PROJECT REPORT

On

"3D BICYCLE"

Submitted in partial fulfillment of

Computer Graphics and Laboratory with mini project(17CSL68)

Sixth Semester of the Degree of Bachelor of Engineering

in

Department of Computer Science and Engineering

Visvesvaraya Technological University, Belagavi

during the year 2019-20

Carried out by

PRASAD B HULYAL

1SB17CS062

VIJESH R NAIK

1SB17CS094

Under the guidance of

REMYA SIVAN

Assistant Professor

Dept. of Computer Science and Engg.

Sri Sairam College of Engineering

Sai Leo Nagar, Guddanahalli Post,
Anekal, Bengaluru – 562106

Sri SAIRAM COLLEGE OF ENGINEERING
Anekal, Bengaluru

Accredited by NAAC
ISO 9001:2015 Certified Institution
Approved by AICTE, New Delhi
Affiliated to Visvesvaraya Technological University
www.sairamce.edu.in

Department of Computer Science and Engineering

CERTIFICATE

Certified that the project entitled "**3D BICYCLE "**is a bonafide work carried out by **PRASAD B HULYAL** **(1SB17CS062)** **VIJESH R NAIK** **(1SB17CS094)**

In partial fulfillment of sixth semester of the degree of bachelor of engineering in **Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year **2019-2020**. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report deposited in the department library. The project has been approved as it satisfies the academic requirements in respect of project work prescribed for the Bachelor of Engineering Degree.

Signature of the Guide	Signature of the HOD
Remya Sivan	Dr.G.Manjula
Asst.Prof,CSE Dept	HOD,CSE Dept
SSCE	SSCE

External Viva

Name of the Examiners: 1._____

2._____

ABSTRACT

3D computer graphics or three-dimensional computer graphics (in contrest with 2D computer graphics) are graphics that use a three-dimensional representation of geometric data that is stored in the computer

The Project is on "**3D-BICYCLE**" computer graphics using open GL Function.
It is used interactive program where in the user can view the Required display by making use of input device like keyboard and mouse. This project mainly consist of a bicycle .The bicycle can be viewed in any direction .

The movement of the bicycle can controled by using keyboard. For viewing , we make use of mouse. The bicycle is ridden on a polygonal surface.
Which looks like a gaint rectangular mesh. Lighting has been incorporated on only one side-Viewer side. The bicycle can move in all direction.
it can be rotated and it can resized at any size.

ACKNOWLEDGEMENT

It has been an honor and privilege to do my project on **3D BICYCLE.**I take this opportunity to convey my sincere thanks and regards to Our Chairman and Chief Executive Officer **Sri. Sai Prakash Leo Muthu**, and **Dr. Arun Kumar R**, Management Representative for offered me a chance to study in this institute and everyone who has helped me in successful completion of this seminar.

I take immense pleasure in expressing my sincere gratitude to **Dr. B SHADAKSHARAPPA, Principal, Sri Sairam College of Engineering, Bengaluru** for giving me a constant encouragement and support to achieve my goal.

My sincere thanks to **Dr.G.Manjula, Professor & Head, Dept. of Computer Science and Engineering** for permitting us to undertake the project work and for her invaluable guidance.

I would like to thank my guide **Remya Sivan,Asst. Prof**, **Dept. of Computer Science and Engineering,** without her constant motivation and guidance at every stage of this project, this work could not have been materialized.

I perceive as this opportunity as a big milestone in my career development. I will strive to use gained skills and knowledge in the best possible way, and will continue to work on their improvement, in order to attain desired career objectives. Hope to continue cooperation with all of you in the future.

CONTENTS

List of Figures

List of Snapshots

1:INTRODUCTION

1.1 COMPUTER GRAPHICS

In this era of computing, computer graphics has become one of the most powerful and interesting fact of computing. It all started with display of data on hardcopy and CRT screen. Now computer graphics is about creation, retrieval, manipulation of models and images.

Graphics today is used in many different areas. Graphics provides one of the most natural means of communicating within a computer, since our highly developed 2D and 3D pattern-recognition abilities allow us to perceive and process pictorial data rapidly and effectively. Interactive computer graphics is the most important means of producing pictures since the invention of photography and television. It has the added advantage that, with the computer, we can make pictures not only of concrete real world objects but also of abstract, synthetic objects, such as mathematical surfaces and of data that have no inherent geometry, such as survey results.

OpenGL is an application program interface (API) offering various functions to implement primitives, models and images. This offers functions to create and manipulate render lighting, coloring, viewing the models. OpenGL offers different coordinate system and frames. OpenGL offers translation, rotation and scaling of objects.

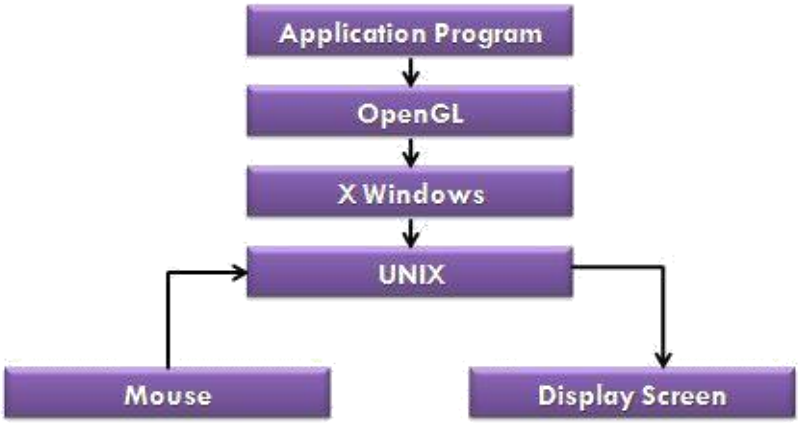


FIG:1.1 OVERVIEW OF COMPUTER GRAPHICS

1.2 OpenGL CONCEPT

1.2.1 INTERFACE

OpenGL is an application program interface (API) offering various functions to implement primitives, models and images. This offers functions to create and manipulate render lighting, coloring, viewing the models. OpenGL offers different coordinate system and frames. OpenGL offers translation, rotation and scaling of objects. Functions in the main GL library have names that begins with *gl* and are stored in a library usually referred to as GL. The second is the **OpenGL Utility Library(GLU)**. The library uses only GL functions but contains code for creating common objects and simplifying viewing.

All functions in GLU can be created from the core GL library but application programmers prefer not to write the code repeatedly. The GLU library is available in all OpenGL implementations; functions in the GLU library begin with the letter *glu*. Rather than using a different library for each system we use a readily available library called OpenGL Utility Toolkit(GLUT),which provides the minimum functionality that should be expected in any modern windowing system.

1.2.2 OVERVIEW

- OpenGL(Open Graphics Library) is the interface between a graphics program and graphics hardware. *It is streamlined*. In other words, it provides low-level functionality. For example, all objects are built from points, lines and convex polygons. Higher level objects like cubes are implemented as six four-sided polygons.
- OpenGL supports features like 3-dimensions, lighting, anti-aliasing, shadows, textures, depth effects, etc.
- It is system-independent. It does not assume anything about hardware or operating system and is only concerned with efficiently rendering mathematically described scenes. As a result, it does not provide any windowing capabilities.
- It is a state machine. At any moment during the execution of a program there is a current model transformation.
- It is a rendering pipeline. The rendering pipeline consists of the following steps:
    * Defines objects mathematically.
    * Arranges objects in space relative to a viewpoint.
    * Calculates the color of the objects.

