EE 356
Notes on Pollock Paint

Figures 1 shows two typical Jackson Pollock paintings. Notice that they consists of paint splatters, loops that resemble a portion of an ellipse, and occasionally a figure eight. These can all be simulated mathematically.

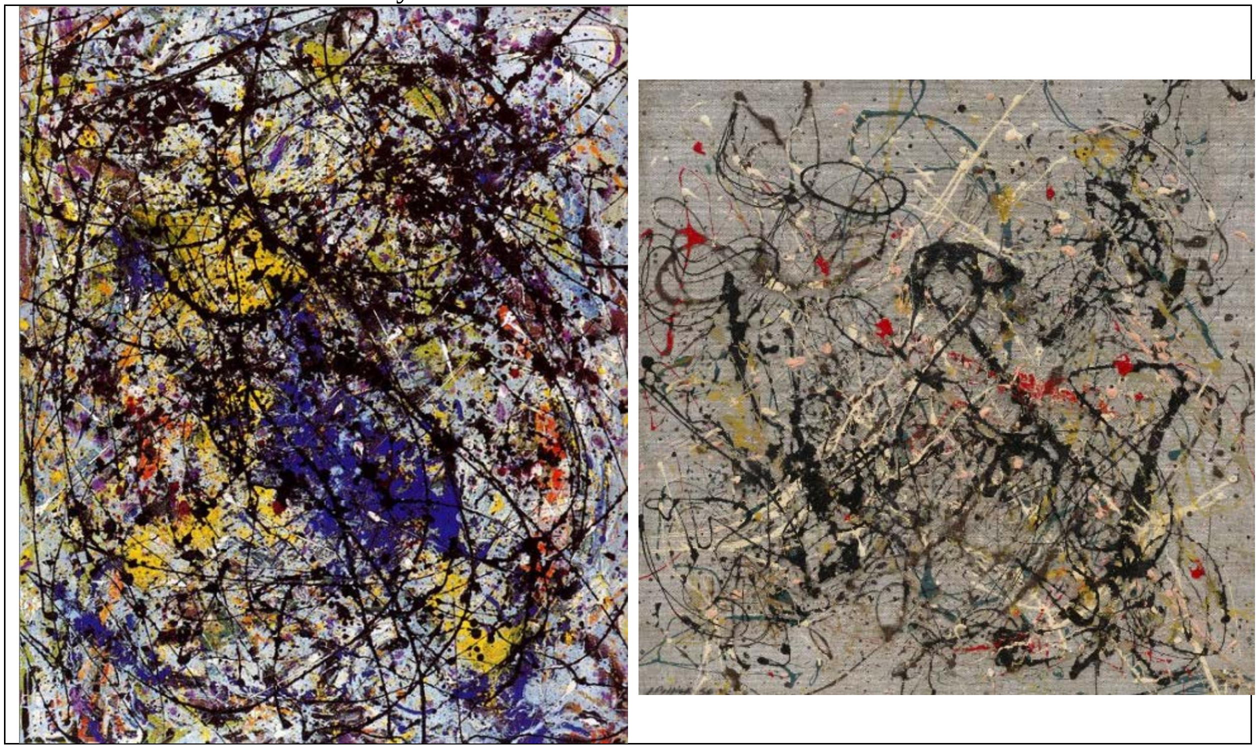

Figure 1
Jackson Pollock painting titled "Reflections of the Big Dipper" (left) and "Number 18" (right).

Ellipse
The equation for an ellipse in parametric form is given by:

$$x = h + r_1 \cos(t - \theta)$$

$$y = k + r_2 \sin(t)$$

The center of the ellipse is located at (h, k), 2r_1 is the length of the x-axis and 2r_2 is the length of the y-axis. The variable t is the parameter and if it goes from -2π to +2π, you get a full ellipse. The variable θ allows you to tilt the x-axis. Figures 2 to 4 show the effects of changing the values of r_2, theta, and t.

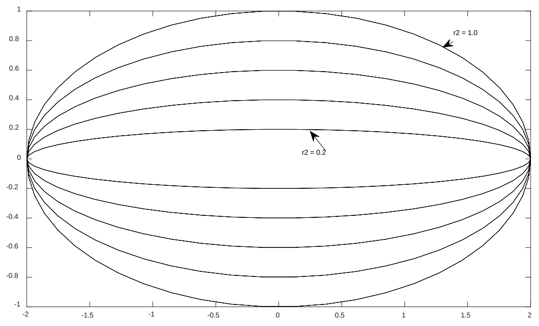

Figure 2

This figure shows the effect of changing r_2.
h = k = theta = 0, t = -2π to +2π, r1 = 2, r2 goes from 0.2 to 1.0 in steps of 0.2

