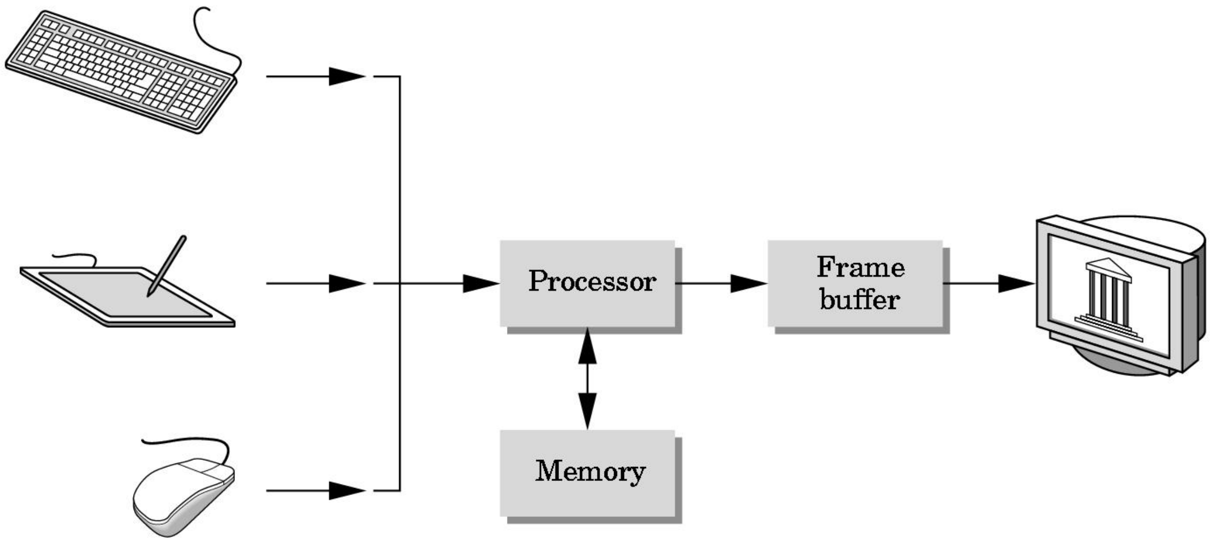
CHAPTER 1
INTRODUCTION

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.



1.2 OpenGL Technology

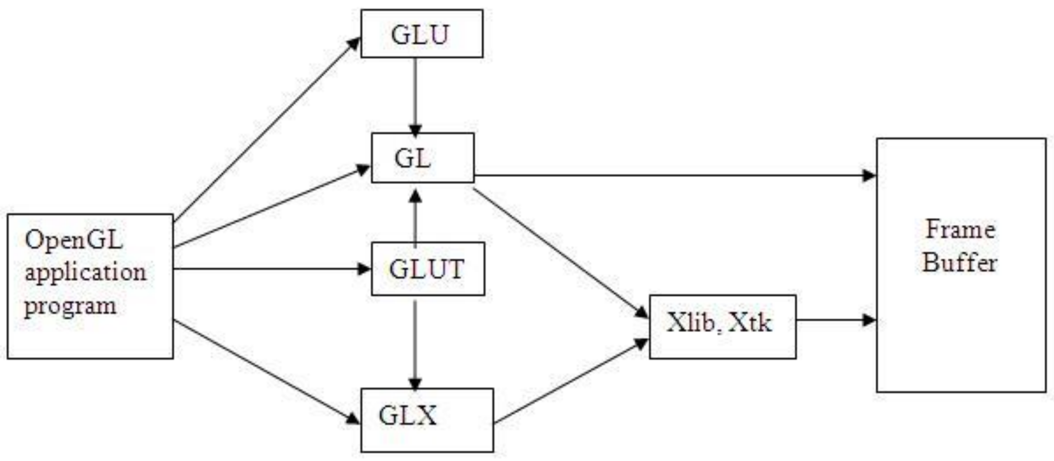
OpenGL is the premier environment for developing portable, interactive 2D and 3D graphics applications. Since its introduction in 1992, OpenGL has become the industry's most widely used and supported 2D and 3D graphics application programming interface (API), bringing thousands of applications to a wide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



CHAPTER 2

LITRETURE SURVEY

The basic functions like glColor3f(…); glRotatef(..),glTranslatef(..) etc that are most commonly used in the code are taken from the prescribed VTU Text book "INTERACTIVE COMPUTER GRAPHICS" 5th edition by Edward Angel.[1].

The lab programs in the syllabus also serve as a basic template for creating a project. The usage of colors and specifications are taken from the various programs that were taught in the lab.[1].

The VTU prescribed text book serves as a huge database of functions and they are used in the project.

The other reference books are:

F.S.Hill Jr: "COMPUTER GRAPHICS USING OPENGL , 2nd EDITION".

James D.Foley , John F.Hughes : "COMPUTER GRAPHICS ,1997."

Donald Hearn and Pauline baker : "COMPUTER GRAPHICS" C-Version,2nd edition, Pearson Education , 2003.

CHAPTER 3
REQUIREMENTS AND SPECIFICATIONS

3.1 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 give input in the game. A **keyboard** is used for controlling and inputting data in the form of characters, numbers i.e. to change the user views . 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.

3.2 Software Requirements

The editor has been implemented on the OpenGL platform and mainly requires an appropriate version of eclipse compiler to be installed and functional in ubuntu. Though it is implemented in OpenGL, it is very 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 10.10

Language Used In Coding

C-language

Tool Used In Coding

Eclipse

CHAPTER 4

SOFTWARE DESIGN:

4.1 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. Even though these properties were supported, it was difficult to render 2D graphics cannot be . Very difficult to get a 3Dimensional object. Even the effects like lighting , shading cannot be provided. So we go for Eclipse software.

