VISVESVARAYA TECHNOLOGICAL UNIVERSITY
BELAGAVI

Mini Project Report on

"GRAVITY SIMULATION"

Submitted in the partial fulfillment for the requirements of Computer Graphics & Visualization Laboratory of 6th semester CSE requirement in the form of the Mini Project work

Submitted By

LIKITH S	USN: 1BY18CS081
MANPREET SINGH CHAWLA	USN: 1BY18CS091

Under the guidance of

Mr. SHANKAR R
Assistant Professor, CSE, BMSIT&M

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BMS INSTITUTE OF TECHNOLOGY & MANAGEMENT
YELAHANKA, BENGALURU - 560064.

2020-2021

BMS INSTITUTE OF TECHNOLOGY & MANAGEMENT
YELAHANKA, BENGALURU – 560064

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the Project work entitled **“GRAVITY SIMULATION”** is a bonafide work carried out by **Likith S (1BY18CS081) and Manpreet Singh Chwala (1BY18CS091)** in partial fulfillment for *Mini Project* during the year 2020-2021. It is hereby certified that this project covers the concepts of *Computer Graphics & Visualization*. It is also certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in this report.

Signature of the Guide with date
Mr. SHANKAR R
Assistant Professor
CSE, BMSIT&M

Signature of HOD with date
Dr. BHUVANESHWARI.C.M

Prof & Head
CSE, BMSIT&M

INSTITUTE VISION

To emerge as one of the finest technical institutions of higher learning, to develop engineering professionals who are technically competent, ethical and environment friendly for betterment of the society.

INSTITUTE MISSION

Accomplish stimulating learning environment through high quality academic instruction, innovation and industry-institute interface.

DEPARTMENT VISION

To develop technical professionals acquainted with recent trends and technologies of computer science to serve as valuable resource for the nation/society.

DEPARTMENT MISSION

Facilitating and exposing the students to various learning opportunities through dedicated academic teaching, guidance and monitoring.

PROGRAM EDUCATIONAL OBJECTIVES

1. Lead a successful career by designing, analysing and solving various problems in the field of Computer Science & Engineering.
2. Pursue higher studies for enduring edification.
3. Exhibit professional and team building attitude along with effective communication.
4. Identify and provide solutions for sustainable environmental development.

ACKNOWLEDGEMENT

We are happy to present this project after completing it successfully. This project would not have been possible without the guidance, assistance, and suggestions of many individuals. We would like to express our deep sense of gratitude and indebtedness to each and everyone who has helped us make this project a success.

We heartily thank our Principal, Dr. MOHAN BABU G N, BMS Institute of Technology &Management, for his constant encouragement and inspiration in taking up this project.

We heartily thank our Professor and Head of the Department, Dr. BHUVANESHWARI.C.M, Department of Computer Science and Engineering, BMS Institute of Technology &Management, for his constant encouragement and inspiration in taking up this project.

We gratefully thank our Project Guide, Mr. Shankar R, Assistant Professor, Department of Computer Science and Engineering for his intangible support and for being a constant backbone for our project.

Special thanks to all the staff members of the Computer Science Department for their help and kind cooperation.

Lastly, we thank our parents and friends for the support and encouragement given throughout in completing this precious work successfully.

LIKITH S (1BY18CS081)

MANPREET SINGH CHWALA (1BY18CS091)

ABSTRACT

Newton's law of gravitation, statement that any particle of matter in the universe attracts any other with a force varying directly as the product of the masses and inversely as the square of the distance between them. Two big objects can be considered as point-like masses, if the distance between them is very large compared to their sizes or if they are spherically symmetric. For these cases, the mass of each object can be represented as a point mass located at its center of mass. In symbols, the magnitude of the attractive force F is equal to G (the gravitational constant, a number the size of which depends on the system of units used and which is a universal constant) multiplied by the product of the masses (m1 and m2) and divided by the square of the distance R: F = G(m1m2)/R2.

This is a simulation portrays the simulation of gravity between particles according to their masses as newton's law of universal gravitation initially a big particle is created at the origin with infinity mass that will attract any new particle created by the user as the new particle is smaller than the big mass, so the gravitation force accelerates the small particles towards the origin (big mass). The acceleration of an object as produced by a net force is directly proportional to the magnitude of the net force, in the same direction as the net force, and inversely proportional to the mass of the object i.e., F = m * a. The application integrates the concepts of computer graphics, Open GL API, C/C++ programming with the concepts of universal gravity and its formulae.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Brief Introduction

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 take 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 Motivation

Gravity simulation makes it an ideal option for scientists to predict and analyze the gravitational nature of masses and use in their solutions in computations. The ability to visualize how gravity exerts a force upon objects and its magnitude estimation will give us valuable insight into its extensive real-life results. The ability to develop this visualization using C programming and the OpenGL API serves as a motivation to develop this application.

