**ABSTRACT**

Computer Graphics is the process of making the design, can many things, including modelling simulation and visualization of an object or a problem. Open GL support this modelling capability as Open GL has additional features to better produce something more Realistic.

Open GL allows us to create a graph that can be run on any operating system only minor adjustment.

The Pac-Man game has its roots as a Japanese arcade game developed by Namco (now Namco Bandai) and licensed for distribution in the U.S. by Midway, first releases in Japan on May 22, 1980. Pac-Man is universally considered as one of the classics of the medium, virtually synonymous with video games, and an icon of 1980's popular culture. When it was released, the game became a social phenomenon.

The Pac-Man game is often credited with being a landmark in video game history, and is among the most famous arcade games of all time. The character also appears in more than 30 officially licensed games as equal, as well as in numerous unauthorized clones and bootlegs.

TABLE OF CONTENTS

Topic Page No.

Chapter 1: **INTRODUCTION**

1.1 Computer Graphics 1.2 Open GL 1.3 Project Goal 1.4 Scope

Chapter 2: **LITERATURE SURVEY**

2.1 Pac-Man

Chapter 3: **REQUIREMENT SPECIFICATION**

3.1 Hardware Requirements 3.2 Software Requirements

Chapter 4: **DESIGN**

4.1 User Defined Functions 4.2 Algorithm 4.3 Data Flow Diagram

Chapter 5: **IMPLEMENTATION**

Chapter 6: **CODE**

Chapter 7:  **SNAPSHOTS**

Chapter 8: **CONCLUSION BIBLIOGRAPHY**

CHAPTER 1

**INTRODUCTION**

**1.1**  **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 advantages that, with the computer, we can make picture not only of concreate 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.

**1.2**  **OpenGL**

OpenGL (Open Graphics Library) is a standard specification defining a cross language cross platform API for writing application 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 nonprofit technology consortium, the Krono's 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 commen in consumer level hardware:

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

**OpenGL Graphics Architecture:**

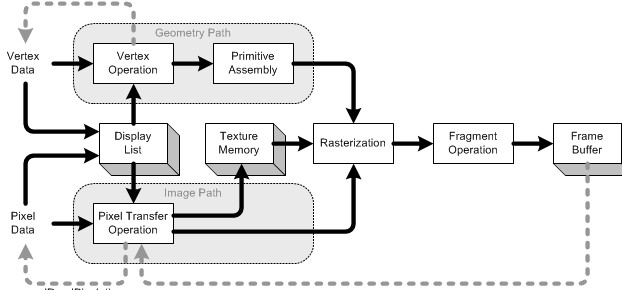


Fig 1.1 OpenGL Graphics Architecture

**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" stages, 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 portion 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.

* **Project Goal:**

The aim of this project is to develop a 3D Game which supports basic operation which include Movement, Artificial Intelligence, collision Detection and also transformation operation like translation, rotation, scaling etc. no objects. The package must also have a user-friendly interface.

* **Scope:**

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

Chapter 2

**LITERATURE SURVEY**

**Pac-Man:**

The game was developed primarily by a young Namco employee Toru Iwatani, over a year, beginning in a April of 1979, employing a nine-man team. The original title was pronounced Paku-man and was inspired by the Japanese onomatopoeic phrase Paku Tabera where Paku describes (the sound of) the mouth movement when widely opened and then closed inn succession.

The Pac-Man game has its roots as a Japanese arcade game developed by Namco (now Namco Bandai) and licensed for distribution in the U.S. by Midway, first released in Japan on May 22, 1980. Pac-Man is universally considered as one of the classes of the medium, virtually synonymous with video games, and an icon of 1980s popular culture. When it was released, the game became a social phenomenon. The Pac-Man game is often credited with being a landmark in video game history, and is among the most famous arcade games of all time. The character also appears in more than 30 officially licensed games as equals, as well as in numerous unauthorized clones and bootlegs.

**Basic Working of Pac-Man:**

The objective of the game is to eat all of the dots placed in the maze while avoiding four colored ghosts. When Pac-Man eats all of the dots, the player advances to the next level. Levels are indicated by fruit icons at the bottom of the screen. If Pac-Man is caught by a ghost, he will lose a life; the game ends when all lives are lost. Placed near the four corners of the maze are large flashing "energizers" or "power pellets." Eating these will cause the ghosts to turn blue with a dizzied expression and to reverse direction.

