BASIC*Pocket†Graphics‡Programs§

George Francis

Orinial text 6jan97. Revised 6jan01.¶

1 The Sierpinski Gasket

Pocket programs are simple programs that do non-simple things. They also contain examples of useful programming techniques. Once upon a time, when everyone had access to a computer that understood BASIC, and when everyone knew how to speak such simple languages, this chapter also served as an introduction to programming. Regrettably, this is no longer true. To work with the pocket programs on a computer you will need additional resources.

We will use a BASIC dialect (originally for a Tandy TRS80 computer) as pseudo-code because its simple syntax and the mnemonic function names make it easy to guess the meaning from the context. Besides, the line numbers make it easy to reference. However, since this document is as much about *what* the programs do as *how* they were written, you should implement the pocket programs in your currently favorite computer language on your most accessible computer. In the current edition of the notes are

*Kemeny and Kurtz invented this *lingua franca* of computing languages at Dartmouth, in the mid sixties.

†Originally they were called "hip-pocket programs", being small enough to carry them around in your hip-pocket and program them from memory when convenient.

‡They produce moving pictures instead of text.

§And they are short enough to fit on a single computer screen or paper page, which simplifies life.

¶Correction 15jan01.

sections that refer to such an implementation in C/OpenGL/GLUT.[1] We label such section by mnemonic **basiCglut**. Sections dealing specifically with programming in BASIC will be labeled **basicBASIC**. But the pocket programs do not really need such an elaborate environment. Any language that can plot points and lines in a rectangle will do as well. The pocket programs translate very nicely into JAVA.

```
10 REM SIERPINSKI GASKET
15 X=100 : Y=50
19 CLS : REM CLEAR SCREEN
20 DATA 0, 32, 100, 0, 100, 63
30 READ X(0),Y(0),X(1),Y(1),X(2),Y(2)
40 I = INT(3*RND(1)) : REM INTEGER PART
50 X=(X+X(I))/2 : Y=(Y+Y(I))/2
60 PSET(X,Y) : REM PLOT POINT
70 GOTO 40 : REM PICK I=0,1,2 RANDOMLY
```

dynamical system

This 8-line program draws a famous fractal, the *Sierpinski Gasket*. It does this by means of a dynamical system.[2] A *dynamical system* is a (usually multidimensional) process which moves points to successive positions according to a rule. If this rule determines the next position of a point strictly from its current position, the system is said to be *deterministic* and *autonomous*.[3] The rule here randomly chooses one of three strategies to compute the next position. This makes it an *iterated function system* (IFS).[4] The Sierpinski Gasket is the attractor of this IFS. An *attractor* of a dynamical system is an invariant subset of its configuration space. such that every orbit converges to it. The set of successive positions of a point is called the *orbit* of its initial point. A geometrical interpretation of the physical states of a dynamical

[1]For such simple programs we prefer C over C++ for its brevity. We use OpenGL because it is the standard way of doing 3-dimensional graphics. We use Mark Kilgard's GLUT library because it greatly simplifies programming in OpenGL. Kilgard has written an excellent single volume reference work on the subject: *OpenGL: Programming for the X Window System*, Addison-Wesley, 1996.

[2]Customarily, the first time a technical term appears it should be italicized so you can find it again when you're looking for its definition. Sometimes, however, it is useful to use a term before defining it. This custom is called "prototyping" in modern computer languages and is much encouraged.

[3]Non-autonomous dynamical systems also depend on time. Non-deterministic, or *stochastic* dynamical systems contain an element of chance. Since a random-number generator on a computer is itself a deterministic process, we don't need to make the distinction here.

[4]The notion of an IFS is due to Michael Barnsley, who wrote about it in *Fractals Everywhere*, Academic Press, 1988.

system is called its *configuration space.*[5] A subset is *invariant* if the orbits of its points stay in the set.

The Sierpinski Gasket looks like the remains of a triangle whose center quarter[6] was *recursively* removed. That is to say, the center quarter of the remaining three triangles is also removed, and so *ad infinitum.*

A geometric description of the algorithm goes like this. The configuration space is a triangle. To compute an orbit of a point, choose one of the three vertices randomly, move the point half-way towards that vertex, and choose again. Note that this is a non-deterministic dynamical system, since the next time an orbit is computed for the same initial position it will probably be different.

basicBASIC

Here is the way you would write, test, debug, and modify the program in BASIC.[7] You would start by writing line 50 because that is the essence of the program. Here, the current position `(X,Y)`, is replaced by a point halfway to the I-th vertex, which was chosen randomly in line 40. Then we want to plot the new point `(X,Y)`. The exact command to use depends on the flavor of BASIC you're using. After this, we are ready to repeat.

Prior to moving the point, we must choose which vertex to move towards. For this we use a pseudo-random number generator which is language/system dependent. Here, the value of the expression `RND(1)` is a decimal fraction (a floating point number between 0 inclusive and 1 exclusive.) Line 40 takes the integer part (also called the floor) of a decimal between 0 and 3. Thus, I is 0,1, or 2 with roughly equal probability.

loop body

That's all we need each time through the iteration. so close the *body of the loop* here with the `GOTO 40` statement. In different languages you would use another way of writing an eternal loop.[8]

The loop itself still has some indeterminates. That is, some variables ap-

[5] This is also called state space, phase space, and even, iteration space by some.

[6] Connect the midpoints of the sides.

[7] Recall that BASIC is an interpreted language, a rarity nowadays. Most of its syntactically correct phrases can be executed by entering them without the initial line number. That initial line number serves the dual purpose of deferring execution until a whole program was written, in any order. The order of execution, then, is determined by the order of the line numbers.

