On Generating Skyscapes through Escape-Time Fractals

T.Gangopadhyay
XLRI
C.H.Area(E), Jamshedpur,
India

ABSTRACT
Scott Draves has used seven variations of affine transformations in his pioneering work on fractal flames. These are linear, sinusoidal, spherical, swirl, horseshoe, polar and bent. In this paper we use average of two such transformations on escape time fractals to produce skyscapes similar to expansion of galaxies.

General Terms
Fractal, Algorithm, Turbo C++, Program.

Keywords
Escape-time, Mandelbrot, IFS, Bailout, affine, sinusoidal

1. INTRODUCTION
This paper takes standard escape-time fractals and applies iterated function system (IFS) techniques to them. Usually in IFS generated fractals an arbitrary point is transformed repeatedly through multiple affine functions to produce fractal shapes. This paper's distinctive features consist of applying average of two nonlinear (often sinusoidal) functions to escape-time fractals using a bailout. As a result, every pixel is comes within the scope of the fractal. The effect is quite distinct from that of usual IFS fractals such as fern (Barnsley [1]), maple leaf (Barnsley [1]), dragon curves (Davis and Knuth [3]), Lissajous figures (Brill [2]) or those obtained through strange attractors (Hofstadter [5], Ruelle [8]).

2. THE ALGORITHM
In iterated function systems Michael Barnsley [1] used affine transformations repeatedly on a starting point to produce fractal shapes. Draves [4] extended the scope of these transformations further. He introduced seven functional variations of the original point. These are described below:

Let (x, y) be the coordinates of the original point. Let $r=\sqrt{x^2+y^2},\ s=r^2$ and $t=\arctan(y/x)$.

a) linear : $f(x, y)=(x, y)$.

b) sinusoidal : $f(x, y)=(\sin x, \sin y)$.

c) spherical : $f(x, y)=(x/s, y/s)$.

d) horseshoe : $f(x, y)=(r\cos(t+r), r\sin(t+r))$.

e) swirl : $f(x, y)=(r\cos(2t), r\sin(2t))$.

f) polar : $f(x, y)=(t/\pi, r-1)$

g) bent : $f(x, y)=(g(x), h(y))$

where

$$g(x)=x \quad \text{if} \quad x\geq 0$$
$$= c*x \text{ otherwise}$$

and

$$h(y)=y \quad \text{if} \quad y\geq 0$$
$$= y/c \text{ otherwise.}$$

In the present paper, we take standard escape time fractals such as Mandelbrot and Julia. For each complex variable z that is iterated, we first separate real(z) and imag(z). This separation is also present in the popcorn frctals developed by Pickover [7]. We apply two of the Draves' functions separately to real(z) and imag(z). In our notation:

$$ax = \mathrm{real}(z),\ ay = \mathrm{imag}(z),\ r = \sqrt{(ax)^2+(ay)^2},$$
$$s = r^2, t = \arctan(\frac{ay}{ax})$$

Then

1st function: $ax_1 = \sin(ax),\quad ay_1 = \sin(ay).$

2nd function: $ax_2 = \frac{ax}{s},\quad ay_2 = \frac{ay}{s}.$

3rd function: $ax_3 = r\cos(t+r),\ ay_3 = r\sin(t+r).$

4th function: $ax_4 = r\cos(2t),\quad ay_4 = r\sin(2t).$

5th function: $ax_5 = t/\pi,\ ay_5 = r-1.$

Suppose we have applied the third and fourth functions. Then during iteration we replace

z= (Real(z), imag(z)) by

$$z = \left(\left(\frac{ax3+ax4}{2}\right),\ \left(\frac{ay3+ay4}{2}\right)\right).$$

Using standard forms of bailout we colour each escaping pixel by iteration number (outside colouring) and each nonescaping pixel by BOF60 (inside colouring) (Peitzen [6]).

In the next section we submit a programme in Turbo C++.

