Mathematics in Computer Graphics

Kevin Wampler

When I was asked to give a presentation here on mathematics within computer graphics, I was at a bit of a loss at what to cover. The problem was not that I couldn't think of what I could cover, it was rather that I was faced with having to decide what *not* to cover. Computer graphics, aside from the mechanics of actually coding the solution one has discovered to a problem, is in many ways simply the problem of creating art with mathematics. Thus almost *any* method used in computer graphics (and there are a great many of them) employs a mathematical formalization of something desired in art.

The use of mathematics in computer graphics generally falls into one of two admittedly fuzzy categories: Cases of computers being used to visualize mathematical objects and cases of math being used to model other phenomena.

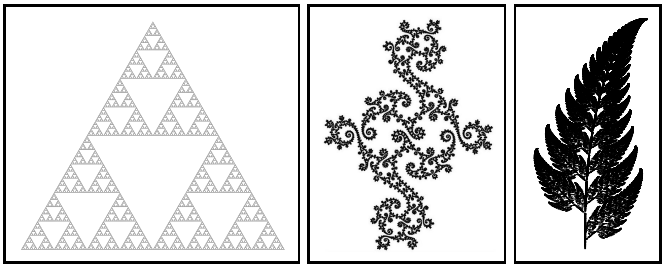

Figure 1: various limit sets [4, 3]

In some cases the mathematics employed in the two cases are very similar. Consider for example the three figures appearing at left. At first glance they appear to be mathematical, ornamental, and natural in appearance. The progression between then would be taken to be one from mathematical visualization to mathematical modeling. In reality they are all the same kind of mathematical object: Limit sets of certain groups (or semigroups) of two dimensional transformations. The leftmost figure, a Sierpinski triangle, and the rightmost fernlike figure are actually very similar to eachother mathematically.

Computer graphics is greatly useful in mathematical visualization; it is, after all no coincidence that the growth of the study of chaos, dynamical systems and fractals has largely coincided with the use of computers to study them. Unfortunately, due to limited time and in the interest of presenting a more focused and manageable stream of information, I will be primarily focusing on the use of mathematics as a tool for computer graphics, rather than computer graphics as a tool for mathematics, though some of the techniques discussed will have an obvious application to mathematical visualization.

There are at least two broad categories in which mathematics is useful in computer graphics: Modeling, and Rendering. Modeling is concerned with the creation of a mathematical object which has certain properties which we find desireable, for example having a shape similar to a flower. Rendering, on the other hand, deals with how to convert these mathematical objects into an actual picture. Both of these are surprisingly diverse fields, so I am forced to present only a small sampling of their contents. I hope that even this small portion helps to give you some feel for the relationship between mathematics and computer graphics.

1 Modeling

1.1 Plants

The picture of the fern leaf above leads naturally to the question of what about a fern leaf lends itself to mathematical modeling. It turns out that this critical property is *self-similarity*. Each of the leaves on the fern has the same shape as the whole fern. Mathematically, we can devise a mapping which will take a the fern and transform it so that it perfectly covers one of its leaves. It turns out that in this case there are four basic ways of doing this. These four mappings are given by the affine transformations [5]:

$$T_1(\mathbf{v}) = \begin{bmatrix} 0 & 0 \\ 0 & 0.16 \end{bmatrix} \mathbf{v},\ T_2(\mathbf{v}) = \begin{bmatrix} 0.85 & 0.04 \\ -0.04 & 0.85 \end{bmatrix} \mathbf{v} + \begin{bmatrix} 0 \\ 1.6 \end{bmatrix},\ T_3(\mathbf{v}) = \begin{bmatrix} 0.2 & -0.26 \\ 0.23 & 0.22 \end{bmatrix} \mathbf{v} + \begin{bmatrix} 0 \\ 1.6 \end{bmatrix},$$
$$T_4(\mathbf{v}) = \begin{bmatrix} -0.15 & 0.28 \\ 0.26 & 0.24 \end{bmatrix} \mathbf{v} + \begin{bmatrix} 0 \\ 0.44 \end{bmatrix}$$

The rules for how one of these transformations acts on a two dimensional point are simple:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix} = \begin{bmatrix} ax + by + e \\ cx + dy + f \end{bmatrix}$$

All that is needed to define the all of the self-similarities in the fern is to combine these four transformations. So we will now have the transformations T_1T_1, $T_3T_1T_4$, $T_2T_2T_4T_1$, etc. All of these transformations, together along with the ways in which we combine two transformations to form a new one is called the *semigroup* generated by $\{T_1, T_2, T_3, T_4\}$. Any element in this semigroup will always transform a point in the fern onto another point in the fern. It furthermore turns out that if we consider *any* two dimensional point, the fern is the limit set the orbit of that point under the actions of members in the aforementioned semigroup. The practical upshot of this is that to draw the fern, we need merely pick any two dimensional point. We randomly choose T_1, T_2, T_3, or T_4, transform the point with it, draw the new point on the screen and repeat. This takes only a few lines of computer code to do, impressive for such a striking result.

