1. INTRODUCTION TO OPEN GL

OpenGL is a software interface to graphics hardware. This interface consists of over 120 distinct commands that we use to specify the objects and operations needed to produce interactive three-dimensional applications.

OpenGL is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms. To achieve these qualities, no commands for performing windowing tasks or obtaining user input are included in OpenGL; instead, we must work through whatever windowing system controls the particular hardware we're using. Similarly, OpenGL doesn't provide high-level commands for describing models of three-dimensional objects. Such commands might allow us to specify relatively complicated shapes such as automobiles, parts of the body, airplanes, or molecules.

With OpenGL, we must build up our desired model from a small set of *geometric primitives* - points, lines, and polygons. A sophisticated library that provides these features could certainly be built on top of OpenGL. The OpenGL Utility Library (GLU) provides many of the modeling features, such as quadric surfaces and NURBS curves and surfaces. GLU is a standard part of every OpenGL implementation.

In OpenGL *rendering*, is the process by which a computer creates images from models. These *models*, or objects, are constructed from geometric primitives - points, lines, and polygons - that are specified by their vertices. The final rendered image consists of pixels drawn on the screen; a pixel is the smallest visible element the display hardware can put on the screen. Information about the pixels (for instance, what color they're supposed to be) is organized in memory into bitplanes. A bitplane is an area of memory that holds one bit of information for every pixel on the screen; the bit might indicate how red a particular pixel is supposed to be, for example. The bitplanes are themselves organized into a *framebuffer*, which holds all the information that the graphics display needs to control the color and intensity of all the pixels on the screen.

OpenGL is designed as a streamlined, hardware independent interface to be implemented on many different hardware platforms. To achieve these qualities, no commands for performing windowing tasks or obtaining user input are included in OpenGL; instead, you must work through whatever windowing system controls the particular hardware you're using. Similarly, OpenGL doesn't provide high-level commands for describing models of three-dimensional objects. Such commands might allow you to specify relatively complicated shapes such as automobiles, parts of the body, airplanes, or molecules.

OpenGL−related Libraries

OpenGL provides a powerful but primitive set of rendering commands, and all higher level drawing must be done in terms of these commands.

The **OpenGL Utility Library (GLU)** contains several routines that use lower level OpenGL commands to perform such tasks as setting up matrices for specific viewing orientations and projections, performing polygon tessellation, and rendering surfaces. This library is provided as part of your OpenGL implementation.

The **OpenGL Extension to the X Window System (GLX)** provides a means of creating an OpenGL context and associating it with a draw-able window on a machine that uses the X Window System. GLX is provided as an adjunct to OpenGL.

2. PROJECT DESCRIPTION

2.1 OBJECTIVES

The objective of the mini OpenGL project is to implement the concepts learnt in lab sessions as well as Theory classes.

2.2 AIM OF THE PROJECT

Aim of the project is to create the game **"Osmos".** It is a fun filled game where you have to eat out a few cubes and at the same time dodge others.

2.3 DESCRIPTION

The player control a cube with the help of a mouse. There are various other cubes going around in random directions. They are of various sizes, bigger and smaller. The player has to 'eat' the smaller cubes by colliding with them, which makes the player's cube bigger. As the player eats more and more cubes, it can eventually eat out all the cubes in the window. However, if the player's cube collides with a bigger cube, the player's cube itself get eaten by other cubes and the size decreases.

The objective of the game is to become the biggest cube in minimum time. A score is calculated based on the amount of time taken to finish the game. The player loses if the cube becomes the smallest in the window.

.

3. APPLICATION PROGRAMMING INTERFACES USED

1. void glBegin(GLenum mode);

It defines the type of primitives that the vertices define. Each subsequent execution og glVertex3f pecifies the x,y,z coordinates of a location in space.

2. void glEnd(void);

It ends the list of vertices.

3. void glClear(GLbitfield mask);

It is used to make the screen solid and white.

4. void glClearColor(GLclampf red, GLclampf green, GLclampf blue, GLclampf alpha);

The background color is set with this API where the last argument specifies a degree of transparency.

