9.4　色々な曲線／フラクタル

(1)スプライン曲線とベジェ曲線

C#では，Graphics クラスの DrawCurve メソッドや DrawBezier メソッドを使うとスプライン曲線(注)や，ベジェ曲線を描くことができます。スプライン曲線では，閉曲線も描くことができます。これらを C#で描いた例を図 9-9 に示します。

さらにメソッドにテンションを引数で受け渡すことで，歪曲度を変更することができます。テンションを連続的に変化させ，テンションの値によって線の色を変えた図を図 9-9(c)に示します。

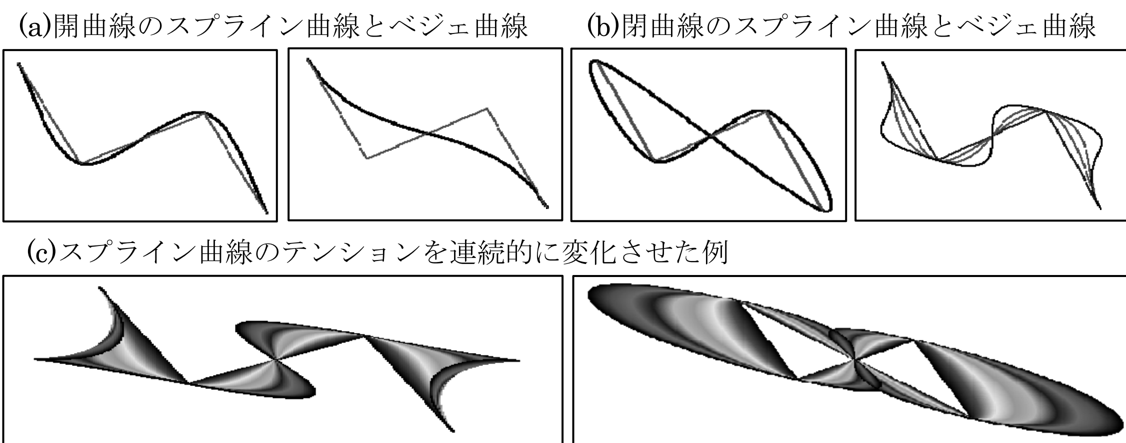


図 9-9　C#によるスプライン曲線とベジェ曲線

2 次元の絵を描きますので，以下の using を追加します。

```
using System.Drawing.Drawing2D;
```

以下の説明では，次のようにデータ宣言されているものとします。

```
private Point[] 構成点={  new Point(200,100), new Point(250,200),
                          new Point(350,150), new Point(400,250)};
private Pen pen1 = new Pen(Color.Black,2);
```

(注) スプライン曲線による補間法については，拙著「Excel と VBA による実用数値解析入門」に詳説しています。手法について知りたい方は同書を参照して下さい。

また，以下の説明中の PaintEventArgs は，次の例に示すオーバライドされた OnPaint から引き渡されるものとします。

```
protected override void OnPaint(PaintEventArgs e )
{ switch (曲線)
  { case "ベジェ曲線"        : ベジェ曲線(e)         ;break;
    case "スプライン曲線"    : スプライン曲線(e)     ;break;
      :
      :
  }
}
```

【ベジェ曲線を描く】DrawBezier メソッドの使用

```
private void ベジェ曲線(PaintEventArgs e )
{ base.OnPaint(e);
  e.Graphics.DrawBezier(pen1, 構成点[0], 構成点[1],
                              構成点[2], 構成点[3]);
}
```

【スプライン曲線を描く】 DrawCurve メソッドの使用

```
private void スプライン曲線(PaintEventArgs e )
{ base.OnPaint(e);
  e.Graphics.DrawCurve(pen1,構成点);
}
```

【スプライン閉曲線を描く】DrawClosedCurve メソッドの使用

```
private void 閉曲線(PaintEventArgs e )
{ base.OnPaint(e);
  e.Graphics.DrawClosedCurve(pen1,構成点);
}
```

【テンションの指定】
DrawCurve メソッド，DrawCurve メソッド共に第 3 引数で指定します。

```
  e.Graphics.DrawCurve(pen1, 構成点, 0.5F);
  e.Graphics.DrawClosedCurve(pen1,構成点,1.0F,FillMode.Alternate);
```

