VISVESVARAYA TECHNOLOGICAL UNIVERSITY

BELAGAVI-590018

A Computer Graphics Mini Project Report

on

3D BICYCLE SIMULATION

Submitted in partial fulfillment of the requirements for the VI semester

Computer Science and Engineering of Visvesvaraya Technological University, Belagavi

Submitted by:

PALAK GUPTA	*1RN20CS096*
PRAHLAD VINAYAK AVADHANI	*1RN20CS099*

Under the Guidance of:

Dr. A N Ramya Shree
Associate Professor
Dept. of CSE

Department of Computer Science and Engineering
(Accredited by NBA up to 30/6/2025)
RNS Institute of Technology
Channasandra, Dr.Vishnuvardhan Road, Bengaluru-560 098

2023

RNS INSTITUTE OF TECHNOLOGY

Channasandra, Dr.Vishnuvardhan Road, Bengaluru-560098

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

(Accredited by NBA up to 30/6/2025)

CERTIFICATE

This is to certify that the mini project work entitled **3D BICYCLE SIMULATION** has been successfully carried out by **PALAK GUPTA** bearing USN **1RN20CS096** and **PRAHLAD VINAYAK AVADHANI** bearing USN **1RN20CS099**, bonafide students of **RNS Institute of Technology** in partial fulfillment of the requirements for the 6th semester **Computer Science and Engineering of Visvesvaraya Technological University"**, Belagavi, during academic year 2023. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library. The project report has been approved as it satisfies the CG laboratory requirements of 6th semester BE, CSE.

Signature of the Guide	Signature of the HoD	Signature of the Principal
Dr. A N Ramya Shree	**Dr. Kiran P**	**Dr. Ramesh Babu H S**
Associate Professor	Professor & Head	Principal
Dept. of CSE	Dept. of CSE	

External Viva:

Name of the Examiners	Signature with Date
1.	
2.	

Acknowledgement

The successful completion of any achievement is not solely dependent on individual efforts but also on the guidance, encouragement, and cooperation of intellectuals, elders, and friends. We would like to take this opportunity to express our heartfelt gratitude to all those who have contributed to the successful execution of this project.

First and foremost, we extend our profound thanks to **Sri. Satish R Shetty**, Managing Trustee of R N Shetty Trust and Chairman of RNS Group of Institutions, and **Sri. Karan S Shetty** , CEO of RNS Group of Institutions, Bengaluru, for providing a conducive environment that facilitated the successful completion of this project.

We would also like to express our sincere appreciation to our esteemed Director, **Dr. M K Venkatesha**, for providing us with the necessary facilities and support throughout the duration of this work.

Our heartfelt thanks go to our respected Principal, **Dr. Ramesh Babu H S**, for his unwavering support, guidance, and encouragement that played a vital role in the completion of this project.

We would like to extend our wholehearted gratitude to our HOD, **Dr. Kiran P**, Professor, and Head of the Department of Computer Science & Engineering, RNSIT, Bangalore, for his valuable suggestions and expert advice, which greatly contributed to the success of this endeavor.

A special word of thanks is due to our project guide,**Dr. A N Ramya Shree**, Associate Professor in the Department of CSE, RNSIT, Bangalore, for her exceptional guidance, constant encouragement, and unwavering assistance throughout the project.

We would also like to express our sincere appreciation to all the teaching and non-teaching staff of the Department of Computer Science & Engineering, RNSIT, for their consistent support and encouragement.

Once again, we express our deepest gratitude to everyone involved, as their support and cooperation were instrumental in the successful completion of this project.

Abstract

The 3D Bicycle Simulation project aims to develop a realistic and immersive virtual environment that replicates the experience of riding a bicycle. By leveraging computer graphics techniques, physics simulation, and intuitive interfaces, the project seeks to provide an engaging and valuable experience for users. The project's applications range from training and skill development to research, rehabilitation, and entertainment. This abstract provides a concise overview of the project's objectives, highlighting its potential impact in various domains. The subsequent sections of the project report delve into the project's history, resource requirements, system design, and other relevant aspects.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview of 3D Bicycle Simulation CGV Project:

The 3D Bicycle Simulation Computer Graphics and Visualization (CGV) project focuses on the development of a realistic and immersive virtual environment that simulates the experience of riding a bicycle. By leveraging cutting-edge computer graphics techniques and visualization technologies, this project aims to create a visually appealing and interactive simulation that closely mimics real-world cycling scenarios.

The primary goal of the 3D Bicycle Simulation CGV project is to provide users with a virtual platform where they can experience the thrill of cycling in a controlled and customizable environment. The simulation utilizes advanced rendering techniques to create visually stunning graphics, including detailed bicycle models, realistic terrains, and dynamic lighting effects. By incorporating high-fidelity visuals, the project seeks to enhance the sense of immersion and realism for the users.

Another crucial aspect of the project is the implementation of accurate physics simulation. The virtual bicycle's movement and behavior are meticulously modeled to replicate the dynamics of real-world cycling. Factors such as gravity, inertia, friction, and wind resistance are taken into account to ensure that the virtual bicycle responds realistically to user input and environmental conditions. This physics simulation adds a layer of authenticity to the simulation, making it a valuable tool for training, research, and analysis.

The 3D Bicycle Simulation CGV project also places a strong emphasis on user interfaces and controls. The goal is to provide intuitive and user-friendly interfaces that allow users to interact with the simulation effortlessly. Through the development of responsive controls and customizable options, users can adjust various parameters such as riding speed, terrain difficulty, and weather conditions to create a personalized and challenging experience.

Furthermore, the project explores additional features and functionalities to enhance the overall experience. These may include virtual competitions, time trials, training programs, and the ability to track performance metrics such as speed, distance, and calories burned. By incorporating these elements, the simulation aims to cater to a wide range of users, from casual riders seeking entertainment to professional cyclists looking to improve their skills.

The 3D Bicycle Simulation CGV project not only serves as a recreational tool but also holds potential in various domains. It can be utilized for virtual training programs, allowing cyclists to practice and refine their skills in a safe and controlled environment. Additionally, the simulation can aid in rehabilitation by providing a platform for physical therapy exercises and motor skill development. Furthermore, researchers and developers can leverage the simulation to explore advancements in virtual reality, computer graphics, and human-computer interaction.

1.2 History:

The history of the 3D Bicycle Simulation CGV project can be traced back to the advancements in computer graphics, virtual reality, and interactive simulation technologies. Over the years, there has been a growing interest in creating realistic virtual environments for various applications, including gaming, training, and research. The idea of simulating the experience of riding a bicycle in a virtual setting emerged as a natural extension of these developments.

The earliest iterations of 3D bicycle simulations can be found in the field of video games. As early as the 1990s, cycling video games started to incorporate rudimentary representations of bicycles and virtual terrains. These games provided a basic simulation of cycling, allowing players to control a virtual cyclist and navigate through various environments. However, the graphical fidelity and physics simulation in these early versions were limited compared to the advancements seen in recent years.

With the progress of computer graphics and simulation technologies, the concept of 3D bicycle simulation expanded beyond gaming. Researchers and developers recognized the potential of creating realistic and immersive virtual environments for training, rehabilitation, and research purposes. This led to the emergence of dedicated projects focused on enhancing the visual quality, physics accuracy, and user interaction aspects of bicycle simulations.

One important milestone in the history of 3D bicycle simulation was the integration of advanced graphics techniques, such as real-time rendering, shading, and texture mapping. These advancements allowed for more detailed and visually appealing virtual environments, bringing the simulation closer to reality. Concurrently, improvements in physics simulation algorithms enabled the accurate modeling of bicycle dynamics, including factors such as weight, balance, aerodynamics, and terrain

interaction.

The development of virtual reality (VR) technologies also had a significant impact on 3D bicycle simulations. VR headsets and motion tracking systems provided a more immersive experience, enabling users to feel as if they were actually riding a bicycle in a virtual world. This integration of VR into bicycle simulations enhanced the sense of presence and realism,contributing to a more engaging and authentic experience.

