2D Graphics Primitives

The application to allow the visualization of basic 2D raster graphic has been developed using Qt Creator, an open-source application development environment. Programming language used to implement the algorithms is C++.

The Primary Components of the application are:

1. The Grid
2. The Drawing Primitives

1. The Grid

Each pixel of the Grid is a cell in a "**tableClass"** type which inherits **QAbstractTableModel**. It allows to change the dimensions of the grid, in terms of rows and columns, and also the pixel size.
The data stored in each cell of the table is an Object of type **QColor**. For this project, the RGB encoding has been used to set the pixel values.

The row, column and a matrix **("gridMatrix")** is passed as arguments to the tableModel object's constructor.
As part of the **"tableClass" object,** variables such as row, column and a matrix representing each cell is defined.

The **constructor of the MainWindow** class creates the "tableClass" object and attaches it to the UI interface provided by QMainWindow Class.
The initial coloring is set as black for pixels along the axes and white for everywhere else.
The pixel size is changed using setDefaultFunctionSize() of horizontal and vertical headers of the table model attached to the UI.
The rows and columns of the grid is changed by changing the dimensions of the matrix "gridMatrix" by reallocation and repainting the screen.
Every time, the middle row and middle column is painted as the axes.

2. The Drawing Primitives

The various algorithms implemented to create the basic 2D graphics primitives have been divided under the following heads displayed as tabs:

a. Lines
b. Circle
c. Ellipse
d. Polygon Fill
e. Line Clipping
f. Polygon Clipping
g. Transformations
h. Bezier Curves.

Upon setting the initial conditions like selecting points, a click on the desired algorithm's button, generates an event which calls the function attached to it (by default: on_pushbutton_<button number>_click()) to execute the algorithm.

Transformations:

A separate "**matrix**" class is defined to represent the points (in homogeneous coordinate form) and the transformation operations. The class supports multiplication of matrices to apply various types of transformations. The transformation is applied only on the endpoints of a line segment. On getting the new endpoints, lines are plotted using Bresenham's algorithm.

The operations are composite in nature, in the sense that firstly if the polygon is scaled to double in size and then shear transformation is applied, the scaled up polygon shall be sheared rather than the original one.

Steps to follow for correct execution:

In order to plot a line, point needs to be selected and point1 button needs to be clicked on. Similarly for second point. Upon selection opt for the desired algorithm to draw the line.

If a primitive requires multiple points at a time as in polygon drawing and Bezier curve, then after clicking on a pixel in the grid, the **Start** button needs to be clicked upon which shall reflect the number of points selected till then. If all the desired points have been selected, then the desired operation button is clicked upon.

For multiple lines, the counter displayed on the button is increased by 1 if clicked on twice.

For the transformations section, the x-axis and y-axis values for the translation, scale and shear operations are obtained from the same field.

Algorithms have been color coded to understand the change, if different algorithms is used to plot the line with the same endpoints. Similarly, the circle drawing algorithm and transformations have also been color coded.

For all algorithms, multiple lines are required to be drawn, Bresenham's algorithm has been used due to its continuity and speed.

Algorithms:

Rotation of a polygon:

1. Clear Screen by repainting all the pixel values to (255,255,255) except the ones for the axes.
2. The line joining the rotation point and the first point in the polygon vertices list is rotated by the angle specified using loopRot() function.
3. Then pairs of vertices are taken in the fashion (1,2) (2,3) and so on and the lines are rotated with respect to the previous end point.
4. Step 3 is repeated till (nth vertex, 1st vertex) is not rotated.
5. Finish

loopRot(int iter,float rad,matrix &point1,int i) function

(iter: vertex to be rotated, rad: angle, point1: previous vertex,i: flag)

1. If the first point is (x0,y0) then the endpoints of the line segment is translated by (-x0) in x-direction and by (-y0) in y-direction.
2. Then the points are rotated with respect to origin by the angle specified.
3. Then the obtained endpoints are again translated by (x0,y0) to get the desired endpoints.

Reflection of a polygon about a line:

1. The point is translated by (-x_line, -y_line) before transformation. x_line and y_line is the endpoint of the reflection line having lower x value.
2. The obtained point is reflected along y-axis by multiplying the y coordinates with (-1).
3. The points are rotated by twice the angle made by the reflection line with positive x-axis.
4. The final endpoints are obtained by translating the coordinates obtained from the previous step by (x_line, y_line).
5. Repeat from step 1 until all the vertices are reflected.
6. Plot the reflected polygons using the obtained vertices joining them using Bresenham's algorithm.

DDA (line Drawing):

1. Slope of the line is determined.
2. Depending on the slope:
   If positive and less than 1, then y is incremented by slope for very single increase in x.
   If positive and greater than 1, then x is incremented by inverse of slope for very single increase in y.
   If negative and greater than -1, then y is incremented by slope for very single increase in x.
   If negative and less than -1, then x is incremented by inverse of slope for very single increase in y.
3. Step 2 is repeated till the other end of the line is reached.

Midpoint method for Circle Drawing:

1. For drawing the circle, an eight-way symmetry method is used.
   Point is determined in one octant, and then reflected about vertical, horizontal, and two lines at 45 and 135 degrees passing through the center of the circle. To get the points in other octants.
2. For a single octant, values of x and y is determined starting with x=0 and y=radius. A decision variable p is computed.
   p(0)=1-radius.
3. If p is negative, then y remains same, x is incremented by 1. Or if p is positive, x is incremented, y is decremented.
   Depending upon the value of decision variable:
   If $p(k) < 0$:
   $p(k+1)=p(k)+2(x+1)+1$
   else
   $p(k+1)=p(k)+2(x+1) + 1 -2(y-1)$
4. Then the obtained values of x and y and their symmetrical counterparts are used to plot the circle 8 points at a time.
5. Step 3 is repeated until x is greater than y.

Cohen Sutherland Line Clipping:

1. Screen is cleared.
2. The position codes for each vertex is computed with respect to the bounding lines.
3. The endpoints of the lines are stored in an array.
4. The Lines are dropped, clipped or retained depending upon the position codes.
   If both have position codes (PC) as 0, then the line is retained.
   If binary AND of PC1 and PC2 is non zero, then the line is completely dropped.
   If either of the above two is not satisfied, then the line is clipped.
   For clipping, the codes are updated until both the codes are converted to 0.
   While converting the codes. the end points are also updated by getting the coordinate values using the value of the bounding line with which the given line intersects.
   After the codes are both zero, the new clipped line is printed.
5. Step 4 is repeated until all the lines have been clipped w.r.t. the clipping window.

Polygon Fill using Flood Fill:

Boundary Color (255,255,255) (white)
Fill Color (187,187,187) (dark shade of grey)

1. The starting point is the point selected inside the polygon or structure. Boundary colour and fill colour is set.
2. The algorithm traverses the region around the selected pixel in four directions (North, South,East and West).
3. If the pixel is not a background color, the function returns to the previous pixel.
   Else, the pixel is colored with the fill color, and again the region around the current colored pixel is explored in the 4 directions.
4. Coloring in any direction stops if boundary or non-background color is encountered.