**CHAPTER 1**

**INTRODUCTION**

Computer graphics are graphics created using computers and more generally, the representation and manipulation of image 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 a profound impact on many types of media and have revolutionized animation, movies and the video game industry.

* 1. **Overview of Computer Graphics**

The term computer graphics has been used in a broad sense to describe “almost everything on computers that is not text or sound. It is one of the most powerful and interesting facts of computer. There is a lot that you can do apart from drawing figures of various shapes.

Today, computers and computer-generated images touch many aspects of daily life. Computer image is found on television, in newspapers, for example in weather reports, in all kinds of medical investigation and surgical procedures. A well-constructed graph can present complex statistics in a form that is easier to understand and interpret. In the media such graphs are used to illustrate papers, reports, and other presentation material.

Many powerful tools have been developed to visualize data. Computer generated imagery can be categorized into several different types: 2D , 3D , 5D, and animated graphics. As technology has improved, 3D computer graphics have become more common, but 2D computer graphics are still widely used. Computer graphics has emerged as a sub-field of computer science which studies methods for digitally synthesizing and manipulating visual content. Over the past decade, other specialized fields have been developed like information visualization, and scientific visualization more concerned with “the visualization of three dimensional phenomena” (architectural, meteorological, medical, biological, etc.), where emphasis is on realistic renderings of volumes, surfaces, illumination sources, and so forth, perhaps with a dynamic (time component).

* 1. **History of Computer Graphics**

Computer graphics was first created as a visualization tool for scientists and engineers in government and corporate research centers such as Bell Labs and Boeing in the 1950s. Later the tools would be developed at Universities in the 6os and 70s at places such as Ohio State University, MIT, University of Utah Cornshell, North Carolina and the New York Institute of technology. The early breakthroughs that took place in academic centers continued at research centers such as the famous Xerox PARC in the 1970’s. These efforts broke first into broadcast video graphics and then major motion pictures in the late 70’s and early 1980’s. Computer graphic research continues today around the production companies. Companies such as George Luca’s Industrial light and magic are constantly redefining the cutting edge of computer graphic technology in order to present the world with a new synthetic digital reality.

* 1. **Applications of Computer Graphics**

Nowadays Computer Graphics used in almost all the areas ranges from science, engineering, medicine, business, industry government, art, entertainment, education and training.

**1.3.1 CG in the field of CAD**

Computer Aided design methods are routinely used in the design of buildings, automobiles, aircraft, watercraft, spacecraft computers, textiles and many other applications.

* + 1. **CG in Presentation Graphics**

Another major application area presentation graphics used to produce illustrations for reports or generate slides. Presentation graphics is commonly used to summarize financial, statistical, mathematical, scientific data for research reports and other types of reports.2D and 3D bar chart to illustrate some mathematical or statistical report.

**1.3.3 CG in computer Art**

CG methods are widely used in both fine art and commercial art applications. Artists use a variety of computer methods including special purpose hardware, artist’s paintbrush program, other pain packages, desktop packages, mathematics packages, animation packages that provide facility for designing object motion. Ex: cartoons decision is an example of computer art which uses CG.

* + 1. **Image Processing**

Concerned with fundamentally different operations. In CG a computer is used to create a picture. Image processing on the other hand applies techniques to modify existing pictures such as photo scans, TV scans.

* + 1. **User Interface**

It is a common for software packages to provide a graphical interface . A major component of a graphical interface is a window manager that allows a user to display multiple window area. Interface also displays menus, icons for fast selection and processing.

* + 1. **Education and Training**

Computer generated models of physical, financial, economic system often acts as education aids. For some training application special system are designed. Ex: specialized system for simulator for practice sessions or training of ship captain, aircraft pilots and traffic control.

* 1. **Statement of the Project**

To design and implement the Balloon Blast game using OpenGL. This game is enriched with OpenGL functions which is learnt from the classes of Computer Graphics.

* 1. **Objectives**

To implement the concepts of Computer Graphics we have learnt.

* 1. **Organization of the Report**

The next chapter 2 deals with the introduction to OpenGL. The chapter 3 gives the concept of this project. The chapter 4 describes the design and implementation of the project work, followed by presentation of few output snapshots. The chapter 6 provides the conclusion and also mentions the future scope. At the end the references used for the project are listed.

* 1. **Hardware Requirements**

This project requires a hardware requirement of a processor of speed 1.0GHz, 512MB or above RAM capacity, a graphic card, a Keyboard and a free space minimum of 500KB on hard disk.

* Intel®Pentium or AMD
* 1GB RAM
* 80GB HDD
* Mouse
* QWERTY Keyboard
* Standard VGA Monitor
  1. **Software Requirements**
* Operating System – GNU/LINUX
* OpenGL Graphics Library files
* Language Tool – C/C++
  1. **Technology Used**

Here graphical implementation is done using OpenGL graphical package. C/C++ programming language is used for development of this project.

**CHAPTER 2**

**INTRODUCTION TO OPENGL**

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.

**2.1 OpenGL Fundamentals**

"OpenGL Fundamentals" briefly explains basic OpenGL concepts, such as what a graphic primitive is and how OpenGL implements a client-server execution model.

**2.1.1 Primitive and Commands**

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.

Primitives are defined by a group of one or more vertices. A vertex defines a point, an endpoint of a line, or a corner of a polygon where two edges meet. Data (consisting of vertex coordinates, colors, normals, texture coordinates, and edge flags) is associated with a vertex, and each vertex and its associated data are processed independently, in order, and in the same way. The only exception to this rule is if the group of vertices must be clipped so that a particular primitive fits within a specified region; in this case, vertex data may be modified and new vertices created. The type of clipping depends on which primitive the group of vertices represents.

Commands are always processed in the order in which they are received, although there may be an indeterminate delay before a command takes effect. This means that each primitive is drawn completely before any subsequent command takes effect. It also means that state-querying commands return data that's consistent with complete execution of all previously issued OpenGL commands.

**2.1.2 Procedural versus Descriptive**

OpenGL provides you with fairly direct control over the fundamental operations of two- and three-dimensional graphics. This includes specification of such parameters as transformation matrices, lighting equation coefficients, antialiasing methods, and pixel update operators. However, it doesn't provide you with a means for describing or modeling complex geometric objects. Thus, the OpenGL commands you issue specify how a certain result should be produced rather than what exactly that result should look like. That is, OpenGL is fundamentally procedural rather than descriptive. Because of this procedural nature, it helps to know how OpenGL works—the order in which it carries out its operations, for example—in order to fully understand how to use it.

**2.1.3 Execution Model**

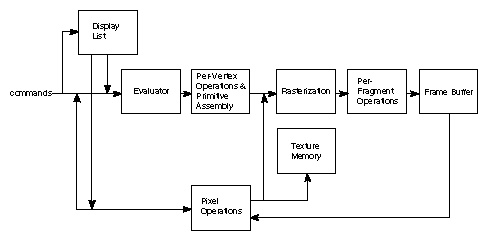
The model for interpretation of OpenGL commands is client-server. An application issues commands, which are interpreted and processed by OpenGL. The server may or may not operate on the same computer as the client. In this sense, OpenGL is network-transparent. A server can maintain several GL contexts, each of which is an encapsulated GL state. A client can connect to any one of these contexts. The required network protocol can be implemented by augmenting an already existing protocol or by using an independent protocol. No OpenGL commands are provided for obtaining user input.

The effects of OpenGL commands on the frame buffer are ultimately controlled by the window system that allocates frame buffer resources. The window system determines which portions of the frame buffer OpenGL may access at any given time and communicates to OpenGL how those portions are structured. Therefore, there are no OpenGL commands to configure the frame buffer or initialize OpenGL. Frame buffer configuration is done outside of OpenGL in conjunction with the window system; OpenGL initialization takes place when the window system allocates a window for OpenGL rendering. (GLX, the X extension of the OpenGL interface, provides these capabilities).

**2.1.4 Basic OpenGL Operation**

The figure shown below gives an abstract, high-level block diagram of how OpenGL processes data.

In the diagram, commands enter from the left and proceed through what can be thought of as a processing pipeline. Some commands specify geometric objects to be drawn, and others control how the objects are handled during the various processing stages.



**Fig:OpenGL Block Diagram**

As shown by the first block in the diagram, rather than having all commands proceed immediately through the pipeline, you can choose to accumulate some of them in a display list for processing at a later time.

The evaluator stage of processing provides an efficient means for approximating curve and surface geometry by evaluating polynomial commands of input values. During the next stage, per-vertex operations and primitive assembly, OpenGL processes geometric primitives—points, line segments, and polygons, all of which are described by vertices. Vertices are transformed and lit, and primitives are clipped to the viewport in preparation for the next stage.

Rasterization produces a series of frame buffer addresses and associated values using a two-dimensional description of a point, line segment, or polygon. Each fragment so produced is fed into the last stage, per-fragment operations, which performs the final operations on the data before it's stored as pixels in the frame buffer. These operations include conditional updates to the frame buffer based on incoming and previously stored z-values (for z-buffering) and blending of incoming pixel colors with stored colors, as well as masking and other logical operations on pixel values.

Input data can be in the form of pixels rather than vertices. Such data, which might describe an image for use in texture mapping, skips the first stage of processing described above and instead is processed as pixels, in the pixel operations stage. The result of this stage is either stored as texture memory, for use in the rasterization stage, or rasterized and the resulting fragments merged into the frame buffer just as if they were generated from geometric data.

All elements of OpenGL state, including the contents of the texture memory and even of the frame buffer, can be obtained by an OpenGL application.

**2.2 OpenGL API**

All Operating System provide a way for applications to use their system resources by using an Application Programming Interface or API. This is usually defined by an extensive list of functions and classes and variables.

It’s not just Operating System that provide APIs. Every programming library (and there are thousands available, both commercial or free) has an API. It’s the job of a programmer to read and understand APIs so that he/she can make the best use of them.

**2.2.1 Display Control Functions**

**glutPostRedisplay(void):**

Request that the image be redrawn**.**

**glutSwapBuffers(void):**

Swap the front and back buffers(used in double buffering).

