1. INTRODUCTION

1.1 Computer Graphics

Computer graphics are graphics created using computers and the representation of image data by a computer specifically with help from specialized graphic hardware and software. The interaction and understanding of computers and interpretation of data has been made easier because of computer graphics. A computer graphic development has had a significant impact on many types of media and has revolutionized animation, movies and the video game industry.

Typically, the term *computer graphics* refers to several different things:

- the representation and manipulation of image data by a computer
- the various technologies used to create and manipulate images
- the sub-field of computer science which studies methods for digitally synthesizing and manipulating visual content.

Computer generated imagery can be categorized into several different types: two dimensional (2D), three dimensional (3D), and animated graphics. As technology has improved, 3D computer graphics have become more common, but 2D computer graphics are still widely used. Computer graphics has emerged as a sub-field of computer science which studies methods for digitally synthesizing and manipulating visual content. Over the past decade, other specialized fields have been developed like information visualization, and scientific visualization more concerned with "the visualization of three dimensional phenomena (architectural, meteorological, medical, biological, etc.), where the emphasis is on realistic renderings of volumes, surfaces, illumination sources, and so forth, perhaps with a dynamic (time) component".

A major use of computer graphics is in design processes, particularly for engineering and architectural systems, but almost all products are now computer designed. Generally referred to as CAD, computer-aided design methods are now routinely used in the design of buildings, automobiles, aircraft, watercraft, Spacecraft, computers, textiles, and many, many other products.

1.2 OpenGL

As a software interface for graphics hardware, OpenGL's main purpose is to render two-dimensional and three-dimensional objects into frame buffer. These objects are described as sequences of vertices (which define geometric objects) or pixels (which define images).OpenGL performs several processing steps on this data to convert it to pixels to form the final described image in the frame buffer.

OpenGL bases on the state variables. There are many values, for example the color, that remain after being specified. That means, you can specify a color once and several polygons, lines or whatever with this color. There are no classes like DirectX. However, it is logically structured. Before we come to the commands themselves, here is another thing:

To be hardware independent, OpenGL provides its own data types. They all begin with "GL". For example: GLfloat, GLint and so on. There are also many symbolic constants , they all begin with "GL_", like GL_POINTS, GL_POLYGON. Finally the commands have the prefix "gl" like glVertex3f( ) . There is a utility library called GLU, here the prefixes are "GLU_" and "glu". GLUT commands begin with "glut", it is the same for every library.

1.3 GLUT

GLUT is a complete API written by Mark Kilgard which facilitates creation of windows and handling messages. It exists for several platforms, that means that a program which uses GLUT can be complied on many platform without (or at least with very few) changes in the code.

GLUT provides some routines for the initialization and cresting the window (or fullscreen mode). Those functions are called first in a GLUT application:

In the first line always *glutInit(&argc,argv)* is written. After this, GLUT must be told which display mode is required-single or double buffering, color index mode or RGB and so on. This is done by calling *glutInitDisplayMode().* The symbolic constants are connected by a logical OR, so you could use *glutInitDisplayMode(GLUT_RGB|GLUT_SINGLE).*

2. PROJECT DESCRIPTION

2.1 SOFTWARE AND HARDWARE SPECIFICATION

2.1.1 HARDWARE REQUIREMENTS

The standard output device is assumed to be a **Color Monitor.** It is quite essential for any graphics package to have this, as provision of color options to the user is a must. The **mouse**, the main input device, has to be functional i.e. used to move the pendulum left and right in the project. A **keyboard** is used to controlling and inputting data in the form characters. Apart from these hardware requirements there should be sufficient **hard disk** space and primary memory available for proper working of the package to execute the program. **Pentium III or** higher processor, 16MB or more **RAM**.

Minimum Requirements expected are cursor movement, creating objects like lines, squares, rectangles, polygon etc. Transformation on object/selected area should be possible. Filling of area with the specified color should be possible.

2.1.2 SOFTWARE REQUIREMENTS

Operating System : UBUNTU

Programming Language : C++

IDE : Code Blocks

Libraries : GLUT, SOIL

2.2 SYSTEM DESIGN

Platform:

The package is implemented using CodeBlocks, under the Ubuntu platform. OpenGL and associated toolkits are used for the package development.

Translation:

Translation is done by adding the required amount of translation quantities to each of the points of the objects in the selected area. If P(x , y) be a point and (tx , ty) be the translation quantities, then the translated point is given by glTranslatef(dx,dy,dz);

Rotation:

The rotation of an object by an angle ‘a’ is accomplished by rotating each of the points of the object. The rotated points can be obtained using the OpenGL function glRotatef(a,vx,vy,vz);

Scaling:

The scaling operation on an object can be carried out by multiplying each of the points (x, y) by the scaling factors sx,sy using the function glScalef(sx,sy,sz);

3. APIs USED

Initialization Functions

3.1 glutInit

glutInit is used to initialize the GLUT library.

Usage

void glutInit(int *argcp, char **argv);

argcp : A pointer to the program's unmodified argc variable from main. Upon return, the value pointed to by argcp will be updated, because glutInit extracts any command line options intended for the GLUT library.

argv : The program's unmodified argv variable from main. Like argcp, the data for argv will be updated because glutInit extracts any command line options understood by the GLUT library.

Description

glutInit will initialize the GLUT library and negotiate a session with the window system. During this process, glutInit may cause the termination of the GLUT program with an error message to the user if GLUT cannot be properly initialized. Examples of this situation include the failure to connect to the window system, the lack of window system support for OpenGL, and invalid command line options. glutInit also processes command line options, but the specific options parse are window system dependent.

3.2 glutInitWindowPosition , glutInitWindowSize

The glutInitWindowPosition and glutInitWindowSize functions specify a desired position and size for windows that *freeglut* will create in the future.

Usage

void glutInitWindowPosition(int x, int y);

void glutInitWindowSize(int width,int height);

Description

The glutInitWindowPostion and glutInitWindowSize functions specify a desired position and size for windows that *freeglut* will create in the furture. The position is measured in pixels from the upper left hand corner of the screen,with “x” increasing to right and “y” increasing towards to the bottom of the screen. The size is measured in *pixels*. *Freeglut* does not promise to follow these specification in create its windows, but it certainly makes am attempt to.

Event Processing Function

3.3 glutMainLoop

The glutMainLoop function enters the event loop

Usage

void glutMainLoop(void);

Description

The glutMainLoop function causes the program to enter the window event loop. An application should call this function at most once. It will call any application callback functions as required to process mouse clicks, mouse motion, key presses, and so on.

Window Function

3.4 glutCreateWindow

glutCreateWindow creates a top-level window.

Usage

int glutCreateWindow(char *name);

name : ASCII character string for use as window name**.**

Description

glutCreateWindow creates a top-level window. The name will be provided to the window system as the window’s name. The intent is that the window system will label the window with the name. Implicitly, the current window is set to the newly created window. Each created window has a unique associated OpenGL context. State changes to a window’s associated OpenGL context can be done immediately after the window is created. The display state of a window is initially for the window to be shown. But the window’s display state is not actually acted upon until glutMainLoop is entered. This means until glutMainLoop is called, rendering to a created window is ineffective because the window cannot yet be displayed. The value returned is a unique small integer identifier for the window. The range of allocated identifiers starts at one. This window identifier can be used when calling glutSetWindow.

Display Functions

3.5 glClear(GLbitfieldmask):

glClear takes a single argument that is the bitwise OR of several values indicating which buffer is to be cleared.

Mask:

Bitwise OR of masks that indicate the buffer to be cleared. The four masks are GL_COLOR_BUFFER_BIT, GL_DEPTH_BIT, GL_ACCUM_BUFFER_BIT, and GL_STENCIL_BUFFER_BIT.

3.6 glColor3f(GLfloat red,GLflaot green,GLflaot blue):

glColor sets a new three valued RGB color. Current color values are stored in floating-point format. Unsigned integer color components are linearly mapped to floating-point values such that the largest representable value maps to 1.0(full intensity), 0 maps to 0.0(zero intensity).

red : The new red values for the current color.

green : The new green value for current color.

blue : The new blue value for current color.

3.7 glClearColor(GLfloat red,GLfloat green,GLfloat blue):

glClearColor specifies the red, green, blue and alpha values used by glClear to clear the color buffers. Values specified by glClearColor are clamped to range[0,1].

red, green, blue, alpha : These values used when the color buffers are cleared. The initial values are all 0.

3.8 glVertex

Specify a vertex

Parameters: x, y, z, w

Specify x, y, z, w coordinates of a vertex.

Not all parameters are present in all forms of the command.

id glVertex3f(GLfloat z, GLfloat y, GLfloat z);

Description

glVertex commands are used within glBegin/glEnd pairs to specify points, lines and polygon vertices. The current color, normal, texture coordinates, and fog coordinates are associated with the vertex when glVertex is called. When x, y, z are specified, w defaults to 1.

3.9 glutPostRedisplay

glutPostRedisplay marks the *current window* as needing to be redisplayed.

Usage

void glutPostRedisplay(void);

Description

Mark the normal plane of *current window* as needing to be redisplayed. The next iteration through glutMainLoop, the window's display callback to redisplay the window's normal plane. Multiple calls to glutPostRedisplay before the next display callback opportunity generates only a single redisplay callback. glutPostRedisplay may be called within a window's display or overlay display callback to re-mark that window for redisplay.

3.10 glutSwapBuffers

glutSwapBuffers swaps the buffers of the *current window* if double buffered.

Usage

void glutSwapBuffers(void);

Description

Performs a buffer swap on the *layer in use* for the *current window*. Specifically, glutSwapBuffers promotes the contents of the back buffer of the *layer in use* of the *current window* to become the contents of the front buffer. The contents of the back buffer then become undefined. The update typically takes place during the vertical retrace of the monitor, rather than immediately after glutSwapbuffers is called.

An implicit glFlush is done by glutSwapBuffers before it returns. Subsequent OpenGL commands can be issued immediately after calling glutSwapBuffers, but are not executed until the buffer exchange is complete. If the *layer in use* is not double buffered, glutSwapBuffers has no effect.

Global Callback Registration Functions.

