Table of Contents

List of Figures

Chapter 1

INTRODUCTION

1.1 Overview

Interactive computer graphics provides us with the most natural means of communicating information through a computer. Over the years advancements in computer graphics have enabled us to not only make pictures of real-world objects but also visualize abstract, synthetic objects such as mathematical surfaces and of data that have no inherent geometry, such as survey results. Some topics in computer graphics include user interface design, sprite graphics, vector graphics, 3D modeling, shaders, GPU design, implicit surface visualization with ray tracing, and computer vision, among others. The overall methodology depends heavily on the underlying sciences of geometry, optics, and physics. The computer on receiving signals from the input device can modify the displayed picture appropriately. To the user it appears that the picture is changing instantaneously in response to his commands. He can give a series of commands, each one generating a graphical response from the computer. In this way he maintains a conversation, or dialogue, with the computer.

1.2 Problem Statement

The aim of this application is to show a basic implementation of a village. The application will be implemented using the C programming language and the OpenGL API. The objective of the application is to demonstrate the day and night view of a village using the principles of Computer Graphics. The application will include user interaction through keyboard events for translation, rotation, and night view; also mouse events for specific actions related to the application.

1.3 Motivation

Opengl functions helps us to give a high level graphical view to the user. A Simple village can be simulated using opengl. It has functions through which many components of a village can be shown. It also has high level functions through which movement is possible.

1.4 Computer Graphics

Computer graphics and multimedia technologies are becoming widely used in educational
applications because they facilitate non-linear, self-learning environments that particularly suited to abstract concepts and technical information.

Computer graphics are pictures and films created using computers. Usually, the term refers to computer-generated image data created with help from specialized graphical hardware and software. It is a vast and recent area in computer science. It is often abbreviated as CG, though sometimes erroneously referred to as CGI. Important topics in computer graphics include user interface design, sprite graphics, vector graphics, 3D modeling, shaders, GPU design, implicit surface visualization with ray tracing, and computer vision, among others.

The overall methodology depends heavily on the underlying sciences of geometry, optics, physics. Computer graphics is responsible for displaying art & image data effectively to the user. It is also used for processing image data received from the physical world.

1.5 OpenGL API

Open Graphics Library (OpenGL) is a cross-language, cross-platform application programming interface (API) for rendering 2D and 3D vector graphics. The API is typically used to interact with a graphics processing unit (GPU), to achieve hardware-accelerated rendering. Silicon Graphics Inc., (SGI) began developing OpenGL in 1991 and released it on June 30, 1992; applications use it extensively in the fields of computer-aided design (CAD), virtual reality, scientific visualization, information visualization, flight simulation, and video games. The OpenGL specification describes an abstract API for drawing 2D and 3D graphics. Although it is possible for the API to be implemented entirely in software, it is designed to be implemented mostly or entirely in hardware. The API is defined as a set of functions which may be called by the client program, alongside a set of named integer constants. In addition to being language-independent, OpenGL is also cross-platform.

1.5.1 OpenGL API Architecture

Display Lists:

All data, whether it describes geometry or pixels, can be saved in a display list for current or later use. When a display list is executed, the retained data is sent from the display list just as if it were sent by the application in immediate mode.

Evaluators:

All geometric primitives are eventually described by vertices. Parametric curves and surfaces may be initially described by control points and polynomial functions called basis functions.

Per Vertex Operations:

For vertex data, next is the "per-vertex operations" stage, which converts the vertices into primitives. Some vertex data are transformed by 4 x 4 floating-point matrices. Spatial coordinates are projected from a position in the 3D world to a position on your screen.

Primitive Assembly:

Clipping, a major part of primitive assembly, is the elimination of portions of geometry which fall outside a half space, defined by a plane.

Pixel Operation:

While geometric data takes one path through the OpenGL rendering pipeline, pixel data takes a different route. Pixels from an array in system memory are first unpacked from one of a variety of formats into the proper number of components. Next the data is scaled, biased, and processed by a pixel map. The results are clamped and then either written into texture memory or sent to the rasterization step.

Rasterization:

Rasterization is the conversion of both geometric and pixel data into fragments. Each fragment square corresponds to a pixel in the frame buffer. Colour and depth values are assigned for each fragment square.

Fragment Operations:

Before values are actually stored into the framebuffer, a series of operations are performed that may alter or even throw out fragments. All these operations can be enabled or disabled.

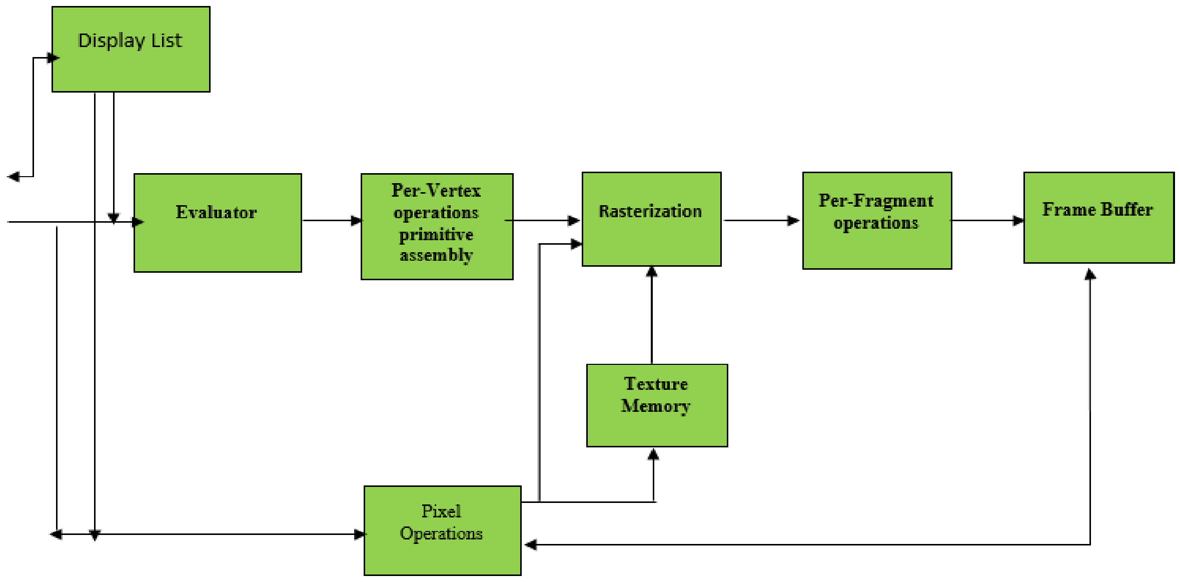

Figure 1.1 Illustration of OpenGL Architecture

1.6 Applications of Computer Graphics

Although many applications span two, three, or even all of these areas, the development of the field was based, for the most part, on separate work in each domain. We can classify applications of computer graphics into four main areas:

1.6.1 Display of Information

Graphics has always been associated with the display of information. Examples of the use of orthographic projections to display floorplans of buildings can be found on 4000-year-old Babylonian stone tablets. Mechanical methods for creating perspective drawings were developed during the Renaissance. Countless engineering students have become familiar with interpreting data plotted on log paper. More recently, software packages that allow interactive design of charts incorporating colour, multiple data sets, and alternate plotting methods have become the norm. In fields such as architecture and mechanical design, hand drafting is being replaced by computer-based drafting systems using plotters and workstations. Medical imaging uses computer graphics in a number of exciting ways.

