Smooth Self-Similar Curves

Craig S. Kaplan
Cheriton School of Computer Science
University of Waterloo
csk@uwaterloo.ca

Abstract

I present a technique for constructing self-similar curves from smooth base curves. The technique is similar to that used in Iterated Function Systems like the Koch curve, except that it does not require a piecewise linear path in order to induce a set of similarities. I explain the mathematical machinery behind the technique, describe a practical numerical approximation that can be implemented in software, and show some results.

1 Introduction

Fractals have long enjoyed a central position in the world of mathematical art. Many classical examples take the form of fractal curves generated from Iterated Function Systems; perhaps the best known of these is the Koch curve. I begin by recapitulating the construction of this curve as a means of motivating the technique presented in the rest of this paper.

Begin with a unit line segment. Divide the segment into three equal pieces and replace the central piece with the other two edges of an equilateral triangle erected upon it. I refer to this four-segment piecewise linear path as the "base curve" (see Figure 1, left). Let T_1, T_2, T_3 and T_4 represent the four similarities of the plane (compositions of translations, rotations, and uniform scalings) that carry the original unit segment to the four segments of the base curve. The fixed set of these transformations (that is, the unique nonempty set S of points satisfying $S = T_1(S) \cup T_2(S) \cup T_3(S) \cup T_4(S)$) forms the final fractal curve [1].

In practice, such curves can be elaborated geometrically to any desired level of detail through a recursive substitution process, in which straight segments are replaced by transformed copies of the base curve. For example, the Koch curve can be drawn via the following pseudocode:

```
procedure KOCH( T, level ):
    if level = 0:
        DRAWLINE( T(0,0), T(1,0) )
    else:
        for i = 1 to 4:
            KOCH( T ∘ T_i, level−1 )
```

The curves derived from the Koch base curve after the first few rounds of substitution are shown in Figure 1.

This drawing algorithm can easily be extended to any piecewise linear base curve beginning at $(0,0)$ and ending at $(1,0)$, as long as all of its segments have length less than 1. Furthermore, there are four choices of similarity for each segment of the base curve, since we may incorporate either or both of a reflection across the segment and a reflection across its perpendicular bisector (see Figure 2).

To what extent can this process be adapted to a *smooth* base curve? A smooth curve does not decompose naturally into straight segments that induce similarities. But if the curve has a well defined tangent everywhere, then we

Figure 1 : *The construction of the Koch curve. On the left, a unit line segment is shown with the base curve above it. The other three drawings show the evolving curve after one, two and three rounds of substitution.*

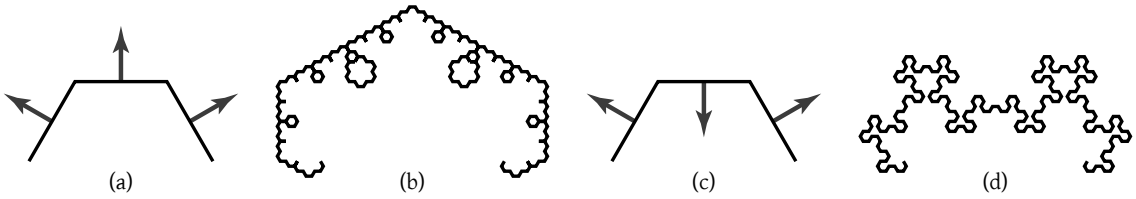

Figure 2 : *A simple example of fractal curve variations based on changing the similarities mapping onto some segments. Two base curves are shown in (a) and (c), with segment orientations labelled by arrows. The middle segment in (c) is flipped relative to that of (a). The two resulting fractal curves in (b) and (d) are very different.*

can define non-linear warps of the plane into a constantly shifting coordinate system made up from the curve's tangents and normals.

In this paper, I present a mathematical technique for constructing self-similar curves, by analogy with the drawing algorithm given above (Section 2). I then describe a software tool that draws numerical approximations of these curves (Section 3). My primary goal is the creation of a practical tool for exploration and artistic inspiration, though I do conclude by discussing some of the mathematical questions evoked by this work, in addition to the aesthetic possibilities (Section 4).

2 Mathematical technique

In this section, I present a mathematical framework that can construct smooth analogues of fractal curves. I begin by describing the elementary step of "curve hoisting" (Section 2.1), which plays the role of the similarities in the construction of the Koch curve, and then move on to the iterated use of hoisting to create self-similar curves to any desired level of detail (Section 2.2).

