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 or 3D pattern-recognition abilities allow us to perceive and process pictorial data rapidly.
- Computers have become a powerful medium for the rapid and economical production of pictures.
- There is virtually no area in which graphical displays cannot be used to some advantage.
- Graphics provide a so natural means of communicating with the computer that they have become widespread.
- Interactive graphics is the most important means of producing pictures since the invention of photography and television.
- We can make pictures of not only the real-world objects but also of abstract objects such as mathematical surfaces on 4D and of data that have no inherent geometry.
- A computer graphics system is a computer system with all the components of the general-purpose computer system. There are five major elements in system: input devices, processor, memory, frame buffer, output devices.

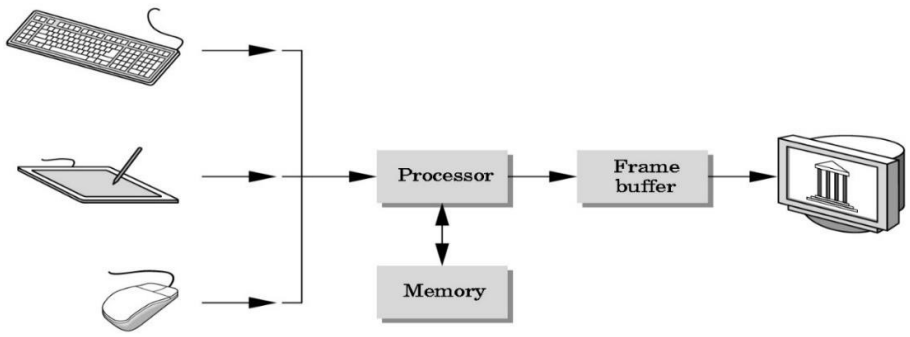

Fig 1.1 Graphics System

1.2 OpenGL Technology

OpenGL is the premier environment for developing portable, interactive 2D and 3D graphics applications. Since its introduction in 1992, OpenGL has become the industries most widely used and supported 2D and 3D graphics application programming interface (API), bringing thousands of applications to a wide variety of computer platforms.

OpenGL fosters innovation and speeds application development by incorporating a broad set of rendering, texture mapping, special effects, and other powerful visualization functions. Developers can leverage the power of OpenGL across all popular desktop and workstation platforms, ensuring wide application deployment.

OpenGL Available Everywhere: Supported on all UNIX® workstations, and shipped standard with every Windows 95/98/2000/NT and MacOS PC, no other graphics API operates on a wider range of hardware platforms and software environments.

OpenGL runs on every major operating system including Mac OS, OS/2, UNIX, Windows 95/98, Windows 2000, Windows NT, Linux, OPENStep, and BeOS; it also works with every major windowing system, including Win32, MacOS, Presentation Manager, and X-Window System. OpenGL is callable from Ada, C, C++, Fortran, Python, Perl and Java and offers complete independence from network protocols and topologies.

The OpenGL interface

Our application will be designed to access OpenGL directly through functions in three libraries namely: gl, glu, glut.

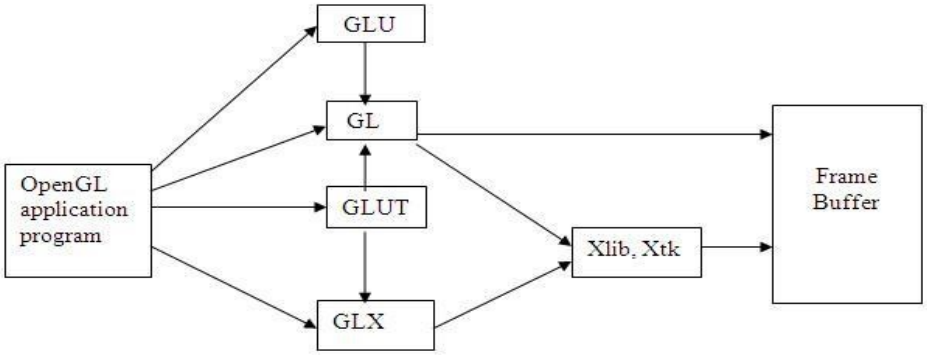

Fig 1.2 OpenGL Libraries

Chapter 2

REQUIREMENTS AND SPECIFICATIONS

2.1 Hardware Requirements

The standard output device is assumed to be a **Color Monitor**. It is quite essential for any graphics package to have this, as provision of color options to the user is a must. The **mouse**, the main input device, has to be functional i.e. used to move the car left or right in the game. A **keyboard** is used for controlling and inputting data in the form of characters, numbers. Apart from these hardware requirements there should be sufficient hard disk space and primary memory available for proper working of the package to execute the program. Pentium III or higher processor, 16MB or more RAM. A functional display card.