As the project progressed, user interfaces and controls became a key focus area. Developers aimed to create intuitive and user-friendly interfaces that would allow users to interact with the simulation seamlessly. This involved designing responsive control systems that accurately translated user input into virtual bicycle movements, as well as providing customizable options for adjusting riding parameters, such as speed, difficulty, and environmental conditions.

The history of the 3D Bicycle Simulation CGV project is a testament to the continuous advancements in computer graphics, simulation, and interactive technologies. It reflects the ongoing pursuit of creating increasingly realistic and immersive virtual environments for cyclists and enthusiasts. The project builds upon the foundations laid by early cycling video games and incorporates the latest developments in graphics, physics simulation, and user interfaces to deliver a state-of-the-art simulation experience.

1.3 Applications:

The 3D Bicycle Simulation project holds various applications in different domains. Here are some of the key applications of the project:

- Training and Skill Development
- Physical Therapy and Rehabilitation
- Research and Analysis
- Virtual Competitions and Events
- Cycling Safety and Education
- Product Testing and Design
- Entertainment and Gaming

1.3.1 Training and Skill Development:

The simulation can serve as a valuable training tool for cyclists of all skill levels. Users can practice and improve their cycling techniques in a controlled and customizable virtual environment. The simulation can simulate different terrains, weather conditions, and riding scenarios, allowing cyclists to hone their skills, test their limits, and explore new strategies.

1.3.2 Physical Therapy and Rehabilitation:

The 3D Bicycle Simulation project can be used in the field of physical therapy to aid in the recovery and rehabilitation of individuals with mobility impairments. It provides a safe and controlled environment where patients can engage in cycling exercises to improve muscle strength, joint flexibility, and overall fitness. The simulation can be tailored to specific rehabilitation programs, allowing therapists to monitor progress and adjust parameters as needed.

1.3.3 Research and Analysis:

The project has applications in research and analysis related to cycling and human movement. Researchers can use the simulation to study various aspects of cycling performance, biomechanics, and physiological responses. It enables the collection of data such as speed, distance, power output, and heart rate, which can be analyzed to gain insights into cycling mechanics, energy expenditure, and training adaptations.

1.3.4 Virtual Competitions and Events:

The 3D Bicycle Simulation project can host virtual cycling competitions, races, and events. Users can participate in virtual races against other simulated cyclists or compete against their own best times. These virtual competitions can provide an entertaining and competitive experience, allowing participants from different locations to challenge each other and compare their performances.

1.3.5 Cycling Safety and Education:

The simulation can be utilized as an educational tool to promote cycling safety and awareness. Users can learn about traffic rules, safe cycling practices, and potential hazards through interactive tutorials and simulated scenarios. The project can help educate cyclists on defensive riding techniques, proper signaling, and decision-making in challenging situations, fostering a culture of safe cycling.

1.3.6 Product Testing and Design:

Bicycle manufacturers and designers can utilize the 3D Bicycle Simulation project for product testing and design iterations. The simulation allows them to evaluate the performance and aerodynamics of different bicycle models, components, and configurations. By virtually testing prototypes and modifications, manufacturers can optimize their designs before physical production, saving time and resources.

1.3.7 Entertainment and Gaming:

The project has a recreational aspect, providing an immersive and enjoyable experience for cycling enthusiasts. Users can explore virtual landscapes, embark on virtual cycling adventures, and engage in gamified challenges within the simulation. This entertainment aspect appeals to both casual riders seeking entertainment and avid cyclists looking for a virtual cycling experience during unfavorable weather conditions or other limitations.

These applications demonstrate the versatility and utility of the 3D Bicycle Simulation project across various domains, from training and research to rehabilitation and entertainment. By leveraging the capabilities of computer graphics, physics simulation, and interactive interfaces, the project offers a valuable tool for cyclists, therapists, researchers, and cycling enthusiasts alike.

Chapter 2

OpenGL

OpenGL (Open Graphics Library) is a standard specification defining a cross-language, cross-platform API for writing applications that produce 2D and 3D computer graphics. The interface consists of over 250 different function calls which can be used to draw complex three-dimensional scenes from simple primitives. OpenGL was developed by Silicon Graphics Inc. (SGI) in 1992 and is widely used in CAD, virtual reality, scientific visualization, information visualization, and flight simulation. OpenGL provides a set of commands to render a three-dimensional scene. That means you provide the data in an OpenGL-useable form and OpenGL will show this data on the screen (render it). It is developed by many companies and it is free to use. You can develop OpenGL applications without licensing. OpenGL is a hardware- and system-independent interface. An OpenGL-application will work on every platform, as long as there is an installed implementation. Because it is system independent, there are no functions to create windows etc., but there are helper functions for each platform. A very useful thing is GLUT.

2.1 OpenGL Libraries

Computer Graphics are created using OpenGL, which became a widely accepted standard software system for developing graphics applications. As a software interface for graphics hardware, OpenGL's main purpose is to render two- and three-dimensional objects into a frame buffer. These objects are described as sequences of vertices (which define geometric objects) or pixels (which define images). OpenGL performs several processing steps on this data to convert it to pixels to form the final desired image in the frame buffer. OpenGL stands for 'open graphics library' graphics library is a collection of API's (Applications Programming Interface). Graphics library functions are:

- GL library (OpenGL in Windows) – Main functions for Windows.

- GLU (OpenGL utility library) - Creating and viewing objects.
- GLUT (OpenGL utility toolkit)- Functions that help in creating an interface of windows
- freeglut (OpenGL utility toolkit) - Functions to help for creating on-screen text

OpenGL draws primitives—points, line segments, or polygons—subject to several selectable modes. You can control modes independently of each other; that is, setting one mode doesn't affect whether other modes are set (although many modes may interact to determine what eventually ends up in the frame buffer). Primitives are specified, modes are set, and other OpenGL operations are described by issuing commands in the form of function calls. These libraries are included in the application program using preprocessor directives OpenGL User Interface Library (GLUI) is a C++ user interface library based on the OpenGL Utility Toolkit (GLUT) which provides controls such as buttons, checkboxes, radio buttons, and spinners to OpenGL applications. It is window and operating system independent, relying on GLUT to handle all system-dependent issues, such as window and mouse management. The OpenGL Utility Library (GLU) is a computer graphics library. It consists of a number of functions that use the base OpenGL library to provide higher-level drawing routines from the more primitive routines that OpenGL provides. It is usually distributed with the base OpenGL package.

2.2 OpenGL Contributions

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

2.3 Limitations

- OpenGL is case sensitive.
- Line Color, Filled Faces and Fill Color not supported.
- Shadow plane is not supported.

Chapter 3

Resource Requirements

The resource requirements for a 3D Bicycle Simulation project can vary depending on the scale and complexity of the simulation. Here are some of the key resources typically needed for such a project:

3.1 Hardware Requirements

- High-performance computer
- Virtual Reality (VR) equipment (optional)

3.1.1 High-performance computer:

A powerful computer system with sufficient processing power, memory, and graphics capabilities is essential for developing and running a 3D Bicycle Simulation. This includes a fast processor, ample RAM, a dedicated graphics card, and sufficient storage capacity.

3.1.2 Virtual Reality (VR) equipment (optional):

If the project incorporates virtual reality, VR headsets, motion controllers, and tracking systems may be required to provide an immersive experience.

3.2 Software Requirements

- Game Engine
- 3D Modeling and Animation Software
- Physics Simulation Libraries

3.2.1 Game Engine:

A robust game engine is typically used for developing the 3D Bicycle Simulation. Popular game engines such as Unity or Unreal Engine offer tools and functionalities for graphics rendering, physics simulation, user interfaces, and scripting.

3.2.2 3D Modeling and Animation Software:

Software tools like Blender, 3ds Max, or Maya are utilized for creating and editing 3D bicycle models, terrains, and other visual assets.

3.2.3 Physics Simulation Libraries:

Libraries such as PhysX or Bullet Physics can be integrated into the simulation to accurately simulate bicycle dynamics, collisions, and interactions with the environment.