2.1 Curve hoisting

Curve hoisting refers to the process of redrawing one curve in the local coordinates defined by another curve. We may think more generally of hoisting any geometry onto a curve, a well known process in computer graphics (used, for example, to warp text to curved paths in illustration software). Here I present a version of hoisting specialized to the iterated process given in Section 2.2, including relevant details about boundary constraints.

Let P be a curve in the plane. P can be represented as a function $P(t) = (p_x(t), p_y(t))$, which produces a point for every t in the interval $[0,1]$. For the purposes of this work, we will require curves to satisfy three constraints:

1. $P(0) = (0,0)$ and $P(1) = (1,0)$.
2. P has a well defined tangent direction everywhere. That is, $P'(t)$ exists and is non-zero for all t in $[0,1]$. (Such curves are known as C^1 *immersions*.)

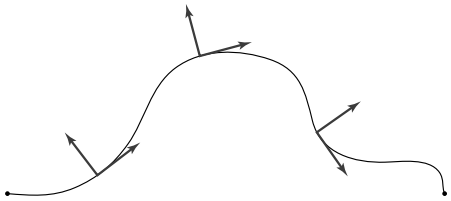

Figure 3 : *A visualization of three frames constructed from tangent and normal axes at various points on a smooth curve.*

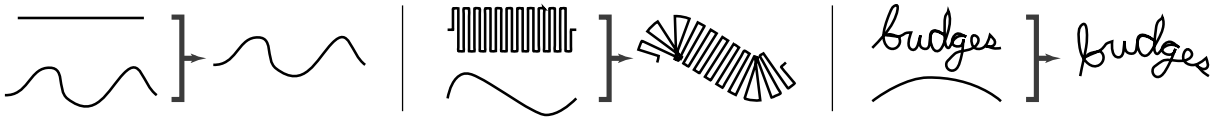

Figure 4 : *Three examples of hoisting a path onto a smooth curve. In each of the three examples, an arbitrary path (top left) and a smooth curve (bottom left) play the roles of Q and P, respectively, in Equation 1. They yield the hoisted results shown on the right in each of the three examples.*

3. P has a constant-speed parameterization; that is, $|P'(t)|$ is constant. With such curves, the distance along the curve from $P(0)$ to any $P(t)$ is proportional to t, making it easy to sample the curve by arclength.

In general, analytical curves with constant-speed parameterization are impractical to work with directly. However, smooth curves can easily be approximated with a finite sequence of position and tangent samples, from which arclength sampling can be approximated (see Section 3.1).

Because such curves have well defined tangents everywhere, it is possible to imagine a 2D coordinate frame erected at every point on the curve. A point $P(t)$ becomes the origin of the coordinate frame. The "tangent axis" is given by $P'(t)/|P'(t)|$, a unit vector in the direction of the tangent. The "normal axis" $\mathcal{N}(t)$ is the unit vector 90 degrees counterclockwise from the tangent axis. Figure 3 shows a sample curve annotated with a few coordinate frames.

Now let $Q(t) = (q_x(t), q_y(t))$ represent any other continuous path in the plane, with $Q(0) = (0,0)$ and $Q(1) = (1,0)$ (we do not require Q to satisfy the constant-speed or tangent constraints imposed upon P). The machinery above allows us to re-express the shape of Q in terms of the continuously roving coordinate frame of P, a process that I will refer to as "hoisting Q onto P". I define the hoisted curve H using the following equation:

$$H(t) = P(q_x(t)) + q_y(t)\mathcal{N}(q_x(t)) \tag{1}$$

The goal of this equation is to consider a point $Q(t_0)$ relative to a "reference line segment" with endpoints $(0,0)$ and $(1,0)$. The horizontal position of this point gets translated into a position along P; its vertical position gets translated into an offset from P. When considered in this way, the reference segment hoisted onto P would yield P itself. Three examples of hoisting are shown in Figure 4.

One immediate problem with this formulation is the legitimate possibility that $q_x(t) < 0$ or $q_x(t) > 1$ for some values of t. The equation for H cannot handle such cases as given, since $P(q_x(t))$ is only meaningful when $0 \leq q_x(t) \leq 1$. We can resolve this issue by extending the definition of P so that it can be evaluated for any real t. There are several possible ways to carry out this extension. Four are equivalent to the segment transformations mentioned in Section 1—we concatenate copies of the curve, alternately incorporating one or both of a reflection in the x axis and a reflection in the y axis. I name these modes **Repeat**, **Reflect Y**, **Reflect X** and **Rotate**. In addition, we can have a **Clamp** mode in which $P(t)$ lies on the x axis for all t outside $[0,1]$. These five modes are illustrated in Figure 5.