Minimum Requirements expected are cursor movement, creating objects like lines, squares, rectangles, polygons, etc. Transformations on objects/selected area should be possible. Filling of area with the specified color should be possible.

2.2 Software Requirements

The editor has been implemented on the OpenGL platform and mainly requires an appropriate version of a C compiler to be installed and functional in Ubuntu. Though it is implemented in Open GL, it is pretty much performed and independent with the restriction that there is support for the execution of C and C++ files. Text Modes is recommended.

Developed Platform

Ubuntu **Language Used in**

Coding C++

Chapter 3

SOFTWARE DESIGN

3.1 SYSTEM DESIGN

Existing System

Existing system for a graphics is the TC++. This system will support only the 2D graphics. 2D graphics package being designed should be easy to use and understand. It should provide various option such as free hand drawing, line drawing, polygon drawing, filled polygons, flood fill, translation, rotation, scaling, clipping etc.

PROPOSED SYSTEM

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

1. Open GL is designed as a streamlined.
2. It's a hardware independent interface i.e. it can be implemented on many different hardware platforms.
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 transformations.
5. It is event driven software.
6. It provides call back function.

3.2 DETAILED DESIGN

Transformation functions

Matrices allow arbitrary linear transformations to be represented in a consistent format, suitable for computation. This also allows transformations to be concatenated easily (by multiplying the matrices).

Linear transformations are not the only ones that can be represented by matrices. Using homogenous coordinates, both affine transformation and perspective projection on $\mathbf{R}^n$ can be represented as linear transformations on $\mathbf{RP}^{n+1}$ (that is, n+1-dimensional real projective space). For this reason, 4x4 transformation matrices are widely used in 3D computer graphics.

3-by-3 or 4-by-4 transformation matrices containing homogeneous coordinates are often called, somewhat improperly, "*homogeneous transformation matrices*". However, the transformations they represent are, in most cases, definitely non-homogeneous and non-linear (like translation, roto-translation or perspective projection). And even the matrices themselves look rather heterogeneous, i.e. composed of different kinds of elements (see below). Because they are multi-purpose transformation matrices, capable of representing both affine and projective transformations, they might be called "*general transformation matrices*", or, depending on the application, "*affine transformation*" or "*perspective projection*" matrices. Moreover, since the homogeneous coordinates describe a projective vector space, they can also be called "*projective space transformation matrices*".

Finding the matrix of a transformation

If one has a linear transformation $T(x)$ in functional form, it is easy to determine the transformation matrix **A** by simply transforming each of the vectors of the standard basis by T and then inserting the results into the columns of a matrix. In other words,

$$\mathbf{A} = \begin{bmatrix} T(\vec{e}_1) & T(\vec{e}_2) & \cdots & T(\vec{e}_n) \end{bmatrix}$$

For example, the function $T(x) = 5x$ is a linear transformation. Applying the above process (suppose that $n = 2$ in this case) reveals that

$$T(\vec{x}) = 5\vec{x} = \begin{bmatrix} 5 & 0 \\ 0 & 5 \end{bmatrix} \vec{x}$$

Examples in 2D graphics

Most common geometric transformations that keep the origin fixed are linear, including rotation, scaling, shearing, reflection, and orthogonal projection; if an affine transformation is not a pure translation it keeps some point fixed, and that point can be chosen as origin to make the transformation linear. In two dimensions, linear transformations can be represented using a 2×2 transformation matrix.

Rotation:

For rotation by an angle θ anticlockwise about the origin, the functional form is $x' = x\cos\theta - y\sin\theta$ and $y' = x\sin\theta + y\cos\theta$. Written in matrix form, this becomes:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Similarly, for a rotation clockwise about the origin, the functional form is $x' = x\cos\theta + y\sin\theta$ and $y' = -x\sin\theta + y\cos\theta$ and the matrix form is:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Scaling:

For scaling (that is, enlarging or shrinking), we have $x' = s_x \cdot x$ and $y' = s_y \cdot y$. The matrix form is:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

When $s_x s_y = 1$, then the matrix is a squeeze mapping and preserves areas in the plane.

Shearing:

For shear mapping (visually similar to slanting), there are two possibilities. For a shear parallel to the x axis has $x' = x + ky$ and $y' = y$; the shear matrix, applied to column vectors, is:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & k \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

A shear parallel to the y axis has $x' = x$ and $y' = y + kx$, which has matrix form:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ k & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Chapter 4

IMPLEMENTATION

4.1 Inbuilt Functions Used

Void glColor3f (float red, float green, float blue);

This function is used to mention the color in which the pixel should appear. The number 3 specifies the number of arguments that the function would take. 'f' gives the type that is float. The arguments are in the order RGB (Red, Green, Blue). The color of the pixel can be specified as the combination of these 3 primary colors.

