**RV COLLEGE OF ENGINEERING® BENGALURU – 560059**

(Autonomous Institution Affiliated to VTU, Belagavi)

## DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



**“Ball In The Basket Using OPENGL”**

## COMPUTER GRAPHICS LAB (16CS73)

**OPEN ENDED EXPERIMENT REPORT**

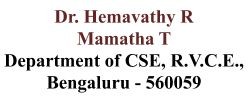
## VII SEMESTER

**2020-2021**

**Submitted by**

**Anita Muddanna Halundi-1RV18CS401**

**Under the Guidance of**



**RV COLLEGE OF ENGINEERING®, BENGALURU - 560059**

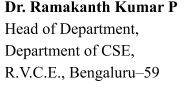
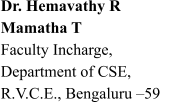
**(Autonomous Institution Affiliated to VTU, Belagavi)**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**



**CERTIFICATE**

Certified that the **Open-Ended Experiment** titled “Ball in the Basket using OpenGL” has been carried out by **Anita Muddanna Halundi(1RV18CS401),** bonafide students of RV College of Engineering, Bengaluru, have submitted in partial fulfillment for the **Internal Assessment of Course: COMPUTER GRAPHICS LAB (16CS73)** during the year 2020-2021. It is certified that all corrections/suggestions indicated for the internal Assessment have been incorporated in the report.



## RV COLLEGE OF ENGINEERING® , BENGALURU - 560059

**(Autonomous Institution Affiliated to VTU)**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

DECLARATION

We, **Anita Muddanna Halundi(1RV18CS401),** the students of Seventh Semester B.E., Computer Science and Engineering, R.V. College of Engineering, Bengaluru hereby declare that the mini-project titled **“**Ball In The Basket Using OPENGL**”** has been carried out by us and submitted in partial fulfillment for the **Internal Assessment of Course: COMPUTER GRAPHICS LAB (16CS73) - Open-Ended Experiment** during the year 2020-2021. We do declare that matter embodied in this report has not been submitted to any other university or institution for the award of any other degree or diploma.

**Place: Bengaluru Anita Muddanna Halundi**

**Date:07/01/2021**

**Table of Contents**

1. **Introduction**
   1. **Computer Graphics**
   2. **OpenGL**
      1. **OpenGL Graphics Architecture**
      2. **Primitives and Attributes**
      3. **Color, Viewing and Control Functions**
   3. **Proposed System**
      1. **Objective of the project**
      2. **Methodology**
      3. **Scope**
2. **Requirement Specifications**
   1. **Hardware Requirements**
   2. **Software Requirements**
3. **System Design and Implementation**
   1. **Data Flow Diagram**
   2. **Structure Chart**
   3. **Modular Description**
4. **Results and Snapshots**
5. **Conclusion**
6. **Bibliography**

**APPENDIX A- SOURCE CODE**

1. **Introduction**
   1. **Computer Graphics**

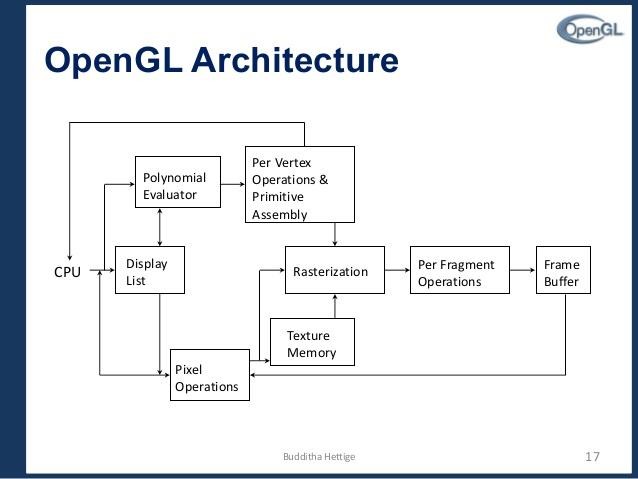
[Graphics](https://www.geeksforgeeks.org/introduction-to-computer-graphics/geeksforgeeks.org/computer-graphics-2/) is defined as any sketch or a drawing or a special network that pictorially represents some meaningful information. Computer Graphics is used where a set of image needs to be manipulated or the creation of the image in the form of pixels and is drawn on the computer. Computer Graphics can be used in digital photography, film, entertainment, electronic gadgets and all other core technologies which are required. It is a vast subject and area in the field of computer science. Computer Graphics can be used in UI design, rendering, geometric object, animation and many more. In most area, computer graphics is an abbreviation of CG.

* 1. **OpenGL**

OpenGL is a cross-platform graphics API that specifies a standard software interface for 3D graphics processing hardware. The API is typically used to interact with a [graphics processing](https://en.wikipedia.org/wiki/Graphics_processing_unit) [unit](https://en.wikipedia.org/wiki/Graphics_processing_unit) (GPU), to achieve [hardware-accelerated](https://en.wikipedia.org/wiki/Hardware_acceleration) [rendering](https://en.wikipedia.org/wiki/Rendering_(computer_graphics)).

* + 1. **OpenGL Graphics Architecture**

The OpenGL architecture is structured as a state-based pipeline. Below is a simplified diagram of this pipeline. Commands enter the pipeline from the left.



**Figure 1. OpenGL Architecture**

Commands may either be accumulated in display lists, or processed immediately through the pipeline. Display lists allow for greater optimization and command reuse, but not all commands can be put in display lists.

The first stage in the pipeline is the evaluator. This stage effectively takes any polynomial evaluator commands and evaluates them into their corresponding vertex and attribute commands.

The second stage is the per-vertex operations, including transformations, lighting, primitive assembly, clipping, projection, and viewport mapping.

The third stage is rasterization. This stage produces fragments, which are series of framebuffer addresses and values, from the viewport-mapped primitives as well as bitmaps and pixel rectangles.

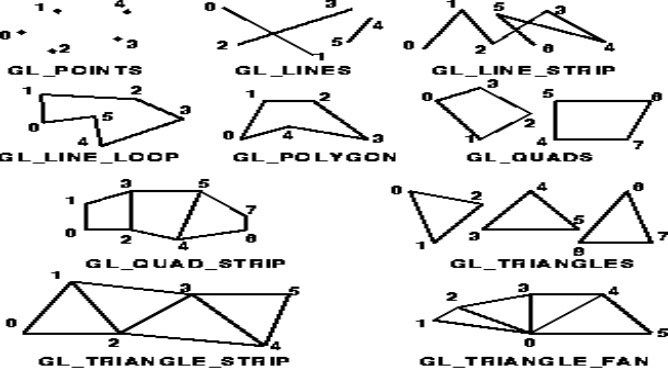
The fourth stage is the per-fragment operations. Before fragments go to the framebuffer, they may be subjected to a series of conditional tests and modifications, such as blending or z-buffering.

Parts of the framebuffer may be fed back into the pipeline as pixel rectangles. Texture memory may be used in the rasterization process when texture mapping is enabled.

## Primitives and Attributes

### OpenGL supports 2 types of primitives:

* + - * Geometric primitives (vertices, line segments) – they pass through the geometric pipeline and are subject of certain geometric functions.
      * Raster primitives (arrays of pixels) – passes through a separate pipeline to the frame buffer. They lack geometry and are array of pixels.



**Figure 2. OpenGL Primitives and Attributes.**

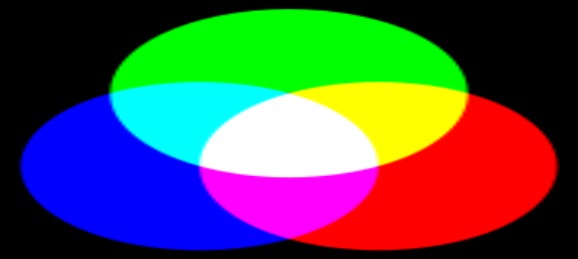
## Color, Viewing and Control Functions

**Color:**

There are many ways to specify a color in computer graphics, but one of the simplest and most widely used methods of describing a color is the RGB color model. RGB stands for the colors red, green and blue: the additive primary colors. Each of these colors is given a value, in OpenGL usually a value between 0 and 1. 1 means as much of that color as possible, and 0 means none of that color. We can mix these three colors together to give us a complete range of colors, as shown to on the left.

For instance, pure red is represented as (1, 0, 0) and full blue is (0, 0, 1). White is the combination of all three, denoted (1, 1, 1), while black is the absence of all three, (0, 0, 0). Yellow is the combination of red and green, as in (1, 1, 0). Orange is yellow with slightly less green, represented as (1, 0.5, 0).

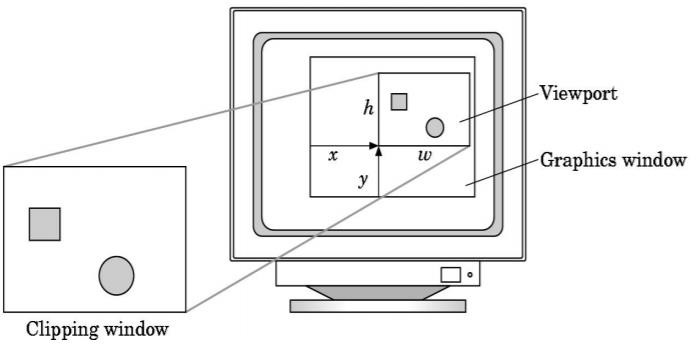
glColor3f() takes 3 arguments: the red, green and blue components of the color you want. After you use glColor3f, everything you draw will be in that color.



**Figure 3. OpenGL Colors.**

**Viewing:**

* + - * Aspect ratio is the ratio of width to height of a particular object. We may obtain undesirable output if the aspect ratio of the viewing rectangle (specified by glOrtho), is not same as the aspect ratio of the window (specified by glutInitWindowSize)
      * Viewport – A rectangular area of the display window, whose height and width can be adjusted to match that of the clipping window, to avoid distortion of the images. void glViewport(Glint x, Glint y, GLsizei w, GLsizei h).



**Figure 4. OpenGL Viewing.**

**Control Functions:**

* + - * 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 colorindex 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.)
      * glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH). If you want a window with double buffering, the RGBA color model, and a depth buffer, you might call
      * 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.
      * int glutCreateWindow(char \*string) creates a window with an OpenGL context. It returns a unique identifier for the new window. Be warned: Until glutMainLoop() is called.
  1. **Proposed System**
     1. **Objective of the project**

The objective of our project is to create a open ended game written in C++ and OpenGL.

* + 1. **Methodology**
* Creating a sphere as a ball of radius 0.2cm
* Creating a basket as a cone shaped and rotating at 90 degree
* Ball will be generated randomly and dropping from a pipe from top to down.
* Basket can be moved right and left to collect the ball .
* As the ball is collected in the box the level is increased and the speed of the ball is increased by 0.008s
* As the ball is collected in the basket the score count is increased.
  + 1. **Scope**

Main scope of the project is to build a game in which the ball is collected in the basket and score is calculated based on count.

1. **Requirement Specification**
   1. **Hardware Requirements**

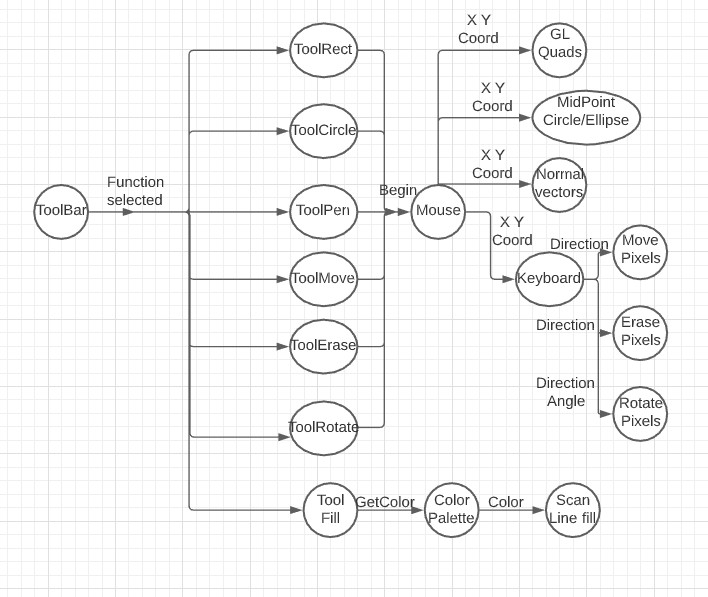
* 1.8 GHz or faster processor. ...
* 2 GB of RAM; 8 GB of RAM recommended (2.5 GB **minimum** if running on a virtual machine)
* Hard disk space: **Minimum** of 800MB up to 210 GB of available space, depending on features installed; typical installations require 20-50 GB of free space.
  1. **Software Requirements**
* Visual Studio
* OpenGl libraries
* Glut and Freeglew libraries



