2. The Lorenz Mask

For the Sierpinski iterated function system we could give an adequate mathematical presentation in few pages. For the Lorenz dynamical system this is not possible. Here we devote a commensurate space to visualizing its famous 3-dimensional attractor, the Lorenz Mask, in a minimal graphical environment. There is a `basiCglut` version of the following program at the end of the section.

There is a separate chapter on the Lorenz Attractor.

```
 10 REM LORENZ ATTRACTOR
 12 CLS
 15 DATA 120,32,1 ,-1,1,.1
 16 READ  XO,YO,UX,UY,X,Y
 20 DATA .05,-50,.01
 21 READ   E,  N,  D
 90 XR = XO-N : XL = XO+N
100 YP = YO + Y*UY
110 PSET(XL+(X-E*Z)*UX,YP)
120 PSET(XR+(X+E*Z)*UX,YP)
130 X1 = X + (10*Y-10*X)*D
140 Y1 = Y +(28*X-X*Z-Y)*D
150 Z  = Z + (X*Y-8*Z/3)*D
160 X=X1:Y=Y1:GOTO 100
```

This also illustrates a dynamical system insofar as a rule inside an eternal loop moves a point to a new position. But this is a *deterministic* system because there is no choice in rules. It is a 3-dimensional system because the point being moved about has 3 components. This raises the problem of how to represent 3-dimensional data in the two dimensions of a picture. The most efficient thing to do on a slow, monochrome (b/w), and coarse grained screen is to use stereo pairs.[1]

Stereopairs are viewed with the help of devices which insure that your right eye sees one image, while your left eye sees the same image, slightly rotated (and sheared,[2] to be exact.) In the absence of such aids, you can reverse the right and left images, and then cross your eyes, until you see three rather than four fuzzy images. Then wait until the middle one comes into sharp focus. On the printed page, the images are smaller and closer together. Here the view for the right eye is on the right. These can be viewed unaided by focusing your eyes at infinity (not crossed). One way of achieving this is to place your nose right up to the image until your eyes are relaxed (unconverged, unfocused). Then move the page back slowly until the fused, 3-D image jumps into focus.[3] To reverse right with left in the program, change the sign of the nose offset, N, or the eye-shear fraction, E, but not both.

[1] Some others possibilities, to be dealt with later, are: linear perspective, depth-cueing, motion parallax, occlusion by z-buffering, lighting and shading, shadows, and VR (if you put them all together and add head-tracking.)

[2] A distortion that moves a rectangle to a parallelogram, without changing its base or altitude, is called a *shear*. Think of a stack of playing cards pushed uniformly to one side.

[3] Don't do this with a CRT, though.

basiCglut

The `basiCglut` examples you should look at while reading this go somewhat further and solve some of the exercises proposed further down. The C-code mimics the BASIC-code. It also is a modification of the C-code for the Sierpienski gasket. But, here we have *factored out* a function `plotAdot(x,y,r,g,b)` from `display()` to simplify writing the position and color of each dot as it is plotted.[4] Moreover, we add some sophistication in C-programming as we progress from one pocket program to the next. Here, the `display()` function combines *initialization* and *iteration*.[5] Note that the display-block is divided into three subblocks. This simplifies factoring this function further. The second block draws the right and left images of the current point, and the third udates it. Note that within a block you can define local variables. They exist only inside the block. Ordinarily they also exist only for the duration of one execution of the block. But you can also preserve the value of a variable from one iteration to the next by declaring it `static`, as in the first sub-block.[6] In the first block, the integer `first` begins its exisence with the value 1, for *true*. The first time the display function is called, the window in which the mask appears is cleared to black. You no longer need to (W)ipe[7] the screen manually, as in the gasket examples. Since you don't want to clear the screen the next time (you want the positions of the point to accumulate on the screen), you lower the flag `first = 0`.

mathematics

The shape you see developing is called the *Lorenz Mask* and it is a very popular example of a *strange attractor*.[8] Two dimensional dynamical systems do not need the complication of stereo-viewing. On the other hand, they don't have strange attractors. The Lorenz is also a favorite character in Nonlinear Mechanics because the rule (lines 130–150) that calculates the velocity at a point is not a linear function of the coordinates of the point (note the products xz and xy.)[9]

graphics

The stereographic model used here goes something like this. The problem to be solved is how to project an object to the image plane so that when viewed by the corresponding eye, the image on the retina is the same as if the object were seen live. If all aspects of perspective, including depth-of-field, were considered, this would be nearly impossible. The eye-mind combination is, fortunately, very

[4] In the future we shall not always explain each factorization, which consists of isolating a part of a function by replacing but with a call to a new, subsidiary function. But you should look for it when comparing two neighboring versions. You should also reflect why the subsidiary code was factored out. After all, we could write a C-program with only *one* function, `main()`.

