Bounce

This pocket program illustrates the principle of a popular screen saver. It serves as a portal not only to ever more fanciful and artistic variations, but also as an introduction to simulating colliding objects and particle systems. It depicts one rectangle bouncing inside a larger rectangle, leaving a trace of its 10 most recent positions.

```
10 REM BOUNCE
15 CLS: N=10
20 READ XM,YM,DX,DY,DU,DV
21 DATA 239,63,1,2,-2,-1
25 READ X(N),Y(N),U(N),V(N)
26 DATA  10, 42, 42, 54
40 LINE(X(N),Y(N))-(U(N),V(N)),1,B
45 LINE(X(1),Y(1))-(U(1),V(1)),0,B
50 FOR J = 2 TO N
52 X(J-1)=X(J) : Y(J-1)=Y(J)
54 U(J-1)=U(J) : V(J-1)=V(J)
55 NEXT J
60 X = X(N)+DX : Y = Y(N)+DY
62 IF(X<0)OR(X>XM)THEN(DX=-DX)ELSE(X(N)=X)
64 IF(Y<0)OR(Y>YM)THEN(DY=-DY)ELSE(Y(N)=Y)
70 U = U(N)+DU : V = V(N)+DV
72 IF(U<0)OR(U>UM)THEN(DU=-DU)ELSE(U(N)=U)
74 IF(V<0)OR(V>VM)THEN(DV=-DV)ELSE(V(N)=V)
80 GOTO 40
```

Line 40 uses a rectangle drawing variant of the line command to draw the tenth *box* specified by opposite corners, `(X,Y), (U,V)`. Line 45 erases the first box, in a sequence of ten boxes.[1] The inside loop, lines 50–55, moves each point one place back in the queue, leaving the N-th point to be updated on line 60. Now, if the new point is out of bounds in the horizontal or vertical directions, then, instead of entering an illegal point in the queue, the displacement increment changes sign, lines 62 and 64. The 70s do the same thing for dialects and not in others.

Exercise 17. A dynamical system which shows the effect of updating a very large number of orbits is called a *particle system*. The above trick suggests a way of displaying a tracer extending back from the current particle position, perhaps with color or gray-shade attenuation. Modify the program first to a 2-particle system (omit the boxes but draw both corners), then to a many particle system. Note that the reassignment in the 50s is inefficient. Rewrite the algorithm so that an index keeps track of the current head of the queue. Instead of the Nth point, erase the current point, update it, then draw it, and advance the index to the eldest place in the queue. In modern graphics systems it is quicker to erase the entire screen and redraw the entire scene, at animation speed, than to manipulate individual points. In that case, you can attenuate the color of each tail as it is redrawn.

Julia Set

[1] We chose 10 because BASIC does not require such short arrays to be specifically allocated. If your BASIC can't draw or erase boxes so conveniently, you'll have to call appropriately written subroutines here.

We complete our sampler of pocket programs with one that generates a Julia set. There is a BASICGLUT version you should try before reading any further. Along with the usual (W)ipe key, be sure to try the lower and upper case X and Y-keys as well. Also, you should brush up on your complex number arithmetic to appreciate Lou Kauffman's algorithm for taking the complex square root. It is the heart of this pocket program.

```
 10 REM JULIA SET
 11 CLS
 15 READ  XO,YO,UX,UY
 16 DATA 120,32,40,40
 20 READ NX, NY, MX, MY
 21 DATA  0, 0.5, -1, 0
100 X = NX - MX : Y = NY - MY
120 R = SQR(X*X+Y*Y)
130 NX = SGN(Y)*SQR((R+X)/2)
140 NY =        SQR((R-X)/2)
150 IF(2*RND(1)<1)THEN NX=-NX:NY=-NY
160 PSET (XO+NX*UX, YO+NY*UY)
166 PSET (XO-NX*UX, YO-NY*UY)
180 GOTO 100
```

The most famous of all fractals, the *Mandelbrot Set*, is based on the same discrete dynamical system as logistic chaos, but the function is iterated over the complex numbers. Recall that complex numbers may be visualized as points in the coordinate plane, except that we write (x, y) as the polynomial $x + iy$. Addition and multiplication is polynomial, except that we may reduce higher powers of i by the identity $i^2 = -1$. Thus

$$(x + iy)^2 = (x^2 - y^2) + i(2xy).$$

We can even divide complex numbers, and take their square roots.[2] The JULIA program uses the square roots in a creative way. Here is the story.