Void glClearColor (int red, int green, int blue, int alpha);

This function is used to clear the color of the screen. The 4 values that are passed as arguments for this function are (RED, GREEN, BLUE, ALPHA) where the red green and blue components are taken to set the background color and alpha is a value that specifies depth of the window. It is used for 3D images.

void glutKeyboardFunc ();

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

Where func is the new keyboard callback function. **glutKeyboardFunc** sets the keyboard callback for the current window. When a user types into the window, each key press generating an ASCII character will generate a keyboard callback. The key callback parameter is the generated ASCII character. The x and y callback parameters indicate the mouse location in window relative coordinates when the key was pressed. When a new window is created, no keyboard callback is initially registered, and ASCII key strokes in the window are ignored. Passing NULL to glutKeyboardFunc disables the generation of keyboard callbacks.

void glFlush ();

Different GL implementations buffer commands in several different locations, including network buffers and the graphics accelerator itself. **glFlush** empties all of these buffers, causing all issued commands to be executed as quickly as they are accepted by the actual rendering engine. Though this execution may not be completed in any particular time period, it does complete in finite time.

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

glutInit will initialize the GLUT library and negotiate a session with the window system. During this process, glutInit may cause the termination of the GLUT program with an error message to the user if GLUT cannot be properly initialized. Examples of this situation include the failure to connect to the window system, the lack of window system support for OpenGL, and invalid command line options. GlutInit also processes command line options, but the specific options parse is window system dependent.

void glutMainLoop (void);

glutMainLoop enters the GLUT event processing loop. This routine should be called at most once in a GLUT program. Once called, this routine will never stop.

gluOrtho2D ();

gluOrtho2D sets up a two-dimensional orthographic viewing region. This is equivalent to calling glOrtho with near = -1 and far = 1.

glLoadIdentity ();

glLoadIdentity replaces the current matrix with the identity matrix. It is semantically equivalent to calling glLoadMatrix() with the identity matrix 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 but in some cases, it is more efficient

glutInit (&argc, argv);

glutInit will initialize the GLUT library and negotiate a session with the window system. During this process, glutInit may cause the termination of the GLUT program with an error message to the user if GLUT cannot be properly initialized. Examples of this situation include the failure to connect to the window system, the lack of window system support for OpenGL, and invalid command line options.

glutInitWindowPosition ();

Windows created by glutCreateWindow will be requested to be created with the current *initial window position.*

glutCreateWindow ();

glutCreateWindow creates a top-level window. The name will be provided to the window system as the window's name. The intent is that the window system will label the window with the name.

glutWindowSize ();

Windows created by glutCreateWindow will be requested to be created with the current *initial size*.

glClearColor ();

glClearColor specifies the red, green, blue, and alpha values used by glClear to clear the color buffers. Values specified by glClearColor are clamped to the range 0 - 1.

glColor3f ();

The GL stores both a current single-valued color index and a current four-valued RGBA color. glColor sets a new four-valued RGBA color. glColor has two major variants: glColor3 and glColor4. glColor3variants specify new red, green, and blue values explicitly and set the current alpha value to 1.0 (full intensity) implicitly. glColor4 variants specify all four-color components explicitly.

4.2 User-Defined Functions Used

void msg(char * arhv);

This function is used to output the message on the main menu.

void display1();

This function is used to draw the background of the main menu of the game.

int tree1 (int argc1 ,int argc2);

This function draws the tree for the left tree.

void tree();

This function specifies where to draw the trees on the left of the background and how the tree moves.

void rtree(int argc1,int argc2);

This function draws the tree for the left tree.

void rtree1();

This function specifies where to draw the trees on the right of the background and how the tree moves.

void myinit()

This function initializes the program's colour and background.

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

The main function of the C++ program, where the execution starts.

void update (int argc);

This function updates the monkey on the screen according to the given mouse commands.

void bitmap_output(int argc, int argc2, char *argv, void * font)

This function prints the desired message unto the screen.

void display();

This function is the main display function .Initializes all other drawing functions and colours and backgrounds.

void grass1(int argc)

This function is used to draw the grass.

void grass()

This function is used to give the positions of the grass to be drawn.

void back_ground()

This function is used to draw the background of the main game.

void Flappy()

This function is used to draw the monkey

void hurdle()

This function is used to draw the hurdle

void load()

This function is used to load the main screen of the game

void delay(int argc)

This function is used to give the delay between loading screens

void restart()

This function is used to restart the game.

void reset()

This function is used to reset the game upon crashing

void mouse1(int button, int state, int x, int y)

This function is used in the main game to control the monkey

void mouse(int button, int state, int x, int y)

This function is used to control the game after game over or monkey crashes.

void keyboard1(unsigned char key, int x, int y)

This function is used to start ,restart or reset the game

void keyboard2(unsigned char key, int x, int y)

This function is used to control the game in the main menu.

