CHAPTER 1

INTRODUCTION

1.1 What is OpenGL?

OpenGL is a software interface to graphics hardware. This interface consists of about150 distinct commands that you use to specify the objects and operations needed to produce interactive three-dimensional applications.

OpenGL is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms. To achieve these qualities, no commands for performing windowing tasks or obtaining user input are included in OpenGL; instead, you must work through whatever windowing system controls the particular hardware you're using. Similarly, OpenGL doesn't provide high-level commands for describing models of three-dimensional objects. Such commands might allow you to specify relatively complicated shapes such as automobiles, parts of the body, airplanes, or molecules. With OpenGL, you must build up your desired model from a small set of *geometric primitives* - points, lines, and polygons.

1.2 What is GLUT?

GLUT is a complete API written by Mark Kilgard which lets you create windows and handle the messages. It exists for several platforms, that means that a program which uses GLUT can be compiled on many platforms without (or at least with very few) changes in the code.

1.3 How does OpenGL work?

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 draw several polygons, lines or what ever with this color then. There are no classes like in 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. You want to know which libraries coexist with the ones called before? There are libraries for every system, Windows has the wgl*-Functions, Unix systems glx* and so on.

A very important thing is to know, that there are two important matrices, which affect the transformation from the 3d-world to the 2d-screen: The projection matrix and the modelview matrix. The projection matrix contains information, how a vertex – let's say a "point" in space – shall be mapped to the screen. This contains, whether the projection shall be isometric or from a perspective, how wide the field of view is and so on. Into the other matrix you put information, how the objects are moved, where the viewer is and so on.

Don't like matrices? Don't be afraid, you probably won't really see them, at least at the beginning. There are commands that do all the maths for you.

Some basic commands are explained later in this tutorial.

1.4 **How can I use GLUT?**

GLUT provides some routines for the initialization and creating the window (or fullscreen mode, if you want to). Those functions are called first in a GLUT application:

In your first line you always write glutInit(&argc, argv). After this, you must tell GLUT, which display mode you want – 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). In later tutorials we will use some more constants here.

After the initialization you call glCreateWindow() with the window name as parameter.Then you can (and should) pass some methods for certain events. The most

important ones are "reshape" and "display". In reshape you need to (re)define the field of view and specify a new area in the window, where OpenGL is allowed to draw to.

Display should clear the so called color buffer – let's say this is the sheet of paper – and draw our objects.

You pass the methods by glut*Func(), for example glutDisplayFunc(). At the end of the main function you call glutMainLoop(). This function doesn't return, but calls the several functions passed by glut*Func.

1.5 OPENGL RENDERING PIPELINE

Most implementations of OpenGL have a similar order of operations, a series of processing stages called the OpenGL rendering pipeline. This ordering, as shown in Figure 1-2, is not a strict rule of how OpenGL is implemented but provides a reliable guide for predicting what OpenGL will do.

If you are new to three-dimensional graphics, the upcoming description may seem like drinking water out of a fire hose. You can skim this now, but come back to Figure 1-2 as you go through each chapter in this book.

The following diagram shows the Henry Ford assembly line approach, which OpenGL takes to processing data. Geometric data (vertices, lines, and polygons) follow the path through the row of boxes that includes evaluators and per-vertex operations, while pixel data (pixels, images, and bitmaps) are treated differently for part of the process. Both types of data undergo the same final steps (rasterization and per-fragment operations) before the final pixel data is written into the framebuffer.

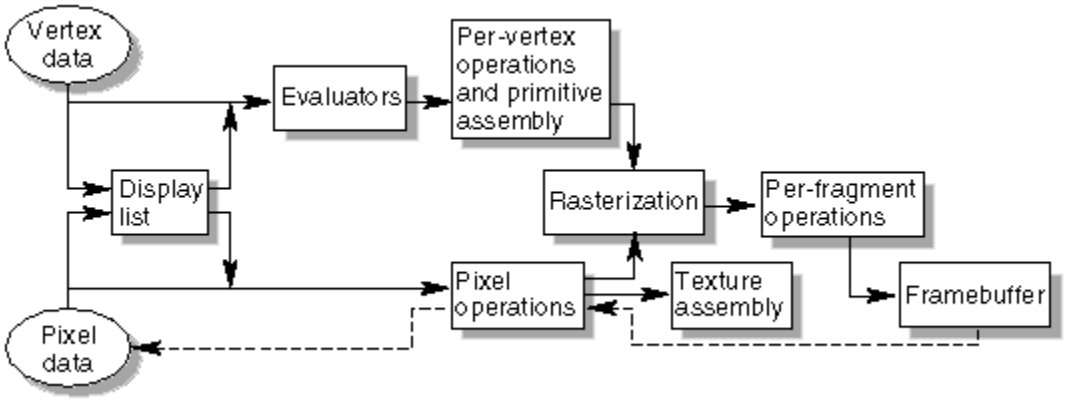


Figure 1.1: Order Of Operations

CHAPTER 2

SYSTEM REQUIREMENTS

2.1 HARDWARE REQUIREMENTS

Minimum hardware specification

Processor : Intel/AMD processor

Main memory : 512 MB RAM

Hard Disk : 40 GB

2.2 SOFTWARE REQUIREMENTS

Minimum software specification

Operating system : Windows

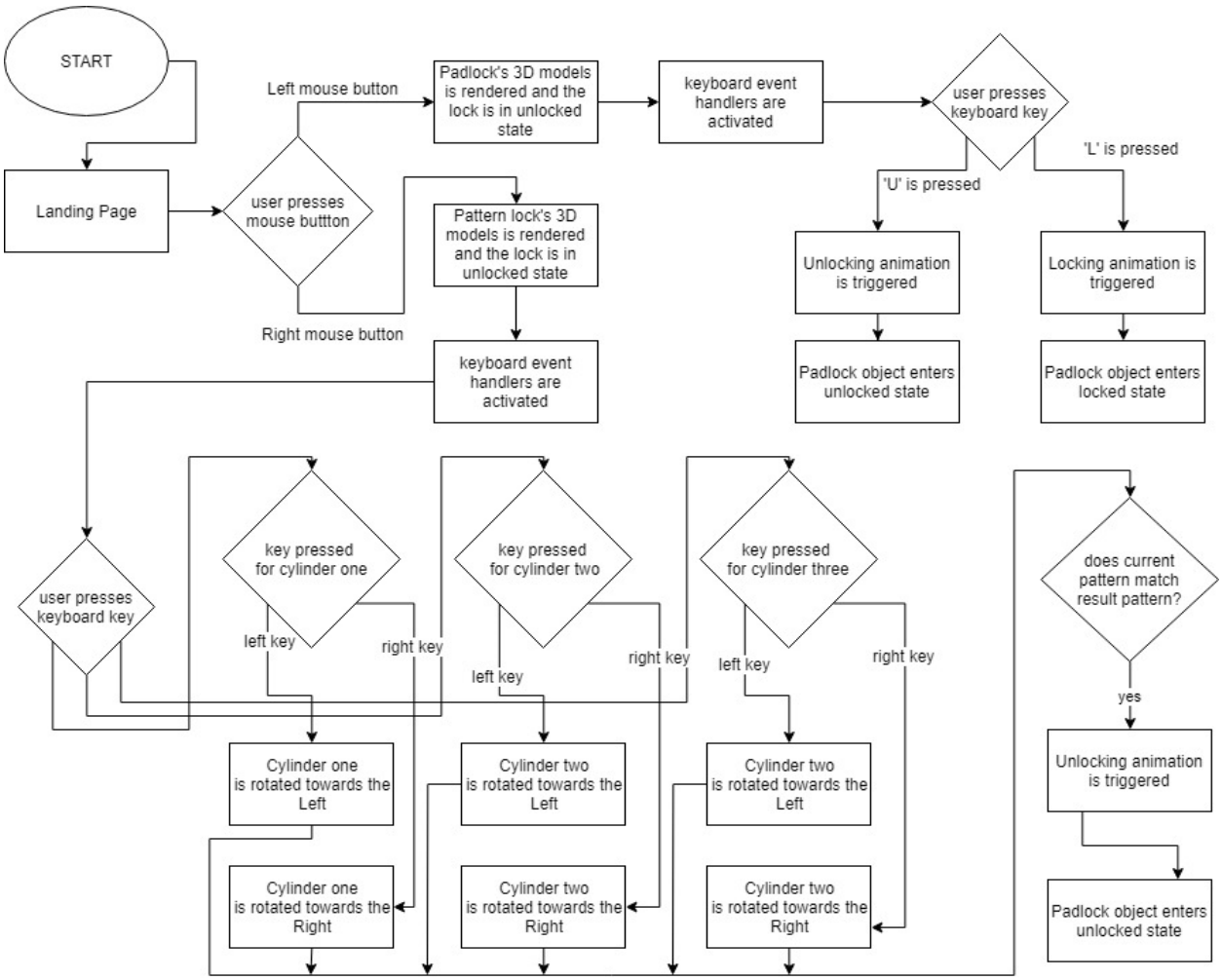
Software Used : Microsoft visual 8.0 using C++

