Part I

Algorithmic Modeling

Luiz Velho

IMPA – Instituto de Matemática Pura e Aplicada
and
University of Toronto

Abstract

Algorithmic modeling (or procedural modeling) is a recent research area in computer graphics which encompasses several methods extending traditional geometric modeling.

Man-made objects are mostly constituted of connected rigid bodies that are well represented by geometric solids. On the other hand, there are in nature extremely complex objects that can have a combination of the following characteristics: irregular geometry, fuzzy boundaries, inhomogeneous material, anisotropic properties. As examples of objects in this category, we find: terrain, plants, clouds, liquids, fire and the living creatures. These objects are part of our environment and, thus, are very familiar to all of us. For this reason, it is even more difficult to represent in the computer their form (visualization) and movement (animation).

Despite its effectiveness to describe manufactured objects, geometric modeling techniques are, in general, inadequate to fully describe organic forms and other natural phenomena. One of the main motivations of algorithmic modeling has been the challenge to capture the shape and behavior of complex objects from the real world.

This part introduces the theory and techniques of algorithmic modeling from the point of view of a conceptual and integrated framework.

Chapter 1

Models and Machines

This chapter investigates mathematical models that can be used to describe complex shapes. Our goal is to define an abstract framework which extends geometric modeling and provides greater expressive power, allowing for a representation of these objects in the computer.

While simple shapes can be described by means of analytical expressions, complex ones may require a more powerful mathematical description. The concept of a mathematical function, i.e. an entity that takes a value as input, executes a procedure and outputs another value, give us such a mechanism. The procedure is the embodiment of an algorithm and consists of all steps that are necessary to perform the computation.

Since we are interested in objects from the real world, we must be able, essentially, to deal with point sets embedded in the continuous three dimensional space. Consequently, we need functions to compute with the real numbers. As we will see, a mathematical model for this type of function is given by a machine over the field of reals.

It is important to note that the sets computable by this machine are the denumerable family of semi-algebraic sets. Based on the results in Part , this indicates that our algorithmic model indeed contains the geometric models. Also, by the Stone-Weierstrass theorem, these sets provide good approximations to most reasonable shapes found in nature.

1.1 Shape Modeling

As we have seen in Part **??**, there are two forms of geometric specification: direct and indirect.

The direct form defines the geometry explicitly by a parametric equation $x = f(u)$, where $x \in \mathbb{R}^3$ is a point of the ambient space and $u \in U \subset \mathbb{R}^k$ is a point of a subset of a parameter space with the same dimension k as the object. The mapping $f : \mathbb{R}^k \rightarrow \mathbb{R}^3$ makes it possible to generate all the points of the object, as long as U is known.

The indirect form defines the geometry implicitly by an equation $f(x) = c$, where x is a point of the three dimensional space and c is a constant or an interval of the real line. The mapping $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ makes it possible to determine which points of the ambient space belong to the object [1]

.

[1] Note that the characteristic function of the object is trivially defined from the implicit equation:

$$\chi(x) = \begin{cases} 1 & \text{if } f(x) = c \\ 0 & \text{otherwise} \end{cases}$$

These two forms of defining geometry are illustrated in Figure 1.1.

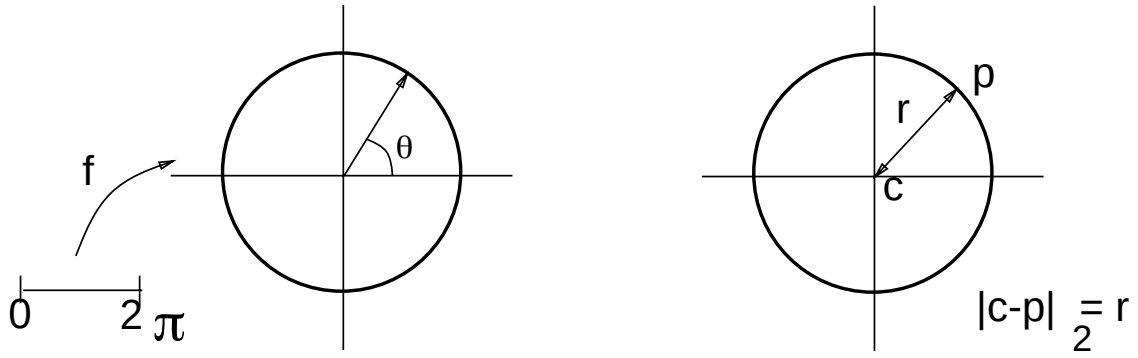

FIGURE 1.1: Direct and indirect forms

In summary, geometric models are defined, in one way or another, by the above equations or some combination of these equations. In this case, the functions f are restricted to analytic expressions (actually, in practice, we work with low degree polynomials, such as quadrics or cubics).

From the previous observations, it becomes clear that geometric modeling can be extended naturally if the restrictions imposed on f are eliminated and arbitrary functions are allowed. This general form of shape modeling, thus, requires a mathematical model of functions over the real numbers. More specifically, in the parametric case f is a function of $\mathbb{R}^k$ into $\mathbb{R}^3$ and, in the implicit case, f is a function of $\mathbb{R}^3$ into $\mathbb{R}$.

1.2 Machines over the Reals

A model for machines over the real numbers comes from the theory of computation and complexity of continuous algorithms (Smale, 1990). This new interdisciplinary field integrates ideas from classical computer theory with methods from numerical analysis in order to develop a formal treatment of problems defined over continuous domains. Central to this discipline is the concept of a universal machine over an ordered ring, developed by (Blum, Shubb and Smale, 1989), which plays in the theory of continuous computation the same role as the Turing machine in the theory of discrete computation (where the ring is $\mathbb{Z}$).

Definition 1.1 *:* A finite dimensional machine M over a ring R consists of three spaces (input space $\mathcal{I}$, state space $\mathcal{S}$ and output space $\mathcal{O}$), together with a finite directed connected graph with four types of nodes (input, output, computation and branch), with associated maps and labeled $1, \ldots, N$.

The spaces $\mathcal{I}$, $\mathcal{S}$ and $\mathcal{O}$ are, respectively, of the form R^l, R^n and R^m, where R^k denotes the direct sum of R with itself k times.

The nodes of M are of the following types:

Input node: has no incoming edge and only one outgoing edge, β_1. It is associated with a linear injective map $I : \mathcal{I} \rightarrow \mathcal{S}$.

Output node: has no outgoing edges. It is associated with a linear map $O : \mathcal{S} \to \mathcal{O}$.

Computational node: has a single outgoing edge. It is associated with a polynomial map $g : \mathcal{S} \to \mathcal{S}$.

Branch node: has two outgoing edges, β_k^- and β_k^+. It is associated with a polynomial $h : \mathcal{S} \to R$, which specifies the next node according to the conditions $h(x) < 0$ and $h(x) \geq 0$.

Figure 1.2 shows a diagram of the machine M and its nodes.

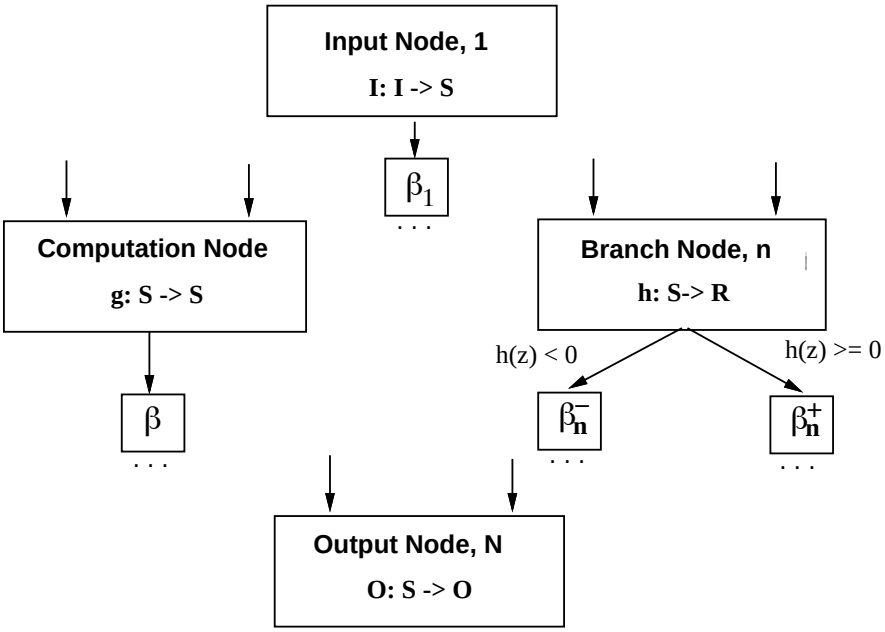

FIGURE 1.2: The nodes of a machine M over R (after Blum)

Let $\mathcal{N} = \{1, \ldots, N\}$ be the set of nodes of M, where 1 is the input node and N is the output node (assuming that there is only one output node). A machine constructed as above is said to be in *normal form.*

Proposition 1.1 *Any machine M over R has an equivalent one in normal form.*

The space of node/state pairs $\mathcal{N} \times \mathcal{S}$ is called the *full state space* of M. The machine is associated with the *computing endomorphism*

$$H : \mathcal{N} \times \mathcal{S} \to \mathcal{N} \times \mathcal{S}$$

of the full state space to itself. H maps each node/state pair (n, x) to the unique *next node / next state* pair $(\beta(n, x), g(n, x))$, where β gives the next node and g the new state. This scheme is determined by the graph of M and the associated maps of its nodes.

The computing endomorphism is a very important tool that, among other things, can be used to define the *input-output map* φ_M of a machine M.

The inner workings of M are revealed by φ_M in the following way: A value $y \in \mathcal{I}$ is input into M by $x_0 = I(y)$. Then, M performs the sequence of computation steps, $x_1 = H(x_0), \ldots, x_k = H(x_{k-1}), \ldots$, until a node/state pair (N, x_T) is produced. If M ever halts, it outputs a value $z = O(x_T)$.

When the above sequence (x_k) is finite, M stops on input y in time T with output $O(x_T)$ and $\varphi_M(y)$ describes a *halting computation*. When the sequence (x_k) is infinite, M does not halt on input y, and $\varphi_M(y)$ is not defined.

The set Ω_M of all points $y \in \mathcal{I}$ on which M halts is called the *halting set* of M.

A map $\varphi : Y \subset R^l \to R^m$ is *computable over* R if and only if there is a machine M over R such that $\Omega_M = Y$ and $\varphi_M = \varphi$ for all $y \in Y$ (i.e. M halts on every element of the domain of φ). In such case, M is said to *compute* φ.

Two machines M_1 and M_2 are called *equivalent* if they compute the same map φ.

It is natural to interpret M as a discrete dynamical system. In this context, we study the orbits of initial points $z_0 = (1, I(y))$ under iterates of H. The qualitative behavior of M can be analyzed through the phase portrait of φ_M. In this way, although the graph of M may be quite complicated, we can represent it in a simple way, as shown in Figure 1.3.

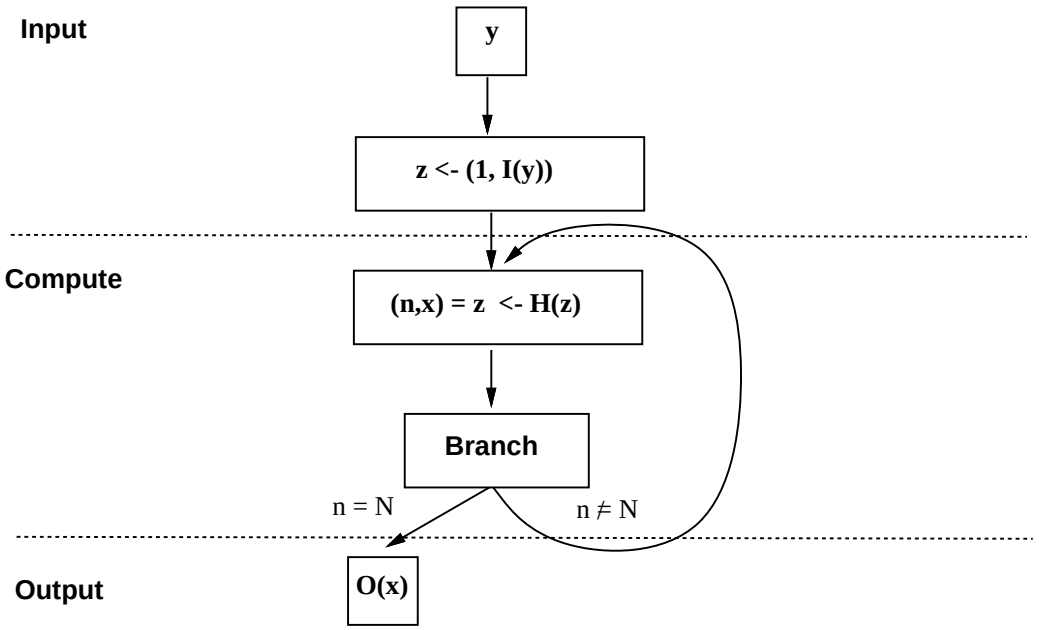

FIGURE 1.3: Canonical schemata for a machine M (after Blum)

1.3 Decidable Point Sets

A set S is *decidable* if its characteristic function χ_S is computable, otherwise it is *undecidable*, (Blum and Smale, 1992). Intuitively, a set is decidable, relative to a universe U, if there is an effective procedure that decides whether or not any given element $u \in U$ belongs to the set.

Proposition 1.2 (Blum-Smale) *A set $S \subset R^l$ is "decidable over R" if and only if both S and its complement S' are halting sets of machines over R.*

Essentially, we need to construct for S a *decision machine* M^* which computes

$$\chi : R^l \to \{0, 1\}, \quad \chi(y) = 1 \text{ iff } y \in S.$$

This can be done by connecting together two machines M and M', which, respectively, have as halting sets S and its complement S' on R^l. The input is fed simultaneously into M and M'. By construction, only one of them will halt. If M halts, the output is 1; if M' halts the output is 0 (see Figure 1.4).

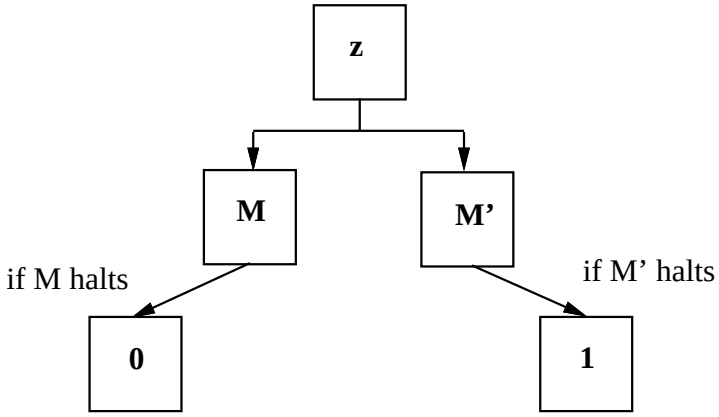

FIGURE 1.4: Decision Machine

Definition 1.2 *A set S is called "semi-decidable" (undecidable) if it is the halting set of a machine M over R, but its complement S' is not.*

This means that, the decision machine M^* on input y will output 1 if and only if y is in S; but, if y is not in S there are no guarantees that M^* will halt.

Proposition 1.3 (Blum-Shub-Smale) *The halting set of a machine M over R is a disjoint countable union of semi-algebraic sets. The input-output map φ_M is a piecewise polynomial map.*

The proof of this proposition is somewhat involved, but the basic idea is simple. Since, besides input and output nodes, M has only computation and branch nodes. the path of any halting computation can be streamlined to a series of polynomial maps. The halting set $\Omega_M = \cup V_\gamma$, where V_γ is defined by polynomial inequalities of the type

$$g_1(\dots g_n(I(y))) < 0.$$

For more details we refer the reader to (Blum, Shubb and Smale, 1989).

The above results clearly establish, in this theory, a link between the notions of decidability and computability. Furthermore, they reinforce the conclusions of Part that semi-algebraic sets are natural candidates to be used in the representation in geometric and solid modeling.

Also, from the direct relationship between the characteristic function of a point set and the specification of a shape by an implicit equation, $f(x) = c$, it follows that the computability of f implies the decidability of $f^{-1}(c)$. The validity of this assertion for the parametric case is immediate because, in that case, the halting set S is the entire domain of the map.

1.4 The Machine at Work

Now we will show through some examples, the application of this model of computation over the reals to define geometric objects.

1.4.1 Parametric Surfaces (Local)

In a parametric surface patch, M has only one computation node and its associated spaces are as follows: $\mathcal{I} = U \subset \mathbb{R}^2$ and $\mathcal{S} = \mathcal{O} = \mathbb{R}^3$. When the surface is smooth, $I(y)$ is the natural injection, $g(x_0)$ is a diffeomorphism and $O(x_1)$ is the identity map.

Example 1.1 (Tensor Product Bezier Patch) A cubic Bezier patch is defined by a 4×4 net of control points $p_{i,j} \in \mathbb{R}^3$. The parameter space U is the unit square $[0,1] \times [0,1]$, g is a tensor product Bezier polynomial

$$g(u,v,0) = \sum_{i=0}^{3} \sum_{j=0}^{3} p_{i,j} B_{i,3}(u) B_{j,3}(v),$$

where $B_{k,l}(u)$ are the Bernstein polynomials.

1.4.2 Boundary Decomposition Schemes

Because it is not possible to find a global parametrization for most surfaces, we have to decompose them into patches, such as in Example 1.1. These pieces are joined by a gluing operation, covering the surface completely. This type of scheme is required when the surface does not have the same topology as $\mathbb{R}^2$.

Note that, some care must be exercised if the surface is not a two-dimensional manifold.

1.4.3 Implicit Solids

An implicit solid is the set of points $x \in \mathbb{R}^3$ which satisfy the equation $f(x) \leq c$. When f is a polynomial over $\mathbb{R}^3$, the point-set is semi-algebraic and, as we demonstrated, the halting set of a machine over R.

1.4.4 CSG Objects

Constructive Solid Geometry objects are generated from basic primitives by a finite process of taking unions ("or's"), intersections ("and's") and complements ("not's).

Proposition 1.4 *Any algebraic CSG object can be expressed as a finite union of basic semi-algebraic sets.*

A basic semi-algebraic *set is defined as the set of $x \in \mathbb{R}^n$ satisfying basic conditions of the type:*

$$\begin{aligned} P(x) &= 0 \\ Q_i(x) &> 0, \end{aligned}$$

where P and Q are polynomials.

Fact: The proof is based on the following facts: If P and Q are polynomials, then

1. $P(x) = 0$ & $Q(x) = 0$ iff $P^2(x) + Q^2(x) = 0$.
2. $Q(x) \not> 0$ iff $-Q(x) > 0 \mid Q(x) = 0$.
3. $P(x) \neq 0$ iff $-P(x) > 0 \mid P(x) > 0$.

Proof: Intersections are eliminated using (1) and complements using (2) and (3).

This demonstrates that an object described by the CSG scheme can be reduced to a simple map computable on a machine over R.

On the other hand, an arbitrary CSG expression can be trivially converted into a machine over R with exactly the same tree structure using the fact that for $P(x) \leq 0$, $Q(x) \leq 0$, the following equivalences apply:

$$P \cup Q = \min(P, Q).$$

$$P \cap Q = \max(P, Q).$$

$$P \setminus Q = P \cap \overline{Q} = \max(P, -Q).$$

1.5 Beyond Geometric Models

The examples above indicate that our algorithmic modeling methods encompass the traditional geometric modeling techniques.

At this point, we need to answer the following crucial questions:

- Can algorithmic methods model complex shapes that are out of the scope of geometric modeling techniques?
- If so, what are the classes of objects that can be described by algorithmic methods?

In the rest of this chapter we will investigate these two issues and their implications.

1.5.1 Fractals

Fractals are examples of highly irregular objects that cannot be constructed using classical geometric modeling techniques, (Mandelbrot, 1977). A fractal is naturally described by a limiting process, well suited to implementation using a recursive algorithm, (Peitgen and Saupe, 1988). For this reason, we would expect that a machine over R is a good candidate to model a fractal. On the other hand, the theorem below is a bit discouraging.

Theorem 1.1 *Halting sets of machines over R have integral Hausdorff dimension.*

Proof: This is a direct corollary of Proposition 1.3. It follows from the fact that semi-algebraic sets (and, hence, countable union of semi-algebraic sets) have integral Hausdorff dimension.

We will see through an example that, although a machine over R cannot be used to compute fractals, it can be used to compute approximations of them.

Example 1.2 (Julia Sets) The set of all points z whose orbits are caotic (not stable) under the action of a complex dynamical system $g(z)$ is the *Julia set* of g. These points are contained in the complement of the basin of attraction of g (points moving to a contracting neighborhood).

Most Julia sets are fractals. Consequently, since fractals have fractional Hausdorf dimension, most Julia sets are undecidable. But, because the complement of a Julia set is "semi-decidable" (i.e. the halting set of a machine over R), it is possible to construct a machine M that computes approximations of a Julia set J.

The idea is to employ a decision machine as in Figure 1.5.

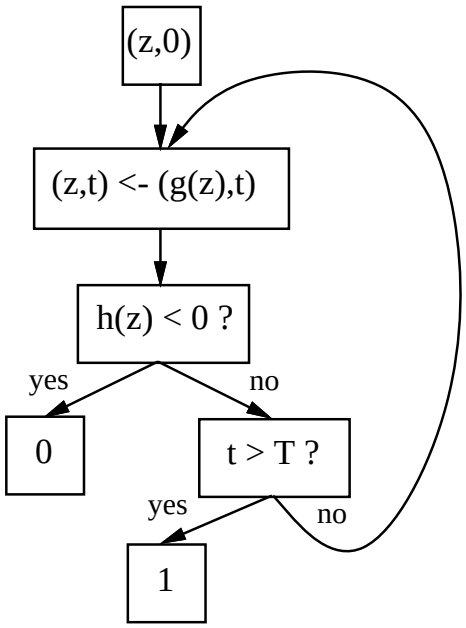

FIGURE 1.5: Julia set machine

From the theory of complex analytic dynamics, there is an $\epsilon > 0$ and a polynomial h such that $h(z) < 0$ if and only if z is in a contracting ϵ-neighborhood of the dynamical system $g(z)$,(Blanchard, 1984). This allows us to identify the points that belong to the complement of J.

Using only the above test, M will halt in a time $t < \infty$ if $z \notin J$, otherwise it will loop forever. Since points closer to the Julia set will take longer to be attracted, an approximation of J is given by the set of points that remain undecided after a large number of iterations T. This approximate method is called *escape time* algorithm by (Barnsley, 1988).

There are a couple of observations that need to be done in relation to the computation of fractals and machines over R: First, the geometry of a fractal set is so complex that cannot be computed exactly. Second, a fractal has a precise mathematical characterization that can be effectively described by a machine over R. In a sense, M gives the best possible representation of a fractal. It allows us to capture the essential aspects of these objects using a direct computational scheme. Third, a fractal shape can be approximated to any degree of accuracy by a union of semi-algebraic sets.

Actually, we can make a much stronger and general affirmative:

Theorem 1.2 (Stone-Weierstrass) *Every continuous function $f : S \to \mathbb{R}$ of a compact metric space S may be represented by a sequence of polynomials converging uniformly to the function.*

Proof: See (?).

1.6 Extending M

So far, we introduced a finite dimensional model of a machine over a ring. It is powerful and very general, allowing us to talk about computation over various domains. In particular, it includes the two important cases, $R = \mathbb{Z}$, the integers and, $R = \mathbb{R}$, the real numbers. In every case, $\mathbb{Z}$ is a natural subring of $\mathbb{R}$.

In order to be complete, and allow for a description of arbitrary algorithmic shapes, this model has to be further developed in several directions.

The model can be extended to include the notion of *infinite dimensional machines* over R. This makes possible the underlying spaces to be R^∞, the infinite direct sum space over R. R^∞ can be thought of as the space of finite but unbounded sequences of R, because a point $y = (y_1, y_2, \ldots) \in R^\infty$ satisfies $y_k = 0$ for k sufficiently large. Consequently, all we need to do is to add a fifth node to the finite dimensional machine to get this extra power. The *fifth node* allows accessing of coordinates of arbitrarily high dimension.

The model can be augmented to incorporate probabilistic algorithms. This requires the addition of a *coin tossing* node with an associated probability distribution.

The model can be modified to be able to deal with parallel and distributed computation. This is a significant change of perspective and requires a formalization of new concepts such as permanency (process) and identity (reference) (Milner, 1989).

1.7 Discussing the Model

A question one may ask is: Why go into the trouble of developing a continuous model of a machine, if the classical theory of computation provides the Turing machine, a discrete model that perfectly reflects the nature of a digital computer?

The problem is that a Turing machine cannot be used to investigate the type of models required for the description of shapes. The theory of digital automata is a chapter of formal logic. It, therefore, emphasizes combinatorics rather than analysis. This prevents us to take advantage of calculus and other tools of continuous mathematics in our models. Furthermore, the Gödel coding of a Turing machine destroys the algebraic structure of the underlying problem spaces, eliminating also the possibility of using algebra [2] .

So, we need a model of a continuous machine!

In spite all those reasons, one can argue that this kind of model is a poor abstraction of the digital computer. How can we input an arbitrary real number into such a discrete machine? Well, this actually cannot be done. In practice the machine can only take in a finite number of decimals. Nonetheless, this limitation does not invalidate our model. The fact that we have to work with rational numbers is not a problem, since they approximate real numbers to a high degree of accuracy. Of course, we have to be careful about round-off errors. This is one of the main subjects of the Numerical Analysis and may be incorporated into the model of a machine over R.

We have already seen that algorithms, even using exact arithmetic, can only solve some numerical problems approximately, to within "accuracy ϵ". Thus, we say that an algorithm *solving a problem*

[2] Note that this in not the case with λ-calculus

approximately is a machine M halting a each input $\epsilon > 0$, $y \in Y$ and satisfying

$$||\varphi_m(\epsilon, y) - \pi^{-1}(y)|| < \epsilon,$$

where the set $\pi^{-1}(y)$ is the exact solution and the norm measures the distance of $\varphi_m(\epsilon, y)$ to it.

Considering the sequence of computations starting with input y:

$$\begin{aligned}
&(n_i, x_i) = H_M(n_{i-1}, x_{i-1}), \quad i = 1 \ldots \\
&n_0 = \text{ input node }, \ x_0 = y \\
&n_T = \text{ output node }, \ z_T = \varphi_M(y).
\end{aligned}$$

We say that $\delta > 0$ is an *admissible round-off error* for (ϵ, y) if for any sequence (n_i, x_i) satisfying $|(n_i, x_i) - H_M(n_{i-1}, x_{i-1})| < \delta$, $n_0 = 1$ and $x_0 = y$, then $|x_T - \varphi_M(\epsilon, y)| < \epsilon$ and n_T is an output node.

Let $\delta_M(\epsilon, y)$ be the supremum of the admissible round-off errors. An algorithm defined by a machine M which solves a problem approximately is *numerically stable* if there exists c and q depending only on M such that

$$\delta_M(\epsilon, y) \leq c(s(y) + |\log \epsilon| + \log w(y))^q,$$

where $s(y)$ is the size of the input and $w(y)$ measures the difficulty of the problem instance y.

Chapter 2

Conceptual Framework

This chapter presents the conceptual framework used to characterize algorithmic modeling. We give an overview of this kind of shape description through the paradigm of the universes. We decompose the modeling process into a hierarchy of abstraction levels and analyze the main aspects at each level. Finally, we discuss the objectives of algorithmic modeling schemes.

2.1 Why Use Algorithmic Models

In the previous chapter we showed that the shape of three-dimensional objects can be modeled by machines over the real numbers. The halting sets of these machines are the countable unions of semi-algebraic sets, exactly the same canonical representation of geometric solids.

This brings up the following question: why use algorithmic models if they represent essentially the same point sets as those given by geometric models? The answer is that we may want to use algorithmic models for several reasons:

- There are objects, such as fractals, which are defined by a limit process and could only be effectively described using an algorithm.
- There are objects whose geometry changes over time or react to external influences. This has to be simulated using an algorithm.
- There are objects that possess a complex geometric structure which could be best captured procedurally.
- Even for simple geometric objects, algorithmic models can be a convenience, because of the extra power and flexibility provided by this scheme.

2.2 General Overview

In order to gain insight into Algorithmic Modeling, we will discuss it globally making comparisons with Geometric Modeling.

2.2.1 Geometric versus Algorithmic Modeling

Traditional solid modeling defines objects as regular point sets. We have seen in Part ?? that they can be constructed according to a representation based on some decomposition of space. In this sense, the representation scheme is a language in which words (sequence of symbols) are the descriptions of geometric solids and sentences (sequence of words) are descriptions of three-dimensional scenes.

Algorithmic modeling incorporates the symbol generation mechanisms into the representation. Consequently, instead of the static structure used in geometric modeling, we are dealing with a dynamic structure in which the rules for generation of symbols are encapsulated into the model itself, producing a geometric description whenever that is required. Note that such scheme allows for adapted realizations of a shape, sensitive to external environment conditions.

The procedural representation is actually a *meta-representation* that introduces another step into the model creation.

2.2.2 Levels of Abstraction

Modeling can be understood using the paradigm of the universes. The description of objects in the computer is viewed as a chain of abstraction levels corresponding to: objects in the real (or imaginary) world; abstract models of these objects; concrete representations of the models; and their computer implementations. In this process, objects are idealized relative to a modeling space and represented through a symbolic description.

In the case of Algorithmic Modeling, the model is a virtual machine, the representation is a program and the implementation a computing system. This hierarchy of abstraction levels is depicted in Figure 2.1.

All of the problems posed in the introductory chapter, related to the use of the universes paradigm, can be restated in this context.

2.3 The Nature of Computation

Algorithmic models can be deterministic or stochastic depending on the nature of the computing strategy adopted in their definition.

Note that, deterministic dynamical systems may exhibit caotic behavior and, on the other hand, well behaved objects may be generated using probabilistic methods.

2.3.1 Deterministic Models

Deterministic models employ only predictable computational methods. All the mathematical relations between elements are fixed. This guarantees that, for a given input y, the machine M always outputs the same result.

2.3.2 Probabilistic Models

Probabilistic models employ stochastic methods of computation which include random components.

Stochastic methods can be related either to properties of the object being modeled, to the computational strategy or to both. An example of the first case is the mathematical characterization of objects that are given in statistical terms, such as those with fuzzy boundaries. For this reason, it is natural to model them as a stochastic process. An example of the second case is the Monte Carlo method to compute integrals.

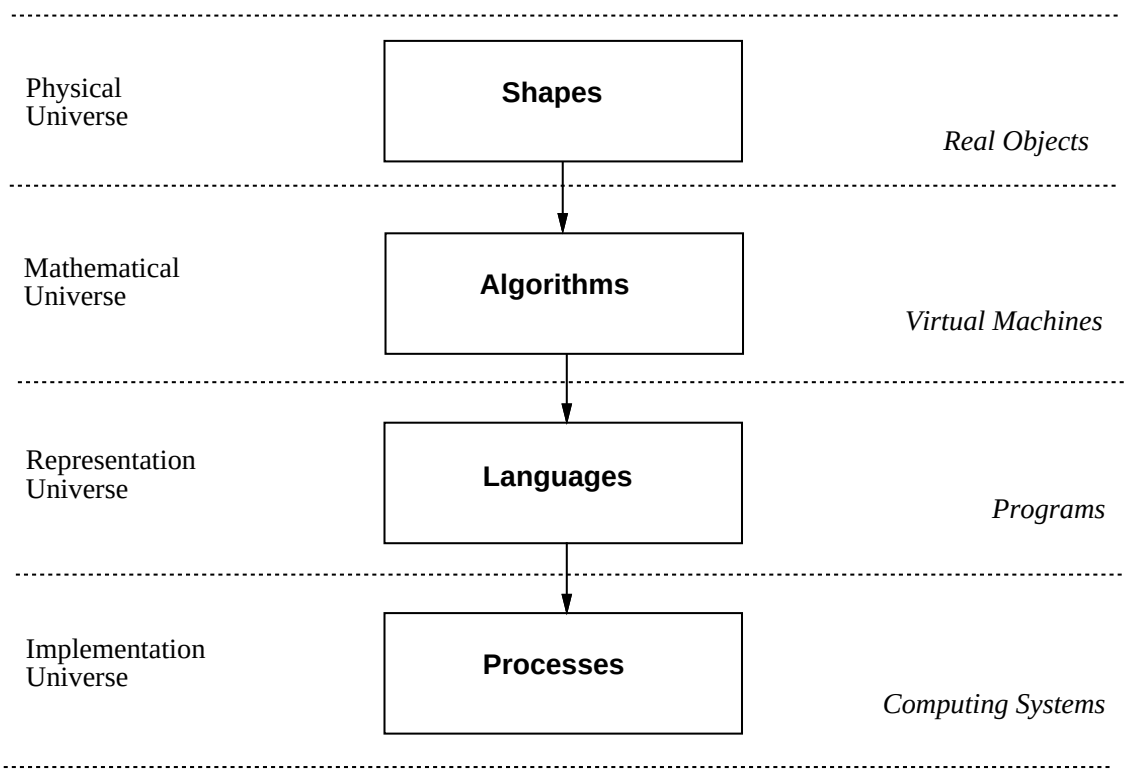

FIGURE 2.1: Levels of Abstraction of Algorithmic Modeling

2.4 Algorithmic Structures

Algorithmic models make use of some fundamental computational structures that allow the construction of more complicated machines from simple ones.

The machine over a ring introduced in the previous chapter is associated with *partial recursive functions over R*. This class of functions are generated by the basic polynomial maps and polynomial inequalities and closed under the following operations:

2.4.1 Composition

Composition is the simplest computational structure. It corresponds to the successive application of a sequence of functions.

2.4.2 Juxtaposition

Juxtaposition is the parallel execution of several independent computations.

2.4.3 Recursion

Recursion is an important characteristic of the algorithmic modeling methods. It allows an effective control of expansion, as well as, subdivision procedures. This computational structure is related to growth processes.

2.4.4 Iteration

Iteration is used by time dependent models and by relaxation algorithms. This computational structure is related to evolution processes.

2.5 Procedural Representations

A language is a way of expressing the notions of some domain. Every language has a set of primitive entities which represent the basic notions of the domain and rules of composition by which compound entities are constructed. These entities are encoded through symbols. The *semantics* of the language corresponds to the meaning of the entities relative to that domain. The *syntax* of the language corresponds to the structure of the symbolic representation.

The syntactic structure of a language (i.e. the ways in which symbols can be combined) is defined by a grammar.

Definition 2.1 *:* A formal grammar is formed by three types of symbols (terminals, nonterminals and start symbol) together with a set of production rules.
Terminals are basic symbols that constitute the strings in the language.
Nonterminals are syntactic categories that denote sets of strings.
Productions are rewriting rules defining the ways in which syntactic categories may be built up. A production is composed of a *left side* and a *right side*.
A *derivation* consists of replacing the symbol on the left side by the symbols on the right side.
The *start symbol* is a nonterminal selected to represent the whole language.

A grammar is *context-free* if the left side of every production consists of a single nonterminal symbol. This means that productions rules can be applied independently in each symbol of a string. A grammar is *context-sensitive* if symbol replacement depends on some neighborhood of the symbol.

The grammar can be used to generate a parser which recognizes the language and, thus, verify whether a program code is syntactically correct or not.

Algorithmic models are represented by the code of a program for a given machine M. A universal machine can simulate all possible machines, and the representation of a machine M is actually a "program" giving the coding of M for this universal machine.

It is possible to associate classes of machines with classes of languages. This result allows:

- The analysis of classes of models through the algorithmic structures of the machines associated with them.
- The categorization of models based on the classes of languages used to encode them.

2.5.1 Expressions

Expressions are simple languages consisting of operators and operands, combined by associativity and precedence relations. *Operators* are functional elements and *operands* are data elements. Operators take as input operands to output a value which can be used by another operator. The *arity* of an

operator is the number of operands it requires. The most common are unary and binary operators. *Associativity* and *precedence relations* define the grouping of elements, eliminating ambiguities of the language.

The syntax of an expression language can be defined by a grammar specifying the operators and their precedence, type of operands, as well as, the form of the expressions. Every expression can always be converted to a *standard form*, `f(x, y)`, where `f` is the operator and `x, y` are the operands. Expressions can also be constructed using: [1]

prefix form: `f @ x ...`
postfix form: `.. x # f`
infix form: `x ~ f ~ y`

An expression may be represented by a tree structure in which the leaf nodes are operands and internal nodes are operators.

In the case of algorithmic modeling, we use *shape expressions* in which the operands are geometric objects and the operators are groups of transformations.

2.5.2 Parallel Grammars

In a parallel grammar the production rules are applied in parallel such that all symbols of a string are replaced simultaneously. Also, there is no distinction in this grammar between terminal and nonterminal symbols, which means that all strings are words in the language. The start symbol is used to generate strings through the successive application of the production rules.

2.5.3 Constraint Languages

Constraint languages are used to express relationships. They define structural connections between a set of elements. Therefore, they are declarative languages, as opposed to most programming languages which are imperative.

A constraint system is composed of the following entities: primitive objects, handles and basic relations. A *handle* is a reference to an object or part of an object. A *constraint declaration* specifies the conditions (relations) linking sets of handles. The system may also include mechanisms to build compound objects and constraints from simpler ones.

We construct constraint networks by associating handles in different constraint declarations. Such a network may be represented by a graph in which the links are relations and the nodes are handles.

In the case of algorithmic modeling, we deal mainly with geometric and shape constraints.

2.6 Implementation Methods

Program codes are static descriptions of algorithmic objects. In order to fulfill their roles and have an effective existence in a computational environment, these sets of symbols need to be processed by a real machine. In this way, a passive program becomes an active process (e.g. the static representation is transformed into a dynamic object).

In practice we use a computing system whose processing power is equivalent to a machine over R. In other words, we employ a real machine which implements the model of a universal machine described in the previous chapter.

[1] This is the notation used by Mathematica.

As we mentioned before, instead of refer to this general model, it is instructive to study models of simpler machines with processing power to implement only a class of algorithmic models. This allows us to identify their structure, as well as their essential characteristics.

Below we will discuss some important classes of virtual machines for algorithmic modeling.

2.6.1 Stack Machines

This class of machine is essentially a simple calculator intended to process expressions. It consists of a pushdown stack and a processor or interpreter that perform basic operations. During execution, partial results are pushed into and poped from the stack, producing a final result in the end.

A shape calculator can be implemented with this kind of machine. In this case, the operands define shapes and the operations are used to modify their appearance and geometry.

2.6.2 Parallel Rewriting Systems

This class of machine performs structural transformations in a representation. It is intend to process grammars expanding a initial representation through a series of derivations. The transformation rules are executed in parallel in each element of the representation.

The structure of this kind of machine is inherently recursive.

2.6.3 Simulation Systems

This class of machine simulates processes that evolve in time or that react to external conditions. The simulation can be continuous or discrete. The difference lies in the way the change of state of the system is modeled. In continuous simulation the transformations defined to vary continuously with time (or space). In discrete simulation the transformations are defined relative to points isolated in time (or space).

Because of the discrete nature of digital computers, even the simulation of a continuous system must be discretized at some point. This can be done by discretizing the *problem* (thus, simulating a discrete model), or by discretizing the *computation of the solution* of the problem (thus, simulating a continuous model).

2.7 Modeling Techniques

Any modeling system must include facilities to allow the creation of new objects, as well as, the modification of existing ones. This set of modeling techniques defines the main form of user interaction with the system.

In algorithmic modeling systems, this is actually a two step process: first a procedural description of the object must be created; then, it may be used to produce geometric descriptions of the object.

The algorithmic model could represent just an individual object or a family of objects. In this last case, the user will interact also with this procedural description in order to specify a particular member of the family. Note that this allows for two kinds of users which manipulate the model in different levels.

Since the algorithmic description is some sort of a program, the modeling system should be a programming environment and the modeling techniques should include tools for program design, debugging and testing.

The modeling system should also incorporate user interface protocols, such as the ones described below, to facilitate control over the model.

2.7.1 Command Based

In a command based interface, the user issues instructions directly to the system. This is usually done using text input. The system's response may include textual and graphical feedback.

This type of interface is well suited to shape calculator systems.

2.7.2 Control Panel

In a control panel interface, the user has access to selected parameters of the model using graphical interaction.

This type of interface is very general and adequate for most modeling systems.

2.7.3 Direct Manipulation

Direct manipulation allows the user to interact with the model geometrically. This can be done using kinematic or dynamic techniques. The ultimate goal is to immerse the user in a virtual world populated by the objects being modeled.

This type of interface is desired in simulation systems.

2.7.4 Natural Selection

Natural selection allows the user to explore large parameter spaces in a efficient manner. This is done through a set of criteria which automatically eliminates most of the elements of a family of objects, leaving to the user the task of selecting the best among few remaining candidates.

This type of interface is indicated for evolutionary systems in which provides a mechanism for mutation of algorithm models.

Chapter 3

System's Issues

Algorithmic models encapsulate data and procedures. They are effectively realized as active processes in the computational environment of a graphics system. For this reason, there is a close interaction between the representation and the system. At the same time, there exists a great independency between classes of models and the system's kernel, because the interaction obeys a well defined protocol. This separation of implementation levels allows more flexibility and expression power.

3.1 Procedural Objects and Computer Graphics

The intrinsic nature of procedural objects imply in a close connection between them and all subareas of computer graphics.

3.1.1 Modeling

The functional characteristics of algorithmic models determine the geometry of the objects they represent. In that sense, we can say that "form follows function".

In the same way that manufactured objects are related to mechanical/industrial engineering, the knowledge base used to develop the algorithmic models is, in general, related to a discipline that studies the object. Physical models, botanical models, and biological models are typical examples of models which can be converted into algorithms that generate the shape of objects.

3.1.2 Visualization

One of the main motivations of using algorithmic models in computer graphics is to produce visual renditions of the objects they represent. For this reason, there is also an emphasis in aspects that influence the appearance of the objects. It is not enough to model only the shape, but also the interaction of light and matter. We need illumination models that are fine tuned for particular classes of objects. One way to evaluate the effectiveness of an algorithmic model is realism of the visualizations it produces, i.e. how close is the synthetic image from a picture of the real object. Although, in some cases, we may not be able to describe the shape of a complex object in every detail, it might be possible to model the visual texture that is produced by its geometry. This is more than sufficient for visualization purposes.

3.1.3 Animation

Algorithm models may also describe motion, as well as, the change of shape over time. Thus, we are dealing with a dynamical system. Objects are now alive, they evolve in time interacting with each other and with the environment. The description of this behavior must be incorporated into the model from the very beginning and the graphics system must take that into account.

3.2 Algorithmic Descriptions

In this section, we discuss different aspects of the algorithmic descriptions from the perspective of a modeling system.

