CHAPTER 1

INTRODUCTION

1.1 Overview of Computer Graphics

Computer Graphics become a powerful tool for the rapid and economical production of pictures. There is virtually no area in which Graphical displays cannot be used to some advantage so it is not surprising to find the use of CG so widespread.

Although early application in engineering & science had to rely on expensive & cumbersome equipment, advances in computer technology have made interactive computer graphics a practical tool.

Today, Computer Graphics is found in a diverse area such as science, engineering, medicine, business, industry, government, art, entertainment, education and training.

Now, so able to answer about computer graphics as generalized tool for drawing and creating pictures and simulate the real world situations within a small computer window.

1.2 History

William fetter was credited with coning the term Computer Graphics in 1960, to describe his work at Boeng. One of the first displays of computer animation was future world (1976), which included an animation of a human face and hand-produced by Carmull and Fred Parkle at the University of Utah.

There are several international conferences and journals where the most significant results in computer-graphics are published. Among them are the SIGGRAPH and Euro graphics conferences and the association for computing machinery (ACM) transaction on Graphics journals.

1.3 Application of Computer Graphics

Today computer graphics is a standard component in most of the computer applications.

- Entertainment: Computer animations, special effects, computer or video games all utilizes rendering, modeling, touch up and background painting software based on computer graphics techniques.
- User interface: GUI's (graphical user interface) are dominant UI (user interface) for the personal computer work stations. Most GUI's use 2D graphics and some advance GUI's are 3D graphics.
- Computer aided graphics and design: Architecture, electronics machining in the aerospace and automotive industries use 3D and 2D graphics to build and analyze design prior manufacturing, simulations based on the design parameters are often presented as animated sequence or VR (virtual reality).
- Health care: Patients are analyzed and diagnosed using graphics representations of information such as MRI (magnetic resonance imaging), CT (computed Tomography) or ultra sound data; treatment of surgical planning is depicted using 3D graphics.
- Cartography: Most GIS (geographic information system) utilize 2D or 3D graphics for applications such as land use planning (zoning), weather or crop prediction.
- Interactive plotting: Business graphics packages use 2D or 3D plots to represent statistical information, risk analysis many other information. Programs such as Microsoft Word, Excel, and Power Point all have embedded in them standard plotting packages.
- Science and Engineering: Visualization of data and computational models 2D,3D and even higher dimensions often requires sophisticated modeling and rendering to produce interactive animated depiction of physical processes.

Advantages of interactive graphics:

- Humans are predominantly visual.
- New medium for producing static images.

- Unprecedented ability to produce new pictures fast.
- Actual change of shape color or other properties of objects being displayed.

1.4 User interface

Now a day's even school children are comfortable with interactive graphics techniques, such as the desktop metaphor for window manipulation and menu and icons selection with mouse. Graphics based user interfaces have made productive users of neophytes, and the desk without its graphics computer is increasingly rare.

At the same time the interactive graphics has become common in user interface and visualization of data and objects, the rendering of 3D objects has become dramatically more realistic, as evidenced by the ubiquitous computer generated commercials and movie special effects. Techniques those were experimental in early 80's now standard practice and more remarkable photorealistic effects are around the corner.

1.5 Statement of the Problem

To stimulate 17 minutes bridge puzzle using OpenGL. There are 4 men who want to cross a bridge; they all begin on the same side. You have 17 minutes to get all of them across to the other side. It is night and there is one flashlight .A maximum of two people can cross at one time. Any party who crosses, either one or two people must have a flashlight with them. The flashlight must be walked back and forth, it cannot be thrown. A pair must walk together at a rate of slower men's pace. Goldman needs one minute to cross, lavender man needs two minutes, pink man needs 5 minutes, green man needs 10 minutes.

1.6 Objectives

- The main objective of the puzzle is to make all men to cross the bridge with the help of flashlight in 17 minutes by applying certain condition.
- Man, bridge, land, water, flashlight will be drawn using OpenGL.

- String will be written using OpenGL drawstring in Pseudo code format.
- After each movement time will be correspondingly decremented and will be displayed on the screen.
- Rules violated and puzzle solved message will be displayed.

1.7 Organization of the Report

Chapter 1 introduces the computer graphics and its applications. Chapter 2 contains the basic concepts and principles involved in OpenGL. Chapter 3 is specific to the problem. Chapter 4 deals with design and implementation of the 17 minutes bridge puzzle. Chapter 5 deals with snapshots and the result regarding project. Chapter 6 deals with conclusion of the project and also mentions its future scope. Lastly reference made for the project.

CHAPTER 2

INTRODUCTION TO OPENGL

2.1 Introduction

OpenGL is a software interface to graphics hardware. This interface consists of about 150 distinct commands that are used to specify the objects and operations needed to produce interactive three-dimensional applications. OpenGL is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms.

Most of the application will be designed to access OpenGL directly through functions in three libraries. Functions in the main GL (or OpenGL in windows) library have names that begin with the letters gl and are stored in a library usually referred to as GL (or OpenGL in windows). The second is the **OpenGL Utility Library** (GLU). This 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 letters glu.

