

**PROJECT REPORT**

**on**

**CLIPPING PLANE**

*Submitted in partial fulfillment of*

Computer Graphics and Visualization Laboratory (15CSL68)

**Sixth Semester of the Degree of Bachelor of Engineering**

in

**COMPUTER SCIENCE AND ENGINEERING of**

**Visvesvaraya Technological University (VTU), Belgaum during the year 2018-19**



**Carried out by**

**Sovan Kar Siddhant Khemka**

**1SB16CS099** **1SB16CS097**

Under the Guidance

of

**Mrs. Sheela Devi M.**

Assistant Professor

Dept. of Computer Science Engineering,

Sri Sairam College of Engineering, Bangalore

**Department of Computer Science and Engineering**

**SRI SAIRAM COLLEGE OF ENGINEERING**

**Anekal, Bangalore - 562106**

**SRI SAIRAM COLLEGE OF ENGINEERING**

**Anekal, Bangalore - 562106**



**Department of Computer Science and Engineering**

**CERTIFICATE**

Certified that project work entitled “CLIPPING PLANE” is a bonafide work carried out by

|  |  |
| --- | --- |
| **Sovan Kar** | **Siddhant Khemka** |
| **(1SB16CS099)** | **(1SB16CS09**7**)** |

In partial fulfillment for the award of the Bachelor of Engineering in Computer Science and Engineering. Of the Viseveswaraya Technological University, Belgaum during the year 2018-19. It is certified that all corrections/suggestions indicated for the internal assessment have been incorporated in the report deposited in the department library. The project has been approved as it satisfies the academic requirements in respect of the project work prescribed for Bachelor of Engineering Degree.

|  |  |  |
| --- | --- | --- |
| Signature of the Guide | Signature of the HOD |  |
| Mrs. Sheela Devi M. | Dr. B. Shadaksharappa |  |
| Asst. Prof., CSE Dept. | HOD, CSE Dept., |  |
| SSCE. | SSCE. |  |

External Viva

Name of the Examiners: 1.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**DECLARATION**

We, the students of the sixth semester of Computer Science and Engineering, Sri Sairam College of Engineering, Anekal, declare that the work entitled **“CLIPPING PLANE”** has been successfully completed under the guidance of Mrs. Sheela Devi M., Computer Science and Engineering Department, Sri Sairam College of Engineering, Anekal. This dissertation work is submitted to Visvesvaraya Technological University in partial fulfillment of the requirements for the award of Degree of Bachelor of Engineering in Computer Science during the academic year 2018 - 2019. Further, the matter embodied in the project report has not been submitted previously by anyone for the award of any degree or diploma to any university.

Place:

Date:

Team members:

1. Sovan Kar (1SB16CS099)

2. Siddhant Khemka (1SB16CS097)

**ACKNOWLEDGEMENT**

The knowledge and satisfaction that accompanies a successful completion of a project is hard to describe. Behind any successful project there are wise people guiding throughout. We thank them for guiding us, correcting our mistakes, and providing valuable feedback. We would consider it as our privilege to express our gratitude and respect to all those who guided and encouraged us in this project.

We extend our heartfelt gratitude to our chairman, **MJF.LN.LEO MUTHU** and also to our beloved principal, **Dr. Y. VIJAYKUMAR** for the success of this project.

We are grateful to **Dr. B. SHADAKSHARAPPA**, Head of CSE Department and Vice Principal, Sri Sairam College of Engineering, for providing support and encouragement.

We convey our sincerest regards to our Lab guide, **Mrs. Sheela Devi M.,** Dept. of CSE, SSCE, for providing guidance and encouragement at all times needed.

**ABSTRACT**

Clipping, in the context of computer graphics, is a method to selectively enable or disable rendering operations within a defined region of interest. Mathematically, clipping can be described using the terminology of constructive geometry. A rendering algorithm only draws pixels in the intersection between the clip region and the scene model. Lines and surfaces outside the view volume (aka. frustum) are removed.

Clip regions are commonly specified to improve render performance. A well-chosen clip allows the renderer to save time and energy by skipping calculations related to pixels that the user cannot see. Pixels that will be drawn are said to be within the clip region. Pixels that will not be drawn are outside the clip region. More informally, pixels that will not be drawn are said to be "clipped."

**CONTENTS**

|  |  |  |
| --- | --- | --- |
| **SL NO** | **TITLE** | **PG NO** |
| **1** | **Introduction** | 1 |
|  |  |  |
| 1.2 | OpenGL Concept | 2 |
| **2** | **Requirement Specification** | 7 |
| 2.1 | Software Requirements Specification | 7 |
| 2.2 | External Interface Requirements | 7 |
| **3** | **System Design** | 8 |
| **4** | **Implementation Source Code** | 9 |
| **5** | **Snapshots** | 16 |
| **6** | **Testing** | 18 |
| **7** | **Conclusion** | 21 |
| **8** | **Future Enhancements** | 22 |
| **9** | **Bibliography** | 23 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **List of Figures** |  |  |
| **Fig. no.** | **Figure Name** | **Page** | |
| 1.1 | Overview of Computer Graphics |  | 1 |
| 1.2 | OpenGL Pipeline Architecture |  | 3 |
| 1.3 | OpenGL Engine And Drivers |  | 3 |
| 1.4 | Application Development(API’s) |  | 4 |
| 5.1 | Clipping Plane, About the demonstration |  | 16 |
| 5.2 | Clipping Plane, clipped at both x-axis and y-axis |  | 16 |
|  | Clipping Plane,clipped at x-axis |  | 17 |
| 5.3  6.1 | Testing Life cycle |  | 18 |

* 1. **INTRODUCTION**

1. **COMPUTER GRAPHICS**

In this era of computing, computer graphics has become one of the most powerful and interesting fact of computing. It all started with display of data on hardcopy and CRT screen. Now computer graphics is about creation, retrieval, manipulation of models and images.

