DECLARATION

We, Samir Sheriff and Satvik N, bearing USN numbers 1RV09CS093 and 1RV09CS095 respectively, hereby declare that the dissertation entitled "**Graphical Editor in OpenGL**", completed and written by us, has not, previously, formed the basis for the award of any degree or diploma or certificate of any other University.

Bangalore

Samir Sheriff
USN:1RV09CS093

Satvik N
USN:1RV09CS095

R V COLLEGE OF ENGINEERING

(Autonomous Institute Affiliated to VTU)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the project entitled, "**Graphical Editor in OpenGL**", has been successfully carried out at R.V.C.E., Bangalore in partial fulfillment of the requirements for the award of degree in Computer Science and Engineering of Visvesvaraya Technological University, under autonomous scheme, during the academic year 2012-2013, by **Samir Sheriff and Satvik N** under our supervision and guidance.

Signature of Lab In-charge
(Mrs. Deepamala N)

Signature of Lab In-charge
(Mrs. Jyothi Shetty)

Signature of HOD
(Dr. N K. Srinath)

Name of Examiner

Signature of Examiner

1:

2:

ACKNOWLEDGEMENT

The euphoria and satisfaction of the completion of the project will be incomplete without thanking the persons responsible for this venture.

We acknowledge RVCE (Autonomous under VTU) for providing an opportunity to create a mini-project in the 5th semester. We express our gratitude towards **Prof. B.S. Satyanarayana**, principal, R.V.C.E for constant encouragement and facilitates extended in completion of this project. We would like to thank **Prof. N.K.Srinath**, HOD, CSE Dept. for providing excellent lab facilites for the completion of the project. We would personally like to thank our project guides **Mrs. Deepamala N and Mrs. Jyothi Shetty** and also the lab in charge, for providing timely assistance & guidance at the time.

We are indebted to the co-operation given by the lab administrators and lab assistants, who have played a major role in bringing out the mini-project in the present form. Bangalore

Samir Sheriff

7th semester, CSE

USN:1RV09CS093

Satvik N

7th semester, CSE

USN:1RV09CS095

ABSTRACT

A Graphics Editor is a feature in any OS that you can use to create drawings on a blank drawing area or in existing pictures. Many of the tools you use in this Graphics Editor are found in the ribbon, which is near the top of the Editor window.

The main objective of the editor is to help the user input graphical data and edit it conveniently. For ease in input and to help the user traverse through the text easily, the editor will provide functionality through the mouse, keyboard, or trackball.

The Graphics Editor in this project has been developed using various algorithm like Mid-point circle drawing algorithm, clipping algorithm, flood-fill Scan-line algorithm, Bezier curves algorithm etc. The editor has been developed in Visual Studio 2010.The various features have been supported by an easy to understand mouse 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, ring drawing, Brush, freehand drawing, curve, text, clipping, and eraser.

This project aims to develop a 2-D Graphics Editor which supports basic operations which include creating objects like lines, circles, polygons, etc and also transformation operations like translation, scaling, etc on such objects.

Contents

Chapter 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, scaling, 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 to 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 History of OpenGl

OpenGL was originally developed by Silicon Graphics, Inc. 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.

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.

1.3 Significance of OpenGL

- With different 3D accelerators, by presenting the programmer To hide the com-

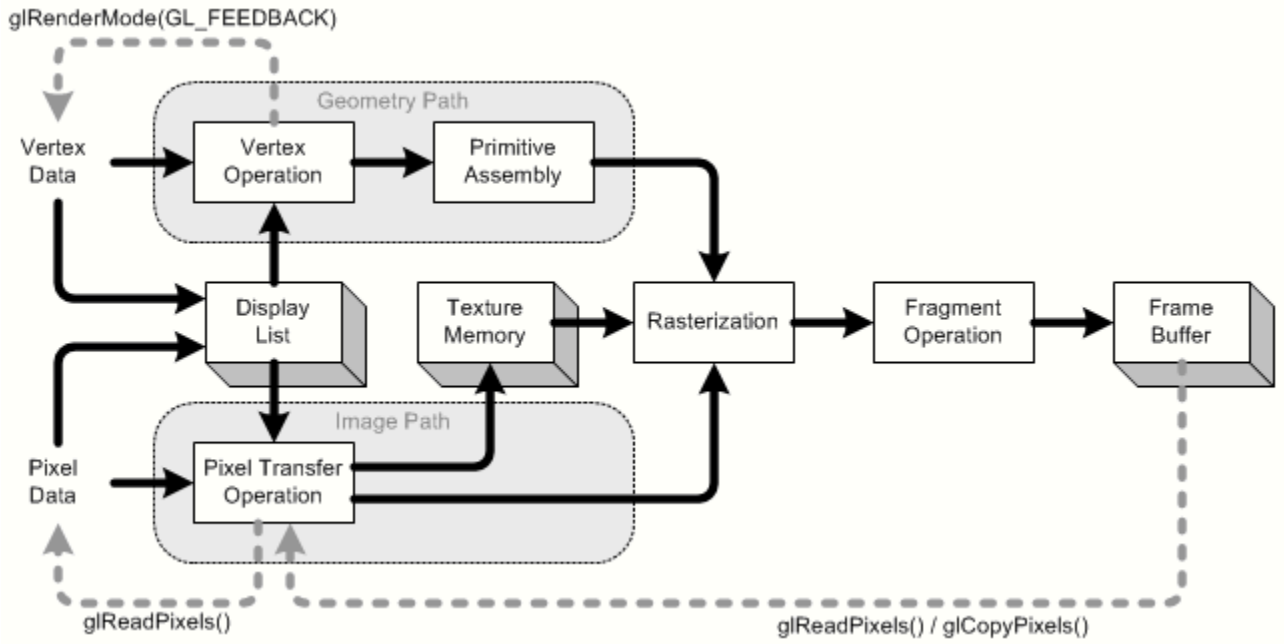


Figure 1.1: The OpenGL Graphics rendering pipeline.

plexities 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 Project in 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.

