**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

"JnanaSangama", Belgaum: 590018



A Mini Project Report (17CSL68)

on

**Blockchain Visualization**

A mini project report submitted in partial fulfilment of the requirement for the award of the degree of

**Bachelor of Engineering**

**in**

## **Computer Science & Engineering**

Submitted by

**Pavan R Bhat (1AY17CS061)**

Under the guidance of

**Prof. Naganandini G**

Department of Computer Science & Engineering



**Acharya Institute of Technology**

**Department of Computer Science & Engineering**

**Soladevanahalli, Bangalore-560107**

**ACKNOWLEDGEMENT**

We express our gratitude to our institution and management for providing us with good infrastructure, laboratory facilities and inspiring staff, and whose gratitude was of immense help in completion of this mini project successfully.

We express our sincere gratitude to our principal, **Dr. M R Prakash** for providing us the required environment and for his valuable suggestions.

Our sincere thanks to **Dr. Prashanth C M,** Head of the Department. Computer Science and Engineering, Acharya Institute of Technology for his valuable support and for rendering us the resources for this mini project.

We heartily thank **Prof. Varalakshmi B D,** Associate Professor**, Prof. Vani K S, Prof. Naganandini N, Prof. Avinash K** AssistantProfessors, Department of Computer Science and Engineering, Acharya Institute of Technology who guided us with valuable suggestions in completing this mini project at every stage.

Our gratitude thanks should be rendered to many people who helped us in all possible ways.

**PAVAN R BHAT**

**1AY17CS061**

**ACHARYA INSTITUTE OF TECHNOLOGY**

(Affiliated to Visvesvaraya Technological University, Belgaum)

Soladevanahalli, Bangalore – 560090

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**Certificate**

Certified that the Computer Graphics Mini Project entitled **“Blockchain Visualization”** is a bonafide work carried out by **Mr. Pavan R Bhat (1AY17CS061)** in partial fulfillment for the award of degree of **Bachelor of Engineering in Computer Science & Engineering of the Visvesvaraya Technological University**, Belgaum during the academic year **2019-2020.** It is certified that all corrections/ suggestions indicated for internal assessments have been incorporated in the Report deposited in the departmental library. The Mini project report has been approved as it satisfies the academic requirements in respect of Mini Project work prescribed for the **Bachelor of Engineering Degree**.

**Signature of Guides Signature of H.O.D**

**Name of the examiners** **Signature with date**

1.

2.

**ABSTRACT**

Many people think “**Block chain Technology”** may replace everything in the future. It is very important to know about the Block chains. The main aim of this project is to show the demonstration of working of Blockchain. This project demonstrates the basics of the Blockchain and it’s working, the project completely uses graphics and has been done using OpenGL. The code used in the program is very simple. We can easily understand the working of Blockchain by seeing the output of this project.

Minimum code is written to show the animation so that the programmers can easily understand the source code as well. In the future I will distribute the code as open source code and anyone can modify the project easily. I have ensured the minimum use of functions and maximum code reuse. I Have used some inbuild OpenGL functions to draw some primitives. There are some primitives which are drawn only using OpenGL user defined functions. 3D and 2D concepts are used based on requirement of the demonstration. Code written is dynamic in nature and behaves according to the user inputs and behaviour. Output window of the OpenGL is completely interactive. Follow the instruction given in the screen to run the project according to user’s comfort. Using the techniques learned, I have tried to make the most efficient code.

**CONTENTS**

1. **Introduction to OpenGL …………………………….…… 1**
   1. **Computer Graphics………………………… 6**
   2. **OpenGL Technology……………………….. 7**
2. **Project Description…………………………………………. 9**
3. **Design Constraints…………………………………………. 10**
   1. **Hardware Constraints**
   2. **Software Constraints**
4. **Architecture…………………………………………………11**
5. **Code Implementation……………………………………….12**
6. **Screen Snapshots……………………………………………25**
7. **Future Enhancements……………………………………….27**
8. **Conclusion……………………………………………………28**
9. **Appendix……………………………………………………...29**
10. **Bibliography……………………………………………33**

**Chapter 1**

**INTRODUCTION TO OPENGL**

Computer graphics are graphics created using computers and, more generally, the representation and manipulation of pictorial data by a computer. The development of computer graphics has made computers easier to interact with and better for understanding and interpreting many types of data. Developments in computer graphics have had a profound impact on many types of media and have revolutionized the animation and video game industry. Today computers and computer-generated images touch many aspects of our daily life. Computer imagery is found on television, in newspapers, in weather reports, and during surgical procedures. A well-constructed graph can present complex statistics in a form that is easier to understand and interpret. Such graphs are used to illustrate papers, reports, theses, and other presentation material. A range of tools and facilities are available to enable users to visualize their data, and computer graphics are used in many disciplines. We implement computer graphics using the OpenGL API. 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.

OpenGL is a low-level graphics library specification. It makes available to the programmer a small set of geometric primitives - points, lines, polygons, images, and bitmaps. OpenGL provides a set of commands that allow the specification of geometric objects in two or three dimensions, using the provided primitives, together with commands that control how these objects are rendered (drawn). Since OpenGL drawing commands are limited to those that generate simple geometric primitives (points, lines, and polygons), the OpenGL Utility Toolkit (GLUT) has been created to aid in the development of more complicated three-dimensional objects such as a sphere, a torus, and even a teapot. GLUT may not be satisfactory for full-featured OpenGL applications, but it is a useful starting point for learning OpenGL.

