VISVESVARAYA TECHNOLOGICAL UNIVERSITY
BELAGAVI-590014

MINI PROJECT ENTITLED
"PRETTY PIXEL"

For the academic year 2016-2017

Submitted by
Shashank S (1MV14CS131)

Project carried out at
SIR M. VISVESVARAYA INSTITUTE OF TECHNOLOGY
BANGALORE-562157

Under the Guidance of
Mrs. Monika Rani H G, Asst.Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SIR M. VISVESVARAYA INSTITUTE OF TECHNOLOGY
Hunasamaranahalli, Bangalore-562157

SIR M. VISVESVARAYA INSTITUTE OF TECHNOLOGY
BANGALORE-562157
(affiliated to Visvesvaraya Technological University, Belagavi, Bangalore-562157)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

Certified that the project work entitled **"PRETTY PIXEL"** is a bonafide work carried out by **Shashank S (1MV14CS131)** in partial fulfillment for the award of the degree of **Bachelor of Engineering** in **Computer Science and Engineering** of the **Visvesvaraya Technological University, Belagavi** during the year 2016-2017 in **Computer Graphics and Visualization Laboratory**. The project report has been approved as it satisfies the academic requirements with respect to the project work prescribed for the course of Bachelor of Engineering Degree.

Signature of the Guide

Mrs.Monika Rani H G
Asst. Prof, Dept of CSE
Sir MVIT

Signature of the HOD

Prof. Dilip K Sen
HOD, Dept of CSE
Sir MVIT

External Examiner

Internal Examiner

ACKNOWLEDGEMENT

Every project requires some form of help. In this section , I would like to thank the people who helped us make "Pretty Pixel" a reality.

I thank Mrs. Monika Rani for helping us understand the concepts of Computer Graphics enough to be able to think of independent ideas of implementation.

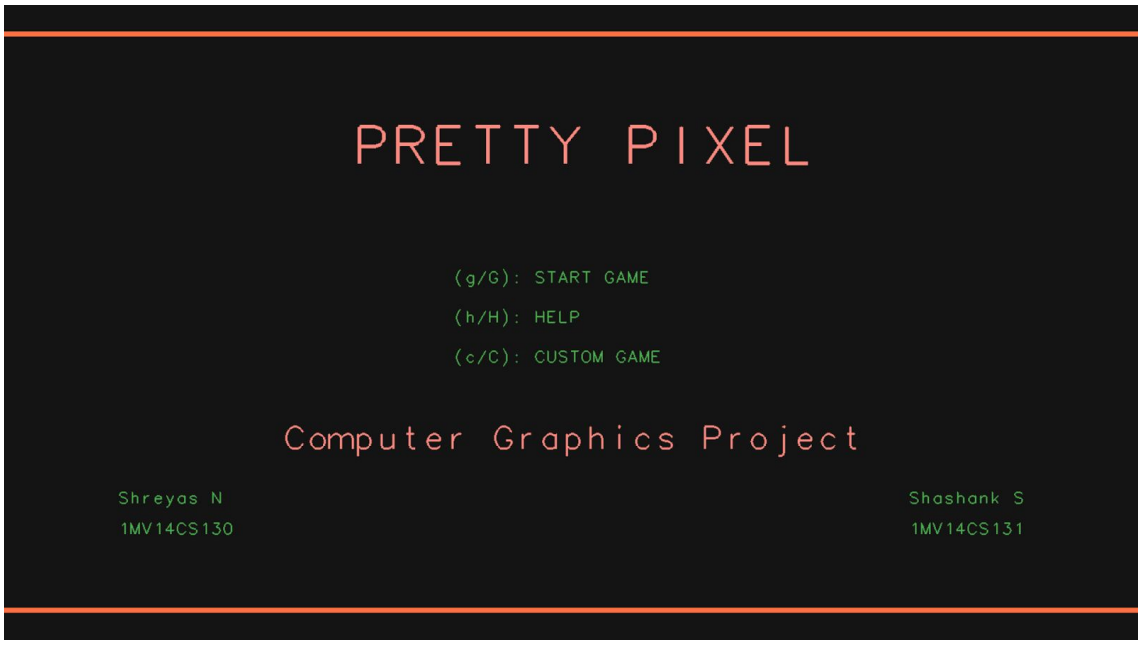
I would like to deeply express my gratitude to Prof. Dilip K Sen, Department of Computer Science and Engineering for his support.

I take this opportunity to thank all the teaching as well as non-teaching staff for their support and motivation.

I would finally like to thank our friends for playing through the initial versions and giving us valuable feedback for improving the game.

A special thanks to the Flying Spaghetti Monster, our lord and savior, who guides us.

Shashank S
1MV14CS131



"If you change the way you look at things, the things you look at change"

- Dr Wayne Dyer

INDEX

LIST OF FIGURES

Chapter 1 INTRODUCTION

This project was created for the purpose of giving people an idea as to how looking at things differently changes what they are looking at.

