EE 356 **August 29, 2014**
Notes on Graphics

Pixel – A pixel is a single dot on a screen. Since 1985 when IBM introduced the Video Graphics Array pixels have been mostly square although there may be some displays which have rectangular pixels producing a very different horizontal and vertical resolutions. Printers typically have 1.5 to 2 times as many pixels in the horizontal direction as in the vertical direction and printer pixels are not square.

Resolution – For a CRT or LCD display the resolution refers to the number of pixels per inch that are displayed on the screen. Typical LCD screens today have a resolution of about 100 pixels per inch. This number is usually expressed as the dot pitch or the pixel pitch and is given as the number of millimeters between dots of the same color. For example a dot pitch of 0.337 would be 0.337 mm between pixels which corresponds to 75.3 dots/inch. Resolution is also quoted as the total number of horizontal pixels x the total number of vertical pixels on a single screen. For example, 1024 x 768 means that the screen has 1,024 horizontal pixels and 768 vertical pixels. For a printer, the resolution refers to the number of ink dots per inch than can be placed on the paper. A typical low-end printer might have a resolution of 300 dots/inch where a high end printer may have a resolution of twice that or 600 dots/inch. A very clear magazine photo might have a resolution of 2,400 dots/inch and a picture in a newspaper will typically have a resolution of 150 to 300 dots/inch.

Note that Windows does not know the size of your monitor so it deals with the number of pixels instead of the number of pixels/inch. To complicate matters further, fonts are typically measured in point sizes where a point is 1/72 inch. For example, a 10-point font will have a height of 10/72 inch in print.

A user cannot directly write to the screen. If you could, your application would be different for each monitor type and resolution. The monitor is driven by a display adapter. This is some logic that is added to a basic computer system which allows connection of the computer and the display monitor. This logic typically contains some video memory where it keeps an image of what is on the screen. It also has all of the electronics necessary to take the data from the memory and put it on the screen at the right point in time. Some screens need to be constantly refreshed and the display adapter takes care of this. The Display adapter is driven by a *device driver* which is installed in the operating system. The device driver serves as a particular interface between the operating system and the Display Adapter. The device driver is all in software and it interfaces to the Graphics Device Interface, or GDI, which is part of the operating system.

In C# there is a Graphics class which provides the interface between say a form and the GDI. Your C# application works entirely with the Graphics object and as a C# developer you can write applications that are mostly independent of what monitor they are used with.

Graphics object → Graphics Device Interface (GDI) in op sys → Display adapter → Monitor

When using Windows Forms we typically create a graphics object and draw our graphics on that object. The object might be a panel for example. We then provide a *Paint* event which takes care of maintaining our graphics on the screen after it has been minimized and restored. This system of doing graphics is called *immediate-mode graphics*. The programmer takes care of all of the window maintenance by providing refresh code as needed.

In WPF, the model has changed to *retained mode graphics*. In this model, you create a graphics object and all of the maintenance of that object is taken care of for you. That is, the object is *retained* for the life of the application.

The term *rendering* is the process of creating an image on the screen from a model by means of a computer program. The model may be a file or, in our case, it can be a set of equations implemented in C# or XAML. In WPF there are three ways to do rendering. These are called *Shapes, Drawings and Geometries,* and *Visuals.* All three of these have a similar base and if you learn to use one it's easy to go to another. For this class we are going to concentrate on Visuals. Visuals are the fastest rendering technique and are best for complex drawings.

To create graphics on the screen we will place an image on the screen from the toolbox and size it to fit our needs. We will then create a bitmap which is the same size as the image. WPF provides tools for creating the bitmap and placing our graphics in bitmap format. Finally, when the bitmap is complete we will *render* the bitmap to the image with a single statement.

Bitmaps are particularly useful when the graphics you are creating is time consuming to compute. If you create a bitmap, modify it, and render it to an image, you can often do so fast enough to do animation.

Coordinate Systems

When we do graphics on a computer we typically are forced to deal with at least two coordinate systems: *screen coordinates* and *world coordinates*. Screen coordinates are the coordinates on the screen. Historically, screen coordinates have developed such that the origin is at the top left corner of the screen. The x-axis is the horizontal axis and increases from left to right as is conventional in mathematics. The y-axis is vertical but increases as you go down the screen. Screen coordinates are numbered in pixels.

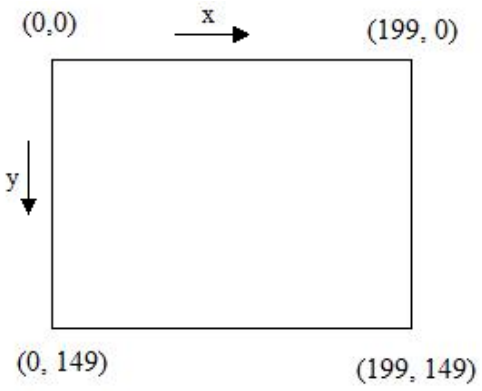

Figure 1

Screen coordinate system. This panel is 200 pixels wide by 150 pixels tall.

The world coordinates are the coordinates you are working in for the problem you are graphing. For example, if you want to graph a sinusoid that goes from -2π to $+2\pi$ in x and from -1.5 to +1.5 in y then your world coordinate system has a width of 4π and a height of 3 and the center point is in the center of the coordinate system.