1.3 Problem Statement

The aim of this application is to show a basic implementation of gravity between particles according to their masses as newton's law of universal gravitation initially a big particle is created at the origin with infinity mass that will attract any new particle created by the user as the new particle is smaller than the big mass so the gravitation force accelerates the small particles towards the origin. The application will be implemented using the C++

programming language and the OpenGL API. This is done using the concepts and principles of computer graphics. The application will also include user interaction through mouse events, for creating new mass elements to simulate gravity that behaves depending on their masses and distance between them and other masses.

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 are 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. The phrase was coined in 1960, by computer graphics researchers Verne Hudson and William Fetter of Boeing. 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, and physics. Computer graphics is responsible for displaying art and image data effectively and meaningfully to the user. It is also used for processing image data received from the physical world. Computer graphic development has had a significant impact on many types of media and has revolutionized animation, movies, advertising, video games, and graphic design generally.

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. Since 2006 OpenGL has been managed by the non-profit technology consortium Kronos Group.

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 that 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.

Given that creating an OpenGL context is quite a complex process, and given that it varies between operating systems, automatic OpenGL context creation has become a common feature of game development and user-interface libraries, including SDL, Allegro, SFML, FLTK, and Qt. A few libraries have been designed solely to produce an OpenGL-capable window. The first such library was OpenGL Utility Toolkit (GLUT), later superseded by free glut. GLFW is a newer alternative.

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 the primitive assembly, is the elimination of portions of geometry that 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. Color and depth values are assigned for each fragment square.

Fragment Operations:

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

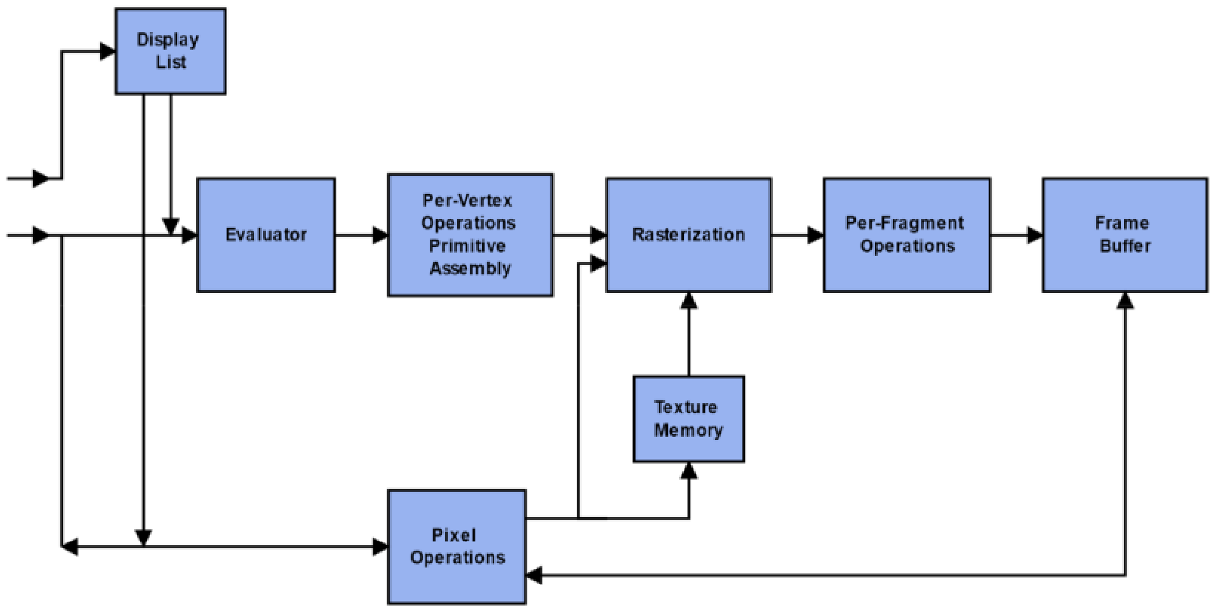


Fig 1.1 An illustration of the graphics pipeline process in OpenGL Architecture

1.6 Applications of Computer Graphics

Although many applications span two, three, or even all 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 the interactive design of charts incorporating color, 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 several 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. Computer-generated images are also the heart of flight simulators, which have become the standard method for training pilots.