PROPOSED SYSTEM :

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

1. OpenGL is designed as a streamlined.
2. It is 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. Its provides double buffering which is vital in providing transformations.
5. It is event driven software.
6. It provides call back function.

4.2 <u>DETAILED DESIGN</u>

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 their 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*cosθ − *y*sinθ and *y*' = *x*sinθ + *y*cosθ. 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*cosθ + *y*sinθ and *y*' = − *x*sinθ + *y*cosθ 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.

CHAPTER 5
IMPLEMENTATION

5.1 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 glMatrixMode(GLenum *mode*);

where `mode` Specifies which matrix stack is the target for subsequent matrix operations. Three values are accepted: **GL_MODELVIEW**, **GL_PROJECTION**, and **GL_TEXTURE**. The initial value is **GL_MODELVIEW**.

glMatrixMode sets the current matrix mode. `mode` can assume one of three values:

GL_MODELVIEW	Applies subsequent matrix operations to the modelview matrix stack.
GL_PROJECTION	Applies subsequent matrix operations to the projection matrix stack.

void glViewport(GLint *x*, GLint *y*, GLsizei *width*, GLsizei *height*)

where *x*, *y* Specify the lower left corner of the viewport rectangle, in pixels. The initial value is (0, 0).

width, *height* Specify the width and height of the viewport. When a GL context is first attached to a surface (e.g. window), *width* and *height* are set to the dimensions of that surface. glViewport specifies the affine transformation of *x* and *y* from normalized device coordinates to window coordinates. Let (x_{nd}, y_{nd}) be normalized device coordinates. Then the window coordinates (x_w, y_w) are computed as follows:

$$x_w = (x_{nd} + 1)\ {}^{width}/_2 + x$$

$$y_w = (y_{nd} + 1)\ {}^{height}/_2 + y$$

Viewport width and height are silently clamped to a range that depends on the implementation. To query this range, call **glGetInteger** with argument `GL_MAX_VIEWPORT_DIMS`.

void glutInit(int *argcp, 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 are window system dependent.

void glutReshapeFunc(void (*func)(int width, int height));

`glutReshapeFunc` sets the reshape callback for the *current window*. The reshape callback is triggered when a window is reshaped. A reshape callback is also triggered immediately before a window's first display callback after a window is created or whenever an overlay for the window is established. The `width` and`height` parameters of the callback specify the new window size in pixels. Before the callback, the *current window* is set to the window that has been reshaped.

If a reshape callback is not registered for a window or `NULL` is passed to `glutReshapeFunc` (to deregister a previously registered callback), the default reshape callback is used. This default callback will simply

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

glutPostRedisplay(): glutPostRedisplay, glutPostWindowRedisplay — marks the current or specified window as needing to be redisplayed.

5.2 **ALGORITHM**

```
int minimax( char _board[9] )
 {
  short int i;
  int bestValue = +INFINITY, index = 0;
  char bestMoves[9] = {0};
  for( i = 0; i < 9; i++ )
  {
      if( _board[i] == empty )
      {
          _board[i] = o;
          int value = maxSearch( _board );
          if( value < bestValue )
          {
              bestValue = value;
              index = 0;
              bestMoves[index] = i;
          }
          else if( value == bestValue )
              bestMoves[index++] = i;
          _board[i] = empty;
      }
  }
  if( index > 0 )
      index = rand() % index;
  return bestMoves[index];
}
int minSearch( char _board[9] )
{
  short int i;
  int positionValue = gameState(_board);
  if( positionValue == DRAW ) return 0;
```

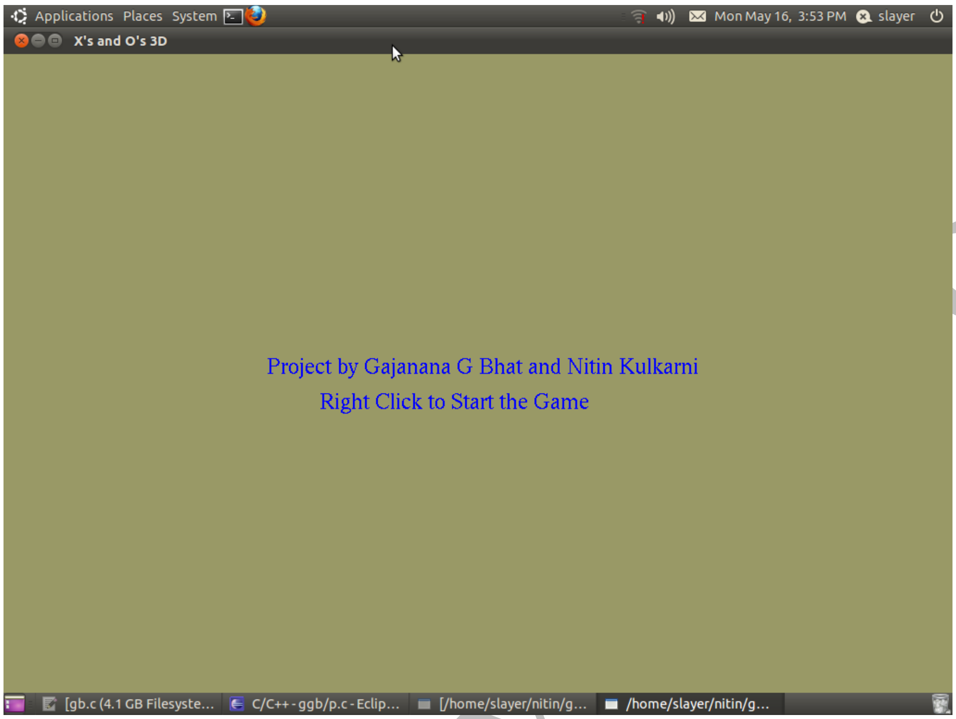
```
    if( positionValue != INPROGRESS ) return positionValue;
    int bestValue = +INFINITY;
    for( i = 0; i < 9; i++ )
    {
      if( _board[i] == empty )
        {
            _board[i] = o;
            int value = maxSearch( _board );
            if( value < bestValue )
                bestValue = value;
            _board[i] = empty;
        }
    }
    return bestValue;
}
int maxSearch( char _board[9] )
{
    short int i;
    int positionValue = gameState(_board);
    if( positionValue == DRAW ) return 0;
    if( positionValue != INPROGRESS ) return positionValue;
    int bestValue = -INFINITY;
    for( i = 0; i < 9; i++ )
    {
        if( _board[i] == empty )
        {
            _board[i] = x;
            int value = minSearch( _board );
            if( value > bestValue )
                bestValue = value;
            _board[i] = empty;
        }
    }
    return bestValue;
```

