VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum: 590018

A Mini Project Report (17CSL68)
on

Blockchain Visualization

A mini project report submitted in partial fulfilment of the requirement for the award of the degree of

Bachelor of Engineering
in
Computer Science & Engineering

Submitted by
Pavan R Bhat (1AY17CS061)

Under the guidance of
Prof. Naganandini G
Department of Computer Science & Engineering

ACHARYA

Acharya Institute of Technology
Department of Computer Science & Engineering
Soladevanahalli, Bangalore-560107

ACKNOWLEDGEMENT

We express our gratitude to our institution and management for providing us with good infrastructure, laboratory facilities and inspiring staff, and whose gratitude was of immense help in completion of this mini project successfully.

We express our sincere gratitude to our principal, **Dr. M R Prakash** for providing us the required environment and for his valuable suggestions.

Our sincere thanks to **Dr. Prashanth C M,** Head of the Department. Computer Science and Engineering, Acharya Institute of Technology for his valuable support and for rendering us the resources for this mini project.

We heartily thank **Prof. Varalakshmi B D,** Associate Professor**, Prof. Vani K S, Prof. Naganandini N, Prof. Avinash K** Assistant Professors, Department of Computer Science and Engineering, Acharya Institute of Technology who guided us with valuable suggestions in completing this mini project at every stage.

Our gratitude thanks should be rendered to many people who helped us in all possible ways.

PAVAN R BHAT

1AY17CS061

ACHARYA INSTITUTE OF TECHNOLOGY

(Affiliated to Visvesvaraya Technological University, Belgaum)
Soladevanahalli, Bangalore – 560090

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Certificate

Certified that the Computer Graphics Mini Project entitled **"Blockchain Visualization"** is a bonafide work carried out by **Mr. Pavan R Bhat (1AY17CS061)** in partial fulfillment for the award of degree of **Bachelor of Engineering in Computer Science & Engineering of the Visvesvaraya Technological University**, Belgaum during the academic year **2019-2020.** It is certified that all corrections/ suggestions indicated for internal assessments have been incorporated in the Report deposited in the departmental library. The Mini project report has been approved as it satisfies the academic requirements in respect of Mini Project work prescribed for the **Bachelor of Engineering Degree**.

Signature of Guides **Signature of H.O.D**

Name of the examiners **Signature with date**

1.

2.

ABSTRACT

Many people think "**Block chain Technology"** may replace everything in the future. It is very important to know about the Block chains. The main aim of this project is to show the demonstration of working of Blockchain. This project demonstrates the basics of the Blockchain and it's working, the project completely uses graphics and has been done using OpenGL. The code used in the program is very simple. We can easily understand the working of Blockchain by seeing the output of this project.

Minimum code is written to show the animation so that the programmers can easily understand the source code as well. In the future I will distribute the code as open source code and anyone can modify the project easily. I have ensured the minimum use of functions and maximum code reuse. I Have used some inbuild OpenGL functions to draw some primitives. There are some primitives which are drawn only using OpenGL user defined functions. 3D and 2D concepts are used based on requirement of the demonstration. Code written is dynamic in nature and behaves according to the user inputs and behaviour. Output window of the OpenGL is completely interactive. Follow the instruction given in the screen to run the project according to user's comfort. Using the techniques learned, I have tried to make the most efficient code.

List of Figures

CONTENTS

Chapter 1

INTRODUCTION TO OPENGL

Computer graphics are graphics created using computers and, more generally, the representation and manipulation of pictorial data by a computer. The development of computer graphics has made computers easier to interact with and better for understanding and interpreting many types of data. Developments in computer graphics have had a profound impact on many types of media and have revolutionized the animation and video game industry. Today computers and computer-generated images touch many aspects of our daily life. Computer imagery is found on television, in newspapers, in weather reports, and during surgical procedures. A well-constructed graph can present complex statistics in a form that is easier to understand and interpret. Such graphs are used to illustrate papers, reports, theses, and other presentation material. A range of tools and facilities are available to enable users to visualize their data, and computer graphics are used in many disciplines. We implement computer graphics using the OpenGL API. OpenGL (Open Graphics Library) is a standard specification defining a cross-language, cross-platform API for writing applications that produce 2D and 3D computer graphics.

OpenGL is a low-level graphics library specification. It makes available to the programmer a small set of geometric primitives - points, lines, polygons, images, and bitmaps. OpenGL provides a set of commands that allow the specification of geometric objects in two or three dimensions, using the provided primitives, together with commands that control how these objects are rendered (drawn). Since OpenGL drawing commands are limited to those that generate simple geometric primitives (points, lines, and polygons), the OpenGL Utility Toolkit (GLUT) has been created to aid in the development of more complicated three-dimensional objects such as a sphere, a torus, and even a teapot. GLUT may not be satisfactory for full-featured OpenGL applications, but it is a useful starting point for learning OpenGL.

GLUT is designed to fill the need for a window system independent programming interface for OpenGL programs. The interface is designed to be simple yet still meet the needs of useful OpenGL programs. Removing window system operations from OpenGL is a sound decision because it allows the OpenGL graphics system to be retargeted to various systems including powerful but expensive graphics workstations as well as mass-production graphics systems like video games, set-top boxes for interactive television, and PCs.

The GLUT application-programming interface (API) requires very few routines to display a graphics scene rendered using OpenGL. The GLUT routines also take relatively few parameters.

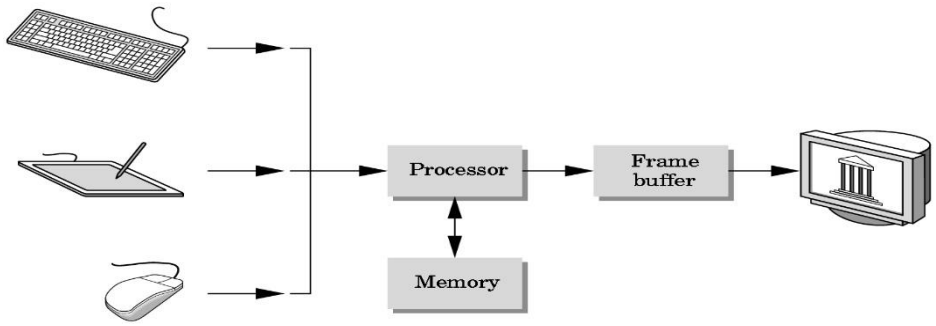
1.1 COMPUTER GRAPHICS

The term computer graphics includes almost everything on computers that is not text or sound. Today nearly all computers use some graphics and users expect to control their computer through icons and pictures rather than just by typing. The term Computer Graphics has several meanings:

- The representation and manipulation of pictorial data by a computer
- The various technologies used to create and manipulate such pictorial data
- The images so produced, and
- The sub-field of computer science which studies methods for digitally synthesizing and manipulating visual content

Today computers and computer-generated images touch many aspects of our daily life. Computer imagery is found on television, in newspapers, in weather reports, and during surgical procedures. A well-constructed graph can present complex statistics in a form that is easier to understand and interpret.

Fig 1.1.1 A Graphics System.



1.2 OPENGL TECHNOLOGY

OpenGL is strictly defined as “a software interface to graphics hardware.” In essence, it is a 3D graphics and modelling library that is highly portable and very fast. Using OpenGL, you can create elegant and beautiful 3D graphics with exceptional visual quality. The greatest advantage to using OpenGL is that it is orders of magnitude faster than a ray tracer or software-rendering engine. Initially, it used algorithms carefully developed and optimized by Silicon Graphics, Inc. (SGI), an acknowledged world leader in computer graphics and animation. Over time, OpenGL has evolved as other vendors have contributed their expertise and intellectual property to develop high-performance implementations of their own.

OpenGL is a software interface to graphics hardware. This interface consists of about 120 distinct commands, which you use to specify the objects and operations needed to produce interactive three-dimensional applications. OpenGL is designed to work efficiently even if the computer that displays the graphics you create isn't the computer that runs your graphics program. The format for transmitting OpenGL commands (called the protocol) from the client to the server is always the same, so OpenGL programs can work across a network even if the client and server are different kinds of computers. If an OpenGL program isn't running across a network, then there's only one computer, and it is both the client and the server. OpenGL is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms.

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. With OpenGL, you must build up your desired model from a small set of geometric primitives - points, lines, and polygons. Computer graphics deals with all aspects of creating images with a computer:

Application: The object is an artist's rendition of the sun for an animation to be shown in a domed environment (planetarium)

Software: Maya for modelling and rendering but Maya is built on top of OpenGL

Hardware: PC with graphics card for modelling and rendering.

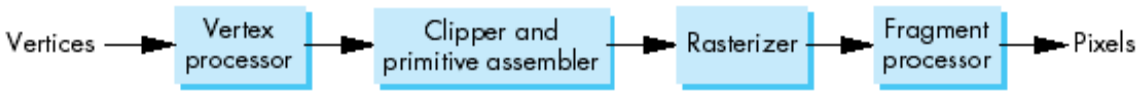


Fig 1.2.1 Graphics pipeline.

