

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 that 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

The graphics object is different if you are doing graphics on a form or if you are doing graphics on a panel or a button. You must obtain a different graphics object for each item on which you wish to draw your graphics.

There are three ways to obtain a graphics object:

1. You can get a graphics object from the paint event for any control. For example, a button has a paint event and you can obtain a graphics object and use it to draw on the button like this:

```
private void btnGraphics_Paint(object sender, PaintEventArgs e)
{Graphics grfx = e.Graphics;
  Pen bluPen = new Pen(Color.Blue);
  grfx.DrawLine(bluPen, 0, 0, 100, 100);
}
```

2. You don't need to have a paint event to create a graphics object. You can obtain one at anypoint in a program by using CreateGraphics like this:

```
private void btnGraphics_Click(object sender, EventArgs e)
{Pen redPen = new Pen(Color.Red);
  Graphics grfx = btnGraphics.CreateGraphics();
  grfx.DrawLine(redPen, 0, 0, 160, 61);
}
```

This example creates a graphics object for the button inside the click event. You could just as well create a graphics object for any other control inside this event.

3. You can create a bitmap image and then create a graphics object from the bitmap. C# provides a bitmap class to make this happen. Here's an example:

```
Pen grnPen = new Pen(Color.Green);           //Green pen
Bitmap btmp = new Bitmap(260, 130);          //Create bitmap
Graphics g = Graphics.FromImage(btmp);      //Get graphics object from bitmap
g.DrawLine(grnPen, 0, 0, 260, 130);         //Draw on bitmap
Graphics gp = pnlGraphics.CreateGraphics(); //Get graphics object for panel
gp.DrawImage(btmp, 0,0);                     //Copy bit map to the panel
```

In most cases you will want to create a graphics object in the paint event. If you don't then you run the risk that your graphics will dissappear when the control is repainted such as when the user minimizes and then restores the application.

Bitmaps are used when the graphics you are creating is time consuming to compute. If you create this kind of graphics in the paint event then the process of repainting the control will become painfully slow. If you create a bitmap elsewhere on a bitmap and use the paint event to copy the bitmap to the control graphics it will become much faster and you can often eliminate flicker.

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.

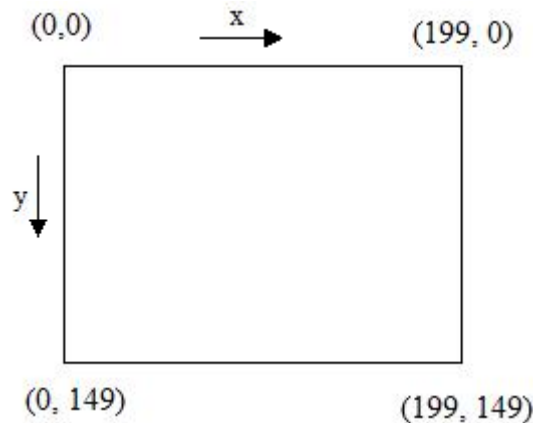


Figure 1

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 $(x_{\text{world}}, y_{\text{world}})$ to the screen at $(x_{\text{screen}}, y_{\text{screen}})$. 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:

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

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

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

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.

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

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

$$x_{\text{world}} = 0 \text{ maps to screen width}/2 \text{ or } x_{\text{screen}} = 100.$$

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

$$x_{\text{world}} = +2\pi \text{ maps to } (\text{screen width})/2 + (\text{screen width})/2 \text{ or } x_{\text{screen}} = 200$$

In general, transformation equations between the world coordinates and the screen coordinates take the form of

$x_{screen} = (\text{proportionality factor}) (x_{world}) + \text{bias}$

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

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

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.

We can draw directly onto a form (or any control) but a form typically has a title bar and may have menus and other items that we would have to draw around or at least be aware of. Instead, it is more convenient to draw on a panel. Since a panel has its own coordinate system, we are free to move the panel anywhere on the form without changing the coordinates of the underlying graphics.