[8] Be sure that you also know how to interrupt an eternal loop you've written before executing it. Often CTL-C or CTL-D stops a runaway loop. If that fails, try closing the window. More drastic measures may be needed.

loop preamble

pearing on the right side of the assigment symbol, =, are themselves not yet assigned. So, in the preparatory part of the program (sometimes called the *preamble* of the loop) assign some values.

In this program, the triangle data is a list, line 20, which is assigned to variables listed on line 30 in the same order.[9] In BASIC there is only one data buffer. The BASIC interpreter fills this buffer, using all the `DATA` statements, such as on line 20, anywhere in the program, before executing the first line. Each `READ` advances a pointer which points to the next datum to be read. If, for some reason, you wish to restore the data pointer all the way to the beginning of the data buffer, while the program is being executed, then use the `RESTORE` command. In many BASIC dialects, writing and drawing is done on the same screen, hence clearing the screen is desirable for drawing. In others, the graphics needs to be invoked first, and sometimes elaborate initializations must be made before anything can be drawn on the screen. This is always *local*, which means that it not only depends on the language, but also on the particular hardware the language is running on.

basiCglut

We next look as the way this translates into C. The entire forty-six line program is at the back of this section. The stars separate the program into five sections.

compiler directives

In the top section we tell the compiler where to look for words we use but do not explicitly define in the program. We include the appropriate *header files*. Lines beginning with the pound-symbol,`#`, are messages to the compiler, called *compiler directives*. They end at the end of the line, and are processed by the *pre-compiler*.Other lines are for the compiler to read and process. Text we don't want the compiler to read, such as comments, is bracketed between /* and */. Another compiler directive,`#define` , is used to define macros. A *macro* is a single, short expression (traditionally all in caps) standing for a longer expression. The compiler substitutes the longer expression wherever it finds the macro in the code. So, wherever `RND` appears later in the code, the pre-compiler substitutes the expression `((float)rand()/RAND_MAX)`. That is why there are so many parentheses around it. Careless construction of macros leads to errors which are difficult to debug.[10]

macro

[9] This is an example of a FIFO-buffer (first-in-first-out) being written and read. It is an efficient and common way of passing data between and within computer programs. Friendlier ways of doing this are are common, but not essential.

[10] In a customary overreaction to a programming abuse, the architects of object-oriented languages (such as C++ and Java) frown on or forbid compiler macros. A similar over-

In C, all globally meaningful variables should be defined at the top, before any functions are defined. That way you are not likely to use a variable before it exists. A variable `foo` is defined by first stating its *type*, then its name, and (optionally) assigning is some values. Here we define the 3 vertices of the triangle to have two floating point coordinates each, and then say what they are. Note that `v[][]` is an array of 3 arrays of 2 floats each, just as `vv[]` is an array of just two floats, i.e. it is a point.

`function();`

Subroutines in C are called *functions* and always have () right after their names.[11] Functions also have types, and the `void` type says that the function does not return a value, even though it does other usefule things. Note that the `main()` function is declared to return an integer value, and its last line tells you that it shall be 0. There are no obviously good reasons for this, and C is full of such mysterious conventions. Functions may or may not have arguments. Note that neither `display()` nor `idle()` have arguments. Good C-style would require the `void` in the definition of the former as it appears in that of the latter. Both `keyboard()` and `main()` do have arguments, and in their definition we declare their types and give them dummy names, used in the definition of the function. The body of a function definition is a *block*, a code fragment between braces. Because braces are easy to lose track of, C-programmers regularly close a brace, bracket, or parenthesis the moment they open one, preferring to insert-edit the content later. Later, when the function is called, its dummy arguments take on a particular identity.

block

Every C-program has one last function, invariably named *main*, which is actually executed first. The `main()` of a program using the GLUT-library is difficult to understand even if you already know how to program in C. So we will discuss this (much) later. What you need to know about it now is the the GLUT library uses callbacks.[12] For example, when you press any key, the `keyboard()` function is called by GLUT and executed. Whenever the graphics system on your computer is ready, the `display()` function is called and executed by GLUT. The `idle()` function, which is called whenever nothing else is being done by GLUT, here refreshes the display window if

callbacks

---

reaction to the `GOTO` command in BASIC and Fortran led the authors of structured programming languages (like Pascal and C) to all but eradicate its use.

[11] In these notes we also adopt a convention referring to a function by its name decorated with `()`. Sort of like the honorific "Prof."

[12] "The way GLUT informs a program when operations need to be performed on particular windows is by triggering a *callback* . A callback is a routine that the GLUT program registers to be called when a certain condition is true." Kilgard, p147.

warranted. [13] After `main` has completed its discussion with GLUT, it falls into an infinite `glutMainLoop();` which our `keyboard()`knows how to exit.

We next turn to the subsidiary function which are used by each other, and ultimately by `main()` . We take a classical approach in basiCglut of never using a function before it has been defined. Of course, we use library functions, which are defined elsewhere.[14]

`display()` uses a local integer variable, `ii`, as an index.[15] It also assign it a new random value each time around. Then comes the midpoint formula calculating the new position, followed by the actual graphics. Note how the graphics portion is bracketed betwen commands `glBegin()` and `glEnd()`. Every command in C is a function, in the sense that it takes *zero* or more arguments, and returns *zero* or more values.[16] The prefix "gl" tells you these are OpenGL functions. There are different graphics routines to `glBegin`. Here it is just drawing `GL_POINTS`.[17]

`keyboard()` is called whenever you press a key. In the body of this function (and nowhere else!) the variable `key` holds the identity of the key you pressed, and `x,y` hold the location of the mouse at that moment.[18] Instead of the "if-then-else" syntax we use a "switch-case" statement here so you have an example to copy. Much of C-syntax is easier to plagiarize than to puzzle out from its absract description. If you press the (ESC)ape key, the

[13]That GLUT distinguishes two automatic activities, the display of a picture and the recalculation of its data, enables you to develop a programming discipline which is very useful for multi-processor multi-display systems like the CAVE.

