Abstract

The Graphics Package is developed using Open GL libraries. OpenGL provides a powerful but primitive set of rendering commands. It is an open specification for an applications program interface for defining 2D and3D objects. OpenGL is a low-level, procedural API, requiring the programmer to dictate the exact steps required to render a scene. OpenGL's low-level design requires programmers to have a good knowledge of the graphics pipeline, but also gives a certain amount of freedom to implement novel rendering algorithms. It is made up by using various algorithms like clipping algorithm, floodfill Scan-line algorithm, etc. The package has been developed using Visual Studio 2008. The various features have been supported by an easy to understand mouse and keyboard interface. All the tools have been organized under the toolbar. The color palette has been organized under the color palette. The drawing work is carried out in the sub window which is a part of the main window. The drawing features supported are Line, Rectangle, Brush, freehand drawing, curve, text, clipping, eraser and selection.

This is a graphics editor that enables the user to input graphical data. Using the editor, the user can also save the information input by him/her into files or open existing files for editing. This editor provides a graphical user-friendly interface to create and edit the files. In keeping with the low-end users, help files have been provided for each of the actions.

The main objective of the editor is to help the user to input graphical data and edit it conveniently. For ease in input and to help the user to traverse through the text easily, the editor provides functionality through the mouse. This project aims to develop a 2-D graphics package which supports basic operations which include creating objects like lines, circles, polygons, spirals, etc and also transformation operations like translation, rotation, etc on such objects. The package must also have a user-friendly interface that may be menu-oriented, iconic or a combination of both.

1. INTRODUCTION

OpenGL is an open specification for an applications program interface for defining 2D and3D objects. The specification is cross-language, cross-platform API for writing applications that produce 2D and 3D computer graphics. It renders 3D objects to the screen, providing the same set of instructions on different computers and graphics adapters. Thus it allows us to write an application that can create the same effects in any operating system using any OpenGL-adhering graphics adapter.

Computer graphics, a 3-dimensional primitive can be anything from a single point to an nsided polygon. From the software standpoint, primitives utilize the basic 3-dimensional rasterization algorithms such as Bresenham's line drawing algorithm, polygon scan line fill, texture mapping and so forth. OpenGL's basic operation is to accept primitives such as points, lines and polygons, and convert them into pixels. This is done by a graphics pipeline known as the OpenGL state machine. Most OpenGL commands either issue primitives to the graphics pipeline, or configure how the pipeline processes these primitives.

OpenGL is a low-level, procedural API, requiring the programmer to dictate the exact steps required to render a scene. OpenGL's low-level design requires programmers to have a good knowledge of the graphics pipeline, but also gives a certain amount of freedom to implement novel rendering algorithms.

1.1 Purpose

The aim of this project is to develop a 2-D graphics package which supports basic operations which include creating objects like lines, circles, polygons, spirals, etc. and also transformation operations like translation, rotation, etc. on such objects. The package must also have a user-friendly interface that may be menu-oriented, iconic or a combination of both.

1.2 Introduction on OpenGL

OpenGL provides the programmer with an interface to graphics hardware. It is a powerful, low-level rendering and modeling software library, available on all major platforms, with wide hardware support. It is designed for use in any graphics applications, from games to modeling to CAD.

OpenGL intentionally provides only low-level rendering routines, allowing the programmer a great deal of control and flexibility. The provided routines can easily be used to build high-level rendering and modeling libraries, and in fact, the OpenGL Utility Library (GLU), which is included in most OpenGL distributions, does exactly that. Note also that OpenGL is just a graphics library; unlike DirectX, it does not include support for sound, input, networking, or anything else not directly related to graphics.

1.2.1 OpenGL History

OpenGL was originally developed by Silicon Graphics, Inc. (SGI) as a multi-purpose, platform-independent graphics API. Since 1992, the development of OpenGL has been overseen by the OpenGL Architecture Review Board (ARB), which is made up of major graphics vendors and other industry leaders, currently consisting of ATI, Compaq, Evans & Sutherland, Hewlett-Packard, IBM, Intel, Intergraph, nVidia, Microsoft, and Silicon Graphics. The role of the ARB is to establish and maintain the OpenGL specification, which dictates which features must be included when one is developing an OpenGL distribution.

Because OpenGL is designed to be used with high-end graphics workstations, it has, until recently, included the power to take full advantage of consumer-level graphics hardware. Furious competition over the last couple of years, however, has brought features once available only on graphics workstations to the consumer level; as a result, there are more and more video cards of which OpenGL can't take full advantage. Eventually, these extensions may become official additions to the OpenGL standard. OpenGL 1.2 was the first version to contain support for features specifically requested by game developers (such as multitexturing), and it is likely that future releases will be influenced by gaming as well.

1.2.2 OpenGL Architecture

OpenGL is a collection of several hundred functions providing access to all the features offered by your graphics hardware. Internally, it acts as a state machine--a collection of states that tell OpenGL what to do. Using the API, you can set various aspects of the state machine, including such things as the current color, lighting, blending, and so on. When rendering, everything drawn is affected by the current settings of the state machine. It's important to be aware of what the various states are, and the effect they have, because it's not uncommon to have unexpected results due to having one or more states set incorrectly.

At the core of OpenGL is the rendering pipeline, as shown in Figure 2.1. You don't need to understand everything that happens in the pipeline at this point, but you should at least be aware that what you see on the screen results from a series of steps. Fortunately, OpenGL handles most of these steps for you.

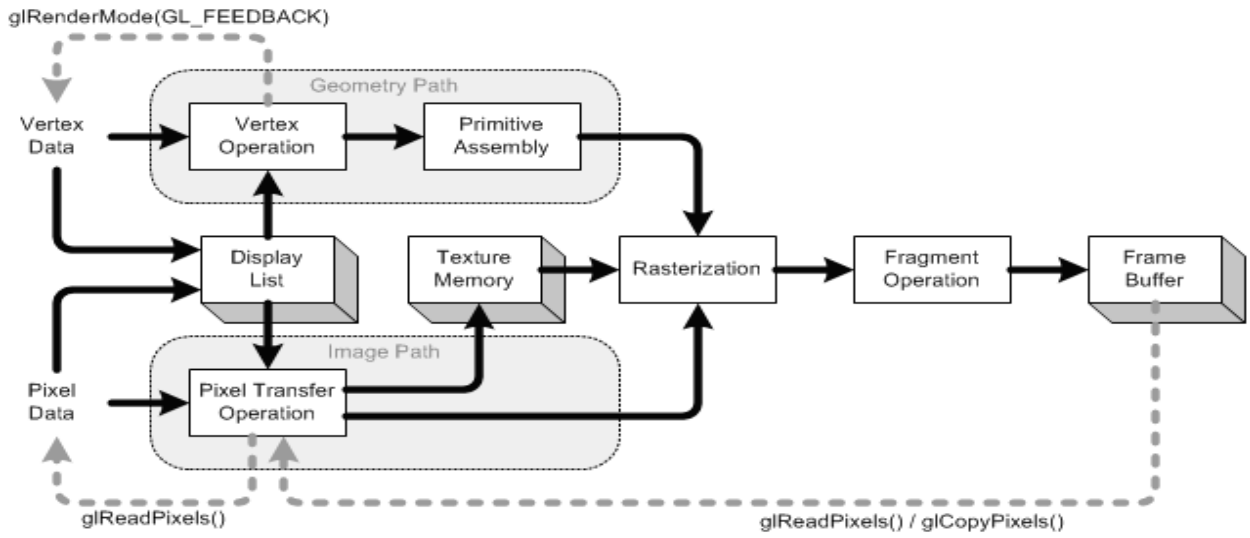

Fig 1.1 The OpenGL rendering pipeline.

Under Windows, OpenGL provides an alternative to using the Graphics Device Interface (GDI). GDI architects designed it to make the graphics hardware entirely invisible to

Windows programmers. This provides layers of abstraction that help programmers avoid dealing with device-specific issues.

However, GDI is intended for use with applications and thus lacks the speed required for games. OpenGL allows you to bypass GDI entirely and deal directly with graphics hardware. Figure 2.2 illustrates the OpenGL hierarchy under Windows.

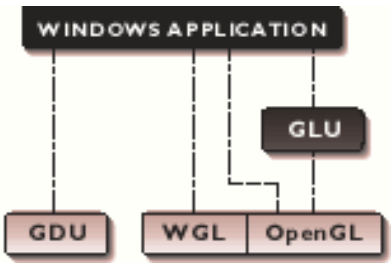

Fig 1.2 OpenGL API hierarchy under Windows systems.

1.2.3 The OpenGL Utility Library

The OpenGL Utility Library, or GLU, supplements OpenGL by providing higher-level functions. GLU offers features that range from simple wrappers around OpenGL functions tocomplex components supporting advanced rendering techniques. Its features include:

- 2D image scaling
- Rendering 3D objects including spheres, cylinders, and disks
- Automatic mipmap generation from a single image
- Support for curves surfaces through NURBS
- Support for tessellation of non-convex polygons
- Special-purpose transformations and matrices

1.2.4 What Is GLUT?

GLUT, short for OpenGL Utility Toolkit, is a set of support libraries available on every major platform. OpenGL does not directly support any form of windowing, menus, or input. That's where GLUT comes in. It provides basic functionality in all of those areas, while remaining platform independent, so that you can easily move GLUT-based applications from, for example, Windows to UNIX with few, if any, changes.

GLUT is easy to use and learn, and although it does not provide you with all the functionality the operating system offers, it works quite well for demos and simple applications.

1.3 Significance of OpenGL

- With different 3D accelerators, by presenting the programmer To hide the complexities of interfacing with a single, uniform API.
- To hide the differing capabilities of hardware platforms, by requiring that all implementations support the full OpenGL feature set (using software emulation if necessary).
- OpenGL is a well-documented API.
- OpenGL is also a clean API and much easier to learn and program.
- OpenGL has the best demonstrated 3D performance for any API.
- OpenGL has a conformance suite to validate that OpenGL implementations correctly implement OpenGL.

1.4 Design Aspects of the Project Using OpenGL

The Graphics Package is designed using the in built graphics library. The objects, which can be drawn using the editor, are stored as functions that can be used according to the requirements.

We can say that based on the design philosophy used during their implementation, the graphics editors can be of two main types:

One is an Object oriented editor where in each thing drawn in the view port is an object. Such objects can be selected individually and can be subjected to any of the transformations provided in the editor. The advantage of such editors is that the code can be easily written in using an Object Oriented Programming language like C++. Also undo functionalities can be easily implemented because all that the editor has to do is to keep a stack of objects being drawn on the screen. The disadvantage is that the user can only select objects and not a part of the screen.

The other kind of an editor is a pixel-based editor where in drawing anything on the view port is like painting on a canvas. Once an object is drawn it cannot be individually selected. Instead only a rectangular portion of the screen can be usually selected and subjected to various transformations or other operations. In other words the smallest object that can be selected and modified is a pixel. The basic advantage is that of the simplicity in code of such an editor where in the smallest unit is a pixel. The disadvantage being that an individual object cannot be selected and subjected to transformations.

Given the advantages and disadvantages of the two ways of implementing, the programmer is free to choose one that is more appealing to him .The end product needs to be a user-friendly interface. Ease of understanding and speed of working are two main requirements for it, which should be kept in mind during each phase of design and implementation.

2. Software Requirement Specifications:

2.1 Overall Description

2.1.1 Product Perspective

This is a graphics editor that enables the user to input graphical data. Using the editor, the user can also save the information input by him/her into files or open existing files for editing. This editor provides a graphical user - friendly interface to create and edit the files.

2.1.2 Product Functions

As mentioned previously, the main objective of the editor is to help the user to input graphical data and edit it conveniently. For ease in input and to help the user to traverse through the text easily, the editor provides functionality through the mouse.

2.1.3 User Characteristics

The editor provides a very easy-to-use interface and does not expect any extra technical knowledge from the user. A basic understanding of all the options provided in the editor would facilitate him in using the editor to the best possible extent. Since it is a mouse-driven interface it is sufficiently easy for any kind of end user to run it.

2.2 Specific Requirements

2.2.1 Software Requirements

- An MS-DOS based operating system like Windows 98, Windows 2000 or Windows XP is the platform required to develop the 2D and 3D graphics applications.
- A Visual C/C++ compiler is required for compiling the source code to make the executable file which can then be directly executed.
- A built in graphics library like glut and glut32, and header file like glut.h and also dynamic link libraries like glut and glut32 are required.

2.2.2 Hardware Requirements

The hardware requirements are very minimal and the software can run on most of the machines.

- Processor - Intel 486/Pentium processor or above.
- Processor Speed - 500 MHz or above
- RAM - 64MB or above Storage Space - 2 MB or above
- Monitor resolution - A color monitor with a minimum resolution of 640*480.