1.6.4 User Interfaces

The interface between the human and the computer has been radically altered using 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) will 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 a cinema without a film. The 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 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 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 is 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 the 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 engineering and Architecture students are concerned with Design. CAD is used to design various structures such as Computers, Aircraft, buildings, 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 that can be like 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. Virtual reality typically incorporates auditory and video feedback but may also allow other types of sensory and force feedback through haptic technology.

Video Games

A video game is an electronic game that involves interaction with a user interface to generate visual feedback on two- or three-dimensional video display devices 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 laptops, desktops, or handheld computing devices.

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: CodeBlocks:: 17.12
- 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
- Secondary Storage: 250GB
- Display Resolution: 1366x768 (1920x1080 recommended)

Chapter 4

SYSTEM DESIGN

4.1 Proposed System

Newton's laws of motion show that objects at rest will stay at rest and those in motion will continue moving uniformly in a straight line unless acted upon by a force. Thus, it is the *straight line* that defines the most natural state of motion. But the planets move in ellipses, not straight lines; therefore, some force must be bending their paths. That force, Newton proposed, was gravity.

In Newton's time, gravity was something associated with Earth alone. Everyday experience shows us that Earth exerts a gravitational force upon objects at its surface. If you drop something, it accelerates toward Earth as it falls. Newton's insight was that Earth's gravity might extend as far as the Moon and produce the force required to curve the Moon's path from a straight line and keep it in its orbit. He further hypothesized that gravity is not limited to Earth, but that there is a general force of attraction between all material bodies. If so, the attractive force between the Sun and each of the planets could keep them in their orbits. (This may seem part of our everyday thinking today, but it was a remarkable insight in Newton's time.)

Once Newton boldly hypothesized that there was a universal attraction among all bodies everywhere in space, he had to determine the exact nature of the attraction. The precise mathematical description of that gravitational force had to dictate that the planets move exactly as Kepler had described them to (as expressed in Kepler's three laws).

Eventually, he was able to conclude that the magnitude of the force of gravity must decrease with increasing distance between the Sun and a planet (or between any two objects) in proportion to the inverse square of their separation. In other words, if a planet were twice as far from the Sun, the force would be (1/2)2, or 1/4 as large. Put the planet three times farther away, and the force is (1/3)2 or 1/9 as large.

Newton also concluded that the gravitational attraction between two bodies must be proportional to their masses. The more mass an object has, the stronger the pull of its gravitational force. The gravitational attraction between any two objects is therefore

given by one of the most famous equations in all of science:

$$Fgravity = G * ((M1M-2) / R2)$$

where Fgravity is the gravitational force between two objects, M1 and M2 are the masses of the two objects, and R is their separation. G is a constant number known as the *universal gravitational constant*, and the equation itself symbolically summarizes Newton's *universal law of gravitation*. With such a force and the laws of motion, Newton was able to show mathematically that the only orbits permitted were exactly those described by Kepler's laws.

4.3 Flowchart

A flowchart is a visual representation of the sequence of steps and decisions needed to perform a process. Each step in the sequence is noted within a diagram shape. Steps are linked by connecting lines and directional arrows. The flowchart shown depicts how the proposed system works in different steps sequentially.

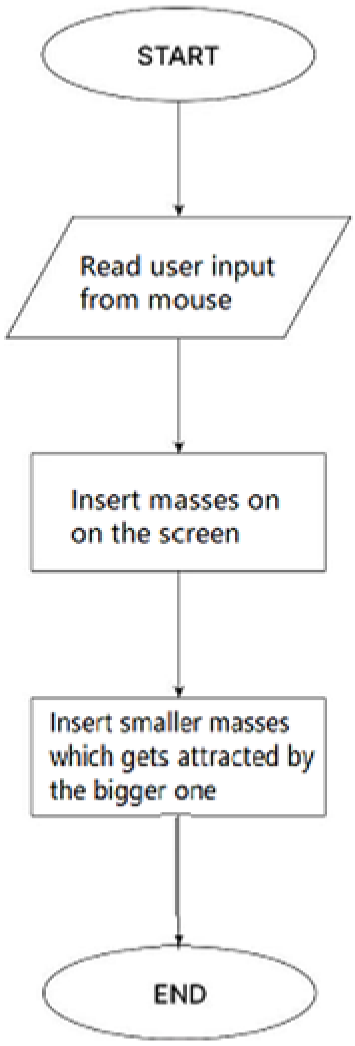

START
Read user input from mouse
Insert masses on on the screen
Insert smaller masses which gets attracted by the bigger one
END

Figure 4.2 Flowchart of the Proposed System

Chapter 5

IMPLEMENTATION

5.1 Module Description

· void timer (int)

This function is responsible for the entire working of the simulation where every move is timed according to the functions and formulae embedded here.

