**CHAPTER 1**

**INTRODUCTION OF OPEN-GL**

**HISTORY**

OpenGL was developed by **‘Silicon Graphics Inc‘** (SGI) on 1992 and is popular in the gaming industry where it competes with the Direct3D in the Microsoft Windows platform. OpenGL is broadly used in CAD(Computer Aided Design),virtual reality, scientific visualization, information visualization, flight simulation and video games development.

OpenGL is a standard specification that defines an API that is multi-language and multi-platform and that enables the codification of applications that output computerized graphics in 2D and 3D.

The interface consists in more than 250 different functions, which can be used to draw complex tridimensional scenes with simple primitives. It consist of many functions that help to create an real world object and an particular existence for an object can be given.

**CHARACTERSTICS**

* OpenGL is a better documented API.
* OpenGL is also a cleaner API and much easier to learn and program.
* OpenGL has the best demonstrated 3D performance for any API.
* Microsoft's Direct3D group is already planning a major API change called Direct Primitive that will leave any existing investment in learning Direct3D immediate mode largely obsolete.

**CHAPTER 2**

**LITERATURE SURVEY**

Stingley invented the synthetic rubber, he tried to find uses for it and someone to manufacture it. He offered his invention to the Bettis Rubber Company (for whom he worked at the time).They turned it down because the material was not very durable. So he took it to the toy company [Wham-O](http://en.wikipedia.org/wiki/Wham-O) and they worked on developing a more durable version. This version is still manufactured by WhamO.

"It took us nearly two years to iron the kinks out of Super Ball before we produced it," according to [Richard Knerr](http://en.wikipedia.org/wiki/Richard_Knerr), President of Wham-O. "It always had that marvelous springiness... But it had a tendency to fly apart. We've licked that with a very high-pressure technique for forming it. Now we're selling millions."

When the Super Ball was first introduced, it became a [fad](http://en.wikipedia.org/wiki/Fad). Peak production was over 170,000 Super Balls per day. By December 1965 over six million had been sold, and US Presidential adviser [McGeorge Bundy](http://en.wikipedia.org/wiki/McGeorge_Bundy) had five dozen Super Balls shipped to the [White House](http://en.wikipedia.org/wiki/White_House) for the amusement of the staff. Knowing that fads are often short-lived, Wham-O Executive Vice-president Richard P. Kerr said, "Each Super Ball bounce is 92% as high as the last. If our sales don't come down any faster than that, we've got it made." Initially the full size Super Ball sold for ninety-eight cents at retail; by the end of 1966 its colorful miniature versions sold for as little as ten cents in vending machines.

In the late 1960s Wham-O made a "giant" Super Ball, roughly the size of a bowling ball, as a promotional stunt. It fell from the 23rd story window of an Australian hotel (or some reports say, from the roof) and destroyed a parked convertible car on the second bounce.

## Physical properties

According to one study "If a pen is stuck in a hard rubber ball and dropped from a certain height, the pen may bounce to several times that height." If a Super Ball is dropped without spin onto a hard surface, with a small ball bearing on top of the Super Ball, the bearing rebounds to a great height.

High school physics teachers use Super Balls to educate students on usual and unusual models of impacts.

The "rough" nature of a Super Ball makes its impact characteristics different from otherwise similar smooth balls. The resulting behavior is quite complex. The Super Ball has been used as an illustration of the principle of [Time Reversal Invariance](http://en.wikipedia.org/wiki/T-symmetry).

A Super Ball is observed to reverse the direction of spin on each bounce. This effect depends on the tangential compliance and frictional effect in the collision, it cannot be explained by rigid body impact theory, and would not occur were the ball perfectly rigid.(Tangential compliance is the degree to which one body clings to rather than slips over another at the point of impact.)

**CHAPTER 3**

**SYSTEM SPECIFICATION**

**3.1 HARDWARE REQUIREMENTS**

The Hardware requirements are very minimal and the program can be run on most of the machines.

* Processor - Intel Pentium 4 processor or above
* Processor Speed - 500 MHz or above
* RAM - 64MB or above
* Storage Space - 2 MB or above
* Monitor Resolution - 1024\*768 or 1336\*768 or 1280\*1024 or better
* Mouse
* Keyboard

**3.2 SOFTWARE REQUIREMENTS**

The software requirements are.

Operating System - Windows / LINUX

* IDE - Dev C++ / Microsoft Visual C++ (version 6 or higher)
* OpenGL libraries

**Header Files:**

1. glut.h

**Object File Libraries:**

1. glu32.lib
2. opengl32.lib
3. glut32.lib

**DLL files (Dynamic Link Libraries):**

1. glu32.dll
2. glut32.dll
3. opengl32.dll

**The programming is done in Visual Studio 10 using visual C++.**

**CHAPTER 4**

**SOFTWARE DESCRIPTION**

Today computers and computer-generated images touch many aspects of our daily life. This imagery is used in 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.

In 1959, the TX-2 computer was developed at - MIT's Lincoln Laboratory. A light pen could be used to draw sketches on the computer using Ivan Sutherland's revolutionary Sketchpad software.IBM released first commercial graphics computer called IBM2250.Many of the most important early breakthroughs in computer graphics research occurred at the University of Utah in the 1970s.

In the 1980s, artists and graphic designers began to see the personal computer, particularly the Commodore Amiga and Macintosh, as a serious design tool, one that could save time and draw more accurately than other methods. In the late 1980s, SGI computers were used to create some of the first fully computer-generated short films at Pixar. The Macintosh remains a highly popular tool for computer graphics among graphic design studios and businesses. Modern computers, dating from the 1980s often use graphical user interfaces (GUI) to present data and information with symbols, icons and pictures, rather than text. Graphics are one of the five key elements of multimediatechnology. 3D Graphics became more popular in 1990 in gaming and multimedia. Quakewas the first 3D game released and Toy Story was first full length computer generated animation movie.

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 toillustrate papers, reports, theses, and other presentation material.

**PRINCIPLES AND CONCEPTS**

The common principles used in computer graphics are as shown below:

### a)Graphics

Graphics are visual presentations on some surface, such as a wall, canvas, computer screen paper, or stone brand, to inform, illustrate, or entertain. Examples are photographs, drawings, line art, graphs, diagrams, typography, numbers, symbols, geometric designs, maps, engineering drawings, or other images. Graphics often combine text, illustration, and color. Graphic design may consist of the deliberate selection, creation, or arrangement of typography alone, as in a brochure, flier, poster, web site, or book without any other element. Clarity or effective communication may be the objective, association with other cultural elements may be sought, or merely, the creation of a distinctive style.