2.3 Supportability

- Good coding standards must be followed. The naming convention must be such that the names of the variables and functions used should indicate their purpose.
- One or two lines of documentation must be provided along with the functions to indicate what they are trying to achieve. Documentation must be provided for every module.

2.4 Design Constraints

- As the software is being built to run on a DOS platform, which gives access to a maximum of only 640kB of conventional memory, efficient use of the memory is very important.
- As the software needs to be run even on low-end machines the code should be efficient and optimal with the minimal redundancies.
- Needless to say, the editor should also be robust and fast.
- It is assumed that the standard output device, namely the monitor, supports colors.
- One of the assumptions made in the file saving and retrieval process is that the required file is in the current directory.

- The user's system is required to have the C++ compiler of the appropriate version.

- The system is also expected to have a mouse connected since most of the drawing and other graphical operations implemented assume the presence of a mouse.

2.5 Interfaces

This section deals with the interfaces that must be supported by the application. It includes the user interfaces that are to be implemented by the system.

2.5.1 User Interfaces

The interface for the editor requires for the user to have a mouse connected, and the corresponding drivers installed. This is because most of the implementation details require and presume the presence of a mouse. For the convenience of the user, there are palettes and icons displayed on the screen.

ICONS

The icons consists of the file saving and opening options. It also has the 'exit' option for termination of the editor's execution.

PALETTES

- Color palette, which displays the different colors available to the user. He/she can pick the required color by clicking on the particular block.

- Shapes palette, which displays the various shapes that can be drawn by the user. This typically consists of line, circle, rectangle and freehand. User can choose the required shape by clicking on the icon representing the shape.

- 2D Transformations palette gives options like translation and rotation for the 2D objects created on the canvas. Again the user has to click on the icon to select the particular transformation.

3. Detailed design

3.1 Window design

Graphics editor uses only one window for all its purposes. The part of the window is used as canvas for drawing , container for holding many tools like color palette, writing tools like pencil, brush etc…

One of the most important things is that window is portable and flexible. The positioning of controls inside the window is generalized. The controls are placed with respect to the relative values of the window width and height. This means that the editor is portable to any system having different resolutions.

3.2 Main Algorithm

The main algorithm for the graphics editor is the one shown below.

1. Initiate the OpenGL graphics mode.
2. Initiate the mouse interface and keyboard interfaces.
3. Draw the toolbar icons, menus, and the main window.
4. Perform any drawing or editing on the canvas using any of the tools.
5. Close the application after releasing any dynamically used resources are released.

As we see above specific functions are registered for mouse and keyboard events. These functions are generally call back functions which are called upon receiving specific events from mouse and keyboard. Upon receiving inputs from these functions , specific actions are performed based on the mouse position.

Data Structures

There are no other data structures explicitly used. This is mainly because, the editor is designed for area based operations and not object-based operations. So it is not necessary maintain information about objects drawn on the drawing area.

3.3 Other Algorithms

Various algorithms have been used in this editor to provide the functionalities it boasts of. Few of them have been explained here.

Free- hand Drawing

Free- hand drawing is a special case of polyline drawing where we have a large number of points obtained by continuously poling the mouse position during the time the mouse' left button is clicked.

Translation

Translation is done by adding the required amount of translation quantities to each of the points of the objects in the selected area. If P(x,y) be the a point and (tx, ty) translation quantities then the translated point is given by

P'(x,y) = p(x+tx,y+ty)

Scaling

The scaling operation on an object can be carried out for an object by multiplying each of the points (x,y) by the scaling factors sx, sy.

Newx = oldx*sx

Newy = oldy*sy

4. Implementation

4.1 Structures used

There is only one structure used for designing our editor which is used to store the screen co-ordinates values. The objects of these basic structures are used in various part of the design. The following listing shows them.

4.1.1 Buffer

Color is a basic structure for color parameters such as red, green, blue. The fig 4.1 shows the color structure. These structures objects are used in other basic functions to store color values required for the objects.

Fig 4.1 Color structure

4.2 Text

The text is written on the drawing canvas using one of the library function available in Open Gl "glutBitmapCharacter()".

Before drawing the text, the pointer is set to proper position using the opengl function call "glRasterPos2i()"

4.3 Button

The buttons in this project are actually state-less objects. They do not graphically respond to the user input. But there input has been recognized by mouse callback function based on the position where user clicks the button.

4.4 Color palette

Color palette is designed using the number of color buttons. The Fig 4.7 shows the color palette.The colors are generated randomly by varying the values of floating point numbers in the function “glColor3f()”. The color chosen by the user is recognized by the mouse callback function based on the co-ordinate points of the mouse.

Fig 4.1 Color palette

4.5 File options

File menu have the options like New, Open, Save, and exit. The New option will create a new blank window for drawing.

The Open option allows to open the file by accepting the filename in command prompt. The file is read by using the “fread” function to a buffer and the buffer is drawn to a screen by using “glDrawPixels” function.

The save option will basically save the contents of the buffer to a file using the “fwrite” system call. The exit option is used to quit the editor.

4.6 Function to draw objects using basic primitives

4.6.1 Function to draw a Pencil

Pencil is used to draw a stream of points on the canvas. A function called “draw_point” has been developed which basically draws a point. So this function is called repeatedly to imitate it as a pencil.

4.6.2 Function to draw a Line

Line drawing is done by using Opengl library functions and by passing the parameter “GL_LINES” to “glBegin()” function.

4.6.3 Function to draw a Rectangle

The above function is achieved by using the opengl library function to draw a line loop.

4.6.4 Function to Render a Text on the Screen

The text is written on the drawing canvas using one of the library function available in Open Gl "glutBitmapCharacter()".

4.6.5 Function to draw a Circle

Circle drawing is achieved by drawing points based on the angle and using circle's parametric equation.

$X=r\cos\theta$

$Y=r\sin\theta$

4.7 2D drawing

2D drawings are done in screen window that is used as canvas. The canvas is nothing but an empty space in a window where the drawing and various other actions are carried out.

4.8 Clipping

There are two types of clipping implemented

- Inside Clipping
- Outside Clipping

4.8.1 Inside Clipping

The area inside a selected region must be removed. And the rest of the area must be retained.

4.8.2 Outside Clipping

The area inside a selected region must be retained, rest all should be removed.

4.9 Window Management

Five routines perform tasks necessary to initialize a window.

- **glutInit**(int **argc*, char ***argv*) initializes GLUT and processes any command line arguments (for X, this would be options like -display and -geometry). **glutInit()** should be called before any other GLUT routine.

- **glutInitDisplayMode**(unsigned int *mode*) specifies whether to use an *RGBA* or color-index color model. You can also specify whether you want a single- or double-buffered window. (If you're working in color-index mode, you'll want to load certain colors into the color map; use **glutSetColor()** to do this.) Finally, you can use this routine to indicate that you want the window to have an associated depth, stencil, and/or accumulation buffer. For example, if you want a window with double buffering, the RGBA color model, and a depth buffer, you might call **glutInitDisplayMode**(*GLUT_DOUBLE* | *GLUT_RGB* | *GLUT_DEPTH*).

- **glutInitWindowPosition**(int *x*, int *y*) specifies the screen location for the upper-left corner of your window.

- **glutInitWindowSize**(int *width*, int *size*) specifies the size, in pixels, of your window.

- **glutCreateWindow**(char **string*) creates a window with an OpenGL context. It returns a unique identifier for the new window. Until **glutMainLoop()** is called the window is not yet displayed.

4.10 The Display Callback

- **glutDisplayFunc**(void (* *func*)(void)) is the first and most important event callback function you will see. Whenever GLUT determines the contents of the window need to be redisplayed, the callback function registered by **glutDisplayFunc()** is executed. Therefore, you should put all the routines you need to redraw the scene in the display callback function.

- **glutPostRedisplay**(void), which gives **glutMainLoop()** a nudge to call the registered display callback at its next opportunity. If your program changes the contents of the window, sometimes you will have to call this function.

4.11 Running the Program

The very last thing you must do is call **glutMainLoop**(void). All windows that have been created are now shown, and rendering to those windows is now effective. Event processing begins, and the registered display callback is triggered. Once this loop is entered, it is never exited!

4.12 OpenGL Geometric Drawing Primitives

Now that you've seen how to specify vertices, you still need to know how to tell OpenGL to create a set of points, a line, or a polygon from those vertices. To do this, you bracket each set of vertices between a call to **glBegin()** and a call to **glEnd()**. The argument passed to **glBegin()** determines what sort of geometric primitive is constructed from the vertices.

Point Details

To control the size of a rendered point, use **glPointSize()** and supply the desired size in pixels as theargument.

- *void **glLoadMatrix**{fd}(const TYPE *m);*

Sets the sixteen values of the current matrix to those specified by m.

- *void* ***glMultMatrix****{fd}(const TYPE *m);*

Multiplies the matrix specified by the sixteen values pointed to by m by the current matrix and stores the result as the current matrix.

- *void* ***glLoadIdentity****(void);*

Sets the currently modifiable matrix to the 4 × 4 identity matrix.

- *void* ***glMatrixMode****(GLenum mode);*

Specifies whether the modelview, projection, or texture matrix will be modified, using the mode attribute. Projecton matrix is used for 2-D purposes whereas Modelview is used for 3-D.

5. Conclusion

5.1 Summary

The editor assumes no detailed knowledge about computer on the part of the user other than the ability to operate a computer i.e. entering the data from the keyboard. The mouse interface and keyboard interface makes it possible for even a novice user to use the graphics editor.

5.2 Limitations

- Multiple workspaces are not provided: It is not possible for a user to open multiple workspaces or multiple documents on which he/she can work on. At a time the user can work on or edit only one document.
- More than one file cannot be opened simultaneously: Similar to workspaces, user can work with (open) only one file at a time. This restriction brings down the user-friendliness of the package.
- Concentrated more on 2D: The main focus of this package has been to perform more of 2D operations. This has led to the neglecting of many 3D components which are sometimes necessary for some user operations.

5.3 Future Enhancements

- Pattern filling can be improved.
- Making the package more user friendly: A better user interface can be given to improve the package. The interface can be designed to be more intuitive.
- Can be enhanced for the efficient usage of memory: As the application is scaled to a higher end to meet higher requirements memory management becomes a core issue. A small prototype like package though may not require the programmer to make efficient use of memory, it can be done in order to allow the package to be scaled to a higher and better version.