【テンションを連続的に変化させながら閉図形を描く】

```
  for(int i=0;i<255;i++)
  {  Pen pen =new Pen(Color.FromArgb(i,0,0));
     e.Graphics.DrawClosedCurve(pen, 構成点,
                      (float)i * 0.5F / 255F, FillMode.Alternate);
  }
```

（2）Cカーブ

現在位置から相対座標に至る C 曲線を描くプログラムです。最大長(maxLength)を小さくすると，より複雑な曲線が得られます。

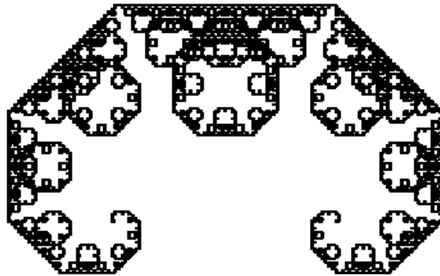


図 9-10　C カーブの例

以下に示すプログラムは，maxLength，X，Y，lineWidth を設定し，次のようにして呼び出します。

```
moveAbsolute(X,Y); // 画面上に描画する位置を指定する
C_Curve(500,0);    // 最初移動する相対位置を指定する
this.Invalidate(); // 再描画
```

■プログラム例

```
private Image image;
private Point currentPoint = new Point();
private Point nextPoint    = new Point();
private double maxLength   = 2; private int lineWidth = 1;

protected override void OnPaint(PaintEventArgs e )  // オーバライド
{ base.OnPaint(e); e.Graphics.DrawImage(image,0,0);}

private void drawRelative(int X, int Y)   // 相対位置による描画
{ nextPoint.X = currentPoint.X + X; nextPoint.Y = currentPoint.Y + Y;
  Graphics g  = Graphics.FromImage(image);
  Pen pen     = new Pen(Color.Black,lineWidth);
  g.DrawLine(pen,currentPoint,nextPoint);
  currentPoint = nextPoint;
}

private void moveAbsolute(int X, int Y)   // 絶対位置に移動
{ currentPoint.X = X; currentPoint.Y = Y;  }

private void C_Curve(double X, double Y)  // C曲線
{ if(X＊X＋Y＊Y <= maxLength＊maxLength) drawRelative((int)X, (int)Y);
  else { C_Curve((X + Y) / 2,(Y - X) / 2);
         C_Curve((X - Y) / 2,(Y + X) / 2 );
        }
}
```

(3)コッホ曲線

線分を三等分し，中央の 1/3 を削除し，そこに 1/3 の長さの正三角形を挿入します。この操作を新たにできた辺に対して施します。これを繰り返して得られる曲線をコッホ曲線といいます(図 9-11 参照)。

図 9-11(b)に示すように，このような簡単な操作にも関わらず実に複雑な形を示しています。なお，ある部分をみると，そこが更に同じ形になっていますので，これを**自己相似形**と呼びます。

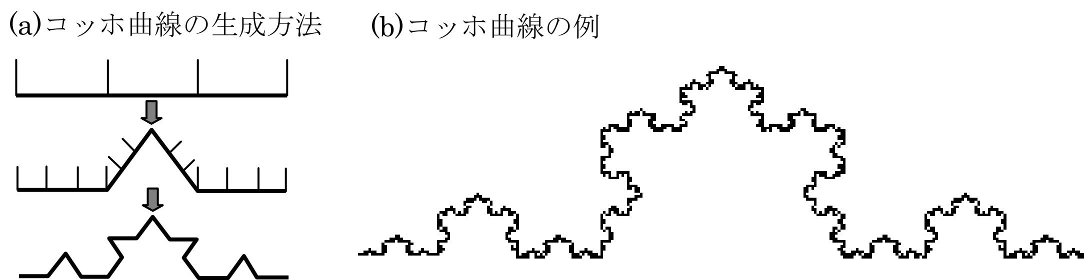


図 9-11　コッホ曲線の生成と例

■**プログラム例**

プログラム例を示す前に，共通的なメソッドを示します。

【追加する using】

```
using System.Drawing.Drawing2D;
```

【データ宣言】

```
public struct 位置データ{ public double X;  public double Y;  }
public 位置データ 現在位置 = new 位置データ();
public double 角度;
```