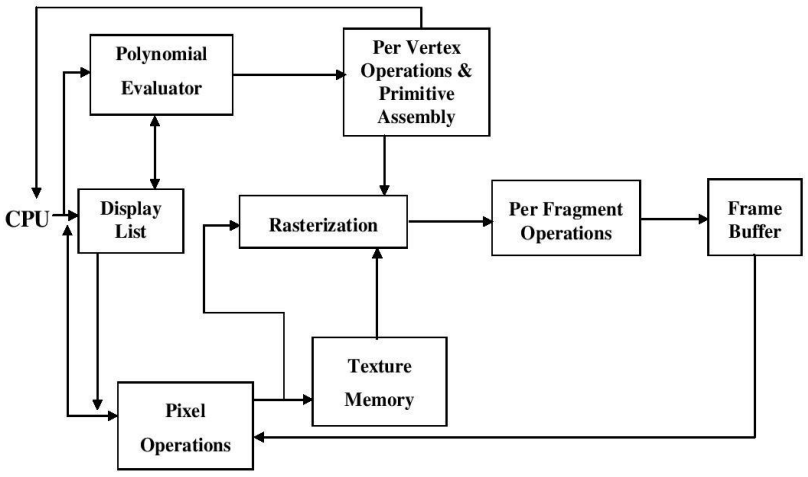


Figure 1.2.2 OpenGL Architecture

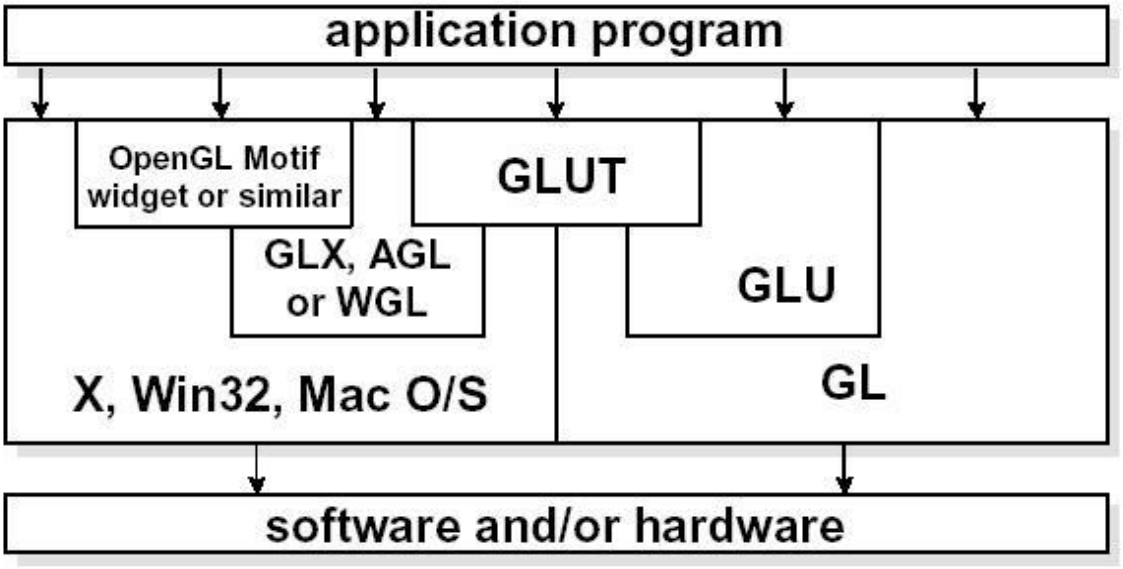


Figure 1.2.3 OpenGL and Related APIs

The above diagram illustrates the relationships of the various libraries and window system components.

Generally, applications which require more user interface support will use a library designed to support those types of features (i.e. buttons, menu and scroll bars, etc.) such as Motif or the Win32 API.

Chapter 2

PROJECT DESCRIPTION

The objective of this project is to demonstrate the working of the **"Blockchain Technology"** through animation. The window on your system will show series of demos and graphics illustrating how the **Blockchain** works. The ideas in this technology such as Hashing, Chaining, Mining, Proof of Work, Consensus etc. The whole project is done using OpenGL.

The output animation of the project gives user a feeling as if he is watching an animated movie. It is important to know about the Blockchain as many people say **Blockchain is the Next Big Thing.** It will replace everything in the future. It includes all the online transactions we make, Military Secrets, Social Media, Decentralized Applications and abundantly many more areas of application. By watching the animation, user can know the working of Blockchain with transactions.

A lot of people think Blockchain is too complex to understand and implement. The goal of this project is to show that Blockchain is not so complex as we thought. In this project you will see the working of Blockchain in a most simple and efficient way as possible. This project illustrates some of the very basic concepts in the technology which is very essential to understand Blockchain as a beginner.

Using OpenGL, free glut, OpenSSL libraries Visualization of the Blockchain is shown in the project.

To make the problem statements better understandable using OpenGL graphics. The main objective of the project is to simplify the understanding of concepts pertaining to the applications of computer science & engineering.

Chapter 3

DESIGN CONSTRAINTS

A major hurdle during the design of the project was displaying the movement of the elements. Since OpenGL contains a large number of API's and built in functions, we were able to clear this hurdle. The hardware and software constraints are:

3.1 HARDWARE CONSTRAINTS

There are no rigorous restrictions on machine configuration. The editor should be capable of working on all machines capable of supporting recent versions of Microsoft Windows.

- Processor : Pentium PC or any higher versions
- RAM : 1 GB or above
- Hard Disk : 20 GB (approx.)
- Display : VGA Colour Monitor

3.2 SOFTWARE CONSTRAINTS

- Operating System: Windows XP/Vista/7/8/10 Linux or Mac
- Language : C/C++
- Compiler : Microsoft Visual Studio 2010
- OpenGL packages
- OpenSSL packages

Chapter 4

ARCHITECTURE

The main, the display, the timer, reshape, keyboard and the menu functions are all defined in CGV_PROJECT_1.0.cpp. The flow of control is shown below.

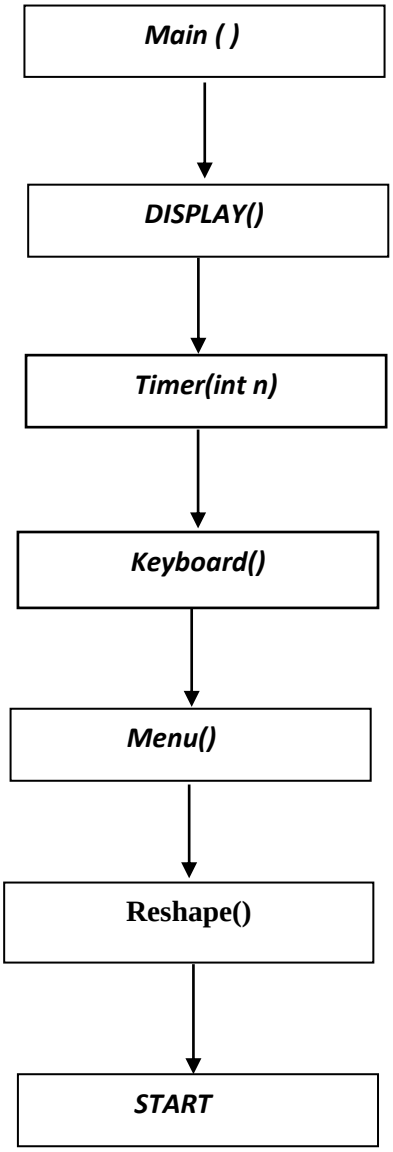


Figure 4.1.1 OpenGL functions and call-back functions flow diagram

4.1 Flow of working

1. Compile the program with g++ -lGL -lGLU -lglut -lcrypto CGV_PROJECT_1.0.cpp
2. Run the program with ./a.out
3. You will see the initial screen with animation.
4. Options which you could perform are shown in the screen as well.
5. Right click for Menu
6. ‘c’ for continue, ‘b’ to go back and other keys which you could use are shown in the screen itself.
7. At any point of time you could use Menu option to go to any window you desire
8. You can select exit from menu or press ‘esc’ for exiting the program.

Chapter 5

CODE IMPLEMENTATION

5.1 Libraries and headers

- **OpenGL**

  GL/glut.h , freeglut.h header files are mostly used for implementation of all primitives drawing and animation. Used functions and primitives will be explained in the next section.

- **OpenSSL**

  openssl.h is used for cryptography and its functions. Blockchain technology uses SHA-256 algorithm for Encryption of the data. This is done through Openssl library. Openssl could be used for many of the encryption techniques. This project uses the SHA-256 encryption.

- **Vectors**

  Vectors are used to store keystrokes in buffer. User could just type in his inputs as messages.

- **Time**

  time.h is used for putting delays and return time of OS.

- **Math**

  math.h is used for all the Mathematics related functions such as Sin Cos Pi etc

5.2 Functions

OpenGL Program Utilities

glutInit()

The first thing we need to do is call the “glutInit()” procedure. It should be called before any other GLUT routine because it initializes the GLUT library. The parameters to “glutInit()” should be the same as those to the main(), specifically “main(int argc, char** argv)” and “glutInit(&argc, argv)”.

glutInitDisplayMode()

The next thing we need to do is call the “glutInitDisplayMode()” procedure to specify the display mode for a window. We must first decide whether we want to use the RGBA (GLUT_RGBA) or color-index (GLUT-INDEX) color model. The RGBA mode stores its color buffers as red, green, blue and an alpha color components. The fourth color component, alpha, corresponds to the notion of opacity. An alpha value of 0.0 gives complete transparency. Color-index mode, in contrast, stores color buffers in 0.0 indices.