5. void glColor3f(GLfloat red, GLfloat green, GLfloat blue);

It sets color to current drawing.

6. void glLoadIdentity(void);

To initialize the current transform matrix to the identity transform.

7. void glMatrixMode(GLenum mode);

It switches matrix mode between the two matrices –
MODEL_VIEW (GL_MODELVIEW)
PROJECTION (GL_PROJECTION)

8. void glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble zNear, GLdouble zFar);

It establishes as a view volume a parallelepiped that extends from left to right in x, bottom to top in y and near to far in z.

9. void glPushMatrix(void);

It is used to push matrix on the stack.

10. void glPopMatrix(void);

It is used to pop matrix from the stack.

11. void glVertex3f(GLfloat x, GLfloat y, GLfloat z);

It is used to represent vertex.

12. void glutCreateWindow(const char *title);

It creates and opens OpenGL window.

13. void glutDisplayFunc(void (GLUTCALLBACK *func)(void));

It sends graphics to screen.

14. void glutIdleFunc(void (GLUTCALLBACK *func)(void));

It is used to increase theta by fixed amount whenever nothing else is happening.

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

It initiates interaction between windowing system and OpenGL.

16. void glutInitDisplayMode(unsigned int mode);

This function specifies how the display should be initialized. The constants GLUT_SINGLE and GLUT_RGB, which are ORed together, indicate that a single display buffer should be allocated and the colors are specified using desired amount of red, green and blue.

17. void glutInitWindowSize(int width, int height);

It sets the size of created window.

18. void glutMainLoop(void);

It causes the program to begin an event-processing loop.

19. void glutSwapBuffers(void);

We can swap the front and back buffers at will from the application programs.

20. void glutSetCursor(int cursor);

It changes the cursor image of the current window.

21. void glutPassiveMotionFunc(void (*func)(int x, int y));

It sets the passive motion callback for the current window.

4. IMPLEMENTATION DETAILS

The game is mainly split into two objects, the player's cube and other cubes. The methods are listed as below which are used to implement both the objects:

- **void DrawPlayer();**

  This method is used to set the player cube's color and draw it. It takes values from a global array that stores the player's edge sizes called player[].

- **int Timeval_Subtract(struct timeval, struct timeval, struct timeval)**

  This method is used to calculate the score and returns the score back to the calling method GameOver(int, int, char*).

- **void ModifyPlayer(int, int)**

  This method is used to modify the position of the player cube based on the arguments provided by MovePlayer(int, int). It simply changes the values of the player cube's edge positions in the global array player[].

- **void MovePlayer(int, int)**

  This method is used to move the player cube. It takes input from the mouse as mouse pointer co-ordinates and called ModifyPlayer(int, int) with the values after inverting y-axis.

- **void InitializeOthers()**

  This method is used to initialize all other cubes. It randomly selects places where all cubes spawn and also randomly selects their size within given parameters. It then randomly selects one of 8 cardinal directions for the cube to move in. Finally, it saves the edge positions, the size of each cube, and the direction it will travel in, in the global 2D array called others[].

- **void DrawOthers()**

  This method is used to draw all other cubes apart from the player's cube. It first check if the cube is smaller or larger than the player cube and sets the color accordingly. It then draws each cube while fetching values from the others[] array which stores their parameters.

- **void ModifyOthers()**

  This method is used to modify the positions of all other cubes. It sets the speed, and then proceeds to set the movement according to the speed and the direction specified in the others[] array. It accordingly changes the position values in the others[] array. If any cube hits the screen wall, it also changes its direction in order to bounce off the wall. This is done for all four walls.

- **void CollisionDetection()**

  This method is used to detect any collisions between the player cube and other cubes. When an edge of the player cube lies between two opposite edges of any other cube, it has collided with it. CollisionDetection() looks for such detections for all 4 edges of the player cube with all other cubes. If it detects a collision, it calls the CollisionHandler(int) method with the index number of the cube which collide with the player cube. CollisionDetection() takes into account multiple collisions at the same instance of time.