3.11 glutTimerFunc

glutTimerFunc registers a timer callback to be triggered in a specified number of milliseconds.

Usage

void glutTimerFunc(unsigned int msecs,void(*func)(int value),value);

Description

glutTimerFunc registers the timer callback *func* to be triggered in at least *msecs* milliseconds. The value parameter to the 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 timer 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 cancelling a registered callback. Instead, ignore a callback based on its value parameter when it is triggered.

Window-Specific Callback Registration Functions.

3.12 glutDisplayFunc

glutDisplayFunc sets the display callback for the *current window*.

Usage

void glutDisplayFunc(void(*func)(void));

The new display callback function.

Description

glutDisplayFunc sets the display callback for the *current window*. When GLUT determines that the normal plane for the window needs to be redisplayed, the display callback window is called. Before the callback, the *current window* is set to the window needing to be redisplayed and (if no overlay display callback is registered) the *layer in use* is set to the normal plane. The display callback I called with no parameters. The entire normal plane region should be redisplayed in response to the callback (this includes ancillary buffers if your program depends on their state).

3.13 glutKeyboardFunc

glutKeyboardFunc sets the keyboard callback for the *current window*.

Usage

void glutKeyboardFunc(void(*func)(unsigned char key, int x, int y));

The new Keyboard callback function.

Description

glutKeyboardfunc sets the keyboard callback for *current window*. When a user types into the window, each key press generating an ASCII character will generate a keyboard callback. The key callback parameter is the generated ASCII character. The state of modifier keys such as Shift cannot be determined directly; their only effect will be on the returned ASCII data. The x and y callback parameters indicate the mouse location in window relative coordinates when the key was pressed. When a new window is created, no keyboard callback is initially registered and ASCII key strokes in the window are ignored. Passing NULL to glutKeyboardFunc disables the generation of keyboard callbacks.

3.14 glutMouseFunc

glutMouseFunc sets the mouse callback for the *current window*.

Usage

void glutMouseFunc(void(*func)(int button, int state, int x, int y));

The new mouse callback function.

Description

glutMouseFunc sets the mouse callback for the *current window*. When a user presses and release mouse buttons in the window, each press and each release generates a mouse callback. The button parameter is one of GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON , or GLUT_RIGHT_BUTTON.

3.15 glMatrixMode (GLenum mode):

It switches matrix mode between the two matrices-

For example : glMatrixMode(GL_MODELVIEW)

3.16 glViewport (GLint x, GLint y,GLsizei width,GLsizei height):

It specifies that the view port will have lower corner(x,y) in screen coordinates and will be width pixels wide and height pixels high.

3.17 glOrtho

Multiply the current matrix with an orthographic matrix.

Usage

Void glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble nearVal, GLdouble farVal);

Parameters

left , *right* : Specify the coordinates for the left and right vertical clipping planes.

bottom , *top* : Specify the coordinates for the bottom and top horizontal clipping planes.

nearVal , *farVal* : Specify the distances to the nearer and farther depth clipping planes. These values are negative if the plane is to be behind the viewer.

Description

glOrtho describes a transformation that produces a parallel projection. glOrtho describes transformation that produces a parallel projection. The current matrix glMatrixMode is multiplied by this matrix and result replaces the current matrix.

3.18 glLoadIdentity

glLoadIdentity — replaces the current matrix with the identity matrix

Usage

void glLoadIdentity(void);

Description

glLoadIdentity replaces the *current matrix* with the identity matrix. It is semantically equivalent to calling glLoadMatrix with the identity matrix.

3.19 glFlush

glFlush - force execution of GL commands in finite time.

Usage

void glFlush(void);

Descritption

Different GL implementations buffer commands in several different locations, including network buffers and the graphics accelerator itself. glFlush empties all of these buffers, causing all issued commands to be executed as quickly as they are accepted by the actual rendering engine. Though this execution may not be completed in any particular time period, it does complete in finite time.

3.20 glPushMatrix

glPushMatrix push the current matrix stack.

Usage

void glPushMatrix(void);

Description

glPushMatrix pushes the *current matrix* stack down by one, duplicating the current matrix. That is, after a glPushMatrix call, the matrix on top of the stack is identical. Initially each stack contains one matrix an identity matrix.

3.21 glPopMatrix

glPopMatrix pops the current matrix stack

Usage

void glPopMatrix(void);

Description

glPopMatrix pops the current matrix stack, replacing the current matrix with the one below it on the stack.

Rendering Functions

3.22 glutSolidSphere

glutSolidSphere render a solid sphere respectively.

Usage

void glutSolidSphere(GL double radius, GLint slices, GLint stacks);

radius : The radius of the sphere.

slices : The number of subdivisions around the Z axis(similar to lines of longitude).

stacks : The number of subdivisions along the Z axis(similar to lines of latitude).

Description

Renders a sphere centered at the modelling coordinates origin of the specified radius. The sphere is subdivided around the Z axis into slices and along the Z axis into stacks.

3.23 glMaterial

glMaterial specify material parameters for the lighting model

Usage

void glMaterialfv(GLenum face, GLenum pname, const GLfloat *params);

faces : Specify which face are beinf updated.Must be one of GL_FRONT, GL_BACK, GL_FRONT_AND_BACK.

pname : Specify the material parameters of the face that is being updated. Must be one of GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR, GL_EMISSION, GL_SHININESS.

params : Specify a pointer to the value that pname will be set to.

Description

glMaterial assign values of material parameters. There are two matched sets of material parameters. One, the *front-facing* set, is used to shade points, lines, bitmaps, and all polygons (when two-sided lighting is disabled), or just front-facing polygons (when two-sided lighting is enabled). The other set, *back-facing*, is used to shade back-facing polygons only when two-sided lighting is enabled.

3.24 glutStrokeCharacter

glutStrokeCharacter renders a stroke character using OpenGL.

Usage

void glutStrokeCharacter(void *font, int character);

font : is the stroke font to use.

character : is the character to render (not confined to 8 bits).

3.25 gluLookAt

gluLookAt defines a viewing transformation.

Usage

void gluLookAt(GLdouble eyeX, GLdouble eyeY, GLdouble eyeZ, GLdouble centerX, GLdouble centerY, GLdouble centerZ, GLdouble upX, GLdouble upY, GLdouble upZ);

eyeX, eyeY, eyeZ : Specify the position of the eye point.

centerX, centerY, centerZ : Specify the position of the reference point.

upX, *upY*, *upZ* : Specify the direction of the *up* vector.

Description

gluLookAt creates a viewing matrix derived from an eye point, a reference point indicating the center of the scene, and an *UP* vector. The matrix maps the reference point to the negative *z* axis and the eye point to the origin. When a typical projection matrix is used, the center of the scene therefore maps to the center of the viewport. Similarly, the direction described by the *UP* vector projected onto the viewing plane is mapped to the positive *y* axis so that it points upward in the viewport. The *UP* vector must not be parallel to the line of sight from the eye point to the reference point.

Menu Management

3.26 glutCreateMenu

glutCreateMenu creates a new pop-up menu.

Usage

int glutCreateMenu(void(*func)(int value);

func : The callback function for the menu that is called when a menu entry from the menu is selected. The value passed to the callback is determined by the value for the selected menu entry.

Description

glutCreateMenu creates a new pop-up menu and returns a unique small integer identifier. The range of allocated identifiers starts at one. The menu identifier range is separate from the window identifier range. Implicitly, the *current menu* is set to the newly created menu.

3.27 glutAddMenuEntry

glutAddMenuEntry adds a menu entry to the bottom of the*current menu*.

Usage

void glutAddMenuEntry(char*name, int value);

name : ASCII character string to display in the menu entry.

value : value to return to the menu's callback function if the menu entry is selected.

Description

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.

3.28 SOIL

SOIL is a C library used primarily for uploading textures into OpenGL.

The various functions of this library used in our project is as follows-

1. **glBindTexture** lets you create or use a named texture.

   void glBindTexture(GLenum *target*, GLuint *texture*);

   This function binds the texture name to the target of the current active texture unit.

2. **SOIL_load_OGL_texture** is used to load an image directly as an OpenGL texture.

```
Tex_2d2 = SOIL_load_OGL_texture (
    "logo.png",                  //BMP, PNG, JPG,TGA, DDS, PSD
    SOIL_LOAD_AUTO,              //  flag which keeps all of
                                 // the loading settings set to default.
    SOIL_CREATE_NEW_ID,
    SOIL_FLAG_COMPRESS_TO_DXT   //  compress and upload
                                // the image as DXT1 or DXT5
);
```

3. **glTexCoord2f** is used to set the texture coordinates.

   glTexCoord2f(GLfloat *s*, GLfloat *t*);

4. **glTexEnvf** is used to control the texture environment for the current active texture.

void glTexEnvf(GLenum *target*, GLenum *pname*, GLfloat *param*);

target : specifies a texture environment

pname : specifies the symbolic name of a single-valued texture environment parameter

param : specifies a single symbolic constant

4. PROJECT IMPLEMENTATION

4.1 Function description

1. void initLighting ( ) : sets the light intensity, color and position.

2. void camera ( ) : set the camera position and usage of gluLookAt function.

3. void timer ( ) : checks the pause flag and sets the time_step_counter needed for animation.

4. void rk4_sys_integrator (int pendno, double g, double l, double alpha, double m, double theta0, double v0, int N, double ti, double tf, double **times, double **thetas, double **vs):

Uses Runge-Kutta method to calculate amplitude, time variation and angular velocity of the pendulum bobs.

5. void drawPendulum (int pendno, double l, double time, double theta, double v) :

Used to model the wooden frame, pendulum bobs and the strings that connect them. For every iteration of call to this method, the relative positions of the strings and the pendulum bobs are set.

6. void printString (const char *str, double x, double y, double size) :

Uses glutStrokeCharacter to print strings by translating and scaling them to the required coordinates and size respectively.

7. void display ( ) : Used to display the contents of the screen. Here we have a total of seven screens. Their functions are precisely explained below:

Screen 1: Sets the Camera position and displays the title of the project. It also displays the college logo along with the names of the students and the guides. In addition to this if the right mouse button it will display the menu which as:

1. Exit: which exit the screen.
2. Demo: which go to screen6.
3. Theory: which go to screen3.

Screen 2: Displays the images of theory and demo wherein, when clicked, goes to the respective pages. In addition to them, an exit button is also present which, when clicked, prompts a confirmation message. The user can either click Yes/No based on his/her wish to exit.