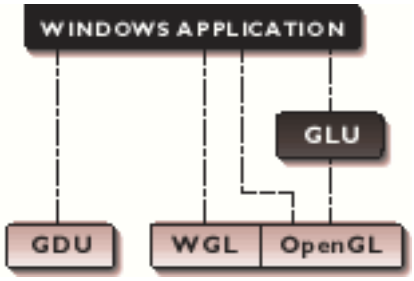


Figure 1.2: OpenGL API hierarchy under Windows system.

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 1.2 illustrates the OpenGL hierarchy under Windows.

Chapter 2

REQUIREMENT SPECIFICATION

Software Requirement Specification (SRS) is an important part of the software development process. We describe the overall description of the Mini-Project, the specific requirements of the Mini-Project, the software requirements and hardware requirements and the functionality of the system.

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 Requirement

2.2.1 Software Requirements

- Operating system: Microsoft Windows 8 / Ubuntu 12.10 / Mac OS X (Mountain Lion)
- 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.

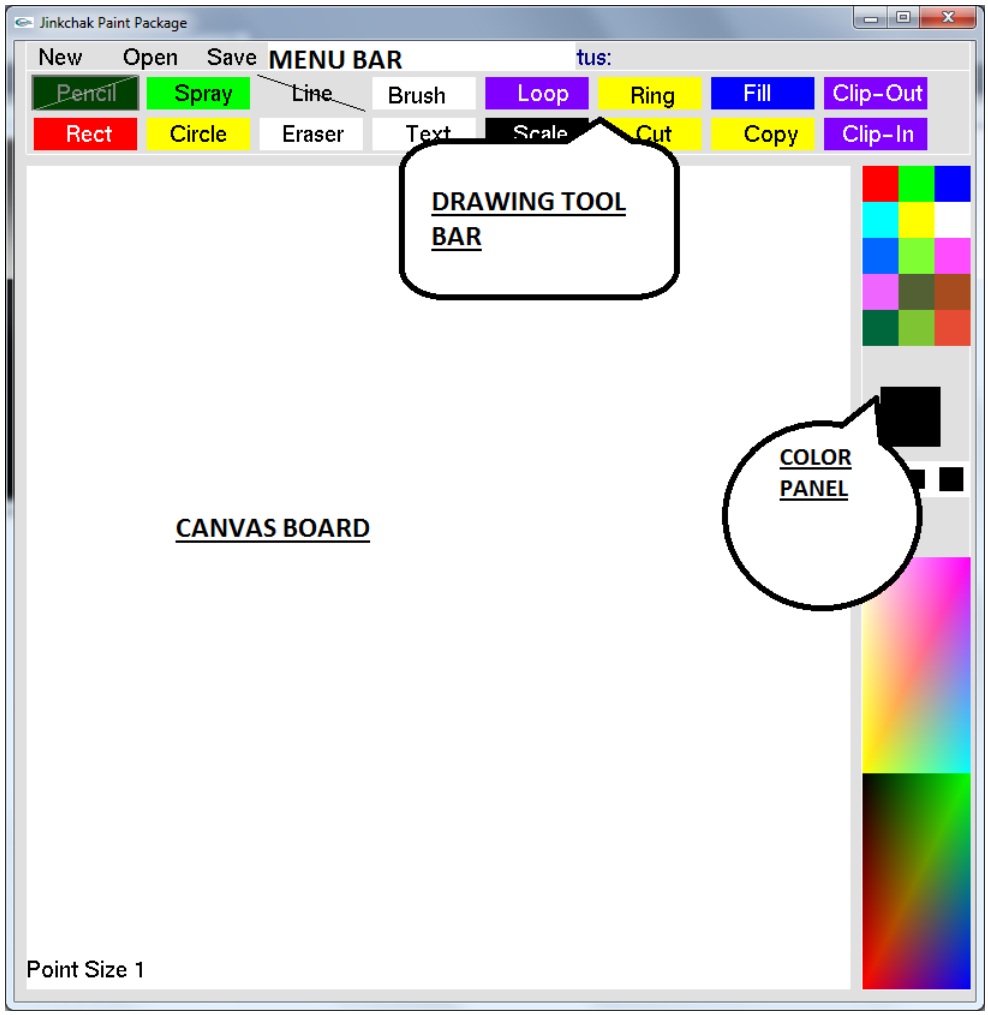
2.2.2 Hardware Requirements

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

- Processor: Intel Core i7 Extreme Edition or higher
- Processor Speed - 500 MHz or above
- RAM: 8 GB or more
- Graphics Card: Any card that supports a resolution of 1920 x 1080 or higher, with more than 3540 million transistors
- Hard disk: 7 TB or more

Chapter 3

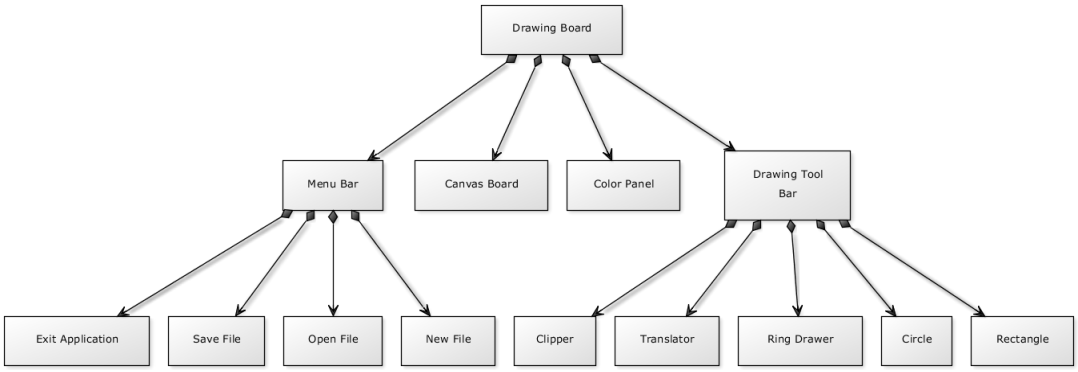
Detailed Design



3.1 Object Oriented Class Design

Object-oriented programming (OOP) is a programming paradigm that represents concepts as "objects" that have data fields (attributes that describe the object) and associated procedures known as methods. Objects, which are instances of classes, are used to inter- act with one another to design applications and computer programs.