Pac-Man can eat blue ghosts for bonus points; when a ghost is eaten, their eyes make their way back to the center box in the maze, where the ghost "regenerates" and resumes their normal activity. Eating multiple blue ghosts in succession increases their point value. After a certain amount of time, blue-colored ghosts will flash white before turning back into their normal form. The 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 3-D Maze.

**Related Data Structure:**

Package uses data structures like Stack for storing the co-ordinats of primitives and also uses Arrays for storing the pixel values.

CHAPTER 3

**HARDWARE AND SOFTWARE REQUIREMENTS**

**3.1**  **Hardware Requirements:**

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

**3.2**  **Softer Requirements**

* This graphics package has been designed for UBUNTU platform and uses ECLIPSE integrated environment.

**Development platform**

UBUNTU 10.10

**Development tool**

ECLIPSE

**Language Used in Coding**

C++

CHAPTER 4

**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 an graphics hardware.

The advantages are:

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

**Detailed Design:**

Transformation Function

* **Translation:**

Translation is done by adding the required amount of translation quantities to each of the points of the object in the selected aarea. If P(x,y) be the a point and (tx, ty) translation quantities then the translated points 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 function glRotatef(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

**IMPLEMENTATION**

**5.1 User Defined Functions:**

**Function for Pac-Man:**

Pacman\_Move()

{

Both the x and y updated coordinates are calculate using the speed of the pacman and trigonometric function.

}

Pacman\_Draw()

{

Here, the pacman is drawn using the glutSolidSphere() function And the eyes of the pacman is rendered using a combination of Random colouring and glusolidSphere() function.

}

**Collision Detection for Pac-Man and Ghost:**

Bool open()

{

Here, the condition to check if the board is open is given This is used for collision detection.

}

**Function for Monsters:**

Monster\_init()

{

Here, all the variable values are initialized at the beginning of the Game

}

Monster\_Move()

{

Both the x and y updated coordinates are calculating using the speed of the Monster and trigonometric functions.

}

Monster\_Updation()

{

Here, the state of the monster is updated. The edibility condition is checked and the flag is set.

If the monster is eaten, then the jail timer starts & The monster is sent to jail.

}

Monster\_Vulnerable()

{

Checks the edible condition for the monsters.

}

Monster\_Chase()

{

Here, depending on the edible condition for the Monsters, they are set to chase Pac-Man or escape from it. This is done by using the x-y coordinates of the Pac-Man. The random movement of the monsters is also handled

}

Monster\_draw()

{

Here, the pacman is drawn using the glutsolidSphere() function.

The color of the monster is changed depending on the table condition.

}

**Function for Board:**

Board\_draw()

{

Here, the boards is rendered.

It's done in 2 steps to avoid complication in depth. Depending on the x-y coordinates, the borad is rendered Using different walls.

The pebbles are also rendered here.

Using the random f(), the color of the pebbles is changed to give it a flicker effect.

}

Render\_scene()

{

This is the default display function.

Here, the collision detection for Pac-Man, the condition for Normal & super pills consumption, with monster movements are covered.

Options are provided for game control.

}

Create\_list()

{

This function is used to create the basic primitive walls using Display lists. Based on the position, the appropriates list is called.

}

**In Built Functions Used:**

* **PushMatrix And PopMatrix**

**Syntax:** glPushMatrix();

glPopMatrix();

**Description:**

Pushes the current transformation matrix onto the matrix stack. The glPushMatrix() function saves the current coordinates system to the stack and glPopMatrix() restores the prior coordinate system.

* **Solid Sphere**

**Syntax:**

void glutSolidSphere(GLdouble radius, GLint slices, GLint stacks);

**Parameters:**

Radius: The radius of the sphere.

Slices: The number of subdivisions around the Z axis (similar to Lines of longitude).

Stacks: The number of subdivisions along the Z axis (similar to Lines of lititude).

**Description:**

Renders a sphere centered at the modeling coordinates origin of the specified radius. The sphere is subdivided around the Z axis into slices and along the Z axis into stacks.

**get Async KeyState Funtion:**

**Syntax:**

SHORT GetAsyncKeyState( int vKey);

**Parameters:**  vKey

**[in]int;**

Specifies one of 256 possible key codes. You can use left-and right-distinguishing constants to specify certain keys.