Appendix A - Source Code

Paint.cpp

```
#include<stdio.h>
#include<math.h>
#include<stdlib.h>
#include<gl/glut.h>

#define PENCIL 1
#define BRUSH 2
#define LINE 3
#define RECT 4
#define FILLRECT 5
#define SPRAY 6
#define CIRCLE 7
#define ERAZE 8
#define QUAD 9
#define NEW 10
#define OPEN 11
#define SAVE 12
#define EXIT 13
#define INSIDECLIP 14
#define OUTSIDECLIP 15
#define TRANSLATE 16
#define SCALING 17
#define SPIRAL 18
#define ERAZE1 19
#define ERAZE2 20
#define ERAZE3 21
#define TRANSLATE1 22
#define MAXX 1018
#define MAXY 700
#define CARDIOID 23
#define LIMACON 24



int linex=0,liney=0,elinex=0,eliney=0,clr=0,count=0;
int
drawline,drawrect,drawcircle,insideclip,outsideclip,translate,scaling,trans_paste,scale_paste,translat
e1,trans_paste1,rotate,reflect,dspiral;
int ax,ay,bx,by,px,py;
int state1,size=20,ret,savec,openc;
float arr[MAXX-50][MAXY-50][3],mat[MAXX-50][MAXY-40][3],clip[MAXX-50][MAXY-40][3],mat1[MAXX-50][MAXY-
40][3], arr1[MAXX-50][MAXY-50][3];
GLfloat
colors[18][3]={{0.0,0.0,0.0},{1.0,0.0,0.0},{0.0,1.0,0.0},{0.0,0.0,1.0},{1.0,1.0,0.0},{1.0,0.0,1.0},{0.
0,1.0,1.0},{1.0,1.0,1.0},{0.75,0.8,0.9},{0.6,0.2,0.7},{0.3,0.3,0.3},{0.1,0.55,0.1},{1.0,0.3,0.0},{0.5,
0.4,1},{0.6,0.0,0.1},{0.5,0.2,0.1},{0.7,0.7,0.7},{0.85,0.89,1.0}};
long matsize=(MAXX-50)*(MAXY-40)*3;
FILE *fp;
int screenwidth=MAXX,screenheight=MAXY;


void lineloop(int x1,int y1,int x2,int y2,int x3,int y3,int x4,int y4)
{
	glBegin(GL_LINE_LOOP);
	glVertex2i(x1,y1);
```

```
    glVertex2i(x2,y2);
       glVertex2i(x3,y3);
       glVertex2i(x4,y4);
       glEnd();
       glFlush();
}

void polygon(int x1,int y1,int x2,int y2,int x3,int y3,int x4,int y4)
{
       glBegin(GL_POLYGON);
       glVertex2i(x1,y1);
       glVertex2i(x2,y2);
       glVertex2i(x3,y3);
       glVertex2i(x4,y4);
       glEnd();
       glFlush();
}

void line(int x1,int y1, int x2, int y2)
{
       glBegin(GL_LINES);
       glVertex2i(x1,y1);
       glVertex2i(x2,y2);
       glEnd();
       glFlush();
}

void draw_text(char *info,int i,int j)
{
       glRasterPos2i(i,j);
       while(*info)
       {
              glutBitmapCharacter(GLUT_BITMAP_HELVETICA_10,*info++);
       }
}

void boxes(int x,int y,GLfloat color1[])
{
       glColor3fv(color1);
       polygon(970+5,x,970+5,y,970+45,y,970+45,x);
}
void boxes1(int x,int y,GLfloat color1[])
{
       glColor3fv(color1);
       polygon(5,x,5,y,45,y,45,x);
}
void triangle(int x1,int y1,int x2,int y2,int x3,int y3)
{
       glBegin(GL_TRIANGLES);
       glVertex2i(x1,y1);
       glVertex2i(x2,y2);
       glVertex2i(x3,y3);
       glEnd();
}
void boxes2(int x,int y,GLfloat color1[])
{
       glColor3fv(color1);
       triangle(970+5,x,970+45,y,970+45,x);
}
void boxes3(int x,int y,GLfloat color1[])
{
       glColor3fv(color1);
       triangle(5,x,45,y,45,x);
```

```
}
void boxes4(int x,int y,GLfloat color1[])
{
	glColor3fv(color1);
	triangle(970+5,x,970+45,y,970+45,x);
	glColor3f(1,0.7,0.8);
			polygon(970+20,365,970+20,370,970+30,370,970+30,365);
			polygon(970+15,360-35,970+15,375-35,970+35,375-35,970+35,360-35);
			polygon(970+10,355-70,970+10,380-70,970+40,380-70,970+40,355-70);
}

void menu(int x1, int x2, int y1, int y2, GLfloat color1[])
{
	glColor3fv(color1);
	polygon(x1,y1,x1,y2,x2,y2,x2,y1);
}

int inside_area(int left,int right,int bottom,int top,int x,int y)
{
	if(x>left && x<right)
		if(y<top && y>bottom)
			return true;
	return false;
}

void draw_pixel(GLint cx,GLint cy)
{
	glBegin(GL_POINTS);
	glVertex2i(cx,cy);
	glEnd();

}

void plot_pixels(GLint h,GLint k,GLint x,GLint y)
{
	draw_pixel(x+h,y+k);
	draw_pixel(-x+h,y+k);
	draw_pixel(x+h,-y+k);
	draw_pixel(-x+h,-y+k);
	draw_pixel(y+h,x+k);
	draw_pixel(-y+h,x+k);
	draw_pixel(y+h,-x+k);
	draw_pixel(-y+h,-x+k);
}

void circle(GLint h,GLint k,GLint r)
{
	GLint d=1-r,x=0,y=r;
	while(y>x)
	{
		plot_pixels(h,k,x,y);
		if(d<0)		d+=2*x+3;
		else
		{
			d+=2*(x-y)+5;
			--y;
		}
		++x;
	}
	plot_pixels(h,k,x,y);

}
```

```
void rect_draw()
{
	glColor3fv(colors[3]);
	lineloop(970+12,535,970+38,535,970+38,550,970+13,550);
}

void fill_draw()
{
	glColor3fv(colors[0]);
	polygon(970+12,500,970+38,500,970+38,515,970+12,515);
}


void circle_draw()
{
	glColor3fv(colors[0]);
	circle(970+25,437.5,9);
}

void pencil()
{
	glColor3fv(colors[6]);
	polygon(970+20,650,970+40,650,970+40,643,970+20,643);
	glColor3fv(colors[0]);
	glBegin(GL_TRIANGLES);
	glVertex2f(970+7,647);
	glVertex2f(970+20,652);
	glVertex2f(970+20,643);
	glEnd();
	glColor3fv(colors[0]);
	glPointSize(3.0);
	lineloop(970+20,650,970+40,650,970+40,643,970+20,643);
	glEnd();
}

void putpixel(int x, int y)
{
	glBegin(GL_POINTS);
	glVertex2i(x,y);
	glEnd();
	glFlush();
}

void spiral(int cx, int cy)
{
	GLfloat t,x3,y3;
	for(t=0;t<6.28*5;t+=0.03)
	{
		x3=t*cos(t)*2+cx;
		y3=t*sin(t)*2+cy;
		putpixel(x3,y3);
		glFlush();
	}
}

void brush()
{
	glColor3f(0.4,0.4,1.0);
	polygon(970+25,608,970+35,618,970+30,623,970+20,613);
	glColor3f(1.0,1.0,0.7);
	polygon(970+25,608,970+20,613,970+12,609,970+21,601);
	glColor3fv(colors[0]);
	glBegin(GL_LINE_LOOP);
```

```
        glVertex2f(970+25,608);
        glVertex2f(970+20,613);
        glVertex2f(970+12,609);
        glVertex2f(970+21,601);
        glEnd();
}

void line()
{
        glColor3fv(colors[3]);
        glBegin(GL_LINE_LOOP);
        glVertex2f(970+17,570);
        glVertex2f(970+35,585);
        glEnd();
}

void eraser()
{
        glColor3f(1,0.7,0.8);
        polygon(970+18,395,970+32,395,970+32,409,970+18,409);
}

void spray_draw()
{
        glColor3fv(colors[3]);
        polygon(970+22,474,970+27,479,970+37,469,970+32,464);
        glBegin(GL_LINE_LOOP);
        glVertex2f(970+22,474);
        glVertex2f(970+27,478);
        glVertex2f(970+19,480);
        glEnd();
        glFlush();
        glColor3fv(colors[0]);
        glPointSize(1.0);
        glBegin(GL_POINTS);
        glVertex2i(970+16,478);
        glVertex2i(970+16,480);
        glVertex2i(970+14,476);
        glVertex2i(970+14,478);
        glVertex2i(970+14,480);
        glVertex2i(970+12,476);
        glVertex2i(970+12,480);
        glVertex2i(970+12,478);
        glVertex2i(970+10,476);
        glVertex2i(970+10,478);
        glVertex2i(970+10,480);
        glVertex2i(970+17,480);
        glEnd();
        glFlush();
}

void disp()
{
        pencil();
        brush();
        line();
        eraser();
        spray_draw();
        circle_draw();
        rect_draw();
        fill_draw();
        glColor3fv(colors[3]);
        draw_text("Inside",12,368);
```

```
        draw_text("Clip",15,358);
        draw_text("Outside",7,333);
        draw_text("Clip",15,323);
        draw_text("Translate",5,293);
        draw_text("Scaling",8,258);
        draw_text("Cardiod",5,223);
        draw_text("Limacon",3,190);
        draw_text("Draw",13,158);
        draw_text("Spiral",12,146);
        glColor3fv(colors[clr]);
        polygon(10,10,10,40,40,40,40,10);
}

void menu_disp()
{
        glColor3f(0.0,0.0,0.0);
        draw_text("New",13,MAXY-20);
    draw_text("Open",60,MAXY-20);
    draw_text("Save",110,MAXY-20);
    draw_text("Exit",160,MAXY-20);
    draw_text("Cut",210,MAXY-15);
        draw_text("Paste",210,MAXY-25);
        draw_text("Copy",260,MAXY-15);
        draw_text("Paste",260,MAXY-25);

        glFlush();
}

void spray(int x,int y)
{
        int randx,randy;
        int x1,y1;
        x1=x+10;
        y1=y+15;
        glBegin(GL_POINTS);
        for(int i=0;i<50;i++)
        {
                randx=x1-size+2/2+rand()%(size/2);
                randy=y1-size+2/2+rand()%(size/2);
                glVertex2i(randx,randy);
        }
        glFlush();
        glEnd();
}

void mouse(int button,int state,int x,int y)
{
        if(state==GLUT_DOWN && button==GLUT_LEFT_BUTTON)
        {
                y=MAXY-y;
                if(inside_area(50,75,5,25,x,y))
                {
                        clr=7;
                        glColor3fv(colors[7]);
                        polygon(10,10,10,40,40,40,40,10);
                        return;
                }
                if(inside_area(200+50,200+75,5,25,x,y))
                {
                        clr=0;
                        glColor3fv(colors[0]);
                        polygon(10,10,10,40,40,40,40,10);
                        return;
```

```
		}

	if(inside_area(75,100,5,25,x,y))
		{
			clr=16;
			glColor3fv(colors[16]);
			polygon(10,10,10,40,40,40,40,10);
			return;
		}
		if(inside_area(200+75,200+100,5,25,x,y))
		{
			clr=10;
			glColor3fv(colors[10]);
			polygon(10,10,10,40,40,40,40,10);
			return;
		}
		if(inside_area(100,125,5,25,x,y))
		{
			clr=6;
			glColor3fv(colors[6]);
			polygon(10,10,10,40,40,40,40,10);
			return;
		}
		if(inside_area(200+100,200+125,5,25,x,y))
		{
			clr=3;
			glColor3fv(colors[3]);
			polygon(10,10,10,40,40,40,40,10);
			return;
		}
		if(inside_area(125,150,5,25,x,y))
		{
			clr=2;
			glColor3fv(colors[2]);
			polygon(10,10,10,40,40,40,40,10);
			return;
		}
		if(inside_area(200+125,200+150,5,25,x,y))
		{
			clr=11;
			glColor3fv(colors[11]);
			polygon(10,10,10,40,40,40,40,10);
			return;
		}
		if(inside_area(150,175,5,25,x,y))
		{
			clr=13;
			glColor3fv(colors[13]);
			polygon(10,10,10,40,40,40,40,10);
			return;
		}
		if(inside_area(200+150,200+175,5,25,x,y))
		{
			clr=9;
			glColor3fv(colors[9]);
			polygon(10,10,10,40,40,40,40,10);
			return;
		}
		if(inside_area(175,200,5,25,x,y))
		{
			clr=15;
			glColor3fv(colors[15]);
			polygon(10,10,10,40,40,40,40,10);
```

```
        return;
    }
    if(inside_area(200+175,200+200,5,25,x,y))
    {
        clr=14;
        glColor3fv(colors[14]);
        polygon(10,10,10,40,40,40,40,10);
        return;
    }
    if(inside_area(200,225,5,25,x,y))
    {
        clr=5;
        glColor3fv(colors[5]);
        polygon(10,10,10,40,40,40,40,10);
        return;
    }
    if(inside_area(200+200,200+225,5,25,x,y))
    {
        clr=1;
        glColor3fv(colors[1]);
        polygon(10,10,10,40,40,40,40,10);
        return;
    }
    if(inside_area(225,250,5,25,x,y))
    {
        clr=4;
        glColor3fv(colors[4]);
        polygon(10,10,10,40,40,40,40,10);
        return;
    }
    if(inside_area(200+225,200+250,5,25,x,y))
    {
        clr=12;
        glColor3fv(colors[12]);
