**CHAPTER 1**

# INTRODUCTION

## COMPUTER GRAPHICS

Graphics provides one of the most natural means of communicating with a computer, since our highly developed 2D and 3D pattern recognition abilities allow us to perceive and process pictorial data rapidly and efficiently. Interactive computer graphics is the most important means of producing pictures since the invention of photography and television. It has the added advantage that, with the computer, we can make pictures not only of concrete real world objects but also of abstract, synthetic objects, such as mathematical surfaces and of data that have no inherent geometry, such as survey results.

## OPENGL

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. The interface consists of over 250 different function calls which can be used to draw complex 3D scenes from simple primitives. OpenGL was developed by Silicon Graphics Inc. (SGI) in 1992 and 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 direct 3D on Microsoft Windows Platforms.OpenGL is managed by the non profit technology consortium, the Khronos group Inc.

OpenGL serves two main purposes :

* To hide the complexities of interfacing with different 3D accelerators, by presenting 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.

OpenGL has historically been influential on the development of 3D accelerator, promoting a base level of functionality that is now common in consumer level hardware:

* Rasterized points, lines and polygons are basic primitives.
* A transform and lighting pipeline .
* Z buffering .
* Texture Mapping.
* Alpha Blending.

**CHAPTER 2**

# PROJECT DSCRIPTION

The aim of this project is to develop a 3D Game which supports basic operations

which include Movement, Artificial Intelligence, collision Detection and also transformation operations like translation, rotation, scaling etc on objects. The package must also have a user friendly interface .

It is developed in ECLIPSE. It has been implemented on UBUNTU platform. The 3-D graphics package designed here provides an interface for the users for handling the display and manipulation of Pac-Man Movements. The Keyboard is the main input device used.

The player controls Pac-Man through a maze, eating dots. When all dots are eaten, Pac-Man game is over. Four ghosts roam the maze, trying to catch Pac-Man. If a ghost touches Pac-Man, a life is lost. When all lives have been lost, the game ends. Near the corners of the maze are four larger, flashing dots known as "Energizers" or "Power Pills", which provide Pac-Man with the temporary ability to eat the ghosts.

The ghosts turn deep blue, reverse direction, and usually move more slowly till it returns to the normal state. When Pac-Man eats ghost during vulnerable state, ghost traverses back to the jail and re-initializes to start a new attack. Collision Detection for Pac-Man and Ghost, and Artificial Intelligence for Ghost has been implemented. A total of 260 points are assigned (1 point for normal pebble and 5 points for super pebble) and max lives of 3 is being setup with a 3-D Maze.

**CHAPTER 3**

# SYSTEM REQUIREMENTS

## 3.1 HARDWARE REQUIREMENTS

* Pentium or higher processor.
* 512 MB or more RAM.

## 3.2 SOFTWARE REQUIREMENTS

This graphics package has been designed for UBUNTU Platform and uses Visual Studio environment.

**CHAPTER 4**

# SYSTEM DESIGN

## 4.1 Proposed System

To achieve three dimensional effects, OpenGL software is proposed. It is software which provides a graphical interface. It is an interface between application program and graphics hardware. The advantages are:

* OpenGL is designed as a streamlined**.**
* It is a hardware independent interface, it can be implemented on many different hardware platforms.
* With OpenGL, we can draw a small set of geometric primitives such as points, lines and polygons etc.
* It provides double buffering which is vital in providing transformations.
* It is event driven software.
* It provides call back function.

**Detailed Design**

**Transformation Functions**

**Translation:**

Translation is done by adding the required amount of translation quantities to each of the points of the objects in the selected area. If P(x,y) be the a point and (tx, ty) translation quantities then the translated point is given by glTranslatef(dx,dy,dz) ;

* + **Rotation:**

The rotation of an object by an angle 'a' is accomplished by rotating each of the points of the object. The rotated points can be obtained using the OpenGL functionsglRotatef(angle, vx,vy,vz);