Screen 3: Displays the theory regarding the frame of pendulum. The screen consists of exit button, and left and right arrow images which, when clicked, goes to the previous /next page respectively.

Screen 4: Displays the instructions which aids in viewing the pendulum motion animation. As in the previous screen, it also consists of exit, left and right buttons.

Screen 5: Displays an image named 'Demo' which indicates that the next screen would display the animation of the pendulum. As in the previous screen, it also consists of exit, left and right buttons and when clicked on the right button, Screen 6 is displayed.

Screen 6: Displays the pendulum frame animation wherein the instructions are as follows:

'Spacebar' : pause/play animation

w : To move Up.

s: To move Down.

a: To move Right.

d: To move Left.

+: To zoom in.

-: To zoom out

Screen 7: Displays a 'Thank You' image which indicates end of the project.

8. void mouse (int button, int state, int x, int y) : This is a callback function for the mouse events.

9. void keyboard (unsigned char key, int x, int y) : This is a callback function for the keyboard events.

10. void init (void): Use to calculate the angular velocity, amplitude, time variation and increases the length of the pendulum.

11. void reshape (int w, int h): This function is a callback function for glutReshapeFunc API. The values *w, h* indicates the width and height of the window respectively.

12. void menu (int ch): This is a callback function for glutCreateMenu API. The value *ch* indicates which option has been selected.

5. SOURCE CODE

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <GL/glut.h>
#include <cstring>
#include "math.h"
#include<SOIL/SOIL.h>

using namespace std;

//Function declarations
void initgLOptions();
void initLighting();
void drawaxes();
void camera();
void timer();
void RK4_Method();
void drawPendulumModel();
void printString();
void display();
void reshape();
void exitBox();
void mouse();
void keyboard();
void init();
void menu();

//global vaiables declaration
int screen=1,exitflag=0;
GLuint tex_2d1;
const int ms_per_frame = 20;  // min ms per frame
int time_step_counter = -1, pause_flag = 1;
double **times, **thetas, **vs;

// pendulum parameters
int NumOfBobs=0; //Num of pendulums
double length[ ] = {0.2115, 0.2182, 0.2252, 0.2325, 0.2402, 0.2482, 0.2567, 0.265, 0.2751,
0.285, 0.2954, 0.3065, 0.3181, 0.3305, 0.3436}; //length of pendulum Bobs
double initial_length=5; //swings, smaller the number, larger the swing area
const double two_pi=2*3.1416;
```

```
//set lighting intensity and color
GLfloat qaAmbientLight[ ] = {0.2, 0.2, 0.2, 2.0};
GLfloat qaDiffuseLight[ ] = {0.8, 0.8, 0.8, 2.0};
GLfloat qaSpecularLight[ ] = {1.0, 1.0, 1.0, 2.0};
GLfloat qaLightPosition[ ] = {5, 10, 1, .1};

//colors
GLfloat black[ ] = {0.0, 0.0, 0.0, 1.0}; //Black Color
GLfloat black30[ ] = {0.0, 0.0, 0.0, .30}; //30% black Color
GLfloat white[ ] = {1.0, 1.0, 1.0, 1.0}; //White Color
GLfloat red[ ] = {1.0, 0.0, 0.0, 1.0};
GLfloat green[ ] = {0.0, 1.0, 0.0, 1.0};
GLfloat blue[ ] = {.0, 0.0, 1.0, 1.0};

//material
GLfloat ambientMaterial[ ] = {.6, 0.65, .65, 1};
GLfloat diffuseMaterial[ ] = {1, 1, 1, 1};
GLfloat specularMaterial[ ] = {0.5, .50,.50, 1};
GLfloat specularMaterialshiness =0.9 ;

GLdouble x_camera=0, y_camera=0, z_camera=0, theta_camera=4.71, phi_camera=0.14,
theta_camera_inc=0.01, radius_camera_inc=0.1, phi_camera_inc=0.01, radius_camera=11.7;

/*
g= gravitational constant
theta0= initial angle
v0= initial angular velocity
ti= initial time
tf= final time duration
alpha= damping factor
m= mass of bob
 */

int N = 3250; //N- factor inversely proportional to speed
double g = 9.8, theta0 = 3.1416/4, v0 = 0, ti = 0, tf = 250, alpha = initial_length*1.6, m = 10;

/* Functions definition*/


//sets the light intensity, color and position.
void initLighting()
{
   // Enable lighting
   glEnable(GL_LIGHTING);
   glEnable(GL_LIGHT0);
```

```
    // Set lighting intensity and color
    glLightfv(GL_LIGHT0, GL_AMBIENT, qaAmbientLight);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, qaDiffuseLight);
    glLightfv(GL_LIGHT0, GL_SPECULAR, qaSpecularLight);
    // Set the light position
    glLightfv(GL_LIGHT0, GL_POSITION, qaLightPosition);
}

//set the camera position and usage of gluLookAt function.
void camera(void)
{
    //drawaxes();
    glLoadIdentity();
    x_camera=radius_camera*cos(theta_camera);
    y_camera=radius_camera*sin(phi_camera);
    z_camera=radius_camera*sin(theta_camera);
    gluLookAt(x_camera,y_camera,z_camera,0,0,0,0,1,0);
}

//checks the pause flag and sets the time_step_counter needed for animation.
void timer(int id)
{
    if(pause_flag == 0)
    {
        time_step_counter++;
        if(time_step_counter >= N)
            time_step_counter = 0;
    }
    glutPostRedisplay();
}

//calculate amplitude, time variation and angular velocity of the pendulum bobs.
void RK4_Method(int pendno,double g, double l, double alpha, double m, double theta0, double
v0, int N, double ti, double tf, double **times, double **thetas, double **vs){
    double h = (tf - ti)/N;
    double t = ti;
    double theta, v, k11, k12, k21, k22, k31, k32, k41, k42;
    int i;
    alpha=alpha/l;
    theta = theta0;
    v = v0;

    times[pendno] = (double *) malloc(sizeof(double)*N);
    thetas[pendno] = (double *) malloc(sizeof(double)*N);
    vs[pendno] = (double *) malloc(sizeof(double)*N);
```

```
   thetas[pendno][0] = theta;
   vs[pendno][0] = v;
   times[pendno][0] = t;

   for(i = 1; i<N; i++)
   {
      k11 = h*v;
      k12 = -h*(alpha/m)*v - h*(g/l)*sin(theta);

      k21 = h*(v + k11/2);
      k22 = -h*(alpha/m)*(v + k12/2) - h*(g/l)*sin(theta + k12/2);

      k31 = h*(v + k21/2);
      k32 = -h*(alpha/m)*(v + k22/2) -h*(g/l)*sin(theta + k22/2);

      k41 = h*(v + k31);
      k42 =  -h*(alpha/m)*(v + k32) - h*(g/l)*sin(theta + k32);

      theta = theta + (k11 + 2*k21 + 2*k31 + k41)/6;
      v = v + (k12 + 2*k22 + 2*k32 + k42)/6;
      t = t + h;

      thetas[pendno][i] = theta;
      vs[pendno][i] = v;
      times[pendno][i] = t;
   }
}

/*Model the wooden frame, pendulum bobs and the strings that connect them. For every iteration
of call to this method, the relative positions of the strings and the pendulum bobs are set.*/
void drawPendulumModel(int pendno,double l,double time, double theta, double v)
 {
   double pend_dist_multiplier=0.4;
   camera( );
   glTranslatef(0,2,0);    //for centering the overall system of pendulum
   glColor3f(0.65,0.50,0.39);

   glBegin(GL_QUADS);

   //1st face
   glNormal3f( 0.0,  0.0,  1.0);
   glVertex3d(-0.5,  0.0,  0.0);
   glVertex3d( 0.5,  0.0,  0.0);
   glVertex3d( 0.5,  0.5,  0.0);
   glVertex3d(-0.5,  0.5,  0.0);
```

```
//2nd face
glVertex3d(-0.5,  0.0,  -6.0);
glVertex3d( 0.5,  0.0,  -6.0);
glVertex3d( 0.5,  0.5,  -6.0);
glVertex3d(-0.5,  0.5,  -6.0);

//3rd face
glVertex3d(-0.5,  0.5,  -6.0);
glVertex3d( -0.5,  0.5,  0.0);
glVertex3d( -0.5,  0.0,  0.0);
glVertex3d(-0.5,  0.0,  -6.0);

//4th face
glVertex3d(0.5,  0.5,  -6.0);
glVertex3d( 0.5,  0.5,  0.0);
glVertex3d( 0.5,  0.0,  0.0);
glVertex3d(0.5,  0.0,  -6.0);

//5th face
glVertex3d(0.5,  0.5,  -6.0);
glVertex3d( 0.5,  0.5,  0.0);
glVertex3d( -0.5,  0.5,  0.0);
glVertex3d(-0.5,  0.5,  -6.0);

//6th face
glVertex3d(0.5,  0.0,  -6.0);
glVertex3d( 0.5,  0.0,  0.0);
glVertex3d( -0.5,  0.0,  0.0);
glVertex3d(-0.5,  0.0,  -6.0);
glEnd();

// Pendulum bob
double pi = 3.1416,x1 = 0, y1 = 0,x2, y2;

if(0 <= theta < pi/2)
{
        x2 = l*cos(pi/2-theta);
        y2 = -l*sin(pi/2-theta);
}
else if(pi/2 <= theta < pi)
{
        x2 = l*cos(theta - pi/2);
        y2 = l*sin(theta - pi/2);
}
else if(-pi/2 <= theta < 0)
```

```
    {
            x2 = -l*cos(pi/2-abs(theta));
            y2 = -l*sin(pi/2-abs(theta));
    }
    else if(-pi <= theta < -pi/2)
    {
            x2 = -l*cos(abs(theta) - pi/2);
            y2 = l*sin(abs(theta) - pi/2);
    }
    else
    {
        printf("invalid theta\n");
    }