The project started off as a simple idea, like any other project. We wanted to do something related to 3-Dimensional space and after a lot of thinking and some inspiration from the internet, we came up with the idea of Pretty Pixel.

It gives you a bird's eye view of several specks or points in space which only make sense when you look at them in a certain way. It's all about perspective.

Going through the code of this game, you will learn about various things like working of stroke text and working of the Modelview and Projection matrix. You will also learn about the working of various input callback functions like the keyboard and mouse. The game mechanics at first glance seems quite complicated but is in fact simple and quite brilliant.

Chapter 2 SYSTEM REQUIREMENTS

Hardware Requirements

- 1.5 GHz or faster processor, Intel Pentium 4 or better
- 256 MB internal RAM
- Windows 7 or higher, Linux
- NVIDIA GeForce FX 5200 Graphics Card or better
- Mouse input
- Keyboard Input
- Display of at least 1600x900 resolution

Software Requirements

- OpenGL Libraries: gl, glu, glut, freeglut etc.
- Text Editor(sublime, gedit etc.)
- C++ 11 support or better
- GNU G++ compiler OR Microsoft Visual Studio 10

Chapter 3 DESCRIPTION / ABSTRACT

This game, as mentioned in the introduction, is all about perspective.

The player is given 2000 points at level 1. Starting a game takes the player to a screen of seemingly random points in space alternately connected by lines. When the mouse is moved, these points also move.

These set of seemingly random points actually make up an image and the aim of the game is to find these images by moving the mouse to rotate the playing field to find the correct position on the screen. When the player reached close to this "Viewpoint", from where the image can be viewed, the points snap into place and the player knows that they have passed.

Left mouse click advances the player to the next level. It also supplies the player with additional 1000 points. The goal is to retain the maximum number of points at the end of seven levels.

It is important to note that if the score counter reaches 0 before the player finds the Viewpoint, s/he loses.

Extra Features

A very important feature is the ability to make custom levels. One can create their own level and play it as well.

The difficulty of the game can also be controlled, which removes the alternate lines and makes the player have to play with only a set of points in space.

The points can be made to blink in random colors, giving it such an aesthetic look if the player so desires.

Chapter 4 GAME PLAY

- (g/G) - Start a new game
- (h/H) - Display help screen
- (m/M) - Main menu
- (f/F) - Toggle fullscreen mode
- (c/C) - Custom game
    - LeftMouseButton click to draw points
    - RightMouseButton click to start custom game
- (e/E) - Activate blink effects
- (t/T) - Toggle difficulty (toggle lines)
- (q/Q) - Quit Game
- Mouse move to rotate the playing field
- LeftMouseButton to advance to next level

Chapter 5.1 SOURCE CODE SNIPPETS (IMPORTANT FUNCTIONS)

5.1.1 - Randomize z-coordinates

Randomize the points in three dimensional space but maintain a particular visual perspective

```
void RandomizeZ(int ray[][3], int siz) {
   srand(time(NULL));
   for (int i = 0; i < siz; ++i) {
         ray[i][2] = (rand() % 600) - 300;
   }
}
```

5.1.2 - Draw and store picture for custom game

Drawing and storing of the picture for manipulation in the custom game screen

```
void DrawCustomPixel(int x, int y) {
   glPushMatrix();
   cout << endl << ind;
   glColor3f(1, 1, 1);
   glPointSize(5);
   glBegin(GL_POINTS);
   glVertex3f(x, y, 0);
   glEnd();
   glutSwapBuffers();
   glPopMatrix();

   //Store points
   custPicture[ind][0] = x,
         custPicture[ind][1] = y,
         custPicture[ind][2] = 0;
   ind++;
}
```

5.1.3 - Text render

Rendering function to render stroke text used in various places across the game

```
void DrawText(const char *text, int x, int y, int sparkle) {
   glPushMatrix();
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
   glTranslatef(x, y, 0);
   if (sparkle == 1) {
         glScalef(.5, .5, 0);
         glLineWidth(5);
   }
   else if (sparkle == 2) {
         glScalef(.15, .15, 0);
         glLineWidth(2);
   }
   else if (sparkle == 3) {
         glScalef(.3, .3, 0);
         glLineWidth(3);
   }

   for (int i = 0; text[i] != '\0'; i++)
         glutStrokeCharacter(GLUT_STROKE_MONO_ROMAN, (int)text[i]);
   glutPostRedisplay();
   glPopMatrix();
}
```

5.1.4 - Draw playground

Drawing the pixels in the actual playing field. Notice the elegant utilization of vertex arrays

```
void DrawPixel() {
   glPushMatrix();
   //Draw points
   epilepsyFlag ? glColor3f((rand() % 100) / 100.0, (rand() % 100) / 100.0, (rand() %
100) / 100.0) : glColor4f(0.96, 0.26, 0.21, 1.0);
   glPointSize(5);
   glEnableClientState(GL_VERTEX_ARRAY);
   glVertexPointer(3, GL_INT, 0, picture);
   glDrawArrays(GL_POINTS, 0, siz);
   glDisableClientState(GL_VERTEX_ARRAY);
   //Draw lines
   if (hard)
   {
         glColor3f(1, 1, 1);
         glLineWidth(1);
         glEnableClientState(GL_VERTEX_ARRAY);
         glVertexPointer(3, GL_INT, 0, picture);
         glDrawArrays(GL_LINES, 0, siz);
         glDisableClientState(GL_VERTEX_ARRAY);
         glPopMatrix();
   }
   glPopMatrix();
}
```

