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.

1.1.1. Application of Computer Graphics

Computer-Aided Design for engineering and architectural systems etc. Objects maybe displayed in a wireframe outline form. Multi-window environment is also favoured for producing various zooming scales and views. Animations are useful for testing performance.

- Presentation Graphics: To produce illustrations which summarize various kinds of data. Except 2D, 3D graphics are good tools for reporting more complex data.
- Computer Art: Painting packages are available. With cordless, pressure-sensitive stylus, artists can produce electronic paintings which simulate different brush strokes, brush widths, and colours. Photorealistic techniques, morphing and animations are very useful in commercial art. For films, 24 frames per second are required. For video monitor, 30 frames per second are required.
- Entertainment: Motion pictures, Music videos, and TV shows, Computer games
- Education and Training: Training with computer-generated models of specialized systems such as the training of ship captains and aircraft pilots.
- Visualization: For analysing scientific, engineering, medical and business data or behavior. Converting data to visual form can help to understand mass volume of data very efficiently.
- Image Processing: Image processing is to apply techniques to modify or interpre existing pictures. It is widely used in medical applications.
- Graphical User Interface: Multiple window, icons, menus allow a computer setup to be utilized more efficiently.

1.2. OpenGL

OpenGL, or the Open Graphics Library, is a 3D graphics language developed by Silicon graphics. As a software interface for graphics hardware, OpenGL's main purpose is to render two- and three-dimensional objects into a 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 desired image in the frame buffer.

Before OpenGL was available, software developers had to write unique 3D graphics code for each operating system platform as well as different graphics hardware. However, with OpenGL, developers can create graphics and special effects that will appear nearly identical on any operating system and any hardware that supports OpenGL. This makes it much easier for developers of 3D games and programs to port their software to multiple platforms.

When programmers write OpenGL code, they specify a set of commands. Each command executes a drawing action or creates a special effect. Using hundreds or even thousands of these OpenGL commands, programmers can create 3D worlds which can include special effects such as texture mapping, transparency (alpha blending), hidden surface removal, antialiasing, fog, and lighting effects. An unlimited amount of viewing and modelling transformations can be applied to the OpenGL objects, giving developers an infinite amount of possibilities. GLUT gives you the ability to create a window, handle input and render to the screen without being Operating System dependent.

1.2.1. OpenGL architecture

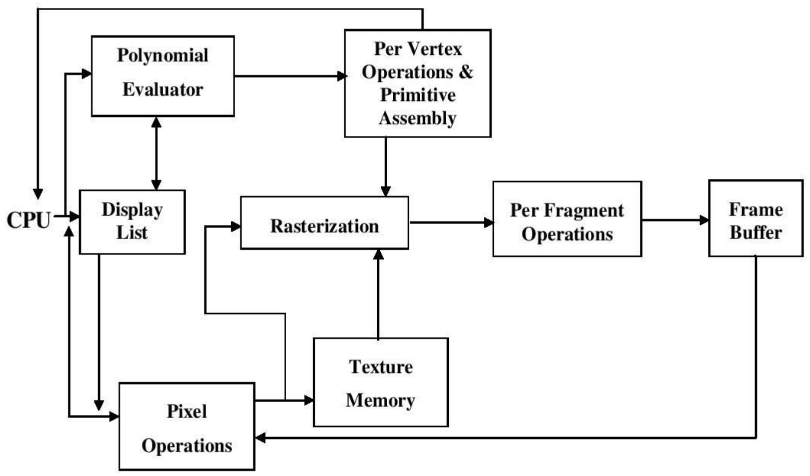


Fig 1.1 OpenGL Architecture

This is the most important diagram we see, representing the flow of graphical information, as it is processed from CPU to the frame buffer.

There are two pipelines of data flow. The upper pipeline is for geometric, vertex-based primitives. The lower pipeline is for pixel-based, image primitives. Texturing combines the two types of primitives together.

