Visvesvaraya Technological University
Jnana Sangam, Belgaum-590018, Karnataka

A mini-project on

DEMONSTRATION OF "FLOWING FOUNTAIN"

Submitted in Partial fulfilment of the requirement as a part of
VI Semester of Computer Graphics Laboratory

BACHELOR OF ENGINEERING
IN
COMPUTER SCIENCE AND ENGINEERING

Submitted By:

BHARGAV K NAIDU
1CG12CS020

Guide

Mrs Veena N D M.Tech
Asst.Professor
Dept. of CSE,
CIT, Gubbi, Tumkur.

HOD

Prof. Shantala C P M.Tech.,(Ph.D.)
Professor & HOD,
Dept. of CSE,
CIT, Gubbi, Tumkur.

Department of Computer Science & Engineering

Channabasaveshwara Institute of Technology
(An ISO 9001:2008 Certified Institution)
NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka.

May-2015

Channabasaveshwara Institute of Technology
(An ISO 9001:2008 Certified Institution)
NH 206 (B.H. Road), Gubbi, Tumkur – 572 216. Karnataka.

Department of Computer Science & Engineering

CERTIFICATE

This is to certify that the project entitled "**FLOWING FOUNTAIN**" is Bonafide work carried out by BHARGAV K NAIDU bearing USN **1CG12CS020** respectively as a partial fulfilment for the award of Bachelor's degree in COMPUTER SCIENCE AND ENGINEERING for COMPUTER GRAPHICS AND VISUALIZATION LAB as prescribed by **VISVESVARAYA TECHNOLOGICAL UNIVERSITY**, **BELGAUM** during the year 2014-15.

Signature of Guide **Signature of HOD**

Mrs.Veena N D M.Tech
Asst.Professor
Dept of CSE
C.I.T, Gubbi, Tumkur

Prof. Shantala C P M.Tech.,(Ph.D.)
Professor & HOD
Dept of CSE
C.I.T,Gubbi,Tumkur

Name of Examiner **Signature of Examiner**

1)

2)

ACKNOWLEDGEMENT

At the outset I express my most sincere grateful thanks to the holy sanctum **"CHANNABASAVESHWARA INSTITUTE OF TECHNOLOGY"** the temple of learning, for giving us an opportunity to pursue the degree course in Computer Science and Engineering thus help shaping our carrier. We wish to extend our gratitude to our beloved **Principal Dr. Suresh Kumar D S** for encouragement throughout our project.

I express my heartiest indebtedness to our **Head of the Department Prof. Shantala C P** for encouragement throughout our project.

I also wish to express my deep sense of gratitude to my **Project Guide**, **Mrs. Veena N D** department of CSE for their continuous and tireless support and advice not only during the course of our project but also during the period of our stay in C.I.T.

Finally I express my gratitude to all teaching staff of **Dept. of CSE** for their timely support and suggestions.

I am conscious of the fact that I have received co-operation in many ways from the teaching and non-teaching staff of the department of Computer Science and Engineering and we are grateful to all of their cooperation and their guidance in completing our task well is time. I thank one and all who have been helped to us in one way or the other in completing our project on time.

Project Associate

Bhargav K Naidu

Abstract

Computer Graphics is the creation, manipulation, and storage of models and images of picture objects by the aid of computers. This was started with the display of data on plotters and CRT. Computer Graphics is also defined as the study of techniques to improve the communication between user and machine, thus Computer Graphics is one of the effective media of communication between machine and user.

Computer Graphics is one of the hottest buzzword in the Information Technology today, with graphics assuming importance in every field. There is a wide scope for graphics based software. Some of the areas in which graphics has made in roads are desktop publishing, image processing, animation, movies etc.

In our package we are clearly going to demonstrate the working of ” Flowing Fountain”. Viewer can change the colors of the fountain through mouse interface and navigate between menus and actual output screen through keyboard interface. Viewer can also view different views of the fountain through keyboard interface.

We have developed this package using OpenGL. OpenGL is a software interface to graphics hardware. The OpenGL Utility Library (GLU) provides many of the modeling features. GLU is a standard part of every OpenGL implementation.

CONTENTS

Page No.

