GeoLib Drawing

Document version 1.0

Contents

Introduction

All GeoLib classes that represent a geometric entity can be drawn to the computer screen using the CGeoDraw class. This class uses the Microsoft Foundation Classes CDC, CPen and CBrush to provide a simple way of drawing geometric objects. This document assumes the use of Microsoft Visual Studio 2005 with C++ and a typical document view architecture.

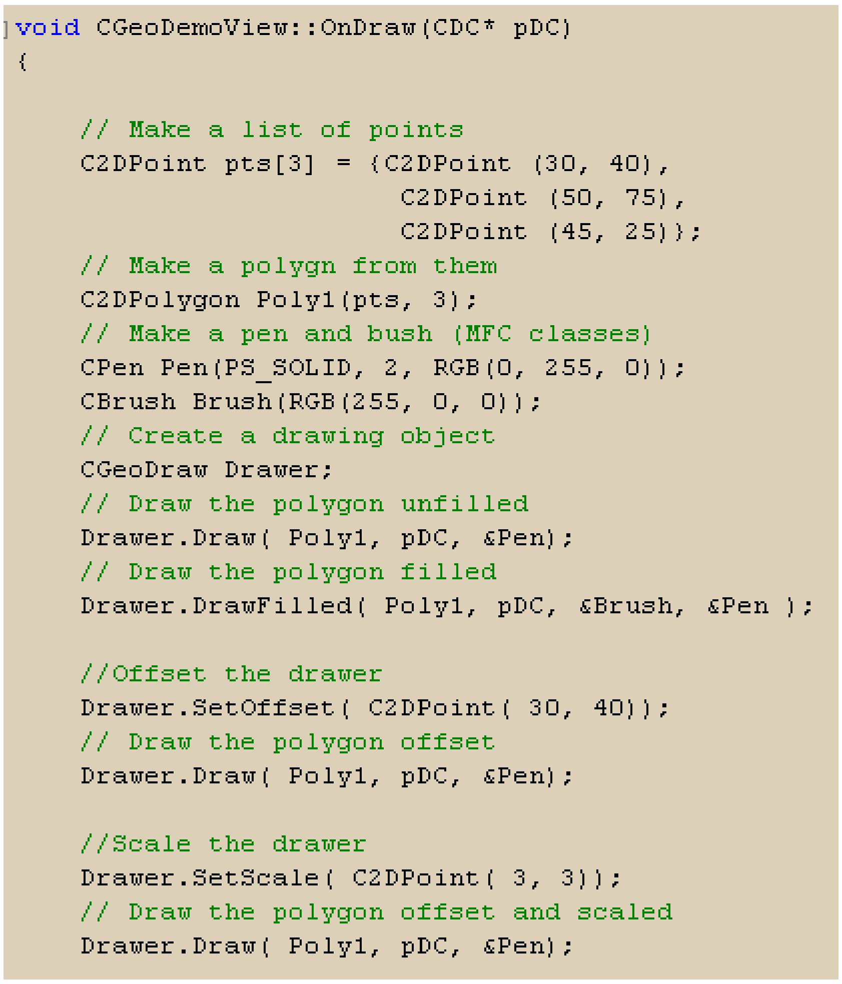
Basic Use

There are 2 drawing functions in the CGeoDraw class – Draw and DrawFilled. The first draws the outline of the shape only whilst the second fills the shape and draws the outline. Both drawing functions takes a reference to the shape to be drawn, a pointer to the CDC and a pen. The DrawFilled function also requires a CBrush object.

The CGeoDraw class defined object also has an offset and a scale which can be set to allow shape that would be drawn off the screen to be positioned as required. All shapes are offset first and then scaled.

A second class is provided, the CScreenManager, to help calculate the offset and scale required.

The most common and advisable way of drawing these items will be through the "On Draw" virtual function in a typical class inherited from the CView class. A simple example is given below.

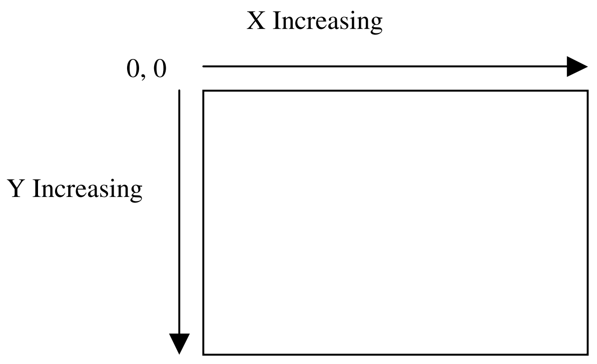


```
void CGeoDemoView::OnDraw(CDC* pDC)
{

    // Make a list of points
    C2DPoint pts[3] = {C2DPoint (30, 40),
                       C2DPoint (50, 75),
                       C2DPoint (45, 25)};
    // Make a polygn from them
    C2DPolygon Poly1(pts, 3);
    // Make a pen and bush (MFC classes)
    CPen Pen(PS_SOLID, 2, RGB(0, 255, 0));
    CBrush Brush(RGB(255, 0, 0));
    // Create a drawing object
    CGeoDraw Drawer;
    // Draw the polygon unfilled
    Drawer.Draw( Poly1, pDC, &Pen);
    // Draw the polygon filled
    Drawer.DrawFilled( Poly1, pDC, &Brush, &Pen );

    //Offset the drawer
    Drawer.SetOffset( C2DPoint( 30, 40));
    // Draw the polygon offset
    Drawer.Draw( Poly1, pDC, &Pen);

    //Scale the drawer
    Drawer.SetScale( C2DPoint( 3, 3));
    // Draw the polygon offset and scaled
    Drawer.Draw( Poly1, pDC, &Pen);
```