Graphics today is used in many different areas. Graphics provides one of the most natural means of communicating within a computer, since our highly developed 2D and 3D pattern-recognition abilities allow us to perceive and process pictorial data rapidly and effectively. Interactive computer graphics is the most important means of producing pictures since the invention of photography and television. It has the added advantage that, with the computer, we can make pictures not only of concrete real world objects but also of abstract, synthetic objects, such as mathematical surfaces and of data that have no inherent geometry, such as survey results.

OpenGL is an application program interface (API) offering various functions to implement primitives, models and images. This offers functions to create and manipulate render lighting, coloring, viewing the models. OpenGL offers different coordinate system and frames. OpenGL offers translation, rotation and scaling of objects.

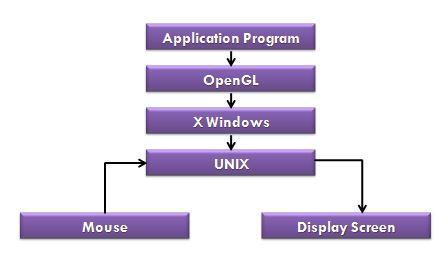


FIG 1.1 **OVERVIEW OF COMPUTER GRAPHICS**

1. **OpenGL CONCEPT**

**1.2.1 INTERFACE**

OpenGL is an application program interface (API) offering various functions to implement primitives, models and images. This offers functions to create and manipulate render lighting, coloring, viewing the models. OpenGL offers different coordinate system and frames. OpenGL offers translation, rotation and scaling of objects. Functions in the main GL library have names that begins with *gl* and are stored in a library usually referred to as GL. The second is the **OpenGL Utility Library(GLU)**. The library uses only GL functions but contains code for creating common objects and simplifying viewing.

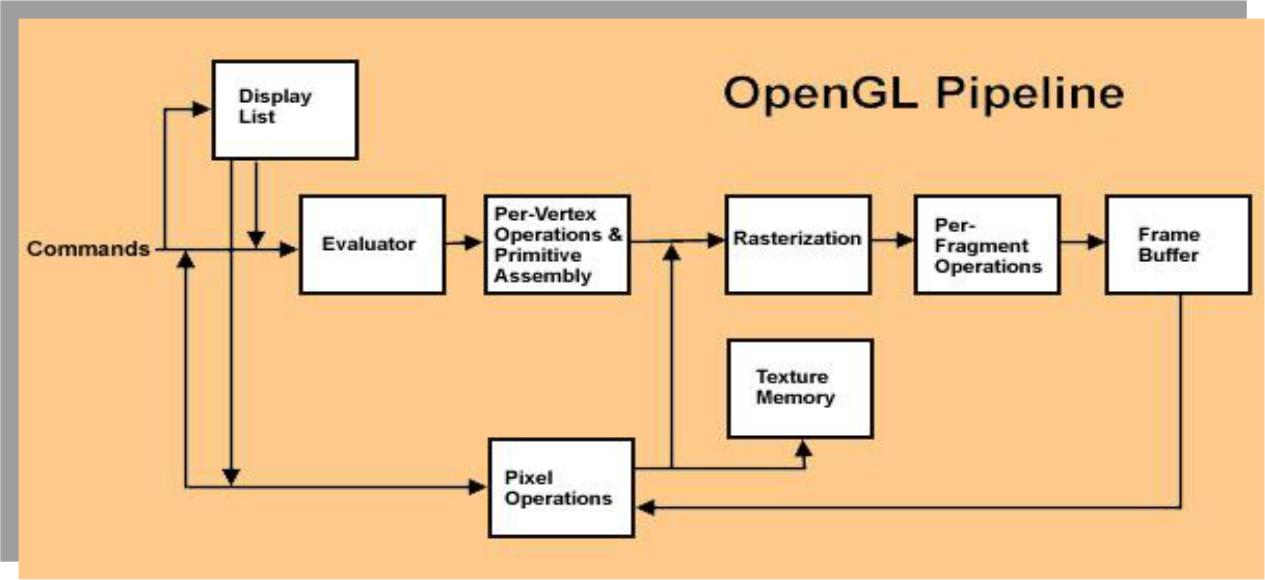
All functions in GLU can be created from the core GL library but application programmers prefer not to write the code repeatedly. The GLU library is available in all OpenGL implementations; functions in the GLU library begin with the letter *glu*. Rather than using a different library for each system we use a readily available library called OpenGL Utility Toolkit(GLUT),which provides the minimum functionality that should be expected in any modern windowing system.

* 1. **OVERVIEW**
* OpenGL(Open Graphics Library) is the interface between a graphics program and graphics hardware. *It is streamlined*. In other words, it provides low-level functionality. For example, all objects are built from points, lines and convex polygons. Higher level objects like cubes are implemented as six four-sided polygons.
* OpenGL supports features like 3-dimensions, lighting, anti-aliasing, shadows, textures, depth effects, etc.
* It is system-independent. It does not assume anything about hardware or operating system and is only concerned with efficiently rendering mathematically described scenes. As a result, it does not provide any windowing capabilities.
* It is a state machine. At any moment during the execution of a program there is a current model transformation.
* It is a rendering pipeline. The rendering pipeline consists of the following steps:
  + - Defines objects mathematically.

\* Arranges objects in space relative to a viewpoint.

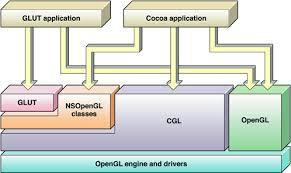
\* Calculates the color of the objects.