· void mouse (int, int, int, int)

This function is responsible for the state and coordinates of the masses placed on the window based on the passed parameters.

· void mouseMotion (int, int)

This function plays the role of detecting the mouse movements and the dragged motion to release a particle towards the huge mass.

· void addParticle (float, float, bool, float, float)

This function places the masses or the particles based on the click of the mouse. If the user presses right a huge particle is placed, and a small particle is placed if the left is clicked.

· void removePartilces()

This function removes all the particles present on the screen and clears the screen.

· void keyboard (unsigned char, int, int)

This function accepts the input from the keyboard but only one key i.e., ‘S’ which is designated as the speed particle whose size is little more than the small particles.

5.2 High Level Code

5.2.1 Built-In Functions

· **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 glutInitWindowSize(int width, int height);**

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

· **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 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 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 the identity matrix.

· **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 glColor3[b I f d ub us ui](TYPE r, TYPE g, TYPE b);**

Sets the present RGB colors. Valid types are bytes(b), int(i), float(f), double(d), unsigned byte(ub), unsigned short(us), and unsigned int(ui). The maximum and minimum values for floating-point types are 1.0 and 0.0 respectively, whereas the maximum and minimum values of the discrete types are those of the type, for eg, 255 and 0 for unsigned bytes.

· void glutMainLoop ( );

This causes the program to enter an event-processing loop.

5.2.2 User Implementation

· **Structure for particle and line**

```
struct Particle {

     float x;

     float y;
```

```
        float r;
        float vx;
        float vy;
        float m;
        float color[3];
};
struct Line {
        float x1;
        float y1;
        float x2;
        float y2;
} line;
```

- **Functions and conditions**

```
void timer(int = 0);
void display();
void mouse(int, int, int, int);
void mouseMotion(int, int);
void addParticle(float, float, bool = true, float = 0, float = 0);
void removeParticles();
void keyboard(unsigned char, int, int);
int Mx, My, WIN;
bool PRESSED_LEFT = false, PRESSED_RIGHT = false,
PRESSED_MIDDLE = false, SPEED_PARTICLES = false;
std::vector<Particle> particles;
```

- **Main Function**

```
int main(int argc, char **argv)

{

        Particle p;

        //initial centered Huge mass particle

        p.x = 0;

        p.y = 0;

        p.vx = p.vy = 0;

        p.m = 10000;

        p.r = 10;

        p.color[0] = 1;

        p.color[1] = 1;

        p.color[2] = 0;

        particles.push_back(p);


        glutInit(&argc, argv);

        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);

        glutInitWindowSize(500, 500);

        glutInitWindowPosition(50, 50);

        WIN = glutCreateWindow("Gravity");

        glClearColor(0, 0, 0, 1);

        glMatrixMode(GL_PROJECTION);

        glLoadIdentity();

        glOrtho(-250.0, 250.0, 250.0, -250.0, 0, 1);

        glutDisplayFunc(display);
```

```
    glutMouseFunc(mouse);

    glutMotionFunc(mouseMotion);

    glutKeyboardFunc(keyboard);

    timer();

    glutMainLoop();

    return 0;

}
```

· Timer Function

```
void timer(int)

{

    display();

    if(PRESSED_LEFT && !SPEED_PARTICLES)

      {

        addParticle(10, 3); //add tiny particle

        PRESSED_LEFT = false;

      }

    if(PRESSED_RIGHT)

      {

        addParticle(10000, 10, 0); //add huge particle

        PRESSED_RIGHT = false;

      }

    if(PRESSED_MIDDLE)

        removeParticles(); //remove all particles

    for(int i = 0; i < particles.size(); i++)

      {

        Particle &p = particles[i];
```

```
            bool not_fall = true;
            for(int j = 0; j < particles.size(); j++)
            {
                if(j == i || p.m >= 10000)
                        continue;
                const Particle &p1 = particles[j];
                float d = sqrt((p1.x - p.x)*(p1.x - p.x) + (p1.y - p.y)*(p1.y - p.y));
                if(d > p1.r)
                {
                        p.vx += 0.03 * p1.m / (d*d) * (p1.x - p.x)/d; //f = ma => a = f/m
                        p.vy += 0.03 * p1.m / (d*d) * (p1.y - p.y)/d;
                }
                else
                        not_fall = false;
            }
            if(not_fall)
            {
                p.x += p.vx;
                p.y += p.vy;
            }
            else
                particles.erase(particles.begin()+i);
          }
        glutTimerFunc(1, timer, 0);
}
```