        polygon(10,10,10,40,40,40,40,10);
        return;
    }
    int i=65,j=40;
    if(inside_area(970+5,970+45,635,660,x,y))
    {
        glColor3fv(colors[17]);
        polygon(970,280,970,385,970+45,385,970+45,280);
        polygon(620,20,620,45,MAXX-5,45,MAXX-5,20);
        boxes2(635,660,colors[8]);
        boxes(600,625,colors[7]);
      boxes(565,590,colors[7]);
        boxes(530,555,colors[7]);
        boxes(495,520,colors[7]);
        boxes(460,485,colors[7]);
        boxes(425,450,colors[7]);
        boxes(390,415,colors[7]);
        boxes1(355,380,colors[7]);
        boxes1(320,345,colors[7]);
        boxes1(285,310,colors[7]);
        boxes1(250,275,colors[7]);
        glColor3f(0,0,0);
        draw_text("PENCIL - used for free hand drawing",625,30);
        disp();
        glFlush();
        state1=PENCIL;return;
    }
    i+=35;j+=35;
    if(inside_area(970+5,970+45,MAXY-i,MAXY-j,x,y))
```

```
		{
			glColor3fv(colors[17]);
			polygon(970,280,970,385,970+45,385,970+45,280);
			polygon(620,20,620,45,MAXX-5,45,MAXX-5,20);
			boxes(635,660,colors[7]);
			boxes2(600,625,colors[8]);
		    boxes(565,590,colors[7]);
			boxes(530,555,colors[7]);
			boxes(495,520,colors[7]);
			boxes(460,485,colors[7]);
			boxes(425,450,colors[7]);
			boxes(390,415,colors[7]);
			boxes1(355,380,colors[7]);
			boxes1(320,345,colors[7]);
			boxes1(285,310,colors[7]);
			boxes1(250,275,colors[7]);
			glColor3f(0,0,0);
			draw_text("BRUSH - used for free hand drawing with size larger than
pencil",625,30);
			disp();
			glFlush();
			state1=BRUSH;return;
		}
		i+=35;j+=35;
	if(inside_area(970+5,970+45,MAXY-i,MAXY-j,x,y))
		{
			glColor3fv(colors[17]);
			polygon(970,280,970,385,970+45,385,970+45,280);
			polygon(620,20,620,45,MAXX-5,45,MAXX-5,20);
			boxes(635,660,colors[7]);
			boxes(600,625,colors[7]);
		    boxes2(565,590,colors[8]);
			boxes(530,555,colors[7]);
			boxes(495,520,colors[7]);
			boxes(460,485,colors[7]);
			boxes(425,450,colors[7]);
			boxes(390,415,colors[7]);
			boxes1(355,380,colors[7]);
			boxes1(320,345,colors[7]);
			boxes1(285,310,colors[7]);
			boxes1(250,275,colors[7]);
			glColor3f(0,0,0);
			draw_text("LINE - used to draw a line of any length between two desired
points",625,30);
			disp();
			glFlush();
			state1=LINE;return;
		}
		i+=35;j+=35;
		if(inside_area(970+5,970+45,MAXY-i,MAXY-j,x,y))
		{
			glColor3fv(colors[17]);
			polygon(970,280,970,385,970+45,385,970+45,280);
			polygon(620,20,620,45,MAXX-5,45,MAXX-5,20);
			boxes(635,660,colors[7]);
			boxes(600,625,colors[7]);
		    boxes(565,590,colors[7]);
			boxes2(530,555,colors[8]);
			boxes(495,520,colors[7]);
			boxes(460,485,colors[7]);
			boxes(425,450,colors[7]);
			boxes(390,415,colors[7]);
			boxes1(355,380,colors[7]);
```

```
                        boxes1(320,345,colors[7]);
                        boxes1(285,310,colors[7]);
                        boxes1(250,275,colors[7]);
                        glColor3f(0,0,0);
                        draw_text("RECTANGLE - used to draw empty rectangle of desired size",625,30);
                        disp();
                        glFlush();
                        state1=RECT;return;
                }
                i+=35;j+=35;
                if(inside_area(970+5,970+45,MAXY-i,MAXY-j,x,y))
                {
                        glColor3fv(colors[17]);
                        polygon(970,280,970,385,970+45,385,970+45,280);
                        polygon(620,20,620,45,MAXX-5,45,MAXX-5,20);
                        boxes(635,660,colors[7]);
                        boxes(600,625,colors[7]);
                    boxes(565,590,colors[7]);
                        boxes(530,555,colors[7]);
                        boxes2(495,520,colors[8]);
                        boxes(460,485,colors[7]);
                        boxes(425,450,colors[7]);
                        boxes(390,415,colors[7]);
                        boxes1(355,380,colors[7]);
                        boxes1(320,345,colors[7]);
                        boxes1(285,310,colors[7]);
                        boxes1(250,275,colors[7]);
                        glColor3f(0,0,0);
                        draw_text("FILLED RECTANGLE - used to draw rectangle filled with selected color
",625,30);
                        disp();
                        glFlush();
                        state1=FILLRECT;return;
                }
                i+=35;j+=35;
                if(inside_area(970+5,970+45,MAXY-i,MAXY-j,x,y))
                {
                        glColor3fv(colors[17]);
                        polygon(970,280,970,385,970+45,385,970+45,280);
                        polygon(620,20,620,45,MAXX-5,45,MAXX-5,20);
                        boxes(635,660,colors[7]);
                        boxes(600,625,colors[7]);
                    boxes(565,590,colors[7]);
                        boxes(530,555,colors[7]);
                        boxes(495,520,colors[7]);
                        boxes2(460,485,colors[8]);
                        boxes(425,450,colors[7]);
                        boxes(390,415,colors[7]);
                        boxes1(355,380,colors[7]);
                        boxes1(320,345,colors[7]);
                        boxes1(285,310,colors[7]);
                        boxes1(250,275,colors[7]);
                        glColor3f(0,0,0);
                        draw_text("SPRAY - press + to increase size , press - to decrease size",625,30);
                        disp();
                        glFlush();
                        state1=SPRAY;return;
                }
                i+=35;j+=35;
                if(inside_area(970+5,970+45,MAXY-i,MAXY-j,x,y))
                {
            glColor3fv(colors[17]);
                        polygon(970,280,970,385,970+45,385,970+45,280);
```

```
                polygon(620,20,620,45,MAXX-5,45,MAXX-5,20);
                boxes(635,660,colors[7]);
                boxes(600,625,colors[7]);
            boxes(565,590,colors[7]);
                boxes(530,555,colors[7]);
                boxes(495,520,colors[7]);
                boxes(460,485,colors[7]);
                boxes2(425,450,colors[8]);
                boxes(390,415,colors[7]);
                boxes1(355,380,colors[7]);
                boxes1(320,345,colors[7]);
                boxes1(285,310,colors[7]);
                boxes1(250,275,colors[7]);
                glColor3f(0,0,0);
                draw_text("CIRCLE - used to draw circle of desired radius",625,30);
                disp();
                glFlush();
                state1=CIRCLE;
                return;
        }
        i+=35;j+=35;
        int ii=i,jj=j;
        if(inside_area(970+5,970+45,MAXY-i,MAXY-j,x,y))
        {
                glColor3fv(colors[17]);
                polygon(620,20,620,45,MAXX-5,45,MAXX-5,20);
                boxes(635,660,colors[7]);
                boxes(600,625,colors[7]);
            boxes(565,590,colors[7]);
                boxes(530,555,colors[7]);
                boxes(495,520,colors[7]);
                boxes(460,485,colors[7]);
                boxes(425,450,colors[7]);
                boxes2(390,415,colors[8]);
                boxes(355,380,colors[7]);
                boxes(320,345,colors[7]);
                boxes(285,310,colors[7]);

                glColor3f(1,0.7,0.8);
                polygon(970+20,365,970+20,370,970+30,370,970+30,365);
                polygon(970+15,360-35,970+15,375-35,970+35,375-35,970+35,360-35);
                polygon(970+10,355-70,970+10,380-70,970+40,380-70,970+40,355-70);

                boxes1(355,380,colors[7]);
                boxes1(320,345,colors[7]);
                boxes1(285,310,colors[7]);
                boxes1(250,275,colors[7]);
                glColor3f(0,0,0);
                draw_text("ERAZER - used to eraze parts of drawing",625,30);
                disp();
                glFlush();
                state1=ERAZE;
                return;
        }
        i+=35;j+=35;
    if(inside_area(970+5,970+45,MAXY-i,MAXY-j,x,y))
        {
                glColor3fv(colors[17]);
                polygon(620,20,620,45,MAXX-5,45,MAXX-5,20);
                boxes(635,660,colors[7]);
                boxes(600,625,colors[7]);
            boxes(565,590,colors[7]);
                boxes(530,555,colors[7]);
```

```
                boxes(495,520,colors[7]);
                boxes(460,485,colors[7]);
                boxes(425,450,colors[7]);
                boxes2(390,415,colors[8]);
                boxes4(355,380,colors[8]);
                boxes(320,345,colors[7]);
                boxes(285,310,colors[7]);
                glColor3f(1,0.7,0.8);
                //polygon(970+20,365,970+20,370,970+30,370,970+30,365);
                polygon(970+15,360-35,970+15,375-35,970+35,375-35,970+35,360-35);
                polygon(970+10,355-70,970+10,380-70,970+40,380-70,970+40,355-70);
                glColor3f(0,0,0);
                draw_text("SMALL ERAZER",625,30);
                disp();
                glFlush();
                state1=ERAZE1;
                return;
        }
i+=35;j+=35;
        if(inside_area(970+5,970+45,MAXY-i,MAXY-j,x,y))
        {
                glColor3fv(colors[17]);
                polygon(620,20,620,45,MAXX-5,45,MAXX-5,20);
                boxes(635,660,colors[7]);
                boxes(600,625,colors[7]);
            boxes(565,590,colors[7]);
                boxes(530,555,colors[7]);
                boxes(495,520,colors[7]);
                boxes(460,485,colors[7]);
                boxes(425,450,colors[7]);
                boxes2(390,415,colors[8]);
                boxes(355,380,colors[7]);
                boxes4(320,345,colors[8]);
                boxes(285,310,colors[7]);
                glColor3f(1,0.7,0.8);
                polygon(970+20,365,970+20,370,970+30,370,970+30,365);
                //polygon(970+15,360-35,970+15,375-35,970+35,375-35,970+35,360-35);
                polygon(970+10,355-70,970+10,380-70,970+40,380-70,970+40,355-70);
                glColor3f(0,0,0);
                draw_text("MEDIUM ERAZER",625,30);
                disp();
                glFlush();
                state1=ERAZE2;
                return;
        }
i+=35;j+=35;
        if(inside_area(970+5,970+45,MAXY-i,MAXY-j,x,y))
        {