**b)Rendering**

Rendering is the process of generating an image from a model, by means of computer programs. The model is a description of three dimensional objects in a strictly defined language or data structure. It would contain geometry, viewpoint, texture, lighting, and shading information. The image is a digital image or raster graphics image.

The term may be by analogy with an "artist's rendering" of a scene. **Rendering** is also used to describe the process of calculating effects in a video editing file to produce final video output.

**c) Shading**

Shading refers to depicting depth in 3D models or illustrations by varying levels of darkness. It is a process used in drawing for depicting levels of darkness on paper by applying media more densely or with a darker shade for darker areas, and less densely or with a lighter shade for lighter area

**APPLICATIONS OF COMPUTER GRAPHICS**

The development of computer graphics has driven both by the needs of the user community and by advances in hardware and software. The applications of computer graphicsare:

.

**a)Design**

* Design is an iterative process.
* Design problems are either over determined, such that they process no solution that satisfies all the criteria, or underdetermined such that they have multiple solutions to satisfy the criteria.
* Designer works iteratively to generate possible design, tests it, then uses the results as the basis for exploring other solutions.

**b) Simulation and animation**

* Engineers and researchers began to use graphics systems as simulators.
* Graphical flight simulators have proved to increase safety and to reduce training expenses.
* Games and education software for home computers are almost as impressive.

**c)User interface**

* Interactions with computers have become dominated by visual paradigm that includes windows, icons, menus and pointing devices.
* Many users have become internet users.
* From user’s perspective, windowing system such as the X Window system, Microsoft Windows and Macintosh OS X differ only in detail.

**4.1FRONT END**

OpenGL(Open Graphics Library) was developed by **‘Silicon Graphics Inc‘**(SGI) on 1992 and is popular in the gaming industry where it competes with the Direct3D in the Microsoft Windows platform. OpenGL is broadly used in CAD(Computer Aided Design),virtual reality, scientific visualization, information visualization, flight simulation and video games development.

OpenGL is a standard specification that defines an API that is multi-language and multi-platform and that enables the codification of applications that output computerized graphics in 2D and 3D.

The interface consists in more than 250 different functions, which can be used to draw complex tridimensional scenes with simple primitives. It consist of many functions that help to create an real world object and an particular existence for an object can be given.

The OpenGL ARB released a series of manuals along with the specifications which have been updated to track changes in the API. These are universally known by the colors of their covers:

* The Red Book

OpenGL Programming Guide, 7th edition. A readable tutorial and reference book – this is a 'must have' book for OpenGL programmers.

* The Blue Book

OpenGL Reference manual, 4th edition. Essentially a hard-copy printout of the [man](http://en.wikipedia.org/wiki/Unix_manual) pages for OpenGL includes a poster-sized fold-out diagram showing the structure of an idealized OpenGL implementation.

* The Green Book

OpenGL Programming for the X Window System.A book about X11 interfacing and [GLUT](http://en.wikipedia.org/wiki/OpenGL_Utility_Toolkit).

* The Alpha Book (white cover)

OpenGL Programming for Windows 95 and Windows NT.A book about interfacing OpenGL with Microsoft Windows.

Then, for OpenGL 2.0 and beyond:

* The Orange Book

OpenGL Shading Language, 3rd edition.

* A readable tutorial and reference book .

**4.2FEATURES**

OpenGL serves two main purposes:

1. To hide the complexities of interfacing different 3D accelerators, by presenting the programmer with a single, uniform API.

2. To hide the different capabilities of hardware platforms,by requiring that all implementations support the full OpenGL feature set(using software emulation if necessary).

OpenGL is a low-level, procedural API, requiring the programmer to dictate the exact steps required to render a scene. This contrast with descriptive scene graph or retained mode APIs, where a programmer only needs to describe a scene and can let the library manage the details of rendering it. OpenGL’s low-level design requires programmers to have a good knowledge of the graphics pipeline, but also give a certain amount of implement novel rendering algorithms.

OpenGL has historically been influential on the development of 3D accelerators, promoting a base level of functionality that is now common in consumer-level-hardware.

Basic features of OpenGL are

* [Rasterised](http://en.wikipedia.org/wiki/Rasterisation) points, lines and polygons as basic [primitives](http://en.wikipedia.org/wiki/Geometric_primitive)
* A [transform and lighting](http://en.wikipedia.org/wiki/Transform_and_lighting) pipeline
* [Z-buffering](http://en.wikipedia.org/wiki/Z-buffering)
* [Texture mapping](http://en.wikipedia.org/wiki/Texture_mapping)
* [Alpha blending](http://en.wikipedia.org/wiki/Alpha_blending)

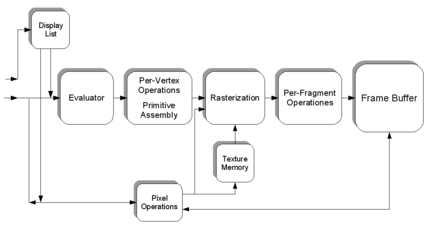


Fig 4.1: The graphics pipeline

A brief description of the process in the graphics pipeline could be:[[5]](http://en.wikipedia.org/wiki/OpenGL#cite_note-4)

1. Evaluation, if necessary, of the polynomial functions which define certain inputs, like [NURBS](http://en.wikipedia.org/wiki/Nonuniform_rational_B-spline) surfaces, approximating curves and the surface geometry.
2. Vertex operations, transforming and lighting them depending on their material. Also clipping non-visible parts of the scene in order to produce the viewing volume.
3. [Rasterisation](http://en.wikipedia.org/wiki/Rasterisation) or conversion of the previous information into pixels. The polygons are represented by the appropriate color by means of interpolation algorithms.
4. Per-fragment operations, like updating values depending on incoming and previously stored depth values, or color combinations, among others.
5. Lastly, fragments are inserted into the [frame buffer](http://en.wikipedia.org/wiki/Framebuffer).

Many modern 3D accelerators provide functionality far above this baseline, but these new features are generally enhancements of this basic pipeline rather than radical revisions of it.

**4.2.1 OpenGL Interface**

OpenGL directory through functions in three libraries, **GL,GLU,GLUT**. Functions in the main GL library have names that begins with the letters gl and stored in a library referred to as GL.