    // string connecting the bob with the wooden plank
    glLineWidth(5);
    glColor3f(0.3, 0.3, 0.3);
    glBegin(GL_LINES);
    glVertex3f(x1,y1,- pendno*pend_dist_multiplier);
    glVertex3f(x2,y2,- pendno*pend_dist_multiplier);
    glEnd( );

    glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, ambientMaterial);
    glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, diffuseMaterial);
    glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, specularMaterial);
    glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, specularMaterialshiness);

    glPushMatrix();
    glColor3f(0.753, 0.753, 0.753);
    glTranslated(x2, y2,- pendno*pend_dist_multiplier);
    glutSolidSphere(0.2, 50, 50);
    glPopMatrix();
}

/*Uses glutStrokeCharacter to print strings by translating and scaling them to the required
coordinates and size respectively.*/
void printString(const char *str, double x, double y, double size)
 {
   glPushMatrix( );
   glTranslatef(x,y,0);
   glScalef(size/153.0,size/153.0,1.0);
   int lineCt = 0;
   int len = strlen(str);
   for (int i = 0; i < len; i++)
    {
      if (str[i] == '\n')
      {
        lineCt++;
        glPopMatrix();
        glPushMatrix();
        glTranslatef(x,y-size*1.15*lineCt,0);
        glScalef(size/153.0,size/153.0,1.0);
      }
      else
      {
        glutStrokeCharacter(GLUT_STROKE_ROMAN,str[i]);
      }
    }
   glPopMatrix( );
}

//Used to display the contents of the screens
void display(void)
{
    glutFullScreen( );
    glutTimerFunc(ms_per_frame,timer,1);
    glClearColor (0.0,0.0,0.0,1.0);
    glClear (GL_COLOR_BUFFER_BIT);

    if(screen==1)
    {

      glLoadIdentity();
      x_camera=radius_camera*cos(theta_camera); //-0.02795
      y_camera=radius_camera*sin(phi_camera); //1.6326
      z_camera=-radius_camera*sin(theta_camera); //11.699
      gluLookAt(-0.0195,0.1326,11.6999,0,0,0,0,1,0);

      glPointSize(3);
      glBegin(GL_POINTS);
      glVertex3d(0.0,6,0.0);
      glEnd( );

      glPointSize(1);
      glColor3f(0.6,0.6,0.9);
      glLineWidth(3);
      printString("PES INSTITUTE OF TECHNOLOGY", -7, 5, 1.0);
      printString ("DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING ",-
8.5,4,0.7);
      glColor3f(1,0.6,0.5);
      printString("A COMPUTER GRAPHICS PROJECT ON",-6,3,0.7);
      glColor3f(0,1,1);
      printString("3D VIEW OF PENDULUM MOTION",-5,2,0.7);
      glColor3f(1,1,1);
      printString("By",4,-3.5,0.5);
      glColor3f(1,1,0);
      printString("Bhavani B M",4,-4.2,0.5);
      printString("(1PE15CS038)",7.5,-4.2,0.5);;
      printString("G L Priya",4,-5,0.5);
      printString("(1PE15CS054)",7.5,-5,0.5);
      printString("Dr. Sarasvathi V",-11,-4.2,0.5);
      printString("Mr. K S V Krishna Srikanth",-11,-5,0.5);
      glColor3f(1,1,1);
      printString("Under the guidance of,",-11,-3.5,0.5);
      glColor3f(1,0,0);
      printString("press enter to go to next page", 4.6, -6.3, 0.4);

    /*PES logo*/
      glEnable(GL_TEXTURE_2D);
      glColor4f(1.0f, 1.0f, 1.0f, 1.0f);
          tex_2d1 = SOIL_load_OGL_texture
                  (
                          "logo1.png",
                          SOIL_LOAD_AUTO,
                          SOIL_CREATE_NEW_ID,
                          SOIL_FLAG_COMPRESS_TO_DXT
                  );
          glBindTexture(GL_TEXTURE_2D, tex_2d1);
          glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
          glBegin(GL_POLYGON);
                  glTexCoord2f(0.0, 0.0);
                  glVertex2f(-1,1);
                  glTexCoord2f(0.0, 1.0);
                  glVertex2f(-1,-1);
          glTexCoord2f(1.0, 1.0);
                  glVertex2f(1,-1);
                  glTexCoord2f(1.0, 0.0);
                  glVertex2f(1,1);
          glEnd();
          glDisable(GL_TEXTURE_2D);
          glutSwapBuffers();
          glFlush();
          glutPostRedisplay();
    }
```

```
else if(screen==2)
{
   /*theory*/
   glEnable(GL_TEXTURE_2D);
   glColor4f(1.0f, 1.0f, 1.0f, 1.0f);
         tex_2d1 = SOIL_load_OGL_texture
                   (
                            "theory2.jpg",
                            SOIL_LOAD_AUTO,
                            SOIL_CREATE_NEW_ID,
                            SOIL_FLAG_COMPRESS_TO_DXT
                   );
         glBindTexture(GL_TEXTURE_2D, tex_2d1);
         glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
         glBegin(GL_POLYGON);
                  glTexCoord2f(0.0, 0.0);
                  glVertex2f(-2,-3);
                   glTexCoord2f(0.0, 1.0);
                  glVertex2f(-2,5);
                  glTexCoord2f(1.0, 1.0);
                  glVertex2f(-9,5);
                  glTexCoord2f(1.0, 0.0);
                  glVertex2f(-9,-3);
         glEnd();

   /*demo*/
  glEnable(GL_TEXTURE_2D);
  glColor4f(1.0f, 1.0f, 1.0f, 1.0f);
  tex_2d1 = SOIL_load_OGL_texture
   (
      "demo1.jpg",
      SOIL_LOAD_AUTO,
      SOIL_CREATE_NEW_ID,
      SOIL_FLAG_COMPRESS_TO_DXT
   );
         glBindTexture(GL_TEXTURE_2D, tex_2d1);
         glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
         glBegin(GL_POLYGON);
                  glTexCoord2f(0.0, 0.0);
                  glVertex2f(9,-3);
         glTexCoord2f(0.0, 1.0);
         glVertex2f(9,5);
         glTexCoord2f(1.0, 1.0);
         glVertex2f(2,5);
         glTexCoord2f(1.0, 0.0);
```

```
            glVertex2f(2,-3);
            glEnd();

            /*exit*/
   glEnable(GL_TEXTURE_2D);
   glColor4f(1.0f, 1.0f, 1.0f, 1.0f);
   tex_2d1 = SOIL_load_OGL_texture
    (
        "exit1.jpeg",
        SOIL_LOAD_AUTO,
        SOIL_CREATE_NEW_ID,
        SOIL_FLAG_COMPRESS_TO_DXT
    );
    glBindTexture(GL_TEXTURE_2D, tex_2d1);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
    glBegin(GL_POLYGON);
            glTexCoord2f(0.0, 0.0);
            glVertex2f(10,-5);
            glTexCoord2f(0.0, 1.0);
            glVertex2f(10,-6);
            glTexCoord2f(1.0, 1.0);
            glVertex2f(11,-6);
            glTexCoord2f(1.0, 0.0);
            glVertex2f(11,-5);
    glEnd();
    glDisable(GL_TEXTURE_2D);
    glutSwapBuffers();
    glFlush();
}
else if(screen==3)
{
    glColor3f(0.957, 0.643, 0.376);
    glBegin(GL_POLYGON);
    glVertex2f(-9.8,-4.5);
    glVertex2f(7.8,-4.5);
    glVertex2f(7.8,5.8);
    glVertex2f(-9.8,5.8);
    glEnd();

    glColor3f(0,1,1);
    glLineWidth(4);
    printString(" 3D View of Pendulum Motion",-5 ,4.9,0.6);
    glBegin(GL_LINES);
            glVertex2d(-4.6,4.7);
            glVertex2d(3,4.7);
    glEnd();
    glLineWidth(3);
    glColor3f(0,0,0);
     printString("   * A steady pendulum frame consists of 15 pendulas, each with unique 
length",-9.7,4.2,0.45);
     printString("   * The length of the longest pendulum has been adjusted so that it executes 51 
",-9.7 ,3.5,0.45);
     printString("    oscillations in 60 seconds period.",-9.7,2.9,0.45);
     printString("   * The length of each successive shorter pendulum is carefully adjusted so that 
it ",-9.7,2.2,0.45);
     printString("    executes one additional oscillation in this period.",-9.7,1.6,0.45);
     printString("   * Thus, the 15th pendulum (shortest) undergoes 65 oscillations.",-
9.7,0.9,0.45);
     printString("   * When all 15 pendulums are started together, they quickly fall out of sync-
their",-9.7,0.2,0.45);
     printString("    relative phases continuously change because of their different periods of 
oscillations.  ",-9.7,-0.4,0.45);
     printString("   * However, after 60 seconds they will all have executed an integral number of 
",-9.7,-1.1,0.45);
     printString("    oscillations and be back in sync again at that instant, ready to repeat the 
dance.",-9.7,-1.7,0.45);
     printString("   * t=2*pi*sqrt(l/g) from which we may calculate ' g ' as ",-7.5,-2.4,0.45);
     printString("   * g=(4*pi^2*l)/t^2",-4.0,-3.1,0.45);



   /*left*/
     glEnable(GL_TEXTURE_2D);
     glColor4f(1.0f, 1.0f, 1.0f, 1.0f);
     tex_2d1 = SOIL_load_OGL_texture
     (
        "left.png",
        SOIL_LOAD_AUTO,
        SOIL_CREATE_NEW_ID,
        SOIL_FLAG_COMPRESS_TO_DXT
     );
     glBindTexture(GL_TEXTURE_2D, tex_2d1);
     glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
     glBegin(GL_POLYGON);
          glTexCoord2f(0.0, 0.0);
          glVertex2f(-11,-5);
          glTexCoord2f(0.0, 1.0);
          glVertex2f(-11,-6);
          glTexCoord2f(1.0, 1.0);
          glVertex2f(-10,-6);
          glTexCoord2f(1.0, 0.0);
        glVertex2f(-10,-5);
    glEnd();