3.3 Content Creation

- 3D Models and Textures
- Sound Effects and Music

3.3.1 3D Models and Textures:

Creating or acquiring high-quality 3D models and textures for bicycles, terrains, objects, and other visual elements is essential. This may involve 3D modeling, texturing, and UV mapping.

Chapter 4

System Design

This encompasses two key aspects: the inbuilt functions and the user-defined functions. The inbuilt functions refer to the pre-existing functions provided by the OpenGL library, which offer a range of functionalities for rendering graphics and interacting with the graphics pipeline. These functions serve as building blocks for creating the visual elements and effects in the bicycle simulation. On the other hand, the user-defined functions are custom functions developed specifically for this project. They encapsulate specific tasks such as drawing the bicycle frame, gears, pedals, and tires, as well as handling user input for controlling the bicycle's movement. Together, these inbuilt and user-defined functions form the core foundation of the system design, enabling the creation of a dynamic and interactive 3D bicycle simulation.

4.1 Inbuilt Functions

OpenGL is a powerful graphics library widely used for rendering 2D and 3D graphics in computer graphics applications. It provides a set of inbuilt functions that facilitate the creation and manipulation of graphical elements. These functions, collectively known as the OpenGL library, are accessible through the OpenGL API (Application Programming Interface) and can be used in C++ programs.

4.1.1 Usage of GL/glut.h and GL/freeglut.h:

These functions are used for tasks such as initializing the OpenGL context, handling window and input events, and performing geometric transformations and rendering operations

4.1.2 Some of the Built-in functions used are:

- `glPushMatrix()` - Pushes the current matrix stack down by one and duplicates the top matrix.

- `glPopMatrix()` - Pops the current matrix stack, discarding the top matrix.
- `glTranslatef(float x, float y, float z)` - Translates the current matrix by the specified offsets in the x, y, and z directions.
- `glRotatef(float angle, float x, float y, float z)` - Applies a rotation to the current matrix by the specified angle around the axis defined by the x, y, and z parameters.
- `glScalef(float x, float y, float z)` - Scales the current matrix by the specified scaling factors in the x, y, and z directions.
- `gluNewQuadric()` - Creates a new quadratic object for rendering quadrics (e.g., cylinders, spheres).
- `gluCylinder(GLUquadric* quad, double base, double top, double height, int slices, int stacks)` - Draws a cylinder with the specified base radius, top radius, height, number of slices, and number of stacks using the given quadric object.
- `glBegin(GLenum mode)` - Specifies the beginning of a primitive or group of primitives to be drawn, with the specified drawing mode (e.g., points, lines, triangles).
- `glEnd()` - Specifies the end of the primitive or group of primitives.
- `glVertex3f(float x, float y, float z)` - Specifies a vertex with the specified coordinates in 3D space.
- `glColor3f(float red, float green, float blue)` - Sets the current color for subsequent vertex specifications using the specified RGB values.
- `glNormal3f(float nx, float ny, float nz)` - Specifies a normal vector for subsequent vertex specifications using the specified components.
- `glShadeModel(GLenum mode)` - Specifies the shading model to be used for the objects, with the specified mode (e.g., smooth shading, flat shading).

4.2 User Defined Functions:

In addition to the inbuilt functions provided by the OpenGL library, developers can also define their own functions to encapsulate specific operations or to enhance code modularity and reusability. User-defined functions in OpenGL can be written in C++ and can be called within the main rendering loop or other relevant parts of the program.

- `ZCylinder(GLfloat radius, GLfloat length)`: This function is used to draw a cylinder along the z-axis with the specified radius and length.
- `XCylinder(GLfloat radius, GLfloat length)`: This function is used to draw a cylinder along the x-axis by calling the `ZCylinder` function after rotating the coordinate system.
- `drawFrame()`: This function is responsible for drawing the frame of the bicycle, including the seat, rods, handle, and wheels.
- `gear(GLfloat innerradius, GLfloat outerradius, GLfloat width, GLint teeth, GLfloat toothdepth)`: This function is used to draw a gear with the specified parameters such as inner radius, outer radius, width, number of teeth, and tooth depth.
- `drawChain()`: This function is responsible for drawing the chain of the bicycle.
- `drawPedals()`: This function is used to draw the pedals of the bicycle.
- `drawTyre()`: This function is responsible for drawing the tires of the bicycle.
- `drawSeat()`: This function is used to draw the seat of the bicycle.
- `help()`: This function displays a help message providing information about the controls of the bicycle simulation.
- `init()`: This function is used for initializing the OpenGL settings and variables.
- `reset()`: This function resets the position, direction, speed, and steering of the bicycle to their initial values.
- `display()`: This function is the main display function that is called for rendering the scene.
- `special(int key, int x, int y)`: This function handles special keyboard key events, such as arrow keys, for controlling the bicycle.
- `idle()`: This function is called when the application is idle and updates the scene and triggers the `display` function.
- `updateScene()`: This function updates the position and orientation of the bicycle based on the speed and steering values.