**Description:**

The GetAsyncKeyState function determines whether a key is up or down at the time the function is called, and whether the key was pressed after a previous call to GetAsyncKeyState.

**Return Value:**

**SHORT**

* **Post Redisplay:**

**Syntax:**

void glutPostRedisplay:

**Description:**

glutPostRedisplay marks the normal plane of current window as needing to be redisplayed. glutPostRedisplay may be called within a window's display or overlay display callback to re-mark that window for redisplay.

* **Timer Funtion:**

**Syntax:**

void glutTimerFunc(unsigned int msecs, void(\*func), int value);

**Parameters:**

msecs: Number of milliseconds to pass before calling the callback.

func: The timer callback function.

Value: Interger value to pass to the timer callback.

**Description:**

glutTimerFunc registers the timer callback func to be triggered in at least msecs milliseconds. The value parameter to the timer callback will be the value of the parameter to glutTimerFunc.

* **Bitmap Character:**

**Syntax:**

void glutBitmapCharacter(void \*font, int character);

**Parameters:**

Font: Bitmap font to use.

Character: Character to render(not to 8 bits).

**Description:**

Without using any display lists, glutBitmapCharacter renders the character in the named bitmapfont. The available fonts are:

GLUT\_BITMAP\_TIMES\_ROMAN\_24: A 24-point proportional spaced times Roman font.

GLUT\_BITMAP\_HELVETICA\_18: A 18-point proportional spaced Helvetica font.

* **Raster Position:**

**Syntax:**

void glRasterPos3f( GLfloat x, GLfloat y, GLfloatz);

**Parameters:**

x: Specifies the x-coordinates for the current raster position.

y: Specifies the y-coordinates for the current position.

z: Specifies the z-coordinates for the current raster position.

**Description:**

OpenGL maintains a 3-D positon in window coordinates. This position, called the raster position, is maintained with subpixel accuracy. It is used to position pixel and bitmap write operations.

* **Color Funtion:**

**Syntax:**

void glColor3ub(GLubyte red, GLubyte green, GLubyte blue);

**Parameters:**

red : The new red value for the current color.

green: The new green value for the current color.

blue: The new blue value for the current color.

**Description:**

This function randomly generates different color based on the rand() function.

* **Keyboard Function:**

**Syntax:**

void glutKeyboardFunc(void (\*func)(unsigned char key, int x, int y));

FUNC: The new keyboard callback function.

**Description:**

glutKeyboardFunc sets the keyboard callback for the current window. When a user types into the window, a keyboard callback.

* **ShadeModel:**

**Syntax:**

void glShadeModel(GLenum model);

**Parameters:**

Mode: Specifies a symbolic value representing a shading technique. Accepted GL\_FLAT and GL\_SMOOTH. The initial value is GL\_SMOOTH.

**Description:**

GL primitives can have either flat or smooth shading. Smooth shading, the default, causes the computed colors of vertices to be interpolated as the primitive is rasterized typically assigning different colors to each resulting pixel fargment.. Flat shading selects the computed color of the just one vertex and assigns it to all the pixel fragments generated by rasterizing a single primitive.

**5.2** **ALGORITHM**

Step 1: Initialize the graphics window and its size using GLUT functions.

Step 2: Register the keyboard call backs in main function.

Step 3: When arrow keys are pressed ghosts are released from jail.

Step 4: If left arrow is pressed the Pac-Man move towards left in the maze eating the pebbles simultaneously points are incremented, when points becomes 260 the game is restored.

Step 5: If right, up, down arrows is pressed Pac-Man moves in respective direction eating pebbles.

Step 6: If Pac-Man eats super pebbles ghosts become edible and vulnerable function

Step 7: If Pac-Man collides the ghosts in vulnerable state ghosts go to jail. If the ghosts are uneaten in vulnerable state update function is called.

Step 8: If Pac-Man collides with ghosts provided ghosts are not in vulnerable state Pacman becomes edible and lives are decremented by one. If lives becomes zero the game is over.