Chapter 5

TESTING AND RESULTS

The full designing and creating of Money Jump has been executed under ubuntu operating system using g++ compiler.This platform satisfies the basic need of a good compiler. Using GL/glut.h library and built in functions make it easy to design good graphics package such as this racing game.

This game allows the user to start the game with keyboard function following a loading screen which prompts to the game. Now the user starts the game by pressing a keyboard key where in the game starts with the Monkey ready to jump left or right.

As and when the obstacles with different speed and size arise, the Monkey has to avoid the obstacles and jump towards left or right increasing the score which is time dependent. When the user fails to avoid the obstacle, then objective of the game is nullified promoting the game to end with a display message quoting “GAME OVER”.

Testing involves unit testing, module testing and system testing.

UNIT TESTING

Here the individual components are tested to ensure that they operate correctly. Each component is tested independently, without other system components.

MODULE TESTING

Module is a collection of dependent components such as procedures and functions. Since the module encapsulates related components can be tested with our other system modules. The testing process is concerned with finding errors which results from erroneous function calls from the main function to various individual functions.

SYSTEM TESING

The Modules are integrated to make up the entire system. The testing process is concerned with finding errors with the results from unanticipated interactions between module and system components. It is also concerned with validating that the system meets its functional and non-functional requirements.

Chapter 6

IN-GAME SNAPSHOTS

Fig 6.1:This is the Main Screen .You can press the S key to start the game

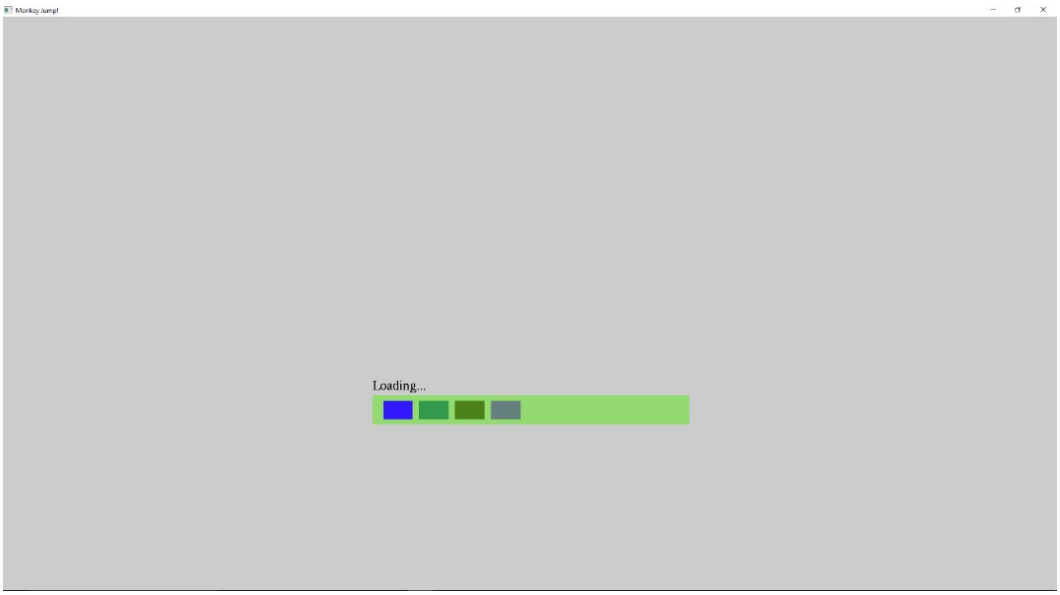

Fig 6.2

This is the loading screen of the Game.This comes everytime you load the game from the main screen.