- **void CollisionHandler(int)**

  This method is only called when a collision is detected by the CollisionDetection() method. CollisionHandler accepts the cube number which collided with the player cube as input and manipulates their sizes depending on which cube is bigger.
  If the player cube is larger, it eats the other cube and its size decreases while the other cube's size increases. If the player cube is smaller, it gets eaten by the other cube and its size gets reduced while the other cube's size increases.

- **void GameWinDetection()**

  This method is used to check if the player has won the game. The player wins the game when the player cube size is largest. It checks if the player cube's size is larger than all other cube's sizes and if it is the largest, it sets a flag and calls the GameOver(int, int, char*) method with the string "CONGRATULATIONS! YOU WIN!"

- **void GameLoseDetection()**

  This method is used to check if the player has lost the game. The player loses the game when the player cube size is smallest, or too small to make a difference even by eating smaller cubes. It checks for both of these conditions and calls the GameOver(int, int, char*) method with the string “GAME OVER! BETTER LUCK NEXT TIME."

- **void GameOver(int, int, char*)**

  This method is used to terminate the game, display the “GAME OVER” message and display the score. The score is calculated using the Timeval_Subtract(struct timeval, struct timeval, struct timeval) method. It then prints the score if the player wins the game and prints “You lose!” if the player loses the game. Finally, it calls the glutIdleFunc() method with a “NULL” parameter to stop the continuous calling of the display method.

- **void display()**

  This method is used to call most other functions. It clears the buffer, draws the player cube by calling DrawPlayer(), draws the other cubes by calling DrawOthers(), and displays to the screen using glFlush(). It then calls ModifyOthers() to modify the other cubes, CollisionDetection() to detect any collisions with the player cube, GameWinDetection() and GameLoseDetection to check if the game is over and if it has been won or lost, and finally it swaps buffers.

- **void myinit()**

  This method is used to set the matrix mode and the viewport size. It also sets default colors and initializes the clipping window size using gluOrtho2D(). Finally myinit() calls method InitializeOthers to initialize all other cubes.

- **int main()**

  The main() method is called first and that is where is program begins from. It is used to call all other methods directly or indirectly. In main(), first we set the display mode to DOUBLE, then we initialize the window size to 1366x768. After that we set the window position and create the output window called “Osmos”.
  This is followed by making display() the glutDisplayFunc, making MovePlayer() the glutPassiveMoveFunc, calling myinit to initialize all parameters and glutIdleFunc(display) to call display continuously. Finally, we call glutMainLoop() and return.