**5.3**  **DATA FLOW DIAGRAM:**

**CHAPTER 7**

**CODE**

#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} };

GLubyte list[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, int x, int y);

bool Open(int a, int b);

void Move()

{

a += speed1 \* cos(M\_PI / 180 \* angle1);

b += 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()

{

x += speed \* cos(M\_PI / 180 \* angle);

y += speed \* sin(M\_PI / 180 \* angle);

}

void Ghost::game\_over()

{

}

void Ghost::Update(void)

{

if ((int)x == 0 && (int)y == 14 && (!(transporting)))

{

angle = 180;

}

if (x < 0.1 && (int)y == 14)

{

x = 26.9;

transporting = true;

}

if ((int)x == 27 && (int)y == 14 && (!(transporting)))

{

angle = 0;

}

if (x > 26.9 && (int)y == 14)

{

x = 0.1;

transporting = true;

}

if ((int)x == 2 || (int)x == 25)

transporting = false;

if (((int)x < 5 || (int)x > 21) && (int)y == 14 && !edible && !eaten)

speed = max\_speed / 2;

speed = max\_speed;

//edibility

if (edible\_timer == 0 && edible && !eaten)

{

edible = false;

speed = max\_speed;

}

if (edible)

edible\_timer--;

//JAIL

if (in\_jail && (int)(y + 0.9) == 11)

{

in\_jail = false;

angle = 180;

}

if (in\_jail && ((int)x == 13 || (int)x == 14))

{

angle = 270;

}

//if time in jail is up, position for exit

if (jail\_timer == 0 && in\_jail)

{

//move right to exit

if (x < 13)

angle = 0;

if (x > 14)

angle = 180;

}

//decrement time in jail counter

if (jail\_timer > 0)

jail\_timer--;

//EATEN GHOST SEND TO JAIL

if (eaten && ((int)x == 13 || (int)(x + 0.9) == 14) && ((int)y > 10 && (int)y < 15))

{

in\_jail = true;

angle = 90;

if ((int)y == 14)

{

eaten = false;

speed = max\_speed;

jail\_timer = 66;

x = 11;

}

}

}

bool Ghost::Catch(double px, double py)

{

// Collision Detection

if (px - x < 0.2 && px - x > -0.2 && py - y < 0.2 && py - y > -0.2)

{

return true;

}

return false;

}

//called when pacman eats a super pebble

void Ghost::Vulnerable(void)

{

if (!(edible))

{

angle = ((int)angle + 180) % 360;

speed = max\_speed;

}

edible = true;

edible\_timer = edible\_max\_time;

//speed1=0.15;

}

void Ghost::Chase(double px, double py, bool\* open\_move)

{

int c;

if (edible)

c = -1;

else

c = 1;

bool moved = false;

if ((int)angle == 0 || (int)angle == 180)

{

if ((int)c \* py > (int)c \* y && open\_move[1])

angle = 90;

else if ((int)c \* py < (int)c \* y && open\_move[3])

angle = 270;

}

else if ((int)angle == 90 || (int)angle == 270)

{

if ((int)c \* px > (int)c \* x && open\_move[0])

angle = 0;

else if ((int)c \* px < (int)c \* x && open\_move[2])

angle = 180;

}

//Random Moves Of Monsters

if ((int)angle == 0 && !open\_move[0])

angle = 90;

if ((int)angle == 90 && !open\_move[1])

angle = 180;

if ((int)angle == 180 && !open\_move[2])

angle = 270;

if ((int)angle == 270 && !open\_move[3])

angle = 0;

if ((int)angle == 0 && !open\_move[0])

angle = 90;

}

void Ghost::Draw(void)

{

if (!edible)

glColor3f(color[0], color[1], color[2]);

else

{

if (edible\_timer < 150)

glColor3f((edible\_timer / 10) % 2, (edible\_timer / 10) % 2, 1);

if (edible\_timer >= 150)

glColor3f(0, 0, 1);

}

if (eaten)

glColor3f(1, 1, 0); //When Eaten By PacMan Change Color To Yellow

glPushMatrix();

glTranslatef(x, -y, 0);

glTranslatef(0.5, 0.6, 0);

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

glutSolidSphere(.5, 10, 10);

glPopMatrix();

}

void tp\_restore(void)

{

for (int ISO = 0; ISO < BOARD\_X; ISO++)

{

for (int j = 0; j < BOARD\_Y; j++)

{

tp\_array[ISO][j] = pebble\_array[ISO][j];