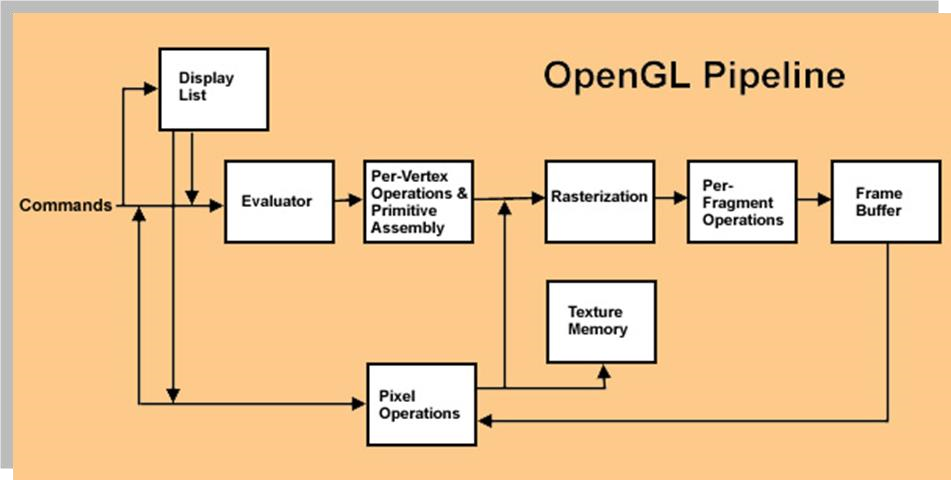
**Scaling:**

The scaling operation on an object can be carried out for an object by multiplying each of the points (x,y,z) by the scaling factors sx, sy and sz. glScalef(sx,sy,sz);

**CHAPTER 5**

# ARCHITECTURE

**OpenGL Graphics Architecture:**



**Fig 5.1 openGl Graphics Architecture**

**Display Lists :**

All data, whether it describes geometry or pixels, can be saved in a display list for current or later use. 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.

* **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.

* **Per Vertex Operations :**

For vertex data, next is the "per-vertex operations" stage, which converts the vertices into primitives. Some vertex data 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.

* **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.

* **Pixel Operation:**

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.

* **Rasterization:**

Rasterization is the conversion of both geometric and pixel data into fragments. Each fragment square corresponds to a pixel in the framebuffer. Color and depth values are assigned for each fragment square.

* **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 6**

# CODE IMPLEMENTATION

#include<ctype.h>

#include<GL/glut.h>

#include<math.h>

#include<stdio.h>

#define M\_PI 3.14159265358979323846264338327950288419716939937510

#define false 0

#define true 1

const int BOARD\_X = 31; const int BOARD\_Y = 28;