1.2.3 OPENGL ARCHITECTURE:

1) Pipeline Architectures

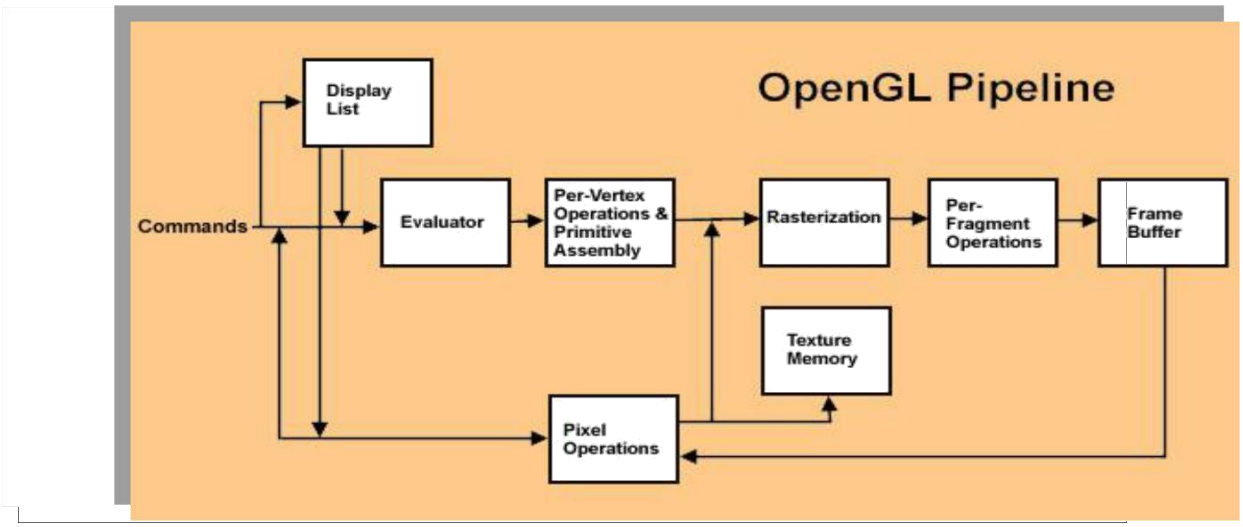


FIG:1.2.3.1 OPENGL PIPELINE ARCHITECTURE

2) OpenGL Engine And Drivers

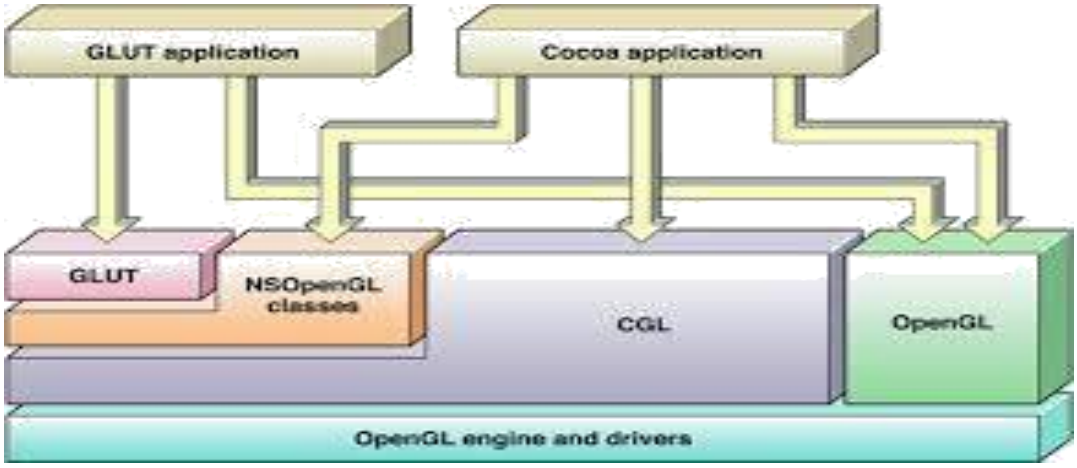


FIG:1.2.3.2 OPENGL ENGINE AND DRIVERS

3) Application Development-API's

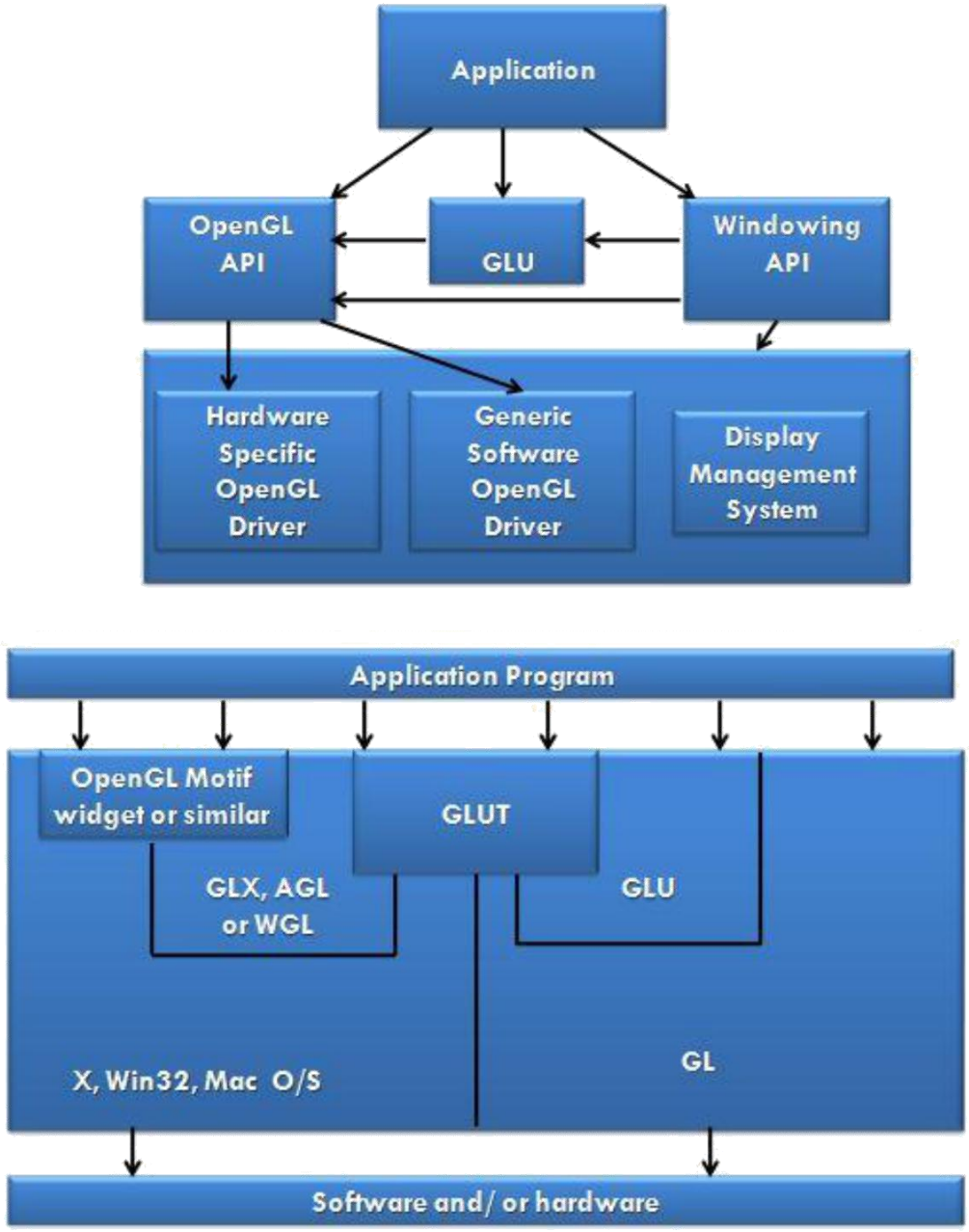


FIG:1.2.3.3 APPLICATIONS DEVELOPMENT(API'S)

The above diagram illustrates the relationship of the various libraries and window system components.

Generally, applications which require more user interface support will use a library designed to support those types of features (i.e. buttons, menu and scroll bars, etc.) such as Motif or the Win32 API. Prototype applications, or ones which do not require all the bells and whistles of a full GUI, may choose to use GLUT instead because of its simplified programming model and window system independence.

Display Lists:

All data, whether it describes geometry or pixels, can be saved in a *display list* for current or later use. (1 alternative to retaining data in a displaylist is processing the data immediately - also known as *immediate mode*.) 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. Evaluators provide a method to derive the vertices used to represent the surface from the control points. The method is a polynomial mapping, which can produce surface normal, texture coordinates, colors, and spatial coordinate values from the control points.

Per-Vertex Operations

For vertex data, next is the "per-vertex operations" stage, which converts. The vertices into primitives. Some vertex data (for example, spatial coordinates) 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.

