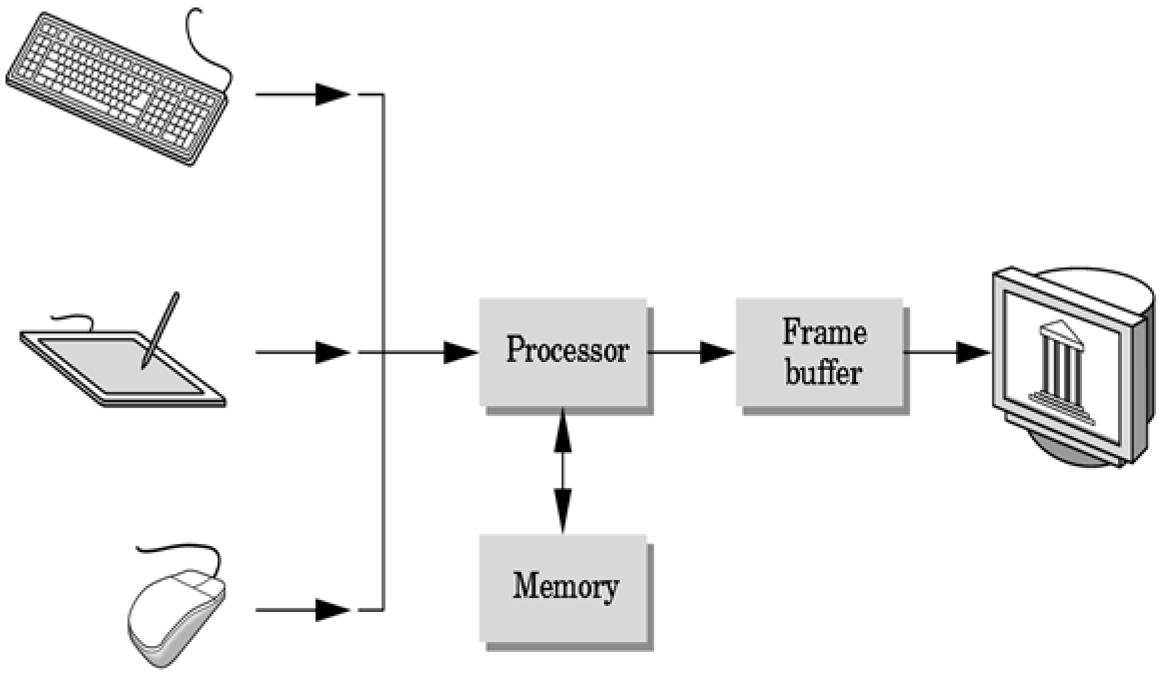


Fig 1.1: Computer Graphics system

1.2 OPENGL TECHNOLOGY

OpenGL is the premier environment for developing portable, interactive 2D and 3D graphics applications. Since its introduction in 1992, OpenGL has become the industry's most widely used and supported 2D and 3D graphics application programming interface (API), bringing thousands of applications to a wide variety of computer platforms. OpenGL fosters innovation and speeds application development by incorporating a broad set of rendering, texture mapping, special effects, and other powerful visualization functions. Developers can leverage the power of OpenGL across all popular desktop and workstation platforms, ensuring wide application deployment. Any visual computing application requiring maximum performance-from 3D animation to CAD to visual simulation-can exploit high-quality, high-performance OpenGL capabilities. These capabilities allow developers in diverse markets such as broadcasting, CAD/CAM/CAE, entertainment, medical imaging, and virtual reality to produce and display incredibly compelling 2D and 3D graphics.

OpenGL (**O**pen **G**raphics **L**ibrary) is a standard specification defining a cross-language, cross platform API for writing applications that produce 2D and3D computer

Graphics interface consists of over 250 different function calls which can be used to draw complex three-dimensional scenes from simple primitives. OpenGL was developed by Silicon Graphics Inc (SGI) in 1992and is widely used in CAD, virtual Reality, scientific visualization, information visualization, and flight simulation. It is also used in video games, where it competes with Direct3D on Microsoft Windows platforms. OpenGL is managed by the non-profit technology consortium, the Khronos Group.
OpenGL serves two main purposes:

- To hide the complexities of interfacing with different 3D accelerators, by presenting the programmer with a single, uniform API.
- To hide the differing capabilities of hardware platforms, by requiring that all implementations support the full OpenGL feature set (using software emulation if necessary).

OpenGL's basic operation is to accept primitives such as points, lines and polygons, and convert them into pixels. This is done by a graphics pipeline known as the OpenGL state machine. Most OpenGL commands either issue primitives to the graphics pipeline, or configure how the pipeline processes these primitives. Prior to the introduction of OpenGL 2.0, each stage of the pipeline performed a fixed function and was configurable only within tight limits. OpenGL 2.0 offers several stages that are fully programmable using GLSL.

OpenGL is a low-level, procedural API, requiring the programmer to dictate the exact steps required to render a scene. This contrast with descriptive APIs, where a programmer only needs to describe a scene and can let the library manage the details of rendering it. OpenGL's low-level design requires programmers to have a good knowledge of the graphics pipeline, but also gives a certain amount of freedom to implement novel rendering algorithms.

Advantages

- **Industry standard: -** An independent consortium, the OpenGL Architecture Review Board, guides the OpenGL specification. With broad industry support, OpenGL is the only truly open, vendor-neutral, multiplatform graphics standard.
- **Stable: -** OpenGL implementations have been available for more than seven years on a wide variety of platforms. Additions to the specification are well controlled, and

proposed updates are announced in time for developers to adopt changes. Backward compatibility requirements ensure that existing applications do not become obsolete.