GLUT is designed to fill the need for a window system independent programming interface for OpenGL programs. The interface is designed to be simple yet still meet the needs of useful OpenGL programs. Removing window system operations from OpenGL is a sound decision because it allows the OpenGL graphics system to be retargeted to various systems including powerful but expensive graphics workstations as well as mass-production graphics systems like video games, set-top boxes for interactive television, and PCs.

The GLUT application-programming interface (API) requires very few routines to display a graphics scene rendered using OpenGL. The GLUT routines also take relatively few parameters.

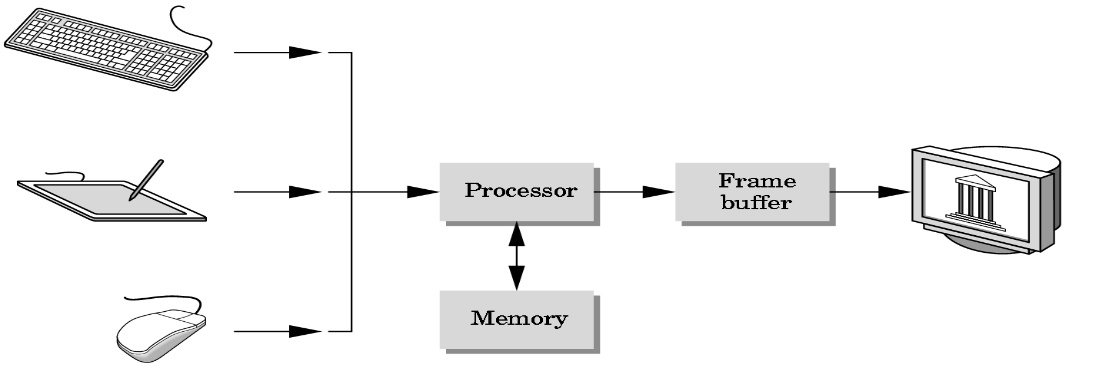
**1.1 COMPUTER GRAPHICS**

The term computer graphics includes almost everything on computers that is not text or sound. Today nearly all computers use some graphics and users expect to control their computer through icons and pictures rather than just by typing. The term Computer Graphics has several meanings:

* The representation and manipulation of pictorial data by a computer
* The various technologies used to create and manipulate such pictorial data
* The images so produced, and
* The sub-field of computer science which studies methods for digitally synthesizing and manipulating visual content

Today computers and computer-generated images touch many aspects of our daily life. Computer imagery is found on television, in newspapers, in weather reports, and during surgical procedures. A well-constructed graph can present complex statistics in a form that is easier to understand and interpret.

Fig 1.1.1 A Graphics System.



**1.2 OPENGL TECHNOLOGY**

OpenGL is strictly defined as “a software interface to graphics hardware.” In essence, it is a 3D graphics and modelling library that is highly portable and very fast. Using OpenGL, you can create elegant and beautiful 3D graphics with exceptional visual quality. The greatest advantage to using OpenGL is that it is orders of magnitude faster than a ray tracer or software-rendering engine. Initially, it used algorithms carefully developed and optimized by Silicon Graphics, Inc. (SGI), an acknowledged world leader in computer graphics and animation. Over time, OpenGL has evolved as other vendors have contributed their expertise and intellectual property to develop high-performance implementations of their own.

OpenGL is a software interface to graphics hardware. This interface consists of about 120 distinct commands, which you use to specify the objects and operations needed to produce interactive three-dimensional applications. OpenGL is designed to work efficiently even if the computer that displays the graphics you create isn't the computer that runs your graphics program. The format for transmitting OpenGL commands (called the protocol) from the client to the server is always the same, so OpenGL programs can work across a network even if the client and server are different kinds of computers. If an OpenGL program isn't running across a network, then there's only one computer, and it is both the client and the server. OpenGL is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms.

OpenGL doesn't provide high-level commands for describing models of three-dimensional objects. Such commands might allow you to specify relatively complicated shapes such as automobiles, parts of the body, airplanes, or molecules. With OpenGL, you must build up your desired model from a small set of geometric primitive - points, lines, and polygons. Computer graphics deals with all aspects of creating images with a computer:

Application: The object is an artist’s rendition of the sun for an animation to be shown in a domed environment (planetarium)

Software: Maya for modelling and rendering but Maya is built on top of OpenGL

Hardware: PC with graphics card for modelling and rendering.

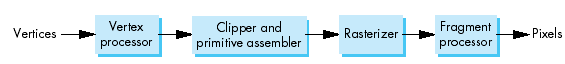


Fig 1.2.1 Graphics pipeline.

1. **PROJECT DESCRIPTION**

The objective of this project is to demonstrate the working of the transformers through animation. The robot is transformed to a vehicle by repositioning the body parts of robot. First we can see a robot in the desert which will automatically be transformed to a vehicle and moves on the road track. The whole project is done using OpenGL.

The output animation of the project gives user a feeling as if he is watching an animated movie. It is important to know about the transformers as many people say transformers will replace everything in the future. By watching the animation, user can know the working of transformers.