1.2.2. GLUT library

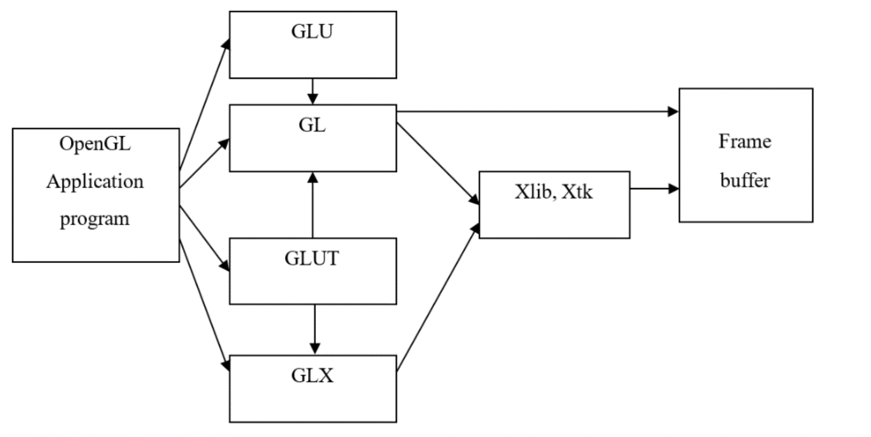


Fig 1.2 Library Organisation of OpenGL

GLUT is the OpenGL utility toolkit, a window system independent toolkit for writing OpenGL programs. It implements a simple windowing API for OpenGL. GLUT makes it easier to learn about and explore OpenGL programming. GLUT provides a portable API so you can write a single OpenGL program that works across all PC and workstation OS platforms. GLUT is designed for constructing small to medium sized OpenGL programs.

While GLUT is well-suited to learning OpenGL and developing simple OpenGL applications, GLUT is not a full-featured toolkit so large applications requiring sophisticated user interfaces are better off using native window system toolkits. The GLUT library has both C, C++ (same as C), FORTRAN, and ADA programming bindings. The GLUT source code distribution is portable to nearly all OpenGL implementations and platforms.

1.2.3. OpenGL contributions

It is very popular in the video games development industry where it competes with Direct3D (on Microsoft Windows).OpenGL is also used in CAD, virtual reality, and scientific visualization programs. OpenGL is very portable. It will run for nearly every platform in existence, and it will run well. It even runs on Windows NT 4.0 etc. The reason OpenGL runs for so many platforms is because of its Open Standard. OpenGL has a wide range of features, both in its core and through extensions. Its extension feature allows it to stay immediately current with new hardware features, despite the mess it can cause.

2. PROJECT DESCRIPTION

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.

2.1. Software and hardware specification

2.1.1. Hardware

➢ Processor	: x86 compatible processor
➢ RAM	: 512 MB or greater
➢ Hard Disk	: 20 GB or greater
➢ Monitor	: VGA/SVGA
➢ Keyboard	: 104 keys standard
➢ Mouse	: 2/3 button. Optical/ Mechanical.