1.6.2 Design

Professions such as engineering and architecture are concerned with design. Although their applications vary, most designers face similar difficulties and use similar methodologies. One of the principal characteristics of most design problems is the lack of a unique solution. Hence, the designer will examine a potential design and then will modify it, possibly many times, in an attempt to achieve a better solution. Computer graphics has become an indispensable element in this iterative process.

1.6.3 Simulation

Some of the most impressive and familiar uses of computer graphics can be classified as simulations. Video games demonstrate both the visual appeal of computer graphics and our ability to generate complex imagery in real time.

1.6.4 User Interfaces

The interface between the human and the computer has been radically altered by the use of computer graphics. Consider the electronic office. The figures in this book were produced through just such an interface. A secretary sits at a workstation, rather than at a desk equipped with a typewriter. This user has a pointing device, such as a mouse, that allows him to communicate with the workstation.

Chapter 2

LITERATURE SURVEY

2.1 History of Computer Graphics

The term “computer graphics” was coined in 1960 by William Fetter, a designer at Boeing, to describe his own job, the field can be said to have first arrived with the publication in 1963 of Ivan Sutherland’s Sketchpad program, as part of his Ph.D. thesis at MIT. Sketchpad, as its name suggests, was a drawing program. Beyond the interactive drawing of primitives such as lines and circles and their manipulation – in particular, copying, moving and constraining – with use of the then recently invented light pen, Sketchpad had the first fully-functional graphical user interface (GUI) and the first algorithms for geometric operations such as clip and zoom. Interesting, as well, is that Sketchpad’s innovation of an object-instance model to store data for geometric primitives foretold object-oriented programming. Coincidentally, on the hardware side, the year 1963 saw the invention by Douglas Engelbart at the Stanford Research Institute of the mouse, the humble device even today carrying so much of GUI on its thin shoulders.

Subsequent advances through the sixties came thick and fast: raster algorithms, the implementation of parametric surfaces, hidden-surface algorithms and the representation of points by homogeneous coordinates, the latter crucially presaging the foundational role of projective geometry in 3D graphics, to name a few. Flight simulators were the killer app of the day and companies such as General Electric and Evans & Sutherland, 6 co-founded by Douglas Evans and Ivan Sutherland, wrote simulators with real-time graphics.

The seventies, brought the Z-buffer for hidden surface removal, texture mapping, Phong’s lighting model – all crucial components of the OpenGL API (Application Programming Interface) we’ll be using soon – as well as keyframe-based animation.

Photorealistic rendering of animated movie keyframes almost invariably deploys ray tracers, which were born in the seventies too.

Through the nineties, as well, the use of 3D effects in movies became pervasive. The Terminator and Star Wars series, and Jurassic Park, were among the early movies to set the standard for CGI. Toy Story from Pixar, 8 released in 1995, has special importance in the history of 3D CGI as the first movie to be entirely computer-generated – no scene was ever pondered through a glass lens, nor any recorded on a photographic reel! It was cinema without film. Quake, released in 1996, the first of the hugely popular Quake series of games, was the first fully 3D game.

Another landmark from the nineties of particular relevance to us was the release in 1992 of OpenGL, the open-standard cross-platform and cross language 3D graphics API, by Silicon Graphics. OpenGL is actually a library of calls to perform 3D tasks, which can be accessed from programs written in various languages and running over various operating systems. That OpenGL was high-level (in that it frees the applications programmer from having to care about such low-level tasks as representing primitives like lines and triangles in the raster, or rendering them to the window) and easy to use (much more so than its predecessor 3D graphics API, PHIGS, standing for Programmer's Hierarchical Interactive Graphics System) first brought 3D graphics programming to the "masses". What till then had been the realm of a specialist was now open to a casual programmer following a fairly amicable learning curve.

Since its release OpenGL has been rapidly adopted throughout academia and industry. It's only among game developers that Microsoft's proprietary 3D API, Direct3D, which came soon after OpenGL bearing an odd similarity to it but optimized for Windows, is more popular.

The story of the past decade has been one of steady progress, rather than spectacular innovations in CG. Hardware continues to get faster, better, smaller and cheaper, continually pushing erstwhile high-end software down market, and raising the bar for new products. The almost complete displacement of CRT monitors by LCD and the emergence of high-definition television are familiar consequences of recent hardware evolution.

2.2 Related Work

- **Computer Aided Design (CAD):**

  Most of engineering and Architecture students are concerned with Design. CAD is used to design various structures such as Computers, Aircrafts, Building, in almost all kinds of Industries. Its use in designing electronic systems is known as electronic design automation (EDA). In mechanical design it is known as mechanical design automation (MDA) or computer-aided drafting (CAD), which includes the process of creating a technical drawing with the use of computer software.

- **Computer Simulation**

Computer simulation is the reproduction of the behavior of a system using a computer to simulate the outcomes of a mathematical model associated with said system. Since they allow to check the reliability of chosen mathematical models, computer simulations have become a useful tool for the mathematical modeling of many natural systems in physics (computational physics), astrophysics, climatology, chemistry, biology and manufacturing, human systems in economics, psychology, social science, health care and engineering. Simulation of a system is represented as the running of the system's model. It can be used to explore and gain new insights into new technology and to estimate the performance of systems too complex for analytical solutions.

- **Digital Art**

Digital art is an artistic work or practice that uses digital technology as part of the creative or presentation process. Since the 1970s, various names have been used to describe the process, including computer art and multimedia art. Digital art is itself placed under the larger umbrella term new media art. With the rise of social media and the internet, digital art application of computer graphics. After some initial resistance, the impact of digital technology has transformed activities such as painting, drawing, sculpture and music/sound art, while new forms, such as net art, digital installation art, and virtual reality, have become recognized artistic practices. More generally the term digital artist is used to describe an artist who makes use of digital technologies in the production of art. In an expanded sense, "digital art" is contemporary art that uses the methods of mass production or digital media.