I was motivated to for this project after I watched the movie ‘Transformers’. I always wonder if I have a small machine which can help me on all my works and that was transformers.

**3. DESIGN CONSTRAINTS**

A major hurdle during the design of the project was displaying the movement of the elements. Since OpenGL contains a large number of API’s and built in functions, we were able to clear this hurdle. The hardware and software constraints are:

**3.1 HARDWARE CONSTRAINTS**

There are no rigorous restrictions on machine configuration. The editor should be capable of working on all machines capable of supporting recent versions of Microsoft Windows.

* Processor : Pentium PC
* RAM : 1 GB
* Hard Disk : 20 GB (approx)
* Display : VGA Color Monitor

**3.2 SOFTWARE CONSTRAINTS**

* Operating System : Windows XP/Vista/7/8
* Language : C/C++
* Compiler : Microsoft Visual Studio 2010
* Must support OpenGL

**4.ARCHITECTURE**

The main, the display, the timer and the calculate functions are all defined in proj.cpp. The flow of control is shown below.

***Main ( )***

***DISPLAY()***

***Timer(int n)***

***Calc()***

***DISPLAY()***

***START***

Fig 6.1 A Project Architecture

**5.CODE IMPLEMENTATION**

The implementation of the project is done using Visual Studio. The program execution starts from main function. The main function calls the call back display. It also calls the timer function recursively after every twenty milliseconds, calculates the transformation values and then calls the display function again and again. In this way it displays the animation.

#include<windows.h>

#include<GL/glut.h>

float yll=36,ylu=130,yln=80,yrh=320,ye=328,yre=325;

float yrf=348,yrnc=295,rlh=492,rrh=708,rla=495;

float rra=705,wty=45;

float sx=1,sy=1,sz=1,fsx=1,fsy=1,fsz=1,tx=0,ty=0,tz=0;

int tpos=640,ftx=0,fty=0,ftz=0,rxa=0,rya=0,flag=0;

int vflag=0,fflag=0,vtx=520,allover=0,i=0;

void tree()

{

//Trunk

glColor3f(0.325,0.208,0.039);

glScalef(1.5,0.3,0.3);

glTranslatef(0,-10,0);

glutSolidSphere(10,18,20);

glTranslatef(0,10,0);

glScalef(1/1.5,1/0.3,1/0.3);

glutSolidCube(12);

//Tree

glColor3f(0,1,0);

for(int c=-5;c<=10;c+=5)

{

glTranslatef(c,25,0);

glScalef(1,0.8,1);

glutSolidSphere(20,20,20);

glScalef(1,1/0.8,1);

glTranslatef(-c,-25,0);

}

}

void clouds()

{

//Clouds

glColor3f(0.8,0.8,1);

glTranslatef(0,0,0);

glScalef(1.6,1,1);

glutSolidSphere(30,25,25);

glScalef(1/1.6,1,1);

glTranslatef(0,0,0);

glTranslatef(0,0,40);

glScalef(1.6,1,1);

glutSolidSphere(30,25,25);

glScalef(1/1.6,1,1);

glTranslatef(0,0,-40);

glTranslatef(0,0,-40);

glScalef(1.6,1,1);

glutSolidSphere(30,25,25);

glScalef(1/1.6,1,1);

glTranslatef(0,0,40);

glTranslatef(40,0,0);

glScalef(1.6,1,1);

glutSolidSphere(30,25,25);

glScalef(1/1.6,1,1);

glTranslatef(-40,0,0);

glTranslatef(-40,0,0);

glScalef(1.6,1,1);

glutSolidSphere(30,25,25);

glScalef(1/1.6,1,1);

glTranslatef(40,0,0);

}

void background()

{

//Floor

glBegin(GL\_QUADS);

glColor3f(0.6,0.4,0.2);

glVertex3i(-2400,0,0);

glVertex3i(3600,0,0);

glVertex3i(3600,0,500);

glVertex3i(-2400,0,500);

glColor3f(0.4,0.4,0.4);

glVertex3i(-1200,120,0);

glVertex3i(2480,120,0);

glVertex3i(2480,160,0);

glVertex3i(-1200,160,0);

//Sky

glColor3f(0.4,0.6,0.8);

glVertex3i(-2400,0,500);

glVertex3i(3600,0,500);

glVertex3i(3600,1200,500);

glVertex3i(-2400,1200,500);

glEnd();

for(int c=-400;c<=1500;c+=300)

{

//Trees on Right Side

glTranslatef(c,170,0);

tree();

glTranslatef(-c,-170,0);

}

glTranslatef(200,560,300);

clouds();

glTranslatef(-200,-560,-300);

glTranslatef(0,560,10);

clouds();

glTranslatef(0,-560,-10);

glTranslatef(1200,560,50);

clouds();

glTranslatef(-1200,-560,-50);

glTranslatef(600,560,400);

clouds();

glTranslatef(-600,-560,-400);

glTranslatef(550,560,50);

clouds();

glTranslatef(-550,-560,-50);

glTranslatef(-1000,560,500);

clouds();

glTranslatef(1000,-560,-500);

glTranslatef(4000,560,1200);

clouds();

glTranslatef(-4000,-560,-1200);

glTranslatef(3750,560,1200);

clouds();

glTranslatef(-3750,-560,-1200);

glTranslatef(1100,480,300);

clouds();

glTranslatef(-1100,-480,-300);

glTranslatef(-2800,560,800);

clouds();

glTranslatef(2800,-560,-800);