To interface with the window system and to get input from external devices into the programs need at least one more system-specific library that provides the “glue” between the window system and OpenGL. For the X window system, this library is functionality that should be expected in any modern windowing system.

Fig 2.1 shows the organization of the libraries for an X Window System environment. For this window system, GLUT will use GLX and the X libraries. The application program, however, can use only GLUT functions and thus can be recompiled with the GLUT library for other window systems.

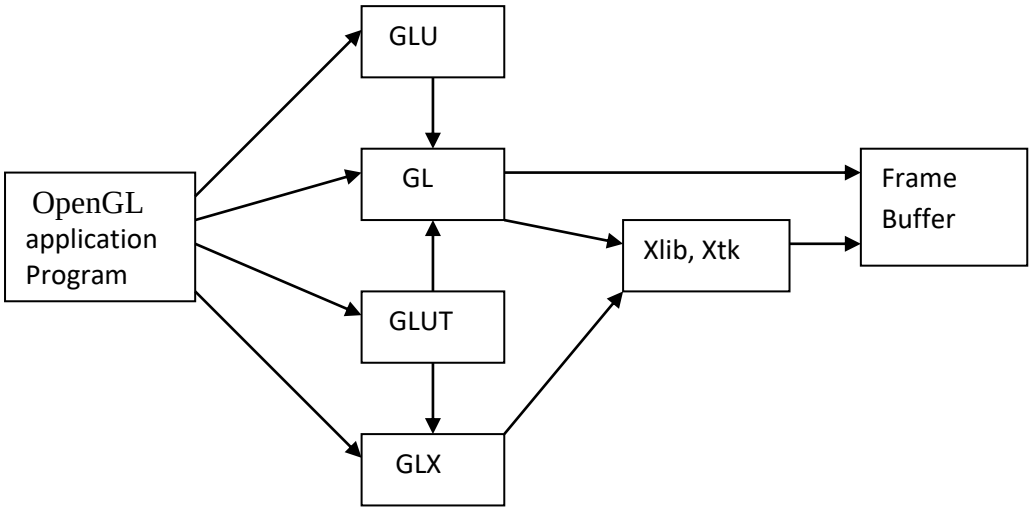


Fig 2.1 Library organization of OpenGL

2.1.1 OpenGL Command Syntax

OpenGL command use the prefix **gl** and initial capital letters for each word making up the command name. Similarly, OpenGL defined constants begin with GL_, use all capital letters, and use underscore to separate words (like GL_COLOR_BUFFERBIT). Some extraneous letters are appended to some command names (for example, the **3f** in **glColor3f() and glVertex3f()**). It's true that the **Color** part of the command name **glColor3f()** is enough to define the command as one that sets the current color. However, more than one such command has been defined so as to use different types of arguments. In particular, the **3** part of the suffix indicates that three arguments are given; another version of the **Color** command takes four arguments. The f part of the suffix indicates that the arguments are floating-point numbers. Having different formats allow OpenGL to accept the user's data. Some OpenGL commands accept as many as 8 different data types for their arguments. The letters used as suffixes to specify theses data types for **ISO C** implementation of OpenGL are shown in table-1, along with the corresponding OpenGL type definition.

Suffix	Data Types	Typical Corresponding C-Language Type	OpenGL Type Definition
B	8-bit integer	Signed char	GLbyte
S	16-bit integer	Short	GLshort
I	32-bit integer	int or long	Glint, GLsizei
F	32-bit floating-point	Float	GLfloat, GLclampf
D	64-bit floating-point	Double	GLdouble GLclampd
Ub	8-bit unsigned Integer	Unsigned char	GLubyte, GLboolean
Us	16-bit unsigned Integer	Unsigned short	GLushort
Ui	32-bit unsigned Integer	Unsigned int or Unsigned long	GLuint, GLenum, GLbitfied

Table 2.1: Command Suffixes and Argument Data Types

Thus, the two commands glVertex2i (1, 3); glVertex2f (1.0, 3.0); are equivalent, except that the first specifies the vertex's coordinates as 32-bit integers, and the second specifies them as single-precision floating-point numbers.

2.1.2 OpenGL as a State Machine

OpenGL is a state machine. The current color is a state variable. The current color can be set to white, red, or any other color, and there after every object is drawn with that many state variables refer to modes that ate enabled or disabled with the command **glEnable()** or **glDisable().**

Each state variable or mode has a default value, and at any point the system can be queried for each variable's current value. Typically one of the six following commands can be used to do this: **glGetBooleanv(), glGetDoublev(), glGetFloatv(), glGetIntegerv(), glGetPointerv(), or glIsEnabled().**

2.1.3 Display Lists

All data, whether it desires geometry or pixels, can be saved in a display list for current or later use. When a display list is executed, the retained data is sent from the

display list just as if it were sent by the application in immediate mode.

