Bounding Iterated Function Systems via Convex Optimization

Orion Sky Lawlor

Department of Computer Science
University of Illinois at Urbana-Champaign

Abstract

We present an algorithm to construct a bounding polyhedron for an affine Iterated Function System (IFS). Our algorithm expresses the IFS-bounding problem as a set of linear constraints on a linear objective function, which can then be solved via standard techniques for linear convex optimization. As such, our algorithm is guaranteed to find the optimum recursively instantiable bounding volume, if it exists.

Key words: Iterated Function Systems, Bounding, Convex Optimization, Linear Programming

1 Introduction

Iterated Function Systems (IFSs) are a widely-used representation for fractal shapes. Much prior work has gone into constructing bounding volumes for IFSs using spheres, including the seminal work by Hart and DeFanti [3], who needed bounding volumes in order to raytrace IFSs.

The definitive work on sphere bounding volumes is that of Rice [5], who finds the smallest bounding sphere by searching over sphere centers using a generic nonlinear optimization package. Our work, though superfically dissimilar, was inspired by this approach, and follows much of his development.

1.1 Iterated Function Systems

An Iterated Function System (IFS) consists of a finite set of functions w_m, which map a space to itself. The Hutchinson operator H is defined as the union of each of the maps w_m. That is, given a subset of space B,

$$H(B) = \bigcup_{m \in M} w_m(B)$$

Under certain conditions on the w_m,[1] it can be shown that repeated applications of H always converge to a unique attractor A. That is, starting with any bounded nonempty set B, $H^\infty(B) = A$. This attractor is the heart of the IFS, and can be suprisingly structured and beautiful, as shown in Figure 1.

We can now establish our fundamental theorem, the recursive bounding theorem, which states that if a shape bounds its image under the operator H, then the shape bounds the attractor.

[1]A sufficient condition is that each w_m be Lipschitz contractive.

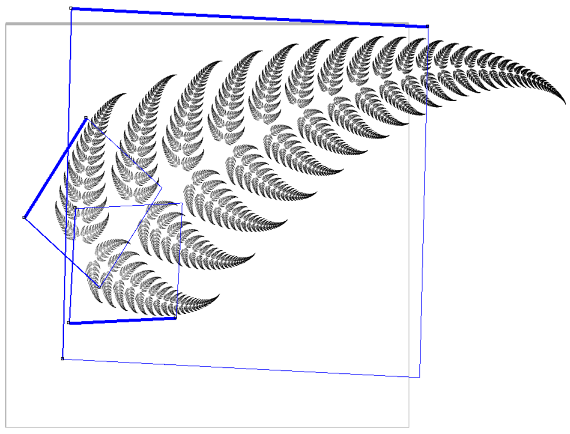
Figure 1. The attractor of a 3-map iterated function system.

Theorem 1.1. *Given a convergent IFS, if B bounds its image under H, B bounds the attractor. That is, if a subset of space B satisfies*

$$\forall_{m \in M}\, w_m(B) \subset B$$

then $H^\infty(B) \subset B$.

Proof: Our proof proceeds by induction on applications of H.

The base case is trivial because by definition $H(B) = \bigcup_{m \in M} w_m(B)$, and by hypothesis each $w_m(B) \subset B$, so because $H(B)$ is the union of shapes bounded by B, $H(B) \subset B$.

For the inductive step, assume for some k, $H^k(B) \subset B$. But then if we define $C = H^k(B) \subset B$, we find that because $C \subset B$, each $w_m(C) \subset w_m(B)$, so

$$H(C) = \bigcup_{m \in M} w_i(C) \subset \bigcup_{m \in M} w_i(B) = H(B) \subset B$$

Thus $H(C) = H^{k+1}(B) \subset B$.

By induction, then $H^\infty(B) \subset B$. $\square$

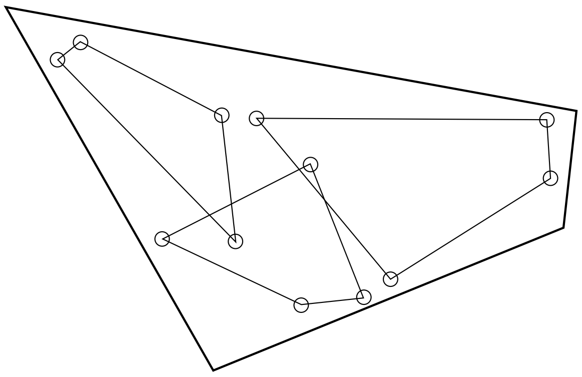

Figure 2. We ensure containment by checking each corner of the hull under each map.