CHAPTER 1

INTRODUCTION

1.1 Computer Graphics

To draw a picture say, fish moving inside the water. Suddenly will get an idea to use paint, but the degree of accuracy, quality of image is not satisfied and become very sad. There is no need to worry, for every problem there will be a solution, so this problem of creating fish moving inside the water can be solved using COMPUTER GRAPHICS without any difficulties.

Computer Graphics become a powerful tool for the rapid and economical production of pictures. There is virtually no area in which Graphical displays cannot be used to some advantage so it is not surprising to find the use of CG so widespread.

Although early application in engineering & science had to rely on expensive & cumbersome equipments, advances in computer technology have made interactive computer graphics a practical tool.

Computer Graphics in a diverse area such as science, engineering, medicine, business, industry, government, art, entertainment, education and training.

Now it can be answered about computer graphics as generalized tool for drawing and creating pictures and simulates the real world situations within a small computer window.

1.2 History

William fetter was credited with coning the term Computer Graphics in 1960, to describe his work at Boeng. One of the first displays of computer animation was future world (1976), which included an animation of a human face and hand-produced by Carmull and Fred Parkle at the University of Utah.

There are several international conferences and journals where the most significant results in computer-graphics are published. Among them are the SIGGRAPH and Euro graphics conferences and the association for computing machinery (ACM) transaction on Graphics journals.

1.3 APPLICATIONS OF COMPUTER GRAPHICS

Nowadays Computer Graphics used in almost all the areas ranges from science, engineering, medicine, business, industry, government, art, entertainment, education and training.

1.3.1 CG in the field of CAD

Computer Aided Design methods are routinely used in the design of buildings, automobiles, aircraft, watercraft, spacecraft computers, textiles and many other applications.

1.3.2 CG in presentation Graphics

Another major application area presentation graphics used to produce illustrations for reports or generate slides. Presentation graphics is commonly used to summarize financial, statistical, mathematical, scientific data for research reports and other types of reports.2D and 3D bar chart to illustrate some mathematical or statistical report.

1.3.3 CG in computer Art

CG methods are widely used in both fine art and commercial art applications. Artists use a variety of computer methods including special purpose hardware, artist's paintbrush program (lumena), other pain packages, desktop packages, maths packages, animation packages that provide facility for designing object motion. Ex: cartoons design is an example of computer art which uses CG.

1.3.4 Entertainment

Computer graphics methods are now commonly used in making motion pictures, music, videos, games and sounds. Sometimes graphics objects are combined with the actors and live scenes.

1.3.5 Education and Training

Computer generated models of physical financial, economic system is often as education aids. For some training application special systems are designed. Ex: specialized system is simulator for practice sessions or training of ship captain, aircraft pilots and traffic control.

1.3.6 Image Processing

Although the methods used in CG image processing overlap, the 2 areas are concerned with fundamentally different operations. In CG a computer is used to create picture. Image processing on the other hand applies techniques to modify existing pictures such as photo scans, TV scans.

1.3.7 User Interface

It is common for software packages to provide a graphical interface. A major component of a graphical interface is a window manager that allows a user to display multiple window area. Interface also displays menus, icons for fast selection and processing.

1.4 Statement of the Problem

The first automatic machine guns had **recoil-based systems**. When we propel a bullet down the barrel, the forward force of the bullet has an opposite force that pushes the gun backward. In a gun built like a revolver, this recoil force just pushes the gun back at the shooter. But in a recoil-based machine gun, moving mechanisms inside the gun absorb some of this recoil force.

1.5 Objectives

- The interactive demo of Machine gun recoil system.
- Graphical approach towards understanding the machine gun recoil system.

1.6 Organization of the Report

- Chapter 1 introduces the computer graphics and its applications.
- Chapter 2 Introduction to OpenGL
- Chapter 3 Basic elements of OpenGL.
- Chapter 4 Working procedure of Flowing Fountain.
- Chapter 5 Deals with design and implementation inbuilt graphics functions and hardware and software requirement with results obtained,
- Chapter 6 Snapshots and results of Flowing Fountain.
- Chapter 7 Conclusion and future scope.