2.1.4 Evaluators

All geometric primitives are eventually described by vertices. Parametric curves and surfaces may be initially described by control points and polynomial function called basis functions. Evaluators provide a method to drive the vertices used to represent the surfaces from the control points. The method is a polynomial mapping, which can produce surface normal, texture coordinates, colors, and spatial coordinate value from the control points.

2.1.5 Primitive Assembly

Clipping, major part of primitive assembly is the elimination of portion 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 perspective division, which makes distant geometric objects appear smaller than closer objects. Then viewport 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.

2.1.6 Pixel Operation

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 result are clamped and then either written into texture memory sent to rasterization step. If pixel data is read from the frame buffer, pixel-transfer operation (scale, bias, mapping, and clamping) are performed. Then these results are packed into an appropriate format and returned to an array in system memory.

2.1.7 Texture Assembly

An OpenGL application may wish to apply texture image onto geometric objects to make them look more realistic. Some OpenGL implementation may have special

resources to accelerate texture performance. There may be specialized, high performance texture memory.

2.1.8 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 anti-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.

2.1.9 Fragment Operations

Before values are actually stored into the frame buffer, a series of operations are performed that may alter or even throw out fragments. All these operations can be enabled or disabled. The first operation which may be encountered is texturing, where a Texel (texture element) is generated from texture memory for each fragment and applied to the fragment.

CHAPTER 3

BASIC CONCEPT AND WORKING PRINCIPLE

3.1 Concepts of 17 minutes bridge puzzle

The aim/concept of 17 minutes bridge puzzle is to make all men to cross the bridge with the help of flashlight in 17 minutes by applying certain logic over each and every movement of the men. In this puzzle, we have to make all men to cross the bridge based on the following rules applied on each man. The applied rules for the proper movement of the balls are

a) The bridge has to be crossed in 17 minutes.

b) At most two men can be moved at a time.

c) It is night and there is only one flashlight. It must be walked back and forth.

d) Goldman needs 1 minute to cross the bridge.

e) Lavender man needs 2 minutes to cross the bridge.

f) Pink man needs 5 minutes to cross the bridge.

g) Green man needs 10 minutes to cross the bridge.

h) Slowest man’s pace will be considered.

For solving out the problem, simply go through the logic that already has been implemented. Apply various logics and go through the conditions applied. Through the concept of hit and trail the desired output can be achieved. These are the problems which required the same logic as used by us in order to solve out the ball problems.

CHAPTER 4

DESIGN AND IMPLEMENTATION

4.1 Design

Men Location-> Men location plays a key role in moving the person on or off the bridge. Using current men location, men are drawn every time the display function is called. Man location is nothing but the x-coordinate of the man. Variable used for this are goldloc, lavloc, pinkloc, and greenloc for gold, lavender, pink and green ball respectively.

Men Number-> Each man is given a unique number. This number helps in indicating which man is currently placed on the bridge. Value of boatseat1 and boatseat2 variable such as 1, 2, 3 and 4 indicates gold, lavender, pink and green ball respectively.

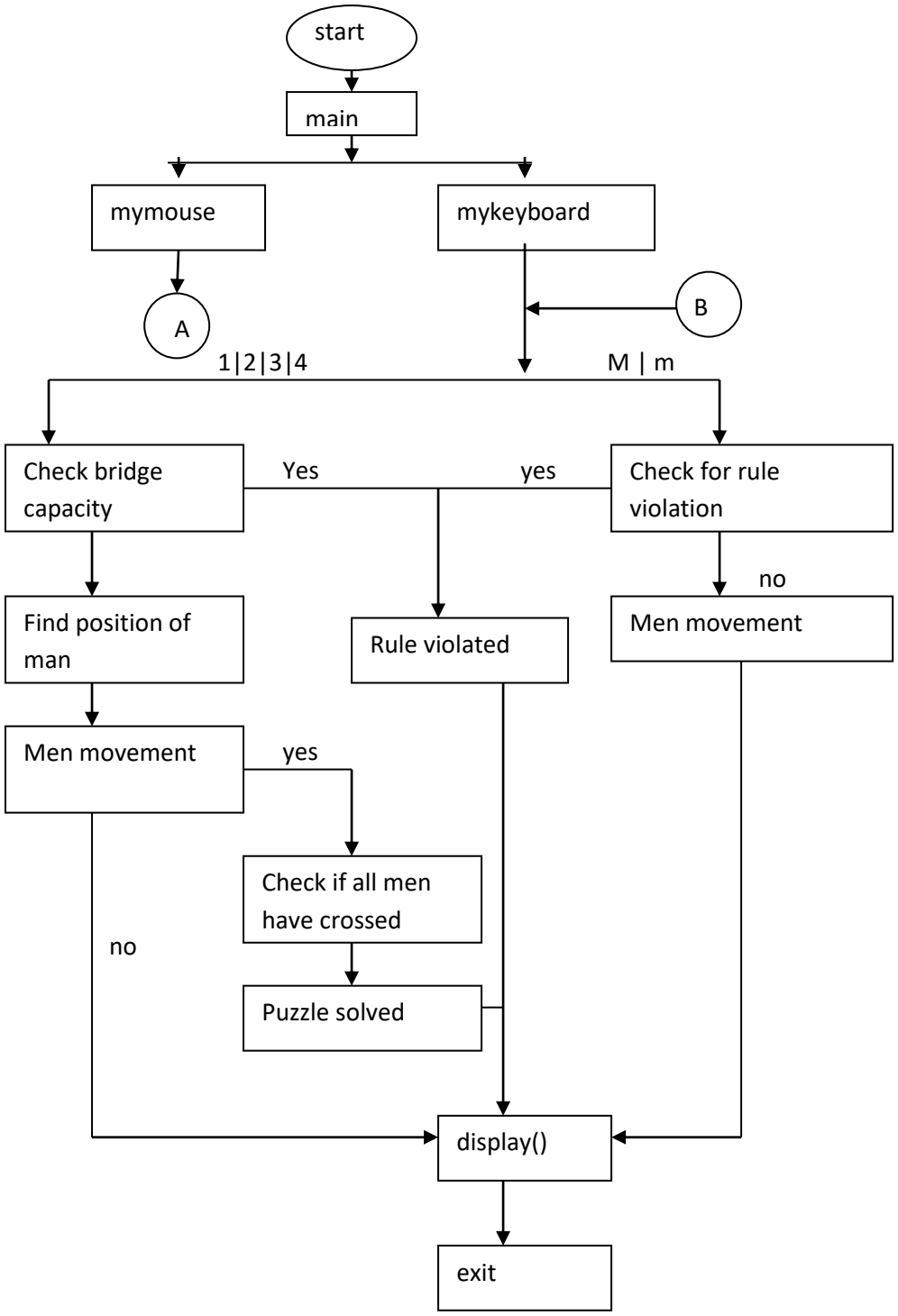
Boat Seat and Capacity-> Boatseat1and boatseat2 are the variable used to move at most two persons on bridge. boatcapacity is the variable used to check how many persons are placed on the bridge and also for checking the maximum bridge capacity.