int board\_array[BOARD\_X][BOARD\_Y] =

{{8,5,5,5,5,5,5,5,5,5,5,5,5,1,1,5,5,5,5,5,5,5,5,5,5,5,5,7},

{6,0,0,0,0,0,0,0,0,0,0,0,0,2,4,0,0,0,0,0,0,0,0,0,0,0,0,6},

{6,0,8,1,1,7,0,8,1,1,1,7,0,2,4,0,8,1,1,1,7,0,8,1,1,7,0,6},

{6,0,2,11,11,4,0,2,11,11,11,4,0,2,4,0,2,11,11,11,4,0,2,11,11,4,0,6},

{6,0,9,3,3,10, 0,9,3,3,3,10,0,9,10,0,9,3,3,3,10,0,9,3,3,10,0,6},

{6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,6},

{6,0,8,1,1,7,0,8,7,0,8,1,1,1,1,1,1,7,0,8,7,0,8,1,1,7,0,6},

{6,0,9,3,3,10,0,2,4,0,9,3,3,11,11,3,3,10,0,2,4,0,9,3,3,10,0,6},

{6,0,0,0,0,0,0,2,4,0,0,0,0,2,4,0,0,0,0,2,4,0,0,0,0,0,0,6},

{9,5,5,5,5,7,0,2,11,1,1,7,0,2,4,0,8,1,1,11,4,0,8,5,5,5,5,10},

{0,0,0,0,0,6,0,2,11,3,3,10,0,9,10,0,9,3,3,11,4,0,6,0,0,0,0,0},

{0,0,0,0,0,6,0,2,4,0,0,0,0,0,0,0,0,0,0,2,4,0,6,0,0,0,0,0},

{0,0,0,0,0,6,0,2,4,0,8,5,5,1,1,5,5,7,0,2,4,0,6,0,0,0,0,0},

{5,5,5,5,5,10,0,9,10,0,6,0,0,0,0,0,0,6,0,9,10,0,9,5,5,5,5,5},

{0,0,0,0,0,0,0,0,0,0,6,0,0,0,0,0,0,6,0,0,0,0,0,0,0,0,0,0},

{5,5,5,5,5,7,0,8,7,0,6,0,0,0,0,0,0,6,0,8,7,0,8,5,5,5,5,5},

{0,0,0,0,0,6,0,2,4,0,9,5,5,5,5,5,5,10,0,2,4,0,6,0,0,0,0,0},

{0,0,0,0,0,6,0,2,4,0,0,0,0,0,0,0,0,0,0,2,4,0,6,0,0,0,0,0},

{0,0,0,0,0,6,0,2,4,0,8,1,1,1,1,1,1,7,0,2,4,0,6,0,0,0,0,0},

{8,5,5,5,5,10,0,9,10,0,9,3,3,11,11,3,3,10,0,9,10,0,9,5,5,5,5,7},

{6,0,0,0,0,0,0,0,0,0,0,0,0,2,4,0,0,0,0,0,0,0,0,0,0,0,0,6},

{6,0,8,1,1,7,0,8,1,1,1,7,0,2,4,0,8,1,1,1,7,0,8,1,1,7,0,6},

{6,0,9,3,11,4,0,9,3,3,3,10,0,9,10,0,9,3,3,3,10,0,2,11,3,10,0,6},

{6,0,0,0,2,4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,4,0,0,0,6},

{2,1,7,0,2,4,0,8,7,0,8,1,1,1,1,1,1,7,0,8,7,0,2,4,0,8,1,4},

{2,3,10,0,9,10,0,2,4,0,9,3,3,11,11,3,3,10,0,2,4,0,9,10,0,9,3,4},

{6,0,0,0,0,0,0,2,4,0,0,0,0,2,4,0,0,0,0,2,4,0,0,0,0,0,0,6},

{6,0,8,1,1,1,1,11,11,1,1,7,0,2,4,0,8,1,1,11,11,1,1,1,1,7,0,6},

{6,0,9,3,3,3,3,3,3,3,3,10,0,9,10,0,9,3,3,3,3,3,3,3,3,10,0,6},

{6,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,6},

{9,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5,10}};

int pebble\_array[BOARD\_X][BOARD\_Y] =

{{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},

{0,1,1,1,1,1,1,1,1,1,1,1,1,0,0,1,1,1,1,1,1,1,1,1,1,1,1,0},

{0,1,0,0,0,0,1,0,0,0,0,0,1,0,0,1,0,0,0,0,0,1,0,0,0,0,1,0},

{0,3,0,0,0,0,1,0,0,0,0,0,1,0,0,1,0,0,0,0,0,1,0,0,0,0,3,0},

{0,1,0,0,0,0,1,0,0,0,0,0,1,0,0,1,0,0,0,0,0,1,0,0,0,0,1,0},

{0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0},

{0,1,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,1,0}, {0,1,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,1,0},

{0,1,1,1,1,1,1,0,0,1,1,1,1,0,0,1,1,1,1,0,0,1,1,1,1,1,1,0},

{0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0},

{0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0},

{0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0},

{0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0},

{0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0},

{0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0},

{0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0},

{0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0},

{0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0},

{0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0},

{0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0},

{0,1,1,1,1,1,1,1,1,1,1,1,1,0,0,1,1,1,1,1,1,1,1,1,1,1,1,0},

{0,1,0,0,0,0,1,0,0,0,0,0,1,0,0,1,0,0,0,0,0,1,0,0,0,0,1,0},

{0,1,0,0,0,0,1,0,0,0,0,0,1,0,0,1,0,0,0,0,0,1,0,0,0,0,1,0},

{0,3,1,1,0,0,1,1,1,1,1,1,1,0,0,1,1,1,1,1,1,1,0,0,1,1,3,0},

{0,0,0,1,0,0,1,0,0,1,0,0,0,0,0,0,0,0,1,0,0,1,0,0,1,0,0,0},

{0,0,0,1,0,0,1,0,0,1,0,0,0,0,0,0,0,0,1,0,0,1,0,0,1,0,0,0},

{0,1,1,1,1,1,1,0,0,1,1,1,1,0,0,1,1,1,1,0,0,1,1,1,1,1,1,0},

{0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0},

{0,1,0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,1,0},

{0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0},

{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}};