**Figure 5. OpenGL Software Requirements**

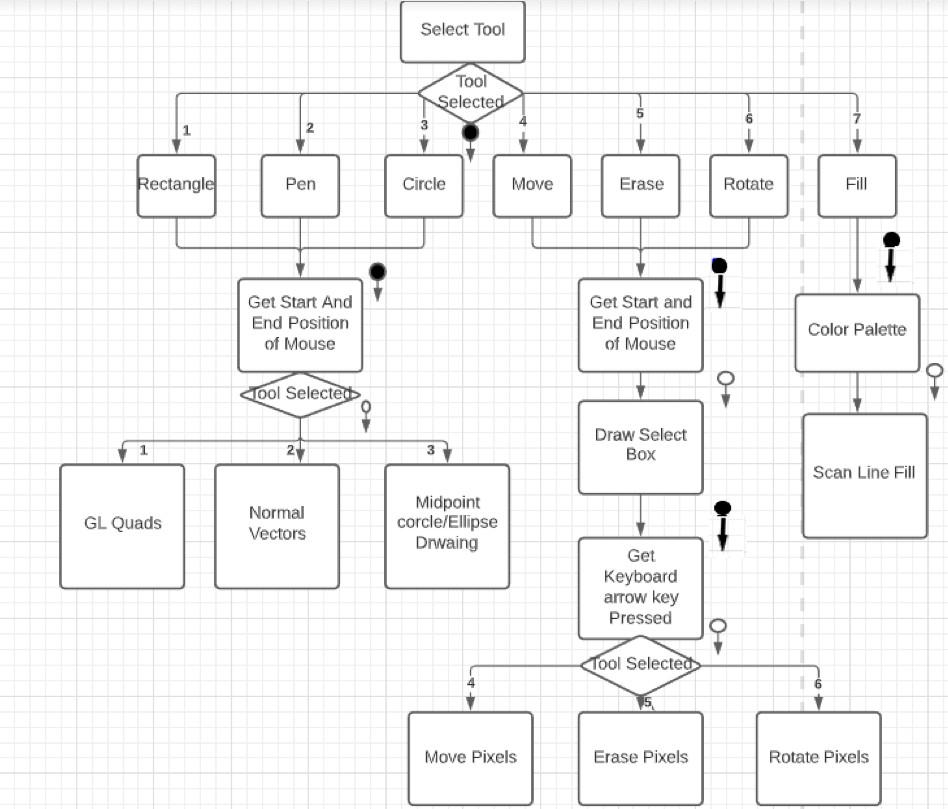
# System Design and Implementation

* 1. **Data Flow Diagram**



**Figure 6. 2D Paint Data Flow Diagram.**

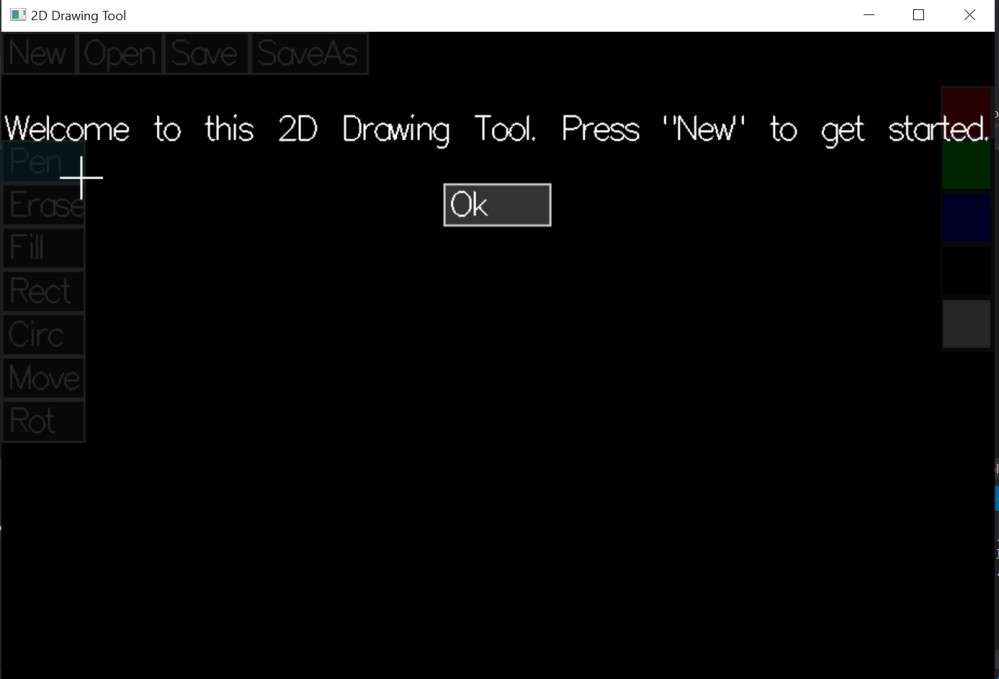
* 1. **Structure Chart**



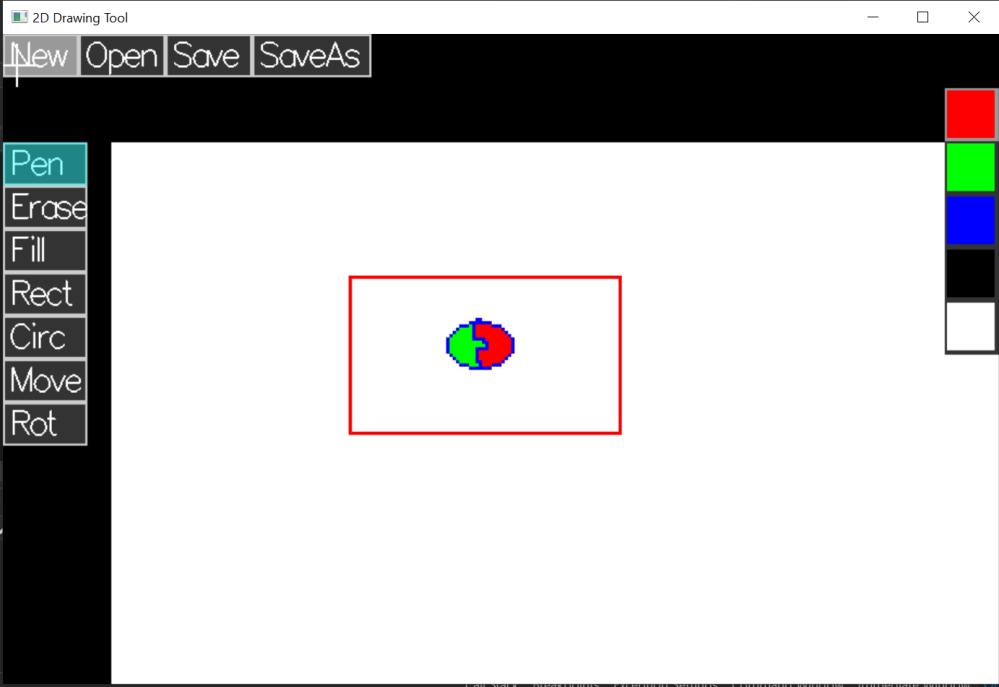
**Figure 7. 2D Paint Structure Chart.**

* 1. **Modular Description**
* Draw with pen tool – This is done using Pixel pipeline and mouse functions.
* Move objects- Using geometric pipeline to select objects and move them using pixel pipeline and keyboard functions.
* Rotate objects- Using geometric pipeline to select objects and rotate them using pixel pipeline and keyboard functions.
* Fill in areas- Using Scan line fill algorithm
* Select from palette of colors- Using rectangles to draw palettes and
* Draw rectangles and circles/ellipses- Using glquads and midpoint circle drawing and ellipse drawing algorithm with help of mouse functions.
* Save and load your files- Save files as .dti files and open them.
* Basic menus- Using Opengl Buttons to create basic menus.
* Some custom dialogues: alert messages, yes/no dialogues, open/save file, etc.

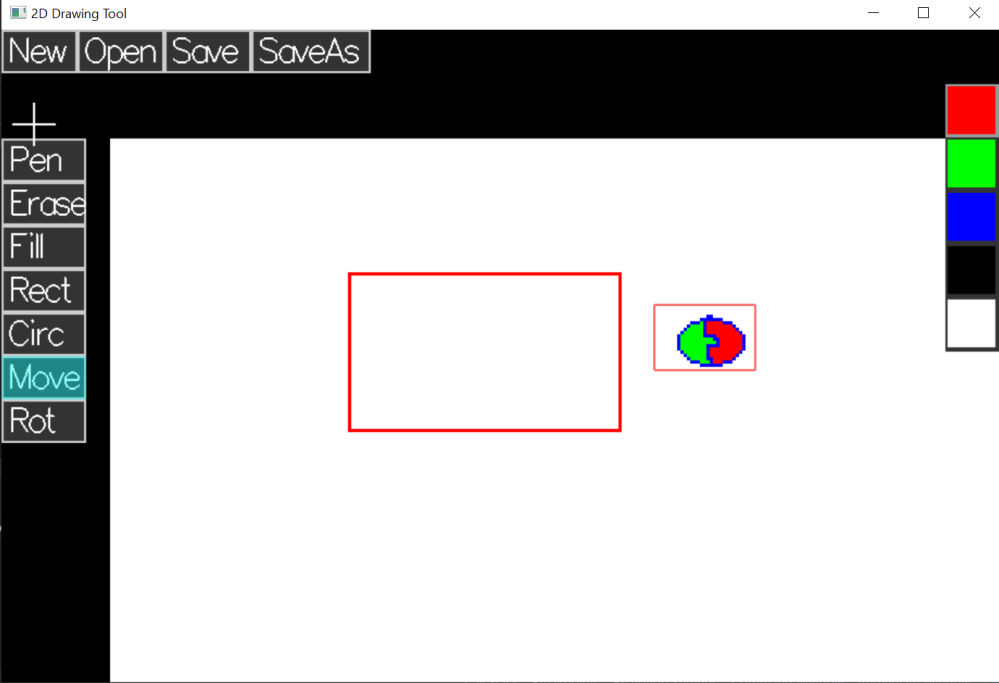
1. **Results and Snapshots**



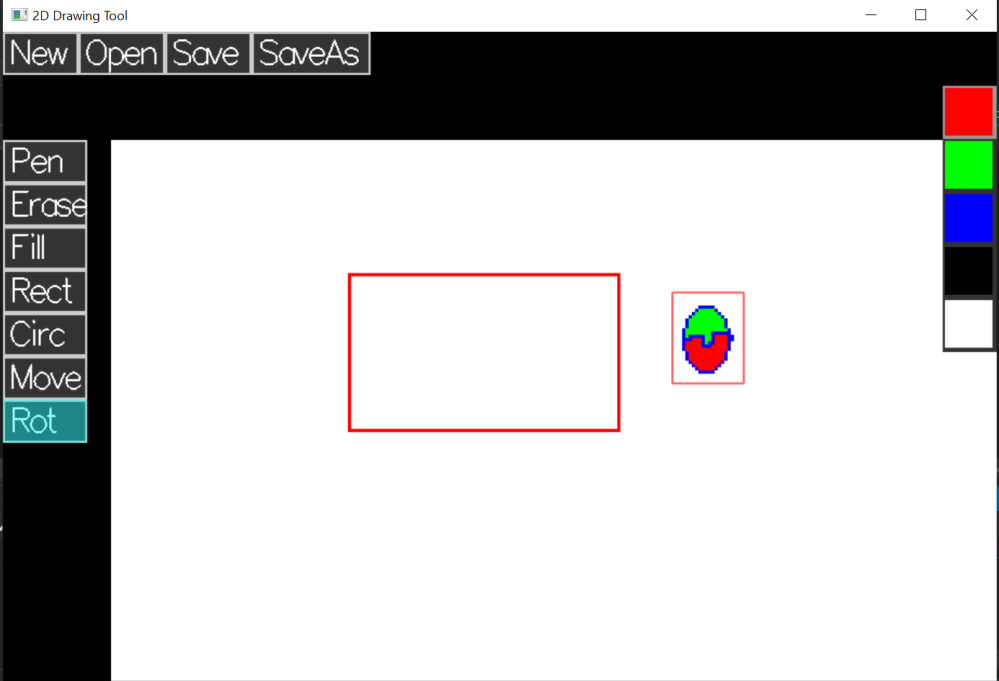
**Figure 8. Alert Box at Beginning of Application.**



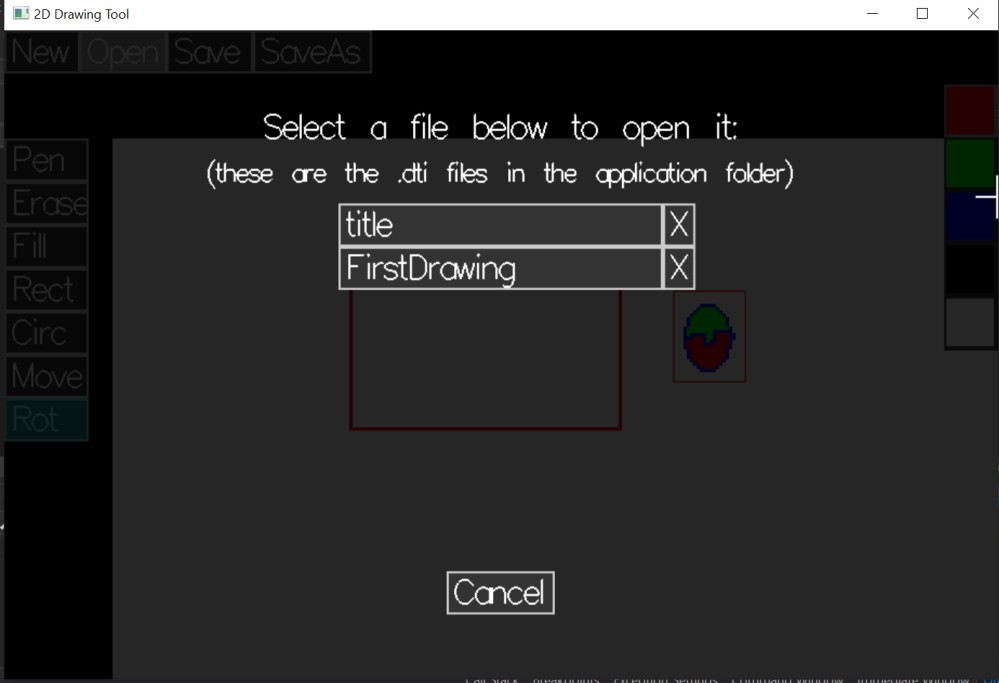
**Figure 9. Drawing using Pen, Circle, Rectangle and Fill Tools.**



**Figure 10. Moving Circle using Move tool.**



**Figure 11. Rotating using rotation tool.**



**Figure 12. Option to save and open files.**

# Conclusion

As a young kid using MS paint was very exciting to use, but little did we know about the efforts put by engineers and mathematicians in creating this simple application, until we were introduced to CG course. MS paint has been an important graphics editor used for various tasks such as editing pictures to creating paintings. Because of its intensive use of graphics function it serves as the best use case to explore various commands of OpenGL graphics library. Hence by creating this simple MS paint like application we are getting to understand practical application of huge number of OpenGL functions learnt in our theory classes.

1. **Bibliography**

* Basic 2D Paint Progra[m - https://github.com/James231/Basic-Paint-Program-OpenGL](https://github.com/James231/Basic-Paint-Program-OpenGL)
* OpenGL Paint https://users.soe.ucsc.edu/~pang/160/f09/projects/proj/ckobata/index.html
* Implementing Surfaces in OpenGL by Hui Zhao A research paper presented to the University of Waterloo
* OpenGL Paint : OpenGL

<https://www3.ntu.edu.sg/home/ehchua/programming/opengl/cg_introduction.html>

**Appendix A-SOURCE CODE**

* **Alert Dialogue.h**

/\*Alert Dialogue.hThis adds a Alert Dialogue which displays a custom message, and "Ok" button.\*/

#pragma onceclass AlertDialogue { public:static bool show;

static Button okButton; static std::string message;

/\*Displays the Alert dialogue with custom message text@param m - The message to display\*/

static void Alert(std::string m) { message = m;show = true;

// Draw the ok button and enable the black semi-transparent cover okButton.Show();

Cover::show = true;

}

/\*Hides the Alert Dialogue\*/ static void Hide()

{

show = false;

// Hide the ok button and disable the black semi-transparent cover okButton.Hide();

Cover::show = false;

}

/\*Callback function when the Ok button is pressed. Hides the Alert Dialogue@param button - The button that was pressed\*/

static void OkPressed(Button button) { Hide();

}

/\*Initialize the Alert Dialogue\*/ static void Init() {

// Create the Ok button and open the dialogue for a welcome screen okButton = Button::Create(0, 140, 100, 40, (char \*)"Ok", OkPressed, true); Alert("Welcome to this 2D Drawing Tool. Press \"New\" to get started.");

}

/\*Displays the Alert Dialogue@param window\_width - Width of the window@param window\_height - Height of the window\*/

static void Display(int window\_width, int window\_height) { if (show) {

// Display the text aligned to the center of the screen

display\_text(message, (window\_width / 2) - (get\_text\_width(message) / 2), window\_height - 100);

// Align the ok button to the center of the screen and display it okButton.HorizontallyCenter(window\_width); okButton.Display(window\_width, window\_height);

}}

/\*Handles button pressed events for alert dialogue@param button - Mouse button pressed@param state - State of mouse event (down or up)@param x - The x coordinate of the mouse when pressed@param y - The y coordinate of the mouse when pressed@return Has the alert dialogue handled the event?\*/

static bool Pressed(int button, int state, int x, int y)

{

if (show)

{

// pass event onto ok button

if (okButton.Pressed(button, state, x, y))

{return true;}

}

return false;

}

/\*Handles mouse move events for alert dialogue@param x - The new x coordinate of the mouse@param y - The new y coordinate of the mouse@return Has the alert dialogue handled the event?\*/

static bool Hover(int x, int y)

{bool output = false; if (show)

{if (okButton.Hover(x, y))

{output = true;}

}

return output;

}

};

* **Button.h :**

/\*Button.hImplements the displaying of a Button and handling of all associated events\*/

#pragma once#include <iostream>#include "Fonts.h"#include <vector>class Button;

// Define a type representing a button pressed callback function typedef void(\*Callback)(Button button);

// Class defining a button

class Button { public:

// The button pressed callback functionButton() :callback(NULL) {}Callback callback;

bool display, hovering;

int x\_pos, y\_pos, width, height;

// Text to display std::string text;

/\*Checks if a point (x,y) is inside the button or not@param x - The x coordinate of the point@param y - The y coordinate of the point@return True if the point lies within the button rectangle\*/

bool checkInside(int x, int y)

{if (x >= x\_pos)

{if (y >= y\_pos)

{if (x < x\_pos + width)

{if (y < y\_pos + height)

{return true;}

}

}

}

return false;

}

/\*Detects if the button should be in the "hover" state@param x - Mouse position x coordinate@param y - Mouse position y coordinate@return True if the button handles the hovering event\*/

bool Hover(int x, int y)

{

hovering = checkInside(x, y); return hovering;

}

/\*Handles mouse pressed events for the button@param button - Mouse button pressed@param state - State of mouse event (down or up)@param x - The x coordinate of the mouse when pressed@param y - The y coordinate of the mouse when pressed@return Has the button handled the event?\*/

bool Pressed(int button, int state, int x, int y)

{

hovering = checkInside(x, y);

// Check the mouse press was left mouse button up if (button == GLUT\_LEFT\_BUTTON)

{

if (state == GLUT\_UP)

{

if (hovering)

{

// The mouse was pressed within the button area so execute the button pressed callback

if (callback) {(\*callback)((Button)\*this);}

// return "true" as the mouse click event was handled return true;}

}

}

// button should always block the event (from UI elements underneath it) if the mouse is over the top of it

return hovering;}

/\*Draws the button on the screenThis is called from inside the "display" function@param window\_width - Width of the window@param window\_height - Height of the window\*/

void Display(int window\_width, int window\_height)

{if (display) {

// button border

glBegin(GL\_QUADS); glColor3f(0.8f, 0.8f, 0.8f);

glVertex2f(x\_pos, window\_height - y\_pos); glVertex2f(x\_pos + width, window\_height - y\_pos);

glVertex2f(x\_pos + width, window\_height - y\_pos - height);glVertex2f(x\_pos, window\_height - y\_pos - height);glEnd();// button interiorglBegin(GL\_QUADS);if (hovering) {// display a different colour in "hover" stateglColor3f(0.6f, 0.6f, 0.6f);}else {glColor3f(0.2f, 0.2f, 0.2f);}glVertex2f(x\_pos + 2, window\_height