5. SOURCE CODE

```
#include<GL/glut.h>
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<sys/time.h>
#define MAX 30

struct timeval begin, end, diff;

GLfloat size = 40;
GLfloat sizeOther[MAX];
//Player square position. Order : LEFT TOP RIGHT BOTTOM
GLint player[4];
//All other squares position AND direction. Yet to be initialised.
GLint others[MAX][5];

GLint flag = 0;

int Timeval_Subtract(struct timeval *result, struct timeval *t2, struct timeval *t1)
{
   long int diff = (t2->tv_usec + 1000000 * t2->tv_sec) - (t1->tv_usec + 1000000 * t1-
>tv_sec);
   result->tv_sec = diff / 1000000;
   result->tv_usec = diff % 1000000;

   return (diff<0);
}

void CollisionHandler(int i)
{
   //sleep(1);

   if(sizeOther[i] > size)
   {
      size = size - (0.02*sizeOther[i]);
      player[2] = player[0]+size;
      player[1] = player[3]+size;


      sizeOther[i] = sizeOther[i] + (0.02*sizeOther[i]);

      others[i][2] = others[i][0] + sizeOther[i];
      others[i][3] = others[i][1] - sizeOther[i];
   }
   else
   {
```

```
        size = size + (0.02*sizeOther[i]);
        player[2] = player[0]+size;
        player[1] = player[3]+size;

        sizeOther[i] = sizeOther[i] - (0.2*sizeOther[i]);

        others[i][2] = others[i][0] + sizeOther[i];
        others[i][3] = others[i][1] - sizeOther[i];
           }

}

void CollisionDetection()
{
   int i, flag = 0;
   for(i = 0; i < MAX; i++)
   {
      if(((others[i][0] >= player[0]) && (others[i][0] <= player[2])) && (((others[i][3] >
player[3]) && (others[i][3]) < player[1]) || ((others[i][1] < player[1]) && (others[i][1] >
player[3]))))
      {
         CollisionHandler(i);
      }
      else if(((others[i][3] <= player[1]) && (others[i][3] >= player[3])) && (((others[i][0]
> player[0]) && (others[i][0]) < player[2]) || ((others[i][2] < player[2]) && (others[i][2] >
player[0]))))
      {
         CollisionHandler(i);
      }
      else if(((others[i][0] <= player[2]) && (others[i][0] >= player[0])) && (((others[i][1]
> player[3]) && (others[i][1]) < player[1]) || ((others[i][3] < player[1]) && (others[i][3] >
player[3]))))
      {
         CollisionHandler(i);
      }
      else if(((others[i][1] >= player[3]) && (others[i][1] <= player[1])) && (((others[i][0]
> player[0]) && (others[i][0]) < player[2]) || ((others[i][2] < player[2]) && (others[i][2] >
player[0]))))
      {
         CollisionHandler(i);
      }
   }
}

void myinit()
{
         glClearColor(1.0,1.0,1.0,1.0);
         glColor3f(1.0,0.0,0.0);
```

```
        gettimeofday(&begin, NULL);

        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0,1366,0,768);
        glMatrixMode(GL_MODELVIEW);
        InitialiseOthers();
}

//Draws player
void DrawPlayer()
{
        glPointSize(2.0);
        glBegin(GL_QUADS);
    glColor3f(0.5,1.0,0.5);
    glVertex2f(player[0], player[3]);
    glVertex2f(player[0], player[1]);
    glVertex2f(player[2], player[1]);
    glVertex2f(player[2], player[3]);
  glEnd();
}

void ModifyPlayer(int x, int y)
{
        player[0] = x;
        player[2] = x+size;
        player[1] = y+size;
        player[3] = y;
}
void MovePlayer(int x, int y)
{
  y=728-abs(y);
  ModifyPlayer(x,y);
}
void InitialiseOthers()
{
        int i, sizebox;
        for(i = 0; i < MAX; i++)
        {
                sizebox = (rand() % 30) + 30;
                others[i][0] = rand() % 750;
                others[i][1] = rand() % 750;
                others[i][2] = others[i][0] + sizebox;
                others[i][3] = others[i][1] - sizebox;
                others[i][4] = rand() % 8;
                sizeOther[i] = (float)sizebox;
        }
}
```

```
void DrawOthers()
{
	int i;
	glPointSize(2.0);
	//glColor3f(0.258824,0.258824,0.435294);

	for(i = 0; i < MAX; i++)
	{
      if(sizeOther[i] > size+1)
	glColor3f(1.0,0.0,0.0);
      else if(sizeOther[i] <= size-1)
	glColor3f(0.0,1.0,0.0);
		glBegin(GL_QUADS);
			glVertex2f(others[i][0], others[i][3]);
			glVertex2f(others[i][0], others[i][1]);
			glVertex2f(others[i][2], others[i][1]);
			glVertex2f(others[i][2], others[i][3]);
		glEnd();
	}
}

void ModifyOthers()
{
	float x, y, speed = 5;
	int i;
	for(i = 0; i < MAX; i++)
	{
		switch(others[i][4])
		{
			case 0:
				x = speed;
				y = 0;
				break;
			case 1:
				x = speed;
				y = -speed;
				break;
			case 2:
				x = 0;
				y = -speed;
				break;
			case 3:
				x = -speed;
				y = -speed;
				break;
			case 4:
				x = -speed;
				y = 0;
				break;
```

```
                        case 5:
                                x = -speed;
                                y = speed;
                                break;
                        case 6:
                                x = 0;
                                y = speed;
                                break;
                        case 7:
                                x = speed;
                                y = speed;
                                break;
                        default:
                                break;
                }
                others[i][0] += x;
                others[i][1] += y;
                others[i][2] += x;
                others[i][3] += y;

//Bounce off LEFT wall
                if(others[i][0] < 0)
                {
                        switch(others[i][4])
                        {
                                case 3:
                                        others[i][4] = 1;
                                        break;
                                case 4:
                                        others[i][4] = 0;
                                        break;
                                case 5:
                                        others[i][4] = 7;
                                        break;
                                default:
                                        break;
                        }
                }
//Bounce off TOP wall
                if(others[i][1] > 768)
                {
                        switch(others[i][4])
                        {
                                case 5:
                                        others[i][4] = 3;
                                        break;
                                case 6:
                                        others[i][4] = 2;
                                        break;
```

```
                        case 7:
                                others[i][4] = 1;
                                break;
                        default:
                                break;
                }
        }
//Bounce off RIGHT wall
        if(others[i][2] > 1366)
        {
                switch(others[i][4])
                {
                        case 0:
                                others[i][4] = 4;
                                break;
                        case 1:
                                others[i][4] = 3;
                                break;
                        case 7:
                                others[i][4] = 5;
                                break;
                        default:
                                break;
                }
        }
//Bounce off BOTTOM wall
        if(others[i][3] < 0)
        {
                switch(others[i][4])
                {
                        case 1:
                                others[i][4] = 7;
                                break;
                        case 2:
                                others[i][4] = 6;
                                break;
                        case 3:
                                others[i][4] = 5;
                                break;
                        default:
                                break;
                }
        }
    }
//      display();
}

void GameWinDetection()
{
```

```
    int i, temp = 0;
    for(i = 0; i < MAX; i++)
    {
        if(sizeOther[i] < size)
            temp++;
    }
    if(temp == MAX)
    {
        flag = 1;
        GameOver(500, 400, "CONGRATULATIONS! YOU WIN!");
    }
}

void GameOver(int x, int y, char *string)
{
    glClear(GL_DEPTH_BUFFER_BIT);
  glColor3f(0,0,0);
  glRasterPos2f(x, y);
  int len, i;
  len = (int)strlen(string);
  for (i = 0; i < len; i++)
  {
    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, string[i]);
  }
  gettimeofday(&end, NULL);
  timeval_subtract(&diff, &end, &begin);
  if(flag == 1)
  {
    //printf("You score: %ld.%02ld!\n", diff.tv_sec, diff.tv_usec);
     printf("You score: %d!\n", 100-(int)diff.tv_sec);
    }
    else
    {
        printf("You Lose!\n");
    }
  glutIdleFunc(NULL);
}

void GameLoseDetection()
{
    int i, temp = 0;
    for(i = 0; i < MAX; i++)
    {
        if(size < sizeOther[i])
            temp++;
    }
    if(temp == MAX || size < 5)
    {
        flag = 0;
```

```
        GameOver(500, 400, "GAME OVER! BETTER LUCK NEXT TIME");
    }
}

void display()
{
        glClear(GL_COLOR_BUFFER_BIT);
        DrawPlayer();
        DrawOthers();
        glFlush();
        ModifyOthers();
        CollisionDetection();
        GameWinDetection();
        GameLoseDetection();
        glutSwapBuffers();
}



int main(int argc, char **argv)
{
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_DOUBLE);
        glutInitWindowSize(1366,768);
        glutInitWindowPosition(100,100);
        glutCreateWindow("Osmos");
        glutDisplayFunc(display);
        glutSetCursor(GLUT_CURSOR_NONE);
        glutPassiveMotionFunc(MovePlayer);
        //glutKeyboardFunc(MovePlayer);
        myinit();
        glutIdleFunc(display);
        glutMainLoop();
        return 0;
}
```