}

CHAPTER 6

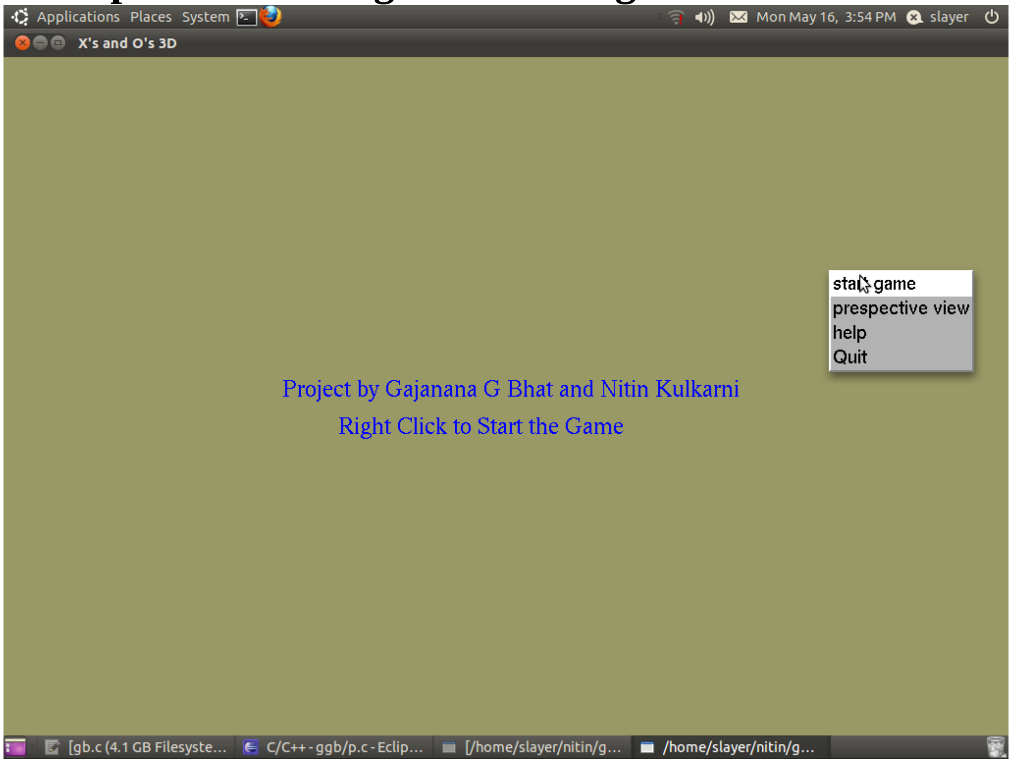
SNAP SHOTS:

1:Snap Shot Showing Instructions before Starting.


Applications Places System
Mon May 16, 3:53 PM slayer
X's and O's 3D
[gb.c (4.1 GB Filesyste...
C/C++ - ggb/p.c - Eclip...
[/home/slayer/nitin/g...
/home/slayer/nitin/g...

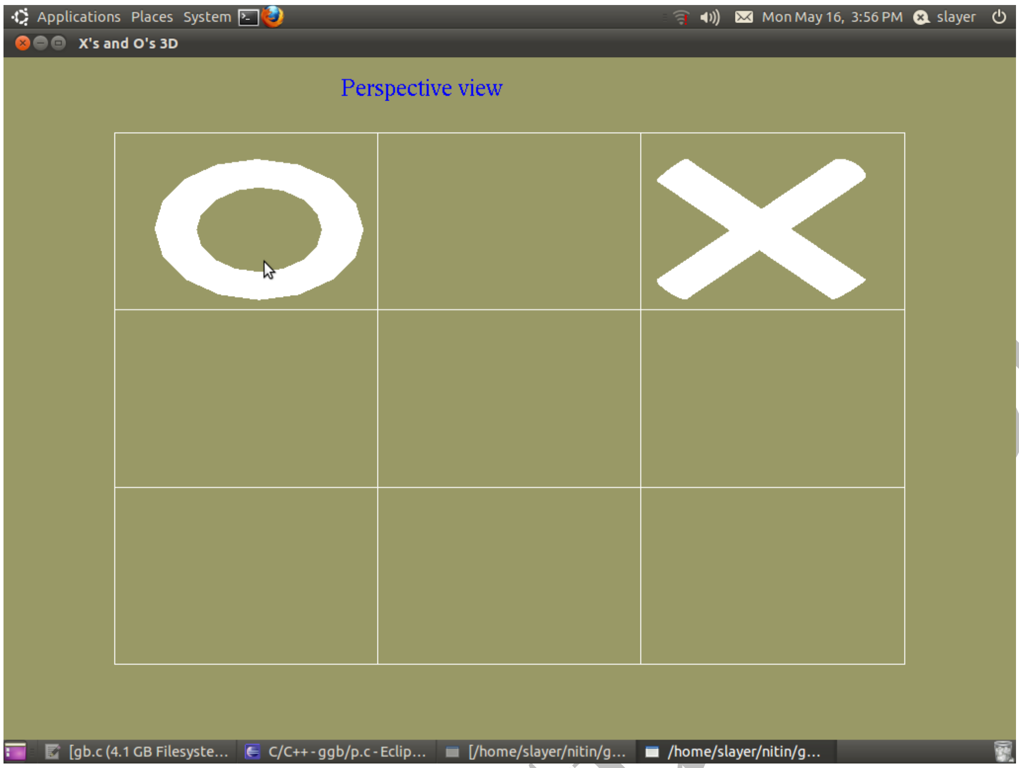
Project by Gajanana G Bhat and Nitin Kulkarni
Right Click to Start the Game

