**CHAPTER 1**

**INTRODUCTION**

Computer Graphics is concerned with all aspects of producing pictures or images using a computer- A particular graphics software system called OpenGL, which has become a widely accepted standard for developing graphics applications . The applications of computer graphics in some of the major areas are as follows

l. Display of information.

2. Design.

3. Simulation and Animation.

4. User interfaces.

OpenGL is a software interface to graphics hardware. This interface consists of about 150 distinct commands that are used to specify the objects and operations needed to produce interactive applications.

This project named "PENDULUM" uses OpenGL software interface and develops an oscillating pendulum. Simple techniques like rotation, transformation, motion, lighting, etc have been used.

**1.1 PROBLEM STATEMENT**

Computer graphics is no longer a rarity. It is an integral part of all computer user

Interfaces and is indispensable for visualizing 2D, 3D and higher dimensional objects.

In the proposed system, the OpenGL is an graphic software system designed as a

streamlined, hardware-independent interface to be implemented on many different hardware platforms. To achieve these qualities, no commands for performing windowing tasks or obtaining user input are included in OpenGL; instead, you must work through whatever windowing system controls the particular hardware you're using.

OpenGL doesn't provide high-level commands for describing models of three dimensional objects. Such commands might allow you to specify relatively complicated shapes such as automobiles, parts of the body, airplanes, or molecules. With OpenGL, you must build up your desired model from a small set of geometric primitives - points, lines, and polygons.

Key board

Drivers

Graphics library (API)

Application Program

Mouse

Display

**Fig1.1 Application programmers model of graphics system**

The interface between an application program and a graphics system can be specified through a set of functions the resides in a graphics library .These specification are called the application programmer's interface (API).The application programmer see only the API and is thus shielded from the details of both the hardware and the software implementation of the graphics library. The software drivers are responsible for interpreting the output of the API and converting this data to a form that is understood by the particular hardware.

**1.2 OBJECTIVES OF THE PROJECT**

The objective of this project is to demonstrate the user how this program interacts with the system and renders the graphic design of rotating objects with a trail following them.

The other objectives of this project are:

* It makes use of interactive programming.
* It provides features of transformations such as translation, rotation, and scaling.
* It makes use of light source and uses concepts of orthogonal and perspective views.

**1.3 SCOPE OF THE PROJECT**

The application program developed can be used in various fields as follows:

* In the field of science to demonstrate the working of a pendulum.
* A number of real time chaotic pendulums can be designed based on this concept.
* Educational purpose as a teaching aid.

**1.4 SUMMARY**

This chapter deals with the first phase of development of the project by knowing the drawbacks of the existing computer graphics and proposing a new system .To determine the objectives and scope of the project.

**CHAPTER 2**

**SYSTEM REQUIREMENTS**

A software requirement definition is an abstract description of the services which the system should provide, and the constraints under which the system must operate. It should only specify the external behavior of the system. The requirements are specified as below:

**3.1 FUNCTIONAL REQUIREMENTS:**

In software engineering, a functional requirement defines a function of a software system or its component. A function is described as a set of inputs, the behavior, and outputs (see also software). Functional requirements may be calculations, technical details, data manipulation and processing and other specific functionality that define what a system is supposed to accomplish. Behavioral requirements describing all the cases where the system uses the functional requirements are captured in use cases.

The various methods used in this project are as follows:-

**Init light-** The module sets the OpenGL light variables.

**Display-** The module draws the output on the screen and the functions in it

**Menu-** This module specifies the action corresponding to menu entry.

**Mouse Movement-** The module specifies the rotation with the mouse and able to see the menu.

**Idle-** This module is used to display the object more times using some delay t.

**Visible-** This module is used to make object visible to the viewer.

**Motion-**This module helps in moving the pendulum 360 degree.

**3.2 NON-FUNCTIONAL REQUIREMENTS:**

These are constraints on the services or functions offered by the system.