Another decision we need to make when setting up the display mode is whether we want to use single buffer (GLUT_SINGLE) or double buffer(GLUT_DOUBLE).Applications that use both front and back colour buffers use double-buffer.

glutInitWindowSize() and glutInitWindowPosition()

We need to create the characteristics of our window. A call to “glutInitWindowSize()”, will be used to specify the size, in pixels, of our initial window. The arguments indicate the height and width (in pixels) of the desired window. Similarly, “glutInitWindowPosition()” is used to specify the screen location for the upper-left corner of our window. The arguments, x and y, indicate the location of the window relative to the entire display.

glutCreateWindow()

To actually create a window, with the previously set characteristics (display mode, size, location, etc.), the programmer uses the “glutCreateWindow( )” function. It takes a string as a parameter, which appears in the tittle bar of the window that we desire. The window is not actually displayed until the “glutMainLoop( )” is called.

glutDisplayFunc()

The glutDisplayFunc() procedure is the first and most important event callback function. A callback function is one where a programmer-specified routine can be registered to be called in response to a specific type of event.

glutMainLoop()

For the code to run, we must call “glutMainLoop()”. All windows that have been created can now be shown, and rendering the windows is now effective. The program will now be able to handle events as they occur (mouse clicks, windows resizing, etc.). In addition, the registered display callback (from our “glutDisplayFunc()”) is triggered. Once this loop is entered, it never exits!

glClearColor( ) and glClear()

Drawing on a computer screen is different from drawing on paper, in the way that the paper starts out white, and all that we have to do is draw the picture. On the computer, the memory holding the picture is usually filled with the last picture we draw, so we typically need to clear it to some background color before we start drawing the new scene. To change the background color, we call and specify the color we select for the background. The default clearing color is (0,0,0, 0) which is black. A subsequent call to “glClear()” will clear the specified buffers to their current clearing values. To clear the color buffer,we use the argument “GL_COLOR_BUFFER_BIT”.

glutReshapeFunc()

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.

glutTimerFunc()

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 value parameter to glutTimerFunc. Multiple timer callbacks at same or differing times may be registered simultaneously.

The number of milliseconds is a lower bound on the time before the callback is generated. GLUT attempts to deliver the timer callback as soon as possible after the expiration of the callback's time interval.

There is no support for canceling a registered callback. Instead, ignore a callback based on its value parameter when it is triggered.

glutAddMenuEntry(char *name, int value);

glutAddMenuEntry adds a menu entry to the bottom of the *current menu*. The string name will be displayed for the newly added menu entry. If the menu entry is selected by the user, the menu's callback will be called passing value as the callback's parameter.

glutAddSubMenu(char *name, int menu);

glutAddSubMenu adds a sub-menu trigger to the bottom of the *current menu*. The string name will be displayed for the newly added sub-menu trigger. If the sub-menu trigger is entered, the sub-menu numbered menu will be cascaded, allowing sub-menu menu items to be selected.

Viewing Transformation

glLoadIdentity()

To start with, we will initialize the viewing matrix by loading it with the identity matrix, using the command “glLoadIdentity()”, and continue to combine it with new matrices according to where we want to place the “camera”.

Projection

Specifying the projection transformation is like choosing a lens for a camera. We can think of this transformation as determining the field of view and therefore which objects are inside it and, to some extent, how they should look. There are two basic types of projections provided for us by OpenGL, orthographic and perspective, respectively.

glFrustum(),glMatrixmode()

1. Perspective:

We will use the perspective projection to get a more realistic rendering of an object. The object(s) will have the unmistakable characteristic of foreshortening: the further an object is from the camera, the smaller it appears in the final image. This is because the viewing volume of perspective projection is a frustum (a truncated pyramid whose top has been cut off by a plane parallel to its base). The command to define this frustum is “glFrustum()”, which takes the values for left, right, top, bottom, near and far edges. The same set of functions to set the projection transformation, but instead of using the “glOrtho( )” function, the “glFrustum( )” function is given.

Translation: The translation function “glTranslate( )” multipliesthe current matrix by a matrix that moves (translates) an object by the given x, y, and z values (or moves the local coordinate system by the same amounts).

5.2 User Defined Functions

void drawtext(const char text_from_main[],float x , float y,int);

Draws the text which is fetched from the display function. This function holds the parameters of coordinates and font to display.

void drawline(float x,float y,float x1,float y1);

Draws the line using GL_LINE primitive. Takes the parameters of coordinates.

void bitcoin()

Draws the bitcoin using stroke characters and a circle.

void draw_block(float x, float y,float z)

Draws the block according to coordinates given in the parameters.

void draw_block_line(float x, float y,float z)

Draw the block with Line Loop.

void draw_key()

Draws the shape of the key using combinations of the openGL primitives

void draw_computer(float x,float y)

Draws the computer monitor and keyboard.

string sha256(const string str)

Encryption function using Openssl.

void draw_pkt_value(const char text_from_main[],float x , float y,int font)

Draws real value of the given data using encryption.

void drawbox(float x_up,float y_left, float x_down , float y_right)

Draws the box of given height and width.

void draw_chain()

Draws the chain display using sin and cos waves.

void drawpkt()

Draws a moving packet.