2 Approach

Our basic approach will be to bound the IFS attractor using a bounding hull built from the intersection of a set of halfspaces. Using the recursive bounding theorem, we will guarantee that the attractor lies within the hull by guaranteeing "containment"—that is, by ensuring that under each map w_m, the map of the hull lies within the original hull.

In general, the map of a polyhedral hull is no longer polyhedral; so checking containment can be quite difficult. Luckily, we are normally interested in affine maps in ordinary Euclidian space, so the map of a convex polyhedral hull is still a convex polyhedron. As such, we can guarantee containment by requiring that each corner of the polyhedron (that is, each intersection of the polyhedron's sides), under each map, satisfies all the halfspaces of the hull, as shown in Figure 2.

2.1 Method

Our bounding hull is a convex polyhedron, and hence consists of a set S of halfspaces. Our halfspaces consist of an outward-facing normal $\vec{n}_s$, represented as a row vector, and a scalar displacement d_s. Then we can determine if a point $\vec{x}$, represented as a column vector, lies inside the halfspace by examining the dot product

$$\vec{n}_s\,\vec{x} \le d_s$$

In 2D, two halfspaces i and j intersect at a point $\vec{x}_{ij}$ if the point simultaniously satisfies the equations of both halfplanes, that is, if

$$\begin{bmatrix} \vec{n}_i \\ \vec{n}_j \end{bmatrix} \vec{x}_{ij} = \begin{bmatrix} d_i \\ d_j \end{bmatrix}$$

To guarantee containment, and hence apply the recursive bounding theorem, we have to make sure each map of each intersection satisfies each of the halfspaces. That is, given a set of intersections I, maps M, and halfspaces S, we require

$$\forall_{(i,j)\in I,\, m\in M,\, s\in S} \quad \vec{n}_s\, w_m(\vec{x}_{ij}) \le d_s \qquad (1)$$

2.2 Linearity of Constraints

We now need to show how the constraints in equation 1 can be made suitable for use in a linear optimizer. We first note that if we fix the normals and define the matrix

$$\mathbf{N}_{ij} = \begin{bmatrix} \vec{n}_i \\ \vec{n}_i \end{bmatrix}^{-1}$$

then $\vec{x}_{ij}$ is a linear function of d_i and d_j

$$\vec{x}_{ij} = \mathbf{N}_{ij} \begin{bmatrix} d_i \\ d_j \end{bmatrix}$$

In 3D, we can similarly define a matrix $\mathbf{N}_{ijk}$ to compute the intersection of three halfplanes given their displacements d_{ijk}.

Since we assumed the maps w_m were affine, we can represent each map w_m as a matrix $\mathbf{W}_m$ and a shift vector $\vec{s}_m$, as in

$$w_m(\vec{x}) = \mathbf{W}_m\,\vec{x} + \vec{s}_m$$

We can now expand out equation 1 and verify that our constraints are now linear in the displacements

$$\forall_{(i,j)\in I,\, m\in M,\, s\in S}\, \vec{n}_s \left(\mathbf{W}_m \mathbf{N}_{ij} \begin{bmatrix} d_i \\ d_j \end{bmatrix} + \vec{s}_m\right) \le d_s \quad (2)$$

This final, linear form of our constraints is suitable for direct use in a constrained linear optimization package; where the displacements d_i are our unknowns and everything else is fixed.

A linear optimization system will also require an objective function. We normally want the "smallest" bounding hull, and the exact choice of the objective function is determined by exactly what we mean by "smallest". Area is a natural choice, but unfortunately the area of the hull is a nonlinear function of the displacements, so we cannot directly minimize area. Another natural choice is to minimize the largest displacement, which is easy to achieve by the standard technique of adding an additional variable to represent the maximum displacement. Our choice for an objective function is to minimize the sum of the displacements, which is quite simple and should give results similar to minimizing either area or maximum displacement.