- **Reliable and portable:** - All OpenGL applications produce consistent visual display results on any OpenGL API-compliant hardware, regardless of operating system or window
- **Evolving:** - Because of its thorough and forward-looking design, OpenGL allows new hardware innovations to be accessible through the API via the OpenGL extension mechanism. In this way, innovations appear in the API in a timely fashion, letting application developers and hardware vendors incorporate new features into their normal product release cycles.
- **Scalable:** - OpenGL API-based applications can run on systems ranging from consumer electronics to PCs, workstations, and supercomputers. As a result, applications can scale to any class of machine that the developer chooses to target.
- **Easy to use:** - OpenGL is well structured with an intuitive design and logical commands. Efficient OpenGL routines typically result in applications with fewer lines of code than those that make up programs generated using other graphics libraries or packages. In addition, OpenGL drivers encapsulate information about the underlying hardware, freeing the application developer from having to design for specific hardware features.
- **Well-documented: -** Numerous books have been published about OpenGL, and a great deal of sample code is readily available, making information about OpenGL inexpensive and easy to obtain.
- **Evolving:** - Because of its thorough and forward-looking design, OpenGL allows new hardware innovations to be accessible through the API via the OpenGL extension mechanism.
- **Scalable:** - OpenGL API-based applications can run on systems ranging from consumer electronics to PCs, workstations, and supercomputers. As a result, applications can scale to any class of machine that the developer chooses to target.
- **Easy to use:** - OpenGL is well structured with an intuitive design and logical commands. Efficient OpenGL routines typically result in applications with fewer lines of code than those that make up programs generated using other graphics libraries or packages. In addition, OpenGL drivers encapsulate information about the underlying hardware.

- **Well-documented: -** Numerous books have been published about OpenGL, and a great deal of sample code is readily available, making information about OpenGL inexpensive and easy to obtain.

Applications of OpenGL

The development of computer graphics has been driven both by the user community and by advances in hardware and software. The application of computers graphics are many and varied; we can however, divide them in to four major areas: -

- Display of Information
- Design
- Simulation and animation
- User interfaces

1.3 PROJECT DESCRIPTION

PROJECT NAME : To implement dual window.

AIM: To design an OpenGL application program to implement the dual window.

DESCRIPTION:

The aim in developing this program was to design a simple program using Open GL application software by applying the skills we learnt in class, and in doing so, to understand the algorithms and the techniques underlying interactive graphics better.

The designed program will incorporate all the basic properties that a simple program must possess .The program is extremely user friendly as the only skill required in executing this program is the knowledge of graphics.

The main idea of our program is to implement the DUAL WINDOW, selecting the objects using menus and applying rotation to them. In this program we are create four objects using textures, which can be rotated using special keys (Arrow Keys).

If we click on the 1st option it adds the Square plate without rotation in first window and with rotation in the second window. Similarly if we click on the 2nd, 3rd and 4th option it adds the second, third and fourth objects without rotation in first window and with rotation in the second window respectively.

We can rotate the objects on output screen by using the Arrow keys in the keyboard. The keys which are used are:-

- **“LEFT ARROW KEY”**: - this key is used to turn to the left.
- **“UP ARROW KEY”**: - this key is used to turn above.
- **“RIGHT ARROW KEY”**: - this key is used to turn to the right.

CHAPTER 2

SYSTEM SPECIFICATION

2.1 SOFTWARE REQUIREMENTS

Minimum Software Requirements	
Operating System	Windows XP
Other	Windows comes with OpenGL, and Visual Studio comes with the OpenGL libraries, but neither of them comes with GLUT .GLUT 3.7.6 has to be included. Prior to installing the latest Windows service packs and critical updates have to be updated.

Fig 2.1a: Software requirements

2.2 HARDWARE REQUIREMENTS

Minimum Hardware Requirements	
Processor	133-MHz Intel Pentium-class processor
Memory	128 MB of RAM, 256 MB recommended
Hard Disk	110 MB of hard disk space required, 40 MB additional hard disk space required for installation (150 MB total)
Display	800 x 600 or higher-resolution display with 256 colors