[14]The contemporary style of prototyping functions so that they can be used without definining them leads to verbose and redundant code which is as difficult to read as the old spaghetti code in BASIC .

[15]The custom of using `ii` instead of a single letter `i`is well established. It helps distinguish between mathematical notation and its computer code. In mathematics we use fancy symbols and strange alphabets to keep the names of objects as short as possible. In computing we replace the elaborate symbols with longish nonsense words which are unlikely to already mean something to the compiler. That is why we would not try to name an integer variable `int` nor a pseudo-real number `float` .

[16]In keeping with such constructs as counting from zero, the empty set in math, and certain iterators in regular expressions, this makes perfectly good sense.

[17]This is a macro which the pre-compiler replaces by a number you need never know what it is.

[18]This assignment was performed by the GLUT-library. The GLUT-library "knows" the name of your keyboard function because you told it with the `glutKeyboardFunc()` in `main()` . You could have named this function `Tastatur()` and the first variable `clef`, just as long as you're consistent. The GLUT library is pretty smart, and this sort of anthropomorphism is customary in *code-speak* .

infinite `glutMainLoop()` we are trapped in is broken and everything stops. If you press the (W)-key, the gl-library does ... well, why don't you see what it does! You can experimentally see what `glutPostRedisplay()` does leaving it out, or putting it somewhere else in the program. But the latter is not recommended.

Exercise 1. Show that you understood the basic idea of buffering data; write a BASIC or pseudo-codefragment[19] that reads the j-th member of a data buffer of length $n, n \geq j$. Of course, you can write it in a real language provided you can make it work.

Exercise 2. You may wish to convince yourself that the initial position has no visible effect on the outcome. Do this by assigning the initial position of `(X,Y)` by some other means, for example, randomly, or with an `INPUT` statement in basicBASIC or the mouse in basiCglut. Does changing the probabilities make a difference?

Exercise 3. The algorithm can be interpreted in terms of three affine transformations.[20] What are they in this case? Morever, each of these is a *contraction*. This means that the distance between the images of two points is less than it was before their transformation. In other words, linear contractions map line segments into shorter line segments. Show geometrically that the contraction factor is $\frac{1}{2}$. Write a program using another set of three linear contractions.

Exercise 4. We can give the Sierpinski Gasket following precise definition. Let A_0 denote the original triangle, including its boundary. Let A_1 denote A_0 minus the *interior* of the middle quarter triangle. We do not remove the points on its boundary. And so forth. Then

$$A_0 \supset A_1 \supset A_2 \supset ... \supset A_\infty$$

where $A_\infty = \cap_{n=0}^{\infty} A_n$ is the Sierpinski Gasket. The existence and properties of this intersection of infinitely many sets is a matter of real-number theory. But can you prove that $A_n \supset A_{n+1}$? Sure you can.

Exercise 5. Number the three original points of the triangle V_0, V_1, V_2, and let T_j denote the Barnsley generator which takes any point P half way to V_j

$$T_j(P) = \frac{P + V_j}{2}.$$

Now prove that

$$T_j(A_n) \subset A_{n+1}.$$

Exercise 6. How does the previous fact demonstrate that A_∞ is an attractor?

Exercise 7. Can you locate an argument, perhaps in Barnsley's book, proving that the Sierpinski Gasket is an invariant set?

Exercise 8. On a color computer choose three colors, one for each J. What difference does it make to set the color of the point on line 75 as opposed to 85.

[19] A set of lines which could be grafted into another program.

[20] This exercise is for people who know some linear algebra.

We close this section with a brief demonstration why the gasket is a *fractal*.[21] As defined in Exercise 4, A_∞ is a planar Cantor set, obtained by recursively removing the central quarter of a triangle. Connecting the midpoints of the sides of a triangle decomposes the area into four congruent triangles, each similar to the parent triangle by a scale facter of 2. This self-similarity shows that doubling the side-size of the Gasket triples the measurable content of the figure.[22] If, by analogy to lines, squares, cubes and tesseracts, *dimension* is defined to be that power d of 2 the content of a figure must be multiplied by in order for it to equal the content of a similar figure obtained by doubling its linear scale,[23] then

$$d = \frac{\log_2(3)}{\log_2(2)}.$$

```
/* Sierpinski Gasket in GLUT, for VC98,  gkf jan01 */
#include <stdlib.h>
#include <stdio.h>
#include <gl/glut.h>
#include <math.h>
float v[3][2] = {{1.,0.},{0.,1.},{0.,0.}};   /* a triangle      */
float vv[2]={.2,.9};                         /* initial pos'n   */
#define RND ((float)rand()/RAND_MAX)         /* random fraction */
#define INT(X) ((int)floor(X))               /* integer part    */
/*********************************************************/
void display(){
  int ii = INT(3*(RND));        /* pick random ii = 0,1,2 */
  vv[0]= (vv[0] + v[ii][0])/2;  /* move half-way toward that vertex */
  vv[1]= (vv[1] + v[ii][1])/2;

  glBegin(GL_POINTS);                  /* draw a */
```

[21] There is no agreement on the best definition of a fractal. Originally it denoted a set with a fractional dimension, as defined by Haussdorff. Because it is sometimes impossible (and usually impractical) to compute this dimension less demanding and definitions have been adopted. But these are nevertheless very precise, and vulgar corruptions of the term should be avoided.