   /*right*/
     glEnable(GL_TEXTURE_2D);
     glColor4f(1.0f, 1.0f, 1.0f, 1.0f);
     tex_2d1 = SOIL_load_OGL_texture
     (
        "down.png",
        SOIL_LOAD_AUTO,
        SOIL_CREATE_NEW_ID,
        SOIL_FLAG_COMPRESS_TO_DXT
     );
    glBindTexture(GL_TEXTURE_2D, tex_2d1);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
    glBegin(GL_POLYGON);
                    glTexCoord2f(0.0, 0.0);
                    glVertex2f(8,-5);
          glTexCoord2f(0.0, 1.0);
          glVertex2f(8,-6);
          glTexCoord2f(1.0, 1.0);
          glVertex2f(9,-6);
          glTexCoord2f(1.0, 0.0);
          glVertex2f(9,-5);
    glEnd();


   /*exit*/
    glEnable(GL_TEXTURE_2D);
    glColor4f(1.0f, 1.0f, 1.0f, 1.0f);
    tex_2d1 = SOIL_load_OGL_texture
    (
       "exit1.jpeg",
       SOIL_LOAD_AUTO,
       SOIL_CREATE_NEW_ID,
       SOIL_FLAG_COMPRESS_TO_DXT
    );
    glBindTexture(GL_TEXTURE_2D, tex_2d1);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
    glBegin(GL_POLYGON);
          glTexCoord2f(0.0, 0.0);
          glVertex2f(10,-5);
          glTexCoord2f(0.0, 1.0);
          glVertex2f(10,-6);
          glTexCoord2f(1.0, 1.0);
```

```
            glVertex2f(11,-6);
            glTexCoord2f(1.0, 0.0);
            glVertex2f(11,-5);
      glEnd();
     glDisable(GL_TEXTURE_2D);
     glutSwapBuffers();
     glFlush();

}
else if(screen==4)
{
      glColor3f(0.957, 0.643, 0.376);
      glBegin(GL_POLYGON);
      glVertex2f(-9.8,-4.5);
      glVertex2f(7.8,-4.5);
      glVertex2f(7.8,6.0);
      glVertex2f(-9.8,6.0);
      glEnd();
      glColor3f(0,1,1);
      glLineWidth(4);
      printString("INSTRUCTIONS",-3,4.9,0.8);
      glLineWidth(3);
      glBegin(GL_LINES);
         glVertex2d(-3.1,4.6);
         glVertex2d(1.7,4.6);
      glEnd();
      glColor3f(0,0,0);
        printString("  * Press '+/-' to Zoom in/Zoom out.",-9.7,4.0,0.45);
        printString("  * Press ' w ' to move Up. ",-9.7,3.0,0.45);
        printString("  * Press ' s ' to move Down.",-9.7,2.0,0.45);
        printString("  * Press ' d ' to move Left.",-9.7,1.0,0.45);
        printString("  * Press ' a ' to move Right.",-9.7,0.0,0.45);
        printString("  * Press ' Spacebar ' to Play/Pause.",-9.7,-1.0,0.45);

      glEnable(GL_TEXTURE_2D);
      glColor4f(1.0f, 1.0f, 1.0f, 1.0f);
      tex_2d1 = SOIL_load_OGL_texture
      (
         "thatsallfolks.jpg",
          SOIL_LOAD_AUTO,
          SOIL_CREATE_NEW_ID,
          SOIL_FLAG_COMPRESS_TO_DXT
      );
      glBindTexture(GL_TEXTURE_2D, tex_2d1);
      glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
```

```
   glBegin(GL_POLYGON);
         glTexCoord2f(0.0, 0.0);
         glVertex2f(-3.5,-2.2);
         glTexCoord2f(0.0, 1.0);
         glVertex2f(-3.5,-4.0);
         glTexCoord2f(1.0, 1.0);
         glVertex2f(2.0,-4.0);
         glTexCoord2f(1.0, 0.0);
         glVertex2f(2.0,-2.2);
   glEnd();
/*left*/
   glEnable(GL_TEXTURE_2D);
   glColor4f(1.0f, 1.0f, 1.0f, 1.0f);
   tex_2d1 = SOIL_load_OGL_texture
   (
      "left.png",
      SOIL_LOAD_AUTO,
      SOIL_CREATE_NEW_ID,
      SOIL_FLAG_COMPRESS_TO_DXT
   );
   glBindTexture(GL_TEXTURE_2D, tex_2d1);
   glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
   glBegin(GL_POLYGON);
         glTexCoord2f(0.0, 0.0);
         glVertex2f(-11,-5);
         glTexCoord2f(0.0, 1.0);
         glVertex2f(-11,-6);
         glTexCoord2f(1.0, 1.0);
         glVertex2f(-10,-6);
         glTexCoord2f(1.0, 0.0);
         glVertex2f(-10,-5);
   glEnd();

 /*right*/
   glEnable(GL_TEXTURE_2D);
   glColor4f(1.0f, 1.0f, 1.0f, 1.0f);
   tex_2d1 = SOIL_load_OGL_texture
   (
      "down.png",
      SOIL_LOAD_AUTO,
      SOIL_CREATE_NEW_ID,
      SOIL_FLAG_COMPRESS_TO_DXT
   );
   glBindTexture(GL_TEXTURE_2D, tex_2d1);
   glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
```

```
      glBegin(GL_POLYGON);
            glTexCoord2f(0.0, 0.0);
            glVertex2f(8,-5);
            glTexCoord2f(0.0, 1.0);
            glVertex2f(8,-6);
            glTexCoord2f(1.0, 1.0);
            glVertex2f(9,-6);
            glTexCoord2f(1.0, 0.0);
            glVertex2f(9,-5);
      glEnd();

    /*exit*/
      glEnable(GL_TEXTURE_2D);
      glColor4f(1.0f, 1.0f, 1.0f, 1.0f);
      tex_2d1 = SOIL_load_OGL_texture
      (
         "exit1.jpeg",
         SOIL_LOAD_AUTO,
         SOIL_CREATE_NEW_ID,
         SOIL_FLAG_COMPRESS_TO_DXT
      );
      glBindTexture(GL_TEXTURE_2D, tex_2d1);
      glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
      glBegin(GL_POLYGON);
            glTexCoord2f(0.0, 0.0);
            glVertex2f(10,-5);
            glTexCoord2f(0.0, 1.0);
            glVertex2f(10,-6);
            glTexCoord2f(1.0, 1.0);
            glVertex2f(11,-6);
            glTexCoord2f(1.0, 0.0);
            glVertex2f(11,-5);
      glEnd();
      glDisable(GL_TEXTURE_2D);
      glutSwapBuffers();
      glFlush();

}
else if(screen==5)
{

    glColor3f(1.0,1.0,0.0);
    glLineWidth(4);
    /*demo image*/
    glEnable(GL_TEXTURE_2D);
    glColor4f(1.0f, 1.0f, 1.0f, 1.0f);
    tex_2d1 = SOIL_load_OGL_texture
    (
            "demo1.jpg",
            SOIL_LOAD_AUTO,
            SOIL_CREATE_NEW_ID,
            SOIL_FLAG_COMPRESS_TO_DXT
     );
    glBindTexture(GL_TEXTURE_2D, tex_2d1);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
    glBegin(GL_POLYGON);
           glTexCoord2f(0.0, 0.0);
           glVertex2f(2,-2);
           glTexCoord2f(0.0, 1.0);
           glVertex2f(2,5);
           glTexCoord2f(1.0, 1.0);
           glVertex2f(-4,5);
           glTexCoord2f(1.0, 0.0);
           glVertex2f(-4,-2);
glEnd();

   /*left*/
   glEnable(GL_TEXTURE_2D);
   glColor4f(1.0f, 1.0f, 1.0f, 1.0f);
   tex_2d1 = SOIL_load_OGL_texture
    (
        "left.png",
        SOIL_LOAD_AUTO,
        SOIL_CREATE_NEW_ID,
        SOIL_FLAG_COMPRESS_TO_DXT
    );
    glBindTexture(GL_TEXTURE_2D, tex_2d1);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
    glBegin(GL_POLYGON);
           glTexCoord2f(0.0, 0.0);
           glVertex2f(-11,-5);
           glTexCoord2f(0.0, 1.0);
           glVertex2f(-11,-6);
           glTexCoord2f(1.0, 1.0);
           glVertex2f(-10,-6);
           glTexCoord2f(1.0, 0.0);
           glVertex2f(-10,-5);
    glEnd();
    /*right*/
    glEnable(GL_TEXTURE_2D);
```

```
glColor4f(1.0f, 1.0f, 1.0f, 1.0f);
tex_2d1 = SOIL_load_OGL_texture
 (
    "down.png",
    SOIL_LOAD_AUTO,
    SOIL_CREATE_NEW_ID,
    SOIL_FLAG_COMPRESS_TO_DXT
 );
 glBindTexture(GL_TEXTURE_2D, tex_2d1);
 glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
 glBegin(GL_POLYGON);
        glTexCoord2f(0.0, 0.0);
        glVertex2f(8,-5);
        glTexCoord2f(0.0, 1.0);
        glVertex2f(8,-6);
        glTexCoord2f(1.0, 1.0);
                glVertex2f(9,-6);
        glTexCoord2f(1.0, 0.0);
        glVertex2f(9,-5);
 glEnd();