5.1.5 - Level initialization

The level-up function along with the level initialization. This takes cares of loading the levels, randomly generating a solution point, leveling up, and toggling custom game level

```
void Leveler() {
   //Set random solution point
   srand(time(NULL));
   cx = (rand() % screenLength) - (screenLength / 2);
   cy = (rand() % screenLength) - (screenLength / 2);
   goalX = cy / (screenLength / 200);
   goalY = cx / (screenLength / 200);

   //Clear picture
   memset(picture, 0, sizeof(int) * siz * 3);

   //Level up
   level++;

   //Play Custom Game
   if (custGame) {
         siz = ind;
         memcpy(picture, custPicture, sizeof(int) * siz * 3);
         score = 1000;
         level = 999;
         custGame = false;
   }

   //Play default game
   else {
         //Load level picture
         switch (level) {
         case 1:
               siz = sizeof(one) / sizeof(one[0]);
               memcpy(picture, one, sizeof(int) * siz * 3);
               break;

         case 2:
               siz = sizeof(two) / sizeof(two[0]);
               memcpy(picture, two, sizeof(int) * siz * 3);
               score += 1000;
               break;

         case 3:
               siz = sizeof(three) / sizeof(three[0]);
               memcpy(picture, three, sizeof(int) * siz * 3);
               score += 1000;
               break;
```

```
            case 4:
                siz = sizeof(four) / sizeof(four[0]);
                memcpy(picture, four, sizeof(int) * siz * 3);
                score += 1000;
                break;

            case 5:
                siz = sizeof(five) / sizeof(five[0]);
                memcpy(picture, five, sizeof(int) * siz * 3);
                score += 1000;
                break;

            case 6:
                siz = sizeof(six) / sizeof(six[0]);
                memcpy(picture, six, sizeof(int) * siz * 3);
                score += 1000;
                break;

            case 7:
                siz = sizeof(seven) / sizeof(seven[0]);
                memcpy(picture, seven, sizeof(int) * siz * 3);
                score += 1000;
                break;

            default://Gameover
                cout << " Score: " << score << "\nGame Over\n";
                siz = sizeof(one) / sizeof(one[0]);
                memcpy(picture, one, sizeof(int) * siz * 3);
                level = 1;
                score = 1000;
                glutDisplayFunc(GameOver);
                break;
            }
        }

        if (level > 1)
            cout << " Score: " << score;
        cout << "\nLevel " << level;

        //Randomize z coordinates
        RandomizeZ(picture, siz);
    }
```

5.2 SOURCE CODE SNIPPETS (INPUT CALLBACK FUNCTIONS)

5.2.1 - Passive motion callback

Mouse callback when no button is clicked. Used in main playing logic.

```
void MouseMove(int x, int y) {
   //Normalization
   x = x - (screenLength / 2),
         y = (screenLength - y) - (screenLength / 2);
   if (!drawSC) {
         rotateX = -y / (screenLength / 200);
         rotateY = x / (screenLength / 200);
         if (--score < 0)
         {
               score = 1000;
               glutDisplayFunc(GameOver);
         }
         glutPostRedisplay();
   }
}
```

5.2.2 - Active motion callback

Mouse callback when either button is clicked. Used in draw custom game logic.

```
void MouseClickMove(int x, int y) {
   //Normalization
   x = x - (screenLength / 2),
         y = (screenLength - y) - (screenLength / 2);
}
```

5.2.3 - Mouse click callback

Mouse button click callback. Handles left and right mouse button clicks

```
void MouseClick(int button, int state, int x, int y) {
   //Normalization
   x = x - (screenLength / 2),
         y = (screenLength - y) - (screenLength / 2);

   switch (button) {
   case GLUT_RIGHT_BUTTON://Right mouse button
         if (state == GLUT_DOWN) {
               if (!drawSC) {
                     cout << "\nGame Over\n";
                     //exit(0);
               }
               else {
                     //WriteCoord();
                     drawSC = false;
                     custGame = true;
                     glutSetWindowTitle("Pretty Pixel");
                     glutFullScreen();
                     glPointSize(1);
                     Leveler();
                     glutDisplayFunc(Pretty);
               }
         }
         break;
   case GLUT_LEFT_BUTTON://left mouse button
         if (state == GLUT_DOWN && pass == true && !drawSC) {
               //Level up
               pass = false;
               Leveler();
         }
         else if (state == GLUT_DOWN && drawSC &&
               x > -350 && x < 350 && y > -350 && y < 350) {
               //User drawing board
               DrawCustomPixel(x, y);
               glFlush();
         }
         break;
   }
}
```