}

void robot()

{

glRotatef(rxa,1,0,0);

glTranslatef(tx,ty,tz);

glScalef(sx,sy,sz);

//Robot Head

glColor3f(0.7,0.7,0.7);

glTranslatef(600,yrh,200);

glScalef(1.8,1,0.2);

glutSolidSphere(25,50,50);

glScalef(1/1.8,1,1/0.2);

glTranslatef(-600,-yrh,-200);

//Left Ear

glColor3f(0.2,0.2,0.7);

glTranslatef(564,ye,200);

glScalef(0.15,1,0.2);

glutSolidCube(50);

glScalef(1/0.15,1,1/0.2);

glTranslatef(-564,-ye,-200);

//Right Ear

//glColor3f(0,0,1);

glTranslatef(636,ye,200);

glScalef(0.15,1,0.2);

glutSolidCube(50);

glScalef(1/0.15,1,1/0.2);

glTranslatef(-636,-ye,-200);

//Robot Eye

//glColor3f(0,0,1);

glTranslatef(600,yre,200);

glScalef(1,0.10,0.15);

glutSolidCube(72);

glScalef(1,1/0.10,1/0.15);

glTranslatef(-600,-yre,-200);

//Robot Forehead

//glColor3f(0,0,1);

glTranslatef(600,yrf,200);

glScalef(1.7,1,0.1);

glutSolidCube(15);

glScalef(1/1.7,1,1/0.1);

glTranslatef(-600,-yrf,-200);

//Robot Neck

glColor3f(0.7,0.7,0.7);

glTranslatef(600,yrnc,200);

glScalef(3.5,1,0.1);

glutSolidCube(15);

glScalef(1/3.5,1,1/0.1);

glTranslatef(-600,-yrnc,-200);

//Robot Left Arm

glColor3f(0.2,0.2,0.2);

glTranslatef(rla,280,200);

glScalef(1.2,0.4,0.2);

glutSolidSphere(25,50,50);

glScalef(1/1.2,1/0.4,1/0.2);

glTranslatef(-rla,-280,-200);

//Robot Right Arm

glColor3f(0.2,0.2,0.2);

glTranslatef(rra,280,200);

glScalef(1.2,0.4,0.2);

glutSolidSphere(25,50,50);

glScalef(1/1.2,1/0.4,1/0.2);

glTranslatef(-rra,-280,-200);

//Robot Left Knee

glColor3f(0.2,0.2,0.2);

glTranslatef(545,yln,200);

glScalef(1.3,0.5,0.2);

glutSolidSphere(25,50,50);

glScalef(1/1.3,1/0.5,1/0.2);

glTranslatef(-545,-yln,-200);

//Robot Right Knee

glColor3f(0.2,0.2,0.2);

glTranslatef(650,yln,200);

glScalef(1.3,0.5,0.2);

glutSolidSphere(25,50,50);

glScalef(1/1.3,1/0.5,1/0.2);

glTranslatef(-650,-yln,-200);

glColor3f(1,0.639,0.102);

glTranslatef(600,155,200);

glScalef(1,0.45,0.2);

glutSolidSphere(60,5,5);

glScalef(1,1/0.45,1/0.2);

glTranslatef(-600,-155,-200);

//Robot Body

glColor3f(0.2,0.2,0.7);

glTranslatef(600,228,200);

glScalef(1.3,1,0.6);

glutSolidCube(120);

glScalef(1/1.3,1,1/0.6);

glTranslatef(-600,-228,-200);

//Robot Left Leg Part 1

//glColor3f(0,0,0.9);

glTranslatef(550,ylu,200);

glScalef(0.8,1.2,0.2);

glutSolidCube(60);

glScalef(1/0.8,1/1.2,1/0.2);

glTranslatef(-550,-ylu,-200);

//Robot Right Leg Part 1

//glColor3f(0,0,0.9);

glTranslatef(650,ylu,200);

glScalef(0.8,1.2,0.2);

glutSolidCube(60);

glScalef(1/0.8,1/1.2,1/0.2);

glTranslatef(-650,-ylu,-200);

//Robot Left Leg Part 2

//glColor3f(0,0,0.9);

glTranslatef(550,yll,200);

glScalef(1,1.2,0.2);

glutSolidCube(60);

glScalef(1,1/1.2,1/0.2);

glTranslatef(-550,-yll,-200);

//Robot Right Leg Part 2

//glColor3f(0,0,0.9);

glTranslatef(650,yll,200);

glScalef(1,1.2,0.2);

glutSolidCube(60);

glScalef(1,1/1.2,1/0.2);

glTranslatef(-650,-yll,-200);

//Robot Left Hand

//glColor3f(0,0,0.9);

glTranslatef(rlh,220,200);

glScalef(0.8,1.8,0.2);

glutSolidCube(60);

glScalef(1/0.8,1/1.8,1/0.2);

glTranslatef(-rlh,-220,-200);