 /*exit*/
 glEnable(GL_TEXTURE_2D);
glColor4f(1.0f, 1.0f, 1.0f, 1.0f);
tex_2d1 = SOIL_load_OGL_texture
 (
    "exit1.jpeg",
    SOIL_LOAD_AUTO,
    SOIL_CREATE_NEW_ID,
    SOIL_FLAG_COMPRESS_TO_DXT
 );
 glBindTexture(GL_TEXTURE_2D, tex_2d1);
 glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
 glBegin(GL_POLYGON);
        glTexCoord2f(0.0, 0.0);
        glVertex2f(10,-5);
        glTexCoord2f(0.0, 1.0);
        glVertex2f(10,-6);
        glTexCoord2f(1.0, 1.0);
        glVertex2f(11,-6);
        glTexCoord2f(1.0, 0.0);
        glVertex2f(11,-5);
 glEnd();
```

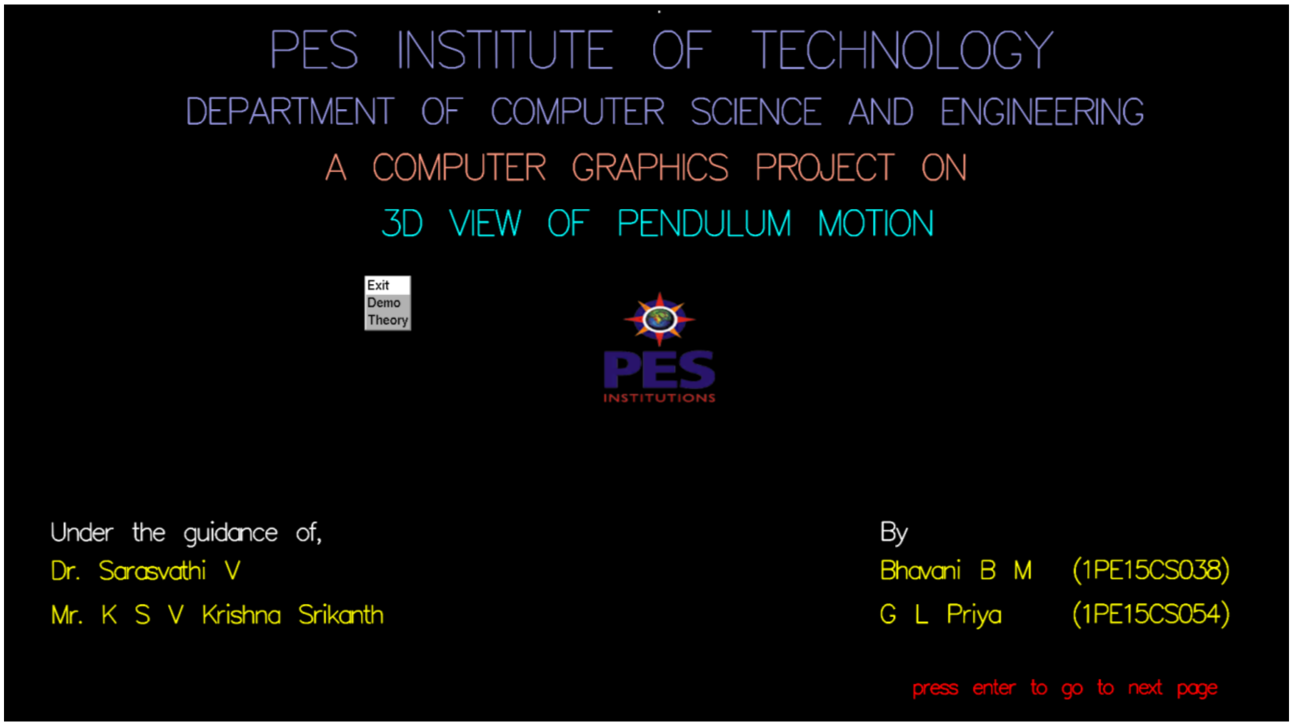
```
    glDisable(GL_TEXTURE_2D);
    glutSwapBuffers();
    glFlush();
}

else if(screen==6)
{
    initLighting();
    for(int i=0; i<NumOfBobs;i++)
    {

            drawPendulumModel(i,length[i],times[i][time_step_counter],thetas[i][time_step_counter]
            ,vs[i][time_step_counter]);
    }
    glutSwapBuffers();
    glFlush();

}
else if(screen==7)
{
   glEnable(GL_TEXTURE_2D);
   glColor4f(1.0f, 1.0f, 1.0f, 1.0f);
   tex_2d1 = SOIL_load_OGL_texture
    (
        "tq2.jpg",
        SOIL_LOAD_AUTO,
        SOIL_CREATE_NEW_ID,
        SOIL_FLAG_COMPRESS_TO_DXT
    );
    glBindTexture(GL_TEXTURE_2D, tex_2d1);
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
    glBegin(GL_POLYGON);
            glTexCoord2f(0.0, 0.0);
            glVertex2f(9.9,3.5);
            glTexCoord2f(0.0, 1.0);
            glVertex2f(11.2,-7.6);
            glTexCoord2f(1.0, 1.0);
            glVertex2f(-11.2,-7.6);
            glTexCoord2f(1.0, 0.0);
            glVertex2f(-9.9,3.5);
    glEnd();
    glDisable(GL_TEXTURE_2D);
    glutSwapBuffers();
    glFlush();
}
```

```
if(exitflag==1)
{
   exitBox();
}
   glFlush();
   glutSwapBuffers();
   glutPostRedisplay();
}

/*callback for glutReshapeFunc*/
void reshape (int w, int h)
{
   glViewport (0, 0, (GLsizei)w, (GLsizei)h);
   glMatrixMode (GL_PROJECTION);
   glLoadIdentity ();
   glOrtho(-1,1,-1,1,-1,1);
   gluPerspective (60, (GLfloat)w / (GLfloat)h, 0.1, 100.0);
   glMatrixMode (GL_MODELVIEW);
}

/*called when exit button is pressed*/
void exitBox( )
{

   glColor3f(1,1,1);
   glBegin(GL_POLYGON);
   glVertex2d(-4,-3.5);
   glVertex2d(4,-3.5);
   glVertex2d(4,-6);
   glVertex2d(-4,-6);
   glEnd();
   glColor3f(0,0,0);
   printString("Do u really want to exit???",-3,-4,0.5);
   glColor3f(0.8,0.8,0.2);
   glBegin(GL_QUADS);
      glVertex2d(-3,-5.5);
      glVertex2d(-2,-5.5);
      glVertex2d(-2,-4.8);
      glVertex2d(-3,-4.8);
   glEnd();
   glBegin(GL_QUADS);
      glVertex2d(2,-5.5);
      glVertex2d(3,-5.5);
      glVertex2d(3,-4.8);
      glVertex2d(2,-4.8);
```

```
    glEnd();
    glColor3f(1.0,1.0,1.0);
    glPointSize(2);
    printString("YES",-2.8,-5.3,0.3);
    printString("NO",2.25,-5.3,0.3);
    glFlush();
}

/*Callback for mouse clicks*/
void mouse (int btn, int state, int x, int y)
{

    if(btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN){

        if(exitflag==1)//exitBox
        {
            if(x<800 && x> 720 && y<975 && y>930) {exitflag=0;exit(0);}
            else if(x<1180 && x>1120  && y<975 && y>930)  {exitflag=0;}
        }

        else if(screen==2) //theory image, demo image
        {

            if(x<800 && x>250 && y<800&& y>140) screen=3;
            if(x< 1670&& x> 1120&& y<800 && y>140) screen=6;
            if (x<1850 && x>1750 && y<1020 && y>930) {exitflag=1;}
        }
        else if(screen==3)//theorypage1
        {
            if(x<150 && x>85 && y<1020 && y>935) screen=2;
            if(x<1670 && x>1595 && y<1020 && y>935) screen=4;
            if(x<1800 && x>1750 && y<1000 && y>950) {exitflag=1;}
        }
        else if(screen==4)//instructions
        {
            if(x<150 && x>85 && y<1020 && y>935) screen=3;
            if(x<1670 && x>1595 && y<1020 && y>935) screen=5;
            if(x<1800 && x>1750 && y<1000 && y>950) {exitflag=1;}

        }
         else if(screen==5)//demo image
        {
            if(x<150 && x>85 && y<1020 && y>935) screen=4;
            if(x<1670 && x>1595 && y<1020 && y>935) screen=6;
            if(x<1800 && x>1750 && y<1000 && y>950) {exitflag=1;}
        }
    }glutPostRedisplay( );
}

/* callback when keys are pressed*/
void keyboard(unsigned char key, int x, int y)
{
    switch (key)
    {
        case 13: if(screen==7) exit(0);
        else
            screen++;
            break;
            glutPostRedisplay( );
        case 27:        //esc button
            exit(0);
            break;
        case 32: if(screen==6){    //space button
            if(pause_flag == 0)
                pause_flag = 1;
            else
                pause_flag = 0;}
            break;
        case 115:       //s
        case 83:if(screen==6){
            phi_camera=phi_camera+phi_camera_inc;}
            break;
        case 97:        //a
        case 65:if(screen==6){
            theta_camera=theta_camera+theta_camera_inc;}
            break;
        case 117:       //w
        case 87:if(screen==6){
            phi_camera=phi_camera-phi_camera_inc;}
            break;
        case 100:       //d
        case 68:if(screen==6){
            theta_camera=theta_camera-theta_camera_inc;}
            break;
        case 43:if(screen==6){
            radius_camera=radius_camera-radius_camera_inc;}
            break;
        case 45:if(screen==6){
            radius_camera=radius_camera+radius_camera_inc;}
            break;
    }
    glutPostRedisplay();
}

/* Use to calculate the angular velocity, amplitude, time variation and increases the length  of the
pendulum*/
void init(void )
{
    times = (double **) malloc(sizeof(double *)*NumOfBobs);
    thetas = (double **) malloc(sizeof(double *)*NumOfBobs);
    vs = (double **) malloc(sizeof(double *)*NumOfBobs);

    for (int i=0;i<NumOfBobs;i++)
    {
        length[i]=length[i]*10;
    }
}

/* callback function for glutCreateMenu API*/
void menu(int ch)
{
    if(screen==1)
  {
    switch(ch)
    {
        case 1:exit(0);
                break;
                case 2: screen=5;
                                break;
                case 3: screen=3;
                                break;
            glutPostRedisplay();
    }
  }
}

//main function
int main (int argc, char **argv)
{
    glutInit (&argc, argv);
    glutInitDisplayMode (GLUT_DOUBLE);
    printf("Enter the number of pendulum bobs(<=15): ");
    scanf("%d",&NumOfBobs);
    glutInitWindowSize (600, 600);
    glutInitWindowPosition (0, 0);
    glutCreateWindow ("Pendulum Wave Simulation");
    init( );
    for(int i=0;i<NumOfBobs;i++)
    {
       RK4_Method(i,g, length[i], alpha, m, theta0, v0, N, ti, tf, &times[0], &thetas[0], &vs[0]);
    }