Had it not been for the presence of the OOP paradigm, our efforts in this project would have gone in vain, and we do not use that term lightly. Code management was a whole lot easier when compared to our past experience with procedural programming. In this chapter, we describe the different classes that were created by us to efficiently manage our code.



3.1.1 DrawingBoard Class

The editor consists of a Drawing Board which, in turn, consists of a color panel on the right, a menu bar and a drawing tool bar on top, and a canvas occupying the rest of the window. Each part of the drawing board is discussed in the sections that follow.

This is the main drawing board class that contains an object of other classes - DrawingToolBar , Color Panel, MenuBar. This class contains some very important methods:

- **initBoard()** - To initialize the board. This method draws all the application window contents - Menubar, Color Panel, Canvas etc.
- **handlKeyPress()** - To handle the user key presses on the keyboard

- **handleMouseClick()** - To handle the mouse click
- **handleMouseClickMotion()** - To handle the mouse click motion event.

The DrawingBoard class only checks in which part of the window a mouse or keyboard event has occurred, and passes control of the program to the respective object (Eg., if the user clicks inside the color panel, the drawing board decides that the color panel and nothing else should handle the click, and hence, it calls the Color Panels click handler

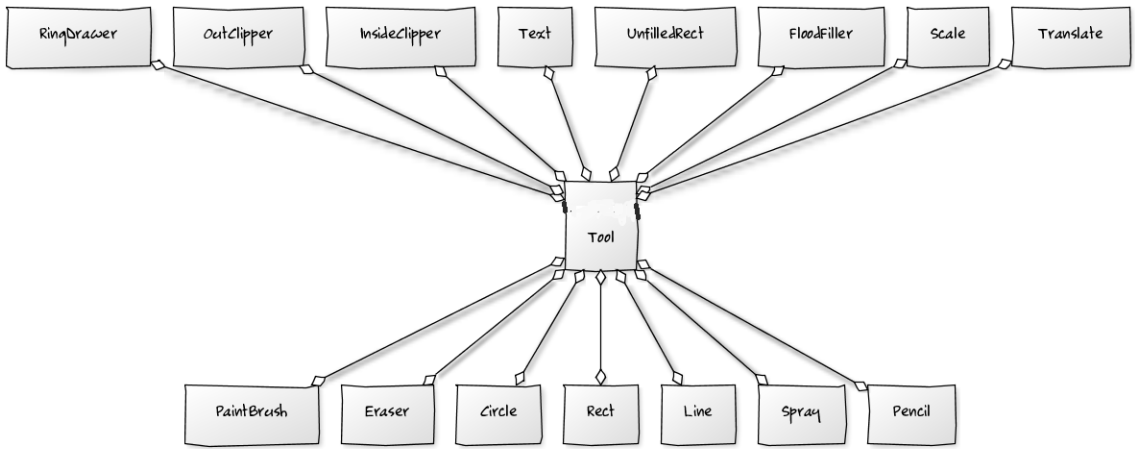
3.1.2 DrawingTool Class

This class manages the toolbar panel below the menu bar. It keeps track of all tools present, and provides methods to check whether the user has clicked anywhere in its vicinity. It also provides a draw method that is called by the display function to refresh the contents of the drawing toolbar, when required by the display function of OpenGL. It also keeps track of the current point size of the OpenGL state machine, and provides methods to increase or decrease its size.

In order to keep track of all the tools in the toolbar and call its respective rendering method when selected, the DrawingTool class makes use of another class called the Tool class. This class is an abstract class from which all other tools are derived through inheritance, as shown in the figure. A derived class should always implement the following two methods:

- **render()** - This defines the appearance of this tools button on the Drawing Toolbar
- **drawOnCanvas()** - This defines how the canvas should be manipulated when this tool is selected by the user.

A 2-D array of type Tool is maintained, which contains pointers to each tool in the toolbar. The DrawingTool object calls these two methods as an when required. Adding new tools is as easy as creating a new class with the Tool class as the base class, and implementing the two methods mentioned above. No other changes need to be made to any other classes with this design (except in emergencies, due to stupid cyclic dependency errors, which C++ cant handle, as in the case of the Text Tool).



Here's a description of some of the tools in our editor:

- A **pencil** is a writing implement or art medium usually constructed of a narrow, solid pigment core inside a protective casing. The pencil in our editor allows the user to create free-hand drawings as though he/she were using a real pencil.
- **Spray** painting is a painting technique where a device sprays a coating (paint, ink, varnish, etc.) through the air onto a surface. The spray in our editor attempts to reproduce this effect virtually.
- The notion of **line or straight line** was introduced by ancient mathematicians to represent straight objects with negligible width and depth. A line, in our editor, is drawn by selecting two points - a start point and an end point. The start point begins with a mouse click and the end point is the point where mouse button is released.
- A **rectangle** is any quadrilateral with four right angles. A user can draw a rectangle, the size of which can be adjusted in real time, by moving the mouse.
- A **circle** is a simple shape of Euclidean geometry that is the set of all points in a plane that are a given distance from a given point, the centre. A user can draw such a circle, the size of which can be adjusted in real time, by moving the mouse.

- An **eraser** (US and Canada) or rubber (elsewhere) is an article of stationery that is used for removing pencil markings, and the eraser in our editor does just that. The size of the eraser can be varied, either from the color panel, or by using the , or . keys.

- The **ring** tool can be used to draw eccentric circles to create a pseudo-3D effect. For instance, clicking and dragging the mouse pointer in the canvas when this tool is selected, produces a pseudo-cone.

- **Text** may refer to the representation of written language. In order to use the text tool, the user must first select it from the toolbar. Then, he/she must select the text operator from the Menubar and type out the necessary text using the typewriter. Once done, the enter key should be pressed, and that's it. The user can click the mouse anywhere in the canvas to replicate the text typed.

- The **scale** ratio of some sort of model which represents an original proportionally is the ratio of a linear dimension of the model to the same dimension of the original. Our editor allows parts of the canvas to be scaled up or down for easy viewing.