2.1.2. Software

➢ Operating System	: Ubuntu/Windows
➢ Programming Language	: C++
➢ API	: Open-GL

2.2. System design

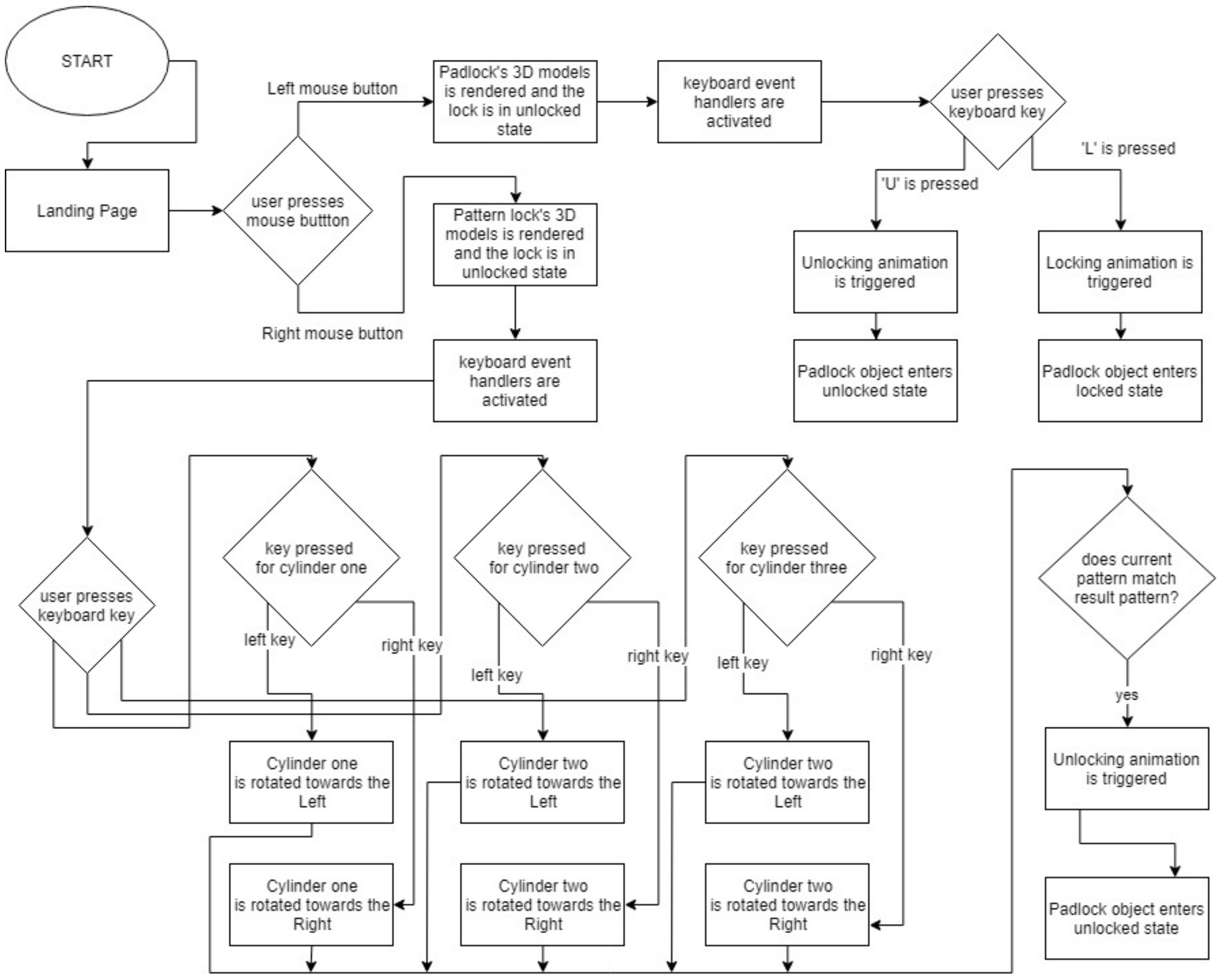

START
Landing Page
user presses mouse buttton
Left mouse button
Right mouse button
Padlock's 3D models is rendered and the lock is in unlocked state
keyboard event handlers are activated
user presses keyboard key
'U' is pressed
'L' is pressed
Unlocking animation is triggered
Locking animation is triggered
Padlock object enters unlocked state
Padlock object enters locked state
Pattern lock's 3D models is rendered and the lock is in unlocked state
keyboard event handlers are activated
user presses keyboard key
key pressed for cylinder one
key pressed for cylinder two
key pressed for cylinder three
left key
right key
Cylinder one is rotated towards the Left
Cylinder one is rotated towards the Right
Cylinder two is rotated towards the Left
Cylinder two is rotated towards the Right
does current pattern match result pattern?
yes
Unlocking animation is triggered
Padlock object enters unlocked state

Fig 2.1 System architecture

3. APIs USED

3.1. GLUT APIs

3.1.1. glutInit

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

glutInit is used to initialize the GLUT library.

3.1.2. glutInitWindowPosition, 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.

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.

3.1.4. glutMainLoop

void glutMainLoop(void);

glutMainLoop enters the GLUT event processing loop.

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.

3.1.6. glutPostRedisplay

void glutPostRedisplay(void);

glutPostRedisplay marks the current window as needing to be redisplayed.

3.1.7. glutSwapBuffers

void glutSwapBuffers(void);

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

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.

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.

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.

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.

3.2. GL APIs

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.

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.

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.

3.2.4. glMatrixMode

void glMatrixMode(GLenum mode);

Specify which matrix is the current matrix.