    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutKeyboardFunc(keyboard);
    glutMouseFunc(mouse);
    if(screen==1)
    {
      glutCreateMenu(menu);
      glutAddMenuEntry("Exit",1);
      glutAddMenuEntry("Demo",2);
      glutAddMenuEntry("Theory",3);
      glutAttachMenu(GLUT_RIGHT_BUTTON);
    }
    initgLOptions();
    glutMainLoop ();
    return 0;
}
```

/*End of Source Code*/

6. RESULT SNAPSHOTS



PES INSTITUTE OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

A COMPUTER GRAPHICS PROJECT ON

3D VIEW OF PENDULUM MOTION

Under the guidance of,
Dr. Sarasvathi V
Mr. K S V Krishna Srikanth

By
Bhavani B M (1PE15CS038)
G L Priya (1PE15CS054)

press enter to go to next page

Fig.1 Front Page with menu option

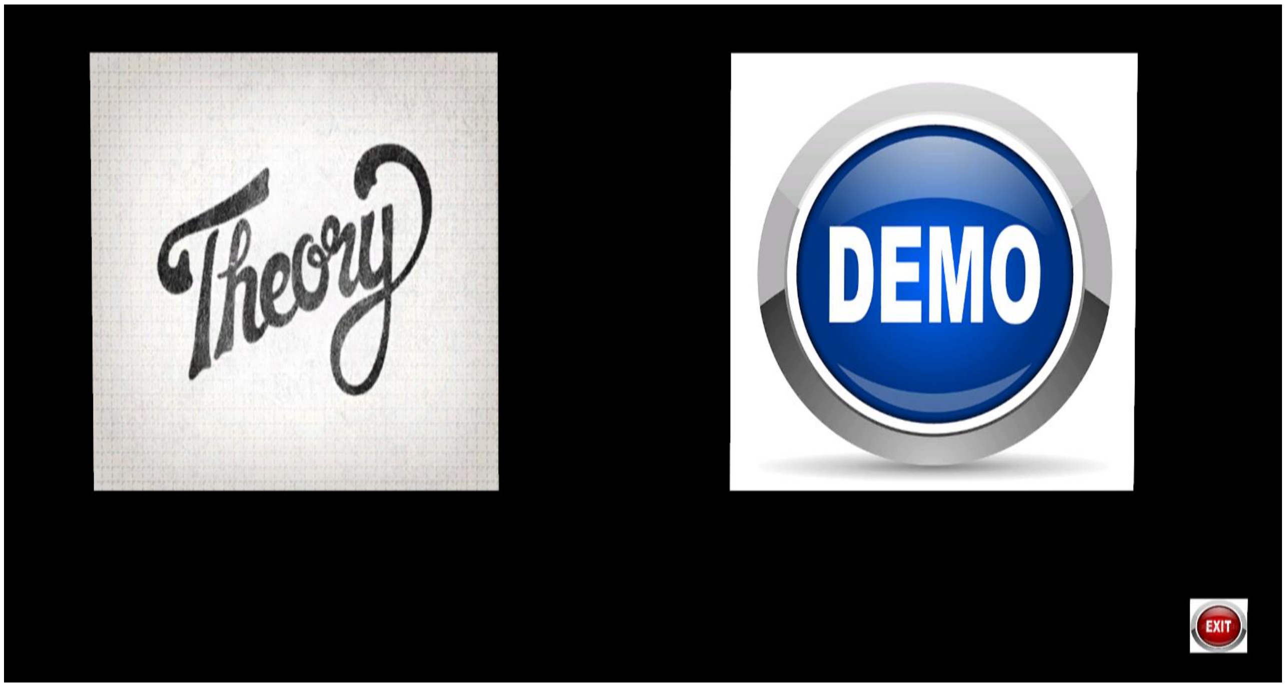


Fig. 2 Theory and Demo with Mouse function

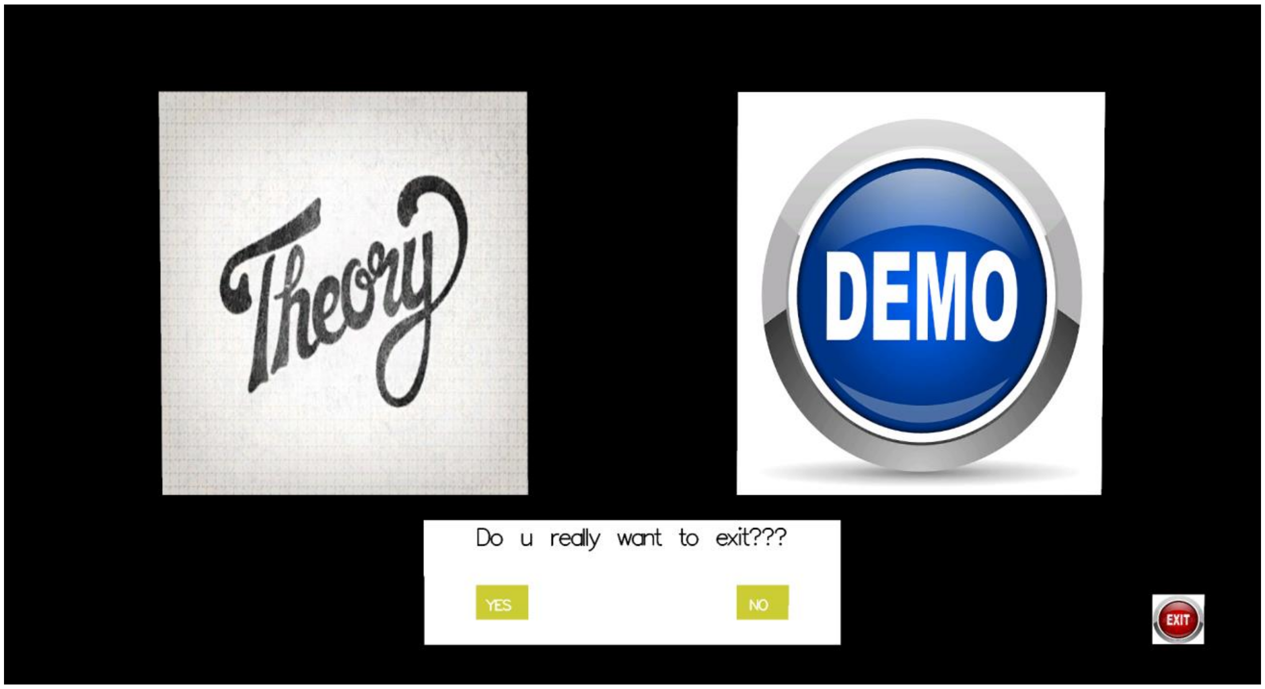
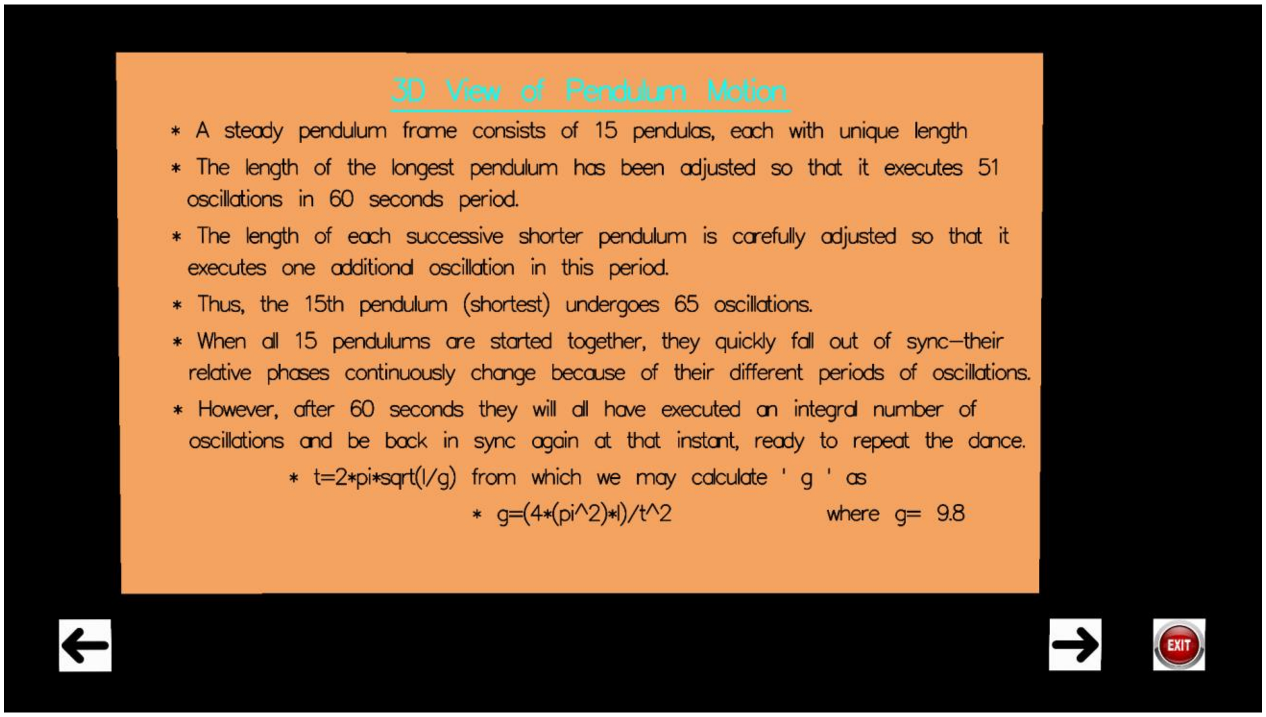


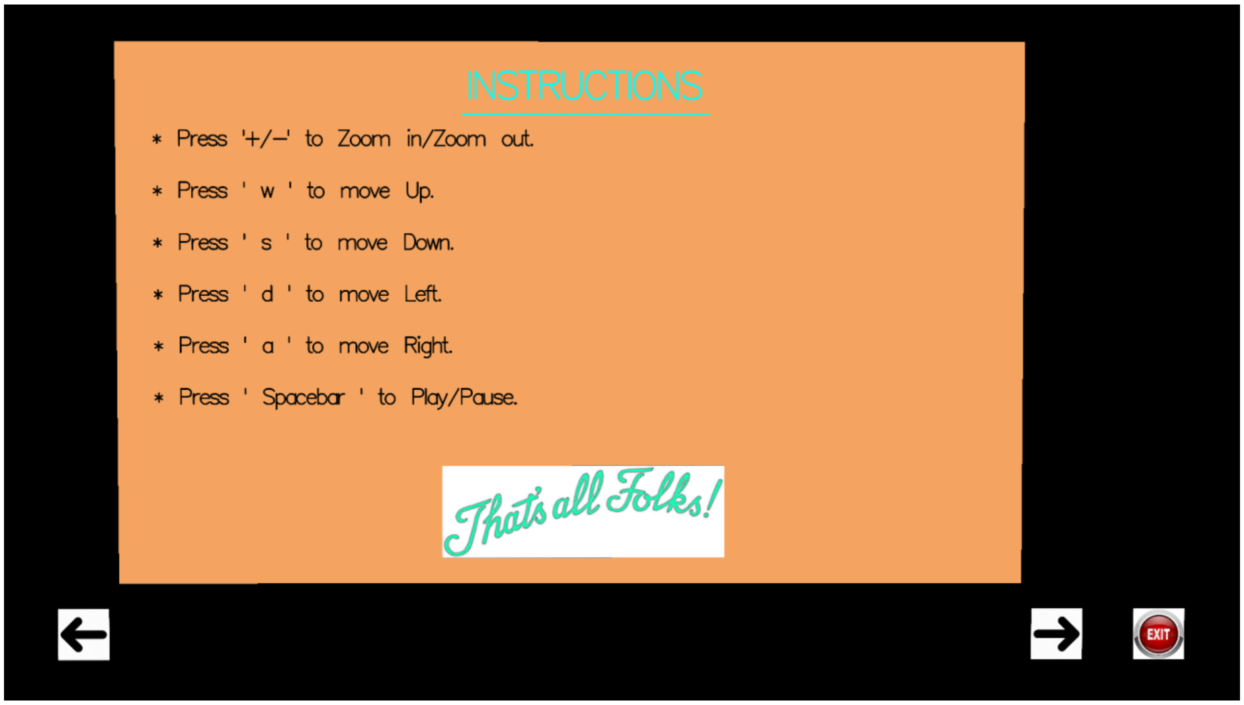
Fig.3 On clicking exit button



3D View of Pendulum Motion

* A steady pendulum frame consists of 15 pendulas, each with unique length
* The length of the longest pendulum has been adjusted so that it executes 51 oscillations in 60 seconds period.
* The length of each successive shorter pendulum is carefully adjusted so that it executes one additional oscillation in this period.
* Thus, the 15th pendulum (shortest) undergoes 65 oscillations.
* When all 15 pendulums are started together, they quickly fall out of sync—their relative phases continuously change because of their different periods of oscillations.