1. **OPENGL ARCHITECTURE:**
   1. **Pipeline Architectures**



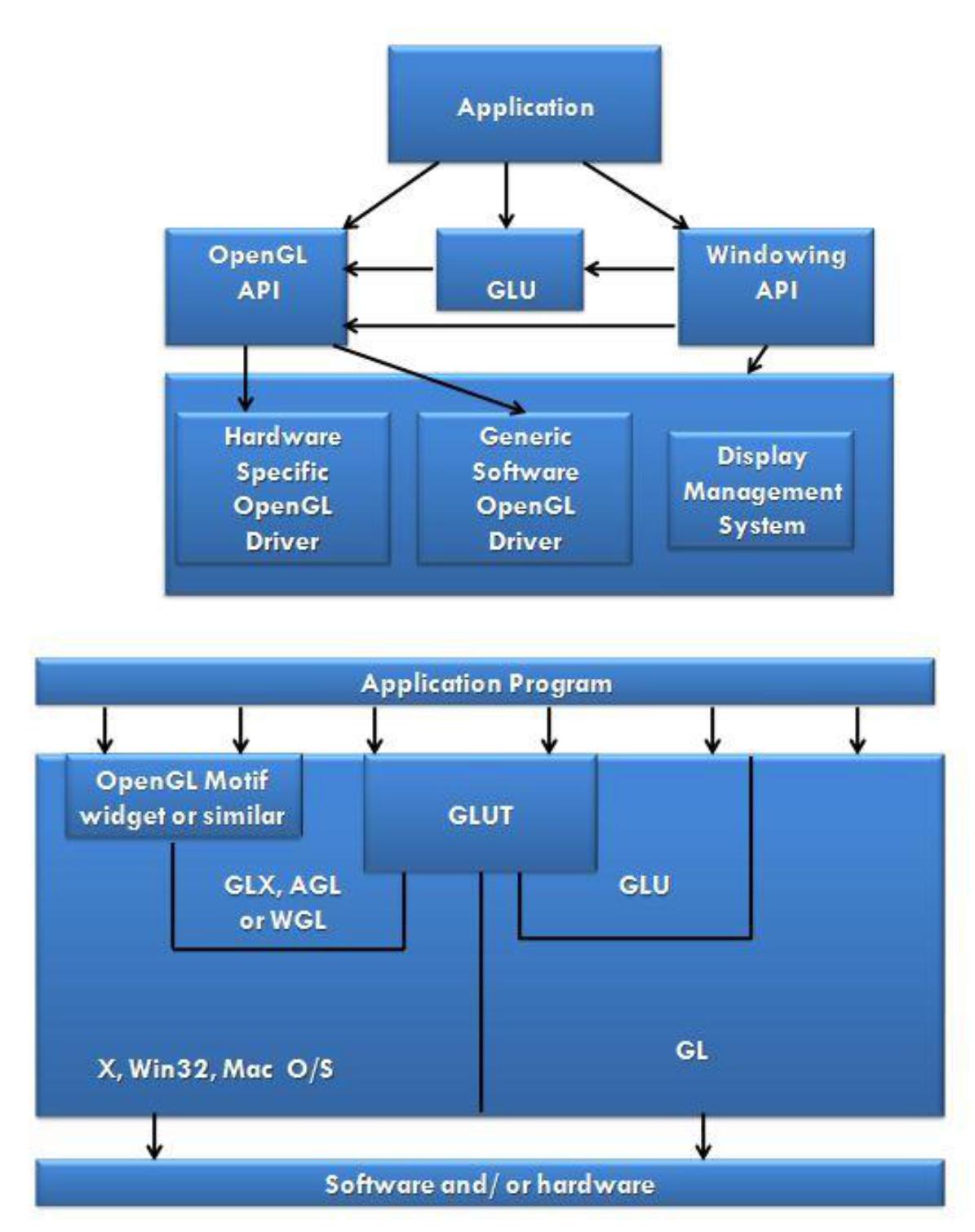
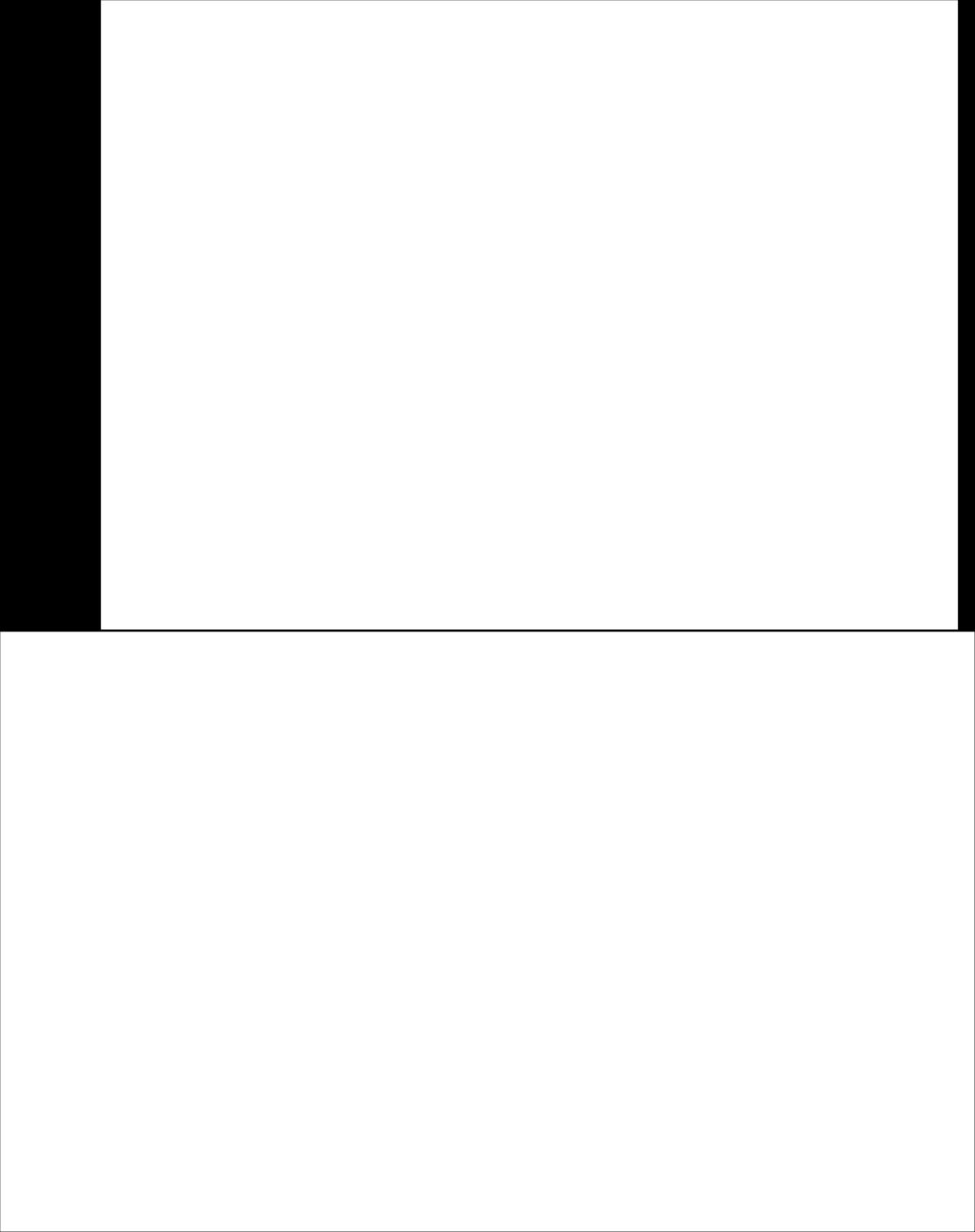
**FIG:1.2 OPENGL PIPELINE ARCHITECTURE**