Accepted values GL_MODELVIEW, GL_PROJECTION, and GL_TEXTURE.

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.

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.

3.2.7. glPushMatrix

void glPushMatrix(void);

push the current matrix stack.

3.2.8. glPopMatrix

void glPopMatrix(void);

pop the current matrix stack.

3.2.9. glVertex*

void glVertexntv(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.

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.

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. 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

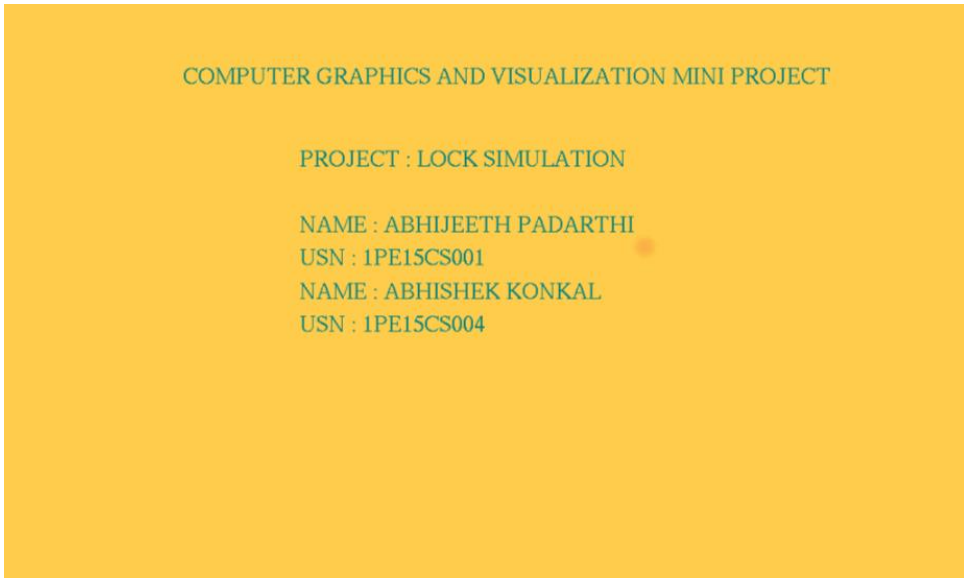
4.1.13. **int main(int argc, char** argv) :** Main Program

5. SOURCE CODE

```
#include <stdio.h>
#include <stdlib.h>
#include <GL/glut.h>
int gameMode = -1; // 0 - pad lock, 1 - number lock
GLfloat keyZ = 0.75; 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 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 SIMULATION",-0.8,1);
       drawDetails("NAME : ABHISHEK KONKAL",-0.8,0.2);
       drawDetails("USN : 1PE15CS004",-0.8,0.0);
       drawDetails("NAME : ABHIJEETH PADARTHI",-0.8,0.6);
       drawDetails("USN : 1PE15CS001",-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 lock(){
   glPushMatrix();    lockCube();    lockFrontDecagon();
   lockKeyholePentagon();    glPushMatrix();    glTranslated(0, lockBarY, 0);    lockBar();
   glPopMatrix();    glPopMatrix();}}
void key(){
   glPushMatrix();    keyBodyPentagon();    keyBlocks();    keyHandle();
   glPopMatrix();
}
void animateKey(){
   if(lockAndKeyAction == 0){
       if(keyZ >= -0.175)            keyZ -= 0.0005;
       if(keyZ < -0.175)            keyPosition = 1;
       if(keyPosition == 1 && keyTheta <= 360)            keyTheta+= 0.5;
       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 -= 0.5;
       if(keyTheta < 0)            keyPosition = 0;
       if(keyPosition == 0 && keyZ <= 0.75)            keyZ += 0.0005;
   } glutPostRedisplay();
}
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();            glTranslated(0, 0, keyZ);
           glRotated(-keyTheta, 0, 0, 1); // TODO - FIX ROTATION
           key();           glPopMatrix();       glPopMatrix();           glPopMatrix();
}
void lockAndKeyKeyboard( unsigned char key, int x, int y){
   if(key == 'u'){
       keyZ = 0.75;       keyTheta = 0;       lockBarY = 0;       keyPosition = 0;
       lockAndKeyAction = 0;       glutPostRedisplay();
   }
   if(key == 'l'){
       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;
}
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;
   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(2000,2000);
		glutInitWindowPosition(0, 35);
		glutCreateWindow("lock and key game");
		glutMouseFunc(mouse);
		glutDisplayFunc(display);
		glutReshapeFunc(myreshape);
		glEnable(GL_DEPTH_TEST);
		glEnable(GL_NORMALIZE);
		glutMainLoop();
		return 0;
}
```