[22] Again, we leave this measuring to the analysts.

[23] Aren't you glad you learned algebra. We can translate this mouthful as follows. Let $A(2)$ denote a figure similar to $A(1)$ constructed on the basis of a linear element twice as large as that for $A(1)$. Then *dimension* is defined by $|A(2)| = 2^d|A(1)|$, where the absolute value means the mysterious content we take for granted. Note that for lines, $d = 1$, and for cubes, $d = 3$ because a cube divides into 8 cubes which are half the linear scale. But for the Gasket, $|A(2)| = 3|A(1)|$.

```
    glColor3f(1.0,1.0,1.0);           /* white  */
    glVertex2fv(vv);                  /* point  */
  glEnd();
}
/*********************************************************/
void keyboard(unsigned char key, int x, int y){
  switch(key){
     case  27: exit(0); break;        /* escape with the  (ESC) key  */
     case 'w': glClear(GL_COLOR_BUFFER_BIT); break; /* (W)ipe screen */
   }
}
/*********************************************************/
void idle(void){ glutPostRedisplay(); }
/*********************************************************/
int main(int argc, char **argv){  /* pure GLUTtery here */
  glutInitWindowSize(400, 400);
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_RGB);
  glutCreateWindow("<< Sierpinski 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;             /* ANSI C requires main to return int. */ }
```

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.[24]

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,[25] 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.[26] 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.

[24] 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.)

[25] 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.

[26] 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.[27] 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*.[28] Note that the display-block is divided into three subblocks, suggesting future factorizations. The second block draws the right and left images of the current point, and the third updates 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 preserve the value of a variable from one iteration to the next by declaring it `static`, as in the first sub-block.[29] 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[30] 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 by setting `first=0`.

mathematics

The shape you see developing is called the *Lorenz Mask* and it is a very popular example of a *strange attractor*.[31] 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

[27] In the future we shall not always explain each factorization. It involves isolating a part of the function code by replacing it with a call to a new, subsidiary function containing that code segment. Look for factors when comparing two neighboring versions of the same program. Reflect why the subsidiary code was factored out. After all, we could write a C-program with only *one* function, `main()`.

[28] 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.

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

[30] 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.

[31] 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.

velocity at a point is not a linear function of the coordinates of the point (note the products xz and xy.)[32]

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 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.[33]

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*[34] approach to Cartesian coordinatization.

In the next two lines, 110 and 120, the horizontal displacement from the left and right origins, `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 of differential equations

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,

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

[32]Nonlinearity has interesting features also in 2-dimensions which linear systems don't have. One of the most interesting examples is the Liénard-Van der Pol system we treat further on.

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

[34]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.

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,[35] 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.

graphics

In lines 130-150, the point `(X,Y,Z)` is moved the small fraction [36] `D` 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. 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.[37] To modify the C-code you will need some standard I/O functions. The visualization of stereo-images can be enhanced by adding more graphical information, which the eye-brain interprets as spatially. One of these is *depth-cueing*. This means that points further back are drawn more dimly than those in front.[38]

Examine how the non-graphical program `heron.c` accepts input from and prints output to the command line window.

Exercise 9: Lorenz Attractor

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

[36] Our h in the previous paragraph.

[37] After the defaults!

[38] 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.

Modify `lorenz.c` to implement depthcueing and painting the third dimension.[39] Vary the parameters[40] $(r, p, b) = (28, 10, 8/3)$ 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.[41] 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 $\alpha > 0$, 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 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

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

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

[40] 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.

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

where $g(x) = f'(x) = (3x^2 - \alpha)$ 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[42] 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.

```
/* 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. ) ; }
```

[42] By drawing board or computer.

```
  {  /*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: printf(" 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. */
}
```

3 Sinewheel

The inexpensive, handheld graphing calculator has replaced (for the most part) the slide-rule and handbook[43] as primary assistant in graphing functions. But there are times when one wants to probe ever deeper into specific aspects of a particular function. This is done on a programmable graphics gadget. Consider the following "poem" which plots the sine function in an unusual, and in an unusually interesting way.

```
10 REM SINEWHEEL
11 CLS : REM CLEAR SCREEN
14 DATA .01745 : REM 1 DEGREE
15 DATA 6, 28, 28, 32
16 READ DG, DX, R, XO, YO
50 FOR X=0 TO 360 STEP DX
60 XR = XO + R*COS(X*DG)
61 YR = YO - R*SIN(X*DG)
62 XX = XO + R + R*X*DG
69 REM DRAW LINE WITH PEN DOWN
70 LINE (XO,YO)-(XR,YR),1
71 LINE (XR,YR)-(XX,YR),1
72 LINE (XX,YR)-(XX,YO),1
80 NEXT
```

Since the original BASIC was not graphics oriented, its graphical syntax varies among different implementations. If you are trying this out on an Apple II you should replace lines 70-72 and 11 with this.

```
75 HPLOT XO,YO TO XR,YR TO XX,YR TO XX,YO
11 HGR : HCOLOR = 15 : REM WHITE
```

Even without a computer, you can figure out what the picture will look like simply by drawing (with a pencil) what the program is telling the computer to do. Lines 50 through 80 form a package called a loop. In essence, this loop generates the 60 points `XR,YR` on a circle centered at `XO,YO` and radius `R` in 6 degree steps. Does the circle go clockwise or counterclockwise ?[44] Now things become tricky. Notice how line 62 computes the x-coordinate

[43] My constant companion in the fifties at homework and study sessions was the "Burington: Handbook of Mathematical Tables and Formulas". Sometimes, when the professor wanted us to improve our scores, we were allowed to take the Burington to exams.