5.2.4 - Keyboard callback

Keyboard key press callback. Handles various keyboard input keys.

```
void Keys(unsigned char key, int x, int y) {

  switch (key)
  {
  case 'm':
  case 'M':
        //Splash screen
        drawSC = false;
        glutDisplayFunc(Splash);
        break;

  case 'h':
  case 'H':
        //Help screen
        drawSC = false;
        glutDisplayFunc(Help);
        break;

  case 'g':
  case 'G':
        //Game screen
        drawSC = false;
  //Level Bug fixed here
        level = 0;
        Leveler();

        glutSetWindowTitle("Pretty Pixel");
        glutDisplayFunc(Pretty);
        break;

  case 'f':
  case 'F':
        //Full screen toggle
        glutFullScreenToggle();
        break;

  case 'c':
  case 'C':
        //Custom game screen
        drawSC = true;
        ind = 0;
        glutReshapeWindow(800, 800);
        glutSetWindowTitle("Pretty Draw");
        glutDisplayFunc(CustomDraw);
```

```
            break;

        case 'e':
        case 'E':
            //Want some epilepsy?
            epilepsyFlag ? epilepsyFlag = false : epilepsyFlag = true;
            break;

        case 't':
        case 'T':
            hard ? hard = false : hard = true;
            break;

        case 'q':
        case 'Q':
            //Quit game
            exit(0);
            break;
    }
}
```

5.3 SOURCE CODE SNIPPET (MAIN FUNCTION)

Main

```
int main(int argc, char **argv) {
   cout << "Hello World!  Pretty Pixel\n";

   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
   glutInitWindowPosition((glutGet(GLUT_SCREEN_WIDTH) - screenLength) / 2,
         (glutGet(GLUT_SCREEN_HEIGHT) - screenLength) / 2);
   glutInitWindowSize(screenLength, screenLength);

   glutCreateWindow("Pretty Pixel");
   glutFullScreen();
   glEnable(GL_DEPTH_TEST);

   glutReshapeFunc(Reshape);
   glutMouseFunc(MouseClick);
   glutMotionFunc(MouseClickMove);
   glutPassiveMotionFunc(MouseMove);
   glutKeyboardFunc(Keys);
   glutDisplayFunc(Splash);
   glClearColor(0.08f, 0.08f, 0.08f, 1.0);
   glutMainLoop();

   return 0;
}
```

Chapter 6

OUTPUT SCREENSHOTS

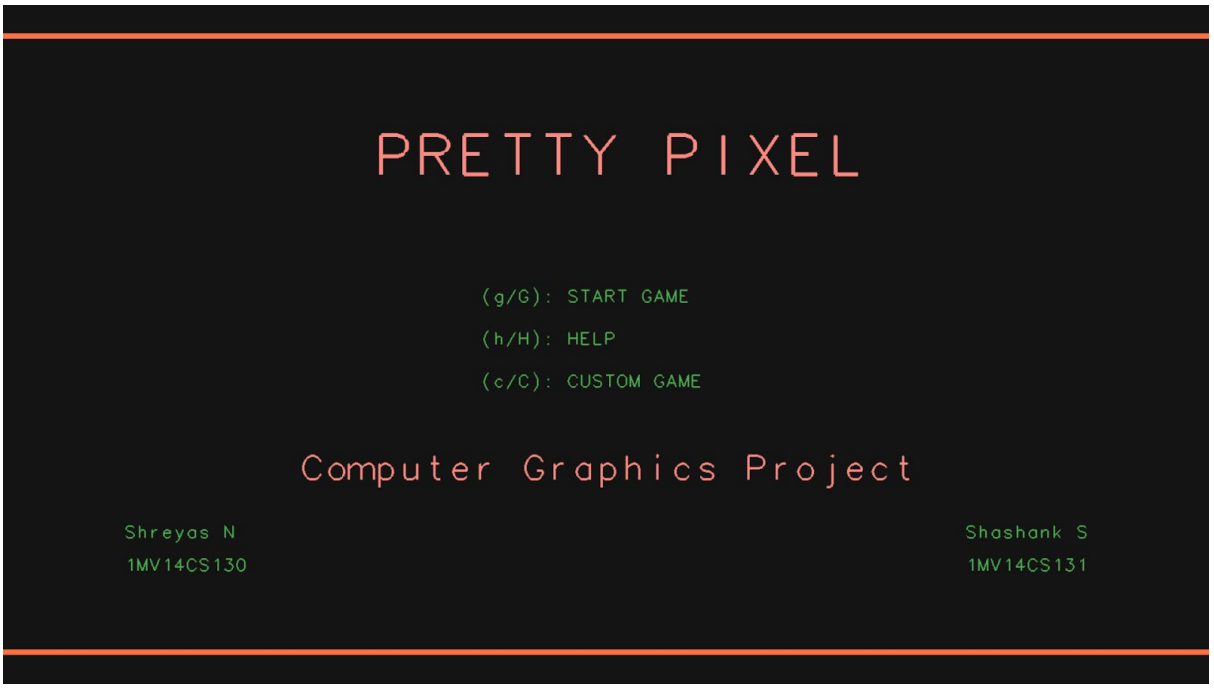


Fig. 6.1 - Splash screen

This is the main screen that greets a player on game initialization. The game starts by default in full screen mode.