//Robot Right Hand

//glColor3f(0,0,0.9);

glTranslatef(rrh,220,200);

glScalef(0.8,1.8,0.2);

glutSolidCube(60);

glScalef(1/0.8,1/1.8,1/0.2);

glTranslatef(-rrh,-220,-200);

glScalef(1/sx,1/sy,1/sz);

glTranslatef(-tx,-ty,-tz);

glRotatef(-rxa,1,0,0);

}

void trans()

{

if (yll<130)

{

yll+=1;

ylu+=1;

yln+=1;

}

else if(yrh>240)

{

yrh-=1;

yre-=1;

yrf-=1;

ye-=1;

yrnc-=1;

}

else if(yll<220)

{

yll+=1;

}

else if(rlh<=560)

{

rlh+=1;

rrh-=1;

rla+=0.9;

rra-=0.9;

}

else if(rxa<=90)

{

rxa+=1;

tz-=2;

ty+=1.2;

}

else

{

flag=1;

rxa=0;

sx=1.3;

sy=0.5;

sz=1;

}

}

void vehicle()

{

glTranslatef(ftx,fty,ftz);

glScalef(fsx,fsy,fsz);

if(vflag==1)

{

//Vehicle Hood

glColor3f(1,0.639,0.102);

glBegin(GL\_QUADS);

glVertex3i(vtx,35,200);

glVertex3i(vtx+60,35,200);

glVertex3i(vtx+60,90,200);

glVertex3i(vtx,58,200);

glEnd();

//Front Wheel

glColor3f(0.2,0.2,0.2);

glTranslatef(540,wty,200);

glScalef(1.7,1,0.2);

glutSolidSphere(12.5,50,50);

glScalef(1/1.7,1,1/0.2);

glTranslatef(-540,-wty,-200);

//Rear Wheel

glColor3f(0.2,0.2,0.2);

glTranslatef(655,wty,200);

glScalef(1.7,1,0.2);

glutSolidSphere(12.5,50,50);

glScalef(1/1.7,1,1/0.2);

glTranslatef(-655,-wty,-200);

}

glColor3f(0.2,0.2,0.7);

glTranslatef(600,60,200);

glScalef(sx,sy,sz);

glRotatef(rya,0,1,0);

glutSolidCube(120);

glRotatef(-rya,0,1,0);

glScalef(1/sx,1/sy,1/sz);

glTranslatef(-600,-60,-200);

glScalef(1/fsx,1/fsy,1/fsz);

glTranslatef(-ftx,-fty,-ftz);

}

void calc()

{

if(rya<=90)

{

rya+=1;

sx+=0.002;

sz-=0.01;

}

else if(fsx<=2.5)

{

fsx+=0.05;

fsy+=0.05;

fsz+=0.05;

ftx-=20;

fty-=10;

}

else

{

vflag=1;

}

if(vflag==1&&vtx>=465)

{

vtx-=1;

}

else if(vtx<465&&wty>=30)

{

wty-=0.5;

}

else if(wty<30)

{

allover=1;

}

if(allover==1)

{

if(i<=540)

{

ftx-=5;

i++;

}

else if(i>540)

fflag=1;

}

}

void display()

{

char pr[20]="OpenGL Project:";

char name[20]="By Mahesh Bhandari";

char title[15]="TRANSFORMERS";

glClear(GL\_COLOR\_BUFFER\_BIT|GL\_DEPTH\_BUFFER\_BIT);

background();

if(tpos>550)

{

tpos-=0.5;

}

glColor3f(0,0,0.502);

glRasterPos2f(1000,tpos);

for(int j=0;j<strlen(pr);j++)

{

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_24, pr[j]);

}

glRasterPos2f(1150,tpos-24);

for(int j=0;j<strlen(name);j++)

{

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_24, name[j]);

}

glRasterPos2f(460,tpos-50);

for(int j=0;j<strlen(title);j++)

{

glutBitmapCharacter(GLUT\_BITMAP\_TIMES\_ROMAN\_24, title[j]);

}

if(flag==0)

{

trans();

robot();

}

else if(flag==1)

{

calc();

vehicle();

}

//Draw Trees alongside the road

for(int c=-400;c<=1500;c+=300)

{

//Trees on Left Side

glTranslatef(c,110,0);

tree();

glTranslatef(-c,-110,0);

}

glFlush();

glutSwapBuffers();

}

void changeSize(int w, int h)

{

if(h == 0)

h = 1;

float ratio = 1.0\* w / h;

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glViewport(0, 0, w, h);

gluPerspective(78,ratio,1,500);

glMatrixMode(GL\_MODELVIEW);

}

void Timer(int value)

{

if(fflag==0)

{

glutTimerFunc(20,Timer,0);

glutPostRedisplay();

}

}

void myinit()

{

glClearColor(1.0,1.0,1.0,1.0);

glOrtho(0,1200,0,640,0,500);

glutTimerFunc(2000,Timer,0);

}

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

{

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_DOUBLE|GLUT\_RGB|GLUT\_DEPTH);

glutInitWindowPosition(0,0);

glutInitWindowSize(1200,640);

glutCreateWindow("OpenGL Project: Transformers");

glutReshapeFunc(changeSize);

glutDisplayFunc(display);