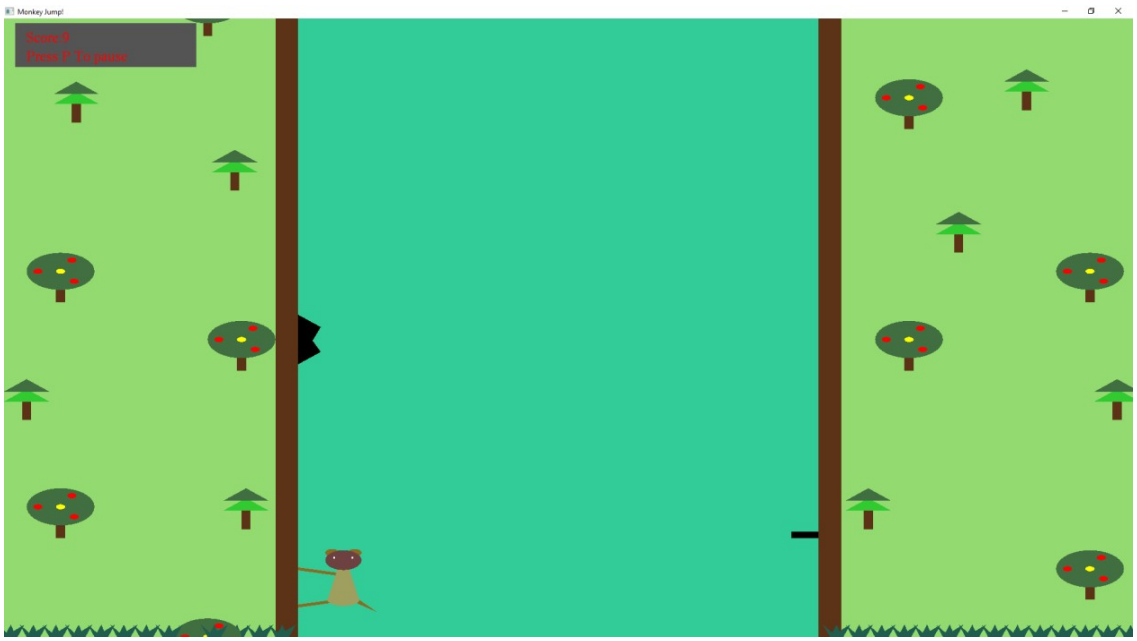

Fig 6.3: This is the Game screen,where we can see that the monkey is flapping from one side of the wall to the other,avoiding obstacles

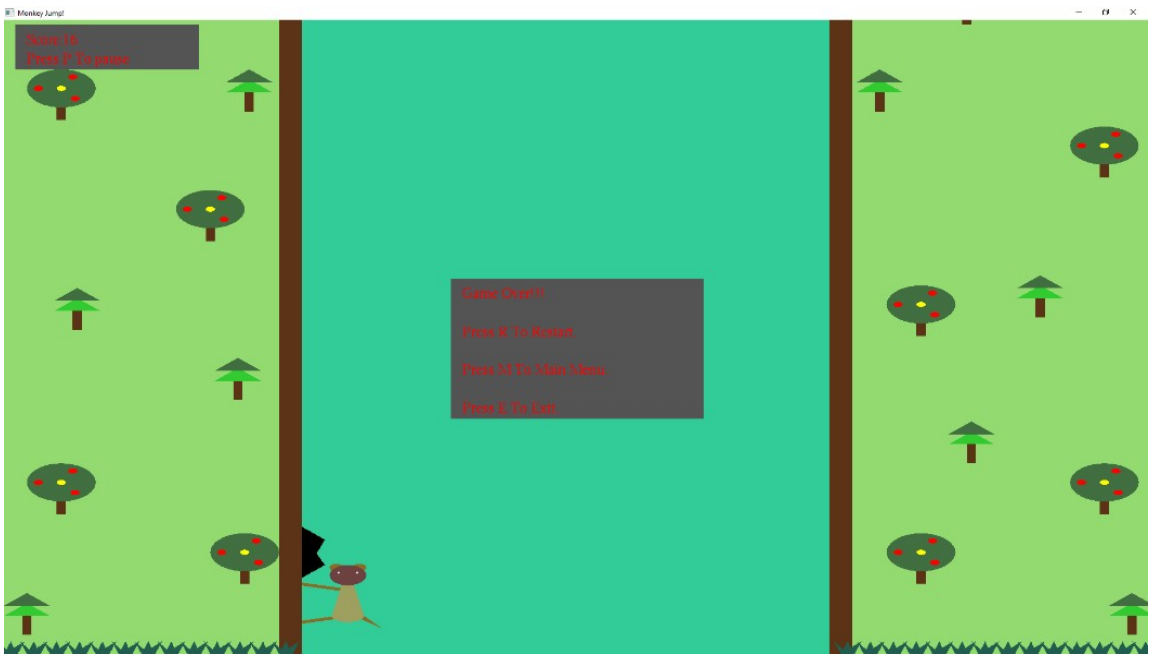

Fig 6.4:If the monkey hits an obstacle ,the game gets over and the score is displayed

REFERENCES

[1] Interactive Computer Graphics A top –down approach with OpenGL – Edward Angel, 5th Edition, Addition-Wesley 2008.