Suppose we consider the feedback loop $w \leftarrow w^2 + \mu$, where μ is a (complex) parameter. From DeMoivre's formula we can see that if $\mu = 0$ life is exceedingly simple. Every point inside the unit circle converges to the origin, every point outside the unit circle goes to ∞, and points on the unit circle stay on it. The unit circle is an invariant set that divides two *basins of attraction* for the attractor 0 and ∞.

When $\mu \neq 0$ it's more complicated, but ∞ is still an attractor. It takes some arithmetic to demonstrate that applying the iteration to a complex number outside

[2] A cheap way to take roots of a complex number $x + iy$ is to rewrite it in *polar form*. That is, factor out the *modulus* $r = \sqrt{x^2 + y^2}$,

$$r(\frac{x}{r} + i\frac{y}{r}) = r(\cos(\theta) + i\sin(\theta)),$$

where $\theta = \arctan(y/x)$. By DeMoivre's theorem, an n-th root of $re^{i\theta}$, which is how this expression was abbreviated by Euler, is $r^{\frac{1}{n}} e^{i\frac{\theta}{n}}$. But this is not the way we take square-roots in JULIA.

the radius-2 circle always diverges to ∞. What happens if you apply the iteration to a given starting point, say $w = 0$, depends on μ. The Mandelbrot set is by definition the set of all μ for which the orbit of the origin stays finite. Thus $\mu = 0$ is in the Mandelbrot set. To determine that a μ is *not* in the Mandelbrot set, all you need to do is to wait and see whether the sequence w_n, where $w_0 = 0$ and $w_{n+1} = w_n^2 + \mu$ ever gets outside the circle of radius two.[3] The pretty pictures you've seen of the crab-like Mandelbrot set are generated by assigning a color to μ depending on how long it took the corresponding sequence to escape the circle of radius 2. The black pixels correspond to those μ which did not escape before the patience of the programmer ran out. The boundary of the Mandelbrot set was thought to be a fractal. But there is some indication that its Hausdorff dimension is, in fact, integral.

You will find much more exciting and satisfying accounts about this set elsewhere.[4] here we are interested in a different set of points in the complex plane, one for each μ. The Julia set of discrete dynamical system in the complex plane is defined to be the set of points that separates the various basins of attraction. You could locate the Julia set by reversing the dynamical system. This is somewhat like finding the continental divide by forcing a drop of water from each ocean to flow backwards in time, flipping a coin to decide which way to go at each fork in the watershed.

Now let's see how the JULIA program reverses the flow $w \leftarrow w^2+\mu$ by implementing $n \leftarrow \sqrt{n-\mu}$. Lines 130 and 140 look like the place n is updated. Note that

$$\begin{array}{rcccc} n_x^2 - n_y^2 & = & \frac{r+x}{2} - \frac{r-x}{2} & = & x \\ 2n_x n_y & = & \sqrt{r^2 - x^2} & = & \sqrt{y^2} \end{array}$$

from which it follows that $x + iy = (n_x + in_y)^2$.

Exercise 18. Explain why the sign of y, written $sgn(y)$, is needed on line 130 to insure that we choose the squareroot correctly. Recall DeMoivre's rule.

Exercise 19. On line 150 we choose one of the two square roots with equal probability. What happens if you skip this, or choose them one with greater probability?

Exercise 20. What happens when the parameter μ is varied? Implement this program on a sufficiently fast computer so that the value of μ can be varied in real time with the mouse. In the BASICGLUT version of Julia, below, the position of μ already can be moved by key-presses. Use the techniques from the Sierpinski Gasket to mark a circle about this point. Later, attach this point to the mouse and paint the orbits to help investigate the properties of this Julia set.

```
/* Julia Set in GLUT, gkf 3jan2K */
#include <stdlib.h>
#include <stdio.h>
#include <gl\glut.h>
```

[3] Note that $w_1 = \mu$, $w_2 = \mu^2 + \mu$, ...