Library Used : OPENGL Library

CHAPTER 3

PROJECT DESIGN

Lock and Key is an interactive 3D game that lets users interact with two different types of locks. The first lock is a pad lock that has animations for locking and unlocking. The second lock is a pattern lock that can be interacted with by the user. Upon selecting the correct pattern, the model will be unlocked.



CHAPTER 4

4. PROJECT IMPLEMENTATION

Lock and key lets the users interact with the game and trigger the locking and unlocking events inside the game. The first lock, which is a padlock has a locking and unlocking animation, where the key is translated into the keyhole and then rotated. The animation for opening the lock bar is then triggered. The second lock, which is a pattern lock contains cylinders with patterns on it and have rotational functionality. The user can rotate these cylinders to trigger the unlocking animation. The project has been implemented using OpenGL libraries and coded in C++.

4.1 HEADER FILES:

The <stdlib.h> Header File

include <stdlib.h>

The ISO C standard introduced this header file as a place to declare certain standard library functions. These include the Memory management functions (malloc, free, et. al.) communication with the environment (abort, exit) and others. Not yet all the standard functions of this header file are supported. If a declaration is present in the supplied header file, then uCR supports it and will continue to support it. If a function is not there, it will be added in time.

The <stdio.h> Header File

include <stdio.h>

Load the file SIMPLEIO.C for our first look at a file with standard I/O. Standard I/O refers to the most usual places where data is either read from, the keyboard, or written to, the video monitor. Since they are used so much, they are used as the default I/O devices and do not need

to be named in the Input/Output instructions. This will make more sense when we actually start to use them so lets look at the file in front of you.

The first thing you will notice is the second line of the file, the #include "stdio.h" line. This is very much like the #define we have already studied, except that instead of a simple substitution, an entire file is read in at this point. The system will find the file named "stdio.h" and read its entire contents in, replacing this statement. Obviously then, the file named "stdio.h" must contain valid C source statements that can be compiled as part of a program. This particular file is composed of several standard #defines to define some of the standard I/O operations. The file is called a header file and you will find several different header files on the source disks that came with your C compiler. Each of the header files has a specific purpose and any or all of them can be included in any program.

4.2. FUNCTIONS USED:

4.1.1. void intro() : Front page function

4.1.2. void lock() : Function that draws padlock.

4.1.3. void key() : Function that draws key.

4.1.4. void animateKey() : Idle function for key.

4.1.5. void displayLockAndKey() : It display both padlock and key.

4.1.6. void lockAndKeyKeyboard(unsigned char key, int x, int y) : Keyboard function for padlock and key.

4.1.7. void numberLock() : Function that draws number lock.

4.1.8. void displayNumberLock() : Function that redraws the window.

4.1.9. void animateNumberLock() : Idle function for number lock.

4.1.10. void numberLockKeyboard(unsigned char key, int x, int y) : Keyboard function for lock and key.

4.1.11. void myreshape(int w, int h) : Handles program on window resize.

4.1.12. void mouse(int btn, int state, int x, int y) : Function that handles mouse interactions

4.1.13. int main(int argc, char argv) :** Main Program

4.3. APIs USED:

4.3.1. GLUT APIs

4.3.1.1. glutInit ()

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

glutInit is used to initialize the GLUT library.

4.3.1.2. glutInit(WindowPosition, glutInitWindowSize)

void glutInitWindowSize(int width, int height);

void glutInitWindowPosition(int x, int y);

glutInitWindowPosition and glutInitWindowSize set the initial window position and size respectively. Width, height, x, y is specified in pixels.

4.3.1.3. glutInitDisplayMode()

void glutInitDisplayMode(unsigned int mode);

glutInitDisplayMode is used to specify the display mode.

Mode is used to specify the mode. Some of the symbolic constants that can be used here are: GLUT_RGB, GLUT_RGBA, GLUT_SINGLE, and GLUT_DOUBLE.

4.3.1.4. glutMainLoop()

void glutMainLoop(void);

glutMainLoop enters the GLUT event processing loop.

4.3.1.5. glutCreateWindow()

int glutCreateWindow(char *name);

glutCreateWindow is used to create top-level window

name is the ACSII character string used as name of the window.

4.3.1.6. glutPostRedisplay()

void glutPostRedisplay(void);

glutPostRedisplay marks the current window as needing to be redisplayed.

4.3.1.7. glutSwapBuffers()

void glutSwapBuffers(void);

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

4.3.1.8. glutDisplayFunc()

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

glutDisplayFunc sets the display callback for the current window.

func is the new display callback function.

4.3.1.9. glutReshapeFunc()

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

glutReshapeFunc sets the reshape callback for the current window.

func is the new reshape callback function.

4.3.1.10. glutKeyboardFunc ()

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

glutKeyboardFunc sets the keyboard callback for the current window.

func is the new callback function.

4.3.1.11. glutMouseFunc()

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

glutMouseFunc sets the mouse callback for the current window.

func is the new mouse callback function

4.3.2 GL APIs

4.3.2.1 glClearColor()

void glClearColor(GLClampf red, GLClampf green, GLClampf blue, GlClampf alpha);

To specify clear values for color buffer.

Initial values are all 1.

4.3.2.2. glClear()

void glClear(GLbitfield mask);

Bitwise OR of masks that indicate the buffers to be cleared.

masks are GL_COLOR_BUFFER_BIT and GL_DEPTH_BUFFER_BIT.

4.3.2.3. glFlush()

void glFlush(void);

glFlush empties all of these buffers, causing all issued commands to be executed as quickly as they are accepted.

force execution of GL commands in finite time.

4.3.2.4. glMatrixMode()

void glMatrixMode(GLenum mode);

Specify which matrix is the current matrix.

Accepted values GL_MODELVIEW, GL_PROJECTION, and GL_TEXTURE.

4.3.2.5. glScale*

void glScaled(GLdouble x, GLdouble y, GLdouble z);

void glScalef(GLfloat x, GLfloat y, GLfloat z);

multiply the current matrix by a general scaling matrix.