[44] In a right handed coordinate system, it is clockwise. Many early computers saved resources by making the y-axis go down the screen instead of up the screen.

of a point that unrolls the circle on a line. Surely you have seen such an animation illustrating the meaning of the sine function.

The picture drawn by this animation[45] reveals a sine-graph only as an illusion. Most of the time, one wants to draw a polygonal line through the points spaced closely enough to give another illusion, that of a smooth curve. To trace the circle, connect successive values of `XR,YR` in the loop. In Applesoft BASIC the `HPLOT TO XR,YR` command connect the current point to the previous point drawn, a syntactic structure suitable for loops. The one point drawing command `HPLOT XR, YR` completes the plotting suite in Applesoft. The same idea is expressed in BASICGLUT by replacing the `glBegin(GL_POINTS)` by `glBegin(GL_LINE_STRIP);` bracketing the stream of point plotting commands.[46]

Exercise 13. Modify the Sinwheel to trace a "continuous" sine curve. Add color to the lines drawn. Generalize the concept to other functions. Make it interactive.

4 Functificator

There are two ways to go from here. We could continue the previous exercise until you can build a general function grapher. To this end we include an pocket program on automatic scaling. But a more interesting way of applying the trick in Sinewheel to visualizing Chaos is described in the next section.

A persistent problem with simple graphics systems is one of *scaling.* The screen coordinates, in pixel-units, are ill adapted for computing. One uses world coordinates for computation, and grown-up function graphers *autoscale.* That is, they automatically scale the function values to fit inside a prescribed rectangle on your screen, called a *viewport.* The Functificator, below, is such an autoscaler in BASIC. The following version served us well in the Apple Lab, see if you can implement it efficiently in a modern language.

[45] On slow computer the drawing proceeds slowly enough to understand what is going on. On a fast computer one would "slow it down" by putting a pause into the loop. If you don't know the correct syntax for the official pause function in the particular computer language at hand, use a long enough "empty" loop `FOR P=0 TO 10000 : NEXT` , for example.

[46] As with human languages, OpenGL syntax has several ways of modifying a stem. The `glBegin()`accepts a word as its argument telling it what to begin. But the vertex drawing stem `glVertex` has many variants. The one closest to `HPLOT` is `glVertex2f()` , the "2f" specifies that the arguments of the function will be the two (floating point) coordinates.

```
10 REM FUNCTIFICATOR
11 READ P, Q, A, B
12 DATA 9, 1, 0, 1
20 READ XM,  YM
21 DATA 300,200
30 DIM Y(XM)
39 DX = (B-A)/XM
40 FOR X = A TO B STEP DX
50 Y = X^P*(1-X)^Q
60 IF Y < MIN  THEN MIN = Y
61 IF Y > MAX  THEN MAX = Y
70 I=I+1 : Y(I)=Y
80 NEXT X
90 DY = (MAX-MIN)/YM
100 CLS
110 FOR I=0 TO XM
120 PSET(I,(Y(I)-MIN)/DY)
130 NEXT I
```

It graphs the function[47] (line 50)

$$y = x^p(1-x)^q,$$

whose parameters (the powers, p, q and the domain $[0, 1]$ of its independent variable, x) are set on lines 11 and 12. `XM` and `YM` hold the dimensions, in pixels, of the viewport. On line 30 BASIC allocates an array `XM` of values of Y, one for each pixel. There are exactly `XM` steps of size `DX` (line 39) on the interval `[A,B]`. We can write a loop as in line 40, using world coordinates, or, we could write it as a counted loop in pixel coordinates, as we do in line 110.

In theory, we need only one loop to draw the graph of the function. But, since we do not know *a priori* the range of the values, we use two loops. The first, 40–80, finds the minimum and maximum of the values at the same time, and we use these to scale the resulting graph into the available vertical space in the viewport. In BASIC, the first time the name of a variable is encountered it has value 0. Critique the way this loop determines the extreme values of a function. How might it fail and why doesn't it for the parabola?

Once we know the range of the y-values, we can scale them into the available

[47] These functions are of central importance in the construction of *splines*, which are curves whose shape are controlled by a few points on or near them. They serve as *weights* or *basis functions* for Bezier splines.

range, line 90. Note that substitution yields

$$\frac{\texttt{Y(I)} - \texttt{MIN}}{\texttt{MAX} - \texttt{MIN}}\texttt{YM}$$

for vertical screen coordinate in line 120.

Exercise 14. A linear mapping of a range $a \leq x \leq b$ into the range $A \leq X \leq B$ can be easily remembered from this formula

$$X = A\frac{b-x}{b-a} + \frac{x-a}{b-a}B.$$

A clever way of reading this expression is that when $x = a$ then $X = A$, and when x finally gets to b then the formula reduces to B. In between, the two fractions always add up to 1. Thus you can think of this as a weighted average of A and B. Incidentally, there is nothing that requires A, B to be numbers. They can be vectors, matrices, anything mathematical that adds and scales. Now, here's the exercise. Write a functificator that automatically scales a parametrized curve,

$$(x, y) = (f(t), g(t)),\ t_{min} \leq t \leq t_{max}$$

into an arbitrary *viewport*. To keep things interesting, try *Lissajoux* figures, where the functions are sinusoidal, $f(t) = \alpha\cos(\lambda t - \phi)$, with differeing amplitudes, α, frequencies, λ, and phase angles, ϕ.