Fig2.2b: Hardware requirements

CHAPTER 3

DESIGN

3.1 DESIGN OF DUAL WINDOW

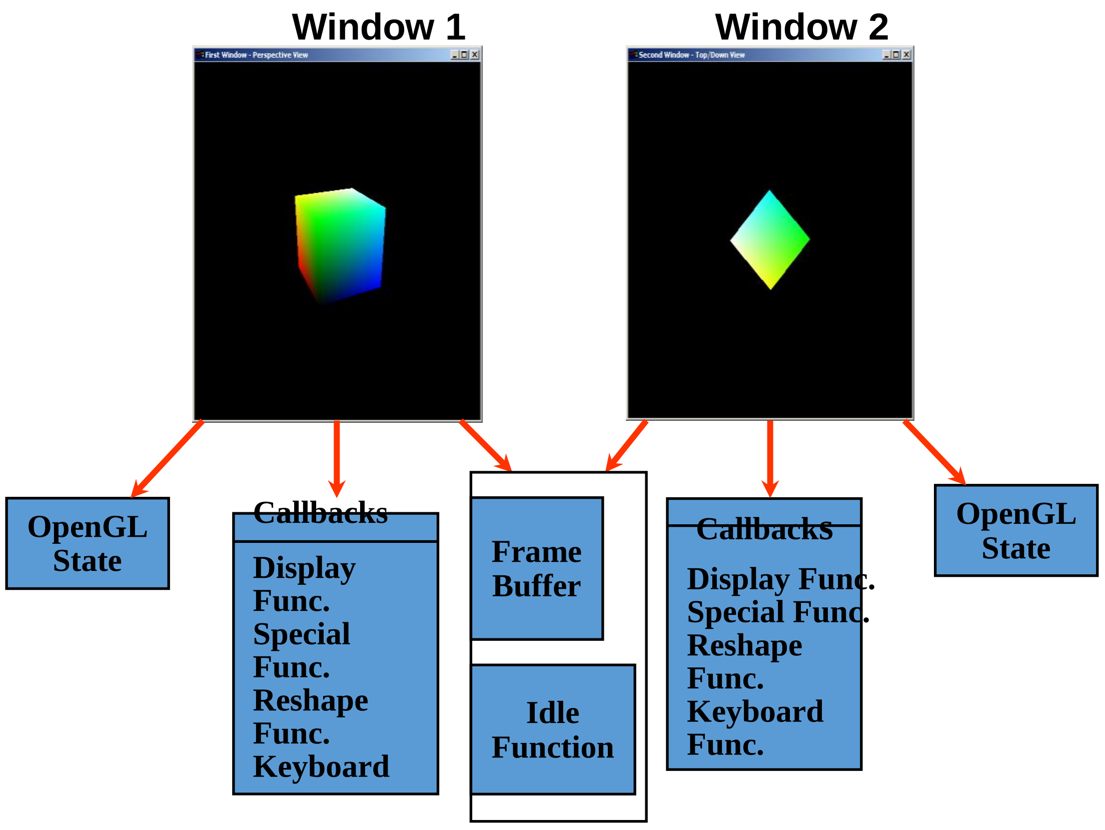


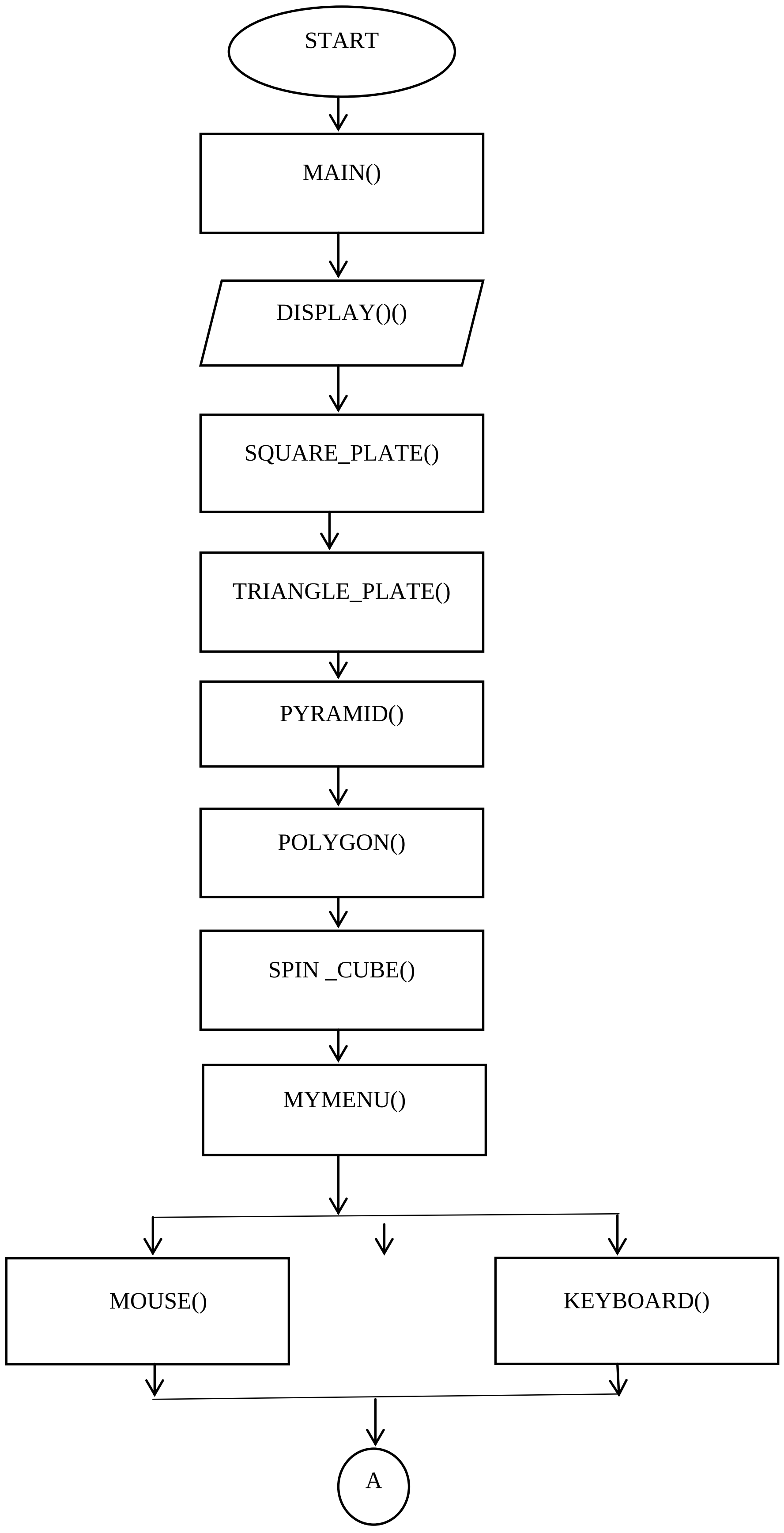
Fig3.1. Design Of Dual Window

Steps to create multiple windows:

Initialize drawing context and frame buffer using glutInit(), glutInitDisplayMode(), and glutInitWindowSize().