Specify scale factors along the x, y, and z axes, respectively.

4.3.2.6. glTranslate*

void glScaled(GLdouble x, GLdouble y, GLdouble z);

void glScalef(GLfloat x, GLfloat y, GLfloat z);

multiply the current matrix by a general translational matrix.

Specify the x, y, and z coordinates of a translation vector.

4.3.2.7. glPushMatrix()

void glPushMatrix(void);

push the current matrix stack.

4.3.2.8. glPopMatrix ()

void glPopMatrix(void);

pop the current matrix stack.

4.3.2.9. glVertex*

void glVertexntv(GLtype v1,…, GLtype vn);

n is the number of vertices.

t is the type of the variables.

4.3.2.10. glColor*

void glColorntv(GLtype v1,…, GLtype vn);

n is the number of vertices,

t is the type of the variables.

V is optional in case of vector input.

CHAPTER 5

SAMPLE CODE

```
#include <stdio.h>
#include <stdlib.h>
#include <GL/glut.h>
#define KEY_DISTANCE 1.5

int gameMode = -1; // 0 - pad lock, 1 - number lock
int won = 0;

GLfloat keyZ = KEY_DISTANCE;
GLfloat keyTheta = 0;
GLfloat lockBarY = 0;
int lockAndKeyThetaX = 0;
int lockAndKeyThetaZ = 0;
int keyPosition = 0;
int lockAndKeyAction = -1; // 0 - outside, 1 - inside the lock

int keySphereThetaX = 0;
int keySphereThetaY = 0;

int numberLockPosition = 0;
int numberLockThetaX = 0;
int numberLockThetaZ = 0;

int ansOne = 1;
int ansTwo = 4;
int ansThree = 2;

int currentFaceOne = 0;
```

```
int currentFaceTwo = 0;
int currentFaceThree = 0;

int cylinderThetaOne = 0;
int cylinderThetaTwo = 0;
int cylinderThetaThree = 0;
void drawDetails(char ch[] , float x,float y)
{
    int i;
    glRasterPos2f(x,y);

    for(i = 0; ch[i] != '\0'; i++)
    {
        glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24,ch[i]);
    }
}

void intro()
{
     glClearColor(1,0.8,0.3,0);
     glClear( GL_DEPTH_BUFFER_BIT | GL_COLOR_BUFFER_BIT);
        glColor3f(0,0.5,0.5);
        drawDetails("COMPUTER GRAPHICS AND VISUALIZATION MINI
PROJECT",-1.5,1.5);
        drawDetails("PROJECT : LOCK AND KEY GAME",-0.8,1);
        drawDetails("NAME : ABHISHEK S",-0.8,0.2);
        drawDetails("USN : 3GN16CS002",-0.8,0.0);
        drawDetails("NAME : KESHAV POLA",-0.8,0.6);
        drawDetails("USN : 3GN16CS029",-0.8,0.4);
}
```

```
GLfloat cubeVertices[][3] = {
    {-0.5, -0.5, -0.5},
     {0.5, -0.5, -0.5},
     {0.5, 0.5, -0.5},
     {-0.5, 0.5, -0.5},
     {-0.5, -0.5, 0.5},
     {0.5, -0.5, 0.5},
     {0.5, 0.5, 0.5},
     {-0.5, 0.5, 0.5}
};

GLfloat lockBarVertices[][3] = {
        {-0.54643, -0.70357, 0},
        {-0.42143, -0.57857, 0.25},
        {-0.29643, -0.45357, 0},
        {-0.42143, -0.57857, -0.25},
        {0.68927, 0.06073, 0},
        {0.81427, 0.18573, 0.25},
        {0.93927, 0.31073, 0},
        {0.81427, 0.18573, -0.25},
        {-1.25, 0, 0},
        {-1.0, 0, 0.25},
        {-0.75, 0, 0},
        {-1.0, 0, -0.25},
        {0, 0.75, 0},
        {0, 1, 0.25},
        {0, 1.25, 0},
        {0, 1, -0.25}
};

GLfloat keyPentagonBodyVertices[][3] = {
```

```
        {0.58779, -0.80902, 1},
        {0.95106, 0.30902, 1},
        {0, 1, 1},
        {-0.95106, 0.30902, 1},
        {-0.58779, -0.80902, 1},
        {0.58779, -0.80902, -1},
        {0.95106, 0.30902, -1},
       {0, 1, -1},
       {-0.95106, 0.30902, -1},
       {-0.58779, -0.80902, -1},
};

GLfloat numberCylinderVertices[][3] = {
       {-0.86603, 0.5, -1.0},
       {0, 1, -1.0},
       {0.86603, 0.5, -1.0},
       {0.86603, -0.5, -1.0},
       {0, -1, -1.0},
       {-0.86603, -0.5, -1.0},
       {-0.86603, 0.5, 1.0},
       {0, 1, 1.0},
       {0.86603, 0.5, 1.0},
       {0.86603, -0.5, 1.0},
       {0, -1, 1.0},
       {-0.86603, -0.5, 1.0}
};

GLfloat cylinderFaceColors[][3] = {
    {0.90196, 0.29020, 0.09804},
    {0.96078, 0.48627, 0.00000},
    {1.00000, 0.62745, 0.00000},
    {0.98431, 0.75294, 0.17647},
    {0.68627, 0.70588, 0.16863},
    {0.40784, 0.62353, 0.21961}
```

```
};

void lockBarFace(int a, int b, int c, int d) {
      glBegin(GL_POLYGON);
         glVertex3fv(lockBarVertices[a]);
         glVertex3fv(lockBarVertices[b]);
         glVertex3fv(lockBarVertices[c]);
         glVertex3fv(lockBarVertices[d]);
      glEnd();
}

void lockBar(){
    glPushMatrix();
    glTranslated(0, 0.3, 0);
    glScaled(0.5, 0.5, 0.5);
    glRotated(-45, 0, 0, 1);
    glColor3d(0.8, 0.8, 0.8);
    lockBarFace(0, 1, 2, 3);
    glColor3d(0.8, 0.8, 0.8);
    lockBarFace(0, 8, 9, 1);
    glColor3d(0.45, 0.45, 0.45);
    lockBarFace(0, 8, 11, 3);
    glColor3d(0.8, 0.8, 0.8);
    lockBarFace(3, 11, 10, 2);
    glColor3d(0.5, 0.5, 0.5);
    lockBarFace(1, 9, 10, 2);
    glColor3d(0.55, 0.55, 0.55);
    lockBarFace(9, 13, 12, 10);
    glColor3d(0.85, 0.85, 0.85);
    lockBarFace(9, 8, 14, 13);
    glColor3d(0.85, 0.85, 0.85);
    lockBarFace(8, 14, 15, 11);
    glColor3d(0.55, 0.55, 0.55);
    lockBarFace(11, 15, 12, 10);
```

```
    glColor3d(0.8, 0.8, 0.8);
    lockBarFace(4, 12, 13, 5);
    glColor3d(0.45, 0.45, 0.45);
    lockBarFace(5, 13, 14, 6);
    glColor3d(0.8, 0.8, 0.8);
    lockBarFace(6, 14, 15, 7);
    glColor3d(0.5, 0.5, 0.5);
    lockBarFace(7, 15, 12, 4);
    glColor3d(0.6, 0.6, 0.6);
    lockBarFace(4, 5, 6, 7);
    glPopMatrix();
}

// -----------------
// LOCK AND KEY CODE
// -----------------
void cubeFace(int a, int b, int c, int d) {
     glBegin(GL_POLYGON);
        glVertex3fv(cubeVertices[a]);
        glVertex3fv(cubeVertices[b]);
        glVertex3fv(cubeVertices[c]);
        glVertex3fv(cubeVertices[d]);
     glEnd();
}

void lockCube(){
    glColor3d(1, 0, 0);
    glPushMatrix();
     glScaled(0.9, 0.9, 0.5);
     glPushMatrix();

     glColor3d(0.9, 0.3, 0);
     cubeFace(0, 3, 2, 1);
```

```
      glColor3d(0.9, 0.3, 0);
      cubeFace(2, 3, 7, 6);

      glColor3d(0.8, 0.3, 0);
      cubeFace(0, 4, 7, 3);

      glColor3d(0.95, 0.3, 0);