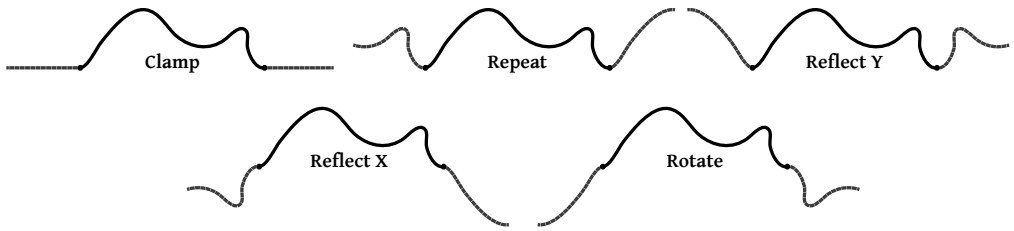

Figure 5 : *Five boundary extension modes for sampling a curve outside of the domain* $[0,1]$. *Each is constructed by concatenating transformed copies of the curve or by clamping to zero.*

In **Clamp** mode, care must be taken to ensure that a particle travelling with constant speed along P will continue with the same speed beyond it. Letting L represent the total arclength from $P(0)$ to $P(1)$, define $P(t) = (Lt, 0)$ for $t < 0$ and $P(t) = (L(t-1), 0)$ for $t > 1$.

Note that of the suggested extension modes, only **Rotate** is guaranteed to produce a smooth curve. In the other cases, the curves will have well defined tangents at points $(n, 0)$ for all integers n if and only if P has horizontal tangents at its endpoints.

2.2 Iterated hoisting

Based on the pseudocode in Section 1 for drawing the Koch curve via substitution, we can now use hoisting to perform an analogous iterated substitution process with smooth curves. Let a curve $P(t)$ be given satisfying the constraints at the beginning of Section 2.1, and let $m > 1$ be an integer (the "multiplicity"). We would like to subdivide P into m equal-length portions by arclength, and hoist a copy of P onto each portion. An easier way to envision this process is to assemble m consecutive copies of P, and then scale the resulting path so it begins at $(0,0)$ and ends at $(1,0)$. This path can then be hoisted onto P all at once using Equation 1.

We might imagine that to handle a second iteration, we could assemble m^2 copies of P and hoist that onto the first generation curve. This approach is problematic, however, because it is difficult to preserve tangent and normal information through the hoisting process. But there is no impediment to working in the other direction, always hoisting onto P. We arrive at the following simple algorithm:

```
function ITERATE( P, m, gens ):
    C = P
    for idx in 1...gens:
        C = HOIST( MULTIPLY( C, m ), P )
    return C
```

In the pseudocode above, MULTIPLY(C, m) lays out m copies of the curve C and scales the result uniformly by $1/m$, and HOIST(A, B) hoists path A onto smooth curve B as explained in the previous section. An example of iterated hoisting is given in Figure 6.

This approach is well suited to a software implementation. A fully analytical approach would require a hoisting method that could compute the tangent at every position along the hoisted curve. In the algorithm above, when C is initialized on the second line it can be set to a piecewise linear approximation of P. The approximation can then be carried through every iteration, so that we are always hoisting a piecewise linear path onto a smooth curve.

As mentioned in the introduction, the MULTIPLY function need not necessarily lay out copies of its input curve via translation alone. In theory, each of the individual copies of the curve could be untransformed, reflected across the x axis, reflected across the y axis, or reflected across both axes (corresponding to a halfturn of the curve).