If we think about the process used to create the fern leaf in more general terms, we can devise more powerful methods of creating certain shapes. We started out with a single point, and then recursively defined that if $\mathbf{v}$ is in the set, then so are $T_1(\mathbf{v})$, $T_2(\mathbf{v})$, $T_3(\mathbf{v})$, and $T_4(\mathbf{v})$ [1]. This notion of recursively defining a set easily created a set with a very interesting and apparently complicated structure, and thus it is not surprising that it appears elsewhere in mathematics.

One of the places where it appears in the the definition of mathematical notation. Normally, we take the syntax of mathematics for granted, and just assume that we know that parentheses should be balanced, a multiplication sign should only occur between two certain kinds of subexpressions etc. Sometimes in mathematics, for instance in symbolic logic, it becomes important to precisely specify exactly what constitutes a valid mathematical expression. The process to do this is very like that by which we drew the fern. We start out with some small set of valid elements. In the case of propositional calculus this might be all of the letters. Then we define that if α and β are valid sentences, then so are $(\neg\alpha)$, $(\alpha \wedge \beta)$, $(\alpha \vee \beta)$, $(\alpha \Rightarrow \beta)$, and $(\alpha \Leftrightarrow \beta)$. This is all that is needed to define all of the allowed sentences in prepositional logic. Even more interesting is that the very notion of proof itself is defined in a very similar manner. When applied to language (in the general sense of a language consisting of certain finite strings of symbols) such a way of defining things is called a *grammar*. So what was given above was a grammar specifying the language of prepositional calculus.

Grammars also have an unexpected use in modeling plants. In fact, grammars have proved so useful in this task that they provide the standard way of modeling a plants. The particular way in which grammars are used if this situation was developed by the Swedish biologist Aristid Lindenmayer and are called L-systems (short for Lindenmayer systems). The key insight in the developmet of L-systems was that strings of symbols could be used to easily create certain shapes. This is done through what is called the *turtle interpertation* of strings. The idea here is that we can have a *turtle* read the string and walk around (possibly) drawing a line as it goes according to the letters it encounters. The letters which are given meaning in a string are (in the most basic case) F, f, $+$, $-$, [, and]. Their interpertations are:

`F` Move forward for some predefined distance d, drawing a line as you go.

`f` Move forward for some predefined distance d, but do not draw a line.

`+` Turn right by some predefined angle δ

`-` Turn left by some predefined angle δ

`[` Save the current state (position and orientation) of the turtle. This is used to begin a branch.

`]` Restore the last saved state of the turtle. This is used to end a branch.

[1] The astute reader may notice that this is sufficient only to define a countable subset of the fern, whereas the fern itself is clearly of an uncountable cardinality. For the purposes of computer graphics this is unimportant, since computers can only operate on finitely enumerated sets and the countable subset has the same structure as the full fern.

In a manner very similar to the definition of the language of mathematics, an L-system takes a word as a starting point and then defines certain transformations which recursively defined a new new word in terms in terms of another already known word. This is done by defining replacements for certain letters. For example, consider the L-system given by [1]:

$n = 4$, $\delta = 90^o$
`F-F-F-F`
`F` $\rightarrow$ `FF-F--F-F`

The starting string for this L-system is `F-F-F-F`. The next string is generated by replacing every `F` in this string by `FF-F--F-F`, resulting in `FF-F--F-F-FF-F--F-F-FF-F--F-F-FF-F--F-F`. The $n = 4$ indicates that this is to be done four times. When interpreted by a turtle, with $\delta = 90^o$, this final string creates the picture:

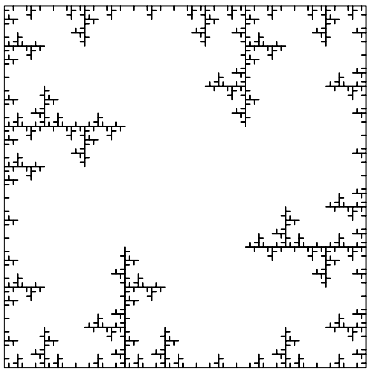

Figure 2: an L-system [2]