**4.2.2 OpenGL Utility Library(GLU)**

This library uses only GL functions but contains code for creating common objects and simplifying viewing.

**4.2.3 OpenGL utility tool kit(GLUT)**

It provides the minimum functionality that should be expected in any modern window system.

**4.2.4Transformation Function**

Transformation functions that allows to carry out transformation of objects such as Translations, Rotation, Scaling.

**Translation**

Translation is an operation that displays points by fixed distances in a given direction. Translation displacement vector d transformed points are given by P’=P+d for all points P on the object

**Rotation**

Rotation is more difficult to specify than translation because we must specify more parameters. We start the simple rotation of point about the origin in 2D plane. Rotations and translations are known as rigid body transformations.

**Scaling**

Scaling is an affine non-rigid body transformation by which we can maximize or miniaturize an object. We need a non-uniform scaling in all direction and scaling in one direction

**CHAPTER 5**

**PROJECT DESCRIPTON**

**5.1 OVERVIEW OF PROJECT**

The designing of the game, bouncing ball is explained here along with all the standard C library functions used and the entire user defined functions. Here the bouncing ball project is executed by using logic ie.,1. when the base is static the ball is jumping. 2.when the base is rotating the ball is static. 3. when the base is rotating the ball will also jump. 4. when the base is static the ball is also static. Each function is explained in detail with respect to the project design as follows.

**5.1.1 standard C-graphics functions**

**initgraph()**

initgraph initializes the graphics system by loading a graphics driver from disk (or validating a registered driver) then putting the system into graphics mode.

Declaration: **void far initgraph(int far \*graphdriver, int far \*graphmouse**

**closegraph()**

closegraphdeallocates all memory allocated by the graphics system.It then restores the screen to the mode it was in before you called initgraph.

Declaration:  **void far closegraph(void);**

**cleardevice()**

cleardevice erases the entire graphics screen and moves the current

position to home (0,0).

Declaration:  **void far cleardevice(void);**

**clearviewport()**

clearviewport erases the viewport and moves the current position to home (0,0), relative to the viewport.

Declaration: **void far clearviewport(void);**

**setcolor()**

sets the current drawing color to color, which can range from 0 to getmaxcolor.

Declaration: **void far setcolor(int color);**

**getcolor()**

getcolor returns the current drawing color.

Declaration: **int far getcolor(void);**

**getimage()**

getimage saves a bit image of the specified region into memory

Declaration: **void far getimage(int left, int top, int right, int bottom,**

**void far \*bitmap);**

**putimage()**

putimage puts the bit image previously saved with getimage back onto the screen, with the upper left corner of the image placed at (left,top).

Declaration: **void far putimage(int left, int top, void far \*bitmap, int op);**

**line()**

line draws a line between two specified points

Declaration: **void far line(int x1, int y1, int x2, int y2);**

**getkey()**

This function gets the value of the key that has been pressed. These values are stored in AL and AH registers.

Declaration: **void getkey();**

**initgraphics()**

This function initializes the graphics mode and if some error occurs in between suitable action is taken.

Declaration: **intinitgraphics();**

**askcontinue()**

When user has finished the game or quits the game this function is invoked asking the user whether he want to play again.

The design part of the game deals with how the game has divided and designed according to the interaction and implementation

Each subsystem provides its own functionalities independent of the services provided by the other subsystem. Each subsystem then further divided into modules which explain the subsystem in detail.

The call-return control model is used to represent the control relationship between the subsystems, which is shown below

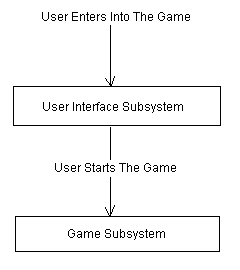


Fig5.1: Relation between subsystems

The modular decomposition of each subsystem is explained below.

**5.2 MODULE DESCRIPTION**

**5.2.1 Translation**

**Translation** is an operation that displaces points by a fixed distance in a given direction. Translation displacement vector d transformed points are given by P’= P+d. for all points

**a.**

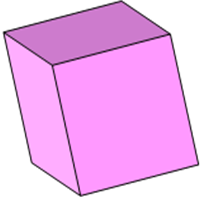


Fig 5.2.1a: object in original position

**b**.

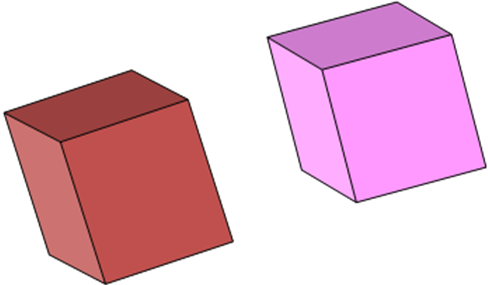


Fig5.2.1b: After translation

**5.2.2 Rotation**

**Rotation** is more difficult to specify than translation because we must specify more parameters. We start the simple rotation of point about the origin in a two dimensional plane. Rotations and translations are known as **rigid body transformations.**

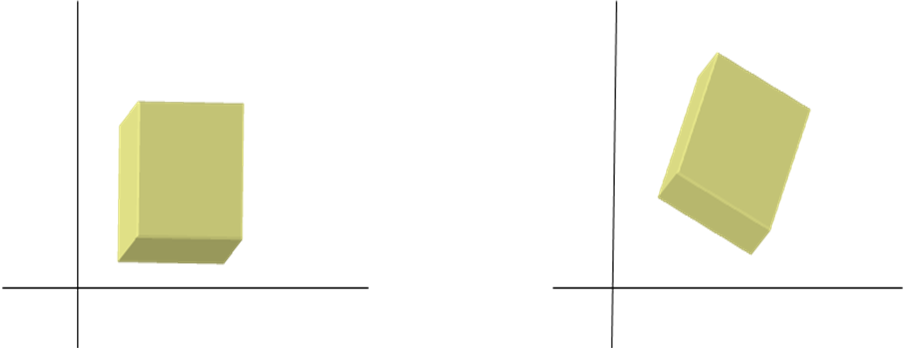


Fig 5.2.2a: [Before rotation] Fig 5.2.2b [After rotation]

**5.2.3 Scaling**

**Scaling** is an affine non rigid body transformation by which we can make an object bigger or smaller. We need a non uniform scaling in all direction and scaling in a single direction. Transformation that we use in modeling and viewing by combining a properly chosen sequence of scaling, translation and rotation.

Fig 5.2.3: [Non-rigid body transformation]

**5.2.4 User Interface Subsystem**

Aim of this subsystem is to provide the user with good interface. This subsystem is decomposed into following interconnected modules.