5 Logistic Chaos

Another application of the figure-ground reversed manner of drawing a function graph you saw in `sinewheel` is this introduction to logistic chaos. You may not be familiar with the mathematics behind this progam, so we shall start with that.

Consider a population of something, say mosquitos in a swamp, which grow at a rate ρ. That is, the next generation of mosquitos is directly proportional to the current generation, $y = \rho x$. By substitution you can calcuate that successive populations, starting with x_0, comprise the geometric sequence,

$$x_0,\ \alpha x_0,\ \alpha^2 x_0,\ ...$$

which has one of three equally dull outcomes. If $\alpha \neq 1$, the mosquito population either dies out or increases indefinitely.

Since the latter is unlikely, we improve the model a tiny bit. Let us agree that there is a maximum possible population determined by the food available

in the swamp. This is called the carrying capacity of the environment. We take this to be 1, which automatically makes our population variable x a fraction of the *carrying capacity.* Thus normalized, we obtain a manageable geometric problem. We can express the effect of nearing the population limit by making the reproductive rate ρ itself proportional to how close we are to 1, $\rho = 4\alpha(1-x)$. Thus our equation becomes

$$y = 4 * \rho(1-x)x \ .$$

This is called a *logistic growth model.* The factor 4 is chosen so that the proportionality constant α exactly measure the maximum altitude of this parabolic arch which touches the x-axis at 0 and 1. Other authors absorb the 4 into the parameter.

You can discover some rudimentary properties of this system by drawing pictures accurately with pencil, paper and a right angle, a filing card for example. The trick is to plot the values of the logistic function not along the x or y-axes, but along the diagonal of the unit square. For example, starting from the input value $x = .33$ roughly a third of the way along the diagonal, find the point above or below on the parabola, and proceed horizontally to the output value $f(x)$. You are now ready to repeat the procedure forever, generating a discrete dynamical system based on the feedback loop $x \leftarrow f(x)$.

Exercise 15. With pencil, paper and ruler check that the population must become extinct for $\alpha < \frac{1}{4}$. For then the parabola remains below the diagonal, and every orbit steps its way to oblivion. Next convince yourself that for $\frac{1}{4} < \alpha < \frac{1}{2}$ the population converges to a single, positive, steady-state value, no matter where it begins. But past $\alpha > \frac{1}{2}$ life is not so simple.

We can speed up our graphical investigations with a computer. First we describe the essential part of our visualization, in BASIC. Then we describe an interactive extension, but now in BASICGLUT, which faithfully emulates the way it was originally written in BASIC. This gives a glimpse of the evolution of computer graphics languages.[48] **basicBASIC**

```
5 REM CHAOS
10 CLS
```

[48]The biological principle that "ontogeny recapulates philogeny" says that the stages an embryo of an animal of passes through mimics the stages of the animal's evolution. Perhaps this principle applies elsewhere too?

```
20 DATA 63, 63, .5, .01
21 READ XM, YM, A, D
90 X = .9
100 REM ETERNAL LOOP
140 Y = 4*A*(1-X)*X
150 XX=X*XM : YY = Y*YM
155 XY=X*YM : YX = Y*XM
160 LINE (XX,XY)-(XX,YY),1
165 LINE (XX,YY)-(YX,YY),1
170 X=Y : REM NEW REPLACES OLD
180 GOTO 100 : REM FEEDBACK
199 END
```

In order to play with this program, i.e. to make it interactive, you might proceed as follows. The line

```
25 INPUT "A= "; A
```

asks you for a value of `A`.[49] Can you change line 90 to accept a user chosen initial value?

Exercise 16. Apply the skills you have gained by completing the exercise for the Sinewheel program to add a subroutine to your Chaos program which draws the graph of the parabola.

```
10
30 INPUT "GRAPH? (Y?N) "; A$
35 IF A$="Y" THEN GOSUB 200 ELSE CLS
200 REM GRAPHIT
210 CLS
299 RETURN
```

The blank line 10 erases the clear-screen command, saving it for when you really want it at 35 or 210. The line 299 returns the program to the place from where the subroutine starting at line 200 was called. Of course, you need to write the parabolic graphing routine and perhaps the unit box and diagonal in the 200's.

basiCglut

We now turn to an example of how a pocket program can mature into a fairly useful mathematical tool. The following program is affectionately called *Allerton* after a conference in Allerton Park.[50]

[49] Incidentally, the keypress CONTROL-C will interrupt your program, and `RUN` will run it again. Of course, there are more elegant ways of controlling your program, even in BASIC.

[50] The conference, on using computers in mathematics instruction, was in the early eighties, a time when the University of Illinois began to switch form its pioneering, but

We shall analyse this code in the order it is written in the program. You should run the program on a computer while reading this. As an exercise, you could design a tutorial which tells a mathematical story *using* Allerton.

```
/* Logistic Chaos in GLUT, gkf 22jan2K on Linux, 10aug on Windows */
/* translated into basiCglut form the original 1984 Applesoft BASIC */
/* Allerton revised 26nov2K and 5jan01 */
#include <stdlib.h>
#include <stdio.h>
#include <gl\glut.h>
#include <windows.h>
#include <math.h>
#define  WIN      640           /* size of the square window */
#define  NAP      100           /* microseconds for ell() only */
#define  SLEEP(u) usleep(u)  /* usleep() in Linux, sginap() in Irix */
#define  FOR(i,a,b) for(i=a;i<b;i++)
int ii, jj, kk; float tmp, temp;
float xmax = WIN , ymax = WIN ,      /* screen size */
      x =.9, y = .9,  A = .99 ,      /* world variables */
      xx, yy, xy, yx,                /* screen variables */
      alt = .96,                     /* altitude of parabola */
      pnt = .9 ,                     /* starting point */
      del = .01 ;                     /* graph step size */
int clr =0,                           /* color index */
    til = 10,                         /* run ells til */
    nth = 1;                          /* iterate */
float rainbow[8][3]={ {1.0, 0.0, 0.0},        /* red */
                      {1.0, 0.5, 0.0},        /* orange */
                      {0.8, 0.8, 0.0},        /* yellow */
                      {0.0, 1.0, 0.0},        /* green */
                      {0.0, 8.0, 0.8},        /* (blue) cyan */
                      {0.0, 5.0, 1.0},        /* indigo */
                      {1.0, 0.0, 1.0},        /* (violet) magenta */
                      {1.0, 1.0, 1.0} };      /* white */
int hasnumber, number, decimal, sign ;  /* SLevy  bump gadget */
/*****************************************************************/
```