**2) OpenGL Engine And Drivers**



**FIG:1.3 OPENGL ENGINE AND DRIVERS**

**3) Application Development-API’s**



**FIG:1.4 APPLICATIONS DEVELOPMENT(API’S)**

The above diagram illustrates the relationship 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.

Prototype applications, or ones which do not require all the bells and whistles of a full GUI, may choose to use GLUT instead because of its simplified programming model and window system independence.

**Display Lists:**

All data, whether it describes geometry or pixels, can be saved in a *display list* for current or later use. (1 alternative to retaining data in a displaylist is processing the data immediately - also known as *immediate mode*.) When a display list is executed, the retained data is sent from the display list just as if it were sent by the application in immediate mode.

**Evaluators:**

All geometric primitives are eventually described by vertices. Parametric curves and surfaces may be initially described by control points and polynomial functions called basis functions. Evaluators provide a method to derive the vertices used to represent the surface from the control points. The method is a polynomial mapping, which can produce surface normal, texture coordinates, colors, and spatial coordinate values from the control points.

**Per-Vertex Operations**

For vertex data, next is the "per-vertex operations" stage, which converts. The vertices into primitives. Some vertex data (for example, spatial coordinates) are transformed by 4 x 4 floating-point matrices. Spatial coordinates are projected from a position in the 3D world to a position on your screen.

If advanced features are enabled, this stage is even busier. If texturing is used, texture coordinates may be generated and transformed here. If lighting is enabled, the lighting calculations are performed using the transformed vertex, surface normal, light source position, material properties, and other lighting information to produce a color value.

**Primitive Assembly**

Clipping, a major part of primitive assembly, is the elimination of portions of geometry which fall outside a half-space, defined by a plane. Point clipping simply passes or rejects vertices; line or polygon clipping can add additional vertices depending upon how the line or polygon is clipped.

In some cases, this is followed by perspective division, which makes distant geometric objects appear smaller than closer objects. Then view port

and depth (z coordinate) operations are applied. If culling is enabled and the primitive is a polygon, it then may be rejected by a culling test. Depending upon the polygon mode, a polygon may be drawn as points or lines. The results or this stage are complete geometric primitives, which are the transformed and clipped vertices with related color, depth, and sometimes texture-coordinate values and guidelines for the rasterization step.

**Pixel Operations**

While geometric data takes one path through the OpenGL rendering pipeline, pixel data takes a different route. Pixels from an array in system memory are first unpacked from one of a variety of formats into the proper number of components. Next the data is scaled, biased, and processed by a pixel map. The results are clamped and then either written into texture memory or sent to the rasterization step. If pixel data is read from the frame buffer, pixel-transfer operations (scale, bias, mapping, and clamping) are performed. Then these results are packed into an appropriate format and returned to an array in system memory. There are special pixel copy operations to copy data in the frame buffer to other parts of the frame buffer or to the texture memory. A single pass is made through the pixel transfer operations before the data is written to the texture memory or back to the frame buffer.

**Texture Assembly**

An OpenGL application may wish to apply texture images onto geometric objects to make them look more realistic. If several texture images are used, it's wise to put them into texture objects so that you can easily switch among them. Some OpenGL implementations may have special resources to accelerate texture performance. There may be specialized, high-performance texture memory. If this memory is available, the texture objects may be prioritized to control the use of this limited and valuable resource.

**Rasterization**

Rasterization is the conversion of both geometric and pixel data into fragments. Each fragment square corresponds to a pixel in the frame buffer. Line and polygon stipples, line width, point size, shading model, and coverage calculations to support ant aliasing are taken into consideration as vertices are connected into lines or the interior pixels are calculated for a filled polygon. Color and depth values are assigned for each fragment square.

**2.REQUIREMENT SPECIFICATION**

1. **SOFTWARE REQUIREMENTS SPECIFICATION**

This section attempts to bring out the requirements and specifications as given out by the Visvesvaraya Technological University for the completion of the package. Minimum requirements expected are cursor movement, editing picture objects like point, line, circle, ellipse and polygons. Transformations on objects/selected area should be possible. User should be able to open the package do the required operations and exit from the environment.

**2.2 EXTERNAL INTERFACE REQUIREMENTS**

**User Interface:**

The interface for the 2D package requires for the user to have a mouse connected, and the corresponding drivers software and header files installed. For the convenience of the user, there are menus and sub -menus displayed on the screen.

**Menus:**

The Menus consists of various operations related to drawing on the area specified. It also has the 'clear' option for clearing the screen and also changes the background color.

**Hardware Interface:**

The standard output device, as mentioned earlier has been 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. A keyboard is also required. .

Apart from these hardware requirements, there should be sufficient hard disk space and primary memory available for proper working of the package.