      cubeFace(1, 2, 6, 5);

      glColor3d(1, 0.3, 0);
      cubeFace(4, 5, 6, 7);

      glColor3d(0.75, 0.3, 0);
      cubeFace(0, 1, 5, 4);

      glPopMatrix();
      glPopMatrix();
}

void lockFrontDecagon(){
    glColor3d(1, 1, 0);
    glPushMatrix();
    glScaled(0.3, 0.3, 0.5);
    glTranslated(0, -0.505, 0.5002);
    glBegin(GL_POLYGON);
        glVertex3d(0, 1, 0);
        glVertex3d(0.58778, 0.80902, 0);
        glVertex3d(0.95106, 0.30902, 0);
        glVertex3d(0.95106, -0.30902, 0);
        glVertex3d(0.58778, -0.80902, 0);
        glVertex3d(0, -1, 0);
        glVertex3d(-0.58778, -0.80902, 0);
        glVertex3d(-0.95106, -0.30902, 0);
        glVertex3d(-0.95106, 0.30902, 0);
```

```
        glVertex3d(-0.58778, 0.80902, 0);
    glEnd();
    glPopMatrix();
}

void lockKeyholePentagon(){
    glColor3d(0, 0, 0);
    glPushMatrix();
    glScaled(0.1, 0.1, 0.5);
    glTranslated(0, -0.415, 0.50040);
    glBegin(GL_POLYGON);
        glVertex3d(0, 1, 0);
        glVertex3d(0.95106, 0.30902, 0);
        glVertex3d(0.58779, -0.80902, 0);
        glVertex3d(0.3, -0.80902, 0);
        glVertex3d(0.3, -3.35, 0);
        glVertex3d(-0.3, -3.35, 0);
        glVertex3d(-0.3, -0.80902, 0);
        glVertex3d(-0.58779, -0.80902, 0);
        glVertex3d(-0.95106, 0.30902, 0);
    glEnd();
    glPopMatrix();

}

void lock(){
    glPushMatrix();
    lockCube();
    lockFrontDecagon();
    lockKeyholePentagon();
    glPushMatrix();
    glTranslated(0, lockBarY, 0);
    lockBar();
    glPopMatrix();
```

```
    glPopMatrix();
}

void keyBodyPentagonFourFace(int a, int b, int c, int d) {
     glBegin(GL_POLYGON);
        glVertex3fv(keyPentagonBodyVertices[a]);
        glVertex3fv(keyPentagonBodyVertices[b]);
        glVertex3fv(keyPentagonBodyVertices[c]);
        glVertex3fv(keyPentagonBodyVertices[d]);
     glEnd();
}

void keyBodyPentagonFiveFace(int a, int b, int c, int d, int e) {
     glBegin(GL_POLYGON);
        glVertex3fv(keyPentagonBodyVertices[a]);
        glVertex3fv(keyPentagonBodyVertices[b]);
        glVertex3fv(keyPentagonBodyVertices[c]);
        glVertex3fv(keyPentagonBodyVertices[d]);
        glVertex3fv(keyPentagonBodyVertices[e]);
     glEnd();
}

void keyBodyPentagon(){
    glPushMatrix();
    glScaled(0.09, 0.09, 0.7);
    glTranslated(0, -0.415, 0.9);
    keyBodyPentagonFiveFace(0, 1, 2, 3, 4);

    glColor3d(0.7, 0.7, 0.7);
    keyBodyPentagonFourFace(0, 1, 6, 5);

    glColor3d(0.8, 0.8, 0.8);
     keyBodyPentagonFourFace(1, 2, 7, 6);
```

```
    glColor3d(0.7, 0.7, 0.7);
    keyBodyPentagonFourFace(2, 3, 8, 7);

    glColor3d(0.6, 0.6, 0.6);
    keyBodyPentagonFourFace(3, 4, 9, 8);

    glColor3d(0.5, 0.5, 0.5);
    keyBodyPentagonFourFace(4, 0, 5, 9);

    keyBodyPentagonFiveFace(5, 6, 7, 8, 9);
    glPopMatrix();
}

void keyBlockOne(float r, float g, float b){
    glColor3f(r, g, b);
    glPushMatrix();
    glScaled(0.05, 0.32, 0.1);
    glTranslated(0, -0.6, -0.1);
     glutSolidCube(1.0);
    glPopMatrix();
}

void keyBlockTwo(float r, float g, float b){
    glColor3f(r, g, b);
    glPushMatrix();
    glScaled(0.05, 0.22, 0.1);
    glTranslated(0, -0.6, 1.1);
     glutSolidCube(1.0);
    glPopMatrix();
}
```

```
void keyBlockThree(float r, float g, float b){
    glColor3f(r, g, b);
    glPushMatrix();
    glScaled(0.05, 0.26, 0.1);
    glTranslated(0, -0.6, 2.3);
     glutSolidCube(1.0);
    glPopMatrix();
}

void keyBlockFour(float r, float g, float b){
    glColor3f(r, g, b);
    glPushMatrix();
    glScaled(0.05, 0.32, 0.1);
    glTranslated(0, -0.6, 3.5);
     glutSolidCube(1.0);
    glPopMatrix();
}

void rightKeyHandle(){
    glColor3f(0.7, 0.7, 0.7);
    glPushMatrix();
    glTranslated(0, -0.04, 1.35);
    glutSolidSphere(0.05, 30, 30);
    glPopMatrix();

    glColor3f(0.4, 1, 0.9);
    glPushMatrix();
    glTranslated(0, -0.04, 1.5);
    glRotated(90, 0, 1, 0);
    glScaled(0.2,0.2,0.2);
    glutSolidTorus(0.15, 0.7, 30, 30);
```

```
    glPopMatrix();
}

void wrongKeyHandle(){
    glColor3f(0.7, 0.7, 0.7);
    glPushMatrix();
    glTranslated(0, -0.04, 1.35);
    glutSolidSphere(0.05, 30, 30);
    glPopMatrix();

    glColor3f(0.9, 0.1, 0.1);
    glPushMatrix();
    glTranslated(0, -0.04, 1.5);
    glRotated(90, 0, 1, 0);
    glScaled(0.2,0.2,0.2);
    glutSolidTorus(0.15, 0.7, 30, 30);
    glPopMatrix();
}

void rightKeyBlocks(){
    glPushMatrix();
        keyBlockOne(0.1, 0.9, 0.1);
        keyBlockTwo(0.1, 0.8, 0.1);
        keyBlockThree(0.1, 0.7, 0.1);
        keyBlockFour(0.1, 0.6, 0.1);
    glPopMatrix();
}

void wrongKeyBlocks(){
    glColor3d(0.6, 0.6, 0.6);
    glPushMatrix();
```

```
        keyBlockOne(0.9, 0.1, 0.1);
        keyBlockTwo(0.8, 0.1, 0.1);
        keyBlockThree(0.7, 0.1, 0.1);
        keyBlockFour(0.6, 0.1, 0.1);
    glPopMatrix();
}

void rightKey(){
    glPushMatrix();
    keyBodyPentagon();
    rightKeyBlocks();
    rightKeyHandle();
    glPopMatrix();
}

void wrongKey(){
    glPushMatrix();
    keyBodyPentagon();
    wrongKeyBlocks();
    wrongKeyHandle();
    glPopMatrix();
}

void animateKey(){
    if(lockAndKeyAction == 0){
        if(keyZ >= -0.175){
            keyZ -= 0.005;
        }
        if(keyZ < -0.175){
            keyPosition = 1;
        }
```

```
        if(keyPosition == 1 && keyTheta <= 360){
            keyTheta+= 1;
        }
        if(keyTheta>360){
            keyPosition = 2;
        }
        if(keyPosition == 2 && lockBarY <= 0.35){
            lockBarY+=0.001;
        }
    }

    if(lockAndKeyAction == 1){
        if(keyPosition == 2 && lockBarY >= 0){
            lockBarY-=0.001;
        }
        if(lockBarY<0){
            keyPosition = 1;
        }
        if(keyPosition == 1 && keyTheta >=0) {
            keyTheta -= 1;
        }
        if(keyTheta < 0){