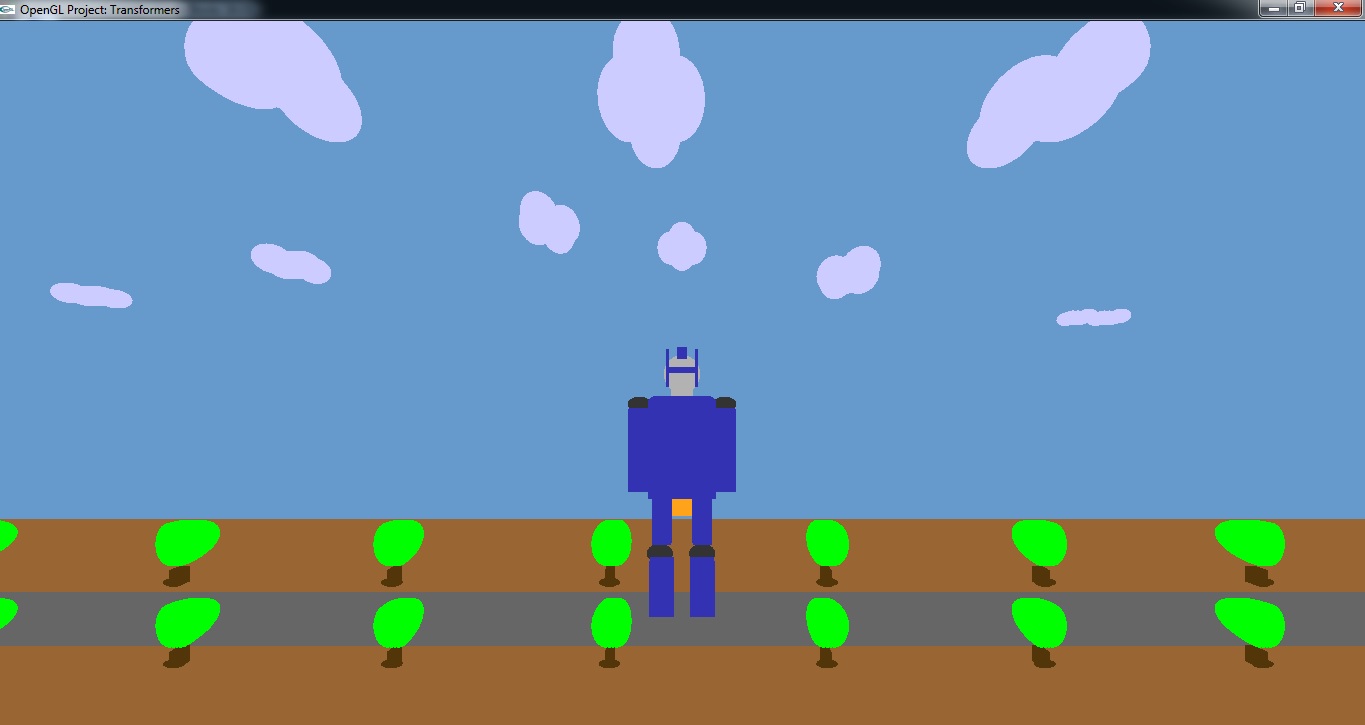
myinit();

trans();

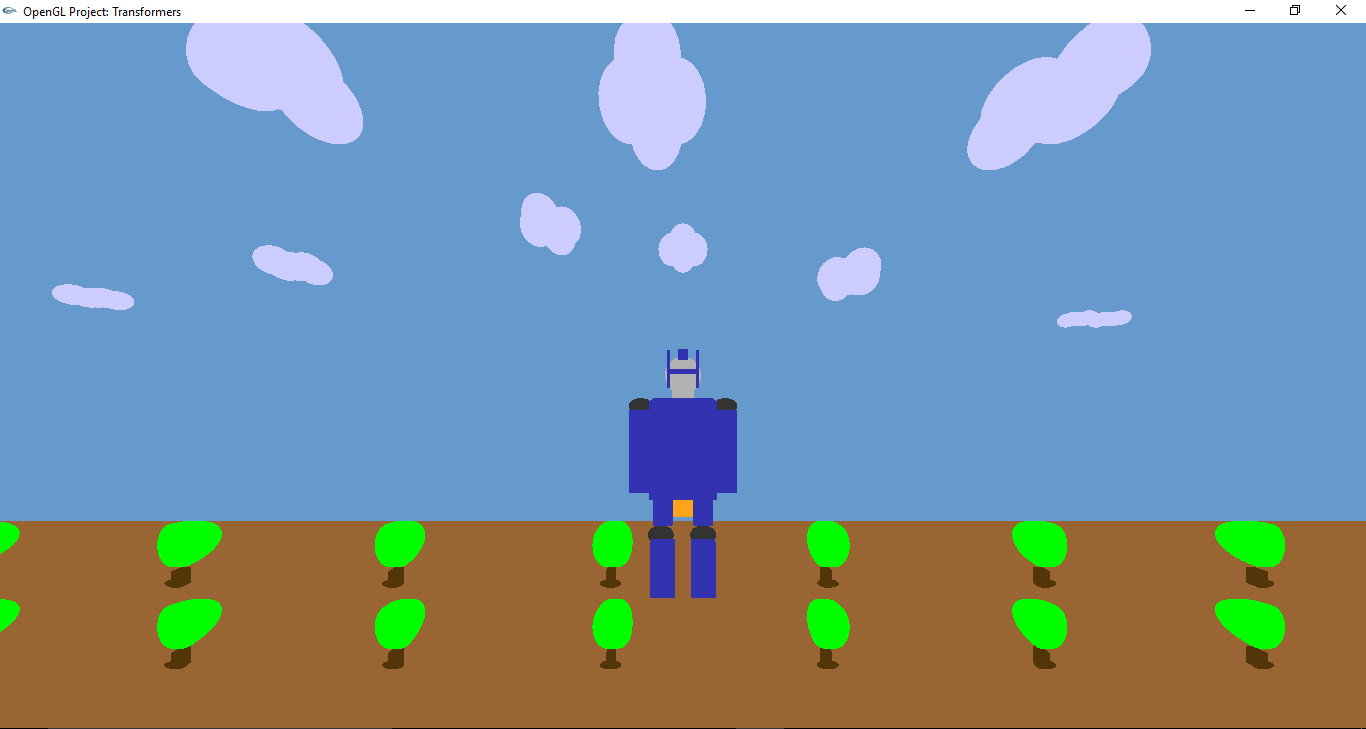
glutMainLoop();