- y\_pos - 2);glVertex2f(x\_pos + width - 2, window\_height - y\_pos - 2);glVertex2f(x\_pos + width - 2, window\_height - y\_pos - height + 2);glVertex2f(x\_pos + 2, window\_height - y\_pos - height + 2);glEnd();// Draw the button text - the constant integers add paddingdisplay\_text(text, x\_pos + 7, window\_height - y\_pos - height + 10);}}/\*Start showing this button in every display call\*/void Show() {display = true;}/\*Hides this button. Call "Show" to display it again\*/void Hide() {display = false;}/\*Set the x position of the button to the center of the screen\*/void HorizontallyCenter(int window\_width)

{x\_pos = (window\_width / 2) - (width / 2);}/\*Set values of button@param x - x position of the button@param y - y position of the button@param w - width of button@param h - height of the button@param t - text to display in the button@param c - function to call when button pressed@param d - display the button?\*/void set\_values(int x, int y, int w, int h, char\* t, Callback c, bool

1. {display = true;hovering = false;x\_pos = x;y\_pos = y;width = w;height = h;text = t;callback = c;display = d;}/\*Creates a new button@param x - x position of the button@param y - y position of the button@param w - width of button@param h - height of the button@param t - text to display in the button@param c - function to call when button pressed@param d - display the button?\*/static Button Create(int x, int y, int w, int h, char\* t, Callback c, bool d) {Button\* newButton = new Button;newButton->set\_values(x, y, w, h, t, c, d);return

\*newButton;}};

## Canvas.h :

/\*Canvas.hThis file implements a Canvas (or "image") which stores an array of pixels (rbg colour values) which can be read/written and displayed\*/#pragma once#include <string>#include <algorithm>#include

<sstream>#include <iostream>#include <cstdlib>#include

<stdlib.h>#include <cmath>class Canvas {public:int width, height;int xOffset, yOffset; // << offsets change for panningstd::string saveFilePath;Colour\*\* pixels; // << 2D array of pixel coloursfloat

zoom;std::string fileName;/\*Sets values for the canvas@param w - width of the canvas@param h - height of the canvas@param xo - x offset of the canvas@param yo - y offset of the canvas\*/void set\_values(int w, int h, int xo, int yo) {fileName = "";width = w;height = h;xOffset = xo;yOffset = yo;zoom = 3.0f;// fill in grid of pixels with white colourstruct Colour white = { 1.0f, 1.0f, 1.0f };pixels = new Colour\*[w];for (int x = 0; x < w; ++x) {pixels[x] = new Colour[h\*5];for (int y = 0; y < h; y++) {pixels[x][y] = white;}}}/\*Returns the pixel colour at the vertex@param x - x coordinate to get the colour from@param y - y coordinate to get the colour from\*/Colour GetPixelColour(int x, int y) {return pixels[x][y];}/\*Assigns a colour to the position@param x - x coordinate to set the colour of@param y - y coordinate to set the colour of@param r - red colour@param g - green colour@param b - blue colour\*/void SetPixelColour(int x, int y, float r, float g, float b)

{struct Colour newColor = { r, g, b };pixels[x][y] = newColor;}/\*Assigns a colour to the position@param x - x coordinate to set the colour of@param y - y coordinate to set the colour of@param c

- the colour to assign\*/void SetPixelColour(int x, int y, Colour c)

{pixels[x][y] = c;}/\*Convert float to string@param value - the float to convert@return The float as a string\*/std::string to\_string(float value){ std::ostringstream os ; os << value ; return os.str()

;}/\*Convert int to string@param value - The int to convert@return The int as a string\*/std::string to\_string(int value){std::ostringstream os;os << value;return os.str();}/\*Expresses canvas in string format@return The string representing the canvas\*/std::string Serialize() {// output width and height on first 2 linesstd::string output = to\_string(width) + "\n" + to\_string(height);// then list r, g, b values for all pixelsfor (int x = 0; x < width; x++) {for (int y

= 0; y < height; y++) {output += "\n" + to\_string(pixels[x][y].r);output += "\n" + to\_string(pixels[x][y].g);output += "\n" + to\_string(pixels[x][y].b);}}return output;}/\*Creates a canvas from a string@param data - Canvas in string format@return New canvas from the string\*/static Canvas Deserialize(std::string data) {std::string curLine = "";// c++ (pre c++ 11) has no string split function :(// first get width and height from first 2 linesint width = -1;int height

= -1;int i = 0;while ((width == -1) || (height == -1)) {// goes through characters and puts them together to give a complete linechar c = data[i];if (c == '\n') {if (width == -1) {width = std::atoi(curLine.c\_str());curLine = "";i++;continue;}if (height == -

1) {height = std::atoi(curLine.c\_str());curLine = "";i++;break;}}else

{curLine += c;}i++;if (i > 10000) {Canvas\* newCanvas = new

Canvas;newCanvas->set\_values(500, 500, 100, 100);return

\*newCanvas;}}// now create the Canvas using width and heightCanvas\* canvasRef = new Canvas;canvasRef->set\_values(width, height, 100, 100);Canvas canvas = \*canvasRef;// continue going through string to fill in the colourscurLine = "";int pixelNum = 0;int coloursDoneInPixel = 0;float r;float g;float b;for (int j = i; j < data.size(); j++) {char c = data[j];if (c == '\n') {if (coloursDoneInPixel == 0) {r = std::atof(curLine.c\_str());coloursDoneInPixel++;}else {if (coloursDoneInPixel == 1) {g = std::atof(curLine.c\_str());coloursDoneInPixel++;}else {b = std::atof(curLine.c\_str());int xIndex = std::floor((double)pixelNum / (double)height);int yIndex = pixelNum % height;canvas.SetPixelColour(xIndex, yIndex, r, g, b);coloursDoneInPixel = 0;pixelNum++;}}curLine = "";}else {curLine += c;}}return canvas;}/\*Draws the canvas on the screenThis is called from inside the "display" function@param window\_width - Width of the window@param window\_height - Height of the window\*/void Draw(int window\_width, int window\_height) {glBegin(GL\_QUADS);for (int x = 0; x

< width; x++) {for (int y = 0; y < height; y++) {// for each pixel draw a quad - size of quad is equal to the zoomglColor3f(pixels[x][y].r, pixels[x][y].g, pixels[x][y].b);glVertex2f((x\*zoom) + xOffset, window\_height - ((y\*zoom) + yOffset));glVertex2f(((x+1)\*zoom) + xOffset, window\_height

* ((y\*zoom) + yOffset));glVertex2f(((x+1)\*zoom) + xOffset, window\_height - (((y+1)\*zoom) + yOffset));glVertex2f((x\*zoom) + xOffset, window\_height - (((y+1)\*zoom) + yOffset));}}glEnd();}/\*Check if a point (x,y) is inside the button or not@param x - x coordinate of the point to check@param y - y coordinate of the point to check@return True if point lies inside the button\*/bool checkInside(int x, int y)

{if (x >= xOffset) {if (y >= yOffset) {if (x < xOffset + (width \* zoom)) {if (y < yOffset + (height \* zoom)) {return true;}}}}return false;}/\*Rounds float to nearest int@param num - float to round@return The rounded integer\*/int round(float num){ return std::ceil(num - 0.5);}};/\*Creates a new canvas with given width, height and offsets@param w - width of the canvas@param h - height of the canvas@param xOffset - x coordinate of canvas in window@param yOffset

* y coordinate of canvas in window@return The new canvas\*/Canvas NewCanvas(int w, int h, int xOffset, int yOffset) {Canvas\* newCanvas = new Canvas;newCanvas->set\_values(w, h, xOffset, yOffset);return

\*newCanvas;}// The canvas which is currently being displayedstatic Canvas currentCanvas;

* + **Colour Pallete.h :**

/\*Colour Palette.hThis file implements the colour palette on the right hand side\*/#pragma once#include "Canvas.h"class ColourPalette

{public:// Array of colours in the palettestatic const Colour colours[];// index of selected colour in above arraystatic int selectedIndex;// x position of the palette, based on window widthstatic int palette\_x\_pos;/\*Draws the colour palette on the screenThis is called from inside the "display" function@param window\_width - Width of the window@param window\_height - Height of the window\*/static void Display(int window\_width, int window\_height)

{palette\_x\_pos = window\_width - 50;// draw the button for each colour (these are not from the Button class! they are implemented here)glBegin(GL\_QUADS);int xPos = palette\_x\_pos;int yPos = 50;for (int i = 0; i < 5; i++) {// Draw border for the colour//different coloured border depending on whether it is selected or notif (i == selectedIndex) {glColor3f(0.6f, 0.6f, 0.6f);}else {glColor3f(0.2f, 0.2f, 0.2f);}glVertex2f(xPos, window\_height - yPos);glVertex2f(xPos + 50, window\_height - yPos);glVertex2f(xPos + 50, window\_height - (yPos

+ 50));glVertex2f(xPos, window\_height - (yPos + 50));// Now draw the colour box itselfglColor3f(colours[i].r, colours[i].g, colours[i].b);glVertex2f(xPos + 2, window\_height - (yPos + 2));glVertex2f(xPos + 46, window\_height - (yPos + 2));glVertex2f(xPos

+ 46, window\_height - (yPos + 46));glVertex2f(xPos + 2, window\_height

* (yPos + 46));yPos += 49;}glEnd();}/\*Checks if (x,y) coordinate is within a given rect@param x - x coordinate of point to test@param y - y coordinate of point to test@param x\_pos - x coordinate of the rect (top left)@param y\_pos - y coordinate of the rect (top left)@param width - width of the rect@param height - height of the rect@return True if the point lies within the rect\*/static bool checkInside(int x, int y, int x\_pos, int y\_pos, int width, int height) {if (x >= x\_pos)

{if (y >= y\_pos) {if (x < x\_pos + width) {if (y < y\_pos + height)

{return true;}}}}return false;}/\*Handles mouse pressed events for the colour palette@param button - Mouse button pressed@param state - State of mouse event (down or up)@param x - The x coordinate of the mouse when pressed@param y - The y coordinate of the mouse when pressed@return Has the colour palette handled the event?\*/static bool Pressed(int button, int state, int x, int y) {if (button == GLUT\_LEFT\_BUTTON) {if (state == GLUT\_UP) {int xPos = palette\_x\_pos;int yPos = 50;for (int i = 0; i < 5; i++) {// Check if the mouse is inside each colour rectif (checkInside(x, y, xPos, yPos, 50, 50)) {// select

the colourselectedIndex = i;selectedColour = colours[i];return true;}yPos += 49;}}}// block the event from continuing if the mouse is over the colour palettereturn checkInside(x, y, palette\_x\_pos, 50, 50, 245);}};

* + **Colour.h :**

/\*Colour.h\*/ #pragma once

// struct for storing colours struct Colour {float r, g, b;};

## Cover.h :

/\*Cover.hThis file handles the "Cover" - the dark semi-transparent layer to cover everything when displaying dialogues\*/#pragma onceclass Cover

{public:// Should the cover be displayedstatic bool show;/\*Displays the cover if necessary\*/static void Display(int window\_width, int window\_height) {if (show) {glBegin(GL\_QUADS);// semi transparent black colourglColor4f(0.0f, 0.0f, 0.0f, 0.85f);// with quad to fill the whole windowglVertex2f(0, window\_height);glVertex2f(0 + window\_width, window\_height);glVertex2f(0 + window\_width, 0);glVertex2f(0, 0);glEnd();}}/\*Handles button pressed events for the cover@param button - Mouse button pressed@param state - State of mouse event (down or up)@param x - The x coordinate of the mouse when pressed@param y - The y coordinate of the mouse when pressed@return Has the Top Menu Bar handled the event?\*/static bool Pressed(int button, int state, int x, int y) {// if showing cover block mouse presses for layers underneath coverreturn show;}/\*Handles mouse move events for cover@param x - The new x coordinate of the mouse@param y - The new y coordinate of the mouse@return Has the top menu bar handled the event?\*/static bool Hover(int x, int y) {// if showing cover block mouse move events for layers underneath coverreturn show;}};

## File Management .h :

/\*File Management.hImplements methods for reading and writing text to files\*/#pragma once#include <string>#include <vector>#include

<fstream>class FileManagement {public:/\*Returns a list of files which have previously been saved and can be opened@return Vector of strings

of file names which can be opened\*/static std::vector<std::string> GetList() {// get the list from the "saved\_files.txt" filestd::vector<std::string> output;std::string line = "";std::ifstream inFile;inFile.open("saved\_files.txt");if (inFile.is\_open()){// read the lines of "saved\_files.txt" and add the file names to the output vectorwhile (getline(inFile, line)){output.push\_back(line);}}if (output.size() == 0) {std::ofstream myfile;std::string fullName = "saved\_files.txt";myfile.open(fullName.c\_str());myfile << "";myfile.close();}return output;}/\*Checks if a file name corresponds to a file which can be opened@param fileName@return True if the given file can be opened\*/static bool CheckExists(std::string fileName) {// retrieve the list of openable filesstd::vector<std::string> files = GetList();// check the given file name is in the listfor (int i = 0; i

< files.size(); i++) {if (files[i] == fileName) {return true;}}return false;}/\*Get the text content of a file with the given name\*/static std::string ReadFile(std::string fileName) {// first make sure we can open this fileif (!CheckExists(fileName)) {return "";}// now open itstd::string contents = "";std::string line = "";std::ifstream inFile;inFile.open((fileName + ".dti").c\_str());if (inFile.is\_open()){// read the contents line by line and add them to the output stringwhile (getline(inFile, line)){if (contents != "")

{contents += "\n";}contents += line;}}return contents;}/\*Writes text to the file name provided@param fileName - The file name to write to@param content - The text to write\*/static void WriteFile(std::string fileName, std::string content) {if (!CheckExists(fileName)) {// update the "saved\_files.txt" adding this file name to the list if it's not already therestd::vector<std::string> files = GetList();files.push\_back(fileName);std::string fileString = "";for (int i = 0; i < files.size(); i++) {if (fileString != "") {fileString

+= "\n";}fileString += files[i];}std::ofstream fileListOutput;std::string name = "saved\_files.txt";fileListOutput.open(name.c\_str());fileListOutput << fileString;fileListOutput.close();}// write the content to the file itselfstd::ofstream myfile;std::string fullName = fileName + ".dti";myfile.open(fullName.c\_str());myfile << content;myfile.close();}/\*Deletes the file with the given name@param fileName - The file name of the file to delete\*/static void DeleteFile(std::string fileName) {if (CheckExists(fileName)) {// if the file exists, remove it from "saved\_files.txt"WriteFile(fileName, "");std::vector<std::string> list = GetList();std::string listText = "";for (int i = 0; i < list.size(); i++) {if (list[i] != fileName) {if

(listText != "") {listText += "\n";}listText += list[i];}}// write empty string "" to the file// this does not delete it but it is the best we can do to remain crossplatform using no additional librariesstd::ofstream fileListOutput;std::string name = "saved\_files.txt";fileListOutput.open(name.c\_str());fileListOutput << listText;fileListOutput.close();}}};

* + **Fonts.h :**

/\*Fonts.hImplements drawing of text using a stroke fontThis code is origionally from the labs\*/#pragma once#include <cstring>/\*Displays text on the screen using a stroke font@param text - The text to display\*/void draw\_text(const char\* text) {size\_t len = strlen(text);for (size\_t i = 0; i<len; i++)glutStrokeCharacter(GLUT\_STROKE\_ROMAN, text[i]);}/\*Displays text on the screen using a stroke font@param text - The text to display\*/void draw\_text(std::string text) {size\_t len = text.size();for (size\_t i = 0; i<len; i++)glutStrokeCharacter(GLUT\_STROKE\_ROMAN, text[i]);}/\*Displays text on the screen at a specific position using a stroke font@param text - The text to display@param x - X coordinate to draw the text@param y - Y coordinate to draw the text\*/void display\_text(const char\* text, float x, float y) {glPushMatrix();glColor3f(1.0f, 1.0f, 1.0f); // white colour// translate to the right place, and scale to a standard sizeglTranslatef(x, y, 0.0f);glScalef(0.22f, 0.22f, 1.0f);draw\_text(text);glPopMatrix();}/\*Displays text on the screen at a specific position using a stroke font@param text - The text to display@param x - X coordinate to draw the text@param y - Y coordinate to draw the text\*/void display\_text(std::string text, float x, float

1. {glPushMatrix();glColor3f(1.0f, 1.0f, 1.0f); // white colour// translate to the right place, and scale to a standard sizeglTranslatef(x, y, 0.0f);glScalef(0.22f, 0.22f, 1.0f);draw\_text(text);glPopMatrix();}/\*Displays text on the screen at a specific position using a stroke font with small font size@param text - The text to display@param x - X coordinate to draw the text@param y - Y coordinate to draw the text\*/void display\_text\_small(const char\* text, float x, float y)