6. SAMPLE OUTPUT



COMPUTER GRAPHICS AND VISUALIZATION MINI PROJECT

PROJECT : LOCK SIMULATION

NAME : ABHIJEETH PADARTHI
USN : 1PE15CS001
NAME : ABHISHEK KONKAL
USN : 1PE15CS004

Fig 6.1 Front Page

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

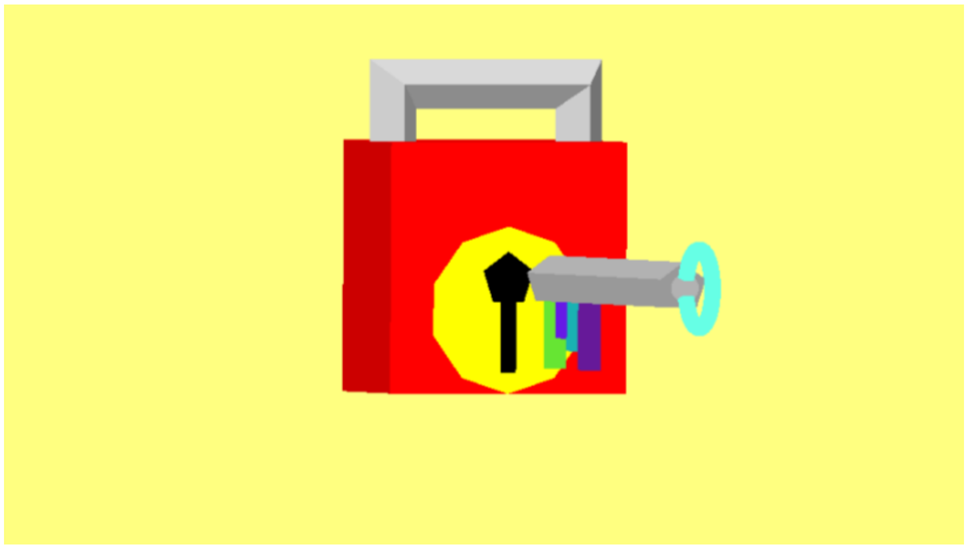


Fig 6.2 Padlock in locked state

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.

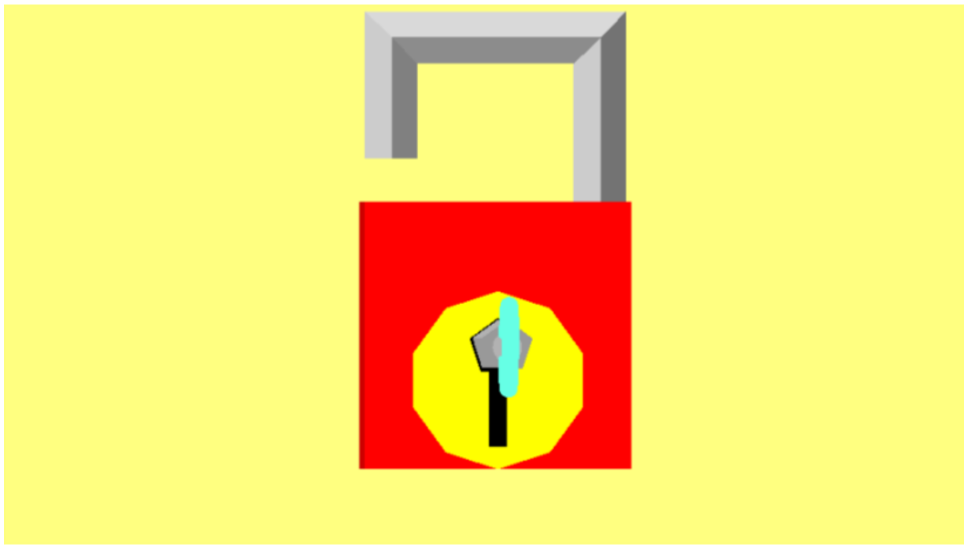


Fig 6.3 Padlock in unlocked state

Upon triggering the locking action, the key is translated into the keyhole of the lock

And then is rotated clockwise to unlock the padlock.