Boat Location-> Variable boatloc is used to indicate where the boat is, and also men's placing on/off the bridge is possible or not.

4.2 Implementation

4.2.1 Flow chart

Figure 4.1 shows the working principle of 17 minutes bridge puzzle. The execution starts from main. User can interact in two ways namely keyboard and mouse. On pressing any key, program checks for the bridge capacity, position of man and checks for rule violation, if some rule is violated it displays rule violation screen else movement of man takes place. If all men have crossed the bridge, puzzle solved screen will be displayed. In mouse interaction on clicking about the puzzle, screen having rules of the game will be displayed. On clicking play, play screen will be displayed and on clicking exit, program will be exited.


start
main
mymouse
mykeyboard
A
B
1|2|3|4
M | m
Check bridge capacity
Yes
yes
Check for rule violation
no
Find position of man
Rule violated
Men movement
Men movement
yes
Check if all men have crossed
no
Puzzle solved
display()
exit

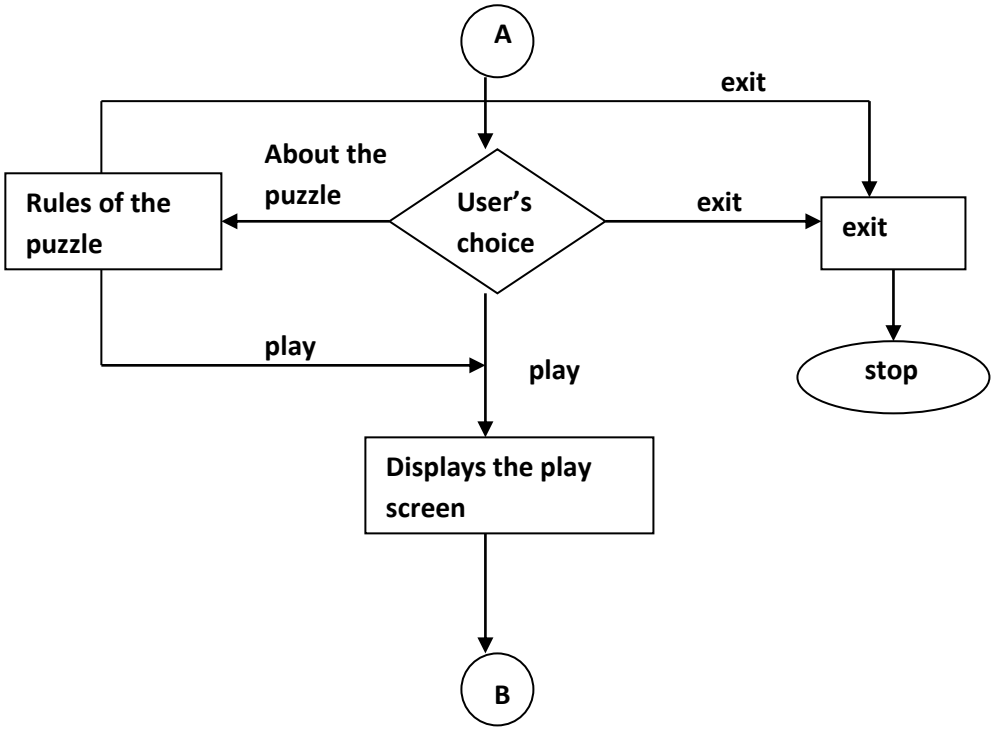


Figure 4.1 Pictorial Representation of 17 minutes bridge puzzle

4.2.2 Built-in functions

OpenGL functions used in the project are:

➔ **void glutCreateWindow(char *title);**

Opens an OpenGL window where the title at the top of the window is given by the string title.