· **Display Function**

```
void display()
{
	glClear(GL_COLOR_BUFFER_BIT);
	//draw the drag line
	glColor3f(0, 0, 1);
	glBegin(GL_LINES);
		glVertex2f(line.x1, line.y1);
		glVertex2f(line.x2, line.y2);
	glEnd();
	//draw particles
	for(int i = 0; i < particles.size(); i++)
	   {
		Particle &p = particles[i];
		glColor3f(p.color[0], p.color[1], p.color[2]);
		glBegin(GL_POLYGON);
		for(float a = 0; a < 2*M_PI; a+=0.2)
			glVertex2f(p.r*cos(a) + p.x, p.r*sin(a) + p.y);
		glEnd();
	   }
	glFlush();
	glutSwapBuffers();
}
```

- **Mouse Function**

```
void mouse(int button, int state, int x, int y)
{	//set the coordinates
```

```
        Mx = x - 250;

        My = y - 250;

        //add speed particles by line dragging

        if(SPEED_PARTICLES)

        {

                if(line.x2 != 0 && line.y2 != 0 && state == GLUT_UP &&
PRESSED_LEFT)

                addParticle(100, 5, 1, line.x1 - line.x2, line.y1 - line.y2); //speed par

                else

                {

                line.x1 = line.x2 = Mx;

                line.y1 = line.y2 = My;

                }

        }

        //check which button is pressed

        if(button == GLUT_LEFT_BUTTON)

                PRESSED_LEFT = state == GLUT_DOWN;

        else if(button == GLUT_RIGHT_BUTTON)

                PRESSED_RIGHT = state == GLUT_DOWN;

        else if(button == GLUT_MIDDLE_BUTTON)

           PRESSED_MIDDLE = state == GLUT_DOWN;

}
```

· **Mouse Motion Function**

```
void mouseMotion(int x, int y)

{

        Mx = x - 250;

        My = y - 250;
        //end of line with dragging
        if(SPEED_PARTICLES && PRESSED_LEFT)
        {
           line.x2 = Mx;
                line.y2 = My;
        }
}
```

· **Function to add new particles**

```
void addParticle(float m, float r, bool randColor, float vx, float vy)
{
        Particle p;
        p.x = Mx;
        p.y = My;
        p.vx = vx / 30; // /30 in case it is a speed particle,
        p.vy = vy / 30; // slow down the speed a little
        p.m = m;
        p.r = r;
        if(randColor)
        {
                p.color[0] = rand()%200 / 200.0;
                p.color[1] = rand()%200 / 200.0;
                p.color[2] = rand()%200 / 200.0;
        }
        else // if is a huge particle make it yellow
        {
                p.color[0] = 1;
                p.color[1] = 1;
                p.color[2] = 0;
        }
        particles.push_back(p);
        if(line.x1 != 0)
                line.x1 = line.x2 = line.y1 = line.y2 = 0;
}
```

- **Function to remove particles**

```
void removeParticles()
{
        for(int i = 0; i < particles.size(); i++)
                particles.pop_back();
}
```

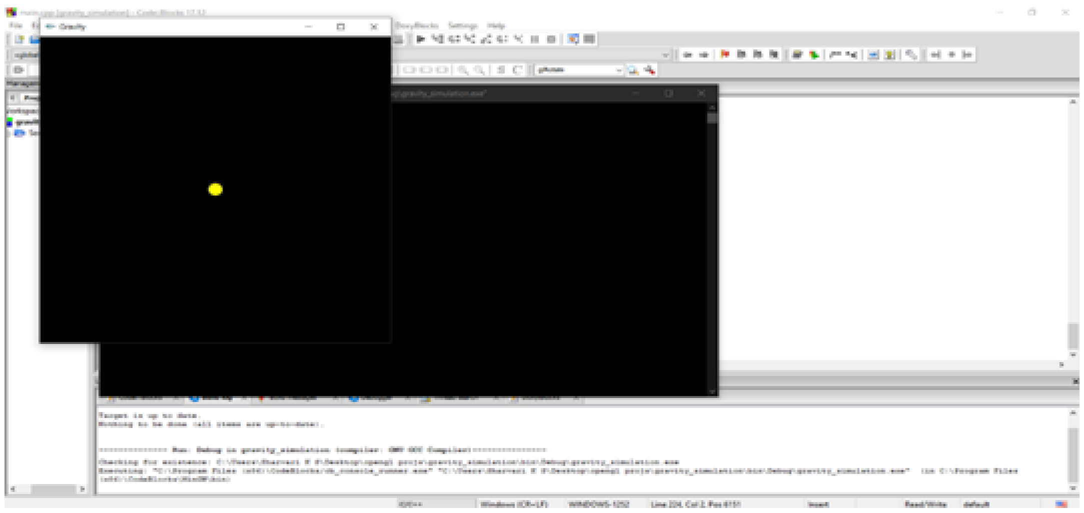
- **Keyboard input function**

```
void keyboard(unsigned char key, int x, int y)
{
        switch(key)
        {
                case 's':
                SPEED_PARTICLES = !SPEED_PARTICLES;
                break;
                case 27:
                 removeParticles();
                glutDestroyWindow(WIN);
                exit(0);
                break;
        }
    }
```