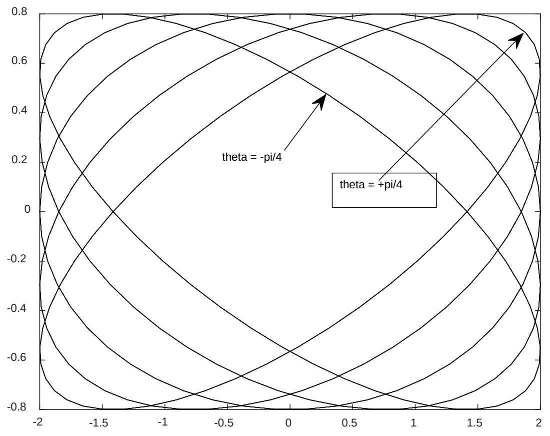

Figure 3

This figure shows the effect of changing θ.
h = k = 0, t = -2π to +2π, r1 = 2, r2 = 0.8 and θ goes from –π/4 to +π/4 in steps of π/8.

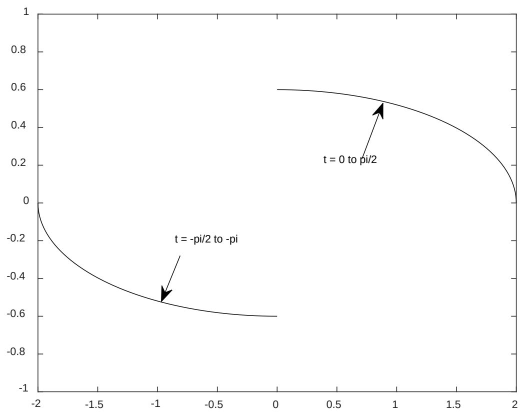

Figure 4

This figure shows the effect of changing θ.
h = k = theta = 0, r1 = 2, r2 = 0.6 and t changes as shown in steps of pi/100

Paint Splatters

Paint splatters can be made by drawing small filled ellipse or circle figures of random sizes at random locations. You can use the Random class in C# to generate random integers like this:
```
private Random ru = new Random();
```
After creating a variable of the Random class you create pseudo-random numbers like this:
```
  int iRand;
  iRand = ru.Next(1, 7); //random ints 1 <= iRand < 7
```
or,
```
  double dRand;
  dRand = ru.NextDouble(); //random double 0 to 1
```
The Random class however, creates only pseudo-random numbers which have a uniform distribution. If you want another distribution you need to create that yourself in code. There are two easy ways to create a *normal* distribution: A) Use sums and B) use the Box-Muller method to approximate the normal distribution.

To use sums you simply add us *n* normally distributed numbers and divide by *n*. The result will be an approximately normally distributed number.
```
double sum = 0, num = 12, xNormal;
int j;
for(j=0;j<num;j++)
    sum += ru.NextDouble();
xNormal = sum/num;
```
If *num* = 1 you get a uniform distribution. As *num* gets larger you get a tighter distribution. Setting *num* to 12 is a reasonable number to use.

The Box-Muller method relies on a mathematical approximation to get a random distribution from a uniform distribution. It can be implemented like this:
```
double u1, u2, randStdNormal;
u1 = 1.0-ru.NextDouble(); //uniform(0,1] random doubles
u2 = 1.0-ru.NextDouble();
randStdNormal = Math.Sqrt(-2.0 * Math.Log(u1)) *
                       Math.Sin(2.0 * Math.PI * u2); //random normal(0,1)
x = mean + stdDev * randStdNormal;
```

Setting the mean to 0 and the stdDev to 0.2 gives a reasonable distribution. See:
http://stackoverflow.com/questions/218060/random-gaussian-variables

To draw paint blobs on the screen you need to do two things: A) Create an array of points where the paint blobs will be located and B) draw a filled in ellipse at each point. You can use either the sums method or the Box-Muller method to create the array of randomly distributed points. Note that you will have to rescale the random array to fit your image size. You can draw an ellipse on the screen using the DrawEllipse method. For example if the drawing context is *dc*.
```
   dc.DrawEllipse(bluBrush, null, new Point(x, y), radX, radY);
```
The ellipse will be filled in with the bluBrush and the second parameter is null so there will be no outline around the ellipse. You can put a pen color here is you want an outline. The Point is the ellipse center and radX, radY is the major and minor ellipse radius.

Figure 5 shows a paint splatter pattern using the Box-Muller normal distribution and small ellipses of size 2 to 7 pixels.