1. Introduction module
2. Score display module
3. Quit module

The control model is used to represent the control relationship between the subsystems, which is shown below.

When the user enters into the game an introduction screen will be displayed and he will be given choices to either see high scores or start the game or edit number of bullets or quit the game. If user decides to see high scores the score display module will be called and again the introduction screen will be displayed. If the user wants to edit the number of bullets to be used in the game, then editbullet module is called which takes care of this operation. If user decides to play the game then control will be transferred to the game subsystem. If the user wants to quit from the game then quit module will be called.

**5.3 DATA FLOW DIAGRAM**

The Flow Diagram below shows the various functions involved in the design of the bouncing ball program. It shows the transfer of control from one function to another.

int

main

Handle key

rotation

Keyboard callback

Mouse callback

Display module

Draw

Drawbackground

coloring

Matrix mode

Fig 5.3:Data flow diagram.

.

**5.4 INPUT DESIGN**

we will pass the arguments to the stack using **glutinit(&argc,argv);**

If we want to write**glutdouble** means when merging it is used.

To get clear image we will use func**GLUT\_RGB**

to interact between windowing and hidden surface remover we use the func**GLUT\_DEPTH;**

Graphics are sent to the screen through a func called **DISPLAY** callback. Here the func name is **FUNC**.

Registers the keyboard callback func f. we use the syntax for this func is void \*f(char key,intwidth,int height)

Registers the MOUSE callback func f. we use the syntax for this func is void \*f(intbutton,intstate,intx,int y)

**5.5 OUTPUT DESIGN**

we are pushing the content of matrix to the stack corresponding to the current matrix mode using **glPushMatrix();**

we are creating sphere by using **#define sphereid 1.**

We are creating box in the base by using func**#define boxid 2**

We are defining plane surface by using the function

**CHAPTER 6**

**CONCLUSION**

* The project consists of bouncing ball will show how ball will bounce over rotating surface
* It will tell how ball will bounce keeping base static
* It will tell how base will rotate keeping ball static
* The project gives an application to the shapes and motion in computer graphics.

**CHAPTER 7**

**FUTURE ENHANCEMENT**

The current game project has immense scope for improvement in many areas. Time constraint has been a major factor in limiting the features being offered in this game. The following features could have been incorporated:

* Mouse interface
* Resizing windows
* Saving the game.
* Better backgrounds
* Choosing color
* Sound implementation
* Changing the control keys

**APPENDIX**

**A.1 SOURCE CODE**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

#include <glut.h>

void Display(void);

void CreateEnvironment(void);

void MakeGeometry(void);

void MakeLighting(void);

void MakeCamera(int,int,int);

void HandleKeyboard(unsigned char key,int x, int y);

void HandleSpecialKeyboard(intkey,int x, int y);

void HandleMouse(int,int,int,int);

void HandleMainMenu(int);

void HandleSpeedMenu(int);

void HandleVisibility(intvis);

void HandleIdle(void);

void DrawTextXY(double,double,double,double,char \*);

void GiveUsage(char \*);

#define TRUE 1

#define FALSE 0

#define PI 3.141592653589793238462643

#define DRAFT 0

#define MEDIUM 1

#define BEST 2

intdrawquality = DRAFT;

intspincamera = TRUE;

intcameradirection = 1;

double updownrotate = 60;

intballbounce = TRUE;

double ballspeed = 2;

#define SPHEREID 1

#define BOXID 2

#define PLANEID 3

#define TEXTID 4

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

{

//inti,j,depth;

intmainmenu,speedmenu;

inti;

for (i=1;i<argc;i++) {

if (strstr(argv[i],"-h") != NULL)

GiveUsage(argv[0]);

if (strstr(argv[i],"-q") != NULL) {

if (i+1 >= argc)

GiveUsage(argv[0]);

drawquality = atoi(argv[i+1]);

if (drawquality< DRAFT)

drawquality = DRAFT;

if (drawquality> BEST)

drawquality = BEST;

i++;

}

}

/\* Set things up and go \*/

glutInit(&argc,argv);

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

glutCreateWindow("BOUNCING A BALL OVER A ROTATING A SURFACE");

glutDisplayFunc(Display);

glutVisibilityFunc(HandleVisibility);

glutKeyboardFunc(HandleKeyboard);

glutSpecialFunc(HandleSpecialKeyboard);

glutMouseFunc(HandleMouse);

CreateEnvironment();

/\* Set up some menus \*/

speedmenu = glutCreateMenu(HandleSpeedMenu);

glutAddMenuEntry("Slow",1);

glutAddMenuEntry("Medium",2);

glutAddMenuEntry("fast",3);

mainmenu = glutCreateMenu(HandleMainMenu);

glutAddMenuEntry("Toggle camera spin",1);

glutAddMenuEntry("Toggle ball bounce",2);

glutAddSubMenu("Ball speed",speedmenu);

glutAddMenuEntry("Quit",100);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

glutMainLoop();

return(0);

}

/\* This is where global settings are made, that is,

things that will not change in time \*/

void CreateEnvironment(void)

{

glEnable(GL\_DEPTH\_TEST);

if (drawquality == DRAFT) {

glShadeModel(GL\_FLAT);

}

if (drawquality == MEDIUM) {

glShadeModel(GL\_SMOOTH);

}

if (drawquality == BEST) {

glEnable(GL\_LINE\_SMOOTH);

glEnable(GL\_POINT\_SMOOTH);

glEnable(GL\_POLYGON\_SMOOTH);

glShadeModel(GL\_SMOOTH);

glDisable(GL\_DITHER); /\* Assume RGBA capabilities \*/

}

glLineWidth(1.0);

glPointSize(1.0);

glPolygonMode(GL\_FRONT\_AND\_BACK,GL\_FILL);

glFrontFace(GL\_CW);

glDisable(GL\_CULL\_FACE);

glClearColor(0.0,0.0,0.0,0.0); /\* Background colour \*/

glEnable(GL\_COLOR\_MATERIAL);

}

/\* This is the basic display callback routine

It creates the geometry, lighting, and viewing position

In this case it rotates the camera around the scene\*/

void Display(void)

{

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

glPushMatrix();

MakeCamera(FALSE,0,0);

MakeLighting();

MakeGeometry();

glPopMatrix();

/\* glFlush(); This isn't necessary for double buffers \*/

glutSwapBuffers();