Chapter 6

Source Code

The implementation of the project is done using Visual Studio. The program execution starts from main function. The main function calls the call back display. It also calls the timer function recursively after every frame, calculates the transformation values and then calls the display function again and again. In this way it displays the animation.

```
#include <GL/freeglut.h>
#include <sstream>
#include <string>
#include<math.h>
#include <iostream>
#include <iomanip>

#include<time.h>
using namespace std;
#include <openssl/sha.h>
#include <vector>
ostringstream oss;
vector< string > names( 1 );
```

```
void *fonts[] = { GLUT_BITMAP_9_BY_15,
                  GLUT_BITMAP_TIMES_ROMAN_10,
                  GLUT_BITMAP_TIMES_ROMAN_24,
                  GLUT_BITMAP_HELVETICA_18,
        GLUT_BITMAP_HELVETICA_12 };


int flag=0;
int move_bitcoin=0;

void display();
void reshape(int , int );
void keyboard( unsigned char key, int x, int y );
void drawtext(const char text_from_main[],float x , float y,int);
void drawline(float x,float y,float x1,float y1);




void delay(float secs)
{
    float end = clock()/CLOCKS_PER_SEC + secs;
    while((clock()/CLOCKS_PER_SEC) < end);
}


void menu(int id)
```

```
{
switch(id)
{
case 1:     flag =1;
            set_false();
            look =true;
        break;
case 2:  flag=1;
            set_false();
            first = true;

            break;
case 3: flag=2;
        set_false();
        chain = true;
        break;
case 4: flag=3;
        set_false();
        POW = true;
        break;
case 5:set_false();crypto=true;break;
case 6:exit(0);
case 7: set_false();consensus=true;break;

}
}
```

```
void bitcoin()
{

   float theta;




   glTranslatef(0,-10,0);
   // glRotatef(-90,0,0,1);
   glColor3f(0.9,0.7,0.1);

   glBegin(GL_POLYGON);
    for (int i = 0; i < 360; i++)
          {
                theta = i * 3.142 /180;
                glVertex2f(2.5*cos(theta),2.5*sin(theta));
          }
   glEnd();



glTranslatef(-1.5,-1.7,0);
glScalef(0.035,0.035,1);
glColor3f(0.4,0,0);
glLineWidth(10);
glutStrokeString(GLUT_STROKE_ROMAN, (unsigned char*)"B");
glLoadIdentity();
```

```
}
void draw_block(float x, float y,float z)
{
   glOrtho(-5,5,-5,5,-5,5);
   //glNewList(upbox,GL_COMPILE_AND_EXECUTE);
   glBegin(GL_POLYGON);
      glColor3f(0.6,0.5,0.4);

      glVertex3f(x+2,y+2.0,z-5);
      glVertex3f(x+2,y-23.0,z-5);
      glVertex3f(x+12,y-23.0,z-5);                    // back
      glVertex3f(x+12,y+2.0,z-5);
   glEnd();
   glBegin(GL_POLYGON);
      glColor3f(0.4,0.5,0.3);

      glVertex3f(x,y-25.0,z);                         // bootom
      glVertex3f(x+10,y-25.0,z);
      glVertex3f(x+12,y-23.0,z-5);
      glVertex3f(x+2,y-23.0,z-5);
   glEnd();
   glBegin(GL_POLYGON);
      glColor3f(0.4,0.5,0.3);
```

```
    glVertex3f(x+2,y+2.0,z-5);          //same top
    glVertex3f(x+2,y-23.0,z-5);         // background        //left
    glVertex3f(x,y-25.0,z);         // same front
    glVertex3f(x,y,z);          //same  front
  glEnd();
  glBegin(GL_POLYGON);
    glColor3f(0.7,0.8,0.6);
    glVertex3f(x,y,z);                  //same front top left
    glVertex3f(x+10,y,z);               // same front top right            // top
    glVertex3f(x+12,y+2.0,0.0);         // top left
    glVertex3f(x+2,y+2.0,z-5);          // same right side top left
  glEnd();
  glBegin(GL_POLYGON);
    glColor3f(0.7,0.7,0.6);


    glVertex3f(x+10,y,z);     //same left bottom front
    glVertex3f(x+10,y-25.0,z);   // same left bottom front // right side
    glVertex3f(x+12,y-23.0,z-5);   // bottom
    glVertex3f(x+12,y+2.0,z-5);     // top
  glEnd();


  glBegin(GL_POLYGON);
    glColor3f(0.3,0.5,0.4);


    glVertex3f(x,y-25.0,z); //bottom right  // front
    glVertex3f(x+10,y-25,z); // bottom left
    glVertex3f(x+10,y,z);   // top left
```

```
      glVertex3f(x,y,z);    // top right   // x y z
   glEnd();
   glLoadIdentity();


}
void draw_key()
{
   float theta;
   glColor3f(0.9,0.7,0.1);


   glBegin(GL_POLYGON);
    for (int i = 45; i < 300; i++)
          {
                theta = i * 3.142 /180;
                glVertex2f(5+0.5*cos(theta),5+0.5*sin(theta));
          }


    glColor3f(0.9,0.8,0.2);
      glVertex2f(5.499,5.0);
     // glVertex2f(6.399,5.199);
     // glVertex2f(6.299,5.179);
      glVertex2f(6.499,5.099);


      glVertex2f(6.499,5.299);


           glEnd();
```

```
   glColor3f(0,1,1);
   glTranslatef(-0.2,0,0);

   glBegin(GL_POLYGON);
   for (int i = 0; i < 360; i++)
         {
      theta = i * 3.142 /180;
               glVertex2f(5+0.10*cos(theta),5+0.10*sin(theta));
         }
    glEnd();
    glLoadIdentity();

}
void draw_computer(float x,float y)
{

     glTranslatef(0,2,0);
     glBegin(GL_POLYGON);
          glColor3f(0.7,0.65,0.83); //stand
          glVertex3f(x+0.7,y-2.4,-2);
          glVertex3f(x+0.7,y-3,-2);
          glVertex3f(x+1.4,y-3,-2);
          glVertex3f(x+1.4,y-2.4,-2);
      glEnd();
       glBegin(GL_POLYGON);
          glColor3f(0.2,0.3,.4); //stand
          glVertex2f(x+0.5,y-3.0);
```

```
        glVertex2f(x+1.6,y-3.0);
        glVertex2f(x+1.6,y-3.2);
        glVertex2f(x+0.5,y-3.2);
    glEnd();
    glBegin(GL_QUAD_STRIP);
        glColor3f(0.2,0.3,0.4);
        glVertex2f(x-0.2,y-2.6);
        glVertex2f(x+2.2,y-2.6);
        glVertex2f(x+2.2,y+0.1);
        glVertex2f(x-0.2,y+0.1);
        glVertex2f(x-0.2,y+0.1);
        glVertex2f(x-0.2,y-2.6);
        glVertex2f(x+2.2,y+0.1);
        glVertex2f(x+2.2,y-2.6);
    glEnd();
   glBegin(GL_POLYGON); //monitor
        glColor3f(0.6,0.73,0.83);
        glVertex2f(x ,     y-2.4);
        glVertex2f(x+2.0 , y-2.4);
        glVertex2f(x+2.0 ,  y   );
        glVertex2f(x ,      y   );      /////// xand y is passed
    glEnd();
    glBegin(GL_POLYGON);
        glColor3f(0.3,0.5,0.6);
        glVertex2f(x-0.2,y-4.2);
        glVertex2f(x+2.4,y-4.2); //keyboard
        glVertex2f(x+2.2,y-3.4);
```

```
        glVertex2f(x,y-3.4);
    glEnd();
}



string sha256(const string str)
{
    unsigned char hash[SHA256_DIGEST_LENGTH];
    SHA256_CTX sha256;
    SHA256_Init(&sha256);
    SHA256_Update(&sha256, str.c_str(), str.size());
    SHA256_Final(hash, &sha256);
    stringstream ss;
    for(int i = 0; i < SHA256_DIGEST_LENGTH; i++)
    {
        ss << hex << setw(2) << setfill('0') << (int)hash[i];
    }
    return ss.str();
}


void keyboard( unsigned char key, int x, int y )
{


    if (key =='s')
    show=!show;
    if( key == 13 )
    {
```

```
    // enter key
    record =1;
    names.push_back(" ");

}
else if( key == 8 )
{
    // backspace
    names.back().pop_back();
}
else
{
    // regular text
    names.back().push_back( key );
}

if (key== 27)  // esc key
exit(0);
if ('m'==key)
{

    mine = !mine;
}
if ('p'==key)
{
    p=!p;
}
```

```
if ('c'==key)
{
    count++;
    switch (count)
    {
    case -1:
          first=true;
          second=false;
          third=false;
    case 1:
       set_false();
       second = true;

       break;
    case 2:
       set_false();
       third = true;


    default:
       break;
    }
}

if ('b'==key)
{
```

```
    count--;
    switch (count)
    {



    case -1:
            set_false();
            first =true;


    case 1:
       set_false();
       second = true;


       break;
    case 2:
       set_false();
       third = true;



    default:
       break;
    }
 }
 if( ' ' == key )
 {
    isAnimating = !isAnimating;
 }
```

```
   if ('1'==key)
   {
      block1 = !block1;
   }
   if ('2'==key)
   {
      block2 = !block2;
   }
   if ('3'==key)
   {

      block3 = !block3;
   }



}

void draw_pkt_value(const char text_from_main[],float x , float y,int font)
{
   std::string text;
    //text = "----------------------------------BLOCK CHAIN - THE TECHNOLOGY OF FUTURE----------------------------------------";

void drawtext(const char text_from_main[],float x , float y,int font)
{
   std::string text;
```

```
    text = text_from_main;
    const char *real_text = text.data();
    int length = text.size();
    glRasterPos2f(x,y);
    for (int i = 0; i < length; i++)
    {
        glColor3f(0.5,0.0,0.1);
        glutBitmapCharacter(fonts[font],(int)real_text[i]);
    }
    glFlush();

}


void drawbox(float x_up,float y_left, float x_down , float y_right)
{
    glBegin(GL_POLYGON);
        glColor3f(0.1,0.0,1.0);
        glVertex2f(x_up,y_left);
        glVertex2f(x_up,y_right);
        glVertex2f(x_down,y_right);
        glVertex2f(x_down,y_left);


    glEnd();
    glFlush();
```

```
}
void draw_chain()
{

 double y=4;
 glOrtho(-10, 10, -10, 10, -10, 10);
 glTranslatef(0,-3,0);
 glColor3f(0.19, 0.27, 0.39);
 glPointSize(3);

   glBegin(GL_POINTS);
         for (double x = -65; x <= -5; x += 0.01)
         {
            glVertex2f(x, sin(y));
            glVertex2f(x,cos(y));
            y = y+0.01;
            }
   glEnd();

 glLoadIdentity();
}

void drawline(float x,float y,float x1,float y1)
{

   glBegin(GL_LINES);
```

```
        glColor3f(0,0,0);
        glVertex2f(x,y);
        glVertex2f(x1,y1);
    glEnd();

}


void drawpkt()
{
   glBegin(GL_POLYGON);
      glColor3f( 0.1, 0.2, 0.3 );
      glVertex2f(x_pos,0.5);
      glVertex2f(x_pos,-0.5);
      glVertex2f(x_pos+0.5,-0.5);
      glVertex2f(x_pos+0.5,0.5);
   glEnd();
}

void reshape(int w ,int h)
{

   glViewport(0,0,GLsizei(w),GLsizei(h));