A more plantlike L-system is given by [1]:

$n = 7$, $\delta = 20^o$
`X`
`X` $\rightarrow$ `F[+X]F[-X]+X`
`F` $\rightarrow$ `FF`

If `X` is interpreted identically to `F` by the turtle, this produces the picture:

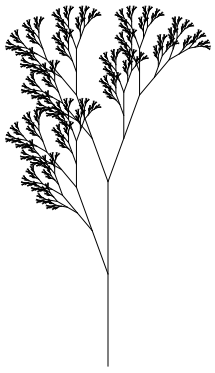

Figure 3: a tree-like L-system [2]

These L-systems are quite simple, and still produce interesting results. Using more advanced techniques, it is possible to use L-systems to create very realistic plants.

1.2 Quaternion Julia sets

L-systems provided one very useful and powerful way of generating certain forms, many of which were fractal in nature. There are, however, other ways of generating certain fractal shapes. One if the most well known of these (at least where 2D fractal images are concerned) is by using *Julia sets*[2]. A Julia set is a subset of the complex numbers. For each complex number c these exists a Julia set which is defined to be the set of all points, z_0, for which the function $z_{n+1} = z_n^2 + c$ does not go to infinity when repeatedly applied[3]. Some of these sets have quite interesting shapes, but unfortunately the process in limited to two dimensions. It is often (probably usually) the case in computer graphics that we wish to do something in three dimensions, and in many cases a standard Julia fractal will not do. The remedy to this, of course, is to compute Julia fractals in higher dimensions. This is not as trivial as it first sounds. The problem is that to compute a Julia fractal in higher dimensions, we need a generalization of complex numbers that works in higher dimensions. What we are really looking for is an *extension* of the complex numbers (in particular what is called a field extension). An extension of a structure is simply a superstructure in which the initial structure exists unaltered. So in this sense, the real numbers are an extension of the rational numbers and the complex numbers are an extension of the real numbers. Normally, we also want an extension of a structure to have all of the "nice" properties of that structure as well. With the complex numbers, there are several such nice properties:

1. multiplication and addition are commutative
2. multiplication and addition are associative
3. all numbers have additive and multiplicative inverses

It has been shown that any extension of the complex numbers to higher dimensions cannot have all of these properties, so the task is instead to find extensions which have as many of these properties as possible. One of the most common ways is with the structure known as the *quaternions*. Quaternions have all of the above mentioned properties except for commutativity of multiplication. Surprisingly, they are not three, but four dimensional. Each quaternion, h can be written as a linear combination of four basis numbers, 1, i, j, and k where $i^2 = j^2 = k^2 = ijk = -1$. Less abstractly, 1, i, j, and k can be represented as the 2×2 complex matrices [6]:

$$\mathsf{U} \equiv \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \mathsf{I} \equiv \begin{bmatrix} i & 0 \\ 0 & -i \end{bmatrix}, \mathsf{J} \equiv \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}, \mathsf{K} \equiv \begin{bmatrix} 0 & i \\ i & 0 \end{bmatrix}$$

We can now define a four dimensional Julia set in the same way as we defined a two dimensional one, but by using quaternions instead of complex numbers. Since the results are four dimensional we still need to find a way to reduce this to three dimensions. This is typically done by displaying only a three dimensional slice of the four dimensional fractal.

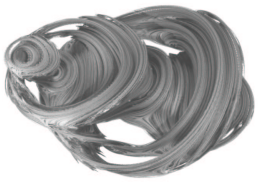

Figure 4: A Quaternion Julia set [7]

[2] In actuality, the use of Julia sets in computer graphics is not terribly common, especially since very few graphics programs (POV-Ray being one notable exception) support them. They still, however, remain a easy and mathematically interesting of creating certain shapes.

[3] The definition of a Julia set is actually more general than this, which should more properly be called a quadratic Julia set. A Julia set is actually defined for each rational function R as the set of points z_0 for which $z_{n+1} = R(z_n)$, does not go to infinity.