CHAPTER 6

RESULTS

INITIAL WINDOW



```
Target is up to date.
Nothing to be done (all items are up-to-date).

-------------- Run: Debug in gravity_simulation (compiler: GNU GCC Compiler)---------------
Checking for existence: C:\Users\Bhargavi K F\Desktop\opengl projs\gravity_simulation\bin\Debug\gravity_simulation.exe
Executing: "C:\Program Files (x86)\CodeBlocks/cb_console_runner.exe" "C:\Users\Bhargavi K F\Desktop\opengl projs\gravity_simulation\bin\Debug\gravity_simulation.exe"  (in C:\Program Files (x86)\CodeBlocks\MinGW\bin)
```

Fig 6.1: Initial Window

This is the window that appears first when the user builds and runs the program. It consists of a single huge particle at the center of the window.

SMALL PARTICLES

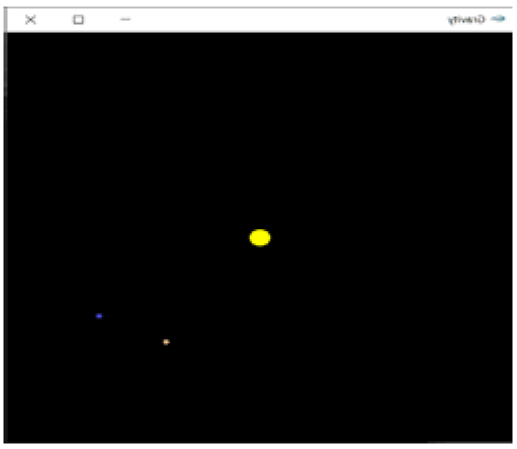


Fig 6.2: Small particles

These small particles are formed when the user clicks the left button on the mouse. These small particles tend to get attracted towards the center huge particle from whichever coordinate they are present in the window.

HUGE PARTICLES

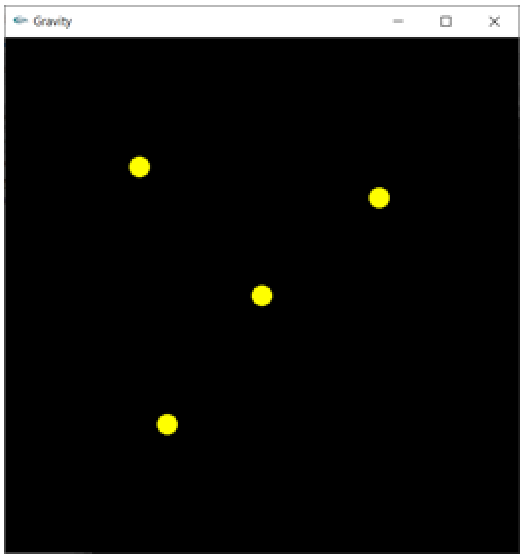


Fig 6.3: Huge particles

Huge particles can be placed by clicking right on the mouse which acts as the heavier particle that attracts the smaller particles.

HUGE AND SMALL PARTICLES

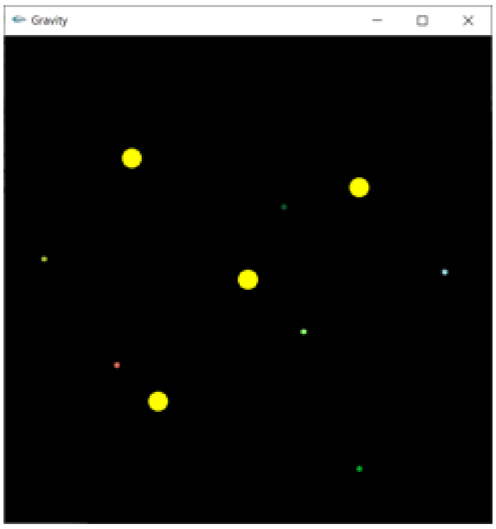


Fig 6.4: Huge and small particles

Here the small particles that are created by left-clicking are being attracted by the huge particles according to the universal law of gravitation.