CHAPTER 2

Introduction to OpenGL

Most of the application will be designed to access OpenGL directly through functions in three libraries. Functions in the main GL (or OpenGL in windows) library have names that begin with the letters gl and are stored in a library usually referred to as GL (or OpenGL in windows). The second is the **OpenGL Utility Library** (GLU). This library uses only GL functions but contains code for creating common objects and simplifying viewing. All functions in GLU can be created from the core GL library but application programmers prefer not to write the code repeatedly. The GLU library is available in all OpenGL implementations; functions in the GLU library begin with letters glu.

To interface with the window system and to get input from external devices into the programs, need at least one more system-specific library that provides the "glue" between the window system and OpenGL. For the X window system, this library is functionality that should be expected in any modern windowing system.

Fig 2.1 shows the organization of the libraries for an X Window System environment. For this window system, GLUT will use GLX and the X libraries. The application program, however, can use only GLUT functions and thus can be recompiled with the GLUT library for other window systems.

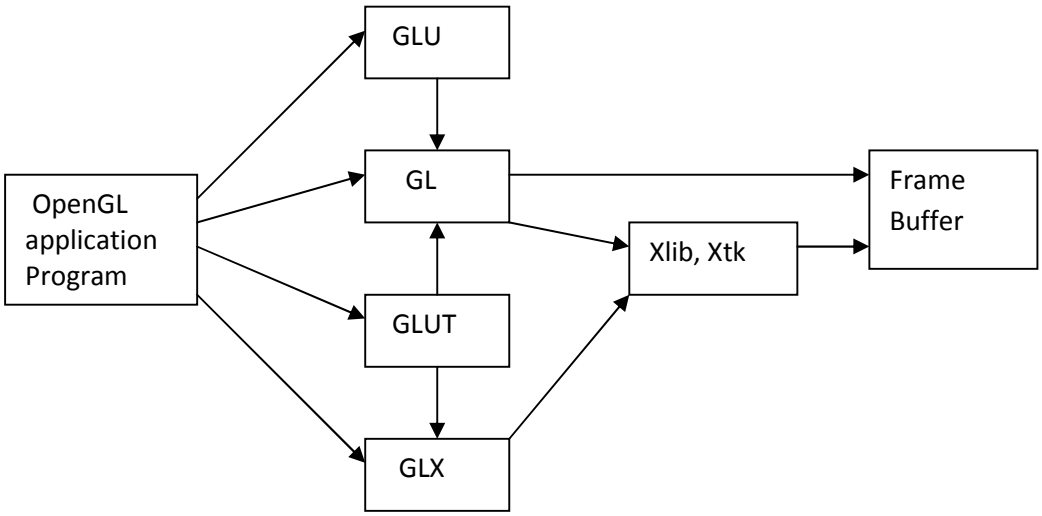


Fig 1.1 Library organization of OpenGL

2.1 OpenGL Command Syntax

As you might have observed from the simple program in the previous section, OpenGL commands use the prefix and initial capital letters for each word making up the command name (recall **glClearColor(),** for example). Similarly, OpenGL defined constants begin with GL_, use all capital letters, and use underscores to separate words (like GL_COLOR_BUFFER_BIT).

Table 1-1: Command Suffixes and Argument Data Types

Suffix	Data Type	Typical Corresponding C-Language Type	OpenGL Type Definition
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 integer	unsigned char	GLubyte, GLboolean
Us	16-bit unsigned integer	unsigned short	GLushort
Ui	32-bit unsigned integer	unsigned int or unsigned long	GLuint,GLenum, GLbitfield

2.2 OpenGL as a State Machine

OpenGL is a state machine. You put it into various states (or modes) that then remain in effect until you change them. As you've already seen, the current color is a state variable. You can set the current color to white, red, or any other color, and thereafter every object is drawn with that color until you set the current color to something else. The current color is only one of many state variables that OpenGL maintains. Others control such things as the current viewing and projection transformations, line and polygon stipple patterns, polygon drawing modes, pixel-packing conventions, positions and characteristics of lights, and material properties of the objects being drawn. Many state variables refer to modes that are enabled or disabled with the command **glEnable()** or **glDisable()**. Each state variable or mode has a default value, and at any point you can query the system for each variable's current value. Typically, you use one of the six following commands to do this: **glGetBooleanv()**, **glGetDoublev()**, **glGetFloatv()**, **glGetIntegerv()**, **glGetPointerv()**, or **glIsEnabled()**. Which of these commands you select depends on what data type you want the answer to be given in. Some state variables have a more specific query command (such as **glGetLight*()**, **glGetError()**, or **glGetPolygonStipple()**). In addition, you can save a collection of state variables on an attribute stack with **glPushAttrib()** or **glPushClientAttrib()**, temporarily modify them, and later restore the values with **glPopAttrib()** or **glPopClientAttrib()**. For temporary state changes, you should use these commands rather than any of the query commands, since they're likely to be more efficient.