Chapter 5

Implementation

```
#include<windows.h>
#include<GL/glut.h>
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<iostream>
#include<GL/freeglut.h>
using namespace std;
#define  PI          3.14159
#define WIN_WIDTH   600
#define WIN_HEIGHT   600
#define CYCLE_LENGTH 3.3f
#define ROD_RADIUS   0.05f
#define NUM_SPOKES   20
#define SPOKE_ANGLE   18
#define RADIUS_WHEEL 1.0f
#define TUBE_WIDTH   0.08f
#define RIGHT_ROD   1.6f
#define RIGHT_ANGLE   48.0f
#define MIDDLE_ROD   1.7f
#define MIDDLE_ANGLE 106.0f
#define BACK_CONNECTOR 0.5f
#define LEFT_ANGLE   50.0f
#define WHEEL_OFFSET 0.11f
#define WHEEL_LEN   1.1f
```

```
#define TOP_LEN      1.5f
#define CRANK_ROD   0.7f
#define CRANK_RODS   1.12f
#define CRANK_ANGLE   8.0f
#define HANDLE_ROD   1.2f
#define FRONT_INCLINE 70.0f
#define HANDLE_LIMIT 70.0f
#define INC_STEERING 2.5f
#define INC_SPEED   0.005f
GLfloat pedalAngle, speed, steering;
GLfloat camx,camy,camz;
GLfloat anglex,angley,anglez;
int prevx,prevy;
GLenum Mouse;
GLfloat xpos,zpos,direction;

void ZCylinder(GLfloat radius,GLfloat length);
void XCylinder(GLfloat radius,GLfloat length);
void drawFrame(void);
void gear( GLfloat inner_radius, GLfloat outer_radius,
       GLfloat width,GLint teeth, GLfloat tooth_depth );
void drawChain(void);
void drawPedals(void);
void drawTyre(void);
void drawSeat(void);
void help(void);
void init(void);
void reset(void);
void display(void);
void special(int key,int x,int y);
void idle(void);
void updateScene(void);
void landmarks(void);
void keyboard(unsigned char key,int x,int y);
void reshape(int w,int h);
void glSetupFuncs(void);
```

```
GLfloat Abs(GLfloat);
GLfloat degrees(GLfloat);
GLfloat radians(GLfloat);
GLfloat angleSum(GLfloat, GLfloat);

void ZCylinder(GLfloat radius,GLfloat length){
   GLUquadricObj *cylinder;
   cylinder=gluNewQuadric();
   glPushMatrix();
      glTranslatef(0.0f,0.0f,0.0f);
      gluCylinder(cylinder,radius,radius,length,15,5);
   glPopMatrix();
}

void XCylinder(GLfloat radius,GLfloat length){
   glPushMatrix();
      glRotatef(90.0f,0.0f,1.0f,0.0f);
      ZCylinder(radius,length);
   glPopMatrix();
}

void updateScene(){
   GLfloat xDelta, zDelta;
   GLfloat rotation;
   GLfloat sin_steering, cos_steering;
   if (-INC_SPEED < speed && speed < INC_SPEED) return;
   if(speed < 0.0f)
        pedalAngle = speed = 0.0f;
   xDelta = speed*cos(radians(direction + steering));
   zDelta = speed*sin(radians(direction + steering));
   xpos += xDelta;
   zpos -= zDelta;
   pedalAngle = degrees(angleSum(radians(pedalAngle), speed/RADIUS_WHEEL));
   sin_steering = sin(radians(steering));
   cos_steering = cos(radians(steering));
   rotation = atan2(speed * sin_steering, CYCLE_LENGTH + speed * cos_steering);
```

```
   direction = degrees(angleSum(radians(direction),rotation));
}

GLfloat angleSum(GLfloat a, GLfloat b){
   a += b;
   if (a < 0) return a+2*PI;
   else if (a > 2*PI) return a-2*PI;
   else return a;
}

void drawFrame(){
   // glColor3f(red, green, blue);
   glColor3f(1.0f,0.0f,0.0f);
   glPushMatrix();
      glPushMatrix();
         glColor3f(0.0f,1.0f,0.0f);
         glPushMatrix();
            glTranslatef(0.0f,0.0f,0.06f);
            glRotatef(-2*pedalAngle,0.0f,0.0f,1.0f);
            gear(0.08f,0.3f,0.03f,30,0.03f);
         glPopMatrix();
         glColor3f(1.0f,0.0f,0.0f);
         glTranslatef(0.0f,0.0f,-0.2f);
         ZCylinder(0.08f,0.32f);
      glPopMatrix();
      glRotatef(RIGHT_ANGLE,0.0f,0.0f,1.0f);
      XCylinder(ROD_RADIUS,RIGHT_ROD);
      glRotatef(MIDDLE_ANGLE-RIGHT_ANGLE,0.0f,0.0f,1.0f);
      XCylinder(ROD_RADIUS,MIDDLE_ROD);
      glColor3f(1.0f,1.0f,0.0f);
      glTranslatef(MIDDLE_ROD,0.0f,0.0f);
      glRotatef(-MIDDLE_ANGLE,0.0f,0.0f,1.0f);
      glScalef(0.3f,ROD_RADIUS,0.25f);
      drawSeat();
      glColor3f(1.0f,0.0f,0.0f);
   glPopMatrix();
```

```
glPushMatrix();
   glRotatef(-180.0f,0.0f,1.0f,0.0f);
   XCylinder(ROD_RADIUS,BACK_CONNECTOR);
   glPushMatrix();
      glTranslatef(0.5f,0.0f,WHEEL_OFFSET);
      XCylinder(ROD_RADIUS,RADIUS_WHEEL+TUBE_WIDTH);
   glPopMatrix();
   glPushMatrix();
      glTranslatef(0.5f,0.0f,-WHEEL_OFFSET);
      XCylinder(ROD_RADIUS,RADIUS_WHEEL+TUBE_WIDTH);
   glPopMatrix();
glPopMatrix();
glPushMatrix();
   glTranslatef(-(BACK_CONNECTOR+RADIUS_WHEEL+TUBE_WIDTH),0.0f,0.0f);
   glPushMatrix();
      glRotatef(-2*pedalAngle,0.0f,0.0f,1.0f);
      drawTyre();
      glColor3f(0.0f,1.0f,0.0f);
         gear(0.03f,0.15f,0.03f,20,0.03f);
      glColor3f(1.0f,0.0f,0.0f);
   glPopMatrix();
   glRotatef(LEFT_ANGLE,0.0f,0.0f,1.0f);
   glPushMatrix();
      glTranslatef(0.0f,0.0f,-WHEEL_OFFSET);
      XCylinder(ROD_RADIUS,WHEEL_LEN);
   glPopMatrix();
   glPushMatrix();
      glTranslatef(0.0f,0.0f,WHEEL_OFFSET);
      XCylinder(ROD_RADIUS,WHEEL_LEN);
   glPopMatrix();
   glTranslatef(WHEEL_LEN,0.0f,0.0f);
   XCylinder(ROD_RADIUS,CRANK_ROD);
   glTranslatef(CRANK_ROD,0.0f,0.0f);
   glRotatef(-LEFT_ANGLE,0.0f,0.0f,1.0f);
   XCylinder(ROD_RADIUS,TOP_LEN);
   glTranslatef(TOP_LEN,0.0f,0.0f);
```

```
glRotatef(-FRONT_INCLINE,0.0f,0.0f,1.0f);
glPushMatrix();
   glTranslatef(-0.1f,0.0f,0.0f);
   XCylinder(ROD_RADIUS,0.45f);
glPopMatrix();
glPushMatrix();
   glRotatef(-steering,1.0f,0.0f,0.0f);
   glTranslatef(-0.3f,0.0f,0.0f);
   glPushMatrix();
      glRotatef(FRONT_INCLINE,0.0f,0.0f,1.0f);
      glPushMatrix();
         glTranslatef(0.0f,0.0f,-HANDLE_ROD/2);
         ZCylinder(ROD_RADIUS,HANDLE_ROD);
      glPopMatrix();
      glPushMatrix();
         glColor3f(1.0f,1.0f,0.0f);
         glTranslatef(0.0f,0.0f,-HANDLE_ROD/2);
         ZCylinder(0.07f,HANDLE_ROD/4);
         glTranslatef(0.0f,0.0f,HANDLE_ROD*3/4);
         ZCylinder(0.07f,HANDLE_ROD/4);
         glColor3f(1.0f,0.0f,0.0f);
      glPopMatrix();
   glPopMatrix();
   glPushMatrix();
      XCylinder(ROD_RADIUS,CRANK_ROD);
      glTranslatef(CRANK_ROD,0.0f,0.0f);
      glRotatef(CRANK_ANGLE,0.0f,0.0f,1.0f);
      glPushMatrix();
         glTranslatef(0.0f,0.0f,WHEEL_OFFSET);
         XCylinder(ROD_RADIUS,CRANK_RODS);
      glPopMatrix();
      glPushMatrix();
         glTranslatef(0.0f,0.0f,-WHEEL_OFFSET);
         XCylinder(ROD_RADIUS,CRANK_RODS);
      glPopMatrix();
      glTranslatef(CRANK_RODS,0.0f,0.0f);
```

```
            glRotatef(-2*pedalAngle,0.0f,0.0f,1.0f);
            drawTyre();
         glPopMatrix();
      glPopMatrix(); /*  End of the rotation of the handle effect */
   glPopMatrix();
}

void gear( GLfloat inner_radius, GLfloat outer_radius, GLfloat width, GLint teeth,
    GLfloat tooth_depth ){
   GLint i;
   GLfloat r0, r1, r2;
   GLfloat angle, da;
   GLfloat u, v, len;
   const double pi = 3.14159264;
   r0 = inner_radius;