                glColor3fv(colors[17]);
                polygon(620,20,620,45,MAXX-5,45,MAXX-5,20);
                boxes(635,660,colors[7]);
                boxes(600,625,colors[7]);
            boxes(565,590,colors[7]);
                boxes(530,555,colors[7]);
                boxes(495,520,colors[7]);
                boxes(460,485,colors[7]);
                boxes(425,450,colors[7]);
                boxes2(390,415,colors[8]);
                boxes(355,380,colors[7]);
                boxes(320,345,colors[7]);
                boxes4(285,310,colors[8]);
                glColor3f(1,0.7,0.8);
                polygon(970+20,365,970+20,370,970+30,370,970+30,365);
```

```
                polygon(970+15,360-35,970+15,375-35,970+35,375-35,970+35,360-35);
                //polygon(970+10,355-70,970+10,380-70,970+40,380-70,970+40,355-70);

                glColor3f(0,0,0);
                draw_text("LARGE ERAZER",625,30);
                disp();
                glFlush();
                state1=ERAZE3;
                return;
        }
ii+=35;jj+=35;
        if(inside_area(5,45,MAXY-ii,MAXY-jj,x,y))
        {
                glColor3fv(colors[17]);
                polygon(620,20,620,45,MAXX-5,45,MAXX-5,20);
                boxes(635,660,colors[7]);
                boxes(600,625,colors[7]);
            boxes(565,590,colors[7]);
                boxes(530,555,colors[7]);
                boxes(495,520,colors[7]);
                boxes(460,485,colors[7]);
                boxes(425,450,colors[7]);
                boxes(390,415,colors[7]);
                glColor3fv(colors[17]);
                polygon(970,280,970,385,970+45,385,970+45,280);
                boxes3(355,380,colors[8]);
                boxes1(320,345,colors[7]);
                boxes1(285,310,colors[7]);
                boxes1(250,275,colors[7]);
                boxes1(238,217,colors[7]);
                boxes1(205,185,colors[7]);
                boxes1(172,141,colors[7]);
                glColor3f(0,0,0);
                draw_text("INSIDE CLIP - to remove parts inside a selected area",625,30);
                disp();
                glFlush();
                state1=INSIDECLIP;
                return;
        }
        ii+=35;jj+=35;
        if(inside_area(5,45,MAXY-ii,MAXY-jj,x,y))
        {
                glColor3fv(colors[17]);
                polygon(620,20,620,45,MAXX-5,45,MAXX-5,20);
                boxes(635,660,colors[7]);
                boxes(600,625,colors[7]);
            boxes(565,590,colors[7]);
                boxes(530,555,colors[7]);
                boxes(495,520,colors[7]);
                boxes(460,485,colors[7]);
                boxes(425,450,colors[7]);
                boxes(390,415,colors[7]);
                glColor3fv(colors[17]);
                polygon(970,280,970,385,970+45,385,970+45,280);
                boxes1(355,380,colors[7]);
                boxes3(320,345,colors[8]);
                boxes1(285,310,colors[7]);
                boxes1(250,275,colors[7]);
                boxes1(238,215,colors[7]);
                boxes1(205,185,colors[7]);
                boxes1(172,141,colors[7]);
                glColor3f(0,0,0);
                draw_text("OUTSIDE CLIP - to remove parts outside a selected area",625,30);
```

```
            disp();
            glFlush();
            state1=OUTSIDECLIP;
            return;
        }
        ii+=35;jj+=35;
        if(inside_area(5,45,MAXY-ii,MAXY-jj,x,y))
        {
            glColor3fv(colors[17]);
            polygon(620,20,620,45,MAXX-5,45,MAXX-5,20);
            boxes(635,660,colors[7]);
            boxes(600,625,colors[7]);
          boxes(565,590,colors[7]);
            boxes(530,555,colors[7]);
            boxes(495,520,colors[7]);
            boxes(460,485,colors[7]);
            boxes(425,450,colors[7]);
            boxes(390,415,colors[7]);
            glColor3fv(colors[17]);
            polygon(970,280,970,385,970+45,385,970+45,280);
            boxes1(355,380,colors[7]);
            boxes1(320,345,colors[7]);
            boxes3(285,310,colors[8]);
            boxes1(250,275,colors[7]);
            boxes1(238,215,colors[7]);
            boxes1(205,185,colors[7]);
            boxes1(172,141,colors[7]);
            glColor3f(0,0,0);
            draw_text("TRANSLATE - to shift selected area to desired location",625,30);
            disp();
            glFlush();
            state1=TRANSLATE;
            return;
        }
        ii+=35;jj+=35;
        if(inside_area(5,45,MAXY-ii,MAXY-jj,x,y))
        {
            glColor3fv(colors[17]);
            polygon(620,20,620,45,MAXX-5,45,MAXX-5,20);
          boxes(635,660,colors[7]);
            boxes(600,625,colors[7]);
          boxes(565,590,colors[7]);
            boxes(530,555,colors[7]);
            boxes(495,520,colors[7]);
            boxes(460,485,colors[7]);
            boxes(425,450,colors[7]);
            boxes(390,415,colors[7]);
            glColor3fv(colors[17]);
            polygon(970,280,970,385,970+45,385,970+45,280);
            boxes1(355,380,colors[7]);
            boxes1(320,345,colors[7]);
            boxes1(285,310,colors[7]);
            boxes3(250,275,colors[8]);
            boxes1(238,215,colors[7]);
            boxes1(205,185,colors[7]);
            boxes1(172,141,colors[7]);
            glColor3f(0,0,0);
            draw_text("SCALING - to zoom the selected area to twice its size",625,30);
            disp();
            glFlush();
            state1=SCALING;
            return;
        }
```

```
ii+=35;jj+=35;
if(inside_area(5,45,MAXY-ii,MAXY-jj,x,y))
{
       glColor3fv(colors[17]);
       polygon(620,20,620,45,MAXX-5,45,MAXX-5,20);
    boxes(635,660,colors[7]);
       boxes(600,625,colors[7]);
    boxes(565,590,colors[7]);
       boxes(530,555,colors[7]);
       boxes(495,520,colors[7]);
       boxes(460,485,colors[7]);
       boxes(425,450,colors[7]);
       boxes(390,415,colors[7]);
       glColor3fv(colors[17]);
       polygon(970,280,970,385,970+45,385,970+45,280);
       boxes1(355,380,colors[7]);
       boxes1(320,345,colors[7]);
       boxes1(285,310,colors[7]);
       boxes1(250,275,colors[7]);
       boxes3(215,238,colors[8]);
       boxes1(205,185,colors[7]);
       boxes1(172,141,colors[7]);
       glColor3f(0,0,0);
       draw_text("Draw a cardioid",625,30);
       disp();
       glFlush();
       state1=CARDIOID;
       return;
}

ii+=35;jj+=35;

if(inside_area(3,45,MAXY-ii,MAXY-jj,x,y))
{
       glColor3fv(colors[17]);
       polygon(620,20,620,45,MAXX-5,45,MAXX-5,20);
    boxes(635,660,colors[7]);
       boxes(600,625,colors[7]);
    boxes(565,590,colors[7]);
       boxes(530,555,colors[7]);
       boxes(495,520,colors[7]);
       boxes(460,485,colors[7]);
       boxes(425,450,colors[7]);
       boxes(390,415,colors[7]);
       glColor3fv(colors[17]);
       polygon(970,280,970,385,970+45,385,970+45,280);
       boxes1(355,380,colors[7]);
       boxes1(320,345,colors[7]);
       boxes1(285,310,colors[7]);
       boxes1(250,275,colors[7]);
       boxes1(238,215,colors[7]);
       boxes3(185,205,colors[8]);
       boxes1(172,141,colors[7]);
       glColor3f(0,0,0);
       draw_text("Draw a Limacon",625,30);
       disp();
       glFlush();
       state1=LIMACON;
       return;
}

ii+=35;jj+=35;
```

```
if(inside_area(5,45,MAXY-ii,MAXY-jj,x,y))
{
       glColor3fv(colors[17]);
       polygon(620,20,620,45,MAXX-5,45,MAXX-5,20);
    boxes(635,660,colors[7]);
       boxes(600,625,colors[7]);
    boxes(565,590,colors[7]);
       boxes(530,555,colors[7]);
       boxes(495,520,colors[7]);
       boxes(460,485,colors[7]);
       boxes(425,450,colors[7]);
       boxes(390,415,colors[7]);
       glColor3fv(colors[17]);
       polygon(970,280,970,385,970+45,385,970+45,280);
       boxes1(355,380,colors[7]);
       boxes1(320,345,colors[7]);
       boxes1(285,310,colors[7]);
       boxes1(250,275,colors[7]);
       boxes1(238,215,colors[7]);
       boxes1(205,185,colors[7]);
       boxes3(141,172,colors[8]);
       glColor3f(0,0,0);
       draw_text("SPIRAL - to draw a spiral",625,30);
       disp();
       glFlush();
       state1=SPIRAL;
       return;
}
if(inside_area(5,45,MAXY-30,MAXY-5,x,y))
{
       menu(5,45,MAXY-30,MAXY-5,colors[8]);
       menu(55,95,MAXY-30,MAXY-5,colors[17]);
       menu(105,145,MAXY-30,MAXY-5,colors[17]);
       menu(155,195,MAXY-30,MAXY-5,colors[17]);
       menu(205,245,MAXY-30,MAXY-5,colors[17]);
       menu(257,297,MAXY-30,MAXY-5,colors[17]);
       menu(312,358,MAXY-30,MAXY-5,colors[17]);
       menu(367,408,MAXY-30,MAXY-5,colors[17]);
       menu_disp();
       state1=NEW;
       return;
}
if(inside_area(55,95,MAXY-30,MAXY-5,x,y))
{
       menu(5,45,MAXY-30,MAXY-5,colors[17]);
       menu(55,95,MAXY-30,MAXY-5,colors[8]);
       menu(105,145,MAXY-30,MAXY-5,colors[17]);
       menu(155,195,MAXY-30,MAXY-5,colors[17]);
       menu(205,245,MAXY-30,MAXY-5,colors[17]);
       menu(257,297,MAXY-30,MAXY-5,colors[17]);
       menu(312,358,MAXY-30,MAXY-5,colors[17]);
       menu(367,408,MAXY-30,MAXY-5,colors[17]);
       menu_disp();
       state1=OPEN;
       openc=0;
       return;
}
if(inside_area(105,145,MAXY-30,MAXY-5,x,y))
{
       menu(5,45,MAXY-30,MAXY-5,colors[17]);
       menu(55,95,MAXY-30,MAXY-5,colors[17]);
       menu(105,145,MAXY-30,MAXY-5,colors[8]);
       menu(155,195,MAXY-30,MAXY-5,colors[17]);
```

```
                menu(205,245,MAXY-30,MAXY-5,colors[17]);
                menu(257,297,MAXY-30,MAXY-5,colors[17]);
                menu(312,358,MAXY-30,MAXY-5,colors[17]);
                menu(367,408,MAXY-30,MAXY-5,colors[17]);
                menu_disp();
                state1=SAVE;
                savec=0;
                return;
        }
        if(inside_area(155,195,MAXY-30,MAXY-5,x,y))
        {
                menu(5,45,MAXY-30,MAXY-5,colors[17]);
                menu(55,95,MAXY-30,MAXY-5,colors[17]);
                menu(105,145,MAXY-30,MAXY-5,colors[17]);
                menu(155,195,MAXY-30,MAXY-5,colors[8]);
                menu(205,245,MAXY-30,MAXY-5,colors[17]);
                menu(257,297,MAXY-30,MAXY-5,colors[17]);
                menu(312,358,MAXY-30,MAXY-5,colors[17]);
                menu(367,408,MAXY-30,MAXY-5,colors[17]);
                menu_disp();