2.3 Display Lists

All data, whether it describes geometry or pixels, can be saved in a *display list* for current or later use. (The alternative to retaining data in a display list is processing the data immediately - also known as *immediate mode*.) When a display list is executed, the retained data is sent from the display list just as if it were sent by the application in immediate mode. (See Chapter 7 for more information about display lists.)

2.4 Evaluators

All geometric primitives are eventually described by vertices. Parametric curves and surfaces may be initially described by control points and polynomial functions called basis functions. Evaluators provide a method to derive the vertices used to represent the surface

from the control points. The method is a polynomial mapping, which can produce surface normal, texture coordinates, colors, and spatial coordinate values from the control points. (See Chapter 12 to learn more about evaluators.)

2.5 Per-Vertex Operations

For vertex data, next is the "per-vertex operations" stage, which converts the vertices into primitives. Some vertex data (for example, spatial coordinates) are transformed by 4 x 4 floating-point matrices. Spatial coordinates are projected from a position in the 3D world to a position on your screen. If advanced features are enabled, this stage is even busier. If texturing is used, texture coordinates may be generated and transformed here. If lighting is enabled, the lighting calculations are performed using the transformed vertex, surface normal, light source position, material properties, and other lighting information to produce a color value.

2.6 Primitive Assembly

Clipping, a major part of primitive assembly, is the elimination of portions of geometry which fall outside a half-space, defined by a plane. Point clipping simply passes or rejects vertices; line or polygon clipping can add additional vertices depending upon how the line or polygon is clipped. In some cases, this is followed by perspective division, which makes distant geometric objects appear smaller than closer objects. Then viewport and depth (z coordinate) operations are applied. If culling is enabled and the primitive is a polygon, it then may be rejected by a culling test. Depending upon the polygon mode, a polygon may be drawn as points or lines. The results of this stage are complete geometric primitives, which are the transformed and clipped vertices with related color, depth, and sometimes texture-coordinate values and guidelines for the rasterization step.

2.7 Pixel Operations

While geometric data takes one path through the OpenGL rendering pipeline, pixel data takes a different route. Pixels from an array in system memory are first unpacked from one of a variety of formats into the proper number of components. Next the data is scaled,

biased, and processed by a pixel map. The results are clamped and then either written into texture memory or sent to the rasterization step. If pixel data is read from the frame buffer, pixel-transfer operations (scale, bias, mapping, and clamping) are performed. Then these results are packed into an appropriate format and returned to an array in system memory. There are special pixel copy operations to copy data in the framebuffer to other parts of the framebuffer or to the texture memory. A single pass is made through the pixel transfer operations before the data is written to the texture emory or back to the framebuffer.

2.8 Texture Assembly

An OpenGL application may wish to apply texture images onto geometric objects to make them look more realistic. If several texture images are used, it's wise to put them into texture objects so that you can easily switch among them. Some OpenGL implementations may have special resources to accelerate texture performance. There may be specialized, high-performance texture memory. If this memory is available, the texture objects may be prioritized to control the use of this limited and valuable resource.

2.9 Rasterization

Rasterization is the conversion of both geometric and pixel data into *fragments*. Each fragment square corresponds to a pixel in the framebuffer. Line and polygon stipples, line width, point size, shading model, and coverage calculations to support antialiasing are taken into consideration as vertices are connected into lines or the interior pixels are calculated for a filled polygon. Color and depth values are assigned for each fragment square.

2.10 Fragment Operations

Before values are actually stored into the framebuffer, a series of operations are performed that may alter or even throw out fragments. All these operations can be enabled or disabled.