【共通処理】

```
public void t_position(double X, double Y) // 現在位置の設定
{ 現在位置.X=X;  現在位置.Y=Y;}

public void t_Degree(double Theta) // 角度の設定
{ 角度=Theta;  }

public void t_turn(double Theta) // 進行方向を変える
{ 角度 += Theta;  }

// 現在進行方向に指定された長さ分だけ進む
public void t_forward(PaintEventArgs e, double length)
{ double TH = 角度 * 3.1415926 / 180;
  double X  = 現在位置.X + length * Math.Cos(TH);
  double Y  = 現在位置.Y + length * Math.Sin(TH);
  Pen pen   = n ew Pen(Color.Black);
  e.Graphics.DrawLine(pen, (int)現在位置.X, 300 - (int)現在位置.Y,
                           (int)X, 300 - (int)Y);
現在位置.X=X; 現在位置.Y=Y;
}
```

【コッホ曲線を描く処理】

次のように行います。

① 現進行方向に 1/3 だけ進み 60 度方向を変え，
② その方向に 1/3 だけ進み－120 度方向を変える。
③ 更に 1/3 だけ進み 60 度方向を変える。

方向は，

60 度→ －120 度→ 60 度

と変わり，最後は最初の方向に戻ります。

```
public void koch( PaintEventArgs e,int n,double length)
{ if(n <= 0) t_forward(e,length);
  else
  { int nn = n - 1; double len = length / 3;
    koch(e, nn, len); t_turn(60); koch(e, nn, len); t_turn(-120);
    koch(e, nn, len); t_turn(60);
    koch(e, nn, len);
  }
}
```

(4)立ち枯れ木立

コッホ曲線では，長さを 1/3 にして，60 度向きを変え，3 回再帰的呼出しを行っていますが，これらの長さの比率，向き，呼出し回数を色々変えると様々な図形が得られます。たとえば，以下のように進行方向，比率を変えてみましょう。なお，向きの合計が 0 度になっていることを確認してください。

	1 回目	2 回目	3 回目	4 回目	合計
向き	0 度	88 度	−176 度	88 度	0 度
比率	0.28	0.28	0.28	0.7	

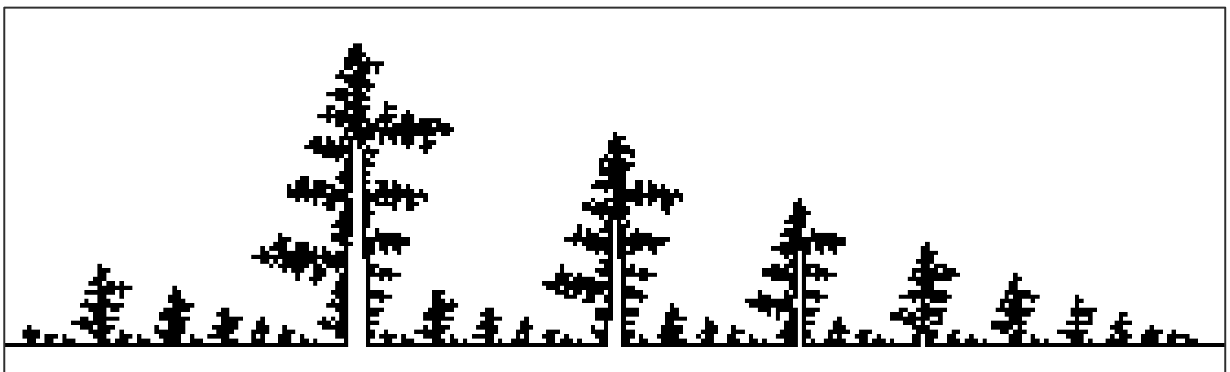
■プログラム例

データ宣言や共通処理は，コッホ曲線と同じです。

```
public double[,] element=new double[,]
     {{ 0.0, 0.28},{88.0, 0.28}, {-176.0, 0.28},{88.0, 0.70}};
public void frctl( PaintEventArgs e, double length)
{ int j;
  if (length <=2.0) t_forward(e,length);
  else { for(j = 0; j < 4; j++)
             { t_turn(element[j,0]);frctl(e, length*element[j,1]);}
       }
}
```

■実行例

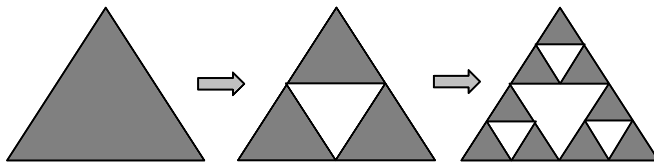
以下のような絵になりますので，立ち枯れ木立と呼ばれます。比率や方向を色々と変えて，色々な絵を生成してみるのも面白いかもしれません。