➔ **void glutInitWindowPosition(int x, int y);**

Position the created window at the given co-ordinate (x, y).

➔ **void glutInitWindowSize(int width, int height);**

Specifies a y*x window at the top left corner of the display.

➔ **void glutDisplayFunc(void (*func) (void));**

Graphics are sent to the screen through this function and registered with the windows system.

➔ **void gluOrtho2D(GLint left, GLint right, GLint bottom, GLint top);**

Defines a two dimensional viewing rectangle in plane z=0.

➔ **void glutMainLoop();**

This will cause the program to begin an event-processing loop. If there are no events to process, the program will sit in a wait state, with the graphics on the screen, until to terminate the program through some external means.

➔ **void glColor3f(float, float, float);**

The three parameters specify the red, green and blue colors respectively of the RGB color model. This is used to set the colors of the objects.

➔ **void glClearColor(float, float, float, float);**

Three parameters specify the red, green and blue colors respectively of the RGB color model and the 4^{th} parameter specifies the transparency or opacity. This is used to clear the drawing window.

➔ **void glBegin(glEnum mode);**

Initiates a new primitive of type mode and starts the collection of vertices. Values of mode include GL_POINTS, GL_LINES and GL_POLYGON.

➔ **void glEnd();**

It terminates a list of vertices.

➔ **void glVertex [234][sifd](TYPE x coordinate, TYPE y coordinate…);**

Specifies the position of a vertex in 2,3,4 dimensions. The coordinates can be specified as shorts(s), ints(i), floats(f) or doubles(d).

➔ **void glFlush();**

It forces any buffered OpenGL commands to execute.

4.2.3 User defined functions

User defined functions used in the project are:

➔ **void delay()**

It delays the display for few times.

➔ **void boat()**

It is uses boat location to draw the boat on the display screen.

➔ **void man(float)**

It is uses men location as parameter to draw the men on the display screen.

➔ **void drawman()**

It is used to pass the man location as parameter to drawman function.

➔ **void lanwa()**

It is used to draw land and water in the display screen.

➔ **void drawstring(float , float , char *)**

It is used to display text on the display sreen.

➔ **void display()**

It is used to display the display screen.

➔ **void display1()**

It is used to display about the puzzle on mouse click.

➔ **void display2()**

It is used to restart the game.

➔ **void aboutthepuzzle()**

It displays rules of the game.

➔ **void bridge()**

Draws the bridge.

➔ **void firstscreen()**

Used to display project associates and guide's names.

➔ **void Flashlight()**

Displays flash light.

➔ **void handlemenu()**

Used to display options on right mouse button click.

➔ **void init()**

Makes initial settings.

➔ **void menuinit()**

Used to display options on right mouse button click.

➔ **void setfont()**

Sets font size.

➔ **void playscreen()**

It is used to display the screen to play.