- Create first window
- Register callbacks for first window
- Create second window
- Position the second window
- Register callbacks for second window
- Register (shared) idle callback
- Enter the main loop

3.2 FLOWCHART



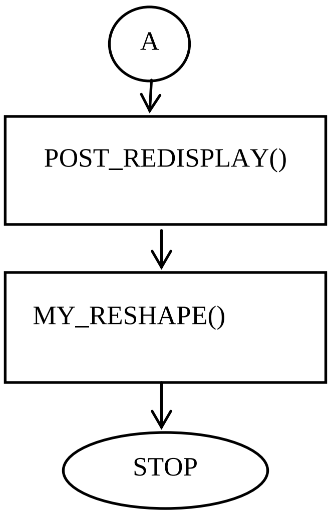


Fig 4.2: flowchart anaylsis

CHAPTER 4

IMPLEMENTATION

4.1 PSEUDOCODE

```
#include <math.h>

s#include <stdlib.h>

#include <stdio.h>

#include <GL/glut.h>

static GLfloat theta[] = {0.0,0.0,0.0};

static GLint axis = 2;

void display()

{

/* display callback, clear frame buffer and z buffer,

rotate cube and draw, swap buffers */
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

glLoadIdentity();

glRotatef(theta[0], 1.0, 0.0, 0.0);

glRotatef(theta[1], 0.0, 1.0, 0.0);

glRotatef(theta[2], 0.0, 0.0, 1.0);

switch(sel)

{

case 1:square_plate();

        break;

case 2:triangle_plate();

        break;
        case 3:colorcube();
                break;
        case 4:pyramid();
                break;
        }
        glutSwapBuffers();
        }
void display1()
{
/* display callback, clear frame buffer and z buffer,
  rotate cube and draw, swap buffers */
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glLoadIdentity();
switch(sel)
{
case 1:square_plate();
                break;
case 2:triangle_plate();
                break;
case 3:colorcube();
                break;
case 4:pyramid();
                break;
}
```

```
glutSwapBuffers();
}
void mySpecialKey(int k, int x, int y)
{
        switch(k)
        {
                case GLUT_KEY_LEFT:
                        axis=0;
                        break;
                case GLUT_KEY_RIGHT:
                        axis=1;
                        break;
                case GLUT_KEY_UP:
                        axis=2;
                        break;
        }
        glutPostRedisplay();
        }
```

4.2 OPENGL FUNCTIONS USED

The OpenGL provides very powerful translation, rotation and scaling facilities which relive the programmers by allowing them to concentrate on their job rather than focusing on how to implement these operations. OpenGL also provides viewing and modeling transformations.

The GLUT library functions used are:

- **glEnable, glDisable** - enable or disable server-side GL capabilities .

void**glEnable**(GLenumcap)

void**glDisable**(GLenumcap)

- **glTranslated,glTranslatef** – multiply the current matrix by a translation matrix.

  void**glTranslated**(GLdoublex, GLdouble*y*, GLdoublez)

  void**glTranslatef**(GLfloat, GLfloat*y*, GLfloat)

  x, *y*,z Specify the *x*, *y* and *z* coordinates of a translation vector.

- **glRotated,glRotatef** - multiply the current matrix by a rotation matrix.

  void**glRotated**(GLdoubleangle,GLdouble*x*, GLdouble*y*,GLdoublez)

  void**glRotatef(**GLfloatangle,GLfloat*x*, GLfloat*y*, GLfloatz)

- **glPushMatrix** - push the current matrix stack.

  CSPECIFICATION:

void**glPushMatrix**(void)

- **glPushMatrix** pushes the current matrix stack down by one duplicating the current matrix that is ,after a **glPushMatrix** call, the matrix on top of the stack is identical to the one below it.

The following functions are used to set the properties of the window and to create it.

This function is used to set the initial position of the window.

- **void glutInit(int *argc, char **argv);**

  argc - A pointer to the unmodifiedargc variable from the main functionargv - A pointer to the unmodifiedargv variable from the main function.

- **voidglutInitWindowPosition(int x, int y)**

PARAMETERS: *x* - the number of pixels from the left of the screen. -1 is the default value, meaning it is up to the window manager to decide where the window will appear. If not using the default values then you should pick a positive value, preferably one that will fit in your screen.

- **void glutInitWindowSize(int width, int height); :**

width - The width of the window

height- the height of the window

This function allocates the necessary buffers from the system, and adjust the structure of the frame buffer.

- **void glutInitDisplayMode(unsigned int mode)**

mode- specifies the display mode

The mode parameter is a Boolean combination (OR bit wise) of the possible predefined values the GLUTlibrary. You usemodeto specify the

colormode,and the number and type of buffers.

The predefined constants to specify the color model are:

GLUT_RGBA or GLUT_RGB – the structure of each pixel.

GLUT_DOUBLE – double is used for smooth animation, it requests the second

Frame buffer from the system.

GLUT_DEPTH - It is another buffer to hold the depth of each pixel. Now, time

has come to create a window after giving its properties.

- **intglutCreateWindow(char *title);**

title- sets the window title

The return value of glutCreateWindow is the window identifier.

A simple initialization code is as follows; if you try to execute it, a window will

show up and then dissapear suddenly.

- **glutKeyboardFunc(void (*func)(unsigned char key,intx,int y)):**

func is called when event of keypress happened. x,y specify mouse position when key is pressed. In this sample function,func is void Keyboard(unsigned char key,intx,int y).

- **glutAddMenuEntry:void** and **glutAddMenuEntry(const char * label**, **int value)**

  Append an item to the current menu. label Menu item text value Menu item callback value

  Inserts a given (label, value) pair into the current menu.label is the text displayed in the value Menu item callback value Inserts a given (label, value) pair into the current menu. label is the text displayed inmenu; value is the identifier received by the callback when the item is selected.The new entry is added to the end of the menu.

- **glutAttachMenu:voidglutAttachMenu (int button)**

  Attach the current menu to the current window. Mouse button to bind to. Associates the button with the current menu in the current window .

- **glClearColor**:specify clear values for the color buffers

  C SPECIFICATION:

voidglClearColor(GLclampf red, GLclampf green, GLclampf blue,

Glclamf alpha);

red, green, blue, alpha

Specify the red, green, blue, and alpha values used when the color buffers are

Cleared. The initial values are all 0.

glClearColor specifies the red, green, blue, and alpha values used by glClear to

clear the color buffers.