(5)シェルピンスキーのギャスケット

シェルピンスキーのギャスケットとは，三角形の中に１辺の長さ 1/2 の三角形を順次描くことによって得られます。これもコッホ曲線と同様，自己相似形を内部に持つ形になります。

(b) シェルピンスキーのギャスケット n の生成方法



(b) 例題プログラムで生成したシェルピンスキーのギャスケット

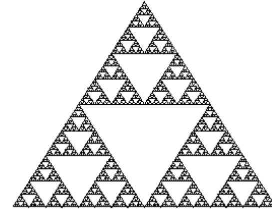


図 9-12　シェルピンスキーのギャスケットの生成と例

■プログラム例

データ宣言や共通処理は，コッホ曲線と同じです。実行例は，図 9-12(b)に示します。

```
public void 三角形(PaintEventArgs e,double X, double Y, double L)
{  Pen pen = new Pen(Color.Black);
   int IX1 = (int)X         ; int IX2 = (int)(X + L );
   int IX3 = (int)(X + L / 2);
   int IY1 = 300 - (int)Y;
   int IY2 = 300 - (int)(Y - L * Math.Sin(60 * 3.1415926 / 180));
   e.Graphics.DrawLine(pen, IX1, IY1, IX2, IY1);
   e.Graphics.DrawLine(pen, IX1, IY1, IX3, IY2);
   e.Graphics.DrawLine(pen, IX2, IY1, IX3, IY2);
}
public void sier( PaintEventArgs e,
                  int n, double X, double Y, double L)
{
   if(n == 0) return;
   double H = L * Math.Sin(60 * 3.1415926 / 180);
   三角形(e,    X + L / 4, Y + H / 2, L / 2);
   sier(e, n-1, X        , Y        , L / 2);
   sier(e, n-1, X + L / 2, Y        , L / 2);
   sier(e, n-1, X + L / 4, Y + H / 2, L / 2);
}
public void sier( PaintEventArgs e)
{
   三角形(e, 290.0, 10.0, -280.0);
   sier(e, 6, 10.0, 10.0, 280.0);
}
```

(6)反復関数系

自己相似図形があれば，その図形を構成する反復関数系を見つけ，各関数に確率を割り振って，乱数で得られた座標に点を打つことで，色々な図形を生成できます。もちろんシェルピンスキーのギャスケットも，この方法で表現できます。

ここでは，別の例を示しますので，シェルピンスキーのギャスケットについては，皆さんでチャレンジしてみてください。なお，フラクタル図形の教科書を参照するのもひとつの手段でしょう。ここでは，以下のような反復関数系を実行してみます。

【反復関数系】

$$(1)\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} 0.8560 & 0.0414 \\ -0.0205 & 0.8580 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} 0.070 \\ 0.147 \end{bmatrix}$$

$$(2)\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} 0.2440 & -0.385 \\ 0.1760 & 0.8224 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} 0.393 \\ 0.102 \end{bmatrix}$$

$$(3)\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} -0.144 & 0.3900 \\ 0.1810 & 0.2590 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} 0.527 \\ -0.014 \end{bmatrix}$$

$$(4)\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} 0. & 0. \\ 0.3550 & 0.2160 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} 0.486 \\ 0.05 \end{bmatrix}$$

【発生確率】

(1) 0.73,　(2) 0.13,　(3) 0.13,　(4) 0.01

【実行例】

■プログラム例

フォームに button1 を配置します。

【追加する using】

```
using System.Drawing.Drawing2D;
```

【データ宣言】

```
public double [,,] Fsys1=new double[,,]
      {{{ 0.856, 0.0414}, {-0.0205, 0.858}},
       {{ 0.244,-0.3850}, { 0.1760, 0.224}},
       {{-0.144, 0.3900}, { 0.1810, 0.259}},
       {{ 0.000, 0.0000}, { 0.3550, 0.216}}};
public double [,] Fsys2 = new double [,]
     {{0.07,0.147},{0.393,0.102},{0.527,-0.014},{0.486,0.05}};
public bool[,] 点    = new bool[1001,1001];
public Matrix matrix = new Matrix();
private Image image;
```