* However, after 60 seconds they will all have executed an integral number of oscillations and be back in sync again at that instant, ready to repeat the dance.
* t=2*pi*sqrt(l/g) from which we may calculate ' g ' as
* g=(4*(pi^2)*l)/t^2 where g= 9.8

Fig. 4 Theory of pendulum motion



INSTRUCTIONS

* Press '+/−' to Zoom in/Zoom out.
* Press ' w ' to move Up.
* Press ' s ' to move Down.
* Press ' d ' to move Left.
* Press ' a ' to move Right.
* Press ' Spacebar ' to Play/Pause.

That's all Folks!

Fig. 5 Instructions for pendulum animation

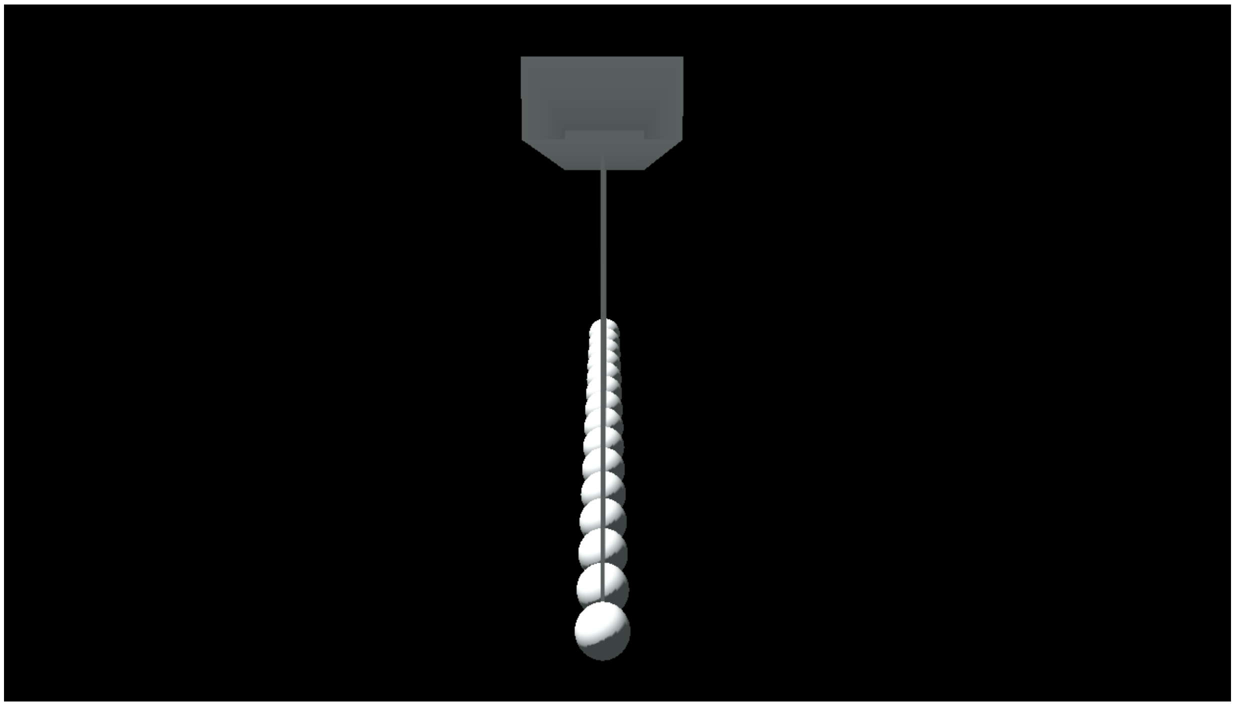


Fig. 6 Still view of pendulum frame

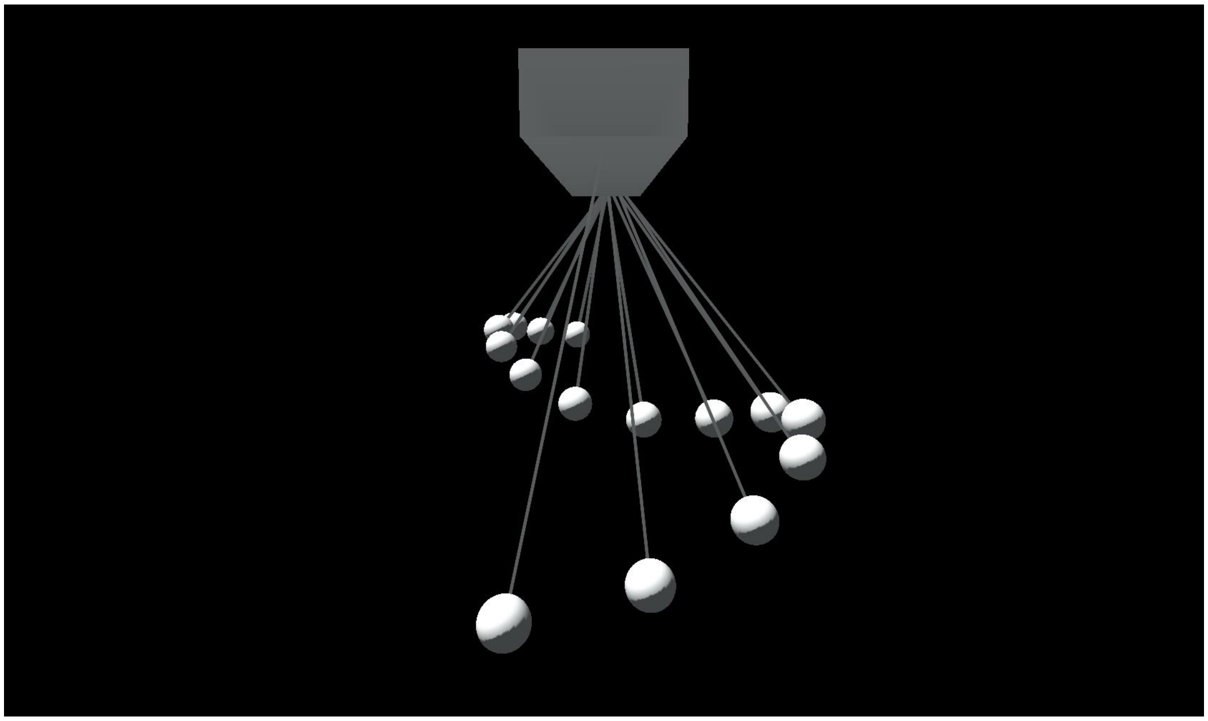


Fig. 7 Pendulum frame in motion

7. CONCLUSION

We had a great experience. In this course of designing this package, we discovered and learnt many things pertaining to the topic of OpenGL programming. We have tried to our best potential to incorporate all the basic level requirements of a normal graphics package for UBUNTU operating system. This is very reliable graphics package supporting various primitive objects like polygon, line loop, etc. Also color selection, mouse, keyboard based interface are included. Transformation like translation, rotation and scaling is provided.

The project can be further developed to include the different methods to find the angular velocity, time variation, amplitude of the pendulum bobs. This would help to design and visualize different 3D pendulum wave motions.

8. BIBLIOGRAPHY

1. Computer Graphics with OpenGL, 4th edition, Pearson Education, 2011

2. Interactive Computer Graphics, A Top Down approach with OpenGL – Edward Angel, 5th edition, Addison-Wesley, 2008

3. https://www.opengl.org