GLubytelist[5];

int tp\_array[31][28];

int pebbles\_left;

double speed1 = 0.1;

double angle1 = 90;

double a=13.5, b=23;

bool animate = false;

int lives=3;

int points=0;

void keys(); unsigned char ckey='w';

void mykey(unsigned char key,intx,int y); bool Open(int a,int b);

void Move()

{

1. += speed1\*cos(M\_PI/180\*angle1);
2. += speed1\*sin(M\_PI/180\*angle1);

if(animate&&ckey==GLUT\_KEY\_UP&& (int) a - a > -0.1 && angle1 != 270) //w

{

if (Open(a,b-1))

{

animate = true;

angle1 = 270;

}

}

else if(animate&&ckey==GLUT\_KEY\_DOWN&& (int) a - a > -0.1 && angle1 != 90)// s

{

if (Open(a,b+1))

{

animate = true;

angle1= 90;

}

}

else if(animate&&ckey==GLUT\_KEY\_LEFT&& (int) b - b > -0.1 && angle1 != 180)//a

{

if (Open(a-1,b))

{

animate = true;

angle1 = 180;

}

}

else if(animate&&ckey==GLUT\_KEY\_RIGHT&& (int) b - b > -0.1 && angle1 != 0)//d

{

if (Open(a+1,b))

{

animate = true;

angle1 = 0;

}

}

}

void Pac(void)

{

//Draw Pacman glColor3f(0,1,1);

glPushMatrix();

glTranslatef(a,-b,0);

glTranslatef(0.5,0.6,0);

glTranslatef((float)BOARD\_X/-2.0f,(float)BOARD\_Y/2.0f,0.5); glutSolidSphere(0.5,15,10);

glPopMatrix();

}

//Monster Drawing And Moving Begins

bool open\_move[4];

bool gameover = false;

int num\_ghosts = 4;

int start\_timer=3;

class Ghost { private:

public:

bool edible;

int edible\_max\_time; int edible\_timer;

bool eaten;

bool transporting;

float color[3];

double speed; double max\_speed; bool in\_jail; int jail\_timer; double angle;

double x, y;

Ghost(double, double);

~Ghost(void);

void Move(); //Move the Monster

void Update(void); //Update Monster State

void Chase(double, double, bool\*); //Chase Pacman

bool Catch(double, double); //collision detection

void Reinit(void);

void Vulnerable(void);

void Draw(void); //Draw the Monster void game\_over(void);

};

Ghost \*ghost[4];

Ghost::~Ghost(void){}

Ghost::Ghost(double tx, double ty)

{

tx = x; ty = y; angle = 90; speed = max\_speed=1; color[0] = 1; color[1] = 0; color[2] = 0; eaten = false; edible\_max\_time =300; edible = false; in\_jail = true;

jail\_timer = 30;

}

void Ghost::Reinit(void)

{

edible = false;

in\_jail = true; angle = 90;

}

//Move Monster void Ghost::Move()

{

1. += speed\*cos(M\_PI/180\*angle);
2. += speed\*sin(M\_PI/180\*angle);