If advanced features are enabled, this stage is even busier. If texturing is used, texture coordinates may be generated and transformed here. If lighting is enabled, the lighting calculations are performed using the transformed vertex, surface normal, light source position, material properties, and other lighting information to produce a color value.

Primitive Assembly

Clipping, a major part of primitive assembly, is the elimination of portions of geometry which fall outside a half-space, defined by a plane. Point clipping simply passes or rejects vertices; line or polygon clipping can add additional vertices depending upon how the line or polygon is clipped.

In some cases, this is followed by perspective division, which makes distant geometric objects appear smaller than closer objects. Then view port

and depth (z coordinate) operations are applied. If culling is enabled and the primitive is a polygon, it then may be rejected by a culling test. Depending upon the polygon mode, a polygon may be drawn as points or lines. The results or this stage are complete geometric primitives, which are the transformed and clipped vertices with related color, depth, and sometimes texture-coordinate values and guidelines for the rasterization step.

Pixel Operations

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. If pixel data is read from the frame buffer, pixel-transfer operations (scale, bias, mapping, and clamping) are performed. Then these results are packed into an appropriate format and returned to an array in system memory. There are special pixel copy operations to copy data in the frame buffer to other parts of the frame buffer or to the texture memory. A single pass is made through the pixel transfer operations before the data is written to the texture memory or back to the frame buffer.

Texture Assembly

An OpenGL application may wish to apply texture images onto geometric objects to make them look more realistic. If several texture images are used, it's wise to put them into texture objects so that you can easily switch among them. Some OpenGL implementations may have special resources to accelerate texture performance. There may be specialized, high-performance texture memory. If this memory is available, the texture objects may be prioritized to control the use of this limited and valuable resource.

Rasterization

Rasterization is the conversion of both geometric and pixel data into fragments. Each fragment square corresponds to a pixel in the frame buffer. Line and polygon stipples, line width, point size, shading model, and coverage calculations to support ant aliasing are taken into consideration as vertices are connected into lines or the interior pixels are calculated for a filled polygon. Color and depth values are assigned for each fragment square.

2.REQUIREMENT SPECIFICATION

2.1 SOFTWARE REQUIREMENTS SPECIFICATION

This section attempts to bring out the requirements and specifications as given out by the Visvesvaraya Technological University for the completion of the package. Minimum requirements expected are cursor movement, editing picture objects like point, line, circle, ellipse and polygons. Transformations on objects/selected area should be possible. User should be able to open the package do the required operations and exit from the environment.

2.2 EXTERNAL INTERFACE REQUIREMENTS

User Interface:

The interface for the 2D package requires for the user to have a mouse connected, and the corresponding drivers software and header files installed. For the convenience of the user, there are menus and sub -menus displayed on the screen.

Menus:

The Menus consists of various operations related to drawing on the area specified. It also has the 'clear' option for clearing the screen and also changes the background color.

Hardware Interface:

The standard output device, as mentioned earlier has been assumed to be a color monitor. It is quite essential for any graphics package to have this, as provision of color options to the user is a must. The mouse, the main input device, has to be functional. A keyboard is also required. .

Apart from these hardware requirements, there should be sufficient hard disk space and primary memory available for proper working of the package.

Software Interface:

The editor has been implemented on the DOS platform and mainly requires an appropriate version of the compiler to be installed and functional. Though it has been implemented on DOS, it is pretty much platform independent with the

Hardware Requirements:

System : Intel
Frequency : 3.0 GHz
RAM : 4 GB
Disk Capacity : 500 GB

Software Requirements:

Operating System : WINDOWS XP/7/8/10
Compiler, IDE : Dev c++,Code blocks.

3.SYSTEM DESIGN

In this project we are presenting 3D Bicycle Animation, which is one the best OpenGL Car Program. This 3D OpenGL Program have so many advance features in it. We can see below the all features that included in this 3D Animated Program.

Features of the 3D Bicycle Animation OpenGL Program:

1. **Start Screen**: As you execute the program, first thing come is the start screen where you will see the details of the project. It also has the user interaction instructions about the use of mouse and keyboard functionality. There is option of going to main screen,

2. **Main Screen**: After you get in to main screen from the start screen you will see the 3D Bicycle . If you press right mouse button you will get the menu options to select for particular functionality to execute. Details is mentioned in below user interaction section.

3**. Comments**: Most of the codes in this 3D OpenGL Program, is commented hence easy to understand the whole program

User interaction section

The user interaction is one of the most important part of any program. This 3D OpenGL Program also have user interaction both using keyboard as well as mouse. Below is the description of the uses and functionality of keys and menus in this 3D OpenGL Program

Keyboard interaction section

- '+' increse the speed
- '-' Redcuce the speed
- 'b' to rotate the handle in clock wise direction
- 'z' to rotate the handle in anti-clock wise direction
- 's' to reset the screen