**glClearColor(GLclampf R, GLclampf G, GLclampf B, GLclampf A):**

Specifies the background color to be used by glClear() , when clearing the buffer.

**glClear(GLbitfield mask):**

Clears one or more of the existing buffers. The mask is logical or (“|”) of any of the following : GL\_COLOR\_BUFFER\_BIT,GL\_DEPTH\_BUFFER\_BIT.

**glFlush():**

OpenGL normally saves or “buffers” drawing commands for greater efficiency. This forces the image to be redrawn , in case the next update will be far in the future.

**2.2.2 OpenGL Basic Drawing**

**glBegin(GLenum mode)…glEnd(void):**

Specify the starting and ending of vertices.

**glVertex\*(…):**

Specify the coordinates of a vertex.

**glNormal\*(...)**:

Specify the surface normal for subsequent vertices. The normal should be of unit length after the modelview transformation is applied. Call glEnable (GL\_NORMALIZE) to have OpenGL do normalization automatically.

**glColor\*(…):**

Specify the color of subsequent vertices . This is normally used if lighting is not enabled . Otherwise , colors are defined by glMaterial\*() , defined below under lighting .

**glLineWidth(GLfloat width):**

Sets the line width for subsequent line drawing .

**2.2.3 GLUT Utilities for Complex Objects**

**glRasterPos\*(...):**

Set the current raster position (in 3D window coordinates) for subsequent glutBitmapCharacter () and other pixel copying operations.

**glutBitmapCharacter (void \*font, int character):**

Draw the given character of the given font at the current raster position and advance the current raster position. Fonts include GLUT\_BITMAP\_9\_BY\_15, GLUT\_BITMAP\_8\_BY\_13, GLUT\_BITMAP\_TIMES\_ROMAN\_10, GLUT\_BITMAP\_TIMES\_ROMAN\_24. When a character is drawn, the raster position is advanced. So to draw a string, initialize the raster position (using glRasterPos\*()) and then call this on each character of the string.

**2.2.4 Transformations and Perspective:**

**glViewport (GLint x, GLint y, GLsizei width, GLsizei height):**

Sets the current viewport, that is, the portion of the window to which everything will be clipped. Typically this is the entire window: glViewport (0, 0, win width, win height), and hence this should be called whenever the window is reshaped.

**glMatrixMode(GLenum mode):**

Set the current matrix mode to one of GL\_MODELVIEW , GL\_PROJECTION, or GL\_TEXTURE. The default is GL\_MODELVIEW.

**glLoadIdentity (void):**

Set the current matrix to the identity.

**glPushMatrix (void):**

Make a copy of the current matrix and push it onto the stack.

**glPopMatrix (void):**

Pop the top of the current matrix.

**glLoadMatrixf (const GLfloat \*M):**

Sets the current matrix to *M*.

**2.2.5 Associated utility libraries**

Several libraries are built on top of or beside OpenGL to provide features not available in OpenGL itself. Libraries such as GLU can be found with most OpenGL implementations, and others such as GLUT and SDL have grown over time and provide rudimentary cross platform windowing and mouse functionality, and if unavailable can easily be downloaded and added to a development environment. Simple graphical user interface functionality can be found in libraries like GLUI or FLTK. Still other libraries like GLAux (OpenGL Auxiliary Library) are deprecated and have been superseded by functionality commonly available in more popular libraries, but code using them still exists, particularly in simple tutorials. Other libraries have been created to provide OpenGL application developers a simple means of managing OpenGL extensions and versioning. Examples of these libraries include GLEW (the OpenGL Extension Wrangler Library) and GLEE (the OpenGL Easy Extension Library).

**GLUT** - The OpenGL Utility Toolkit

GLUT (pronounced like the glut in gluttony) is the OpenGL Utility Toolkit, a window system independent toolkit for writing OpenGL programs. It implements a simple windowing application programming interface (API) for OpenGL. GLUT makes it considerably easier to learn about and explore OpenGL programming. GLUT provides a portable API so you can write a single OpenGL program that works across all PC and workstation OS platforms. GLUT is designed for constructing small to medium sized OpenGL programs. While GLUT is well-suited to learning OpenGL and developing simple OpenGL applications, GLUT is not a full-featured toolkit so large applications requiring sophisticated user interfaces are better off using native window system toolkits. GLUT is simple, easy, and small. The GLUT library has C, C++ (same as C), FORTRAN, and ADA programming bindings. The GLUT source code distribution is portable to nearly all OpenGL implementations and platforms.

**CHAPTER 3**

**PROJECT DESCRIPTION**

Some of the basic requirements for the development of this project are as follows:

**3.1 GRAPHICS FUNCTIONS AND REQUIREMENT**

**Header Files:**

* GL-Used for commands that begin with GL
* String-used for string functions
* Glut-used for glut library functions

**glColor3f (float, float, float):**

This function will set the current drawing color.

**gluOrtho(GLdouble left, GLdouble right, GLdouble bottom,**

**GLdoubletop,GLdouble zNear,Gldouble zfar):**

The ortho function multiplies the current matrix by an orthography matrix.

**glClear( ):**

Takes a single argument that is the bitwise OR of several values indicating which buffer is to be cleared.

**glClearColor ():**

Specifies the red, green, blue, and alpha values used by **glClear** to clear the color buffers.

**glLoadIdentity( ):**

The current matrix with the identity matrix.

**glMatrixMode(mode):**

Sets the current matrix mode, mode can be GL\_MODELVIEW, GL\_PROJECTION or GL\_TEXTURE.

**void glutInit (int \*argc, char\*\*argv):**

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

**void glutInitDisplayMode (unsigned int mode):**

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

**void glutInitWindowSize (int width, int height):**

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

**void glutDisplayFunc (void (\*func) (void)):**

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

**glutPostReDisplay ( )** :

Which requests that the display callback be executed after the current callback returns.

**void glutMainLoop ():**

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

**glFlush():**

Forces and buffers any openGL commands to execute

**glPushMatrix():**

Make a copy of current matrix and push it onto the stack.

**glPopMatrix():**

Pop the top of the current matrix.

**glutBitmapCharacter(void\* font,int character):**

Write a character to output.

**glRasterpos3f(x,y,z):**

To set the coordinates of raster text.

**3.2 USER DEFINED FUNCTIONS**

* **void setfont(void \*font):**

Function to set the font of the text to be displayed.

* **void drawstring(char string[],float x1,float y1,float z1):**

Function to write the characters on the screen.

* **void screen1( ):**

To initialise the welcome screen contents.

* **void screen3( ):**

To initialize the final screen contents.

* **void water( ),void lines( ):**

To set the initial coordinates of the sea water.

* **void base( ),void earth( ):**

To set the initial coordinates of the ground.

* **void pillars( ):**

To set the initial coordinates of the pillars.

* **void bridge( ):**

To set the initial coordinates of the bridge.

* **void track( ):**

To set the initial coordinates of the railway track.

* **void ship( ),void train( ),void aero( ):**

To set the initial coordinates of ship, train and aeroplane respectively.

* **void lighthouse( ),void signal( ):**

To set the initial coordinates of lighthouse & signal.

* **void light( ):**

To set the initial coordinates the signal light and set the colors.

* **void new1( ),void new2( ),void new3( ),void new4( ),void new5( ):**

To control the movements of bridge, ship, train, plane and water.

* **void update( ):**

To update the movement values of bridge, ship, train, plane and water.

* **void mydisplay( ):**

To control the display of multiple screens.

* **void mykeyboard( ):**

To get user input through keyboard.

**3.3 IMPLEMENTATION**

#include<GL/glut.h>

#include<string.h>

int i,flag=0,flag2=0,flagb=1,flags=0,flagt=0,flagp=0,flagw=1,flagx=0;

float a=0.0f , b=0.0f , c=0.0f , m=0.0f , n=0.0f, o=0.0f , p=0.0f , q=0.0f , r=0.0f , x=0.0f , y=0.0f , z=0.0f , a1=0.0 , a2=0.0 , a3=0.0,j;

void \*currentfont;

void setFont(void \*font){

currentfont=font;

}

void drawstring(char string[],float x1,float y1,float z1){

int i , j;

j=strlen(string);

glRasterPos3f(x1,y1,z1);

for (i=0;i<j;i++){

glutBitmapCharacter(currentfont, string[i]);

}

}

void screen1(){

glClearColor(0.5,0.2,0.8,0.0);

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

setFont(GLUT\_BITMAP\_TIMES\_ROMAN\_24);

glColor3f(1.0,1.0,1.0);

char str1[]="COORG INSTITUTE OF TECHNOLOGY";

drawstring(str1,-0.5,0.9,0.0);

glColor3f(0.9,0.9,0.9);

char str2[]="DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING";

drawstring(str2,-0.7,0.8,0.0);

glColor3f(0.8,0.8,0.8);

char str3[]="A MINI PROJECT ON";

drawstring(str3,-0.3,0.7,0.0);

glColor3f(1.0,0.7,0.7);

char str4[]="-\*- VERTICAL LIFT BRIDGE SIMULATION -\*-";

drawstring(str4,-0.65,0.6,0.0);

glColor3f(0.2,0.8,0.5);

char str5[]="BY :";

drawstring(str5,-0.5,0.5,0.0);

glColor3f(1.0,1.0,0.0);

char str6[]="AMAL KURIAKOSE (4CI14CS010)";

drawstring(str6,-0.5,0.4,0.0);

char str7[]="M ARJUN PRADEEP (4CI14CS038)";

drawstring(str7,-0.5,0.3,0.0);

glColor3f(0.2,0.8,0.5);

char str8[]="GUIDES :";

drawstring(str8,-0.5,0.1,0.0);

glColor3f(1.0,1.0,0.0);

char str9[]="Mr . BHARATH (Asst.prof. Dept of CSE)";

drawstring(str9,-0.5,0.0,0.0);

char str10[]="Mr . RAVISHANKAR (Asst.prof. Dept of CSE)";

drawstring(str10,-0.5,-0.1,0.0);

glColor3f(1.0,0.9,1.0);

char str11[]="INSTRUCTIONS ";

drawstring(str11,-0.3,-0.5,0.0);

char str12[]="----------------";

drawstring(str12,-0.35,-0.55,0.0);

char str13[]=" \* \* \* PRESS -ENTER - TO START AND -ESC- TO EXIT \* \* \*";

drawstring(str13,-0.9,-0.7,0.0);

char str14[]="PRESS -S- TO START ANIMATION AND PRESS -T- TO STOP ANIMATION";

drawstring(str14,-1.05,-0.85,0.0);

glFlush();