- In human-computer interaction, **cut and paste and copy and paste** are related commands that offer a user-interface interaction technique for transferring pixels from a source to a destination. Our editor allows the user to draw a bounding box around the region that has to be cut/copied. Once done, the user can **click and drag** anywhere else around the canvas to paste the selection. Once the paste location is decided, the user can release the mouse button.

- The **fill** tool of our editor determines which parts of the canvas to fill with a color selected from the color panel.

- Any procedure which identifies that portion of a picture which is either inside or outside a region is referred to as **clipping**. Our editor provides this extremely useful feature.

3.1.3 Color Panel Class

The object of this class represents the color panel that is present in the right part of the application. This panel consists of the following main sections

- **Basic Color Block Section** : This section shows the basic color block squares which can be selected by clicking on them.
- **Selected Color Box:** The selected color either in the basic color block or in the color selection triangle is shown in this color box.
- **Size Selectors :** This section is used to select the drawing size of the current tool. There are three sizes small, medium, large.
- **Color Blender** - There are two color blenders based on RGB color model and CMYK color model. A variety of colors can be selected from this section

3.1.4 MenuBar Class

In a restaurant , a menu is a presentation of food and beverage offerings. However, you will be pleased to know that the menu in our editor is not like that. Our menu is a list of options or commands presented to an operator by a computer or communications system (the Chicken Curry editor, in our case)

In order to keep track of all the menu operations in the menubar and call its respective rendering method when selected, the **MenuBar** class makes use of another class called the **MenuOp** class. This class is an abstract class from which all other menu operations are derived through inheritance, as shown in the figure. A derived class should always implement the following two methods:

- **render()** - This defines the appearance of this toolś button on the Drawing Toolbar
- **performOperation()** - This defines the operation that should be carried out when this op is clicked

The MenuBar object calls these two methods as and when required. Adding new tools is as easy as creating a new class with the **MenuOp** class as the base class, and

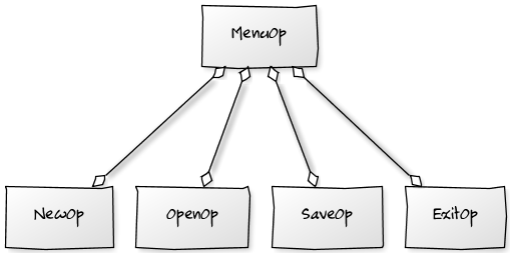


Figure 3.1: Application Window - Start Up Screen

implementing the two methods mentioned above. No other changes need to be made to any other classes with this design.

Menu Options supported in the Graphics editor

- **New** - To create a new canvas used for drawing.
- Open File - To open an already saved canvas drawing file.
- Save File - To save the current canvas drawing to a file.
- Exit application - To exit our Application.

When the user clicks the open or save buttons, it ain't a good idea to ask him/her to enter the filename in the console. It would be better for everyone involved, to accept input from the user's keyboard in the editor's window itself. The TypeWriter class was born for this very purpose. It saves the day by offering the following services to the MenuOp class:

- It can be started by one operation and requested not to accept more requests until it is done
- While it is buffering input from the user after being turned on, it doesn't accept any more requests, hence, preventing race conditions. It turns on a busy indicator.
- Once done buffering input (when the user presses the enter key), it switches off the busy indicator LED and the requesting party is provided with the contents of the typewriter's buffer to do whatever it feels like with it.
- It writes its output on the right side of the Menu Bar.

3.1.5 Canvas Class

A canvas is an extremely heavy-duty plain-woven fabric, popularly used by artists as a painting surface, typically stretched across a wooden frame. Our canvas is a virtual representation of such an entity. It allows the user to draw whatever he/she feels like, using the tools and color palette provided.

It contains a 2D array that records every artistic endeavour of the user. It provides methods to save this array to a file so that it can be loaded later.

What happens if the user finds out that he/she has made a mistake and wants to revert to an older state. Well, say hello to our custom History engine that tracks as many as 30 major updates of the canvas, and allows the user to undo changes, with only a touch of the F2 button. A redo facility, mapped to the F3 key, has also been provided, to head back to the future!

Chapter 4

CONCLUSION AND FUTURE WORK

4.1 Summary

After toiling for an unknown amount of time, we have managed to create a state-of-the-art Graphical editor, designed with object-oriented principles. Our unique design and its documentation allows the Editor's functionality to be easily extended. This editor was, is and will always be free and, hopefully, can be used in schools, colleges and universities, as an alternative to Microsoft Paint, and as a teaching tool for Object-Oriented Programming and OpenGL.

4.2 Limitations

- The graphics editor doesn't work in full screen.
- The drawn picture cannot be saved as a picture file like png, jpeg. It is saved as a binary file.
- There is a lag when using tools like pencil, spray, etc. are used.
- Three dimensional viewing and editing of pictures are not possible.

4.3 Future enhancements

- This package was developed on Windows 7, and can be made a platform independent application.
- The drawings can be made to save as pictures and help the user share them in various social networking sites like facebook, google+ etc.
- This package can be made efficient by exploiting the opengl graphics pipeline system.
- Various special tools like Scan fillers, image enhancing tools can be added.

Bibliography

[1] OpenGL Book - A free online opengl programming book. -http://openglbook.com/

[2] OpenGL - Wikipedia, the free encyclopedia - http://en.wikipedia.org/wiki/OpenGL

[3] OpenGL Programming Guide - https://www.glprogramming.com/red/

[4] OpenGL SuperBible - http://www.starstonesoftware.com/OpenGL/

[5] Opengl Graphics Editor in Google Code- http://code.google.com/p/opengl-graphics-editor/