- **Virtual Reality**

Virtual reality (VR) is an experience taking place within a computer-generated reality of immersive environments can be similar to or completely different from the real world. Applications of virtual reality can include entertainment (i.e. gaming) and educational purposes (i.e. medical or military training). Other, distinct types of VR style technology include augmented reality and mixed reality. Currently standard virtual reality systems use either virtual reality headsets or multi-projected environments to generate realistic images, sounds and other sensations that simulate a user's physical presence in a virtual environment. A person using virtual reality equipment is able to look around the artificial world, move around in it, and interact with virtual features or items. The effect is commonly created by VR headsets consisting of a head-mounted display with a small screen in front of the eyes, but can also be created through specially designed rooms with multiple large screens.

- **Video Games**

A video game is an electronic game that involves interaction with a user interface to generate visual feedback on a two- or three-dimensional video display device such as a TV screen, virtual reality headset or computer monitor. Since the 1980s, video games have become an increasingly important part of the entertainment industry, and whether they are also a form of art is a matter of dispute. The electronic systems used to play video games are called platforms. Video games are developed and released for one or several platforms and may not be available on others. Specialized platforms such as arcade games, which present the game in a large, typically coin-operated chassis, were common in the 1980s in video arcades, but declined in popularity as other, more affordable platforms became available. These include dedicated devices such as video game consoles, as well as general-purpose computers like a laptop, desktop or handheld computing device.

Chapter 3

SYSTEM REQUIREMENTS

3.1 Software Requirements

Software requirements deal with defining software resource requirements and prerequisites that need to be installed on a computer to provide optimal functioning of an application.

The following are the software requirements for the application:

- Operating System: Windows 10
- Compiler: GNU C/C++ Compiler
- Development Environment: Visual Studio 2019 Community Edition
- API: OpenGL API & Win32 API for User Interface and Interaction.

3.2 Hardware Requirements

The most common set of requirements defined by any operating system or software application is the physical computer resources, also known as hardware.

- CPU: Intel or AMD processor
- Cores: Dual-Core (Quad-Core recommended)
- RAM: minimum 4GB (>4GB recommended)
- Graphics: Intel Integrated Graphics or AMD Equivalent
- Display Resolution: 1366x768 (1920x1080 recommended.

Chapter 4

SYSTEM DESIGN

4.1 Proposed System

This project intends to give an animation of a simple village. The user is allowed to provide inputs through key board and mouse and view changes accordingly. Initially a scene with 2 houses, sun, moving cloud, pond with fishes inside is displayed to the user. On keying ‘x’ or ‘y’ the fishes get repositioned(translated) along x and y axes respectively. When ‘r’ key is pressed, the fishes rotate through 360 degrees. When ‘n’ key is pressed, Night mode can be visualized.

When the options ‘Open doors’ and ‘Turn Lights ON’ are chosen from the menu on right click of the mouse, then mouse event occurs which performs the appropriate actions as desired by the user.

In both night and day views, the clouds are in continuous movement due to the idle function that is implemented and called in the main function.

4.2 Data Flow Diagram

A data-flow diagram (DFD) is a way of representing a flow of a data of a process or a system (usually an information system). The DFD also provides information about the outputs and inputs of each entity and the process itself. A data-flow diagram has no control flow, there are no decision rules and no loops. For each data flow, at least one of the endpoints (source and / or destination) must exist in a process. The refined representation of a process can be done in another data-flow diagram, which subdivides this process into sub-processes. The data-flow diagram is part of the structured-analysis modelling tools. When using UML, the activity diagram typically takes over the role of the data-flow diagram. A special form of data-flow plan is a site-oriented data-flow plan.

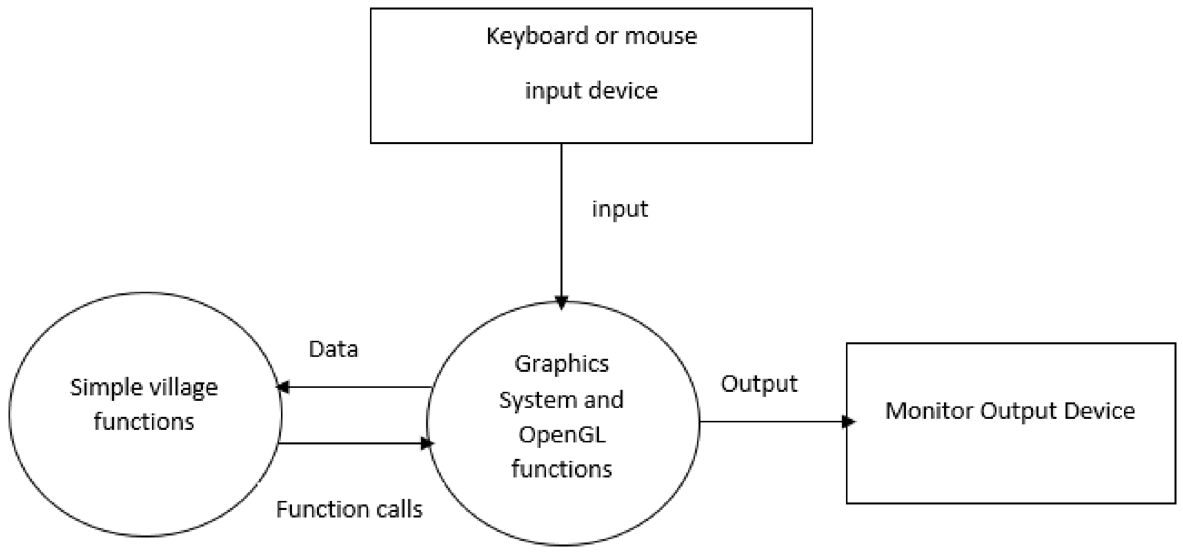

Figure 4.2 Level 0 Dataflow Diagram of the Proposed System

Figure 4.1 shows the Level 0 Dataflow diagram of the proposed application for simple village animation. The keyboard and mouse devices are used for input to the application. The graphics system processes these user keyboard/mouse interactions using built in OpenGL functions like glutKeyboardFunc(void *) and glutMouseFunc(*void). The Structure is constructed in the memory using the user inputs sent to it by the Graphics System. The structure built in the memory is used by the graphics system to draw the tree onto the screen using OpenGL Functions.