            keyPosition = 0;
        }
        if(keyPosition == 0 && keyZ <= KEY_DISTANCE){
            keyZ += 0.005;
        }
    }

    glutPostRedisplay();
}
```

```
void keySphere(){

    //place right key
    glPushMatrix();
        glTranslated(0, 0, keyZ);
        glRotated(-keyTheta, 0, 0, 1); // TODO - FIX ROTATION
        rightKey();
    glPopMatrix();

    //place wrong keys
    if(won == 0){

        glPushMatrix();
            glRotated(90, 1, 0, 0);
            glPushMatrix();
                glTranslated(0, 0, keyZ);
                glRotated(90, 0, 0, 1);
                wrongKey();
            glPopMatrix();
        glPopMatrix();

        glPushMatrix();
            glRotated(90, 0, 1, 0);
            glPushMatrix();
                glTranslated(0, 0, keyZ);
                glRotated(90, 0, 0, 1);
                wrongKey();
            glPopMatrix();
        glPopMatrix();
```

```
        glPushMatrix();
            glRotated(270, 0, 1, 0);
            glPushMatrix();
                glTranslated(0, 0, keyZ);
                glRotated(90, 0, 0, 1);
                wrongKey();
            glPopMatrix();
        glPopMatrix();

        glPushMatrix();
            glRotated(180, 1, 0, 0);
            glPushMatrix();
                glTranslated(0, 0, keyZ);
                glRotated(90, 0, 0, 1);
                wrongKey();
            glPopMatrix();
        glPopMatrix();



        glPushMatrix();
            glRotated(270, 1, 0, 0);
            glPushMatrix();
                glTranslated(0, 0, keyZ);
                glRotated(90, 0, 0, 1);
                wrongKey();
            glPopMatrix();
        glPopMatrix();

    }
}
```

```
void displayLockAndKey(){

    glPushMatrix();
     glRotated(lockAndKeyThetaZ, 0, 1, 0);
     glRotated(lockAndKeyThetaX, 1, 0, 0);
        glPushMatrix();
            //glScaled(1.5, 1.5, 1.5);
            glPushMatrix();
            lock();
            glPopMatrix();

            glPushMatrix();
                glRotated(keySphereThetaX, 1, 0, 0);
                glPushMatrix();
                    glRotated(keySphereThetaY, 0, 1, 0);
                    keySphere();
                glPopMatrix();
            glPopMatrix();

        glPopMatrix();
     glPopMatrix();
}

void lockAndKeyKeyboard( unsigned char key, int x, int y){
    if(key == 'u' && won == 0 && keySphereThetaX == 0 &&
keySphereThetaY == 0){
        won = 1;
        keyZ = KEY_DISTANCE;
        keyTheta = 0;
        lockBarY = 0;
        keyPosition = 0;
```

```
        lockAndKeyAction = 0;
        glutPostRedisplay();
    }

    if(key == 'u' && won == 1){
        keyZ = KEY_DISTANCE;
        keyTheta = 0;
        lockBarY = 0;
        keyPosition = 0;
        lockAndKeyAction = 0;
        glutPostRedisplay();
    }
    if(key == 'l' && won == 1){
        keyZ = -0.175;
        keyTheta = 360;
        lockBarY = 0.35;
        keyPosition = 2;
        lockAndKeyAction = 1;
        glutPostRedisplay();
    }

    if(key == 'w'){
        --lockAndKeyThetaX;
    }
    if(key == 's'){
        ++lockAndKeyThetaX;
    }
    if(key == 'a'){
        --lockAndKeyThetaZ;
    }
    if(key == 'd'){
```

```
        ++lockAndKeyThetaZ;
    }

    if(key == 'i' && won == 0){
        keySphereThetaX = (keySphereThetaX-5) % 360;
    }
    if(key == 'k' && won == 0){
        keySphereThetaX = (keySphereThetaX+5) % 360;
    }
    if(key == 'j' && won == 0){
        keySphereThetaY = (keySphereThetaY-5) % 360;
    }
    if(key == 'l' && won == 0){
        keySphereThetaY = (keySphereThetaY+5) % 360;
    }
    if(key == 'q'){
        exit(0);
    }
}
// -----------------
// LOCK AND KEY CODE
// -----------------

// -----------------
// NUMBER LOCK CODE
// -----------------
void numberLockCube(){
    glColor3d(0.6, 0.3, 0.9);
    glPushMatrix();
     glScaled(0.9, 0.9, 0.5);
```

```
    glColor3d(0.6, 0.3, 0.9);
    cubeFace(0, 3, 2, 1);

    glColor3d(0.65, 0.35, 0.95);
    cubeFace(2, 3, 7, 6);

    glColor3d(0.5, 0.2, 0.8);
    cubeFace(0, 4, 7, 3);

    glColor3d(0.7, 0.4, 1);
    cubeFace(1, 2, 6, 5);

    glColor3d(0.6, 0.3, 0.9);
    cubeFace(4, 5, 6, 7);

    glColor3d(0.45, 0.15, 0.75);
    cubeFace(0, 1, 5, 4);

    glPopMatrix();
}

void numberCylinderFourFace(int a, int b, int c, int d) {
    glBegin(GL_POLYGON);
       glVertex3fv(numberCylinderVertices[a]);
       glVertex3fv(numberCylinderVertices[b]);
       glVertex3fv(numberCylinderVertices[c]);
       glVertex3fv(numberCylinderVertices[d]);
    glEnd();
}
```

```
void numberCylinderSixFace(int a, int b, int c, int d, int e,
int f) {
     glBegin(GL_POLYGON);
        glVertex3fv(numberCylinderVertices[a]);
        glVertex3fv(numberCylinderVertices[b]);
        glVertex3fv(numberCylinderVertices[c]);
        glVertex3fv(numberCylinderVertices[d]);
        glVertex3fv(numberCylinderVertices[e]);
        glVertex3fv(numberCylinderVertices[f]);
     glEnd();
}

void numberCylinder(){
    glColor3d(0.6, 0.8, 0.6);
    glPushMatrix();
    glColor3fv(cylinderFaceColors[0]);
    numberCylinderFourFace(0, 1, 7, 6);
    glColor3fv(cylinderFaceColors[1]);
    numberCylinderFourFace(1, 2, 8, 7);
    glColor3fv(cylinderFaceColors[2]);
    numberCylinderFourFace(2, 3, 9, 8);
    glColor3fv(cylinderFaceColors[3]);
    numberCylinderFourFace(3, 4, 10, 9);
    glColor3fv(cylinderFaceColors[4]);
    numberCylinderFourFace(4, 5, 11, 10);
    glColor3fv(cylinderFaceColors[5]);
    numberCylinderFourFace(5, 0, 6, 11);
    glColor3d(0.6, 0.6, 0.6);
    numberCylinderSixFace(0, 1, 2, 3, 4, 5);
    numberCylinderSixFace(6, 7, 8, 9, 10, 11);
    glPopMatrix();
```

```
}

void numberCylinderOne(){
    glPushMatrix();
    glScaled(0.15, 0.1, 0.15);
    glTranslated(2, 3, 1.25);
    glRotated(30 - cylinderThetaOne, 0, 1, 0);
    glRotated(90, 1, 0, 0);