}

void screen3(){

glClearColor(1.0,0.5,0.0,0.0);

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

setFont(GLUT\_BITMAP\_TIMES\_ROMAN\_24);

glColor3f(0.0,0.0,0.0);

char str1[]=" \* \* \* THANK YOU \* \* \* ";

drawstring(str1,-0.2,0.2,0.0);

char str2[]=" ------------------------- ";

drawstring(str2,-0.3,0.1,0.0);

glFlush();

}

void water(){

glBegin(GL\_QUADS);

glColor3f(0.0,0.4,0.7);

glVertex3f(-5.0,-0.415,5.0);

glVertex3f(5.0,-0.415,5.0);

glVertex3f(5.0,-0.415,-5.0);

glVertex3f(-5.0,-0.415,-5.0);

glEnd();

}

void lines(){

float t1,t2;

glBegin(GL\_LINES);

for(t1=0.0;t1<=10.0;t1+=0.4){

for(t2=0.0;t2<=10.0;t2+=0.4){

glColor3f(0.7,0.7,0.7);

glVertex3f(-5.0+t2,-0.41,-4.5+t1);

glVertex3f(-4.95+t2,-0.41,-4.5+t1);

}

}

glEnd();

}

void base()

{

float i;

for(i=0.0;i<1.0;i+=0.8) {

glBegin(GL\_QUADS);

glColor3f(0.1,0.1,0.1); //front

glVertex3f(-0.5+i,-0.4,0.2);

glVertex3f(-0.3+i,-0.4,0.2);

glVertex3f(-0.3+i,-0.25,0.2);

glVertex3f(-0.5+i,-0.25,0.2);

glColor3f(0.3,0.3,0.3); //left

glVertex3f(-0.5+i,-0.4,0.2);

glVertex3f(-0.5+i,-0.4,-0.2);

glVertex3f(-0.5+i,-0.25,-0.2);

glVertex3f(-0.5+i,-0.25,0.2);

glColor3f(0.1,0.1,0.1); //back

glVertex3f(-0.5+i,-0.4,-0.2);

glVertex3f(-0.3+i,-0.4,-0.2);

glVertex3f(-0.3+i,-0.25,-0.2);

glVertex3f(-0.5+i,-0.25,-0.2);

glColor3f(0.3,0.3,0.3); //right

glVertex3f(-0.3+i,-0.4,0.2);

glVertex3f(-0.3+i,-0.4,-0.2);

glVertex3f(-0.3+i,-0.25,-0.2);

glVertex3f(-0.3+i,-0.25,0.2);

glColor3f(0.5,0.5,0.5); //top

glVertex3f(-0.5+i,-0.25,0.2);

glVertex3f(-0.3+i,-0.25,0.2);

glVertex3f(-0.3+i,-0.25,-0.2);

glVertex3f(-0.5+i,-0.25,-0.2);

glEnd();

}

}

void earth(){

float i;

for(i=0.0;i<4.0;i+=3.5){

glBegin(GL\_QUADS);

glColor3f(0.0,0.6,0.0);

glVertex3f(-3.0+i,-0.4,0.17); //front

glVertex3f(-0.5+i,-0.4,0.17);

glVertex3f(-0.5+i,-0.25,0.17);

glVertex3f(-3.0+i,-0.25,0.17);

glVertex3f(-3.0+i,-0.4,-0.17); //back

glVertex3f(-0.5+i,-0.4,-0.17);

glVertex3f(-0.5+i,-0.25,-0.17);

glVertex3f(-3.0+i,-0.25,-0.17);

glColor3f(0.0,0.5,0.0); //top

glVertex3f(-3.0+i,-0.25,0.17);

glVertex3f(-0.5+i,-0.25,0.17);

glVertex3f(-0.5+i,-0.25,-0.17);

glVertex3f(-3.0+i,-0.25,-0.17);

glEnd();

}

}

void pillars(){

glBegin(GL\_QUADS);

glColor3f(0.1,0.1,0.5); //front

glVertex3f(-0.35,-0.25,0.2);

glVertex3f(-0.31,-0.25,0.2);

glVertex3f(-0.31,0.2,0.2);

glVertex3f(-0.35,0.2,0.2);

glColor3f(0.1,0.1,0.3); //left

glVertex3f(-0.35,-0.25,0.2);

glVertex3f(-0.35,-0.25,0.15);

glVertex3f(-0.35,0.2,0.15);

glVertex3f(-0.35,0.2,0.2);

glColor3f(0.1,0.1,0.5); //back

glVertex3f(-0.35,-0.25,0.15);

glVertex3f(-0.31,-0.25,0.15);

glVertex3f(-0.31,0.2,0.15);

glVertex3f(-0.35,0.2,0.15);

glColor3f(0.1,0.1,0.3); //right

glVertex3f(-0.31,-0.25,0.2);

glVertex3f(-0.31,-0.25,0.15);

glVertex3f(-0.31,0.2,0.15);

glVertex3f(-0.31,0.2,0.2);

glColor3f(0.1,0.1,0.5); //front

glVertex3f(-0.35,-0.25,-0.2);

glVertex3f(-0.31,-0.25,-0.2);

glVertex3f(-0.31,0.2,-0.2);

glVertex3f(-0.35,0.2,-0.2);

glColor3f(0.1,0.1,0.3); //left

glVertex3f(-0.35,-0.25,-0.2);

glVertex3f(-0.35,-0.25,-0.15);

glVertex3f(-0.35,0.2,-0.15);

glVertex3f(-0.35,0.2,-0.2);

glColor3f(0.1,0.1,0.5); //back

glVertex3f(-0.35,-0.25,-0.15);

glVertex3f(-0.31,-0.25,-0.15);

glVertex3f(-0.31,0.2,-0.15);

glVertex3f(-0.35,0.2,-0.15);

glColor3f(0.1,0.1,0.3); //right

glVertex3f(-0.31,-0.25,-0.2);

glVertex3f(-0.31,-0.25,-0.15);

glVertex3f(-0.31,0.2,-0.15);

glVertex3f(-0.31,0.2,-0.2);

glColor3f(0.2,0.2,0.5);//top

glVertex3f(-0.35,0.2,0.3);

glVertex3f(-0.31,0.2,0.3);

glVertex3f(-0.31,0.25,0.3);

glVertex3f(-0.35,0.25,0.3);

glColor3f(0.3,0.3,0.5);

glVertex3f(-0.35,0.2,0.3);

glVertex3f(-0.35,0.2,-0.3);

glVertex3f(-0.35,0.25,-0.3);

glVertex3f(-0.35,0.25,0.3);

glColor3f(0.2,0.2,0.5);

glVertex3f(-0.35,0.2,-0.3);

glVertex3f(-0.31,0.2,-0.3);

glVertex3f(-0.31,0.25,-0.3);

glVertex3f(-0.35,0.25,-0.3);

glColor3f(0.3,0.3,0.5);

glVertex3f(-0.31,0.2,0.3);

glVertex3f(-0.31,0.2,-0.3);

glVertex3f(-0.31,0.25,-0.3);

glVertex3f(-0.31,0.25,0.3);

glColor3f(0.3,0.3,0.6);

glVertex3f(-0.35,0.25,0.3);

glVertex3f(-0.31,0.25,0.3);

glVertex3f(-0.31,0.25,-0.3);

glVertex3f(-0.35,0.25,-0.3);

//top piller

glColor3f(0.1,0.1,0.5); //front

glVertex3f(-0.35,0.25,0.2);

glVertex3f(-0.31,0.25,0.2);

glVertex3f(-0.31,0.35,0.2);

glVertex3f(-0.35,0.35,0.2);

glColor3f(0.1,0.1,0.3); //left

glVertex3f(-0.35,0.25,0.2);

glVertex3f(-0.35,0.25,0.15);

glVertex3f(-0.35,0.35,0.15);

glVertex3f(-0.35,0.35,0.2);

glColor3f(0.1,0.1,0.5); //back

glVertex3f(-0.35,0.25,0.15);

glVertex3f(-0.31,0.25,0.15);

glVertex3f(-0.31,0.35,0.15);

glVertex3f(-0.35,0.35,0.15);

glColor3f(0.1,0.1,0.3); //right

glVertex3f(-0.31,0.25,0.2);

glVertex3f(-0.31,0.25,0.15);

glVertex3f(-0.31,0.35,0.15);

glVertex3f(-0.31,0.35,0.2);

glColor3f(0.1,0.1,0.5); //front

glVertex3f(-0.35,0.25,-0.2);

glVertex3f(-0.31,0.25,-0.2);

glVertex3f(-0.31,0.35,-0.2);

glVertex3f(-0.35,0.35,-0.2);

glColor3f(0.1,0.1,0.3); //left

glVertex3f(-0.35,0.25,-0.2);

glVertex3f(-0.35,0.25,-0.15);

glVertex3f(-0.35,0.35,-0.15);

glVertex3f(-0.35,0.35,-0.2);

glColor3f(0.1,0.1,0.5); //back

glVertex3f(-0.35,0.25,-0.15);

glVertex3f(-0.31,0.25,-0.15);

glVertex3f(-0.31,0.35,-0.15);

glVertex3f(-0.35,0.35,-0.15);

glColor3f(0.1,0.1,0.3); //right

glVertex3f(-0.31,0.25,-0.2);

glVertex3f(-0.31,0.25,-0.15);

glVertex3f(-0.31,0.35,-0.15);

glVertex3f(-0.31,0.35,-0.2);

glEnd();