4.3 Results and snapshots

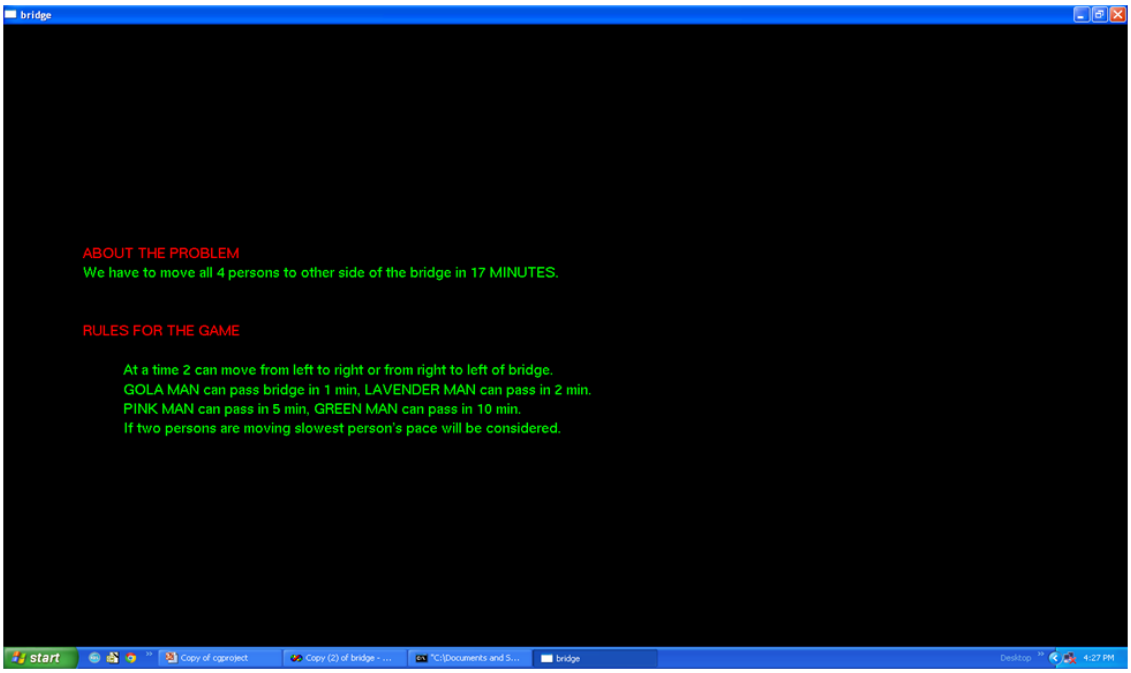


Figure 4.2 Rules displaying screen.

This figure shows the rules of the game. This screen will be displayed on clicking ‘about the puzzle’ option.

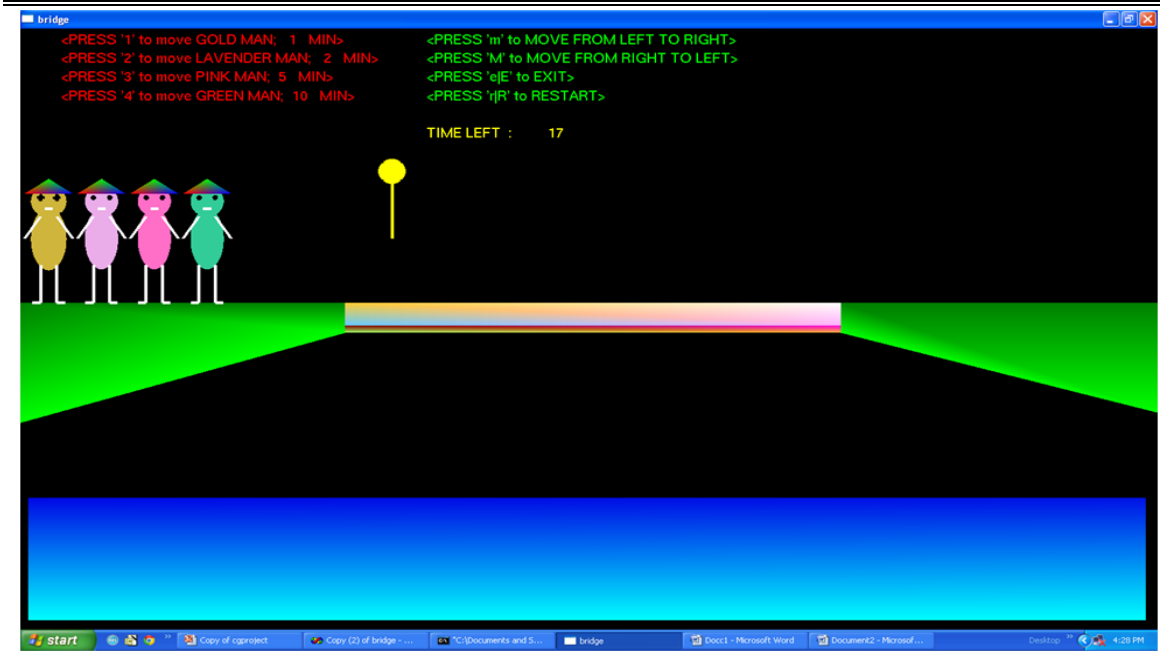


Figure 4.3 Initial condition

This figure shows the initial condition of the game.