                state1=EXIT;
                return;
        }
        if(inside_area(205,245,MAXY-30,MAXY-5,x,y))
        {
                menu(5,45,MAXY-30,MAXY-5,colors[17]);
                menu(55,95,MAXY-30,MAXY-5,colors[17]);
                menu(105,145,MAXY-30,MAXY-5,colors[17]);
                menu(155,195,MAXY-30,MAXY-5,colors[17]);
                menu(205,245,MAXY-30,MAXY-5,colors[8]);
                menu(257,297,MAXY-30,MAXY-5,colors[17]);
                menu(312,358,MAXY-30,MAXY-5,colors[17]);
                menu(367,408,MAXY-30,MAXY-5,colors[17]);
                menu_disp();
                state1=TRANSLATE;
                return;
        }
if(inside_area(255,295,MAXY-30,MAXY-5,x,y))
        {
                menu(5,45,MAXY-30,MAXY-5,colors[17]);
                menu(55,95,MAXY-30,MAXY-5,colors[17]);
                menu(105,145,MAXY-30,MAXY-5,colors[17]);
                menu(155,195,MAXY-30,MAXY-5,colors[17]);
                menu(205,245,MAXY-30,MAXY-5,colors[17]);
                menu(257,297,MAXY-30,MAXY-5,colors[8]);
                menu(312,358,MAXY-30,MAXY-5,colors[17]);
                menu(367,408,MAXY-30,MAXY-5,colors[17]);
                menu_disp();
                state1=TRANSLATE1;
                return;
        }

        if(inside_area(205,245,MAXY-30,MAXY-5,x,y))
        {
                menu(5,45,MAXY-30,MAXY-5,colors[17]);
                menu(55,95,MAXY-30,MAXY-5,colors[17]);
                menu(105,145,MAXY-30,MAXY-5,colors[17]);
                menu(155,195,MAXY-30,MAXY-5,colors[17]);
                menu(205,245,MAXY-30,MAXY-5,colors[8]);
                menu(257,297,MAXY-30,MAXY-5,colors[17]);
                menu(312,358,MAXY-30,MAXY-5,colors[17]);
                menu(367,408,MAXY-30,MAXY-5,colors[17]);
                menu_disp();
```

```
                state1=INSIDECLIP;
                return;
        }
        if(inside_area(257,297,MAXY-30,MAXY-5,x,y))
        {
                menu(5,45,MAXY-30,MAXY-5,colors[17]);
                menu(55,95,MAXY-30,MAXY-5,colors[17]);
                menu(105,145,MAXY-30,MAXY-5,colors[17]);
                menu(155,195,MAXY-30,MAXY-5,colors[17]);
                menu(205,245,MAXY-30,MAXY-5,colors[17]);
                menu(257,297,MAXY-30,MAXY-5,colors[8]);
                menu(312,358,MAXY-30,MAXY-5,colors[17]);
                menu(367,408,MAXY-30,MAXY-5,colors[17]);
                menu_disp();
                state1=OUTSIDECLIP;
                return;
        }
        if(inside_area(312,358,MAXY-30,MAXY-5,x,y))
        {
                menu(5,45,MAXY-30,MAXY-5,colors[17]);
                menu(55,95,MAXY-30,MAXY-5,colors[17]);
                menu(105,145,MAXY-30,MAXY-5,colors[17]);
                menu(155,195,MAXY-30,MAXY-5,colors[17]);
                menu(205,245,MAXY-30,MAXY-5,colors[17]);
                menu(257,297,MAXY-30,MAXY-5,colors[17]);
                menu(312,358,MAXY-30,MAXY-5,colors[8]);
                menu(367,408,MAXY-30,MAXY-5,colors[17]);
                menu_disp();
                state1=TRANSLATE;
                return;
        }
        if(inside_area(367,408,MAXY-30,MAXY-5,x,y))
        {
                menu(5,45,MAXY-30,MAXY-5,colors[17]);
                menu(55,95,MAXY-30,MAXY-5,colors[17]);
                menu(105,145,MAXY-30,MAXY-5,colors[17]);
                menu(155,195,MAXY-30,MAXY-5,colors[17]);
                menu(205,245,MAXY-30,MAXY-5,colors[17]);
                menu(257,297,MAXY-30,MAXY-5,colors[17]);
                menu(312,358,MAXY-30,MAXY-5,colors[17]);
                menu(367,408,MAXY-30,MAXY-5,colors[8]);
                menu_disp();
                state1=SCALING;
                return;
        }
        if(state1==SPIRAL)
        {
                spiral(x,y);
        }
}
if(state1==NEW)
{
        glColor3fv(colors[7]);
        polygon(50,50,MAXX-50,50,MAXX-50,MAXY-40,50,MAXY-40);
        glColor3fv(colors[0]);
    lineloop(50,50,MAXX-50,50,MAXX-50,MAXY-40,50,MAXY-40);
        line(51,51,51,MAXY-39);
        line(51,MAXY-41,MAXX-51,MAXY-41);
        glColor3fv(colors[17]);
                polygon(970,280,970,385,970+45,385,970+45,280);

 glColor3fv(colors[17]);
 polygon(0,MAXY-40,0,MAXY,MAXX,MAXY,MAXX,MAXY-40);
```

```
polygon(MAXX-50,50,MAXX,50,MAXX,MAXY-40,MAXX-50,MAXY-40); //right
polygon(0,0,0,MAXY-20,50,MAXY-20,50,0);  //left
	menu(5,45,MAXY-30,MAXY-5,colors[17]);
	menu(55,95,MAXY-30,MAXY-5,colors[17]);
	menu(105,145,MAXY-30,MAXY-5,colors[17]);
	menu(155,195,MAXY-30,MAXY-5,colors[17]);
	menu_disp();
	glColor3fv(colors[7]);
	int x=40,y=65,i;
	for(i=0;i<8;i++)
	{
		polygon(970+5,MAXY-x,970+5,MAXY-y,970+45,MAXY-y,970+45,MAXY-x);
		x+=35;
		y+=35;
	}
	for(i=0;i<4;i++)
	{
		polygon(5,MAXY-x,5,MAXY-y,45,MAXY-y,45,MAXY-x);
		x+=35;
		y+=35;
	}
	disp();
}
if(state1==SAVE && savec==0)
{
	char filename[30];
	savec=1;
	glReadPixels(50,50,MAXX-50,MAXY-40,GL_RGB,GL_FLOAT,mat);
	printf("Enter the filename to save : ");
	scanf("%s",filename);
	fp=fopen(filename,"w");
	fwrite(mat,matsize,sizeof(GLubyte),fp);
	fclose(fp);
	printf("\nFILE SAVED\n");
	menu(5,45,MAXY-30,MAXY-5,colors[17]);
	menu(55,95,MAXY-30,MAXY-5,colors[17]);
	menu(105,145,MAXY-30,MAXY-5,colors[16]);
	menu(155,195,MAXY-30,MAXY-5,colors[17]);
	menu_disp();
	glFlush();
}
if(state1==OPEN && openc==0)
{
	char filename[30];
	openc=1;
	printf("\nEnter filename to open : ");
	scanf("%s",filename);
	fp=fopen(filename,"r");
	fread(mat,matsize,sizeof(GLubyte),fp);
	fclose(fp);
	glColor3fv(colors[7]);
	polygon(50,50,MAXX-50,50,MAXX-50,MAXY-40,50,MAXY-40);
	glFlush();
	glRasterPos2i(50,50);
	glDrawPixels(MAXX-50,MAXY-40,GL_RGB,GL_FLOAT,mat);
	printf("\nFILE OPENED\n");
	menu(5,45,MAXY-30,MAXY-5,colors[17]);
	menu(55,95,MAXY-30,MAXY-5,colors[17]);
	menu(105,145,MAXY-30,MAXY-5,colors[16]);
	menu(155,195,MAXY-30,MAXY-5,colors[17]);
	menu_disp();
	glFlush();
}
```

```
if(trans_paste==1)
{
	glRasterPos2i(50,50);
	glDrawPixels(MAXX-50,MAXY-50,GL_RGB,GL_FLOAT,arr);
	if(x>=50)
		if(y>=50)
			if((x+px-1)<=(MAXX-50))
				if((y+py-1)<=(MAXY-40))
				{
					glColor3fv(colors[7]);
					polygon(ax,by,ax,by+py,ax+px,by+py,ax+px,by);
					glRasterPos2i(x,y);
					glDrawPixels(px-1,py-1,GL_RGB,GL_FLOAT,clip);
				}
	glFlush();
	trans_paste=0;
}

if(trans_paste1==1)
{
	glRasterPos2i(50,50);
	glDrawPixels(MAXX-50,MAXY-50,GL_RGB,GL_FLOAT,arr);
	if(x>=50)
		if(y>=50)
			if((x+px-1)<=(MAXX-50))
				if((y+py-1)<=(MAXY-40))
				{
					//glColor3fv(colors[7]);
					//olygon(ax,by,ax,by+py,ax+px,by+py,ax+px,by);
					glRasterPos2i(x,y);
					glDrawPixels(px-1,py-1,GL_RGB,GL_FLOAT,clip);
				}
	glFlush();
	trans_paste1=0;
}
if(scale_paste==1)
{
	glRasterPos2i(50,50);
	glDrawPixels(MAXX-50,MAXY-50,GL_RGB,GL_FLOAT,arr);
	if(x>=50)
		if(y>=50)
			if((x+px-1)<=(MAXX-50))
				if((y+py-1)<=(MAXY-40))
				{
					glColor3fv(colors[7]);
					polygon(ax,by,ax,by+py,ax+px,by+py,ax+px,by);
					glPixelZoom(2,2);
					glRasterPos2i(x,y);
					glDrawPixels(px-1,py-1,GL_RGB,GL_FLOAT,clip);
				}
	glFlush();
	glPixelZoom(1,1);
	scale_paste=0;
}
if(state1==EXIT)
{
	exit(0);
}
if(state==GLUT_UP)
{
	if(outsideclip)
	{
		px=elinex-linex; py=liney-eliney;
```

```
		glReadPixels(linex,eliney,px,py,GL_RGB,GL_FLOAT,clip);
		glColor3fv(colors[7]);
		polygon(50,50,MAXX-50,50,MAXX-50,MAXY-40,50,MAXY-40);
		glFlush();
		glRasterPos2i(linex,eliney);
		glDrawPixels(px,py,GL_RGB,GL_FLOAT,clip);
		outsideclip=0;
	}
	if(insideclip)
	{
		glColor3fv(colors[7]);
		printf("%d %d %d %d",linex,liney,elinex,eliney);
		polygon(linex,liney,linex,eliney,elinex,eliney,elinex,liney);
		lineloop(linex,liney,linex-1,eliney,elinex,eliney,elinex,liney);
		glFlush();
		insideclip=0;
	}

	if(translate)
	{
		ax=linex;ay=liney;bx=elinex;by=eliney;
		px=elinex-linex; py=liney-eliney;
		glReadPixels(linex,eliney+1,px-1,py-1,GL_RGB,GL_FLOAT,clip);
		trans_paste=1;
		translate=0;
	}
	if(translate1)
	{
		ax=linex;ay=liney;bx=elinex;by=eliney;
		px=elinex-linex; py=liney-eliney;
		glReadPixels(linex,eliney+1,px-1,py-1,GL_RGB,GL_FLOAT,clip);
		trans_paste1=1;
		translate1=0;
	}
	if(scaling)
	{
		ax=linex;ay=liney;bx=elinex;by=eliney;
		px=elinex-linex; py=liney-eliney;
		glReadPixels(linex,eliney+1,px-1,py-1,GL_RGB,GL_FLOAT,clip);
		scale_paste=1;
		scaling=0;
	}
	if(drawline)
	{
		glBegin(GL_LINES);
		glVertex2i(linex,liney);
		glVertex2i(elinex,eliney);
		glEnd();
		glFlush();
		linex=0;
		liney=0;
		elinex=0;
		eliney=0;
		drawline=0;
	}
	if(drawrect)
	{
		lineloop(linex,liney,linex-1,eliney,elinex,eliney,elinex,liney);
		linex=0;
		liney=0;
		elinex=0;
		eliney=0;
		drawrect=0;
```

```
		}
		if(drawcircle)
		{
			double r=sqrt(pow((double)(elinex-linex),2)+(pow((double)(eliney-liney),2)));
			if((linex-r)<50)
				r=linex-50;
			if((liney-r)<50)
				r=liney-50;
			if((linex+r)>(MAXX-51))
				r=MAXX-51-linex;
			if((liney+r)>(MAXY-41))
				r=MAXY-41-liney;
			circle(linex,liney,r);
			glFlush();
			linex=0;
			liney=0;
			elinex=0;
			eliney=0;
			drawcircle=0;
		}
		glReadPixels(50,50,MAXX-50,MAXY-50,GL_RGB,GL_FLOAT,arr1);
	}

}

void mymove(int mx,int my)
{
	GLint x=mx;
	GLint y=MAXY-my;
	if(state1==INSIDECLIP)
	{
		if(x<MAXX-49 && x>49 && y>50 && y<MAXY-39)
		{
			if(!linex && !liney)
			{
				glReadPixels(50,50,MAXX-50,MAXY-50,GL_RGB,GL_FLOAT,arr);
				linex=x;
				liney=y;
			}
			else
			{
				drawrect=1;
				insideclip=1;
				elinex=x;
				eliney=y;
				glRasterPos2i(50,50);
				glDrawPixels(MAXX-50,MAXY-50,GL_RGB,GL_FLOAT,arr);
				lineloop(linex,liney,linex-1,eliney,elinex,eliney,elinex,liney);
				glColor3fv(colors[0]);
			}
		}
	}
	if(state1==OUTSIDECLIP)
	{
		if(x<MAXX-49 && x>49 && y>50 && y<MAXY-39)
		{
			if(!linex && !liney)
			{
				glReadPixels(50,50,MAXX-50,MAXY-50,GL_RGB,GL_FLOAT,arr);
				linex=x;
				liney=y;
			}
			else
```

```
                {
                        drawrect=1;
                        outsideclip=1;
                        elinex=x;
                        eliney=y;
                        glRasterPos2i(50,50);
                        glDrawPixels(MAXX-50,MAXY-50,GL_RGB,GL_FLOAT,arr);
                        lineloop(linex,liney,linex-1,eliney,elinex,eliney,elinex,liney);
                        glColor3fv(colors[0]);
                }
        }
}
if(state1==TRANSLATE)