4.3 Flowchart

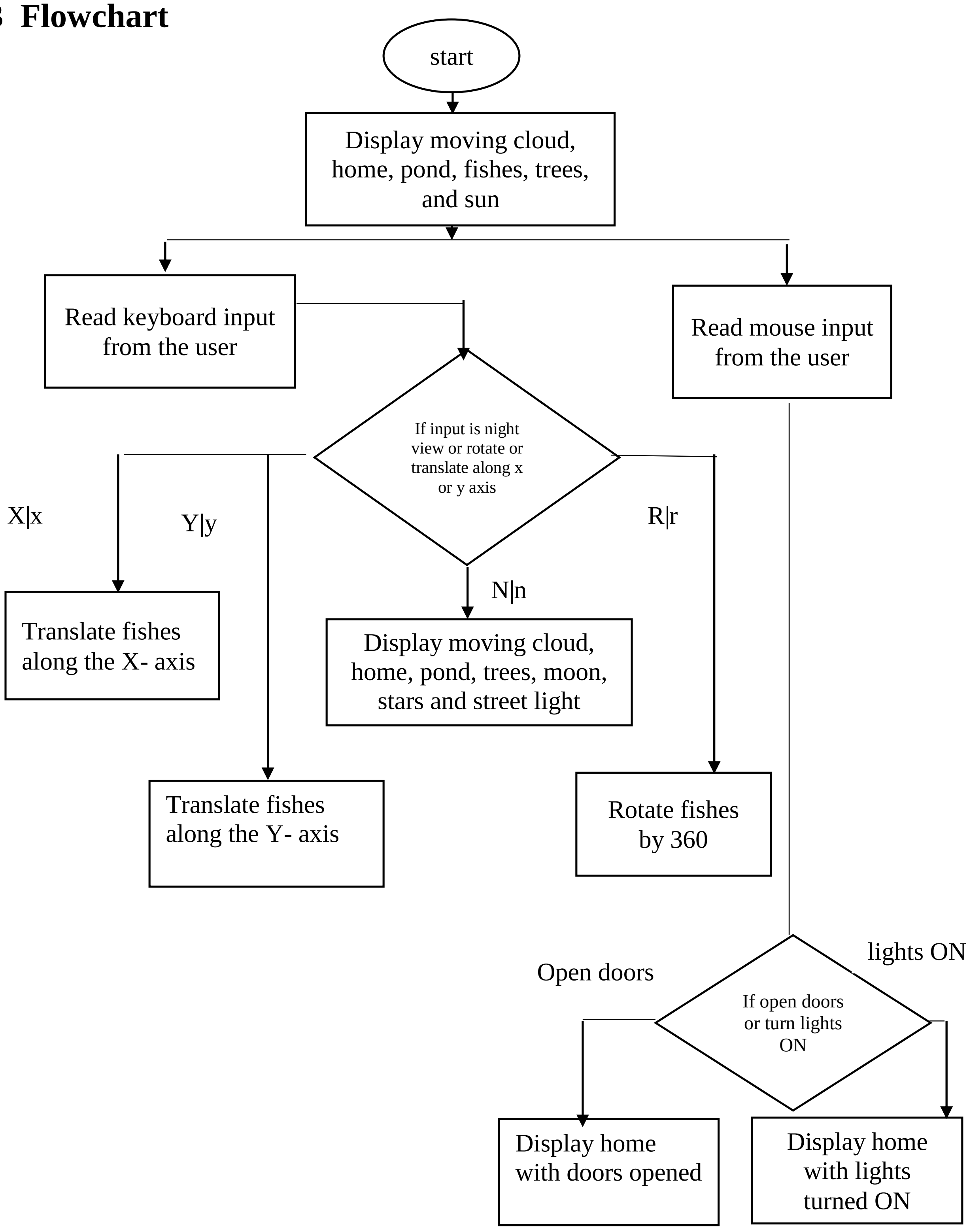
start
Display moving cloud, home, pond, fishes, trees, and sun
Read keyboard input from the user
Read mouse input from the user
If input is night view or rotate or translate along x or y axis
X|x
Y|y
R|r
N|n
Translate fishes along the X- axis
Display moving cloud, home, pond, trees, moon, stars and street light
Translate fishes along the Y- axis
Rotate fishes by 360
lights ON
Open doors
If open doors or turn lights ON
Display home with doors opened
Display home with lights turned ON