}

1. **SCREEN SNAPSHOTS**
   1. **Initial screen:**

****

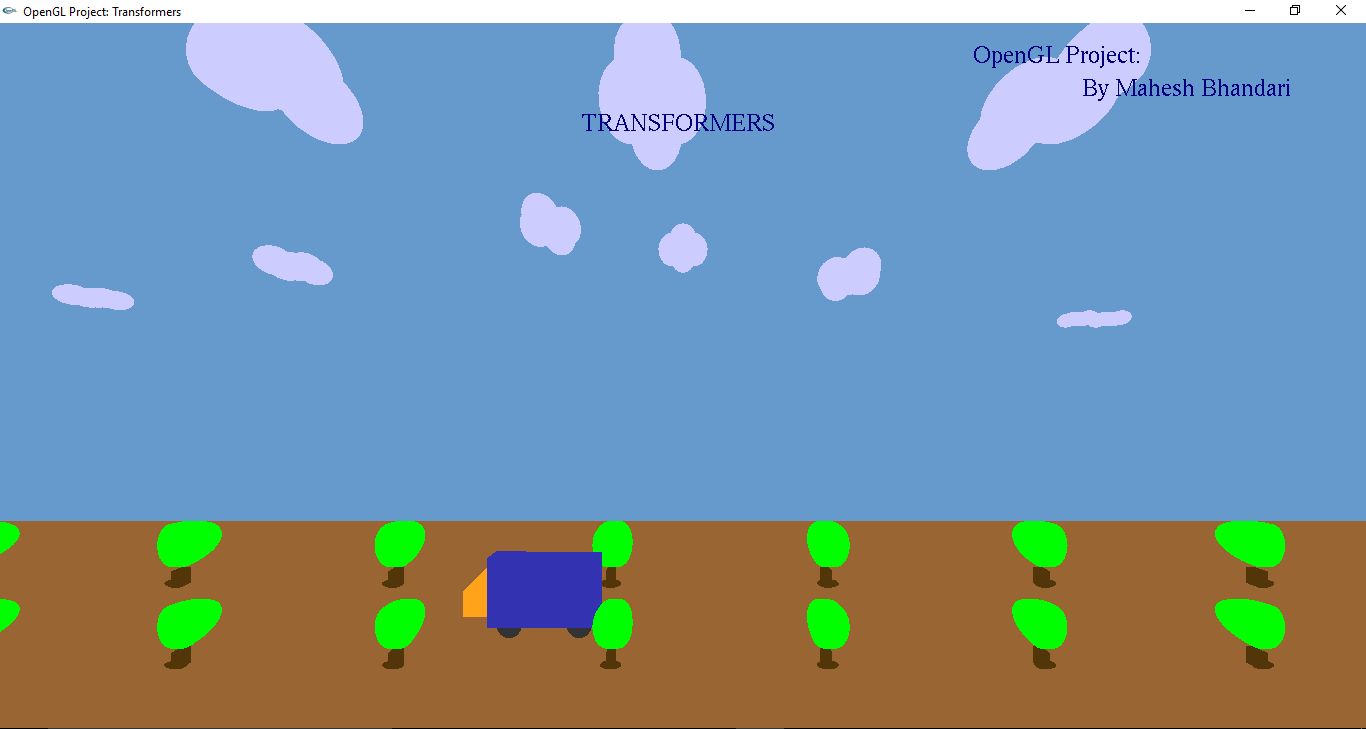
* 1. **The robot starts to transform:**

****

* 1. **The vehicle rotates to look like a vehicle**



* 1. **The vehicle moves along the way**

****

**7. FUTURE ENHANCEMENTS**

The following are some of the future scopes:

* The project can be developed in by using the less codes and the mathematical functions also can be used for calculating the values.
* It can also be used as an animation video to demonstrate the working of transformers.
* The output screen can be recorded and we can make a short animated video.
* The animation will look much better if we show the movement of both robot and the vehicle.

**8.CONCLUSION**

The project entitled “**TRANSFORMERS**” is developed with the best of our effort within a limited period of time. Due to time constraint and out of coverage in the syllabus, the complexity of this project has not been fully furnished to attain perfection.