2. Snap Shot Showing to start the game:

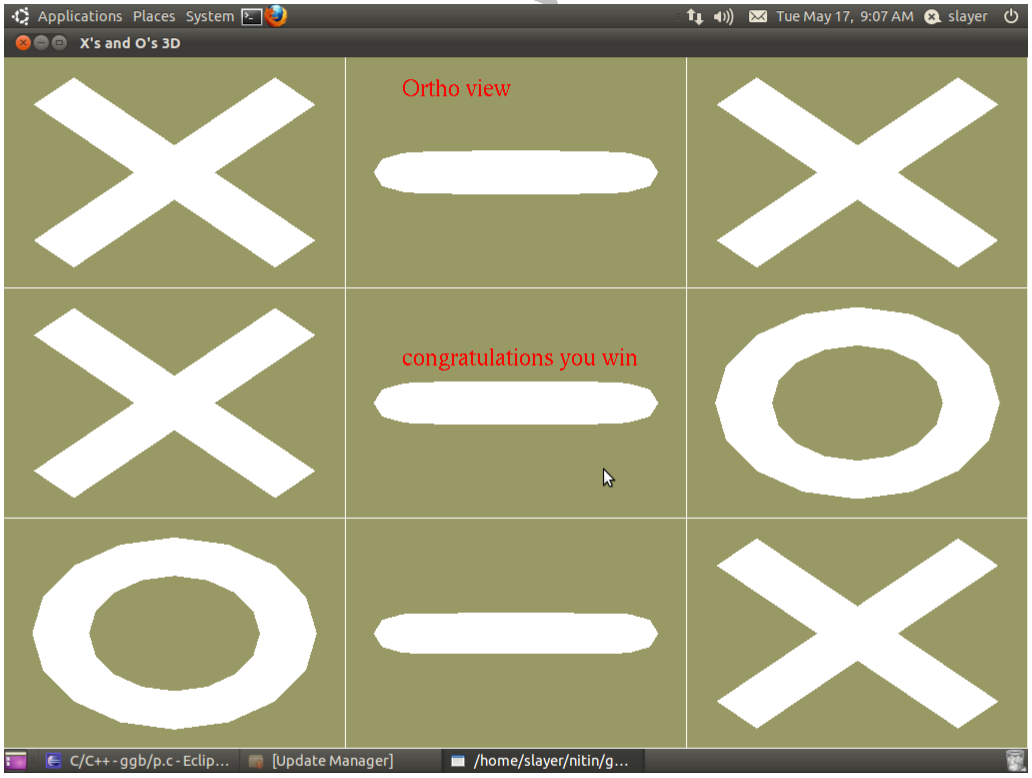

Applications Places System
Mon May 16, 3:54 PM slayer
X's and O's 3D
start game
prespective view
help
Quit
[gb.c (4.1 GB Filesyste...
C/C++ - ggb/p.c - Eclip...
[/home/slayer/nitin/g...
/home/slayer/nitin/g...

Project by Gajanana G Bhat and Nitin Kulkarni
Right Click to Start the Game

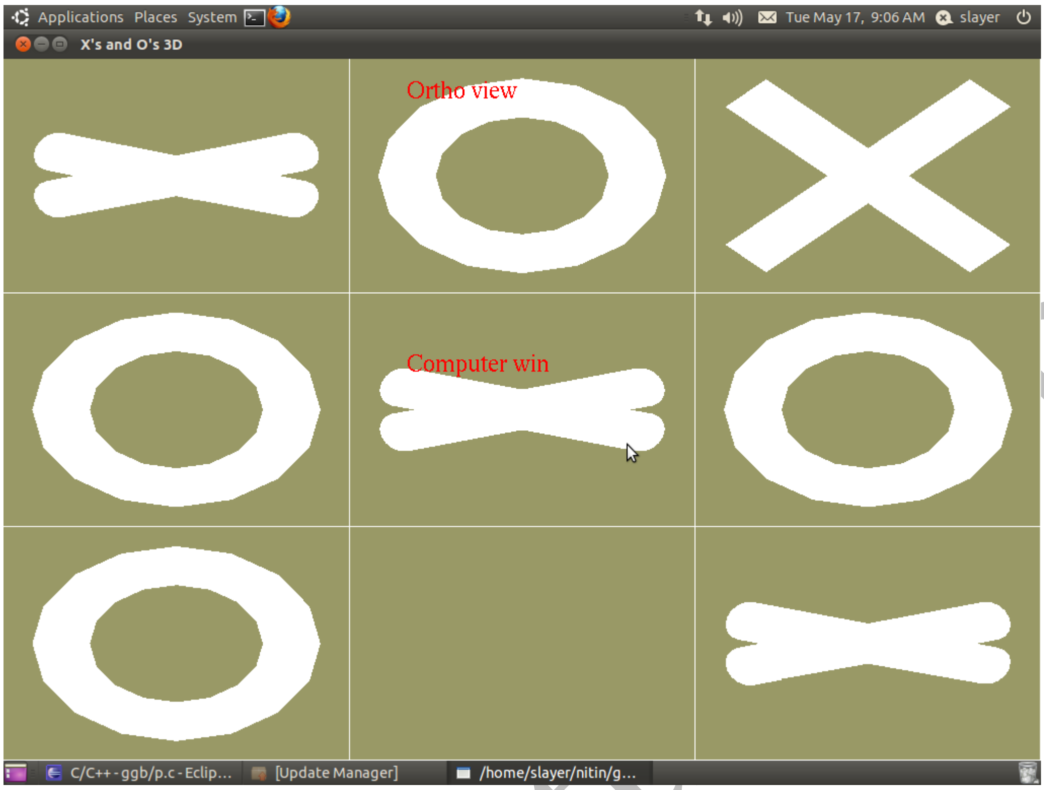
3. Snap Shot Showing 0's and X's as the input :