   r1 = outer_radius - tooth_depth/2.0;
   r2 = outer_radius + tooth_depth/2.0;
   da = 2.0*pi / teeth / 4.0;
   glShadeModel( GL_FLAT );
   glNormal3f( 0.0, 0.0, 1.0 );
   glBegin( GL_QUAD_STRIP );
   for (i=0;i<=teeth;i++) {
      angle = i * 2.0*pi / teeth;
      glVertex3f( r0*cos(angle), r0*sin(angle), width*0.5 );
      glVertex3f( r1*cos(angle), r1*sin(angle), width*0.5 );
      glVertex3f( r0*cos(angle), r0*sin(angle), width*0.5 );
      glVertex3f( r1*cos(angle+3*da), r1*sin(angle+3*da), width*0.5 );
   }
   glEnd();
   glBegin( GL_QUADS );
   da = 2.0*pi / teeth / 4.0;
   for (i=0;i<teeth;i++) {
      angle = i * 2.0*pi / teeth;
      glVertex3f( r1*cos(angle),  r1*sin(angle),   width*0.5 );
      glVertex3f( r2*cos(angle+da), r2*sin(angle+da), width*0.5 );
      glVertex3f( r2*cos(angle+2*da), r2*sin(angle+2*da), width*0.5 );
```

```
   glVertex3f( r1*cos(angle+3*da), r1*sin(angle+3*da), width*0.5 );
}
glEnd();
glNormal3f( 0.0, 0.0, -1.0 );
glBegin( GL_QUAD_STRIP );
for (i=0;i<=teeth;i++) {
   angle = i * 2.0*pi / teeth;
   glVertex3f( r1*cos(angle), r1*sin(angle), -width*0.5 );
   glVertex3f( r0*cos(angle), r0*sin(angle), -width*0.5 );
   glVertex3f( r1*cos(angle+3*da), r1*sin(angle+3*da), -width*0.5 );
   glVertex3f( r0*cos(angle), r0*sin(angle), -width*0.5 );
}
glEnd();
glBegin( GL_QUADS );
da = 2.0*pi / teeth / 4.0;
for (i=0;i<teeth;i++) {
   angle = i * 2.0*pi / teeth;
   glVertex3f( r1*cos(angle+3*da), r1*sin(angle+3*da), -width*0.5 );
   glVertex3f( r2*cos(angle+2*da), r2*sin(angle+2*da), -width*0.5 );
   glVertex3f( r2*cos(angle+da), r2*sin(angle+da), -width*0.5 );
   glVertex3f( r1*cos(angle),  r1*sin(angle),  -width*0.5 );
}
glEnd();
glBegin( GL_QUAD_STRIP );
for (i=0;i<teeth;i++) {
   angle = i * 2.0*pi / teeth;
   glVertex3f( r1*cos(angle),  r1*sin(angle),   width*0.5 );
   glVertex3f( r1*cos(angle),  r1*sin(angle),  -width*0.5 );
   u = r2*cos(angle+da) - r1*cos(angle);
   v = r2*sin(angle+da) - r1*sin(angle);
   len = sqrt( u*u + v*v );
   u /= len;
   v /= len;
   glNormal3f( v, -u, 0.0 );
   glVertex3f( r2*cos(angle+da), r2*sin(angle+da), width*0.5 );
   glVertex3f( r2*cos(angle+da), r2*sin(angle+da), -width*0.5 );
```

```
      glNormal3f( cos(angle), sin(angle), 0.0 );
      glVertex3f( r2*cos(angle+2*da), r2*sin(angle+2*da), width*0.5 );
      glVertex3f( r2*cos(angle+2*da), r2*sin(angle+2*da), -width*0.5 );
      u = r1*cos(angle+3*da) - r2*cos(angle+2*da);
      v = r1*sin(angle+3*da) - r2*sin(angle+2*da);
      glNormal3f( v, -u, 0.0 );
      glVertex3f( r1*cos(angle+3*da), r1*sin(angle+3*da), width*0.5 );
      glVertex3f( r1*cos(angle+3*da), r1*sin(angle+3*da), -width*0.5 );
      glNormal3f( cos(angle), sin(angle), 0.0 );
   }
   glVertex3f( r1*cos(0.0), r1*sin(0.0), width*0.5 );
   glVertex3f( r1*cos(0.0), r1*sin(0.0), -width*0.5 );
   glEnd();
   glShadeModel( GL_SMOOTH );
   glBegin( GL_QUAD_STRIP );
   for (i=0;i<=teeth;i++) {
      angle = i * 2.0*pi / teeth;
      glNormal3f( -cos(angle), -sin(angle), 0.0 );
      glVertex3f( r0*cos(angle), r0*sin(angle), -width*0.5 );
      glVertex3f( r0*cos(angle), r0*sin(angle), width*0.5 );
   }
   glEnd();
}

void drawChain(){
   GLfloat depth;
   static int mode=0;
   glColor3f(0.0f,1.0f,0.0f);
   glEnable(GL_LINE_STIPPLE);
   mode=(mode+1)%2;
   if(mode==0 && speed>0)
      glLineStipple(1,0x1c47);
   else if(mode==1 && speed>0)
      glLineStipple(1,0x00FF);
   glBegin(GL_LINES);
   for(depth=0.06f;depth<=0.12f;depth+=0.01f){
```

```
      glVertex3f(-1.6f,0.15f,ROD_RADIUS);
      glVertex3f(0.0f,0.3f,depth);
      glVertex3f(-1.6f,-0.15f,ROD_RADIUS);
      glVertex3f(0.0f,-0.3f,depth);
   }
   glEnd();
   glDisable(GL_LINE_STIPPLE);
}

void drawSeat(){
   glBegin(GL_POLYGON);
       glVertex3f(-0.1f, 1.0f, -0.5f);
       glVertex3f( 1.0f, 1.0f, -0.3f);
       glVertex3f( 1.0f, 1.0f, 0.3f);
       glVertex3f(-0.1f, 1.0f, 0.5f);
       glVertex3f(-0.5f, 1.0f, 1.0f);
       glVertex3f(-1.0f, 1.0f, 1.0f);
       glVertex3f(-1.0f, 1.0f, -1.0f);
       glVertex3f(-0.5f, 1.0f, -1.0f);
   glEnd();
   glBegin(GL_POLYGON);
    glColor3f(0.4,0.0,0.2);
      glVertex3f(-0.1f, -1.0f, -0.5f);
      glVertex3f( 1.0f, -1.0f, -0.3f);
      glVertex3f( 1.0f, -1.0f, 0.3f);
      glVertex3f(-0.1f, -1.0f, 0.5f);
      glVertex3f(-0.5f, -1.0f, 1.0f);
      glVertex3f(-1.0f, -1.0f, 1.0f);
      glVertex3f(-1.0f, -1.0f, -1.0f);
      glVertex3f(-0.5f, -1.0f, -1.0f);
   glEnd();
   glBegin(GL_QUADS);
      glVertex3f(1.0f,1.0f,-0.3f);
      glVertex3f(1.0f,1.0f,0.3f);
      glVertex3f(1.0f,-1.0f,0.3f);
      glVertex3f(1.0f,-1.0f,-0.3f);
```

```
      glVertex3f(1.0f,1.0f,0.3f);
      glVertex3f(-0.1f,1.0f,0.5f);
      glVertex3f(-0.1f,-1.0f,0.5f);
      glVertex3f(1.0f,-1.0f,0.3f);
      glVertex3f(1.0f,1.0f,-0.3f);
      glVertex3f(-0.1f,1.0f,-0.5f);
      glVertex3f(-0.1f,-1.0f,-0.5f);
      glVertex3f(1.0f,-1.0f,-0.3f);
      glVertex3f(-0.1f,1.0f,0.5f);
      glVertex3f(-0.5f,1.0f,1.0f);
      glVertex3f(-0.5f,-1.0f,1.0f);
      glVertex3f(-0.1f,-1.0f,0.5f);
      glVertex3f(-0.1f,1.0f,-0.5f);
      glVertex3f(-0.5f,1.0f,-1.0f);
      glVertex3f(-0.5f,-1.0f,-1.0f);
      glVertex3f(-0.1f,-1.0f,-0.5f);
      glVertex3f(-0.5f,1.0f,1.0f);
      glVertex3f(-1.0f,1.0f,1.0f);
      glVertex3f(-1.0f,-1.0f,1.0f);
      glVertex3f(-0.5f,-1.0f,1.0f);
      glVertex3f(-0.5f,1.0f,-1.0f);
      glVertex3f(-1.0f,1.0f,-1.0f);
      glVertex3f(-1.0f,-1.0f,-1.0f);
      glVertex3f(-0.5f,-1.0f,-1.0f);
      glVertex3f(-1.0f,1.0f,1.0f);
      glVertex3f(-1.0f,1.0f,-1.0f);
      glVertex3f(-1.0f,-1.0f,-1.0f);
      glVertex3f(-1.0f,-1.0f,1.0f);
   glEnd();
}

void drawPedals(){
 //Right pedal
   glColor3f(0.0f,0.0f,1.0f);

   glPushMatrix();
```

```
    glTranslatef(0.0f,0.0f,0.105f);
    glRotatef(-pedalAngle,0.0f,0.0f,1.0f);
    glTranslatef(0.25f,0.0f,0.0f);

    glPushMatrix();
      glScalef(0.5f,0.1f,0.1f);
      glutSolidCube(1.0f);
    glPopMatrix();

//Right Leg put
    glPushMatrix();
      glTranslatef(0.25f,0.0f,0.15f);
      glRotatef(pedalAngle,0.0f,0.0f,1.0f);
      glScalef(0.2f,0.02f,0.3f);
      glutSolidCube(1.0f);
    glPopMatrix();
glPopMatrix();

//Left pedal
glPushMatrix();
    glTranslatef(0.0f,0.0f,-0.105f);
    glRotatef(180.0f-pedalAngle,0.0f,0.0f,1.0f);
    glTranslatef(0.25f,0.0f,0.0f);

    glPushMatrix();
      glScalef(0.5f,0.1f,0.1f);
      glutSolidCube(1.0f);
    glPopMatrix();

    //Left leg put pedal
    glPushMatrix();
      glTranslatef(0.25f,0.0f,-0.15f);
      glRotatef(pedalAngle-180.0f,0.0f,0.0f,1.0f);
      glScalef(0.2f,0.02f,0.3f);
      glutSolidCube(1.0f);
    glPopMatrix();
```

```
   glPopMatrix();
   glColor3f(1.0f,0.0f,0.0f);
}

void drawTyre(void){
   int i;