//right side

glBegin(GL\_QUADS);

glColor3f(0.1,0.1,0.5); //front

glVertex3f(0.35,-0.25,0.2);

glVertex3f(0.31,-0.25,0.2);

glVertex3f(0.31,0.2,0.2);

glVertex3f(0.35,0.2,0.2);

glColor3f(0.1,0.1,0.3); //left

glVertex3f(0.35,-0.25,0.2);

glVertex3f(0.35,-0.25,0.15);

glVertex3f(0.35,0.2,0.15);

glVertex3f(0.35,0.2,0.2);

glColor3f(0.1,0.1,0.5); //back

glVertex3f(0.35,-0.25,0.15);

glVertex3f(0.31,-0.25,0.15);

glVertex3f(0.31,0.2,0.15);

glVertex3f(0.35,0.2,0.15);

glColor3f(0.1,0.1,0.3); //right

glVertex3f(0.31,-0.25,0.2);

glVertex3f(0.31,-0.25,0.15);

glVertex3f(0.31,0.2,0.15);

glVertex3f(0.31,0.2,0.2);

glColor3f(0.1,0.1,0.5); //front

glVertex3f(0.35,-0.25,-0.2);

glVertex3f(0.31,-0.25,-0.2);

glVertex3f(0.31,0.2,-0.2);

glVertex3f(0.35,0.2,-0.2);

glColor3f(0.1,0.1,0.3); //left

glVertex3f(0.35,-0.25,-0.2);

glVertex3f(0.35,-0.25,-0.15);

glVertex3f(0.35,0.2,-0.15);

glVertex3f(0.35,0.2,-0.2);

glColor3f(0.1,0.1,0.5); //back

glVertex3f(0.35,-0.25,-0.15);

glVertex3f(0.31,-0.25,-0.15);

glVertex3f(0.31,0.2,-0.15);

glVertex3f(0.35,0.2,-0.15);

glColor3f(0.1,0.1,0.3); //right

glVertex3f(0.31,-0.25,-0.2);

glVertex3f(0.31,-0.25,-0.15);

glVertex3f(0.31,0.2,-0.15);

glVertex3f(0.31,0.2,-0.2);

glColor3f(0.2,0.2,0.5);//top

glVertex3f(0.35,0.2,0.3);

glVertex3f(0.31,0.2,0.3);

glVertex3f(0.31,0.25,0.3);

glVertex3f(0.35,0.25,0.3);

glColor3f(0.3,0.3,0.5);

glVertex3f(0.35,0.2,0.3);

glVertex3f(0.35,0.2,-0.3);

glVertex3f(0.35,0.25,-0.3);

glVertex3f(0.35,0.25,0.3);

glColor3f(0.2,0.2,0.5);

glVertex3f(0.35,0.2,-0.3);

glVertex3f(0.31,0.2,-0.3);

glVertex3f(0.31,0.25,-0.3);

glVertex3f(0.35,0.25,-0.3);

glColor3f(0.3,0.3,0.5);

glVertex3f(0.31,0.2,0.3);

glVertex3f(0.31,0.2,-0.3);

glVertex3f(0.31,0.25,-0.3);

glVertex3f(0.31,0.25,0.3);

glColor3f(0.3,0.3,0.6);

glVertex3f(0.35,0.25,0.3);

glVertex3f(0.31,0.25,0.3);

glVertex3f(0.31,0.25,-0.3);

glVertex3f(0.35,0.25,-0.3);

//top piller

glColor3f(0.1,0.1,0.5); //front

glVertex3f(0.35,0.25,0.2);

glVertex3f(0.31,0.25,0.2);

glVertex3f(0.31,0.35,0.2);

glVertex3f(0.35,0.35,0.2);

glColor3f(0.1,0.1,0.3); //left

glVertex3f(0.35,0.25,0.2);

glVertex3f(0.35,0.25,0.15);

glVertex3f(0.35,0.35,0.15);

glVertex3f(0.35,0.35,0.2);

glColor3f(0.1,0.1,0.5); //back

glVertex3f(0.35,0.25,0.15);

glVertex3f(0.31,0.25,0.15);

glVertex3f(0.31,0.35,0.15);

glVertex3f(0.35,0.35,0.15);

glColor3f(0.1,0.1,0.3); //right

glVertex3f(0.31,0.25,0.2);

glVertex3f(0.31,0.25,0.15);

glVertex3f(0.31,0.35,0.15);

glVertex3f(0.31,0.35,0.2);

glColor3f(0.1,0.1,0.5); //front

glVertex3f(0.35,0.25,-0.2);

glVertex3f(0.31,0.25,-0.2);

glVertex3f(0.31,0.35,-0.2);

glVertex3f(0.35,0.35,-0.2);

glColor3f(0.1,0.1,0.3); //left

glVertex3f(0.35,0.25,-0.2);

glVertex3f(0.35,0.25,-0.15);

glVertex3f(0.35,0.35,-0.15);

glVertex3f(0.35,0.35,-0.2);

glColor3f(0.1,0.1,0.5); //back

glVertex3f(0.35,0.25,-0.15);

glVertex3f(0.31,0.25,-0.15);

glVertex3f(0.31,0.35,-0.15);

glVertex3f(0.35,0.35,-0.15);

glColor3f(0.1,0.1,0.3); //right

glVertex3f(0.31,0.25,-0.2);

glVertex3f(0.31,0.25,-0.15);

glVertex3f(0.31,0.35,-0.15);

glVertex3f(0.31,0.35,-0.2);

glEnd();

//slops front

glBegin(GL\_QUADS);

glColor3f(0.8,0.6,0.5);

glVertex3f(-0.6,-0.25,0.17);

glVertex3f(-0.5,-0.25,0.17);

glVertex3f(-0.35,0.2,0.17);

glVertex3f(-0.35,0.25,0.17);

//BACK

glVertex3f(-0.6,-0.25,-0.17);

glVertex3f(-0.5,-0.25,-0.17);

glVertex3f(-0.35,0.2,-0.17);

glVertex3f(-0.35,0.25,-0.17);

//right front

glVertex3f(0.6,-0.25,0.17);

glVertex3f(0.5,-0.25,0.17);

glVertex3f(0.35,0.2,0.17);

glVertex3f(0.35,0.25,0.17);

//Back

glVertex3f(0.6,-0.25,-0.17);

glVertex3f(0.5,-0.25,-0.17);

glVertex3f(0.35,0.2,-0.17);

glVertex3f(0.35,0.25,-0.17);

glEnd();

}

void bridge(){

glBegin(GL\_QUADS);

glColor3f(0.1,0.2,0.3);

glVertex3f(-0.3,-0.25,0.15);

glVertex3f(0.3,-0.25,0.15);

glVertex3f(0.3,-0.23,0.15);

glVertex3f(-0.3,-0.23,0.15);

glColor3f(0.3,0.2,0.3);

glVertex3f(-0.3,-0.25,0.15);

glVertex3f(-0.3,-0.25,-0.15);

glVertex3f(-0.3,-0.23,-0.15);

glVertex3f(-0.3,-0.23,0.15);

glColor3f(0.1,0.2,0.3);

glVertex3f(-0.3,-0.25,-0.15);

glVertex3f(0.3,-0.25,-0.15);

glVertex3f(0.3,-0.23,-0.15);

glVertex3f(-0.3,-0.23,-0.15);

glColor3f(0.3,0.2,0.3);

glVertex3f(0.3,-0.25,0.15);

glVertex3f(0.3,-0.25,-0.15);

glVertex3f(0.3,-0.23,-0.15);

glVertex3f(0.3,-0.23,0.15);

glColor3f(0.3,0.3,0.4);

glVertex3f(-0.3,-0.23,0.15);

glVertex3f(0.3,-0.23,0.15);

glVertex3f(0.3,-0.23,-0.15);

glVertex3f(-0.3,-0.23,-0.15);

glEnd();

//pillers

//left front

glBegin(GL\_QUADS);

glColor3f(0.3,0.2,0.1);

glVertex3f(-0.3,-0.23,0.15);

glVertex3f(-0.28,-0.23,0.15);

glVertex3f(-0.28,0.1,0.15);

glVertex3f(-0.3,0.1,0.15);

glColor3f(0.3,0.2,0.3);

glVertex3f(-0.3,-0.23,0.15);

glVertex3f(-0.3,-0.23,0.12);

glVertex3f(-0.3,0.1,0.12);

glVertex3f(-0.3,0.1,0.15);

glColor3f(0.3,0.2,0.1);

glVertex3f(-0.3,-0.23,0.12);

glVertex3f(-0.28,-0.23,0.12);

glVertex3f(-0.28,0.1,0.12);

glVertex3f(-0.3,0.1,0.12);

glColor3f(0.3,0.2,0.3);

glVertex3f(-0.28,-0.23,0.15);

glVertex3f(-0.28,-0.23,0.12);

glVertex3f(-0.28,0.1,0.12);

glVertex3f(-0.28,0.1,0.15);

glEnd();

//right front

glBegin(GL\_QUADS);

glColor3f(0.3,0.2,0.1);

glVertex3f(0.3,-0.23,0.15);

glVertex3f(0.28,-0.23,0.15);

glVertex3f(0.28,0.1,0.15);

glVertex3f(0.3,0.1,0.15);

glColor3f(0.3,0.2,0.3);

glVertex3f(0.3,-0.23,0.15);

glVertex3f(0.3,-0.23,0.12);

glVertex3f(0.3,0.1,0.12);

glVertex3f(0.3,0.1,0.15);

glColor3f(0.3,0.2,0.1);

glVertex3f(0.3,-0.23,0.12);

glVertex3f(0.28,-0.23,0.12);

glVertex3f(0.28,0.1,0.12);

glVertex3f(0.3,0.1,0.12);

glColor3f(0.3,0.2,0.3);

glVertex3f(0.28,-0.23,0.15);

glVertex3f(0.28,-0.23,0.12);

glVertex3f(0.28,0.1,0.12);

glVertex3f(0.28,0.1,0.15);

glEnd();