3. THE CODE

We use the variable names standardized in Stevens [9].

```
void fractal(double,double,double,double,int,int);

main()

{

double xmax=-.8,xmin=-1.1,ymax=.150,ymin=-.09.
            …(1)

int max_iterations=56;int max_size= 4.0;            …(2)

fractal(xmax,xmin,ymax,ymin,max_iterations,max_size);

getch();

closegraph();

 }

double cabs(complex z)

{return sqrt(norm(z));}

void fractal(double xmax,double xmin,double ymax,double
ymin,int max_iterations,int max_size)

{complex c,z;    float zmin,index;

int color;

float col,row;

double deltap,deltaq;

deltap=(xmax-xmin)/640;

deltaq=(ymax-ymin)/480;

for(col=0;col<640;col++)

{for(row=0;row<480;row++)

{z=c=complex(xmin+col*deltap,ymax-row*deltaq);

color=0;   color=index=0;zmin=1000;

while((color<max_iterations)&&(norm(z)<max_size))

{ float ax=real(z),ay=imag(z);              …(3)

   float ax1,ay1,ax2,ay2,ax3,ay3,ax4,ay4,ax5,ay5;

 float r=sqrt(ax*ax+ay*ay),t=atan(ay/ax),s=r*r;

ax1=sin(ax); ay1=sin(ay); ax2=ax/s;ay2=ay/s;

ax3=r*cos(t+r);ay3=r*sin(t+r);
ax4=r*cos(2*t);ay4=r*sin(2*t);

ax5=t/3.14; ay5=r-1;


z=complex((ax4+ax3)/2,(ay4+ay3)/2);             …(4)
z=z*z+c;                                        …(5)

color++;

if( cabs((z)) < zmin)

  { zmin = cabs((z)) ;}

  }

  index=zmin  ;

if(color>=max_iterations)

          putpixel(col,row,index);

 else

          putpixel(col,row,color);

}  }}
```

The output of this sample code is illustrated in Figure 1.

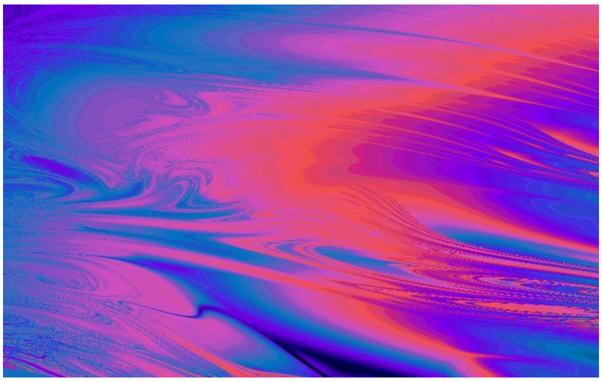


Fig 1 : Output of the sample code

4. VARIATIONS ON THE SAME THEME

The following variations on the code given in section 2 yields interesting effects.

VARIATION 1

Replace …(1) by

double xmax=.32,xmin=.24,ymax=-.270,ymin=-.550;

Replace …(4) by

z=complex((ax+ax3)/2,(ay+ay3)/2);

The output is illustrated in Figure 2.

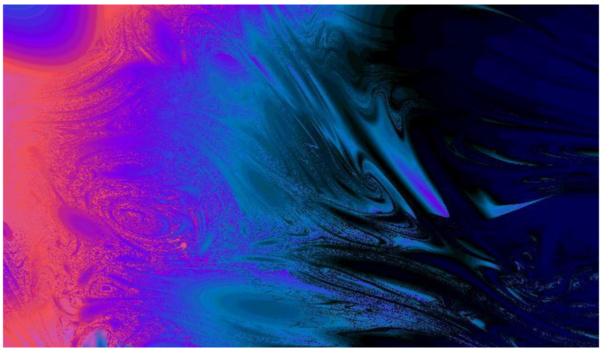


Fig. 2 : Output of Variation 1

VARIATION 2

Replace …(1) by

double xmax=-.078,xmin=-.65,ymax=1.510,ymin=.509;

Replace …(4) by

z=complex((ax5+ax4)/2,(ay5+ay4)/2);

The output is illustrated in Figure 3.

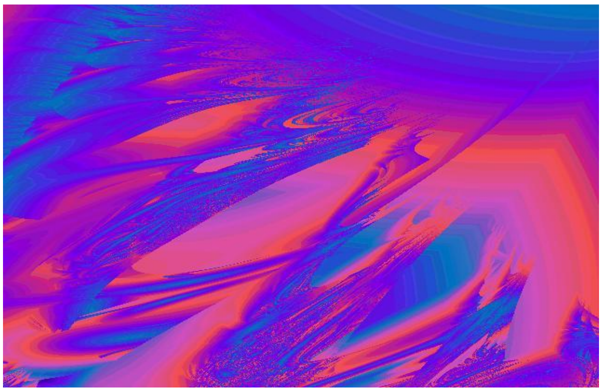


Fig. 3 : Output of Variation 2

VARIATION 3

Replace …(1) by

double xmax=-.78,xmin=-1.31,ymax=.10,ymin=-.209;