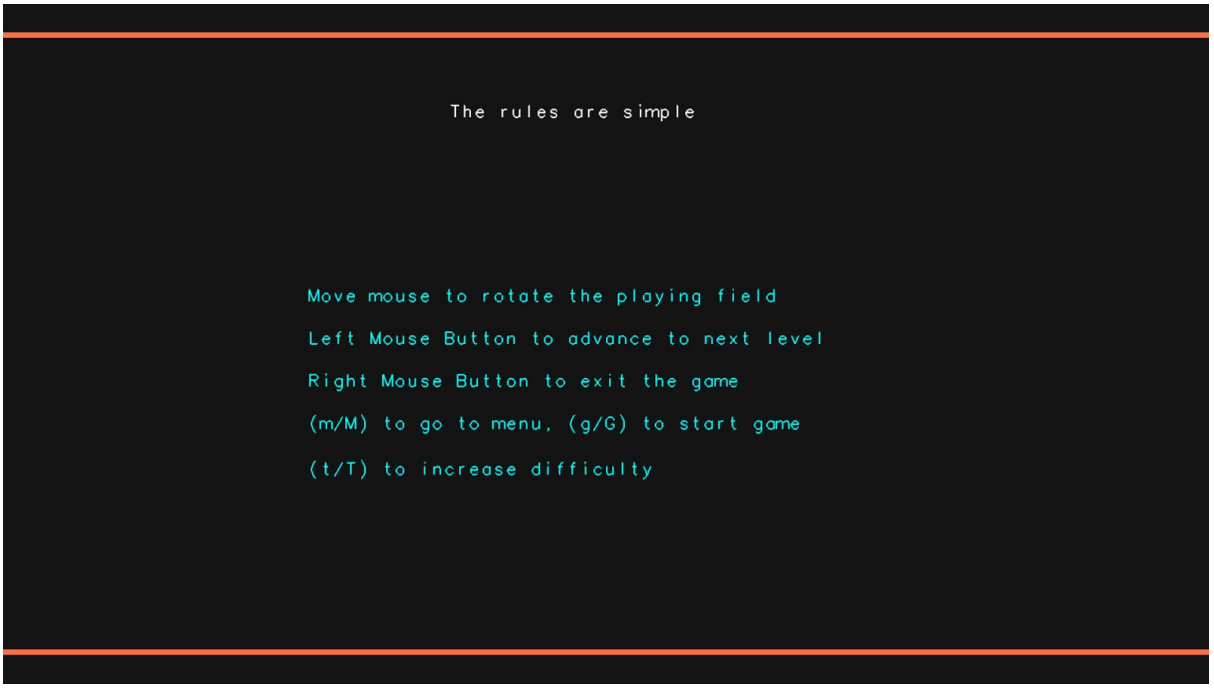


Fig. 6.2 - Help screen

Specifies various aspects of the gameplay.