}

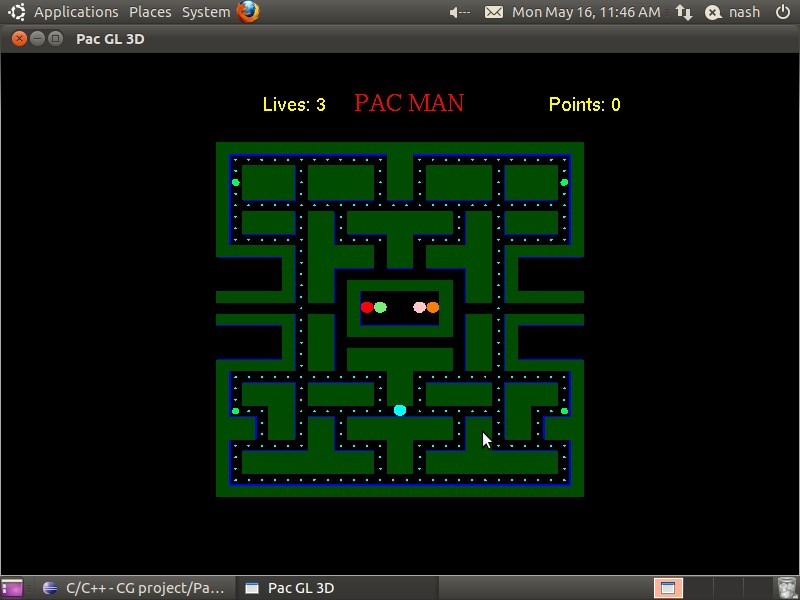
void Ghost::game\_over()

{

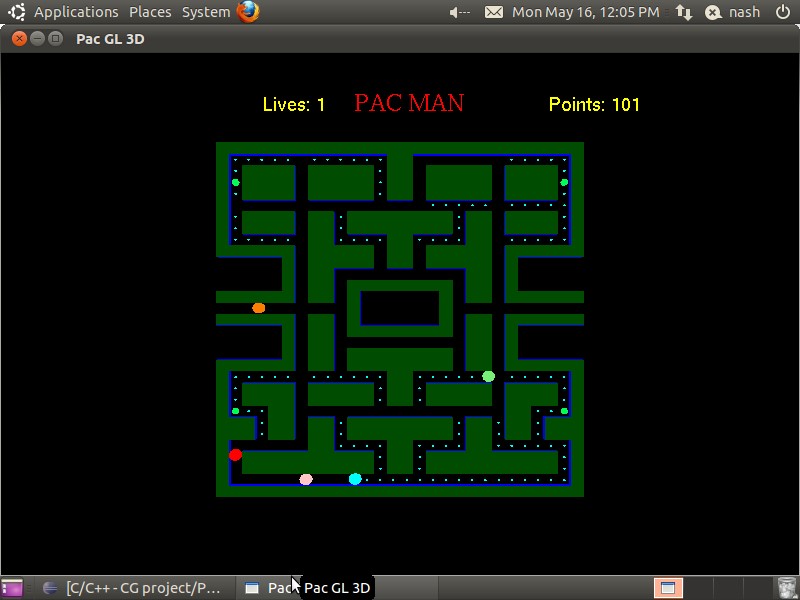
}

**CHAPTER 7**

# SNAPSHOTS



**Fig 7.1 Initial View Of Pac-Man**



**Fig 7.2 Ghosts Chasing The Pac-Man**



**Fig 7.3 Ghosts In The Vulnerable State**



**Fig 7.4 Game Over**

**CHAPTER 8**

# FUTURE ENHANCEMENTS

The mini project designed can be implemented in the future with more effects in lighting as well as function. The scene can be viewed in 3D display further we can implement more user interaction as well as more function.

In future the following enhancements could be done:

* Providing Camera Movement.
* Providing More Number of Levels.
* Providing High Quality Graphics.
* Implementing Shortest Path Algorithm for Ghosts.

# CONCLUSION

We have tried our level best to build the project efficiently and correctly and have succeeded in building a better project, but may not be a best project. We have implemented the required functions which we had stated earlier. After all testing process, the game is now ready to be played.

The mini project helps me in learning how to use simple codes to build simple graphics content. These simple graphics content can help render a complex set of graphical objects easily. It also helps me understand the mechanics of OpenGL programming, handling various functionalities to give the user a different experience.

# REFERENCE

1. The Red Book-OpenGL programming Guide,6th edition
2. Edward Angel Interactive Computer Graphics A Top-Down Approach with

OpenGL, 5th edition, Addison and Wesley

1. http://www.opengl.org/registry/
2. [www.codeguru.com](http://www.codeguru.com/)
3. [www.openglforum.com](http://www.openglforum.com/)
4. [www.nehe.com.](http://www.nehe.com/)