    numberCylinder();
    glPopMatrix();
}

void numberCylinderTwo(){
    glPushMatrix();
    glScaled(0.15, 0.1, 0.15);
    glTranslated(2, 0, 1.25);
    glRotated(30 - cylinderThetaTwo, 0, 1, 0);
    glRotated(90, 1, 0, 0);
    numberCylinder();
    glPopMatrix();
}

void numberCylinderThree(){
    glPushMatrix();
    glScaled(0.15, 0.1, 0.15);
    glTranslated(2, -3, 1.25);
    glRotated(30 - cylinderThetaThree, 0, 1, 0);
    glRotated(90, 1, 0, 0);
    numberCylinder();
    glPopMatrix();
}
```

```
void numberCylinders(){
    glPushMatrix();
    numberCylinderOne();
    numberCylinderTwo();
    numberCylinderThree();
    glPopMatrix();
}

void guessSphereOne(){
    glPushMatrix();
    glTranslated(-0.3, 0, 0.25);
    glColor3fv(cylinderFaceColors[1]);
    glutSolidSphere(0.05, 10, 10);
    glPopMatrix();
}

void guessSphereTwo(){
    glPushMatrix();
    glTranslated(-0.15, 0, 0.25);
    glColor3fv(cylinderFaceColors[4]);
    glutSolidSphere(00.05, 10, 10);
    glPopMatrix();
}

void guessSphereThree(){
    glPushMatrix();
    glTranslated(0, 0, 0.25);
    glColor3fv(cylinderFaceColors[2]);
    glutSolidSphere(0.05, 10, 10);
    glPopMatrix();
```

```
}

void guessSpheres(){
    glPushMatrix();
    guessSphereOne();
    guessSphereTwo();
    guessSphereThree();
    glPopMatrix();
}

void numberLock(){
    glPushMatrix();
    numberLockCube();
    numberCylinders();
    guessSpheres();

    glPushMatrix();
    // FINAL LOCATION
    glTranslated(0, lockBarY, 0);
    lockBar();
    glPopMatrix();

    glPopMatrix();
}

void displayNumberLock(){
    glPushMatrix();
     glRotated(numberLockThetaZ, 0, 1, 0);
     glRotated(numberLockThetaX, 1, 0, 0);
     glPushMatrix();
     glScaled(1.5, 1.5, 1.5);
```

```
     numberLock();
     glPopMatrix();
     glPopMatrix();
}

void animateNumberLock(){
    if(currentFaceOne == ansOne && currentFaceTwo == ansTwo &&
currentFaceThree == ansThree && lockBarY <= 0.35){
        numberLockPosition = 1;
    }
    if(numberLockPosition == 1 && lockBarY <= 0.35){
        lockBarY += 0.001;
    }
    if(lockBarY > 0.35 && (currentFaceOne != ansOne ||
currentFaceTwo != ansTwo || currentFaceThree != ansThree)){
        numberLockPosition = 0;
    }
    if(numberLockPosition == 0 && lockBarY >0){
        lockBarY -= 0.001;
    }
    glutPostRedisplay();
}

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

    // CYLINDER ONE KEY HANDLER
    if(key == '7'){
        cylinderThetaOne += 60;
        currentFaceOne = (++currentFaceOne) % 6;
        printf("current face of cylinder 1 : %d\n",
currentFaceOne);
```

```
    }
    if(key == '9'){
        cylinderThetaOne -= 60;
        currentFaceOne = (--currentFaceOne) % 6;
        if(currentFaceOne == -1) currentFaceOne = 5;
        printf("current face of cylinder 1 : %d\n",
currentFaceOne);
    }

    // CYLINDER TWO KEY HANDLER
    if(key == '4'){
        cylinderThetaTwo += 60;
        currentFaceTwo = (++currentFaceTwo) % 6;
        printf("current face of cylinder 2 : %d\n",
currentFaceTwo);
    }
    if(key == '6'){
        cylinderThetaTwo -= 60;
        currentFaceTwo = (--currentFaceTwo) % 6;
        if(currentFaceTwo == -1) currentFaceTwo = 5;
        printf("current face of cylinder 2 : %d\n",
currentFaceTwo);
    }

    // CYLINDER THREE KEY HANDLER
    if(key == '1'){
        cylinderThetaThree += 60;
        currentFaceThree = (++currentFaceThree) % 6;
        printf("current face of cylinder 3 : %d\n",
currentFaceThree);
    }
```

```
    if(key == '3'){
        cylinderThetaThree -= 60;
        currentFaceThree = (--currentFaceThree) % 6;
        if(currentFaceThree == -1) currentFaceThree = 5;
        printf("current face of cylinder 3 : %d\n",
currentFaceThree);
    }

    if(key == 'w'){
        --numberLockThetaX;
    }
    if(key == 's'){
        ++numberLockThetaX;
    }
    if(key == 'a'){
        --numberLockThetaZ;
    }
    if(key == 'd'){
        ++numberLockThetaZ;
    }
    if(key == 'q'){
        exit(0);
    }



    glutPostRedisplay();
}
// -----------------
// NUMBER LOCK CODE
// -----------------
```

```
void mouse(int btn, int state, int x, int y){
    if(btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN){
        gameMode = 0;
    }
    if(btn == GLUT_RIGHT_BUTTON && state == GLUT_DOWN){
        gameMode = 1;
    }
    glutPostRedisplay();
}

void display(void) {
    glClearColor(0.9, 1, 1.0, 1.0);
     glClear( GL_DEPTH_BUFFER_BIT | GL_COLOR_BUFFER_BIT);

     if(gameMode==-1)
    {
        intro();
    }
    if(gameMode == 0){
        glClearColor(1, 1,.5, 1.0);
     glClear( GL_DEPTH_BUFFER_BIT | GL_COLOR_BUFFER_BIT);
        displayLockAndKey();
        glutKeyboardFunc(lockAndKeyKeyboard);
        glutIdleFunc(animateKey);
    } if(gameMode==1) {
        displayNumberLock();
        glutKeyboardFunc(numberLockKeyboard);
        glutIdleFunc(animateNumberLock);
    }
     glutSwapBuffers();
    glFlush();
```

```
}
void myreshape(int w,int h) {
     glViewport(0,0,w,h);
     glMatrixMode(GL_PROJECTION);
     glLoadIdentity();
     if(w<=h)
           glOrtho(-2.0,2.0,-
2.0*(GLfloat)h/(GLfloat)w,2.0*(GLfloat)h/(GLfloat)w,-10.0,10.0);
     else
           glOrtho(-
2.0*(GLfloat)w/(GLfloat)h,2.0*(GLfloat)w/(GLfloat)h,-2.0,2.0,-
10.0,10.0);
     glMatrixMode(GL_MODELVIEW);
    glutPostRedisplay();
}

int main(int argc, char **argv) {
     glutInit(&argc, argv);
     glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
     glutInitWindowSize(500,500);
     glutInitWindowPosition(0, 35);
     glutCreateWindow("lock and key game");
     glutMouseFunc(mouse);
     glutDisplayFunc(display);
     glutReshapeFunc(myreshape);
    glEnable(GL_DEPTH_TEST);
     glEnable(GL_NORMALIZE);
     glutFullScreen();
     glutMainLoop();
     return 0;
}
```