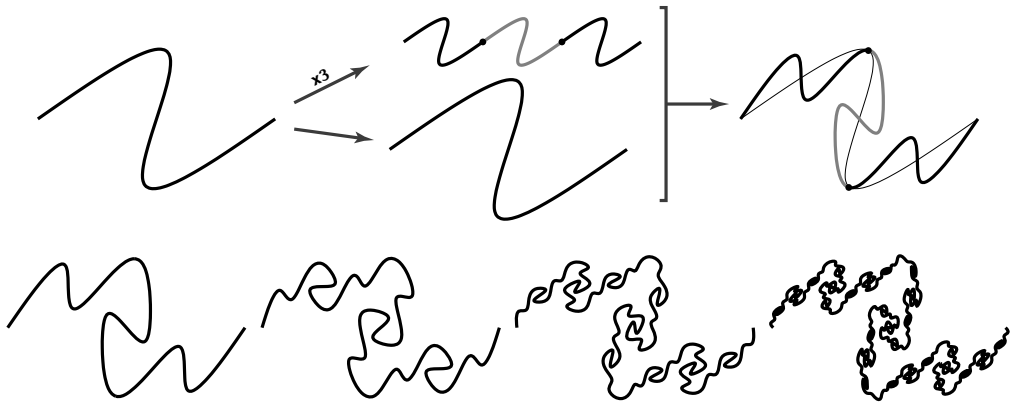

Figure 6 : *An illustration of the iterated hoisting process for generating a self-similar curve. The top row shows the original curve, which is then "multiplied" by three and hoisted onto a copy of itself to yield the first generation of substitution. The bottom row shows the first four generations of the curve.*

This flexibility opens up new creative possibilities for self-similar curves. In Section 3.2, I discuss some of the simple alternatives I have explored.

It is also possible to introduce flexibility by allowing for non-integer multiplicities. If m is not an integer, we can still assemble m copies of a curve, where the final copy might consist of only an initial segment of the curve. In this case the end of the multiplied curve might not lie at $(1,0)$; I then scale and rotate the curve to restore this requirement.

3 Implementation

The mathematical theory presented here is sound, but it relies on an analytical view of curves that can never be fully realized algorithmically. Inevitably, the theory must give way to a numerical approximation that allows us to construct the expected result with sufficient accuracy. In this section I discuss the details of this technique's software implementation, and describe the user interface I have constructed to interact with it.

3.1 Algorithmic details

Any implementation will ultimately rely on a concrete curve implementation. We would like to use smooth curves that are easy and intuitive to manipulate. The typical standard in computer graphics is cubic polynomial spline curves [5, Chapter 15]. For experimentation purposes, I implemented B-Splines and Catmull-Rom splines. B-Splines have better continuity properties across curve segments, but the curve only approximates its control points; Catmull-Rom splines interpolate (pass through) their control points but do not have continuous second derivatives.

The major difficulty with cubic splines (and, indeed, most other non-trivial curves) is that they do not support constant-speed parameterization. The formula for the arclength of a cubic curve derives from an integration, and can not be given in a closed form. Therefore I approximate the spline curve using a discrete set of sample positions along it. Each sample records a position on the curve and the tangent direction at that position. The curve can then be (approximately) sampled at any fraction of arclength by linearly interpolating position and tangent information from the two nearest samples. I sample the curve adaptively: the sampling process is controlled by a "flatness" parameter that recursively adds samples in regions of high curvature. It is this re-sampled curve that

plays the role of P in the algorithm of Section 2.2.

The successive values of C in the iterative algorithm can use a simplified version of the same approximation. C is initialized to an adaptive sampling of the base curve, keeping only position samples. This piecewise linear path is then multiplied and hoisted onto P to yield a new C. The iteration process can continue to any desired number of generations, always yielding a polygonal approximation of the true curve.

3.2 User interface

I have created a small Java program to explore the space of self-similar curves generated using this algorithm. The core algorithm is relatively short; most of the code is used to construct the user interface. The interface shows a base curve (either a B-Spline or a Catmull-Rom spline) with all of its control points. All control points can be edited interactively except the first and last, which remain fixed to ensure that the curve begins at $(0,0)$ and ends at $(1,0)$. Superimposed upon the base curve is the final curve produced after a user-specified number of iterations.

A panel of controls allows the parameters governing the editing, iteration and display process to be manipulated interactively. In addition to controls for the multiplicity and the number of iterations, I have implemented the following:

- **Boundary constraints:** a selector that decides which method to use when sampling P outside the range $[0,1]$. As described in Section 2.1, the choices are **Clamp**, **Repeat**, **Reflect Y**, **Reflect X** and **Rotate**.
- **Segment constraints:** a selector that chooses a pattern of transformations to apply to successive copies of the curve in the MULTIPLY step of the algorithm. In theory, every copy of the curve could be assigned one of four transformations independently (as noted in Section 2.2). In this interface, I support modes called **Repeat** (all segments untransformed), **Reflect Y** (alternate segments reflected through the y axis), and **Rotate** (alternate segments rotated through a halfturn). With more implementation work, the user could be permitted to assign a transformation to each segment individually.
- **Editing constraints:** a selector for symmetry constraints applied when manipulating curve control points. Aesthetically pleasing fractal curves frequently have base curves that are either bilaterally or rotationally symmetric. The **Mirror** and **Rotate** options force curve control points to move in synchrony, guaranteeing that the resulting base curve has the desired symmetry.