{
        if(x<MAXX-49 && x>49 && y>50 && y<MAXY-39)
        {
                if(!linex && !liney)
                {
                        glReadPixels(50,50,MAXX-50,MAXY-50,GL_RGB,GL_FLOAT,arr);
                        linex=x;
                        liney=y;
                }
                else
                {
                        drawrect=1;
                        translate=1;
                        elinex=x;
                        eliney=y;
                        glRasterPos2i(50,50);
                        glDrawPixels(MAXX-50,MAXY-50,GL_RGB,GL_FLOAT,arr);
                        lineloop(linex,liney,linex-1,eliney,elinex,eliney,elinex,liney);
                        glColor3fv(colors[0]);
                }
        }
}

if(state1==TRANSLATE1)
{
        if(x<MAXX-49 && x>49 && y>50 && y<MAXY-39)
        {
                if(!linex && !liney)
                {
                        glReadPixels(50,50,MAXX-50,MAXY-50,GL_RGB,GL_FLOAT,arr);
                        linex=x;
                        liney=y;
                }
                else
                {
                        drawrect=1;
                        translate1=1;
                        elinex=x;
                        eliney=y;
                        glRasterPos2i(50,50);
                        glDrawPixels(MAXX-50,MAXY-50,GL_RGB,GL_FLOAT,arr);
                        lineloop(linex,liney,linex-1,eliney,elinex,eliney,elinex,liney);
                        glColor3fv(colors[0]);
                }
        }
}
if(state1==SCALING)
{
        if(x<MAXX-49 && x>49 && y>50 && y<MAXY-39)
        {
```

```
                if(!linex && !liney)
                {
                        glReadPixels(50,50,MAXX-50,MAXY-50,GL_RGB,GL_FLOAT,arr);
                        linex=x;
                        liney=y;
                }
                else
                {
                        drawrect=1;
                        scaling=1;
                        elinex=x;
                        eliney=y;
                        glRasterPos2i(50,50);
                        glDrawPixels(MAXX-50,MAXY-50,GL_RGB,GL_FLOAT,arr);
                        lineloop(linex,liney,linex-1,eliney,elinex,eliney,elinex,liney);
                        glColor3fv(colors[0]);
                }
        }
}
if(state1==BRUSH)
{
        if(x<(MAXX-57) && x>49 && y>49 && y<(MAXY-47))
        {
                GLint brushsize=8;
                glRecti(x,y,x+brushsize,y+brushsize);
        }
        glFlush();
}
if(state1==PENCIL)
{
        if(x<(MAXX-49) && x>49 && y>49 && y<(MAXY-39))
        {
                glBegin(GL_POLYGON);
                glVertex2f(x,y);
                glVertex2f(x,y+1.8);
                glVertex2f(x+1.8,y+1.8);
                glVertex2f(x+1.8,y);
                glEnd();
                glFlush();
        }
}
if(state1==ERAZE1)
{

        if(x<(MAXX-57) && x>49 && y>49 && y<(MAXY-47))
        {
                glColor3f(1.0,1.0,1.0);
                GLint brushsize=8;
                glRecti(x,y,x+brushsize,y+brushsize);
        }
        glFlush();
}
if(state1==ERAZE2)
{

        if(x<(MAXX-57) && x>49 && y>49 && y<(MAXY-47))
        {
                glColor3f(1.0,1.0,1.0);
                GLint brushsize=16;
                glRecti(x,y,x+brushsize,y+brushsize);
        }
        glFlush();
}
```

```
if(state1==ERAZE3)
{

	if(x<(MAXX-57) && x>49 && y>49 && y<(MAXY-47))
	{
		glColor3f(1.0,1.0,1.0);
		GLint brushsize=24;
		glRecti(x,y,x+brushsize,y+brushsize);
	}
	glFlush();
}
if(state1==SPRAY)
{
	if(x<(MAXX-50) && x>59 && y>54 && y<(MAXY-45))
	{
		glPointSize(1.0);
		spray(x,y);
		glFlush();
	}
}
if(state1==LINE)
{
	if(x<MAXX-49 && x>49 && y>49 && y<MAXY-39)
	{
		if(!linex && !liney)
		{
			glReadPixels(50,50,MAXX-50,MAXY-50,GL_RGB,GL_FLOAT,arr);
			linex=x;
			liney=y;
		}
		else
		{
			drawline=1;
			elinex=x;
			eliney=y;
			glRasterPos2i(50,50);
			glDrawPixels(MAXX-50,MAXY-50,GL_RGB,GL_FLOAT,arr);
			glBegin(GL_LINES);
			glVertex2i(linex,liney);
			glVertex2i(x,y);
			glEnd();
			glFlush();
		}
	}
}
if(state1==RECT)
{
	if(x<MAXX-49 && x>49 && y>50 && y<MAXY-39)
	{
		if(!linex && !liney)
		{
			glReadPixels(50,50,MAXX-50,MAXY-50,GL_RGB,GL_FLOAT,arr);
			linex=x;
			liney=y;
		}
		else
		{
			drawrect=1;
			elinex=x;
			eliney=y;
			glRasterPos2i(50,50);
			glDrawPixels(MAXX-50,MAXY-50,GL_RGB,GL_FLOAT,arr);
			lineloop(linex,liney,linex-1,eliney,elinex,eliney,elinex,liney);
```

```
                }
            }
        }
        if(state1==FILLRECT)
        {
            if(x<MAXX-49 && x>49 && y>49 && y<MAXY-38)
            {
                if(!linex && !liney)
                {
                    glReadPixels(50,50,MAXX-50,MAXY-50,GL_RGB,GL_FLOAT,arr);
                    linex=x;
                    liney=y;
                }
                else
                {
                    drawline=1;
                    elinex=x;
                    eliney=y;
                    glRasterPos2i(50,50);
                    glDrawPixels(MAXX-50,MAXY-50,GL_RGB,GL_FLOAT,arr);
                    glBegin(GL_POLYGON);
                    glVertex2i(linex,liney);
                    glVertex2i(linex,eliney);
                    glVertex2i(elinex,eliney);
                    glVertex2i(elinex,liney);
                    glEnd();
                glFlush();
                }
            }
        }
        if(state1==CIRCLE)
        {
            if(x<MAXX-49 && x>49 && y>49 && y<MAXY-39)
            {
                if(!linex && !liney)
                {
                    glReadPixels(50,50,MAXX-50,MAXY-50,GL_RGB,GL_FLOAT,arr);
                    linex=x;
                    liney=y;
                }
                else
                {
                    drawcircle=1;
                    elinex=x;
                    eliney=y;
                    glRasterPos2i(50,50);
                    glDrawPixels(MAXX-50,MAXY-50,GL_RGB,GL_FLOAT,arr);
                    double r=sqrt(pow((double)(elinex-linex),2)+(pow((double)(eliney-
liney),2)));
                    if((linex-r)<50)
                        r=linex-50;
                    if((liney-r)<50)
                        r=liney-50;
                    if((linex+r)>(MAXX-51))
                        r=MAXX-51-linex;
                    if((liney+r)>(MAXY-41))
                        r=MAXY-41-liney;
                    circle(linex,liney,r);
                    glFlush();
                }
            }
        }
}
```

```
void display()
{
        glReadPixels(50,50,MAXX-50,MAXY-50,GL_RGB,GL_FLOAT,arr1);
        glClear(GL_COLOR_BUFFER_BIT);
        glColor3fv(colors[17]);
        polygon(0,MAXY-40,0,MAXY,MAXX,MAXY,MAXX,MAXY-40); //top
        menu_disp();
    glColor3fv(colors[17]);
        polygon(MAXX-50,50,MAXX,50,MAXX,MAXY-40,MAXX-50,MAXY-40); //right
        polygon(0,0,0,MAXY-20,50,MAXY-20,50,0);  //left
        polygon(0,0,0,50,MAXX,50,MAXX,0); //bottom
        glColor3fv(colors[7]);
        polygon(50,5,50,25,75,25,75,5);
        glColor3fv(colors[0]);
        polygon(200+50,5,200+50,25,200+75,25,200+75,5);
        glColor3fv(colors[16]);
        polygon(75,5,75,25,100,25,100,5);
        glColor3fv(colors[10]);
        polygon(200+75,5,200+75,25,200+100,25,200+100,5);
        glColor3fv(colors[6]);
        polygon(100,5,100,25,125,25,125,5);
        glColor3fv(colors[3]);
        polygon(200+100,5,200+100,25,200+125,25,200+125,5);
        glColor3fv(colors[2]);
        polygon(125,5,125,25,150,25,150,5);
        glColor3fv(colors[11]);
        polygon(200+125,5,200+125,25,200+150,25,200+150,5);
        glColor3fv(colors[13]);
        polygon(150,5,150,25,175,25,175,5);
        glColor3fv(colors[9]);
        polygon(200+150,5,200+150,25,200+175,25,200+175,5);
        glColor3fv(colors[15]);
        polygon(175,5,175,25,200,25,200,5);
        glColor3fv(colors[14]);
        polygon(200+175,5,200+175,25,200+200,25,200+200,5);
        glColor3fv(colors[5]);
        polygon(200,5,200,25,225,25,225,5);
        glColor3fv(colors[1]);
        polygon(200+200,5,200+200,25,200+225,25,200+225,5);
        glColor3fv(colors[4]);
        polygon(225,5,225,25,250,25,250,5);
        glColor3fv(colors[12]);
        polygon(200+225,5,200+225,25,200+250,25,200+250,5);
        glColor3fv(colors[7]);
        glPointSize(2.0);
        lineloop(50,5,50,25,450,25,450,5);
        line(50,5,50,25);
        line(75,5,75,25);
        line(100,5,100,25);
        line(125,5,125,25);
        line(150,5,150,25);
        line(175,5,175,25);
        line(200,5,200,25);
        line(225,5,225,25);
        line(250,5,250,25);
        line(275,5,275,25);
        line(300,5,300,25);
        line(325,5,325,25);
        line(350,5,350,25);
        line(375,5,375,25);
        line(400,5,400,25);
        line(425,5,425,25);
```

```
	line(450,5,450,25);
	glColor3fv(colors[7]);
	int x=40,y=65,i;
	for(i=0;i<8;i++)
	{
		polygon(970+5,MAXY-x,970+5,MAXY-y,970+45,MAXY-y,970+45,MAXY-x);
		x+=35;
		y+=35;
	}
	for(i=0;i<4;i++)
	{
		polygon(5,MAXY-x,5,MAXY-y,45,MAXY-y,45,MAXY-x);
		x+=35;
		y+=35;
	}
	disp();
	glFlush();
	glColor3fv(colors[0]);
	polygon(10,10,10,40,40,40,40,10);
	lineloop(9,10,10,40,40,40,40,10);
	glColor3fv(colors[7]);
	polygon(50,50,MAXX-50,50,MAXX-50,MAXY-40,50,MAXY-40);
	glColor3fv(colors[0]);
	glPointSize(2.0);
	lineloop(50,50,MAXX-50,50,MAXX-50,MAXY-40,50,MAXY-40);
	if(count!=0)
	{
		glRasterPos2i(50,50);
		glDrawPixels(MAXX-50,MAXY-50,GL_RGB,GL_FLOAT,arr1);
	}
	else count++;
	draw_text(" ",MAXX-500,MAXY-10);
	glFlush();
}

void keyb(unsigned char key,int x,int y)
{
	if(key=='+') size++;
	if(key=='-') size--;
}


void myReshape(int screenwidth,int screenheight)
{
	glViewport(0, 0, screenwidth, screenheight);
	glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluOrtho2D(0.0,MAXX,0.0,MAXY);
	glColor3f(0.0,0.0,0.0);
}


void main(int argc,char **argv)
{
	glutInit(&argc,argv);
	glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
	glutInitWindowSize(MAXX,MAXY);
	glutInitWindowPosition(0,0);
	glutCreateWindow("Editor");
	glutReshapeFunc(myReshape);
	glutDisplayFunc(display);
	glutMouseFunc(mouse);
```

```
        glutKeyboardFunc(keyb);
        glutMotionFunc(mymove);
        glutMainLoop();
}
```