Applications Places System
Mon May 16, 3:56 PM slayer
X's and O's 3D
Perspective view
[gb.c (4.1 GB Filesyste...
C/C++ - ggb/p.c - Eclip...
[/home/slayer/nitin/g...
/home/slayer/nitin/g...

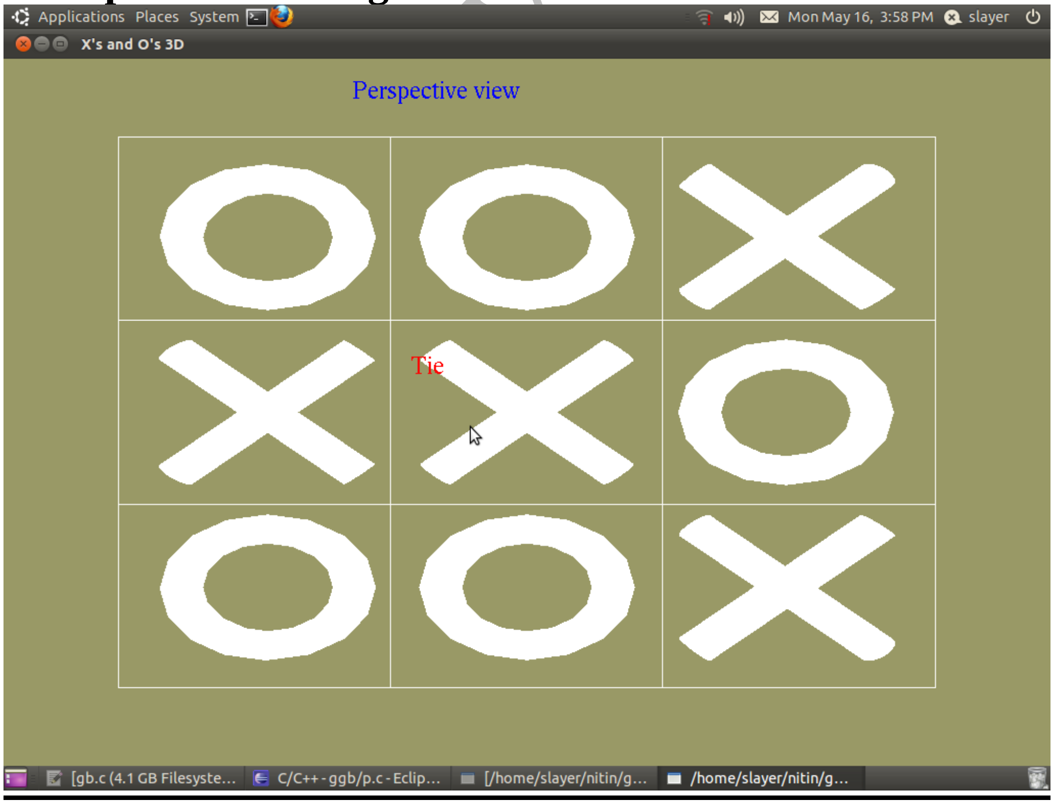
4. Snap Shot Showing if player wins:



5. Snap Shot Showing if computer wins:


Applications Places System
Tue May 17, 9:06 AM slayer
X's and O's 3D
Ortho view
Computer win
C/C++ - ggb/p.c - Eclip...
[Update Manager]
/home/slayer/nitin/g...

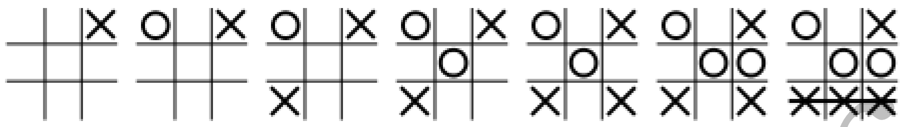
6.Snap Shot Showing a Tie:


Applications Places System
Mon May 16, 3:58 PM slayer
X's and O's 3D
Perspective view
Tie
[gb.c (4.1 GB Filesyste...
C/C++ - ggb/p.c - Eclip...
[/home/slayer/nitin/g...
/home/slayer/nitin/g...

CHAPTER 7
CONCLUSION AND FUTURE SCOPE:

The full designing and creating of **Tic-Tac-Toe** has been executed under ubuntu operating system using Eclipse, this platform provides a and 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 simple game.

The following example game is won by the first player, X:



A player can play perfect tic-tac-toe (win or draw) given they move according to the highest possible moves:

1. **Win**: If the player has two in a row, play the third to get three in a row.
2. **Block**: If the opponent has two in a row, play the third to block them.
3. **Fork**: Create an opportunity where you can win in two ways.
4. **Block opponent's fork**:
   - **Option 1**: Create two in a row to force the opponent into defending, as long as it doesn't result in them creating a fork or winning. For example, if "X" has a corner, "O" has the center, and "X" has the opposite corner as well, "O" must not play a corner in order to win. (Playing a corner in this scenario creates a fork for "X" to win.)
   - **Option 2**: If there is a configuration where the opponent can fork, block that fork.
5. **Center**: Play the center.
6. **Opposite corner**: If the opponent is in the corner, play the opposite corner.
7. **Empty corner**: Play in a corner square.
8. **Empty side**: Play in a middle square on any of the 4 sides.