**Software Interface:**

The editor has been implemented on the DOS platform and mainly requires an appropriate version of the compiler to be installed and functional. Though it has been implemented on DOS, it is pretty much platform independent with the

**Hardware Requirements:**

|  |  |  |
| --- | --- | --- |
| **System** | **:** | **Intel** |
| **Frequency** | **:** | **3.0 GHz** |
| **RAM** | **:** | **4 GB** |
| **Disk Capacity** | **:** | **500 GB** |

**Software Requirements:**

**Operating System :** **WINDOWS XP/7/8/10**

**Compiler, IDE : Dev C++**

**3.SYSTEM DESIGN**

1. **Program Structure :**

The entire program is inside clipping\_plane.cpp

Logically, the program consists of a single *“View”*. The view is the main window which lets users load and use the demonstration. There are a set of functions used by both the program and the compiler.

**3.2 Clipping Plane Theory:**

In clipping plane demonstration there are five keys information about which does not appear anywhere. In the code there are keyboard function where keys f,1,2,3,q are used to do the interaction. Key f toggles the clipping plane sphere from wireframe mode to shaded mode. 1,2 and 3 clip the sphere wrt the three planes in which the three axes lie. If left mouse button is pressed then the sphere rotates.

Firstly a complete sphere is displayed. There are 3 clipping planes along the co-ordinate axes namely the x-axis, y-axis, and the z-axis. The key mappings are provided in the code for clipping the sphere along these axes and also to clip the sphere. One or more clipping planes can be used at the same time. The clipped amount can also be varied as per the users wish. The clipped plane can also be reversed based on some predefined key bindings. Finally there is a rotate option for complete viewing of the clipped object.

**4. IMPLEMENTATION SOURCE CODE**

***Clipping\_Plane.cpp***

#include <GL/glut.h>

#include <stdlib.h>

#include <math.h>

#include <stdio.h>

//-----------------------------------------------------------------------------

// function prototypes

//-----------------------------------------------------------------------------

void idleFunc( );

void displayFunc( );

void reshapeFunc( GLsizei width, GLsizei height );

void keyboardFunc( unsigned char, int, int );

void mouseFunc( int button, int state, int x, int y );

void initialize( );

//-----------------------------------------------------------------------------

// global variables and #defines

//-----------------------------------------------------------------------------

#define INC\_VAL 1.0f

#ifndef M\_PI

#define M\_PI 3.14159265359

#endif

// width and height of the window

GLsizei g\_width = 480;

GLsizei g\_height = 360;

// whether to animate

GLboolean g\_rotate = GL\_TRUE;

// fill mode

GLenum g\_fillmode = GL\_FILL;

// light 0 position

GLfloat g\_light0\_pos[4] = { 2.0f, 1.2f, 4.0f, 1.0f };

// light 1 parameters

GLfloat g\_light1\_ambient[] = { .2f, .2f, .2f, 1.0f };

GLfloat g\_light1\_diffuse[] = { 1.0f, 1.0f, 1.0f, 1.0f };

GLfloat g\_light1\_specular[] = { 1.0f, 1.0f, 1.0f, 1.0f };

GLfloat g\_light1\_pos[4] = { -2.0f, 0.0f, -4.0f, 1.0f };

// toggle each of 3 clipping planes

GLboolean g\_clip1 = GL\_FALSE;

GLboolean g\_clip2 = GL\_FALSE;

GLboolean g\_clip3 = GL\_FALSE;

// clipping planes

GLdouble eqn1[4] = { 1.0, 0.0, 0.0, 0.0 };

GLdouble eqn2[4] = { 0.0, 1.0, 0.0, 0.0 };

GLdouble eqn3[4] = { 0.0, 0.0, 1.0, 0.0 };

// modelview stuff

GLfloat g\_angle\_y = 32.0f;

GLfloat g\_inc = 0.0f;

GLfloat g\_eye\_y = 0;

// translation for the clipping planes

GLfloat g\_clip\_x = 0.0f;

GLfloat g\_clip\_y = 0.0f;

GLfloat g\_clip\_z = 0.0f;

//-----------------------------------------------------------------------------

// Name: main( )

// Desc: entry point

//-----------------------------------------------------------------------------

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

{

// initialize GLUT

glutInit( &argc, argv );

// double buffer, use rgb color, enable depth buffer

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

// initialize the window size

glutInitWindowSize( g\_width, g\_height );

// set the window postion

glutInitWindowPosition( 100, 100 );

// create the window

glutCreateWindow( "Clipping Planes in OpenGL" );

// set the idle function - called when idle

glutIdleFunc( g\_rotate ? idleFunc : NULL );

// set the display function - called when redrawing

glutDisplayFunc( displayFunc );

// set the reshape function - called when client area changes

glutReshapeFunc( reshapeFunc );

// set the keyboard function - called on keyboard events

glutKeyboardFunc( keyboardFunc );

// set the mouse function - called on mouse stuff

glutMouseFunc( mouseFunc );

// do our own initialization

initialize();

// let GLUT handle the current thread from here

glutMainLoop();

return 0;

}

//-----------------------------------------------------------------------------

// Name: initialize( )

// Desc: sets initial OpenGL states and initializes any application data

//-----------------------------------------------------------------------------

void initialize()

{

// set the GL clear color - use when the color buffer is cleared

glClearColor( 0.0f, 0.0f,0.0f, 1.0f );

// set the shading model to 'smooth'

glShadeModel( GL\_SMOOTH );

// enable depth

glEnable( GL\_DEPTH\_TEST );

// set the front faces of polygons

glFrontFace( GL\_CCW );

// set fill mode

glPolygonMode( GL\_FRONT\_AND\_BACK, g\_fillmode );

// enable lighting

glEnable( GL\_LIGHTING );

// enable lighting for front

glLightModeli( GL\_FRONT, GL\_TRUE );

// material have diffuse and ambient lighting

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

// enable color

glEnable( GL\_COLOR\_MATERIAL );

// enable light 0

glEnable( GL\_LIGHT0 );