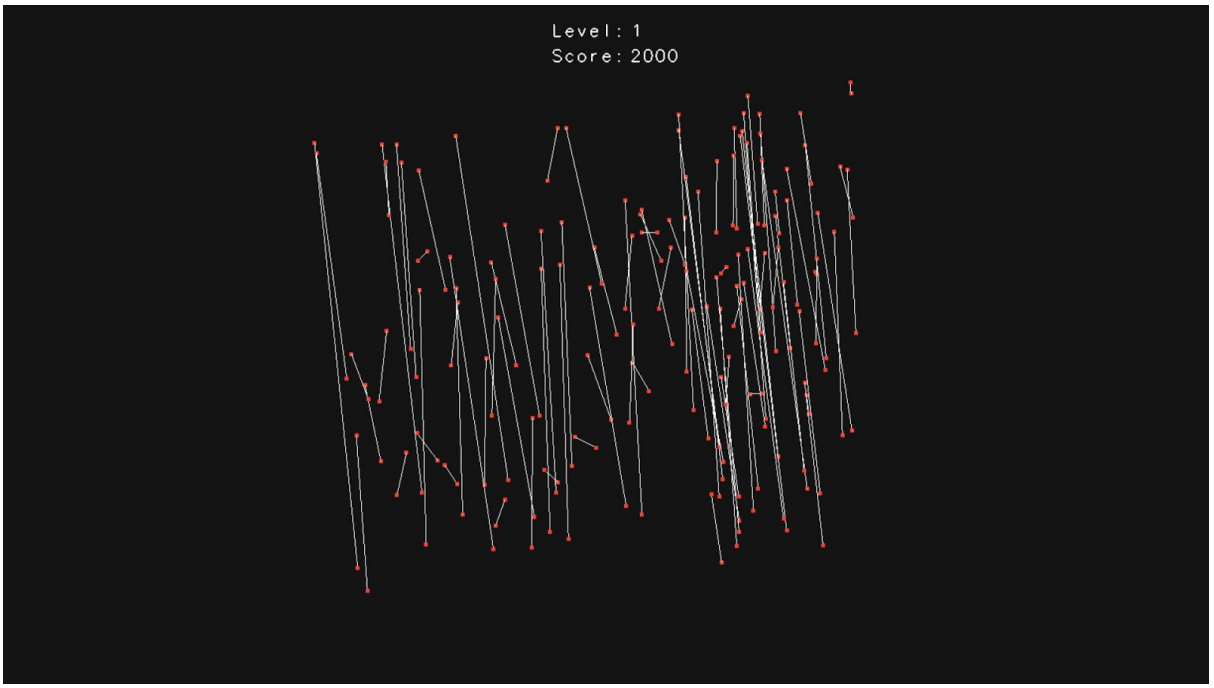


Fig. 6.3 - Level one(unsolved)
Notice the seemingly random lines. The lines reorient themselves as the playing field is rotated.

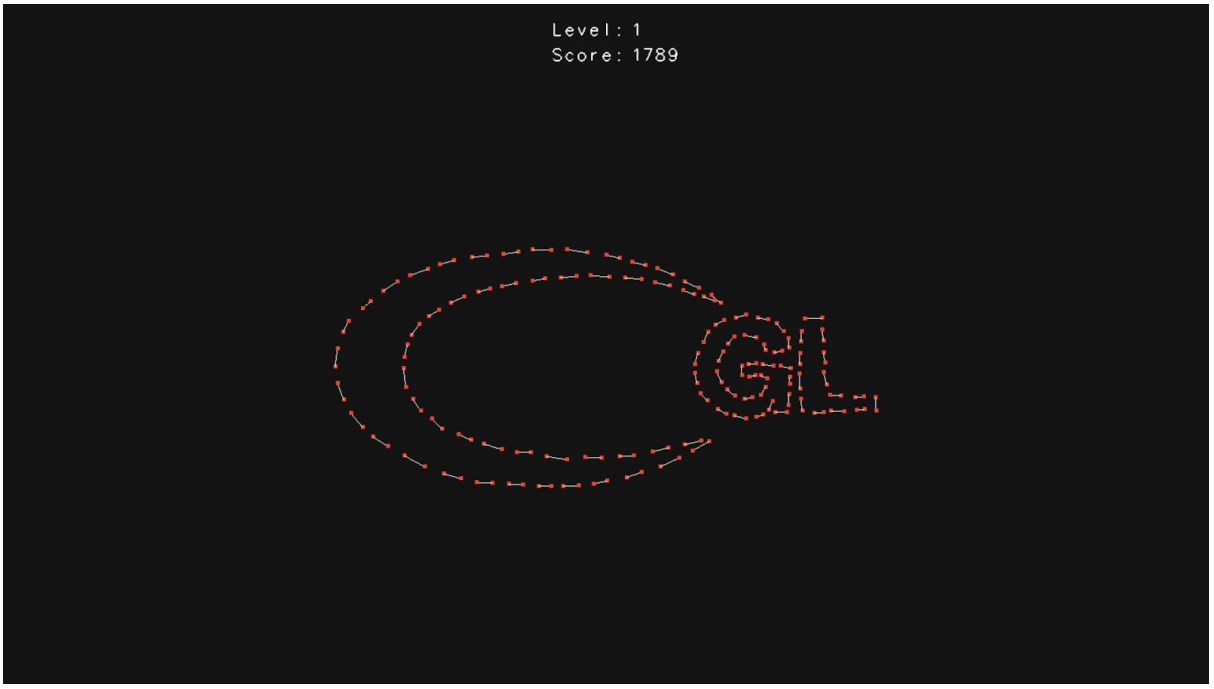


Fig. 6.4 - Level one(solved)
The lines snap to the figure as the mouse moves to the correct perspective point. This is the logo of OpenGL, partial.