4. IMPLEMENTATION SOURCE CODE

```
/**************************************
* Beginning of the header file "cycle.h"
**************************************/
#include<GL/glut.h>
#include<stdio.h>
#include<stdlib.h>
#include<math.h>

#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   2.0f
#define INC_SPEED     0.05f

/**************************************
*    All the Global Variables are Here
**************************************/
/**************************************
*   Cycle - related variables
**************************************/
GLfloat pedalAngle, speed, steering;
```

```
/*******************************
*  User view realted variables
*******************************/
GLfloat camx,camy,camz;
GLfloat anglex,angley,anglez;

/*****************************
*  Mouse related variables
*****************************/
int prevx,prevy;
GLenum Mouse;

/*************************
*  Cycle position related
*  variables
*************************/
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 idle(void);
void updateScene(void);
void landmarks(void);
void special(int key,int x,int y);
void keyboard(unsigned char key,int x,int y);
void mouse(int button,int state,int x,int y);
void motion(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);

/************************************
```

```
*   End of the header file
************************************/

/************************************
*  Start of the source file "cycle.c"
************************************/
//#include "cycle.h"

/*****************************************
*        A
*       1  ========== 2
*        /\      | B
*       /   \     / 5
*     E /   \D   /
*      /     \  / C
*     /     \/
*    3 ==========/ 4
*       F
*     1   =  212,82
*     2   =  368,82
*     5   =  369,94
*     3   =  112,220
*     4   =  249,232
*
*     214  =  73
*     124  =  55
*     142  =  52
*     143  =  73
*     134  =  50
*     431  =  57
*****************************************/

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();
```

```
}

// called by idle()
void updateScene()
{
  GLfloat xDelta, zDelta;
  GLfloat rotation;
  GLfloat sin_steering, cos_steering;

  // if the tricycle is not moving then do nothing
  if (-INC_SPEED < speed && speed < INC_SPEED) return;

  if(speed < 0.0f)
      pedalAngle = speed = 0.0f;

  // otherwise, calculate the new position of the tricycle
  // and the amount that each wheel has rotated.
  // The tricycle has moved "speed*(time elapsed)".
  // We assume that "(time elapsed)=1".

  xDelta = speed*cos(radians(direction + steering));
  zDelta = speed*sin(radians(direction + steering));
  xpos += xDelta;
  zpos -= zDelta;
  pedalAngle = degrees(angleSum(radians(pedalAngle), speed/RADIUS_WHEEL));

  // we'll be using sin(steering) and cos(steering) more than once
  // so calculate the values one time for efficiency
  sin_steering = sin(radians(steering));
  cos_steering = cos(radians(steering));

  // see the assignment 3 "Hint"
  rotation = atan2(speed * sin_steering, CYCLE_LENGTH + speed * cos_steering);
  direction = degrees(angleSum(radians(direction),rotation));
}

// angleSum(a,b) = (a+b) MOD 2*PI
// a and b are two angles (radians)
//  both between 0 and 2*PI
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;
}

/*********************************************
```

```
*  Draw the metal frame of the cycle and also
*  draw the seat and the back wheel with
*  this.
*  All these parts are always together in the
*  same plane.They never move out ot the
*  PLANE!  ;)
**********************************************/
void drawFrame()
{
  glColor3f(1.0f,0.0f,0.0f);

  /*******************************
  *  First draw the all the items
  *  at the center of the frame.
  *  Draw the bigger gear,the small
  *  cylinder acting as the socket
  *  for the pedals.Also DON'T
  *  forget to draw the two
  *  connecting cemtral rods
  *******************************/
  glPushMatrix();
    /*******************************
    *  Allow me to draw the BIGGER
    *  gear and the socket cylinder
    *******************************/
    glPushMatrix();
      /*****************************
      *  Let the gear have the
      *  green color
      *****************************/
      glColor3f(0.0f,1.0f,0.0f);

      /*************************
      *  The gear should be
      *  outside the frame !!!
      *  This is the bigger
      *  GEAR
      *************************/
      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();
      /*************************
      *  Restore the color of the
      *  frame
      *************************/
      glColor3f(1.0f,0.0f,0.0f);
```

```
    glTranslatef(0.0f,0.0f,-0.2f);
    ZCylinder(0.08f,0.32f);
  glPopMatrix();
  /*****************************
  *  Lets first draw the
  *  rightmost rod of the frame
  *****************************/
  glRotatef(RIGHT_ANGLE,0.0f,0.0f,1.0f);
  XCylinder(ROD_RADIUS,RIGHT_ROD);

  /*****************************
  *  Now draw the centre rod of
  *  the frame which also supports
  *  the seat
  *****************************/
  glRotatef(MIDDLE_ANGLE-RIGHT_ANGLE,0.0f,0.0f,1.0f);
  XCylinder(ROD_RADIUS,MIDDLE_ROD);
  /*****************************
  *  We have drawn the support.So
  *  let's draw the seat with a
  *  new color
  *****************************/
  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();
  /**********************
  *  Restore the color !
  **********************/
  glColor3f(1.0f,0.0f,0.0f);
glPopMatrix();
/*******************************
*  Draw the horizontal part of
*  the frame.
*******************************/

/*******************************
*  Draw the main single rod
*  connecting the center of the
*  frame to the back wheel of the
*  cycle
*******************************/
glPushMatrix();
  glRotatef(-180.0f,0.0f,1.0f,0.0f);
  XCylinder(ROD_RADIUS,BACK_CONNECTOR);

  /*********************************
```

```
   *  Draw the two rods on the either
   *  side of the wheel
   *  These rods are part of the
   *  horizontal part of the cycle
   ************************************/
   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();

 /************************************
 *  Draw the leftmost rods of the
 *  frame of the cycle
 ************************************/
 glPushMatrix();
   glTranslatef(-(BACK_CONNECTOR+RADIUS_WHEEL+TUBE_WIDTH),0.0f,0.0f);
     /*********************************
   *  Transalted to the back wheel
   *  position.Why not draw the back
   *  wheel and also the gear ? :))
   *********************************/
   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);

   /************************************
   *  Draw the two rods on the either
   *  side of the wheel connecting the
   *  backwheel and topmost horizontal
   *  part of the wheel
   ************************************/
   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();

/*****************************
*  Draw the single rod of the
*  same setup
*****************************/
glTranslatef(WHEEL_LEN,0.0f,0.0f);
XCylinder(ROD_RADIUS,CRANK_ROD);

/*****************************
*  Now Draw The topmost
*  Horizontal rod
*****************************/
glTranslatef(CRANK_ROD,0.0f,0.0f);
glRotatef(-LEFT_ANGLE,0.0f,0.0f,1.0f);
XCylinder(ROD_RADIUS,TOP_LEN);

/*******************************
*  Now instead of again traversing
*  all the way back and again
*  forward.WHY NOT DRAW THE
*  HANDLE FROM HERE ITSELF?
*******************************/
/*****************************
*  Now draw the handle and
*  small support rod which
*  is incorporated in the
*  frame itself.
*  Set y-axis at te required
*  incline.
*****************************/
glTranslatef(TOP_LEN,0.0f,0.0f);
glRotatef(-FRONT_INCLINE,0.0f,0.0f,1.0f);

/*****************************
*  Draw the small rod that acts
*  as the socket joining the
*  frame and the handle
*****************************/
glPushMatrix();
  glTranslatef(-0.1f,0.0f,0.0f);
  XCylinder(ROD_RADIUS,0.45f);
glPopMatrix();

/*****************************
*  I Hope the handle can rotate
*  about its mean position
*****************************/
```

```
glPushMatrix();
  glRotatef(-steering,1.0f,0.0f,0.0f);
  /*****************************
  *  Roll back to the height of
  *  the handle to draw it
  *****************************/
  glTranslatef(-0.3f,0.0f,0.0f);

  /*****************************
  *  We cannot use the incline
  *  the incline to draw the
  *  horizontal part of the rod
  *****************************/
  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();

  /*****************************
  *  Using this incline now draw
  *  the handle.Maybe use this again
  *  to draw the wheel. ;)
  *****************************/
  glPushMatrix();
    /*****************************
    *  Draw the main big rod
    *****************************/
    XCylinder(ROD_RADIUS,CRANK_ROD);

    /*****************************
    *  Why not draw the two rods and
    *  the WHEEL?  :)
    *  Yes!So,first go to the
    *  end of the main rod.
    *****************************/
```

```
        glTranslatef(CRANK_ROD,0.0f,0.0f);
        glRotatef(CRANK_ANGLE,0.0f,0.0f,1.0f);

        /*******************************
        *   Draw the two rods connecting
        *   the handle and the front
        *   wheel.
        *   The two rods are at a incline
        *   to the connector.
        *******************************/
        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();
          /*******************************
        *   Why not draw the wheel.
        *   The FRONT wheel to be precise
        *******************************/
        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();
}

// Portions of this code have been borrowed from Brian Paul's Mesa
// distribution.
/*
* Draw a gear wheel.  You'll probably want to call this function when
* building a display list since we do a lot of trig here.
*
* Input:  inner_radius - radius of hole at center
*     outer_radius - radius at center of teeth
*     width - width of gear
*     teeth - number of teeth
*     tooth_depth - depth of tooth
*/

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 );

/* draw front face */
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();

/* draw front sides of teeth */
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 );

/* draw back face */
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();

 /* draw back sides of teeth */
 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();


 /* draw outward faces of teeth */
 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 );

   /* draw inside radius cylinder */
   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();

}

/****************************************
*   Could not model the exact chain
*   Think it eats up a lot of power if
*   approximated by a lot of spheres
*   So approximated with the stippled
*   lines instead
****************************************/
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()
{
  /*********************************
  *  Draw the top of the seat
  *********************************/
  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();

  /*********************************
  *  Draw the bottom base part of the
  *  seat
  *********************************/
  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();

  /**********************
  *  Draw the sides!
  **********************/
  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()
{
  glColor3f(0.0f,0.0f,1.0f);
  /*************************
  *  Lets draw the two pedals
  *  offset from the center
  *  of the frame.
  *************************/

  /*****************************
  *  First draw the one visible
  *  to the viewer
  *****************************/
  glPushMatrix();
```

```
  glTranslatef(0.0f,0.0f,0.105f);
  glRotatef(-pedalAngle,0.0f,0.0f,1.0f);
  glTranslatef(0.25f,0.0f,0.0f);
  /*************************
  *  Draw the pedal rod
  *************************/
  glPushMatrix();
    glScalef(0.5f,0.1f,0.1f);
    glutSolidCube(1.0f);
  glPopMatrix();

  /************************
  *  Draw the resting pad
  ************************/
  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();

/*******************************
*  Draw the one on the other
*  side  of the frame
*******************************/
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);

  /***************************
  *  Now again draw the pedal
  *  rod
  ***************************/
  glPushMatrix();
    glScalef(0.5f,0.1f,0.1f);
    glutSolidCube(1.0f);
  glPopMatrix();

  /***************************
  *  Draw the resting pad of
  *  the 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;
  //   Draw The Rim
  glColor3f(1.0f,1.0f,1.0f);
  glutSolidTorus(0.06f,0.92f,4,30);
  //   Draw The Central Cylinder
  //   Length of cylinder  0.12f
  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);

  //   Draw The Spokes
  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();
  }

  //   Draw The Tyre
  glColor3f(0.0f,0.0f,0.0f);
  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(0.0f,1.0f,0.0f);

    /************************************
    *  Draw the ground for the cycle
    *  Looks incomplete with it!Don't
    *  forget to define the normal
    *  vectors for the vertices.
    *  gotta fix this bug!
    ************************************/
    glBegin(GL_LINES);
    for(i=-100.0f ; i<100.0f ; i += 1.0f)
    {
      glVertex3f(-100.0f,-RADIUS_WHEEL,i);
      glVertex3f( 100.0f,-RADIUS_WHEEL,i);
      glVertex3f(i,-RADIUS_WHEEL,-100.0f);
      glVertex3f(i,-RADIUS_WHEEL,100.0f);
    }
    glEnd();
}

void display(void)
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glEnable(GL_NORMALIZE);
```

```
  glPushMatrix();
    /*******************************
     *   Prepare the rotations
     *   and start doing the
     *   remaining scene
     *******************************/
    glRotatef(angley,1.0f,0.0f,0.0f);
    glRotatef(anglex,0.0f,1.0f,0.0f);
    glRotatef(anglez,0.0f,0.0f,1.0f);

    /***********************
     *   Start rendering
     *   the scene;
     *   the bicycle ;)
     ***********************/

    landmarks();

    /****************************
    *   Move the cycle.
    ****************************/
    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();
}

/************************
*   Returns the absolute
*   value of a given
*   float number
************************/
GLfloat Abs(GLfloat a)
{
  if(a < 0.0f)
    return -a;
```

```
  else
    return a;
}

/************************
*  Returns the value of
*  the given angle in
*  degrees
************************/
GLfloat degrees(GLfloat a)
{
  return a*180.0f/PI;
}

/************************
*  Returns the value of
*  the given angle in
*  radians
************************/
GLfloat radians(GLfloat a)
{
  return a*PI/180.0f;
}

/*************************
*  The idle function of
*  the program which makes
*  the contniuous loop
*************************/
void idle(void)
{
  updateScene();
  glutPostRedisplay();
}

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();
}

/*****************************
*  Reset The scene
*****************************/
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 's':
    case 'S':
      reset();
      break;
    case 'z':
      if(steering < HANDLE_LIMIT)
          steering += INC_STEERING;
      break;
    case 'b':
      if(steering > -HANDLE_LIMIT)
        steering -= INC_STEERING;
      break;
    case '+':
      speed += INC_SPEED;
      break;
    case '-':
      speed -= INC_SPEED;
      break;
    case 27:
      exit(1);
```