[4] B. Mandelbrot, The Fractal Geometry of Nature, W. H. Freeman, 1982.
H.-O. Peitgen and P.H. Richter, The Beauty of Fractals.

```
#include <math.h>
#define RND ((float)rand()/RAND_MAX)         /* random fraction */
#define SGN(x) ((x)<0?-1:1)                   /* signum function */
#define PIXEL 2                               /* fat dots */
#define NAP  1000                             /* microseconds */
#define SLEEP(u) usleep(u)
float red = 1., green = 1., blue = 1. ;        /* default colors */
float nx = 0., ny = 0.5, mx = -1, my = 0;      /* julia data */
float x, y, r;
/*********************************************************/
void usleep(int nap){int ii; for(ii=0;ii< nap; ii++);}
/*********************************************************/
void dotit(float xx, float yy){
     glPointSize(PIXEL);
     glColor3f(red,green,blue);
     glBegin(GL_POINTS); glVertex2f(xx,yy); glEnd();
}
/*********************************************************/
void wipeit(){
    glClear(GL_COLOR_BUFFER_BIT); glClearColor(0.,0.,0.,0.);
    nx = RND; ny = RND;        /* also start from a random place */
}
/*********************************************************/
void zapit(){mx = -1; my = 0; wipeit();}
/*********************************************************/
void display(){
  float x, y, r;
  x = nx - mx ; y = ny - my;      /* z = nz - m */
  r = sqrt(x*x + y*y);            /* r = |z|   */
  nx = SGN(y)*sqrt((r+x)/2);      /* nz = sqrt(z) */
  ny =        sqrt((r-x)/2);
  if(2*RND < 1){ nx = -nx; ny = -ny;}   /* choose one */
  dotit( nx,  ny);                /* but plot both */
  dotit(-nx, -ny);
  SLEEP(NAP);
}
/*********************************************************/
void keyboard(unsigned char key, int x, int y){
  switch(key){
     case  27: fprintf(stderr," Thanks for using GLUT ! \n"); exit(0); break;
     case 'x': mx += .1; break;  /* move modulus x-ward */
     case 'X': mx -= .1; break;
     case 'y': my += .1; break;  /* move modulus y-ward */
     case 'Y': my -= .1; break;
     case 'w': wipeit(); break; case 'z': zapit(); break;
```

```
   }
}
/*********************************************************/
void idle(void){ glutPostRedisplay(); }
/*********************************************************/
int main(int argc, char **argv){
  glutInitWindowSize(400, 400);  glutInitDisplayMode(GLUT_RGB);
  glutCreateWindow("<< Julia Set in GLUT >>");
  glutDisplayFunc(display); glutKeyboardFunc(keyboard); glutIdleFunc(idle);
  glMatrixMode(GL_PROJECTION); glLoadIdentity();
     glOrtho(-2.0,2.0,-2.0,2.0,-2.0,2.0);
  glMatrixMode(GL_MODELVIEW); glLoadIdentity();
  glutMainLoop();
  return 0;
}
```

Conclusion

Here is a table of the the pocket programs we have discussed so far.

	GASKET	LORENZ	SINEWHEEL	FUNCTIFIC	CHAOS	BOUNCE	JULIA
Draws straight line			X		X		
Graphs a function			X	X	X		
Uses a figure macro	X					X	
Pseudo-random nrs	X						X
Give 3-D illusion		X					
Continuous dyn sys		X				X	
Iterated function sys	X				X		X
Attractor	X	X			X		X
World/screen coords		X	X	X	X		
IF/THEN structure				X	X	X	X
Subroutines					X		
FOR/NEXT loops			X	X	X		X
Trigonometry			X				
Complex numbers							X
Auto-scaling				X			
Fractal geometry	X				X		X
Strange attractor	X				X		X