//left back

glBegin(GL\_QUADS);

glColor3f(0.3,0.2,0.1);

glVertex3f(-0.3,-0.23,-0.15);

glVertex3f(-0.28,-0.23,-0.15);

glVertex3f(-0.28,0.1,-0.15);

glVertex3f(-0.3,0.1,-0.15);

glColor3f(0.3,0.2,0.3);

glVertex3f(-0.3,-0.23,-0.15);

glVertex3f(-0.3,-0.23,-0.12);

glVertex3f(-0.3,0.1,-0.12);

glVertex3f(-0.3,0.1,-0.15);

glColor3f(0.3,0.2,0.1);

glVertex3f(-0.3,-0.23,-0.12);

glVertex3f(-0.28,-0.23,-0.12);

glVertex3f(-0.28,0.1,-0.12);

glVertex3f(-0.3,0.1,-0.12);

glColor3f(0.3,0.2,0.3);

glVertex3f(-0.28,-0.23,-0.15);

glVertex3f(-0.28,-0.23,-0.12);

glVertex3f(-0.28,0.1,-0.12);

glVertex3f(-0.28,0.1,-0.15);

glEnd();

//right back

glBegin(GL\_QUADS);

glColor3f(0.3,0.2,0.1);

glVertex3f(0.3,-0.23,-0.15);

glVertex3f(0.28,-0.23,-0.15);

glVertex3f(0.28,0.1,-0.15);

glVertex3f(0.3,0.1,-0.15);

glColor3f(0.3,0.2,0.3);

glVertex3f(0.3,-0.23,-0.15);

glVertex3f(0.3,-0.23,-0.12);

glVertex3f(0.3,0.1,-0.12);

glVertex3f(0.3,0.1,-0.15);

glColor3f(0.3,0.2,0.1);

glVertex3f(0.3,-0.23,-0.12);

glVertex3f(0.28,-0.23,-0.12);

glVertex3f(0.28,0.1,-0.12);

glVertex3f(0.3,0.1,-0.12);

glColor3f(0.3,0.2,0.3);

glVertex3f(0.28,-0.23,-0.15);

glVertex3f(0.28,-0.23,-0.12);

glVertex3f(0.28,0.1,-0.12);

glVertex3f(0.28,0.1,-0.15);

glEnd();

//top left

glBegin(GL\_QUADS);

glColor3f(0.4,0.3,0.2);

glVertex3f(-0.3,0.1,0.15);

glVertex3f(-0.28,0.1,0.15);

glVertex3f(-0.28,0.13,0.15);

glVertex3f(-0.3,0.13,0.15);

glColor3f(0.5,0.3,0.2);

glVertex3f(-0.3,0.1,0.15);

glVertex3f(-0.3,0.1,-0.15);

glVertex3f(-0.3,0.13,-0.15);

glVertex3f(-0.3,0.13,0.15);

glColor3f(0.4,0.3,0.2);

glVertex3f(-0.3,0.1,-0.15);

glVertex3f(-0.28,0.1,-0.15);

glVertex3f(-0.28,0.13,-0.15);

glVertex3f(-0.3,0.13,-0.15);

glColor3f(0.5,0.3,0.2);

glVertex3f(-0.28,0.1,0.15);

glVertex3f(-0.28,0.1,-0.15);

glVertex3f(-0.28,0.13,-0.15);

glVertex3f(-0.28,0.13,0.15);

glEnd();

//top roght

glBegin(GL\_QUADS);

glColor3f(0.4,0.3,0.2);

glVertex3f(0.3,0.1,0.15);

glVertex3f(0.28,0.1,0.15);

glVertex3f(0.28,0.13,0.15);

glVertex3f(0.3,0.13,0.15);

glColor3f(0.5,0.3,0.2);

glVertex3f(0.3,0.1,0.15);

glVertex3f(0.3,0.1,-0.15);

glVertex3f(0.3,0.13,-0.15);

glVertex3f(0.3,0.13,0.15);

glColor3f(0.4,0.3,0.2);

glVertex3f(0.3,0.1,-0.15);

glVertex3f(0.28,0.1,-0.15);

glVertex3f(0.28,0.13,-0.15);

glVertex3f(0.3,0.13,-0.15);

glColor3f(0.5,0.3,0.2);

glVertex3f(0.28,0.1,0.15);

glVertex3f(0.28,0.1,-0.15);

glVertex3f(0.28,0.13,-0.15);

glVertex3f(0.28,0.13,0.15);

glEnd();

//sides front

glBegin(GL\_QUADS);

glColor3f(0.4,0.3,0.5);

glVertex3f(-0.28,-0.15,0.15);

glVertex3f(0.28,-0.15,0.15);

glVertex3f(0.28,-0.12,0.15);

glVertex3f(-0.28,-0.12,0.15);

glVertex3f(-0.28,-0.15,0.12);

glVertex3f(0.28,-0.15,0.12);

glVertex3f(0.28,-0.12,0.12);

glVertex3f(-0.28,-0.12,0.12);

//back

glVertex3f(-0.28,-0.15,-0.15);

glVertex3f(0.28,-0.15,-0.15);

glVertex3f(0.28,-0.12,-0.15);

glVertex3f(-0.28,-0.12,-0.15);

glVertex3f(-0.28,-0.15,-0.12);

glVertex3f(0.28,-0.15,-0.12);

glVertex3f(0.28,-0.12,-0.12);

glVertex3f(-0.28,-0.12,-0.12);

glEnd();

//top

glBegin(GL\_QUADS);

glColor3f(0.4,0.3,0.5);

glVertex3f(-0.28,0.1,0.15);

glVertex3f(0.28,0.1,0.15);

glVertex3f(0.28,0.13,0.15);

glVertex3f(-0.28,0.13,0.15);

glVertex3f(-0.28,0.1,0.12);

glVertex3f(0.28,0.1,0.12);

glVertex3f(0.28,0.13,0.12);

glVertex3f(-0.28,0.13,0.12);

//back

glVertex3f(-0.28,0.1,-0.15);

glVertex3f(0.28,0.1,-0.15);

glVertex3f(0.28,0.13,-0.15);

glVertex3f(-0.28,0.13,-0.15);

glVertex3f(-0.28,0.1,-0.12);

glVertex3f(0.28,0.1,-0.12);

glVertex3f(0.28,0.13,-0.12);

glVertex3f(-0.28,0.13,-0.12);

glEnd();

//house

glColor3f(0.8,0.1,0.1);

glPushMatrix();

glTranslatef(0.0,0.25,0.0);

glutSolidCube(0.25);

glPopMatrix();

glBegin(GL\_QUADS); //WINDOW

glColor3f(1.0,1.0,1.0);

glVertex3f(-0.05,0.18,0.16);

glVertex3f(0.05,0.18,0.16);

glVertex3f(0.05,0.25,0.16);

glVertex3f(-0.05,0.25,0.16);

glEnd();

glBegin(GL\_TRIANGLES); //ROOF

glColor3f(1.0,0.8,0.0);

glVertex3f(-0.16,0.35,0.16);

glVertex3f(0.16,0.35,0.16);

glVertex3f(0.0,0.5,0.0);

glVertex3f(-0.16,0.35,0.16);

glVertex3f(-0.16,0.35,-0.16);

glVertex3f(0.0,0.5,0.0);

glVertex3f(-0.16,0.35,-0.16);

glVertex3f(0.16,0.35,-0.16);

glVertex3f(0.0,0.5,0.0);

glVertex3f(0.16,0.35,0.16);

glVertex3f(0.16,0.35,-0.16);

glVertex3f(0.0,0.5,0.0);

glEnd();

}

void track(){

glBegin(GL\_LINES);

glColor3f(0.0,0.0,0.0);//left

glVertex3f(-3.0,-0.23,0.12);

glVertex3f(-0.3,-0.23,0.12);

glVertex3f(-3.0,-0.23,0.1);

glVertex3f(-0.3,-0.23,0.1);

glVertex3f(-3.0,-0.23,-0.12);

glVertex3f(-0.3,-0.23,-0.12);

glVertex3f(-3.0,-0.23,-0.1);

glVertex3f(-0.3,-0.23,-0.1);

glVertex3f(3.0,-0.23,0.12);

glVertex3f(0.3,-0.23,0.12);

glVertex3f(3.0,-0.23,0.1);

glVertex3f(0.3,-0.23,0.1);

glVertex3f(3.0,-0.23,-0.12);

glVertex3f(0.3,-0.23,-0.12);

glVertex3f(3.0,-0.23,-0.1);

glVertex3f(0.3,-0.23,-0.1);

glEnd();

glBegin(GL\_LINES);

glColor3f(0.0,0.0,0.0);

for(j=0.0;j<=2.6;j+=0.1) {

glVertex3f(-3.0+j,-0.23,0.1);

glVertex3f(-3.0+j,-0.23,-0.1);

}

for(j=0.0;j<=3;j+=0.1){

glVertex3f(0.3+j,-0.23,0.1);

glVertex3f(0.3+j,-0.23,-0.1);

}

glEnd();

}

void ship(){

glBegin(GL\_QUADS);

glColor3f(0.8,0.8,0.8); //base

glVertex3f(-0.2,-0.4,-3.5);

glVertex3f(0.2,-0.4,-3.5);

glVertex3f(0.2,-0.3,-3.5);

glVertex3f(-0.2,-0.3,-3.5);

glColor3f(0.8,0.8,1.0);

glVertex3f(-0.2,-0.4,-3.5);

glVertex3f(-0.2,-0.4,-4.8);

glVertex3f(-0.2,-0.3,-5.0);

glVertex3f(-0.2,-0.3,-3.5);

glColor3f(0.8,0.8,0.8);

glVertex3f(-0.2,-0.4,-4.8);

glVertex3f(0.2,-0.4,-4.8);

glVertex3f(0.2,-0.3,-5.0);

glVertex3f(-0.2,-0.3,-5.0);

glColor3f(0.8,0.8,1.0);

glVertex3f(0.2,-0.4,-3.5);

glVertex3f(0.2,-0.4,-4.8);

glVertex3f(0.2,-0.3,-5.0);

glVertex3f(0.2,-0.3,-3.5);

glColor3f(1.0,0.8,1.0);

glVertex3f(-0.2,-0.3,-3.5);

glVertex3f(0.2,-0.3,-3.5);

glVertex3f(0.2,-0.3,-5.0);

glVertex3f(-0.2,-0.3,-5.0);

glColor3f(1.0,0.0,0.7);

glVertex3f(-0.18,-0.3,-3.7);

glVertex3f(0.18,-0.3,-3.7);

glVertex3f(0.18,-0.2,-3.7);

glVertex3f(-0.18,-0.2,-3.7);

glColor3f(1.0,0.0,0.5);

glVertex3f(-0.18,-0.3,-3.7);

glVertex3f(-0.18,-0.3,-4.8);

glVertex3f(-0.18,-0.2,-4.8);

glVertex3f(-0.18,-0.2,-3.7);

glColor3f(1.0,0.0,0.7);

glVertex3f(-0.18,-0.3,-3.7);

glVertex3f(0.18,-0.3,-4.8);

glVertex3f(0.18,-0.2,-4.8);

glVertex3f(-0.18,-0.2,-3.7);

glColor3f(1.0,0.0,0.5);

glVertex3f(0.18,-0.3,-3.7);

glVertex3f(0.18,-0.3,-4.8);

glVertex3f(0.18,-0.2,-4.8);

glVertex3f(0.18,-0.2,-3.7);

glColor3f(1.0,0.1,0.8);

glVertex3f(-0.18,-0.2,-3.7);

glVertex3f(0.18,-0.2,-3.7);

glVertex3f(0.18,-0.2,-4.8);

glVertex3f(-0.18,-0.2,-4.8);

glEnd();