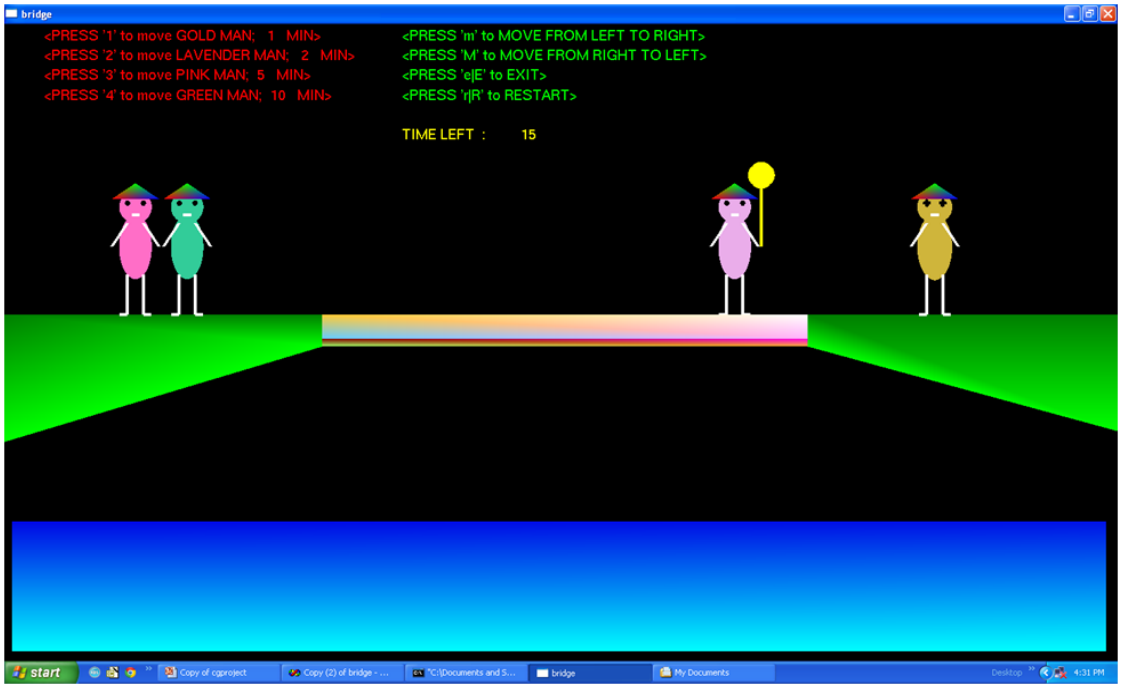


Figure 4.4 Working condition

This figure is displayed when the user is playing the game.

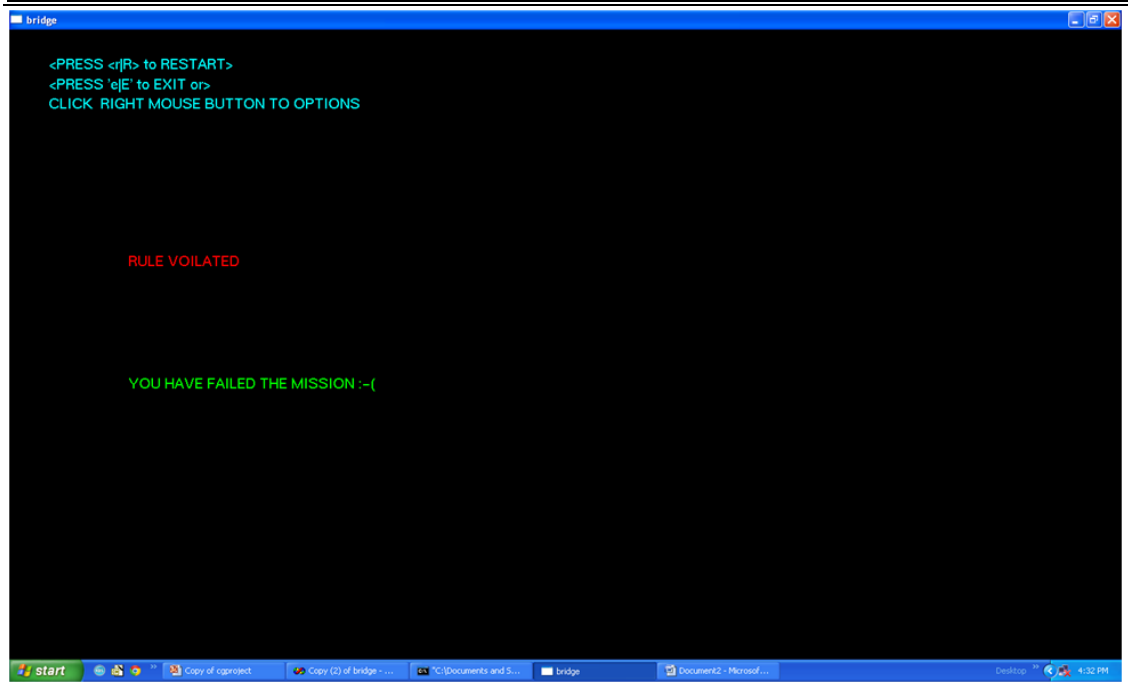


Figure 4.5: Rule violation message display

This screen is displayed when user violates the rule and game will be over.