When a satisfactory curve has been created, it can be exported as a Postscript drawing.

The simple adaptive sampling process I use will never *reduce* the number of points in the curve. If the initial C in the algorithm contains n samples, then after k iterations with multiplicity m the new version of C will have at least nm^k points. It can therefore be useful to rebuild this final curve as a spline using a smaller number of control points. I use the Simplify tool in Adobe Illustrator for this final step.

Some results are shown in Figures 7–11, highlighting the effect on the final curve as various controls are modified.

4 Discussion

In undertaking this work, my primary goal was to develop a working algorithm, and not to determine the analytical properties of the resulting curves. Nevertheless, the resulting technique raises some interesting questions. The most immediate avenue for further exploration is to study under what circumstances this process can be shown to converge. If it does converge, what sort of shape do we obtain in the limit? If these curves possess anything resembling self-similarity, it must be approximate (as in the Mandelbrot set) and not exact (as in the Koch curve). Is there a workable notion of fractal dimension?

Figure 7 : *An example showing how a self-similar curve changes as a single control point moves. The base curve is a B-Spline. The control point marked by an arrow moves northeast. The successive curves show the results after the marked point moves by equal intervals.*

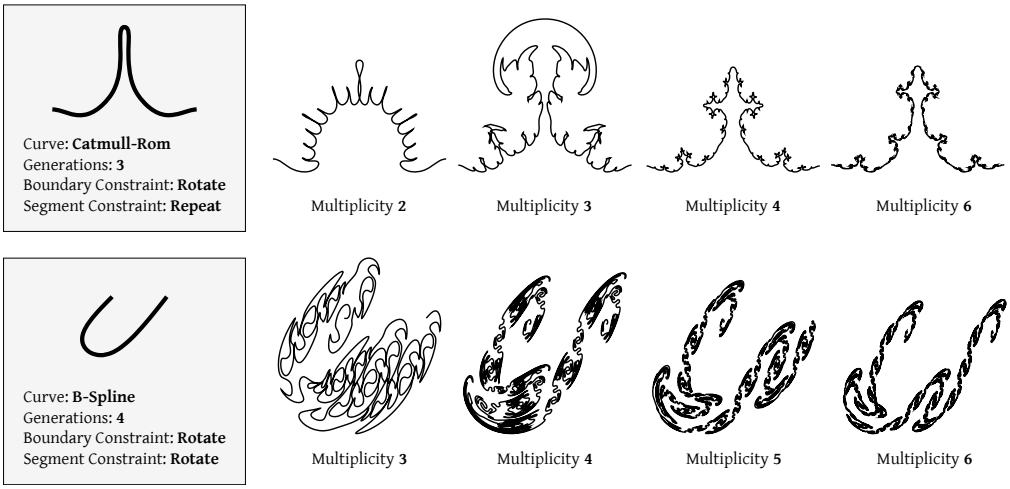

Figure 8 : *Two examples showing the effect of varying the multiplicity. Two base curves are shown (each with its relevant parameters). Each curve is then elaborated over a few generations at different multiplicities.*

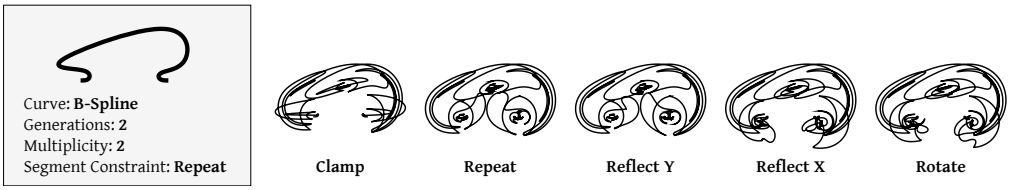

Figure 9 : *An example showing the effect of varying the boundary constraint. The base curve extends beyond the $x = 0$ and $x = 1$ lines in the plane, which will cause the boundary constraints to be used starting with the first substitution. The result is shown using each of the different boundary modes.*