//front

glBegin(GL\_TRIANGLES);

glColor3f(0.5,0.5,0.7);

glVertex3f(-0.2,-0.4,-3.5);

glVertex3f(-0.2,-0.3,-3.5);

glVertex3f(0.0,-0.15,-2.2);

glColor3f(0.5,0.8,0.7);

glVertex3f(-0.2,-0.4,-3.5);

glVertex3f(0.2,-0.4,-3.5);

glVertex3f(0.0,-0.15,-2.2);

glColor3f(0.5,0.5,0.7);

glVertex3f(0.2,-0.4,-3.5);

glVertex3f(0.2,-0.3,-3.5);

glVertex3f(0.0,-0.15,-2.2);

glEnd();

//TOP PILLARS

glBegin(GL\_QUADS);

glColor3f(1.0,0.8,0.1);

glVertex3f(-0.05,-0.2,-3.8);

glVertex3f(0.05,-0.2,-3.8);

glVertex3f(0.05,0.1,-3.8);

glVertex3f(-0.05,0.1,-3.8);

glColor3f(1.0,0.8,0.2);

glVertex3f(-0.05,-0.2,-3.8);

glVertex3f(-0.05,-0.2,-4.0);

glVertex3f(-0.05,0.1,-4.0);

glVertex3f(-0.05,0.1,-3.8);

glColor3f(1.0,0.8,0.1);

glVertex3f(-0.05,-0.2,-4.0);

glVertex3f(0.05,-0.2,-4.0);

glVertex3f(0.05,0.1,-4.0);

glVertex3f(-0.05,0.1,-4.0);

glColor3f(1.0,0.8,0.2);

glVertex3f(0.05,-0.2,-3.8);

glVertex3f(0.05,-0.2,-4.0);

glVertex3f(0.05,0.1,-4.0);

glVertex3f(0.05,0.1,-3.8);

//back

glColor3f(1.0,0.5,0.1);

glVertex3f(-0.05,-0.2,-4.2);

glVertex3f(0.05,-0.2,-4.2);

glVertex3f(0.05,0.2,-4.2);

glVertex3f(-0.05,0.2,-4.2);

glColor3f(1.0,0.6,0.2);

glVertex3f(-0.05,-0.2,-4.2);

glVertex3f(-0.05,-0.2,-4.5);

glVertex3f(-0.05,0.2,-4.5);

glVertex3f(-0.05,0.2,-4.2);

glColor3f(1.0,0.5,0.1);

glVertex3f(-0.05,-0.2,-4.5);

glVertex3f(0.05,-0.2,-4.5);

glVertex3f(0.05,0.2,-4.5);

glVertex3f(-0.05,0.2,-4.5);

glColor3f(1.0,0.6,0.2);

glVertex3f(0.05,-0.2,-4.2);

glVertex3f(0.05,-0.2,-4.5);

glVertex3f(0.05,0.2,-4.5);

glVertex3f(0.05,0.2,-4.2);

glEnd();

}

void train(){

glBegin(GL\_QUADS); //engine

glColor3f(0.8,0.6,0.4);

glVertex3f(1.0,-0.23,0.1);

glVertex3f(1.15,-0.23,0.1);

glVertex3f(1.15,-0.14,0.1);

glVertex3f(1.0,-0.14,0.1);

glColor3f(0.5,0.5,0.8);

glVertex3f(1.0,-0.23,0.1);

glVertex3f(1.0,-0.23,-0.1);

glVertex3f(1.0,-0.14,-0.1);

glVertex3f(1.0,-0.14,0.1);

glColor3f(0.8,0.6,0.4);

glVertex3f(1.0,-0.23,-0.1);

glVertex3f(1.15,-0.23,-0.1);

glVertex3f(1.15,-0.14,-0.1);

glVertex3f(1.0,-0.14,-0.1);

//FRONT

glColor3f(0.0,0.1,0.9);

glVertex3f(1.0,-0.14,0.1);

glVertex3f(1.15,-0.05,0.1);

glVertex3f(1.15,-0.05,-0.1);

glVertex3f(1.0,-0.14,-0.1);

glColor3f(0.8,0.6,0.2);

glVertex3f(1.02,-0.12,0.1);

glVertex3f(1.13,-0.05,0.1);

glVertex3f(1.13,-0.05,-0.06);

glVertex3f(1.02,-0.12,-0.06);

glEnd();

//side

glBegin(GL\_TRIANGLES);

glColor3f(0.0,0.0,0.0);

glVertex3f(1.0,-0.14,0.1);

glVertex3f(1.15,-0.14,0.1);

glVertex3f(1.15,-0.05,0.1);

glVertex3f(1.0,-0.14,-0.1);

glVertex3f(1.15,-0.14,-0.1);

glVertex3f(1.15,-0.05,-0.1);

glEnd();

//bogies

glBegin(GL\_QUADS);

for(j=0.0;j<2;j+=0.27){

glColor3f(0.5,0.0,0.1);

glVertex3f(1.15+j,-0.23,0.1);

glVertex3f(1.4+j,-0.23,0.1);

glVertex3f(1.4+j,-0.05,0.1);

glVertex3f(1.15+j,-0.05,0.1);

glColor3f(0.5,0.0,0.5);

glVertex3f(1.15+j,-0.23,0.1);

glVertex3f(1.15+j,-0.23,-0.1);

glVertex3f(1.15+j,-0.05,-0.1);

glVertex3f(1.15+j,-0.05,0.1);

glColor3f(0.5,0.0,0.1);

glVertex3f(1.15+j,-0.23,-0.1);

glVertex3f(1.4+j,-0.23,-0.1);

glVertex3f(1.4+j,-0.05,-0.1);

glVertex3f(1.15+j,-0.05,-0.1);

glColor3f(0.5,0.0,0.5);

glVertex3f(1.4+j,-0.23,0.1);

glVertex3f(1.4+j,-0.23,-0.1);

glVertex3f(1.4+j,-0.05,-0.1);

glVertex3f(1.4+j,-0.05,0.1);

glColor3f(0.8,0.3,0.5);

glVertex3f(1.15+j,-0.05,0.1);

glVertex3f(1.4+j,-0.05,0.1);

glVertex3f(1.4+j,-0.05,-0.1);

glVertex3f(1.15+j,-0.05,-0.1);

}

glEnd();

}

void aero(){

glBegin(GL\_POLYGON);

glColor3f(0.9,0.9,.9);

glVertex3f(-3.2,0.7,-0.8);

glVertex3f(-3.0,0.67,-0.8);

glVertex3f(-2.4,0.67,-0.8);

glVertex3f(-2.4,0.73,-0.8);

glVertex3f(-3.2,0.73,-0.8);

glVertex3f(-3.2,0.7,-0.65);

glVertex3f(-3.0,0.67,-0.65);

glVertex3f(-2.4,0.67,-0.65);

glVertex3f(-2.4,0.73,-0.65);

glVertex3f(-3.2,0.73,-0.65);

glEnd();

glBegin(GL\_QUADS);

glColor3f(0.7,0.7,.7);

glVertex3f(-3.2,0.7,-0.65);

glVertex3f(-3.2,0.7,-0.8);

glVertex3f(-3.2,0.73,-0.8);

glVertex3f(-3.2,0.73,-0.65);

glColor3f(0.7,0.7,.7);

glVertex3f(-2.4,0.67,-0.65);

glVertex3f(-2.4,0.67,-0.8);

glVertex3f(-2.4,0.73,-0.8);

glVertex3f(-2.4,0.73,-0.65);

glColor3f(0.7,0.8,0.8);

glVertex3f(-3.2,0.73,-0.65);

glVertex3f(-2.4,0.73,-0.65);

glVertex3f(-2.4,0.73,-0.8);

glVertex3f(-3.2,0.73,-0.8);

//wings

glColor3f(0.8,0.2,0.5);

glVertex3f(-2.8,0.7,-0.7);

glVertex3f(-2.62,0.7,-0.7);

glVertex3f(-2.75,0.7,-0.2);

glVertex3f(-2.85,0.7,-0.2);

glVertex3f(-2.8,0.7,-0.8);

glVertex3f(-2.62,0.7,-0.8);

glVertex3f(-2.75,0.7,-1.3);

glVertex3f(-2.85,0.7,-1.3);

glVertex3f(-3.2,0.73,-0.725);

glVertex3f(-3.1,0.73,-0.725);

glVertex3f(-3.18,0.82,-0.725);

glVertex3f(-3.25,0.82,-0.725);

glEnd();

glBegin(GL\_TRIANGLES);

glColor3f(1.0,0.5,0.2);

glVertex3f(-2.4,0.73,-0.8);

glVertex3f(-2.4,0.73,-0.65);

glVertex3f(-2.2,0.68,-0.725);

glColor3f(1.0,0.7,0.4);

glVertex3f(-2.4,0.67,-0.65);

glVertex3f(-2.4,0.73,-0.65);

glVertex3f(-2.2,0.68,-0.725);

glColor3f(1.0,0.5,0.2);

glVertex3f(-2.4,0.68,-0.65);

glVertex3f(-2.4,0.68,-0.65);

glVertex3f(-2.2,0.68,-0.725);

glColor3f(1.0,0.7,0.4);

glVertex3f(-2.4,0.69,-0.8);

glVertex3f(-2.4,0.73,-0.8);

glVertex3f(-2.2,0.68,-0.725);

glEnd();

char str[]=" AIR CIT . . .";

setFont(GLUT\_BITMAP\_HELVETICA\_18);

glColor3f(0.0,0.0,0.0);

drawstring(str,-3.05,0.69,-0.65);