CHAPTER 6

SAMPLE OUTPUT

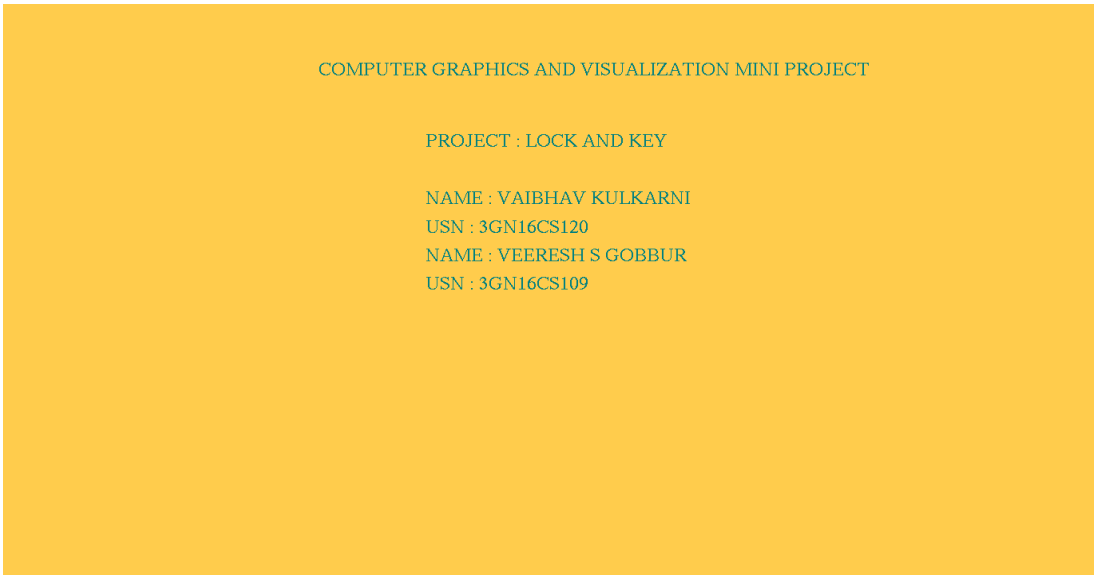


Fig 6.1 Front Page

This is the Landing page that will be visible to the application user upon launching of application.

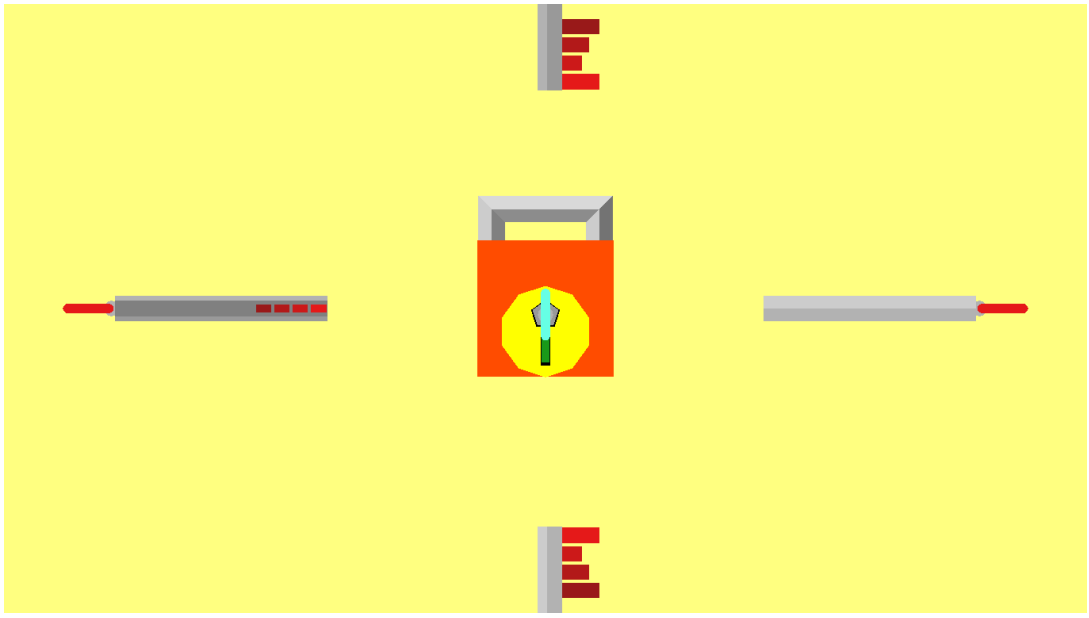


Fig 6.2 Padlock in locked state

The padlock is initially in the locked state. In this state, there are multiple keys and the lock bar lies half way inside the lock.

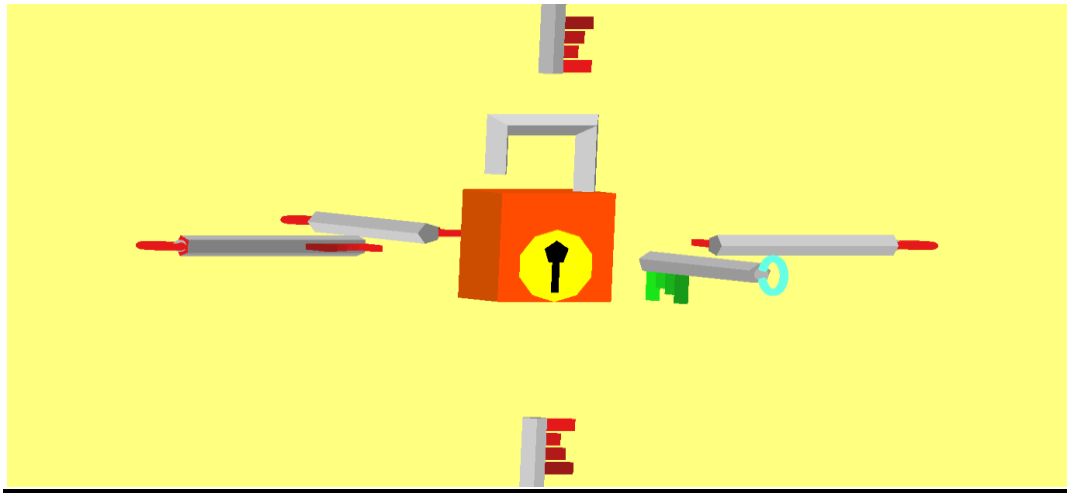


Fig 6.3 Padlock in unlocked state

Upon triggering the locking action, the key is translated into the keyhole of the lock And then is rotated clockwise to unlock the padlock.