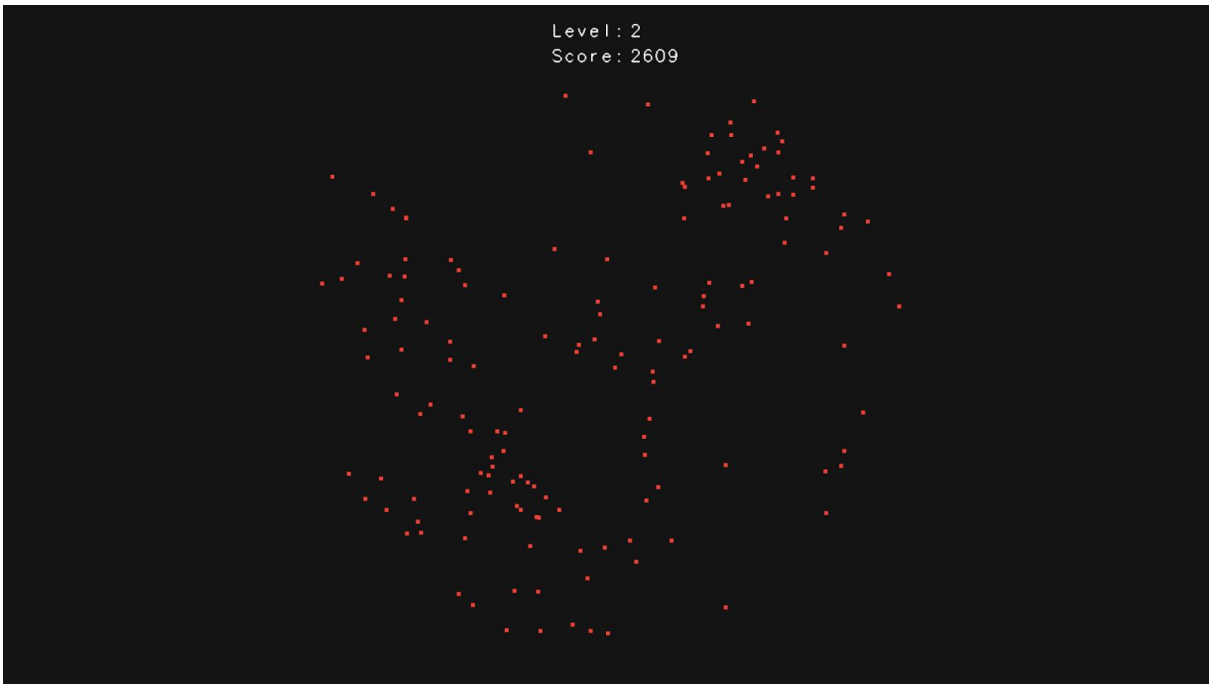


Fig. 6.5 - Hard mode toggle
The lines are erased. The points simulate stars in the formation of a constellation.

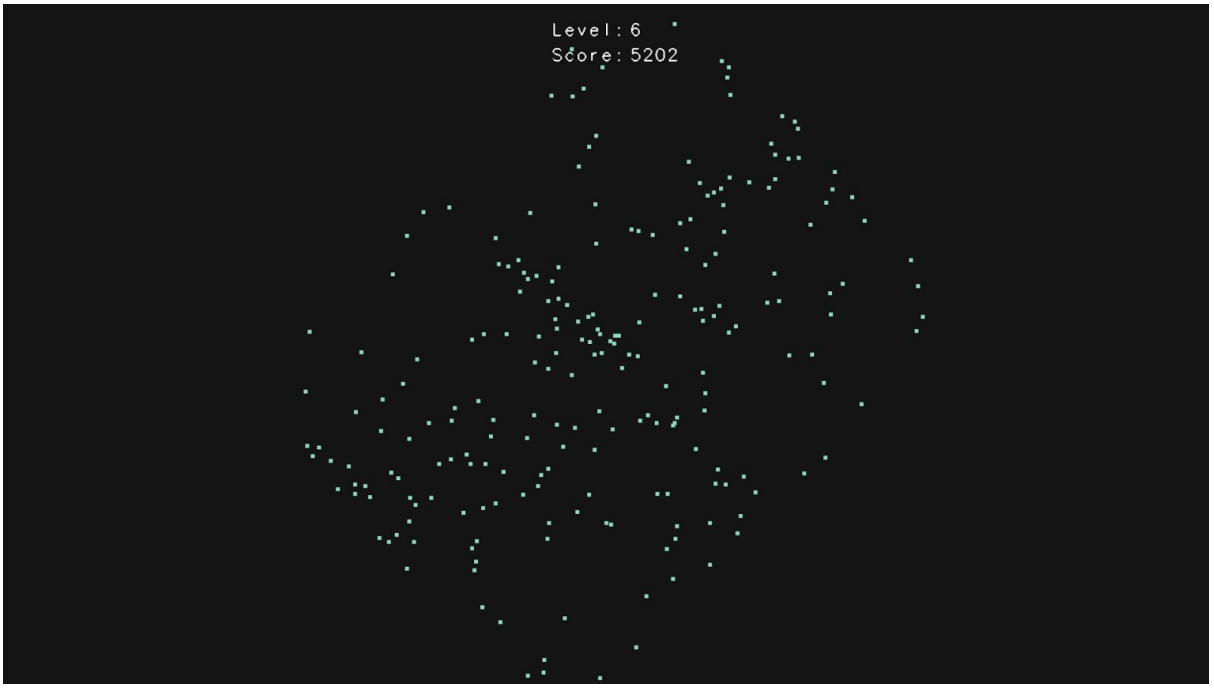


Fig. 6.6 - Blink mode toggle
The pixels randomly change colors to simulate blinking stars.

Fig. 6.7 - Custom draw mode
This is the final goal perspective.