[6] Opengl on github - https://github.com/samiriff/GraphicalEditor

Appendices

Appendix A : Source Code

```
1  #include "Canvas.h"
2  #include "ColorPanel.h"
3  #include "DrawingToolBar.h"
4  #include "MenuBar.h"
5  #include "SizeSelector.h"
6  #include<gl/glut.h>
7  /*
8   *This is the main drawing board class.
9   * supposed to contain the object of other classes like toolbar,
10  colorpanel, menubar, and the drawing canvas on
11  which everything is drawn.
12  */
13 class DrawingBoard
14 {
15 private:
16          Canvas *canvas_board;
17          ColorPanel *color_panel;
18          MenuBar *menu_bar;
19          DrawingToolBar *drawingToolBar;
20          SizeSelector *sizeSelector;
21 public:
22          DrawingBoard();
23          void initBoard();
24          void handleMouseClick(int, int, int, int);
25          void handleMouseClickMotion(int, int);
26          void handleKeyPress(unsigned char, int, int);
27          void handleSpecialKeyPress(int, int, int);
28          ~DrawingBoard();
```

```
29
30          Canvas *getCanvas();
31
32  };
33  DrawingBoard::DrawingBoard()
34  {
35          canvas_board = new Canvas(CANVAS_LEFT,CANVAS_BOTTOM, CANVAS_RIGHT,
                CANVAS_TOP);
36          color_panel = new ColorPanel(COLOR_PANEL_LEFT, COLOR_PANEL_BOTTOM,
                COLOR_PANEL_RIGHT, COLOR_PANEL_TOP);
37          menu_bar = new MenuBar(MENUBAR_LEFT, MENUBAR_BOTTOM,
                MENUBAR_RIGHT, MENUBAR_TOP, canvas_board);
38          drawingToolBar = new DrawingToolBar( TOOLBAR_LEFT, TOOLBAR_BOTTOM,
                TOOLBAR_RIGHT, TOOLBAR_TOP);
39          sizeSelector = new SizeSelector(SIZ_SELECTOR_LEFT, SIZ_SELECTOR_BOTTOM,
                SIZ_SELECTOR_RIGHT, SIZ_SELECTOR_TOP);
40  }
41  void DrawingBoard::handleKeyPress(unsigned char c, int x, int y)
42  {
43          switch(c)
44          {
45          case ',':
46                  drawingToolBar->IncreasePointSize();
47                  break;
48          case '.':
49                  drawingToolBar->DecreasePointSize();
50                  break;
51          }
52          menu_bar->getTypeWriter()->addChar(c);
53          glutPostRedisplay();
54
55  }
```

```
56
57 void DrawingBoard::handleSpecialKeyPress(int c, int x, int y)
58 {
59         LOG("DrawingBoard Special Key Function");
60
61         switch(c)
62         {
63         case GLUT_KEY_F2:
64                 canvas_board->restoreOld();
65                 break;
66         case GLUT_KEY_F3:
67                 canvas_board->restoreNew();
68                 break;
69         }
70         glutPostRedisplay();
71 }
72
73 void DrawingBoard::handleMouseClick(int button, int state, int x, int y)
74 {
75         if(button==GLUT_LEFT_BUTTON && state==GLUT_DOWN)
76         {
77                 LOG("Mouse Click (x,y) = ");
78                 LOG(x<<' '<<y);
79                 if(canvas_board->isClickInside(x,y))
80                 {
81                         LOG(" Inside CanvasBoard");
82                         drawingToolBar->getSelectedTool()->start();
83                         canvas_board->drawWithTool(drawingToolBar->getSelectedTool(),
84                                 color_panel->getSelectedColor(), x, y);
85                 }
86                 if(color_panel->isClickInside(x,y))
87                 {
```

```
88                      LOG(" Inside Color Panel");
89                      color_panel->selectClickedColorFromGrid(x, y);
90                      color_panel->selectClickedColorFromTriangle(x, y);
91                      if(sizeSelector->isClickInsideSizeSelector(x,y))
92                              drawingToolBar->setSize(sizeSelector->getSelectedSize());
93              }
94              if(drawingToolBar->isClickInside(x,y))
95              {
96                      LOG(" Inside Drawing Toolbar");
97                      drawingToolBar->selectClickedToolFromGrid(x, y);
98                      drawingToolBar->setSize(sizeSelector->getSelectedSize());
99              }
100             if(menu_bar->isClickInsideMenu(x,y))
101             {
102                     LOG("Inside Menu Bar");
103                     //menu_bar->performMenuOperation(canvas_board);
104                     menu_bar->selectClickedOpFromGrid(x, y);
105             }
106     }
107
108
109     if(button==GLUT_LEFT_BUTTON && state==GLUT_UP)
110     {
111             drawingToolBar->getSelectedTool()->stop();
112
113             if(canvas_board->hasCanvasChanged())
114             {
115                     LOG("Canvas Has Changed...Recording");
116                     canvas_board->recordInHistory();
117             }
118     }
119 }
```

```
120
121 void DrawingBoard::handleMouseClickMotion(int x, int y)
122 {
123         sizeSelector->updateXYInfo(x,y);
124         if(color_panel->isClickInside(x,y))
125         {
126                 color_panel->selectClickedColorFromTriangle(x, y);
127         }
128         if(canvas_board->isClickInside(x,y))
129         {
130                 LOG(" Inside CanvasBoard");
131                 canvas_board->drawWithTool(drawingToolBar->getSelectedTool(), color_panel
                        ->getSelectedColor(), x, y);
132         }
133 }
134 void DrawingBoard::initBoard()
135 {
136         canvas_board->drawBoard();
137         color_panel->drawPanel();
138         menu_bar->addMenuBar();
139         drawingToolBar->drawToolBar();
140         sizeSelector->draw();
141 }
142
143 DrawingBoard::~DrawingBoard()
144 {
145         delete canvas_board;
146         delete color_panel;
147         delete menu_bar;
148         delete drawingToolBar;
149 }
```