[2] Computer Graphics Using OpenGL - F.S.Hill.Jr. 2nd Edition, Pearson Education, 2009

[3] www.opengl.org

[4] https://www.youtube.com/watch?v=527bR2JHSR0

[5] https://www.youtube.com/watch?v=lKbFZ0RUm3k

[6] https://www.youtube.com/watch?v=u55fkIYlBs8

APPENDIX

```
void msg(char *string)
{
	while (*string)
		glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, *string++);
}
void update(int value)
{
	if (!gameoverflag && !pause)
	{
		hmove1 += 5;
		hmove2 += 5;
		tmove1 += 5;
		tmove2 += 5;
		if (hmove1 > 500) { hmove1 = -100 ;}
		if (hmove2 > 500) { hmove2 = -100 ;}
		if (tmove1 > 500) { tmove1 = -500 ;}
		if (tmove2 > 1000) { tmove2 = 0 ;}
		glFlush();
		glutSwapBuffers();
		glutPostRedisplay();
		glutTimerFunc(20 , update, 0); //1st arg gives delay time
	}
}
void tree1(int t1, int t2)
{
	glColor3f(0.196078, 0.8, 0.196078);//Leaf2
	glBegin(GL_POLYGON);
	glVertex2f(t1-8, t2+15);
	glVertex2f(t1+12, t2+15);
	glVertex2f(t1+2, t2+25);
	glEnd();
```

```
        glColor3f(0.258824, 0.435294, 0.258824);//Leaf1
        glBegin(GL_POLYGON);
        glVertex2f(t1 - 8, t2 + 23);
        glVertex2f(t1+12, t2+23);
        glVertex2f(t1+2, t2+33);
        glEnd();
        glColor3f(0.36, 0.20, 0.09);//Trunk
        glBegin(GL_POLYGON);
        glVertex2f(t1, t2+15);
        glVertex2f(t1+4, t2+15);
        glVertex2f(t1+4, t2);
        glVertex2f(t1, t2);
        glEnd();

        glutSwapBuffers();
}
void tree()
{
        tree1(100,0 - tmove1);tree1(48,125 - tmove1);tree1(105,327 - tmove1);tree1(8,420 -
tmove1);////Trees In Left Bg

        tree1(390,0 - tmove1);tree1(460,135 - tmove1);tree1(380,327 -
tmove1);tree1(490,420 - tmove1);//Trees In Right Bg

        tree1(100,600 - tmove2);tree1(30,660 - tmove2);tree1(105,827 -
tmove2);tree1(50,920 - tmove2);//Trees In Left Bg - Frame Above

        tree1(380,827 - tmove2);tree1(450,670 - tmove2);tree1(420,550 -
tmove2);tree1(490,920 - tmove2);//Trees In Right Bg - Frame Above
        glutSwapBuffers();
}
void rtree1(int rt1, int rt2)
{
        glColor3f(0.36, 0.20, 0.09);//Round Tree Trunk
```

```
glBegin(GL_POLYGON);
glVertex2f(rt1 - 2, rt2);
glVertex2f(rt1 + 2, rt2);
glVertex2f(rt1 + 2, rt2 - 25);
glVertex2f(rt1 - 2, rt2 - 25);
glEnd();

glColor3f(0.258824, 0.435294, 0.258824);//Round Tree
glBegin(GL_POLYGON);
for (int k = 0; k <= 36; k++)
{
        glVertex2f(rt1 + 15 * cos(2 * 3.14*k / 36), rt2 + 15 * sin(2 * 3.14*k / 36));
}
glEnd();

glColor3f(1.0, 1.0, 0.0);//Fruit1
glBegin(GL_POLYGON);
for (int k = 0; k <= 36; k++)
{
        glVertex2f(rt1 + 2 * cos(2 * 3.14*k / 36), rt2 + 2 * sin(2 * 3.14*k / 36));
}
glEnd();

glColor3f(1.0, 0.0, 0.0);//Fruit2
glBegin(GL_POLYGON);
for (int k = 0; k <= 36; k++)
{
        glVertex2f((rt1 - 10) + 2 * cos(2 * 3.14*k / 36), rt2 + 2 * sin(2 * 3.14*k / 36));
}
glEnd();

glColor3f(1.0, 0.0, 0.0);//Fruit3
glBegin(GL_POLYGON);
for (int k = 0; k <= 36; k++)
```

```
        {
                glVertex2f((rt1 + 5) + 2 * cos(2 * 3.14*k / 36), rt2 + 9 + 2 * sin(2 * 3.14*k /
36));
        }
        glEnd();

        glColor3f(1.0, 0.0, 0.0);//Fruit4
        glBegin(GL_POLYGON);
        for (int k = 0; k <= 36; k++)
        {