Finally, we can keep the displacements non-negative by expressing the maps w_m in a coordinate system where the displacment origin always lies within the hull. For example, we can place the origin at one of the maps' fixed points, since each of the maps' fixed points must lie within the attractor.

2.3 Extension to RIFS

In a Recurrent Iterated Function System (RIFS), we do not apply the individual maps randomly. Instead, after applying map i, we are only allowed to apply map j if there is an edge in the "control graph" G between the nodes representing maps i and j. A theorem by Barnsley [1] shows the attractor for a RIFS can be described as a set of overlapping "attractorlets."

A coarse bound for an RIFS can be found by simply ignoring the control graph G, in effect converting the RIFS into an ordinary IFS. As usual, this requires $|M||I||S|$ constraints.

A tighter bound can be obtained by independently bounding each of the attractorlets with a separate convex bounding hull. The constraints for such a system would look identical to the constraints seen previously, but since each attractorlet must be bounded separately, this will require $|M||S|$ unknowns and $|M|^2|I||S|$ constraints.

3 Limitations

Our bounding hull approach has several limitations.

3.1 Corners

Because our bounding hull has corners, it may not be possible to bound an IFS by simply increasing the hull's size. For example, consider a single-map 2D IFS that consists of a 30-degree rotation together with a very gentle contraction. The attractor for this IFS consists of a single point. However, no 4-sided hull can recursively bound this IFS, because the hull's corners will always stick out—see Figure 3. Increasing the sides of the box does not help, because rotation and scaling are scale-independent.[2]

The solution to this corner problem is to add sides—in this case, a 12-sided hull will match itself under rotation, and can then be shrunk exactly to the attractor.

In 2D, if the angle between two hull corners is α, the corners will never stick out if all the map contraction factors s_m satisfy $s_m < \cos\alpha/2$, as can be seen in Figure 4. This means if the largest contraction factor of any map is s, no corner will protrude if we use at least $h = 2\pi/\alpha > \frac{\pi}{\cos^{-1} s}$ sides.

For contraction factors very close to 1, this means we may require an unaffordably large number of sides to acheive any bound; but as we will show, in practice most IFS only require a very small number of sides.

3.2 Fixed normals

To implement the algorithm as a linear optimization problem, we first choose fixed normal directions for the faces of our convex hull. A natural choice, which we have

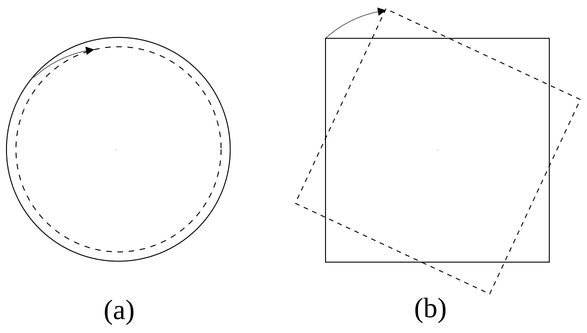

Figure 3. For a one-map IFS, which rotates and slightly scales space, the sphere bound (a) fits nicely; but the corners of a 4-sided box (b) will never fit inside the box.

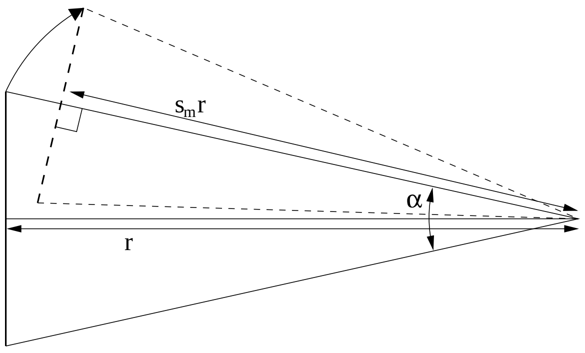

Figure 4. Corners never stick out if $s_m < \cos\alpha/2$.

[2]We did not consider translation here, because by choosing a big enough bounding volume we can make any finite translation seem small.

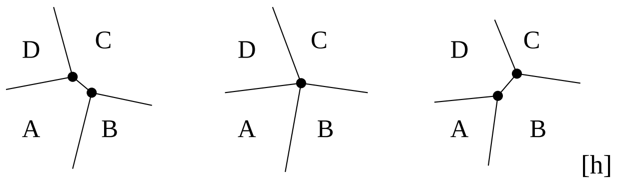

Figure 5. In 3D, vertices that are the intersection of 4 faces, as shown in the center, can be shifted to have two different sets of intersections, as shown on the left and right.

made, is to pick equally spaced directions.