{glPushMatrix();glColor3f(1.0f, 1.0f, 1.0f); // white colourglTranslatef(x, y, 0.0f);// translate to the right place, and scale to a small sizeglScalef(0.17f, 0.17f, 1.0f);draw\_text(text);glPopMatrix();}/\*Calculates the width of the text at standard font size@param text - The text to caluclate the width of@return The width of the text when displayed\*/float

get\_text\_width(const char\* text) {size\_t len = strlen(text);int total\_width = 0;for (size\_t i = 0; i<len; i++) {total\_width += glutStrokeWidth(GLUT\_STROKE\_ROMAN, text[i]);}return total\_width \* 0.22f;}/\*Calculates the width of the text at standard font size@param text - The text to caluclate the width of@return The width of the text when displayed\*/float get\_text\_width(std::string text) {size\_t len = text.size();int total\_width = 0;for (size\_t i = 0; i<len; i++)

{total\_width += glutStrokeWidth(GLUT\_STROKE\_ROMAN, text[i]);}return total\_width \* 0.22f;}/\*Calculates the width of the text at a small font size@param text - The text to caluclate the width of@return The width of the text when displayed\*/float get\_text\_width\_small(const char\* text) {size\_t len = strlen(text);int total\_width = 0;for (size\_t i = 0; i<len; i++) {total\_width += glutStrokeWidth(GLUT\_STROKE\_ROMAN, text[i]);}return total\_width \* 0.17f;}

* + **Main.cpp :**

/\*main.cppEntry point for 2D Drawing Tool\*/

#ifdef \_\_APPLE #include <GLUT/glut.h>#else#include

<GL/glut.h>#endif#include <stddef.h>#include <iostream>#include

<math.h>// For each static class we import from a header file, we need to redefine its variables here#include "Colour.h"Colour selectedColour

= { 1.0f, 0.0f, 0.0f };bool canvasAssigned = false;#include "File Management.h"#include "Button.h"#include "Fonts.h"#include "Pointer.h"#include "Cover.h"bool Cover::show = false;#include "Canvas.h"#include "Colour Palette.h"const Colour ColourPalette::colours[] = {{ 1.0f, 0.0f, 0.0f },// red{ 0.0f, 1.0f,

0.0f },// green{ 0.0f, 0.0f, 1.0f },// blue{ 0.0f, 0.0f, 0.0f },// black{ 1.0f, 1.0f, 1.0f }// white};int ColourPalette::selectedIndex = 0;int ColourPalette::palette\_x\_pos = 750;#include "Alert Dialogue.h"bool AlertDialogue::show = false;Button AlertDialogue::okButton;std::string AlertDialogue::message = "";#include "Yes No Dialogue.h"bool YesNoDialogue::show = false;Button YesNoDialogue::yesButton;Button YesNoDialogue::noButton;std::string YesNoDialogue::message = "";Callback YesNoDialogue::yesCallback;#include "Open File Dialogue.h"bool OpenFileDialogue::show = false;Button OpenFileDialogue::cancelButton;std::vector<Button> OpenFileDialogue::fileButtons;std::vector<Button> OpenFileDialogue::crossButtons;std::string OpenFileDialogue::deletionPendingFileName = "";#include "Save File Dialogue.h"bool SaveFileDialogue::show = false;Button

SaveFileDialogue::cancelButton;Button SaveFileDialogue::saveButton;std::string SaveFileDialogue::fileName = "";bool SaveFileDialogue::showTooLongText = false;#include "Top Menu Bar Callbacks.h"#include "Top Menu Bar.h"std::vector<Button> TopMenuBar::buttons;#include "Toolbar.h"int Toolbar::selectedButton = 0;Button Toolbar::penButton;Button Toolbar::eraseButton;Button Toolbar::moveButton;Button Toolbar::rotateButton;Button Toolbar::fillButton;Button Toolbar::rectButton;Button Toolbar::circleButton;#include "Tool\_Pen.h"bool Tool\_Pen::isMouseDown

= false;int Tool\_Pen::mouseLastX = 0;int Tool\_Pen::mouseLastY = 0;#include "Tool\_Fill.h"#include "Tool\_Rect.h"bool Tool\_Rect::isMouseDown = false;int Tool\_Rect::startMouseX = 0;int Tool\_Rect::startMouseY = 0;#include "Tool\_Circ.h"bool Tool\_Circ::isMouseDown = false;int Tool\_Circ::startMouseX = 0;int Tool\_Circ::startMouseY = 0;#include "Tool\_Move.h"int Tool\_Move::flickerFrameCount;bool Tool\_Move::flickerColor;bool Tool\_Move::isMouseDown;int Tool\_Move::startMouseX;int Tool\_Move::startMouseY;int Tool\_Move::endMouseX;int Tool\_Move::endMouseY;bool Tool\_Move::isDisplaying;#include"Tool\_Erase.h"int Tool\_Erase::flickerFrameCount;bool Tool\_Erase::flickerColor;bool Tool\_Erase::isMouseDown;int Tool\_Erase::startMouseX;int Tool\_Erase::startMouseY;int Tool\_Erase::endMouseX;int Tool\_Erase::endMouseY;bool Tool\_Erase::isDisplaying;#include "Tool\_Rotate.h"int Tool\_Rotate::flickerFrameCount;bool Tool\_Rotate::flickerColor;bool Tool\_Rotate::isMouseDown;int Tool\_Rotate::startMouseX;int Tool\_Rotate::startMouseY;int Tool\_Rotate::endMouseX;int Tool\_Rotate::endMouseY;bool Tool\_Rotate::isDisplaying;/\*OpenGL display function\*/void display(){glClear(GL\_COLOR\_BUFFER\_BIT);glPushMatrix();// Rescale to "pixel" scale - position (x, y) is x pixels along, y pixels up// Allows user to resize window without stretching UI elementsglScalef((double)(800) / (double)(glutGet(GLUT\_WINDOW\_WIDTH)), (double)(600) / (double)(glutGet(GLUT\_WINDOW\_HEIGHT)), 1.0f);// Draw the canvas and any overlays from tools in useif (canvasAssigned)

{currentCanvas.Draw(glutGet(GLUT\_WINDOW\_WIDTH), glutGet(GLUT\_WINDOW\_HEIGHT));ToolEvents::Display(glutGet(GLUT\_WINDOW\_W IDTH), glutGet(GLUT\_WINDOW\_HEIGHT));}// Draw the colour paletteColourPalette::Display(glutGet(GLUT\_WINDOW\_WIDTH), glutGet(GLUT\_WINDOW\_HEIGHT));// Draw the toolbar on left hand sideToolbar::Display(glutGet(GLUT\_WINDOW\_WIDTH), glutGet(GLUT\_WINDOW\_HEIGHT));// Draw the top menu bar buttons (new, open, save, etc)TopMenuBar::Display(glutGet(GLUT\_WINDOW\_WIDTH),

glutGet(GLUT\_WINDOW\_HEIGHT));// Draw the dark semi-transparent cover if necessaryCover::Display(glutGet(GLUT\_WINDOW\_WIDTH), glutGet(GLUT\_WINDOW\_HEIGHT));// Draw File Dialgues if necessaryOpenFileDialogue::Display(glutGet(GLUT\_WINDOW\_WIDTH), glutGet(GLUT\_WINDOW\_HEIGHT));SaveFileDialogue::Display(glutGet(GLUT\_WI NDOW\_WIDTH),

glutGet(GLUT\_WINDOW\_HEIGHT));AlertDialogue::Display(glutGet(GLUT\_WINDO W\_WIDTH),

glutGet(GLUT\_WINDOW\_HEIGHT));YesNoDialogue::Display(glutGet(GLUT\_WINDO W\_WIDTH), glutGet(GLUT\_WINDOW\_HEIGHT));// Draw mouse pointer last (so it appears above everything else)Display\_Pointer();glPopMatrix();glutSwapBuffers();}/\*Handles mouse click events@param button - Mouse button pressed@param state - State of mouse event (down or up)@param x - The x coordinate of the mouse when pressed@param y - The y coordinate of the mouse when pressed\*/void mouse\_click(int button, int state, int x, int y){// If we are currently using a tool which wants all mouse events first, give it the mouse events// Otherwise pass the mouse event onto each UI element in turn until it gets handled, in the order of depthif ((!canvasAssigned) || (!ToolEvents::BlockMousePress(button, state, x, y))) {// First pass the event onto the Dialoguesif (AlertDialogue::Pressed(button, state, x, y)) {return;}if (YesNoDialogue::Pressed(button, state, x, y)) {return;}if (OpenFileDialogue::Pressed(button, state, x, y)) {return;}if (SaveFileDialogue::Pressed(button, state, x, y)) {return;}// If not handled, maybe the Cover will block itif (Cover::Pressed(button, state, x, y)) {return;}// If not handled pass it onto buttons/toolbarsif (TopMenuBar::Pressed(button, state, x, y))

{return;}if (Toolbar::Pressed(button, state, x, y)) {return;}if (ColourPalette::Pressed(button, state, x, y)) {return;}}// If it hasn't been handled, pass it on to the selected tool if we have a canvasif (canvasAssigned) {if (ToolEvents::Pressed(button, state, x, y)) {return;}}}/\*Handles all mouse movement events@param x - The new x coordinate of the mouse@param y - The new y coordinate of the mouse\*/void mouse\_motion(int x, int y){// Remember new cursor position for the PointercursorX = x;cursorY = y;// Pass the mouse move event onto Dialoguesif (AlertDialogue::Hover(x, y)) {return;}if (YesNoDialogue::Hover(x, y)) {return;}if (OpenFileDialogue::Hover(x, y)) {return;}if (SaveFileDialogue::Hover(x, y)) {return;}// If not handled, maybe the Cover will block itif (Cover::Hover(x, y))

{return;}// If not handled pass it onto buttons/toolbarsif (Toolbar::Hover(x, y)) {return;}if (TopMenuBar::Hover(x, y))

{return;}// If it hasn't been handled, pass it on to the selected tool

if we have a canvasif (canvasAssigned) {if (ToolEvents::Hover(x, y))

{return;}}}/\*Handles standard keyboard events@param key - The key that was pressed@param x - The x position of the mouse@param y - The y position of the mouse\*/void keyboard(unsigned char key, int x, int y){// Save File Dialogue should steal the input events if active// But Yes/No Dialogue can appear above it so should block inputs if it is being displayedif (YesNoDialogue::show) {return;}// Now pass on to Save File Dialogueif (SaveFileDialogue::KeyboardPressed(key, x, y))

{return;}// otherwise check for quitting or zoomswitch (key){case 'q': exit(1); break;case 's':// zoom inif (canvasAssigned)

{currentCanvas.zoom++;}break;case 'S':// zoom outif (canvasAssigned)

{if (currentCanvas.zoom > 1) {currentCanvas.zoom--;}}break;}}/\*Handles special keyboard events@param key - The key that was pressed@param x - The x position of the mouse@param y - The y position of the mouse\*/void special(int key, int x, int y){if (!Cover::show) {// Pass onto selected tool (Move tool uses arrow keys)if (canvasAssigned) {if (ToolEvents::SpecialKey(key, x, y)) {return;}}// If not handled check arrow keys for panning cameraswitch (key){case GLUT\_KEY\_LEFT: if (canvasAssigned) { currentCanvas.xOffset -= 6; } break;case GLUT\_KEY\_RIGHT: if (canvasAssigned) { currentCanvas.xOffset += 6; } break;case GLUT\_KEY\_UP: if (canvasAssigned) { currentCanvas.yOffset -= 6; } break;case GLUT\_KEY\_DOWN: if (canvasAssigned) { currentCanvas.yOffset += 6; } break;}}}/\*Called to intialize all classes\*/void init(){// Initialize classes where neededTopMenuBar::Init();Toolbar::Init();OpenFileDialogue::Init();Save FileDialogue::Init();AlertDialogue::Init();YesNoDialogue::Init();// Enable transparency (e.g. black semi-transparent cover over screen appears with dialogues)glEnable(GL\_BLEND);glBlendFunc(GL\_SRC\_ALPHA, GL\_ONE\_MINUS\_SRC\_ALPHA);// Set initial scale for window coordinatesgluOrtho2D(0, glutGet(GLUT\_WINDOW\_WIDTH), 0, glutGet(GLUT\_WINDOW\_HEIGHT));glMatrixMode(GL\_PROJECTION);glLoadIdentit y();glClearColor(0.0f, 0.0f, 0.0f, 0.0f);}/\*idle function\*/void idle()

{// force a redraw// so we get through as many frames as possible (needed for things like blinking of Move tool)glutPostRedisplay();}/\*Main entry point of application\*/int main(int argc, char\* argv[]){// create window with title and fixed start sizeglutInit(&argc, argv);glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGBA);glutInitWindowSize(800, 600);glutCreateWindow("2D Drawing Tool");// define the display functionglutDisplayFunc(display);// handlers for keyboard inputglutKeyboardFunc(keyboard);glutSpecialFunc(special);// define mouse pressed event handlerglutMouseFunc(mouse\_click);// define mouse movement event handlersglutPassiveMotionFunc(mouse\_motion); // << when

mouse is not being pressedglutMotionFunc(mouse\_motion); // << when mouse is being pressed// define idle and initglutIdleFunc(idle);// initialize everythinginit();// start first render cycleglutMainLoop();return 0;}

* + **Open File Dialogue.h :**

/\*Open File Dialogue.hThis adds the Open File Dialogue listing the list of the users saved files allowing them to be opened or deleted\*/#pragma once#include <fstream>#include <vector>#include "Canvas.h"class OpenFileDialogue {public:static bool show;static Button cancelButton;// The buttons for opening a file (at most 8)static std::vector<Button> fileButtons;// The buttons for deleting a file (at most 8)static std::vector<Button> crossButtons;// If we are deleting a file, this stores the file to delete while we are waiting for the user to confirm their actionstatic std::string deletionPendingFileName;/\*Opens the Open File Dialogue\*/static void Show() {// Get the list of files to displaystd::vector<std::string> files = FileManagement::GetList();for (int i = 0; i < fileButtons.size(); i++) {if (i < files.size()) {// display the first

10 filesfileButtons[i].text = files[i].c\_str();fileButtons[i].Show();crossButtons[i].Show();}else

{// If more files than buttons, hide the unneeded buttonsfileButtons[i].Hide();crossButtons[i].Hide();}}Cover::show = true;show = true;cancelButton.Show();}/\*Hides the Open File Dialogue\*/static void Hide() {Cover::show = false;show = false;cancelButton.Hide();// Hide all the buttonsfor (int i = 0; i < fileButtons.size(); i++)

{fileButtons[i].Hide();crossButtons[i].Hide();}}/\*Callback invoked when the Cancel button is pressed\*/static void CancelPressed(Button button) {Hide();}/\*Callback invoked when a file is selected\*/static void SelectPressed(Button button) {std::string content = FileManagement::ReadFile(button.text);Canvas newCanvas = Canvas::Deserialize(content);currentCanvas = newCanvas;canvasAssigned

= true;currentCanvas.fileName = button.text;Hide();}/\*Callback invoked when the 'X' button is pressed for a file listed\*/static void DeletePressed(Button button) {// get the fileName from this buttonstd::string fileName = "";for (int i = 0; i < fileButtons.size(); i++) {if (crossButtons[i].y\_pos == button.y\_pos)

{fileName = fileButtons[i].text;}}// Ask the user if they are sure they want to delete the fileYesNoDialogue::Show("Are you sure you want to delete " + fileName + ".dti?", DeleteConfirmedCallback);deletionPendingFileName = fileName; // <<

remember the file name so we can access it in the Callback function below}/\*Callback invoked when the user confirms they are sure they want to delete a file\*/static void DeleteConfirmedCallback(Button button) {// Delete the file and display the updated list of filesFileManagement::DeleteFile(deletionPendingFileName);Show();}/\*Ini tializes the Open File Dialogue\*/static void Init() {show = false;// create the cancel buttoncancelButton = Button::Create(0, 500, 100, 40, (char \*)"Cancel", CancelPressed, false);// create the 8 buttons and delete buttonsfor (int i = 0; i < 8; i++) {Button newButton = Button::Create(0, 160 + (i \* 40), 300, 40, (char \*)"asdf", SelectPressed, false);fileButtons.push\_back(newButton);Button newCrossButton = Button::Create(0, 160 + (i \* 40), 30, 40, (char

\*)"X", DeletePressed, false);crossButtons.push\_back(newCrossButton);}}/\*Displays the open file dialogue@param window\_width - the width of the window@param window\_height - the height of the window\*/static void Display(int window\_width, int window\_height) {if (show) {// Display the 2 lines of centered textchar\* text = (char \*)"Select a file below to open it:";display\_text(text, (window\_width / 2) - (get\_text\_width(text) / 2), window\_height - 100);char\* textTwo = (char \*)"(these are the .dti files in the application folder)";display\_text\_small(textTwo, (window\_width / 2) - (get\_text\_width\_small(textTwo) / 2), window\_height - 140);// Display the centered cancel buttoncancelButton.HorizontallyCenter(window\_width);cancelButton.Displ ay(window\_width, window\_height);// Display the list of open and delete buttonsfor (int i = 0; i < fileButtons.size(); i++)