}

}

pebbles\_left = 244;

}

void Draw(void)

{

glColor3f(1, 0, 1);

//split board drawing in half to avoid issues with depth

for (int ISO = 0; ISO < BOARD\_X; ISO++)

{

for (int j = 0; j < BOARD\_Y / 2; j++)

{

glColor3f(0, 0, 1);

int call\_this = 0;

glPushMatrix();

glTranslatef(-(float)BOARD\_X / 2.0f, -(float)BOARD\_Y / 2.0f, 0);

glTranslatef(j, BOARD\_Y - ISO, 0);

glPushMatrix();

glTranslatef(0.5, 0.5, 0);

switch (board\_array[ISO][j])

{

case 4:

glRotatef(90.0, 0, 0, 1);

case 3:

glRotatef(90.0, 0, 0, 1);

case 2:

glRotatef(90.0, 0, 0, 1);

case 1:

call\_this = 1;

break;

case 6:

glRotatef(90.0, 0, 0, 1);

case 5:

call\_this = 2;

break;

case 10:

glRotatef(90.0, 0, 0, 1);

case 9:

glRotatef(90.0, 0, 0, 1);

case 8:

glRotatef(90.0, 0, 0, 1);

case 7:

call\_this = 3;

break;

}

glScalef(1, 1, 0.5);

glTranslatef(-0.5, -0.5, 0);

glCallList(list[call\_this]);

glPopMatrix();

//now put on the top of the cell

if (call\_this != 0 || board\_array[ISO][j] == 11)

{

glTranslatef(0, 0, -0.5);

glCallList(list[4]);

}

glPopMatrix();

if (tp\_array[ISO][j] > 0)

{

glColor3f(0, 300, 1 / (float)tp\_array[ISO][j]);

glPushMatrix();

glTranslatef(-(float)BOARD\_X / 2.0f, -(float)BOARD\_Y / 2.0f, 0);

glTranslatef(j, BOARD\_Y - ISO, 0);

glTranslatef(0.5, 0.5, 0.5);

glutSolidSphere(0.1f \* ((float)tp\_array[ISO][j]), 6, 6);

glPopMatrix();

}

}

}

int ISO;

for (ISO = 0; ISO < BOARD\_X; ISO++)

{

for (int j = BOARD\_Y - 1; j >= BOARD\_Y / 2; j--)

{

glColor3f(0, 0, 1);

int call\_this = 0;

glPushMatrix();

glTranslatef(-(float)BOARD\_X / 2.0f, -(float)BOARD\_Y / 2.0f, 0);

glTranslatef(j, BOARD\_Y - ISO, 0);

glPushMatrix();

glTranslatef(0.5, 0.5, 0);

switch (board\_array[ISO][j])

{

case 4:

glRotatef(90.0, 0, 0, 1);

case 3:

glRotatef(90.0, 0, 0, 1);

case 2:

glRotatef(90.0, 0, 0, 1);

case 1:

call\_this = 1;

break;

case 6:

glRotatef(90.0, 0, 0, 1);

case 5:

call\_this = 2;

break;

case 10:

glRotatef(90.0, 0, 0, 1);

case 9:

glRotatef(90.0, 0, 0, 1);

case 8:

glRotatef(90.0, 0, 0, 1);

case 7:

call\_this = 3;

break;

}

glScalef(1, 1, 0.5);

glTranslatef(-0.5, -0.5, 0);

glCallList(list[call\_this]);

glPopMatrix();

//now put on top

if (call\_this != 0 || board\_array[ISO][j] == 11)

{

glTranslatef(0, 0, -0.5);

glCallList(list[4]);

}

glPopMatrix();

if (tp\_array[ISO][j] > 0)

{

glColor3f(0, 300, 1 / (float)tp\_array[ISO][j]);

glPushMatrix();

glTranslatef(-(float)BOARD\_X / 2.0f, -(float)BOARD\_Y / 2.0f, 0);

glTranslatef(j, BOARD\_Y - ISO, 0);

glTranslatef(0.5, 0.5, 0.5);

glutSolidSphere(0.1f \* ((float)tp\_array[ISO][j]), 6, 6);

glPopMatrix();

}

}

}

Pac();

}

bool Open(int a, int b)

{

if (board\_array[b][a] > 0)