APPENDIX:

```
// Tic Tac Toe or X's and O's.
// Keyboard input
// 'v' = view ortho/perspective
// 'l' = lighting on/of


#include <GL/glut.h>  // glut (gl utility toolkit) basic windows functions,
keyboard, mouse.
#include <stdio.h>    // standard (I/O library)
#include <stdlib.h>   // standard library (set of standard C functions
#include <math.h>     // Math library (Higher math functions )
#include<string.h>

// lighting
GLfloat LightAmbient[]= { 0.5f, 0.5f, 0.5f, 1.0f };
GLfloat LightDiffuse[]= { 0.5f, 0.5f, 0.5f, 1.0f };
GLfloat LightPosition[]= { 5.0f, 25.0f, 5.0f, 1.0f };
GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
int abc=0;

// mouse variables: Win = windows size, mouse = mouse position
int mouse_x, mouse_y, Win_x, Win_y, object_select;

// state variables for Orho/Perspective view, lighting on/off
static int view_state = 0, light_state = 0;

// Use to spin X's and O's
int spin, spinboxes;

// Win = 1 player wins, -1 computer wins, 2 tie.
// player or computer; 1 = X, -1 = O
// start_game indicates that game is in play.
int player, computer, win, start_game;

// alingment of boxes in which one can win
// We have 8 posiblities, 3 accross, 3 down and 2 diagnally
//
// 0 | 1 | 2
// 3 | 4 | 5
// 6 | 7 | 8
//
// row, colunm, diagnal information

static int box[8][3] = {{0, 1, 2}, {3, 4, 5}, {6, 7, 8}, {0, 3, 6},
                {1, 4, 7}, {2, 5, 8}, {0, 4, 8}, {2, 4, 6}};


// Storage for our game board
// 1 = X's, -1 = O's, 0 = open space

int box_map[9];
// center x,y location for each box
int object_map[9][2] = {{-6,6},{0,6},{6,6},{-6,0},{0,0},{6,0},{-6,-6},{0,-
6},{6,-6}};

// quadric pointer for build our X
```

```
GLUquadricObj *Cylinder;

// Begin game routine
void init_game(void)
{
int i;

// Clear map for new game
for( i = 0; i < 9; i++)
      {
      box_map[i] = 0;
    }

// Set 0 for no winner
win = 0;
start_game = 1;
}

// Check for three in a row/colunm/diagnal
// returns 1 if there is a winner
int check_move( void )
{

int i, t = 0;

//Check for three in a row
for( i = 0; i < 8; i++)
      {
       t = box_map[box[i][0]] + box_map[box[i][1]] + box_map[box[i][2]];
       if ( (t == 3) || ( t == -3) )
         {
          spinboxes = i;
          return( 1 );
         }
    }
t = 0;


// check for tie
for( i = 0; i < 8; i++)
      {
       t = t + abs(box_map[box[i][0]]) + abs( box_map[box[i][1]]) + abs(
box_map[box[i][2]]);
    }

if ( t == 24 ) return( 2 );
return( 0 );
}

// Do we need to block other player?
int blocking_win(void)
{
int i, t;
for( i = 0; i < 8; i++)
      {
       t = box_map[box[i][0]] + box_map[box[i][1]] + box_map[box[i][2]];
       if ( (t == 2) || ( t == -2) )
         {
         // Find empty
         if (box_map[box[i][0]] == 0) box_map[box[i][0]] = computer;
         if (box_map[box[i][1]] == 0) box_map[box[i][1]] = computer;
         if (box_map[box[i][2]] == 0) box_map[box[i][2]] = computer;
              return( 1 );
       }
    }
            return( 0 );
}


// check for a free space in corner
int check_corner(void)
{
int i;

if ( box_map[0] == 0)
      {
      box_map[0] = computer;
      i = 1;
      return( 1 );
    }

if ( box_map[2] == 0)
      {
      box_map[2] = computer;
      i = 1;
      return( 1 );
    }

if ( box_map[6] == 0)
      {
      box_map[6] = computer;
      i = 1;
      return( 1 );
    }

if ( box_map[8] == 0)
      {
      box_map[8] = computer;
      i = 1;
      return( 1 );
    }
           return( 0 );
}

// Check for free space in row
int check_row(void)
{

if ( box_map[4] == 0)
      {
      box_map[4] = computer;
      return( 1 );
    }

if ( box_map[1] == 0)
      {
      box_map[1] = computer;
      return( 1 );
    }

if ( box_map[3] == 0)
      {
```

```
      box_map[3] = computer;
      return( 1 );
    }
if ( box_map[5] == 0)
      {
      box_map[5] = computer;
      return( 1 );
    }
if ( box_map[7] == 0)
      {
      box_map[7] = computer;
      return( 1 );
    }

            return( 0 );
}

// logic for computer's turn
int computer_move()
{
if ( blocking_win() == 1) return( 1 );
if ( check_corner() == 1) return( 1 );
if ( check_row() == 1) return( 1);

    return( 0 );
}


// I use this to put text on the screen
void Sprint( int x, int y, char *st)
{
      int l,i;

      l=strlen( st ); // see how many characters are in text string.
      glRasterPos2i( x, y); // location to start printing text
      for( i=0; i < l; i++)  // loop until i is greater then l
            {
            glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, st[i]); //
Print a character on the screen
      }
}

// This creates the spinning of the cube.
static void TimeEvent(int te)
{

    spin++;  // increase cube rotation by 1
      if (spin > 360) spin = 180; // if over 360 degress, start back at
zero.
      glutPostRedisplay();  // Update screen with new rotation data
      glutTimerFunc( 8, TimeEvent, 1);  // Reset our timmer.
}


// Setup our Opengl world, called once at startup.
void init(void)
{
   glClearColor (0.6,0.6,0.4,0.0);  // When screen cleared, use black.
   glShadeModel (GL_SMOOTH);  // How the object color will be rendered
smooth or flat
   glEnable(GL_DEPTH_TEST);   // Check depth when rendering
```

```
    // Lighting is added to scene
    glLightfv(GL_LIGHT1 ,GL_AMBIENT, LightAmbient);
      glLightfv(GL_LIGHT1 ,GL_DIFFUSE, LightDiffuse);
      glLightfv(GL_LIGHT1 ,GL_POSITION, LightPosition);
      glEnable(GL_LIGHTING);  // Turn on lighting
      glEnable(GL_LIGHT1);    // Turn on light 1

    start_game = 0;
    win = 0;

    // Create a new quadric
    Cylinder = gluNewQuadric();
    gluQuadricDrawStyle( Cylinder, GLU_FILL );
    gluQuadricNormals( Cylinder, GLU_SMOOTH );
    gluQuadricOrientation( Cylinder, GLU_OUTSIDE );
}


void Draw_O(int x, int y, int z, int a)
{

glPushMatrix();
glTranslatef(x, y, z);
glRotatef(a, 1, 0, 0);
glutSolidTorus(0.5, 2.0, 8, 16);
glPopMatrix();
}

void Draw_X(int x, int y, int z, int a)
{

glPushMatrix();
glTranslatef(x, y, z);
glPushMatrix();
glRotatef(a, 1, 0, 0);
glRotatef(90, 0, 1, 0);
glRotatef(45, 1, 0, 0);
glTranslatef( 0, 0, -3);
gluCylinder( Cylinder, 0.5, 0.5, 6, 16, 16);
//glutSolidCone( 2.5, 3.0, 16, 8 );
glPopMatrix();
glPushMatrix();
glRotatef(a, 1, 0, 0);
glRotatef(90, 0, 1, 0);
glRotatef(315, 1, 0, 0);
glTranslatef( 0, 0, -3);
gluCylinder( Cylinder, 0.5, 0.5, 6, 16, 16);
//glutSolidCone( 2.5, 3.0, 16, 8 );
glPopMatrix();
glPopMatrix();

}

// Draw our world
void display(void)
{
      if(abc==3)
      {
             //int mk=0;
             // glColor3f(0.0,1.0,0.0);
                  glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