// setup and enable light 1

glLightfv( GL\_LIGHT1, GL\_AMBIENT, g\_light1\_ambient );

glLightfv( GL\_LIGHT1, GL\_DIFFUSE, g\_light1\_diffuse );

glLightfv( GL\_LIGHT1, GL\_SPECULAR, g\_light1\_specular );

glEnable( GL\_LIGHT1 );

printf( "----------------------------------------------------\n" );

printf( "Clipping Planes sample in OpenGL\n" );

printf( "----------------------------------------------------\n" );

printf( "'1' - toggle x=0 halfplane\n" );

printf( "'2' - toggle y=0 halfplane\n" );

printf( "'3' - toggle z=0 halfplane\n" );

printf( "'j', 'l' - translate x=0 halfplane (when toggled)\n" );

printf( "',', 'i' - translate y=0 halfplane (when toggled)\n" );

printf( "'u', 'm' - translate z=0 halfplane (when toggled)\n" );

printf( "'x', 'y', 'z' - reverse the corresponding half plane\n" );

printf( "\n" );

printf( "'-', '=' - rotate about y-axis\n" );

printf( "(L/R) mouse buttons - rotate about y-axis\n" );

printf( "'[', ']' - rotate viewpoint about x-axis\n" );

printf( "'f' - toggle fill/wireframe drawmode\n" );

printf( "----------------------------------------------------\n" );

printf( "\n" );

}

//-----------------------------------------------------------------------------

// Name: reshapeFunc( )

// Desc: called when window size changes

//-----------------------------------------------------------------------------

void reshapeFunc( GLsizei w, GLsizei h )

{

// save the new window size

g\_width = w; g\_height = h;

// map the view port to the client area

glViewport( 0, 0, w, h );

// set the matrix mode to project

glMatrixMode( GL\_PROJECTION );

// load the identity matrix

glLoadIdentity( );

// create the viewing frustum

gluPerspective( 45.0, (GLfloat) w / (GLfloat) h, 1.0, 300.0 );

// set the matrix mode to modelview

glMatrixMode( GL\_MODELVIEW );

// load the identity matrix

glLoadIdentity( );

// position the view point

gluLookAt( 0.0f, 3.5f \* sin( g\_eye\_y ), 3.5f \* cos( g\_eye\_y ),

0.0f, 0.0f, 0.0f,

0.0f, ( cos( g\_eye\_y ) < 0 ? -1.0f : 1.0f ), 0.0f );

// set the position of the lights

glLightfv( GL\_LIGHT0, GL\_POSITION, g\_light0\_pos );

glLightfv( GL\_LIGHT1, GL\_POSITION, g\_light1\_pos );

}

//-----------------------------------------------------------------------------

// Name: keyboardFunc( )

// Desc: key event

//-----------------------------------------------------------------------------

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

{

switch( key )

{

case 'f':

g\_fillmode = ( g\_fillmode == GL\_FILL ? GL\_LINE : GL\_FILL );

glPolygonMode( GL\_FRONT\_AND\_BACK, g\_fillmode );

break;

// toggle the 3 clipping planes

case '1':

g\_clip1 = !g\_clip1;

break;

case '2':

g\_clip2 = !g\_clip2;

break;

case '3':

g\_clip3 = !g\_clip3;

break;

case 'q':

exit( 0 );

break;

// set the rotation along the y axis

case '-':

g\_angle\_y -= INC\_VAL \* 4.0f;

break;

case '=':

g\_angle\_y += INC\_VAL \* 4.0f;

break;

// move the view point up and down

case '[':

g\_eye\_y -= .1f;

break;

case ']':

g\_eye\_y += .1f;

break;

// translate each clipping plane

case 'j':

if( g\_clip1 && g\_clip\_x > -1.0f ) g\_clip\_x -= .1f;

break;

case 'l':

if( g\_clip1 && g\_clip\_x < 1.0f ) g\_clip\_x += .1f;

break;

case ',':

if( g\_clip2 && g\_clip\_y > -1.0f ) g\_clip\_y -= .1f;

break;

case 'i':

if( g\_clip2 && g\_clip\_y < 1.0f ) g\_clip\_y += .1f;

break;

case 'u':

if( g\_clip3 && g\_clip\_z > -1.0f ) g\_clip\_z -= .1f;

break;

case 'm':

if( g\_clip3 && g\_clip\_z < 1.0f ) g\_clip\_z += .1f;

break;

// reverse the half space that is removed

case 'x':

eqn1[0] \*= -1;

break;

case 'y':

eqn2[1] \*= -1;

break;

case 'z':

eqn3[2] \*= -1;

break;

}

// do a reshape since g\_eye\_y might have changed

reshapeFunc( g\_width, g\_height );

glutPostRedisplay( );

}

//-----------------------------------------------------------------------------

// Name: mouseFunc( )

// Desc: handles mouse stuff

//-----------------------------------------------------------------------------

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

{

if( button == GLUT\_LEFT\_BUTTON )

{

// rotate

if( state == GLUT\_DOWN )

g\_inc -= INC\_VAL;

else

g\_inc += INC\_VAL;

}

else if ( button == GLUT\_RIGHT\_BUTTON )

{

if( state == GLUT\_DOWN )

g\_inc += INC\_VAL;

else

g\_inc -= INC\_VAL;

}

else

g\_inc = 0.0f;

glutPostRedisplay( );

}

//-----------------------------------------------------------------------------

// Name: idleFunc( )

// Desc: callback from GLUT

//-----------------------------------------------------------------------------

void idleFunc( )

{

// render the scene

glutPostRedisplay( );

}

//-----------------------------------------------------------------------------

// Name: displayFunc( )

// Desc: callback function invoked to draw the client area

//-----------------------------------------------------------------------------

void displayFunc( )

{

// clear the color and depth buffers

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

glPushMatrix( );

// rotate the sphere about y axis

glRotatef( g\_angle\_y += g\_inc, 0.0f, 1.0f, 0.0f );

// set up the clipping planes

glPushMatrix( );

glTranslatef( g\_clip\_x, 0.0f, 0.0f );

glClipPlane( GL\_CLIP\_PLANE0, eqn1 );

glPopMatrix( );

glPushMatrix( );

glTranslatef( 0.0f, g\_clip\_y, 0.0f );

glClipPlane( GL\_CLIP\_PLANE1, eqn2 );

glPopMatrix( );

glPushMatrix( );

glTranslatef( 0.0f, 0.0f, g\_clip\_z );

glClipPlane( GL\_CLIP\_PLANE2, eqn3 );

glPopMatrix( );