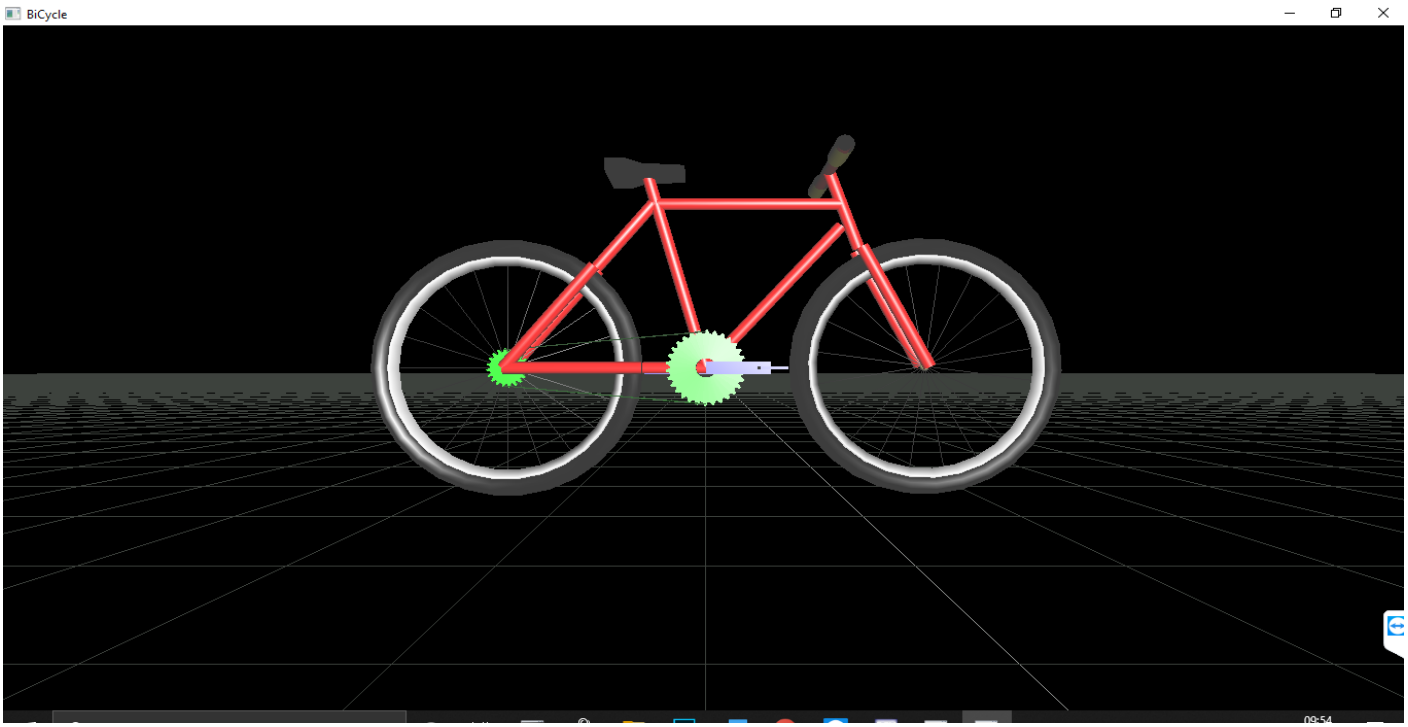
```
  }

  /*******************************
  *  Where is my Cycle?
  *******************************/

  /*******************************
  *  When you rotate the handle the
  *  cycle as a whole does not rotate
  *  at once immediately.
  *  For each unit of time, the
  *  handle slowly begins to align
  *  with the rest of the body of the
  *  cycle.
  *  Tough this is a gross approximation
  *  it's workig fine now, maybe i'll
  *  get some bugs. :<
  *  I Think that the rate at which the
  *  handle aligns with the body is
  *  dependant on the speed too!!
  *  The rate is given by 'delta'
  *  and the speed is given by 'speed'
  *  Now there should be no problems
  *********************************/

  /**********************
  *  Check out the error
  *  conditions  ;>
  **********************/
  pedalAngle += speed;
  if(speed < 0.0f)
    speed = 0.0f;
  if(pedalAngle < 0.0f)
    pedalAngle = 0.0f;
  if(pedalAngle >= 360.0f)
    pedalAngle -= 360.0f;

  /******************
  *  Go! Display ;)
  ******************/
  glutPostRedisplay();
}

void mouse(int button,int state,int x,int y)
{
  switch(button)
  {
    case GLUT_LEFT_BUTTON:
```

```
      if(state==GLUT_DOWN)
      {
        Mouse=GLUT_DOWN;
        prevx=x;
        prevy=y;
      }
      if(state==GLUT_UP)
      {
        Mouse=GLUT_UP;
      }
      break;
    case GLUT_RIGHT_BUTTON:
      /*  DO NOTHING  */
      break;
  }
  glutPostRedisplay();
}

void passive(int x,int y)
{
/*  DO NOTHING  */
}

void motion(int x,int y)
{
  if(Mouse==GLUT_DOWN)
  {
    int deltax,deltay;
    deltax=prevx-x;
    deltay=prevy-y;
    anglex += 0.5*deltax;
    angley += 0.5*deltay;
    if(deltax!=0 && deltay!=0)
      anglez += 0.5*sqrt(deltax*deltax + deltay*deltay);

    if(anglex < 0)
      anglex+=360.0;
    if(angley < 0)
      angley+=360.0;
    if(anglez < 0)
      anglez += 360.0;

    if(anglex > 360.0)
      anglex-=360.0;
    if(angley > 360.0)
      angley-=360.0;
    if(anglez > 360.0)
      anglez-=360.0;
```