- **glMatrixMode** — specify which matrix is the current matrix

C SPECIFICATION:

voidglMatrixMode(GLenum mode);

glMatrixMode sets the current matrix mode. mode can assume one of four values:

1.GL_MODELVIEW

Applies subsequent matrix operations to the modelview matrix stack.

2.GL_PROJECTION

Applies subsequent matrix operations to the projection matrix stack.

- **glutMainLoop** enters the GLUT event processing loop.

USAGE : void glutMainLoop(void);

glutMainLoop enters the GLUT event processing loop. This routine should be called at most once in a GLUT program. Once called, this routine will never return. It will call as necessary any callbacks that have been registered.

- **glutSwapBuffers** swaps the buffers of the current window if double buffered.

USAGE :voidglutSwapBuffers(void);

Performs a buffer swap on the layer in use for the current window. Specifically, glutSwapBuffers promotes the contents of the back buffer

of the layer in use of the current window to become the contents of the front buffer. The contents of the back buffer then

become undefined. The update typically takes place during the vertical retrace of the monitor, rather than immediately after glutSwapBuffers is called.

An implicit glFlush is done by glutSwapBuffers before it returns. Subsequent OpenGL commands can be issued immediately after calling glutSwapBuffers, but are not executed until the buffer exchange is completed. If the *layer in use* is not double buffered, glutSwapBuffers has no effect.

- **glutIdleFunc** sets the global idle callback.

USAGE :voidglutIdleFunc(void (*func)(void));

glutIdleFunc sets the global idle callback to be func so a GLUT program can perform background processing tasks or continuous animation when window system events are not being received. If enabled, the idle callback is continuously called when events are not being received. The callback routine has no parameters. The current window and current menu will not be changed before the idle callback. Programs with multiple windows and/or menus should explicitly set the current window and/or current menu and not rely on its current setting.

The amount of computation and rendering done in an idle callback should be minimized to avoid affecting the program's interactive response. In general, not more than a single frame of rendering should be done in an idle callback. Passing NULL to glutIdleFunc disables the generation of the idle callback.

- **glutPostRedisplay** marks the current window as needing to be redisplayed.
  USAGE: void glutPostRedisplay(void);

Mark the normal plane of current window as needing to be redisplayed. The next iteration through glutMainLoop, the window's display callback will be called to redisplay the window's normal plane. Multiple calls to glutPostRedisplay before the next display callback opportunity generates only a single redisplay callback. glutPostRedisplay may be called within a window's display or overlay display callback to re-mark that window for redisplay.

Logically, normal plane damage notification for a window is treated as a glutPostRedisplay on the damaged window. Unlike damage reported by the window system, glutPostRedisplay will not set to true the normal plane's damaged status.

CHAPTER 5

TESTING

SCOPE

Testing is done for each module. After testing all the modules, the modules are integrated and testing of the final system is done with the test data, specially designed to show that the system will operate successfully in all its aspects conditions. The procedure level testing is made first. By giving improper inputs, the error occurred are noted and eliminated. Thus the system testing is a confirmation that all is correct and an opportunity to show the user that the system works. The final step involves Validation testing, which determines whether the software function as the user expected. The end-user rather than the system developer conduct this test most software developer as process called "Alpha and Beta test" to uncover that only the end user seems able to find.

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software testing also provides an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include, but are not limited to, the process of executing a program or application with the intent of finding software bugs (errors or other defects).

Software testing can also be stated as the process of validating and verifying that a software program/application/product:

- meets the business and technical requirements that guided its design and development;
- woSrks as expected; and
- can be implemented with the same characteristics.

Software testing, depending on the testing method employed, can be implemented at any time in the development process. However, most of the test effort occurs after the requirements have been defined and the coding process has been completed. As such, the methodology of the test is governed by the software development methodology adopted.

Different software development models will focus the test effort at different points in the development process. Newer development models, such as Agile, often employ test driven development and place an increased portion of the testing in the hands of the developer, before it reaches a formal team of testers. In a more traditional model, most of the test execution occurs after the requirements have been defined and the coding process has been completed.

5.1 UNIT TESTING

Unit testing refers to tests that verify the functionality of a specific section of code, usually at the function level. In an object-oriented environment, this is usually at the class level, and the minimal unit tests include the constructors and destructors.

These types of tests are usually written by developers as they work on code (white-box style), to ensure that the specific function is working as expected. One function might have multiple tests, to catch corner cases or other branches in the code. Unit testing alone cannot verify the functionality of a piece of software, but rather is used to assure that the building blocks the software uses work independently of each other. Unit testing is also called component testing.

5.2 INTEGRATION TESTING

Integration testing is any type of software testing that seeks to verify the interfaces between components against a software design. Software components may be integrated in an iterative way or all together ("big bang"). Normally the former is considered a better practice since it allows interface issues to be localised more quickly and fixed.

Integration testing works to expose defects in the interfaces and interaction between integrated components (modules). Progressively larger groups of tested software components corresponding to elements of the architectural design are integrated and tested until the software works as a system.