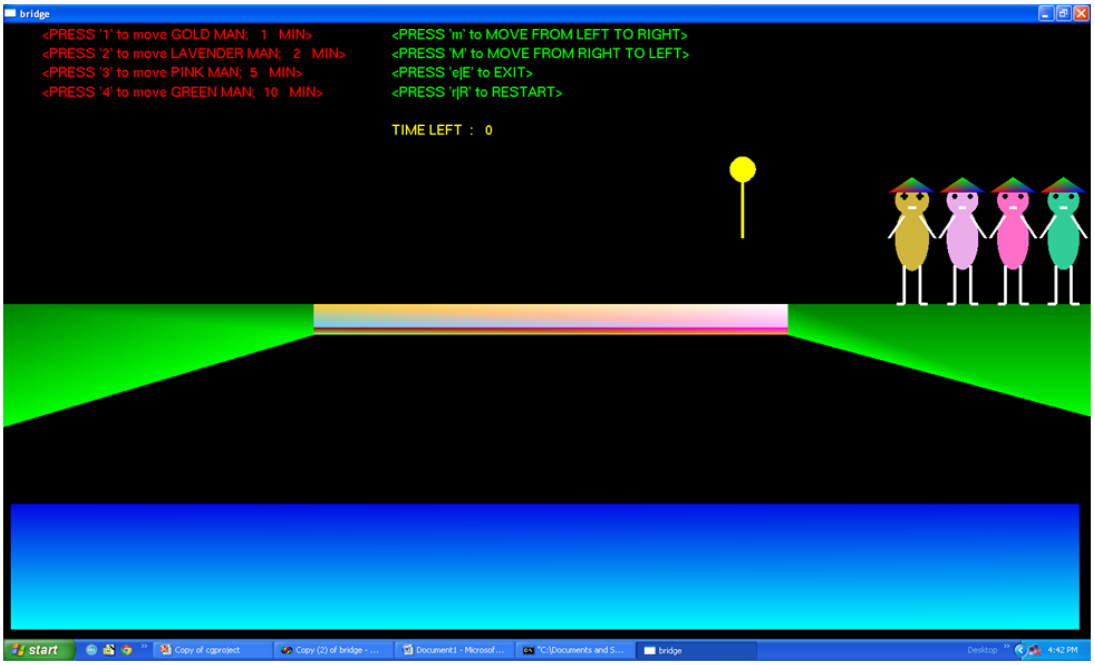


Figure 4.6 Puzzle solved

This figure will displayed when the user solves the puzzle.

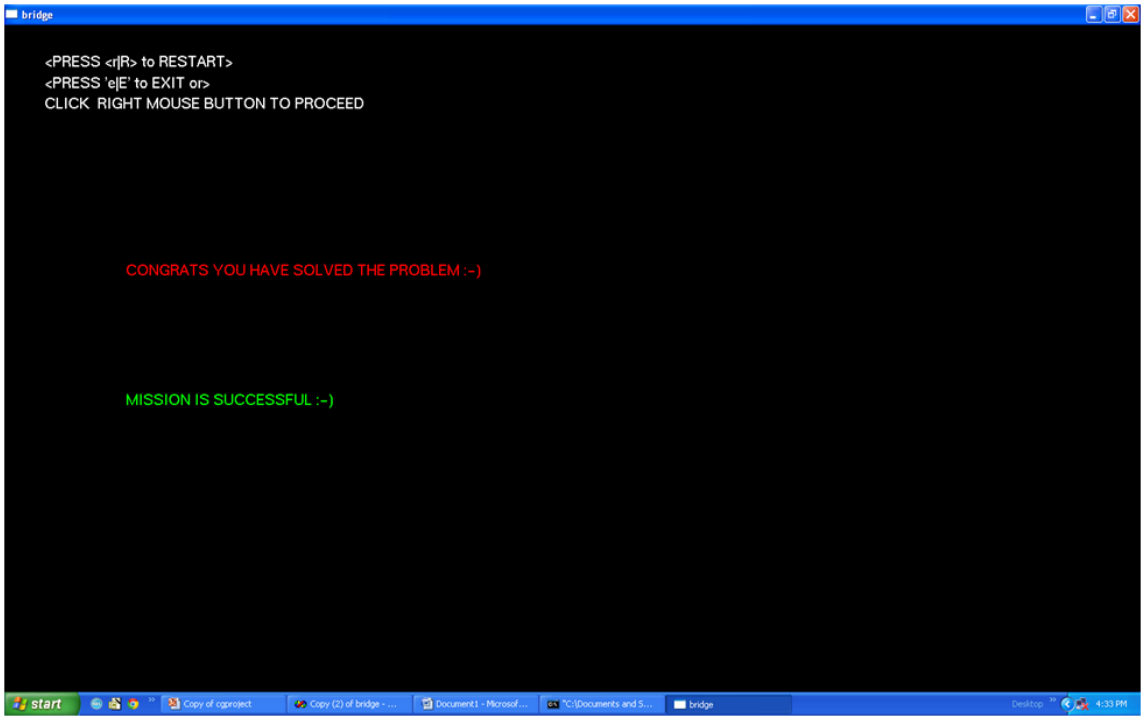


Figure 4.7: Puzzle solved message display

This screen is displayed when the user successfully solves the puzzle.

CHAPTER 5

CONCLUSION AND FUTURE SCOPE

5.1 Conclusion

The concept of solving 17 minutes bridge puzzle is well understood by applying different logical method. In this puzzle the basic method of hit and trial is used in which just go on checking the logic frequently until we don't obtain the desired result.

5.2 Future scope

In future, object can redesigned with 3D viewing with individual shading which can make them realistic. Mouse can be implemented in order to have the movement of the objects. Sound effect can be introduced in the movement of the objects.

REFERENCES

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 Using OpenGL 3E, 3rd edition, By F. S. Hill, Jr. and Stephen M. Kelley, Jr.