Appendix B : Screen Shots

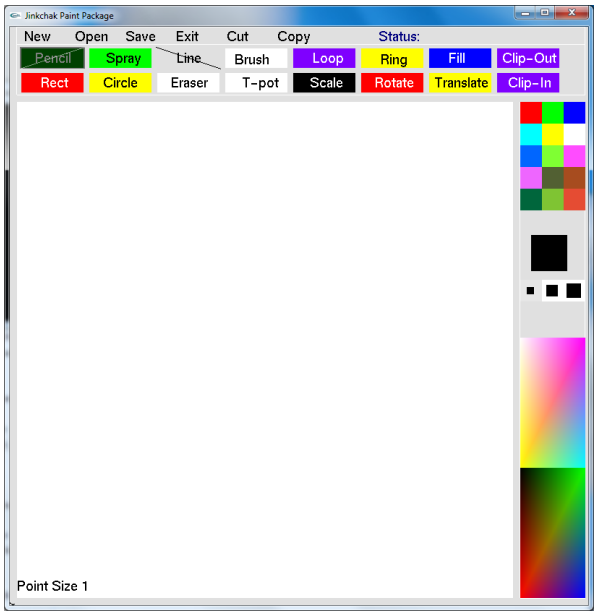


Figure 4.1: Application Window - Start Up Screen

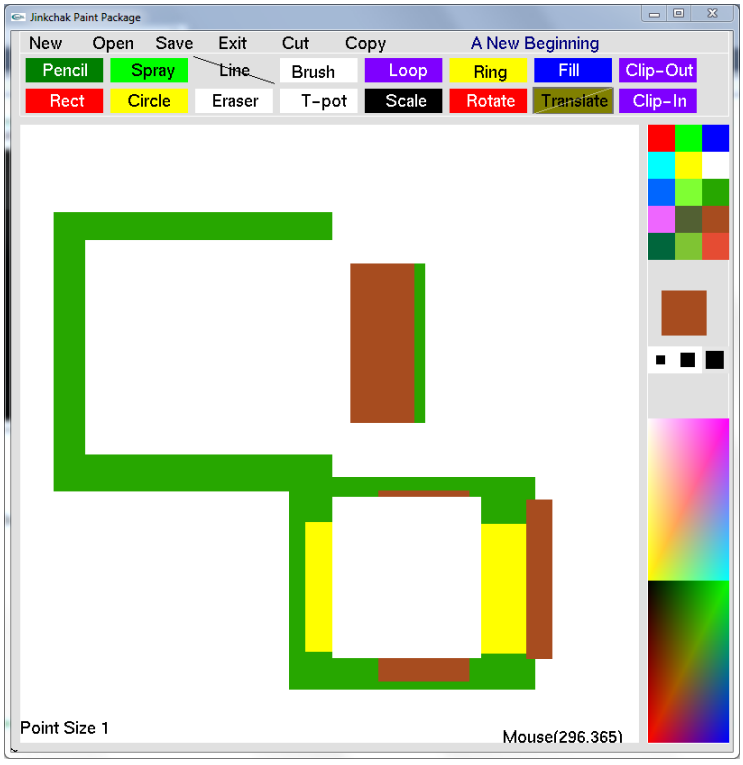


Figure 4.2: Translating a portion of a drawing using the Translate tool.

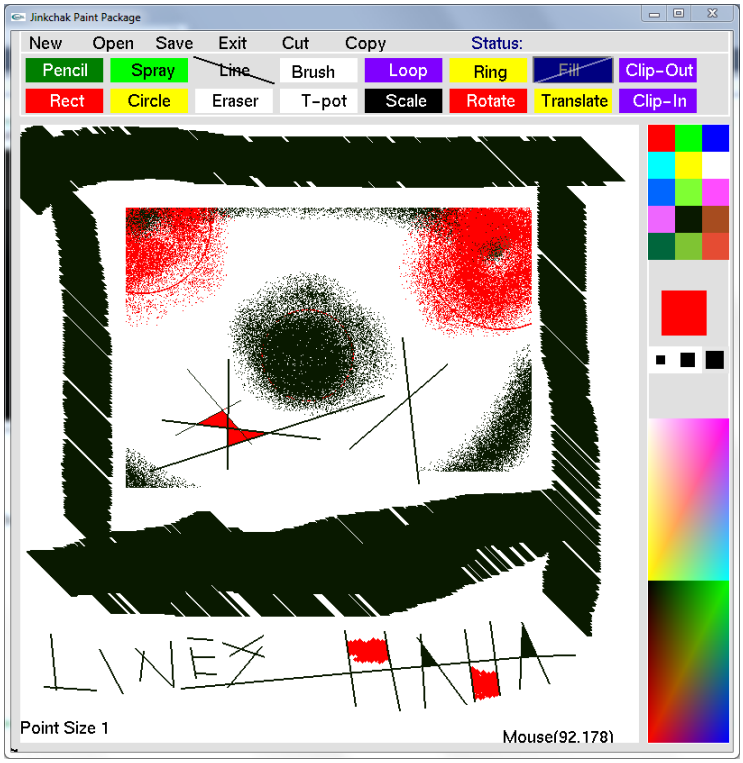


Figure 4.3: Drawing on the canvas with lines.

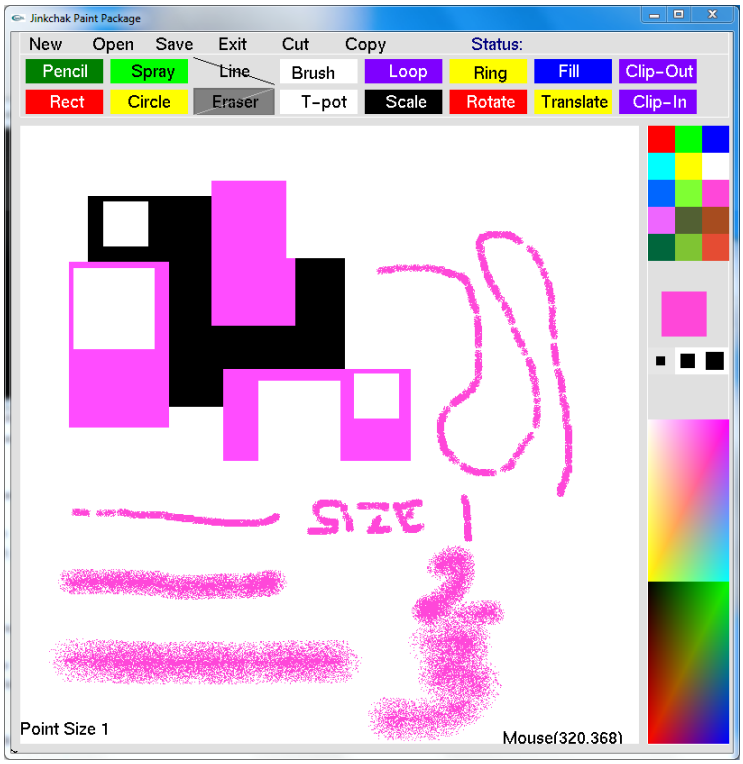


Figure 4.4: Drawing the rectangles and spraying the colors.