}

/\* Create the geometry\*/

void MakeGeometry(void)

{

// inti;

double radius = 0.5;

static double theta = 0;

GLfloat mshin1[] = {5.0}; /\* For the sphere \*/

GLfloat mspec1[] = {0.5,0.5,0.5,1.0};

GLfloat mdiff1[] = {0.6,0.0,0.6,1.0};

GLfloat mamb1[] = {0.1,0.0,0.1,1.0};

GLfloat mdiff2[] = {0.0,1.0,0.0,1.0}; /\* Green plane \*/

GLfloat mamb2[] = {0.0,0.2,0.0,1.0};

GLfloat mdiff3[] = {0.5,0.5,0.5,1.0}; /\* Grey boxes \*/

GLfloat mamb3[] = {0.2,0.2,0.2,1.0};

/\* Create a green ground plane \*/

glLoadName(PLANEID);

if (drawquality> DRAFT)

glBegin(GL\_POLYGON);

else

glBegin(GL\_POLYGON);

glColor3f(0.0,1.0,1.0);

if (drawquality> DRAFT) {

glMaterialfv(GL\_FRONT\_AND\_BACK,GL\_DIFFUSE,mdiff2);

glMaterialfv(GL\_FRONT\_AND\_BACK,GL\_AMBIENT,mamb2);

}

glNormal3f( 0.0, 1.0, 0.0);

glVertex3f( 2.0, 0.0, 2.0);

glVertex3f( 2.0, 0.0,-2.0);

glVertex3f(-2.0, 0.0,-2.0);

glVertex3f(-2.0, 0.0, 2.0);

glEnd();

/\* Place a few grey boxes around the place \*/

glLoadName(BOXID);

glColor3f(1.0,0.0,1.0);

if (drawquality> DRAFT) {

glMaterialfv(GL\_FRONT\_AND\_BACK,GL\_DIFFUSE,mdiff3);

glMaterialfv(GL\_FRONT\_AND\_BACK,GL\_AMBIENT,mamb3);

}

glPushMatrix();

glTranslatef(1.8,0.2,1.8);

if (drawquality> DRAFT)

glutSolidCube(0.4);

else

glutWireCube(0.4);

glTranslatef(-3.6,0.0,0.0);

if (drawquality> DRAFT)

glutSolidCube(0.4);

else

glutWireCube(0.4);

glTranslatef(0.0,0.0,-3.6); //modification for first left box

if (drawquality> DRAFT)

glutSolidCube(0.4);

else

glutWireCube(0.4);

glTranslatef(3.6,0.0,0.0); //for second right box

if (drawquality> DRAFT)

glutSolidCube(0.4);

else

glutWireCube(0.4);

glPopMatrix();

/\* Create a magenta bouncing ball \*/

glLoadName(SPHEREID);

glColor3f(0.6,0.0,0.6);

if (drawquality> DRAFT) {

glMaterialfv(GL\_FRONT,GL\_SHININESS,mshin1);

glMaterialfv(GL\_FRONT,GL\_SPECULAR,mspec1);

}

glPushMatrix();

glTranslatef(0.0,radius+0.5\*(1+sin(PI\*theta/180)),0.0);

glScalef(radius,radius,radius);

if (ballbounce)

theta += ballspeed;

if (drawquality> DRAFT)

glutSolidSphere(1.0,16,16);

else

glutWireSphere(1.0,32,32);

glPopMatrix();

glColor3f(1.0,1.0,.0);

/\* Write some text \*/

glLoadName(TEXTID);

DrawTextXY(-2.0,-0.25,2.0,0.001,"An OpenGL Miniprojectby:Madhushree.N.V and Bhanupriya.B.S");

}

/\* Set up the lighing environment\*/

void MakeLighting(void)

{

GLfloatglobalambient[] = {0.3,0.3,0.3,1.0};

/\* The specifications for 3 light sources \*/

GLfloat pos0[] = {1.0,1.0,1.0,0.0}; /\* w = 0 == infinite distance \*/

GLfloat dif0[] = {0.8,0.8,0.8,1.0};

GLfloat pos1[] = {5.0,-5.0,0.0,0.0}; /\* Light from below \*/

GLfloat dif1[] = {0.4,0.4,0.4,1.0}; /\* Fainter \*/

if (drawquality> DRAFT) {

/\* Set ambient globally, default ambient for light sources is 0 \*/

glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT,globalambient);

glLightfv(GL\_LIGHT0,GL\_POSITION,pos0);

glLightfv(GL\_LIGHT0,GL\_DIFFUSE,dif0);

glLightfv(GL\_LIGHT1,GL\_POSITION,pos1);

glLightfv(GL\_LIGHT1,GL\_DIFFUSE,dif1);

glEnable(GL\_LIGHT0);

glEnable(GL\_LIGHT1);

glEnable(GL\_LIGHTING);

}

}

/\* Set up the camera optionally creating a small viewport about the mouse click point for

object selection \*/

void MakeCamera(intpickmode,intx,int y)

{

static double theta = 0;

GLint viewport[4];

/\* Camera setup \*/

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

if (pickmode == TRUE) {

glGetIntegerv(GL\_VIEWPORT,viewport); /\* Get the viewport bounds \*/

gluPickMatrix(x,viewport[3]-y,3.0,3.0,viewport);

}

gluPerspective(70.0, /\* Field of view \*/

1.0, /\* aspect ratio \*/

0.1,1000.0); /\* near and far \*/

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

gluLookAt(5\*cos(theta\*PI/180)\*sin(updownrotate\*PI/180),

5\*cos(updownrotate\*PI/180),

5\*sin(theta\*PI/180)\*sin(updownrotate\*PI/180),

0.0,0.0,0.0, /\* Focus \*/

0.0,1.0,0.0); /\* Up \*/

if (spincamera)

theta += (cameradirection \* 0.2);

}

/\* Deal with plain key strokes\*/

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

{

switch (key) {

case 27: /\* ESC \*/

case 'Q':

case 'q': exit(0); break;

case 's':

case 'S': spincamera = !spincamera; break;

case 'b':

case 'B': ballbounce = !ballbounce; break;

}

}

/\* Deal with special key strokes \*/

void HandleSpecialKeyboard(intkey,int x, int y)

{

switch (key) {

case GLUT\_KEY\_LEFT: cameradirection = -1; break;

case GLUT\_KEY\_RIGHT: cameradirection = 1; break;

case GLUT\_KEY\_UP: updownrotate -= 2; break;

case GLUT\_KEY\_DOWN: updownrotate += 2; break;