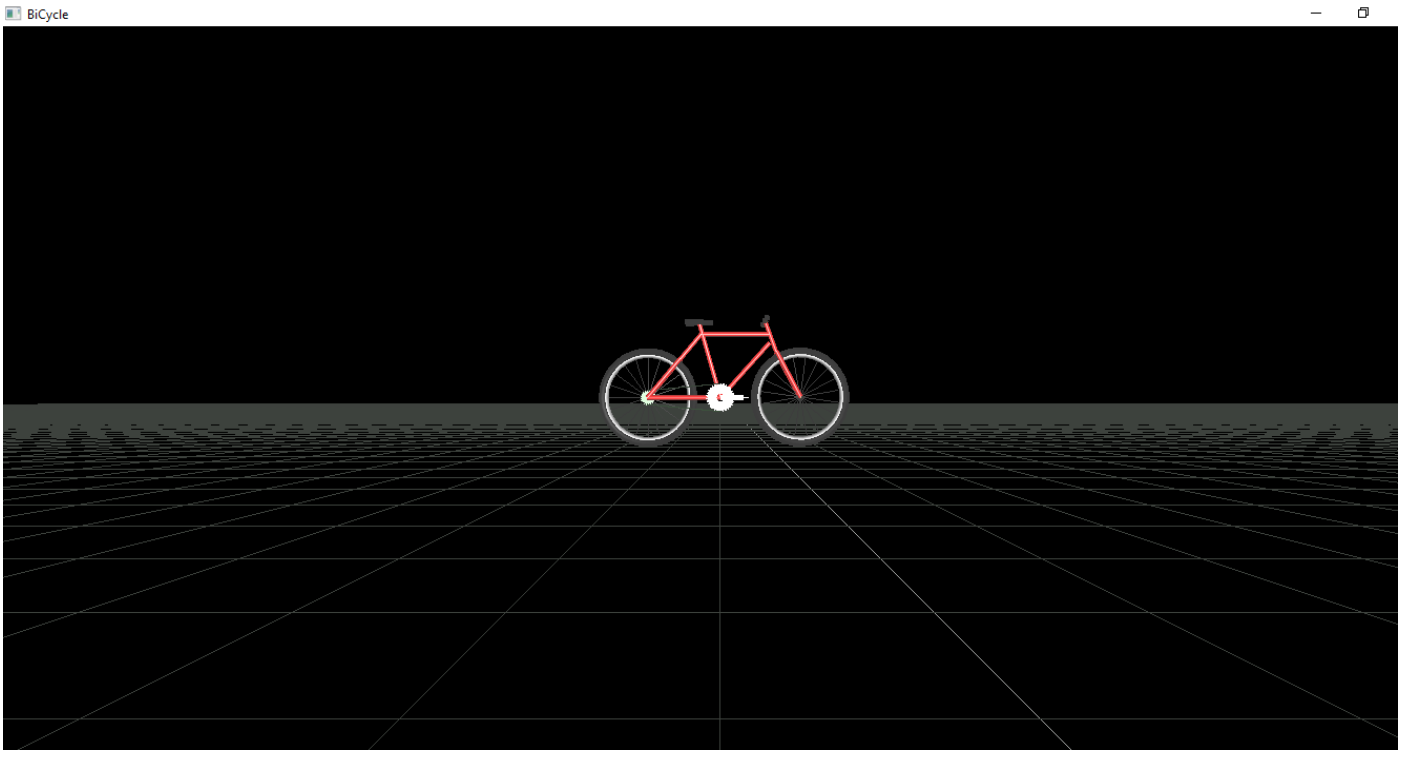
```
  }
  else
  {
    Mouse=GLUT_UP;
  }
  prevx=x;
  prevy=y;
  glutPostRedisplay();
}

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);
  //Angle,Aspect Ratio,near plane,far plane
  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);
  glutSpecialFunc(special);
  glutKeyboardFunc(keyboard);
  glutMouseFunc(mouse);
  glutMotionFunc(motion);
  glutPassiveMotionFunc(passive);
  glutSetCursor(GLUT_CURSOR_CROSSHAIR);
}

void help(void)
{
  printf("Hierarchical 3D Model of a Bicycle\n");
  printf("TCS2111- Computer Graphics\n");
  printf("Group Project\n\n");
  printf("'+' to increase the speed\n");
  printf("'-' to decrease the speed\n");
  printf("'b' to rotate the handle in clockwise direction\n");
  printf("'z' to rotate the handle in anti-clockwise direction\n");
  printf("'s' or 'S' to reset the scene\n");
  printf("Arrow keys to move the camera\n");
  printf("Mouse to move the scene\n");
}
```

```
int main(int argc,char *argv[])
{
  help();
  glutInit(&argc,argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
  glutInitWindowPosition(100,100);
  glutInitWindowSize(WIN_WIDTH,WIN_HEIGHT);
  glutCreateWindow("BiCycle");
  init();
  glSetupFuncs();
  glutMainLoop();
}
```