However, it is often possible to find a smaller hull via a better choice of normal directions; although optimizing over normals is a nonlinear optimization problem. Choosing normal directions is the subject of ongoing work.

3.3 Fixed intersections

We must also pick the vertices on the convex hull where faces intersect. In 2D, this is simple–every two adjacent convex hull edges intersect at a hull vertex, and any other edge intersections will lie outside the hull and can be safely ignored. The only way the set of intersections can change is if three edges intersect at a point, in which one of the edges vanishes; in this case we end up checking the same point twice, but this causes no problems.

But in 3D, the set of points on the hull boundary can change. For example, if four faces A, B, C, and D intersect at a point, perturbing the faces one way leaves the intersections ABC and ADC on the hull; while another perturbation leaves ABD and BCD on the hull, as shown in Figure 5. Since we have to choose the set of intersections beforehand, we cannot allow more than three faces to intersect at a point.

4 Implementation

We implemented this convex optimization bounding method for 2D IFS. We wrote a C++ program to generate constraints for the widely available convex linear optimization package lp_solve [2]. In this section, we examine the results and performance of this implementation of the method.

4.1 Example

Figure 6 shows the coarse bounding volume computed by our implementation using just 8 sides; Figure 7 shows the much tighter bounding volume computed using 30 sides.

In both cases, the bounding volume indeed appears to be optimal, in that any smaller volume with the same number and orientation of sides would not recursively bound the attractor. Both figures also lie quite close to the actual attractor.

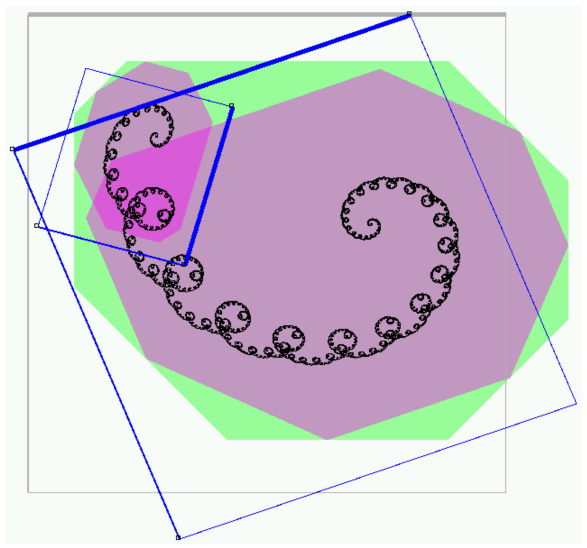

Figure 6. An IFS and attractor, with an 8-sided bounding volume computed by the algorithm.

4.2 Performance

Given an IFS with M maps, and a bounding volume consisting of I intersections between H halfspaces, we will need exactly MIH constraints, of the form given in equation 2, and H unknowns. In 2D, the number of intersections is equal to the number of halfspaces, so this is $O(MH^2)$ constraints.

Although there polynomial-worst-case algorithms for solving linear programs exist, such as Karmarkar's [4]; the package we used, `lp_solve`, is based on the well known exponential-worst-case simplex method. The relationship between the number of sides and the run time is shown in Figure 8; the algorithm's total run time appears to be approximately $O(H^{4.6})$. Large numbers of sides are thus computationally infeasible; but a couple dozen sides can be computed quickly. Luckily, as shown below, more sides than this are rarely required.

The experimental relationship between the number of sides and the area of the resulting hull is summarized in Figure 9. Because we distribute the side normals evenly, the area plot jumps up and down as useful normals are found and then passed by. As can be seen in the plot, in practice a fairly small number of sides suffices to bound most IFSs.

5 Conclusions

We have presented an algorithm based on convex linear optimization for constructing an optimal convex bounding volume for the attractor of an Iterated Function System or Recurrent Iterated Function System.

The algorithm is easy to implement and runs at interactive rates for small problems.