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glOrtho(-10,10,-10,10,-10,10);
   //gluPerspective(60,1,2.0,50.0);
```

```
   glMatrixMode(GL_MODELVIEW);


}


void timer1(int  )
{
   glutPostRedisplay();
   glutTimerFunc(1000/60,timer1,0 );


      if (isAnimating)
         {


            angle=angle+0.8;
            switch (state)
               {
                     case 1:
                           if (x_pos<4)
                           {
                              x_pos+=0.05;
                           }
                           else
                              state = -1;
                           break;


                     case -1 :
                              if (x_pos>-5)
                              {
```

```
                                x_pos-=0.05;


                                // state = 0;  stop right
                            }
                            else
                                state=1;
                              // state = 0;  //stop left


                            break;


                    case 0 : break;
                }
            }


    if (true)
    {
        if (first)
        {
                move_bitcoin-=1;
        }
    }

}

void display()
{
```

```
glClear(GL_COLOR_BUFFER_BIT| GL_DEPTH_BUFFER_BIT);
glLoadIdentity();
if (first)
{

    glTranslatef(0,-x_pos,0);
    draw_chain();

    glTranslatef(7,-x_pos,0);
    draw_chain();

    glTranslatef(0,-x_pos,0);
    glColor3f(0.3,0.5,0.4);
    drawtext("BLOCK 1",-8.0,-4.0,3);
    glColor3f(0.0,0.0,1.0);
    drawtext("(GENISYS BLOCK)",-8.3,-4.5,0);
    draw_block(-8.5*5,3.0*5,5.0);

    glTranslatef(0,-x_pos,0);
    drawtext("BLOCK 2",-0.5,-4.0,3);
    draw_block(-1.0*5,3.0*5,5.0);

    glTranslatef(0,-x_pos,0);
    drawtext("BLOCK 3",7.0,-4.0,3);
    draw_block(5.5*5,3.0*5,5.0);
```

```
        glTranslatef(0,-x_pos,0);
        glScalef(1.5,1.5,0);
        glRotatef(angle,0,1,0);
        bitcoin();

        glTranslatef(-1,-x_pos,0);
        glScalef(1.5,1.5,0.0);
        draw_computer(0.0,5.0);
        glLoadIdentity();




    }

    if (look)
    {



      //drawbox(-1.0,-4.0,3.0,4.0);
      glColor3f(0.3,0.5,0.4);
      drawtext("BLOCK",0.5,6.0,2);
      glTranslatef(2,0,0);
```

```
    glScalef(2,1,2);
    glRotatef(angle,0,1,0);

     draw_block_line(-1.0*5,3.0*5,5.0);
    glLoadIdentity();
    glTranslatef(-6,-2,0);
        glColor3f(0.5,0.0,0.1);
        drawtext("Index :",-0.6,3.0,2);
        drawtext("Transaction (data) :",-0.6,2.0,2);
        drawtext("Timestamp :",-0.6,1.0,2);
        drawtext("Previous hash :",-0.6,0.0,2);
        drawtext("POW (nounce) :",-0.6,-1.0,2);

      glLoadIdentity();


   /*
      draw_chain();
      glTranslatef(7,0,0);
      draw_chain();


      if (block1)
      {
         //drawbox(-8.5,-3.0,-6.0,3.0);
         draw_block(-8.5*5,3.0*5,5.0);
         drawtext("BLOCK 1",-8.0,-4.0,3);
```

```
    glColor3f(0.0,0.0,1.0);
    drawtext("(GENISYS BLOCK)",-8.3,-4.5,0);
}
if (block2)
{
    //drawbox(-1.0,-3.0,1.5,3.0);

    draw_block(-1.0*5,3.0*5,5.0);
    drawtext("BLOCK 2",-0.5,-4.0,3);

}
if (block3)
{
    //drawbox(8.5,-3.0,6.0,3.0);
    draw_block(5.5*5,3.0*5,5.0);
    drawtext("BLOCK 3",7.0,-4.0,3);
}

drawpkt();

const char* data;
glColor3f(0.5,0.0,0.1);
if(block1)
{
    int index=1;
    data="My name is root";
    float time_stamp=0.1;
```

```
        const char* pre_hash="";  //genysys block
        bool nounce=0;            // nounce
        if (!p)
        {
           if(!block3&&!block2)
           drawtext(data,-8.3,-2.0,2);
        }
        if (p&&!block3&&!block2)
        {
          draw_pkt_value(data,-8.0,-2.0,2);
        }

    }

    if (!p)
    {
       drawtext("P1",x_pos+0.5,-1.0,4);
    }
    if(p)
    {
       draw_pkt_value(data,x_pos+0.5,-1.0,4);
    }
    drawtext("Press 'm' for mining ",-9.0,-7.0,4);
    */
  }

 if(chain)
```

```
{



    glTranslatef(7,0,0);
    draw_chain();
    if (block1)
    {
       draw_chain();
       //drawbox(-8.5,-3.0,-6.0,3.0);
       draw_block(-8.5*5,3.0*5,5.0);
       drawtext("BLOCK 1",-8.0,-4.0,3);
       glColor3f(0.0,0.0,1.0);
       drawtext("(GENISYS BLOCK)",-8.3,-4.5,0);
    }
    if (block2)
    {
       //drawbox(-1.0,-3.0,1.5,3.0);


       draw_block(-1.0*5,3.0*5,5.0);
       drawtext("BLOCK 2",-0.5,-4.0,3);


    }
    if (block3)
    {
       //drawbox(8.5,-3.0,6.0,3.0);
       draw_block(5.5*5,3.0*5,5.0);
```

```
    drawtext("BLOCK 3",7.0,-4.0,3);
}

    glTranslatef(-1.0,1.0,0);
    draw_computer(0.0,5.0);
    glLoadIdentity();
const char* data;
glColor3f(0.5,0.0,0.1);
if (!p)
{
    drawtext("HASH",-8.0,1.0,3);
    drawtext("HASH",-0.5,1.0,3);
    drawtext("HASH",6.0,1.0,3);
    drawtext("PREVIOUS",-0.5,-1.0,3);
    drawtext("PREVIOUS",6.0,-1.0,3);
}
if(p)
{
    if(block1)
    {
    draw_pkt_value("BLOCK1",-8.0,1.0,3);
    glColor3f(1,0.1,0.3);
    draw_pkt_value("BLOCK1",-0.5,-1.0,3);
    }
    if(block1==false)
    {
        draw_pkt_value("BLOCK2",-0.5,1.0,3);
```

```
            glColor3f(1,0.1,0.3);
            draw_pkt_value("BLOCK2",x_pos,-1.0,3);
          }
       }



}

   if (POW)
   {



     int y=5,y_1=6,y_2=5;

   //glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);

   for( size_t i = 0; i < names.size(); ++i )
   {



      if(record&&show)
    {
      glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
       for( size_t i = 0; i < names.size()-1; ++i )     // for displaying the added transactions
       {
            const char* n = names[i].data();
```

```
            drawtext(n,-3,y_1=y_1-2,3);
            draw_pkt_value(n,-3,y=y-2,3);
        }
      break;
    }


     ostringstream oss;
     oss << "Enter the transaction"<<i+1 << ": " << names[i];

     void* font = GLUT_BITMAP_TIMES_ROMAN_24;

     glRasterPos2i(-3,y_2--);
     glutBitmapString( font, (const unsigned char*)( oss.str().c_str() ) );
  }
  glScalef(1.5,1.5,1.5);
  draw_computer(-5.0,0.0);
  glLoadIdentity();
  if(mine)
  {
  glColor3f(0.5,0.0,0.1);
  drawtext("Mining",-7.0,2,3);
  drawtext("Successful",-7.0,1,3);
  drawtext("NEW BLOCK",3,-9,3);
  glTranslatef(0,-7,0);
    glScalef(0.5,0.5,0.5);
    draw_block(5.5*5,3.0*5,5.0);
```

```
  }

  glTranslatef(0,-7,0);
    glScalef(0.5,0.5,0.5);
    draw_block(-1.0*5,3.0*5,5.0);
    glTranslatef(0,-7,0);
    glScalef(0.5,0.5,0.5);
    draw_block(-8.5*5,3.0*5,5.0);

  if(!mine)
  {
    drawtext("Verification",-7.0,2,3);
    //draw_block(5.5*5,3.0*5,5.0);
  }
  }

 if(consensus)
 {


      glTranslatef(-4,7,0);
      glScalef(0.5,0.5,0.5);
      draw_computer(-5.0,0.0);
      glLoadIdentity();

      glTranslatef(-4,-6,0);
      glScalef(0.5,0.5,0.5);
```

```
        draw_computer(-5.0,0.0);
        glLoadIdentity();

        glTranslatef(6,7,0);
        glScalef(0.5,0.5,0.5);
        draw_computer(-5.0,0.0);
        glLoadIdentity();

        glTranslatef(6,-6,0);
        glScalef(0.5,0.5,0.5);
        draw_computer(-5.0,0.0);
        glLoadIdentity();

      glTranslatef(-5,-3,0);
      glScalef(0.5,0.5,0.5);
      draw_block(-1.0*5,3.0*5,5.0);
      glTranslatef(-3,-3,0);
      glScalef(0.5,0.5,0.5);
      draw_block(-8.5*5,3.0*5,5.0);

//
      glTranslatef(-5,3,0);
      glScalef(0.5,0.5,0.5);
      draw_block(-1.0*5,3.0*5,5.0);
      glTranslatef(-3,3,0);
      glScalef(0.5,0.5,0.5);
      draw_block(-8.5*5,3.0*5,5.0);
```

```
    //
    glTranslatef(4,3,0);
    glScalef(0.5,0.5,0.5);
    draw_block(-1.0*5,3.0*5,5.0);

    glTranslatef(6,3,0);
    glScalef(0.5,0.5,0.5);
    draw_block(-8.5*5,3.0*5,5.0);



   //
    glTranslatef(4,-3,0);
    glScalef(0.5,0.5,0.5);
    draw_block(-1.0*5,3.0*5,5.0);

    glTranslatef(6,-3,0);
    glScalef(0.5,0.5,0.5);
    draw_block(-8.5*5,3.0*5,5.0);

    if(!mine)
    drawtext("ENTER M FOR VERIFYING BLOCKS",-2,7,3);

    if (mine)
    {
       glTranslatef(0,5,0);
       glScalef(0.5,0.5,0.5);
```

```
        bitcoin();

          glTranslatef(-6.5,3,0);
          glScalef(0.5,0.5,0.5);

          draw_block1(5.5*5,3.0*5,5.0);
          glTranslatef(-6.5,-3,0);
          glScalef(0.5,0.5,0.5);
          draw_block1(5.5*5,3.0*5,5.0);
          glTranslatef(2.5,3,0);
          glScalef(0.5,0.5,0.5);
          draw_block1(5.5*5,3.0*5,5.0);
          glTranslatef(2.5,-3,0);
          glScalef(0.5,0.5,0.5);
          draw_block1(5.5*5,3.0*5,5.0);

    drawtext("CONSENSUS COMPLETE",-2,7,3);
    drawtext("NEW BLOCK SUCCESSFULLY ADDED",-3,6,3);
    }

  }

      glFlush();
      glColor3f(1,0.1,0.3);
      drawtext("press   'b'   to go back",-9.0,-7.5,4);
```

```
        drawtext("----------------------------------------------------BLOCK CHAIN - THE NEXT BIG THING----------------------------------------------------",-9.0,9.0,3);

        drawtext("press   'c'   to continue to next page",-9.0,-8.0,4);

        drawtext("press   'p'   to  know packet info",-9.0,-8.5,4);

        glColor3f(1,0,0);

        drawtext("Right click for menu",-9.0,-9.5,4);

    glutSwapBuffers(); // Required to copy color buffer onto the screen.



}



int main(int argc,char **argv)

{

    glutInit(&argc,argv);

    glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE);

    glutInitWindowPosition(100,1000);

    glutInitWindowSize(1200,1000);

    glutCreateWindow("BLOCK CHAIN - THE NEXT BIG THING");


    glutDisplayFunc(display);



static int animeringsmeny;

static int springmeny;

static int menu_opt;

static int menyid,sec_menu;
```