// enable each clipping plane

if( g\_clip1 )

glEnable( GL\_CLIP\_PLANE0 );

else

glDisable( GL\_CLIP\_PLANE0 );

if( g\_clip2 )

glEnable( GL\_CLIP\_PLANE1 );

else

glDisable( GL\_CLIP\_PLANE1 );

if( g\_clip3 )

glEnable( GL\_CLIP\_PLANE2 );

else

glDisable( GL\_CLIP\_PLANE2 );

// draw spheres inside of spheres

glColor3f( 0.4f, .4f, 1.0f );

glutSolidSphere( .23, 16, 16 );

glColor3f( 1.0f, .4f, .4f );

glutSolidSphere( .45, 16, 16 );

glColor3f( 1.0f, .8f, .4f );

glutSolidSphere( .71, 16, 16 );

glColor3f( 0.4f, 1.0f, .4f );

glutSolidSphere( 1.0, 16, 16 );

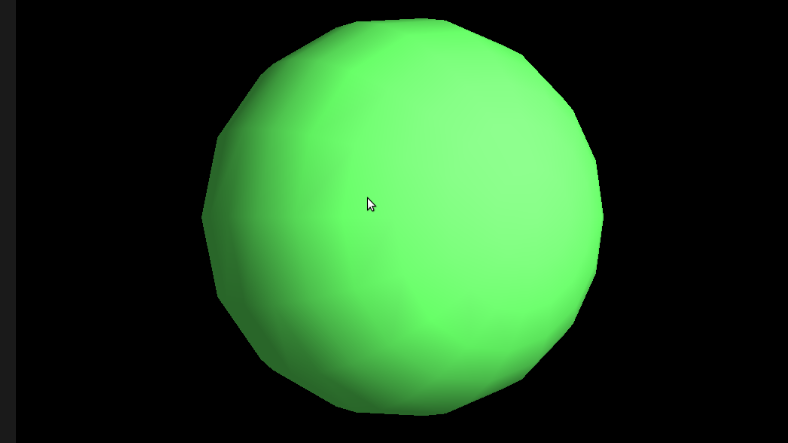
glPopMatrix( );

glFlush( );

glutSwapBuffers( );

}

**5.SNAPSHOTS**



**Fig 5.1**- Clipping Plane, About the demonstration

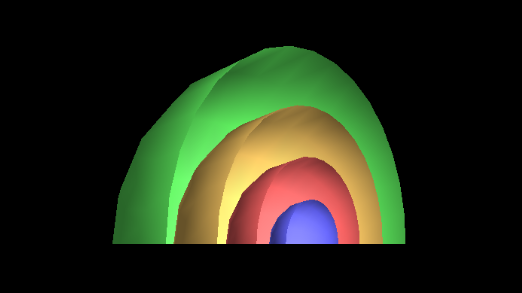
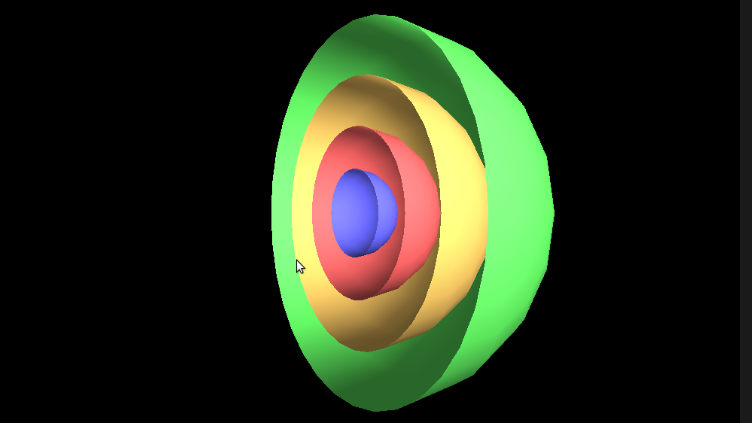
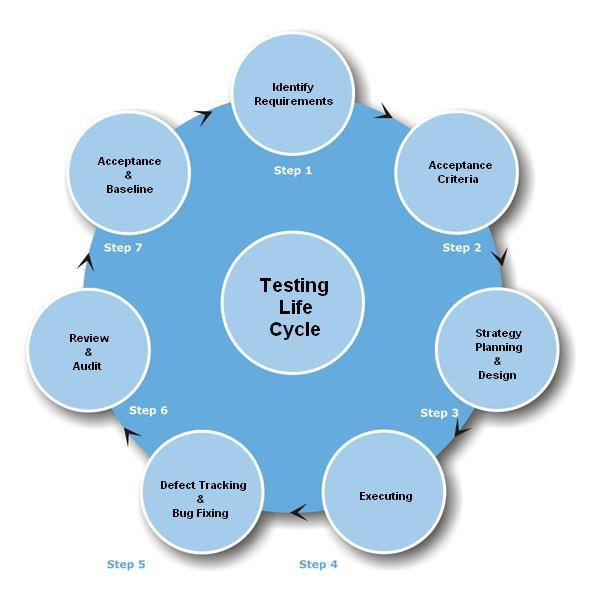
****

Fig 5.2- Clipping Plane, clipped at both x-axis and y-axis



**Fig 5.3 –** Clipping Plane, clipped at both x-axis.

**6.TESTING**



**FIG 6.1 TESTING LIFE CYCLE**

**6.2 TYPES OF TESTING**

**6.2.1 MANUAL TESTING**

Manual testing includes testing a software manually, i.e., without using any automated tool or any script. In this type, the tester takes over the role of an end-user and tests the software to identify any unexpected behavior or bug. There are different stages for manual testing such as unit testing, integration testing, system testing, and user acceptance testing.

**6.2.2 AUTOMATION TESTING**

Automation testing, which is also known as Test Automation, is when the tester writes scripts and uses another software to test the product. This process involves

automation of a manual process. Automation Testing is used to re-run the test scenarios that were performed manually, quickly, and repeatedly.