[5] The former is done once, the first time the function is called. The latter is done each time the function is called. It is better programming style to write separate initialization and iteration functions, but this requires more careful modifications when grafting this routine in other programs.

[6] Global variables are "static" for the program.

[7] We adopt this typographical convention, both in the text and in the heads-up displays on the screen, to mean that the action is accomplished by pressing that key.

[8] There is no agreement as to what features makes one attractor "stranger" than another. Points of equilibrium and limit cycles are not strange but fractals are. More significant is the unpredictable way the trajectory of a single initial point switches from orbiting about one or the other equilibrium points in the Lorenz Mask.

[9] Nonlinearity in 2-dimensions also has interesting features absent from linear systems. One of the most interesting examples is the Liénard-Van der Pol system we treat further on.

forgiving. Here the shortcut is to ...

- ... ignore anything but a tiny (about 2.5 degrees) rotation of one image against the other,
- ... replace this rotation by a shear.[10]

The endless iteration loop begins at line 100. This program uses both *world* and *screen* coordinates. On line 100, the vertical screen coordinate, `YP` is obtained by adding the fraction `Y` of the vertical unit `UY` to the vertical origin `YO.` This interprets the world coordinate `Y` as a fraction (proper and improper) of the fixed vertical displacement. It is an example of the *axonometric*[11] approach to Cartesian coordinatization.

In the next two lines, 110 and 120, the horizontal displacement from the left, resp. right, origin, `XL, XR` is computed. The true displacement, `X`, is seen only by the cyclopean eye (in the middle of your forehead). Both eyes see this only when `Z = 0` . It becomes progressively greater as the point recedes into the background, ie as the Z-coordinate becomes greater.

numerical integration

The iteration rule here is the simplest (and least accurate) numerical integration method, known as *Euler's Method.* Here is how it works on a system of first order differential equations,

$$\dot{X} = F(X),$$

where X is a vector and $F(X)$ is a vector valued function of the components of X. First, switch from Newton to Leibniz notation,

$$\frac{dX}{dt} = \dot{X},$$

then multiply through by dt,

$$dX = \dot{X}dt = F(X)dt.$$

Now replace the infinitesimal time step dt by a finite one h,[12] and the infinitesimal displacement of X by the corresponding finite one,

$$\triangle X = X_{new} - X_{old}.$$

We can now solve for X_{new}

$$X_{new} = X_{old} + F(X_{old})h.$$

If this reminds you of the beginning of a Taylor's series then you have learned something in your calculus class.

In lines 130-150, the point `(X,Y,Z)` is moved the small fraction `D`[13] along a displacement, the *velocity vector*, which itself depends on the current position. That is why temporary values, `X1,Y1,` are used until `Z` has been updated too. **graphics**

Some default values are set in the preamble, lines 0-99. To understand them better, you should experiment by changing variables. In `basic`, insert an `INPUT` statement[14].To modify the C-code you will need some standard I/O functions.[15] The visualization of stereo-images can be helped by adding more graphical information which the eye-brain interprets as spatial information. One of these is *depth-cueing.* This means that points further back are drawn more lightly than those in front. In the decade of the UIMATH.Applelab, Ted Emerson and Cary Sandvig, *inter alia* , extended Applesoft BASIC on our Apple IIs with machine language subroutines. They not only achieve animation speed acceleration but built remarkably mature 2-D graphics language, called *GS&*. Depth-cued with 16 levels of gray, or painted with 16 colors the Lorenz Mask achieved great popularity, leading to a generation of 3-D implementations in 3-D graphics and in the CAVE.

Exercise 9: Lorenz Attractor

Modify `lorenz.c` to implement depthcueing and painting[16] the third dimension. Vary the parameters[17] $(r, p, b) = (28, 10, 8/3)$ used by Lorenz. Try $(16, 45.9, 4)$. Vary the starting place of the orbit. Draw four views of the 3-D object on the screen simultenously (in the tradition of drafting) by drawing the 3 orthographic views.[18] The fourth view should be an axonometric general view.

Exercise 10: Liénard-Van der Pol.

This 2-dimensional dynamical system depends on a function $y = f(x)$, for example the cubic $f(x) = x^3 - \alpha x$, where α is a parameter. Note that for $a > 0$, f factors into

$$f(x) = x(x - \sqrt{\alpha})(x + \sqrt{\alpha})$$

and its graph is an inflected cubic with two humps. What happens to this graph as α goes negative?

Exercise 11: LVdP continued

The velocity, $(\dot{x}, \dot{y})$, at a point $P = (x, y)$ is obtained geometrically as follows. Drop a perpendicular from P to the graph of f at $(x, f(x))$, proceed horizontally to $(0, f(x))$ on the y-axis, return

[10] In effect, replacing the Cosine of a small angle by 1, and its Sine by a small number.