They include timing constraints, constraints on the development process and standards. Nonfunctional requirements often apply to the system as a whole. The Non-Functional Requirements are as follows:-

**3.2.1 Dependability**

The dependability of a computer system is a property of the system that equates to its trustworthiness. Trustworthiness essentially means the degree of user confidence that the system will operate as they expect and that the system will not 'fail' in normal use.

**3.2.2 Availability**

It is the ability of the system to deliver services when requested. There is no error in the program while executing the program.

**3.2.3 Reliability**

The ability of the system to deliver services as specified. The program is compatible with all types of operating system without any failure.

**3.2.4 Safety**

It is the ability of the system to operate without catastrophic failure. This program is user friendly and it will never effects the system

**3.2.5 Security**

It is the ability of the system to protect itself against accidental or deliberate intrusion.

**3.3 DETAILS OF THE SOFTWARE**

Here, the coding of the project is done in Microsoft Visual Studio which is a commercial integrated development environment (IDE) with OpenGL (Open Graphics Library) which is a standard specification to produce 2D and 3D computer graphics.

The OpenGL Utility Toolkit called GLUT which is a library of utilities for OpenGL programs is also used.

**3.3.1 Microsoft Visual C++**

Microsoft Visual g++ is a commercial integrated development environment (IDE) product engineered by Microsoft for the C, C++ and C#/CLI programming languages. It has tools for developing and debugging C++ code, especially code written for the Microsoft Windows API, OpenGL API, the DirectX API and the Microsoft .NET Framework.

**3.3.2 OpenGL and GLUT**

OpenGL (Open Graphics Library) is a standard specification defining a cross language, cross-platform API for writing applications that produce 2D and 3D computer graphics, describing a set of functions and the precise behaviors that they must perform. From this specification, hardware vendors create implementations - libraries of functions created to match the functions stated in the OpenGL specification, making use of hardware acceleration where possible. Hardware vendors have to meet specific tests to be able to qualify their implementation as an OpenGL implementation.

GLUT is the OpenGL Utility Toolkit, a window system independent toolkit for writing

Programs in OpenGL. It implements a simple windowing application programming interface (API) for OpenGL. GLUT makes it considerably easier to learn about and explore OpenGL programming. GLUT provides a portable API so you can write a single OpenGL program that works across all PC and workstation OS platforms.

**3.4 SOFTWARE REQUIREMENTS**

* OpenGL(Graphics Library Utility Toolkit)
* Microsoft Visual C++ 6.0
* Operating System- Windows XP

**3.5 HARDWARE REQUIREMENTS:**

* Hard Disk - 30 GB and Above
* Memory - 256 MB and Above
* Mouse - 3 button mouse
* Keyboard
* Visual Display Unit

**CHAPTER 3**

**PROBLEM DESCRIPTION**

**LEAVE MANAGEMENT SYSTEM**

ABC COLLEGE OF ENGINEERING has Teaching and Non-Teaching staff categories. Professors, Associate professors and Assistant professors come under Teaching and Lab-Instuctors under Non-Teaching. Teaching faculty has 15 CL’s and Non-teaching has 12 CL’s in a year. The college uses a LEAVE MANAGEMENT SYSTEM to track the different types of leave details for each faculty. The system uses the following tables for maintaining this details.

* FACULTY
* FACULTYLEAVEDETAILS
* APPLIEDLEAVEDETAILS

The table details are as follows

FACULTY

|  |  |  |  |
| --- | --- | --- | --- |
| **COLUMN NAME** | **DTATYPE & SIZE** | **CONSTRAINTS** | **DESCRIPTION** |
| FID | VARCHAR(6) | PRIMARY KEY | The facultyID column accepts alphanumerics of length 6 charcter |
| FNAME | VARCHAR(30) |  | Full name of the faculty |
| DEPT | CHAR(2) | NOT NULL | Department is of charecter of length 2 |
| DESG | CHAR(7) |  | Use PROF for Professor and ASSTP for Associate Professor AP for Assistant Profesor. LABINST for LabInstrectors |
| DOJ | DATE | NOT NULL | Date of Joining of faculty |
| SALARY | FLOAT(6,2) |  | Salary of faculties |
| CATEG | CHAR(2) | NOT NULL | Use ‘T’ for Teaching and ‘NT’ for Non-Teaching |