}

}

/\* Handle mouse events\*/

void HandleMouse(intbutton,intstate,intx,int y)

{

inti,maxselect = 100,nhits = 0;

GLuintselectlist[100];

if (state == GLUT\_DOWN) {

glSelectBuffer(maxselect,selectlist);

glRenderMode(GL\_SELECT);

glInitNames();

glPushName(-1);

glPushMatrix();

MakeCamera(TRUE,x,y);

MakeGeometry();

glPopMatrix();

nhits = glRenderMode(GL\_RENDER);

if (button == GLUT\_LEFT\_BUTTON) {

} else if (button == GLUT\_MIDDLE\_BUTTON) {

} /\* Right button events are passed to menu handlers \*/

if (nhits == -1)

fprintf(stderr,"Select buffer overflow\n");

if (nhits> 0) {

fprintf(stderr,"\tPicked %d objects: ",nhits);

for (i=0;i<nhits;i++)

fprintf(stderr,"%d ",selectlist[4\*i+3]);

fprintf(stderr,"\n");

}

}

}

/\* Handle the main menu\*/

void HandleMainMenu(intwhichone)

{

switch (whichone) {

case 1: spincamera = !spincamera; break;

case 2: ballbounce = !ballbounce; break;

case 100: exit(0); break;

}

}

/\* Handle the ball speed sub menu\*/

void HandleSpeedMenu(intwhichone)

{ switch (whichone) {

case 1: ballspeed = 0.5; break;

case 2: ballspeed = 2; break;

case 3: ballspeed = 10; break;

}

}

/\* How to handle visibility\*/

void HandleVisibility(int visible)

{ if (visible == GLUT\_VISIBLE)

glutIdleFunc(HandleIdle);

else

glutIdleFunc(NULL);

}

/\* What to do on an idle event\*/

void HandleIdle(void)

{ glutPostRedisplay();

}

/\* Draw text in the x-y plane

The x,y,z coordinate is the bottom left corner (looking down -ve z axis)\*/

void DrawTextXY(double x,doubley,doublez,doublescale,char \*s)

{ inti;

glPushMatrix();

glTranslatef(x,y,z);

glScalef(scale,scale,scale);

for (i=0;i<strlen(s);i++)

{

glutStrokeCharacter(GLUT\_STROKE\_ROMAN,s[i]);

}

glPopMatrix();

}

/\* Display the program usage information\*/

void GiveUsage(char \*cmd)

{ fprintf(stderr,"Usage: %s [-h] [-q n]\n",cmd);

fprintf(stderr," -h this text\n");

fprintf(stderr," -q n quality, 0,1,2\n");

fprintf(stderr,"Key Strokes and Menus:\n");

fprintf(stderr," q - quit\n");

fprintf(stderr," s - toggle camera spin\n");

fprintf(stderr," b - toggle ball bounce\n");

fprintf(stderr," left arrow - change rotation direction\n");

fprintf(stderr," right arrow - change rotation direction\n");

fprintf(stderr," down arrow - rotate camera down\n");

fprintf(stderr," up arrow - rotate camera up\n");

exit(-1);