CHAPTER 3

BASIC OPENGL ELEMENTS

3.1 Basic 2D Drawing

glBegin (mode);

glVertex2f (x_1, y_1);
glVertex2f (x_2, y_2);
glVertex2f (x_3, y_3);
glVertex2f (x_4, y_4);

glEnd ();

where mode = GL_POINTS, GL_LINES, GL_LINE_STRIP, GL_POLYGON, GL_LINE_LOOP, etc.

Lines of zero length are not visible.

A vertex is considered to be just a location with zero mass.

Only 10 of the OpenGL functions are legal for use between glBegin and glEnd.

3.2 Some Primitive Attributes

glClearColor (red, green, blue, alpha); - Default = (0.0, 0.0, 0.0, 0.0)
glColor3f (red, green, blue); - Default = (1.0, 1.0, 1.0)
glLineWidth (width); - Default = (1.0)
glLineStipple (factor, pattern) - Default = (1, 0xffff)
glEnable (GL_LINE_STIPPLE);
glPolygonMode (face, mode) - Default = (GL_FRONT_AND_BACK, GL_FILL)
glPointSize (size); - Default = (1.0)

3.3 Projection Transformations

glMatrixMode (GL_PROJECTION);

glLoadIdentity ( );

glFrustum (left, right, bottom, top, near, far);

gluPerspective (fov, aspect, near, far);

glOrtho (left, right, bottom, top, near, far); - Default = (-1.0, 1.0, -1.0, 1.0, -1.0, 1.0)

gluOrtho2D (left, right, bottom, top);

3.4 Modelview Transformations

glMatrixMode (GL_MODELVIEW);

glLoadIdentity ( );

gluLookAt (eye_x, eye_y, eye_z, at_x, at_y, at_z, up_x, up_y, up_z);

glTranslatef (dx, dy, dz);

glScalef (sx, sy, sz);

glRotatef (angle, axisx, axisy, axisz);

3.5 Writing Bitmapped Text

glPixelStorei (GL_UNPACK_ALIGNMENT, 1);

glColor3f (red, green, blue);

glRasterPos2f (x, y);

glutBitmapCharacter (font, character);

where font = GLUT_BITMAP_8_BY_13, GLUT_BITMAP_HELVETICA_10, etc.

3.6 Managing the Frame Buffer

glutInit (&argc, argv);

glutInitDisplayMode (GLUT_RGB | mode);

glutInitWindowSize (width, height);

glutInitWindowPosition (x, y);

glutCreateWindow (label);

```
glClear (GL_COLOR_BUFFER_BIT);
glutSwapBuffers ( );
```

where mode = GLUT_SINGLE or GLUT_DOUBLE.

3.7 Registering Callbacks

```
glutDisplayFunc (callback);
glutReshapeFunc (callback);
glutDisplayFunc (callback);
glutMotionFunc (callback);
glutPassiveMotionFunc (callback);
glutMouseFunc (callback);
glutKeyboardFunc (callback);
id = glutCreateMenu (callback);
glutMainLoop ( );
```

3.8 Display Lists

```
glNewList (number, GL_COMPILE);
glEndList ( );
glCallList (number);
glDeleteLists (number, 1);
```

3.9 Managing Menus

```
id = glutCreateMenu (callback);
glutDestroyMenu (id);
glutAddMenuEntry (label, number);
glutAttachMenu (button);
glutDetachMenu (button);
```

where button = GLUT_RIGHT_BUTTON or GLUT_LEFT_BUTTON

CHAPTER 4

WORKING PROCEDURE OF FLOWING FOUNTAIN

4.1 WORKING PROCEDURE

- For displaying menu in first, the display list is defined. For **each different screen different display list** is used. We have about 4 different screens defined in menu function.
- Structure is defined for each of the directions and a class is created for dropping the water via fountain. The Class declaration contains all data members and member functions.
- We have four main functions defined for **velocity of water, time of flowing of water, acceleration for water and setting of new position** of the droplets of water flowing via the fountain.
- Time and Position variable used along with the accelerating factor to determine the vertex position so we have water seems flowing at different levels. A **fountain has several steps, each with its own height**.
- The next work is to create the reservoir which is done by taking outer and inner radius and doing so mathematical work. The important part in this was to draw a perfect circular cylinder shape for reservoir.
- To make the Fountain looks real, with water flow we need to shaply apply a good mathematical algorithm. For accomplish this task we defined speed factor as well as random generation of water. Also different angle of water was needed to make it flow (in your eyes), we defined the angles for them. All this clubbed in a function and was used in for loop so it (flowing fountain's water) continuously flows.
- We have defined a function that allows to have different color for the flowing water in the fountain.