{fileButtons[i].HorizontallyCenter(window\_width);fileButtons[i].Displa y(window\_width, window\_height);crossButtons[i].x\_pos = fileButtons[i].x\_pos + fileButtons[i].width;crossButtons[i].Display(window\_width, window\_height);}}}/\*Handles button pressed events for Open File Dialogue@param button - Mouse button pressed@param state - State of mouse event (down or up)@param x - The x coordinate of the mouse when pressed@param y - The y coordinate of the mouse when pressed@return Has the Open File Dialogue handled the event?\*/static bool Pressed(int button, int state, int x, int y) {if (show) {// Go through all the buttons and check if they handle the eventif (cancelButton.Pressed(button, state, x, y)) {return true;}for (int i = 0; i < fileButtons.size(); i++) {if (fileButtons[i].Pressed(button, state, x, y)) {return true;}if (crossButtons[i].Pressed(button, state, x, y)) {return true;}}}return false;}/\*Detects if the Open File Dialogue should be in the "hover" state@param x - Mouse position x coordinate@param y - Mouse position y coordinate@return True if the

button handles the hovering event\*/static bool Hover(int x, int y)

{bool output = false;if (show) {// Go through all buttons and check if they handle the eventif (cancelButton.Hover(x, y)) {output = true;}for (int i = 0; i < fileButtons.size(); i++) {if (fileButtons[i].Hover(x, y)) {output = true;}if (crossButtons[i].Hover(x, y)) {output = true;}}}return output;}};

* **Pointer.h :**

/\*Pointer.hDisplays a small white cross at the mouse position\*/ #pragma onceint cursorX = 0;

int cursorY = 0;/\*Draws a white cross on the screen on a (-1, 1) scale\*/void Draw\_Cross() {glBegin(GL\_LINE\_LOOP);static float pointOne[2] = { -1.0f, 0.0f };static float pointTwo[2] = { 1.0f, 0.0f

};glVertex2fv(pointOne);glVertex2fv(pointTwo);glEnd();glBegin(GL\_LINE\_ LOOP);static float pointThree[2] = { 0.0f, 1.0f };static float pointFour[2] = { 0.0f, -1.0f

};glVertex2fv(pointThree);glVertex2fv(pointFour);glEnd();}/\*Draws a small white cross on the screen\*/void Display\_Pointer()

{glMatrixMode(GL\_MODELVIEW);glColor3f(1.0f, 1.0f, 1.0f);glLineWidth(2.0f);glPushMatrix();glTranslatef(cursorX, glutGet(GLUT\_WINDOW\_HEIGHT) - cursorY, 1.0f);glScalef(20.0f, 20.0f, 1.0f);Draw\_Cross();glPopMatrix();}

* **Save File Dialogue.h :**

/\*Save File Dialogue.hThis adds the Save File Dialogue where the user can type the name of the file and save it\*/#pragma once#include "Canvas.h"#include <iostream>#include <fstream>#include <string>class SaveFileDialogue {public:static bool show;// cancel and save buttons to displaystatic Button cancelButton;static Button saveButton;// the file name which has been typed instatic std::string fileName;// has user tried to enter a file name over 40 charsstatic bool showTooLongText;/\*resets the save file dialogue\*/static void Reset() {fileName = "";saveButton.Hide();showTooLongText = false;}/\*start showing the save file dialogue\*/static void Show() {Cover::show = true;show = true;cancelButton.Show();}/\*hide the save file dialogue\*/static void Hide()

{Cover::show = false;show = false;cancelButton.Hide();}/\*Callback invoked when the cancel button is pressed\*/static void CancelPressed(Button button)

{Hide();}/\*Callback invoked when the save button is pressed\*/static void SavePressed(Button button) {// check the file name is a valid lengthif

((fileName.size() > 0) && (fileName.size() < 40)) {// If the file already exists, ask if the user wants to overwrite itif (FileManagement::CheckExists(fileName)) {YesNoDialogue::Show("File already exists. Overwrite?", SaveOverwriteCallback);return;}// File doesn't already exists so save itFileManagement::WriteFile(fileName, currentCanvas.Serialize());currentCanvas.fileName = fileName;Hide();}}/\*Callback when user confirms they want to save, overwriting a file\*/static void SaveOverwriteCallback(Button button) {// save the fileFileManagement::WriteFile(fileName, currentCanvas.Serialize());currentCanvas.fileName = fileName;Hide();}/\*Initializes the open file dialogue\*/static void Init()

{show = false;cancelButton = Button::Create(0, 200, 100, 40, (char

\*)"Cancel", CancelPressed, false);saveButton = Button::Create(0, 200, 75, 40, (char \*)"Save", SavePressed, false);}/\*Displays the save file dialogue@param window\_width - the width of the window@param window\_height - the height of the window\*/static void Display(int window\_width, int window\_height) {if (show) {// display text at topchar\* text = (char \*)"Type in a file name to save it:";display\_text(text, (window\_width / 2) - (get\_text\_width(text) / 2), window\_height - 100);display\_text(fileName, (window\_width / 2) - (get\_text\_width(fileName) / 2), window\_height - 155);// If the user is trying to enter a file name too long, display error messageif (showTooLongText)

{char\* text2 = (char \*)"40 character file name limit";display\_text\_small(text2, (window\_width / 2) - (get\_text\_width\_small(text2) / 2), window\_height - 184);}// display the cancel and save buttons (centered then offset from center)cancelButton.HorizontallyCenter(window\_width);cancelButton.x\_pos -= 55;saveButton.HorizontallyCenter(window\_width);saveButton.x\_pos += 65;cancelButton.Display(window\_width, window\_height);saveButton.Display(window\_width, window\_height);}}/\*Handles button pressed events for Save File Dialogue@param button - Mouse button pressed@param state - State of mouse event (down or up)@param x - The x coordinate of the mouse when pressed@param y - The y coordinate of the mouse when pressed@return Has the Save File Dialogue handled the event?\*/static bool Pressed(int button, int state, int x, int y) {if (show) {// pass the event on to cancel and save buttonsif (cancelButton.Pressed(button, state, x, y)) {return true;}if (saveButton.Pressed(button, state, x, y)) {return true;}}return false;}/\*Detects if the Save File Dialogue should be in the "hover" state@param x - Mouse position x coordinate@param y - Mouse position y coordinate@return True if the dialogue handles the hovering event\*/static bool Hover(int x, int y) {bool output = false;if (show) {// pass the event on to cancel and save buttonsif (cancelButton.Hover(x, y)) {output = true;}if (saveButton.Hover(x, y)) {output = true;}}return output;}/\*Handles keyboard pressed events@param key - The key that was pressed@param x - The x position of the mouse@param y - The y position of the mosue@return True if the event

was handled\*/static bool KeyboardPressed(unsigned char key, int x, int y) {if (show) {// Limit the characters allowed in file names (e.g. ? is not allowed in file names on windows)std::string allowedChars = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789-

\_+=[]{}!�$^&()`�,;@' ";if (allowedChars.find(key) != std::string::npos)

{saveButton.Show();if (fileName.size() < 40) {// add the character to file namefileName += key;}else {showTooLongText = true;}return true;}if (key == '\b') {// backspace key was pressedif (fileName.size() > 0) {// remove last character form file namefileName = fileName.substr(0, fileName.size() - 1);}if (fileName.size() == 0) {saveButton.Hide();}showTooLongText = false;return true;}}return false;}};

## Tool\_Circle.h :

/\*Tool\_Circ.hImplements the Circle drawing tool\*/#pragma once/\*Handles mouse press events passed onto the Circ tool@param button - Mouse button pressed@param state - State of mouse event (down or up)@param x - The x coordinate of the mouse when pressed@param y - The y coordinate of the mouse when pressed@return Has the tool handled the event?\*/// Function to put pixels// at subsequence points//mid point elipse drawing algorithmvoid midptellipse(int rx, int ry,int xc, int yc){float dx, dy, d1, d2, x, y;x = 0;y = ry;// Initial decision parameter of region 1 d1 = (ry \* ry) - (rx \* rx

\* ry) +(0.25 \* rx \* rx);dx = 2 \* ry \* ry \* x;dy = 2 \* rx \* rx \* y;// For region 1 while (dx < dy){// Print points based on 4-way symmetry currentCanvas.SetPixelColour(xc + x, yc + y, selectedColour);currentCanvas.SetPixelColour(xc - x, yc + y, selectedColour);currentCanvas.SetPixelColour(xc + x, yc - y, selectedColour);currentCanvas.SetPixelColour(xc - x, yc - y, selectedColour);// Checking and updating value of // decision parameter based on algorithm if (d1 < 0){x++;dx = dx + (2 \* ry \* ry);d1 = d1 + dx + (ry \* ry);}else{x++;y--;dx = dx + (2 \* ry \* ry);dy = dy - (2 \* rx \* rx);d1 = d1 + dx - dy + (ry \* ry);}}// Decision parameter of region 2 d2 = ((ry \* ry) \* ((x

+ 0.5) \* (x + 0.5))) +((rx \* rx) \* ((y - 1) \* (y - 1))) -(rx \* rx \* ry \* ry);// Plotting points of region 2 while (y >= 0){// Print points based on 4- way symmetry currentCanvas.SetPixelColour(xc + x, yc + y, selectedColour);currentCanvas.SetPixelColour(xc - x, yc + y, selectedColour);currentCanvas.SetPixelColour(xc + x, yc - y, selectedColour);currentCanvas.SetPixelColour(xc - x, yc - y, selectedColour);// Checking and updating parameter // value based on algorithm if (d2 > 0){y--;dy = dy - (2 \* rx \* rx);d2 = d2 + (rx \* rx) - dy;}else{y--;x++;dx = dx + (2 \* ry \* ry);dy = dy - (2 \* rx \* rx);d2 = d2 + dx

* dy + (rx \* rx);}}}//Bresenhams Line drawing algovoid drawCircle(int xc, int yc, int x, int y){currentCanvas.SetPixelColour(xc+x, yc+y, selectedColour);currentCanvas.SetPixelColour(xc-x, yc+y, selectedColour);currentCanvas.SetPixelColour(xc+x, yc-y, selectedColour);currentCanvas.SetPixelColour(xc-x, yc-y, selectedColour);currentCanvas.SetPixelColour(xc+y, yc+x, selectedColour);currentCanvas.SetPixelColour(xc-y, yc+x, selectedColour);currentCanvas.SetPixelColour(xc+y, yc-x, selectedColour);currentCanvas.SetPixelColour(xc-y, yc-x, selectedColour);}// Function for circle-generation// using Bresenham's algorithmvoid circleBres(int xc, int yc, int r){int x = 0, y = r;int d = 3 - 2 \* r;drawCircle(xc, yc, x, y);while (y >= x){// for each pixel we will// draw all eight pixelsx++;// check for decision parameter// and correspondingly// update d, x, yif (d > 0){y--;d = d + 4 \* (x - y) + 10;}elsed = d + 4 \* x + 6;drawCircle(xc, yc, x, y);}}bool Tool\_Circ::Pressed(int button, int state, int x, int y) {if (currentCanvas.checkInside(x, y)) {// convert mouse position into canvas coordinatesint cx = (x - currentCanvas.xOffset) / currentCanvas.zoom;int cy = (y - currentCanvas.yOffset) / currentCanvas.zoom;// remember the start mouse position if this is start of a dragif ((button == GLUT\_LEFT\_BUTTON) && (state == GLUT\_DOWN) && !isMouseDown)

{isMouseDown = true;startMouseX = cx;startMouseY = cy;return true;}// draw the circle if this is the end of a dragif ((button == GLUT\_LEFT\_BUTTON) && (state == GLUT\_UP) && isMouseDown) {if ((startMouseX == cx) && (startMouseY

== cy)) {// if the mouse hasn't moved, just colour one pixelcurrentCanvas.SetPixelColour(cx, cy, selectedColour);}else {// get the rect coordinates to put the circle inint minX = std::min(cx, startMouseX);int maxX = std::max(cx, startMouseX);int minY = std::min(cy, startMouseY);int maxY = std::max(cy, startMouseY);// work out the radii and center coordsint radH = (maxX - minX) / 2;int radV = (maxY - minY) / 2;int centX = (maxX + minX) / 2;int centY = (maxY + minY) / 2;// call Bresenhams algoritmif (radH

== radV) {circleBres(centX, centY, radH);}else {midptellipse(radH, radV,centX, centY);}}isMouseDown = false;return true;}}return false;}/\*Should this tool take priority for receiving mouse events@param button - Mouse button pressed@param state - State of mouse event (down or up)@param x - The x coordinate of the mouse when pressed@param y - The y coordinate of the mouse when pressed@return Should this tool take priority for receiving mouse events\*/bool Tool\_Circ::BlockMousePress(int button, int state, int x, int y)

{// if during a drag, this tool should take the mouse events firstif (isMouseDown) {if (currentCanvas.checkInside(x, y)) {return true;}}isMouseDown = false;return false;}

## Tool\_Erase.h :

/\*Tool\_Erase.hImplements the Move tool\*/#pragma once/\*Initializes tool when selected\*/void Tool\_Erase::Start() {isDisplaying = false;isMouseDown = false;flickerFrameCount = 20;flickerColor = false;}/\*Disables tool when another is selected\*/void Tool\_Erase::End() {isDisplaying = false;isMouseDown

= false;}/\*Displays rect around selected pixels@param window\_width - the width of the window@param window\_height - the height of the window\*/void Tool\_Erase::Display(int window\_width, int window\_height) {if ((!Tool\_Erase::isDisplaying) || isMouseDown) {// display white cover over canvas while user needs to draw rectangle to select pixelsglBegin(GL\_QUADS);glColor4f(1.0f, 1.0f, 1.0f, 0.7f);glVertex2f(currentCanvas.xOffset, window\_height - currentCanvas.yOffset);glVertex2f(currentCanvas.xOffset + (currentCanvas.width \* currentCanvas.zoom), window\_height - currentCanvas.yOffset);glVertex2f(currentCanvas.xOffset + (currentCanvas.width \* currentCanvas.zoom), window\_height - currentCanvas.yOffset - (currentCanvas.height \* currentCanvas.zoom));glVertex2f(currentCanvas.xOffset, window\_height - currentCanvas.yOffset - (currentCanvas.height \* currentCanvas.zoom));glEnd();}else {// display the blue/red flickering rectangle around selected pixelsglBegin(GL\_LINES);if (flickerColor)

{glColor4f(0.4f, 0.4f, 1.0f, 1.0f);}else {glColor4f(1.0f, 0.4f, 0.4f, 1.0f);}glVertex2f(currentCanvas.xOffset + (startMouseX \* currentCanvas.zoom), window\_height - currentCanvas.yOffset - (startMouseY \* currentCanvas.zoom));glVertex2f(currentCanvas.xOffset + (endMouseX \* currentCanvas.zoom), window\_height - currentCanvas.yOffset - (startMouseY \* currentCanvas.zoom));glVertex2f(currentCanvas.xOffset + (endMouseX \* currentCanvas.zoom), window\_height - currentCanvas.yOffset - (startMouseY \* currentCanvas.zoom));glVertex2f(currentCanvas.xOffset + (endMouseX \* currentCanvas.zoom), window\_height - currentCanvas.yOffset - (endMouseY \* currentCanvas.zoom));glVertex2f(currentCanvas.xOffset + (endMouseX \* currentCanvas.zoom), window\_height - currentCanvas.yOffset - (endMouseY \* currentCanvas.zoom));glVertex2f(currentCanvas.xOffset + (startMouseX \* currentCanvas.zoom), window\_height - currentCanvas.yOffset - (endMouseY \* currentCanvas.zoom));glVertex2f(currentCanvas.xOffset + (startMouseX \* currentCanvas.zoom), window\_height - currentCanvas.yOffset - (endMouseY \* currentCanvas.zoom));glVertex2f(currentCanvas.xOffset + (startMouseX \* currentCanvas.zoom), window\_height - currentCanvas.yOffset - (startMouseY \* currentCanvas.zoom));glEnd();// change the flickering colour every 20 framesflickerFrameCount--;if (flickerFrameCount <= 0) {flickerFrameCount = 20;flickerColor = !flickerColor;}}}/\*Handles mouse press events passed onto

the Move tool@param button - Mouse button pressed@param state - State of mouse event (down or up)@param x - The x coordinate of the mouse when pressed@param y - The y coordinate of the mouse when pressed@return Has the tool handled the event?\*/bool Tool\_Erase::Pressed(int button, int state, int x, int y) {if (currentCanvas.checkInside(x, y)) {// get mouse position in canvas coordinatesint cx = (x - currentCanvas.xOffset) / currentCanvas.zoom;int cy = (y - currentCanvas.yOffset) / currentCanvas.zoom;// remember the drag start position (mouse down) and end position (mouse up)if ((button == GLUT\_LEFT\_BUTTON) && (state == GLUT\_DOWN) && !isMouseDown) {isMouseDown = true;startMouseX = cx;startMouseY = cy;return true;}if ((button == GLUT\_LEFT\_BUTTON) && (state == GLUT\_UP) && isMouseDown)