{

return false;

}

return true;

}

void RenderScene();

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

{

if (start\_timer > 0)

{

start\_timer--;

}

}

void specialDown(int key, int x, int y)

{

if (start\_timer > 0)

start\_timer--;

ckey = key;

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

{

if (Open(a, b - 1))

{

animate = true;

angle1 = 270;

}

}

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

{

if (Open(a, b + 1))

{

animate = true;

angle1 = 90;

}

}

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

{

if (Open(a - 1, b))

{

animate = true;

angle1 = 180;

}

}

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

{

if (Open(a + 1, b))

{

animate = true;

angle1 = 0;

}

}

}

void specialUp(int key, int x, int y)

{

}

void P\_Reinit()

{

a = 13.5;

b = 23;

angle1 = 90;

animate = false;

Pac();

}

void G\_Reinit(void)

{

start\_timer = 3;

//ghost initial starting positions

int start\_x[4] = { 11,12,15,16 };

float ghost\_colors[4][3] = { {255,0,0},{120,240,120},{255,200,200},{255,125,0} };

for (int i = 0; i < num\_ghosts; i++)

{

ghost[i]->Reinit();

ghost[i]->x = start\_x[i];

ghost[i]->y = 14;

ghost[i]->eaten = false;

ghost[i]->jail\_timer = i \* 33 + 66;

ghost[i]->max\_speed = 0.1 - 0.01 \* (float)i;

ghost[i]->speed = ghost[i]->max\_speed;

//colorize ghosts

for (int j = 0; j < 3; j++)

ghost[i]->color[j] = ghost\_colors[i][j] / 255.0f;

}

}

void renderBitmapString(float x, float y, void\* font, char\* string)

{

char\* c;

glRasterPos2f(x, y);

for (c = string; \*c != '\0'; c++)

{

glutBitmapCharacter(font, \*c);

}

}

void Write(char\* string)

{

while (\*string)

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_18, \*string++);

}

void print(char\* string)

{

while (\*string)

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_24, \*string++);

}

//Display Function->This Function Is Registered in glutDisplayFunc

void RenderScene()

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

//Through Movement->From One End To The Other

if ((int)a == 27 && (int)b == 14 && angle1 == 0)

{

a = 0;

animate = true;

}

else

if ((int)(a + 0.9) == 0 && (int)b == 14 && angle1 == 180)

{

a = 27;

animate = true;

}

//Collision Detection For PacMan

if (animate)

Move();

if (!(Open((int)(a + cos(M\_PI / 180 \* angle1)),

(int)(b + sin(M\_PI / 180 \* angle1)))) &&

a - (int)a < 0.1 && b - (int)b < 0.1)

animate = false;

if (tp\_array[(int)(b + 0.5)][(int)(a + 0.5)] == 1)

{

tp\_array[(int)(b + 0.5)][(int)(a + 0.5)] = 0;

pebbles\_left--;

points += 1;

}

//Super Pebble Eating

else if (tp\_array[(int)(b + 0.5)][(int)(a + 0.5)] == 3)

{

tp\_array[(int)(b + 0.5)][(int)(a + 0.5)] = 0;

pebbles\_left--;

points += 5;

for (int i = 0; i < 4; i++)

{

if (!ghost[i]->eaten)

ghost[i]->Vulnerable(); //Calls A Function To Make Monster Weak

}

}

//All The Pebbles Have Been Eaten

if (pebbles\_left == 0)

{

G\_Reinit();

P\_Reinit();

tp\_restore();

points = 0;

lives = 3;

}

if (!gameover)

Draw();

for (int d = 0; d < num\_ghosts; d++)

{

if (!gameover && start\_timer == 0)

ghost[d]->Update();

if (!ghost[d]->in\_jail &&

ghost[d]->x - (int)ghost[d]->x < 0.1 && ghost[d]->y - (int)ghost[d]->y < 0.1)

{

bool open\_move[4];

//Finding Moves

for (int ang = 0; ang < 4; ang++)

{

open\_move[ang] = Open((int)(ghost[d]->x + cos(M\_PI / 180 \* ang \* 90)),

(int)(ghost[d]->y + sin(M\_PI / 180 \* ang \* 90)));

}

//Chase Pac Man

if (!ghost[d]->eaten)