   glColor3f(1.0f,1.0f,1.0f);
   glutSolidTorus(0.06f,0.92f,4,30);
   glColor3f(1.0f,1.0f,0.5f);
   glPushMatrix();
      glTranslatef(0.0f,0.0f,-0.06f);
      ZCylinder(0.02f,0.12f);
   glPopMatrix();
   glutSolidTorus(0.02f,0.02f,3,20);
   glColor3f(1.0f,1.0f,1.0f);
   for(i=0;i<NUM_SPOKES;++i){
      glPushMatrix();
         glRotatef(i*SPOKE_ANGLE,0.0f,0.0f,1.0f);
         glBegin(GL_LINES);
            glVertex3f(0.0f,0.02f,0.0f);
            glVertex3f(0.0f,0.86f,0.0f);
         glEnd();
      glPopMatrix();
   }
   glColor3f(0.0f,0.0f,0.1f);
   glutSolidTorus(TUBE_WIDTH,RADIUS_WHEEL,10,30);
   glColor3f(1.0f,0.0f,0.0f);
}

void init(){
   GLfloat mat_specular[]={1.0,1.0,1.0,1.0};
   GLfloat mat_shininess[]={100.0};
   GLfloat light_directional[]={1.0,1.0,1.0,1.0};
   GLfloat light_positional[]={1.0,1.0,1.0,0.0};
   GLfloat light_diffuse[]={1.0,1.0,1.0};
```

```
   reset();
   glShadeModel(GL_SMOOTH);
   glLightfv(GL_LIGHT0,GL_POSITION,light_directional);
   glLightfv(GL_LIGHT0,GL_AMBIENT,light_diffuse);
   glLightfv(GL_LIGHT0,GL_DIFFUSE,light_diffuse);
   glMaterialfv(GL_FRONT,GL_SHININESS,mat_shininess);
   glMaterialfv(GL_FRONT,GL_SPECULAR,mat_specular);
   glColorMaterial(GL_FRONT,GL_DIFFUSE);
   glEnable(GL_LIGHTING);
   glEnable(GL_LIGHT0);
   glEnable(GL_COLOR_MATERIAL);
   glEnable(GL_DEPTH_TEST);
}

void landmarks(void){
      GLfloat i;
      glColor3f(-0.6f,-0.6f,-0.5f);
      glBegin(GL_QUADS);
      for(i=-100.0f ; i<100.0f ; i += 1.0f){
         glVertex3f(-86.0f,-RADIUS_WHEEL,i);
         glVertex3f( 100.0f,-RADIUS_WHEEL,i);
         glVertex3f(i,-RADIUS_WHEEL,-100.0f);
         glVertex3f(i,-RADIUS_WHEEL,100.0f);
      }
      glEnd();
}

void displayKeyBindings(void){
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glColor3f(0,1,0);
   glRasterPos2f(-0.5, 2.5);
   glutBitmapString(GLUT_BITMAP_HELVETICA_18,(const unsigned char*)"3D Bicycle
       Simulation");
   glRasterPos2f(-5, 2.3);
   glutBitmapString(GLUT_BITMAP_HELVETICA_18,(const unsigned char*)"Key Bindings:
       (Enable keyboard from menu)");
```

```
    glRasterPos2f(-5, 2.1);
    glutBitmapString(GLUT_BITMAP_HELVETICA_18,(const unsigned char*)"Right click -
        Menu");
    glRasterPos2f(-5, 1.9);
    glutBitmapString(GLUT_BITMAP_HELVETICA_18,(const unsigned char*)"W, A, S, D -
        Move Bicycle");
    glRasterPos2f(-5, 1.7);
    glutBitmapString(GLUT_BITMAP_HELVETICA_18,(const unsigned char*)"Esc - Exit");
}

void display(void){
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    displayKeyBindings();

    glEnable(GL_NORMALIZE);
    glPushMatrix();
       glRotatef(angley,1.0f,0.0f,0.0f);
       glRotatef(anglex,0.0f,1.0f,0.0f);
       glRotatef(anglez,0.0f,0.0f,1.0f);
      landmarks();
       glPushMatrix();
          glTranslatef(xpos,0.0f,zpos);
          glRotatef(direction,0.0f,1.0f,0.0f);
          drawFrame();
          drawChain();
          drawPedals();
       glPopMatrix();
    glPopMatrix();
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(camx,camy,camz, camx,0.0,0.0, 0.0,1.0,0.0);
    glutSwapBuffers();
}

void special(int key,int x,int y){
```

```
   switch(key){
      case GLUT_KEY_UP:
         camz -= 0.1f;
         break;
      case GLUT_KEY_DOWN:
         camz += 0.1f;
         break;
      case GLUT_KEY_LEFT:
         camx -= 0.1f;
         break;
      case GLUT_KEY_RIGHT:
         camx += 0.1f;
         break;
   }
   glutPostRedisplay();
}

GLfloat Abs(GLfloat a){
   if(a < 0.0f)
      return -a;
   else
      return a;
}

GLfloat degrees(GLfloat a){
   return a*180.0f/PI;
}

GLfloat radians(GLfloat a){
   return a*PI/180.0f;
}

void idle(void){
   updateScene();
   glutPostRedisplay();
}
```

```
void reset(){
   anglex=angley=anglez=0.0f;
   pedalAngle=steering=0.0f;
   Mouse=GLUT_UP;
   pedalAngle=speed=steering=0.0f;
   camx=camy=0.0f;
   camz=5.0f;
   xpos=zpos=0.0f;
   direction=0.0f;
}

void keyboard(unsigned char key,int x,int y){
   GLfloat r=0.0f;

   switch(key){
      case 'r':
      case 'R':
         reset();
         break;
      case 'A':
      case 'a':
         if(steering < HANDLE_LIMIT)
               steering += INC_STEERING;
         break;
      case 'D':
      case 'd':
         if(steering > -HANDLE_LIMIT)
            steering -= INC_STEERING;
         break;
      case 'W':
      case 'w':
         speed += INC_SPEED;
         break;
      case 'S':
```

```
      case 's':
         speed -= INC_SPEED;
         break;
      case 27:
         exit(1);
   }
   pedalAngle += speed;
   if(speed < 0.0f)
      speed = 0.0f;
   if(pedalAngle < 0.0f)
      pedalAngle = 0.0f;
   if(pedalAngle >= 360.0f)
      pedalAngle -= 360.0f;
   glutPostRedisplay();
}


void reshape(int w,int h){
   glViewport(0,0,(GLsizei)w,(GLsizei)h);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluPerspective(60.0,(GLfloat)w/(GLfloat)h,0.1,100.0);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
   gluLookAt(camx,camy,camz, 0.0,0.0,0.0, 0.0,1.0,0.0);
}

void glSetupFuncs(void){
   glutDisplayFunc(display);
   glutReshapeFunc(reshape);
   glutIdleFunc(idle);
   glutSetCursor(GLUT_CURSOR_CROSSHAIR);
}

void menu(int value){
   switch(value){
```

```
      case 1:
         glutSetCursor(GLUT_CURSOR_UP_DOWN);
         glutKeyboardFunc(keyboard);
         glutChangeToMenuEntry(1,"Disable Bicycle Movement",5);

         break;
      case 2:
         glutSetCursor(GLUT_CURSOR_LEFT_RIGHT);
         glutSpecialFunc(special);
         glutChangeToMenuEntry(1,"Disable Camera Movement",6);
         break;
      case 3:
         reset();
         glutChangeToMenuEntry(1,"Enable Bicycle Movement",1);
         glutChangeToMenuEntry(2,"Enable Camera Movement",2);
         break;
      case 4:
         exit(1);
      case 5:
         glutSetCursor(GLUT_CURSOR_LEFT_RIGHT);
         glutKeyboardFunc(NULL);
         glutChangeToMenuEntry(1,"Enable Bicycle Movement",1);
         break;
      case 6:
         glutSetCursor(GLUT_CURSOR_UP_DOWN);
         glutSpecialFunc(NULL);
         glutChangeToMenuEntry(1,"Enable Camera Movement",2);
         break;
   }
}

int main(int argc,char *argv[]){
   glutInit(&argc,argv);
   glutInitWindowSize(980,680);
   glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
   glutInitWindowPosition(100,100);
```

```
    glutInitWindowSize(WIN_WIDTH,WIN_HEIGHT);
    glutCreateWindow("InfiniteN00b BiCycle");

    glutCreateMenu(menu);
    glutAddMenuEntry("Enable Bicycle Movement", 1);
    glutAddMenuEntry("Enable Camera Movement", 2);
    glutAddMenuEntry("Reset Position", 3);
    glutAddMenuEntry("Exit From Simulation", 4);
    glutAttachMenu(GLUT_RIGHT_BUTTON);

    glSetupFuncs();
    init();
    glutMainLoop();
}
```