Figure 4.3. Flowchart of the proposed system

Chapter 5

IMPLEMENTATION

5.1 Module Description

```
void draw_pixel(GLint cx, GLint cy)
{

   glBegin(GL_POINTS);
   glVertex2i(cx, cy);
   glEnd();
}

void plotpixels(GLint h, GLint k, GLint x, GLint y)
{
   draw_pixel(x + h, y + k);
   draw_pixel(-x + h, y + k);
   draw_pixel(x + h, -y + k);
   draw_pixel(-x + h, -y + k);
   draw_pixel(y + h, x + k);
   draw_pixel(-y + h, x + k);
   draw_pixel(y + h, -x + k);
   draw_pixel(-y + h, -x + k);
}
void draw_circle(GLint h, GLint k, GLint r)
{
   GLint d = 1 - r, x = 0, y = r;
   while (y > x)
   {
      plotpixels(h, k, x, y);
      if (d < 0) d += 2 * x + 3;
      else
      {
         d += 2 * (x - y) + 5;
         --y;
      }
      ++x;
   }
   plotpixels(h, k, x, y);

}
```

```
void sky()
{
   glColor3f(skycolors[0], skycolors[1], skycolors[2]);
   glBegin(GL_POLYGON);
   glVertex2i(-500, 500);
   glVertex2i(500, 500);
   glVertex2i(500, 200);
   glVertex2i(-500, 200);
   glEnd();
}
void cloud()

{
   int l;

   for (l = 0; l <= 35; l++)
   {
      glColor3f(1.0, 1.0, 1.0);
      draw_circle(-400 + m, 420, l);
      draw_circle(-450 + m, 420, l);
   }

   for (l = 0; l <= 20; l++)
   {
      glColor3f(1.0, 1.0, 1.0);
      draw_circle(-350+m, 420, l);

   }
   for (l = 0; l <= 35; l++)
   {
      glColor3f(1.0, 1.0, 1.0);
      draw_circle(-250 + m, 400, l);
      draw_circle(-200 + m, 400, l);
   }

   for (l = 0; l <= 20; l++)
   {
      glColor3f(1.0, 1.0, 1.0);
      draw_circle(-160 + m, 400, l);
   }

   for (l = 0; l <= 35; l++)
   {
      glColor3f(1.0, 1.0, 1.0);
      draw_circle(50+m, 350, l);
      draw_circle(90+m, 350, l);
```

```
    }

    for (l = 0; l <= 20; l++)
    {
        glColor3f(1.0, 1.0, 1.0);
        draw_circle(130+m, 350, l);
    }

}
void sidewall(int x, int y)
{
    glColor3f(sx, sy, sz);
    glBegin(GL_POLYGON);
    glVertex2i(-350 + x, 112 + y);
    glVertex2i(-300 + x, 112 + y);
    glVertex2i(-300 + x, 55 + y);
    glVertex2i(-350 + x, 55 + y);
    glEnd();

}

void windowline(int x,int y)
{
    glColor3f(0.0, 0.0, 0.0);
    glLineWidth(2);
    glBegin(GL_LINES);
    glVertex2i(-325 + x, 112 + y);
    glVertex2i(-325 + x, 55 + y);
    glVertex2i(-350 + x, 81 + y);
    glVertex2i(-300 + x, 81 + y);
    glEnd();

}
void tree(int x, int y)
{
    int l;
    glColor3f(0.5, 0.35, 0.05);
    glBegin(GL_POLYGON);
    glVertex2i(200+x, 200+y);
    glVertex2i(250+x, 200+y);
    glVertex2i(250+x, 50+y);
    glVertex2i(200+x, 50+y);
    glEnd();
    for (l = 0; l <= 35; l++)
    {
        glColor3f(0.0, 0.7, 0.0);
        draw_circle(170+x, 200+y, l);
        draw_circle(220+x, 220+y, l);
        draw_circle(270+x, 200+y, l);
        draw_circle(180+x, 270+y, l);
        draw_circle(255+x, 270+y, l);
        draw_circle(220+x, 290+y, l);
    }
}
void sun()
{
    int l;
    for (l = 0; l <= 35; l++)
    {
        glColor3f(suncolors[0], suncolors[1], suncolors[2]);
        draw_circle(0, 450, l);
    }
}
void door(int x, int y)
{
    glColor3f(dcx, dcy, dcz);
    glBegin(GL_POLYGON);
    glVertex2i(-470 + x, 100 + y);
    glVertex2i(-440 + x, 100 + y);
    glVertex2i(-440 + x, 60 + y);
    glVertex2i(-470 + x, 60 + y);
    glEnd();
    //glFlush();
}

void grass()
{
    glColor3f(grasscolors[0], grasscolors[1], grasscolors[2]);
    glBegin(GL_POLYGON);
    glVertex2i(-500, 200);
    glVertex2i(500, 200);
    glVertex2i(500, -500);
    glVertex2i(-500, -500);
    glEnd();
}
void home(int x, int y)
{
    glColor3f(1.0, 0.6, 1.0);
    //roof
    glBegin(GL_POLYGON);
    glVertex2i(-450+x, 300+y);
    glVertex2i(-300+x, 300+y);
    glVertex2i(-250+x, 200+y);
    glVertex2i(-400+x, 200+y);
    glEnd();

    //triangle
    glColor3f(1.0, 0.0, 1.0);
    glBegin(GL_TRIANGLES);
    glVertex2i(-450+x, 300+y);
    glVertex2i(-500+x, 200+y);
    glVertex2i(-400+x, 200+y);
    glEnd();

    //below traingle
    glColor3f(0.5, 0.35, 0.05);
    glBegin(GL_POLYGON);
    glVertex2i(-500+x, 200+y);
    glVertex2i(-400+x, 200+y);
    glVertex2i(-400+x, 50+y);
    glVertex2i(-500+x, 50+y);
    glEnd();

    // Front Door
    glColor3f(0.8, 0.5, 0.2);
    glBegin(GL_POLYGON);
    glVertex2i(-400+x, 200+y);
    glVertex2i(-250+x, 200+y);
    glVertex2i(-250+x, 50+y);
    glVertex2i(-400+x, 50+y);
    glEnd();

    // Front Door Lock
    door(x,y);

    //side Wall
    sidewall( x, y);

    // line of window one
    windowline(x, y);

}

void pond()
{
    glColor3f(pondcolors[0], pondcolors[1], pondcolors[2]);
    glBegin(GL_POLYGON);
```

```
   glVertex2i(-500, -500);
   glVertex2i(-500, 10);
   glVertex2i(-450, -10);
   glVertex2i(-350, -20);
   glVertex2i(-250, -30);
   glVertex2i(-100, -150);
   glVertex2i(-90, -200);
   glVertex2i(-50, -500);
   glEnd();
}

void dooropen(int x, int y)
{
   glColor3f(1.0, 0.0,1.0);
   glBegin(GL_POLYGON);
   glVertex2i(-470 + x, 100 + y);
   glVertex2i(-493 + x, 95 + y);
   glVertex2i(-493 + x, 55 + y);
   glVertex2i(-470 + x, 60 + y);
   glEnd();

}
void fish(int x, int y, int fcx, int fcy, int fcz)
{
   glColor3f(fcx, fcy, fcz);
   glBegin(GL_TRIANGLES);
   glVertex2i(-350 + x + mx, -30 + y + my);
   glVertex2i(-360 + x + mx, -40 + y + my);
   glVertex2i(-360 + x + mx, -30 + y + my);
   glEnd();
   glBegin(GL_TRIANGLES);
   glVertex2i(-350 + x + mx, -30 + y + my);
   glVertex2i(-360 + x + mx, -40 + y + my);
   glVertex2i(-340 + x + mx, -60 + y + my);
   glEnd();
   glBegin(GL_TRIANGLES);
   glVertex2i(-345 + x + mx, -80 + y + my);
   glVertex2i(-340 + x + mx, -60 + y + my);
   glVertex2i(-325 + x + mx, -70 + y + my);
   glEnd();
}
void pole()
{
   int l = 0;
   glColor3f(0.5, 0.35, 0.05);
   glBegin(GL_POLYGON);
   glVertex2i(450.0,-150.0);
   glVertex2i(450.0, 250.0);
   glVertex2i(460.0,250.0);
   glVertex2i(460.0, -150.0);
   glEnd();
   glBegin(GL_POLYGON);
   glVertex2i(300.0, 150.0);
   glVertex2i(450.0, 150.0);
   glVertex2i(450.0, 140.0);
   glVertex2i(300.0, 165.0);
   glEnd();
   for (l = 0; l <= 25; l++)
   {
      glColor3f(1.0, 1.0, 1.0);
      draw_circle(300, 140, l);
   }
}
void stars()
{
   glColor3f(1.0, 1.0, 1.0);
   glPointSize(5);
   glBegin(GL_POINTS);
   glVertex2i(-400, 400);
   glEnd();
   glBegin(GL_POINTS);
   glVertex2i(-300, 450);
   glEnd();
   glBegin(GL_POINTS);
   glVertex2i(-350, 300);
   glEnd();
   glBegin(GL_POINTS);
   glVertex2i(20, 350);
   glEnd();
   glBegin(GL_POINTS);
   glVertex2i(60, 430);
   glEnd();
   glBegin(GL_POINTS);
   glVertex2i(400, 250);
   glEnd();
   glBegin(GL_POINTS);
   glVertex2i(40, 220);
   glEnd();

}
```

5.2 High Level Code

5.2.1 Built-In Functions

- **void glClear(glEnum mode);**

Clears the buffers namely color buffer and depth buffer. mode refers to GL_COLOR_BUFFER_BIT or DEPTH_BUFFER_BIT.