HUGE AND SPEED PARTICLES

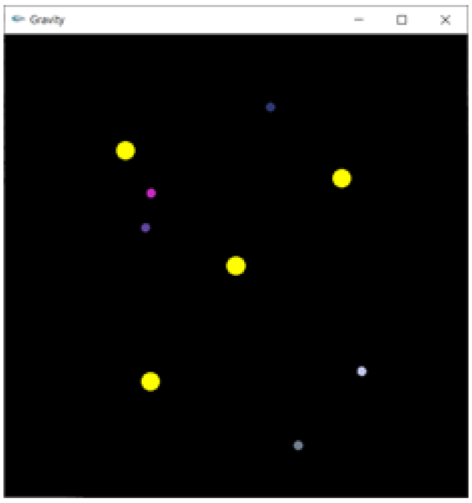


Fig 6.5: Huge and speed particles

Here the speed particles are created by pressing s on the keyboard and clicking left on the mouse. These speed particles are a little heavier than the small particles and they get attracted towards the huge particles.

SPEED PARTICLES BY DRAGGING

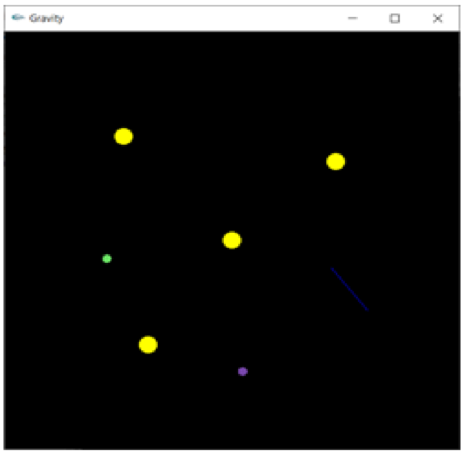


Fig 6.6: Speed particles by dragging

By dragging the mouse towards the required direction, the particles can be directed when the clicked left button is released.

REMOVING PARTICLES

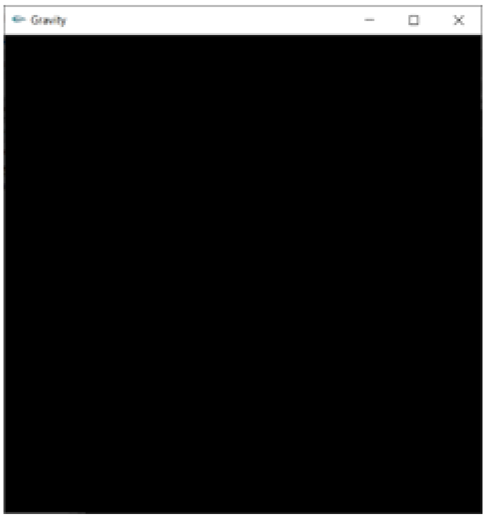


Fig 6.7: Removing of particles

By clicking the middle button on the mouse all the particles are removed and the screen is cleared.

SMALL PARTICLES

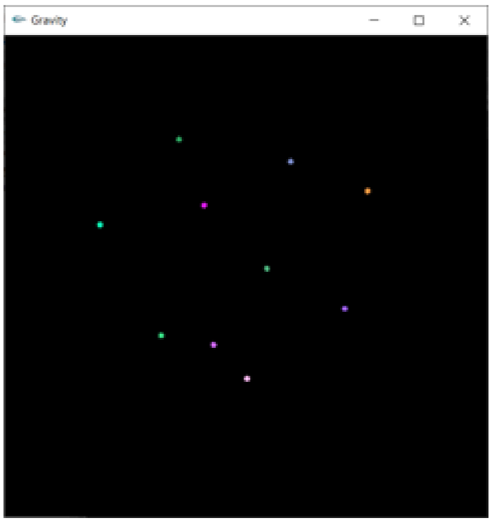


Fig 6.8: Only small particles

When only small particles are created on the screen, they tend to attract each other and combine. The movement of every small particle is determined by its neighboring small particles.