[11] The official description of an *axonometric projection* goes like this. In a plane, draw three line segments, the x, y, z-axes, from a common point, the origin. The axonometric image of a point (ξ, η, ζ) in 3-space is found by moving parallel to the axes, a fraction ξ, η, ζ, of the length of the corresponding axis.

[12] In the programs we use more dramatic names for the time-step h, `D` and `dd` respectively.

[13] Our h in the previous paragraph.

[14] After the defaults!

[15] Study how the non-graphical example `heron.c` is written. It accepts inputs and prints outputs into the command window.

[16] Also called "color-coding" and "false-coloring", we assign colors to spots in the picture not to emulate nature but to transmit information.

[17] These are sometimes called the Reynold Number, r, Prandtl Number, p, and Box Ratio, b, respectively. Perhaps a web-search will discover the reasons for the names.

[18] The *plan*-view is from the top, the two *elevations* are side views, 90 degrees apart.

from there to P along vector $(x, y - f(x))$, turn right, facing down $(y - f(x), -x)$:

$$\begin{aligned} \dot{x} &= y - f(x) \\ \dot{y} &= -x \end{aligned}$$

which becomes

$$\begin{aligned} X_{new} &= X + (Y - F(X)) * D \\ Y_{new} &= Y - X * D \end{aligned}$$

by Euler's Method. To see what this dynamical system looks like you should write a program. But it is also possible to do this graphically provided you are willing to follow Liénard's construction accurately enough. If you own a drawing board, a T-square and a right-angle, this construction is almost faster than programming it.

Exercise 12: LVdP concluded

To give this system physical meaning, and reveal its historical origin, eliminate the phase-variable y by differentiating the first differential equation, and substitute the second:

$$\ddot{x} = -x - f'(x)\dot{x}$$

or

$$\ddot{x} + g(x)\dot{x} + x = 0,$$

where $g(x) = f'(x) = (3x^2 - a)$ represents a non-linear, position dependent, frictional term. For negative α we have positive friction, which will eventually damp out this oscillator. But for positive α, there is always a region on either side of the equilibrium $x = 0$ where this system is given a kick (negative friction). Demonstrate[19] that this sustains the oscillator. The LVdP system has a globally attracting limit cycle. This is the toy-version of the vacuum tube equation. Find a reference and relate it to the LVdP equation.

[19] By drawing board or computer.

```
/* Lorenz Mask in GLUT, gkf 3jan2K */
#include <stdlib.h>
#include <stdio.h>
#include <gl\glut.h>
#include <math.h>
/*********************************************************/
/* assuming the window has an 800 x 400 pixels */
float xo = 400 , yo = 200 ,             /* screen origin */
      ux = 8 , uy =-8 ,             /*  screenunits */
      xx = 5, yy = -1 , zz = 0.,    /* world particle */
      ee = .05,            /* epsilon eye shear */
      ns = 200,             /* nose displacement */
      dd = .01;            /* delta stepsize  */
/*********************************************************/
void plotAdot(float xx, float yy, float red, float green, float blue){
     glColor3f(red,green,blue);
     glBegin(GL_POINTS); glVertex2f(xx,yy);
     glEnd();
}
/*********************************************************/
void display(){
  {static first=1; if(first)        /* first time clear the slate */
    glClear(GL_COLOR_BUFFER_BIT);
    glClearColor(0.,0.,0.,0.);       /* to black */
    first =0;}

  { /* draw the dots */
    float xleft = xo - ns, xriht = xo + ns;
    plotAdot(xleft  + (xx - ee*zz)*ux, yo - yy*uy, 1.,0.,1. ) ;
    plotAdot(xriht  + (xx + ee*zz)*ux, yo - yy*uy, 0.,1.,1. ) ; }

  {  /*update a dot */
    float xn,yn,zn;  /*local variables */
    xn = xx + (10*yy - 10*xx)*dd;
    yn = yy + (28*xx - xx*zz - yy)*dd;
    zn = zz + (xx*yy - 8*zz/3)*dd;
    xx = xn; yy = yn; zz = zn; }

  /*  usleep(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 'w': {glClear(GL_COLOR_BUFFER_BIT);
```

```
                glClearColor(0.,0.,0.,0.);  break;};
   }
}
/*********************************************************/
void idle(void){ glutPostRedisplay(); }
/*********************************************************/
int main(int argc, char **argv){
  glutInitWindowSize(800, 400);
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_RGB);
  glutCreateWindow("<< Sierpinski in GLUT >>");
  glutDisplayFunc(display);
  glutKeyboardFunc(keyboard);
  glutIdleFunc(idle);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glOrtho(0,800,0,400,-10.0,10.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glutMainLoop();
  return 0;             /* ANSI C requires main to return int. */
}
```