FACULTYLEAVEDETAILS

|  |  |  |  |
| --- | --- | --- | --- |
| **COLUMN NAME** | **DTATYPE & SIZE** | **CONSTRAINTS** | **DESCRIPTION** |
| FID | VARCHAR(6) | PRIMARY KEY AND REFERENCES FID FROM FACULTY | The facultyID column accepts alphanumerics of length 6 charcter and should be there in FACULTY table |
| CL | FLOAT(2,1) |  | CASUAL LEAVE |
|  |  |  |  |
| OOD | FLOAT(2,1) |  | OFFICIAL ON DUTY |
| SL | FLOAT(2,1) |  | SICK LEAVE |
| VL | FLOAT(2,1) |  | VACATION LEAVE |
| EL | FLOAT(2,1) |  | EARNED LEAVE |
| LOP | FLOAT(2,1) |  | LOSS OF PAY |

APPLIEDLEAVEDETAILS

|  |  |  |  |
| --- | --- | --- | --- |
| **COLUMN NAME** | **DTATYPE & SIZE** | **CONSTRAINTS** | **DESCRIPTION** |
| AID | INT | PRIMARY KEY | Application ID |
| FID | VARCHAR(6) | REFERENCES FID FROM FACULTY | FID from FACULTY table |
| TYPEOFLEAVE | VARCHAR(4) |  | CL,OOD,VL,EL,SL,LOP |
| DATEFROM | DATE |  | Leave start date |
| DATETO | DATE |  | Leave end date |
| NOOFDAYS | FLOAT(2,1) |  | Number of days |
| STATUSS | CHAR(10) |  | ‘PENDING’ or ‘APPROVED’ |
| CLASSALTERED | CHAR(2) |  | For Teaching ‘Y’ or ‘N’ and Non-Teaching ‘NA’ |

**CHAPTER 4**

**SYSTEM DESIGN**

**(Design should include ER Diagram and Schema Diagram of your Database)**

Graphics systems used general - purpose computers with the standard von Neumann architecture. Such computers are characterized by a single processing unit that processes a single instruction at a time. Information had to be sent to the display at a rate high enough to avoid flicker on the display. In the early days of computer graphics, computers were so slow that refreshing even simple images, containing a few hundred line segments, would burden an expensive computer.

The following diagram shows the Henry ford assembly line approach which OpenGL takes to process data.



**4.1 Display Processors:**

Built special-purpose graphics systems were concerned primarily with relieving the general purpose computer from the task of refreshing the display continuously. These display processes had conventional architectures but included instructions to display primitives on the CRT. The main advantage of the display processor was that the instructions to generate the image could be assembled once in the host and sent to the display processor, where they were stored in the display processor's own memory as a display list, or display file. The display processor would then repetitively execute the

program in the display list, at a rate sufficient to avoid flickering independently of the host, thus freeing the host for other tasks. This architecture has become closely associated with the client-server architectures.

**4.2 Pipeline Architectures:**

The major advances in graphics architectures closely parallel the advances in work stations. In both cases, the ability to create special-purpose VLSI chops was the key enabling technology development. In addition, the availability of inexpensive solid state memory led to the universality of raster displays. For computer-graphics applications, the most important use of customs VLSI circuits has been in creating pipeline architectures.

**4.3 The graphics pipeline:**

3 major steps in the imaging process:

* **Vertex processing**
* **Clipping and primitive assembly**
* **Fragment processing**

**4.3.1 Vertex Processing:**

In the first block of our pipeline, each vertex is processed independently. The two major functions of the block are to carry out coordinate transformations and to compute a color for each vertex. The assignment of vertex colors can be as simple as the program specifying a color or as complex as the computation of a color from a physically realistic lighting model that incorporates the surface properties of the object and the characteristic light sources in the scent.