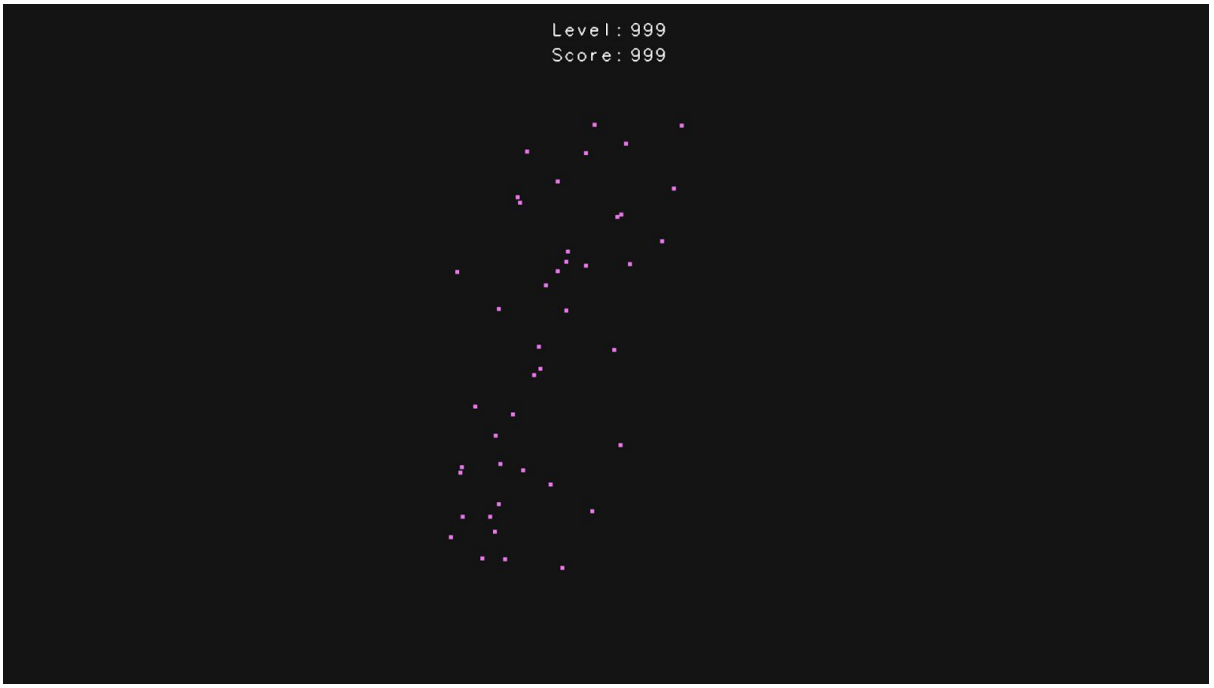


Fig 6.8 - Custom play mode
Notice level number. You get 999 points to solve your custom level.

Chapter 7 CONCLUSION

This project served to further our understanding of the Open Graphics Library. It is a game which emphasises on the importance of perspective and sends a message that not everything is as it seems, sometimes all it takes is to look at things from a different angle.

Pretty Pixel is a game which anyone can play and has no age limits. One can create levels of their own and even play with their friends. It is a game simple in its mechanics but elegant in its execution.

As a final note we would like to tell a little bit about the origins of the name. We are handling, orienting and looking at dots on the screen, hence the “Pixel”. And you have to admit, that is some real nice sparkling pixels too. You can see why we call them “Pretty”.

Chapter 8 FUTURE ENHANCEMENTS

Now

- There are currently 7 predefined levels which can be played on this game.
- The player can create one custom level and play it.

Possibilities

- Points to be anti-aliased to make them look like stars.
- The number of levels can be increased as per requirement. This would only require changes in some of the code and inclusion of image points in a header file.
- The difficulty can be increased further by reducing point size and the buffer for solution snapping dynamically.
- Another version of the game with the player being able to play with more than one custom level at once can be made.
- The game can be ported to Android. It has tremendous potential as an android application to harness the touch input to enhance the gameplay.

REFERENCES

The following technologies were instrumental in the completion of this game

- **OpenGL** - Open Graphics Library, a cross-language, cross-platform application programming interface (API) for rendering 2D and 3D vector graphics
- **FreeGLUT** - An open source alternative to the OpenGL Utility Toolkit (GLUT) library
- **Sublime Text 3** - A sophisticated text editor for code, markup and prose
- **Visual Studio** - An integrated development environment (IDE) from Microsoft
- **GNU/GCC** - A compiler system produced by the *GNU* Project supporting various programming languages, here C++
- **GitHub** - A web-based Git or version control repository and Internet hosting service
- **StackOverflow** - An online community for programmers to learn, share their knowledge, and advance their careers
- **Windows** - A metafamily of graphical operating systems developed, marketed, and sold by Microsoft
- **Fedora** - An operating system based on Linux kernel and GNU programs

The following websites also proved to be of tremendous help

- https://www.opengl.org/
- www.glprogramming.com/red/

A special shout-out to **Armor Games**, a website that hosts free Flash-based browser games. The game **Starlight**, created by **zedarus**, was the inspiration behind Pretty Pixel.

- http://armorgames.com/play/4998/starlight
- http://armorgames.com/user/zedarus


GameOver
(m/M): MAIN MENU
(q/Q): EXIT
Your High Score:5774