{isMouseDown = false;endMouseX = cx;endMouseY = cy;isDisplaying = true;return true;}}return false;}/\*Should this tool take priority for receiving mouse events@param button - Mouse button pressed@param state - State of mouse event (down or up)@param x - The x coordinate of the mouse when pressed@param y - The y coordinate of the mouse when pressed@return Should this tool take priority for receiving mouse events\*/bool Tool\_Erase::BlockMousePress(int button, int state, int x, int y) {if (isMouseDown) {// take priority with mouse events when tool is selected and mouse is over canvasif (currentCanvas.checkInside(x, y)) {return true;}}isMouseDown = false;return false;}/\*Handles special key events (arrow keys) for the tool@param key - the key that was pressed@param x - x position of the mouse@param y - y position of the mouse@return Has the tool handled the event?\*/bool Tool\_Erase::SpecialKey(int key, int x, int y) {if (isDisplaying) {// get rect coordinatesint minX = std::min(startMouseX, endMouseX);int maxX = std::max(startMouseX, endMouseX);int minY = std::min(startMouseY, endMouseY);int maxY = std::max(startMouseY, endMouseY);switch (key){case GLUT\_KEY\_LEFT:// left key pressedif (minX > 0) {Colour white = { 1.0f, 1.0f, 1.0f };for (int x = 0; x < maxX - minX; x++) {for (int y = 0; y < maxY - minY; y++) {// move all pixels left//currentCanvas.SetPixelColour(minX + x - 1, minY + y, currentCanvas.GetPixelColour(minX + x, minY + y));currentCanvas.SetPixelColour(minX + x, minY + y, white);}}// rectangle has movedstartMouseX--;endMouseX--;return true;}break;// do the same for other arrow keyscase GLUT\_KEY\_RIGHT:if (maxX < currentCanvas.width) {Colour white = { 1.0f, 1.0f, 1.0f };for (int x = maxX - minX - 1; x >= 0; x--) {for (int y = 0; y < maxY - minY; y++) {//currentCanvas.SetPixelColour(minX + x + 1, minY + y, currentCanvas.GetPixelColour(minX + x, minY + y));currentCanvas.SetPixelColour(minX + x, minY + y, white);}}startMouseX++;endMouseX++;return true;}break;case GLUT\_KEY\_UP:if (minY > 0) {Colour white = { 1.0f, 1.0f, 1.0f };for (int x = 0; x < maxX - minX; x++) {for (int y = 0; y < maxY - minY; y++)

{currentCanvas.SetPixelColour(minX + x, minY + y, white);}}startMouseY--

;endMouseY--;return true;}break;case GLUT\_KEY\_DOWN:if (maxY < currentCanvas.height) {Colour white = { 1.0f, 1.0f, 1.0f };for (int x = 0; x

< maxX - minX; x++) {for (int y = maxY - minY - 1; y >= 0; y--)

{//currentCanvas.SetPixelColour(minX + x, minY + y + 1, currentCanvas.GetPixelColour(minX + x, minY + y));currentCanvas.SetPixelColour(minX + x, minY + y, white);}}startMouseY++;endMouseY++;return true;}break;}return true;}return false;}

## Tool\_Fill.h :

/\*Tool\_Fill.hImplements the Fill tool\*/#pragma once#include <vector>// struct for storing coordinatesstruct Tuple {int x;int y;};/\*Fill algorithm from a position@param startColour - The colour we should be replacing@param cx - x coordinate of pixel to fill from@param cy - y coordinate of pixel to fill from\*/void Tool\_Fill::Fill(Colour startColour, int cx, int cy) {// algorithm expands from point filling an area// vec stores outer pixels for the next iterationstd::vector<Tuple> vec;std::vector<Tuple> newvec;Tuple startCoord =

{ cx, cy };vec.push\_back(startCoord);while (vec.size() > 0) {for (int i = 0; i < vec.size(); i++) {// change pixel colour for pixels on outside of the fill areaColour colourAtPixel = currentCanvas.GetPixelColour(vec[i].x, vec[i].y);if ((colourAtPixel.r == startColour.r) && (colourAtPixel.g == startColour.g) && (colourAtPixel.b == startColour.b))

{currentCanvas.SetPixelColour(vec[i].x, vec[i].y, selectedColour);// add neighbours to the outside vector for the next iterationif (vec[i].x < currentCanvas.width - 1) {Tuple newCoord = { vec[i].x + 1, vec[i].y

};newvec.push\_back(newCoord);}if (vec[i].x > 0) {Tuple newCoord = { vec[i].x

* 1, vec[i].y };newvec.push\_back(newCoord);}if (vec[i].y < currentCanvas.height - 1) {Tuple newCoord = { vec[i].x, vec[i].y + 1

};newvec.push\_back(newCoord);}if (vec[i].y > 0) {Tuple newCoord = { vec[i].x, vec[i].y - 1 };newvec.push\_back(newCoord);}}}vec.empty();vec = newvec;std::vector<Tuple> b;newvec = b;}}/\*Handles mouse press events passed onto the Fill tool@param button - Mouse button pressed@param state - State of mouse event (down or up)@param x - The x coordinate of the mouse when pressed@param y - The y coordinate of the mouse when pressed@return Has the tool handled the event?\*/bool Tool\_Fill::Pressed(int button, int state, int x, int y) {if ((button == GLUT\_LEFT\_BUTTON) && (state == GLUT\_DOWN)) {int canvasCoordX = (x - currentCanvas.xOffset) / currentCanvas.zoom;int canvasCoordY = (y - currentCanvas.yOffset) / currentCanvas.zoom;Colour colourWherePressed = currentCanvas.GetPixelColour(canvasCoordX, canvasCoordY);if ((colourWherePressed.r != selectedColour.r) || (colourWherePressed.g != selectedColour.g) || (colourWherePressed.b != selectedColour.b)) {Tool\_Fill::Fill(colourWherePressed, canvasCoordX, canvasCoordY);return true;}}return false;}

## Tool\_Move.h :

/\*Tool\_Move.hImplements the Move tool\*/#pragma oncestatic std::map

<std::string, struct Colour> pix\_color;/\*Initializes tool when selected\*/void Tool\_Move::Start() {isDisplaying = false;isMouseDown = false;flickerFrameCount = 20;flickerColor = false;}/\*Disables tool when another is selected\*/void Tool\_Move::End() {isDisplaying = false;isMouseDown

= false;}/\*Displays rect around selected pixels@param window\_width - the width of the window@param window\_height - the height of the window\*/void Tool\_Move::Display(int window\_width, int window\_height) {if ((!Tool\_Move::isDisplaying) || isMouseDown) {// display white cover over canvas while user needs to draw rectangle to select pixelsglBegin(GL\_QUADS);glColor4f(1.0f, 1.0f, 1.0f, 0.7f);glVertex2f(currentCanvas.xOffset, window\_height - currentCanvas.yOffset);glVertex2f(currentCanvas.xOffset + (currentCanvas.width\*currentCanvas.zoom), window\_height - currentCanvas.yOffset);glVertex2f(currentCanvas.xOffset + (currentCanvas.width\*currentCanvas.zoom), window\_height - currentCanvas.yOffset - (currentCanvas.height\*currentCanvas.zoom));glVertex2f(currentCanvas.xOffset, window\_height - currentCanvas.yOffset - (currentCanvas.height\*currentCanvas.zoom));glEnd();}else {// display the blue/red flickering rectangle around selected pixelsglBegin(GL\_LINES);if (flickerColor) {glColor4f(0.4f, 0.4f, 1.0f, 1.0f);}else {glColor4f(1.0f, 0.4f, 0.4f, 1.0f);}glVertex2f(currentCanvas.xOffset + (startMouseX\*currentCanvas.zoom), window\_height - currentCanvas.yOffset - (startMouseY\*currentCanvas.zoom));glVertex2f(currentCanvas.xOffset + (endMouseX\*currentCanvas.zoom), window\_height - currentCanvas.yOffset - (startMouseY\*currentCanvas.zoom));glVertex2f(currentCanvas.xOffset + (endMouseX\*currentCanvas.zoom), window\_height - currentCanvas.yOffset - (startMouseY\*currentCanvas.zoom));glVertex2f(currentCanvas.xOffset + (endMouseX\*currentCanvas.zoom), window\_height - currentCanvas.yOffset - (endMouseY\*currentCanvas.zoom));glVertex2f(currentCanvas.xOffset + (endMouseX\*currentCanvas.zoom), window\_height - currentCanvas.yOffset - (endMouseY\*currentCanvas.zoom));glVertex2f(currentCanvas.xOffset + (startMouseX\*currentCanvas.zoom), window\_height - currentCanvas.yOffset - (endMouseY\*currentCanvas.zoom));glVertex2f(currentCanvas.xOffset + (startMouseX\*currentCanvas.zoom), window\_height - currentCanvas.yOffset - (endMouseY\*currentCanvas.zoom));glVertex2f(currentCanvas.xOffset + (startMouseX\*currentCanvas.zoom), window\_height - currentCanvas.yOffset - (startMouseY\*currentCanvas.zoom));glEnd();// change the flickering colour

every 20 framesflickerFrameCount--;if (flickerFrameCount <= 0)

{flickerFrameCount = 20;flickerColor = !flickerColor;}}}/\*Handles mouse press events passed onto the Move tool@param button - Mouse button pressed@param state - State of mouse event (down or up)@param x - The x coordinate of the mouse when pressed@param y - The y coordinate of the mouse when pressed@return Has the tool handled the event?\*/bool Tool\_Move::Pressed(int button, int state, int x, int y) {if (currentCanvas.checkInside(x, y)) {// get mouse position in canvas coordinatesint cx = (x - currentCanvas.xOffset)

/ currentCanvas.zoom;int cy = (y - currentCanvas.yOffset) / currentCanvas.zoom;// remember the drag start position (mouse down) and end position (mouse up)if ((button == GLUT\_LEFT\_BUTTON) && (state == GLUT\_DOWN) && !isMouseDown) {isMouseDown = true;startMouseX = cx;startMouseY = cy;return true;}if ((button == GLUT\_LEFT\_BUTTON) && (state == GLUT\_UP) && isMouseDown)

{isMouseDown = false;endMouseX = cx;endMouseY = cy;isDisplaying = true;return true;}}return false;}/\*Should this tool take priority for receiving mouse events@param button - Mouse button pressed@param state - State of mouse event (down or up)@param x - The x coordinate of the mouse when pressed@param y - The y coordinate of the mouse when pressed@return Should this tool take priority for receiving mouse events\*/bool Tool\_Move::BlockMousePress(int button, int state, int x, int y) {if (isMouseDown) {// take priority with mouse events when tool is selected and mouse is over canvasif (currentCanvas.checkInside(x, y)) {return true;}}isMouseDown = false;return false;}/\*Handles special key events (arrow keys) for the tool@param key - the key that was pressed@param x - x position of the mouse@param y - y position of the mouse@return Has the tool handled the event?\*/bool Tool\_Move::SpecialKey(int key, int x, int y) {std::string x\_y;if (isDisplaying) {// get rect coordinatesint minX = std::min(startMouseX, endMouseX);int maxX = std::max(startMouseX, endMouseX);int minY = std::min(startMouseY, endMouseY);int maxY = std::max(startMouseY, endMouseY);switch (key){case GLUT\_KEY\_LEFT:// left key pressedif (minX > 0)

{Colour white = { 1.0f, 1.0f, 1.0f };for (int x = 0; x < maxX - minX; x++)

{for (int y = 0; y < maxY - minY; y++) {// move all pixels leftif (x == 0){x\_y = std::to\_string(minX - 1) + "\*" +std:: to\_string(minY + y);pix\_color[x\_y] = currentCanvas.GetPixelColour(minX - 1, minY + y);}currentCanvas.SetPixelColour(minX + x - 1, minY + y, currentCanvas.GetPixelColour(minX + x, minY + y));if (x == maxX - minX - 1)

{// leave column of white pixels on the rightx\_y = std::to\_string(minX+x) + "\*" + std::to\_string(minY + y);if (pix\_color.count(x\_y) > 0)

{currentCanvas.SetPixelColour(minX + x, minY + y, pix\_color[x\_y]);pix\_color.erase(x\_y);}else {currentCanvas.SetPixelColour(minX

+ x, minY + y, white);}}}}// rectangle has movedstartMouseX--;endMouseX--

;return true;}break;// do the same for other arrow keyscase GLUT\_KEY\_RIGHT:if (maxX < currentCanvas.width) {Colour white = { 1.0f, 1.0f, 1.0f };for (int x

= maxX - minX - 1; x >= 0; x--) {for (int y = 0; y < maxY - minY; y++) {if (x

== maxX - minX - 1){x\_y = std::to\_string(minX + x + 1) + "\*" + std::to\_string(minY + y);pix\_color[x\_y] = currentCanvas.GetPixelColour(minX + x + 1, minY + y);}currentCanvas.SetPixelColour(minX + x + 1, minY + y, currentCanvas.GetPixelColour(minX + x, minY + y));if (x == 0) {x\_y = std::to\_string(minX + x) + "\*" + std::to\_string(minY + y);if (pix\_color.count(x\_y) > 0) {currentCanvas.SetPixelColour(minX + x, minY + y, pix\_color[x\_y]);pix\_color.erase(x\_y);}else {currentCanvas.SetPixelColour(minX

+ x, minY + y, white);}}}}startMouseX++;endMouseX++;return true;}break;case GLUT\_KEY\_UP:if (minY > 0) {Colour white = { 1.0f, 1.0f, 1.0f };for (int x = 0; x < maxX - minX; x++) {for (int y = 0; y < maxY - minY; y++) {if (y==0){x\_y = std::to\_string(minX + x ) + "\*" + std::to\_string(minY + y - 1);pix\_color[x\_y] = currentCanvas.GetPixelColour(minX + x, minY + y - 1);}currentCanvas.SetPixelColour(minX + x, minY + y - 1, currentCanvas.GetPixelColour(minX + x, minY + y));if (y == maxY - minY - 1)

{x\_y = std::to\_string(minX + x) + "\*" + std::to\_string(minY + y);if (pix\_color.count(x\_y) > 0) {currentCanvas.SetPixelColour(minX + x, minY + y , pix\_color[x\_y]);pix\_color.erase(x\_y);}else {currentCanvas.SetPixelColour(minX

+ x, minY + y, white);}}}}startMouseY--;endMouseY--;return true;}break;case GLUT\_KEY\_DOWN:if (maxY < currentCanvas.height) {Colour white = { 1.0f, 1.0f, 1.0f };for (int x = 0; x < maxX - minX; x++) {for (int y = maxY - minY - 1; y

>= 0; y--) {if (y == maxY - minY - 1){x\_y = std::to\_string(minX + x ) + "\*" + std::to\_string(minY + y + 1);pix\_color[x\_y] = currentCanvas.GetPixelColour(minX + x, minY + y + 1);}currentCanvas.SetPixelColour(minX + x, minY + y + 1, currentCanvas.GetPixelColour(minX + x, minY + y));if (y == 0) {x\_y = std::to\_string(minX + x) + "\*" + std::to\_string(minY + y);if (pix\_color.count(x\_y) > 0) {currentCanvas.SetPixelColour(minX + x, minY + y, pix\_color[x\_y]);pix\_color.erase(x\_y);}else {currentCanvas.SetPixelColour(minX

+ x, minY + y, white);}}}}startMouseY++;endMouseY++;return true;}break;}return true;}return false;}

## Tool\_Pen.h :

/\*Tool\_Pen.hImplements the Pen tool\*/#pragma once/\*Handles mouse press events passed onto the Pen tool@param button - Mouse button pressed@param state - State of mouse event (down or up)@param x - The x coordinate of the mouse when pressed@param y - The y coordinate of the mouse when pressed@return Has the tool handled the event?\*/bool Tool\_Pen::Pressed(int button, int state, int x, int y) {if (currentCanvas.checkInside(x, y)) {if ((button == GLUT\_LEFT\_BUTTON) && (state == GLUT\_DOWN) && !isMouseDown) {std::cout << "start drag" << std::endl;isMouseDown = true;mouseLastX = (x -

currentCanvas.xOffset) / currentCanvas.zoom;mouseLastY = (y - currentCanvas.yOffset) / currentCanvas.zoom;return true;}if ((button == GLUT\_LEFT\_BUTTON) && (state == GLUT\_UP) && isMouseDown) {std::cout << "end drag" << std::endl;isMouseDown = false;return true;}}return false;}/\*Handles mouse movement events passed to the tool@param x - Mouse position x coordinate@param y - Mouse position y coordinate@return True if the tool handles the hovering event\*/bool Tool\_Pen::Hover(int x, int y) {if (isMouseDown) {if (currentCanvas.checkInside(x, y)) {// we want to colour all pixels on the line between the last and current mouse positions// First convert current mouse position to canvas coordinatesint canvasCoordX = (x - currentCanvas.xOffset) / currentCanvas.zoom;int canvasCoordY = (y - currentCanvas.yOffset) / currentCanvas.zoom;// If the mouse hasn't moved, colour the single pixel at its positionif ((canvasCoordX == mouseLastX) && (canvasCoordY == mouseLastY)) {currentCanvas.SetPixelColour(canvasCoordX, canvasCoordY, selectedColour);}else {int minX = std::min(canvasCoordX, mouseLastX);int maxX = std::max(canvasCoordX, mouseLastX);int minY = std::min(canvasCoordY, mouseLastY);int maxY = std::max(canvasCoordY, mouseLastY);// Find a unit vector moving along the line from last mouse position to current mouse positiondouble length = std::sqrt(std::pow(maxX - minX, 2) + std::pow(maxY - minY, 2));double moveX = (canvasCoordX - mouseLastX) / length;double moveY = (canvasCoordY - mouseLastY) / length;// we start at current mouse positiondouble curX = mouseLastX;double curY = mouseLastY;// keep moving in the direction of the unit vector and colouring in pixels until outside rangewhile ((curX <= maxX) && (curY <= maxY) && (curX