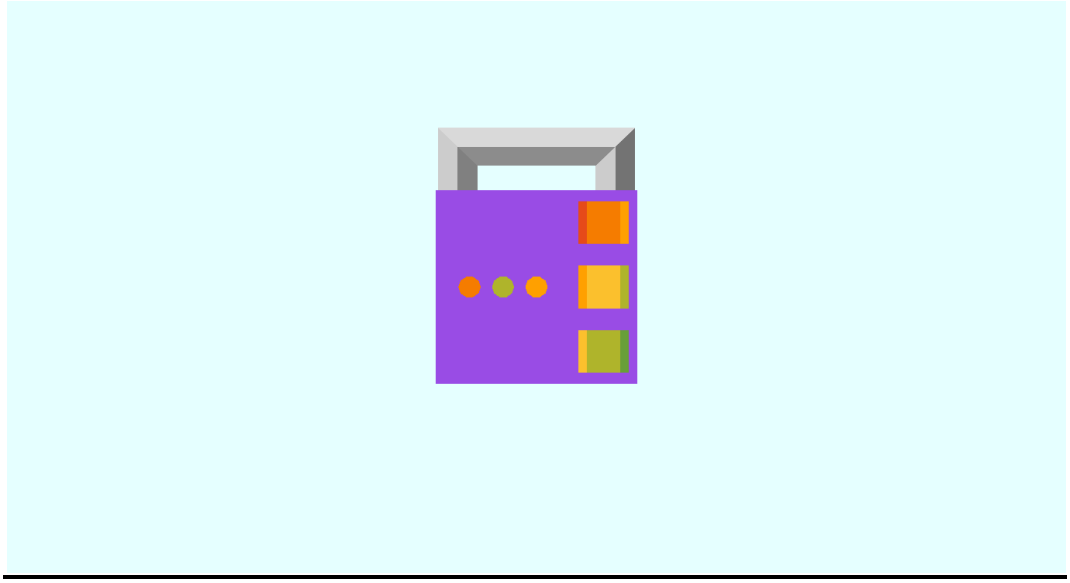


Fig 6.4 pattern lock in locked state

The pattern lock consists of three cylinders with each cylinder having six patterns, Which can be rotated using number keys.

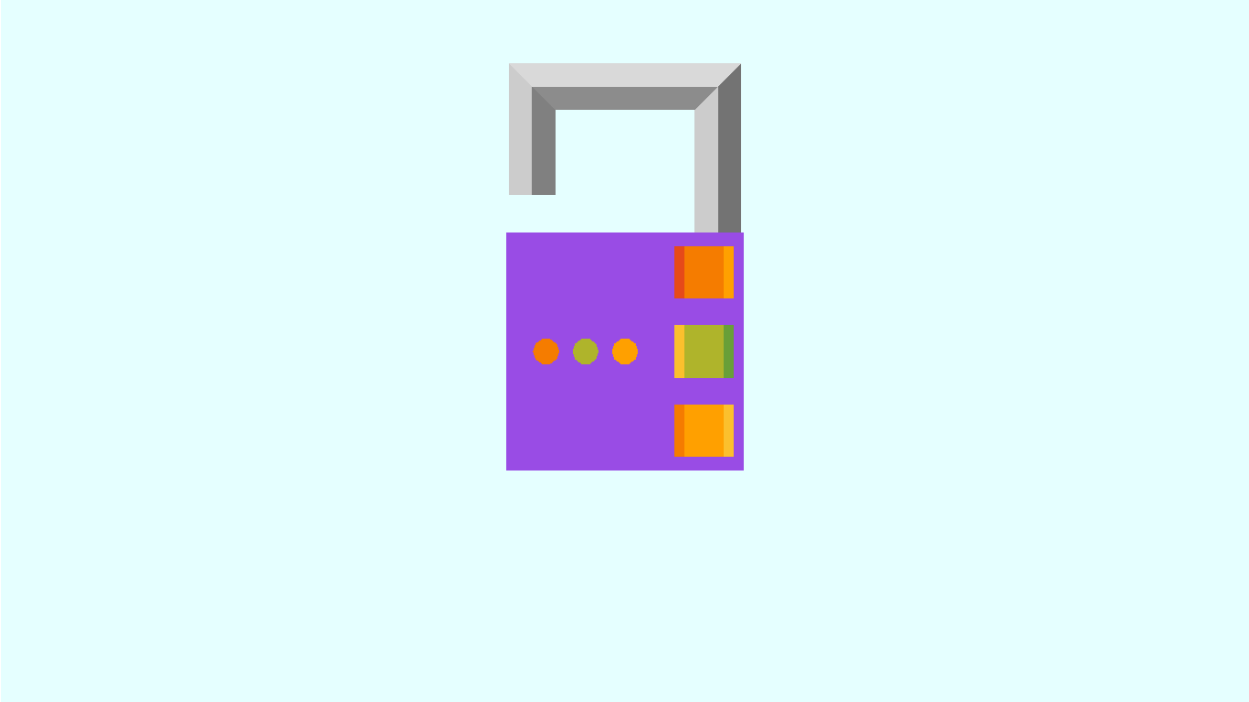


Fig 6.5 Pattern lock in unlocked state

Upon selecting the correct pattern, the unlocking action is triggered, where the lock bar Is translated upwards to enter the unlocking state.

CHAPTER 7

COST ESTIMATION

Project cost estimating scares a lot of people. They don't know how much something will cost, but they know whatever value they give, they will be held to it by their manager.

The challenge with estimating is that it always involves some uncertainty. Some of the factors that contribute to this uncertainty include...

• **Experience with similar projects**: The less experience you have with similar projects, the greater the uncertainty. If you've managed similar projects, you will be able to better estimate the costs of the project.

• **Planning Horizon:** The longer the planning horizon, the greater the uncertainty. The planning horizon you are considering may be the whole project or just a certain phase. Either way, you will be able to better estimate costs for the time periods that are closer to the present.

• **Project Duration:** The longer the project, the greater the uncertainty. This is similar to planning horizon in the sense that if a project is of a shorter duration you are more likely to account for most of the costs.

• **People:** The quantity of people and their skill will be a huge factor in estimating their costs. Early in the project, you may not even know the specific people that will be on the project. That will increase the uncertainty of your cost estimates.

TABULAR COLUMN:

SPECIFICATIONS	AMOUNT
i. Software Requirements	400
ii. Hardware Requirements	200
iii. Report	200
iv. Miscellaneous	400
Total	1200

CHAPTER 8

CONCLUSION

The objective of this project was to develop a final product which enhanced our knowledge with the wonders of OpenGL and successfully implement a 3D simulation which contained many concepts that are used during game design and development in the real world. With Lock and Key we are proud to say that we have accomplished the above objective along with gaining knowledge of the various API's and functions involved with OpenGL and its accompanying libraries.

We have imbibed in this project a variety of concepts and hope to further enhance this project by making it more robust, adding more ways the user can interact with the game and adding multiple other types of locks.

CHAPTER 9

BIBLIOGRAPHY

BOOK REFERENCES

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

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

WEBSITE REFERENCES

[1] - https://www.khronos.org/opengl/wiki/Getting_Started

[2] - https://www.opengl.org/sdk/docs/tutorials/OGLSamples/

[3] - https://www.geeksforgeeks.org/getting-started-with-opengl/

[4] - https://www.khronos.org/opengl/wiki//Code_Resources

[5] - http://www.lighthouse3d.com/tutorials/glut-tutorial/

[6] - http://code-blocks.blogspot.in/2014/12/bresenhams-circle-drawing-algorithm.html

[7] - https://github.com/sprintr/opengl-examples/blob/master/OpenGL-Menu.cpp

[8] - https://stackoverflow.com/questions/24201804/drawing-unfilled-rectangle-shape-in-opengl?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa