Extension of algorithmic geometry to fractal structures

Anton Mishkinis

SPIM

Thèse de Doctorat

école doctorale **sciences pour l'ingénieur et microtechniques**

UNIVERSITÉ DE BOURGOGNE

Extension des méthodes de géométrie algorithmique aux structures fractales

ANTON MISHKINIS

SPIM

Thèse de Doctorat

école doctorale **sciences pour l'ingénieur et microtechniques**

UNIVERSITÉ DE BOURGOGNE

THÈSE présentée par

ANTON MISHKINIS

pour obtenir le

Grade de Docteur de
l'Université de Bourgogne

Spécialité : **Informatique**

Extension des méthodes de géométrie algorithmique aux structures fractales

Soutenue publiquement le 27 novembre 2013 devant le Jury composé de :

MICHAEL BARNSLEY	Rapporteur	Professeur de l'Université nationale australienne
MARC DANIEL	Examinateur	Professeur de l'école Polytech Marseille
CHRISTIAN GENTIL	Directeur de thèse	HDR, Maître de conférences de l'Université de Bourgogne
STEFANIE HAHMANN	Examinateur	Professeur de l'Université de Grenoble INP
SANDRINE LANQUETIN	Coencadrant	Maître de conférences de l'Université de Bourgogne
RONALD GOLDMAN	Rapporteur	Professeur de l'Université Rice
ANDRÉ LIEUTIER	Rapporteur	"Technology scientific director" chez Dassault Systèmes

Contents

Chapter 1

Introduction

1.1 Context

Our research is conducted as part of the project ANR ModItère. The global objective of this project is to develop a new type of geometric modellers for computer-aided design systems (CAD), based on iterative methods following the principle of fractal geometry.

In CAD systems, objects are often modelled based on the classic matching process, that is, by combining different simple parts. Defining shapes by iteration allows us to generate new structures with specific properties, such as roughness or lacunarity, which cannot be achieved with classic modelling. However these types of objects are conceivable and may be of interest: porous structures can be used for lighter objects while maintaining satisfactory mechanical properties, rough surfaces can be used for acoustic absorption (see figure 1.1).

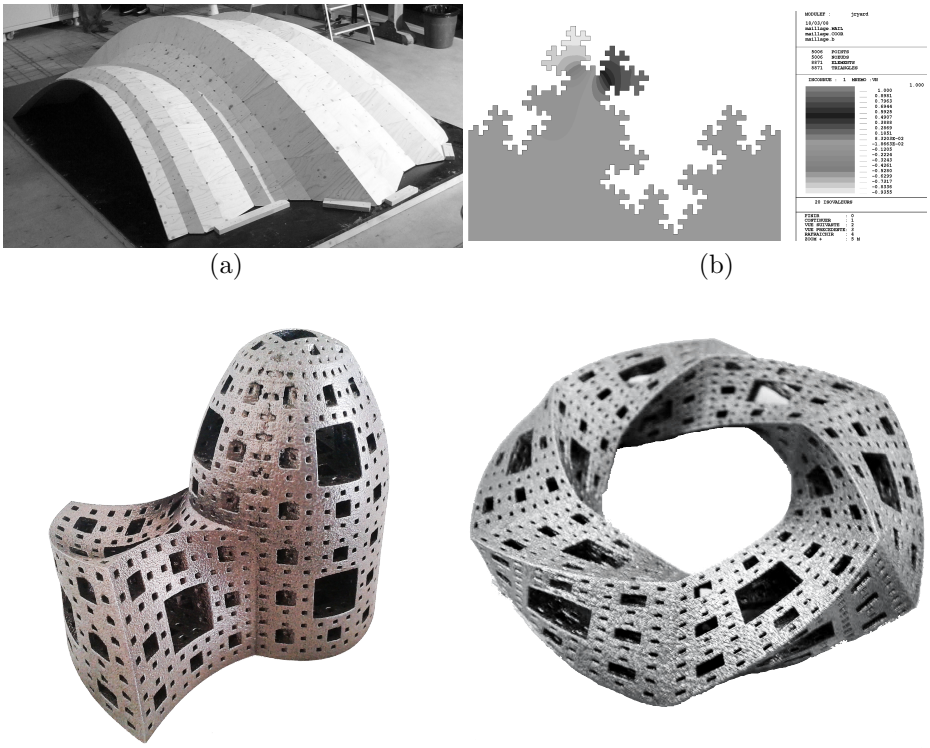
We propose to elaborate a set of modelling tools to adapt the fractal model, keeping the facilities of existing CAD systems, while extending their capabilities and application areas. Thus we provide algorithms for CAD systems to design new kind of shapes based on the concept of fractal geometry in order to model and fabricate these shapes.

1.2 Problems

Fractals are self-similar shapes composed of several parts, each of which is similar to the entire shape. In mathematics, fractals are sets of points in a metric space with a non-integer dimension (in the sense of Minkowski or Hausdorff metric), that is, the metric dimension differs from the topological one.

Most of existing CAD modellers are based on classic geometry. They permit modelling analytic curves and surfaces, which are usually polynomial or rational and smooth. The shapes modelled by iterative methods are generally not smooth because of their fractal structure. The topology can be controlled in order to build curves and surfaces represented by parametrized functions, but these functions are usually nowhere differentiable.

Generally, it is difficult or impossible to represent these shapes using classic geometric models, although this iterative construction principle is exploited to evaluate certain objects, such as subdivision surfaces [WW02, Lan04]. Specific



(a) (b)

(c)

Figure 1.1: Several applications of fractal structures. (a) Example of application in architecture, EPFL 2010 (b) The study of acoustical properties of fractal structures (Emmanuel Redon LVA LYON I) (c) Examples of fractal shapes fabrication (Project ANR ModItre).

iterative models were developed to generalize the answer to this problem: L-system [PL90], IFS [Bar88, Gen92], LRIFS/CIFS [PH94], BCIFS [TBSG$^+$06]. These models are mainly used to generate fractal structures in computer graphics, but also to compress data [BS87, Fis95]. In this thesis, we use the term fractal to denote a shape represented by these iterative models.

An Iterated Function System (IFS) is determined by a set of transformations. The attractor of an IFS may be evaluated by iterating these functions. A Controlled Iterated Function System (CIFS) is a more general system that allows us to control certain parts of the IFS attractor. A CIFS denotes an IFS with restrictions on the transformation sequences imposed by a control graph and defines objects whose geometry can be complex. The attractor of a CIFS can be evaluated by an automaton [MW88] defined on the control graph. Each state of the automaton corresponds to different parts or sub-parts of the modelled object. Transitions between states indicate that one sub-part is contained in another one. It is then possible to control the attractor more precisely. However the evaluation, analysis and characterization (location, size) of the resulting shape can be complex.

From IFS and CIFS, various shapes can be modelled. However, it is difficult to control their topological properties. Shapes are determined by a set of geometry operators. Modifying these operators leads to both global and local changes in the shape, this affects not only geometry but also topology.

To separate global and local controls, attractors can be constructed in barycentric spaces and then projected to a set of control points in the modelling space [SLG05, Tos06, Gou09, ZT96]. The global shape can therefore be modified by moving the control points, while the local geometry (geometric texture) is defined by subdivision operators, i.e. the CIFS transformations.

In order to control the topological structure of the modelled shapes, a CIFS is enriched by integrating a topological model to obtain a Boundary Controlled Iterated Function System (BCIFS) [TBSG$^+$06]. It is thus possible to specify incidence and adjacency constraints [Tos06, Gou09] for the subdivision and thereby to control the resulting topology: classic topology (curve, surface,...) or fractal topology. The topological model in this case is more general than the classic one [Gou09]. A topological cell may be a fractal object. For example, a face can be a Sierpinski triangle or an edge can be a Cantor set, but the topological structure remains consistent.

Fractal shapes can thus be modelled using BCIFS, but shape modelling in CAD systems is complex, that is not confined to simply representing the shape. To assist the user, geometric modellers are enriched by constructing operations and algorithms. These operations are usually based on geometric properties of modelled objects. Since for fractal shapes these properties are more complex and more subtle [Ben09, BGN08] and sometimes even not defined, existing CAD systems are not suitable to manipulate fractal structures. On the other hand, Zaïr and Tosan [ZT96] showed that the result of certain operations can be obtained formally. For example, a tensor product of two fractal curves can be obtained by forming the tensor product of BCIFS operator matrices.

Today, there is no CAD software to design or to model fractals. This is justified by the fact that this problem was not really of interest. Indeed, there was no manufacturing process able to produce such objects. However, the emergence of new techniques, such as stereo-lithography or laser fusion, opens up new possibilities yet unused and even unexplored.

The passage of the theoretical model to the physical object requires a number of treatments. For example, curves and surfaces must be thickened (dilated with respect to HAUSDORFF distance) to become 3-dimensional. This must be done without loss of rendering accuracy in order to preserve the aesthetics and geometric properties such as topology (lacunarity) and differentiability (roughness). The generated shapes must thus have a good translation into the physical world.

The goal of our project is to develop a set of tools and algorithms that permits one to evaluate, to characterize and to analyse different geometric properties (localisation, convex hull, volume, fractal dimension) of fractal shapes. It is thus necessary to be able to evaluate a set operations (intersection, union, ...) between fractal objects and classic CAD objects (splines, NURBS, cylinders, ...). Finally, in order to fabricate these objects, it is also important to take into account the manufacturing constraints. The idea is to integrate these constraints as early as possible, i.e. during design. This leads to the following advantages:

- the sequence conception-simulation-fabrication is more direct;
- the manufacturing time is shortened;
- the creativity of users (designers, architects, ...) is improved by opening a wide range of modelling possibilities, that were not free in existing CAD modellers;
- this creativity is better exploited, i.e. designed shapes are closer to the constructed shapes, there are fewer losses and degradation of information.

1.3 Organization of thesis

This thesis consists of six parts and is organised as follows.

In chapter 2 we describe popular existing tools for geometric modelling, and mention the various constructing operations used to assist the user in computer-aided shape modelling. There exist many techniques to generate solid objects, such as sweeping 2D surfaces, solid modelling, free-form surfaces. We describe basic ideas of these techniques as well as discuss approaches in the iterative geometric modelling. After listing the constructing operations presented in most of CAD systems, we identify the basic properties required for their realisation. Preliminary analysis shows that CAD operations are based on four properties: affine invariance, topological structure, parametrization and differential properties. The result is given in the form of a dependency graph.

Chapter 3 defines IFS, CIFS and BCIFS. We discuss mathematical aspects that allow us to define irregular curves and surfaces. We also identify a system of topological constraints and the notion of control polygons.

In chapter 4 we classify CAD operations adapting to BCIFS and we also introduce our algorithms to approximate the operation. The approximation algorithms are generic, i.e. defined for an arbitrary operation satisfying some constraints that we explicit. These algorithms can thus be applied once we implement the required set of operators (interface). The proof of convergence of these algorithms is based on the results of domain theory.

In chapter 5 we define a self-similarity property of the operation and introduce a generic algorithm to compute a formal representation of this result, i.e. to compute a specific CIFS with the operation image as the attractor. Most of CAD operations are semi-decidable, that is the result can be computed only approximately. The presenting algorithm computes a CIFS automaton for the required accuracy. The attractor of this CIFS is thus close enough to the required operation image. In special cases, when the automaton degenerates into an single state, the constructed model represents a specific IFS and has a linear complexity on the number of iterations.

Chapter 6 presents various examples and special cases and summarizes all the results. After a general conclusion we announce prospects of our work and discuss some open questions.

Appendix A describes the methods we used to compute a bounding ball (not necessarily a tight one) of the IFS attractor. For CIFS attractors we thus compute a set of bounding balls. We use these approaches in the initialization steps of presented approximation algorithms.

Chapter 2

CAD operations

Shape modelling in CAD systems is enriched by constructing operations and algorithms. These operations are usually based on geometric properties of modelled objects. However, for fractal shapes, these properties are more complex and sometime not defined. Since most of existing CAD modellers are based on classic geometry, the modellers are not suitable to manipulate fractal structures. We study how standard CAD operators can be defined and evaluated on fractals.

Our goal is to identify the different cases that arise when we want to implement standard CAD operations on fractals. For this, we list the main standard CAD operations presented in most of CAD systems and identify the properties on which these operations are based. We then identify the basic properties required for their realisation. Preliminary analysis shows that operations are based on four properties: affine invariance, topological structure, parametrization and differential properties. The result is given in the form of a dependency graph.

Finally, we discuss existing iterative models, such as subdivision surfaces and L-Systems, that define shapes by iteration. There are various existing algorithms to manipulate these models, that will be described, before concluding.

2.1 Geometric modelling

In this section we identify general tools widely used for geometric modelling. This survey does not pretend to be exhaustive; we considered only the methods related to our work. There exist many techniques to generate solid objects, such as sweeping 2D surfaces, solid modelling, free-form surfaces. We describe basic ideas while details are available in many resources [Req77, Chi88, Hof89, HR91, Gal00, Rao04].

2.1.1 Sweep or Extrusion

There are number of useful techniques for creating objects, such as *extrude, revolve, sweep, loft*. These 3D geometric constructing methods extended from 2D, normally can be combined into two classes:

- linear extrusion or translational sweep,

- rotational sweep.

In general, for extrusion sweeping a 2D surface along a curve (see figure 2.1) generates a 3D object. In linear extrusion, sweeping is applied along a straight line. There exist further variations in sweep for generating more complex geometry. For example, it is possible to sweep in a linear direction with a taper along the direction.

There are some cases where a sweep may fail to generate a valid solid. For example, a line when swept transforms to a surface, but not a solid. Sweeping must therefore forbid such cases by imposing some restrictions on the topology and the geometry chosen for linear extrusion in order to produce a valid solid.

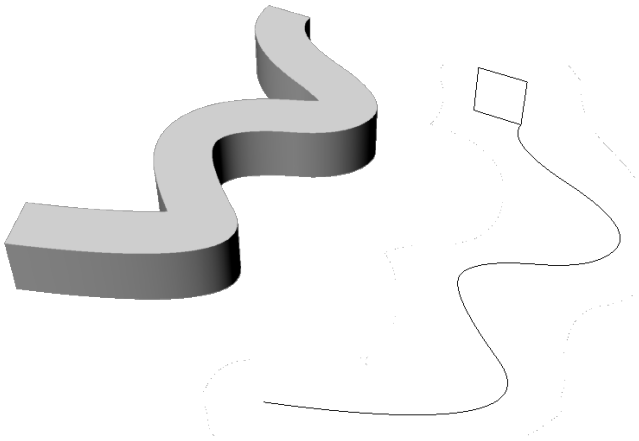


Figure 2.1: A 2D surface swept along a curve

Another type of construction is the rotational sweep (sometimes called "revolve" or "surface of revolution"), which can be used for axi-symmetric shapes, such as bottles, glasses, vases and so on. It is also possible to add twist to sweeping in the third dimension. Similarly, the rotational sweep can be enhanced by adding axial or radial offset while sweeping to get helical or spiral objects.

2.1.2 Solid modelling

A well-known method for 3D solid constructing is the solid modelling technique, often called "constructive solid geometry" (CSG). The idea is to create complex objects by adding or subtracting predefined solid primitives, as illustrated in figure 2.2.

For combining primitives one usually use basic set operators, called Boolean operators. There are following classic boolean operations:

- Union of the sets A and B, denoted $A \cup B$, is the set of all objects that are a member of A, or B, or both.
- Intersection of the sets A and B, denoted $A \cap B$, is the set of all objects that are members of both A and B.
- Set difference of A and B, denoted $A \setminus B$, is the set of all members of A that are not members of B.

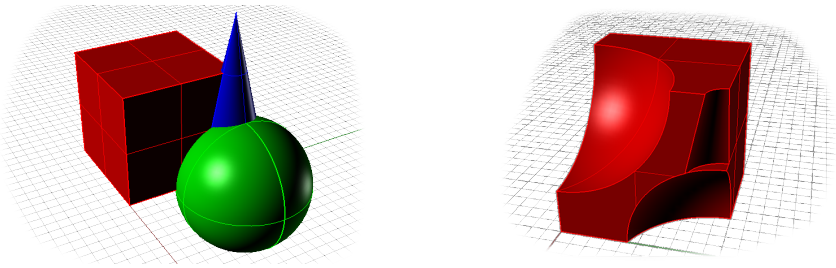


Figure 2.2: Complex objects are created by adding or subtracting predefined solid primitives. For combining these primitives one usually uses basic set operators, called Boolean operators.

- Symmetric difference of sets A and B, denoted $A \triangle B$, is the set of all objects that are member of exactly one of A or B (elements which are in one of the sets, but not in both): $(A \cup B) \setminus (A \cap B)$
- Cartesian product of A and B, denoted $A \times B$, is the set whose members are all possible ordered pairs (a, b) where a is a member of A and b is a member of B.

CSG is a powerful technique for representing fairly complex objects relatively easily. The boolean operators always guarantee that the object formed by these rules is physically constructable. However, note that these operations can affect both the geometry and the topology. To produce valid solid objects, one should use regularized boolean operations [GVS12].

2.1.3 Free-form surfaces

Free-form surfaces are generally created by manipulating the control points, as illustrated in figure 2.3. Surfaces are not stored or defined in CAD software in terms of polynomial equations, but by their control points and a surface degree. The degree of a surface determines its geometric properties.

There are many types of free-form surfaces that are widely used in CAD systems, such as ruled, Bézier or B-spline surfaces.

Some operations (chamfering, rounding/filleting) are applied directly to the model faces, edges and vertices to create a desired modification. A lofted surface is a surface constructed by transitioning between two or more curves by using a smooth blend between each section of the surface. Surface fillet is a tool to automatically generate the fillet radius between two surfaces, which could be uniform or vary linearly.

2.1.4 Transforms

All the modelling techniques have a number of simple and natural geometric transformations. There is an important class of geometric transformations, called *affine transformations*. In geometry, an affine transformation is a transformation that preserves straight lines (i.e. images of all points lying on a line, also lie on a line) and ratios of distances between points lying on a straight line (that is, the midpoint of a line segment remains the midpoint after transformation).

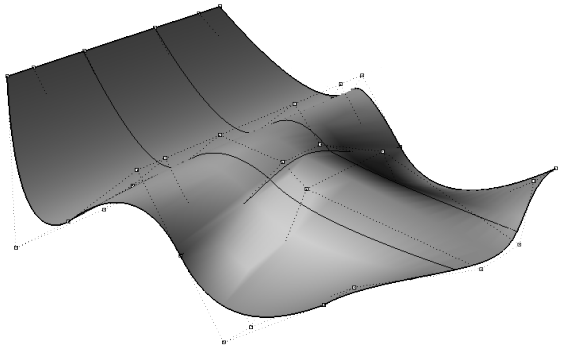


Figure 2.3: Free-form surfaces are generally created by manipulating the surface control points

Examples of affine transformations include translation, homothety, rotation, shear and also their composites.

Other examples of CAD operations simplifying geometric modelling are "copy-paste" and "array".

2.2 Operations graph

To exploit fractals in CAD systems we must be able to evaluate most standard constructing operations for fractal objects. In this section we study the constructing operations presented in the CAD system Rhino. The choice of this software was suggested by our industrial partners.

Some of these operations are composites, that is, we have to calculate several elementary operations and then combine results. For example, to determine a G_1 connection between two curves, we must be able to calculate the tangents at each curve end. But we also must be able to construct a curve with these imposed tangents.

For preliminary analysis we structure operations by determining dependencies between them. After listing the CAD operations, we construct a diagram illustrating these relations in the form of a graph (see figure 2.4). This graph illustrates the dependencies between the CAD operations and object properties. The basic properties and methods are marked by diamonds. Note that this graph does not pretend to be exhaustive neither for operations nor dependencies. However, it is constructed by considering at least one solution for each operation.

All CAD operations are divided into two categories depending on the type of the result:

- where a formal solution is provided (depicted by rectangles)
- where an approximate solution is provided (depicted by ellipses)

We now explain these categories.

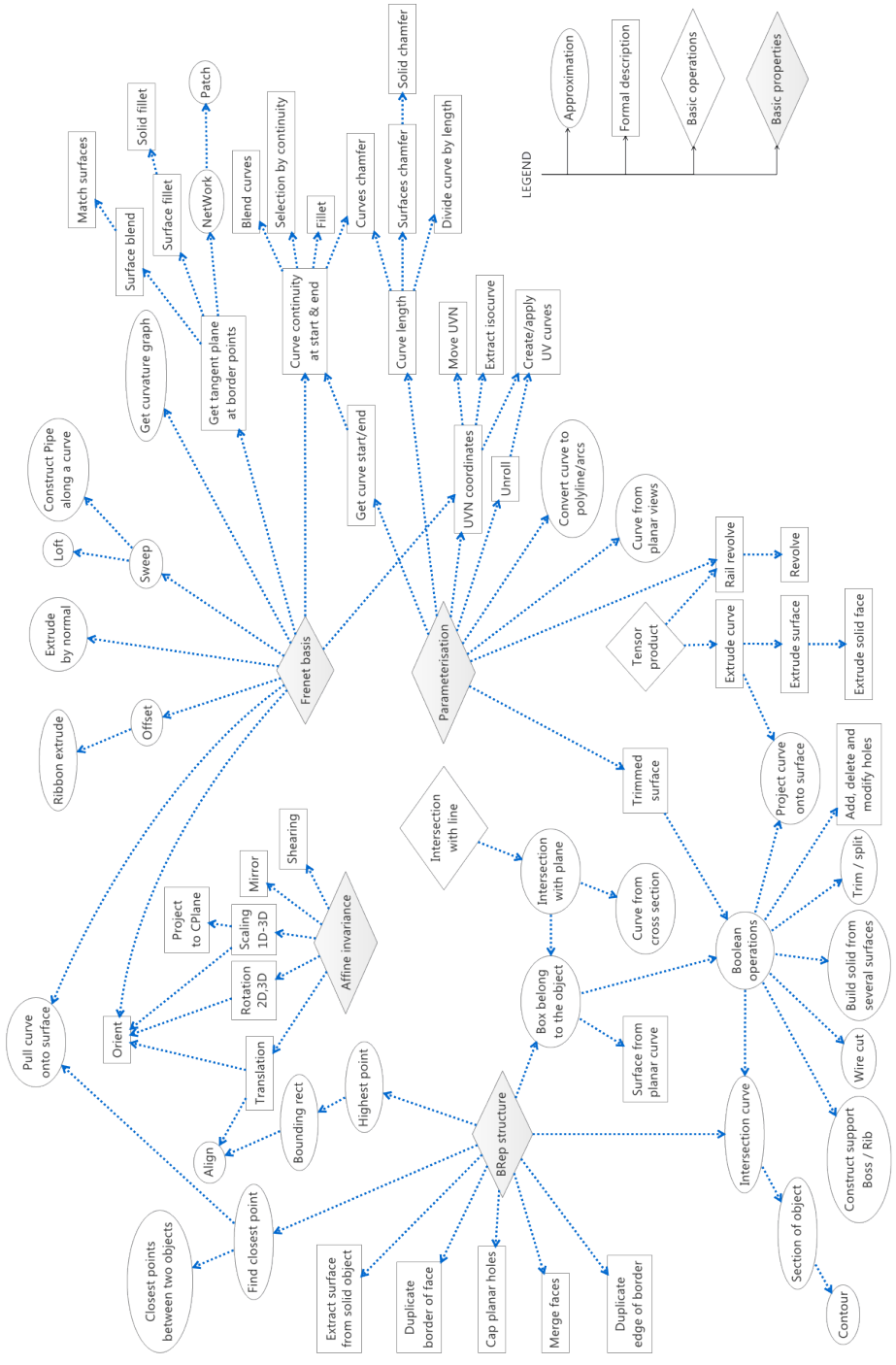


Figure 2.4: Graph of the relations between the CAD operations and the object properties.

2.2.1 Operations with a formal solution

In this category there are operations that produce a formal representation of the required result. That is, after an appropriate transform T, the NURBS surface S, for example, remains a NURBS surface, and moreover, the result represents the exact mathematical image $T(S)$.

For instance, we consider the affine transformations. Let C be a NURBS curve, defined by a set of control points $\{P_i \mid i = 1, \ldots, k\}$ as follows:

$$C(u) = \sum_{i=1}^{k} R_{i,n}(u) P_i,$$

where $R_{i,n}$ are the rational basis functions:

$$R_{i,n}(u) = \frac{N_{i,n}(u) w_i}{\sum_{j=1}^{k} N_{j,n}(u) w_j},$$

with $N_{i,n}$ the B-spline basis functions, n the degree of the B-spline basis functions and w_i the corresponding weights. The denominator is a normalizing factor.

Every affine transformation is the composite of two functions: a translation and a linear transformation. The linear part is represented as a multiplication by a matrix L and the translation as the addition of a vector b, an affine transformation T acting on the curve C can thus be represented as

$$C'(u) = T(C(u)) = b + LC(u) = b + \sum_{i=1}^{k} R_{i,n}(u) L P_i$$

Since all the $R_{i,n}$ sums to 1:

$$C'(u) = \sum_{i=1}^{k} R_{i,n}(u) b + \sum_{i=1}^{k} R_{i,n}(u) L P_i = \sum_{i=1}^{k} R_{i,n}(u)(b + L P_i) = \sum_{i=1}^{k} R_{i,n}(u) T(P_i).$$

To apply an affine transformation to a NURBS curve, we can thus simply transform the control points of the curve. Moreover, the new curve obtained by transforming the control points is also a NURBS curve. We have thus the formal representation of the image under affine transformation.

Another example is determining curve endpoints. This can be done in a natural way from the parametrisation of a curve by passing the bounding values of a parameter.

2.2.2 Operations with an approximate solution

Sometimes it is difficult or even impossible to preserve the representation of an object after applying a CAD operation. This category consists of operations for which the result is computed approximately. A simple example is intersecting between subdivision surfaces [LFKN03, BKZ01]. There is no method to represent formally the result. CAD systems therefore modify the initial representation of objects after computing this operation.

2.3 Iterative modelling and manipulations

There exists another approach to represent geometric solids, called iterative modelling, where we define the shape implicitly by an algorithm that generates it. These geometric models are usually defined by a set of rules which are applied to generate the ensuing iteration in a recursive manner, in contrast to shapes represented by boundary surfaces and volumetric models (CSG and Spatial-partitioning). However, these distinctions are often blurred: for instance, geometric shapes such as circles can be defined by implicit mathematical equations, and a fractal model reduces to a constructive model when its recursive definition is truncated to a finite depth.

Iteration allows us to generate structures with complex properties, such as roughness or lacunarity, which cannot be achieved with classic modelling. In this section we discuss existing iterative models, that define shapes by iteration. There are various existing algorithms to manipulate these models, which will also be described.

2.3.1 Subdivision surfaces

One powerful iterative method for representing shape is called *subdivision*. Subdivision surfaces allow smooth free-form surface modelling without topological constraints. Subdivision have become a fundamental representation for smooth geometry, particularly in the animated movies.

Subdivision defines smooth curve or surface as the limit of a sequence of successive refinements of a mesh. The geometric domain is represented by piecewise linear objects, usually polygons or polyhedra. The iteration consists of two distinct phases:

- **Splitting**: Each edge or face is split into edges or faces.
- **Averaging**: Each new vertex introduced by splitting is positioned at a fixed affine combination of its neighbour's positions. The initial points could also be repositioned.

Subdivision thus produces a hierarchy of polyhedra that approximate the final limit shape. This process is used mostly to produce highly smooth surfaces from polyhedra of arbitrary topology.

The main benefit of subdivision is that it is simple to implement. In order to implement a subdivision scheme, each vertex of a shape is tagged by specifying whether the descendants of the vertex lie on a vertex, edge or face of the final limit shape. During subdivision, the appropriate averaging mask can be chosen based on this tag. By truncating to a finite depth the curved limit shape can be approximated to any desired tolerance.

There exist a large number of different subdivision schemes: schemes that operate on quadrilateral [CC98, Doo78] or triangular meshes [Loo87], schemes whose limit surface either interpolates a control mesh [DLG90, Kob96] or approximates it [CC98, Doo78], as well as schemes that incorporate a rotation of the grid directions [PR97, Kob00, LG00, VZ01]. There are also schemes that use the flexibility of subdivision to mix refinement patterns in the same mesh [SL03, SW05, PS04]. Comprehensive classifications of refinement patterns are provided by Cashman [Cas12], Han [Han03] and Ivrissimtzis et al. [IDS04].

Manipulation with subdivision surfaces

Given a subdivision scheme, the fundamental difficulties are to determine the properties of the limit shape, to convert other representations, such as NURBS or CSG, into a subdivision representation, and to perform geometric operations, such as intersection, lofting or fairing.

T. Cashman [Cas12] surveys the high-performance evaluating techniques for subdivision surfaces on both CPU and GPU which makes it possible to consider subdivision surfaces in real-time applications such as computer games [KMDZ09]. The survey also includes various techniques to analyse the first and the second-order smoothness of subdivision surfaces.

To trim subdivision surfaces it is usually required to compute the parametric pre-image of a trim curve in order to exclude for evaluation the relevant parts of the surface. Litke et al. [LLS01] pointed out that with Levin's [Lev99] combined subdivision scheme, which can satisfy boundary constraints, it is possible to meet a desired trim curve exactly without computing an exact pre-image. The compromise they make is to modify (within a specified tolerance) the surface near the trim curve.

Much work has been devoted to interference detection and surface intersection for analytical surfaces [RS97] and NURBS surfaces [KKM97, KKM99, KM97], however few previous works have attempted to solve this problem for subdivision surfaces. Subdivision surface intersection often involves truncating the evaluation tree to a finite depth and then applying a mesh intersection algorithm without considering properties of subdivision [BKZ01], which can be costly to compute. The problem of interference detection was approximately solved for a large class of subdivision surfaces [DKT98, GS01, WP04].

Lanquetin et al. [LFKN03] proposed an algorithm to intersect the subdivision surfaces generated by the Loop scheme. The algorithm was later improved by Severn and Samavati [SS06] by using the strong convex hull property in order to produce a fast intersection for subdivision surfaces with arbitrary precision. This approach can therefore be used with any subdivision scheme that has this strong convex hull property. Biermann et al. [BKZ01] presented a method for approximating boolean operations for subdivision surfaces. There were attempts to improve robustness by using voxelization [LC07] or by operating on the limit mesh [JS09].

J. Stam [Sta98] showed that Catmull-Clark subdivision surface and all its derivatives can be evaluated in terms of a set of eigenbasis functions which depend only on the subdivision scheme and he derives analytical expressions for these basis functions. He implemented a technique to compute high quality curvature plots of subdivision surfaces. The cost of the evaluation scheme is comparable to that of a bi-cubic spline. Therefore, this method allows many algorithms developed for parametric surfaces to be applied to Catmull-Clark subdivision surfaces.

Subdivision schemes generate self-similar curves and surfaces. S. Schaefer et al. [SLG05] demonstrated that subdivision curves and surfaces are in fact special cases of the IFS attractors, that is, they can be generated by iterating contractive transformations. The authors derived the associated IFS for many different subdivision curves and surfaces without extraordinary vertices, including B-splines, piecewise Bezier, interpolatory four-point subdivision, bi-cubic subdivision. Moreover they showed how to build subdivision schemes to gener-

ate traditional fractals such as the Sierpinski triangle and the Koch curve.

2.3.2 L-systems

An L-system or Lindenmayer [Lin68] system is a type of formal grammar. A systematic study and formalisation was undertaken by P. Prusinkiewicz [PSS10, PL90].

An L-system consists of an alphabet of symbols, a collection of production rules that expand each symbol into some larger string of symbols, an initial string from which to begin construction, and an algorithm to translate the generated strings into geometric structures. The central concept of L-systems is the notion of rewriting. Rewriting is a technique to build complex objects by replacing parts of a simple initial object using local rewriting rules. L-systems describe growing structures in terms of automatically maintained topological (neighbourhood) relations between components. The context for applying the rules is therefore always available.

Typically, L-systems are characterized by repetitive applying simple rules to discrete structures with a changing number of components. More recently, interpretations of L-systems based on affine geometry have been presented [DeR89, Gol02]. These interpretations provide a brief description of subdivision curves. In fact, the generating algorithm is similar to the one for generating fractal curves and also resembles subdivision algorithms. Their modification rules can thus be described in local terms and formalized as L-systems production rules.

P. Prusinkiewicz et al. [PSSK02] introduced parametric context-sensitive L-systems with affine geometry interpretations as an alternative technique for specifying and generating subdivision curves. Subdivision algorithms can thus be formalized by L-systems in an intuitive manner, which leads furthermore to their computer implementation.

Such fundamental algorithms as the Lane-Riesenfeld algorithm for generating uniform B-splines of arbitrary degree, the de Casteljau algorithm for generating Bezier curves, and their extensions to rational curves, can also be expressed using L-systems with affine geometry interpretations [PSS10].

I. Zammouri et al. [ZTA08] showed the exact mathematical relation between L-systems and iterated function systems (IFS). They proposed a new method to describe fractal shapes using parametric L-systems.

2.4 Conclusion

In this chapter we identified the general tools widely used for geometric modelling and we described the various constructing operations used to assist users in computer-aided shape modelling. There exist many techniques to generate solid objects, such as sweeping 2D surfaces, solid modelling, free-form surfaces. We described the basic ideas of these techniques.

After enumerating all the operations presented in the CAD system Rhino, we constructed a diagram that illustrates dependencies between these operations in the form of a graph. Operations were then classified into two categories depending on type of the result: a formal representation of the result or an approximate result.

The constructing operations are usually based on geometric properties of modelled objects. A preliminary analysis shows that CAD operations are based on four basic properties: affine invariance, topological structure, parametrization and differential properties. For example, in order to align objects we must be able to compute the bounding box and to translate the objects. If the objects are not affine invariant, the computed bounding box will no longer be correct after translating. Extracting surfaces of a solid or duplicating their borders can be easily computed using the boundary representation. These basic properties are therefore essential and we must find their equivalents in fractal geometry. In fact, for fractal shapes, these properties are more complex and sometime even not defined. That is why existing CAD systems are often not suitable to manipulate fractal structures.

In fact, most of existing CAD modellers are based on classic geometry. These modellers permit modelling analytic curves and surfaces, which are usually polynomial or rational and smooth. The shapes modelled by iterative methods generally are not smooth because of their fractal structure. Their topology can be controlled in order to build curves and surfaces represented by parametrized functions, but these functions are usually nowhere differentiable.

Iterative models, such as L-systems, give a recursive definition of shapes allowing us to generate structures with specific properties, such as roughness or lacunarity. Shapes with these properties cannot be achieved with classic modelling. To exploit these shapes in CAD systems, we must be able to evaluate most standard constructing operations for fractal objects.

Being a parametric and an implicit model, a fractal reduces to a constructive geometry when its recursive definition is truncated to a finite depth. The CAD operations can therefore be applied directly to an approximation of a fractal. However, the attractor approximation can be time-consuming, because the number of objects grows exponentially with an increasing number of iterations. In the next chapter we adapt CAD operations to fractals, we formalize the notion of approximation by identifying situations for which it is possible to control this approximation. We identify the constraints on operations in order to apply general or optimised algorithms. These algorithms can thus be applied once we implement the required set of operators (interface).

Together, the principles of iterative, geometric and solid modelling form the foundation of computer-aided design and in general support the creation, visualization, animation and annotation of digital models for physical objects.

Chapter 3

Iterated Function Systems and generalisations

In this chapter we define the iterative systems that will be used to represent fractal objects. We discuss mathematical aspects that allow us to define irregular curves and surfaces. We also identify a system of topological constraints and the notion of control polygons.

3.1 IFS

Iterated Function Systems (IFS) were introduced by HUTCHINSON [Hut81] as a strictly mathematical model. Barnsley further developed and popularized this model for applications in computer graphics [Bar88]. Much research has followed on from this idea [Gen92, Mas97, BHS08, Tos06].

IFS are based on the self-similarity property. A modelled object is made up of the union of several copies of itself; each copy is transformed by a function. These functions are usually contractive, that is, bringing points closer together and making shapes smaller. Hence, the modelled object, called *attractor*, is made up of several possibly-overlapping smaller copies of itself, each copy also made up of copies of itself, ad infinitum.

3.1.1 Definition of IFS

In this section we formally define IFS and we thus establish the notation used in this thesis.

Definition. Given a complete metric space $(\mathbb{X}, d)$ with the associated metric d, an IFS is defined by a finite set of continuous transformations $T = \{T_i\}_{i=0}^{N-1}$ in the space $\mathbb{X}$. Let $\Sigma = \{0, \ldots, N-1\}$ be the set of IFS transformation indices, thus $|\Sigma| = N$. The IFS is then denoted by $\{\mathbb{X}; T_i \mid i \in \Sigma\}$.

A simple example of an IFS can be constructed for a representation of real numbers in $[0, 1]$. Let

$$T_i : x \to \frac{x+i}{3} : [0,1] \to [0,1],$$

where $i \in \Sigma = \{0, 1, 2\}$. The IFS $\{[0,1]; T_0, T_1, T_2\}$ can thus express the representation of any real number in $[0, 1]$.

We are substantially interested in so-called *hyperbolic* IFS, whose transformations T_i are all contractive.

Definition. A transformation $T : \mathbb{X} \to \mathbb{X}$ is called *contractive* if and only if there exists a real s, $0 \leqslant s < 1$ such that $d(T(x), T(y)) < s \cdot d(x, y)$ for all $x, y \in \mathbb{X}$. The minimal coefficient s which satisfies the inequality is called the *contraction coefficient* of the transformation T. To distinguish contraction coefficients of different transformations we also use the function notation $s(T)$ to denote the contraction coefficient of the transformation T.

Definition. For the set of non-empty compact subsets of $\mathbb{X}$, denoted $\mathcal{H}(\mathbb{X})$, we define the HAUSDORFF distance $d_{\mathbb{X}}$ induced by the metric d:

$$d_{\mathbb{X}}(A, B) = \max\{\sup_{a \in A} \inf_{b \in B} d(a, b), \sup_{b \in B} \inf_{a \in A} d(a, b)\}.$$

Since the metric space $(\mathbb{X}, d)$ is complete, the set $(\mathcal{H}(\mathbb{X}), d_{\mathbb{X}})$ is also complete [Bar88].

In the early 80's, Hutchinson [Hut81] used the BANACH fixed point theorem to deduce the existence and the uniqueness of an attractor for a hyperbolic IFS, i.e. the fixed point of the associated contractive map. It is thus possible to define an operator $\mathbb{T} : \mathcal{H}(\mathbb{X}) \to \mathcal{H}(\mathbb{X})$, called HUTCHINSON operator [Bar88], as the union of the IFS transformations T_i:

$$\mathbb{T}(K) = \bigcup_{i=0}^{N-1} T_i(K).$$

The contraction coefficient of the HUTCHINSON operator is thus defined by:

$$s_{max} = \max_{i=0,\cdots,N-1} s(T_i).$$

Definition. If $s_{max} < 1$ then $\mathbb{T}$ is also contractive in the complete metric space $(\mathcal{H}(\mathbb{X}), d_{\mathbb{X}})$. According to the BANACH fixed point theorem [Bar88], $\mathbb{T}$ has a unique fixed point $\mathcal{A}$. This fixed point is named the attractor of the IFS:

$$\mathcal{A} = \mathbb{T}(\mathcal{A}) = \bigcup_{i=0}^{N-1} T_i(\mathcal{A}). \tag{3.1}$$

3.1.2 Examples of an IFS

In the following examples we use the EUCLIDEAN plane $\mathbb{R}^2$ with the associated EUCLIDEAN metric.

The first example is the SIERPINSKI triangle illustrated in figure 3.1. We define an IFS with the three contractive transformations T_0, T_1 and T_2, defined by the homotheties with centres p_i and ratios $\frac{1}{2}$ as follows:

$$T_0 = H^{p_0}_{\frac{1}{2}}$$
$$T_1 = H^{p_1}_{\frac{1}{2}}$$
$$T_2 = H^{p_2}_{\frac{1}{2}}.$$

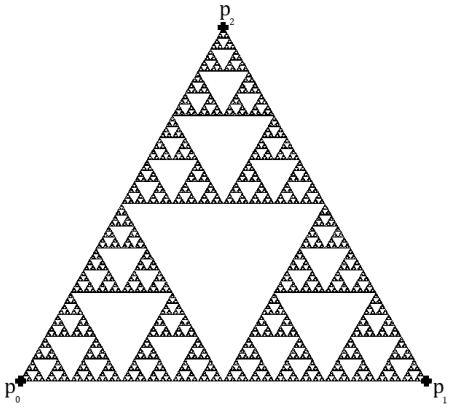


Figure 3.1: Sierpinski triangle

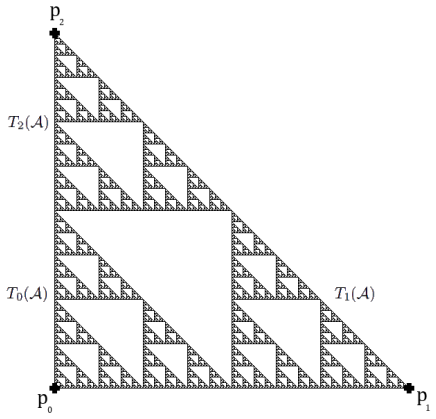


Figure 3.2: Attractor of an IFS composed of three homotheties with ratios $\frac{1}{2}$ and centres p_i.

Here H_s^p denotes the homothety centred at point p with ratio s.

The Sierpinski triangle is the attractor of this IFS, that is a non-empty compact set satisfying the equation (3.1):

$$\mathcal{A} = H_{\frac{1}{2}}^{p_0}(\mathcal{A}) \cup H_{\frac{1}{2}}^{p_1}(\mathcal{A}) \cup H_{\frac{1}{2}}^{p_2}(\mathcal{A}).$$

The shape of the IFS attractor depends entirely on the transformations T_i. The Sierpinski triangle has a number of particularities, which can generally be found in other IFS attractors. One of the main characteristics, common to all IFS attractors, is self-similarity with changing scale. Inside the attractor there are subsets similar to the whole set, and this pattern occurs at any scale.

Figure 3.2 illustrates the same triangle with a slightly modified geometry. Here, IFS transformations have the same ratio, but we modify the centre of the homothety T_2.

The following example demonstrates an IFS composed of three homotheties

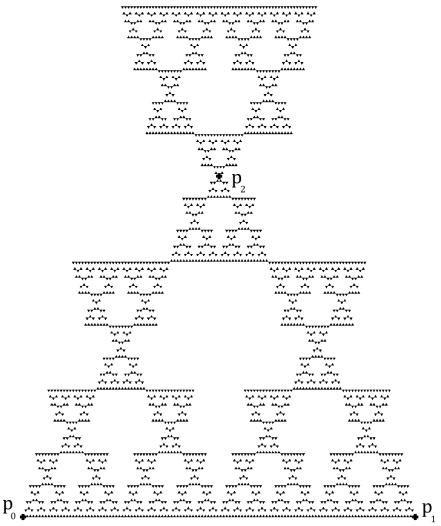


Figure 3.3: Attractor of an IFS composed of two homotheties with positive ratios $\frac{1}{2}$ and one homothety with negative ratio $-\frac{1}{2}$. The centres of these homotheties are p_i for $i = 0, \ldots, 2$.

whose centres form an equilateral triangle, but we change their ratios as follows:

$$T_0 = H^{p_0}_{\frac{1}{2}}$$
$$T_1 = H^{p_1}_{\frac{1}{2}}$$
$$T_2 = H^{p_2}_{-\frac{1}{2}},$$

The attractor of this IFS is shown in figure 3.3.

These examples show that a simple IFS system can produce complex self-similar structures, but it is quite difficult to control the geometry and topology of these objects, even though the geometry depends continuously on the transformations [Bar88, Gen92].

3.1.3 IFS construction algorithms

The Banach fixed point theorem [Bar88] describes the existence and uniqueness of the attractor for a hyperbolic IFS $\{\mathbb{X}; T_i \mid i \in \Sigma\}$. In this section we describe methods to approximate and to visualize the attractor.

There are two canonical ways to evaluate the attractor of a dynamical system:

- By constructing a decreasing sequence of non-empty compact sets which shrink down to the attractor. The smaller the compact set in the sequence, the better the attractor approximation with respect to the Hausdorff distance. Here, the initial compact set has to cover the attractor. This method provides an approximation of the fractal exterior.

- By constructing an increasing sequence of non-empty compact sets whose union has the attractor as its closure. Here, we have to start with a

compact set included in the attractor. In this case, the bigger the compact set in the sequence, the better the attractor approximation.

Other construction and visualisation methods exist [PS88, HPS92], such as the *dynamic chaos game* or *non-deterministic* algorithms [Bar88, BD85]. However, we are substantially interested in the deterministic method allowing us to approximate most operations. In fact, in both these cases, we use a similar approximation algorithm, i.e. we recursively apply transformations to the initial non-empty compact subset.

The attractor of an IFS may therefore be evaluated recursively. That is, the attractor can be approximated by a sequence $\{K_n\}_{n\in\mathbb{N}}$ converging to $\mathcal{A}$. The initial element in the sequence is defined by means of a primitive $K \in \mathcal{H}(\mathbb{X})$. The following elements are defined recursively:

$$\begin{aligned} K_0 &= K \\ K_{n+1} &= \bigcup_{i\in\Sigma} T_i(K_n). \end{aligned}$$

The elements K_n are images of composite functions applied to K.

Each element in the sequence represents an approximation of the IFS attractor. Each term K_n is composed of N^n images of K by a composite of n functions. For example, a sequence of the attractor approximations for an IFS $\{\mathbb{X}; T_0, T_1\}$ is presented here:

$$\begin{aligned} K_0 &= K, \\ K_1 &= \mathbb{T}(K_0) = T_0(K) \cup T_1(K), \\ K_2 &= \mathbb{T}(K_1) = T_0T_0(K) \cup T_0T_1(K) \cup T_1T_0(K) \cup T_1T_1(K), \\ K_3 &= \mathbb{T}(K_2) = T_0T_0T_0(K) \cup T_0T_0T_1(K) \cup T_0T_1T_0(K) \cup T_0T_1T_1(K) \cup \\ &\qquad T_1T_0T_0(K) \cup T_1T_0T_1(K) \cup T_1T_1T_0(K) \cup T_1T_1T_1(K), \\ &\quad\ \vdots \qquad \vdots \qquad \vdots \\ K_n &= \mathbb{T}(K_2) = \bigcup_{\alpha_i\in\Sigma} T_{\alpha_1}T_{\alpha_n}(K_{n-1}). \end{aligned}$$

In this iterative algorithm, a set of transformed primitives K is constructed recursively and calculations can be represented by an *evaluation tree*. Each node on the i-th level of the tree corresponds to the image of a composite of i IFS transformations. This tree is traversed up to a given depth n, where we display the image of K by the composite function associated with the current node, as shown in figure 3.4.

Note that these composite functions are calculated from left to right. The primitive K is finally transformed by a constructed composite function. In practice, the IFS transformations T_i are affine operators and can therefore be represented by matrices. A composite affine transformation can thus be represented by a product of transformation matrices.

In chapter 4 we use the notion of the evaluation tree to decompose an operation image and to optimize the approximation.

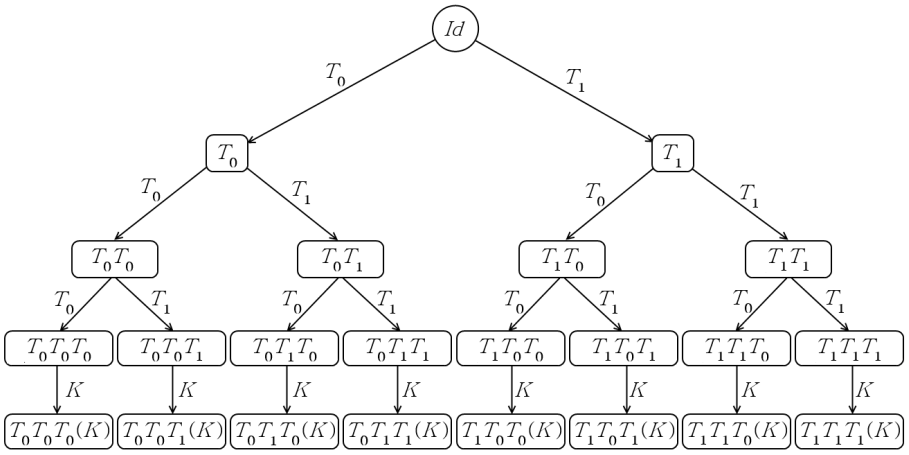


Figure 3.4: The IFS evaluation tree calculated to the third level. Internal nodes correspond to the calculation of a composite function. Leaves correspond to subsets of K_3 to construct or to visualise.

3.2 Controlled IFS

In IFS all the transformations are applied at each iteration. It is possible to enrich this model by adding rules to control the iterations. This is the principle of a *CIFS* (Controlled IFS).

CIFS are more general systems allowing us to control certain parts of the IFS attractor. A CIFS denotes an IFS with restrictions on transformation sequences imposed by a control graph. This system is similar to "Restricted IFS" (RIFS) [Bar88], and is also described [PH94, TT93] by means of formal languages, called LRIFS (Language-Restricted Iterated Function System). CIFS allow us to handle fractals with subsets of different dimensions. CIFS define objects whose geometry can be complex. However CIFS attractors are more convenient and controllable than IFS attractors for manufacturability purposes.

The attractor of a CIFS can be evaluated by an automaton [MW88] defined on the control graph. Each state of the automaton corresponds to different parts of the modelled object. States are associated with construction spaces. Transitions between states indicate that one sub-part is contained in another one. It is then possible to control the attractor more precisely. However, it is usually NP-complex to evaluate, to analyse and to characterize (location, size) the resulting shape.

3.2.1 Definition of CIFS

Definition. A CIFS is given by an automaton, where each state q is associated with an attractor $\mathcal{A}^q \in \mathbb{X}^q$, and each transition from q to w is associated with an operator $\mathbb{X}^w \to \mathbb{X}^q$. Here is a list of parameters describing the CIFS:

- An automaton (Σ, Q, δ), where Σ is an alphabet, Q is a set of states and δ is a transition function $\delta : Q \times \Sigma \to Q$;

- A set of complete metric spaces associated with the automaton states $\{\mathbb{X}^q\}_{q\in Q}$;
- An operator associated to each transition $T_i^q : \mathbb{X}^{\delta(q,i)} \to \mathbb{X}^q$;
- A compact set $K^q \in \mathcal{H}(\mathbb{X}^q)$, called *primitive*, associated with each state $q \in Q$. Primitives are not used to define the attractor, but only to approximate it;
- Finally, the automaton is provided by an initial state, noted by $\natural$.

In the following, we denote by Σ^q the restriction of Σ by outgoing transitions from the state q, i.e.:

$$\Sigma^q = \{i \in \Sigma,\ \delta(q,i) \in Q\}$$

3.2.2 Attractor of a CIFS

The attractor of an IFS can be evaluated recursively.

Definition. A CIFS defines a family of attractors associated with the states: $\{\mathcal{A}^q\}_{q\in Q}$, where $\mathcal{A}^q \in \mathcal{H}(\mathbb{X}^q)$. The attractors $\mathcal{A}^q$ are mutually defined recursively:

$$\mathcal{A}^q = \bigcup_{i\in\Sigma^q} T_i^q(\mathcal{A}^{\delta(q,i)}) \tag{3.2}$$

A CIFS attractor, denoted $\mathcal{A}$, is the attractor associated with the initial state $\natural$, i.e.:

$$\mathcal{A} = \mathcal{A}^{\natural}.$$

As for IFS, each CIFS attractor can be approximated by a sequence $\{K_n^q\}_{n\in\mathbb{N}}$ converging to $\mathcal{A}^q$. Each state $q \in Q$ is associated with a primitive $K^q \in \mathcal{H}(\mathbb{X}^q)$, which defines the initial element in the sequence. The following elements are mutually defined recursively:

$$\begin{array}{lcl} K_0^q & = & K^q \\ K_{n+1}^q & = & \bigcup_{i\in\Sigma^q} T_i^q(K_n^{\delta(q,i)}) \end{array}$$

In this iterative algorithm a set of transformed primitives K^q is recursively constructed and the calculations can also be represented by an *evaluation tree*. Each node on the i-th level of the tree corresponds to the image of a composition of i CIFS transformations, i.e. a path of length i in the automaton. This tree is traversed up to a given depth n, where we display the image of K^q by the composite function associated with the current node, as shown in figure 3.5.

Consider an example of the CIFS attractor, illustrated in figure 3.6. The system is described by an automaton with two states: a and b. There are two subdividing operators from the state a and three from b. Figure 3.7 shows the automaton of this CIFS. Transition functions are the following:

$$\begin{array}{ll} \delta(a,0) = a & \delta(b,0) = b \\ \delta(a,1) = b & \delta(b,1) = b \\ & \delta(b,2) = b \end{array}$$

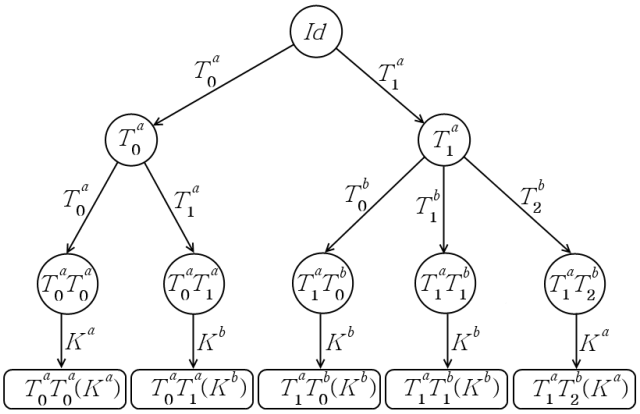


Figure 3.5: CIFS evaluation tree calculated to the third level. Internal nodes correspond to the calculation of a composite function. Leaves correspond to subsets of K_3 to construct or to visualise.

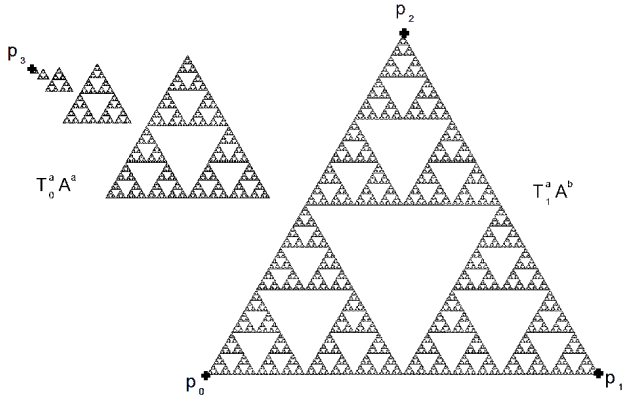


Figure 3.6: Attractor of a CIFS described by an automaton with two states: a and b.

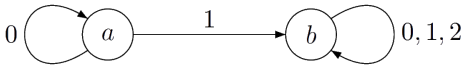


Figure 3.7: The automaton of the CIFS considered in the example. There are two subdividing operators from the state a to construct an array of the SIERPINSKI triangles and three operators from b to construct each triangle.

Each transition $\delta(q, i) = w$ of the automaton is associated with an operator $T_i^q : \mathbb{X}^w \to \mathbb{X}^q$. Assume that $\mathbb{X}^a$ and $\mathbb{X}^b$ are both the same EUCLIDEAN affine plane. Let the operator between these spaces be the identity map Id and the operators defined from the space into itself be the following homotheties:

$$\begin{array}{ll} T_0^a = H_{0.42}^{p_3} & T_0^b = H_{0.5}^{p_0} \\ T_1^a = Id & T_1^b = H_{0.5}^{p_1} \\ & T_2^b = H_{0.5}^{p_2}, \end{array}$$

where H_s^p denotes the homothety centred at point p with ratio s.

Thus, the attractors $\mathcal{A}^a$ and $\mathcal{A}^b$ satisfy the following equations:

$$\mathcal{A}^a = \bigcup_{i\in\Sigma^a} T_i^a(\mathcal{A}^{\delta(a,i)}) = T_0^a(\mathcal{A}^a) \cup T_1^a(\mathcal{A}^b) = H_{0.42}^{p_3}(\mathcal{A}^a) \cup \mathcal{A}^b$$

$$\mathcal{A}^b = \bigcup_{i\in\Sigma^b} T_i^b(\mathcal{A}^{\delta(b,i)}) = T_0^b(\mathcal{A}^b) \ \cup \ T_1^b(\mathcal{A}^b) \ \cup \ T_2^b(\mathcal{A}^b) = \\ = H_{0.5}^{p_0}(\mathcal{A}^b) \cup H_{0.5}^{p_1}(\mathcal{A}^b) \cup H_{0.5}^{p_2}(\mathcal{A}^b)$$

As can be deduced from defining $\mathcal{A}^b$ (see equation 3.2), this attractor is the Sierpinski triangle with vertices p_0, p_1 and p_2.

In figure 3.6 we can see that the attractor $\mathcal{A}^a$ is an infinite set of contracted Sierpinski triangles.

Remark Attractors of a CIFS are uniquely defined if operators associated with each cycle in the control graph are contractive. In the previous example, the transition between the states does not need to be contractive.

3.2.3 Projection onto the modelling space

From IFS and CIFS, various shapes can be modelled. However, it is difficult to control their topological properties. The shapes are determined by a set of geometry operators. Modifying these operators leads to both global and local changes in the shape and affects not only geometry but also topology.

To separate global and local control, we construct the attractor in a barycentric space and then project this attractor according to a set of control points in the modelling space. The global shape can therefore be modified by moving control points, while the local geometry (geometric texture) is defined by subdividing operators, i.e. CIFS transformations [ZT96, SLG05].

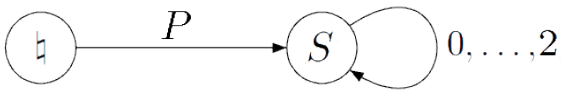


Figure 3.8: The automaton of the CIFS considered in the example. There are three subdividing operators from state S to itself and one projecting operator from S to $\natural$.

Consider an example of the Sierpinski triangle as an attractor of a projecting CIFS. The system is described by an automaton with two states: $\natural$ (for the modelling space) and S (to construct the triangle). There are three subdividing operators from state S to itself and one projecting operator from S to $\natural$. Figure 3.8 shows the automaton of this CIFS. Transition operators are the following:

$$T_0^{\natural} = P \quad \begin{array}{l} T_0^S = H_{0.5}^{(1,0,0)} \\ T_1^S = H_{0.5}^{(0,1,0)} \\ T_2^S = H_{0.5}^{(0,0,1)}, \end{array}$$

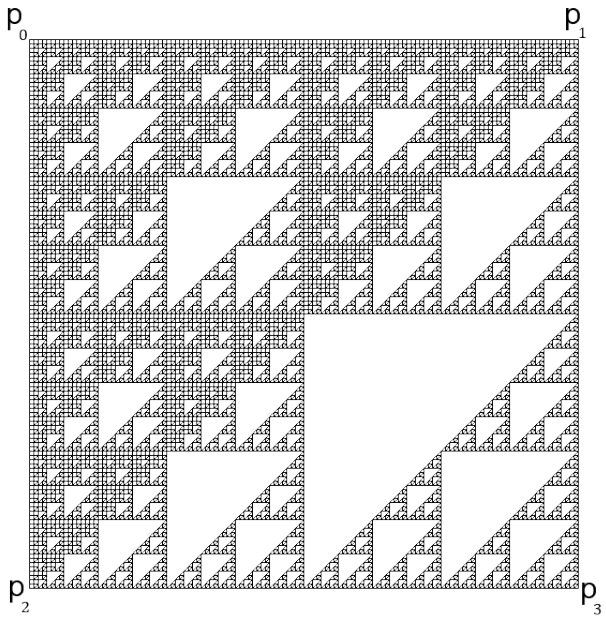


Figure 3.9: Attractor of a CIFS described by an automaton with three states ♮, C and S.

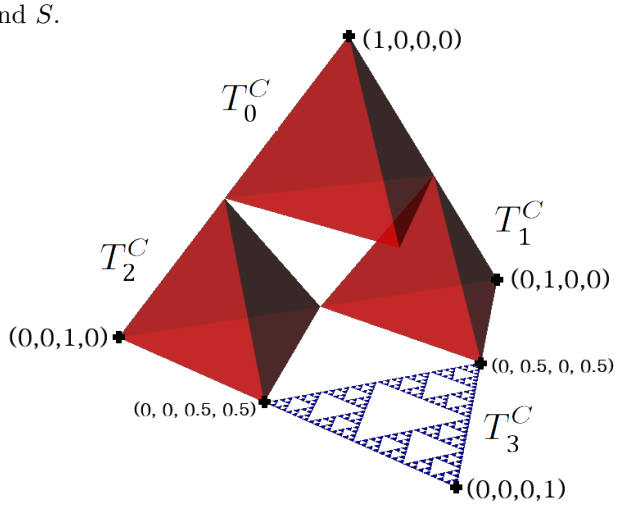


Figure 3.10: 4D barycentric space $\mathbb{X}^C$ for constructing a square. Three subdividing operators transform the space into itself by scaling with a ratio 0.5. The fourth operator projects the SIERPINSKI triangle constructed in the space $\mathbb{X}^S$.

where P is the projecting operator and H_s^p denotes the homothety centred at point p with ratio s.

In this case, the SIERPINSKI triangle is constructed in a 3D barycentric space. We choose the centres of homotheties as the unit vectors to simplify the following projection onto the modelling space. The projecting operator P has the following matrix representation:

$$P = \begin{pmatrix} p_0[0] & p_1[0] & p_2[0] \\ p_0[1] & p_1[1] & p_2[1] \\ \vdots & \vdots & \vdots \\ p_0[d] & p_1[d] & p_2[d] \end{pmatrix},$$

where d is the dimension of the modelling space and $\{p_0, p_1, p_2\}$ is a set of control points, called the *control polygon*. Here $p_i[j]$ denotes the j-th coordinate of point p_i. The projecting matrix is thus composed of the control point coordinates in columns.

According to definition (3.2), the attractor $\mathcal{A}^\natural$ of this CIFS satisfies the following equation:

$$\mathcal{A}^\natural = P(\mathcal{A}^S).$$

Note that applying affine transformations becomes simple. In order to apply a transformation R to the attractor, we modify the projecting operator as follows:

$$P' \leftarrow R \circ P,$$

where P is the initial projecting operator and P' is the new one. Indeed, the attractor of the modified CIFS, denoted by $\mathcal{A}^{\natural'}$, can be determined as follows:

$$R(\mathcal{A}^\natural) = RP(\mathcal{A}^S) = P'(\mathcal{A}^S) = \mathcal{A}^{\natural'}.$$

Consider another example of a CIFS, illustrated in figure 3.9. The system is described by an automaton with three states: $\natural$, C and S. State C is to construct a square, and is associated with the 4D barycentric space $\mathbb{X}^C$, illustrated in figure 3.10. The square is then projected to the modelling space $\mathbb{X}^\natural$. State S is to construct the SIERPINSKI triangle in the 3D barycentric space $\mathbb{X}^S$, which is then projected onto $\mathbb{X}^C$.

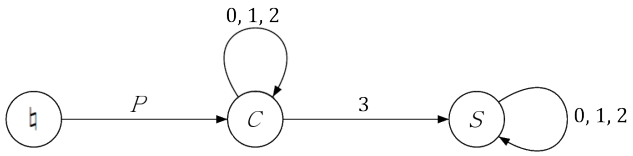


Figure 3.11: The automaton of the CIFS considered in the example. There are four subdividing operators from C to construct a square, but the SIERPINSKI triangle is inserted as one quarter. There are three subdividing operators from S to construct the SIERPINSKI triangle. The result is then projected onto the modelling space $\mathbb{X}^\natural$.

There are four subdividing operators from state C and three from S. Figure 3.11 shows the automaton of this CIFS. Transition functions are the following:

$$\begin{array}{ll} \delta(C,0) = C & \delta(S,0) = S \\ \delta(C,1) = C & \delta(S,1) = S \\ \delta(C,2) = C & \delta(S,2) = S \\ \delta(C,3) = S & \end{array}$$

Each transition $\delta(q,i) = w$ of the automaton is associated with an operator $T_i^q : \mathbb{X}^w \to \mathbb{X}^q$. The subdividing operators are defined as follows:

$$\begin{array}{ll} T_0^C = H_{0.5}^{(1,0,0,0)} & T_0^S = H_{0.5}^{(1,0,0)} \\ T_1^C = H_{0.5}^{(0,1,0,0)} & T_1^S = H_{0.5}^{(0,1,0)} \\ T_2^C = H_{0.5}^{(0,0,1,0)} & T_2^S = H_{0.5}^{(0,0,1)}, \end{array}$$

where H_s^p denotes the homothety centred at point p with ratio s. The projecting map T_3^C of the SIERPINSKI triangle onto the space $\mathbb{X}^C$ is defined as follows:

$$T_3^C = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0.5 & 0 & 0 \\ 0.5 & 0.5 & 1 \end{pmatrix}.$$

This matrix projects the 3D barycentric space $\mathbb{X}^S$, in which the triangle is constructed, onto the 4D barycentric space $\mathbb{X}^C$ by conforming the unit vectors of $\mathbb{X}^S$ to the control points $(0,0,0.5,0.5)$, $(0,0.5,0,0.5)$ and $(0,0,0,1)$, as illustrated in figure 3.10. Finally, the projecting map P is defined as follows:

$$P = \begin{pmatrix} p_0[0] & p_1[0] & p_2[0] & p_3[0] \\ p_0[1] & p_1[1] & p_2[1] & p_3[1] \\ \vdots & \vdots & \vdots & \vdots \\ p_0[d] & p_1[d] & p_2[d] & p_3[d] \end{pmatrix},$$

where d is the dimension of the modelling space and $\{p_i \mid i = 0, \ldots, 3\}$ is the control polygon.

According to definition (3.2), the attractors of this CIFS satisfy the following equations:

$$\begin{aligned} \mathcal{A}^\natural &= P(\mathcal{A}^C), \\ \mathcal{A}^C &= T_0^C(\mathcal{A}^C) \cup T_1^C(\mathcal{A}^C) \cup T_2^C(\mathcal{A}^C) \cup T_3^C(\mathcal{A}^S), \\ \mathcal{A}^S &= T_0^S(\mathcal{A}^S) \cup T_1^S(\mathcal{A}^S) \cup T_2^S(\mathcal{A}^S). \end{aligned}$$

Figure 3.12 illustrates the same attractor projected onto a different control polygon.

3.3 BCIFS

From IFS and CIFS, various shapes can be modelled. However, it is difficult to control their topological properties. The shapes are determined by a set of geometry operators. Modifying these operators leads to both global and local changes in the shape and affects not only geometry but also topology.

In order to control the topological structure of the modelled shape, one can enrich CIFS by integrating the topological model to obtain *BCIFS* (Boundary Controlled Iterated Function System) [TBSG+06].

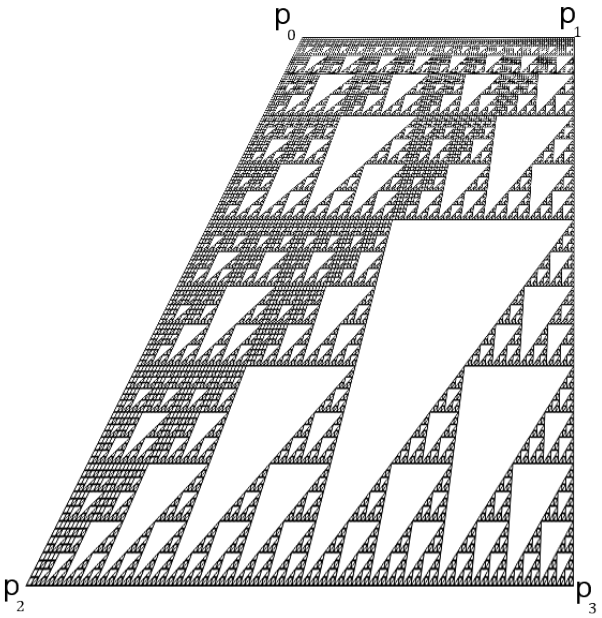


Figure 3.12: Attractor of the CIFS considered in the previous example with a different projecting map P.

It is thus possible to specify incidence and adjacency constraints [Tos06, Gou09] for subdivision and thereby to control the resulting topology: classic topology (curve, surface,...) or fractal topology. The topological model used is more general than the classic one. A topological cell may be fractal. For example, a face may be the SIERPINSKI triangle or an edge may be the CANTOR set, but the topological structure remains consistent.

3.3.1 Definition of BCIFS

The topological model is commonly used in CAD systems to represent geometric objects. This formalism was introduced to separate the topology and the geometric properties of a considered object. The topological structure is encoded by a set of topological cells (faces, edges, vertices) interconnected by a set of incidence and adjacency relations. The incidence relations are based on nesting of cells: each face is bounded by a set of edges, and each edge is bounded by two vertices. The adjacency relations are based on sharing cells: two adjacent faces share a common edge, and two adjacent edges are bounded by a common vertex. BCIFS is thus an extension of a CIFS enriched by the topological model.

Classic shape description includes a finite number of components. Each shape is represented by a finite set of cells with incidence and adjacency relations using polynomial or rational (NURBS) functions. Topological structure of these shapes is thus encoded by a finite graph. An iterative description of fractal shapes thus consists of an infinite number of components generated by subdivision. At each step, the algorithm generates an approximate mesh with an increasing (potentially infinite) number of components. The topological struc-

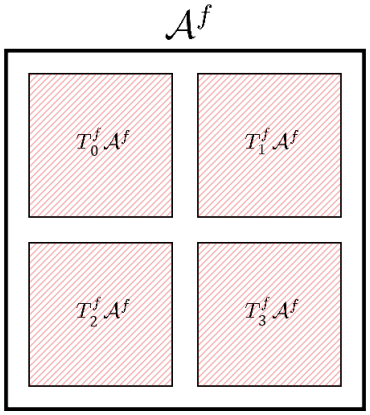


Figure 3.13: Subdivision of a face in the considered example of a BCIFS.

ture is encoded by a finite set of rules, according to which the current level of approximation is obtained from the previous one. All topological rules on the n-th subdivision level can thus be derived from the rules on the first level.

Each topological cell corresponds to an attractor in a certain space. A face corresponds to the attractor $\mathcal{A}^f$ of the IFS $(\mathbb{X}^f; T_0^f, T_1^f, T_2^f, T_3^f)$ (see figure 3.13). An edge corresponds to the attractor $\mathcal{A}^e$ of the IFS $(\mathbb{X}^e; T_0^e, T_1^e)$ (see figure 3.14). A vertex corresponds to the attractor $\mathcal{A}^v$ of the IFS $(\mathbb{X}^v; T_0^v)$.

Figure 3.15 illustrates how a square can be decomposed into the topological cells. Incidence operators are used to generalize the concept of edge. Each included cell is defined by an embedding operator. Each face of the square is bounded by four edges and each edge is bounded by two vertices. Embedding of edges $\mathcal{A}^e$ in $\mathbb{X}^f$ is defined by four embedding operators $\amalg_i^f : \mathbb{X}^e \to \mathbb{X}^f$ for $i = 0, \ldots, 3$:

$$\amalg_i^f \mathcal{A}^e \subset \mathcal{A}^f.$$

Embedding of vertices $\mathcal{A}^v$ in $\mathbb{X}^e$ is defined by two embedding operators $\amalg_i^e : \mathbb{X}^v \to \mathbb{X}^e$ for $i = 0, 1$:

$$\amalg_i^e \mathcal{A}^v \subset \mathcal{A}^e.$$

There are two types of transition in the BCIFS automaton:

- transitions subdividing a topological cell;
- transitions embedding a topological cell in another one.

The alphabet Σ is also divided into:

- symbols of subdivision $\Sigma_\div = \{\div_i \mid i \in \mathbb{N}\}$;
- symbols of incidence $\Sigma_\partial = \{\partial_i \mid i \in \mathbb{N}\}$.

Each subdividing transition $\delta(q, \div_i) = w$ is associated with a subdividing operator $T_i^q : \mathbb{X}^q \to \mathbb{X}^w$, where $q, w \in Q$ and $\div_i \in \Sigma_\div$. Similarly, each embedding transition $\delta(q, \partial_i) = w$ is associated with an embedding operator $\amalg_i^q : \mathbb{X}^q \to \mathbb{X}^w$, where $q, w \in Q$ and $\partial_i \in \Sigma_\partial$.

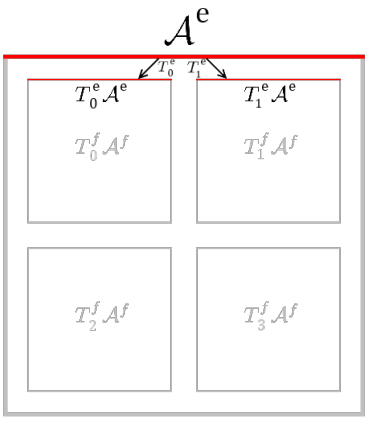


Figure 3.14: Subdivision of an edge in the considered example of a BCIFS.

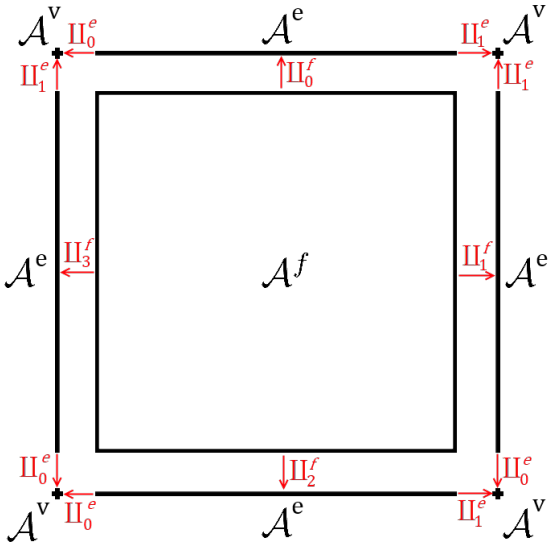


Figure 3.15: Decomposition of the square into the topological cells (face, edge, vertex).

Usually edges share certain vertices. An example of shared vertices is illustrated in figure 3.15. These relations can be translated into adjacency constraints on the embedding operators $\amalg_i^f, i = 0, \ldots, 3$:

$$\begin{aligned}
\amalg_0^f \amalg {}_0^e &= \amalg_3^f \amalg {}_1^e \\
\amalg_0^f \amalg {}_1^e &= \amalg_1^f \amalg {}_1^e \\
\amalg_1^f \amalg {}_0^e &= \amalg_2^f \amalg {}_1^e \\
\amalg_2^f \amalg {}_0^e &= \amalg_3^f \amalg {}_0^e,
\end{aligned}$$

There is a correspondence between subdivided borders of a face and the borders of subdivided faces, which can be expressed by the following incidence constraints:

$$\begin{aligned}
T_0^f \amalg {}_0^f &= \amalg_0^f T_0^e \\
T_1^f \amalg {}_0^f &= \amalg_0^f T_1^e \\
T_1^f \amalg {}_1^f &= \amalg_1^f T_1^e \\
T_3^f \amalg {}_1^f &= \amalg_1^f T_0^e \\
T_3^f \amalg {}_2^f &= \amalg_2^f T_1^e \\
T_2^f \amalg {}_2^f &= \amalg_2^f T_0^e \\
T_2^f \amalg {}_3^f &= \amalg_3^f T_0^e \\
T_0^f \amalg {}_3^f &= \amalg_3^f T_1^e,
\end{aligned}$$

One can prove that these constraints imply coherence between the borders on all levels of subdivision.

To ensure the continuity of the shape, subdivided faces can also share edges. These relations can be translated into adjacency constraints between borders of the subdivided faces:

$$\begin{aligned}
T_0^f \amalg {}_1^f &= T_1^f \amalg {}_3^f \\
T_1^f \amalg {}_2^f &= T_3^f \amalg {}_0^f \\
T_3^f \amalg {}_3^f &= T_2^f \amalg {}_1^f \\
T_2^f \amalg {}_0^f &= T_0^f \amalg {}_2^f,
\end{aligned}$$

All the incidence and adjacency constraints are thus illustrated in figure 3.16. In this case, there are twelve constraints that guarantee the absence of holes in the surface. These constraints define a system of constraints on subdividing and embedding operators and establish an equivalence of paths in the automaton for the first level of subdivision. The equivalence on the next levels of subdivision can thus be derived from this system of equations.

Objects modelled using BCIFS do not necessarily have specific fractal properties; shapes with a classic topology can be modelled as well. The topological constraints of BCIFS guarantee the topological structure of the limit shape. However, by choosing appropriate initial compact sets, these constraints guarantee the coherence and integrity of the topological structure for all the approximation levels as well. Moreover, the topological constraints are easy to solve regardless of the nature of the embedding operators [Gou09].

For simplification purposes, in this thesis, we primarily describe the ideas for an IFS, keeping in mind that generalization to a CIFS is immediate.

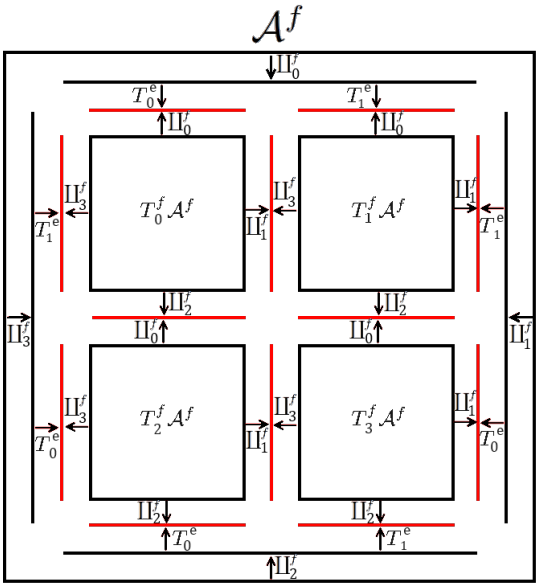


Figure 3.16: All the incidence and adjacency constraints for a square subdivided into four similar squares.

3.4 Manipulations with iterative models

Despite the difficulty of controlling the final shape defined by an iterative system there exist construction algorithms and methods to manipulate their attractors. In this section we describe some existing algorithms.

3.4.1 Model reconstruction

Much work has been done on the so-called "inverse problem" for IFS and its generalisations. This problem involves developing methods to find a set of contractive mappings whose attractor is close to a given shape in the sense of a predefined error measure.

Barnsley [Bar88] proposed a method to determine the transformations associated with a given attractor, called the *Collage Theorem*. The theorem states that to find an IFS whose attractor is close to a given set, one must determine a set of contraction mappings on a suitable space, within which the given set lies. The union of the images of the given set under these transformations must be close to the given set with respect to the HAUSDORFF metric. More precisely, the theorem implies that finding an IFS $\{\mathbb{X}; T_i \mid i \in \Sigma\}$ whose attractor is close to a given image K is equivalent to the minimization of the distance

$$d_{\mathbb{X}}(K, \bigcup_{i\in\Sigma} T_i(K)).$$

Roll, Lutton et al. [RLG+95] developed a method to solve the "general" inverse problem for IFS within a reasonable computation time by using variable-

sized structures in the genetic programming algorithm, which seems to perform a more efficient search in a large space.

3.4.2 Bounding the attractor

Methods to approximate the convex hull of an IFS attractor have already been developed.

Strichartz, Wang, Kenyon et al. [SW98, KLSW99] studied the boundaries and the convex hulls of self-affine tiles that can be considered as attractors of very special affine IFS, for which all the transformations have the same linear part.

Lawlor and Hart [LH03] presented an algorithm to construct a tight bounding polyhedron of the IFS attractor. An algorithm expresses the IFS-bounding as a set of linear constraints on a linear objective function, which can then be solved via standard techniques for linear convex optimization. This method works for a predetermined number of convex hull faces and shows the interactive rate only when this number is small.

More recently, Duda [Dud07] and Martyn [Mar09a] have presented methods to calculate the approximate convex hull of an affine IFS attractor. The two methods are similar. The first is based on so-called "width function" that maps a point to the nearest bounding half-space in a given direction. The second is based on constructing a sequence of balls that bound corresponding parts of the attractor to approximate the convex hull. This latter approach is presented in 2D only.

Another well studied problem involves determining a bounding ball for the attractor of an IFS.

Gentil [Gen92] described an approach based on the dichotomous search for the minimal radius of a ball bounding the IFS attractor. The approach can be applied in a multi-dimensional space and the result is calculated for a given precision.

Hart and DeFanti [HD91] introduced a method which starts with a unit ball centred at the origin. The algorithm iteratively produces a sequence of balls converging to the limit ball that bounds the attractor. Rice [Ric96] improved the approach of Hart and DeFanti by optimizing the radius of the bounding ball using the generic optimization package. He also showed that the centre of the limit ball can be determined analytically by solving a system of linear equations. Martyn [Mar09a] showed that the solution of this system is the centroid of the attractor with particular weights. To obtain a better approximation, he presented an heuristic iterative method, called "balancing the attractor". The algorithm is not limited by the dimension of the space in which the attractor lies.

More recently, Martyn [Mar09b] presented a novel approach to approximate the smallest disc to enclose an affine IFS attractor at any accuracy. The method is based on a concept of spanning points he introduced to describe the extent of the IFS attractor.

We studied the convex hull approximation for a given affine IFS attractor [MGLS12]. Our work can be considered as a generalization and an optimization of Martyn's method. The algorithm constructs a sequence of convex hull approximations using the self-similarity property of the attractor in order

to reduce the number of necessary operations. In addition, we have introduced a method to simplify the approximate convex hull without losing accuracy.

3.4.3 Combination approaches

Thollot [Tho96] proposed a model generalising IFS based on Languages Theory. This model permits us to compose two attractors by means of specific language operations, such as union, condensation, mixing and intersection. The author studied the relation between attractors defined by languages L and L' and also the attractor $L \odot L'$ defined by combining these languages. Certain operations were introduced and studied. The attractor defined by condensation of these languages can be interpreted as the attractor associated with the first language fabricated by iterating the attractor of the second one. Mixing is the operation that "fills the space" between two initial attractors. The language intersection of two IFS is equivalent to the intersection of their sets of transformations. Note that the language intersection is not equivalent to a classic set intersection. In fact two different languages may have the same attractor and the result thus depends on the choice of languages. In addition, this study presented an algorithm to construct the result of these operations in the case of rational languages.

Zaïr and Tosan [ZT96], Thollot et al. [TZTV97] presented an approach to fractal modelling which is based on IFS theory. The authors used free-form techniques to provide a practical and efficient way to build controlled fractal attractors. They also showed that it is possible to manipulate fractals with methods used for classic geometric design, such as control point editing, deformations or tensor products.

Martyn [Mar04] studied the affine stable morphing of 2D affine IFS fractals. The affine stability of metamorphosis can be described by the two following properties:

- intermediate shapes should preserve the topology under affine deformations of the key fractals;
- intermediate shapes should be at least as similar to the key fractals as they are to each other.

The approach presented in the paper satisfies these conditions. The author also considered morphing fractals specified by IFS with different numbers of transformations and gave a number of solutions.

Recently, Podkorytov et al. [PGSL13] developed tools based on BCIFS to construct and to control a junction between two fractal curves. The authors also deduced conditions that guarantee continuity of the intermediate curve by studying the eigenvalues of the subdividing operators. Moreover, the authors suggested a method to control the differential behaviour at the connection points between the initial curves and the intermediate one.

Gentil and Neveu [GN13] developed a formalism based on IFS that builds parametric shapes (curves, surfaces, ...) with a non-uniform local aspect: to every point is assigned a "geometric texture" that evolves continuously from a smooth to a rough aspect. The principle is to blend shapes with uniform aspects to define the shape with a variable aspect. A blending function controls the influence of each initial shape.

Chapter 4

Calculation of CAD operations on fractals

Our goal is to identify the different cases that arise when we want to evaluate CAD operations on fractals and to study the approximation of these operations. In section 2.2, we listed the main standard CAD operations presented in most CAD systems and identified the properties on which the operations are based. A preliminary analysis shows that the operations are based on four basic properties: affine invariance, topological structure, parametrization and differential properties. These properties are therefore essential and we must determine their equivalents in fractal geometry.

In this chapter, we classify these operations by adapting them to fractals in three categories. The first category consists of operators for which the result can be formally represented. The second is composed of operators that require extending or adapting to fractals because of their specific properties. And the third category corresponds to cases for which we can approximate the result.

We then discuss in detail the last category, that in which we can approximate the operation. In studying and identifying the constraints on operations to evaluate this result, we introduce several algorithms. The approximation algorithms are generic, i.e. defined for an arbitrary operation satisfying certain constraints that we state. These algorithms can be applied once we implement the required set of operators (interface).

The proof of convergence of these algorithms is based on Domain theory and computability of functions, as presented by Edalat in solid modelling in [EL02]. In studying measure theory and IFS fractals [Eda95], A. Edalat described a computational model for dynamical systems using the upper space and the probabilistic power domains. More particularly, he developed a constructive framework to study IFS attractors and he described when the attractor of a dynamical system is computable.

4.1 Classification in the context of fractals

Fractals have more complex and more subtle properties than classic shapes, and existing CAD systems are not suitable to manipulate fractal structures. In this section, we study to what extent the standard CAD operators can be defined

and evaluated on fractals. We are primarily interested in shapes represented by BCIFS described in section 3.3.

By analysing the graph of operations presented in section 2.2, we identify the basic properties on which the CAD operations are based. These properties are thus required for their realisation.

A preliminary analysis shows that CAD operations are based on four basic properties: affine invariance, topology, parametrization and differential properties. For shapes described by a BCIFS, the affine invariance property is immediately verified, because objects are defined using control points. In order to apply an affine transformation we can simply modify the control points in the same way as for NURBS surfaces. The topology is formalized by the model and extends concepts of classic topology to fractal topology [Gou09,GTWS10,GHT11]. Parametrization is obtained in terms of a BCIFS address function [Bar88]. Finally, differential properties have been rarely studied [Ben09, BGN08] and, for example, the notion of Frenet basis has still not been adapted for fractal curves.

These basic properties are thus partially defined and it is then possible to adapt CAD operations and algorithms to fractals. Here, we distinguish three cases:

1. The operation result has a formal expression;
2. The operation requires a generalization because of the specific properties of fractals;
3. The operation result cannot be represented formally at the moment, but it can be approximated.

The first category is the same as in the classification described in section 2.2. An operation Op preserves the BCIFS of an object S, and the output BCIFS has $T(S)$ as its attractor. The first example is consists of applying an affine transformation. As for NURBS surfaces, we simply need to apply the transformation to control points, hence the BCIFS attractor is projected according to the set of transformed control points (see section 3.2.3).

There are certain other examples mentioned in section 3.4, such as tensor products of fractal curves [ZT96], specific language operations [Tho96] (union, condensation, mixing, language intersection), some studies on blending between two fractal curves [PGSL13].

The second category corresponds to operations that have not yet been defined for fractals. For example, "pulling curve onto surface" cannot be used in the original form. This operation projects a given curve onto a given surface along the normal to the surface. Since fractal surfaces are nowhere differentiable, this projection would be discontinuous. To obtain a more intuitive result, it is therefore necessary to redefine or to generalise the operation.

In fact, most of operations in this category depend on the differential properties of shapes. Because of the lack of continuity, applying this kind of operation becomes inconvenient. Examples are "network", determining tangents, or the extrusion by the normal direction. Nevertheless, some of these operations may be generalised to a more convenient analogue. Inversely, in some cases the operations can be restricted to a set of suitable cases, in which they may be applied. For example, we can require a surface to have a tangent plane at any point of an edge to apply a "patch".

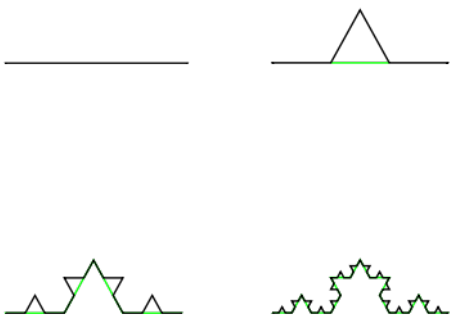


Figure 4.1: Construction steps for the KOCH curve (from left to right, and from top to bottom). The curve is constructed by starting with a single line segment, and by repeating the following three steps recursively: (1) divide the line segment into three segments of equal length, (2) draw an equilateral triangle that has the middle segment obtained from step 1 as its base and points outward, (3) remove the line segment that is the base of the triangle from step 2.

Furthermore, some operations, such as the curvature graph or calculating curve length, are based on properties that are not defined for fractals. For instance, consider the attractor of the VON KOCH curve. Starting with a single line segment, the VON KOCH curve can be constructed by recursively altering each line segment as follows (see figure 4.1):

1. divide the line segment into three segments of equal length;
2. draw an equilateral triangle that has the middle segment obtained from step 1 as its base and points outward;
3. remove the line segment that is the base of the triangle from step 2.

The number of sides N_n on the n-th iteration is therefore:

$$N_n = N_{n-1} \cdot 4 = 4^n,$$

where $N_0 = 1$. After each iteration, the number of sides increases four times, because we add two segments in the middle of each segment.

Let l be the length of the initial segment. The length of each segment L_n on the n-th iteration is thus:

$$L_n = \frac{L_{n-1}}{3} = \frac{l}{3^n}$$

After each iteration, the length of each segment is three times smaller. The length of the VON KOCH curve after n iterations can therefore be determined by the following equation:

$$P_n = N_n \cdot L_n = l \cdot \left(\frac{4}{3}\right)^n \xrightarrow[n\to\infty]{} \infty.$$

The von Koch curve itself is of infinite length.

Operations such as the chamfer operation or dividing a curve by its length cannot be applied directly. These operations must either be restricted to fractal curves with a finite length, or redefined in a more convenient way so that they can be applied to curves with an infinite length (the same for surface and volume).

Certain new operations that are natural for fractals can also be included in this category, such as fractal dimensions, for example.

After all, it is possible to apply operations from this category to an attractor approximation as a classic object. Approximating the attractor is described in detail in chapter 3.3. The time complexity, in this case, depends exponentially on the number of iterations.

For the last category, we develop generic algorithms to approximate the operator result. These algorithms are defined for any operator satisfying certain explicit properties. We show that with some additional constraints these algorithms can be optimized by exploiting the self-similarity property.

4.2 Approximation of a CAD operator image

In this section, we formalise application of an arbitrary CAD operation and we establish the notation used in this thesis. To simplify explanations we describe the idea on IFS; a generalization to CIFS is thus immediate.

To denote a CAD operation we use the symbol Op and the function notation. Note that the operator Op may be any algorithm that takes a non-empty compact subset B in a complete metric space $\mathbb{X}$ as input, and generates the resulting set $Op(B)$. We then describe the algorithm to approximate the image $Op(\mathcal{A})$, where $\mathcal{A}$ is the attractor of our iterative system, and we show how to optimize the algorithm when the operator verifies certain specific properties.

Definition. Given a hyperbolic IFS $\{T_i\}_{i=0}^{N-1}$ defined in a complete metric space $(\mathbb{X}, d_{\mathbb{X}})$ with the attractor $\mathcal{A}$. Let $\Sigma = \{0, \ldots, N-1\}$ be the set of IFS transformation indices. An arbitrary operator Op could be any algorithm that takes a non-empty compact subset in a complete metric space $\mathbb{X}$ as input, i.e.:

$$Op: \ \mathcal{H}(\mathbb{X}) \to \mathbb{Y},$$

where $\mathbb{Y}$ denotes the target space of the operator. This space may be a metric space, as for most CAD operations. However, it may also be a non-metric space, as for the membership predicate or any other classifying operation.

In studying measure theory and IFS [Eda95], A. Edalat developed a constructive framework to study IFS attractors. He determined when the attractor of a dynamical system is computable. As mentioned in section 3.1.3, there are two canonical ways to evaluate the attractor of a dynamical system:

- By constructing a decreasing sequence of non-empty compact sets which shrinks down to the attractor.
- By constructing an increasing sequence of non-empty compact sets whose union has the attractor as its closure.

In fact, in both cases we use similar approximation algorithms, i.e. we recursively apply transformations to an initial non-empty compact subset.

Starting with an arbitrary compact subset $B \in \mathcal{H}(\mathbb{X})$, we construct a sequence of compact sets converging to the attractor $\mathcal{A}$:

$$(B_n)_{n\in\mathbb{N}} = \mathbb{T}^n(B),$$

where $\mathbb{T}$ is the HUTCHINSON operator corresponding to a given IFS. Figure 4.2 illustrates this process.

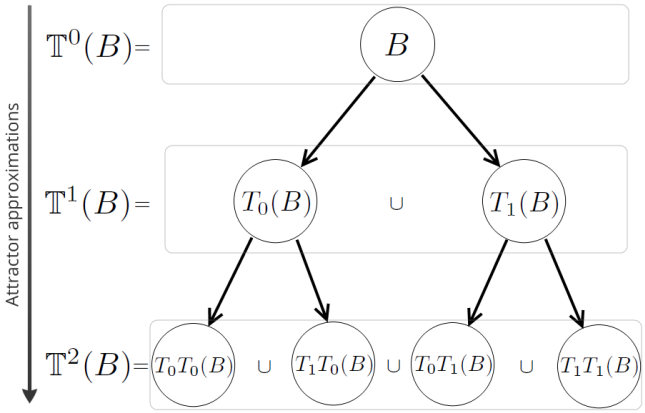


Figure 4.2: Approximation of the attractor illustrated by a tree whose leaves are the images of an initial compact subset B by applying sequences of IFS transformations.

Statement. We denote $r = d_{\mathbb{X}}(\mathcal{A}, B)$. The HUTCHINSON operator is contractive, we therefore have:

$$d_{\mathbb{X}}(\mathcal{A}, B_n) \leqslant s(\mathbb{T})^n \cdot r \xrightarrow[n\to\infty]{} 0,$$

where $s(\mathbb{T}) < 1$ is the contraction coefficient of the HUTCHINSON operator and n is the number of iterations.

Therefore, $\forall\varepsilon > 0 \;\; \exists m \in \mathbb{N} : \forall n \geqslant m \;\; d_{\mathbb{X}}(\mathcal{A}, B_n) < \varepsilon$, or equivalently,

$$B_n \xrightarrow[n\to\infty]{} \mathcal{A}.$$

The HUTCHINSON operator is thus continuous at the attractor $\mathcal{A}$.

The idea of our generic algorithm is to apply the operation to attractor approximations, i.e. to calculate $Op(B_n)$, in order to obtain a sequence of approximate results. If the operator Op is continuous at the point $\mathcal{A}$, then this sequence converges to the required result $Op(\mathcal{A})$.

For instance, we consider the operator determining the distance from a given point to the attractor. Let $p \in \mathbb{X}$ be a point; the operator Op is then defined as follows:

$$Op_p(\mathcal{A}) : \mathcal{A} \mapsto \min_{a\in\mathcal{A}} d(p, a).$$

In this case, the space $\mathbb{Y}$ is the set of real numbers ($\mathbb{R}$). One can show that this operator is continuous at point $\mathcal{A}$ and, moreover, that it is a LIPSCHITZ function.

We can therefore construct the attractor approximations B_n and compute the images $Op(B_n)$.

Statement. For a given accuracy ε, the number of iterations is determined as follows:

$$n = \left\lceil \frac{\log(\frac{\varepsilon}{Lr})}{\log(s(\mathbb{T}))} \right\rceil, \tag{4.1}$$

where $r = d_{\mathbb{X}}(\mathcal{A}, B)$, $s(\mathbb{T}) < 1$ is the contraction coefficient of the HUTCHINSON operator and L is the LIPSCHITZ constant of the operation ($L = 1$ in this example).

4.3 Continuity of the generic algorithm

Definition. An operator Op is called *continuous at point* $\mathcal{A}$ iff for any sequence $(B_n)_{n\in\mathbb{N}}$ of non-empty compact subsets of $\mathbb{X}$ converging to attractor $\mathcal{A}$, the corresponding sequence $(Op(B_n))_{n\in\mathbb{N}}$ converges to $Op(\mathcal{A})$.

In general, most CAD operations are not continuous. However, in our case we construct only particular sequences defined by applying the HUTCHINSON operator $\mathbb{T}$ and convergence for any sequence is unnecessary. This issue is closely related to computability in solid modelling [EL02] and more particularly to the question of domain choice.

For example, such basic predicates and operations as membership, subset inclusion and intersection are not continuous and, therefore, are not computable. However, in some cases we can restrict the domain of the operator Op (that is, the choice of the sequences $(B_n)_{n\in\mathbb{N}}$) to guarantee the continuity at the attractor $\mathcal{A}$.

4.3.1 Directed complete and continuous posets

In this section we introduce the basic notation of domain theory and we explain how continuous domains can be effectively presented to guarantee the convergence of our algorithms.

Definition. A set P with a binary relation $\sqsubseteq$ is called a *partially ordered set* (poset), if the following conditions are satisfied $\forall x, y, z \in P$:

- $x \sqsubseteq x$ (Reflexivity)
- $x \sqsubseteq y \;\&\; y \sqsubseteq z \Rightarrow x \sqsubseteq z$ (Transitivity)
- $x \sqsubseteq y \;\&\; y \sqsubseteq x \Rightarrow x = y$ (Antisymmetry)

Figure 4.3 illustrates two simple examples of partially ordered sets.

Definition. A non-empty subset $D \subseteq P$ of a poset $(P, \sqsubseteq)$ is *directed* iff for any pair of elements $x, y \in D$ there exists an upper bound $z \in D$ with $x, y \sqsubseteq z$. In other words, a directed set is a non-empty set with a reflexive and transitive binary relation $\sqsubseteq$, with the additional property that every pair of elements has an upper bound.

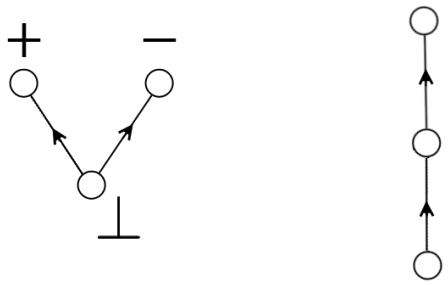


Figure 4.3: Two simple partially ordered sets. The pointed domain of booleans $\{+,-\}_\perp$ is illustrated on the left, and an increasing chain on the right.

Directed sets are used to define nets, which generalize sequences. An increasing chain is the simplest example of a directed set. In the theory of information, a directed set D corresponds to a consistent set of inputs and outputs of a given program (see figure 4.4): for any two elements in D, there exists an element which refines the information of both elements. The total information in a directed subset must therefore be represented by an element of the domain. A domain should thus contain a least upper bound of each directed subset.

Definition. A *directed complete partial order (dcpo)* is a partial order in which every directed subset D has a least upper bound (lub), denoted by $\bigsqcup D$.

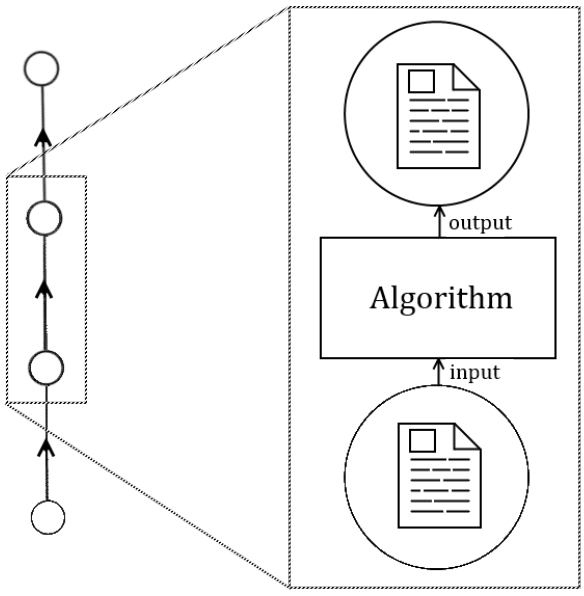


Figure 4.4: In the theory of information, a directed set corresponds to a consistent set of inputs and outputs of a given program.

There is a natural topology on dcpos given by Scott.

Definition. Let $(P, \sqsubseteq)$ be a dcpo. A set $O \subseteq P$ is open iff:

1. $x \in O$ and $x \sqsubseteq y \Rightarrow y \in O$;
2. whenever $D \subset P$ is directed set and $\bigsqcup D \in O$ then $D \cap O \neq \emptyset$.

The first condition thus introduces the nature of open sets in the SCOTT topology and the second condition means that the border of the set O is not included.

Definition. An operator $F : P \to P'$ from a dcpo P to another dcpo P' is *monotone* if for any directed set $D \subseteq P$ and for any pair of elements $x, y \in D$,

$$x \sqsubseteq_P y \Rightarrow F(x) \sqsubseteq_{P'} F(y).$$

Definition. An operator F is *continuous* iff F is monotone and it preserves a least upper bound of directed sets, i.e.

$$\bigsqcup_{x \in D} F(x) = F(\bigsqcup_{x \in D} x),$$

for all directed subsets $D \subseteq P$.

The key idea of transitioning to domain theory is that functions may be discontinuous with respect to the HAUSDORFF metric, but continuous in the less restrictive SCOTT topology.

Consider the following example. Let $f : \mathbb{R} \to \mathbb{R}$ be a real-valued function defined by:

$$f(x) = \begin{cases} x + 1 & \text{if } x > a, \\ x & \text{otherwise.} \end{cases}$$

The plot of this function is shown in figure 4.5.

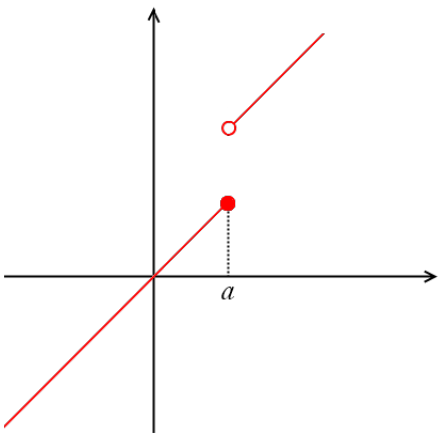


Figure 4.5: Plot of the function f considered in the example.

In the standard topology this function is not continuous. As illustrated in figure 4.6, in the standard topology, a neighbourhood of the point a is bounded on both sides. By considering the pre-image of this neighbourhood, we do not

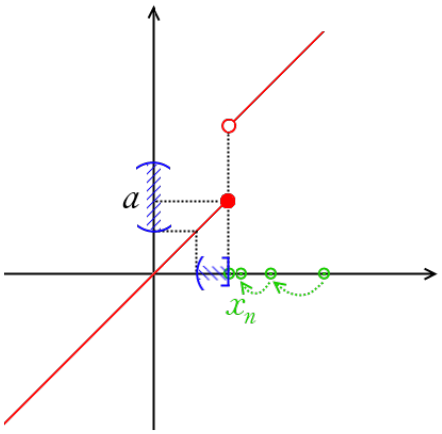


Figure 4.6: In the standard topology this function is not continuous, but lower semi-continuous.

obtain an open set. Thus, there exists a sequence $x_n \to a$ whose tail does not fall entirely in the neighbourhood.

However, this function is lower semi-continuous. Note that a function $F : \mathbb{X} \to \mathbb{R}$ for a topological space $\mathbb{X}$ is lower semi-continuous iff the set $\{x \in \mathbb{X} \mid F(x) > r\}$ is open for all $r \in \mathbb{R}$. This function is thus continuous with respect to the Scott topology on $\mathbb{R}$. Indeed, a neighbourhood of the point a is only bounded on the bottom. As illustrated in figure 4.7, the pre-image of this neighbourhood is always open. Thus, we remove from consideration the sequences converging on the right side.

This simple example demonstrates that convergence in the Scott topology is less restrictive than in the classic topology. In general, most CAD operations are not continuous. Continuity in the Scott topology requires that a smaller set of sequences converge. Moreover, when applying the operation to fractals we construct only a single sequence defined by applying an iterative algorithm and convergence for all sequences is unnecessary.

We will now define the so-called order of approximation, which is also more suggestively called the way-below relation.

Definition. An element $x \in P$ is *way-below* $y \in P$, or equivalently x *approximates* y, denoted by $x \ll y$, iff whenever $y \sqsubseteq \bigsqcup D$ for a directed set D with a lub, there exists $z \in D$ such that $x \sqsubseteq z$. This condition implies that $x \sqsubseteq y$, since the singleton set $\{y\}$ is directed.

Consider the interval domain with the subset inclusion order:

$$[a, b] \sqsubseteq [c, d] \Leftrightarrow [a, b] \supseteq [c, d],$$

where $[a, b]$ and $[c, d]$ are intervals. One can show that

$$[a, b] \ll [c, d] \Leftrightarrow (a, b) \supseteq [c, d].$$

Indeed, if the intervals share the left border a, there exists a directed set $D = \{[a - \frac{1}{n}, a] \mid n \in \mathbb{N}\}$ with the lub a. There is no element $z \in D$ such that

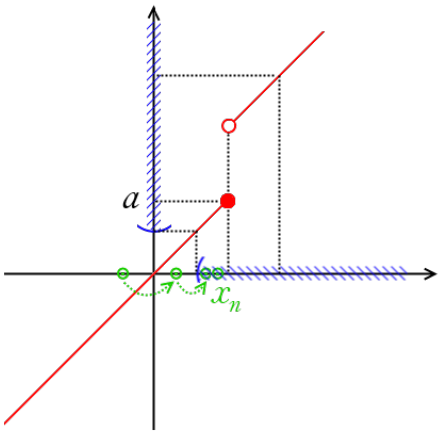


Figure 4.7: Lower semi-continuous functions are continuous in the SCOTT topology.

$[a, d] \sqsubseteq z$. The interval $[a, d]$ thus does not approximate the interval $[a, b]$. By analogy, there is no interval $[c, b]$ that approximates $[a, b]$. Whenever the interval $[c, d]$ is strictly included in $[a, b]$, one can find a small enough interval z that lies inside $[a, b]$. The way-below relation thus means that all the boundaries of the object x are refined by passing to the approximated element y.

However, being way-below of an element is a relative notion and does not reveal much about the element alone. For example, one would like to characterize finite sets in an order-theoretic way, but even infinite sets can be way-below any other set. The special property of these finite elements is that they are way below themselves, i.e. $x \ll x$. An element with this property is also called *compact*. Yet, such elements need not be finite nor compact in any other mathematical usage of the terms. Compact elements of a domain have the important special property that they cannot be obtained as a limit of a directed set in which these elements did not already occur.

The previous thoughts raise another question: is it possible to guarantee that any element of a domain can be obtained as a limit of much simpler elements? This is quite relevant in practice, since we cannot compute infinite objects but we may still hope to approximate these objects arbitrarily closely. More generally, we would like to restrict to a certain subset of elements as being sufficient to obtain all other elements as least upper bounds.

Definition. A *basis* of a poset P is a subset $B \subseteq P$, such that, for each $x \in P$, the set of elements in B that are way-below x contains a directed set with lub x. We say P is *continuous* if it has a basis. The poset is ω-continuous if it has a countable basis.

A dcpo is *pointed* if it has a least element. We can always add a bottom element to a domain to make it pointed.

4.3.2 Solid domain

In this section we introduce a domain chosen to represent the attractor of iterative systems and to extend the operator in a continuous and computable way.

Definition. The solid domain of a topological space $\mathbb{X}$ is the set of ordered pairs (A, B) of disjoint open subsets of $\mathbb{X}$ endowed with the information order:

$$(A_1, B_1) \sqsubseteq (A_2, B_2) \Leftrightarrow A_1 \subseteq A_2 \text{ and } B_1 \subseteq B_2.$$

An element (A, B) is called a *partial solid*: A and B are intended to capture, respectively, the interior and the exterior of a solid, possibly, at some finite stage of computation (see figure 4.8). The border of the solid is thus somewhere in between. The solid domain is a mathematical model to represent rigid solids.

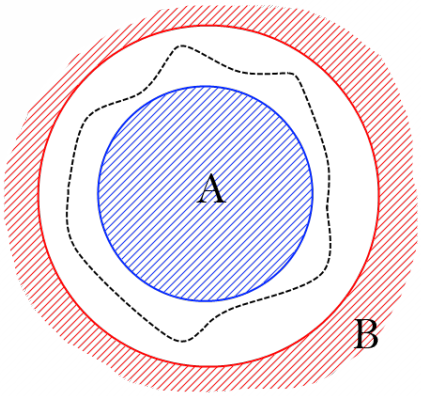


Figure 4.8: Partial solid (A, B). The sets A and B are intended to capture, respectively, the interior and the exterior of a solid, possibly, at some finite stage of computation.

A. Edalat and A. Lieutier have shown [EL02] that the basic predicates and operations are continuous and computable in the solid domain $(S\mathbb{X}, \sqsubseteq)$. We can thus represent the attractor in the solid domain and then apply continuous operations to it.

A. Edalat presented a domain-theoretic model for iterated function systems [Eda97]. He constructed computational models for classic spaces using continuous dcpo and provided applications in IFS theory. A domain-theoretic model for IFS gives a unifying framework to study various properties and provides a set of new results in the theory and applications of these systems. In particular, this domain-theoretic approach allows us to prove convergence of the developing algorithms and to approximate operations on fractals.

In the study of computable partial solids, A. Lieutier [Lie99] introduced conditions on the domain to be an ω-continuous dcpo and showed when a continuous function $f : D \to E$ on the ω-continuous domains D and E is computable.

Definitions.

- A function is called *recursive* if the program computing this function ends within a finite number of steps for each input.
- A function is called *partial recursive* iff for some values given as inputs, a program computing this function may run forever.
- *Computable functions* are partial functions that can be computed by the TURING machine.

Definitions. We use the equivalent notion of *partial recursive functions* as a map that can be computed by a general purpose computer or a (finite length) program written in a general purpose language. For a countable set $\mathbb{X}$:

- a subset $O \subseteq \mathbb{X}$ is called *recursive*, if there is an algorithm which decides, whether a given element $k \in \mathbb{X}$ is in O or not;
- a subset $O \subseteq \mathbb{X}$ is called *recursively enumerable*, if it is empty or if there is a recursive function which lists all elements $k \in O$;

Definition. In a metric space $\mathbb{X}$, a sequence $(O_n)_{n\in\mathbb{N}}$ is said to converge *effectively* toward $O \subseteq \mathbb{X}$ if and only if there exists a recursive function f such that:

$$\forall m, n \in \mathbb{N}, m \geqslant f(n) \Rightarrow d_{\mathbb{X}}(O, O_m) \leqslant 2^{-n}.$$

Note that, in defining the computability of $O \subseteq \mathbb{X}$, the existence alone of a computable sequence of approximations converging to O is not sufficient; the convergence has to be effective.

If $\mathbb{X}$ is a second countable locally compact HAUSDORFF space, the solid domain is thus ω-continuous bounded complete dcpo's. An ω-continuous domain can be effectively presented with respect to the enumeration of a basis by requiring that the way-below relation restricted to the basis elements be recursively enumerable. This requirement allows us to compute the number of iterations n needed to achieve accuracy ε.

4.3.3 Attractor in the solid domain

Generally speaking, most CAD operations are not continuous and therefore not computable. Figure 4.9 illustrates the overall idea of our method. We construct a representation of the attractor in the solid domain and then apply a continuous extended operation to it. We use the symbol SOp to denote the continuous extension of the operator Op to the solid domain:

$$SOp \ : \ S\mathbb{X} \to S\mathbb{Y}.$$

For any subset K of a topological space, ∂K, $\bar{K}$, K^o and K^{c} denote, respectively, the border, the closure, the interior and the complement of K.

To represent the attractor $\mathcal{A}$ in the solid domain we consider the partial solid $(\mathcal{A}^o, \mathcal{A}^{\mathsf{c}})$. However, constructing the attractor interior is quite complex, and we therefore often use the empty set as an approximate interior, i.e. $(\emptyset, \mathcal{A}^{\mathsf{c}})$, as illustrated in figure 4.10. This partial solid is not the maximal element and, generally, is not a classic solid. However, to apply for example the boolean operations or the MINKOWSKI sum, it is sufficient to consider the exterior only.

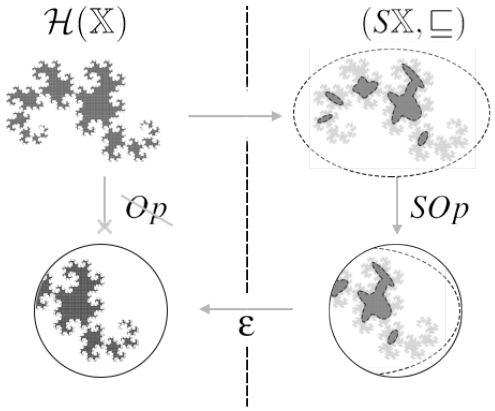


Figure 4.9: We consider a continuous extension of the operator SOp on an ω-continuous dcpo $(D, \sqsubseteq)$. Thus we construct a representation of the attractor in this domain, apply the continuous operation to it and then return to the modelling space $\mathbb{X}$ by approximating the attractor with the accuracy ε.

Note that for such basic operations as membership predicate, subset inclusion or difference between two fractals, one must also find a non-empty approximation of the attractor interior.

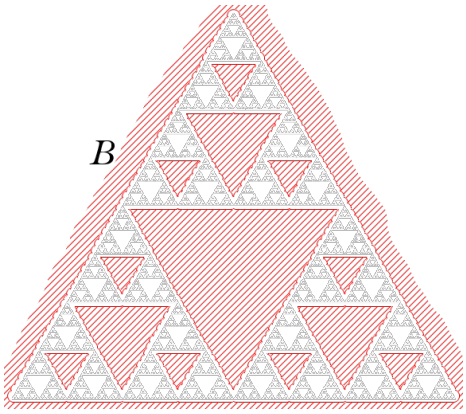


Figure 4.10: Attractor $\mathcal{A}$ represented in the solid domain by a pair of open sets $(\emptyset, \mathcal{A}^c)$. We use the empty set as an interior approximation.

In fact, when the interior of a partial solid is empty, this means that we do not know the attractor interior A^o. The operation is therefore computed using only the exterior approximations. The membership predicate, for example, is semi-decidable in this case, because we can say if a point is outside the attractor, but cannot determine if the point is inside.

Statement. When avoiding the approximation of the attractor interior, we do not distinguish the objects (A_1, B) and (A_2, B) if the following equality holds:

$$(\bar{A}_1)^o = (\bar{A}_2)^o.$$

The objects illustrated in figure 4.11 will not be distinguished without approximating the interior. However, this is never the case for IFS (or CIFS) attractors. These models are defined on a set of non-empty compact subsets of a complete metric space $\mathbb{X}$. The attractors are therefore compact sets and the attractors with the same exterior cannot have different interiors.

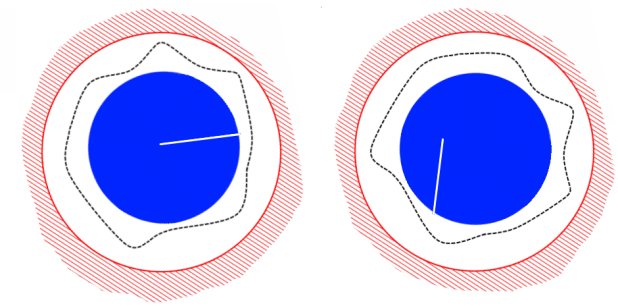


Figure 4.11: Two partial solids which differ on a set of a zero measure. These partial solids will not be distinguished without approximating the interior.

4.3.4 Algorithm convergence

In this section we discuss convergence of constructed partial solids to the attractor of a given iterative system. We identify necessary conditions for the initial partial solid to constitute a directed set with the attractor as the lub. Here we assume the operator SOp to be continuous in the Scott topology.

Theorem. Let A_n be a sequence of open sets, B_n be a sequence of non-empty compact sets and $S\mathcal{A} = (\mathcal{A}^o, \mathcal{A}^c)$ be the attractor of our iterative system in the Scott topology. Let SOp be a continuous operator (for example, a boolean operator or the Minkowski sum [EL02]). If the partial solids $SB_n = (A_n, B_n^c)$ constitute a directed set with the lub $S\mathcal{A}$ then we can write the following equation:

$$\bigsqcup_n SOp((A_n, B_n^c)) = SOp(\bigsqcup_n (A_n, B_n^c)) = SOp((A^o, \mathcal{A}^c)).$$

The proof is based on defining continuous functions in the solid domain.

The simplest and most commonly used example is when this sequence SB_n constitutes an increasing chain, where we thus improve the information about the attractor with each iteration. The sequence $SOp(SB_n)$ thus converges to the image of the attractor $S\mathcal{A}$ and this sequence can be computed with respect to the effective structure on $S\mathbb{X}$, as described in section 4.3.2. Note that the image under the Hutchinson operator $\mathbb{T}(SB)$ gives $(\mathbb{T}(A), \mathbb{T}(B)^c)$. We can then apply the IFS transformation to partial solids in the same manner.

In fact, constructing $SOp((A_n, B_n^c))$ is equivalent to constructing $Op(A_n)$ and $Op(B_n)$. In this case, we use very particular sequences of approximations and we therefore converge to $Op(\mathcal{A})$.

To exploit the solid domain, we should firstly verify that the sequence of the partial solids (A_n, B_n^c) constitutes a directed set and, more precisely, an

increasing chain. We need to verify the following inequality:

$$(A_0, B_0^{\mathsf{c}}) \sqsubseteq (A_1, B_1^{\mathsf{c}}) \sqsubseteq \ldots \sqsubseteq (A_n, B_n^{\mathsf{c}}),$$

or equivalently:

$$\begin{aligned} A_0 &\subseteq A_1 \subseteq \ldots \subseteq A_n \\ B_0^{\mathsf{c}} &\subseteq B_1^{\mathsf{c}} \subseteq \ldots \subseteq B_n^{\mathsf{c}}, \end{aligned}$$

It is easy to show that by choosing an initial compact subset B such that $T_i(B) \subseteq B \; \forall i \in \Sigma$ (see appendix A) the following inclusions hold:

$$\mathbb{T}^n(B) \subseteq \ldots \subseteq \mathbb{T}^2(B) \subseteq \mathbb{T}(B) \subseteq B,$$

or equivalently:

$$B^{\mathsf{c}} = B_0^{\mathsf{c}} \subseteq B_1^{\mathsf{c}} \subseteq B_2^{\mathsf{c}} \subseteq \ldots \subseteq B_n^{\mathsf{c}} = (\mathbb{T}^n(B^{\mathsf{c}})).$$

The interior part $A = A_0$ of the initial partial solid must thus be chosen to satisfy the following constraint:

$$A \subseteq \mathbb{T}(A).$$

If the interior A is chosen properly, the sequence of partial solids (A_n, B_n^{c}) constitutes a directed set and the sequence of $Op(B_n)$ converges to the required image $Op(\mathcal{A})$. Note that if the initial interior part A_0 is chosen as the empty set, the sequence of partial solids (A_n, B_n^{c}) also constitutes a directed set. However, in this case the lub of this set would be $(\emptyset, \mathcal{A}^{\mathsf{c}})$.

Let $B \in \mathcal{H}(\mathbb{X})$ be a non-empty compact subset and $A \subset B$ be a non-empty open set. Let the sequence $SB_n = (A_n, B_n^{\mathsf{c}})$ be a directed set constructed by applying the HUTCHINSON operator to (A, B^{c}). In order to identify the lub of this sequence, consider the partial solid (Q_i, Q_e) such that:

$$(A_n, B_n^{\mathsf{c}}) \sqsubseteq (Q_i, Q_e) \;\; \forall n.$$

By the definition of partial order in the solid domain, this system of inequalities is equivalent to the following:

$$A_n \subseteq Q_i \text{ and } B_n^{\mathsf{c}} \subseteq Q_e,$$

or

$$A_n \subseteq Q_i \text{ and } Q_e^{\mathsf{c}} \subseteq B_n. \tag{4.2}$$

Now we consider the relation between the attractor $\mathcal{A}$ and the set Q_e. Let us assume that $\mathcal{A} \subset Q_e^{\mathsf{c}}$. Let a be a point of $Q_e^{\mathsf{c}} \setminus \mathcal{A}$. This means that a does not belong to the attractor and there exists a non-zero distance between them:

$$d(a, \mathcal{A}) = \epsilon > 0.$$

Since the sequence of compact subsets B_n converges to the attractor $\mathcal{A}$ with respect to the HAUSDORFF distance, there exists a number $m \in \mathbb{N}$ such that:

$$\forall n \geqslant m \;\; d_{\mathbb{X}}(\mathcal{A}, B_n) < \epsilon$$

This implies that the point a does not belong to any of these sets B_n for $\forall n \geqslant m$:

$$a \notin B_n \text{ but } a \in Q_e^{\mathsf{c}} \setminus \mathcal{A} \subset Q_e^{\mathsf{c}} \subseteq B_n.$$

The contradiction means that the set $Q_e^{\mathsf{c}} \setminus \mathcal{A} = \emptyset$ or equivalently:

$$Q_e^{\mathsf{c}} \subset \mathcal{A} \Leftrightarrow \mathcal{A}^{\mathsf{c}} \subset Q_e.$$

Both sets Q_e and $\mathcal{A}^{\mathsf{c}}$ are open and the set Q_e can thus be represented as the union of $\mathcal{A}^{\mathsf{c}}$ with an open set $C \subseteq \mathbb{X}$, i.e.:

$$Q_e = \mathcal{A}^{\mathsf{c}} \cup C. \tag{4.3}$$

Since the open sets Q_i and Q_e are disjoint and by inclusion (4.2), we can therefore write the following:

$$A_n \subseteq Q_i \subset Q_e^{\mathsf{c}} = \mathcal{A} \cap C^{\mathsf{c}} \subseteq \mathcal{A}.$$

Thus:

$$Q_i \subseteq \mathcal{A}^o. \tag{4.4}$$

Hence the set A_n is open and non-empty, there exists a compact subset $A' \subset A_0$ and we can write the following:

$$\mathbb{T}^n(A') \subset A_n \subset \mathcal{A} \cap C^{\mathsf{c}} \subseteq \mathcal{A}.$$

One can thus derive the following inequality for the distances:

$$d_{\mathbb{X}}(\mathbb{T}^n(A'), \mathcal{A} \cap C^{\mathsf{c}}) \leqslant d_{\mathbb{X}}(\mathbb{T}^n(A'), \mathcal{A}) \xrightarrow[n\to\infty]{} 0.$$

By uniqueness of the limit, the sets $\mathcal{A} \cap C^{\mathsf{c}}$ and $\mathcal{A}$ are equal, i.e.:

$$\mathcal{A} \subseteq C^{\mathsf{c}} \Leftrightarrow C \subseteq \mathcal{A}^{\mathsf{c}}.$$

We can thus simplify the definition (4.3) as follows:

$$Q_e = \mathcal{A}^{\mathsf{c}} \cup C = \mathcal{A}^{\mathsf{c}}. \tag{4.5}$$

One can similarly show that:

$$\bar{Q}_i = \mathcal{A}.$$

Thus we identified the following restrictions on the lub of the directed set SB_n:

$$Q_i \subseteq \mathcal{A}^o; \tag{4.6}$$
$$\bar{Q}_i = \mathcal{A}; \tag{4.7}$$
$$Q_e = \mathcal{A}^{\mathsf{c}}. \tag{4.8}$$

That is, the lub (Q_i, Q_e) exists only when the exterior part Q_e is equal to $\mathcal{A}^{\mathsf{c}}$ and the closure of the interior part equals $\mathcal{A}$.

Note that if the attractor interior is empty, that is $\mathcal{A}^o = \emptyset$, one can write:

$$A_n \subseteq Q_i \subseteq \mathcal{A}^o = \emptyset,$$

which implies that the set A_0 must also be the empty set.

Thus we identified the lub of the sequence SB_n to be the partial solid $(D, \mathcal{A}^c) \sqsubseteq S\mathcal{A}$, where $\bar{D} = \mathcal{A}$. The only possibility for SB_n to have the lub $S\mathcal{A}$ is to choose the initial open set $A_0 = \mathcal{A}^o$, which corresponds to the case $D = \mathcal{A}^o$. In fact, another set D prevents the convergence $SB_n \to S\mathcal{A}$ with respect to the SCOTT topology.

Consider the following example. Let $\{\mathbb{X}; T_i \mid i = 0, \ldots, 3\}$ be an IFS subdividing square in four parts, as illustrated in figure 4.12.

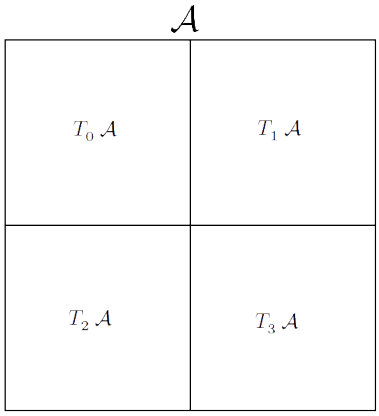


Figure 4.12: Subdivision of a square by four transformations.

Let $A = A_0 \subseteq \mathcal{A}^o$ be an open set and $B = B_0$ be a bounding ball of this square such that $\mathcal{A} \subseteq \mathbb{T}(B) \subseteq B$, as illustrated in figure 4.13. On the following iteration we apply the HUTCHINSON operator to this partial solid (see figure 4.14). The resulting set is no longer connected, since the images of a border appear in the middle of the square.

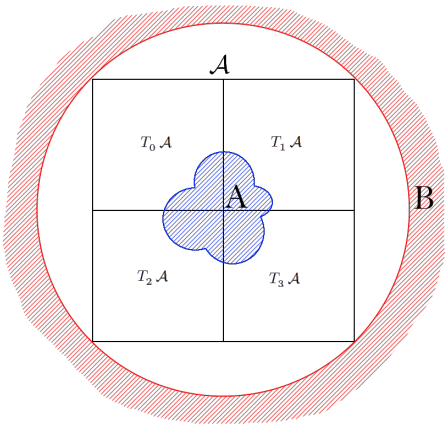


Figure 4.13: Initial partial solid (A_0, B_0^c) considered in the example.

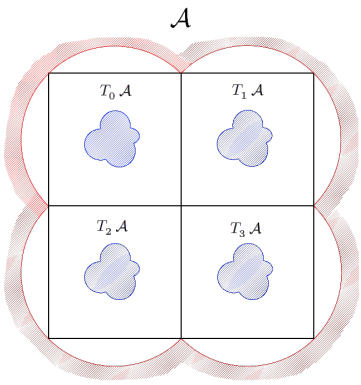


Figure 4.14: The first iteration partial solid (A_1, B_1^{c}) considered in the example.

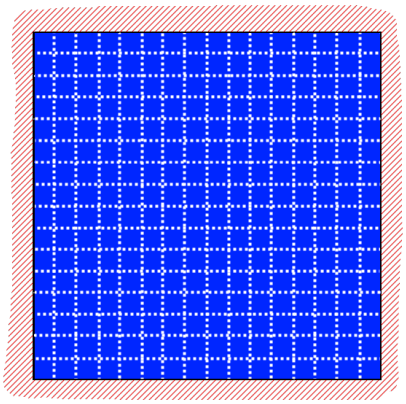


Figure 4.15: The n-th iteration partial solid (A_n, B_n^{c}) considered in the example.

On the n-th iteration we obtain the set B_n, which is close enough to $\mathcal{A}^{\mathsf{c}}$, and the set A_n, for which there is always some empty space near the middle of the square.

Objects described by BCIFS are usually so-called just-touching IFS (CIFS). This means that the subdivided attractor parts intersect only by the boundary. The interiors of these parts do not intersect with each other:

$$\forall i, j \in \Sigma,\ i \neq j \quad T_i(\mathcal{A}^o) \cap T_j(\mathcal{A}^o) = \emptyset.$$

By choosing the initial interior set $A \subseteq \mathcal{A}^o$ we always obtain a case similar to that considered in the example. For these BCIFS, the empty set is the single option for the initial interior part in order to construct a directed set:

$$SB_n = (\emptyset, B_n^{\mathsf{c}}).$$

However, this directed set has the lub $(\emptyset, \mathcal{A}^{\mathsf{c}})$.

Nevertheless, for some operations such as the distance from a point for example (see section 6.2), the interior part of the operator image $SOp(SB)$ can be constructed even without approximating the interior part of the attractor.

In addition, constructing a directed set is not the only way to have the convergence $SOp(SB) \to SOp(SA)$. In some cases, even if the partial solids $SB_n = (A_n, B_n^{\mathsf{c}})$ do not constitute a directed set the convergence can still be stated.

Statement. If the sequence SB_n converges to the attractor $S\mathcal{A}$ with respect to the Scott topology, then by the Scott-continuity of the operator SOP we can write the following:

$$SOp((A_n, B_n^{\mathsf{c}})) \to SOp((A^o, \mathcal{A}^{\mathsf{c}})).$$

Consider the following example. Let $\{\mathbb{R}; T_0; T_1\}$ be an IFS with the following transformations:

$$T_0 : \mathbb{R} \to \mathbb{R} : x \mapsto x/2,$$
$$T_1 : \mathbb{R} \to \mathbb{R} : x \mapsto (x+1)/2.$$

The attractor of this IFS system is thus the segment $[0, 1]$.

Consider the images of an arbitrary interval (a, b) under the IFS transformations:

$$\begin{aligned} T_0((a,b)) &= (a/2, b/2), \\ T_1((a,b)) &= ((a+1)/2, (b+1)/2). \end{aligned}$$

We note that these images do not intersect when $b - a > 1$.

Let (A_0, B_0^{c}) be the initial partial solid defined as follows:

$$(A_0, B_0^{\mathsf{c}}) = ((1,3), (-\infty, 0) \cup (3, \infty)).$$

Several first elements of the sequence SB_n are illustrated in figure 4.16. One can prove that the sequence $SB_n = (1/2^n, 1 + 1/2^n)$ converges to the attractor $(\mathcal{A}^o, \mathcal{A}^{\mathsf{c}})$ according to the Scott topology.

Indeed, for any neighbourhood V of the attractor $S\mathcal{A}$ in the Scott topology there exists an open set $U = (U_i, U_e) \sqsubseteq S\mathcal{A}$. By the definition of open sets, the interior part U_i of U has no common borders with the attractor $\mathcal{A} = [0, 1]$:

$$\bar{U_i} \subset \mathcal{A}.$$

There is a non-zero distance between these sets, which implies that $\exists m \in \mathbb{N}$ from which U_i is included in all A_n.

By analogy, there exists a number after which the exterior part U_e of U is included in all B_n^{c}. This consequence implies that after a certain number of iterations we will exceed U_e, that is $U \sqsubseteq SB_n$, or by the definition of Scott open sets:

$$\exists m \in \mathbb{N} \;\; \forall n \geqslant m \;\; SB_n \in V.$$

The sequence SB_n thus converges to the attractor $\mathcal{A}$ according to the Scott topology. The elements SB_n are not comparable to each other and, therefore, do not constitute a directed set.

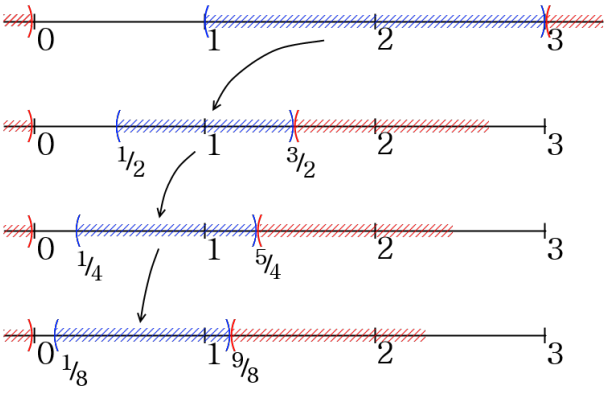


Figure 4.16: Several elements of the sequence SB_n in the considered example. The interior and the exterior parts are in blue and red, respectively.

Finally, constructing a directed set is not limited to constructing an increasing chain. To satisfy the definition of directed sets, we need to find a common maximum for each pair of elements:

$$\begin{aligned}(A_n, B_n^{\mathsf{c}}) &\sqsubseteq (C_{(n,m)}, D_{(n,m)}^{\mathsf{c}}),\\ (A_m, B_m^{\mathsf{c}}) &\sqsubseteq (C_{(n,m)}, D_{(n,m)}^{\mathsf{c}}).\end{aligned}$$

These inequalities are equivalent to the following:

$$\begin{aligned}C_{(n,m)} &\supseteq A_n \cup A_m,\\ D_{(n,m)} &\subseteq B_n \cap B_m.\end{aligned}$$

We consider a special case of this property, called *IFS with condensation*. Let $A = A_0 \subseteq \mathbb{X}$ be an initial open set and $B = B_0$ be the attractor bounding ball such that $\mathbb{T}(B) \subseteq B$. Let $A \subset B$ for (A, B^{c}) be a partial solid. All the following partial solids are constructed by applying the HUTCHINSON operator.

Let $C_0 = A_0$ and $C_1 = A_0 \cup A_1 = A_0 \cup \mathbb{T}(A_0) = \mathbb{T}'(C_0)$, where $\mathbb{T}'$ is the HUTCHINSON operator associated with the IFS: $\{\mathbb{X}; F, T_i \mid i \in \Sigma\}$. Thus we add a transformation $F : K \mapsto A_0$, for $K \subseteq \mathbb{X}$ to the set of transformations.

Introduce the two following sequences:

$$\begin{aligned}C_n &= \mathbb{T}'(C_{n-1}) = \mathbb{T}'^n(C_0) = A_0 \cup A_1 \cup \ldots \cup A_n,\\ D_n &= B_{n-1} \cap B_n = B_n.\end{aligned}$$

The sequence (C_n, D_n^{c}) thus constitutes a directed set. In the beginning of this section, we identified the restrictions on the lub of the directed set SB_n. The new directed set (C_n, D_n^{c}) thus has the same restrictions and does not converge to the attractor $S\mathcal{A}$ according to the SCOTT topology (see figure 4.17).

Finally, one can imagine a completely different method to construct a directed set converging to the attractor. For example, after applying the HUTCHINSON operator we can merge certain subdivided parts based on the adjacency constraints of the BCIFS. We leave this question open for future studies.

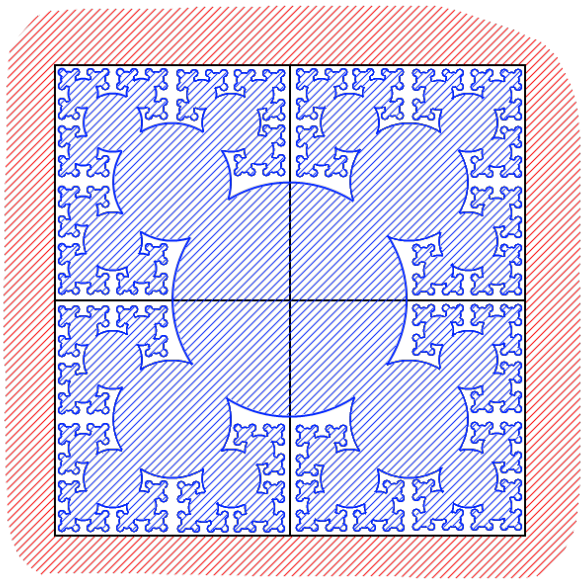


Figure 4.17: Example of the directed set constructed by applying an IFS with condensation. Condensation set is the circle in the center. The interior and the exterior parts are in blue and red, respectively.

In conclusion, we have described a generic algorithm calculating an approximate operation image for a fractal shape represented by an IFS or a CIFS. We use the solid domain to ensure the calculability of most standard CAD operations, and we can therefore state that whenever the initial approximation SB is properly chosen, the sequence of operator images $SOp(SB_n)$ converges to the required result $SOp(S\mathcal{A})$. However, approximating the attractor itself can be time consuming. Applying the operator directly to SB_n is thus not always the optimal way to compute the result. In the following sections we consider certain properties of the operator that allow us to optimize the $SOp(SB_n)$ construction.

4.4 Image decomposition

In the previous section we established the generic algorithm and proved convergence to the required result. In the following sections we show how this algorithm can be optimized by exploiting certain specific properties of fractals.

Approximating the attractor can be time-consuming, because the number of objects (in $\mathbb{T}^n(SB)$) grows exponentially as the number of iterations increases. First of all, each IFS transformation is applied to the initial partial solid SB. On the next iteration, we repeat the same procedure. Since $T(A \cup B) = T(A) \cup T(B) \ \ \forall A, B \in S\mathbb{X}$, the calculation of SB_n can be illustrated by a tree whose nodes are images of the initial compact subset SB obtained by applying sequences of the IFS transformations (see figure 4.2) [Eda95]. The union of compact subsets in the n-th level of the tree represents the corresponding approximation SB_n.

If SOp satisfies certain conditions, the construction of $SOp(SB_n)$ can be

optimized by exploiting the self-similarity property of our iterative system. In the following explanations we assume that the required number of iterations n is chosen according to the effective structure of the domain used, as described in section 4.3.2 (in simple cases, this method reduces to the expression 4.1).

In the algorithm, we calculate the approximation SB_n and then apply the operator SOp to this approximation. The complexity of the algorithm is thus $O\left(f_{SOp}\ (N^n)\right)$, where $f_{SOp}\ (x)$ is the complexity of the operator SOp evaluated on the union of x objects and N is the number of IFS transformations. Generally, applying SOp to the union of an exponential number of objects is expensive. The idea of this optimization is to evaluate $SOp(SB_n)$ by applying the operator to more simple objects, that is, to each leaf of the tree (see figure 4.18).

Statement. If there exists an associative operation Θ defined on $S\mathbb{Y}$ as follows:

$$SOp(A \cup B) = SOp(A)\ \Theta\ SOp(B), \tag{4.9}$$

the image of $SB_n = \mathbb{T}^n(SB)$ can be obtained by composing the images of each leaf, i.e.:

$$SOp(\bigcup_i T_i(SB)) = \underset{i}{\Theta}\ SOp(T_i(SB)).$$

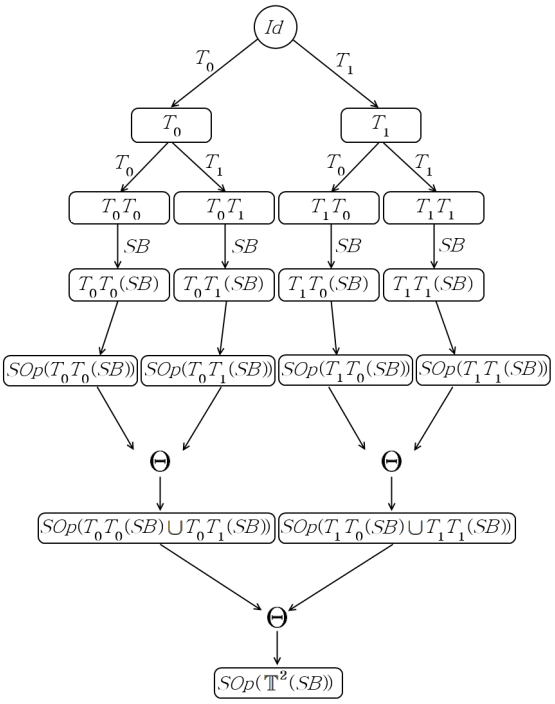


Figure 4.18: Calculation of the approximate operation result by applying the operation to each leaf of the tree. We then compose the results by means of an associative binary operation Θ.

Statement. The property of the image decomposition is not always verified. For example, calculating the volume Vol cannot be decomposed by Θ:

$$\forall \Theta : S\mathbb{Y} \times S\mathbb{Y} \to S\mathbb{Y} \;\; \exists A, B \in S\mathbb{X} : Vol(A \cup B) \neq Vol(A) \;\Theta\; Vol(B).$$

Indeed, the volume operator is translation invariant and on the right side we therefore do not have the information about the relative positions of A and B or their intersection. That is, we cannot determine the common part of these sets to identify $Vol(A \cup B)$. Nevertheless, in some cases, the volume of the IFS attractor can be computed as a sum of the volumes of its parts:

$$\forall i, j \in \Sigma \;\; Vol(T_i(\mathcal{A}) \cap T_j(\mathcal{A})) = 0 \Leftrightarrow Vol(\mathcal{A}) = \sum_{i \in \Sigma} Vol(T_i(\mathcal{A})).$$

Defining the operator Θ allows us to redefine the algorithm in many ways, depending on how we traverse the evaluation tree. This choice primarily affects the order in which we combine the results of the tree leaves. There exist two canonical ways to traverse the tree defined up to the ordering of branches: the iterative breadth-first and the recursive depth-first algorithms. Our generic algorithm can accordingly be redefined in these two ways. We describe the two algorithms in the following sections.

4.4.1 Breadth-first algorithm

In this section we describe the breadth-first algorithm for approximating the operator SOp image. We choose the required number of iterations n according to a given accuracy ε, as described in section 4.3.2. The idea of the breadth-first algorithm is to first calculate all the operator results at the n-th level of the tree, and then compose these results by means of the operation Θ.

The following function illustrates an implementation of this algorithm:

Function BFAlgo
Global variables:
SB is the initial partial solid
SOp is the operator required to evaluate
$\{T_i\}_{i \in \Sigma}$ is the set of IFS transformations
Input:
n is the number of levels left to pass
Output: an approximate operator result $SOp(\mathbb{T}^n(SB))$
Body:
// Compute set of composite IFS transformations
$ts \leftarrow \{T_{\alpha_1} \circ \ldots \circ T_{\alpha_n} \mid \alpha_i \in \Sigma\}$
// Combine all the results
return $\underset{T \in ts}{\Theta} SOp(T(SB))$
End

Note that in the breadth-first algorithm we apply the operator SOp to a transformed initial partial solid in each node of the n-th tree level, i.e. N^n times. The operation Θ is evaluated only once to compose all the operator results. The complexity of this algorithm is therefore:

$$O\left(N^n \cdot f_{SOp}\,(1) + f_\Theta(N^n)\right) = O\left(N^n + f_\Theta(N^n)\right),$$

where f_{SOp} is the complexity of the operator SOp evaluation, f_Θ is the complexity of the operation Θ as a function of the number of parameters and N is

the number of IFS transformations. Note also that $f_{SOp}\ (1)$ does not depend on the number of iterations.

4.4.2 Depth-first algorithm

In this section we describe the depth-first algorithm to approximate the image of the operator SOp. Here also, we choose the required number of iteration n according to a given accuracy ε, as described in section 4.3.2. A depth-first algorithm recursively calculates results for each branch of the tree and then composes the results by using the operation Θ.

The following recursive function illustrates an implementation of this algorithm:

Function DFAlgo
Global variables:
SB is the initial partial solid
SOp is the operator required to evaluate
$\{T_i\}_{i\in\Sigma}$ is the set of IFS transformations
Input:
T is a composite IFS transformation
n is the number of levels left to pass
Output: the approximate operator result $SOp(\mathbb{T}^n(SB))$
Body:
if $n = 0$ // End of the recursion
  return $SOp(T(SB))$
endif
return $\underset{i\in\Sigma}{\Theta}$ DFAlgo$(T \circ T_i,\ n-1)$
End

In the depth-first algorithm we also apply the operator SOp to a transformed initial partial solid in each node of the n-th tree level, i.e. N^n times. However, in this case, the operation Θ is evaluated in each recursive call, i.e. $1+N+N^2+\ldots+N^{n-1}=\frac{N^n-1}{N-1}$ times to compose N results. The algorithm complexity is therefore:

$$O\left(N^n \cdot f_{SOp}\ (1) + N^{n-1} \cdot f_\Theta(N)\right) = O\left(N^n\right),$$

where f_{SOp} is the complexity of the operator SOp evaluation, f_Θ is the complexity of the operation Θ as a function of the number of parameters and N is the number of IFS transformations.

Statement. The described algorithms have different complexities and the optimal choice depends on the operation Θ. If it is better to compose the N^n results once, we choose the breadth-first algorithm, and if it is better to combine N operator results $\frac{N^n-1}{N-1}$ times, we use the depth-first algorithm. One can prove that if evaluating the operator Θ is more complex than linear, i.e.:

$$\lim_{x\to\infty} \frac{f_\Theta(x)}{x} = \infty, \tag{4.10}$$

then the depth-first algorithm is obviously the better choice.

4.5 Adaptive tree cutting

In this section we describe how to optimise the algorithms, described in section 4.4, by collecting information about the solution in intermediate levels of the evaluation tree.

Both algorithms have an exponential complexity according to the number of iterations n, since we explore the entire tree to approximate the operator result. Due to the self-similarity of IFS attractors, in some cases we can predict the operator results in the leaves of the tree by calculating SOp in intermediate levels of the tree. If these results do not affect the overall composition by Θ, then it is not necessary to compute the corresponding branch of the tree. We call this method adaptive tree cutting optimization.

To describe this optimization more precisely, we introduce the following notation. Firstly, we define the subdomain $S_b\mathbb{Y}$ of bounded partial solids: $S_b\mathbb{Y} = \{(A, B) \in S\mathbb{Y} \mid B^{\mathsf{c}} \text{ is compact}\} \cup \{(\emptyset, \emptyset)\}$, ordered by inclusion. Recall that $S\mathbb{Y}$ is the target space of the operator SOp. The continuous predicate [EL02] $\subseteq$: $S_b\mathbb{Y} \times S\mathbb{Y} \to \{+, -\}_\perp$ is then defined by:

$$(A, B) \subseteq (C, D) = \begin{cases} + & \text{if } B \cup C = \mathbb{Y}, \\ - & \text{if } A \cap D \neq \emptyset, \\ \perp & \text{otherwise.} \end{cases}$$

Definition. The information about where the solution can be found, contained in each node of the evaluation tree, is called *solution set*. We denote the solution sets by $SOp(T(SB))$, where T is a composite IFS transformation. The solution set thus corresponds to an approximate part of $SOp(S\mathcal{A})$. All these sets are subsequently composed to approximate the resulting image $SOp(S\mathcal{A})$. Since this image is composed iteratively, to optimize we can define the approximate result as a global variable and use it in the following iterations.

Statement. Consider a solution set of some node $SOp(T(SB))$, where T is a composite IFS transformation. By composing this set with the global solution set S, we can deduce whether it is suitable, i.e. whether it changes the overall composition. More precisely, if the following conditions hold:

$$SOp(T(SB)) \sqsubseteq SOp(T \circ \mathbb{T}(SB)), \tag{4.11}$$

$$SOp(T(SB)) \subseteq S, \tag{4.12}$$

we can stop iterations for this branch of the tree without changing the resulting solution set S. Relation $\sqsubseteq$ here is the partial order of the solid domain. Condition (4.11) implies that by applying the HUTCHINSON operator we increase the precision of the solution set and that if this current solution does not improve the result (see condition (4.12)), we can then avoid unnecessary computations.

Note that condition (4.11) is always verified when the partial solids SB_n constitute a directed set, due to the continuity of SOp and T. Otherwise, this condition has to be verified additionally. Hereafter, we use the term stopping criterion to refer to condition (4.12) and assume that the initial partial solid SB is properly chosen (see section 4.3).

Composing the solution sets by Θ at each iteration, we thus restrict the set of possible solutions, or, in terms of domain theory, we approximate the resulting partial solid $SOp(S\mathcal{A})$.

The algorithms described in section 4.4 can both be optimized by constructing a global solution set S, i.e. an approximation of the solution, and by verifying the stopping criterion (4.12). The following recursive function illustrates this optimization applied to the breadth-first algorithm:

Function BFAlgoOpt
Global variables:
$SB = (\emptyset, B^c)$ is an initial partial solid such that $\mathbb{T}(B) \subseteq B$
SOp is the operator required to evaluate
S is the solution set
Input:
n is a number of levels to pass
Output: an approximate operator result $SOp(\mathbb{T}^n(SB))$
Body:
$ts \leftarrow [Id]$ // A set of composite IFS transformations
for k = 1 to n do
    $ts' \leftarrow []$ // Build a new set of composite IFS transformations
    $S \leftarrow (\emptyset, \mathbb{Y})$ // Reset S to avoid the comparison with the precedent tree level
    foreach $T \in ts$ such that $SOp(T(SB)) \nsubseteq S$ // Stopping criterion (4.12)
        $S \leftarrow S \ \Theta \ SOp(T(SB))$
        $ts'.push(TT_i), \forall i \in \Sigma$
    endfor
    $ts \leftarrow ts'$
endfor
return $\underset{T \in ts}{\Theta} SOp(T(SB))$
End

The price we must pay for this optimization is applying the operator $\frac{N^{n+1}-1}{N-1}$ times, instead of N^n, that is roughly $\frac{N}{N-1}$ times more, and applying Θ roughly 2 times more, as illustrated in figure 4.19. More precisely, the complexity of the optimized breadth-first algorithm is:

$$O\left(N^n \cdot f_{SOp}\ (1) + f_\Theta(N^n) + N^{n-1} \cdot f_\Theta(2)\right) =$$

$$O\left(N^n + f_\Theta(N^n)\right),$$

where f_{SOp} is the complexity of the operator SOp evaluation, f_Θ is the complexity of the operation Θ as a function of the number of parameters and N is the number of IFS transformations. Note that the best-case running time for this optimized algorithm is $O(1)$. This is the case in which $SOp(SB)$ returns the empty partial solid $(\emptyset, \mathbb{Y})$.

One can also, by analogy, adapt the depth-first algorithm. The following function thus shows the complexity of the optimized depth-first algorithm:

$$O\left(N^n \cdot f_{SOp}\ (1) + N^{n-1} \cdot f_\Theta(N) + N^n \cdot f_\Theta(2)\right) = O\left(N^n\right).$$

In optimistic non-trivial scenarios the depth-first algorithm shows better running-times. However, on average, the breadth-first algorithm cuts the tree much earlier than the depth-first algorithm and the breadth-first algorithm therefore has a better average running-time (see section 7). The evaluation tree of the optimized depth-first algorithm is presented in figure 4.20.

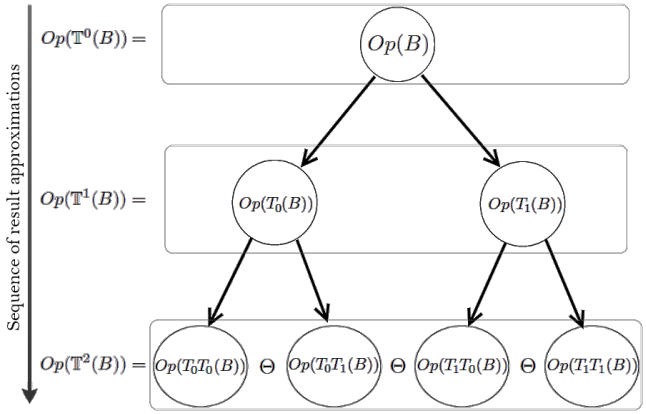


Figure 4.19: Modifying the evaluation tree allows us to optimize the generic algorithms. We apply the operator SOp on intermediate levels of the tree to prevent unnecessary calculations as early as possible.

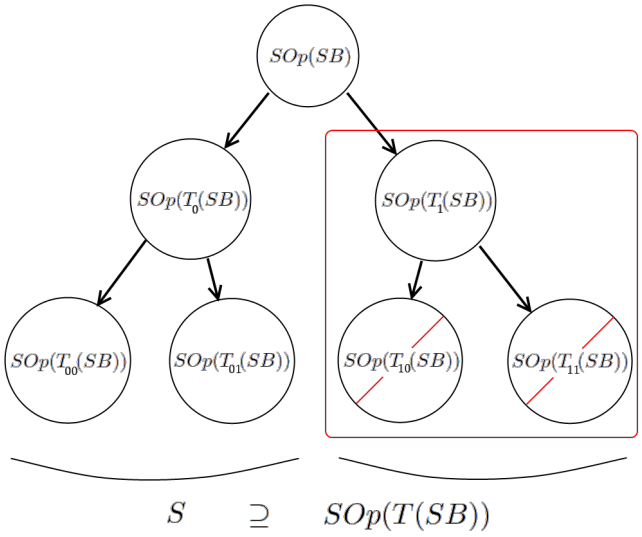


Figure 4.20: Modifying the evaluation tree that allows us to optimize the depth-first generic algorithm. We apply the operator SOp on intermediate levels of the tree to prevent unnecessary calculations as early as possible. During the calculation we construct the solution set S and verify the stopping criterion 4.12.

4.5.1 Image with an unknown interior

Let T be a composite transformation considered in an arbitrary step of the algorithm. To test the inclusion $SOp(T(SB)) \subseteq S$ we must be able to compute the interior part of these partial solids.

We have already described the constraints required for the interior part of the initial partial solid SB (see section 4.3.4). It is sometimes difficult to construct the interior of $SOp(SB)$, and accordingly the interior of all $SOp(T(SB))$.

We therefore use the empty set $\emptyset$ to approximate the interior; that is, only the exterior is used to approximate the solution. In this case, the stopping criterion (4.12) is verified only for the empty partial solid $(\emptyset, \mathbb{Y})$. To implement, we can then simply verify if $SOp(T(SB))$ is equal to $(\emptyset, \mathbb{Y})$ and avoid calculating the solution set S, as illustrated in figure 4.21. This leads to slightly different algorithms with the same complexities.

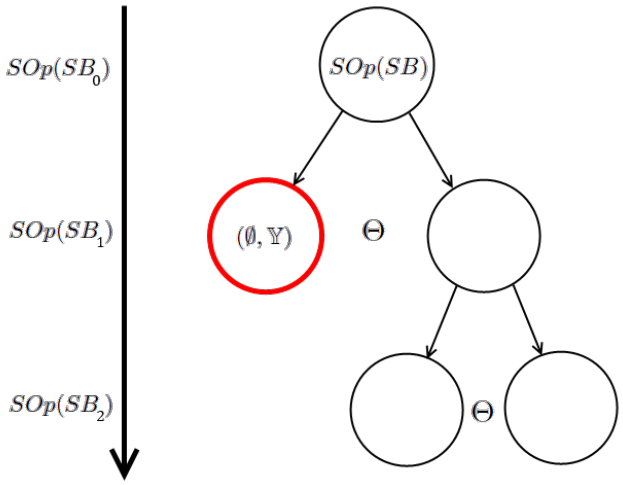


Figure 4.21: Simplification of the stopping criterion. We also apply the operator SOp on the intermediate levels of the tree, but we compare the result with the empty partial solid $(\emptyset, \mathbb{Y})$ only. Thus we avoid constructing the global solution set S.

For instance, we consider an operator determining whether a given point belongs to the attractor. Let $p \in \mathbb{X}$ be a point; the extended operator SOp is then defined as follows:

$$SOp((A, B)) = \begin{cases} + & \text{if } p \in A, \\ - & \text{if } p \in B, \\ \bot & \text{otherwise.} \end{cases}$$

In this case, the space $\mathbb{Y}$ is a set $\{+, -\}_\bot$ without a metric. The operator SOp satisfies all conditions and we can write the following:

$$\left(p \in \bigcup_i T_i(B) \right) \Leftrightarrow \underset{i}{\text{OR}} \left(p \in T_i(B) \right).$$

The composing operator Θ is therefore the boolean operator OR.

Note that, since we do not construct the interior of the initial partial solid SB, the operator SOp never returns $+$. We therefore cannot determine whether or not the point belongs to the attractor. The problem is therefore semi-decidable. However, we can bound the distance from point p to the attractor, and after n iterations, we can determine whether it is small enough, i.e.:

$$SOp(SB_n) = \bot \Rightarrow p \notin B_n^{\mathsf{c}} \Rightarrow p \in B_n \Rightarrow d_{\mathbb{X}}(p, \mathcal{A}) < \varepsilon.$$

In this example, the stopping criterion (4.12) is only verified for the element $-$. That is, we need to determine if $SOp(SB) = -$. If this is the case, we stop the algorithm and deduce that $SOp(\mathbb{T}^n(SB)) = -$, that is, the operator result is false. Otherwise, we determine $SOp(T_i(SB)), \forall i \in \Sigma$. If all results are equal to $-$ we deduce that $SOp(\mathbb{T}(SB)) = -$ and that the point does not belong to $\mathbb{T}^n(B)$. If there exists i_0 such that $SOp(T_{i_0}(SB) \neq -$, we then continue explorations by determining $SOp(T_{i_0}T_j(SB)$, $\forall j \in \Sigma$. For the images that give $-$ we stop the exploration, for the others we continue up to level n.

4.6 Conclusion

In this chapter we studied the possibility of applying an arbitrary CAD operation to fractals represented by a BCIFS (see section 3.3). We classify most standard constructing operations by adapting them to BCIFS. Thus we encountered three categories depending on the type of result. The first category corresponds to cases, in which we have a formal expression of the operation result. Applying operations from the second category is not immediate. The properties of fractals are more specific than those of conventional geometric objects (concepts of roughness or lacunarity, for example) and we have to study the underlying concepts of these operations in order to generalize or to adapt. The last category consists of operations for which we can construct an approximate result without specifying a formal representation.

In studying and identifying constraints on the operations to evaluate an approximate result, we introduced seven algorithms, which are the following:

- direct application to the approximate attractor;
- the breadth-first and the depth-first algorithms using image decomposition (see section 4.4);
- two optimized algorithms using the additional stopping criterion (see section 4.5);
- two simplified algorithms without constructing the interior part (see section 4.5.1).

These algorithms are generic, i.e. they can be applied to any CAD operation satisfying certain constraints once we implement the required interface. The first step is to verify the continuity of the operator. If it is possible to construct a continuous extension of the operator on the solid domain, we can thus construct the approximate image by using the effective structure on this domain. Moreover, this allows us to optimize the approximations. Otherwise, we cannot say with certainty that the sequence of approximate images converges to the required result.

If there exists an associative operation Θ described in detail in section 4.4, the approximate image can be obtained by composing images of each leaf, i.e. we can decompose the approximate result on separate parts. This leads to the breadth-first and depth-first algorithms.

Image decomposition allows us to subdivide the calculations in order to abandon the unnecessary parts, therefore noticeably optimizing the calculations. By using the solid domain we formalize solution set notation. Thus, we identify a

set of possible solutions which are suitable for the current approximation level. This set is therefore restricted while iterating the algorithm. The presented algorithms can thus be optimized by verifying the stopping criterion (4.12). In certain cases, this criterion can also be simplified by simply comparing the result with $(\emptyset, \mathbb{Y})$, as described in section 4.5.1.

At the moment, we cannot provide the full automation of this process. This means that to apply a CAD operator we have to perform some preliminary work. Firstly, we need to define the extended operator, which must be continuous on the solid domain. We then have to design and code the image decomposition operator Θ, which may be any algorithm satisfying the definition (4.9). Finally, we must be able to verify the stopping criterion (4.12), by calculating the inclusion. Once we implement this interface we can apply the presented generic algorithms.

In the next chapter, we define a self-similarity property of the operation and introduce a new generic algorithm to formally represent an approximate operation result, i.e. to compute a specific CIFS with the approximate image as the attractor. In special cases, when the automaton degenerates into a single state, the constructed model is a specific IFS and has a linear complexity on the number of iterations.

In chapter 6, we consider some examples and special cases to evaluate different CAD operations. Based on the operation properties, we choose the appropriate algorithms presented in this section. We then analyse results and compare the performance of our algorithms.

Chapter 5

Formal representation of the operator image

In chapter 4 we presented several algorithms which could be used to approximate an operation. Once we implement the required interface, the operation image can be computed for the required accuracy ε. In this chapter, we identify some additional properties of the operation used to formally represent the approximate result. The idea is to build a specific iterated system (CIFS or IFS) for which the operation image would be the attractor. To this end we have to formalize the self-similarity property of the constructing set.

If the operation image satisfies the self-similarity property, we can then construct a CIFS with the generalised HUTCHINSON operator generating the attractor close to the operation image. First, we define this property and then introduce a new generic algorithm to construct this CIFS. In order to keep the process as simple as possible, we describe the idea on IFS. Since CIFS can contain transitions between spaces of different dimensions, generalizing to CIFS involves additional manipulations that we discuss in section 5.1.1.

Because of the semi-decidability of the real numbers comparison, the result can only be computed approximately. The final construction algorithm thus computes a CIFS automaton for a given accuracy $\gamma > 0$. In certain cases, when the automaton degenerates into a single state, the constructed model represents an IFS and has a linear complexity on the number of iterations.

5.1 Self-similarity property

In all the algorithms described in chapter 4, we apply the extended operator SOp to a transformed initial partial solid B in each node of the n-th tree level, i.e. N^n times (see figure 4.18), or even more in the optimized versions. These calculations can be expensive if the operator SOp has a relatively high complexity or if the number of iterations n is large, sometimes these calculations can also be optimized.

Let us remind you that we approximate the operation by constructing a sequence of partial solids:

$$SOp(SB_n) = SOp((A_n, B_n^{\mathsf{c}})) \xrightarrow[n\to\infty]{} SOp((\mathcal{A}^o, \mathcal{A}^{\mathsf{c}})).$$

Here, $(B_n)_{n\in\mathbb{N}} = \mathbb{T}^n(B)$, $(A_n)_{n\in\mathbb{N}} = \mathbb{T}^n(A)$ and $\mathbb{T}$ is the HUTCHINSON operator corresponding to a given iterative system. The initial partial solid $SB = (A, B^c)$ is chosen as described in section 4.3.4.

In this section we define the self-similarity property of the images $SOp(SB_n)$. The constructing algorithm uses the image decomposition, described in section 4.4, and we also assume that the operator SOp is continuous to guarantee the convergence of the sequence $SOp(SB_n)$ to $SOp(S\mathcal{A})$ (see section 4.3.4).

The principle of the self-similarity property is that instead of applying the operator to transformed partial solids, we factor out the transformation, and thus always apply the operator to the same partial solid SB before transforming it, as illustrated in figure 5.1. However, the operator has to be modified, in this case, preserving the continuity and image decomposition, described in chapter 4.

Definition. The self-similarity property of an operator SOp image is defined for all transition operators T of the model as follows:

$$\forall T\ \exists \tilde{T} : S\mathbb{Y} \to S\mathbb{Y} \text{ such that } SOp(T(SB)) = \tilde{T}SOp^T(SB), \tag{5.1}$$

where SOp^T is the operator SOp modified using the corresponding transformation T.

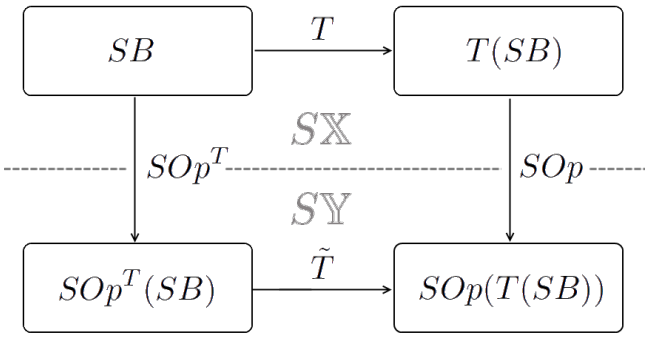


Figure 5.1: Self-similarity property of the operator SOp. The principle is that instead of applying the operator to transformed partial solids, we factor out the transformation, and apply the operator to the initial partial solid SB before transforming it. However, the operator has to be modified preserving continuity and image decomposition.

If the operator SOp is, for example, a binary operation applied to a fractal A and an object K:

$$SOp : A \mapsto A \odot K,$$

we can write the following:

$$SOp(T(SB)) = T(SB) \odot K.$$

In simple cases, when transformation T has distributivity under binary operation $\odot$, i.e.:

$$T(A \odot B) = T(A) \odot T(B),$$

and the transformation is right-invertible, by factoring it out we obtain

$$SOp(T(SB)) = T(SB) \odot K = T(SB \odot T^{-1}_{right}(K)) = TSOp^T(SB),$$

where T^{-1}_{right} is the right-inverse operator.

However, generally, transformation T does not have this distributivity property, and we therefore have to perform some additional actions to maintain the equality, that is:

$$SOp(T(SB)) = T(SB) \odot K = \tilde{T}(SB \odot T^{-1}_{right}(K)) = \tilde{T}SOp^T(SB).$$

Here T^{-1}_{right} is the right-inverse operator, transformation $\tilde{T}$ can differ from T.

If transformation T is not invertible, by factoring it out we obtain

$$SOp(T(SB)) = T(SB) \odot K = \tilde{T}(SB \odot K') = \tilde{T}SOp^T(SB),$$

where

$$T(K') = K.$$

We should thus find all points that are transformed into K under T.

5.1.1 Examples and special cases

As a first example, consider the operator intersecting an IFS $\{\mathbb{X}; T_i \mid i \in \Sigma\}$ with a line L:

$$SOp : S\mathcal{A} \mapsto S\mathcal{A} \cap L : S\mathbb{X} \to S\mathbb{X}.$$

Assume that all the IFS transformations are invertible. One can prove that for any non-empty partial solid $SB \in S\mathbb{X}$ and an invertible transformation $T : S\mathbb{X} \to S\mathbb{X}$ the following property holds:

$$T(SB) \cap L = T(SB \cap T^{-1}(L)),$$

or in terms of the operator SOp:

$$SOp(T(SB)) = T(SOp^T(SB)),$$

where SOp^T is the same intersecting operator applied to another object $T^{-1}(L)$, that is, $SOp^T(SB) = SB \cap T^{-1}(L)$. This operator thus has the same properties (continuity, image decomposition) as the initial operator SOp and the image is self-similar [HD91, Gen92].

Consider another example introduced in section 3.2.3, which corresponds to a case of non-invertible transformation. In a CIFS each state of the automaton is associated with a space. A transformation between two spaces with different dimensions is generally non-invertible. In this example we demonstrate how to deal with such cases. Figure 5.2 illustrates the attractor of a given CIFS and a line to intersect with it. The CIFS is described by an automaton with three states: $\natural$, C and S. State C corresponds to a square, and is associated with the 4D barycentric space $\mathbb{X}^C$, illustrated in figure 3.10. The result is then projected onto the 2D modelling space $\mathbb{X}^\natural = \mathbb{R}^2$. State S is used to construct the Sierpinski triangle in the 3D barycentric space $\mathbb{X}^S$, which is then projected onto $\mathbb{X}^C$ as one of the quarters.

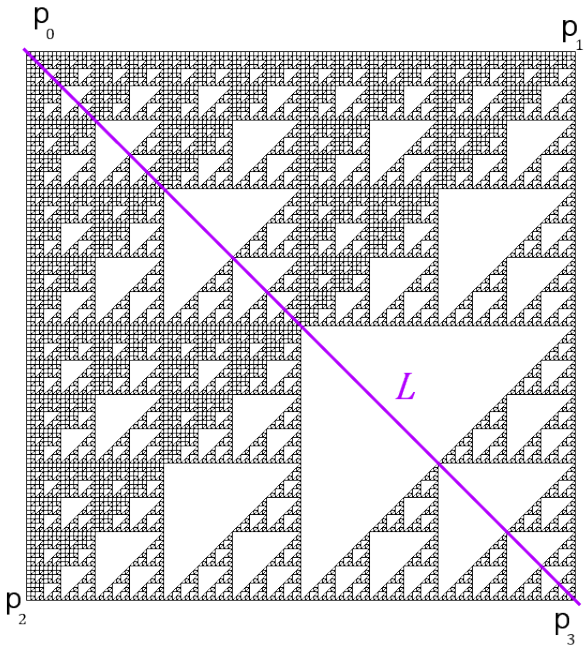


Figure 5.2: Example of an intersection between a CIFS and a line L. The system is described by an automaton with three states: $\natural$, C and S. The image illustrates attractor $\mathcal{A}^{\natural}$ in the 2D modelling space.

Let L be a line in the modelling space defined parametrically for $t \in \mathbb{R}$ as follows:

$$L = \begin{pmatrix} 0 \\ 1 \end{pmatrix} + t \begin{pmatrix} 1 \\ -1 \end{pmatrix} = \begin{pmatrix} t \\ 1-t \end{pmatrix}.$$

To simplify notation, we here use the equality sign, meaning that each point of set L has this structure for a given value of parameter $t \in \mathbb{R}$, that is, in fact:

$$L = \{p = \begin{pmatrix} t \\ 1-t \end{pmatrix} \mid t \in \mathbb{R}\}.$$

Let $SOp : S\mathbb{X}^{\natural} \to S\mathbb{X}^{\natural}$ be extended to the solid domain operator calculating the intersection with a line L.

$$SOp : S\mathcal{A} \mapsto S\mathcal{A} \cap SL,$$

where SL is a representation of the line L in the solid domain. We approximate the operation by constructing a directed set $\{SOp(SB_n) \mid n \in \mathbb{N}\}$:

$$SOp(SB_n) \xrightarrow[n\to\infty]{} SOp(S\mathcal{A}) = SOp((\emptyset, \mathcal{A}^{\mathsf{c}})),$$

where SB_n is an approximation of the attractor $\mathcal{A} = \mathcal{A}^{\natural}$ in the solid domain, such that (see section 4.3.4):

$$SB_0 \sqsubseteq SB_1 \sqsubseteq \ldots \sqsubseteq SB_n.$$

Each transition $\delta(q, i) = w$ of the automaton is associated with an operator $T_i^q : \mathbb{X}^w \to \mathbb{X}^q$. Looped subdividing operators are defined as follows:

$$\begin{array}{ll} T_0^C = H_{0.5}^{(1,0,0,0)} & T_0^S = H_{0.5}^{(1,0,0)} \\ T_1^C = H_{0.5}^{(0,1,0,0)} & T_1^S = H_{0.5}^{(0,1,0)} \\ T_2^C = H_{0.5}^{(0,0,1,0)} & T_2^S = H_{0.5}^{(0,0,1)}, \end{array}$$

where H_s^p denotes the homothety centred at point p with ratio s. These operators are invertible, and we thus have the self-similarity property (5.1):

$$\tilde{T}_i^q = T_i^q, \text{ for } i = 0, 1, 2 \text{ and } q \in C, S,$$

since the operators are continuous and therefore have distributivity under the intersection.

Let P be a projection to the 2D modelling space defined as follows:

$$P = \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix}.$$

Projection P is not invertible and for the self-similarity property we therefore have to find a set of points, denoted L', such that:

$$P(L') = L.$$

Let L' be defined parametrically in a barycentric space for $a, b, c \in \mathbb{R}$ as follows:

$$L' = \begin{pmatrix} a \\ b \\ c \\ 1 - a - b - c \end{pmatrix}.$$

We therefore have

$$P(L') = \begin{pmatrix} 1 - a - c \\ a + b \end{pmatrix} = \begin{pmatrix} t \\ 1 - t \end{pmatrix} = L,$$

which implies

$$\begin{cases} 1 - a - c & = t \\ a + b & = 1 - t. \end{cases}$$

Here we have two equations for four variables. Set L' thus defines a plane.

Let t and c be parameters, we can write the following:

$$L' = \begin{pmatrix} 1 - c - t \\ c \\ c \\ t - c \end{pmatrix}.$$

This plane L' is projected onto line L by P. This operator thus verifies the self-similarity property (5.1) and the operator SOp^P is defined as follows:

$$SOp^P : SB \mapsto SB \cap L',$$

which implies:

$$P(SOp^P(SB)) = P(SB \cap L') = P(SB) \cap L = SOp(P(SB)).$$

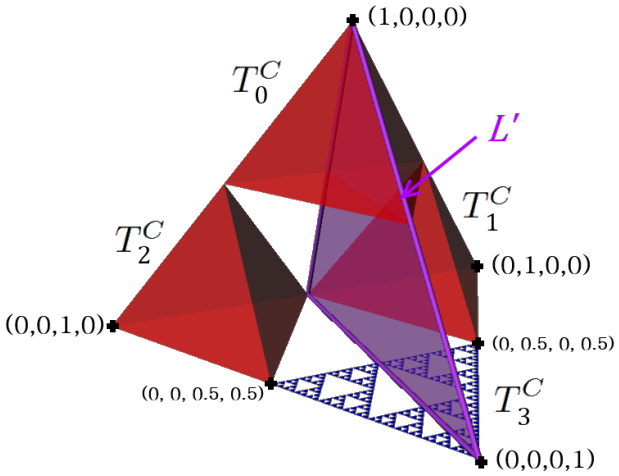


Figure 5.3: Instead of intersecting attractor $\mathcal{A}^\natural$ with line L in the 2D modelling space, now we have to intersect the attractor $\mathcal{A}^C$ with the plane L' in the 4D barycentric space. Plane L' is then projected onto L by the operator P.

Instead of intersecting attractor $\mathcal{A}^\natural$ with line L in the 2D modelling space, now we have to intersect the attractor $\mathcal{A}^C$ (an attractor associated with the state C) with the plane L' in the 4D barycentric space $\mathbb{X}^C$ as shown in figure 5.3.

Since P projects the 4D subset L' to the 2D subset L, the solution L' of the equation $P(L') = L$ can always be found. The last transformation corresponds to another case, where we have to project a 3D subset to the 4D barycentric space. The projection T_3^C of the SIERPINSKI triangle to the space $\mathbb{X}^C$ is defined as follows:

$$T_3^C = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0.5 & 0 & 0 \\ 0.5 & 0.5 & 1 \end{pmatrix}.$$

Also we have to find a set of points, denoted L'', that gives L' after transformation by T_3^C:

$$T_3^C(L'') = L'.$$

This equation generally has no solution, because we have to match the 3D subset L'' with the 4D subset L' and the system would consist of four equations for three variables. However, the transformation T_3^C places the attractor $\mathcal{A}^S$ following the control polygon $((0, 0, 0.5, 0.5), (0, 0.5, 0, 0.5)$ and $(0, 0, 0, 1))$. Thus we know that the set $T_3^C(\mathcal{A}^S)$ lies in a plane H formed by these control points. We have to find a set L'' that gives $L' \cap H$ after transforming by T_3^C:

$$T_3^C(L'') = L' \cap H.$$

This is equivalent to finding a set L'' such that $PT_3^C(L'') = L$. We will use the second equation, as it is easier to implement.

One can derive the following:

$$PT_3^C = \begin{pmatrix} 0.5 & 1 & 1 \\ 0 & 0.5 & 0 \end{pmatrix}.$$

Let L'' be defined parametrically as follows:

$$L'' = \begin{pmatrix} e \\ f \\ 1-e-f \end{pmatrix},$$

where $e, f \in \mathbb{R}$ are real variables.

We therefore have

$$PT_3^C(L'') = \begin{pmatrix} 1-0.5e \\ 0.5f \end{pmatrix} = \begin{pmatrix} t \\ 1-t \end{pmatrix} = L,$$

which implies

$$\begin{cases} 1-\frac{e}{2} = t \\ \frac{f}{2} = 1-t \end{cases}$$

Therefore

$$L'' = \begin{pmatrix} e \\ e \\ 1-2e \end{pmatrix}.$$

By construction, this line projects by T_3^C onto the intersection $L' \cap H$ and then projects onto the line L. Instead of intersecting the transformed attractor $T_3^C(\mathcal{A}^S)$ with the plane L' in the 4D barycentric space $\mathbb{X}^C$, we have now to intersect the attractor $\mathcal{A}^S$ with the line L'' in the 3D barycentric space $\mathbb{X}^S$.

The self-similarity property (5.1) is satisfied for all the transition operators and the algorithm can therefore be applied to construct a CIFS automaton to approximate the operator SOp image. In the next section we introduce this construction algorithm. The intersection between the attractor and the considered line is illustrated in figure 5.4.

5.2 Approximate iterative model

In this section we introduce a generic algorithm to formally represent the approximate operation result. For simplicity purposes, we first describe the idea on IFS, a generalization to CIFS requires additional manipulations to verify the self-similarity property (see section 5.1.1).

5.2.1 Preliminaries

Let $\{\mathbb{X}; T_i \mid i \in \Sigma\}$ be a hyperbolic IFS defined in a complete metric space $(\mathbb{X}, d_{\mathbb{X}})$ with the attractor $\mathcal{A}$. An extension of an arbitrary operator $Op : \mathcal{H}(\mathbb{X}) \to \mathbb{Y}$ to the solid domain $SOp : S\mathbb{X} \to S\mathbb{Y}$ could be any algorithm that takes a non-empty partial solid as input (see section 4.3.4). We approximate the operation by constructing a directed set $\{SOp(SB_n) \mid n \in \mathbb{N}\}$:

$$SOp(SB_n) \xrightarrow[n\to\infty]{} SOp(S\mathcal{A}) = SOp((\mathcal{A}^o, \mathcal{A}^c)),$$

where $SB_n = \mathbb{T}^n(SB)$ is an approximation of the attractor $\mathcal{A}$ in the solid domain, and $SB \in S\mathbb{X}$ is an initial partial solid chosen as described in section 4.3.4.

Assume also that the operator SOp satisfies the image decomposition (see section 4.4) and is continuous to guarantee the convergence of the sequence $SOp(SB_n)$ to $SOp(S\mathcal{A})$.

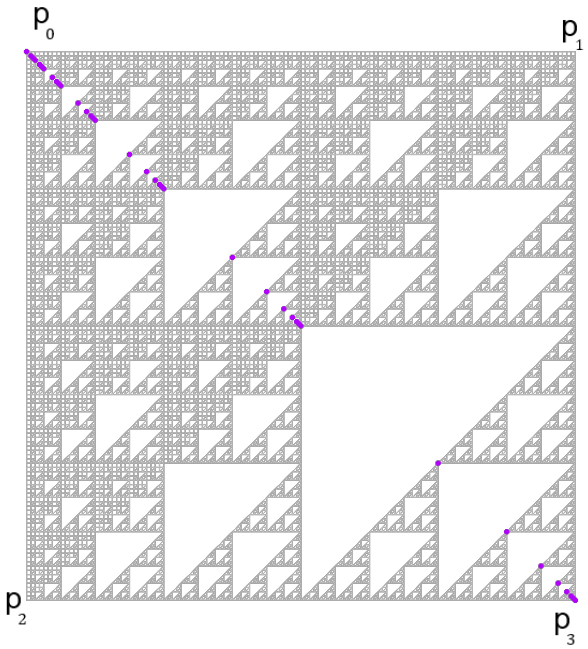


Figure 5.4: Intersection between a CIFS and a line L. The output CIFS automaton consists of three states, we denote these states by the corresponding calculating sets $\mathcal{A}^{\natural} \cap L$, $\mathcal{A}^{C} \cap L'$ and $\mathcal{A}^{S} \cap L''$. The intersection between the line and the SIERPINSKI triangle is thus computed in the state $\mathcal{A}^{S} \cap L''$, it is then placed in the 4D barycentric space according to the control points, and finally projected onto the modelling space.

The self-similarity property of the operator SOp image is defined for each IFS transformation T_i, $i \in \Sigma$ as follows:

$$\exists \tilde{T}_i : S\mathbb{Y} \to S\mathbb{Y} \text{ such that } SOp(T_i(SB)) = \tilde{T}_i SOp^{T_i}(SB). \tag{5.2}$$

Here SOp^{T_i} is the operator SOp modified somehow using the corresponding transformation T_i (see section 5.1).

5.2.2 Construction algorithm

In this section we describe an algorithm to construct a specific CIFS with an approximate operator image as the attractor. The attractors of this CIFS are mutually recursively defined for each state $q \in Q$ by the generalised HUTCHINSON operator:

$$S\mathcal{A}^{q} = \bigoplus_{i \in \Sigma^{q}} \tilde{T}_i^{q}(S\mathcal{A}^{\delta(q,i)}), \tag{5.3}$$

We describe in detail the steps for constructing the CIFS automaton. In general, the algorithm produces an automaton with an infinite number of states, and we therefore have to bound the approximating accuracy to be able to compute the result in finite time.

The key idea of the algorithm is to apply a classic evaluation and verify the similarities between constructed sub-attractors, as illustrated in figure 5.5. Thus

we start with the initial state and then loop through all the IFS transformations checking which of the alternatives the corresponding sub-attractor hits:

- the sub-attractor is empty. We do not add any special state to construct the empty set, so we continue the iteration.
- the sub-attractor is similar to one that was already parsed, denotes s'. In this case, we close the transition to the state s'.
- otherwise, we should continue the iteration recursively.

The algorithm thus finds the similar parts in the output set and constructs a CIFS automaton with the corresponding attractor.

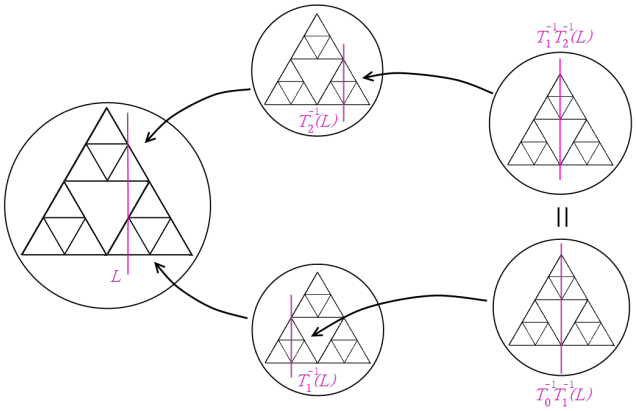


Figure 5.5: The key idea of the algorithm is to apply a classic evaluation and verify the similarities between constructed sub-attractors. If the sub-attractor is similar to one that was already parsed, we process these sub-attractors as the same set and unite the corresponding states.

If the operator SOp verifies the self-similarity property (see equality 5.2) we can construct a specific CIFS with the attractor $SOp(S\mathcal{A})$ as follows. We start with the initial state associated to the attractor $SOp(S\mathcal{A})$ construction. We then add the transitions T_i for all $i \in \Sigma$ from the initial state to the new states associated to the corresponding attractors $SOp^{T_i}(S\mathcal{A})$ and continue this process recursively.

If a partial solid $SOp^{T_i}(SB)$ is empty, that is equal to $(\emptyset, \mathbb{Y})$, then it is not necessary to add a new state, because such state is associated with the empty attractor.

Consider the k-th iteration of this recursion. Let consider two composite transformations given by two sets of indices:

$$\alpha_i \in \Sigma \text{ for } i = 1, \ldots, k$$
$$\beta_j \in \Sigma \text{ for } j = 1, \ldots, l.$$

Here k and l, $l \leqslant k$ represent the lengths of these composites. α_i and β_j are the IFS transformations indices. We are going to add a new state corresponding

to the composition $T_{\alpha_1} \circ \ldots \circ T_{\alpha_k}$. If there exists a state q associated to the composition $T_{\beta_1} \circ \ldots \circ T_{\beta_l}$ for which the following equality holds:

$$SOp^{T_{\alpha_1} \circ \ldots \circ T_{\alpha_k}}(SB) = SOp^{T_{\beta_1} \circ \ldots \circ T_{\beta_l}}(SB),$$

then the corresponding states are associated with the same sub-attractors. It is thus not necessary to add a new state, and we simply close the transition to the state q.

Figure 5.6 illustrates an example of a CIFS automaton produced by the presented algorithm. In general, the operator image $SOp(S\mathcal{A})$ is an aperiodic set, and the algorithm thus produces an automaton with an infinite number of states. Sub-attractors equality is semi-decidable and is related to the issue of comparing real numbers, where we can only say if the numbers are close enough, but we cannot determine if the numbers are equal.

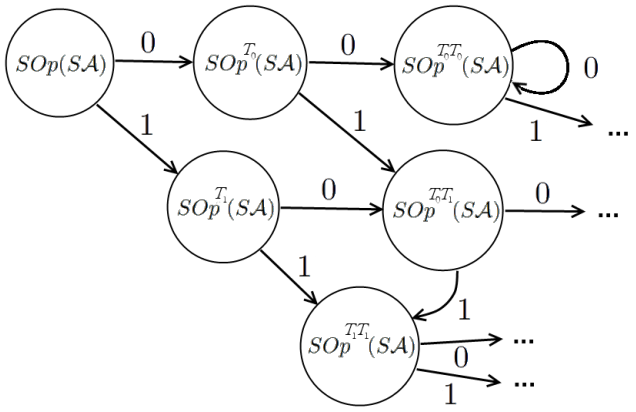


Figure 5.6: CIFS automaton constructed by the presented algorithm. Each node corresponds to a state of the automaton with the associated CIFS attractor. Attractors are defined by the generalised HUTCHINSON operator. Directed edges correspond to the transitions and they are marked by the associated transformations. The initial state of the automaton is associated with the attractor $SOp(S\mathcal{A})$.

By setting the accuracy at $\gamma > 0$ we instead verify if the sub-attractors are close enough. This allows us to determine whether the sub-attractors are relatively similar and, moreover, to bound the required number of iterations. Thus we verify the following inequality:

$$d_{S\mathbb{Y}}\left(SOp^{T_{\alpha_1} \circ \ldots \circ T_{\alpha_k}}(SB), SOp^{T_{\beta_1} \circ \ldots \circ T_{\beta_l}}(SB)\right) < \gamma,$$

To simplify the notation, we use a relation $\underset{\gamma}{\simeq}$ to denote this inequality, i.e.:

$$SOp^{T_{\alpha_1} \circ \ldots \circ T_{\alpha_k}}(SB) \underset{\gamma}{\simeq} SOp^{T_{\beta_1} \circ \ldots \circ T_{\beta_l}}(SB). \tag{5.4}$$

Note that in the evaluation algorithm, the constructing sub-attractors will be placed later on the output attractor by a composite of contractive transformations (see figure 5.5). The accuracy for this inequality, denoted ϵ, could

be greater than the initial γ. More precisely, the accuracy should satisfy the following inequality:

$$\epsilon \leqslant \gamma \cdot \min\{s(T_{\alpha_1} \circ \ldots \circ T_{\alpha_k}), s(T_{\beta_1} \circ \ldots \circ T_{\beta_l})\}. \tag{5.5}$$

The inequality (5.4) holds when the sub-attractors are relatively close to each other. In this case, it is not necessary to overly complicate the automaton. By definition, the attractor $\mathcal{A}$ is a compact set, it therefore has a finite covering by sub-attractors which are not close to each other. Our algorithm thus always stops.

Consider the previous example of intersecting with a line $L \in S\mathbb{X}$:

$$SOp : S\mathcal{A} \mapsto S\mathcal{A} \cap L : S\mathbb{X} \to S\mathbb{X}.$$

As described in section 5.1, for an invertible transformation $T : S\mathbb{X} \to S\mathbb{X}$ with a distributivity under the boolean intersection, i.e.

$$T(SB) \cap L = T(SB \cap T^{-1}(L)),$$

the self-similarity property (5.2) is satisfied, as we can rewrite this in terms of the operator SOp:

$$SOp(T(SB)) = T(SOp^T(SB)),$$

where SOp^T is the operator calculating the intersection with a line $T^{-1}(L)$, that is:

$$SOp^T(SB) = SB \cap T^{-1}(L).$$

This operator has the same properties (continuity, image decomposition, self-similarity) as the initial operator SOp.

To determine similar sub-attractors we verify the following inequality:

$$T_{\alpha_k}^{-1} \circ \ldots \circ T_{\alpha_1}^{-1}(L) \underset{\epsilon}{\simeq} T_{\beta_l}^{-1} \circ \ldots \circ T_{\beta_1}^{-1}(L), \tag{5.6}$$

where the tolerance ϵ can be chosen by the inequality (5.5). It implies the inequality (5.4) and does not contain applications of SOp, here we only verify whether two transformed lines are close enough.

The constructing algorithm thus goes through the IFS evaluation tree and checks the emptiness of sub-attractors and the inequality (5.4). In the first case, we stop the iterations, in the second one we close the transition to the corresponding state. Figure 5.6 illustrates an example of a CIFS automaton produced by this algorithm. The following function illustrates the algorithm implementation:

Function CIFSAlgo
Input:
SOp is the operator required to evaluate
γ is the calculation tolerance
Output: a CIFS with the attractor close to $SOp(S\mathcal{A})$
Body:
$Q \leftarrow [(Id, \natural)]$ // an output set of CIFS states and composite transformations
$F \leftarrow []$ // an output set of CIFS transitions
$S \leftarrow [(Id, \natural)]$ // a temporary list of states and composite transformations
while (S is not empty)
  $S' \leftarrow []$
  for $\forall (t, s) \in S$

```
        for ∀T_i  i ∈ Σ
            t' ← t ∘ T_i
            if SOp^{t'}(SB) = (∅, 𝕐)
                go to next iteration
            endif
            // Find similar sub-attractors
            if ∃s' : (tt, s') ∈ Q and SOp^{t'} ≃_ε SOp^{tt}, where ε = γ · min{s(t'), s(tt)}
                F.push(a new transition s ⟶_{T_i} s')
            else
                s' ← a new state
                Q.push((t', s'))
                S'.push((t', s'))
                F.push(a new transition s ⟶_{T_i} s')
            endif
        endfor
    endfor
    S ← S'
endwhile
return (Q[1], F)
End
```

Here Q is a list of pairs (t, s), where s is a CIFS state and t is the corresponding composite transformation. In the return statement we denote by $Q[1]$ a set of states in Q, that is:

$$Q[1] = \{s \mid \exists (t, s) \in Q\}.$$

The algorithm runs as follows. We start with an initial state $\natural$. The corresponding composite transformation is obviously the identity map. We then loop through all the IFS transformations T_i for $i \in \Sigma$, and check which of the alternatives the corresponding sub-attractor hits:

- the sub-attractor is empty. We do not add any special state to construct the empty set, so we continue the iteration.
- the sub-attractor is similar to one that was already parsed, denoted s'. In this case, we close the transition at the state s'.
- otherwise, we should continue the iteration recursively.

The algorithm returns a set of states and transitions composing a CIFS automaton. By construction, all the sub-attractors are close enough to corresponding parts of $SOp(S\mathcal{A})$.

Statement. For the attractor $S\mathcal{B}$ of the output CIFS the following inequality holds:

$$S\mathcal{B} \underset{\gamma}{\simeq} SOp(S\mathcal{A}) \Leftrightarrow d_{S\mathbb{Y}}(S\mathcal{B}, SOp(S\mathcal{A})) < \gamma.$$

As usual, we can approximate the CIFS attractor $S\mathcal{B}$ by an increasing chain of partial solids.

Let $\{\mathbb{X}; T_i \mid i \in \Sigma\}$ be a hyperbolic IFS defined in a complete metric space $(\mathbb{X}, d_{\mathbb{X}})$ with the attractor $\mathcal{A}$. Let $SOp : S\mathbb{X} \to S\mathbb{Y}$ be an extended operation that satisfied all the required conditions to apply the presented algorithm, and $\gamma > 0$ be a calculation accuracy. The algorithm returns a CIFS automaton

(Q, F), where Q is a set of states and F is a set of CIFS transitions. Denote the attractor of this system by $S\mathcal{B}$. By construction, one can state the following:

$$S\mathcal{B} \underset{\gamma}{\simeq} SOp(S\mathcal{A}).$$

Statement. The number of states $|Q|$ is equal to the number of iterations, for which we did not find any similar (for the accuracy γ) sub-attractor or for which the sub-attractor was empty. In other words, it is equal to a number of sub-attractors that make up the attractor, which are not similar to each other. Since the attractor is compact and the distances between sub-attractors are greater than γ, this number of states $|Q|$ is finite.

The maximal number of states can roughly be calculated as follows. The idea is that we cover the attractor $S\mathcal{B}$ by sub-attractors that are not similar to each other. the similarity is defined with respect to the HAUSDORFF distance between the sub-attractors which cannot thus be less than γ. Assuming that all sub-attractors are points, we cover the attractor $\mathcal{B}$ by balls of radius γ. In the evaluation algorithm we can use a bounding ball as the initial compact subset. The radius r of this ball can be mutually recursively determined (see appendix A). By applying the evaluation algorithm we obtain a coverage of the attractor by balls of radius at most γ. The maximal number of iterations k can be computed as follows:

$$k = \left\lceil \frac{\log(\gamma/r)}{\log(s)} \right\rceil,$$

where s is the maximal contraction coefficient of the IFS transformations. In the worst case, we thus need N^k balls to cover the entire attractor $\mathcal{B}$. This means that the maximal number of iterations required to obtain a resulting CIFS for the accuracy γ, is equal to N^{k+1}.

$$|Q| \leqslant |F| \leqslant N^{k+1},$$

where N is the number of the IFS transformations.

In the worst case, the number of iterations exponentially depends on k, and k logarithmically depends on the accuracy γ, the algorithm complexity is thus:

$$O(N^k) = O\left(\left(\frac{\gamma}{r}\right)^{1/\log_N(s)}\right).$$

5.3 Exact iterative model for solution

In the previous section we introduced a generic algorithm to formally represent the operation result. In this section we discuss how to construct an exact iterative model of the CAD operation image, that is the CIFS with the attractor $SOp(S\mathcal{A})$, defined by the generalised HUTCHINSON operator. Recall that the attractors of this CIFS are mutually recursively defined for each state $q \in Q$ by the generalised HUTCHINSON operator:

$$S\mathcal{A}^q = \bigoplus_{i \in \Sigma^q} \tilde{T}_i^q(S\mathcal{A}^{\delta(q,i)}), \tag{5.7}$$

In fact, the presented algorithm expresses a finite list of well-defined instructions for formally representing the operation image. The only instruction

that cannot be calculated exactly is the one used to verify sub-attractors self-similarity (5.4), because it involves real numbers comparisons, which are semi-decidable. However, all these instructions can be carried out manually, that is if we can prove that two sub-attractors are self-similar, then the closed transition represents a part of the automaton with the exact solution as the attractor. The problem is that, in general, there is an infinite number of such verifications. Nevertheless, there exist cases where the solution $SOp(S\mathcal{A})$ itself can be represented by an automaton with a finite number of states. In these cases, performing the algorithm instructions manually allows us to determine an exact model of the solution.

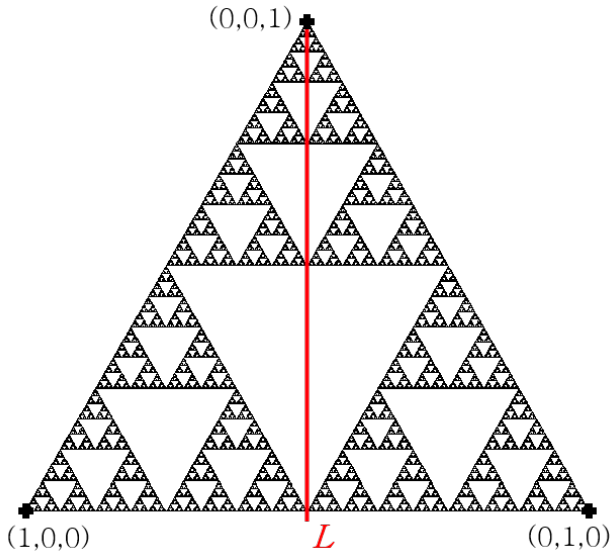


Figure 5.7: Intersecting the SIERPINSKI triangle with a line L in the 3D barycentric space.

Consider an example of intersecting the SIERPINSKI triangle $\mathcal{A}$ with a line L in the 3D barycentric space $\mathbb{X}$, illustrated in figure 5.7. We define an IFS with the three contractive transformations T_0, T_1 and T_2, defined by homotheties centered at the unit vectors with the ratios $\frac{1}{2}$ as follows:

$$T_0 = H_{\frac{1}{2}}^{(1,0,0)}$$
$$T_1 = H_{\frac{1}{2}}^{(0,1,0)}$$
$$T_2 = H_{\frac{1}{2}}^{(0,0,1)}$$

Let the line L be defined as follows:

$$L = \begin{pmatrix} t \\ t \\ 1-2t \end{pmatrix},$$

where $t \in \mathbb{R}$ is a real variable. Let SL be a representation of the line L in the solid domain $S\mathbb{X}$. Let $SB \in S\mathbb{X}$ be an initial partial solid, which represents a

bounding ball centered in the point

$$c = \begin{pmatrix} 1/3 \\ 1/3 \\ 1/3 \end{pmatrix},$$

with radius $\sqrt{2/3}$ and with an empty interior.

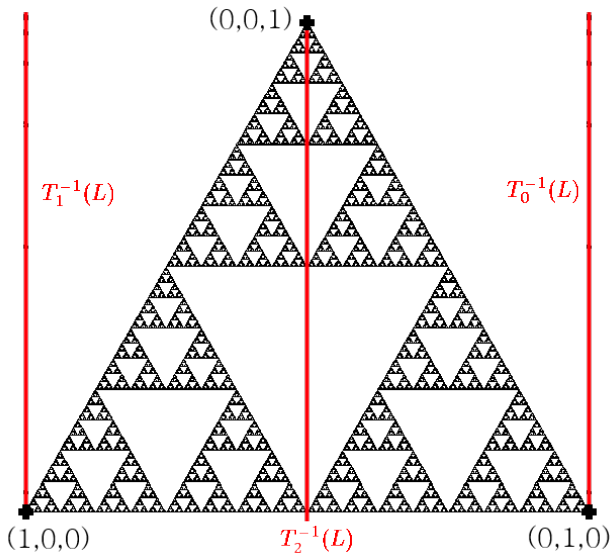


Figure 5.8: Intersecting the SIERPINSKI triangle with the lines $T_i^{-1}(L)$ for $i \in \Sigma$.

In order to verify self-similarity on the first iteration (see figure 5.8) we must compare the line SL with $T_i^{-1}(SL)$ for $i \in \Sigma$. To avoid overly complicated explanations, we present here the calculations for the line L, instead of SL. For the first IFS transformation T_0 we thus have the following:

$$T_0^{-1}(L) = \begin{pmatrix} 1 & -1 & -1 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix} \begin{pmatrix} t \\ t \\ 1-2t \end{pmatrix} = \begin{pmatrix} 2t-1 \\ 2t \\ 2-4t \end{pmatrix} = \begin{pmatrix} x \\ x \\ 1-2x \end{pmatrix},$$

where $x \in \mathbb{R}$. We therefore have the system of equations:

$$\begin{cases} 2t - 1 = x \\ 2t = x \\ 2 - 4t = 1 - 2x \end{cases},$$

which obviously has no solution. This means that the lines are not the same, and neither are the sub-attractors. For the second IFS transformation T_1 we have the same, by symmetry.

Finally, for the last IFS transformation T_2 we can write the following:

$$T_2^{-1}(L) = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ -1 & -1 & 1 \end{pmatrix} \begin{pmatrix} t \\ t \\ 1-2t \end{pmatrix} = \begin{pmatrix} 2t \\ 2t \\ 1-4t \end{pmatrix} = \begin{pmatrix} x \\ x \\ 1-2x \end{pmatrix},$$

where $x \in \mathbb{R}$. We therefore have the following system of equations:

$$\begin{cases} 2t = x \\ 2t = x \\ 1-4t = 1-2x \end{cases},$$

which has the solution $x = 2t$. This means that the line $T_2^{-1}(L)$ is the same as L, and the corresponding sub-attractors are also the same. We can thus close the current transition to the initial state.

On the next iteration, we compare the line $T_0^{-1}(L)$ with $T_i^{-1}T_0^{-1}(L)$ for $i \in \Sigma$. For $i = 0$ we have:

$$T_0^{-2}(L) = \begin{pmatrix} 1 & -3 & -3 \\ 0 & 4 & 0 \\ 0 & 0 & 4 \end{pmatrix} \begin{pmatrix} t \\ t \\ 1-2t \end{pmatrix} = \begin{pmatrix} 4t-3 \\ 4t \\ 4-8t \end{pmatrix}.$$

This line does not intersect with the ball SB as well as the line $T_2^{-1}T_0^{-1}(L)$.

For $i = 1$ we can write the following:

$$T_1^{-1}T_0^{-1}(L) = \begin{pmatrix} 2 & -2 & -2 \\ -1 & 3 & -1 \\ 0 & 0 & 4 \end{pmatrix} \begin{pmatrix} t \\ t \\ 1-2t \end{pmatrix} = \begin{pmatrix} 4t-2 \\ 4t-1 \\ 4-8t \end{pmatrix}.$$

Comparing this line with the line $T_0^{-1}(L)$ we obtain the following system of equations:

$$\begin{cases} 4t-2 = 2x-1 \\ 4t-1 = 2x \\ 4-8t = 2-4x \end{cases},$$

where $x \in \mathbb{R}$. This system has a solution $x = 2t - \frac{1}{2}$, which implies that the line $T_1^{-1}T_0^{-1}(L)$ equals to the line $T_0^{-1}(L)$, and the corresponding sub-attractors are therefore the same. We can thus close the current transition to the corresponding state. By analogy, we can deduce that $T_1^{-1}(L) = T_0^{-1}T_1^{-1}(L)$.

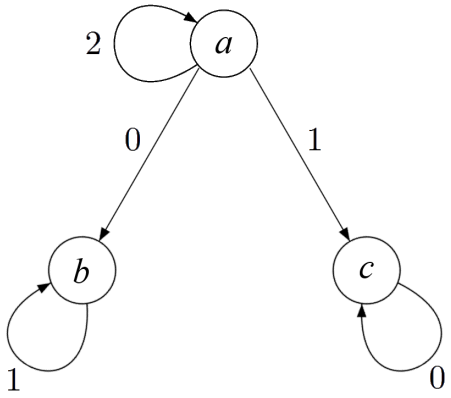


Figure 5.9: Automaton of the CIFS considered in the example. The attractor of this CIFS is exactly the set $SL \cap S\mathcal{A}$.

Since all other lines do not intersect with the ball SB the algorithm stops and we thus obtain an exact CIFS with the attractor $SOp(S\mathcal{A}) = S\mathcal{A} \cap SL$. The

automaton is illustrated in figure 5.9. In special cases, when the automaton degenerates into a single state, the constructed model represents an IFS and has a linear complexity on the number of iterations. We discuss conditions for constructing an IFS in the following section 5.3.1.

Figure 5.10 illustrates another example of intersection between the SIERPINSKI triangle and different lines.

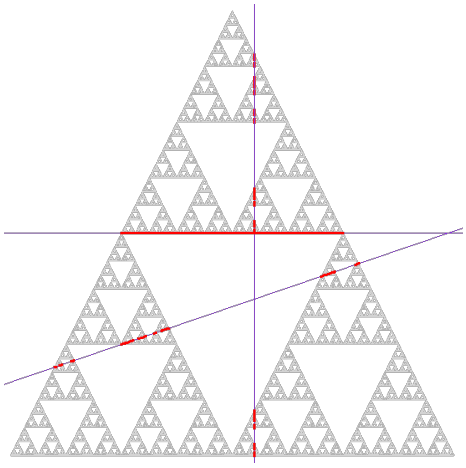


Figure 5.10: Example of intersection between the SIERPINSKI triangle and different lines.

5.3.1 Iterated function system

In section 5.2 we described the algorithm constructing a specific CIFS with the approximate operator image as the attractor. To determine similar sub-attractors we verify the inequality (5.4). In section 5.3 we showed that if the operation image is self-similar, it is possible to manually construct the exact CIFS of the image. In certain cases, when the inequality (5.4) holds for all the transformations, the output CIFS degenerates to an IFS. In this section, we identify the condition for constructing iterative model to be an IFS.

To avoid overly complicated explanations, we describe the idea in IFS, a generalization to CIFS is thus immediate with minor additional manipulations (see section 5.1.1).

Given a hyperbolic IFS $\{\mathbb{X}; T_i \mid i \in \Sigma\}$ defined in a complete metric space $(\mathbb{X}, d_{\mathbb{X}})$ by a finite set of contractive transformations T_i. Let $SB = (A, B^{\mathsf{c}}) \in S\mathbb{X}$ be an initial partial solid such that $\mathbb{T}(B) \subseteq B$ and $A \subseteq \mathcal{A}^o$ (see section 4.3.3). Suppose that the extended operator SOp is continuous and verifies the self-similarity property (5.2), i.e. $\forall i \in \Sigma$:

$$\exists \tilde{T}_i : S\mathbb{Y} \to S\mathbb{Y} \text{ such that } SOp(T_i(SB)) = \tilde{T}_i SOp^{T_i}(SB).$$

Statement. If all the modified operators are equal, i.e.:

$$SOp^{T_i}(SB) = SOp^{T_j}(SB) \text{ for } \forall i, j \in \Sigma, \tag{5.8}$$

then the presented algorithm (see section 5.2) produce an IFS with the attractor $SOp(S\mathcal{A})$ as shown in figure 5.11. The produced IFS has the same set of transformations as the initial one and it is possible to redefine a generalised HUTCHINSON operator $\tilde{\mathbb{T}} : S\mathbb{Y} \to S\mathbb{Y}$, by composing the transformations $\tilde{T}_i$:

$$\tilde{\mathbb{T}}(SY) = \underset{i\in\Sigma}{\Theta} \tilde{T}_i(SY),$$

where $SY \in S\mathbb{Y}$. Θ here is the decomposition operator chosen as described in section 4.4.

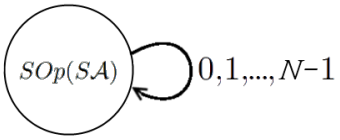


Figure 5.11: Graph for operators satisfying the condition (5.8). The automaton is degenerated to a single state associated with the attractor $SOp(S\mathcal{A})$. All the transitions are closed to this state. The result is an IFS.

The sequence defined by applying this operator thus converges to the operator image $SOp(S\mathcal{A})$. Consider a sequence of partial solids $\tilde{\mathbb{T}}^n(SOp(SB))$. For the first iteration ($n = 1$) we have:

$$\tilde{\mathbb{T}}(SOp(SB)) = \underset{i\in\Sigma}{\Theta} \tilde{T}_i(SOp(SB)) = \underset{i\in\Sigma}{\Theta} SOp(T_i(SB)) = SOp(\mathbb{T}(SB))$$

and on the n-th iteration:

$$\tilde{\mathbb{T}}^n(SOp(SB)) = SOp(\mathbb{T}^n(SB)) = SOp(SB_n) \xrightarrow[n\to\infty]{} SOp(S\mathcal{A}).$$

To approximate the operator image $SOp(S\mathcal{A})$ we thus apply the generalised HUTCHINSON operator n times, where the number of iterations n is chosen according to the effective structure of the used domain, as described in section 4.3.2 (in simple cases, by expression 4.1).

To verify the condition (5.8) for the constructing iterative model to be an IFS one must perform a theoretical analysis of the operation to verify the self-similarity property (5.2). Once the transformations $\tilde{T}_i$ are determined and the condition (5.8) holds, we can define the generalised HUTCHINSON operator and thus construct an IFS with the attractor $SOp(S\mathcal{A})$. Note that we thus obtain a precise formal model of the operation image.

In this case, the complexity of constructing $SOp(S\mathcal{A})$ equals to the complexity of evaluating the IFS, that is it linearly depends on the number of iterations n:

$$O\left(n \cdot f_\Theta(N)\right),$$

where f_Θ is the complexity of the operation Θ as a function of parameters count and N is the number of IFS transformations.

For instance, consider the convex hull operator $Op = conv$ and an extended operator SOp defined by:

$$SOp : (A, B) \mapsto (conv(A), (conv(B^{\mathsf{c}}))^{\mathsf{c}}).$$

There exists an associative binary operation Θ merging two convex hulls, i.e.:

$$(A, B)\ \Theta\ (C, D) = (merge(A, B), merge(C^{\mathsf{c}}, D^{\mathsf{c}})^{\mathsf{c}}).$$

Suppose that the IFS transformations are affine, we can therefore write the following equality:

$$conv(T_i(\mathcal{A})) = T_i(conv^{T_i}(\mathcal{A})) = T_i(conv(S\mathcal{A}))\ \ \forall i \in \Sigma,$$

the self-similarity property (5.2) of the operator is therefore verified.

Moreover, the equality of all sub-attractors holds:

$$SOp^{T_i}(S\mathcal{A}) = SOp(S\mathcal{A})\ \ \forall i \in \Sigma.$$

All the conditions for constructing the IFS with the attractor $SOp(S\mathcal{A})$ are satisfied. Let $SB \in S\mathbb{X}$ be an initial non-empty partial solid. It is thus possible to define the generalised HUTCHINSON operator:

$$\tilde{\mathbb{T}}(SB) = \underset{i\in\Sigma}{merge}\ T_i(SB).$$

And the operator image can be determined by constructing the corresponding attractor:

$$SOp(S\mathcal{A}) = \tilde{\mathbb{T}}(SOp(S\mathcal{A})) = \lim_{n\to\infty}(\tilde{\mathbb{T}}^n(SB)).$$

Since merging N convex hulls requires $O(hN^2)$, where h is the number of convex hull vertices and $N = |\Sigma|$ is the number of IFS transformations, the total time complexity for constructing the convex hull is $O(nhN^2)$. Note that it linearly depends on the number of iterations n.

Constructing an iterative function system significantly reduces the number of operations performed to compute the operator image, and this therefore provides a qualitative gain in execution time. This is the fundamental advantage of this method, since there exist many tasks in which it is necessary to calculate the operator result in real time. Another important advantage is that this method does not depend on computational accuracy. Indeed, the operation image is represented formally.

5.4 Conclusion

In this chapter we introduced an algorithm constructing a CIFS with the generalised HUTCHINSON operator (5.7), whose attractor is close enough to the operation result with respect to the HAUSDORFF metric. The output automaton consists of states for sub-attractors of the operation image and transitions to embed these sub-attractors together. Each transition is associated with one of the initial transformations, the algorithm does thus not produce any new transformation.

On each iteration we check, which of the alternatives the corresponding sub-attractor hits:

- the sub-attractor is empty.
- the sub-attractor is similar to one that was already parsed.

- otherwise, we should continue the iteration recursively.

In the case of the empty sub-attractor, we do not add any special state to construct the empty set, so we continue the iteration. To preserve the self-similarity of the output set we verify the similarity of sub-attractors that are already constructed. Thus we defined a condition (5.1) to determine the similar sub-attractors. The self-similarity property is defined for every transition operator in the input model.

Because of the semi-decidability of real number comparisons, we cannot verify this equation automatically. In order to construct an exact model of the required solution, if it exists, we must prove all the necessary similarities manually.

To evaluate the algorithm in finite time one can set the computation accuracy γ. Thus we identified the self-similarity property in terms of approximate calculations and proved that the algorithm requires a finite number of iterations in this case. For a given accuracy γ the algorithm constructs a CIFS with the attractor, defined by the generalised HUTCHINSON operator, that is close enough to the operation image. Using the evaluation algorithm, described in section 3, we can approximate the attractor by the union of the transformed primitives. However, in the case of the set calculation tolerance γ, the distance between the attractor and the operation image cannot be less that γ.

In some cases, as demonstrated in section 5.3, the output iterative model degenerates to an IFS. We specify a condition when this is the case, however to verify this condition one must perform a theoretical analysis of the operation. Once the condition (5.8) holds, we can thus construct an IFS with the operation image as attractor. Note that we thus obtain an exact formal model of the operation image. Evaluating the operation is thus reduced to evaluating this IFS which has a linear complexity on the number of iterations.

Chapter 6

Examples and special cases

In this chapter we present various examples and special cases. We consider several examples of CAD operators applied to fractal shapes. We use the algorithms presented in this thesis based on operator properties. After discussing the examples we provide experimental results by comparing the running-time of the approximation algorithms and we then summarize all the results.

For purposes of simplicity, we describe most examples for IFS, a generalization to the CIFS is thus immediate (see section 5.1.1).

6.1 Prerequisites

In this section we summarize the results already presented in chapters 4 and 5. More precisely, we describe methods which need to be implemented manually in order to apply the presented generic algorithms.

Given a CIFS defined by an automaton (Σ, Q, δ), where Σ is an alphabet, Q is a set of states and δ is a transition function $\delta : Q \times \Sigma \to Q$. An initial partial solid $SB^q \in S\mathbb{X}^q$ is associated to each state $q \in Q$. Thus we have to implement a function *GetInitial* : $q \mapsto SB^q$ that gives the initial partial solid for a given state $q \in Q$.

After that, the operator $SOp : S\mathbb{X} \to S\mathbb{Y}$ has to be defined. For this we should decide how the solution will be represented, that is to say that we define the nature of solids in $S\mathbb{Y}$. In the presented algorithms, the following methods are applied to partial solids in $S\mathbb{Y}$:

- The inclusion predicate $\subseteq$;
- GetAccuracy - returns an accuracy value for a given partial solid $(C, D) \in S\mathbb{Y}$, i.e. $d_{\mathbb{Y}}(C, D^{\mathsf{c}})$.

The first method is required to optimize calculations, and the second is used to stop the algorithm when the required accuracy is achieved. In cases where the solution interior is empty we use the simplified stopping criterion (see section 4.5.1) and do not use the *GetAccuracy* method. Note that we do not need to be able to represent any partial solid in $S\mathbb{Y}$. We only deal with the transformed initial bounding balls SB^q and their compositions by Θ. For example, when Θ is the union operator, the partial solid can be represented as a list of

pairs (T, SB^q), T is a composite CIFS transformation and SB^q is the initial partial solid for a given state $q \in Q$. After defining these methods the operator *SOp* itself should be implemented.

If the operator image can be decomposed, as described in section 4.4, the corresponding operator $\Theta : S\mathbb{Y} \times S\mathbb{Y} \to S\mathbb{Y}$ has to be implemented as well. In the breadth-first algorithm this operator could also compose a whole list of intermediate results, i.e. $\Theta : S\mathbb{Y}^k \to S\mathbb{Y}$, where k is the number of input results.

Figure 6.1 illustrates a class diagram for our algorithms. We also implemented *GetDFA, GetBFA, GetDirectAlgo* methods returning the corresponding algorithm to apply to a CIFS. The *SetX* and *SetY* classes represent partial solids respectively of $S\mathbb{X}$ and $S\mathbb{Y}$. The *SetY* class is thus abstract, because its implementation depends on the operation.

Constructing the formal representation, as it is described in chapter 5, requires additional theoretical analysis of the operator. First we should verify the self-similarity property (5.1) and define all the corresponding transformations $\tilde{T} : S\mathbb{Y} \to S\mathbb{Y}$. Second, we have to be able to verify the relation $\underset{\gamma}{\simeq}$ (see the inequality (5.4)) between the automaton's states. Here we can store the approximate attractor for each state, however in some cases the inequality can be transformed to an even simpler comparison (see the inequality (5.6)).

During the study of the self-similarity property (5.1) one can also verify the condition (5.8). If this condition is verified, applying the operator *SOp* is thus reduced to applying the generalised HUTCHINSON operator. Otherwise the approximate solution can be generated. Once we identify the transformations $\tilde{T}$ and implement the method to verify self-similarities, the algorithm, presented in section 5.2.2, can be applied to produce a CIFS with an attractor that is close to the operator image $SOp(S\mathcal{A})$.

Finally, after computing the approximate partial solid $SOp(S\mathcal{A})$ one should be able to obtain the required approximate solution $Op(\mathcal{A})$ by applying the *BuildSolution* method.

These are all the methods which have to be implemented to apply the operator to fractals.

6.2 Distance from a point

In this section we consider the operator for computing the distance from a given point. We also present the implementation of prerequisites, described in section 6.1.

Given an IFS defined by a finite set of contractive transformations $\{T_i\}_{i\in\Sigma}$ and the operator $Op : \mathbb{X} \to \mathbb{R}^+$ defined as follows:

$$p \in \mathbb{X} \quad Op : A \mapsto d(p, A) = \min_{a\in A} d(p, a).$$

It is easy to show, that for any sequence $(B_n)_{n\in\mathbb{N}}$ of non-empty compact subsets of $\mathbb{X}$ converging to $\mathcal{A}$, the corresponding sequence of the distances $(d(p, B_n))_{n\in\mathbb{N}}$ converges to $d(p, \mathcal{A})$, that is, the distance operator Op is continuous at $\mathcal{A}$. We can thus apply the generic algorithm approximating the distance to the attractor.

Since the following equality holds:

$$d(p, A \cup B) = \min\{d(p, A), d(p, B)\},$$

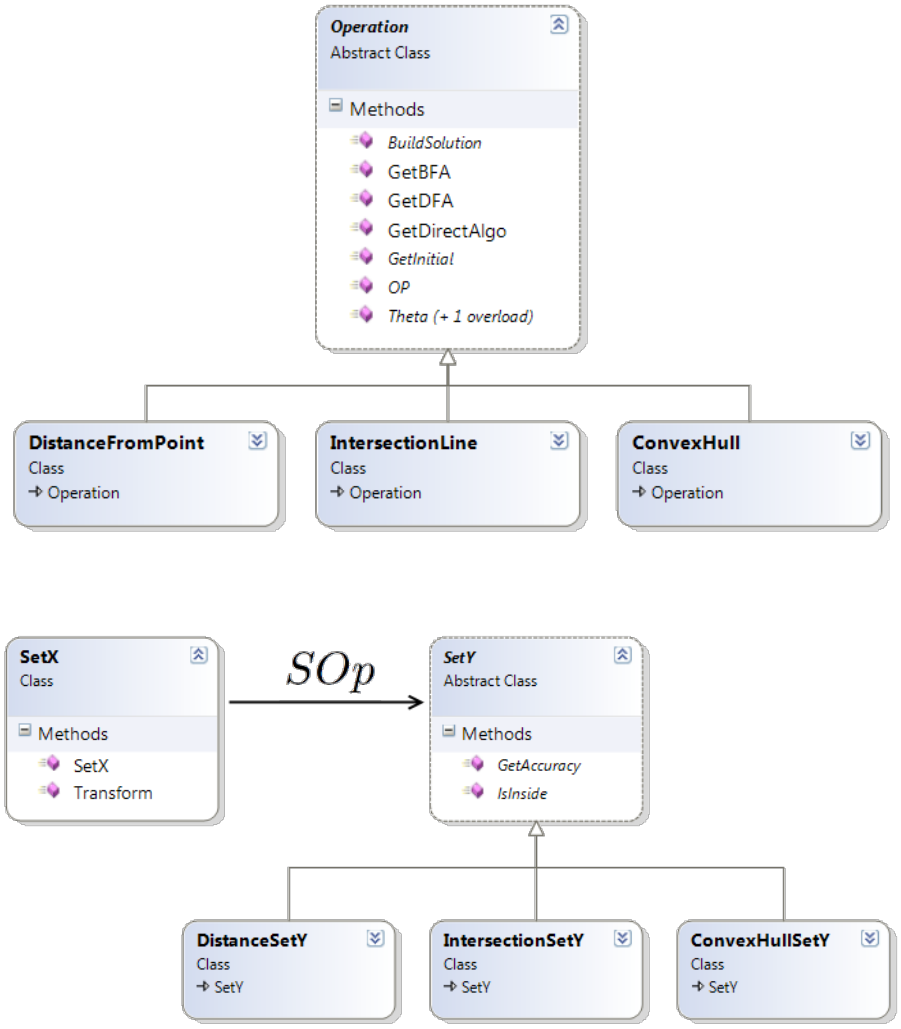


Figure 6.1: Class diagram for the presented solution.

there exists an associative operation $\Theta = \min\{\ldots\}$ decomposing the image $Op(B_n)$. The complexity of the operator Θ as a function of a count of parameters is therefore:

$$f_\Theta(x) = O(x).$$

The initial compact set B can be chosen as a point, that is to say, each time we apply the operator to a single transformed point. The complexity of the operator Op is constant:

$$f_{Op}\ (1) = O(1).$$

As described in section 4.4, we defined two approximation algorithms. There is no difference between applying Θ one time to many elements and applying Θ many times to N elements (see 4.4.2). Both the breadth-first and depth-first algorithms have the same complexity $O\left(N^n\right)$, where N is the number of the IFS transformations and n is the number of iterations chosen by equation (4.1). However, both algorithms have the same complexity as that of simply computing $Op(B_n)$.

The optimization described in section 4.5 requires the notion of interior and exterior bounds of the approximate solution. In order to optimize the computation we should thus extend the operator Op to the solid domain $S\mathbb{R}^+$.

As described in section 4.3, we choose the initial compact subset B as an attractor bounding ball, such that $T_i(B) \subseteq B\ \forall i \in \Sigma$. Thus we consider the operator defined by:

$$SOp : (\emptyset, B^\complement) \mapsto ((\max_{b\in B} d(p,b), +\infty), (0, \min_{b\in B} d(p,b))).$$

The values $\min\limits_{b\in B} d(p,b)$ and $\max\limits_{b\in B} d(p,b)$ correspondingly indicate the lower and upper bounds for the operator result $SOp(S\mathcal{A})$. On the next iterations we will precise these bounds, that is to say that we will restrict the set of possible solutions up to the required precision ε. The complexity of the extended operator SOp evaluated for a single transformed ball is the same.

In the optimized algorithm we always compute the operator on a single transformed initial partial solid. By choosing the initial compact subset B as an attractor bounding ball $B(c,r)$ as described in section 4.3, the lower and upper bounds can be simply computed as follows:

$$\min_{b\in B} d(p,b) = d(p,c) - r,$$
$$\max_{b\in B} d(p,b) = d(p,c) + r$$

The operator Θ, in this case, is the continuous union operation:

$$(A,B)\ \Theta\ (C,D) = (A \cup B, C \cap D).$$

The complexity of the operator Θ as a function of the number of parameters is therefore $f_\Theta(x) = O(x)$, due to the fact that each composition requires two comparisons of real numbers.

The algorithm steps are the following. At the beginning, we identify the initial solution set, as illustrated in figure 6.2:

$$S = SOp(SB) = ((d+r, +\infty), (0, d-r)),$$

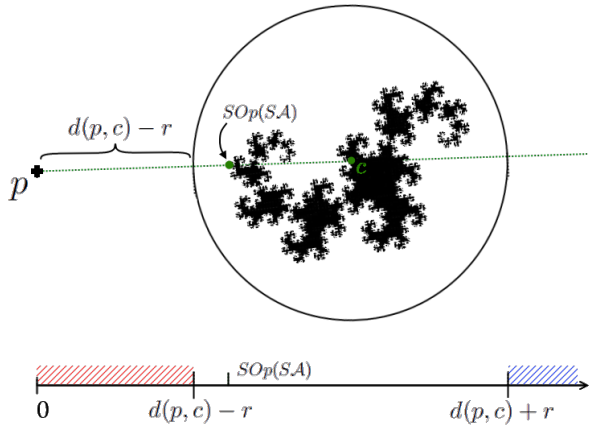


Figure 6.2: Initial solution set. The lower and upper bounds of the solution are computed as the minimal and maximal distances to the attractor bounding ball. The exact result $SOp(S\mathcal{A})$ is thus situated somewhere in between.

where $B = B(c, r)$ is the initial bounding ball and $d = d(p, c)$ is the distance from its center to the point p.

To be able to restrict the set of possible solutions, we should not compare solution sets between different tree levels. Thus we reset S to be able to obtain a more accurate result on the next iteration. We then start a loop for all the IFS transformations T_i, $\forall i \in \Sigma$.

Let us consider, for example, the first transformation T_0. We compute the solution set:

$$SOp(T_0(SB)) = ((d_0 + r_0, +\infty), (0, d_0 - r_0)),$$

where $d_0 = d(p, T_0(c))$, $r_0 = r \cdot s_0$ and s_0 is the contraction coefficient of T_0. The stopping criterion obviously does not return $+$. We therefore specify the global solution set S by composing with $SOp(T_0(SB))$:

$$\begin{aligned} &S \leftarrow S \ominus SOp(T_0(SB)) = \\ &= ((\min\{d + r, d_0 + r_0\}, +\infty), (0, \min\{d - r, d_0 - r_0\})) \\ &= ((d_0 + r_0, +\infty), (0, d - r)), \end{aligned}$$

and we continue the iterations for this branch of the tree by adding T_0 to the list (ts' in the implementation BFAlgoOpt) of transformations for the next iteration.

For the next transformation T_1 we execute the same steps. We compute the corresponding solution set:

$$SOp(T_1(SB)) = ((d_1 + r_1, +\infty), (0, d_1 - r_1)),$$

where $d_1 = d(p, T_1(c))$, $r_1 = r \cdot s_1$ and s_1 is the contraction coefficient of T_1. We verify the stopping criterion (4.12) for this solution set:

$$SOp(T_1(SB)) \subseteq S = \begin{cases} + & \text{if } d_1 - r_1 \geqslant d_0 + r_0, \\ - & \text{if } d_1 + r_1 \leqslant d_0 - r_0, \\ \perp & \text{otherwise.} \end{cases}$$

If it returns $+$, we can deduce that for all solutions in this branch of the tree there exists a better one in the previous branch associated with T_0. We can therefore avoid unnecessary computations and stop iterations for this branch T_1. Otherwise, we also add T_1 to the list (ts' in the implementation BFAlgoOpt) of transformations for the next iteration.

After n iterations of the algorithm we obtain a solution set of the following form:

$$((b, +\infty), (0, a)),$$

where $a, b \in \mathbb{R}^+$ are correspondingly the lower and upper solution bounds. The exact solution is therefore somewhere in the segment $[a, b]$. We also know that the distance between the exact solution and a (the same for b) is inferior to the given precision ε. That is to say, any point between a and b is an appropriate result.

In the pessimistic case where the stopping criterion is never verified, we have to go over the entire tree. The worst-case running time is therefore exponential $O(N^n)$, where N is the number of the IFS transformations and n is the required number of iterations. However, the best-case running time of the closest ball search is equal to the depth of the tree, that is to say, $O(\log N^n) = O(n)$.

6.2.1 Implementation

In this section we present the implementation of prerequisites for the operator used in computing the distance from a given point.

As described in section 6.1, the first function to be implemented determines the initial partial solid for a given state $q \in Q$, i.e. $q \mapsto SB^q$. We compute the bounding balls B^q for each CIFS attractor $\mathcal{A}^q$ using the algorithm, presented in appendix A. The initial partial solids SB^q are thus the pairs $(\emptyset, B^{q\complement})$.

As described in section 6.2, we represent the solution sets in $S\mathbb{Y}$ by pairs of the following form: $((a, +\infty), (0, b))$, where $a, b \in \mathbb{R}$ are two real values. The following pseudo-code illustrates how we define the solids of $S\mathbb{Y}$:

Object *DistanceSetY* implements *SetY*
Body:
$in \in \mathbb{R}$ is the interior part of the partial solid
$ex \in \mathbb{R}$ is the exterior part of the partial solid

Function GetAccuracy
Output: An accuracy of a given partial solid.
Body:
return $|in - ex|$
End

Function Inclusion predicate $\subseteq$
Input: A, B are objects of type *DistanceSetY*.
Output: $A \subseteq B$.
Body:

$$\text{return} \begin{cases} + & \text{if } B.in \leqslant A.ex, \\ - & \text{if } A.in \leqslant B.ex, \\ \perp & \text{otherwise.} \end{cases}$$

End

End

Thus we defined the nature of solids in $S\mathbb{Y}$, and the operator SOp itself can now be implemented. The following function illustrates the implementation of the operator SOp:

Function OP
Input:
SB is the input partial solid in $S\mathbb{X}$ representing a transformed ball
Output: The distance from the point $p \in \mathbb{X}$ to the partial solid SB
Body:
$dist \leftarrow |p - SB.center|$
return $DistanceSetY$ with
  $in = dist + SB.radius$
  $ex = \max\{0, dist - SB.radius\}$
End

In simpler terms, for a transformation T we approximate the transformed ball $T(B(c,r))$ by the ball $B(T(c), s(T) \cdot r)$. Thus we denote the center of the transformed input ball SB by $SB.center$ and its radius (or the semi-major axis for ellipses) by $SB.radius$.

The decomposing operator Θ is simply defined for two partial solids in $S\mathbb{Y}$ by the following function:

Function Θ
Input: A, B are objects of the type $DistanceSetY$.
Output: $A \Theta B$
Body:
return $DistanceSetY$ with
  $in = \min\{A.in, B.in\}$
  $ex = \min\{A.ex, B.ex\}$
End

In the breadth-first algorithm this operator could also compose a whole list of intermediate results, i.e. $\Theta : S\mathbb{Y}^k \rightarrow S\mathbb{Y}$, where k is the number of input results. For this we simply apply the operator Θ to the first two elements in the list, and we then recursively compose the result with the following elements.

Finally, after computing the approximate partial solid $SOp(S\mathcal{A})$ of the type $DistanceSetY$, one should be able to obtain the real distance from the given point p to the attractor $\mathcal{A}$. We therefore define a mapping $S\mathbb{Y} \rightarrow \mathbb{R}$ as follows:

Function BuildSolution
Input: A is an object of the type $DistanceSetY$ approximating $SOp(S\mathcal{A})$.
Output: An approximate operator image $Op(\mathcal{A})$
Body:
return $\frac{A.in+A.ex}{2}$
End

These are all the methods we need to implement in order to use the presented algorithm.

6.3 Convex hull

In this section we consider the convex hull computing operator for a given affine IFS.

Consider an IFS defined by a finite set of affine contractive operators $\{T_i\}_{i\in\Sigma}$ and the convex hull operator $Op = conv$. There exists an associative binary operation Θ merging two convex hulls, i.e.:

$$conv(A \cup B) = merge(conv(A), conv(B)).$$

One can prove that the operator *conv* is continuous at point $\mathcal{A}$. Merging convex hull runs faster that recalculating the overall convex hull, because in merging we use information according to which the initial sets are convex [MGLS12].

However, in order to apply the optimized algorithm, we should extend the operator to the solid domain. Let us consider the convex hull of the fixed points $fix(T_i)$ for $\forall i \in \Sigma$. We can only find fixed points approximately. Let $\delta > 0$ be the accuracy used in calculating the fixed point. We then compute the offset on a distance $-\delta$ to ensure that the resulting set will be inside the exact convex hull $conv(\{fix(T_i) \mid i \in \Sigma\})$. This set is an interior part of the initial partial solid SB. The exterior can be chosen simply as a bounding ball of the attractor.

Let us consider an extended operator SOp defined by:

$$SOp : (A, B) \mapsto (conv(A), conv(B^{\mathsf{c}})^{\mathsf{c}}).$$

One can see that $SOp(SB) = SB$ and the interior part of the image can thus be constructed. The interior of partial solids $SOp(T(SB))$ for an arbitrary composite transformation T can be constructed as well. We therefore have both the interior and the exterior parts of the solution. Moreover, the operation images constitute an increasing chain, that is we can apply the optimization described in section 4.5.

The operator Θ is merging:

$$(A, B)\ \Theta\ (C, D) = (merge(A, B), merge(C^{\mathsf{c}}, D^{\mathsf{c}})^{\mathsf{c}}).$$

Merging two convex hulls is linear on the number of points in the convex hulls. So, merging x convex hulls requires $O(h \cdot x^2)$ time, where h is the number of convex hull vertices. The complexity of the operator Θ as a function of the parameters count is therefore:

$$f_\Theta(x) = O(x^2).$$

Note that the equation (4.10) is satisfied, that is to say that, the depth-first algorithm is better than the breadth-first one.

The algorithm performs the following steps. We start by identifying the initial partial solid SB as described in section 4.3.4 and get down to the n-th level of the IFS tree. We then compute the solution set $SOp(T_0^n(SB))$ and therefore assign the global solution set to $S = SOp(T_0^n(SB))$. similarly, on the next N iterations we compute the solution sets $SOp(T_0^{n-1}T_i(SB))$ for $\forall i \in \Sigma$ and merge the corresponding convex sets by Θ.

On the next sub-branch of the tree we have to compute the solution set $SOp(T_0^{n-2}T_1(SB))$ to verify the stopping criterion (4.12). If this solution set is inside the global solution set S, we can deduce that all the tree leaves will also lie inside S, that is to say, all this sub-tree has already been counted in the solution. We therefore stop iterations for this sub-tree. Otherwise we should continue to explore the tree, because each leaf can potentially extend the final result.

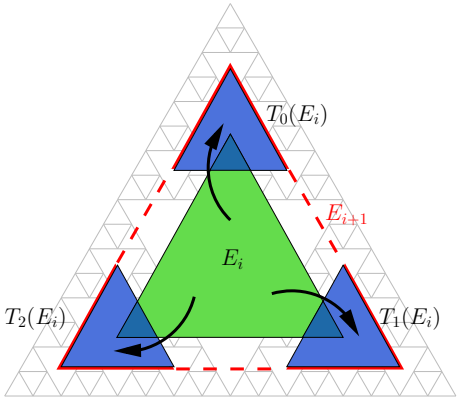


Figure 6.3: Iterative construction of approximations. Given an ε_i-approximation of the convex hull, we calculate its images by applying all the IFS transformations to its vertices. We then merge all these images to obtain the ε_{i+1}-approximation of the convex hull.

The process is repeated iteratively to achieve an approximate convex hull of the required precision. Figure 6.3 illustrates these iterations.

At a given time, we compute the operator Θ for N convex hulls maximum. This takes $O(hN^2)$ time, where h is the number of convex hull vertices, and N is the number of the IFS transformations. The complexity of evaluating Θ is therefore constant on the number of iterations n. On the other side, the breadth-first algorithm compute one final operation Θ on at most N^n elements. This takes $O(hN^{2n})$ time, which is exponentially worse than the chosen algorithm.

Note that if the output convex hull has an infinite number of vertices, these algorithms quickly go beyond the space and time limits, because merging depends on the number of convex hull vertices and is evaluated at most N^n times to achieve the n-th level of the tree.

As described in section 5.3.1, it is possible to formally represent the convex hull. Since the IFS transformations are affine, the self-similarity property of the operator is verified:

$$\forall i \in \Sigma \;\; conv(T_i(SB)) = T_i(conv^{T_i}(SB)) = T_i(conv(SB)).$$

Moreover, we also have the equality of all the sub-attractors (see 5.8):

$$conv^{T_i}(SB) = conv(SB) \;\; \forall i \in \Sigma.$$

In other words, the algorithm starts with the initial state $\natural$. Since all the sub-attractors $conv^{T_i}(SB)$ are equal, for each transformation T_i we add a cyclic transition from the state $\natural$ to itself and then the algorithm stops. The produced generalised IFS constructs the attractor $conv(S\mathcal{A})$ by iteratively applying the generalised Hutchinson operator $\tilde{\mathbb{T}}(SB) = \underset{i\in\Sigma}{merge}(T_i(conv(SB)))$, as described in section 5.3.1. Evaluating the convex hull is thus reduced to evaluating this IFS.

Note that in this method we do not construct the IFS evaluation tree with its exponential number of objects. Instead, we compose all the results on each

iteration, which is, in fact, a more natural way of evaluating the IFS. In this case, both the space and time complexities are reduced, because our composing operator simplifies the resulting approximate object in contrast to the union operator.

Since merging N convex hulls requires $O(hN^2)$, where h is the number of convex hull vertices and $N = |\Sigma|$ is the number of IFS transformations, the total time complexity necessary for constructing the convex hull is $O(nhN^2)$. Note that it linearly depends on the number of iterations n.

6.3.1 Simplification of the convex hull approximation

The presented results confirm our previous analysis for the convex hull of affine IFS [MGLS12]. However, sometimes the algorithms could be optimized by using operator specific properties. For instance, the approximate convex hull can be simplified without any loss of accuracy.

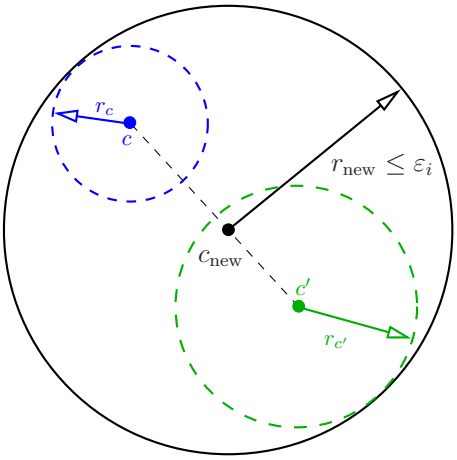


Figure 6.4: Illustration of the simplification. If two neighboring points c and c' are close, i.e. $d(c, c') + r_c + r_{c'} \leqslant 2\varepsilon_i$ we can then replace these points with the middle point without any loss of accuracy.

We simplify the approximate convex hull at each iteration by replacing vertices that are close enough to one another with one single vertex. To identify such vertices, we simply need to check the neighbouring points on the approximate convex hull.

If two neighbouring points c and c' satisfy:

$$d(c, c') + r_c + r_{c'} \leqslant 2\varepsilon_i \tag{6.1}$$

we can then replace these points with the middle point without any loss of accuracy, as shown in figure 6.4.

Thus, the number of points considered in constructing intermediate convex hulls is reduced and intermediate convex hulls are simplified without losing any accuracy. To avoid a constant rebuilding of data structures the simplification can be performed during the merge. This method of simplification can therefore

easily be integrated into our approach by redefining the composing operator Θ in the following way:

$$(A, B)\ \Theta\ (C, D) = (mergeAndSimplify(A, B), mergeAndSimplify(C^c, D^c)^c),$$

where the method $mergeAndSimplify$ takes into account distances between neighbouring points, as described above.

6.3.2 Results and discussion

We examined the presented algorithm for various IFS attractors and different precision levels in 2D and 3D spaces. It should be pointed out that all tests were performed on an ordinary PC (Intel®Core™2 Duo T7500 2.2GHz 3GB RAM) without any additional computational facilities.

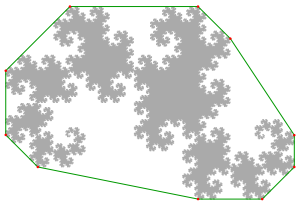
Several results for various IFS attractors are presented in figures 6.5 and 6.6. For each attractor, the program generates an approximation of the convex hull at the specified accuracy. The execution time of our algorithm is presented in Table 1.

Attractor	N	s_{max}	dim	ε	time(ms)
Sierpinski triangle	3	0.5	2	10^{-9}	16
von Koch curve	4	0.25	2	10^{-9}	27
Dragon curve	2	0.70711	2	10^{-9}	62
Tree	5	0.66762	2	10^{-9}	94
Sierpinski tetrahedron	4	0.5	3	10^{-9}	109
IFS dragon	2	0.91944	2	10^{-9}	265
Barnsley fern	4	0.85094	2	10^{-9}	484
FIF	4	0.86603	3	10^{-3}	3541
3D Barnsley fern	4	0.85586	3	10^{-3}	10951

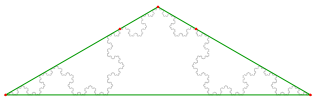
Table 1. *Execution time of our algorithms for several IFS attractors with the precision ε. Here N is the number of the IFS transformations, s_{max} is their maximal contraction coefficient and* dim *is the dimension of the space in which the attractor lies.*

An approximation of the convex hull is computed in an interactive rate for all of the 2D attractors and also for the attractors whose convex hulls have a finite number of vertices.

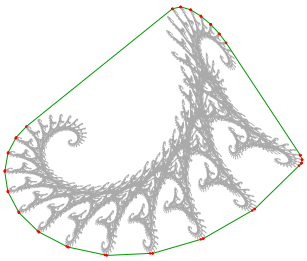
We compare our method for computing an approximate convex hull with Martyn's approach [Mar09a]. Table 2 demonstrates the execution time of our algorithm compared to Martyn's for 2D IFS attractors.



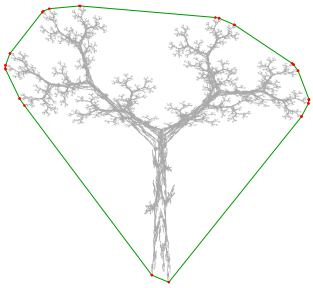
Dragon curve



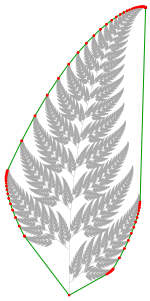
von Koch curve



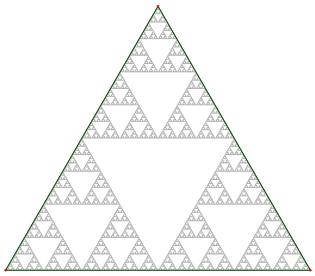
IFS dragon



Tree

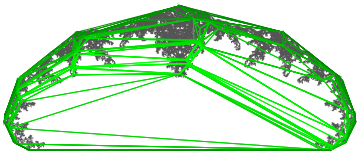


Barnsley fern

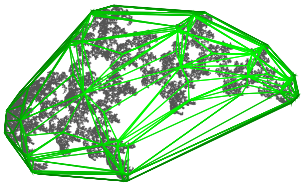


Sierpinski triangle

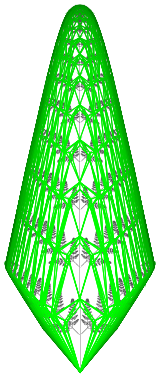
Figure 6.5: Results for 2D IFS attractors at precision $\varepsilon = 10^{-9}$



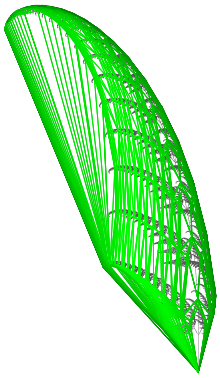
Fractal interpolation function



Fractal interpolation function



3D Barnsley fern



3D Barnsley fern

Figure 6.6: Results for 3D IFS attractors at precision $\varepsilon = 10^{-2}$

Attractor	N	s_{max}	Martyn's method time, ms	Our algorithm time, ms
von Koch curve	4	0.25	125	16
Sierpinski triangle	3	0.5	647	16
Dragon curve	2	0.70711	756	16
Tree	5	0.66762	1131	31
Barnsley fern	4	0.85094	45630	250
IFS dragon	2	0.91944	more than 60 sec	78

Table 2. *Execution time of the algorithms for several IFS attractors with a given precision $\varepsilon = 10^{-3}$. Here N is the number of the IFS transformations and s_{max} is their maximal contraction coefficient.*

One can see that Martyn's approach strongly depends on the maximal contraction coefficient s_{max} whereas our algorithm depends more on the number of the approximate convex hull vertices h.

The stopping criterion (4.12) can significantly reduce the number of operations, and our optimization therefore provides a qualitative gain in execution time.

Simplifying the convex hull is another important process. To improve collision tests performance it is necessary to verify whether an object intersects with the convex hull of our attractor. This process requires testing all the faces of the convex hull faces. The simplified convex hull involves a smaller number of vertices and faces, verifying the intersections will therefore take less time. The number of approximate convex hull vertices for $\varepsilon = 10^{-3}$ are presented in the following table:

Attractor	without simplification	with simplification
Tree	30 pts.	25 pts.
IFS dragon	67 pts.	59 pts.
Barnsley fern	186 pts.	101 pts.
3D Barnsley fern	4577 pts.	359 pts.

On average, the efficiency of the simplification is about 46%, in 2D, but it decreases, when we increse the level of precision. In 3D, this process is much more efficient - we eliminate about 90% of vertices. However, such efficiency can be achieved only for the attractors whose convex hulls have an infinite number of vertices.

Another important advantage is that we can improve the precision of the obtained approximation by further iterating our algorithm. That is to say, to improve the precision of the process it is not necessary to recalculate all k iterations, we start with the approximation first obtained and continue to iterate the algorithm. This can be useful advantage when it is necessary to compute a sequence of the approximate convex hulls (to obtain different levels of detail for example).

6.4 Boolean intersection

In this section we present an example of the operator used in calculating the intersection between a given IFS and a given set of $\mathbb{X}$. We also present the

implementation of prerequisites, as it was described in section 6.1.

Given an IFS defined by a finite set of contractive transformations $\{T_i\}_{i\in\Sigma}$ and the operator $Op : \mathbb{X} \to \mathbb{X}$ defined as follows:

$$L \subseteq \mathbb{X} \quad Op : A \mapsto A \cap L,$$

where L is an object intersect a given IFS.

The boolean intersection is not continuous for the HAUSDORFF metric. To apply the generic algorithms and also to optimize the process as described in section 4.5, we should extend the operator Op to the solid domain $S\mathbb{X}$. More precisely, we consider the operator defined by:

$$SOp : (\emptyset, B^{\mathsf{c}}) \mapsto (\emptyset, (B \cap L)^{\mathsf{c}}).$$

Here, the initial compact set SB is chosen as $(\emptyset, B(c,r)^{\mathsf{c}})$, where $B(c,r)$ is an attractor bounding ball. In this case, we do not construct the interior part, because it is difficult and not always possible. The interior is therefore approximated by the empty set $\emptyset$, that is to say, only the exterior is used to approximate the solution, as described in section 4.5.1. We also consider some modified algorithms without constructing the global solution set S.

It can easily be shown, that the operator Θ is the union operator:

$$\begin{aligned} Op(T_i(SB) \cup T_j(SB)) &= (\emptyset, ((T_i(B) \cup T_j(B)) \cap L)^{\mathsf{c}}) = \\ &= (\emptyset, ((T_i(B) \cap L) \cup (T_j(B) \cap L))^{\mathsf{c}}) = \\ &= (\emptyset, (T_i(B) \cap L)^{\mathsf{c}} \cap (T_j(B) \cap L)^{\mathsf{c}}) = \\ &= Op(T_i(SB)) \cup Op(T_j(SB)), \end{aligned}$$

where T_i and T_j are the IFS transformations, L is the object intersecting them, and B is the initial compact subset. The complexity of the operator Θ as a function of the parameters count is therefore linear:

$$f_\Theta(x) = O(x).$$

While evaluating the algorithm, we apply the operator SOp only to transformed balls. Intersecting between classic shapes does not depend on the number of iterations. The complexity of the operator SOp is therefore:

$$f_{SOp}\ (1) = O(1).$$

As described in section 4.4, we defined two approximation algorithms. As composing by Θ is linear on the number of parameters, there is no difference between applying Θ one time to many elements or applying Θ many times to N elements (see 4.4.2). Both the breadth-first and depth-first algorithms have the same complexity $O\left(N^n\right)$, where N is the number of IFS transformations and n is the chosen number of iterations as described in section 4.3.2.

We do not construct the interior and the stopping criterion (4.12) therefore reduces to the following:

$$SOp(T(SB)) = (\emptyset, \mathbb{Y}).$$

This simplification reduces the running-time, because we do not construct the global solution set S. Nevertheless, the asymptotic complexity remains the same.

Moreover, the self-similarity property is also verified, which allows us to factor out the transformation, and to always apply the operator to the same partial solid SB before transforming, as described in section 5.1. One can thus find the corresponding transformations $\tilde{T}_i$ (see section 5.1.1) in order to construct the approximate iterative model (see section 5.2).

The modified operator SOp^T, for a corresponding composite transformation T, differs from the original operator SOp by intersecting the initial partial solid SB with the modified object L. In the case of an invertible transformation T we have:

$$SOp^T : (\emptyset, B^{\mathsf{c}}) \mapsto (\emptyset, (B \cap T^{-1}(L))^{\mathsf{c}}).$$

How to deal with the non-invertible transformation was described in section 5.1.1. The generic algorithm, described in section 5.2, produces a CIFS with the attractor that is close enough to the operator image $SOp(S\mathcal{A})$.

In some cases, when the solution is exactly self-similar, the algorithm stops for any approximation accuracy. This is the case when only a finite number of semi-decidable equations have to be verified. If all these equations are proved manually the resulting CIFS formally represents the exact solution, that is to say, its attractor $SOp(S\mathcal{A})$ is defined by applying the generalised HUTCHINSON operator.

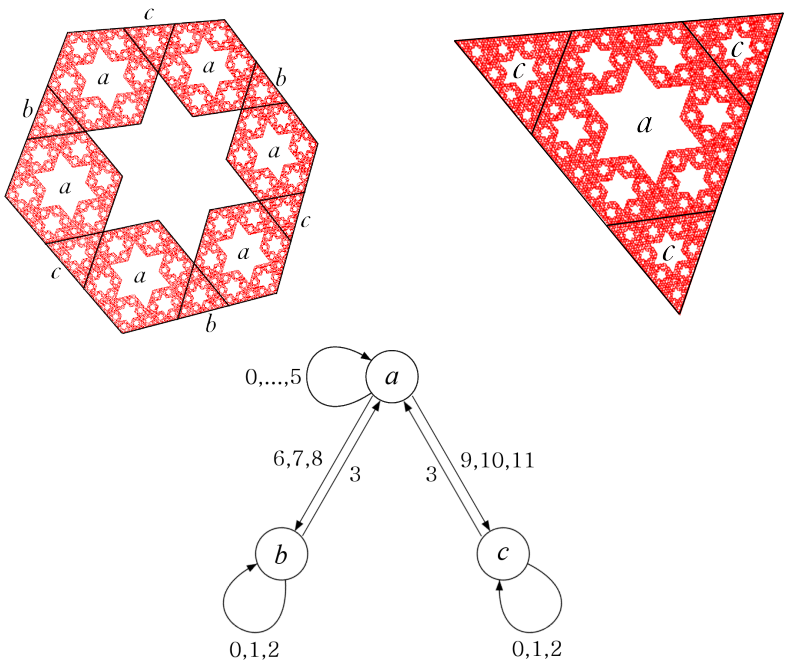
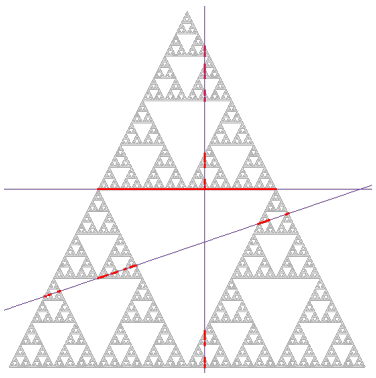
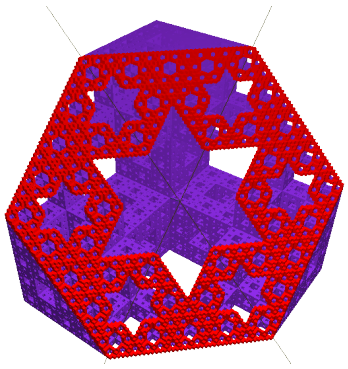


Figure 6.7: Subdivision of the intersection between a plane and the MENGER sponge.

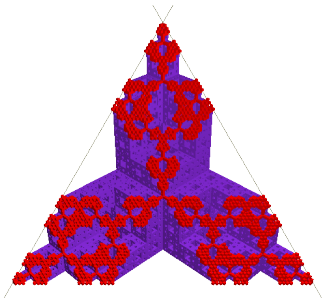
Figure 6.7 shows an example of intersection between a plane and the MENGER

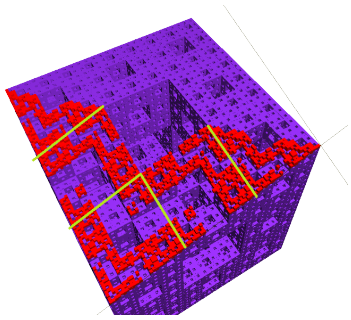


SIERPINSKI triangle sections

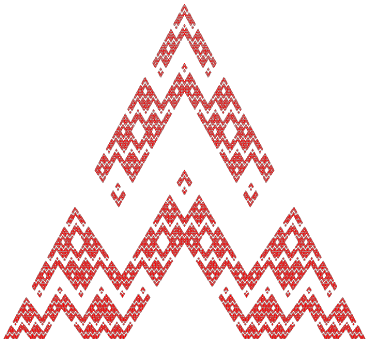


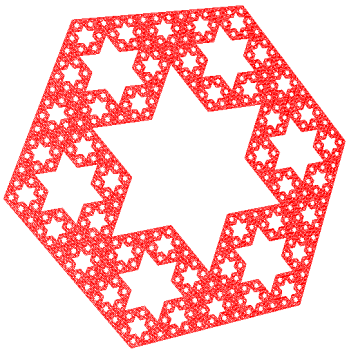
Approximate MENGER sponge section





Approximate models of the MENGER sponge sections





Exact models of the MENGER sponge sections

Figure 6.8: Attractors of various intersection images.

sponge. The output automaton's states illustrated on the CIFS attractors are shown in the figure. There are three states: a, b and c. The attractor $\mathcal{A}^a$ is in the shape of an hexagon, and the attractors $\mathcal{A}^b$ and $\mathcal{A}^c$ are triangles. Figure 6.8 illustrates several examples of intersection.

One can also compute a set difference and a symmetric difference by expressing these operators in terms of union, intersection and complement.

6.4.1 Implementation

In this section we present an implementation of the prerequisites for the operator to compute the intersection with a given line L. Let the line L be defined by a point $L.ref$ and a vector $L.dir$ representing the reference point on the line and the direction correspondingly.

As for the distance example in section 6.2.1, the first function which has to be implemented determines the initial partial solid as a bounding ball B^q of the attractor $\mathcal{A}^q$ for a given state $q \in Q$ using the algorithm, presented in appendix A. The initial partial solids SB^q are thus the pairs $(\emptyset, B^{q\complement})$.

Solution sets in $S\mathbb{Y}$ are represented by a set of segments on the line L as the exterior part and the empty interior part. The following pseudo-code illustrates how to define partial solids in $S\mathbb{Y}$:

Object *IntersectionSetY* implements *SetY*
Body:
segments is the list of segments

Function GetAccuracy
Output: an accuracy of a given partial solid.
Body:
// We do not use this stopping criteria
return ∞
End

// We use the simplified stopping criterion (see section 4.5.1).
Function Inclusion predicate $\subseteq$
Not applicable

End

Thus we defined the nature of partial solids in $S\mathbb{Y}$, and the operator SOp itself can now be implemented. The following function illustrates the operator SOp implementing:

Function OP
Input:
SB is the input partial solid in $S\mathbb{X}$ representing a transformed bounding ball
Output: an intersection $SB \cap L$ represented by *IntersectionSetY*
Body:
$v \leftarrow SB.center - L.ref$
$h \leftarrow v \cdot L.dir$
if $SB.radius^2 - |v|^2 + h^2 \leqslant 0$
    return $(\emptyset, \mathbb{Y})$
endif
$l \leftarrow \sqrt{SB.radius^2 - |v|^2 + h^2}$
return *IntersectionSetY* with
    $segments = [(h - l, h + l)]$
End

The decomposing operator Θ is simply defined for two partial solids in $S\mathbb{Y}$ by the following function:

Function Θ
Input:
A, B are objects of the type *IntersectionSetY*.
Output: a composition $A \Theta B$
Body:
return *IntersectionSetY* with
$segments = A.segments \cup B.segments$
End

Here the union of two lists means the pairwise unite of all segments in the first list with segments in the second one. Whenever two segments (a, b) and (c, d) intersect with each other we union them, i.e. $(\min\{a, c\}, \max\{b, d\})$. The segments, which do not intersect, may be left unchanged.

In the breadth first algorithm the operator Θ could also compose a whole list of the intermediate results, i.e. $\Theta : S\mathbb{Y}^k \to S\mathbb{Y}$, where k is the number of input results. We simply apply the defined operator Θ to the first two elements in the list, and then we recursively compose the result with each the following element.

Finally, after computing the approximate partial solid $SOp(S\mathcal{A})$ in the form of the *IntersectionSetY* object, the output result can be obtained by using its *segments*:

Function BuildSolution
Input:
A is the objects of the type *IntersectionSetY* approximating $SOp(S\mathcal{A})$.
Output: an approximate operator image $Op(\mathcal{A})$
Body:
return $A.segments$
End

As described in section 5.2.2, to determine similar sub-attractors we verify the inequality (5.6). It implies the inequality (5.4) and does not contain any applications of the operator SOp, that is to say that we verify it only if two transformed lines are close enough.

These are all the methods which have to be implemented to apply the operator using the presented algorithm. Moreover, for this example we can construct a CIFS with the attractor that is close to the solution $SOp(S\mathcal{A})$.

6.5 Offset

The offsetting map produces a shape driven out from an original shape in equal distance and direction, normally or perpendicularly to the original shape. One usually distinguishes between the offsetting on a positive distance and on a negative distance. In this section we consider both offsetting maps and describe whether the maps could be computed by our approach.

Consider first the operator $Op : \mathbb{X} \to \mathbb{X}$ defined through the Minkowski sum as follows:

$$r \geqslant 0 \quad Op : A \mapsto A \setminus (\bar{A} \oplus B(0, r)),$$

where $B(0,r)$ is a ball centered at the origin with the radius r. The operator thus performs offsetting at a negative distance r.

Consider $Op(A \cup B)$ calculating, where A and B are two intersecting sets, as illustrated in figure 6.9. There is no relatively simple method to compose the results $Op(A)$ and $Op(B)$ in order to obtain $Op(A \cup B)$, because $Op(A)$ and $Op(B)$ do not contain any information about the intersection $A \cap B$. The operation Op image cannot thus be decomposed and to offset on a negative distance we have to apply the operation directly to the approximate attractor, as described in section 4.2.

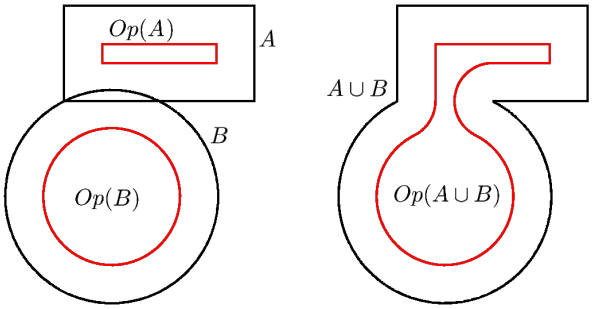


Figure 6.9: Counterexample where the offsetting on a negative distance cannot be decomposed by Θ operator.

Now we consider the positive offsetting operator $Op : \mathbb{X} \to \mathbb{X}$, defined as follows:

$$r \geqslant 0 \quad Op : A \mapsto A \oplus B(0,r),$$

where $B(0,r)$ is a ball centered at the origin with the radius r. Thus the operator offsets at a positive distance r.

The Minkowski sum can be extended [EL02] for partial solids of $\mathbb{X} = \mathbb{R}^d$ as a function $\oplus : S_b\mathbb{R}^d \times S\mathbb{R}^d \mapsto S\mathbb{R}^d$ defined by:

$$(A_1, B_1) \oplus (A_2, B_2) = (A_1 \oplus A_2, (B_1^{\mathsf{c}} \oplus B_2^{\mathsf{c}})^{\mathsf{c}}).$$

This map is thus well defined and continuous [EL02].

For this example, the initial compact set SB is chosen as $(\emptyset, B(c,r)^{\mathsf{c}})$, where $B(c,r)$ is an attractor bounding ball, as for the distance example (see 6.2). We do not construct the interior part, because it is difficult and not always possible. The interior is therefore approximated by the empty set $\emptyset$, that is to say, only the exterior is used to approximate the solution, as described in section 4.5.1. The operator extended to the solid domain is thus:

$$SOp : SB \mapsto SB \oplus SR,$$

where $SR = (B(0,r)^o, B(0,r)^{\mathsf{c}})$.

It can be shown, that the operator Θ is the union operator:

$$\begin{aligned} SOp(T_i(SB) \cup T_j(SB)) &= \{x + b \mid x \in T_i(SB) \cup T_j(SB), b \in B(0,r)\} = \\ &= \{x + b \mid x \in T_i(SB), b \in B(0,r)\} \cup \\ &\cup \{x + b \mid x \in T_j(SB), b \in B(0,r)\} = \\ &= SOp(T_i(SB)) \cup SOp(T_j(SB)), \end{aligned}$$

where T_i and T_j are the IFS transformations, SB is the initial partial solid. The complexity of the operator Θ as a function of the parameters count is therefore linear:

$$f_\Theta(x) = O(x).$$

Evaluating the algorithm only involves applying the operator SOp to transformed balls. If the IFS transformations are affine, the initial bounding ball is transformed into ellipses and offsetting these ellipses can be computed in constant time.

As described in section 4.4, we defined two approximation algorithms. Composing by Θ is linear on the number of parameters, there is thus no difference between applying Θ one time to many elements or applying Θ many times to N elements (see 4.4.2). Both the breadth-first and depth-first algorithms have the same complexity $O(N^n)$, where N is the number of the IFS transformations and n is the number of iterations chosen as described in section 4.3.2.

If we do not construct the interior, the stopping criterion (4.12) is simplified to the following:

$$SOp(T(SB)) = (\emptyset, \mathbb{Y}),$$

This is never the case because:

$$(T(B) \oplus B(0,r))^{\mathsf{c}} = \mathbb{Y} \Leftrightarrow T(B) \oplus B(0,r) = \emptyset.$$

This simplification does not reduce the running-time, because we always reach all the leaves of the evaluation tree and there are N^n leaves.

Now we consider the self-similarity of the Minkowski addition. Assume that all the IFS transformations are affine and can therefore be represented as follows:

$$T_i(x) = L_i(x) + v_i, \text{ for } i \in \Sigma, x \in \mathbb{X},$$

where L_i is the linear part of the transformation and v_i is a vector used for translation. We can thus write the following for any composite IFS transformation $T(x) = L(x) + v$, $x \in \mathbb{X}$:

$$\begin{aligned} T(B) \oplus B(0,r) &= \{T(x) + b \mid x \in B, b \in B(0,r)\} = \\ &= \{L(x) + v + b \mid x \in B, b \in B(0,r)\} = \\ &= \{L(x) + b - v + 2v \mid x \in B, b \in B(0,r)\} = \\ &= \{L(x + L^{-1}(b - v)) + 2v \mid x \in B, b \in B(0,r)\} = \\ &= \{T(x + T^{-1}(b)) + v \mid x \in B, b \in B(0,r)\} = \\ &= \{\tilde{T}(x + T^{-1}(b)) \mid x \in B, b \in B(0,r)\} \\ &= \tilde{T}(B \oplus T^{-1}(B(0,r))), \end{aligned}$$

where $\tilde{T}(x) = T(x) + v$ is the corresponding transformation providing the self-similarity. The operator SOp thus satisfies the self-similarity property and the corresponding modified operators are the following:

$$SOp^T : SB \mapsto SB \oplus T^{-1}(SR).$$

As described in section 5.1.1, in the case of non-invertible transformation T we have to find a set B' such that:

$$T(B') = SR,$$

where $SR = (B(0,r)^o, B(0,r)^c)$. Once we determine this set we can write the following:

$$\begin{aligned} T(B) \oplus B(0,r) &= \{T(x) + b \mid x \in B, b \in B(0,r)\} = \\ &= \{T(x) + T(b') \mid x \in B, b' \in B'\} = \\ &= \{T(x + b') + v \mid x \in B, b' \in B'\} = \\ &= \{\tilde{T}(x + b') \mid x \in B, b' \in B'\} = \\ &= \tilde{T}(B \oplus B'). \end{aligned}$$

We could thus continue the CIFS automaton construction as described in section 5.2.

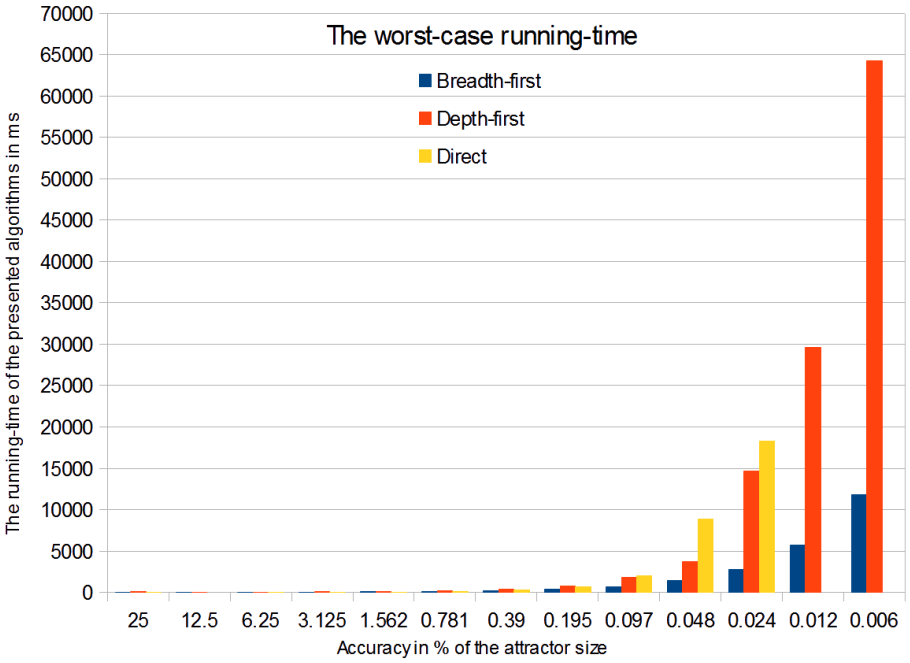


Figure 6.10: Worst-case running time of the optimized algorithms and the direct $Op(B_n)$ calculating for the SIERPINSKI triangle. The operation is the distance from a point, defined as described in section 6.2.

6.6 Results and discussion

In this section we show the gain derived from using the optimized algorithms and we compare the presented algorithms on different examples.

All the presented algorithms were designed and coded in C# language and tested on an ordinary laptop (Intel Core i7-2720QM CPU @2.2GHz).

Figure 6.10 illustrates the worst-case running-time for the presented algorithms. All the algorithms have an exponential running-time. However, the tree-cutting optimisation of the breadth-first algorithm is triggered earlier compared with the depth-first algorithm. As illustrated on the chart, the running time of the breadth-first algorithm is better than that of the depth-first algorithm. But the space complexity of the breadth-first algorithm is exponential, since on the i-th iteration we have to keep all the N^i tree nodes in accessible memory to compute the approximate result. The algorithm therefore runs out of memory very quickly. This does not happen with the depth-first algorithm, since on each iteration we need to keep only one branch of the evaluation tree in accessible memory.

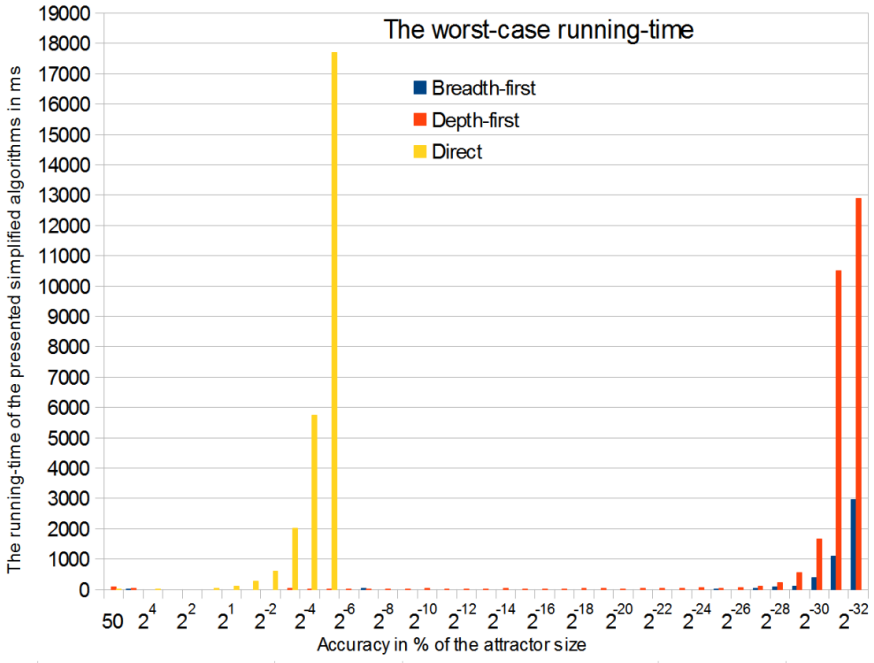


Figure 6.11: Worst-case running time of the optimized algorithms without constructing the solution interior part, and the direct $Op(B_n)$ calculating for the Sierpinski triangle. The operation computes the intersection with a line, as described in section 6.4.

Figure 6.11 illustrates the worst-case running-time for the presented simplified algorithms. As described in section 4.5.1, we do not construct the interior part of the approximate result. Only the exterior is therefore used to approximate the solution. In this case, the stopping criterion (4.12) is only verified for the empty partial solid allowing us to simplify the verifications. The algorithms

also have an exponential running-time.

As in the previous example, compared to the depth-first algorithm, the breadth-first algorithm is triggered earlier and has a better running-time. However, the algorithm also runs out of memory very quickly, which does not happen with the depth-first algorithm. Compared to the direct algorithm both optimized algorithms still show a considerable gain in running time.

The best-case running time of both the optimized and simplified algorithms is constant, representing the gain over the direct algorithm which exponentially depends on the number of iterations.

The following table illustrates the complexities of the algorithms described in this thesis:

Algorithm	Time	Space
Direct	$O\left(f_{SOp}\left(N^n\right)\right)$	$O\left(N^n\right)$
Breadth-first (BFA)	$O\left(N^n + f_\Theta(N^n)\right)$	$O\left(N^n\right)$
Optimized BFA	$O\left(N^n + f_\Theta(N^n)\right)$	$O\left(N^n\right)$
Simplified BFA	$O\left(N^n + f_\Theta(N^n)\right)$	$O\left(N^n\right)$
Depth-first (DFA)	$O\left(N^n\right)$	$O\left(N\right)$
Optimized DFA	$O\left(N^n\right)$	$O\left(N\right)$
Simplified DFA	$O\left(N^n\right)$	$O\left(N\right)$
CIFSAlgo	$O\left(N^n\right)$	$O\left(N^n\right)$
Generalised IFS	$O\left(n \cdot f_\Theta(N)\right)$	$O\left(N\right)$

We recall that $f_{SOp}\left(x\right)$ is the complexity of the operator SOp evaluated on the union of x objects, f_Θ is the complexity of the operation Θ as a function of parameters count, N is the number of the IFS transformations and n is the required number of iterations. Optimized algorithms do not depend on the operator SOp complexity because the operation is evaluated only for a single object. The function $f_{SOp}\left(x\right)$ is therefore constant on the number of iterations n.

Chapter 7

Conclusion and prospects

Our research is conducted as part of the ANR ModItère project . The global objective of this project is to develop a new type of geometric modellers for computer-aided design systems (CAD), based on iterative methods following the principles of fractal geometry.

Most existing CAD modellers are based on classic geometry. These modellers allow a user to model analytic curves and surfaces, which are usually polynomial or rational and smooth. Fractal shapes have more complex and specific geometric properties. Fractal shapes can generally be rough, lacunar, porous, they are not smooth and usually not differentiable in any way. It is therefore difficult or impossible to represent these shapes by using classic geometric models. To generalise the answer to this problem specific iterative models such as IFS, CIFS and BCIFS were developed, in which we are substantially interested. Defining shapes by iteration allows us to generate fractal structures with all the diversity of their specific properties, which cannot be achieved with classic modelling.

Shape modelling in CAD systems is complex, in that it is not confined to the simple shape representation. To assist the user, geometric modellers are improved by construction operations and algorithms. These operations are usually based on the geometric properties of the modelled objects. Existing CAD systems are not suitable to manipulate fractal structures because of their specific properties. After listing the constructing operations presented in most of CAD systems, we identify the basic properties required for their realization. A preliminary analysis shows that the operations are based on four properties: affine invariance, topological structure, parametrization and differential properties. The result is given in the form of a dependency graph.

From this graph, we classified CAD operations by adapting to BCIFS : certain operations require generalization because of the specific properties of fractals, for certain operations the result can be approximated. For approximating CAD operations, we thus introduced generic algorithms defined for arbitrary operations satisfying some constraints that we explicit.

The proof of convergence of these algorithms is based on domain theory. Most CAD operations are not continuous. We consider continuous extensions of CAD operations to the solid domain. Thus we represent the attractor in this domain, apply the continuous operation to the approximate attractor and then return to the modelling space (see figure 4.9).

The presented approximation algorithms are iterative. Domain theory allows

us to formalize the notion of solution sets that represent the information on the solution at the current approximation level. On each iteration we compute a local solution set representing a part of the global solution. In the optimized algorithms we use a criterion determining whether a local solution set could potentially improve the global one. The global solution set is restricted up to the required accuracy while iterating the algorithm. We then identify cases in which this criterion can also be simplified to a comparison with a constant.

Given a CAD operation which satisfies the image decomposition, it is possible to construct a CIFS with a generalised HUTCHINSON operator, whose attractor is close enough to the operation result with respect to the HAUSDORFF metric. Thus we introduce a generic algorithm to compute such CIFS for a given accuracy. We define the self-similarity property of the operation. This property defines a set of transformations, which are used in the output iterative system. Starting with the initial state, the algorithm iteratively traverses the evaluation tree of the input BCIFS. In order to provide a self-similar nature of the output set the algorithm verifies the similarity of the sub-attractors that were already parsed.

In order to construct an exact CIFS (with a generalized HUTCHINSON operator) of the operation, if it exists, we must prove all the necessary similarities manually. We also explicit the condition of the operation to be represented by an IFS with a generalised HUTCHINSON operator. In this case, only this condition should be proved manually.

We studied the complexity of all the presented algorithms and performed some experiments: a convex hull calculation, boolean set operations (intersection, union, difference) between fractal objects and classic CAD objects (lines, planes, splines, cylinders). A primary goal for our future works could be the integration of these approaches to our geometric modeller.

Several questions remain unanswered. As we described in section 4.3.4, there are restrictions on the choice of the initial partial solid. Constructing a suitable initial interior part is generally challenging, and we often have no choice except to work without approximating the interior. This makes it impossible to apply important operations such as the membership predicate and the subset inclusion.

Another question is related to representing the operation result. As described in chapter 5, when the self-similarity property (5.1), is satisfied, it is possible to construct a CIFS whose the attractor is close enough to the operation image. However, it is also interesting to derive the topological model of the output iterative system, that is to say to construct a BCIFS representing the operator image. For example, in applying a boolean intersection, some constraints of incidence and adjacency can be induced from the initial BCIFS constraints. However, the additional constraints for the new edges and faces somehow have to be identified.

Finally, a whole class of operations working with multiple attractors remains unexplored. Of course, for approximate computations we could replace the attractors with their approximations, and calculate the result using available techniques. However, the effectiveness of this approach may to be questioned. The key difficulty here is that we have to increase the level of approximation in all attractors in parallel, that is to say, to find a balance between the rates of their evolution.

Appendices

Appendix A

Computing the attractor bounding ball

A.1 Bounding ball for the attractor of an IFS

Methods to determine a bounding ball for the attractor of an IFS have already been developed.

Gentil [Gen92] described an approach based on the dichotomous search for the minimal radius of a ball that bounds the attractor of an IFS. The approach can be applied in a multi-dimensional space and calculates the result for a given precision.

Hart and DeFanti [HD91] introduced a method which starts with the unit ball centered at the origin. The algorithm iteratively produces a sequence of balls converging to the limit ball that bounds the attractor.

Rice [Ric96] improved on Hart and DeFanti's approach by optimizing the radius of the bounding ball using a generic optimization package. He also showed that the centre of the limit ball can be determined analytically by solving a system of linear equations.

Martyn [Mar03] showed that the solution of this system is the centroid of the attractor with particular weights. To obtain a better approximation, he presented a heuristic iterative method called "balancing the attractor". The algorithm is not limited by the dimension of the space in which the attractor lies.

More recently, Martyn [Mar09b] presented a novel approach to approximate the smallest disc to enclose the affine IFS attractor at any accuracy. The method is based on the concept of spanning points he introduced to describe the extent of an IFS attractor.

In this thesis, we require the initial compact subset B to be not only the bounding ball (not necessarily a tight one), but also the set, for which the following inclusion holds:

$$\mathbb{T}(B) \subseteq B,$$

because with this property we have the continuous approximation of the attractor exterior. Each element of this sequence is included in the next one.

We have chosen the method suggested in [Gen92] because it satisfies all the requirements and also because of it is simple to implement. In addition to

efficiency, this approach allows us to achieve better convergence as it computes the minimum bounding ball for a given accuracy.

A.2 Bounding ball for the attractor of a CIFS

Here we describe a method for computing a set of bounding balls for the attractor of a CIFS. For each CIFS attractor we thus find a bounding ball (not necessarily a tight one) centered in a point of the attractor.

A *path* in the CIFS graph is a sequence of the automaton states $\{q_k\}_{k=1}^{l+1}$ which satisfies the following conditions:

- $\forall k = 1, \cdots, l \; \exists i : q_{k+1} = \delta(q_k, i)$
- $q_i \neq q_j \; \forall i, j, \; i \neq j$

Here $l \in \mathbb{N}$ is called *length* of the path and denotes a number of transformations applied to reach the state q_{l+1}.

A *simple cycle* in the CIFS graph is a path which is closed on its first state:

- $q_1 = q_{l+1}$

An operator of the simple cycle is a composite of the transformations $T_i^{q_k}$ between the cycle states. Let us denote the simple cycle from the state a (i.e. $q_1 = q_{l+1} = a$) by ξ^a.

Finally, let θ^a be a set of all simple cycles from the state a.

The CIFS system defines a family of attractors associated with the states. Since the attractors are mutually recursively defined, we construct the balls by depth-first searching in the CIFS control graph.

Let us consider an example of a CIFS attractor, illustrated in figure A.1. The system is described by an automaton with the initial state a and several simple cycles.

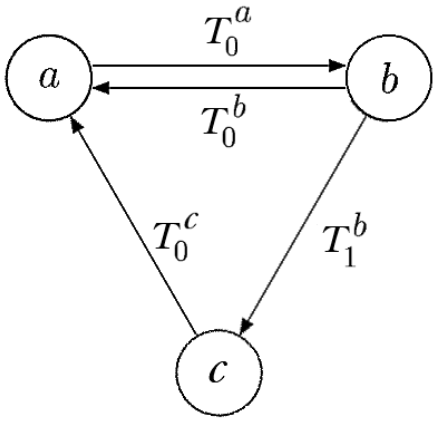


Figure A.1: An example of a CIFS. The system is described by an automaton with the initial state a and several simple cycles

The attractor $\mathcal{A}^a$ is defined as follows:

$$\begin{aligned}\mathcal{A}^a &= \bigcup_{i\in\Sigma^a} T_i^a(\mathcal{A}^{\delta(a,i)}) \\ &= T_0^a(\mathcal{A}^b) \\ &= T_0^aT_0^b(\mathcal{A}^a) \ \cup \ T_0^aT_1^b(\mathcal{A}^c) \\ &= T_0^aT_0^b(\mathcal{A}^a) \ \cup \ T_0^aT_1^bT_0^c(\mathcal{A}^a)\end{aligned}$$

This CIFS is thus equivalent to the IFS with two transformations $T_0^aT_0^b$ and $T_0^aT_1^bT_0^c$.

We compute the ball $B^a = B(c^a, r^a)$ satisfying the following inclusions:

$$B^a \supseteq T_0^aT_0^b(B^a),$$

$$B^a \supseteq T_0^aT_1^bT_0^c(B^a).$$

The radius r^a is computed as follows:

$$r^a = \max_{\xi^a\in\theta^a} \frac{d(T_{\xi^a}(c^a), c^a)}{1 - s_{\xi^a}},$$

where T_{ξ^a} is an operator of the simple cycle ξ^a and s_{ξ^a} is its contraction coefficient. A ball center c^a is computed as a fixed point of T_{ξ^a}, where ξ^a is any cycle in θ^a.

It is convenient to compute the radius r^a for each simple cycle $\xi^a \in \theta^a$ separately and then take the maximum one.

Let us consider another example of a CIFS attractor, illustrated in figure A.2. There is a path $\{q_1 = \natural, q_2, \ldots, q_l, q_{l+1} = a\}$ to the several simple cycles $\xi_i^a \in \theta^a$. As described above, we firstly compute a ball B^a using the simple cycle transformations. Secondly, we compute the balls B^{q_i} recursively verifying the following inclusions:

$$B(c^{q_i}, r^{q_i}) = B^{q_i} \supseteq T^{q_i}(B^{q_{i+1}}), \ \forall i = l, \ldots, 1.$$

That is, we compute the radius r^{q_i} as follows:

$$r^{q_i} = \max(r^{q_i}, d(T^{q_i}(c^{q_{i+1}}), c^{q_i}) + r^{q_{i+1}} \cdot s(T^{q_i})),$$

If the ball center c^{q_i} has not defined yet, we compute it as the image of $c^{q_{i+1}}$.

The attractor of any CIFS is formed by schemes shown in figure A.2. Since each simple cycle is considered separately, we can then find all schemes by depth-first searching in the CIFS control graph. For each simple cycle ξ^a found we firstly compute the ball B^a, and secondly we compute the balls through the path to this cycle.

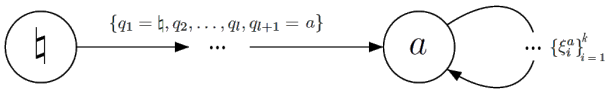


Figure A.2: An automaton of the CIFS. There is a path $\{q_1 = \natural, q_2, \ldots, q_l, q_{l+1} = a\}$ to the several simple cycles $\xi_i^a \in \theta^a$.

To perform a depth-first search in the control graph, we use a recursive function. The algorithm starts off by calling this function for the initial state $\natural$. When the whole graph is gone through, all the required radii and centres will be defined.

Given a CIFS with an automaton (Σ, Q, δ) and an initial state $\natural$. Here is the described initialization method implemented on pseudocode:

```
// To perform a depth-first search in the automaton's
// graph, we use a recursive function. Algorithm starts
// off by calling this function with q = ♮ and empty lists
Function CheckState
Input:
q is a current state q ∈ Q
states is a list of traversed states
t is a list of applied transformations
Output: initial bounding balls B^q = B(c^q, r^q) ∀q ∈ Q
Body:
{
    // We are looking for a cycle
    index ← states.indexof(q)
    if index = −1
    {
        // It is not a cycle, continue to search
        for each i ∈ Σ^q
        {
            CheckState(δ(q, i), states + [q], t + [T_i^q])
        }
        // Whole branch has been traversed and
        // the required radii and centres were defined
        exit
    }
    // We found a cycle. It is a tail of the list t
    // Calculate a cycle length
    cLen ← t.count−index
    // Extend the list t on cycle transforms
    // to simplify the following loop
    t ← t + t.sublist(index)
    // Change the cycle state balls
    for i = states.count−1 to index
    {
        w ← states[i]
        t_cycle ← ∏_{j=i}^{i+cLen} t[j]
        if c^w is not defined
        then c^w ← a fixed point of t_cycle
        else r^w ← max(r^w, d(t_cycle(c^w), c^w) / (1 − s_cycle))
    }
    // Change the balls on the path to simple cycle
    for i = index − 1 to 0
    {
        w ← states[i]
        y ← states[i + 1]
        if c^w is not defined
        then c^w ← t[i](c^y)
        else r^w ← max(r^w, d(t[i](c^y), c^w) + r^y · s(t[i]))
    }
}
```

Bibliography

[Bar88] M. Barnsley. *Fractals everywhere*. Academic Press Professional, Inc., San Diego, CA, USA, 1988.

[BD85] M. F. Barnsley and S. Demko. Iterated function systems and the global construction of fractals. *Proceedings of the Royal Society of London*, A399:245 – 257, 1985.

[Ben09] H. Bensoudane. *Étude différentielle des formes fractales*. PhD thesis, Université de Bourgogne, October 2009. Jury: M. Barnsley, S. Hahmann, E. Tosan, J. Levy Vehel, D. Michelucci, M. Sabin, M. Neveu, C. Gentil.

[BGN08] H. Bensoudane, C. Gentil, and M. Neveu. The local fractional derivative of fractal curves. *Signal Image Technology and Internet Based Systems, SITIS '08*, pages 422 – 429, 2008.

[BHS08] M. Barnsley, J. Hutchinson, and O. Stenflo. V-variable fractals: Fractals with partial self similarity. *Advances in Mathematics*, 218(6):2051 – 2088, 2008.

[BKZ01] H. Biermann, D. Kristjansson, and D. Zorin. Approximate boolean operations on free-form solids. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, pages 185 – 194, New York, NY, USA, 2001. ACM.

[BS87] M. F. Barnsley and A. Sloan. Methods and apparatus for image compression by iterated function system. United States Patent 4,941,193, 1987.

[Cas12] T. J. Cashman. Beyond catmull-clark? a survey of advances in subdivision surface methods. *Comput. Graph. Forum*, 31(1):42 – 61, February 2012.

[CC98] E. Catmull and J. Clark. Seminal graphics. chapter Recursively generated B-spline surfaces on arbitrary topological meshes, pages 183 – 188. ACM, New York, NY, USA, 1998.

[Chi88] H. Chiyokura. *Solid modelling with Designbase: theory and implementation*. Addison-Wesley, 1988.

[DeR89] T. D. DeRose. Theory and practice of geometric modeling. chapter A coordinate-free approach to geometric programming, pages 291 – 305. Springer-Verlag New York, Inc., New York, NY, USA, 1989.

[DKT98] T. DeRose, M. Kass, and T. Truong. Subdivision surfaces in character animation. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '98, pages 85 – 94, New York, NY, USA, 1998. ACM.

[DLG90] N. Dyn, D. Levine, and J. A. Gregory. A butterfly subdivision scheme for surface interpolation with tension control. *ACM Trans. Graph.*, 9(2):160 – 169, April 1990.

[Doo78] D. Doo. A subdivision algorithm for smoothing down irregularly shaped polyhedrons. In *in Proceedings on Interactive Techniques in Computer Aided Design*, pages 157 – 165. IEEE, 1978.

[Dud07] J. Duda. Analysis of the convex hull of the attractor of an ifs. http://arXiv/0710.3863, 2007.

[Eda95] A. Edalat. Dynamical systems, measures, and fractals via domain theory. *Information and computation*, 120:3248, 1995.

[Eda97] A. Edalat. Domains for computation in mathematics, physics and exact real arithmetic. *Bulletin of Symbolic Logic*, 3:401–452, 1997.

[EL02] A. Edalat and A. Lieutier. Foundation of a computable solid modelling. *Theoretical Computer Science*, 284:319345, 2002.

[Fis95] Y. Fisher. Fractal image compression, theory and application. *Springer-Verlag*, 1995.

[Gal00] J.H. Gallier. *Curves and Surfaces in Geometric Modeling: Theory and Algorithms*. The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling. Morgan Kaufmann Publishers, 2000.

[Gen92] C. Gentil. *Les fractales en synthése d'images: le modèle IFS*. PhD thesis, Université LYON I, March 1992. Jury: D. Vandorpe, P. Chenin, J. Mazoyer, J. P. Reveilles, J. Levy Vehel, M. Terrenoire, E. Tosan.

[GHT11] G. Gouaty, H. Hnaidi, and E. Tosan. Vers un modeleur basé sur un formalisme itératif et sur la géométrie fractale. Grenoble, 2011. Journées du Groupe de Travail en Modélisation Géométrique.

[GN13] C. Gentil and M. Neveu. Mixed-aspect fractal surfaces. *Comput. Aided Des.*, 45(2):432 – 439, February 2013.

[Gol02] R. Goldman. On the algebraic and geometric foundations of computer graphics. *ACM Trans. Graph.*, 21(1):52 – 86, January 2002.

[Gou09] G. Gouaty. *Modélisation géométrique itérative sous contraintes*. PhD thesis, École polytechnique fédérale de Lausanne EPFL, January 2009. Jury: I. Smith, Y. Weinand, E. Tosan, C. Gentil, P. Lienhardt, D. Thalmann.

[GS01] E. Grinspun and P. Schrder. Normal bounds for subdivision-surface interference detection. In *In Proc. of IEEE Scientific Visualization, IEEE*, pages 333 – 340. IEEE, 2001.

[GTWS10] G. Gouaty, E. Tosan, Y. Weinand, and I. Stotz. Modélisation géométrique itérative et approche b-rep. Dijon, 2010. Journées du Groupe de Travail en Modélisation Géométrique.

[GVS12] J. Gomes, L. Velho, and M. C. Sousa. *Design and Implementation of 3D Graphics Systems*. Taylor & Francis, 2012.

[Han03] B. Han. Classification and construction of bivariate subdivision schemes. In *Curve and Surface Fitting: Saint-Malo 2002*, pages 187 – 197. Nashboro Press, 2003.

[HD91] J. C. Hart and T. A. DeFant. Efficient anti-aliased rendering of 3d linear fractals. *Computers and Graphics*, 25(4):91 – 100, 1991.

[Hof89] C.M. Hoffmann. *Geometric and solid modeling: an introduction*. Morgan Kaufmann series in computer graphics and geometric modeling. Morgan Kaufmann, 1989.

[HPS92] D. H. Hepting, P. Prusinkiewicz, and D. Saupe. Rendering methods for iterated function systems. In *Fractals in the Fundamental and Applied Sciences*, pages 183 – 224, 1992.

[HR91] H. Hagen and D. Roller. *Geometric modeling: methods and applications*. Symbolic computation: Computer graphics–systems and applications. Springer-Verlag, 1991.

[Hut81] J. Hutchinson. Fractals and self-similarity. *Indiana University Journal of Mathematics*, 30:713 – 747, 1981.

[IDS04] I. P. Ivrissimtzis, N. A. Dodgson, and M. A. Sabin. A generative classification of mesh refinement rules with lattice transformations. *Comput. Aided Geom. Des.*, 21(1):99 – 109, January 2004.

[JS09] D. Jiang and N. F. Stewart. Numerical validation in current hardware architectures. chapter Robustness of Boolean Operations on Subdivision-Surface Models, pages 161 – 174. Springer-Verlag, Berlin, Heidelberg, 2009.

[KKM97] J. Keyser, S. Krishnan, and D. Manocha. Efficient and accurate b-rep generation of low degree sculptured solids using exact arithmetic. In *Proceedings of the fourth ACM symposium on Solid modeling and applications*, SMA '97, pages 42 – 55, New York, NY, USA, 1997. ACM.

[KKM99] J. Keyser, S. Krishnan, and D. Manocha. Efficient and accurate b-rep generation of low degree sculptured solids using arithmetic: Ii-computation. *Comput. Aided Geom. Des.*, 16(9):861 – 882, October 1999.

[KLSW99] R. Kenyon, J. Li, R. S. Strichartz, and Y. Wang. Geometry of self-affine tiles ii. *Indiana University Mathematics Journal*, 48:25 – 42, 1999.

[KM97] S. Krishnan and D. Manocha. An efficient surface intersection algorithm based on lower-dimensional formulation. *ACM Trans. Graph.*, 16(1):74 – 106, January 1997.

[KMDZ09] D. Kovacs, J. Mitchell, S. Drone, and D. Zorin. Real-time creased approximate subdivision surfaces. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, I3D '09, pages 155 – 160, New York, NY, USA, 2009. ACM.

[Kob96] L. Kobbelt. Interpolatory subdivision on open quadrilateral nets with arbitrary topology. In *Computer Graphics Forum*, pages 409 – 420, 1996.

[Kob00] L. Kobbelt. Root of 3-subdivision. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '00, pages 103 – 112, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[Lan04] S. Lanquetin. *Étude des surfaces de subdivision: intersection, precision et profondeur de subdivision*. PhD thesis, Université de Bourgogne, October 2004. Jury: S. Akkouche, M. Daniel, S. Hahmann, F. Merienne, G. Morin, M. Neveu.

[LC07] S. Lai and F. Cheng. Robust and error controllable Boolean operations on free-form solids represented by Catmull-Clark subdivision surfaces. *Computer-Aided Design and Applications*, 4(1-6):487 – 496, 2007.

[Lev99] A. Levin. Combined subdivision schemes for the design of surfaces satisfying boundary conditions. *Computer Aided Geometric Design*, 16:345 – 354, 1999.

[LFKN03] S. Lanquetin, S. Foufou, H. Kheddouci, and M. Neveu. A graph based algorithm for intersection of subdivision surfaces. In Vipin Kumar, Marina L. Gavrilova, Chih Jeng Kenneth Tan, and Pierre L'Ecuyer, editors, *ICCSA (3)*, volume 2669 of *Lecture Notes in Computer Science*, pages 387 – 396. Springer, 2003.

[LG00] U. Labsik and G. Greiner. Interpolatory sqrt(3)-subdivision. *Comput. Graph. Forum*, 19(3), 2000.

[LH03] O.S. Lawlor and J.C. Hart. Bounding recursive procedural models using convex optimization. In *Computer Graphics and Applications, 2003. Proceedings. 11th Pacific Conference on*, pages 283 – 292, 2003.

[Lie99] A. Lieutier. Invited paper: Computable partial solids and voxels sets. In Gilles Bertrand, Michel Couprie, and Laurent Perroton,

editors, *Discrete Geometry for Computer Imagery, 8th International Conference, DCGI 99, Marne-la-Vallee, France, March 17-19, 1999, Proceedings*, volume 1568 of *Lecture Notes in Computer Science*, pages 349 – 360. Springer, 1999.

[Lin68] A. Lindenmayer. Mathematical models for cellular interaction in development: Parts i and II. *Journal of Theoretical Biology*, 18:280 – 315, 1968.

[LLS01] N. Litke, A. Levin, and P. Schrder. Trimming for subdivision surfaces. *Computer Aided Geometric Design*, 18(5):463 – 481, 2001.

[Loo87] C. Loop. Smooth Subdivision Surfaces Based on Triangles. Master's thesis, The University of Utah, August 1987.

[Mar03] T. Martyn. Tight bounding ball for affine ifs attractor. *Computers & Graphics*, 27(4):535 – 552, 2003.

[Mar04] T. Martyn. A new approach to morphing 2d affine ifs fractals. *Computers & Graphics*, 28(2):249 – 272, 2004.

[Mar09a] T. Martyn. The attractor-wrapping approach to approximating convex hulls of 2d affine ifs attractors. *Computers & Graphics*, 33(1):104 – 112, 2009.

[Mar09b] T. Martyn. The smallest enclosing disc of an affine ifs fractal. *Fractals*, 17(03):269 – 281, 2009.

[Mas97] P. R. Massopust. Fractal functions and their applications. *Chaos, Solitons & Fractals*, 8(2):171 – 190, 1997.

[MGLS12] A. Mishkinis, C. Gentil, S. Lanquetin, and D. Sokolov. Approximate convex hull of affine iterated function system attractors. *Chaos, Solitons & Fractals*, 45(11):1444 – 1451, 2012.

[MW88] R. D. Mauldin and S. C. Williams. Hausdorff dimension in graph directed constructions. *Transactions of the American Mathematical Society*, 309(2):811 – 829, 1988.

[PGSL13] S. Podkorytov, C. Gentil, D. Sokolov, and S. Lanquetin. Geometry control of the junction between two fractal curves. *Computer-Aided Design*, 45(2):424 – 431, 2013. Solid and Physical Modeling 2012.

[PH94] P. Prusinkiewicz and M. Hammel. Language-restricted iterated function systems, koch constructions, and l-systems. *SIGGRAPH'94 Course Notes*, 1994.

[PL90] P. Prusinkiewicz and A. Lindenmayer. The algorithmic beauty of plants. *Springer-Verlag*, 1990.

[PR97] J. Peters and U. Reif. The simplest subdivision scheme for smoothing polyhedra. *ACM Trans. Graph.*, 16(4):420 – 431, 1997.

[PS88] P. Prusinkiewcz and G. Sandness. Koch curves as attractors and repellers. *Computer Graphics and Applications, IEEE*, 8(6):26–40, 1988.

[PS04] J. Peters and L. J. Shiue. Combining 4- and 3-direction subdivision. *ACM Trans. Graph.*, 23(4):980 – 1003, October 2004.

[PSS10] P. Prusinkiewicz, M. Shirmohammadi, and F. Samavati. L-systems in geometric modeling. In *DCFS*, pages 3 – 14, 2010.

[PSSK02] P. Prusinkiewicz, F. Samavati, C. Smith, and R. Karwowski. L-system description of subdivision curves. *INTERNATIONAL JOURNAL OF SHAPE MODELING*, 9:41 – 59, 2002.

[Rao04] P. N. Rao. *CAD/CAM: Principles and Applications.* Mechanical engineering series. McGraw-Hill Education (India) Pvt Limited, 2004.

[Req77] A.A.G. Requicha. *Mathematical Models of Rigid Solid Objects, TM-28.* Technical memorandum. Production Automation Project, University of Rochester, 1977.

[Ric96] J. Rice. Spatial bounding of self-affine iterated function system attractor sets. In Wayne A. Davis and Richard M. Bartels, editors, *Proceedings of the Graphics Interface 1996 Conference, May 22-24, 1996, Toronto, Ontario, Canada*, pages 107 – 115. Canadian Human-Computer Communications Society, 1996.

[RLG+95] C. Roll, E. Lutton, P. Glevarec, J. Lévy-Véhel, and G. Cretin. Mixed ifs: resolution of the inverse problem using genetic programming. *Complex Systems*, 5(5):375 – 398, 1995.

[RS97] A. Rappoport and S. Spitz. Interactive boolean operations for conceptual design of 3-d solids. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '97, pages 269 – 278, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[SL03] J. Stam and C. Loop. Quad/triangle subdivision. *Comput. Graph. Forum*, 22(1):79–86, 2003.

[SLG05] S. Schaefer, D. Levin, and R. Goldman. Subdivision schemes and attractors. In *Proceedings of the third Eurographics symposium on Geometry processing*, SGP '05, Aire-la-Ville, Switzerland, Switzerland, 2005. Eurographics Association.

[SS06] A. Severn and F. Samavati. Fast intersections for subdivision surfaces. In *Proceedings of the 6th international conference on Computational Science and Its Applications - Volume Part I*, ICCSA'06, pages 91 – 100, Berlin, Heidelberg, 2006. Springer-Verlag.

[Sta98] J. Stam. Exact evaluation of catmull-clark subdivision surfaces at arbitrary parameter values. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '98, pages 395 – 404, New York, NY, USA, 1998. ACM.

[SW98] R. S. Strichartz and Y. Wang. Geometry of self-affine tiles i. *Information science research, AT&T LABS-RESEARCH, FLORHAM PARK, NJ 07932 SCHOOL OF MATHEMATICS, GEORGIA INSTITUTE OF TECHNOLOGY, ATLANTA, GA 30332 E-MAIL: WANG@MATH.GATECH.EDU*, 48:1 – 23, 1998.

[SW05] S. Schaefer and J. Warren. On c^2 triangle/quad subdivision. *ACM TRANS. GRAPHICS*, 24(1):28 – 36, 2005.

[TBSG+06] E. Tosan, I. Bailly-Salins, G. Gouaty, I. Stotz, P. Buser, and Y. Weinand. Une modélisation géométrique itérative basée sur les automates. pages 155 – 169. LURPA EA 138, AFIG, GDR ALP. 2, Journées du Groupe de Travail en Modélisation Géométrique, March 2006.

[Tho96] J. Thollot. *Extension du modèle IFS pour une géométrie fractale constructive*. PhD thesis, September 1996.

[Tos06] E. Tosan. Surfaces fractales définies par leurs bords. Grenoble, 2006. Journées Courbes, surfaces et algorithmes.

[TT93] J. Thollot and E. Tosan. Construction of fractales using formal languages and matrices of attractors. In Harold P. Santos, editor, *Conference on Computational Graphics and Visualization Techniques, Compugraphics*, pages 74 – 78, Alvor, Portugal, 1993.

[TZTV97] J. Thollot, C. E. Zair, E. Tosan, and D. Vandorpe. Modeling fractal shapes using generalizations of IFS techniques. In Jacques Lévy Véhel, Evelyne Lutton, and Claude Tricot, editors, *Fractals in Engineering*, Arcachon, France, 1997. Springer.

[VZ01] L. Velho and D. Zorin. 4-8 subdivision. *Comput. Aided Geom. Des.*, 18(5):397 – 427, June 2001.

[WP04] X. Wu and J. Peters. Interference detection for subdivision surfaces. *Comput. Graph. Forum*, 23(3):577 – 584, 2004.

[WW02] J. Warren and H. Weimer. *Subdivision methods for geometric design : a constructive approach*. The Morgan Kaufmann series in computer graphics and geometric modeling. Morgan Kaufmann, San Francisco, 2002.

[ZT96] C. E. Zair and E. Tosan. Fractal modeling using free form techniques. *EUROGRAPHICS'96*, 15(3), 1996.

[ZTA08] I. Zammouri, E. Tosan, and B. Ayeb. A theoretical framework mapping Grammar based systems and Fractal description. *Fractals*, 16(4):389 – 401, December 2008.

Résumé :

La définition de formes par ces procédés itératifs génère des structures avec des propriétés spécifiques intéressantes : rugosité, lacunarité. . . . Cependant, les modèles géométriques classiques ne sont pas adaptés à la description de ces formes.
Dans le but de développer un modeleur itératif pour concevoir des objets fractals décrits à l'aide du BCIFS, nous avons développé un ensemble d'outils et d'algorithmes génériques qui nous permettent d'évaluer, de caractériser et d'analyser les différentes propriétés géométriques (la localisation, le calcul de l'enveloppe convexe, de la distance à partir d'un point, etc) de fractals. Nous avons identifié les propriétés des opérations standards (intersection, union, offset, . . .) permettant de calculer une approximation d'image des fractales et de plus d'optimiser ces algorithmes d'approximation.
Dans certains cas, il est possible de construire un CIFS avec l'opérateur de HUTCHINSON généralisé dont l'attracteur est suffisamment proche du résultat de l'opération par rapport à la métrique de Hausdorff. Nous avons développé un algorithme générique pour calculer ces CIFS pour une précision donnée. Nous avons défini la propriété d'auto-similarité de l'opération, qui définie un ensemble de transformations utilisé dans un système itératif résultant.
Pour construire un CIFS exact de l'image, si il existe, il faut prouver tous les similitudes nécessaires manuellement. Nous explicitons également la condition de l'opération, quand le résultat peut être représenté par un IFS avec un opérateur de HUTCHINSON généralisé. Dans ce cas, il n'est que cette condition à prouver manuellement.

Mots-clés : Fractal, modélisation géométrique, conception assistée par ordinateur, géométrie algorithmique, informatique graphique

Abstract:

Defining shapes by iteration allows us to generate new structures with specific properties (roughness, lacunarity), which cannot be achieved with classic modelling.
For developing an iterative modeller to design fractals described by a BCIFS, we developed a set of tools and algorithms that permits one to evaluate, to characterize and to analyse different geometric properties (localisation, convex hull, volume, fractal dimension) of fractals. We identified properties of standard CAD operations (intersection, union, offset, . . .) allowing us to approximate them for fractals and also to optimize these approximation algorithms.
In some cases, it is possible to construct a CIFS with generalised HUTCHINSON operator, whose attractor is close enough to the operation result with respect to the HAUSDORFF metric. We introduced a generic algorithm to compute such CIFS for a given accuracy. We defined the self-similarity property of the operation defining a set of transformations, which are used in the output iterative system.
In order to construct an exact CIFS of the image, if it exists, we must prove all the necessary similarities manually. We explicit also the condition of the operation to be represented by an IFS with a generalised HUTCHINSON operator. In this case, only this condition should be proved manually.

Keywords: fractal, geometric modelling, computer-aided design, computational geometry, computer graphics

SPIM

■ École doctorale SPIM - Université de Bourgogne/UFR ST BP 47870 F - 21078 Dijon cedex
■ tél. +33 (0)3 80 39 59 10 ■ ed-spim@univ-fcomte.fr ■ www.ed-spim.univ-fcomte.fr

UB UNIVERSITÉ DE BOURGOGNE