1.3 Isosurfaces

If we again think more generally, this time about the definition of a Julia set, we can come up with a way of defining shapes which is very useful in computer graphics. In general terms, we defined a Julia set to be the set of all points which had a certain property, namely when some function was applied to them, the result remained small. A similar and more general way of defining surfaces is for each function $f : \mathbb{R}^3 \to \mathbb{R}$ to define a surface which is the set of all points, v for which $f(v) \leq 0$. A surface defined in such a manner is called an *isosurface*. Isosurfaces are very versatile and useful in modeling a great variety of shapes, from mountains to clouds to a coffee cup. One of the simplest examples of an isosurface (given here in two dimensions, rather than three) is a disk, which is defined by the equation:

$$\sqrt{x^2 + y^2} - R \leq 0 \quad [4]$$

One aspect of the power of isosurfaces is seen with the ease by which this equation can be modified to create more interesting variations on a disk:

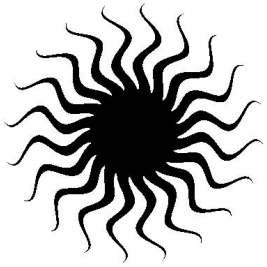

Figure 5: $\sqrt{x^2 + y^2} - \frac{1}{2} - \left(\frac{tri(\frac{20atan(x,y)+2sin(10\sqrt{x^2+y^2})}{\pi})+1}{2} \right)^2$

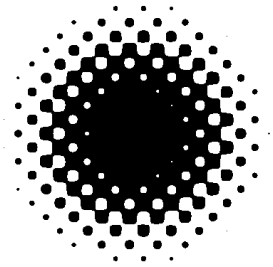

Figure 6: $\sqrt{x^2 + y^2} + \frac{1}{2}sin(20x)cos(20y) - 1$

If used well, isosurfaces provide a very mathematically elegant and powerful way of modeling a great many types of shapes. They are often particularly useful in modeling natural objects, as these often have a definite mathematical structure, but one which is hard to describe in terms of geometric primitives (spheres, boxes, cones, etc.). Furthermore, the idea behind isosurfaces ban easily be extended to cover volumetric objects, such as clouds and smoke.

1.4 Parametric surfaces

Although powerful, isosurfaces have several notable limitations. Greatest among these are that they are hard to controll locally, they are hard to texture in certain ways, and that they only lend themselves to certain

[4]The square root used here is not necessary, as the same surface is defined by $x^2 + y^2 - R^2 \leq 0$. It is used here for the sake of consistency with later equations.

rendering algorithms (typically not the fast ones either). These two issues are remedied by *parametric surfaces*, although these in turn have their own disadvantages. While an isosurface defined the interior and exterior of an object, a parametric surface only defines the surface of an object. This is done by defining a function $f : \mathbb{R}_{[a,b]} \times \mathbb{R}_{[c,d]} \to \mathbb{R}^3$. The surface specified by such an equation is simply the image of $\mathbb{R}_{[a,b]} \times \mathbb{R}_{[c,d]}$. For example, a sphere is defined by the function:

$$f(u,v) = (cos(u)cos(v), cos(u)sin(v), sin(u)), \text{ where } (u,v) \in \left[-\tfrac{\pi}{2}, \tfrac{\pi}{2}\right] \times [0,\pi]$$

To render an picture of such a parametric surface, we merely consider small squares, or more typically triangles, in the domain of the function and then draw squares/triangles between the images of the corners of the original square/triangle. If the squares are made small enough, the resulting picture will very closely approximate the original surface. This method of rendering an object as being composed of many triangles is quite fast and is used, for example, by 3D graphics cards. Furthermore, since a parametric surface is defined in terms of a two dimensional rectangle, it is easy to use the equation for the surface to map pixels on a two dimensional image onto the three dimensional surface. This results in an image which wraps around the surface, a technique which is used very often in texturing a three dimensional model. Furthermore, certain functions which interpolate between a set of points or curves are easily represented as parametric equations. There is class of such functions which are called *splines* and are used extensively in the modeling of a great many shapes (like humans, cars, etc.)

References

[1] Prusinkiewicz, Przemyslaw and Lindenmayer, Aristid. The Algorithmic Beauty of Plants. New York: Springer-Verlag New York Inc, 1990.

[2] http://www.cs.unh.edu/ charpov/Programming/L-systems/

[3] http://www.mcs.vuw.ac.nz/ jfouhy/postscript/

[4] http://klein.math.okstate.edu/IndrasPearls/RoadMap/

[5] http://mathworld.wolfram.com/BarnsleysFern.html

[6] http://mathworld.wolfram.com/Quaternion.html

[7] http://astronomy.swin.edu.au/ pbourke/fractals/quaternionjulia/

University of Arizona

wamplerk@u.arizona.edu