When providing a scale and an offset to the drawing function, the offset is always calculated first and then the result is scaled. Providing an offset means that every point in the geometric entity has this taken from it. A scale multiplies the result by the x and y provided in the scale point. The drawing function is constant and as such leaves the object unchanged.

Window Considerations

When drawing within Windows, the rectangle provided is upside down when compared to a Cartesian coordinate system. In other words, the top of a window will be 0 height and the bottom will be whatever the height is and positive.



For this reason, it is common to make the y part of the scale point negative.

It is recommended that the drawing is double bufferred i.e. a comaptible CDC object is created and the drawing is done in memory before being copied to the main CDC provided in the draw frunction. This eliminates flickering.

Calculating a Scale and Offset

If the points used to create geometric entities e.g. polygons are gained through user input to the screen e.g. through mouse clicks, the drawing is easy because no offset or scale is required. However, there are many cases when the objects to be drawn have external values and need to be manipulated to fit on a sceen. For example, consider a set of polygons representing postal codes with units in metres. In order to draw them onto the screen its likely they need to be scaled down to fit on the screen. If there are negative values for some points it is certain we need an offset as there are no negative pixels in the drawing rectangle in Windows. This is all complicated by the fact that we need to flip them all over for Windows. Drawing in this way is made easy through the CScreenManager class which calculates the values you need.

CScreenManager

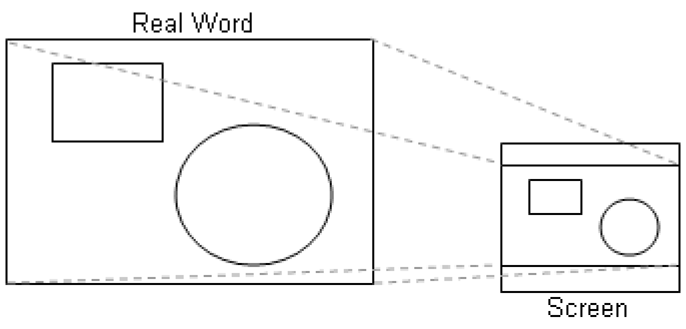
This class calculates the scale and offset required to display geometric shapes on the computer screen in the correct position. All that is required is the “Real World” rectangle to be drawn and the “Windows” rectangle that is to be drawn on. You decide the real world rectangle which is defined by the shapes you have. You can call the get bounding rectangle function to find that which bounds your entire set. The windows rectangle can be found through the MFC CWnd::GetClientRect function.

The Windows rectangle is provided as a CRect MFC object whilst the real world rectangle is a C2DRect which is a GeoLib object.

Once these 2 have been provided, the Calculate function is called and then the offset and scaled can be accessed through GetOffset and GetScale.

The screen manager will, by default, be set to calculate a negative y scale so that the image is correctly displayed for windows. This can be changed through the “SetYFlip” function.

The screen manager will calculate an offset and scale such that the real world rectangle is entirely within the windows rectangle when drawn. The aspect ratio of the real world rectangle will be maintained so that there will inevitably be a buffer, either above and below it or to the sides as illustrated.



The real world rectangle represented by the entire screen area is also calculated and can be accessed through the GetRealWindowsRect function. In other worlds, this is the real world rectangle that would fill the whole of the screen area including the buffers.

There are a number of other functions to help zoom in and out and scroll the screen and also to map a point on the screen to that in the real world.

The following is an example.

```
// Make a polygon
C2DPolygon Poly1(pts, 3);
// Create a drawing object
CGeoDraw Drawer;
// Create a screen manager
CScreenManager ScreenManager;
// Find the screen rectangle
CRect WindowsRect;
GetClientRect( &WindowsRect);
// Find the real world rect
C2DRect RealWorldRect = Poly1.GetBoundingRect();
// Initialise the screen manager
ScreenManager.SetRealRect( RealWorldRect );
ScreenManager.SetWindowsRect( WindowsRect );
ScreenManager.Calculate();
// Draw the shape
Drawer.SetOffset( ScreenManager.GetOffset());
Drawer.SetScale( ScreenManager.GetScale());
Drawer.Draw( Poly1, pDC, &Pen);
```