

-----  
Pseudocode illustrate method for generating EscapeTime fractals.  
built with 3-D points. Lang: Pascal. It's not a stand alone application.

All references in the code are to the following articles:

- J.C. Sprott, C.A. Pickover, *Automatic generation of general quadratic map basins*, Computers&Graphics, Vol. 19, No. 2 (1995), pp. 309-313
- Nikiel S., Goinski A., *Generation of volumetric quadratic map basins*, Computers&Graphics, Vol. 27 No. 6, pp. 977-982, 2003

-----

program fractals;

**const**

MaxColors = 80; { number of layers in 3D }

{ variables }

**var**

x, y, z : **Real**; { current point in fractal space }

a : **array** [0..29] of **Real**; { fractal code (fractal coefficients) }

fDimX, fDimY, fDimZ : **Real**; { dimensions of fractal space (e.g. = 8.0f) }

iSizeX, iSizeY, iSizeZ : **Integer**; { e.g. = 300, 3D dimensions of solid }

fScaleX, fScaleY, fScaleZ : **Real**; { scales of dimensions (e.g. = 4.0f) }

n : **Integer**; { a color/layer of a point }

iKernelColor : **Integer**; { a threshold color, in other words

it is the number of the first visible layer

of a fractal (e.g. = 5). Layers are numbered from 0 }

iMaxBound : **Integer**; { e.g. = 1000 }

fMaxSolution : **Real**; { e.g. = 1000000.0f }

-----

{ Calculate current solution for x, y and z. Increment.

-----

**procedure** AdvanceXY;

**var**

xnew, ynew : **Real**;

**begin**

xnew := a[0] + x\*a[1] + y\*a[2] + z\*a[3] +  
x\*x\*a[4] + y\*y\*a[5] + z\*z\*a[6] +  
x\*y\*a[7] + x\*z\*a[8] + y\*z\*a[9];

ynew := a[10] + x\*a[11] + y\*a[12] + z\*a[13] +  
x\*x\*a[14] + y\*y\*a[15] + z\*z\*a[16] +  
x\*y\*a[17] + x\*z\*a[18] + y\*z\*a[19];

z := a[20] + x\*a[21] + y\*a[22] + z\*a[23] +

```
x*x*a[24] + y*y*a[25] + z*z*a[26] +  
x*y*a[27] + x*z*a[28] + y*z*a[29];
```

```
x := xnew;  
y := ynew;  
n := n + 1;  
end;
```

```
{ ----- }  
{ Generates points of fractal. Display }  
{ ----- }
```

```
procedure GeneratePoints;
```

```
var
```

```
  k, i, j : Integer;  
  fScreenX, fScreenY, fScreenZ : Real;
```

```
begin
```

```
  fWidthDiv := iWidth / fScaleX;  
  fHeightDiv := iHeight / fScaleY;  
  fDepthDiv := iDepth / fScaleZ;
```

```
  for k:=0 to iSizeZ do
```

```
    for i:=0 to iSizeX do
```

```
      for j:=0 to iSizeY do
```

```
        begin
```

```
          x := fDimX + i / fWidthDiv;  
          y := fDimY - j / fHeightDiv;  
          z := fDimZ - k / fDepthDiv;
```

```
          n := 0;
```

```
          while (n<MaxColors) and ((x*x + y*y + z*z)<fMaxSolution) do  
            AdvanceXY;
```

```
          { render only one layer of fractal }
```

```
          if n = iKernelColor then
```

```
            begin
```

```
              { put point on screen in 3D space (centering it) }
```

```
              fScreenX := i - iSizeX * 0.5;
```

```
              fScreenY := j - iSizeY * 0.5;
```

```
              fScreenZ := k - iSizeZ * 0.5;
```

```
              PutPoint3D( fScreenX, fScreenY, fScreenZ );
```

```
            end;
```

```
          end;
```

```
  end;
```

```
{ ----- }  
{ Sets the parameters of a fractal }
```

```
{ (from -1.2 to 1.2) and prepares a fractal environment
{-----}
procedure SetParams;
var
  i : Integer;
begin
  x := 0;
  y := 0;
  z := 0;

  { randomly chosen parameters }
  for i:=0 to 30 do
    a[i] := ( random(25) - 12 )/10.0;
end;

{-----}
{ Test the solution. See Picover and Sprott's for details
{-----}
procedure TestSoln;
begin
  if n = iMaxBound then   { solution is bounded }
    n := 0;

  if ( x*x + y*y + z*z ) > fMaxSolution then { solution escaped }
    begin
      if n > 100 then
        n := iMaxBound
      else
        n := 0;
    end;
end;
end;

begin
  randomize;

  { set some initial values }
  iSizeX := 100; iSizeY := 100; iSizeZ := 100;
  fDimX := 8.0; fDimY := 8.0; fDimZ := 8.0;
  fScaleX := -4.0; fScaleY := 4.0; fScaleZ := 4.0;

  n := 0;
  iKernelColor := 6;
  iMaxBound := 1000;
  fMaxSolution := 1000000.0;

  { main loop }
  repeat
    if n = 0 then
      SetParams;
```

```
AdvanceXY;
TestSoln;

if n = iMaxBound then
begin
    GeneratePoints(); { draw fractal }
    n := 0;
end;
until keypressed;    { loop until a key is pressed }
end.
```