                glVertex2f((rt1 + 6) + 2 * cos(2 * 3.14*k / 36), rt2 - 8 + 2 * sin(2 * 3.14*k /
36));
        }
        glEnd();

        glutSwapBuffers();

}

void rtree()
{
        rtree1(18,25 - tmove1);rtree1(90,240 - tmove1);rtree1(25,350 -
tmove1);rtree1(105,480 - tmove1);//Rtrees In Left Bg

        rtree1(470,25 - tmove1);rtree1(400,175 - tmove1);rtree1(480,300 -
tmove1);rtree1(400,480 - tmove1);//Rtrees In Right Bg

        rtree1(25,540 - tmove2);rtree1(90,750 - tmove2);rtree1(25,850 - tmove2);//Rtrees In
Left Bg - Frame Above

        rtree1(480,800 - tmove2);rtree1(400,680 - tmove2);rtree1(420,920 -
tmove2);rtree1(480,540 - tmove2);//Rtrees In Right Bg - Frame Above
        glutSwapBuffers();
}
```

```
void grass1(int g1)
{
	glColor3f(0.13, 0.37, 0.31);//Grass
	glBegin(GL_POLYGON);
	glVertex2f(g1, 0 );
	glVertex2f(g1-3, 10 );
	glVertex2f(g1+2, 2 );
	glVertex2f(g1+3, 10 );
	glVertex2f(g1+5, 2 );
	glVertex2f(g1+9, 10 );
	glVertex2f(g1+6, 0 );
	glEnd();

	glutSwapBuffers();
}

void grass()
{
	grass1(0);grass1(10);grass1(20);grass1(30);grass1(40);grass1(50);grass1(60);grass1(70);grass1(80);grass1(90);grass1(100);grass1(110);grass1(120);//Grass In Left Bg

	grass1(365);grass1(375);grass1(385);grass1(395);grass1(405);grass1(415);grass1(425);grass1(435);grass1(445);grass1(455);grass1(465);grass1(475);grass1(485);grass1(495);//Grass In Right Bg
	glutSwapBuffers();
}

void Flappy()
{
	glColor3f(0.556863, 0.419608, 0.137255);//Upper Hand
	glLineWidth(5);
	glBegin(GL_LINES);
	glVertex2f(m2 , 55);
```

```
glVertex2f(m1 , 50);
glEnd();

glColor3f(0.556863, 0.419608, 0.137255);//Lower Hand
glLineWidth(5);
glBegin(GL_LINES);
glVertex2f(m2 , 25);
glVertex2f(m1 , 30);
glEnd();

glColor3f(0.556863, 0.419608, 0.137255);//Tail
glBegin(GL_POLYGON);
glVertex2f(m3 , 28);
glVertex2f(m4 , 30);
glVertex2f(m5 , 20);
glEnd();

glColor3f(0.623529, 0.623529, 0.372549);//Bottom Body
glBegin(GL_POLYGON);
for (int k = 0; k <= 36; k++)
{
        glVertex2f(m1 + 7 * cos(2 * 3.14*k / 36), 30 + 5 * sin(2 * 3.14*k / 36));
}
glEnd();

glColor3f(0.623529, 0.623529, 0.372549);//Body
glBegin(GL_POLYGON);
glVertex2f(m1 , 70);
glVertex2f(m1 - 7 , 30);
glVertex2f(m1 + 7 , 30);
glEnd();

glColor3f(0.556863, 0.419608, 0.137255);//Left Ear
glBegin(GL_POLYGON);
```

```
        for (int k = 0; k <= 36; k++)
        {
                glVertex2f((m1 - 5) + 3 * cos(2 * 3.14*k / 36), 68 + 3 * sin(2 * 3.14*k / 36));
        }
        glEnd();

        glColor3f(0.556863, 0.419608, 0.137255);//Right Ear
        glBegin(GL_POLYGON);
        for (int k = 0; k <= 36; k++)
        {
                glVertex2f((m1 + 5) + 3 * cos(2 * 3.14*k / 36), 68 + 3 * sin(2 * 3.14*k / 36));
        }
        glEnd();

        glColor3f(0.435294, 0.258824, 0.258824);//Head
        glBegin(GL_POLYGON);
        for (int k = 0; k <= 36; k++)
        {
                glVertex2f(m1 + 8 * cos(2 * 3.14*k / 36), 62 + 8 * sin(2 * 3.14*k / 36));
        }
        glEnd();
        glFlush();

        glColor3f(1.0, 1.0, 1.0);//Eyes
        glPointSize(3);
        glBegin(GL_POINTS);
        glVertex2f(m1 - 4 , 64);
        glVertex2f(m1 + 4 , 64);
        glEnd();

        glutSwapBuffers();
}
void hurdle()