- Glutmenu function is used to make the project interesting. This allows to have user interaction. User can choose from a range of menu that contains help menu, different functionality and color chooser for water.
- Various key have been assigned with different work for user interaction like moving up and down.
- The most important part played in this project is the use of Display list. Also this is project we **utilized the concept of OOP** principle.

CHAPTER 5

DESIGN AND IMPLEMENTATION

5.1 Graphic functions

5.1.1 void glBegin(glEnum mode);

Initiates a new primitive of type mode and starts the collection of vertices. Values of mode include GL_POINTS, GL_LINES and GL_POLYGON.

5.1.2 void glEnd( );

Terminates a list of vertices.

5.1.3 void glColor3f[i f d] (TYPE r, TYPE g, TYPE b);

Sets the present RGB colors. Valid types are int (i), float (f) and double (d). The maximum and minimum values of the floating-point types are 1.0 and 0.0, respectively.

5.1.4 void glClearColor(GLclampf r,GLclampf g,GLclampf b,GLclampf a);

Sets the present RGBA clear color used when clearing the color buffer. Variables of GLclampf are floating-point numbers between 0.0 and 1.0.

5.1.5 int glutCreateWindow(char *title);

Creates a window on the display. The string title can be used to label the window. The return value provides a reference to the window that can be used where there are multiple windows.

5.1.6 void glutInitWindowSize(int width, int height);

Specifies the initial height and width of the window in pixels.

5.1.7 void glutInitWindowPosition(int x, int y);

Specifies the initial position of the top-left corner of the window in pixels.

5.1.8 void glutInitDisplayMode(unsigned int mode);

Request a display with the properties in mode. The value of mode is determined by the logical OR of operation including the color model (GLUT_RGB, GLUT_INDEX) and buffering (GLUT_SINGLE, GLUT_DOUBLE);

5.1.9 void glFlush( );

Forces any buffered any OpenGL commands to execute.

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

Initializes GLUT. The arguments from main are passed in and can be used by the application.

5.1.11 void glutMainLoop( );

Cause the program to enter an event processing loop. It should be the last statement in main.

5.1.12 void glutDisplayFunc(void (*func) (void));

Registers the display function func that is executed when the window needs to be redrawn.

5.1.13 gluOrtho2D(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top);

Defines a two-dimensional viewing rectangle in the plane Z=0;

5.1.14 void glutBitmapCharacter(void *font, int char);

Renders the character with ASCII code char at the current raster position using the raster font given by font. Fonts include GLUT_BITMAP_TIMES_ROMAN_10 and GLUT_BITMAP_TIMES_ROMAN_8_Y_13. The raster position is incremented by the width of the character.

5.1.15 void glClear(GL_COLOR_BUFFER_BIT);

To make the screen solid and white.

5.1.16 void MouseFunc(myMouse);

It is used for the implementation of mouse interface.

Passing the control to

void myMouse(int button,int state,int x,int y);

5.1.17 void KeyboardFunc(key);

It is used for the implementation of keyboard interface.

Passing control to

void key(unsigned char key,int x,int y);

5.1.18 void translate[fd](TYPE x,TYPE y,TYPE z);

Alters the current matrix by displacement of (x,y,z).Type is either GLfloat or GLdouble.

5.1.19 void glPushMatrix(); void glPopMatrix();

Pushes to and pops from the matrix stack corresponding to current matrix mode.

5.1.20 void glLoadMatrix[fd](TYPE *m);

Loads the 16 element array of TYPE GLfloat or GLdouble as a current matrix.