SPEED PARTICLES

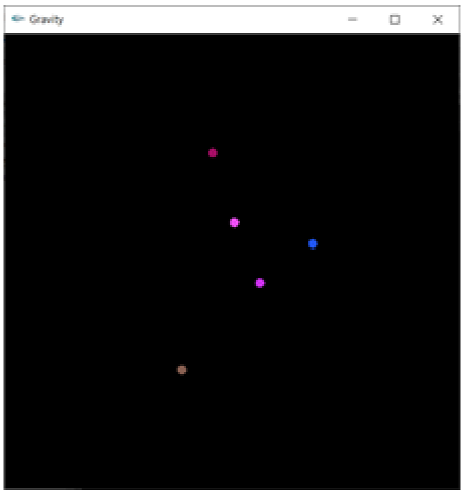


Fig 6.9: Only speed particles

When only speed particles are created on the screen, they tend to attract each other and combine. The movement of every speed particle is determined by its neighboring speed particles.

SPEED AND SMALL PARTICLES

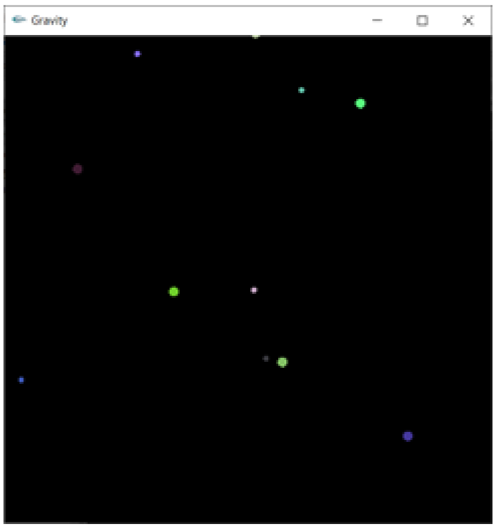


Fig 6.10: Speed and small particles

When speed and small particles are created on the screen, they tend to attract each other and combine. The movement of every particle is determined by its neighboring particles which pull towards each other by following the law of universal gravity.

SPEED AND SMALL PARTICLES BY DRAGGING

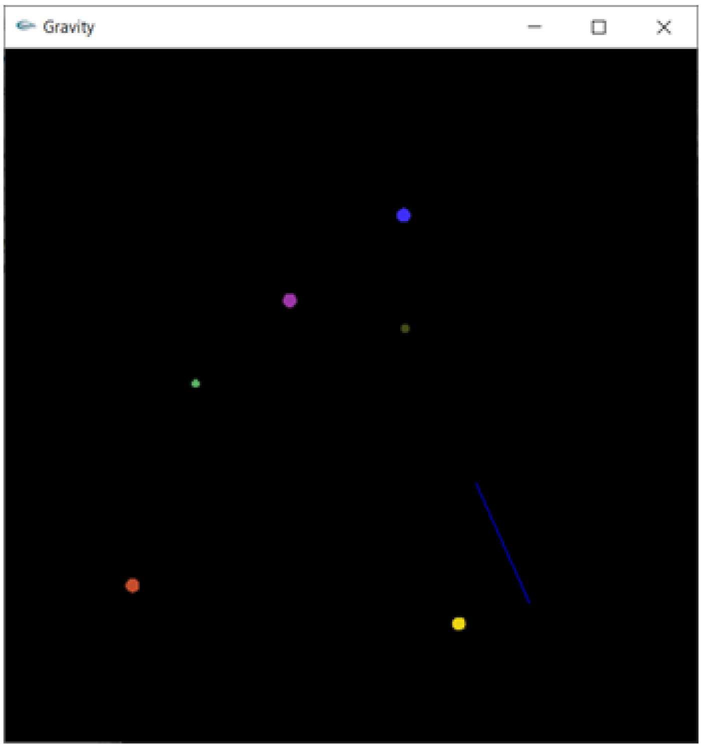


Fig 6.11: Speed and small particles by dragging

Chapter 7

CONCLUSION AND FUTURE ENHANCEMENTS

In conclusion, based on the acceptable Newton's gravitational law of gravitation, gravitation is a mutual force. Every particle in the universe attracts every other particle with a force that is directly proportional to the product of their masses and inversely proportional to the square of the distance between them. Therefore, the gravitational force is depending on the masses of the bodies and the distance between the two bodies. In the process of building this application, various OpenGL API Functions and variables were utilized that made it easier to visualize the applications working along with the working of the functions that were used.

In the future, this application can be further enhanced by adding more functionality to show more operations on the working of the simulation. There can be the implementation of the same in 3D which helps in better understanding and more view on all the perspectives of the actions. More built-in functions and optimal codes can be used to have a smooth simulation.

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

4. Gravity Simulation: http://phet.colorado.edu/sims/html/gravity-and-orbits/latest/gravity-and-orbits_en.html