**4.3.2 Clipping and primitive assembly:**

The second fundamental block in the implementation of the standard graphics pipeline is for clipping and primitive assembly. We must do clipping because of the limitation that no imaging system can see the whole world at once. The human retina has a limited size corresponding to an approximately 9O-degree field of view. Cameras have a film of limited size, and we can adjust their fields of view by selecting different lenses.

**4.3.3 Rasterization**.

The primitives that emerge from the clipper are still represented in terms of their vertices and must be further processed to generate pixels in the frame buffer. For example, if three vertices specify a triangle filled with a solid color, the raster must determine which pixels in the frame buffer are inside the polygon. The output of rasterization is a set fragment for each primitive. A fragment can be thought of as a potential pixel that carries with it information, including its color and location, that is used to update the corresponding pixel in the frame buffer. Fragment can also carry along depth information that allows later stages to determine if a particular fragment lies behind other previously rasterized fragments for a given pixel.

**CHAPTER 5**

**IMPLEMENTATION**

* **(need not to have complete code, should contain important modules of your code)**
* **glPushMatrix**

**Syntax:** glPushMatrix( );

**Purpose**: It saves the presentvalues of attributes and matrices by placing or pushing them on top of the appropriate stack.

* **glPopMatrix**

**Syntax:** glPopMatrix( );

**Purpose:** It recovers the attributes and matrices by removing or popping them from the stack.

* **glloadldentity**

**Syntax**: glloadldentityQ;

**Purpose:** It loads the idartity matrix.

* **glMatrixMode**

**Syntax:** a) glMatrixMode(GL\_PROJECTION);

b) glMatrixMode(GL\_MODEL\_VIEW);

**Purpose:** It helps in selecting the matrix(model view or projection matrix) to which the operations apply by setting the matrix mode a variable that is set to one tlpe of matrix and also part of the state.

* **glOrtho**

**Syntax:** glorthd Gldouble left, Gldouble right, Gldouble bottom, Glxlouble top, Gldouble near, Gldouble far);

**Purpose**: It gives an orthogmphic projection with a right parallelepiped viewing volume. All the parameters me distances measured from the camera.

* **glBegin**

**Syntax:** glBegin(glEnum mode);

**Purpose:** Initiates a new primitive of type mode and and starts the collection of vertices. Values of mode include GL\_POINTS, GL\_LINES and GL\_POLYGON.

* **glEnd**

**Syntax:** glEnd0;

**Purpose:** Terminates a list of vertices.

* **glutlnit**

**Syntax**: glutlnit( int \*argc, char \*\* argv);

**Purpose:** Initializes GLUT. The arguments from main are passed

In and can be used by the application.

* **glVertex**

**Syntax:**a)glVertex[23a][sifd( TYPE xcoordinate, TYPE ycoordinate, .);

b) glVertex[B4][sifd]v(TYPE \*coordinate);

**Purpose:** Specifies the position of a vertex inZ,3,or 4 dimensions. The coordinates can be specifie as short s, int i, float f or double d. If the v is presant, the argument is a pointer to an array containing the coordinates.

* **glutCreateWindow**

**Syntax:** glutCreateWindow(char \* title);

**Purpose:** It creates a window on the display. The string title can be used to label the window.

* **glutlnitDisplayMode**

**Syntax**: glutlnitDisplayMode( unsigned int mode);

**Purpose:** It requests a display with the properties in mode. The value of mode is determined by the logical OR of options including the color model(GLUT\_RGB, GLUT\_INDEX) and buffering(GLUT\_SINGLE, GLUT\_DOUBLE).

* **glutMainloop**

**Syntax:** glutMainlLoop0;

**Purpose:** It causes the program to enter an event processing loop. It should be the last statement in main.

* **glutKeyboardFunc**

**Syntax:** glutKepoardFunc(void \*(char key, int width, int height)

**Purpose:** Registers the keyboard callback function f. The callback function returns the ASCtr code of the key pressed and the position of the mouse.