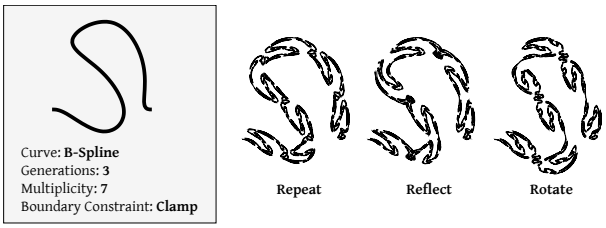

Figure 10 : *An example showing the effect of varying the segment constraint.*

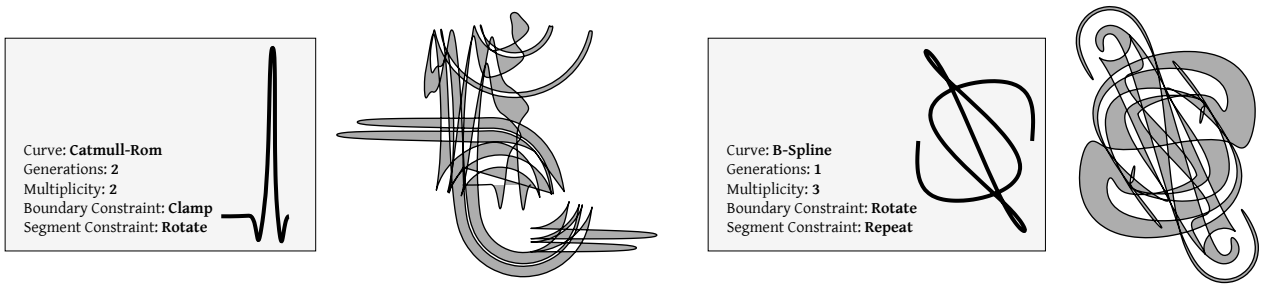

Figure 11 : *Two aesthetically guided experiments in using self-similar curves for design. In both cases the resulting curve is filled (using an even-odd fill rule). In the example on the right, the curve was edited manually to render it closed.*

Mathematical questions aside, this work can also be seen a design tool for producing attractive planar paths for art or design. To some extent humans have an innate aesthetic response to self-similarity, one that has sustained fractals as an artistic medium for decades. We enjoy interpreting a design as having information at a range of spatial frequencies; previous work by Finkelstein and Salesin [2] and Hertzmann et al. [3] allowed for direct control of curve content at different frequencies.

The self-similar curves presented here frequently have continuous derivatives, a property not seen with typical piecewise linear Iterated Function Systems. Curved fractal forms have certainly been featured in previous work; for example, see Mandelbrot's *Monkey Tree Curve* [4], or McKenna's *Thirteenski*. However, in those examples the curviness was applied as a post-process to a piecewise linear path after fractal substitution was complete.

As with other examples of using fractals for design, some points in the parameter space are more successful than others. For example, symmetry in the base curve leads naturally to a symmetric final curve, which can be interpreted as having a more cohesive design. Also, self-avoiding curves tend to endure as classic fractals; more work is needed to understand the constraints on the base curve that will produce a self-avoiding result. Even when the self-similarity of the process is suppressed by the choice of base curve, multiplicity and number of generations, the algorithm can still produce interesting and unexpected curves (see Figure 11). Either way, this technique provides one more opportunity for the design of aesthetic mathematical form.

Acknowledgments

Doug McKenna and Robert Fathauer provided some helpful thoughts early in the process of preparing this paper. Thanks also to the anonymous reviewers, particularly John Sullivan, for their constructive feedback.

References

[1] Michael F. Barnsley. *Fractals Everywhere*. Morgan Kauffman, second edition, 2000.

[2] Adam Finkelstein and David H. Salesin. Multiresolution curves. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, SIGGRAPH '94, pages 261–268. ACM, 1994.

[3] Aaron Hertzmann, Nuria Oliver, Brian Curless, and Steven M. Seitz. Curve analogies. In *Proceedings of the 13th Eurographics workshop on Rendering*, EGRW '02, pages 233–246. Eurographics Association, 2002.

[4] Benoît B. Mandelbrot. *The Fractal Geometry of Nature*. Times Books, 1983.

[5] Peter Shirley. *Fundamentals of Computer Graphics*. A K Peters, second edition, 2005.