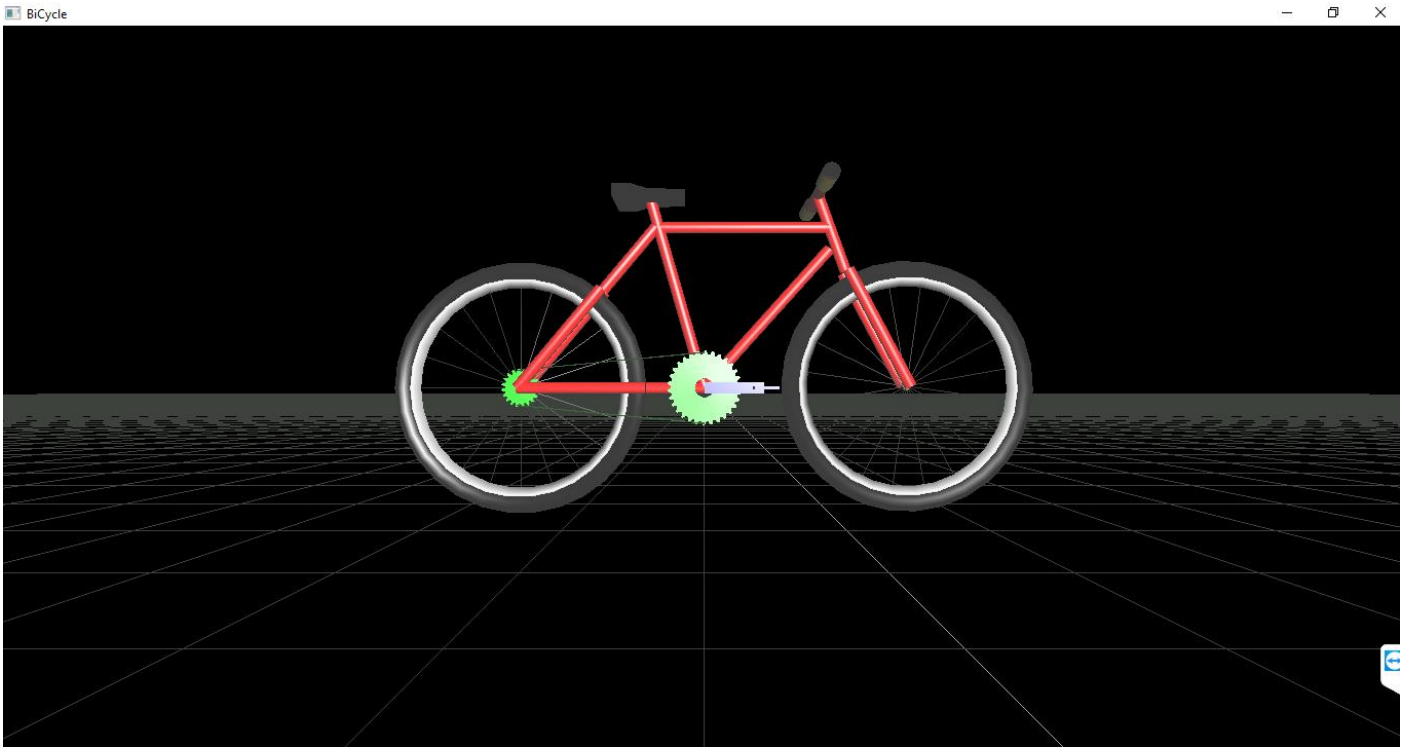
SNAPSHOTS



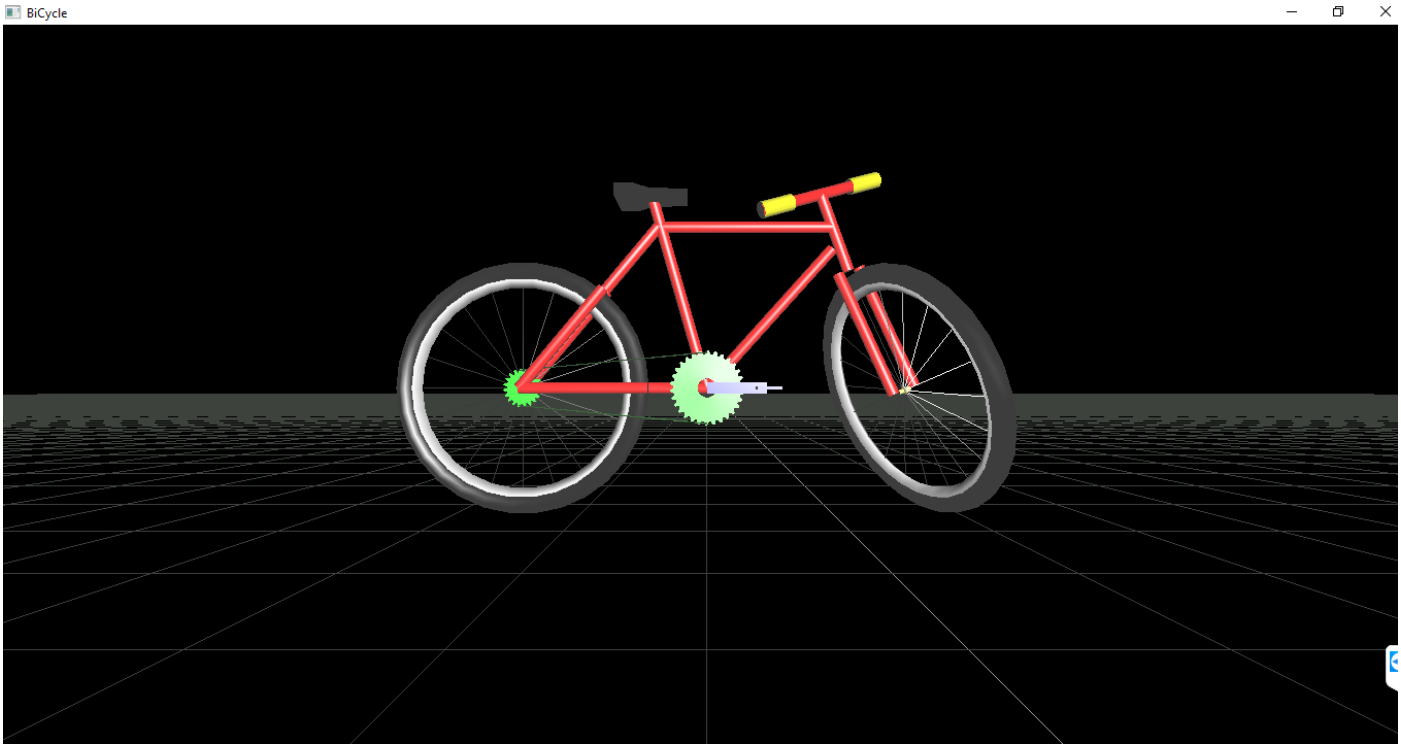
1:Main screen



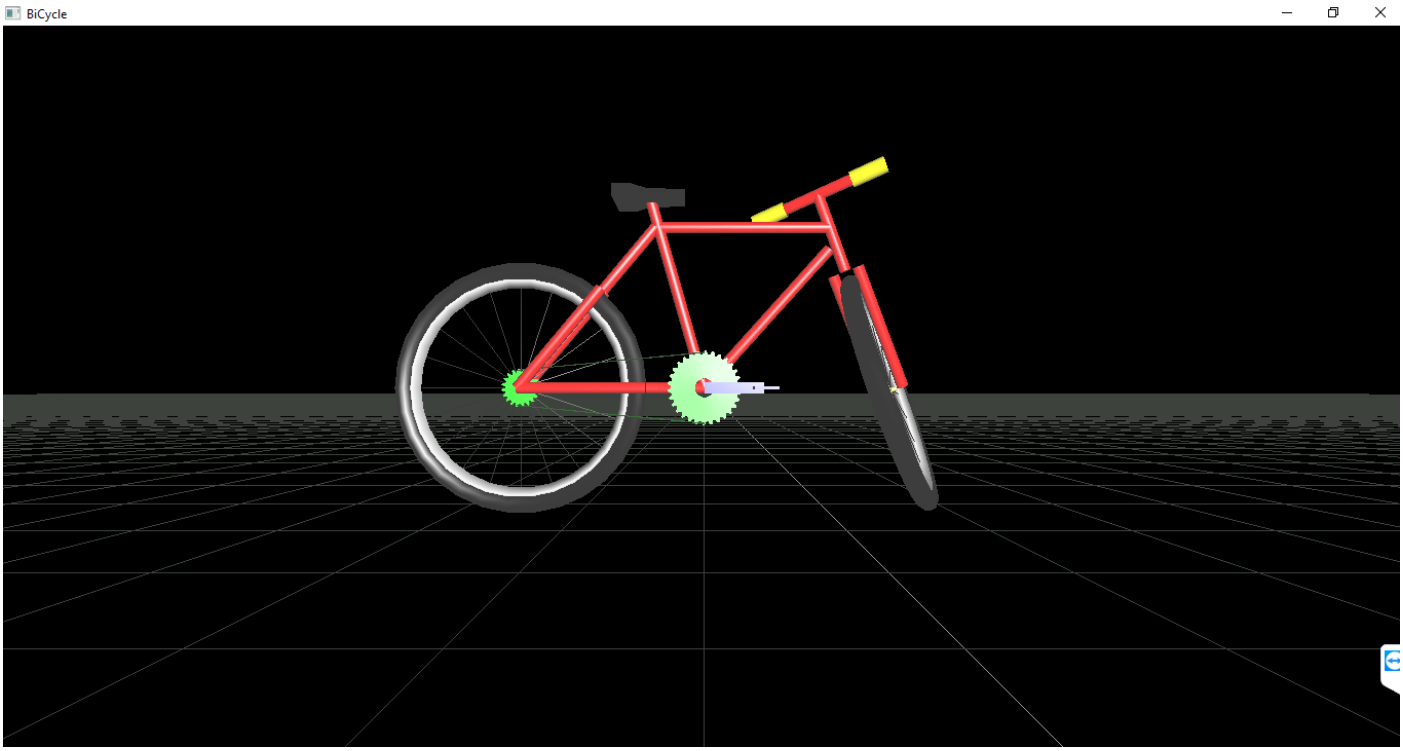
2:Zoom out



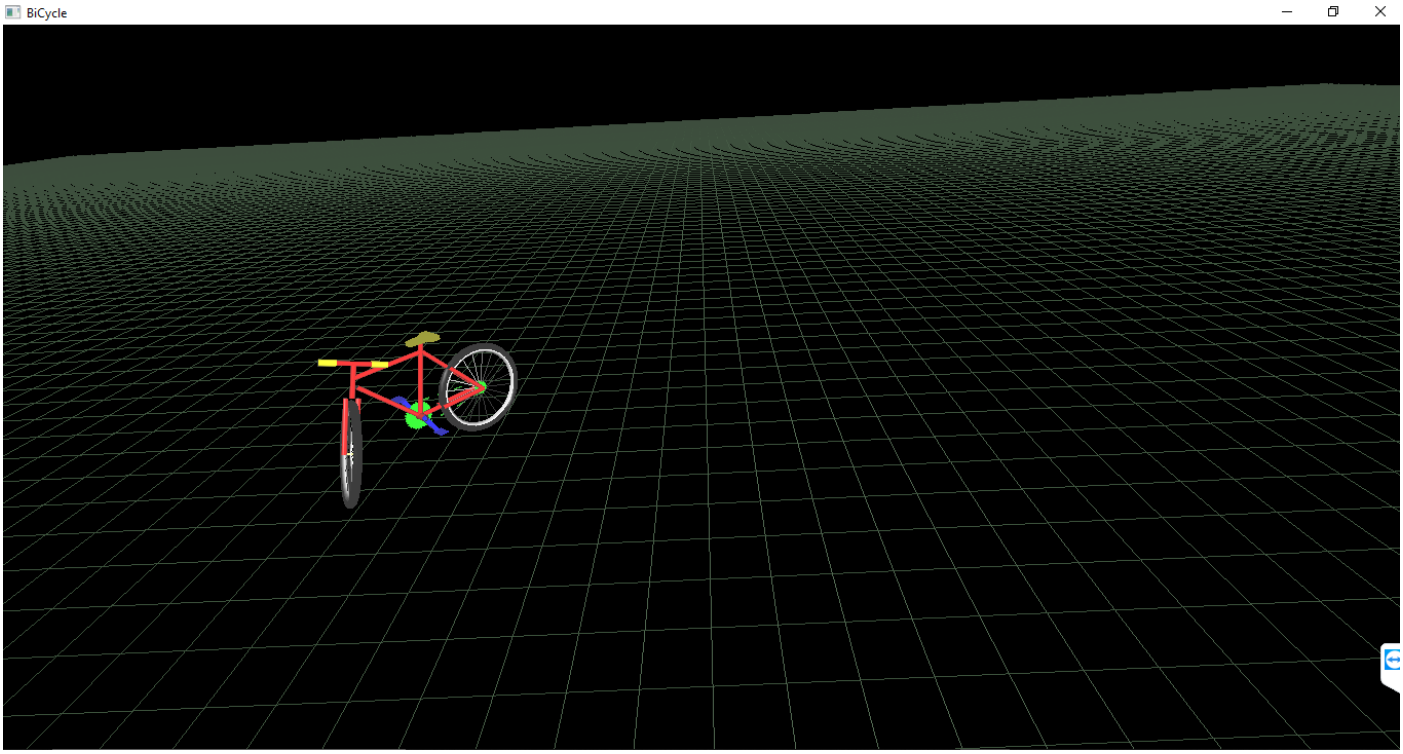
3:Zoom in



4:Rotating in clockwise direction



5:Rotating in anti-clock wise direction



6:Viewing in top

6.TESTING

6.1 LIFE CYCLE OF TESTING

FIG 6.1 TESTING LIFE CYCLE

6.2 TYPES OF TESTING

6.2.1 MANUAL TESTING

Manual testing includes testing a software manually, i.e., without using any automated tool or any script. In this type, the tester takes over the role of an end-user and tests the software to identify any unexpected behavior or bug. There are different stages for manual testing such as unit testing, integration testing, system testing, and user acceptance testing.

6.2.2 AUTOMATION TESTING

Automation testing, which is also known as Test Automation, is when the tester writes scripts and uses another software to test the product. This process involves

automation of a manual process. Automation Testing is used to re-run the test scenarios that were performed manually, quickly, and repeatedly.

6.2.3 BLACK-BOX TESTING

The technique of testing without having any knowledge of the interior workings of the application is called black-box testing. The tester is oblivious to the system architecture and does not have access to the source code. Typically, while performing a black-box test, a tester will interact with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon.

6.2.4 WHITE-BOX TESTING

White-box testing is the detailed investigation of internal logic and structure of the code. White-box testing is also called **glass testing** or **open-box testing**. In order to perform **white-box** testing on an application, a tester needs to know the internal workings of the code. The tester needs to have a look inside the source code and find out which unit/chunk of the code is behaving inappropriately.

6.2.5 GREY-BOX TESTING

Grey-box testing is a technique to test the application with having a limited knowledge of the internal workings of an application. In software testing, the phrase the more you know, the better carries a lot of weight while testing an application.

6.2.6 FUNCTIONAL TESTING

This is a type of black-box testing that is based on the specifications of the software that is to be tested. The application is tested by providing input and then the results are examined that need to conform to the functionality it was intended for. Functional testing of a software is conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements.

6.2.7 UNIT TESTING

This type of testing is performed by developers before the setup is handed over to the testing team to formally execute the test cases. Unit testing is performed by the respective developers on the individual units of source code assigned areas. The developers use test data that is different from the test data of the quality assurance team. The goal of unit testing is to isolate each part of the program and show that individual parts are correct in terms of requirements and functionality.

6.2.8 INTEGRATION TESTING

Integration testing is defined as the testing of combined parts of an application to determine if they function correctly. Integration testing can be done in two ways: Bottom-up integration testing and Top-down integration testing.

6.2.9 SYSTEM TESTING

System testing tests the system as a whole. Once all the components are integrated, the application as a whole is tested rigorously to see that it meets the specified Quality Standards. This type of testing is performed by a specialized testing team.

6.3.0 REGRESSION TESTING

Whenever a change in a software application is made, it is quite possible that other areas within the application have been affected by this change. Regression testing is performed to verify that a fixed bug hasn't resulted in another functionality or business rule violation. The intent of regression testing is to

ensure that a change, such as a bug fix should not result in another fault being uncovered in the application.

6.3.1 ACCEPTANCE TESTING

This is arguably the most important type of testing, as it is conducted by the Quality Assurance Team who will gauge whether the application meets the intended specifications and satisfies the client's requirement. The QA team will have a set of pre-written scenarios and test cases that will be used to test the application. By performing acceptance tests on an application, the testing team will deduce how the application will perform in production. There are also legal and contractual requirements for acceptance of the system.