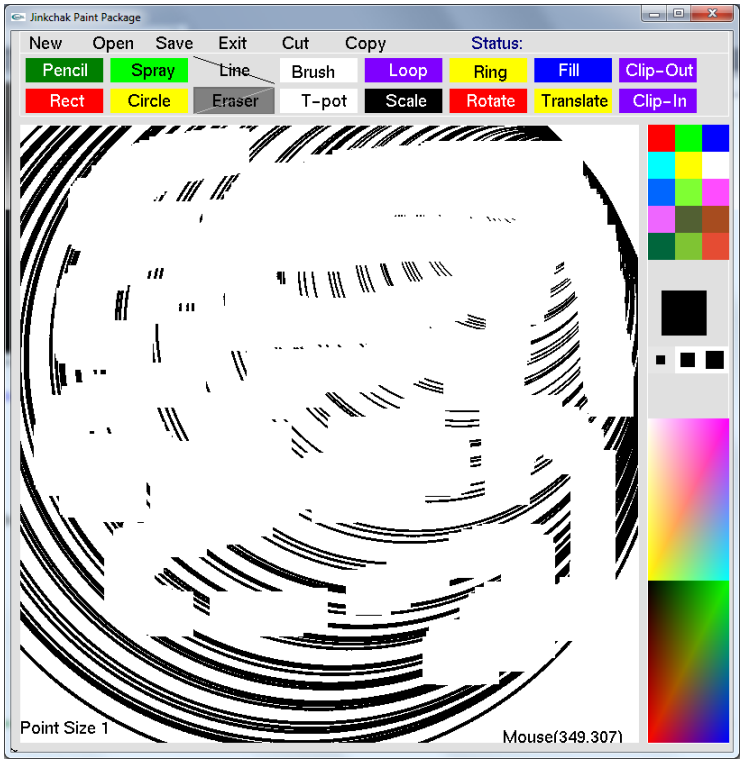


Figure 4.5: Erasing the drawing with the eraser.

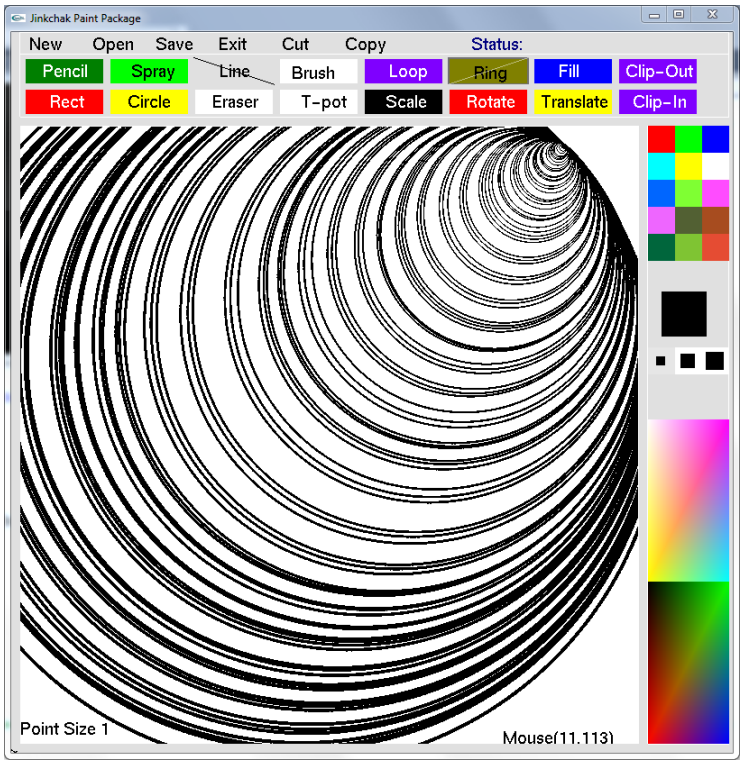


Figure 4.6: Ring Pattern - Drawn with the ring drawer special tool.

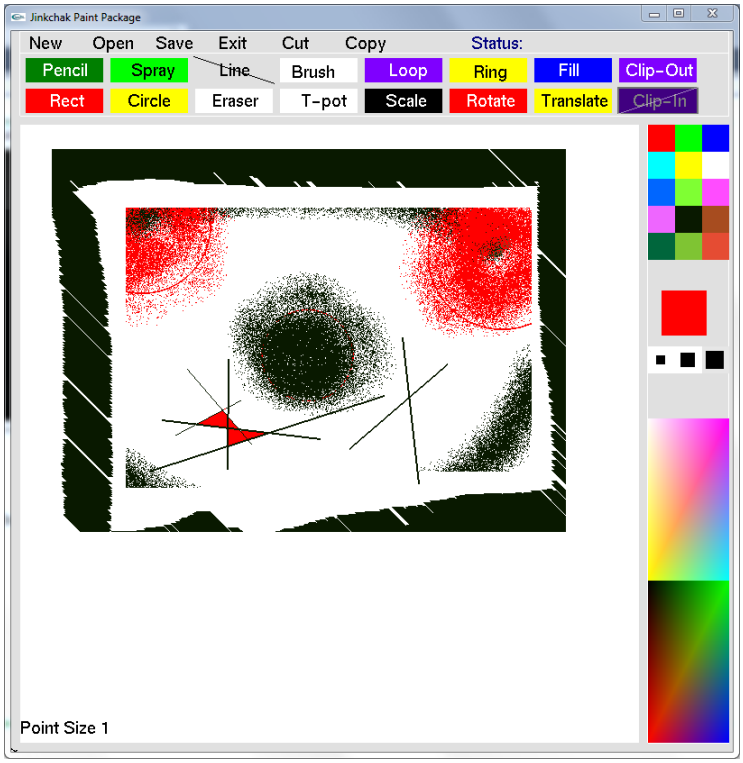


Figure 4.7: Inside Clipping with the inside clipper tool.