To calculate the points to be plotted we work in the world coordinate system but to actually put something on the screen we must transform the world coordinates to screen coordinates. As an example, suppose we want to map the world coordinates for a sine wave which goes from -2π to $+2\pi$ and from -1.5 to + 1.5 to a screen coordinate system that goes from 0 to 199 in x and 0 to 149 in y. We want to map an arbitrary point (x_{world}, y_{world}) to the screen at (x_{screen}, y_{screen}). The entire width of the world coordinate system will get mapped into the entire width of the screen coordinate system. We can write the following proportion:

$$\frac{x_{world}}{x_{screen}} = \frac{\text{world width}}{\text{screen width}}$$

Since we know x_{world}, the world width, and the screen width, we can solve this for x_{screen}.

$$x_{screen} = \frac{\text{screen width}}{\text{world width}} x_{world}$$

However, this transformation from world to screen does not account for the differences in where the zero location is. Zero in the world is mid-screen and zero for the screen is on the far left. We must therefore add in an offset to account for this difference.

$$x_{screen} = \frac{\text{screen width}}{\text{world width}} x_{world} + \frac{\text{screen width}}{2}$$

To check this transformation to see that it works we use it to map some world coordinates to screen coordinates. Take screen width = 200 and world width = 4π.
We see that

$x_{world} = 0$ maps to screen width/2 or $x_{screen} = 100$.
$x_{world} = -2\pi$ maps to –(screen width)/2 + (screen width)/2 or $x_{screen} = 0$.
$x_{world} = +2\pi$ maps to (screen width)/2 + (screen width)/2 or $x_{screen} = 200$

In general, transformation equations between the world coordinates and the screen coordinates take the form of
x_{screen} = (proportionality factor) (x_{world}) + bias

In the same way we can develop a transformation equation for the y coordinates. This comes out to be

$$y_{screen} = \frac{-\text{screen height}}{\text{world height}} y_{world} + \frac{\text{screem height}}{2}$$

The minus sign is necessary because the world coordinates in y increase in the upward direction and the screen coordinates in y increase in the downward direction.

The following program illustrates how to do a graphics application in WPF. It plots a sinusoid with an x and a y axis and writes text to the screen.

```
private double[,] data = new double[600, 2];     //Data to be plotted
private double xRange, yRange, xOffset, yOffset;
private double winWidth, winHeight;

/// <summary>
/// Creates a sinusoid and plot it to the screen by way of a
///   visual and a bitmap
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void btnPlot_Click(object sender, RoutedEventArgs e)
  {int i;
   double x, y, xIncr, f;
   double xOld, yOld;
   FormattedText fmtxt;
   winWidth = imgPlot.Width;winHeight = imgPlot.Height;
   Pen blkPen = new Pen(Brushes.Black, 1);
   Brush bluBrush = Brushes.AliceBlue;
   //Create a "visual" to draw on.
   DrawingVisual vis = new DrawingVisual();
   //Create a drawing contest for this visual
   DrawingContext dc = vis.RenderOpen();
   f = 1;
   xRange = 12;yRange = 4;
   //Initialize x and y in world coordinates
   x = -xRange / 2;
   y = Math.Sin(2 * Math.PI * f * x);
   xIncr = xRange / (data.Length / 2);
   //Load the data array
   for (i = 0; i < data.Length / 2; i++)
      {data[i, 0] = x;
       data[i, 1] = Math.Sin(2 * Math.PI * f * x);
       x = x + xIncr;
      }
   xOffset = winWidth/2+10; yOffset = winHeight/2 - 10;
   World2Screen(data[0, 0], data[0, 1], out xOld, out yOld);
   //Convert coordinates to screen and draw lines between points
   for (i = 1; i < data.Length / 2; i++)
      {World2Screen(data[i, 0], data[i, 1], out x, out y);
       dc.DrawLine(blkPen, new Point(xOld, yOld), new Point(x, y));
       xOld = x; yOld = y;
      }
   //Draw x and y axis
   dc.DrawLine(blkPen, new Point(xOffset, 0), new Point(xOffset, winHeight));
   dc.DrawLine(blkPen, new Point(0, yOffset), new Point(winWidth, yOffset));
   //Just for fun write text to screen
   fmtxt = GetFormattedText("Hello Mom!", 14);
   dc.DrawText(fmtxt, new Point(10, 10));
   dc.Close();
```

```
    //Create a bit map
    RenderTargetBitmap bmp = new RenderTargetBitmap(490, 260, 96, 96,
                                                    PixelFormats.Pbgra32);
    //Render the visual to the bitmap
    bmp.Render(vis);
    //Make the image source the bitmap
    imgPlot.Source = bmp;
   }

/// <summary>
/// Changes world coordinates to screen coordinates
/// </summary>
/// <param name="xWld"></param>
/// <param name="yWld"></param>
/// <param name="xScr"></param>
/// <param name="yScr"></param>
private void World2Screen(double xWld, double yWld,
                    out double xScr, out double yScr)
    {xScr = xWld * winWidth / xRange + xOffset;
     yScr = -yWld * winHeight / yRange + yOffset;
    }

/// <summary>
/// Creates a formatted string of text
/// </summary>
/// <param name="sTmp"></param>
/// <param name="typeSize"></param>
/// <returns></returns>
private FormattedText GetFormattedText(string sTmp, int typeSize)
    {FormattedText fmtxt;
     fmtxt = new FormattedText(sTmp,
                 System.Globalization.CultureInfo.CurrentCulture,
     FlowDirection.LeftToRight,  new Typeface("Times New Roman"), typeSize,
                                 System.Windows.Media.Brushes.Black);
     return fmtxt;
    }
```