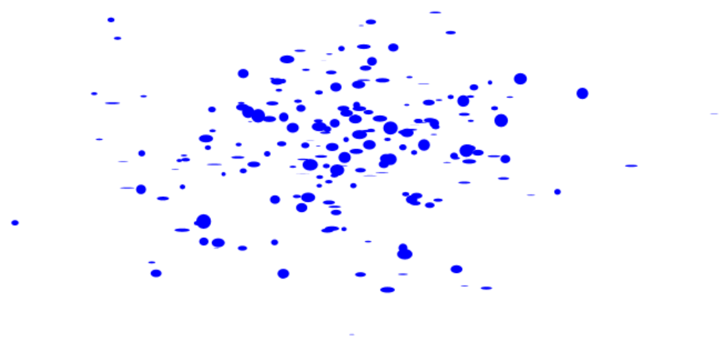

Figure 5
Two-hundred Paint spots in a normal distribution in the screen center.

Lines and Line Thickness
You can draw a line with the DrawLine command like this:
```
Pen bluPen = new Pen(Brushes.Blue, 1);
dc.DrawLine(bluPen, new Point(x1, y1), new Point(x2, y2));
```
where x1, y1 and x2, y2 are the end points of the line and the bluPen gives the color of the line and its thickness, in this case a blue line with a thickness of 1. But thickness is a double and can be changed gradually to make a line that appears to change thickness as it gets longer. The following code fragment draws a red line in 10 segments. The first line segment has a thickness of 0.5 and each succeeding line segments gets one unit thicker. The array linePts is a double array which holds the end points of each line segment.
```
  thick = 0.5;
  for(i=0;i<9;i++)
     {x1 = linePts[i,0];y1 = linePts[i,1];
      x2 = linePts[i+1,0];y2 = linePts[i+1, 1];
      Pen nPen = new Pen(Brushes.Red, thick);
      dc.DrawLine(nPen, new Point(x1, y1), new Point(x2, y2));
      thick += 1;
     }
```
Figure 6 shows the result.

The same general idea can be applied to other drawings which can be formulated as a sequence of line segments. Recall from the discussion of the ellipse that the parametric equations are given by:

$$x = h + r_1 \cos(t - \theta)$$

$$y = k + r_2 \sin(t)$$

Where the ellipse has a center at (h, k) and the x-axis is of length $2r_1$ and the y-axis is of length $2r_2$.

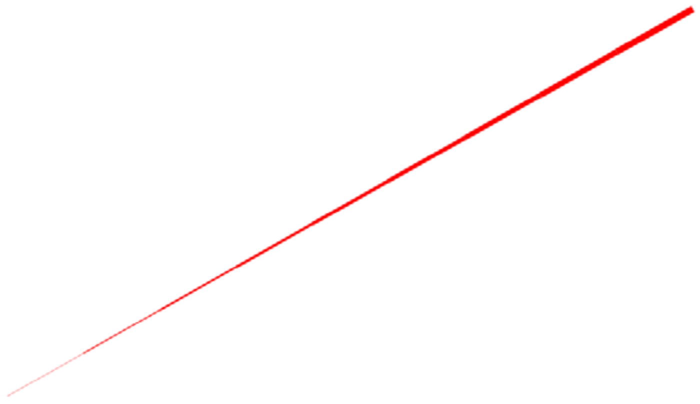

Figure 6
This line was broken into ten segments and each segment was drawn separately with a different thickness.

The function name GetPointOnEllipse below fills an array of doubles with the points on an ellipse segment using the parmetric equations above. The array has 20 points so the variable t goes from 0 to π/2.

```
        private void GetPointOnEllipse(double [,] ellPts)
            {double r1, r2, theta = 0, t;
             double h, k;
             int i = 0;
             r1 = 300;r2 = 60;
             h = winWidth/2.0;k = winHeight/2.0;
             t = 0;
             for(i=0;i<ellPts.GetLength(0);i++)
                {ellPts[i, 0] = h + r1*Math.Cos(t - theta);
                 ellPts[i, 1] = k + r2*Math.Sin(t);
                 t += Math.PI/40;
                }
            }
```

Figure 7 shows a plot of the data when the line thickness between the 20 segments increases in steps of 2 unit from 0.5.

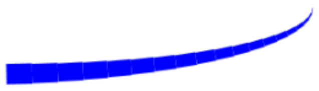

Figure 7
A portion of an ellipse drawn in line segments where the thickness of each segment gradually increases.

There are many other equations for curves that might be suitable for a Pollock painting. For example see:
http://jwilson.coe.uga.edu/EMAT6680Fa08/Broderick/assignment10/assignment10.html