```
//Clear the screen
                     glColor3f(0.0,1.0,0.0);
                  glMatrixMode (GL_PROJECTION);  // Tell opengl that we are
doing project matrix work
                  glLoadIdentity();  // Clear the matrix
                  glOrtho(-9.0, 9.0, -9.0, 9.0, 0.0, 30.0);  // Setup an
Ortho view
                  glMatrixMode(GL_MODELVIEW);  // Tell opengl that we are
doing model matrix work. (drawing)
                  glLoadIdentity(); // Clear the model matrix

                  glDisable(GL_COLOR_MATERIAL);
                  glDisable(GL_LIGHTING);
                  glColor3f(0.0, 0.0, 1.0);


                   Sprint(-2, 0, "Project by");
                     Sprint(-2, -1, "Gajanan and Nitin");
                     Sprint(-3, -2, "To Start press right button");
                     Sprint(-3, -3, "right button for X's");
                     Sprint(-3, -4, "and left for O's");
            glutSwapBuffers();
      }
      else if(abc==0)
{

      glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);  //Clear the
screen

      glMatrixMode (GL_PROJECTION);  // Tell opengl that we are doing
project matrix work
      glLoadIdentity();  // Clear the matrix
      glOrtho(-9.0, 9.0, -9.0, 9.0, 0.0, 30.0);  // Setup an Ortho view
      glMatrixMode(GL_MODELVIEW);  // Tell opengl that we are doing model
matrix work. (drawing)
      glLoadIdentity(); // Clear the model matrix

      glDisable(GL_COLOR_MATERIAL);
      glDisable(GL_LIGHTING);
      glColor3f(0.0, 0.0, 1.0);
         Sprint(-4, 0, "Project by Gajanana G Bhat and Nitin Kulkarni");
         Sprint(-3, -1, "Right Click to Start the Game");
      glutSwapBuffers();

}
else
{
int ix, iy;
int i;
int j;

glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);  //Clear the screen

glMatrixMode (GL_PROJECTION);  // Tell opengl that we are doing project
matrix work
glLoadIdentity();  // Clear the matrix
glOrtho(-9.0, 9.0, -9.0, 9.0, 0.0, 30.0);  // Setup an Ortho view
glMatrixMode(GL_MODELVIEW);  // Tell opengl that we are doing model matrix
work. (drawing)
glLoadIdentity(); // Clear the model matrix
```

```
glDisable(GL_COLOR_MATERIAL);
glDisable(GL_LIGHTING);
glColor3f(1.0, 0.0, 0.0);

//printing final result of the game
if (win == 1) Sprint( -2, 1, "congratulations you win");
if (win == -1) Sprint( -2, 1, "Computer win");
if (win == 2) Sprint( -2, 1, "Tie");

// Setup view, and print view state on screen
if (view_state == 1)
      {
    glColor3f( 0.0, 0.0, 1.0);
    Sprint(-3, 8, "Perspective view");
    glMatrixMode (GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60, 1, 1, 30);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    }else
      {
      glColor3f( 1.0, 0.0, 0.0);
    Sprint(-2, 8, "Ortho view");
    }