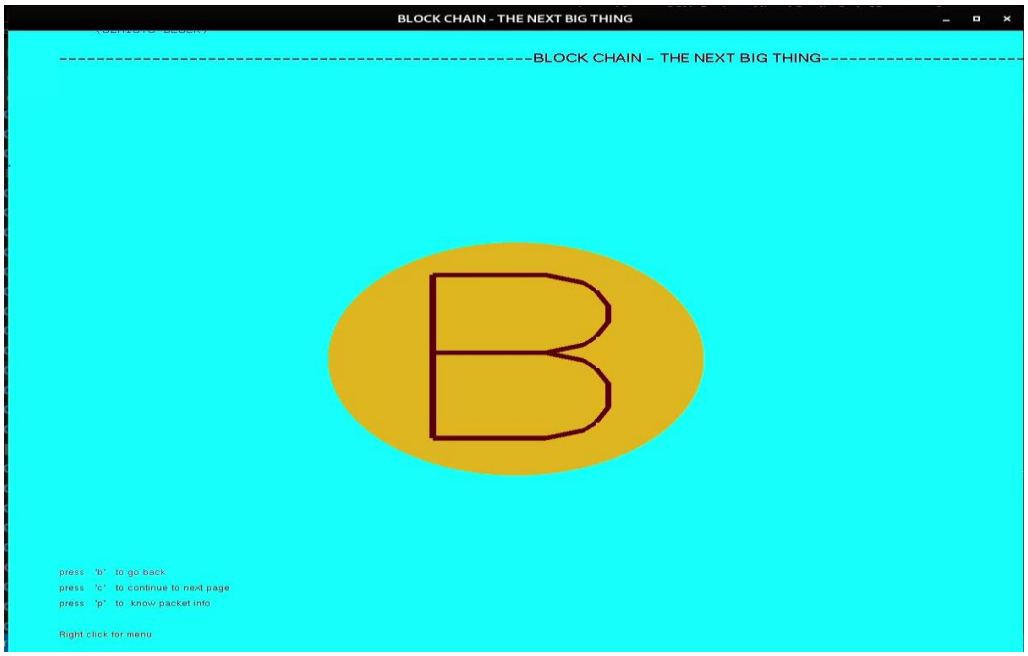
```
static int val = 0;

animeringsmeny = glutCreateMenu(menu);

        // Add sub menu entry
        glutAddMenuEntry("  BLOCK  ", 1);
        glutAddMenuEntry("  GO BACK  ", 2);

menu_opt = glutCreateMenu(menu);
   glutAddMenuEntry("  CHAINING    ",3);
   glutAddMenuEntry("  MINING    ",4);
sec_menu=glutCreateMenu(menu);
   glutAddMenuEntry("CONSENSUS",7)    ;
   glutAddMenuEntry("CRYPTOGRAPHY",5);
        springmeny = glutCreateMenu(menu);

        glutAddSubMenu("    LOOK INSIDE A BOX  ", animeringsmeny);
        //glutAddSubMenu("    INTRO   ", menu_opt);

        // Create the menu, this menu becomes the current menu
        menyid = glutCreateMenu(menu);

        // Create an entry
   glutAddMenuEntry("MY MENU :", 0);
        glutAddSubMenu("    INTRODUCTION    ", springmeny);
        glutAddSubMenu("    TRANSACTION   ", menu_opt);
        // Create an entry
```

```
        glutAddSubMenu("    SECURITY ", sec_menu);
   glutAddMenuEntry("EXIT",6);
glutAttachMenu(GLUT_RIGHT_BUTTON);
glBlendFunc(GL_SRC_ALPHA,GL_DST_ALPHA);

   glutReshapeFunc(reshape);
   glutKeyboardFunc(keyboard) ;

   glutTimerFunc(0,timer1,0);

   //glutTimerFunc(0,timer,0);

   glEnable(GL_DEPTH_TEST);
   init();
   glutMainLoop();

}
```