>= minX) && (curY >= minY)) {currentCanvas.SetPixelColour(round(curX), round(curY), selectedColour);curX += moveX;curY += moveY;}}// remember the mouse position for next timemouseLastX = canvasCoordX;mouseLastY = canvasCoordY;}return true;}return currentCanvas.checkInside(x, y);}/\*Should this tool take priority for receiving mouse events@param button - Mouse button pressed@param state - State of mouse event (down or up)@param x - The x coordinate of the mouse when pressed@param y - The y coordinate of the mouse when pressed@return Should this tool take priority for receiving mouse events\*/bool Tool\_Pen::BlockMousePress(int button, int state, int x, int y)

{if (isMouseDown) {// Should take mouse events if the mouse is already down and over the canvasif (currentCanvas.checkInside(x, y)) {return true;}}isMouseDown = false;return false;}

## Tool\_Rect.h :

/\*Tool\_Rect.hImplements the Rect tool\*/

#pragma once/\*Handles mouse press events passed onto the Rect tool@param button - Mouse button pressed@param state - State of mouse event (down or up)@param x - The x coordinate of the mouse when pressed@param y - The y coordinate of the mouse when pressed@return Has the tool handled the event?\*/bool Tool\_Rect::Pressed(int button, int state, int x, int y) {if (currentCanvas.checkInside(x, y)) {// convert mouse position into canvas coordinatesint cx = (x - currentCanvas.xOffset) / currentCanvas.zoom;int cy = (y - currentCanvas.yOffset) / currentCanvas.zoom;// remember the start mouse position if this is start of a dragif ((button == GLUT\_LEFT\_BUTTON) && (state

== GLUT\_DOWN) && !isMouseDown) {isMouseDown = true;startMouseX = cx;startMouseY = cy;return true;}// draw the rect if this is the end of a dragif ((button == GLUT\_LEFT\_BUTTON) && (state == GLUT\_UP) && isMouseDown)

{if ((startMouseX == cx) && (startMouseY == cy)) {// if the mouse hasn't moved, just colour one pixelcurrentCanvas.SetPixelColour(cx, cy, selectedColour);}else {// get the rect coordinatesint minX = std::min(cx, startMouseX);int maxX = std::max(cx, startMouseX);int minY = std::min(cy, startMouseY);int maxY = std::max(cy, startMouseY);for (int px = 0; px <= maxX

* minX; px++) {for (int py = 0; py <= maxY - minY; py++) {// fill in the pixelsif (px == 0 || px == (maxX - minX)) {currentCanvas.SetPixelColour(minX

+ px, minY + py, selectedColour);}else {if (py == 0 || py == (maxY - minY))

{currentCanvas.SetPixelColour(minX + px, minY + py, selectedColour);}}}}}isMouseDown = false;return true;}}return false;}/\*Should this tool take priority for receiving mouse events@param button - Mouse button pressed@param state - State of mouse event (down or up)@param x - The x coordinate of the mouse when pressed@param y - The y coordinate of the mouse when pressed@return Should this tool take priority for receiving mouse events\*/bool Tool\_Rect::BlockMousePress(int button, int state, int x, int y)

{// if during a drag, this tool should take the mouse events firstif (isMouseDown) {if (currentCanvas.checkInside(x, y)) {return true;}}isMouseDown = false;return false;}

## Tool\_Rotate.h :

/\*Tool\_Rotate.hImplements the Move tool\*/#include<stdlib.h>#include "cmath"#pragma oncestatic float rotate\_about\_x, rotate\_about\_y, x\_new, y\_new, prev\_key; float angle =90;static float radian = angle \* 3.141593/180;/\*Initializes tool when selected\*/void Tool\_Rotate::Start()

{isDisplaying = false;isMouseDown = false;flickerFrameCount = 20;flickerColor

= false;}/\*Disables tool when another is selected\*/void Tool\_Rotate::End()

{isDisplaying = false;isMouseDown = false;}/\*Displays rect around selected

pixels@param window\_width - the width of the window@param window\_height - the height of the window\*/void Tool\_Rotate::Display(int window\_width, int window\_height) {if ((!Tool\_Rotate::isDisplaying) || isMouseDown) {// display white cover over canvas while user needs to draw rectangle to select pixelsglBegin(GL\_QUADS);glColor4f(1.0f, 1.0f, 1.0f, 0.7f);glVertex2f(currentCanvas.xOffset, window\_height - currentCanvas.yOffset);glVertex2f(currentCanvas.xOffset + (currentCanvas.width \* currentCanvas.zoom), window\_height - currentCanvas.yOffset);glVertex2f(currentCanvas.xOffset + (currentCanvas.width \* currentCanvas.zoom), window\_height - currentCanvas.yOffset - (currentCanvas.height \* currentCanvas.zoom));glVertex2f(currentCanvas.xOffset, window\_height - currentCanvas.yOffset - (currentCanvas.height \* currentCanvas.zoom));glEnd();}else {// display the blue/red flickering rectangle around selected pixelsglBegin(GL\_LINES);if (flickerColor)

{glColor4f(0.4f, 0.4f, 1.0f, 1.0f);}else {glColor4f(1.0f, 0.4f, 0.4f, 1.0f);}glVertex2f(currentCanvas.xOffset + (startMouseX \* currentCanvas.zoom), window\_height - currentCanvas.yOffset - (startMouseY \* currentCanvas.zoom));glVertex2f(currentCanvas.xOffset + (endMouseX \* currentCanvas.zoom), window\_height - currentCanvas.yOffset - (startMouseY \* currentCanvas.zoom));glVertex2f(currentCanvas.xOffset + (endMouseX \* currentCanvas.zoom), window\_height - currentCanvas.yOffset - (startMouseY \* currentCanvas.zoom));glVertex2f(currentCanvas.xOffset + (endMouseX \* currentCanvas.zoom), window\_height - currentCanvas.yOffset - (endMouseY \* currentCanvas.zoom));glVertex2f(currentCanvas.xOffset + (endMouseX \* currentCanvas.zoom), window\_height - currentCanvas.yOffset - (endMouseY \* currentCanvas.zoom));glVertex2f(currentCanvas.xOffset + (startMouseX \* currentCanvas.zoom), window\_height - currentCanvas.yOffset - (endMouseY \* currentCanvas.zoom));glVertex2f(currentCanvas.xOffset + (startMouseX \* currentCanvas.zoom), window\_height - currentCanvas.yOffset - (endMouseY \* currentCanvas.zoom));glVertex2f(currentCanvas.xOffset + (startMouseX \* currentCanvas.zoom), window\_height - currentCanvas.yOffset - (startMouseY \* currentCanvas.zoom));glEnd();// change the flickering colour every 20 framesflickerFrameCount--;if (flickerFrameCount <= 0) {flickerFrameCount = 20;flickerColor = !flickerColor;}}}/\*Handles mouse press events passed onto the Move tool@param button - Mouse button pressed@param state - State of mouse event (down or up)@param x - The x coordinate of the mouse when pressed@param y - The y coordinate of the mouse when pressed@return Has the tool handled the event?\*/bool Tool\_Rotate::Pressed(int button, int state, int x, int y) {if (currentCanvas.checkInside(x, y)) {// get mouse position in canvas coordinatesint cx = (x - currentCanvas.xOffset) / currentCanvas.zoom;int cy = (y - currentCanvas.yOffset) / currentCanvas.zoom;// remember the drag start position (mouse down) and end position (mouse up)if ((button == GLUT\_LEFT\_BUTTON) && (state == GLUT\_DOWN)

&& !isMouseDown) {isMouseDown = true;startMouseX = cx;startMouseY = cy;return true;}if ((button == GLUT\_LEFT\_BUTTON) && (state == GLUT\_UP) && isMouseDown)

{isMouseDown = false;endMouseX = cx;endMouseY = cy;isDisplaying = true;return true;}}return false;}/\*Should this tool take priority for receiving mouse events@param button - Mouse button pressed@param state - State of mouse event (down or up)@param x - The x coordinate of the mouse when pressed@param y - The y coordinate of the mouse when pressed@return Should this tool take priority for receiving mouse events\*/bool Tool\_Rotate::BlockMousePress(int button, int state, int x, int y) {if (isMouseDown) {// take priority with mouse events when tool is selected and mouse is over canvasif (currentCanvas.checkInside(x, y)) {return true;}}isMouseDown = false;return false;}/\*Handles special key events (arrow keys) for the tool@param key - the key that was pressed@param x - x position of the mouse@param y - y position of the mouse@return Has the tool handled the event?bool Tool\_Rotate::SpecialKey(int key, int x, int y) {std::string x\_y;if (isDisplaying) {// get rect coordinatesint minX = std::min(startMouseX, endMouseX);int maxX = std::max(startMouseX, endMouseX);int minY = std::min(startMouseY, endMouseY);int maxY = std::max(startMouseY, endMouseY);switch (key){case GLUT\_KEY\_LEFT:// left key pressedif (minX > 0)

{Colour white = { 1.0f, 1.0f, 1.0f };for (int x = 0; x < maxX - minX; x++)

{for (int y = 0; y < maxY - minY; y++) {// move all pixels leftif (x == 0){x\_y = std::to\_string(minX - 1) + "\*" + std::to\_string(minY + y);pix\_color[x\_y] = currentCanvas.GetPixelColour(minX - 1, minY + y);}currentCanvas.SetPixelColour(minX + x - 1, minY + y, currentCanvas.GetPixelColour(minX + x, minY + y));if (x == maxX - minX - 1)

{// leave column of white pixels on the rightx\_y = std::to\_string(minX + x) + "\*" + std::to\_string(minY + y);if (pix\_color.count(x\_y) > 0)

{currentCanvas.SetPixelColour(minX + x, minY + y, pix\_color[x\_y]);pix\_color.erase(x\_y);}else {currentCanvas.SetPixelColour(minX

+ x, minY + y, white);}}}}// rectangle has movedstartMouseX--;endMouseX--

;return true;}break;// do the same for other arrow keyscase GLUT\_KEY\_RIGHT:if (maxX < currentCanvas.width) {Colour white = { 1.0f, 1.0f, 1.0f };for (int x

= maxX - minX - 1; x >= 0; x--) {for (int y = 0; y < maxY - minY; y++) {if (x

== maxX - minX - 1){x\_y = std::to\_string(minX + x + 1) + "\*" + std::to\_string(minY + y);pix\_color[x\_y] = currentCanvas.GetPixelColour(minX + x + 1, minY + y);}currentCanvas.SetPixelColour(minX + x + 1, minY + y, currentCanvas.GetPixelColour(minX + x, minY + y));if (x == 0) {x\_y = std::to\_string(minX + x) + "\*" + std::to\_string(minY + y);if (pix\_color.count(x\_y) > 0) {currentCanvas.SetPixelColour(minX + x, minY + y, pix\_color[x\_y]);pix\_color.erase(x\_y);}else {currentCanvas.SetPixelColour(minX

+ x, minY + y, white);}}}}startMouseX++;endMouseX++;return true;}break;case GLUT\_KEY\_UP:if (minY > 0) {Colour white = { 1.0f, 1.0f, 1.0f };for (int x = 0; x < maxX - minX; x++) {for (int y = 0; y < maxY - minY; y++) {if (y == 0){x\_y = std::to\_string(minX + x) + "\*" + std::to\_string(minY + y -

1);pix\_color[x\_y] = currentCanvas.GetPixelColour(minX + x, minY + y - 1);}currentCanvas.SetPixelColour(minX + x, minY + y - 1, currentCanvas.GetPixelColour(minX + x, minY + y));if (y == maxY - minY - 1)

{x\_y = std::to\_string(minX + x) + "\*" + std::to\_string(minY + y);if (pix\_color.count(x\_y) > 0) {currentCanvas.SetPixelColour(minX + x, minY + y, pix\_color[x\_y]);pix\_color.erase(x\_y);}else {currentCanvas.SetPixelColour(minX

+ x, minY + y, white);}}}}startMouseY--;endMouseY--;return true;}break;case GLUT\_KEY\_DOWN:if (maxY < currentCanvas.height) {Colour white = { 1.0f, 1.0f, 1.0f };for (int x = 0; x < maxX - minX; x++) {for (int y = maxY - minY - 1; y

>= 0; y--) {if (y == maxY - minY - 1){x\_y = std::to\_string(minX + x) + "\*" + std::to\_string(minY + y + 1);pix\_color[x\_y] = currentCanvas.GetPixelColour(minX + x, minY + y + 1);}currentCanvas.SetPixelColour(minX + x, minY + y + 1, currentCanvas.GetPixelColour(minX + x, minY + y));if (y == 0) {x\_y = std::to\_string(minX + x) + "\*" + std::to\_string(minY + y);if (pix\_color.count(x\_y) > 0) {currentCanvas.SetPixelColour(minX + x, minY + y, pix\_color[x\_y]);pix\_color.erase(x\_y);}else {currentCanvas.SetPixelColour(minX

+ x, minY + y, white);}}}}startMouseY++;endMouseY++;return true;}break;}return true;}return false;}\*/bool Tool\_Rotate::SpecialKey(int key, int x, int y) {if (isDisplaying) {// get rect coordinatesfloat minX = std::min(startMouseX, endMouseX);float maxX = std::max(startMouseX, endMouseX);float minY = std::min(startMouseY, endMouseY);float maxY = std::max(startMouseY, endMouseY);float x\_dist = maxX - minX;float y\_dist = maxY - minY;rotate\_about\_x = minX;rotate\_about\_y = minY;switch (key){case GLUT\_KEY\_DOWN:Colour white = { 1.0f, 1.0f, 1.0f };for (y = 0; y < maxY - minY

; y++) {for (x = 0; x < maxX - minX ; x++) {x\_new = ((minX + x - rotate\_about\_x) \* std::cos(radian)) + ((minY + y - rotate\_about\_y) \* std::sin(radian)) + rotate\_about\_x - x\_dist ;y\_new = -((minX + x - rotate\_about\_x) \* std::sin(radian)) + ((minY + y - rotate\_about\_y) \* std::cos(radian)) + rotate\_about\_y + y\_dist;currentCanvas.SetPixelColour(x\_new, y\_new, currentCanvas.GetPixelColour(minX + x, minY + y));currentCanvas.SetPixelColour(minX + x, minY + y, white);}}startMouseX = ((startMouseX - rotate\_about\_x) \* std::cos(radian)) - ((startMouseY- rotate\_about\_y) \* std::sin(radian)) + rotate\_about\_x ;startMouseY = ((startMouseX - rotate\_about\_x) \* std::sin(radian)) + ((startMouseY- rotate\_about\_y) \* std::cos(radian)) + rotate\_about\_y + y\_dist;endMouseX = ((endMouseX - rotate\_about\_x) \* std::cos(radian)) - ((endMouseY - rotate\_about\_y) \* std::sin(radian)) + rotate\_about\_x;endMouseY = ((endMouseX

* rotate\_about\_x) \* std::sin(radian)) + ((endMouseY - rotate\_about\_y) \* std::cos(radian)) + rotate\_about\_y + y\_dist;break;}return true;}return false;}

## Toolbar.h :

/\*Toolbar.hImplements the toolbar on the left of the screen\*/#pragma once#include <map>#include<string>#include<iostream>#include <string>// define class structures// class to pass on events to the selected toolclass ToolEvents {public:static void Start();static void End();static void Display(int window\_width, int window\_height);static bool Pressed(int button, int state, int x, int y);static bool Hover(int x, int y);static bool SpecialKey(int key, int x, int y);static bool BlockMousePress(int button, int state, int x, int y);};class Tool\_Erase {public:static bool isMouseDown;static int startMouseX;static int startMouseY;static int endMouseX;static int endMouseY;static bool isDisplaying;static int flickerFrameCount;static bool flickerColor;static void Start();static void End();static void Display(int window\_width, int window\_height);static bool Pressed(int button, int state, int cx, int cy);static bool BlockMousePress(int button, int state, int x, int y);static bool SpecialKey(int key, int x, int y);};// classes for the toolsclass Tool\_Pen