Chapter 6

Testing

In unit testing, the program modules that make up the system are tested individually. Unit testing focuses to locate errors in the working modules that are independent to each other. This enables to detect errors in coding and the logic within the module alone. This testing is also used to ensure the integrity of the data stored. The various routines were checked by passing the inputs and the corresponding output is tested.Table 6.1 gives details of validation. Test cases used in the project as follows:

Table 6.1: Test Case Validation

Test Case No.	**Metric**	**Description**	**Observation**
1.	Keyboard Function	Moves the Bicycle in the direction of key press(W, A, S ,D) for movements	Results are obtained as expected.
2	Display Function	Outer Frame of the Bicycle is drawn and the view scale changes according to movement	Results are obtained as expected.
3	Animation	Bicycle Handle movements are bound to keyboard keys and Wheels rotation on the direction of movement	Results obtained as expected

Chapter 7

7.1 Results & Snapshots

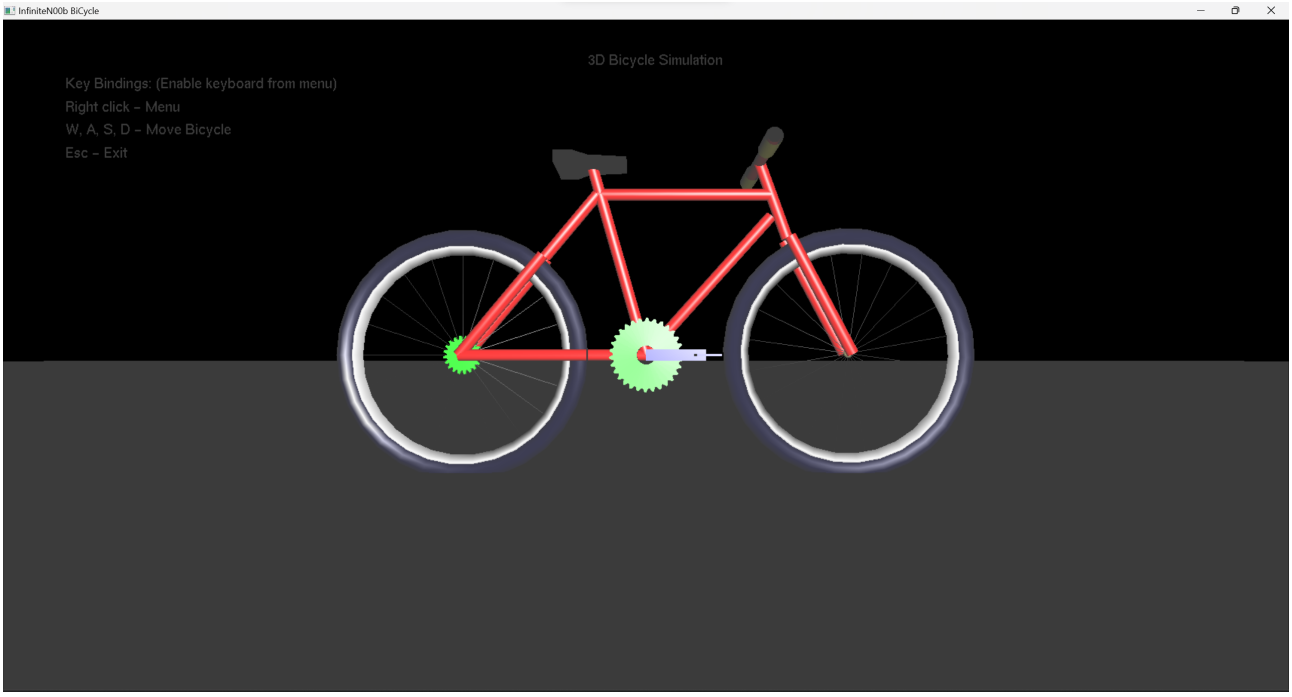

Figure 7.1: Main Screen with Key information

Figure 7.1 describes:
The static Entry screen which contains on-screen information on the key bindings to be used to move the bicycle and key to access the menu of the window

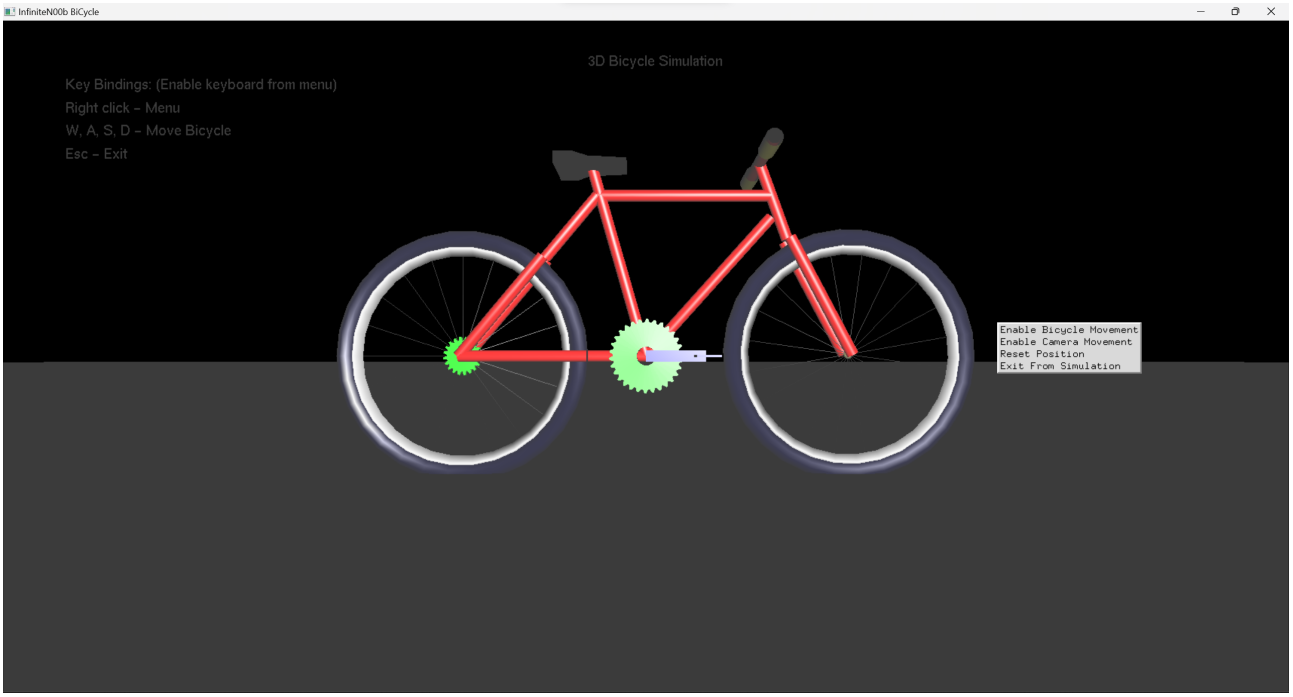

Figure 7.2: Menu Options

Figure 7.2 describes:
Display four menu options,
Enable Bicycle Movement, Enable Camera Movement, Reset Position, Exit from simulation,