CHAPTER 6

RESULTS AND SNAPSHOTS

6.1 SNAPSHOT

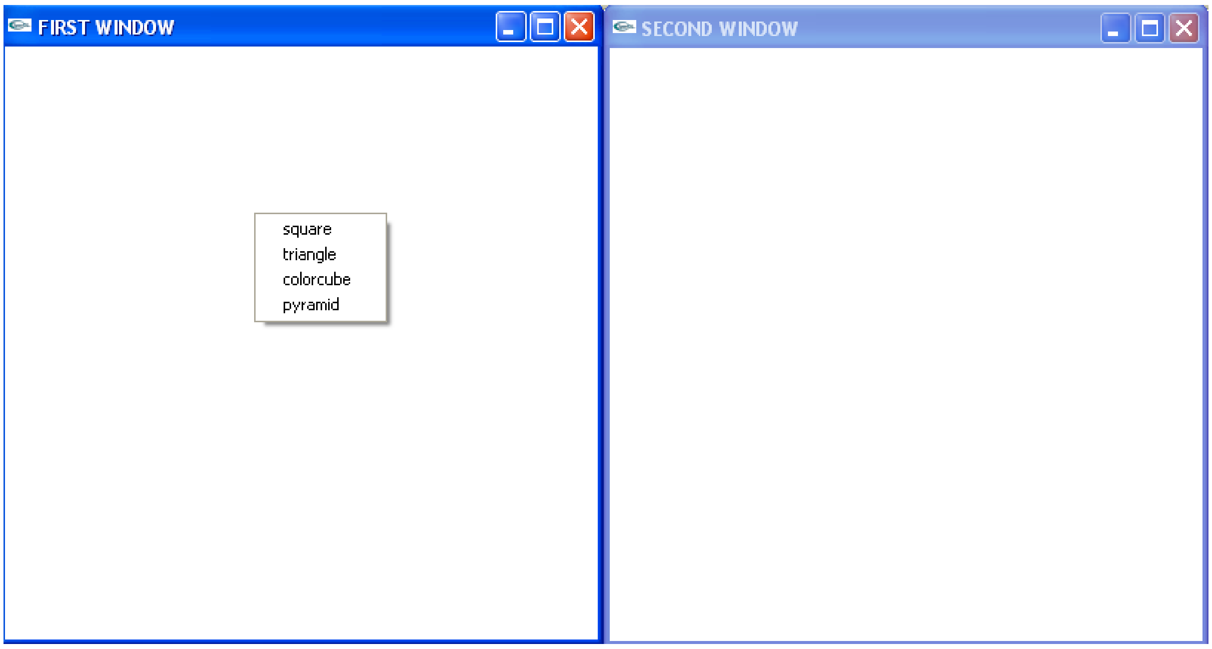


Fig6.1a:Output options

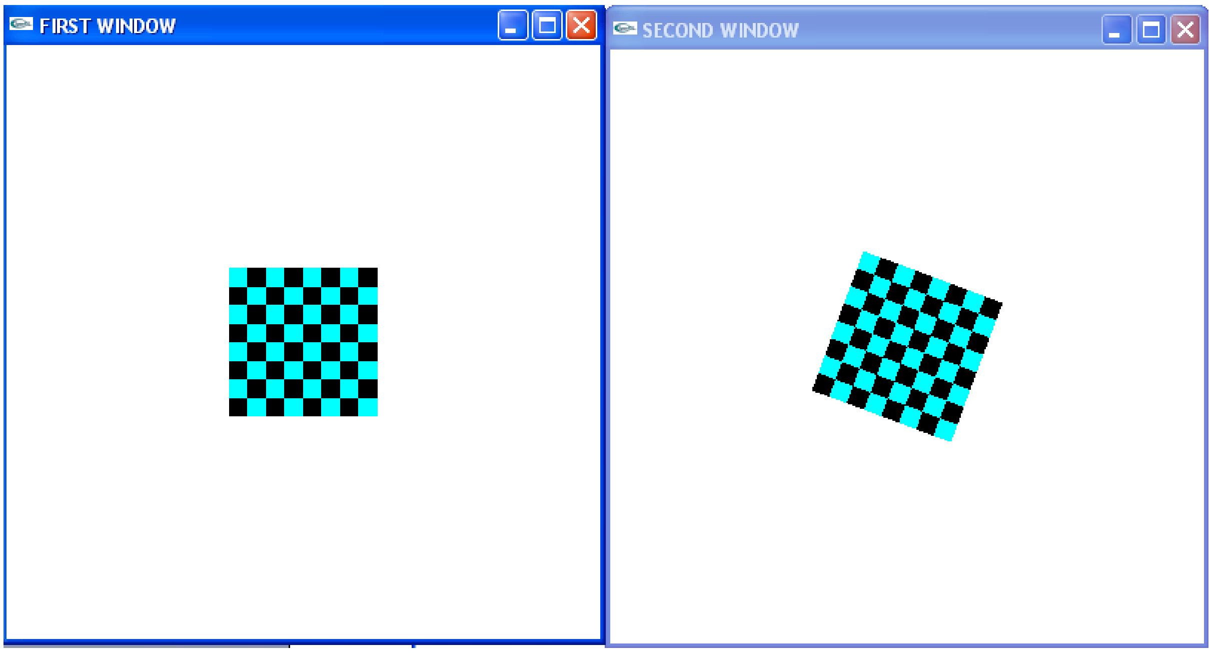


Fig 6.1b: square_plate

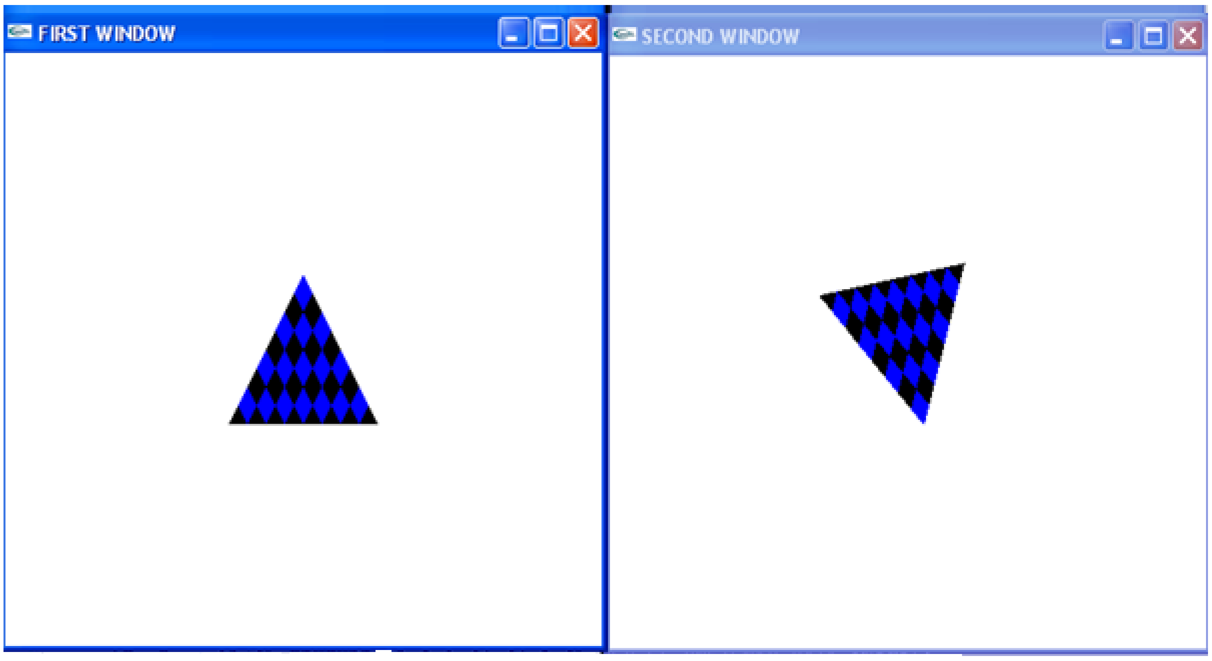


Fig 6.1c: Triangle_plate

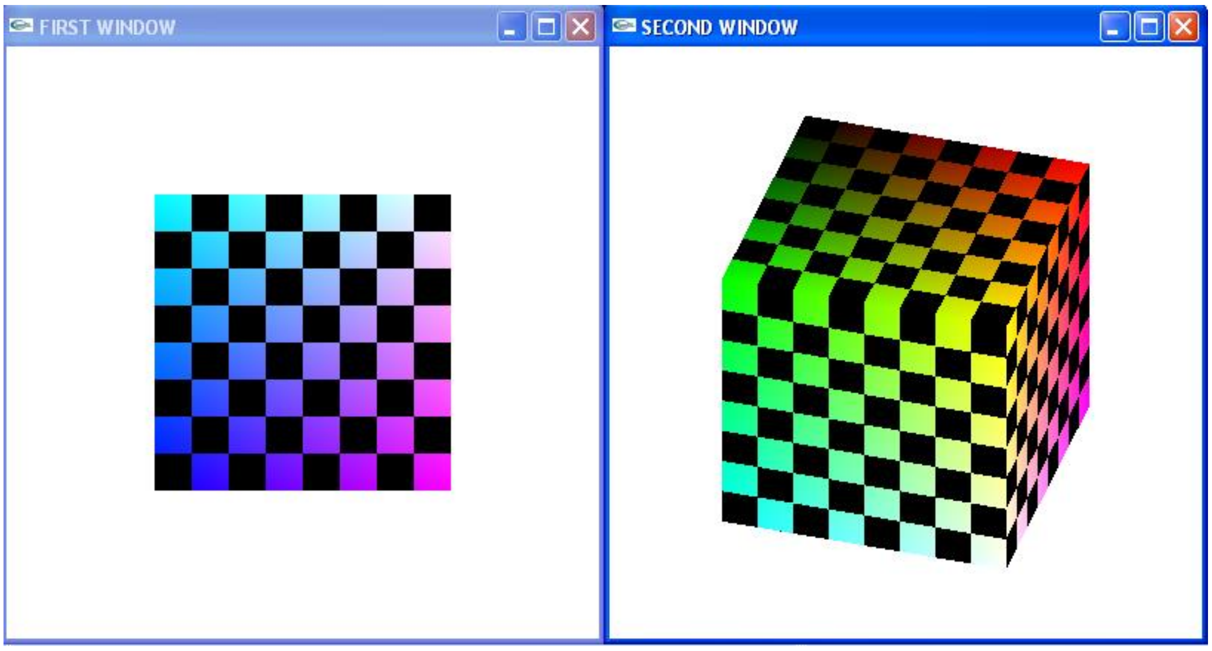


Fig 6.1d: Color cube

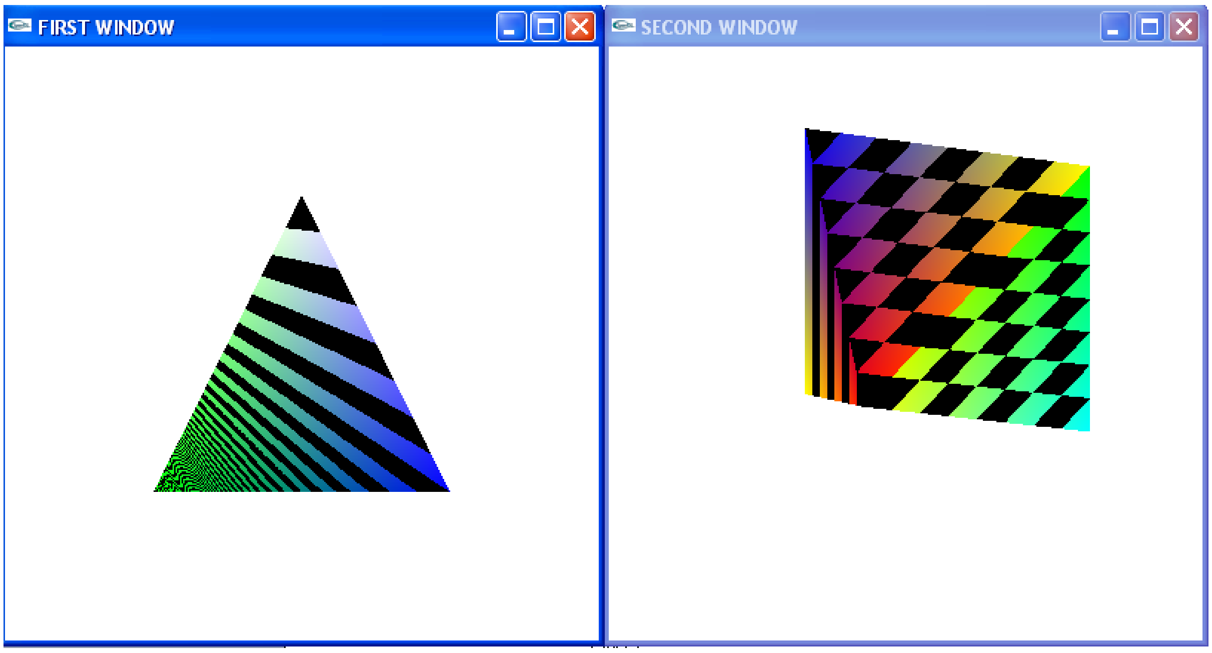
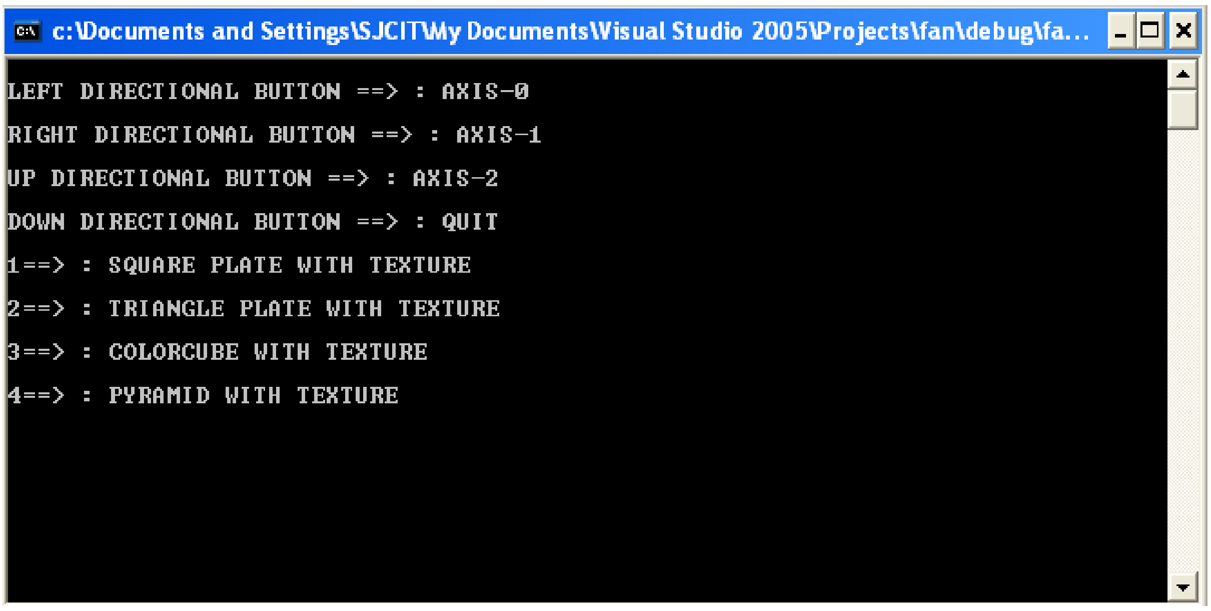


Fig6.1e: Pyramid


c:\Documents and Settings\SJCIT\My Documents\Visual Studio 2005\Projects\fan\debug\fa...

```
LEFT DIRECTIONAL BUTTON ==> : AXIS-0
RIGHT DIRECTIONAL BUTTON ==> : AXIS-1
UP DIRECTIONAL BUTTON ==> : AXIS-2
DOWN DIRECTIONAL BUTTON ==> : QUIT
1==> : SQUARE PLATE WITH TEXTURE
2==> : TRIANGLE PLATE WITH TEXTURE
3==> : COLORCUBE WITH TEXTURE
4==> : PYRAMID WITH TEXTURE
```

Fig6.1f:Comment Screen

CONCLUSION AND FUTURE ENHANCEMENT

CONCLUSION

A **Dual Window Implementation** Graphics package has been developed Using OpenGL.The illustration of graphical principles and OpenGl features are included and application program is efficiently developed.