6. OUTPUT

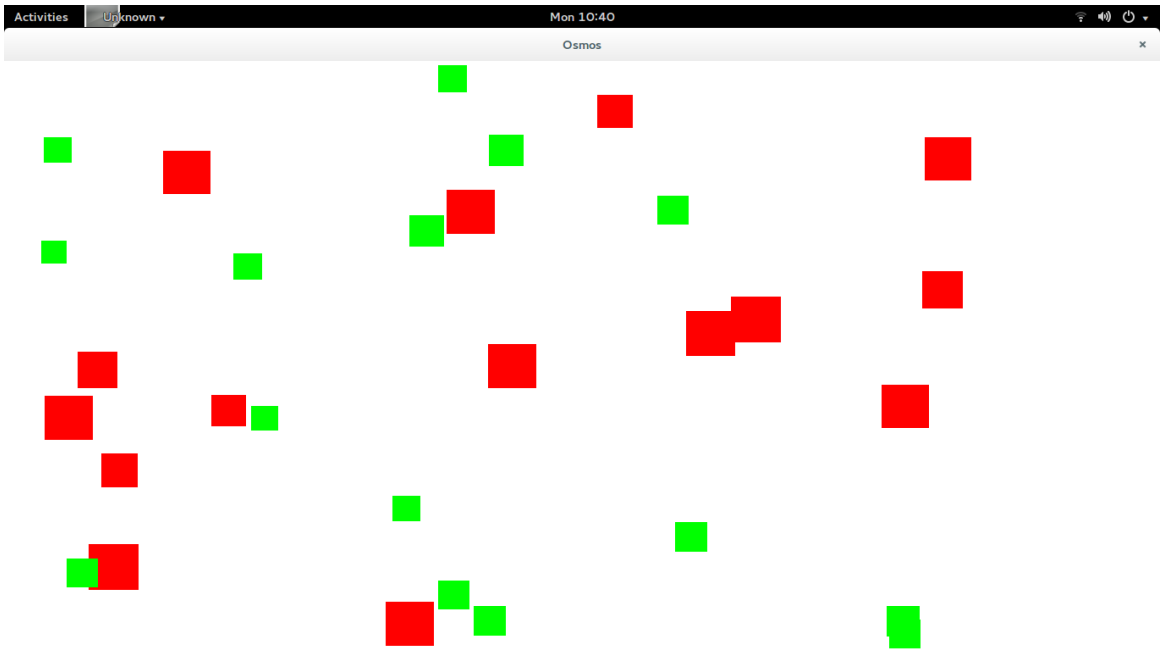

Fig1. The game window; Green cubes are smaller than the player's cube while red cubes are bigger than the player's cube.