}

void lighthouse(){

glBegin(GL\_QUADS);

glColor3f(0.8,0.2,0.0);

glVertex3f(0.2,-0.42,-5.01);

glVertex3f(0.35,-0.42,-5.01);

glVertex3f(0.3,-0.1,-5.01);

glVertex3f(0.25,-0.1,-5.01);

glColor3f(1.0,1.0,1.0);

glVertex3f(0.23,-0.1,-5.01);

glVertex3f(0.32,-0.1,-5.01);

glVertex3f(0.32,0.0,-5.01);

glVertex3f(0.23,0.0,-5.01);

glColor3f(1.0,1.0,1.0);

glVertex3f(0.215,-0.3,-5.0);

glVertex3f(0.33,-0.3,-5.0);

glVertex3f(0.315,-0.2,-5.0);

glVertex3f(0.23,-0.2,-5.0);

glEnd();

glBegin(GL\_TRIANGLES);

glColor3f(1.0,0.2,0.2);

glVertex3f(0.2,0.0,-5.0);

glVertex3f(0.35,0.0,-5.0);

glVertex3f(0.27,0.05,-5.0);

glEnd();

}

void signal(){

glBegin(GL\_QUADS);

glColor3f(0.1,0.2,0.1);

glVertex3f(0.7,-0.25,-0.17);

glVertex3f(0.73,-0.25,-0.17);

glVertex3f(0.73,0.15,-0.17);

glVertex3f(0.7,0.15,-0.17);

glColor3f(0.1,0.1,0.2);

glVertex3f(0.67,0.15,-0.17);

glVertex3f(0.76,0.15,-0.17);

glVertex3f(0.76,0.3,-0.17);

glVertex3f(0.67,0.3,-0.17);

glEnd();

}

void light(){

if(b>0.0)

glColor3f(1.0,0.0,0.0);

else

glColor3f(0.0,1.0,0.0);

if(p<-3.5)

glColor3f(1.0,0.0,0.0);

glPushMatrix();

glTranslatef(0.715,0.25,-0.17);

glutSolidSphere(0.03,10,10);

glPopMatrix();

}

void new1(){

glTranslatef(a,b,c);

bridge();

}

void new2(){

glTranslatef(m,n,o);

ship();

}

void new3(){

glTranslatef(p,q,r);

train();

}

void new4(){

glTranslatef(x,y,z);

aero();

}

void new5(){

glTranslatef(a1,a2,a3);

lines();

}

void update(int value){

if(flagx==1){

if(flagb==1){

b+=0.02f;

if(b>0.5){

flagb=2;

flags=1;

}

}

if(flags==1){

o+=0.07f;

if(o>2.0)

flagp=1;

if(o>6.0){

flagb=0;

}

}

if(flagb==0){

b-=0.02f;

if(b<0.01){

flagb=1;

flagt=1;

}

}

if(flagt==1){

p-=0.05f;

}

if(flagp==1){

x+=0.035;

}

if(flagw==1){

a1+=0.006;

}

}

glutPostRedisplay();

glutTimerFunc(100,update,0);

}

void display(){

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

glClearColor(0.0,0.5,1.0,0.0);

glPushMatrix();

glRotatef(20.0,0.25,0.5,0.0);

base();

glPopMatrix();

glPushMatrix();

glRotatef(20.0,0.25,0.5,0.0);

pillars();

glPopMatrix();

glPushMatrix();

glRotatef(20.0,0.25,0.5,0.0);

earth();

glPopMatrix();

glPushMatrix();

glRotatef(20.0,0.25,0.5,0.0);

track();

glPopMatrix();

glPushMatrix();

glRotatef(20.0,0.25,0.5,0.0);

glPopMatrix();

glPushMatrix();

glRotatef(20.0,0.25,0.5,0.0);

new1();

glPopMatrix();

glPushMatrix();

glRotatef(20.0,0.25,0.5,0.0);

new2();

glPopMatrix();

glPushMatrix();

glRotatef(20.0,0.25,0.5,0.0);

new3();

glPopMatrix();

glPushMatrix();

glRotatef(20.0,0.25,0.5,0.0);

new4();

glPopMatrix();

glPushMatrix();

glRotatef(20.0,0.25,0.5,0.0);

water();

glPopMatrix();

glPushMatrix();

glRotatef(20.0,0.25,0.5,0.0);

new5();

glPopMatrix();

glPushMatrix();

glColor3f(1.0,1.0,0.0);

glTranslatef(1.2,0.9,-5.1);

glutSolidSphere(0.08,20,20);

glPopMatrix();

glPushMatrix();

glRotatef(20.0,0.25,0.5,0.0);

lighthouse();

glTranslatef(0.28,-0.05,-5.0);

glColor3f(0.0,0.0,0.0);

glutSolidSphere(0.02,20,20);

glPopMatrix();

glPushMatrix();

glRotatef(20.0,0.25,0.5,0.0);

signal();

glPopMatrix();

glPushMatrix();

glRotatef(20.0,0.25,0.5,0.0);

light();

glPopMatrix();

glFlush();

}

void mydisplay(){

if(flag==0)

screen1();

if(flag==1)

display();

if(p<-6.0)

screen3();

if(p<-6.8)

exit(0);

}

void mykeyboard(unsigned char key,int x,int y){

switch(key){

case 13 :flag=1;break;

case 83 :if(flag==1)

flagx=1;break;

case 115:if(flag==1)

flagx=1;break;

case 84 :flagx=0;break;

case 116 :flagx=0;break;

case 27:exit(0);

}

}

void reshape(int w,int h){

glViewport(0,0,w,h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

if(w<=h)

glOrtho(-1.1,1.1,1.1\*(GLfloat)h/(GLfloat)w,1.1\*(GLfloat)h/(GLfloat)w,-10.0,10.0);

else

glOrtho(-1.1\*(GLfloat)w/(GLfloat)h,1.1\*(GLfloat)w/(GLfloat)h,-1.1,1.1,-10.0,10.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

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

glutInit(&argc,argv);

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

glutInitWindowSize(1500,1000);

glutCreateWindow("VLBS");

glClearColor(0.0,0.0,0.0,0.0);

glEnable(GL\_DEPTH\_TEST);

glutReshapeFunc(reshape);

glutDisplayFunc(mydisplay);

glutKeyboardFunc(mykeyboard);

glutTimerFunc(200,update,0);

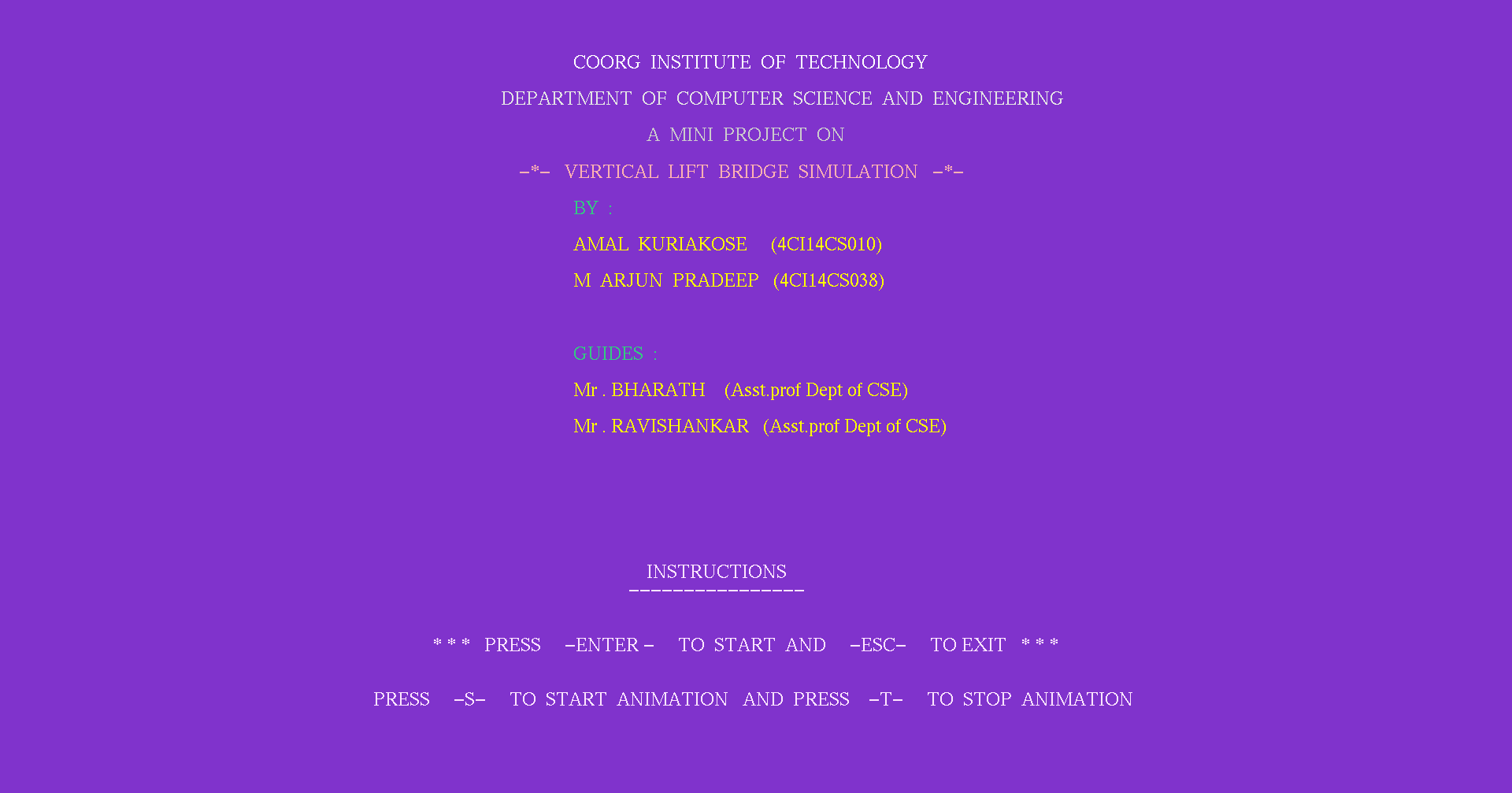
glutMainLoop();

return 0;