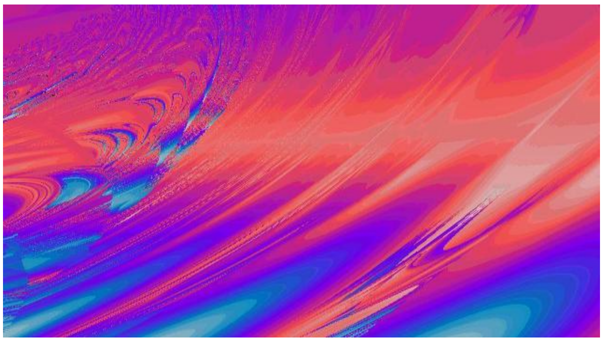
Replace …(3) by

float ax=real(z+c),ay=imag(z+c);

Replace …(5) by

z=z*z;

The output is illustrated in Figure 4.



VARIATION 4

Replace …(1) by

Fig. 4 : Output of Variation 3

double xmax=-.8,xmin=-1.02,ymax=.230,ymin=.14;

Replace …(5) by

z=z*z*z*z+c;

The output is illustrated in Figure 5.

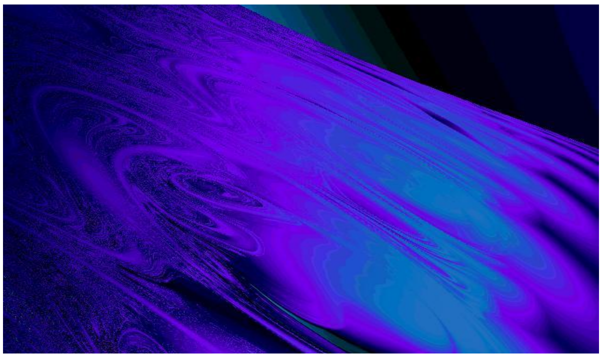


Fig. 5 : Output of Variation 4

VARIATION 5

Replace …(1) by

double xmax=-.03, xmin=-.8, ymax=-.0, ymin=-.75:

Replace …(2) by

int max_iterations=56;int max_size= 31004.0;

Replace …(5) by

z=sin(z)+ c;

The output is illustrated in Figure 6.

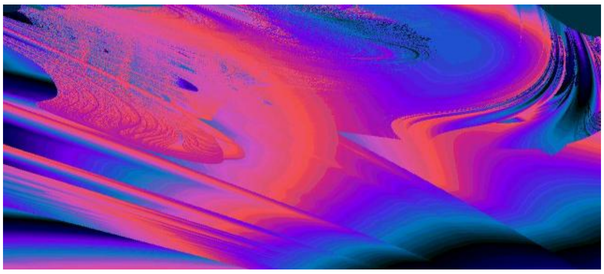


Fig. 6 : Output of Variation 5

4. CONCLUSION

This paper presents one way of using iterative functions to escape-time fractals. However, instead of using average of functions one could use a probabilistic system. It would also be interesting to note the effect of similar study on escape-time fractals other than Mandelbrot and Julia. These would be explored in future.

5. ACKNOWLEDGMENTS

The author wishes to acknowledge his debt to the referee(s) for their constructive suggestions and encouragement

6. REFERENCES

[1] Barnsley, M. 1983 Fractals Everywhere, Academic Press.

[2] Brill, R. 1995 Embellished Lissajous Figures, The Pattern Book(ed. Pickover, C.).

[3] Davis, C. and Knuth, D.E. 1970 Number representations and dragon curves,Journal of Recreational Mathematics 3(1970) 66-81 and 133-149..

[4] Draves, S. 1992 The Fractal Flame Algorithm, flame3.com/flame-draves.pdf.

[5] Hofstadter, D.R. 1982 Strange attractors: Mathematical patterns delicately poised between order and chaos, Scientific American 245(May 1982)16-29.

[6] Peitzen, H. 1987 Beauty of Fractals, Springer.

[7] Pickover C. quoted in Fractint formula documentation,www.nahee.com/spanky/www/fractint/popcorn_type.html.

[8] Ruell,D. 1980 Strange attractors, Math Intelligencer 2(1980)126-137.

[9] Stevens, R. 1989 Fractal Programming in C, M&T Books.