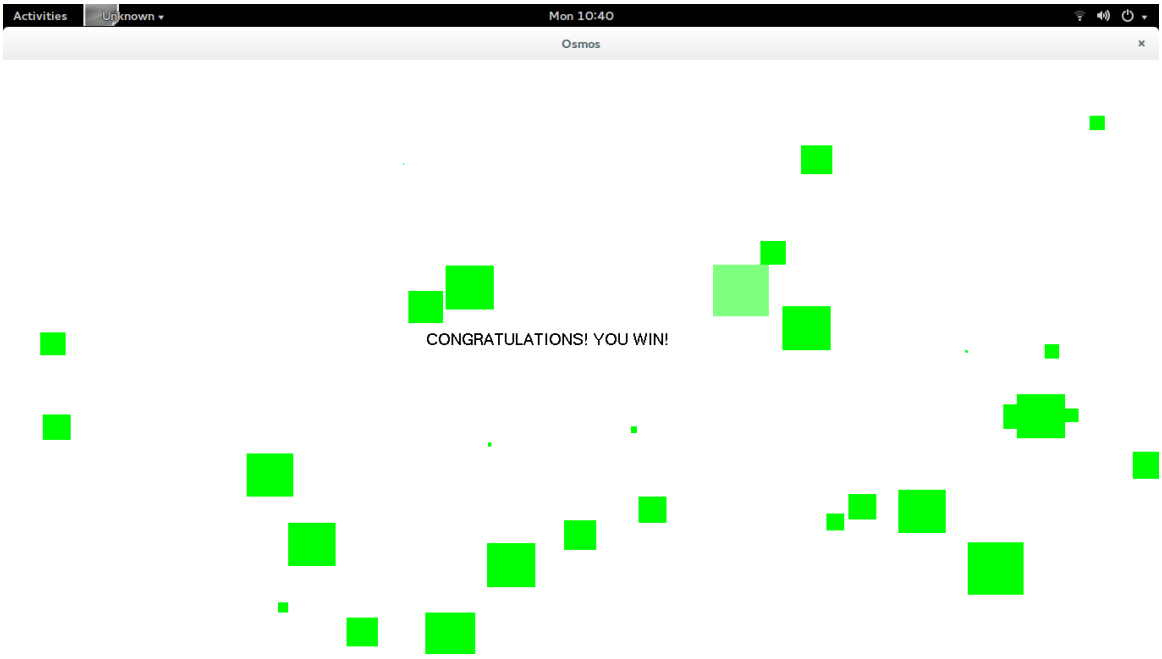

Fig 2. Game Win

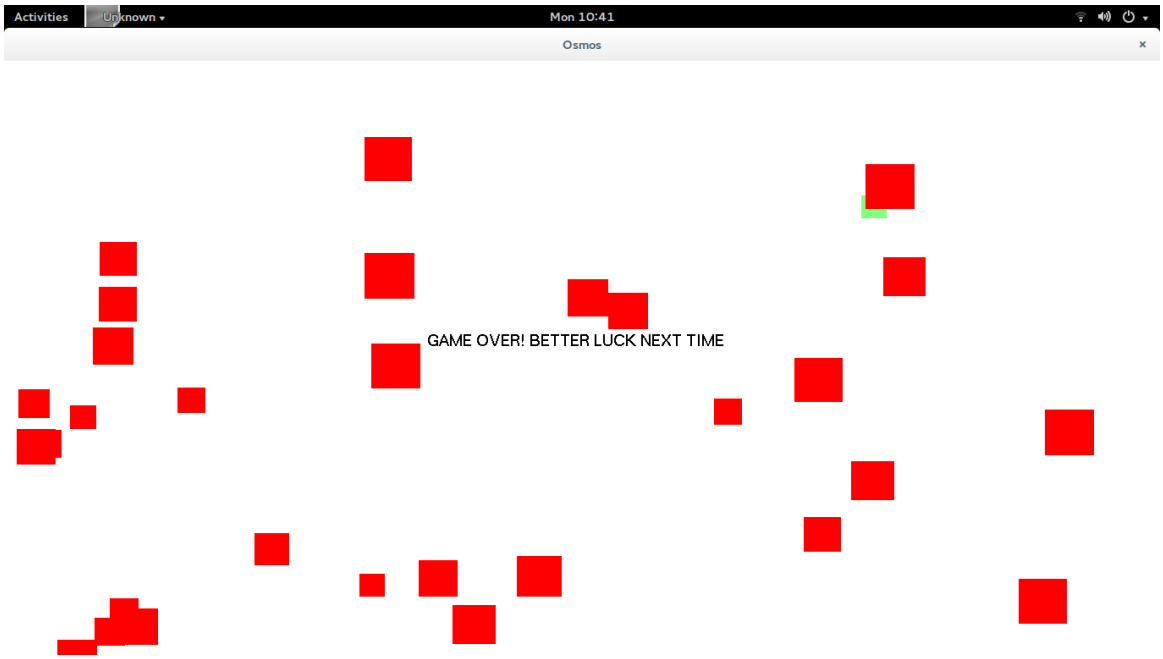

Fig 3. Game Lose

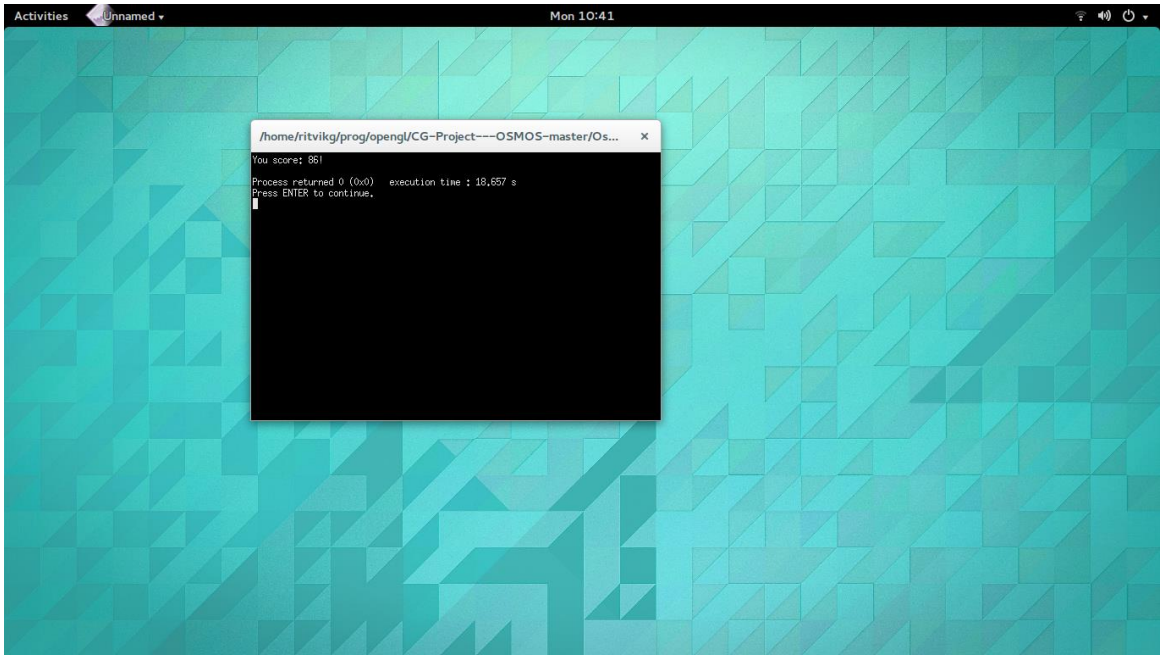
Activities
Unnamed
Mon 10:41
/home/ritvikg/prog/opengl/CG-Project---OSMOS-master/Os...
You score: 861
Process returned 0 (0x0) execution time : 18.657 s
Press ENTER to continue.

Fig 4. Score

7. CONCLUSION

This project helps us to understand the different concepts used in Computer Graphics and Visualization lab like transformation.

We have also learnt how the OpenGL uses built-in functions and algorithms to create complex graphical objects and render it to display.

The OpenGL Utility Toolkit (GLUT) is a programming interface with ANSI C bindings for writing window system independent OpenGL programs.

One of the major accomplishments in the specification of OpenGL was the isolation of window system dependencies from OpenGL's rendering model. The result is that OpenGL is window system independent. Window system operations such as the creation of a rendering window and the handling of window system events are left to the native window system to define. Necessary interactions between OpenGL and the window system such as creating and binding an OpenGL context to a window are described separately from the OpenGL specification in a window system dependent specification.

The GLUT application-programming interface (API) requires very few routines to display a graphics scene rendered using OpenGL. The GLUT API (like the OpenGL API) is stateful. Most initial GLUT state is defined and the initial state is reasonable for simple programs. The GLUT routines also take relatively few parameters. No pointers are returned. The only pointers passed into GLUT are pointers to character strings (all strings passed to GLUT are copied, not referenced) and opaque font handles.

8. BIBLIOGRAPHY

Books referred

1. Interactive Computer Graphics By Edward Angel (5th edition)
2. Computer graphics using OpenGL: F. S. Hill, Jr

Websites referred

http://www.opengl.org/