- **void glTranslate[fd](TYPE x, TYPE y, TYPE z);**

Alters the current matrix by displacement of (x, y, z), TYPE is either GLfloat or GLdouble.
.

- **void glMatrixMode(GLenum mode);**

Specifies which matrix will be affected by subsequent transformations, Mode can be GL_MODELVIEW or GL_PROJECTION.

- **void glLoadIdentity( );**

Sets the current transformation matrix to identity matrix.

- **void glPushMatrix() and void glPopMatrix();**

Pushes to and pops from the matrix stack corresponding to the current matrix mode.

- **void glutInit(int *argc, char **argv);**

Initializes GLUT; the arguments from main are passed in and can be used by the application.

- **void glutInitDisplayMode(unsigned int mode);**

Requests a display with the properties in the mode; the value of mode is determined by the logical OR of options including the color model (GLUT_RGB, GLUT_INDEX) and buffering (GLUT_SINGLE, GLUT_DOUBLE).

- **void glutCreateWindow(char *title);**

Creates a window on display; the string title can be used to label the window. The return value provides a reference to the window that can be used when there are multiple windows.

- **void glutMainLoop();**

Causes the program to enter an event-processing loop

- **void glutDisplayFunc(void (*func)(void))**

Registers the display function func that is executed when the window needs to be redrawn.

- **void glutMouseFunc(void *f(int button, int state, int x, int y)**

Registers the mouse callback function f. The callback function returns the button (GLUT_LEFT_BUTTON,etc., the state of the button after the event (GLUT_DOWN), and the position of the mouse relative to the top-left corner of the window.

- **void glutKeyboardFunc(void *f(char key, int width, int height))**

Registers the keyboard callback function f. The callback function returns the ASCII code of the key pressed and the position of the mouse.

- **void glClearColor(GLclampf r, GLclampf g, GLclamp b, Glclamp a)**

Sets the present RGBA clear color used when clearing the color buffer. Variables of type GLclampf are floating point numbers between 0.0 and 1.0.

- **void glutInitWindowSize(int width, int height);**

Specifies the initial height and width of the window in pixels.

- **void createMenu(void);**

This function is used to create menus which are used as options in program

5.2.2 User Implementation

- **display function**

```
void display()
{

   glClear(GL_COLOR_BUFFER_BIT);
   sky();
   cloud();
   sun();
   grass();
   home(0, 0);
   home(750, -450);
   pond();
   tree(-50, 0);
   tree(150, 100);
   tree(-400, 100);
   if (nightflag != 1)
   {

      glPushMatrix();
```

```
        if (fishrot == 1)
        {

            glTranslatef(-300, -300, 0.0);
            glRotatef(180.0, 0.0, 0.0, 1.0);
            glTranslatef(300, 300, 0.0);
        }
        fish(-40, -300, 1.0, 1.0, 0.0);
        fish(-90, -100, 0.8, 0.5, 0.8);
        fish(-60, -450, 1.0, 0.8, 0.6);
        fish(20, -200, 0.5, 0.2, 1.0);
        fish(100, -200, 0.0, 0.0, 0.0);
        glPopMatrix();
    }
    if (nightflag == 1)
    {
        stars();
        pole();
    }
    if (dflag == 1)
    {
        dooropen(0, 0);
        dooropen(750, -450);
    }


    glFlush();
}
```

- **idle function**

```
void idle()
{
    m += 1;

    if (m > 500)
        m = -500.0;
    display();
}
```

- **mouse function**

```
void select(int c)
{
    switch (c) {
    case 1:
        dflag = 1;
        if (lflag == 1) {
            dcx = 1.0;
            dcy = 1.0;
            dcz = 0.0;
        }
        else {
            dcx = 0.3;
            dcy = 0.5;
            dcz = 0.5;
        }
        glutPostRedisplay();
        break;
    case 2:
        lflag = 1;
        if (dflag == 1)
        {
            dcx = 1.0;
            dcy = 1.0;
            dcz = 0.0;
        }
        sx = 1.0;
        sy = 1.0;
        sz = 0.0;
        glutPostRedisplay();

    }
}
```

- **keyboard function**

```
void keys(unsigned char key)
{
    if (key == 'x' || key == 'X')
    {
          if (mx < 110)
              mx += 5;

    }
    if (key == 'y' || key == 'Y')
    {
            if (my <= 100)
                my += 5;

    }
    if (key == 'r' || key == 'R')
```

```
        fishrot = 1;

    if (key == 'n' || key == 'N')
    {
        nightflag = 1;
        skycolors[0] = 0.0;
        skycolors[1] = 0.0;
        skycolors[2] = 0.0;
        suncolors[0] = 1.0;
        suncolors[1] = 1.0;
        suncolors[2] = 1.0;
        grasscolors[0] = 0.2;
        grasscolors[1] = 0.6;
        grasscolors[2] = 0.3;
        pondcolors[0] = 0;
        pondcolors[1] = 0;
        pondcolors[2] = 1;


    }
}
int main(int argc, char** argv)
{
    printf("Press x and y for translation of fishes");
    printf("Press r for rotation of fishes");
    printf("Press n for night mode");
    printf("Right Click to view Menu");
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(800.0, 700.0);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("Simple Village Animation");
    init();
    glutDisplayFunc(display);
    glutIdleFunc(idle);
    glutCreateMenu(select);
    glutAddMenuEntry("Open Doors", 1);
    glutAddMenuEntry("Turn Lights ON", 2);
    glutKeyboardFunc(keys);
    glutAttachMenu(GLUT_RIGHT_BUTTON);
    glutMainLoop();
}
```

