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 pseudocode because its simple syntax and 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 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 bf basicBASIC.

But the pocket programs do not need such an elaborate environment. Any language that can plot points and lines in a rectangle will do. They translate very nicely into JAVA.

*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.

[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 the 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.

```
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
```

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 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.

[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.

[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.

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.

That's all we need each time through the iteration, so close 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 appearing 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.

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. **basiCglut**

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 appropriat *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 pre-compiler. Other lines are for the compiler to read and process. Text we don't want the

[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. Try closing the window. More drastic measures may be needed.

[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.

compiler to read, such as comments, are between /* and */. The only othere compiler directive we use here is to define two 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 your 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]

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.

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 former as it appears in the latter. Both `keyboard()` and `main()` do have arguments, and in their definition we declare their type and a dummy name to be used for them in the body of the function definition. 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. Every C-program has one last function, invariably named "main" which is executed first. The `main()` of a program using the GLUT-library is quite 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 that when you press any key, the `keyboard()` function is called and executed. Otherwise, the `display()` function is called and executed as often as the graphics system on your computer can handle it. The `idle()` function is also executed "all the time".[12] Next, the subsidiary functions.

[10] In a customary overreaction, the architects of object-oriented languages, such as C and Java, frown on or forbid compiler macros. A similar reaction to abusing 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 title "Prof."

[12] That GLUT has two functions like that is a great help learning to program multi-processor multi-display systems like the CAVE.

`display()` uses a local integer variable, `ii`, as an index.[13] 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()`. In C, "every" commmand function, in that it has zero or more arguments, and returns something some value, which you generally ignore. The prefix "gl" tells you these are OpenGL functions. There are many graphics routines you will begin. Here it's `GL_POINTS`.[14]

`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.[15] Instead of the "if-then-else" syntax we use a "switch-case" statement here so you will learn that too. But it is easier to note how these are written and do likewise, than to follow a description. If you press the (ESC)ape key the infinite `glutMainLoop()` the GLUT-library is stuck 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.

We close this section with a brief demonstration why the gasket is a *fractal*.[16] 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.[17] 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

[13] 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".

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

[15] 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 computing.

[16] There is no agreement on the best definition of a fractal. The originally it is is a set with a fractional dimension, as defined by Haussdorff. Because it is sometimes impossible, and usually impractical to compute this dimension less demanding definitions are in common use.

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

of a similar figure obtained by doubling its linear scale,[18] then

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

Exercise 1. To show that you have understood the concept of bufferering data, write pseudo-code fragment[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 you know provided you can demonstrate that it really works.

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.

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. Be careful that you write the matrix multiplication so as not to stall[21] BASIC.

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. Now 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.

[18] 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,

$$|A(2)| = 3|A(1)|.$$

Why?

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

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

[21] Well, perhaps not stall, but reduce to a crawl.

```
/* 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 */
    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. */ }
```