The functions in the package have been implemented and tested to ensure the efficiency of operation and they were found to be quite satisfactory.

After the completion of the development and study of the project I have come to the conclusion that computer graphics using OpenGL can be used to developing a much better and complex application that include 2D and 3D image processing.

Lastly and importantly I thank our teachers who were a great source of inspiration clubbed with technical help , my seniors who guided us throughout the course of project, my college for giving us all the facilities we need and my parents who have supported me in all respects.

**General Constraints**

* As the code is written in Microsoft visual studio, which is supported by only Graphics Interface Operating System and not by Command Interface Operating Systems like MS-DOS etc, therefore the base platform should be kept in mind before code compilation.
* Needless to say the program should be robust and fast.
* It is assumed that the standard output device, namely the monitor, supports colors.

**Assumptions and Dependencies**

* The user’s system is required to have the C++ compiler of the appropriate versions.

**9.APPENDIX**

This appendix contains the detail description of the functions that are used in the source code. These functions can be explained as follows,

***OpenGL Functions***

**1. Specifying Simple Geometry**

voidglVertex[234][sifd](TYPE coordinate, TYPE coordinate,..)

voidglVertex[234][sifd]v(TYPE \*coordinates)

Specifies the position of a vertex in 2, 3, or 4 dimensions. The coordinates can be specified as shorts s, ints i, floats f or doubles d. If v is present, the argument is a pointer to an array containing the coordinates.

voidglBegin(glEnum mode)

Initiates a new primitive of type mode and stars the collection of vertices. Values of mode include GL\_POINTS, GL\_LINES, and GL\_POLYGON.

voidglEnd( )

Terminates a list of vertices.

**2. Attributes**

void glColor3[b i f d ub us ui](TYPE r, TYPE g, TYPE b)

void glColor4[b i f d ub us ui](TYPE r, TYPE g, TYPE b, TYPE a)

voidglColor[34][b i f d ub us ui]v(TYPE \*color)

Sets the present RGB (or RGBA) colors. Valid types are byte (b),int (i), float (f), double (d), unsigned byte (ub), unsigned short (us), and unsigned int (ui).

voidglClearColor(GLclampf r, GLclampf g, GLclampf b, GLclampf a)

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

voidglLineWidth(GLfloat size)

Sets the line width attribute in pixels.

**3. Working With The Window System**

Void glFlush( )

forces any buffered OpenGL commands to execute.

voidglutInit(int \*argc, char \*\*argv)

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

intglutCreateWindow(char \*title)

Creates a window on the display. The string title can be used to label the window.

voidglutInitDisplayMode(unsigned int mode)

Requests a display with the properties in mode. The value of mode is determined by the logical OR of options including the color model and buffering.

voidglutInitWindowSize(int width, int height)

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

voidglutInitWindowPosition(int x, int y)

Specifies the initial position of the top-left corner of the window in pixels.

voidglutMainLoop()

Cause the program to enter an event-processing loop. It should be the last statement in main.

voidglutDisplayFunc(void (\*func)(void))

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

voidglutSwapBuffers()

Swaps the front and back buffers.

**6. Defining Descrete Primitives**

Void glutBitmapCharacter(void\* font, int char)

Renders the character with ASCII code at the current raster position using the raster font given by font.

**7. Transformations**

Void glMatrixMode(GLenum mode)

specifies which matrix will be affected by subsequent transformations.

void glLoadIdentity()

Sets the current transformation matrix to an identity matrix.

**8. Viewing**

void glOrtho(GLdoble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far)

Defines an orthographic viewing volume with all parameters measured from the centre of the projection plane.

void gluPerspective(GLdoble fovy, GLdouble aspect, GLdouble znear, GLdouble zfar)

It is used for viewing the object in 3D with certain angle. It is a bit realistic than parallel viewing.

# 10.BIBLIOGRAPHY:

A number of different sources were thoroughly searched for reference material. The major source being of our **library books** and the **internet** (numerous websites and pages including those of reputed universities and companies). Paper presentations available on the net were immense help. Previous works and ideas by seniors have been incorporated to produce this editor. Help pages provided excellent understanding of the concepts associated with C- programming.

**Books referred:**

1. **Interactive Computer Graphics** **A Top-Down Approach with OpenGL** -Edward Angel, 5th Edition, Addison-Wesley, 2008.
2. **Computer Graphics Using OpenGL**– F.S. Hill,Jr.  2nd Edition, Pearson Education, 2001.
3. **Computer Graphics**– James D Foley, Andries Van Dam, Steven K Feiner, John F Hughes,  Addison-wesley 1997.
4. **Computer Graphics - OpenGL** **Version**– Donald Hearn and Pauline Baker, 2nd Edition, Pearson Education, 2003.
5. **Compter Graphics Using OpenGL**– F.S. Hill,Jr. – 2nd Edition, Pearson education, 2001.
6. **Interactive Computer Graphics A Top-Down Approach with OpenGL Edward Angel**– 2ndEdition, Addison-Wesley, 2000.

**WEBSITES:**

* <http://www.opengl.org/sdk/docs/man/>Online man pages.
* <http://www.cs.rutgers.edu/~decarlo/428/glman.html>Online man pages.
* <http://nehe.gamedev.net> OpenGL tutorials.