The aim in developing this program was to design a simple program using Open GL application software by applying the skills we learnt in class, and in doing so, to understand the algorithms and the techniques underlying interactive graphics better.

The designed program will incorporate all the basic properties that a simple program must possess.

The program is user friendly as the only skill required in executing this program is the knowledge of graphics.

The main idea of the program is to implement gear wheels by creating the gears and applying rotation to the object.

FUTURE ENHANCEMENT

The aim in developing this program was to design a simple program using Open GL application software by applying the skills we learnt in class, and in doing so, to understand the algorithms and the techniques underlying interactive graphics better.

Here in our project we have implemented dual windows in such a way that the first window keeps the object (Square, triangle, colorcube, pyramid) constant without any rotation whereas in the second window it rotates about x, y and z axis as per the choice. The dual window implementation can be enhanced in future in such a way that either both the windows can rotate the object that is chosen or the window selected can make the suitable rotations on the chosen object.

In our project we have included only 4 objects (Square, triangle, colorcube, pyramid), but it can be improvised and more number of objects can be included in future as per requirements. The same project can be implemented on multiple windows also.

BIBLIOGRAPHY

BOOKS

>**Edward Angel,** "Interactive Computer Graphics", 5th edition, Pearson Education.

>**Jackie. L. Neider, Mark Warhol, Tom. R. Davis,** "OpenGL Red Book", Second Revised Edition.

>**Donald D Hearn and M. Pauline Baker,** "Computer Graphics with OpenGL",3rd Edition.

WEBSITES:

[1].http://www.opengl.org

[2].http://www.wikipedia.com

[3].http://basic4gl.wikispaces.com

[4].www.OpenGL.org/documantations

[5].www.pudn.org