**CHAPTER 6**

**SCREEN SHOTS**

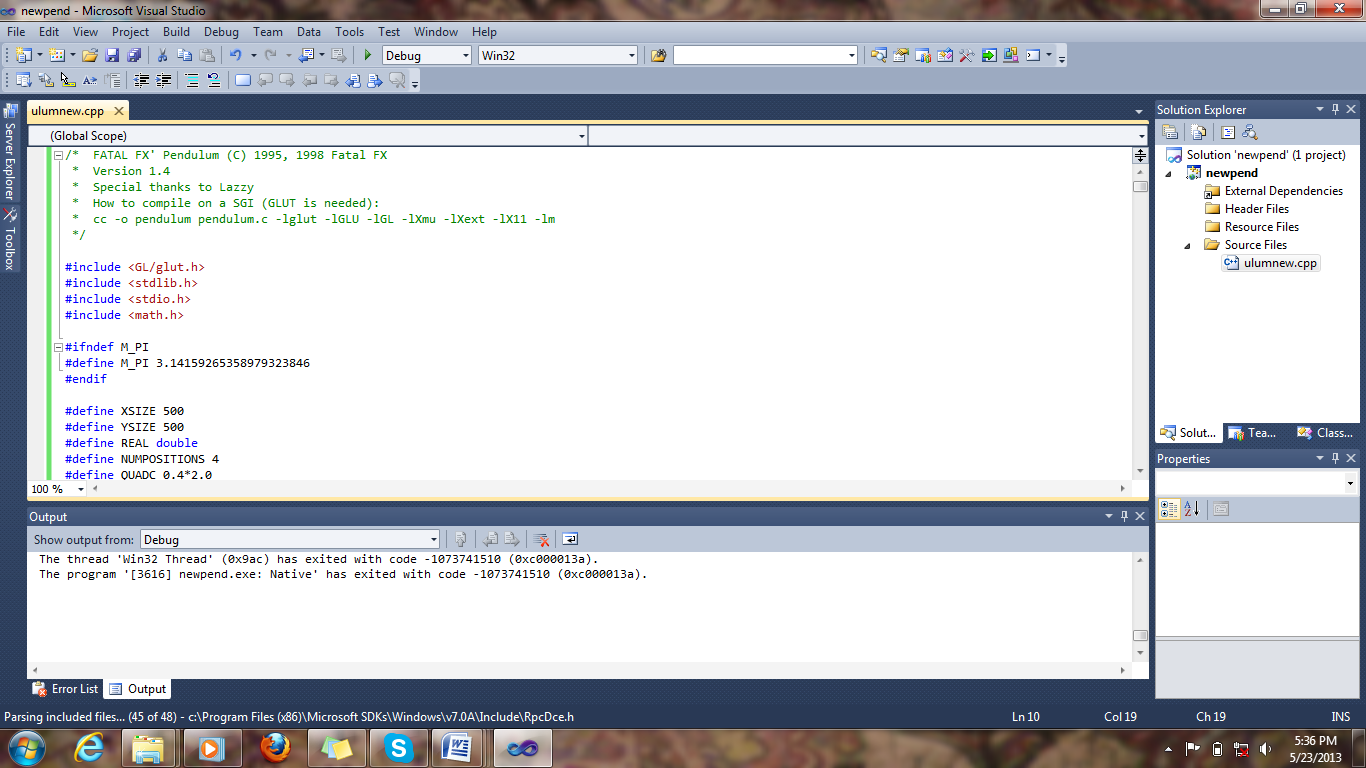
****

Figure 7.1 LOGIN PAGE

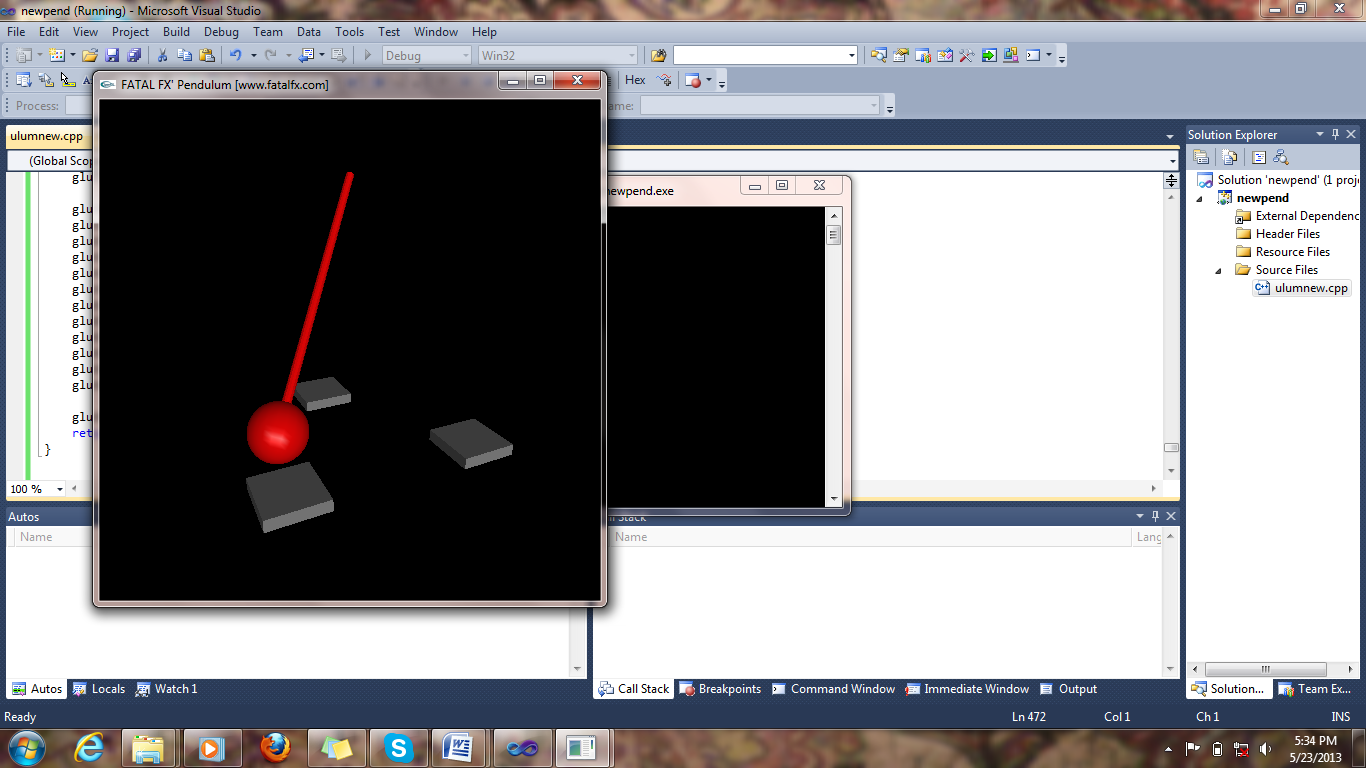


Figure 7.2 USER DETAILS

**CHAPTER 7**

**CONCLUSION & FUTURE SCOPE**

In this project, the design and implementation of 3-dimensional graphics Interfacing program has been attempted. We have seen the functions performed by this application and code for the same. In OpenGL, C/C++ provide enormous flexibility in the design and the use of C/C++ graphics programs.

The presence of many in-built OpenGL functions and libraries to take care of many of the functionalities reduces the burden of coding and makes the implementation simpler.

The project started with the designing phase in which we figured the requirements needed the system design, Data flow diagram etc. Then comes the details of the implementation phase where in we have included various functionalities. And now after the testing phase, the project comes to an end.

.