{

if (ghost[d]->x - (int)ghost[d]->x < 0.1 && ghost[d]->y - (int)ghost[d]->y < 0.1)

ghost[d]->Chase(a, b, open\_move);

}

else

{

if (ghost[d]->x - (int)ghost[d]->x < 0.1 && ghost[d]->y - (int)ghost[d]->y < 0.1)

ghost[d]->Chase(13, 11, open\_move);

}

}

if (ghost[d]->in\_jail && !(Open((int)(ghost[d]->x + cos(M\_PI / 180 \* ghost[d]->angle)),

(int)(ghost[d]->y + sin(M\_PI / 180 \* ghost[d]->angle)))) && ghost[d]->jail\_timer > 0

&& ghost[d]->x - (int)ghost[d]->x < 0.1 && ghost[d]->y - (int)ghost[d]->y < 0.1)

{

ghost[d]->angle = (double)(((int)ghost[d]->angle + 180) % 360);

}

if (!gameover && start\_timer == 0)

ghost[d]->Move();

ghost[d]->Draw();

if (!(ghost[d]->eaten))

{

bool collide = ghost[d]->Catch(a, b);

//Monster Eats PacMan

if (collide && !(ghost[d]->edible))

{

lives--;

if (lives == 0)

{

gameover = true;

lives = 0;

ghost[d]->game\_over();

}

P\_Reinit();

d = 4;

}

//PacMan Eats Monster And Sends It To Jail

else if (collide && ((ghost[d]->edible)))

{

ghost[d]->edible = false;

ghost[d]->eaten = true;

ghost[d]->speed = 1;

}

}

}

if (gameover == true)

{

glColor3f(1, 0, 0);

renderBitmapString(-5, 0.5, GLUT\_BITMAP\_HELVETICA\_18, (char\*)"gameover");

}

char tmp\_str[40];

glColor3f(1, 1, 0);

glRasterPos2f(10, 18);

sprintf\_s(tmp\_str, "Points: %d", points);

Write(tmp\_str);

glColor3f(1, 0, 0);

glRasterPos2f(-5, 18);

sprintf\_s(tmp\_str, "PAC MAN");

print(tmp\_str);

glColor3f(1, 1, 0);

glRasterPos2f(-12, 18);

sprintf\_s(tmp\_str, "Lives: %d", lives);

Write(tmp\_str);

glutPostRedisplay();

glutSwapBuffers();

}

void create\_list\_lib()

{

//Set Up Maze Using Lists

list[1] = glGenLists(1);

glNewList(list[1], GL\_COMPILE);

//North Wall

glBegin(GL\_QUADS);

glColor3f(0, 0, 1);

glNormal3f(0.0, 1.0, 0.0);

glVertex3f(1.0, 1.0, 1.0);

glVertex3f(1.0, 1.0, 0.0);

glVertex3f(0.0, 1.0, 0.0);

glVertex3f(0.0, 1.0, 1.0);

glEnd();

glEndList();

list[2] = glGenLists(1);

glNewList(list[2], GL\_COMPILE);

glBegin(GL\_QUADS);

//North Wall

glColor3f(0, 0, 1);

glNormal3f(0.0, 1.0, 0.0);

glVertex3f(1.0, 1.0, 1.0);

glVertex3f(1.0, 1.0, 0.0);

glVertex3f(0.0, 1.0, 0.0);

glVertex3f(0.0, 1.0, 1.0);

//South Wall

glColor3f(0, 0, 1);

glNormal3f(0.0, -1.0, 0.0);

glVertex3f(1.0, 0.0, 0.0);

glVertex3f(1.0, 0.0, 1.0);

glVertex3f(0.0, 0.0, 1.0);

glVertex3f(0.0, 0.0, 0.0);

glEnd();

glEndList();

list[3] = glGenLists(1);

glNewList(list[3], GL\_COMPILE);

glBegin(GL\_QUADS);

//North Wall

glColor3f(0, 0, 1);

glNormal3f(0.0f, 1.0f, 0.0f);

glVertex3f(1.0, 1.0, 1.0);

glVertex3f(1.0, 1.0, 0.0);

glVertex3f(0.0, 1.0, 0.0);

glVertex3f(0.0, 1.0, 1.0);