5.2 Hardware and Software requirements:

Hardware Requirements:

- Processor of 2.2G Hz or higher speed
- 20MB Hard Disk Space
- 1GB RAM
- Keyboard
- Mouse

Software Requirement:

- Microsoft Visual Studio 6.0
- Windows-98/xp/vista/win7 Operating System
- MS-Office
- Graphics package available in Microsoft Visual Studio 6.0

5.3 Flowing Fountain System Design

The main functions used in the Flowing Fountain design are:

- DROP CLASS

The class define the following functions –

void SetConstantSpeed (SVertex NewSpeed); - Set the Constant Speed of water.

void SetAccFactor(GLfloat NewAccFactor); - Use for Accelerating water flow

void SetTime(GLfloat NewTime); - Time for setting at particular postion.

void GetNewPosition(SVertex * PositionVertex); - This Functions will increments time, gets the new position

- CREATLIST ();

This Function is use for reservoir and it's boundary area.

- INITFOUNTAIN ();

This function optimized the flowing fountain look as well as speed and make it look real.

- RANDCOLOR ();

Draw different color for different color selected via KEYBOARD.

- DRAWFOUNTAIN ();

DrawFountain is used to intergrated all the function in one and draw the fountain and it's other structures.

- MENU ();

There are two menu functions to allow user to select the different options available to them. MENU() have both MOUSE() and KEYBOARD() functions attached.

- MOUSE ();

The functionality of mouse interaction is defined in the MENU(); Here right mouse click is used to trigger the selection.

- KEYBOARD ();

  Select the key for different functions like flowing the fountain, color changing, up and down movement. The control is passed to the DISPLAY(); function sending the selected button which result in the movement and display the flowing of water through fountain .

- DISPLAY();

  It shows all the internal as well as external structure of Flowing Fountain system and display the all function after selecting the keyboard button it will call the display function.