{public:static bool isMouseDown;static int mouseLastX;static int mouseLastY;static bool Pressed(int button, int state, int cx, int cy);static bool Hover(int x, int y);static bool BlockMousePress(int button, int state, int x, int y);};class Tool\_Fill {public:static void Fill(Colour startColour, int x, int y);static bool Pressed(int button, int state, int x, int y);};class Tool\_Rect {public:static bool isMouseDown;static int startMouseX;static int startMouseY;static bool Pressed(int button, int state, int cx, int cy);static bool BlockMousePress(int button, int state, int x, int y);};class Tool\_Circ {public:static bool isMouseDown;static int startMouseX;static int startMouseY;static bool Pressed(int button, int state, int cx, int cy);static bool BlockMousePress(int button, int state, int x, int y);};class Tool\_Move {public:static bool isMouseDown;static int startMouseX;static int startMouseY;static int endMouseX;static int endMouseY;static bool isDisplaying;static int flickerFrameCount;static bool flickerColor;static void Start();static void End();static void Display(int window\_width, int window\_height);static bool Pressed(int button, int state, int cx, int cy);static bool BlockMousePress(int button, int state, int x, int y);static bool SpecialKey(int key, int x, int y);};class Tool\_Rotate

{public:static bool isMouseDown;static int startMouseX;static int startMouseY;static int endMouseX;static int endMouseY;static bool isDisplaying;static int flickerFrameCount;static bool flickerColor;static void Start();static void End();static void Display(int window\_width, int window\_height);static bool Pressed(int button, int state, int cx, int cy);static bool BlockMousePress(int button, int state, int x, int y);static bool SpecialKey(int key, int x, int y);};/\*This class implements the toolbar on the left of the window\*/class Toolbar {public:// the index of the selected toolstatic int selectedButton;// buttons for each tool in the menustatic

Button penButton;static Button eraseButton;static Button fillButton;static Button rectButton;static Button circleButton;static Button moveButton;static Button rotateButton;/\*Callback invoked when a tool button is pressed from the toolbar@param button - the button that was pressed\*/static void ToolButtonPressed(Button button) {ToolEvents::End();if (button.text == "Pen")

{ selectedButton = 0; }if (button.text == "Erase") { selectedButton = 1; }if (button.text == "Fill") { selectedButton = 2; }if (button.text == "Rect") { selectedButton = 3; }if (button.text == "Circ") { selectedButton = 4; }if (button.text == "Move") { selectedButton = 5; }if (button.text == "Rot") { selectedButton = 6; }ToolEvents::Start();}/\*Initializes the toolbar\*/static void Init() {// start with pen tool selectedselectedButton = 0;// create the buttons for the toolbarpenButton = Button::Create(0, 100, 78, 40, (char

\*)"Pen", ToolButtonPressed, true);eraseButton = Button::Create(0, 140, 78, 40, (char \*)"Erase", ToolButtonPressed, true);fillButton = Button::Create(0, 180, 78, 40, (char \*)"Fill", ToolButtonPressed, true);rectButton =

Button::Create(0, 220, 78, 40, (char \*)"Rect", ToolButtonPressed,

true);circleButton = Button::Create(0, 260, 78, 40, (char \*)"Circ",

ToolButtonPressed, true);moveButton = Button::Create(0, 300, 78, 40, (char\*)"Move", ToolButtonPressed, true);rotateButton = Button::Create(0, 340, 78, 40, (char\*)"Rot", ToolButtonPressed, true);}/\*Displays the toolbar@param window\_width - the width of the window@param window\_height - the height of the window\*/static void Display(int window\_width, int window\_height) {// draw the buttonspenButton.Display(window\_width, window\_height);eraseButton.Display(window\_width, window\_height);fillButton.Display(window\_width, window\_height);rectButton.Display(window\_width, window\_height);circleButton.Display(window\_width, window\_height);moveButton.Display(window\_width, window\_height);rotateButton.Display(window\_width, window\_height);// draw a blue overlay on the selected buttonglColor4f(0.0f, 1.0f, 1.0f, 0.4f);glBegin(GL\_QUADS);int selY = window\_height - ( 100 + (selectedButton \* 40));glVertex2f(0, selY);glVertex2f(78, selY);glVertex2f(78, selY - 40);glVertex2f(0, selY - 40);glEnd();}/\*Handles mouse press events passing them on to the selected tool@param button - Mouse button pressed@param state

* State of mouse event (down or up)@param x - The x coordinate of the mouse when pressed@param y - The y coordinate of the mouse when pressed@return Has the event been handled?\*/static bool Pressed(int button, int state, int x, int y) {if ((selectedButton != 0) && (penButton.Pressed(button, state, x, y))) {return true;}if ((selectedButton != 1) && (eraseButton.Pressed(button, state, x, y))) {return true;}if ((selectedButton != 2) && (fillButton.Pressed(button, state, x, y))) {return true;}if ((selectedButton

!= 3) && (rectButton.Pressed(button, state, x, y))) {return true;}if ((selectedButton != 4) && (circleButton.Pressed(button, state, x, y)))

{return true;}if ((selectedButton != 5) && (moveButton.Pressed(button, state,

x, y))) {return true;}if ((selectedButton != 6) && (rotateButton.Pressed(button, state, x, y))) {return true;}return false;}/\*Handles mouse movement events passed to the tool@param x - Mouse position x coordinate@param y - Mouse position y coordinate@return True if the event gets handled\*/static bool Hover(int x, int y) {bool output = false;if (penButton.Hover(x, y)) {output = true;}if (eraseButton.Hover(x, y))

{output = true;}if (fillButton.Hover(x, y)) {output = true;}if (rectButton.Hover(x, y)) {output = true;}if (circleButton.Hover(x, y))

{output = true;}if (moveButton.Hover(x, y)) {output = true;}if (rotateButton.Hover(x, y)) {output = true;}return output;}};/\*Passes on start event to the selected tool\*/void ToolEvents::Start() {switch (Toolbar::selectedButton) {case 5:Tool\_Move::Start();break;case 6:Tool\_Rotate::Start();break;case 1:Tool\_Erase::Start();break;}}/\*Passes on end event ot the selected tool\*/void ToolEvents::End() {switch (Toolbar::selectedButton) {case 5:Tool\_Move::End();break;case 6:Tool\_Rotate::End();break;case 1:Tool\_Erase::End();break;}}/\*Passes display event onto selected tool@param window\_width - the width of the window@param window\_height - the height of the window\*/void ToolEvents::Display(int window\_width, int window\_height) {switch (Toolbar::selectedButton) {case 5:Tool\_Move::Display(window\_width, window\_height);break;case 6:Tool\_Rotate::Display(window\_width, window\_height);break;case 1:Tool\_Erase::Display(window\_width, window\_height);break;}}/\*Passes Mouse pressed event onto the selected tool@param button - Mouse button pressed@param state - State of mouse event (down or up)@param x - The x coordinate of the mouse when pressed@param y - The y coordinate of the mouse when pressed@return Has the tool handled the event?\*/bool ToolEvents::Pressed(int button, int state, int x, int y) {switch (Toolbar::selectedButton) {case 0:if (Tool\_Pen::Pressed(button, state, x, y))

{return true;}break;case 1:if (Tool\_Erase::Pressed(button, state, x, y))

{return true;}break;case 2:if (Tool\_Fill::Pressed(button, state, x, y))

{return true;}break;case 3:if (Tool\_Rect::Pressed(button, state, x, y))

{return true;}break;case 4:if (Tool\_Circ::Pressed(button, state, x, y))

{return true;}break;case 5:if (Tool\_Move::Pressed(button, state, x, y))

{return true;}break;case 6:if (Tool\_Rotate::Pressed(button, state, x, y))

{return true;}break;}return false;}/\*Passes mouse movement events onto the selected tool@param x - Mouse position x coordinate@param y - Mouse position y coordinate@return True if the event gets handled\*/bool ToolEvents::Hover(int x, int y) {switch (Toolbar::selectedButton) {case 0:if (Tool\_Pen::Hover(x, y)) {return true;}break;}return false;}/\*Passes special key events (arrow keys) on to the selected tool@param key - the key that was pressed@param x - x position of the mouse@param y - y position of the mouse@return Has the event been handled?\*/bool ToolEvents::SpecialKey(int key, int x, int y) {switch (Toolbar::selectedButton) {case 1:if (Tool\_Erase::SpecialKey(key, x, y)) {return true;}break;case 5:if

(Tool\_Move::SpecialKey(key, x, y)) {return true;}break;case 6:if (Tool\_Rotate::SpecialKey(key, x, y)) {return true;}break;}return false;}/\*Should the selected tool take priority for receiving mouse events@param button - Mouse button pressed@param state - State of mouse event (down or up)@param x - The x coordinate of the mouse when pressed@param y - The y coordinate of the mouse when pressed@return Should the selected tool take priority for receiving mouse events\*/bool ToolEvents::BlockMousePress(int button, int state, int x, int y) {switch (Toolbar::selectedButton) {case 0:if (Tool\_Pen::BlockMousePress(button, state, x, y)) {return true;}break;case 1:if (Tool\_Erase::BlockMousePress(button, state, x, y)) {return true;}break;case 3:if (Tool\_Rect::BlockMousePress(button, state, x, y)) {return true;}break;case 4:if (Tool\_Circ::BlockMousePress(button, state, x, y))

{return true;}break;case 5:if (Tool\_Move::BlockMousePress(button, state, x, y)) {return true;}break;case 6:if (Tool\_Rotate::BlockMousePress(button, state, x, y)) {return true;}break;}return false;}

## Top Menu Bar Callbacks.h :

/\*Top Menu Bar Calllbacks.hThis file adds callbacks to the top menu bar buttons. These are functions which are called when the "New", "Open", "Save" and "Save As" buttons are pressed\*/#pragma once#include "Button.h"#include "Cover.h"#include "Open File Dialogue.h"#include "Save File Dialogue.h"/\*"New" pressed then "Yes" pressed in "are you sure" dialogue@param button - The button that was pressed\*/void NewConfirmedCallback(Button button) {canvasAssigned = true;currentCanvas = NewCanvas(500, 500, 100, 100);}/\*"New" pressed@param button - The button that was pressed\*/void NewButtonPressed(Button button) {if (canvasAssigned)

{YesNoDialogue::Show("You will loose any unsaved changes. Continue?", NewConfirmedCallback);}else {canvasAssigned = true;currentCanvas = NewCanvas(500, 500, 100, 100);}}/\*"Open" pressed@param button - The button that was pressed\*/void OpenButtonPressed(Button button)

{OpenFileDialogue::Show();}/\*"Save" pressed@param button - The button that was pressed\*/void SaveButtonPressed(Button button) {// If there is no current canvas, show an errorif (!canvasAssigned) {AlertDialogue::Alert("No canvas has been created. Create one before saving.");return;}// If the current canvas is new (without a file path), go to Save File Dialogue (just like Save As)if (currentCanvas.fileName == "")

{SaveFileDialogue::Reset();SaveFileDialogue::Show();return;}// If it is new (with a file path) save it to the pathFileManagement::WriteFile(currentCanvas.fileName,

currentCanvas.Serialize());AlertDialogue::Alert("Saved to " + currentCanvas.fileName + ".dti");}/\*"SaveAs" pressed@param button - The button that was pressed\*/void SaveAsButtonPressed(Button button) {if (!canvasAssigned) {AlertDialogue::Alert("No canvas has been created. Create one before saving.");return;}SaveFileDialogue::Reset();SaveFileDialogue::Show();}

## Top MenuBar.h :

/\*Top Menu Bar.hImplements the drawing of the Top Menu Bar and handling of all associated events\*/#pragma once#include <vector>class TopMenuBar

{public:// Store a vector containing top menu bar buttonsstatic std::vector<Button> buttons;/\*Initializes the top menu bar\*/static void Init() {// Create instances of the button classes and add them to the vectorbuttons.push\_back(Button::Create(0, 0, 70, 40, (char \*)"New",

NewButtonPressed, true));buttons.push\_back(Button::Create(70, 0, 80, 40, (char \*)"Open", OpenButtonPressed, true));buttons.push\_back(Button::Create(150, 0, 80, 40, (char \*)"Save",

SaveButtonPressed, true));buttons.push\_back(Button::Create(230, 0, 110, 40, (char \*)"SaveAs", SaveAsButtonPressed, true));}/\*Displays the top menu bar@param window\_width - the width of the window@param window\_height - the height of the window\*/static void Display(int window\_width, int window\_height) {for (int i = 0; i < buttons.size(); i++) {// Pass on event to each buttonbuttons[i].Display(window\_width, window\_height);}}/\*Handles button pressed events for top menu bar@param button - Mouse button pressed@param state - State of mouse event (down or up)@param x - The x coordinate of the mouse when pressed@param y - The y coordinate of the mouse when pressed@return Has the Top Menu Bar handled the event?\*/static bool Pressed(int button, int state, int x, int y) {for (int i = buttons.size() - 1; i >= 0; i--) {// Pass on event to each buttonif (buttons[i].Pressed(button, state, x, y)) {return true;}}return false;}/\*Handles mouse move events for top menu bar@param x - The new x coordinate of the mouse@param y - The new y coordinate of the mouse@return Has the top menu bar handled the event?\*/static bool Hover(int x, int y)

{bool output = false;for (int i = buttons.size() - 1; i >= 0; i--) {// Pass on event to each buttonif (buttons[i].Hover(x, y)) {output = true;}}return output;}};

## Yes No Dialogue.h :

/\*Yes No Dialogue.hThis file adds a Yes/No Dialogue - a custom message with "Yes" and "No" buttonsThe No button closes the dialogue, the Yes button Invokes a custom callback function\*/#pragma onceclass YesNoDialogue

{public:// should the yes/no dialogue be displayedstatic bool show;// The yes and no buttonsstatic Button yesButton;static Button noButton;// message to displaystatic std::string message;// function to call when 'yes' button pressedstatic Callback yesCallback;/\*starts displaying the yes/no dialogue@param m - The message to display in the dialogue@param callback - The function to invoke when the yes button gets pressed\*/static void Show(std::string m, Callback callback) {message = m;yesCallback = callback;show = true;yesButton.Show();noButton.Show();}/\*stops displaying the yes/no dialogue\*/static void Hide() {show = false;yesButton.Hide();noButton.Hide();}/\*Callback invoked when the yes button is pressed@param button - the button that was pressed\*/static void YesPressed(Button button) {Hide();if (yesCallback) {// trigger the callback provided for the yes button(\*yesCallback)(yesButton);}}/\*Callback invoked when the no button is pressed\*/static void NoPressed(Button button)

{Hide();}/\*Initializes the yes/no dialogue\*/static void Init() {// creates the yes and no buttonsnoButton = Button::Create(0, 140, 100, 40, (char

\*)"No", NoPressed, true);yesButton = Button::Create(0, 140, 100, 40, (char

\*)"Yes", YesPressed, true);}/\*Displays the yes/no dialogue@param window\_width

* the width of the window@param window\_height - the height of the window\*/static void Display(int window\_width, int window\_height) {if (show)

{// Display another cover (as yes/no dialogues can appear above other dialogues)glBegin(GL\_QUADS);glColor4f(0.0f, 0.0f, 0.0f, 0.85f);glVertex2f(0, window\_height);glVertex2f(0 + window\_width, window\_height);glVertex2f(0 + window\_width, 0);glVertex2f(0, 0);glEnd();// display the messagedisplay\_text(message, (window\_width / 2) - (get\_text\_width(message) / 2), window\_height - 100);// display the buttons, offset from the centernoButton.HorizontallyCenter(window\_width);noButton.x\_pos -= 55;yesButton.HorizontallyCenter(window\_width);yesButton.x\_pos += 55;noButton.Display(window\_width, window\_height);yesButton.Display(window\_width, window\_height);}}/\*Handles mouse pressed events for the yes/no dialogue@param button - Mouse button pressed@param state - State of mouse event (down or up)@param x - The x coordinate of the mouse when pressed@param y - The y coordinate of the mouse when pressed@return Has the yes/no dialogue handled the event?\*/static bool Pressed(int button, int state, int x, int y) {if (show) {// pass the event onto the yes and no buttonsif (noButton.Pressed(button, state, x, y)) {return true;}if (yesButton.Pressed(button, state, x, y)) {return true;}return true;}return false;}/\*Handles mouse movement events for the yes/no dialogue@param x - Mouse position x coordinate@param y - Mouse position y

coordinate@return True if the event gets handled\*/static bool Hover(int x, int y) {bool output = false;if (show) {// pass the event onto the yes and no buttonsif (noButton.Hover(x, y)) {output = true;}if (yesButton.Hover(x, y))

{output = true;}return true;}return false;}};