not portable, PLATO system to microcomputers. I had already translated this example from PLATO to Applesoft but gave my talk the old way, with transparencies. Tom Shilgalis brought an Apple computer to the conference and convinced me once and forever that a single live computer demo is worth a hundred slides.

```
void usleep(int nap){int ii; for(ii=0;ii< nap; ii++);} //a bad kludge
/*********************************************************************/
```

The foregoing preamble to *Allerton* is pretty much self-explanatory. The abbreviation `FOR(i,a,b)` (starting a counted loop) is easier to type than its more versatile counterpart in C. It is also less likely to be mistyped. The `rainbow` contains an approximation to those colors. You can tweak these RGB values to make the colors look better.

This program introduces Stuart Levy's gadget for writing into the graphics window and reading entries made there. It mimics the command lines at the bottom of the Applesoft high resolution screen. We will see how it works later.

From the backslash in `<gl\glut.h>`, and the `windows.h` header file in the include lines you can see this is the Windows version of Allerton. The pause function is system dependent, so we alias it with the macro `SLEEP()` . Not finding the Windows counterpart, we temporarily rewrote the `usleep()`as a long loop.[51]

The body of Allerton begins with the logistic equation. Here you could put additional feedback functions. Use a switch-case statement and a key-controlled choosing gadget to compare the different iteration schemes interactively.

```
/*********************************************************************/
float func(int nth, float x){
      FOR(ii,0,nth)x= 4 * alt*(1-x)*x; return x;
}
/*********************************************************************/
void  drawell(void){
      y=func(nth,x);     /* evaluate */
      xx = x*xmax ; yy = y*ymax;    /* world to screen coords */
      xy = x*ymax ; yx = y*xmax;
      glBegin(GL_LINE_STRIP);
      glVertex2f(xx,xy); glVertex2f(xx,yy); glVertex2f(yx,yy);
      glEnd();
      x = y;              /*feedback*/
```

[51] This is an example of a "hack" involving an inelegant solution, or "kludge".

```
      SLEEP(NAP);
}
/*********************************************************************/
void frame(void){  /* box with diagonal */
    int bot=1, top=WIN;
    glBegin(GL_LINE_STRIP);
    glVertex2f(top,top); glVertex2f(bot,top); glVertex2f(bot,bot);
    glVertex2f(top,top); glVertex2f(top,bot); glVertex2f(bot,bot);
    glEnd();
}
/*********************************************************************/
void wipe(void){glClear(GL_COLOR_BUFFER_BIT); glClearColor(0.,0.,0.,0.);
 }
/*********************************************************************/
void graph(void){ float x;   glColor3fv(rainbow[clr]);
     glBegin(GL_LINE_STRIP);
     for(x=0; x < 1+del; x += del)
         glVertex2f(x*xmax, func(nth,x)*ymax);
     glEnd();
}
/*********************************************************************/
void run(void){ glColor3fv(rainbow[clr]); FOR(jj,0,10){drawell();} }
/*********************************************************************/
```

Type the keys for (W)ipe, (F)rame, (G)raph, el(L), and (R)un to see what the four functions do. Look at the code to see why they do it. These four are also marked on the (H)elp button.

More help is on the (J) button. It refers to the six gadgets that display a parameter and accept input to change it. How Stuart Levy made this work *inside* the graphics window requires more explanation than we give here. If you are proficient in C, can you figure it out?

```
/*********************************************************************/
float getnumber(float dflt){      /* return new or default number  */
      if(!hasnumber)return dflt;
      tmp = sign ? -number : number;
      return decimal>0 ? tmp/(float)decimal : tmp ;
}
```

```
/*********************************************************************/
void graffiti(char strng[128], float par){  /* from avn by SLevy */
     char buf[128], *p ;
     glColor3f(0,0,0); glRecti(5,3,WIN,20);  /* erase old graffiti */
     glColor3fv(rainbow[clr]);  /* create and draw new graffiti */
     sprintf(buf, strng, par);
     glRasterPos2i(5,5);
     for(p=buf; *p; p++)glutBitmapCharacter(GLUT_BITMAP_8_BY_13, *p);
}
/*********************************************************************/
void altit(void) {
      alt= getnumber(alt); graffiti("%0.3f=(A)lt", alt); }
void point(void){
   x=pnt = getnumber(pnt); graffiti("%0.3f=(P)nt",pnt); }
void delta(void){
     del = getnumber(del); graffiti("%0.3f=(D)el",del); }
void until(void){
     tmp = getnumber((float)til); graffiti("%f=(T)il",tmp); til=(int)tmp; }
void power(void){
   tmp = getnumber((float)nth); graffiti("%f=(N)th",tmp); nth=(int)tmp;}
void color(void){
   tmp = getnumber((float)clr); graffiti("%f=clr",tmp); clr = (int)tmp; }
#if 0
void help(void){ fprintf(stdout,
     "(ESC)ape (w)ipe (f)rame (g)raph el(l) (r)un (h)elp \n \
      (a)ltitude (p)oint (d)elta (n)th (t)il (c)olor \n ",0);
#endif
void helpH(void){
     graffiti("(ESC)ape (W)ipe (F)rame (G)raph el(L) (R)un (H)(J)",0); }
void helpJ(void){
     graffiti("(A)ltitude (P)oint (D)elta (N)th (T)il (C)olor",0);}
/*********************************************************************/
```