CHAPTER 6

SNAPSHOTS OF FLOWING FOUNTAIN

6.1 RESULTS

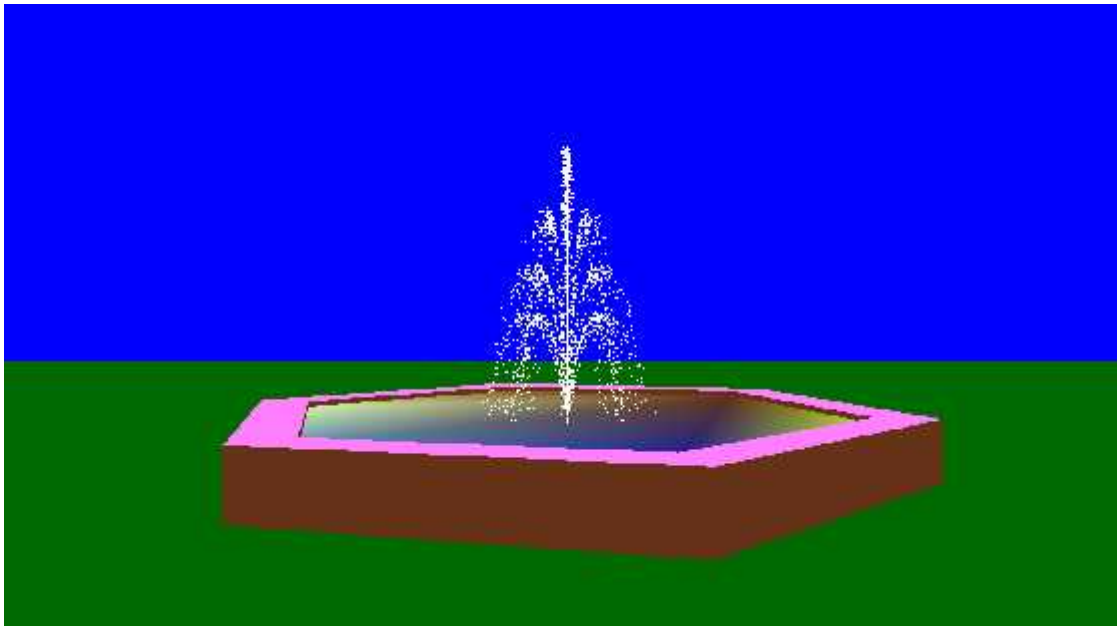
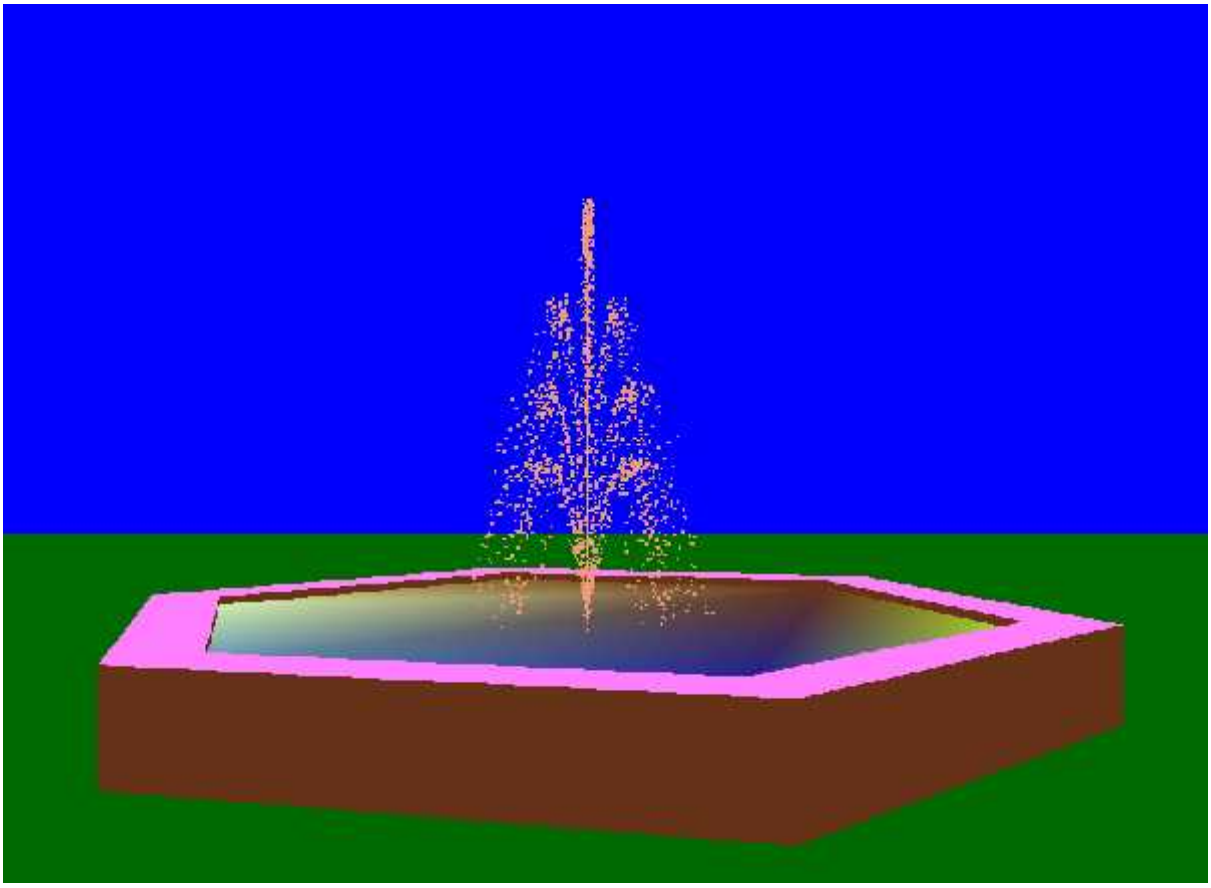


Fig 5.1: Final Result

Fig 5.2: Menu & Main Screen

Fig 5.3:Help Menu



Fg 5.4: Fountain with Different Color

CHAPTER 7

CONCLUSION AND FUTURE SCOPE

7.1 CONCLUSION

- We are introducing the implementation of Flowing Fountain System that shows the architecture of a Fountain and its working.
- The demo is made more interactive with a keyboard and mouse interaction module in the program.

7.2 FUTURE SCOPE

- This project may be useful to follow the archicture of Fountain and its further development in future with more options.
- Even though demo designed is enriched with many options, it is a two dimensional demo, in future it can be re designed with 3D animation and sound effects.

BIBLIOGRAPHY

I. Edward Angel-**Interactive Computer Graphics: A Top-Down Approach using OpenGL** Fifth Edition, Published by Pearson Education, 2009

II. The OpenGL Programming Guide, 5th Edition. The Official guide to learning OpenGL Version 2.1 by OpenGL Architecture Review Board.

III. http://en.wikipedia.org/wiki/Fountain

IV. http://opengl.org/