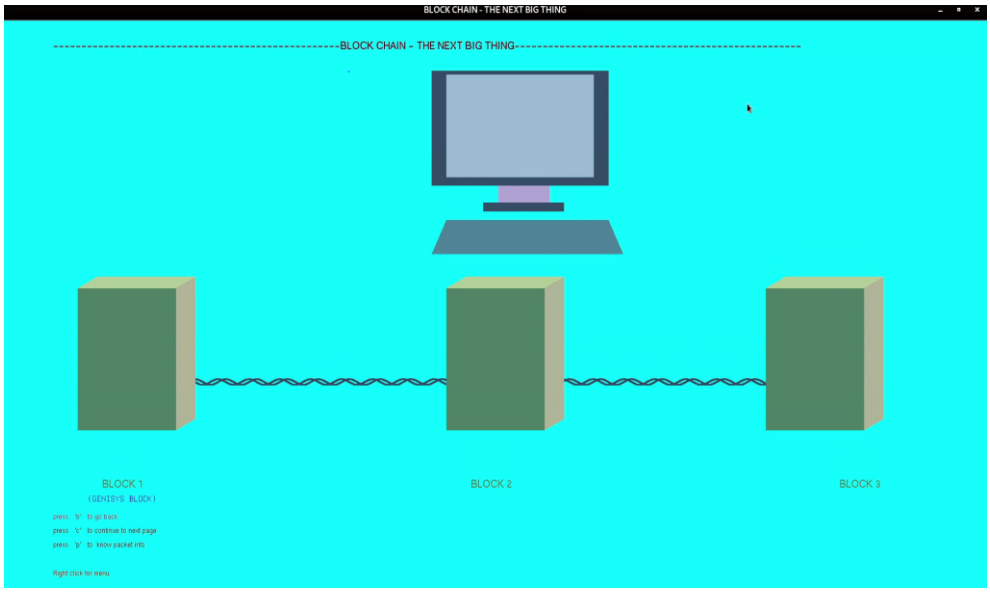
Chapter 7

SCREEN SNAPSHOTS

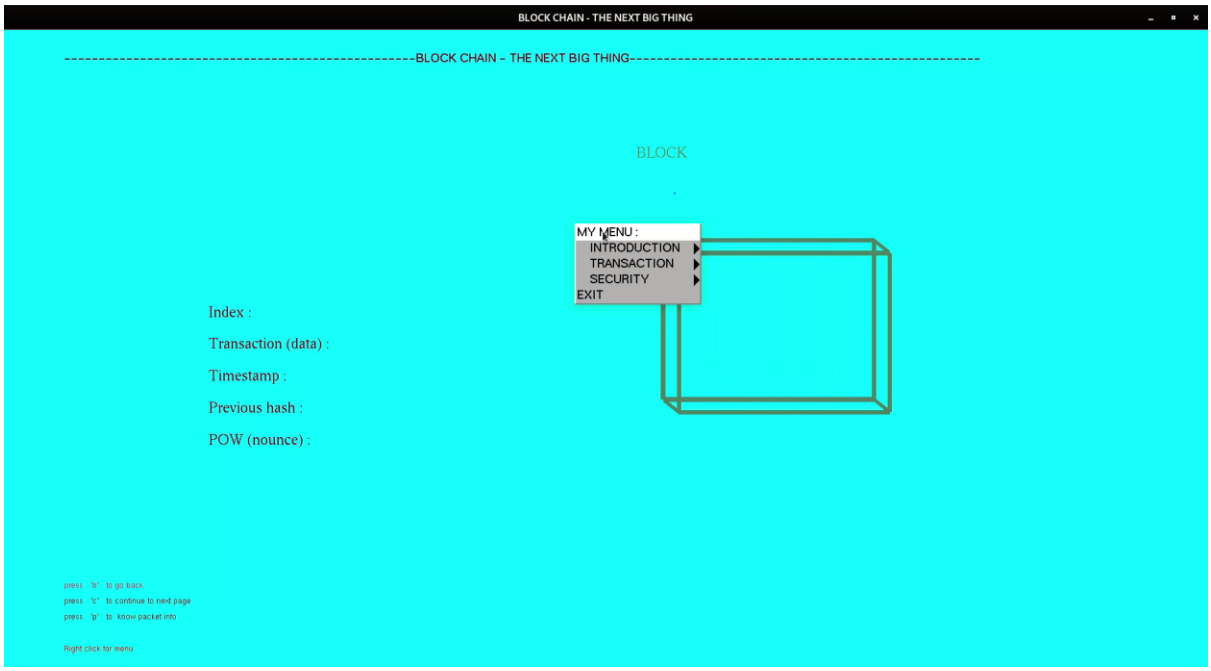
a. Initial screen: Bitcoin



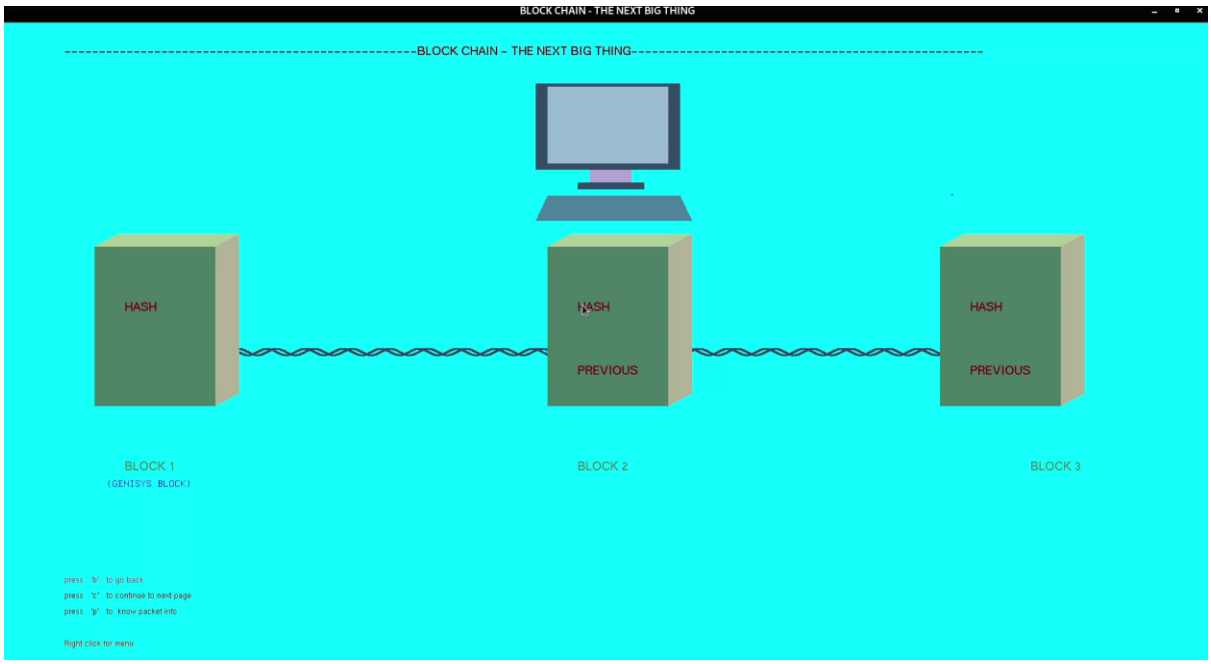
b. Behind bit coin Blockchain animation



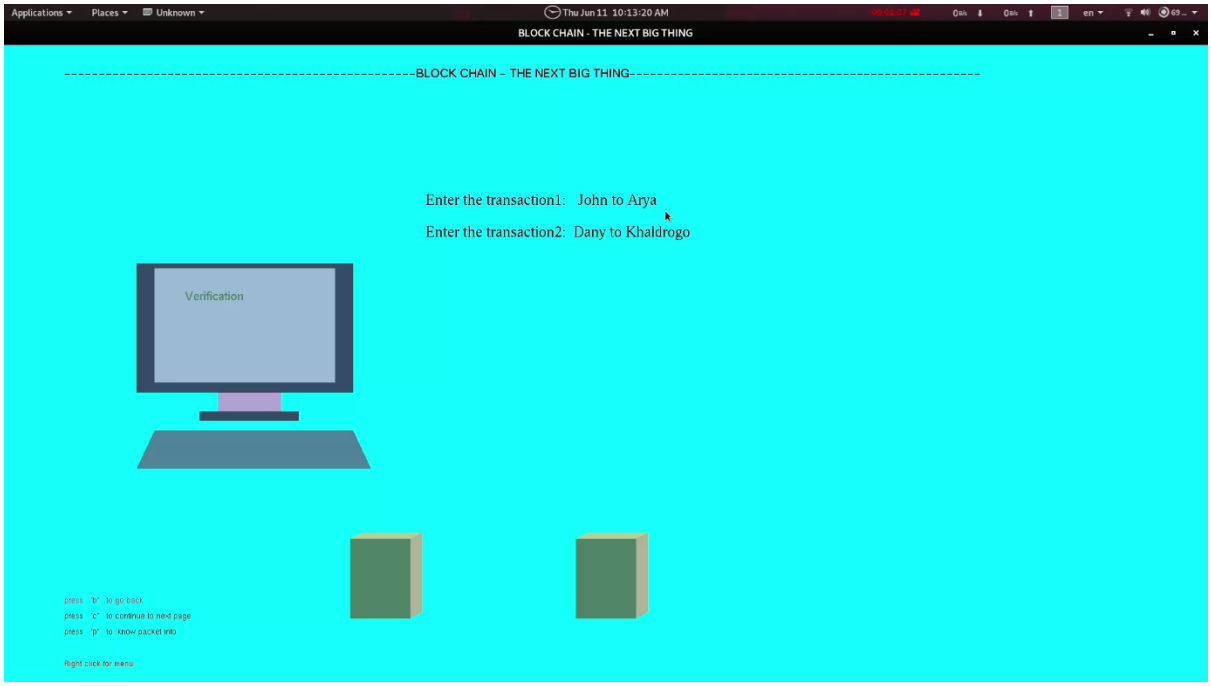
c. A look inside a block with Menu



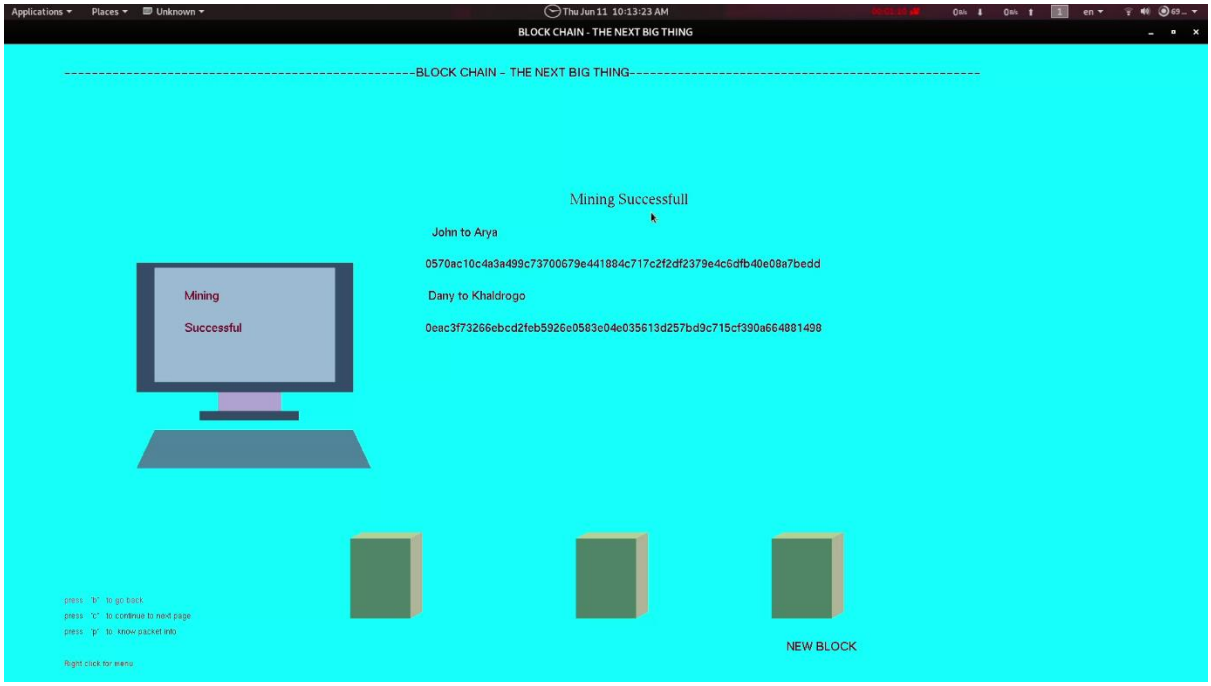
d. Chaining



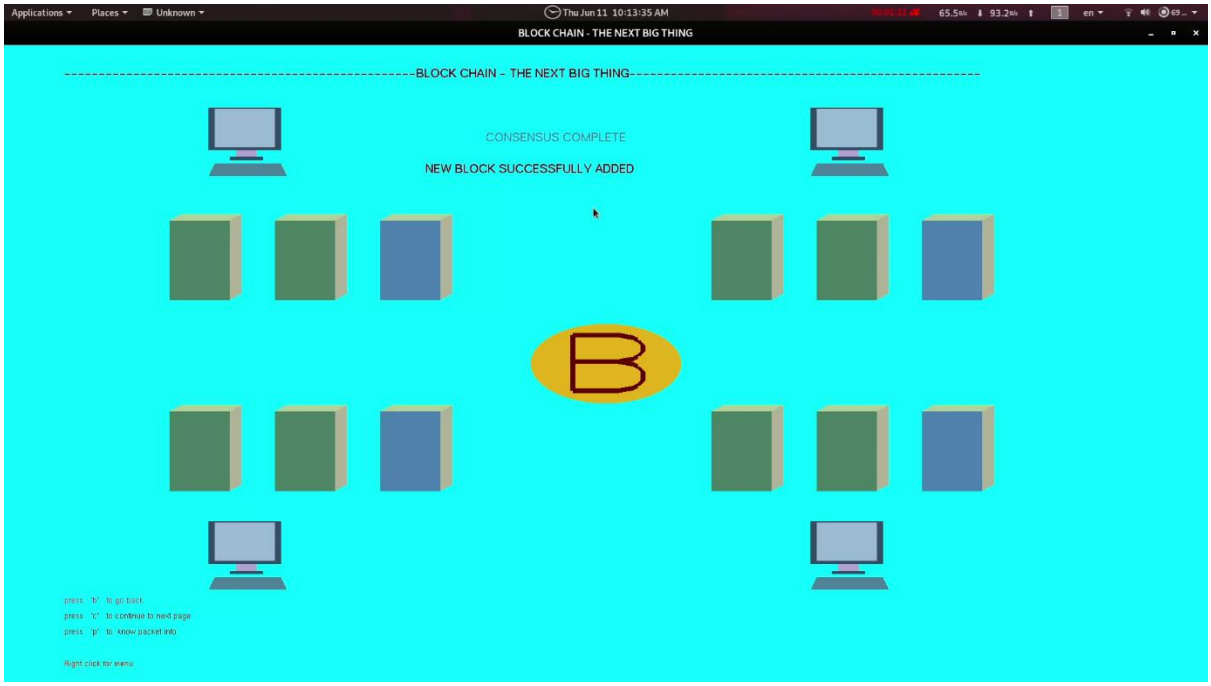
e. Entering the transaction for mining



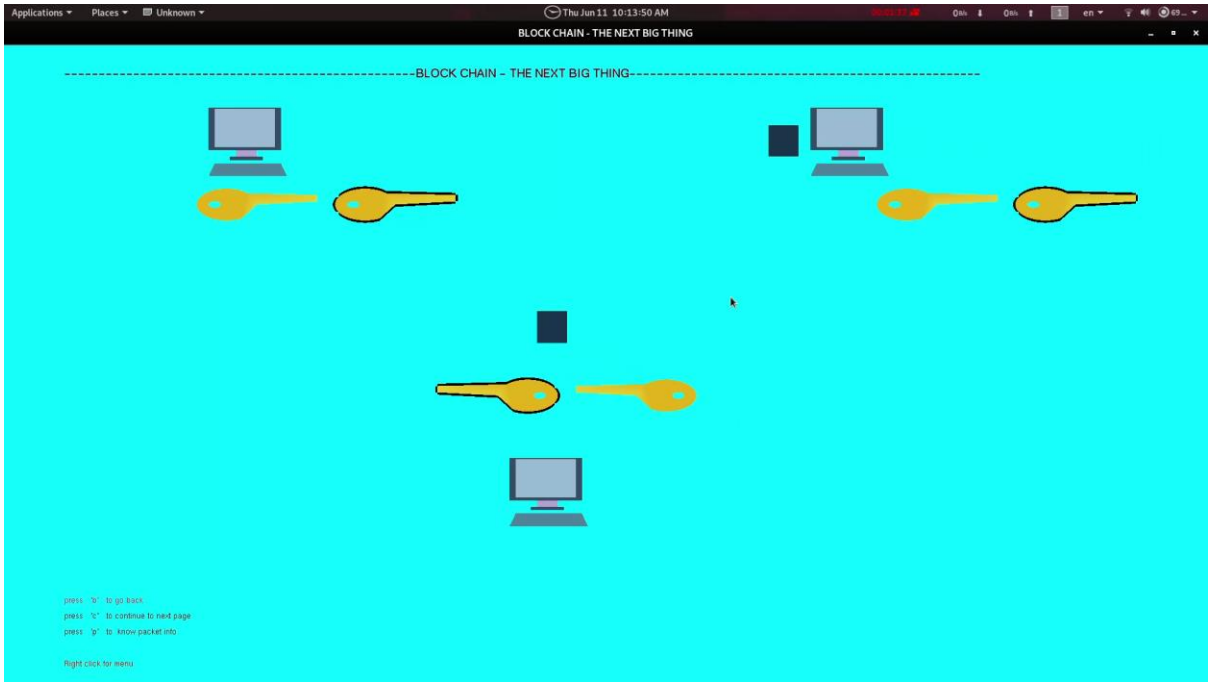
Mining Complete



f. Consensus



g. Cryptography



Chapter 8

FUTURE ENHANCEMENTS

The following are some of the future scopes:

- The project can be developed in by using the less codes and the mathematical functions also can be used for calculating the values.

- It can also be used as an animation video to demonstrate the working of Blockchain Technology.

- The output screen can be recorded and we can make a short animated video.

- The more detailed explanation could be given.

- This project only gives the basic idea of the Blockchain Technology. But there are more facts and more precise way in which Blockchain works. So we could demonstrate that as well.

Chapter 9

CONCLUSION

The project entitled "**Blockchain Visualization**" is developed with the best of my effort within a limited period of time. Due to time constraint and out of coverage in the syllabus, the complexity of this project has not been fully furnished to attain perfection.

The functions in the package have been implemented and tested to ensure the efficiency of operation and they were found to be quite satisfactory.

After the completion of the development and study of the project I have come to the conclusion that computer graphics using OpenGL can be used to developing a much better and complex application that include 2D and 3D image processing.

Lastly and importantly I thank our teachers who were a great source of inspiration clubbed with technical help , my seniors who guided us throughout the course of project, my college for giving us all the facilities we need and my parents who have supported me in all respects.