}

**A.2 SCREEN SHOTS**

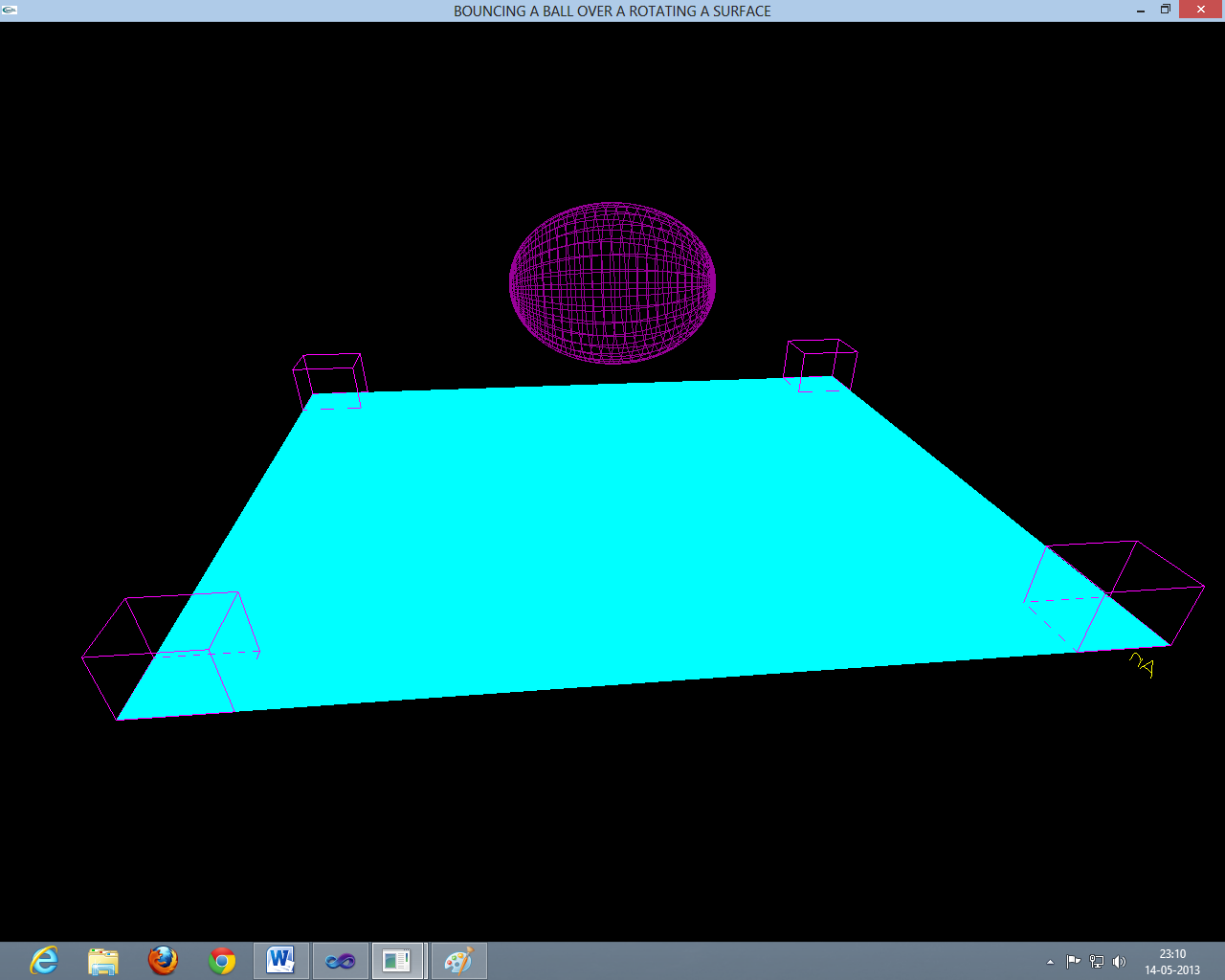


Fig A.1: bouncing ball over static surface

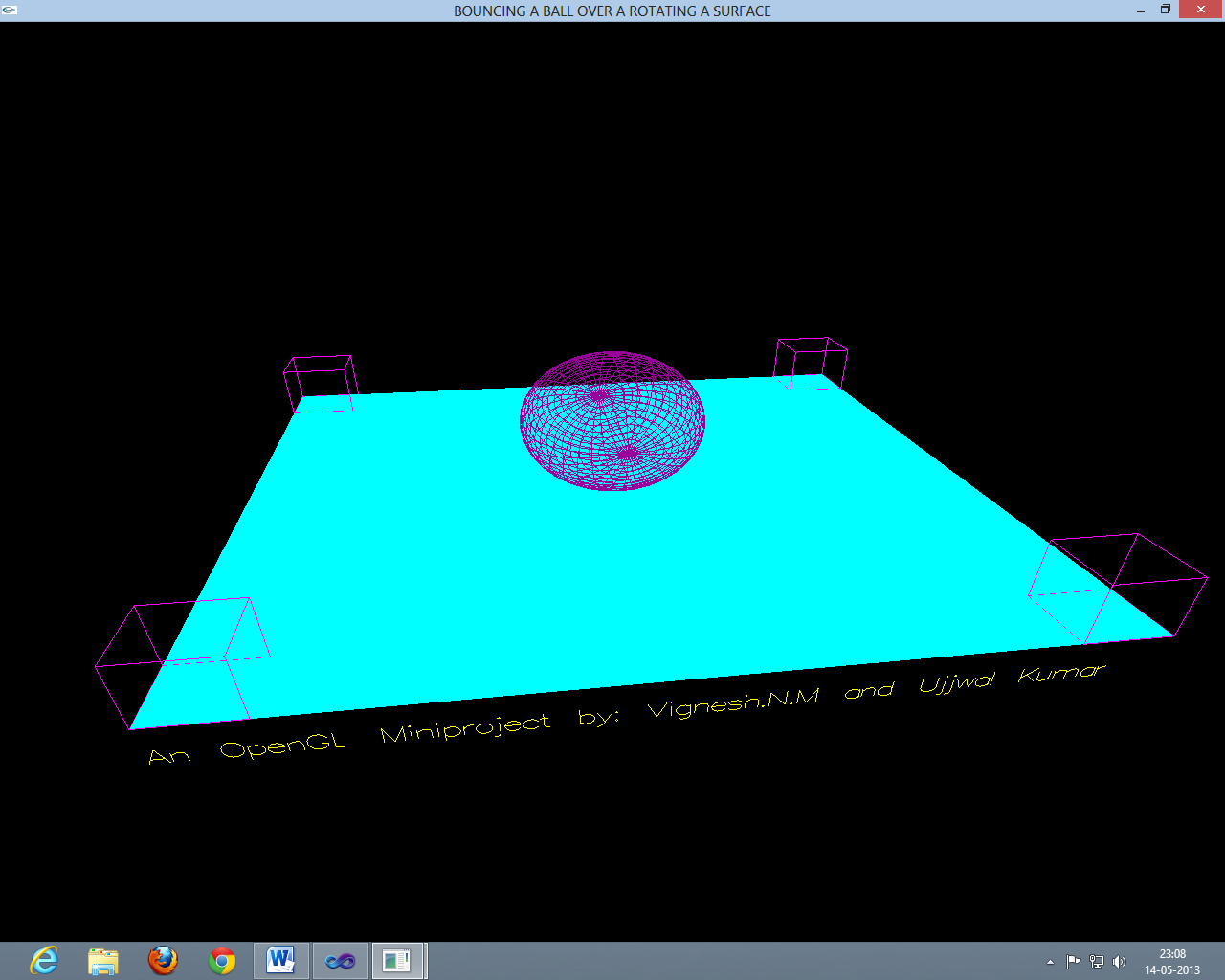


Fig A.2: bouncing ball over rotating surface

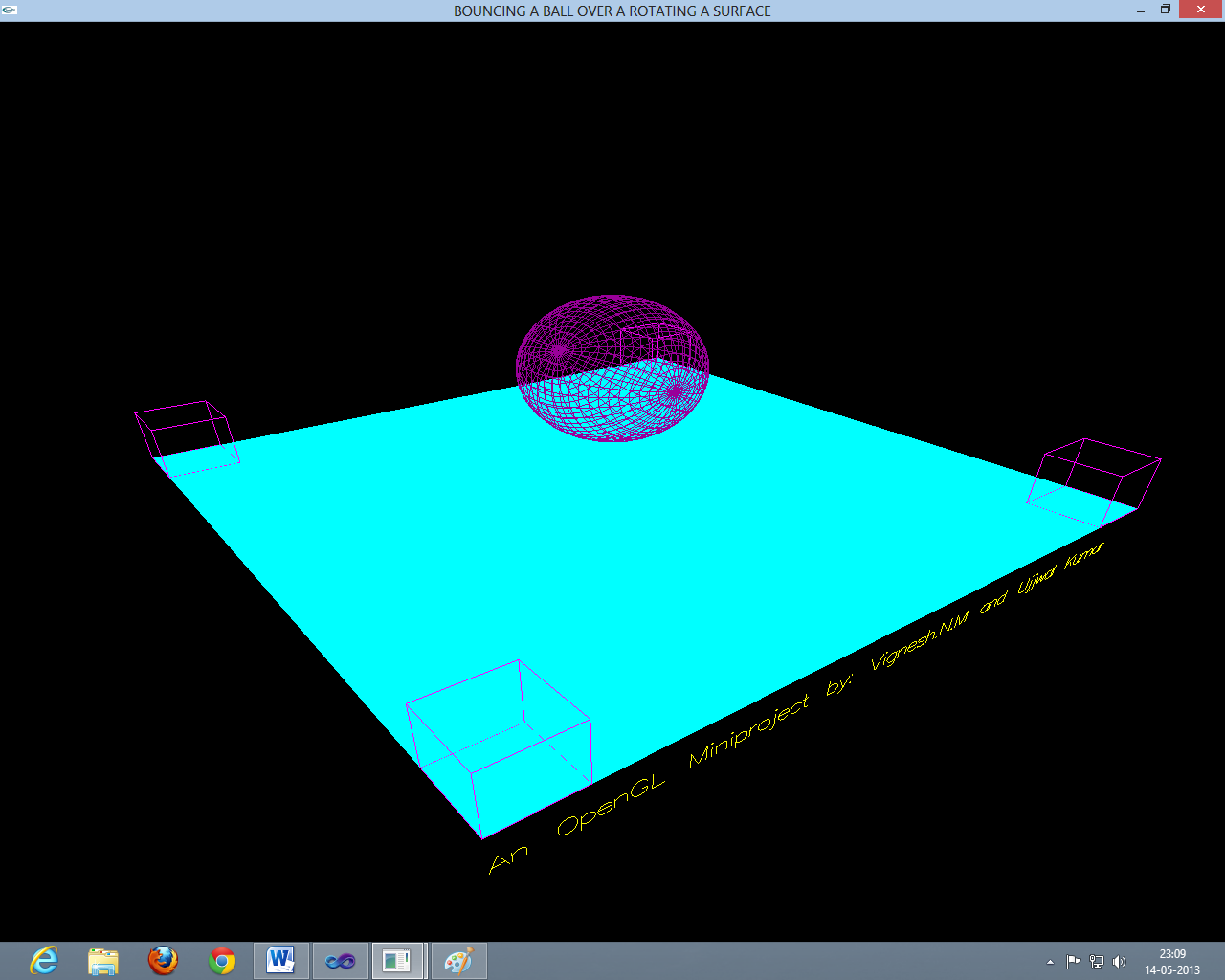


Fig A.3: static ball static surface

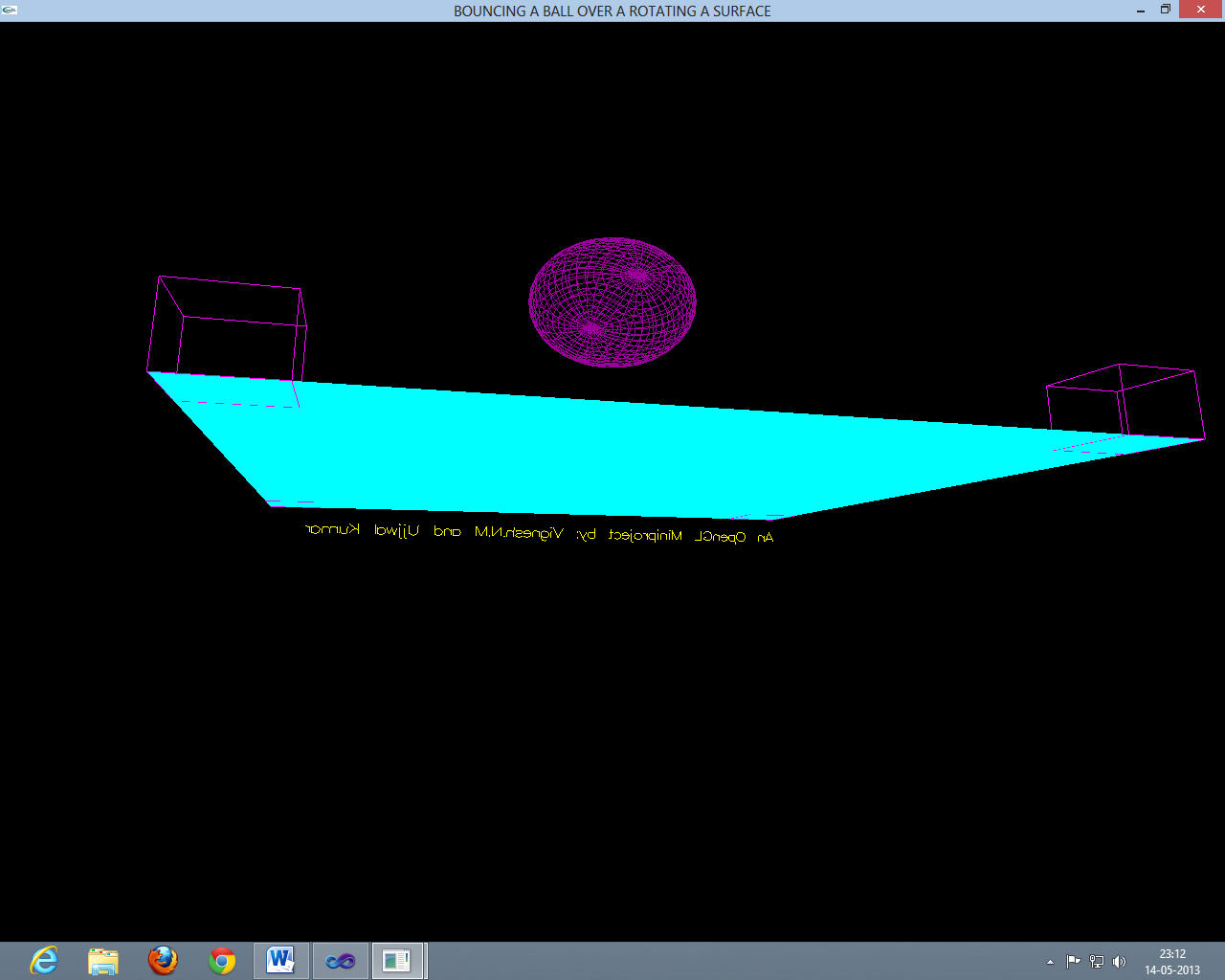


Fig A.4: 360º camera rotation capability

**REFERENCES**

**Reference Books and E-books**

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

[2]OpenGL Red Book and Blue Book for reference.

**Websites for Reference**

[3] [www.cprogramming.com/tutorial/opengl\_introduction.html](http://www.cprogramming.com/tutorial/opengl_introduction.html)

[4] [www.wikipedia.org](http://www.wikipedia.org)

[5] [www.**opengl**.org/resources/code/samples/redbook/](http://www.opengl.org/resources/code/samples/redbook/)

[6] www.opengl.org/sdk/docs/man/xhtml/