Chapter 6

RESULTS

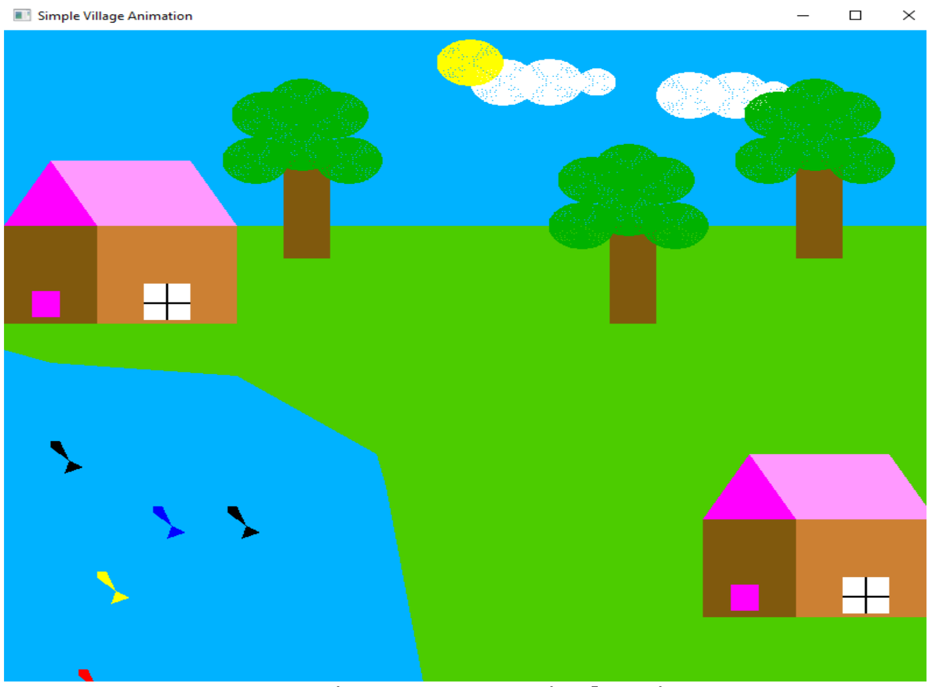

Figure 6.1 Scene in day view

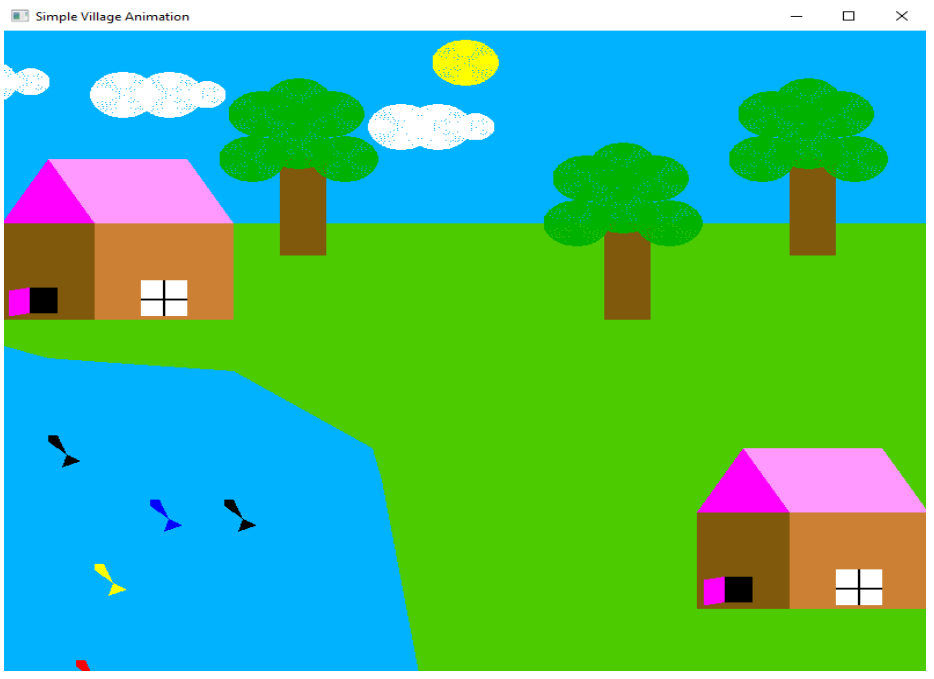

Figure 6.2 Scene when doors are opened

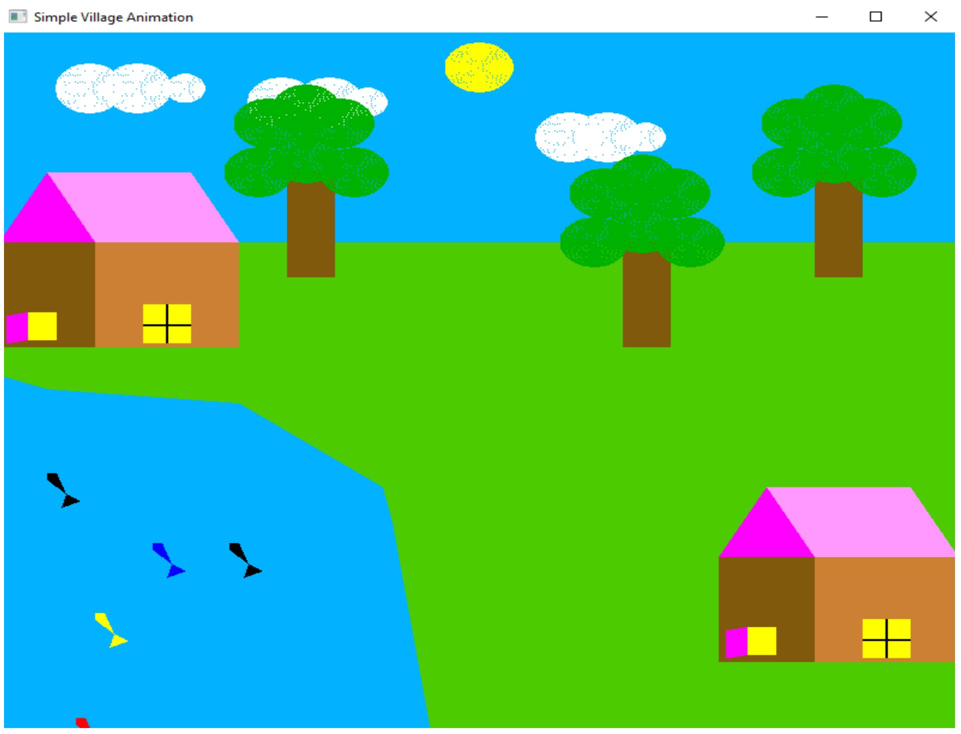

Figure 6.3 Scene when lights are turned ON

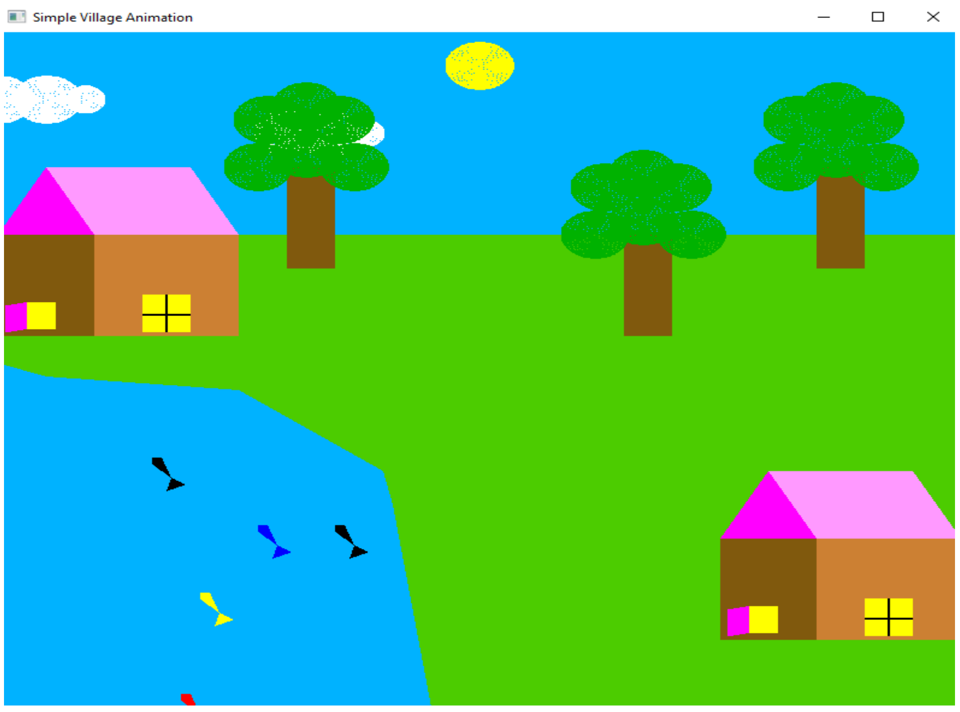

Figure 6.4 Scene when fishes are translated

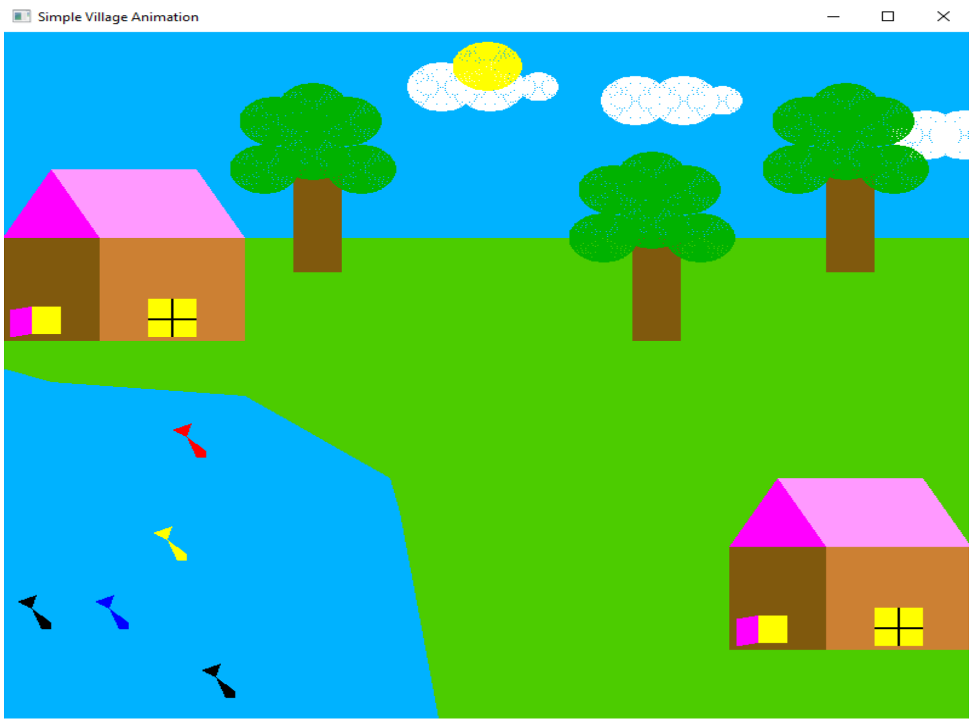

Figure 6.5 Scene when fishes are rotated

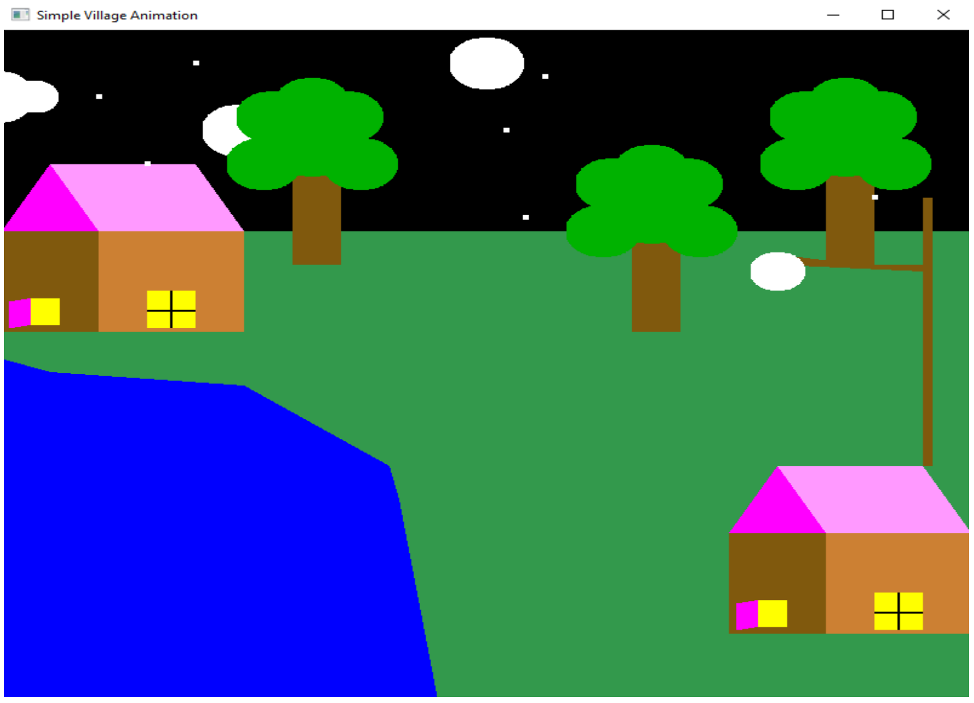

Figure 6.6 Scene in night view

Chapter 7

CONCLUSION AND FUTURE ENHANCEMENTS

This project intends to give an animation of a simple village. From the project, we can witness how the opengl functions work. We also notice how the input events given by the mouse and the keyboard are handled using various opengl functions.

Overall, the project gives a high level graphical view of how a simple village looks like. It allows the user to visualize the changes according to the events generated by him.

Future Enhancements:

- Allow viewer to switch between day and night views when two different keys are pressed.
- Allow viewer to shut the opened doors and turn the lights OFF.
- Addition of new components like cattle area, firewood etc.

BIBLIOGRAPHY

[1]Edward Angel: Interactive Computer Graphics: A Top Down Approach 5th Edition, Addison – Wesley, 2008

[2] Donald Hearn and Pauline Baker: OpenGL, 3rd Edition, Pearson Education, 2004

[3] Wikipedia: Computer Graphics – https://en.wikipedia.org/wiki/ComputerGraphics