Development of this computer graphics project has given us a better understanding of the topics and concepts learnt in theory. The project increased our knowledge in the field of computer graphics.

The hands on experience of developing a graphics project has helped us inculcate the good practices and principles of software engineering. This project has imbibed within us the spirit of teamwork.

Chapter 10

BIBLIOGRAPHY:

A number of different sources were thoroughly searched for reference material. The major source being of the **internet** (numerous websites and pages including those of reputed universities and companies). Paper presentations available on the net were immense help. Previous works and ideas by seniors have been incorporated to produce this editor. Help pages provided excellent understanding of the concepts associated with C- programming.

REFERENCES

[1] Donald Hearn & Pauline Baker: Computer Graphics with OpenGL Version,3rd / 4th Edition, Pearson Education,2011

[2] Edward Angel: Interactive Computer Graphics- A Top Down approach with OpenGL, 5 th edition. Pearson Education, 2008

[3] James D Foley, Andries Van Dam, Steven K Feiner, John F Huges Computer graphics with OpenGL: pearson education

[4] Xiang, Plastock : Computer Graphics , sham's outline series, 2nd edition, TMG

[5] https://www.opengl.org/

[6] https://learnopengl.com/Getting-started/OpenGL

[7] https://en.wikipedia.org/wiki/Computer_graphics

[8] Interactive Computer Graphics A Top-Down Approach with OpenGL -Edward Angel, 5th Edition, Addison-Wesley, 2008.

[9] Computer Graphics Using OpenGL – F.S. Hill,Jr. 2nd Edition, Pearson Education, 2001.

[10] Computer Graphics – James D Foley, Andries Van Dam, Steven K Feiner, John F Hughes, Addison-wesley 1997.

[11] Computer Graphics - OpenGL Version – Donald Hearn and Pauline Baker, 2nd Edition, Pearson Education, 2003.

[12] Compter Graphics Using OpenGL – F.S. Hill,Jr. – 2nd Edition, Pearson education, 2001.

[13] Interactive Computer Graphics A Top-Down Approach with OpenGL Edward Angel – 2ndEdition, Addison-Wesley, 2000.

[14] http://www.opengl.org/sdk/docs/man/Online man pages.

[15] http://www.cs.rutgers.edu/~decarlo/428/glman.htmlOnline man pages.

[16] http://nehe.gamedev.net OpenGL tutorials.