【描画処理】

```
public void 描画()
{  Brush brush = new SolidBrush(Color.DarkGreen);
   Graphics g  = Graphics.FromImage(image);
   g.Clear(Color.White);
   for(int i = 0; i < 1000; i++)
     for(int j = 0; j < 1000; j++)
       if(点[i,j]) g.FillRectangle(brush, i, (1000 - j), 1, 1);
}
protected override void OnPaint(PaintEventArgs e)
{  base.OnPaint(e);
   e.Graphics.Transform = matrix;
   e.Graphics.DrawImage(image, 100, 100);
}
```

【変換マトリックス設定】

```
private void window(float X1,float Y1,float X2, float Y2, float R)
{  float W  = ClientSize.Width ;
   float H  = ClientSize.Height;
   float SX = W / (X2 - X1)    ; float SY = H / (Y2 -Y1);
   float MX = - SX * X1        ; float MY = - SY * Y1   ;
   matrix.Scale(SX, SY)        ; matrix.Rotate(R)       ;
   matrix.Translate(MX, MY)    ;
}
```

【初期化】

```
private void 初期化()
{  for(int i= 0; i < 1001; i++)
        for(int j = 0; j < 1001; j++) 点[i, j] = false;
   描画();
}
private void Form1_Load(object sender, System.EventArgs e)
{  初期化(); window(0,0,1000,1000,0);}
```

【ボタン Click 処理】

```
private void button1_Click(object sender, System.EventArgs e)
{
    int ID;double X1,Y1;
    Random RD = new Random();
    double X0 = 1.0; double Y0 = 1.0;
    for(int i = 0; i < 20000; i++)
    {
        double R = RD.NextDouble();
        if      (R < 0.73) ID = 0;
        else if (R < 0.86) ID = 1;
        else if (R < 0.99) ID = 2;
        else               ID = 3;
        X1 = Fsys1[ID,0,0] * X0 + Fsys1[ID,0,1] * Y0 + Fsys2[ID,0];
        Y1 = Fsys1[ID,1,0] * X0 + Fsys1[ID,1,1] * Y0 + Fsys2[ID,1];
        if(X1 < 1.0 && Y1 < 1.0 && X1 >= 0 && Y1 >= 0)
            点[(int)(X1 * 1000.0), (int)(Y1 * 1000.0)] = true;
        X0 = X1; Y0 = Y1;
    }
    描画(); this.Invalidate();
}
```

色々な反復関数系を考えることができます。
興味ある方は，色々チャレンジしてみましょう。
ここでは２つの例を挙げておきます。

【反復関数系】

(1) $\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} 0.8 & 0.0 \\ 0.0 & 0.8 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} 0.10 \\ 0.04 \end{bmatrix}$

(2) $\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} 0.5 & 0.0 \\ 0.0 & 0.5 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} 0.25 \\ 0.40 \end{bmatrix}$

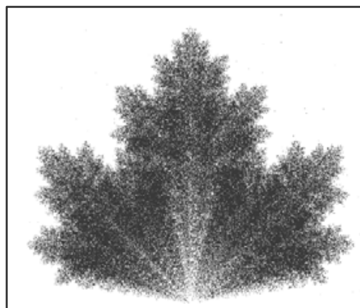
(3) $\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} 0.355 & -0.355 \\ 0.355 & 0.355 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} 0.266 \\ 0.078 \end{bmatrix}$

(4) $\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} 0.355 & 0.355 \\ -0.355 & 0.355 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} 0.378 \\ 0.434 \end{bmatrix}$

【発生確率】

(1) 0.5　(2)0.168　(3)0.166　(4)0.166

【実行例】



【反復関数系】

(1) $\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} 0.05 & 0.0 \\ 0.0 & 0.6 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} 0.0 \\ 0.0 \end{bmatrix}$

(2) $\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} 0.05 & 0.0 \\ 0.0 & -0.5 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} 0.0 \\ 1.0 \end{bmatrix}$

(3) $\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} 0.46 & -0.32 \\ 0.39 & 0.38 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} 0.0 \\ 0.6 \end{bmatrix}$

(4) $\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} 0.47 & -0.15 \\ 0.17 & 0.42 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} 0.0 \\ 1.1 \end{bmatrix}$

(5) $\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} 0.43 & 0.28 \\ -0.25 & 0.45 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} 0.0 \\ 1.0 \end{bmatrix}$

(6) $\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} 0.42 & 0.26 \\ -0.35 & 0.31 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} 0.0 \\ 0.7 \end{bmatrix}$

【発生確率】

(1) 0.1　(2)0.2　(3)0.2　(4)0.2　(5)0.2　(6)0.1

【実行例】