Figure 7.3: Bicycle movement enabled

Figure 7.3 describes:
Bicycle movement has been enabled, and menu options are changed to,
Disable Bicycle Movement, Enable Camera Movement, Reset Position, Exit from simulation,

Figure 7.4: Enabling Camera Movement

Figure 7.4 describes:
Bicycle movement and Camera movement has been enabled, and menu options are changed to, Disable Bicycle Movement, Disable Camera Movement, Reset Position, Exit from simulation,

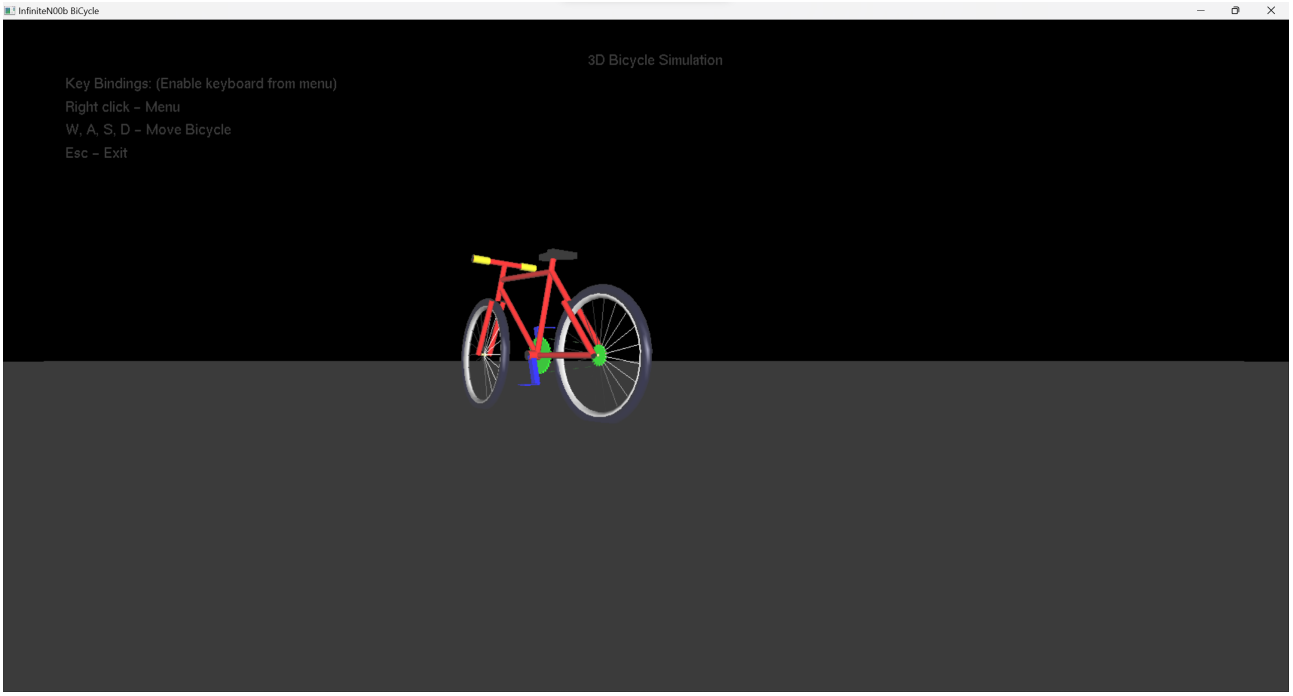

Figure 7.5: Bicycle movement in the key direction

Figure 7.5 describes:
The bicycle has been moved along a direction, and menu options are,
Disable Bicycle Movement, Disable Camera Movement, Reset Position, Exit from simulation,

Figure 7.6: Camera scale changed

Figure 7.6 describes:
The bicycle moved and the camera scale was increased,
Arrow keys are to be used to change the camera views.

Chapter 8

Conclusion & Future Enhancements

8.1 Conclusion

The 3D Bicycle Simulation project has achieved its objective of creating a realistic and immersive virtual environment for bicycle riding. By utilizing computer graphics techniques and physics simulation, the project offers users an engaging and valuable experience. With applications ranging from training and research to rehabilitation and entertainment, the simulation has proven to be a versatile tool. Leveraging the power of OpenGL, the project demonstrates the capabilities of graphics rendering, showcasing visually appealing 3D models, lighting effects, and camera control.

In conclusion, the 3D Bicycle Simulation project successfully delivers an immersive virtual bicycle riding experience. It exemplifies the potential of computer graphics and simulation technologies, paving the way for future advancements and opportunities in the field of virtual bicycle simulations.

8.2 Future Enhancements

While the 3D Bicycle Simulation project has achieved its initial objectives, there are several potential areas for future enhancements and improvements. These enhancements can further enhance the realism, interactivity, and overall user experience of the simulation. Here are some possible avenues for future development:

- Advanced Physics Simulation
- Expanded Environments and Terrains
- Multiplayer Functionality

- Real-time Weather Effects
- Customization Options
- VR (Virtual Reality) Integration
- Training and Analytics

8.2.1 Advanced Physics Simulation:

Enhancing the physics simulation component can result in more realistic and accurate bicycle dynamics. This could involve incorporating advanced physics models, such as tire deformation, suspension systems, and more intricate interactions with the terrain and other objects.

8.2.2 Expanded Environments and Terrains:

Adding a variety of environments and terrains can provide users with different riding experiences. Expanding the simulation to include diverse landscapes, such as mountains, urban areas, or off-road trails, can create a more immersive and engaging experience.

8.2.3 Multiplayer Functionality:

Implementing multiplayer functionality would allow users to interact and ride alongside other virtual cyclists. This could include features such as cooperative or competitive modes, online leaderboards, and the ability to challenge friends or other users.

8.2.4 Real-time Weather Effects:

Introducing real-time weather effects, such as rain, snow, or fog, can enhance the realism of the simulation. Weather conditions could affect the bicycle's handling, visibility, and overall riding experience, adding another layer of immersion.

8.2.5 Customization Options:

Providing users with the ability to customize their bicycles and adjust various parameters can add personalization and enhance the engagement. This could include options for choosing different bicycle models, adjusting components (such as frame, wheels, or gears), and selecting colors or decals.

8.2.6 VR (Virtual Reality) Integration:

ntegrating virtual reality technology into the simulation can greatly enhance the immersion and realism. By allowing users to wear VR headsets and interact with the virtual environment using motion controllers, they can experience a more immersive and hands-on riding experience.

8.2.7 Training and Analytics:

Implementing training features and analytics can transform the simulation into a valuable tool for cyclists. This could include features like performance tracking, analyzing riding technique, providing feedback and suggestions for improvement, and offering training programs or challenges.

References

[1] Edward Angel, *"Interactive Computer Graphics A Top-Down Approach With OpenGL"* 5th Edition, Addison-Wesley, 2008.

[2] F.S. Hill,*"Computer Graphics Using OpenGL"*, 2nd Edition, Pearson Education, 2001.

[3] @online OpenGL Official ,`https://www.opengl.org/`

[4] @online OpenGL Overview , `https://https://www.khronos.org/opengl/`

[5] @online OpenGL Overview , `https://https://stackoverflow.com/`