//East Wall

glColor3f(0, 0, 1);

glNormal3f(1.0, 0.0, 0.0);

glVertex3f(1.0, 1.0, 0.0);

glVertex3f(1.0, 1.0, 1.0);

glVertex3f(1.0, 0.0, 1.0);

glVertex3f(1.0, 0.0, 0.0);

glEnd();

glEndList();

list[4] = glGenLists(1);

glNewList(list[4], GL\_COMPILE);

glBegin(GL\_QUADS);

//Top Wall

glColor3f(-1, 0.3, 0);

glNormal3f(1.0, 0.0, 1.0);

glVertex3f(1, 1, 1.0);

glVertex3f(0, 1, 1.0);

glVertex3f(0, 0, 1.0);

glVertex3f(1, 0, 1.0);

glEnd();

glEndList();

}

void init()

{

/\* float color[4];

Enable Lighting.

glEnable(GL\_LIGHT0);

glEnable(GL\_LIGHTING);

Ambient And Diffuse Lighting

glColorMaterial(GL\_FRONT, GL\_AMBIENT\_AND\_DIFFUSE);

glEnable(GL\_COLOR\_MATERIAL);

color[0] = 1.0f; color[1] = 1.0f; color[2] = 0.0f; color[3] = 0.0f;

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, color);

color[0] = 1.0f; color[1] = 0.0f; color[2] = 1.0f; color[3] = 1.0f;

glLightfv(GL\_LIGHT0, GL\_AMBIENT, color);\*/

glEnable(GL\_NORMALIZE);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluPerspective(60, 1.33, 0.005, 100);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

gluLookAt(-1.5, 0, 40, -1.5, 0, 0, 0.0f, 1.0f, 0.0f);

}

void erase()

{

glColor3f(0.1, 0.0, 0.0);

glBegin(GL\_POLYGON);

glVertex2f(0, 0);

glVertex2f(0.5, 0);

glVertex2f(0.25, 0.5);

glEnd();

}

int main(int argc, char\*\* argv)

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_RGBA | GLUT\_DOUBLE | GLUT\_DEPTH);

glutInitWindowSize(1200, 780);

glutInitWindowPosition(0, 0);

glutCreateWindow("Pac GL 3D");

init();

glutDisplayFunc(RenderScene);

create\_list\_lib();

glutKeyboardFunc(mykey);

glutSpecialFunc(specialDown);

glutSpecialUpFunc(specialUp);

glEnable(GL\_DEPTH\_TEST);

int start\_x[4] = { 11,12,15,16 };

for (int ISO = 0; ISO < num\_ghosts; ISO++)

{

ghost[ISO] = new Ghost(start\_x[ISO], 14);

}

float ghost\_colors[4][3] = { {255,0,0},{120,240,120},{255,200,200},{255,125,0} };

int ISO;

for (ISO = 0; ISO < num\_ghosts; ISO++)

{

ghost[ISO]->x = start\_x[ISO];

ghost[ISO]->y = 14;

ghost[ISO]->eaten = false;

ghost[ISO]->max\_speed = 0.1 - 0.01 \* (float)ISO;

ghost[ISO]->speed = ghost[ISO]->max\_speed;

//colorize ghosts

for (int j = 0; j < 3; j++)

ghost[ISO]->color[j] = ghost\_colors[ISO][j] / 255.0f;

}

for (ISO = 0; ISO < BOARD\_X; ISO++)

{

for (int j = 0; j < BOARD\_Y; j++)

{

tp\_array[ISO][j] = pebble\_array[ISO][j];

}

}

pebbles\_left = 244;

glShadeModel(GL\_SMOOTH);

glutMainLoop();

return 0;

}

**CHAPTER 8**

**CONCLUSION AND FUTURE**

**ENHANCEMENTS**

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.

In future the following enhancement could be done:

* Providing Camara Movement
* Providing More Number of Levels.
* Providing High Quality Graphics.

**BIBILOGRAPHY**

1. The Red Book-OpenGL programming Guide, 6th edition
2. Edward Angel Interactive Computer Graphics A Top-Down Approch with OpenGL, 5th edition, Addison and Wesly.
3. <http://www.opengl.org/registry/>
4. [www.codeguru.com](http://www.codeguru.com)
5. [www.openglforum.com](http://www.openglforum.com)
6. [www.nehe.com](http://www.nehe.com)