Appendix B – Screenshots

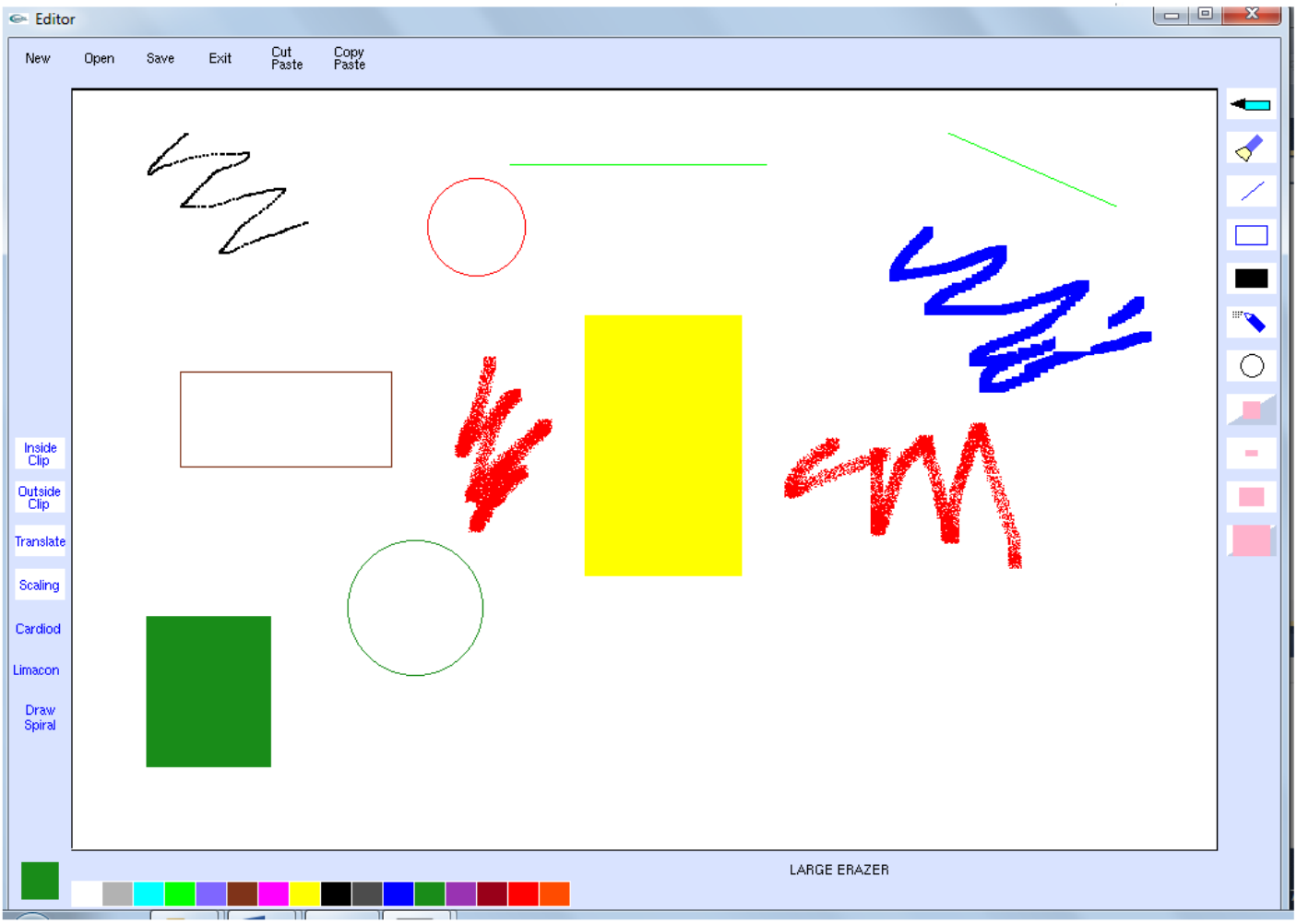

Fig B.1 Tools

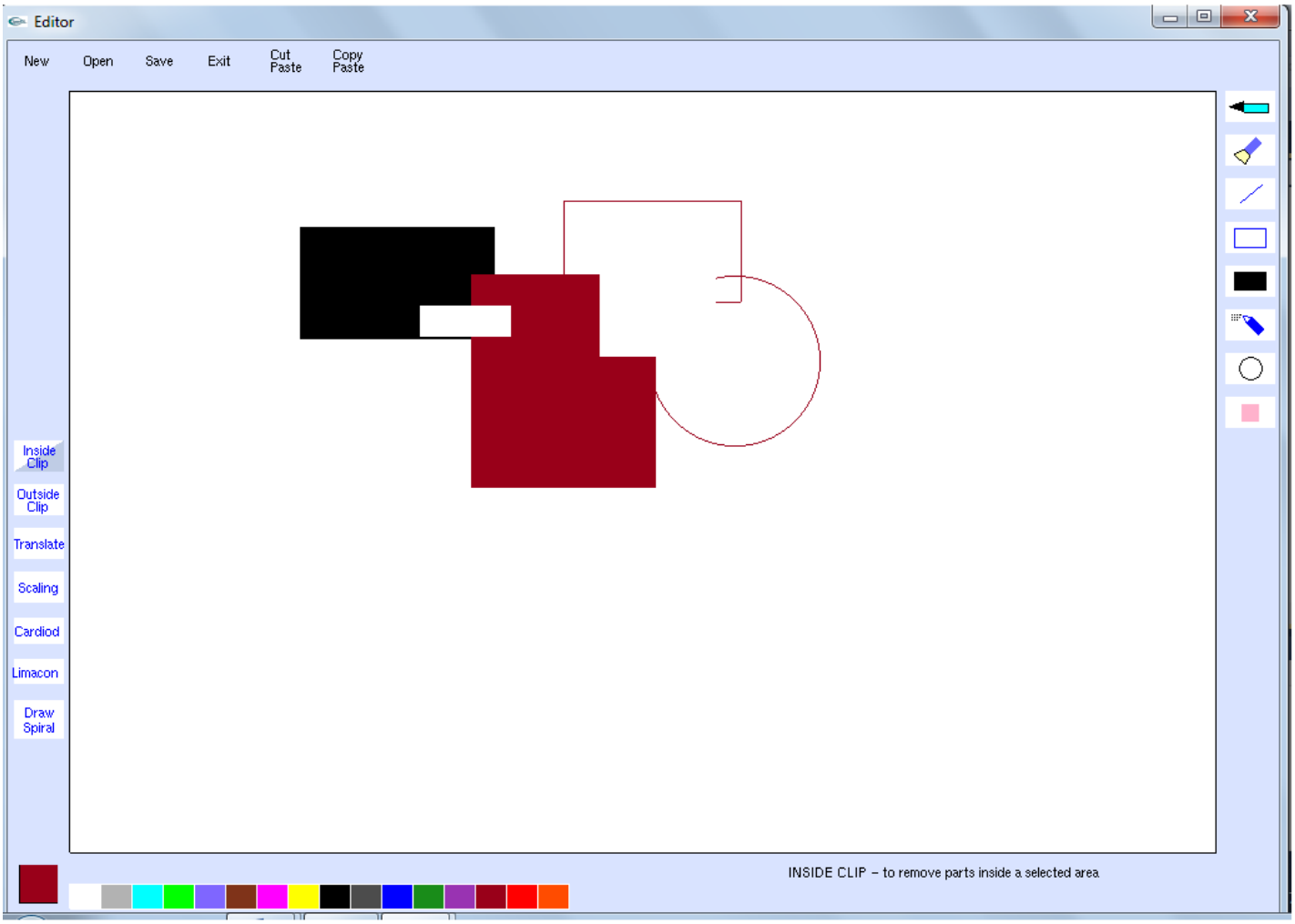

Fig B.2 Clippings

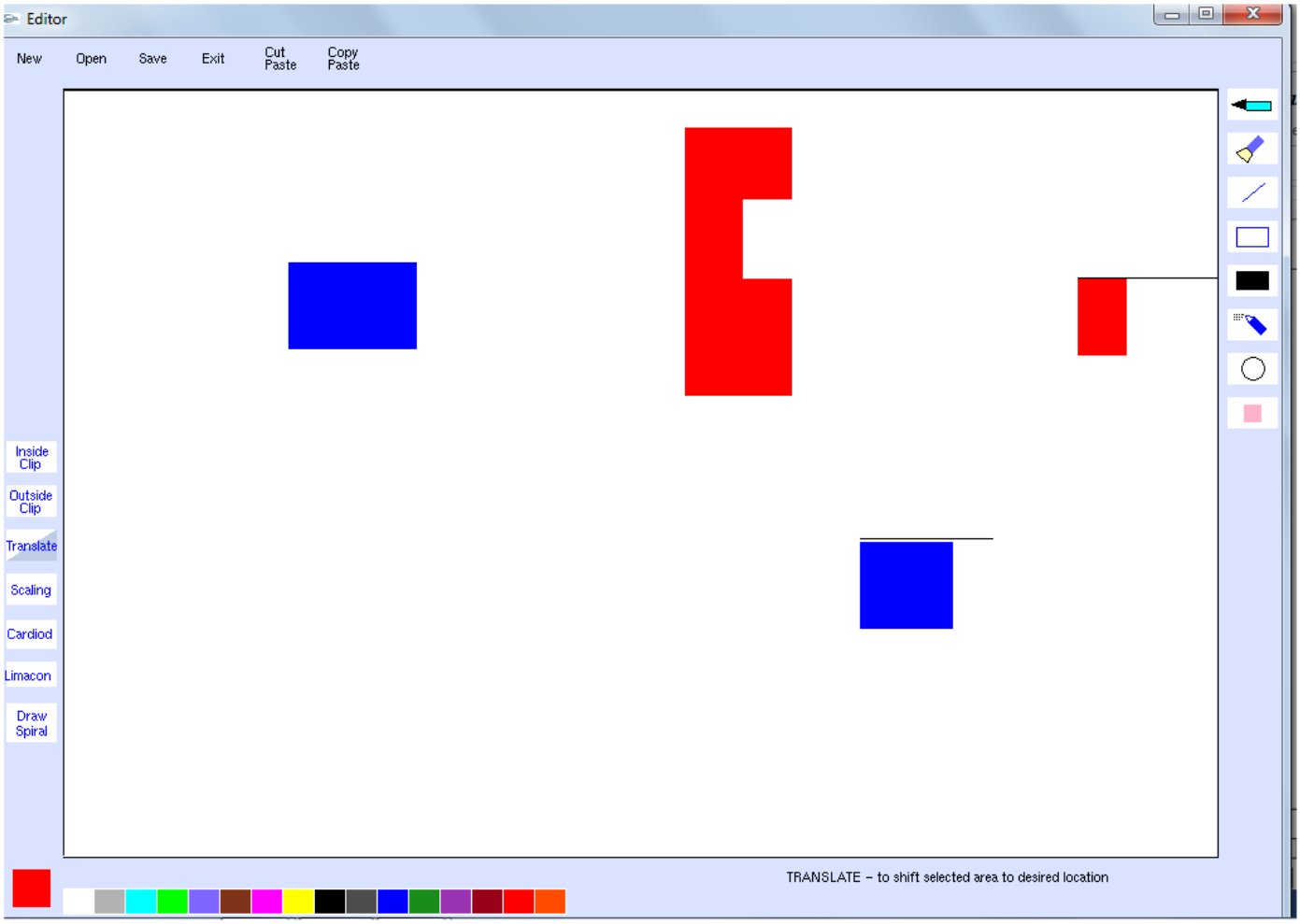

Fig B.3 Translation

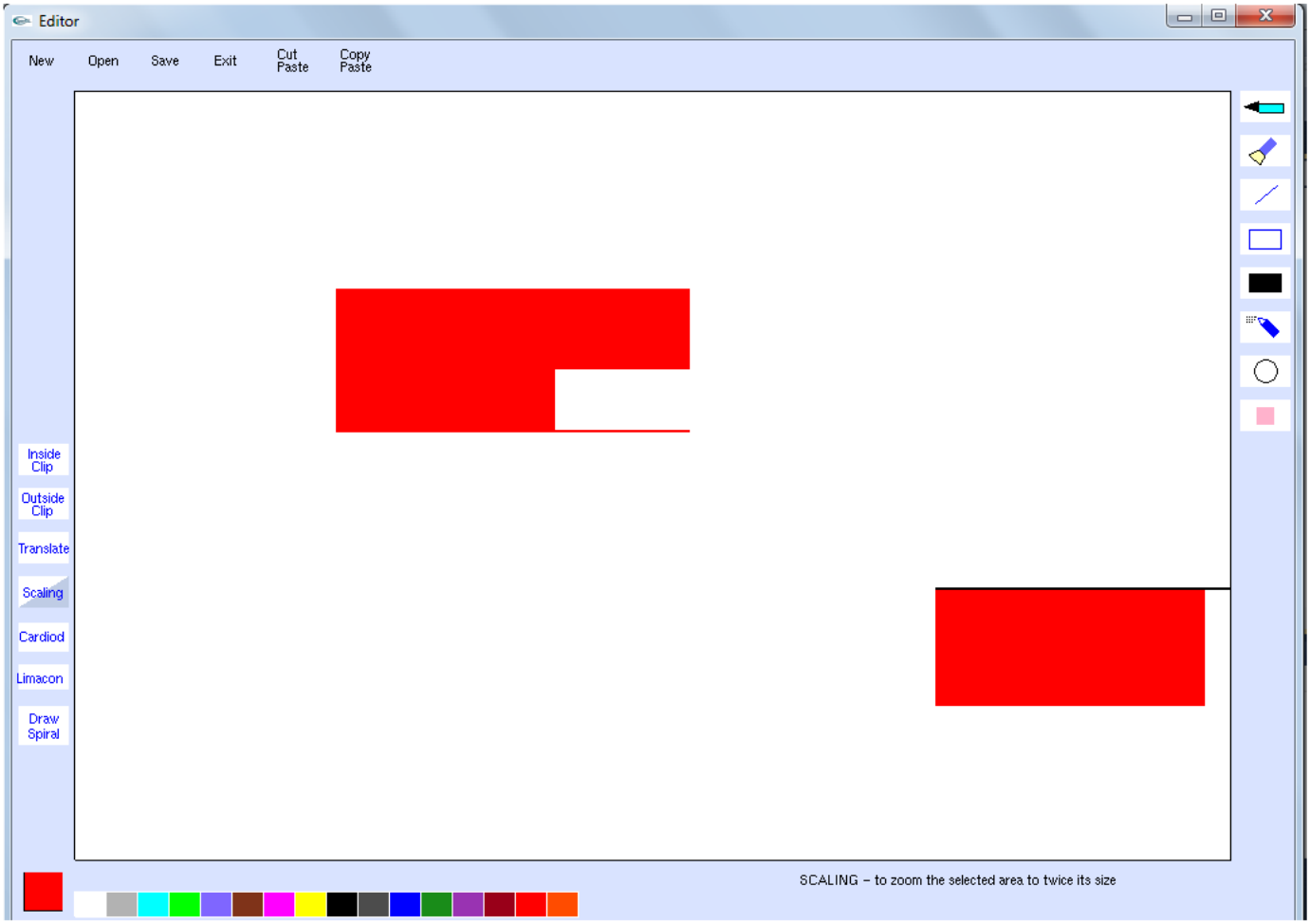

Fig B.4 Scaling

Appendix C – List of Figures

7. Bibliography

[1] http://www.files32.com/Types-Of-Clipping-In-Computer-Graphics.asp

[2] Interactive Computer Graphics A Top-Down Approach Using OpenGL by Edward Angel Pearson Education, 5th Edition.

[3] Donald Hearn, M. Pauline Baker : Computer Graphics 'C' version Pearson Education,2nd Edition.

[4] F. S. Hill, Jr: Computer Graphics Using OpenGL Pearson Education, 3rd Edition. Foley Van Dam Feiner Hughes : Computer Graphics Principles and practice Pearson Education, 2nd Edition in 'C'.

[5] http://www.opengl.org/

[6] http://nehe.gamedev.net/