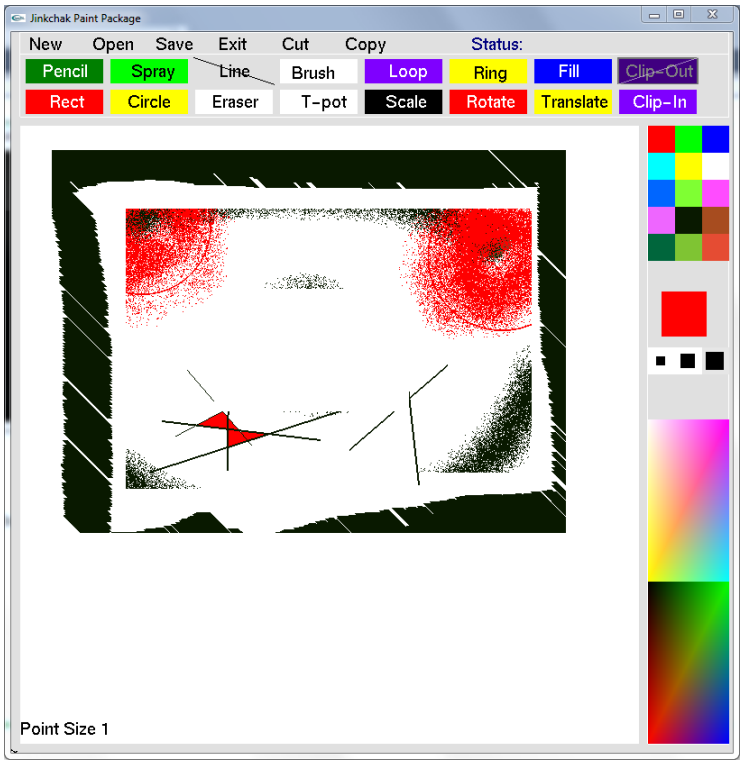


Figure 4.8: Outside clipping with the outside clipper tool.

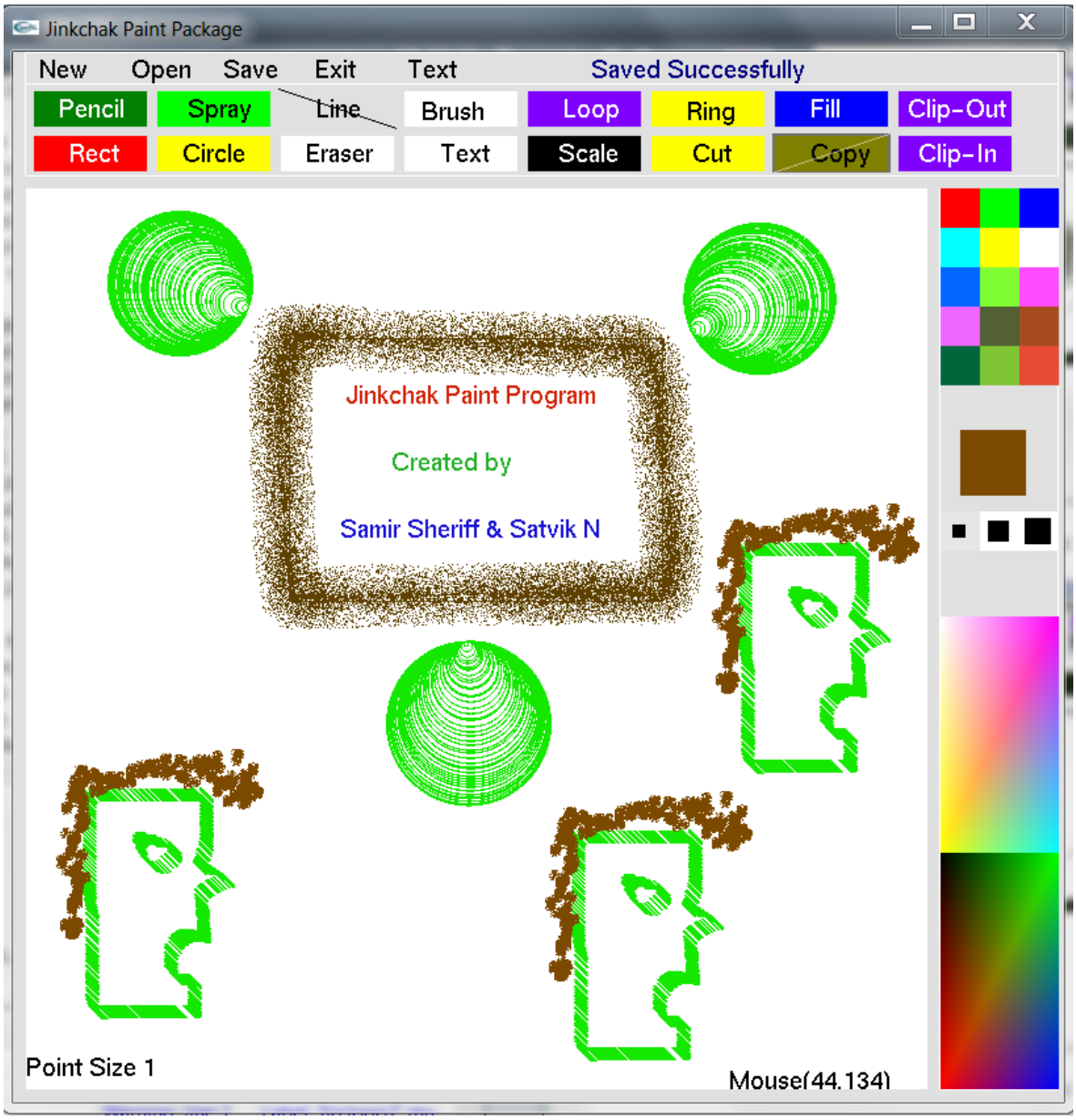


Figure 4.9: Image created in its entirety by our Editor