**6.2.3 BLACK-BOX TESTING**

The technique of testing without having any knowledge of the interior workings of the application is called black-box testing. The tester is oblivious to the system architecture and does not have access to the source code. Typically, while performing a black-box test, a tester will interact with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon.

**6.2.4 WHITE-BOX TESTING**

White-box testing is the detailed investigation of internal logic and structure of the code. White-box testing is also called **glass testing** or **open-box testing**. In order to perform **white-box** testing on an application, a tester needs to know the internal workings of the code. The tester needs to have a look inside the source code and find out which unit/chunk of the code is behaving inappropriately.

**6.2.5 GREY-BOX TESTING**

Grey-box testing is a technique to test the application with having a limited knowledge of the internal workings of an application. In software testing, the phrase the more you know, the better carries a lot of weight while testing an application.

**6.2.6 FUNCTIONAL TESTING**

This is a type of black-box testing that is based on the specifications of the software that is to be tested. The application is tested by providing input and then the results are examined that need to conform to the functionality it was intended for. Functional testing of a software is conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements.

**6.2.7 UNIT TESTING**

This type of testing is performed by developers before the setup is handed over to the testing team to formally execute the test cases. Unit testing is performed by the respective developers on the individual units of source code assigned areas. The developers use test data that is different from the test data of the quality assurance team. The goal of unit testing is to isolate each part of the program and show that individual parts are correct in terms of requirements and functionality.

**6.2.8 INTEGRATION TESTING**

Integration testing is defined as the testing of combined parts of an application to determine if they function correctly. Integration testing can be done in two ways: Bottom-up integration testing and Top-down integration testing.

**6.2.9 SYSTEM TESTING**

System testing tests the system as a whole. Once all the components are integrated, the application as a whole is tested rigorously to see that it meets the specified Quality Standards. This type of testing is performed by a specialized testing team.

**6.3.0 REGRESSION TESTING**

Whenever a change in a software application is made, it is quite possible that other areas within the application have been affected by this change. Regression testing is performed to verify that a fixed bug hasn't resulted in another functionality or business rule violation. The intent of regression testing is to ensure that a change, such as a bug fix should not result in another fault being uncovered in the application.

**6.3.1 ACCEPTANCE TESTING**

This is arguably the most important type of testing, as it is conducted by the Quality Assurance Team who will gauge whether the application meets the intended specifications and satisfies the client’s requirement. The QA team will have a set of pre-written scenarios and test cases that will be used to test the application. By performing acceptance tests on an application, the testing team will deduce how the application will perform in production. There are also legal and contractual requirements for acceptance of the system.

**6.3.2 ALPHA TESTING**

This test is the first stage of testing and will be performed amongst the teams (developer and QA teams). Unit testing, integration testing and system testing when combined together is known as alpha testing. During this phase, the following aspects will be tested in the application:

* Spelling Mistakes
* Broken Links
* Cloudy Directions
* The Application will be tested on machines with the lowest specification to test loading times and any latency problems.
  1. **BETA TESTING**

This test is performed after alpha testing has been successfully performed. In beta testing, a sample of the intended audience tests the application. Beta testing is also known as **pre-release testing**. Beta test versions of software are ideally distributed to a wide audience on the Web, partly to give the program a "real-world" test and partly to provide a preview of the next release. In this phase, the audience will be testing the following:

* Users will install, run the application and send their feedback to the project team
* Typographical errors, confusing application flow, and even crashes.
* Getting the feedback, the project team can fix the problems before releasing the software to the actual users.
  1. **NON-FUNCTIONAL TESTING**

This section is based upon testing an application from its non-functional attributes. Non-functional testing involves testing a software from the requirements which are nonfunctional in nature but important such as performance, security, user interface, etc. Some of the important and commonly used non-functional

* Load Testing
* Performance Testing
* Stress Testing

**7. CONCLUSION**

This approach to the beginning graphics course is oriented toward the science student and fits the growing use of visualization in the sciences, but it should be possible to focus the projects towards many other fields to support visualization in these fields. Versions of this course have been taught twice (at two different universities) with significant student interest and success, although it has so far been difficult to reach significant numbers of science students. Some stability in the course content and some marketing of the course to the sciences are needed to increase interest.

Conversations about and presentations of this course to a number of people in computer science, computational science, and the sciences lead the author to believe that this approach is worth sharing with the graphics community in spite of the relative lack of experience, because other fields are eager to gain the advantages of learning about graphics without having to master fundamental graphics principles and without having to rely on pre-written graphics applications. The project materials described here are being developed with funding from the National Science Foundation. These materials include descriptions of projects for each of the sciences, source code for the examples, supplements on various graphics topics, and pointers to example materials from various sources

Before concluding, it is worth mentioning that this project would never have been possible without the tremendous amount of encouragement by the staff and guides of our department.

We are content with the outcome of this project and are hopeful that it meets the requirements expected and we wish that it may inspire others to be creative and critical in the field of computer graphics and have a newfound appreciation for the Open Graphics Library.

* 1. **FUTURE ENHANCEMENTS**
* Although it isn’t noticeable on a relatively newer and powerful processor, the method of applying the clipping plane procedure is a hectic work. A possible feature to implement could be to perform all the design & play using multithreading to split the workload onto multiple threads, reducing the minimum computational time required between time steps.
* Since Contrast modifies the RGB values instead of HSL, further attempts can also be made.
* Other possibilities include clipping any object inserted within the clipping window.

**9. BIBILIOGRAPHY**

**REFERENCE BOOKS:**

1. **Edward angel: Interactive computer graphics A TOP-DOWN Approach with OpenGL, 2nd edition, Addison-Wesley, 2000.**
2. **F.S. Hill, Jr.: computer graphics using OpenGL, 2nd edition, Pearson education, 2001.**

**WEB URL’S:**

[*Http://msdn.microsoft.com*](Http://msdn.microsoft.com)

[*http://codeproject.com*](http://codeproject.com)

[*http://stackoverflow.com*](http://stackoverflow.com)