6.3.2 ALPHA TESTING

This test is the first stage of testing and will be performed amongst the teams (developer and QA teams). Unit testing, integration testing and system testing when combined together is known as alpha testing. During this phase, the following aspects will be tested in the application:

- Spelling Mistakes
- Broken Links
- Cloudy Directions
- The Application will be tested on machines with the lowest specification to test loading times and any latency problems.

6.3.3BETA TESTING

This test is performed after alpha testing has been successfully performed. In beta testing, a sample of the intended audience tests the application. Beta testing is also known as **pre-release testing**. Beta test versions of software are ideally distributed to a wide audience on the Web, partly to give the program a "real-world" test and partly to provide a preview of the next release. In this phase, the audience will be testing the following:

- Users will install, run the application and send their feedback to the project team

- Typographical errors, confusing application flow, and even crashes.
- Getting the feedback, the project team can fix the problems before releasing the software to the actual users.

6.3.4 NON-FUNCTIONAL TESTING

This section is based upon testing an application from its non-functional attributes. Non-functional testing involves testing a software from the requirements which are nonfunctional in nature but important such as performance, security, user interface, etc. Some of the important and commonly used non-functional

- Load Testing
- Performance Testing
- Stress Testing

7. CONCLUSION

It has been an interesting journey through the development of this project. At the beginning we used our limited knowledge to implement only the basic features. However, through the months of development, new issues and bugs led to new ideas which led to newer methods of implementation which, in turn, led to us learning even more features of the OpenGL and apply more creative and efficient ways to perform the older functions. New methods helped us in adding flexibility to the various parts of the program, making the further addition of newer features easier and less time consuming which, again, led to the possibility of adding even more features. This sequential chain reaction of progress and ideas has enabled to learn so much through the months of working on this project and we have done our best to add as many features as we could and provide a user interface that is easy and intuitive to use.

Before concluding, it is worth mentioning that this project would never have been possible without the tremendous amount of encouragement by the staff and guides of our department.

We are content with the outcome of this project and are hopeful that it meets the requirements expected and we wish that it may inspire others to be creative and critical in the field of computer graphics and have a newfound appreciation for the Open Graphics Library.

Although it isn't noticeable on a relatively newer and powerful processor, the method of applying the transforms pixel by pixel is a very taxing process, especially for larger images of 1080p or higher. A possible feature to implement could be to perform all the transforms using multithreading to split the workload onto multiple threads, reducing the minimum computational time required between time steps.

• Since Contrast modifies the RGB values instead of HSL, the contrast should be the last thing that should be changed after changing any brightness or saturation. • Addition of complete pixel blur in both directions, and several other transforms is also notable.

• Allowing the user to select his own color model from RGB, HSL, HSV etc. to allow more fine tuning of the Image.

• More efficient memory management techniques to prevent heavy usage of memory

8. BIBILIOGRAPHY

REFERENCE BOOKS:

- Interactive Computer Graphics A top –down approach with OpenGL – Edward Angel, 5th Edition, Addition-Wesley 2008.
- Computer Graphics Using OpenGL - F.S.Hill.Jr. 2nd Edition, Pearson Education, 2001.

Web Sites:

www.opengl.org

www.sourcecode.com