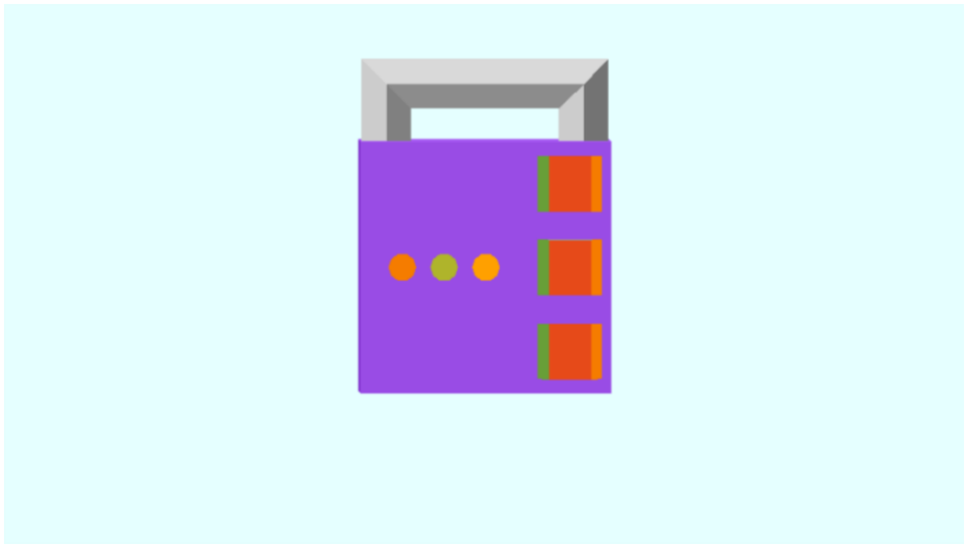


Fig 6.4 pattern lock in locked state

The pattern lock consists of three cylinders with each cylinder having six patterns,

Which can be rotated using number keys.

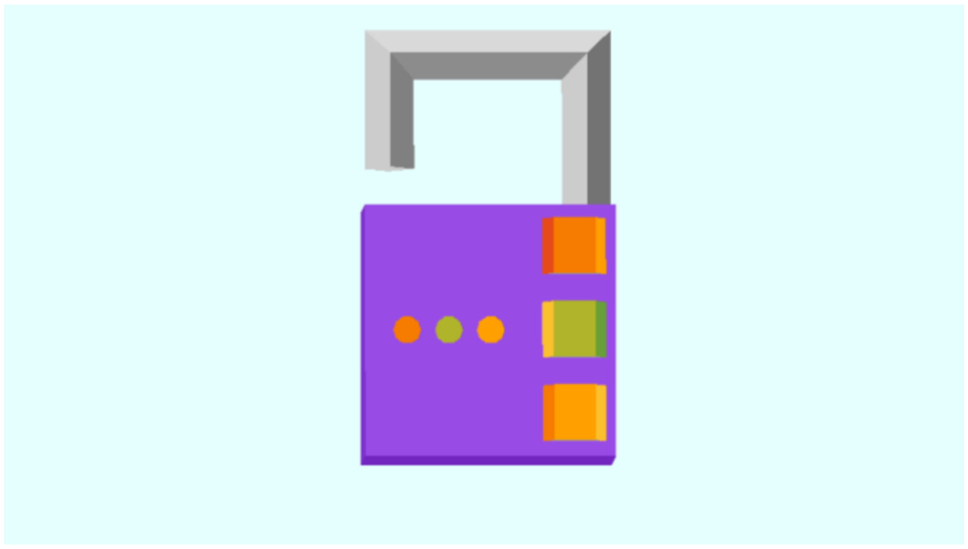


Fig 6.5 Pattern lock in unlocked state

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

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.

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