gadgets

Both `getnumber()`and `graffiti()` are too advanced for now, and we explain only their operation. They are factors of the next six gadgets, all of which have the same estructure.

For example, the altitude of the parabolic arch is a parameter that many functions may want to use. So it is a global variable. In the gadget, `alt = getnumber(alt);` assigns a new value to the altitude if you typed one in

before pressing the (A). Otherwise, it retains the old value. Then it writes the current value (new or old) to the screen.

Now experiment with the (A)ltitude, the initial (P)oint, the stepsize (D)elta for the graph, the (N)th power the logistic is iterated before the next position is drawn. Check out the interplay of (N) and (G).[52] The (C)olor gadget is even trickier. Not only can input the color, but pressing (C) repeatedly cycles the colors to help you tell a story with Allerton.

Finally, there are three helpers in Allerton. The original `help()`is commented out in a peculiar way. The compiler is directed to ignore the code between `#if` and `#endif` because the "0" is always false.[53]

You have already met the `keyboard()` in earlier pocket programs. But this one is a step more sophisticated. For one, it has the rest of Stuart Levy's gadget. For another it chooses the display callback function for the GLUT library at your command.

keyboard

```
/*********************************************************************/
void cycle(int *par, int bas){
     if(hasnumber){*par = getnumber(0); return;} /* the 0 is a herring */
     *par = (*par + 1)%bas ;
}
/*********************************************************************/
void keyboard(unsigned char key, int x, int y){
#define PLOT(foo) glutDisplayFunc(foo); glutPostRedisplay();
  switch(key){
     case  27: fprintf(stderr," Thanks for using GLUT ! \n"); exit(0); break;
     case 's': {exit(0); break;}; /* stop */
     case 'w': {wipe(); break;};
     case 'f': {PLOT(frame); break;};
     case 'g': {PLOT(graph); break;};
     case 'l': { glColor3fv(rainbow[clr=(clr++)%7]);  /* cycle colors */
                 PLOT(drawell); break;};
     case 'r': {PLOT(run); break;};
```

[52]The gadget saying un(T)il when a (R)un of el(L)s should be drawn is "disabled" by having *hard-coded* the default value of `til` in the loop of `run()` . Change that!

[53]This is a useful abuse of the compiler directives. If you don't like graffiti and want to write to the command window instead of to the graphics window then use this instead of the other helpers. In Unix the `fprintf(stdout ...` works normally. In Windows you'll have to experiment.

```
     case 'h': {helpH(); break;};
     case 'j': {helpJ(); break;};
     case 'a': {altit(); break;};
     case 'p': {point(); break;};
     case 'd': {delta(); break;};
     case 'n': {power(); break;};
     case 't': {until(); break;};
     case 'c': {cycle(&clr,7); color(); break;};
   }
   glFlush(); /* superstition ? */
   /* Stuart Levy's gadget parser from avn.c 1998 */
   if(key >= '0' && key <= '9'){ hasnumber = 1;
         number = 10*number + (key - '0'); decimal *= 10;}
   else if(key == '.'){ decimal = 1;}
   else if(key == '-'){ sign = -1;}
   else {hasnumber = number = decimal = sign = 0;}
}
/*****************************************************************/
```

The "advanced" factor, `cycle()` , is used for the (C)olor cycler. In the keyboard function we define a macro `PLOT()`which does two things. It hands the GLUT-library to a new display function to "call back" when appropriate. And it tells the GLUT-library to update itself, in case it has fallen asleep.

The switch-case construction which turns your key-presses into action, can pretty much be puzzled out. Note the two ways of writing a cycling gadget.[54] The rest is advanced wizardry.

We now come the part that uses the GLUT-library.

```
/*****************************************************************/
void nothing(void){  } /* Glut3 forbids glutDisplayFunc(NULL); */
/*****************************************************************/
int main(int argc, char **argv){
  glutInitWindowSize(WIN, WIN);
  glutInitWindowPosition(10,10);
```

[54]In computer graphics there is a well-defined notion of a *widget* in a *graphical user interface*(GUI). A *gadget* is a poor man's widget. It gets the job done with a minimum of fuss, not a minumum of inconvenience. We will discuss gadgets at greater lengths in the second part of the course.

```
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_RGB);
  glutCreateWindow("<< Logistic Chaos in GLUT >>");
  glutDisplayFunc(nothing);
  glutKeyboardFunc(keyboard);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glOrtho(0,WIN,0,WIN,-10.0,10.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glutMainLoop();
  return 0;             /* ANSI C requires main to return int. */
}
```

6 Bouncing Shapes

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 simulations of 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.[55] 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.

7 Julia Set

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
```

[55] 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.

```
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.[56] 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 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

[56] 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.

outside the circle of radius two.[57] 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.[58] 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

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

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 */
```

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

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

```
#include <stdlib.h>
#include <stdio.h>
#include <gl\glut.h>
#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;
}
```

8 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