{
```

```
	glBegin(GL_POLYGON);
	glColor3f(0.0, 0.0, 0.0);
	glVertex2f(130, 460 - hmove1);
	glVertex2f(140, 470 - hmove1);
	glVertex2f(130, 495 - hmove1);
	glVertex2f(140, 490 - hmove1);
	glVertex2f(130, 500 - hmove1);
	glEnd();

	newx1 = 130; newy1 = 460 - hmove1;

	glBegin(GL_POLYGON);
	glColor3f(0.0, 0.0, 0.0);
	glVertex2f(360, 320 - hmove2);
	glVertex2f(348, 320 - hmove2);
	glVertex2f(348, 325 - hmove2);
	glVertex2f(360, 325 - hmove2);
	glEnd();

	newx2 = 360; newy2 = 320 - hmove2;

	{
		if ((mx1 == newx1 - (newx1 % 5) && my == newy1 - (newy1 % 5)) || (mx1
== newx2 - (newx2 % 5) && my == newy2 - (newy2 % 5)) )
		{
			gameoverflag = 1;
		}
	}

	glutSwapBuffers();
}

void display()
```

```
{
        glClear(GL_COLOR_BUFFER_BIT);
        glClearColor(1.0, 1.0, 1.0, 1.0);
        fflush(stdin);
        glutKeyboardFunc(keyboard2);
        glutMouseFunc(mouse);
        back_ground();
        Flappy();
        tree();
        grass();
        rtree();
        hurdle();

        if(!gameoverflag)
        {
                score+=0.2;
        }

        int sc = score;//For Score Box
        glBegin(GL_POLYGON);
        glColor3f(0.329412, 0.329412, 0.329412);
        glVertex2f(5, 495);
        glVertex2f(85, 495);
        glVertex2f(85, 460);
        glVertex2f(5, 460);
        glEnd();
        glColor3f(1, 0, 0);
        glRasterPos2f(10, 480);
        printf(tmp, "Score:%d", sc);
        msg(tmp);
        glColor3f(1, 0, 0);
        bitmap_output(10, 465, "Press P To pause", GLUT_BITMAP_TIMES_ROMAN_24);

        if(dmsg)
```

```
	{
		glBegin(GL_POLYGON);
		glColor3f(0.329412, 0.329412, 0.329412);
		glVertex2f(215, 255);
		glVertex2f(280,  255);
		glVertex2f(280,  235);
		glVertex2f(215,  235);
		glEnd();
		dmsg = 0;
		glColor3f(1, 0, 0);
		bitmap_output(220, 240, "Press S To Begin",
GLUT_BITMAP_TIMES_ROMAN_24);

	}

	if(pause)
	{
		glBegin(GL_POLYGON);
		glColor3f(0.329412, 0.329412, 0.329412);
		glVertex2f(215, 255);
		glVertex2f(290,  255);
		glVertex2f(290,  235);
		glVertex2f(215,  235);
		glEnd();
		dmsg = 0;
		glColor3f(1, 0, 0);
		bitmap_output(220, 240, "Press R To Resume",
GLUT_BITMAP_TIMES_ROMAN_24);
	}

	if(gameoverflag)
	{
		//PlaySoundA(NULL, NULL,
SND_ASYNC|SND_FILENAME|SND_LOOP);
```

```
                //PlaySoundA("end.wav", NULL, SND_ASYNC|SND_FILENAME);
                glBegin(GL_POLYGON);
                glColor3f(0.329412, 0.329412, 0.329412);
                glVertex2f(195, 295);
                glVertex2f(305,  295);
                glVertex2f(305,  185);
                glVertex2f(195,  185);
                glEnd();
                glColor3f(1, 0, 0);
                bitmap_output(200, 280, "Game Over!!!",
GLUT_BITMAP_TIMES_ROMAN_24);
                glColor3f(1, 0, 0);
                bitmap_output(200, 250, "Press R To Restart.",
GLUT_BITMAP_TIMES_ROMAN_24);
                glColor3f(1, 0, 0);
                bitmap_output(200, 220, "Press M To Main Menu.",
GLUT_BITMAP_TIMES_ROMAN_24);
                glColor3f(1, 0, 0);
                bitmap_output(200, 190, "Press E To Exit.",
GLUT_BITMAP_TIMES_ROMAN_24);
                glFlush();
        } glFlush();
        glutSwapBuffers();
}
```