// Lighting on/off
if (light_state == 1)
      {
      glDisable(GL_LIGHTING);
    glDisable(GL_COLOR_MATERIAL);
    }else
      {
    glEnable(GL_LIGHTING);
    glEnable(GL_COLOR_MATERIAL);
    }

gluLookAt( 0, 0, 20, 0, 0, 0, 0, 1, 0);

// Draw Grid
for( ix = 0; ix < 4; ix++)
      {
       glPushMatrix();
       glColor3f(1,1,1);
         glBegin(GL_LINES);
         glVertex2i(-9 , -9 + ix * 6);
         glVertex2i(9 , -9 + ix * 6 );
         glEnd();
         glPopMatrix();
      }
      for( iy = 0; iy < 4; iy++ )
         {
       glPushMatrix();
       glColor3f(1,1,1);
         glBegin(GL_LINES);
         glVertex2i(-9 + iy * 6, 9 );
         glVertex2i(-9 + iy * 6, -9 );
         glEnd();
         glPopMatrix();
         }

glColorMaterial(GL_FRONT, GL_AMBIENT);
```

```
glColor4f(0.5, 0.5, 0.5, 1.0);
glColorMaterial(GL_FRONT, GL_EMISSION);
glColor4f(0.0, 0.0, 0.0, 1.0 );
glColorMaterial(GL_FRONT, GL_SPECULAR);
glColor4f(0.35, 0.35, 0.35, 1.0);
glColorMaterial(GL_FRONT, GL_DIFFUSE);
glColor4f(0.69, 0.69, 0.69, 1.0);
//glDisable(GL_COLOR_MATERIAL);
glColor3f( 0.0, 0.0, 0.0);  // Cube color
//glEnable(GL_COLOR_MATERIAL);
// Draw object in box's

for( i = 0; i < 9; i++)
   {
   j = 0;
   if (abs( win ) == 1 )
      {
      if ( (i == box[spinboxes][0]) || (i == box[spinboxes][1]) || (i ==
box[spinboxes][2]))
         {
          j = spin;
          }else j = 0;
        }
   if(box_map[i] == 1) Draw_X( object_map[i][0], object_map[i][1], -1, j);

   if(box_map[i] == -1) Draw_O( object_map[i][0], object_map[i][1], -1, j);
   }

//glDisable(GL_COLOR_MATERIAL);

glutSwapBuffers();
}
}

// This is called when the window has been resized.
void reshape (int w, int h)
{
   Win_x = w;
   Win_y = h;
   glViewport (0, 0, (GLsizei) w, (GLsizei) h);
   glMatrixMode (GL_PROJECTION);
   glLoadIdentity ();
}

// Read the keyboard
void keyboard (unsigned char key, int x, int y)
{
   switch (key)
   {

      case 'v':
        case 'V':
              view_state = abs(view_state -1);
              break;
        case 'b':
        case 'B':
              light_state = abs(light_state -1);
              break;
        case 27:
         exit(0); // exit program when [ESC] key presseed
         break;
      default:
         break;
   }
}

void mouse(int button, int state, int x, int y)
{
// We convert windows mouse coords to out openGL coords
mouse_x =  (18 * (float) ((float)x/(float)Win_x))/6;
mouse_y =  (18 * (float) ((float)y/(float)Win_y))/6;

// What square have they clicked in?
object_select = mouse_x + mouse_y * 3;

if ( start_game == 0)
      {
    if ((button == GLUT_RIGHT_BUTTON) && (state == GLUT_DOWN))
      {
       player = 1;
         computer = -1;
         init_game();
         computer_move();
         return;
      }

    if ((button == GLUT_LEFT_BUTTON) && (state == GLUT_DOWN))
      {
        player = -1;
      computer = 1;
        init_game();

        return;
      }
   }

if ( start_game == 1)
      {
    if ((button == GLUT_LEFT_BUTTON) && (state == GLUT_DOWN))
      {
      if (win == 0)
         {
         if (box_map[ object_select ] == 0)
            {
           box_map[ object_select ] = player;
             win = check_move();
             if (win == 1)
                {
                start_game = 0;
              return;
              }
             computer_move();
               win = check_move();
               if (win == 1)
                  {
                 win = -1;
                 start_game = 0;
                 }
             }
           }
        }
}
```

```
if ( win == 2 )start_game = 0;

}

void menu(int choice)
{

            switch(choice)
            {
            case 1: abc=1;
                    glutMouseFunc(mouse);
                    break;

            case 2:
                view_state = abs(view_state -1);
                break;
            case 3: abc=3;
              glutMouseFunc(mouse);
            break;
            case 4:
                    exit(0);
                    break;


            }
}


// Main program
int main(int argc, char** argv)
{
   glutInit(&argc, argv);
   glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);

   glutInitWindowSize (850,600);
   glutInitWindowPosition (10, 10);
   glutCreateWindow (argv[0]);
   glutSetWindowTitle("X's and O's 3D");
   init ();
   glutCreateMenu(menu);
   glutAddMenuEntry("start game",1);
   glutAddMenuEntry("perspective view",2);
   glutAddMenuEntry("help",3);
   glutAddMenuEntry("Quit",4);
   glutAttachMenu(GLUT_RIGHT_BUTTON);
   glutDisplayFunc(display);
   glutReshapeFunc(reshape);
   glutKeyboardFunc(keyboard);
   //glutMouseFunc(mouse);
   glutTimerFunc( 50, TimeEvent, 1);
   glutMainLoop();
   return 0;

}
```

BIBLIOGRAPHY

1. Reference from Interactive Computer Graphics by EDWARD ANGEL

2. Internet source

   - ✓ www.angelfire.com
   - ✓ Wikipedia.org.
   - ✓ Google.
   - ✓ Opengl.org.