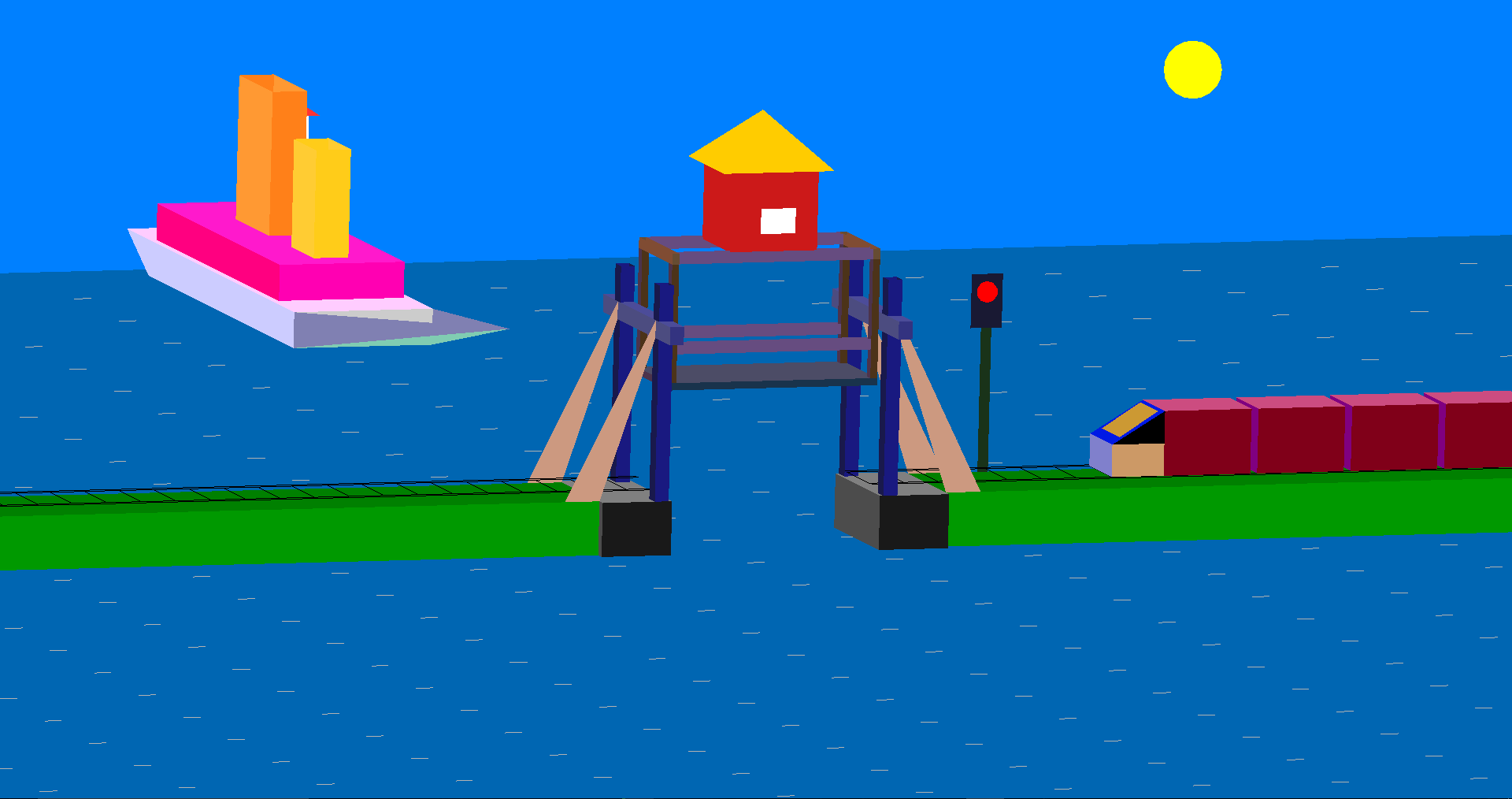
}

**CHAPTER 4**

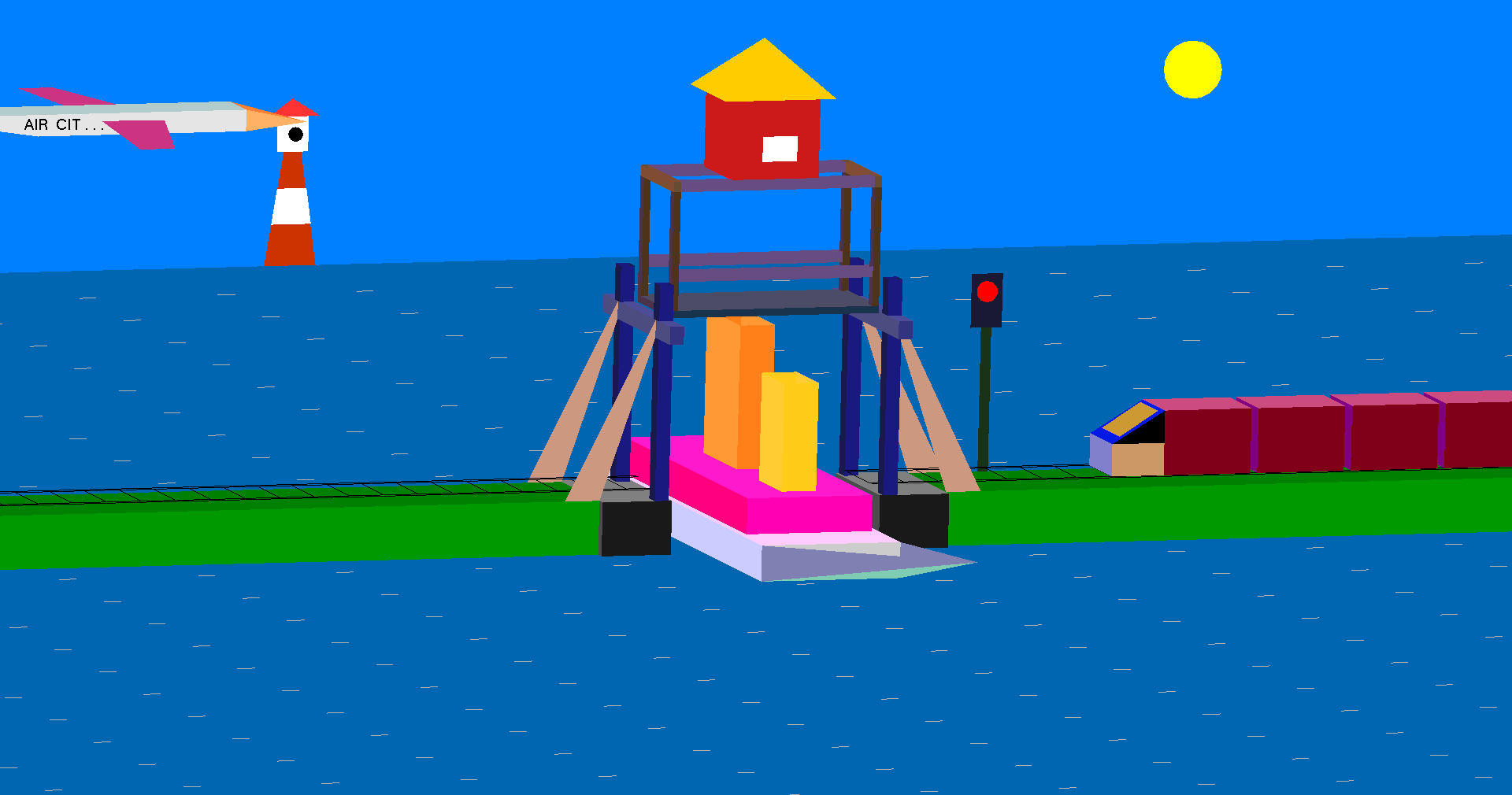
**SNAPSHOTS**

****

**WELCOME SCREEN**

****

**LIFTING OF THE BRIDGE**

****

**SHIP CROSSING THE BRIDGE**

****

**LOWERING OF THE BRIDGE**

****

**TRAIN CROSSING THE BRIDGE**

****

**EXIT SCREEN**

**CHAPTER 5**

**CONCLUSION AND FUTURE SCOPE**

**5.1 CONCLUSION**

Designing and implementing project in graphics is a great experience. We understood and analyzed about the concepts of OpenGL which is very useful for our future.

**“VERTICAL LIFT BRIDGE SIMULATION”** is developed to provide a GUI. An attempt has been made to develop an openGL package which meets necessary requirements of the user.

The development of the mini project has given us a good exposure to openGL by which we have learnt some of the techniques which help in the development of interactive application

**5.2 FUTURE SCOPE**

This application is like open source where anyone can design and add his own codes to modify. Even more features can be included.

**REFERENCES**

**TEXT BOOKS:**

[1]. Edward Angel, Interactive Computer Graphics A Top-Down Approach

Using OpenGL,Pearson Education Asia,Fifth Edition,2008.

**WEBSITE:**

[1]. [www.OpenGL.org](http://www.OpenGL.org)

[2]. [www.Github.com](http://www.Github.com)

[3]. [www.stackoverflow.com](http://www.stackoverflow.com)

[4]. [www.youtube.com](http://www.youtube.com)

[5]. [www.openglprojects.in](http://www.openglprojects.in)