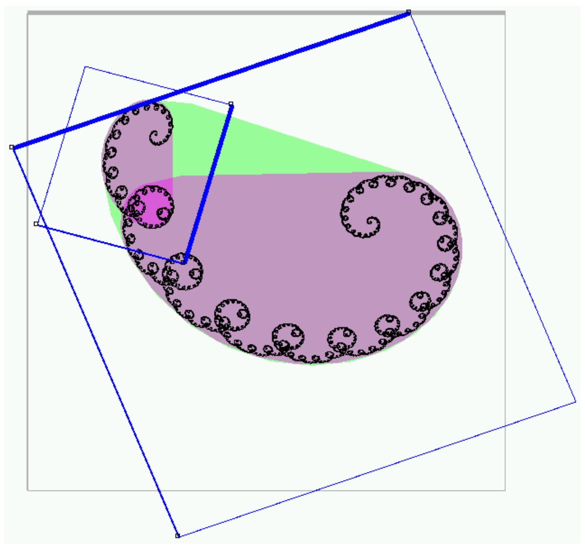

Figure 7. The same IFS using a 30-sided bounding volume.

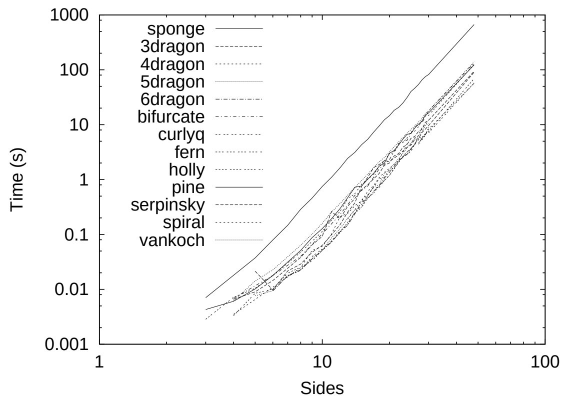

Figure 8. Log-log plot of the time to determine the optimal bound for a variety of sides for a variety of IFS. Runs on a 1.3 GHz AMD Athlon PC running Linux.

6 Future Directions

We have only implemented this bounding technique in 2D. The 3D version should be virtually identical, and would allow us to experiment with raytracing IFS attractors using our computed convex bounding volumes.

Choosing the side normals is a difficult problem, and should be done automatically somehow. Many approaches for this ”normal search” seem promising: we might begin with a small number of sides, then add sides where they seem necessary; or begin with a large number of sides and prune those that are not useful. It may even be possible to make good a priori normal estimates by examining some feature of the individual IFS maps.

There are a variety of possibilities to improve the speed of the algorithm. It may be possible to take advantage of the spatial structure of this problem’s constraints to speed up the optimization process; or use another solver for convex linear optimization.

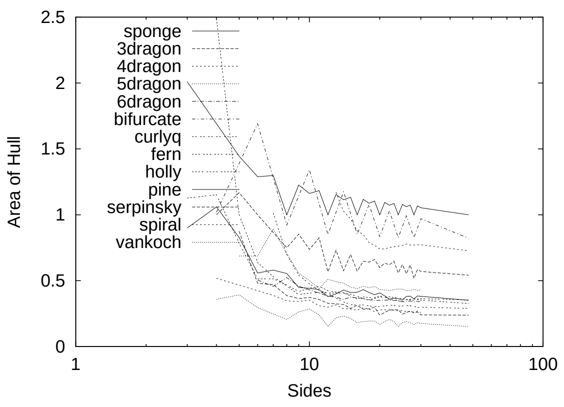

Figure 9. Log-linear plot of the area of the bound found by the algorithm for a variety of sides and a variety of IFS.

References

[1] M. F. Barnsley, J. H. Elton, and D. P. Hardin. Recurrent iterated function systems. *Constructive Approximation*, 5:3–31, 1989.

[2] Michel Berkelaar. Mixed integer linear program solver lp_solve. Available from `ftp://ftp.ics.ele.tue.nl/pub/lp_solve/`.

[3] J.C. Hart and T. A. DeFanti. Efficient anti-aliased rendering of 3d linear fractals. In *SIGGRAPH ’91 Proceedings*, pages 91–100, July 1991.

[4] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–396, 1984.

[5] Jonathan Rice. Spatial bounding of self-affine iterated function system attractor sets. In *Graphics Interface*, pages 107–115, May 1996.