3.2.1 Models

While in geometric modeling we have shapes with a very well prescribed geometry and topology, that is not always the case in algorithm modeling. Therefore, the mathematical tools used for the construction of objects in geometric modeling come from traditional areas of mathematics: Geometry, Topology, Algebra, Approximation Theory, Numerical Analysis etc. On the other hand, algorithmic modeling systems are pressed by the need to find new mathematical tools expanding the class of objects that can be represented. Besides the traditional areas mentioned above, constructing techniques in algorithmic modeling use new areas of mathematics: Dynamical Systems, Geometric Measure Theory, Probability Theory etc.

3.2.2 Representations

Algorithmic models are represented by computational processes. They are particular incarnations of generic procedural descriptions of a given class of objects. The modeling system must provide a operational environment in which all individual procedural objects may coexist and interact. For this reason, there are strong links between the modeling system and representations of the objects.

3.2.3 Implementation

The implementation of algorithmic modeling systems has been influenced by several emerging concepts in Computer Science. Among them, the most important is certainly the Object Oriented Programming. This software development technique is perfectly identified with the principles of algorithmic modeling and allow the effective creation of its computational basis.

In the context of object oriented programming, a *class* of objects is composed of a *state* (local memory) and a set of procedures, or *methods* activated by signals or *messages*.

Classes can be organized in hierarchical or graph structures which determine the *inheritance* of methods by subclasses of a class. The makes possible the definition of objects in different levels of abstraction.

Objects are created through *instancing* operations, which generates an individual representation of an object initializing its state with particular values. Objects interact sending messages back and forth.

These ideas can be implemented in object oriented operating systems, such as in Smalltalk, or Clos (Common Lisp Object System), in object oriented programming languages, such as C++, and even using this methodology with a conventional programming language, such as C or Pascal.

3.3 Properties of the Representation

We will not make a rigorous analysis of the properties of the algorithmic representation scheme. We only mention that, in this respect, they appear to be in opposition to geometric representation schemes.

Algorithmic representations are, in general:

- Ambiguous – can have multiple interpretations.
- Non-unique – can have various models for the same object.
- Relative Validity – the concept of validity is at best subjective. (i.e based on the visual comparison between images of the synthetic and real objects).

This properties are, in part, a consequence of the fact that it is not possible in general to obtain exact geometric representations from algorithmic models, and also, in part, a consequence of the inherent difficulty to deal with algorithmic models in a formal manner.

3.4 General Characteristics

Algorithmic models have several characteristics in common that we discuss below.

3.4.1 Data Base Amplification

Algorithmic models have a very compact representation. The geometric data base is expanded using procedural methods. This can be done for the whole object or for a small part of interest. The expansion is usually performed during the execution of specific task, such as rendering.

It is desirable the use of recursive algorithms that result in a exponential growth of the geometric data base.

3.4.2 Variable Levels of Detail

Algorithmic models are often adaptive. This allows, among other things, to control the level of geometric detail generated by the model There are several methods to do this according to the type of algorithmic model. The basic ones are: control of the geometric expansion, simplification of geometry, and use of separate procedures for different levels of detail.

3.4.3 Stochastic Methods

Algorithmic models may use stochastic methods to introduce in a controlled way variations into the model. These methods employ random processes that are based on the statistical characteristics of the objects being modeled.

The use of tables of random numbers guarantees the efficiency and repeatability of the processes.

3.5 Conversion between Representations

Conversion between procedural representations is a difficult subject. Even in the classical theory of computation there are only a few results.

Ideally, we should have a canonical form for every procedural representation scheme, such that we could make assessments about particular descriptions of a model. Also, we should be able to convert any procedural representation into one for a universal machine.

The problem is that the computational power of virtual machines associated with different algorithmic models is not the same. Therefore, it is not always possible to map one machine into the other.

Chapter 4

Algorithmic Models

This chapter analyses the main classes of algorithmic models. A natural way to classify these models is through their procedural structure. Using this criterium, algorithmic models can be divided into: geometry based models; functional based models; grammar based models; and physics based models.

1. *Geometry based models* describe shapes by mappings of spaces. These models can be parametrized to define a family of objects.
2. *Functional based models* employ functions of space and composition of those functions to create and transform the geometry of objects.
3. *Grammar based models* use a geometric or topological language to define objects.
4. *Physics based models* employ the laws of Mechanics to determine the movement and deformation of objects.

In the following section we will discuss each one of these types of models.

4.1 Geometry Based Models

Geometry based models incorporate the data of geometric objects and the procedures that are necessary to manipulate such data.

4.1.1 Families of Shapes

Geometric models can often be parametrized and used to represent a family of shapes. The algorithms are the same for a given family. Parameters are used to specify an individual element of the family. The interaction with the model is through a communication protocol defining what operations can be performed with it.

A simple example can illustrate these ideas more concretely:

Example 4.1 (Sphere)

- Parameters:
  - geometry: center, radius

  - appearance: color, transparency

- Operations:
  - transformation: translation, rotation, scaling
  - visualization: wire-frame, shaded
  - query: ray intersection, surface normal, volume

As we can see, the parameters determine completely the object. Operations define what can be done with the object and may have colateral effects (e.g. generation of an image). Some operations return a value.

This type of representation has many advantages: It allows to isolate implementation details from the model definition. It creates a mechanism to exploit particular aspects of a family of objects in many operations (for example, in rendering). It permits a wide choice of parametrizations, contributing to a natural and intuitive way to specify the objects. It makes possible using, in the same model, multiple forms of geometric description each one most appropriate to a type of problem (for example, a shape can be described parametrically and implicitly). It establishes a uniform interface with the graphics system.

4.1.2 Generative Models

Generative models are objects defined by the action of a transformation group of k parameters, $T : \mathbb{R}^m \times \mathbb{R}^k \rightarrow \mathbb{R}^n$, on a parametric function $F : \mathbb{R}^l \rightarrow \mathbb{R}^m$, called *generator*. The resulting object is given by the parametric map $T(F(x); q)$ from $\mathbb{R}^{l+k}$ to $\mathbb{R}^n$, where l and k are respectively the degrees of freedom determined by the function F and the transformation T.

This type of model is a procedural generalization of parametric sweep models, (Snyder and Kajiya, 1992).

Example 4.2 (Sweep Surface) Consider the surface $S(u, v)$ formed by applying the continuous transformation $\delta : \mathbb{R}^3 \times \mathbb{R} \rightarrow \mathbb{R}^3$ to the parametric curve $\gamma : \mathbb{R} \rightarrow \mathbb{R}^3$:

$$S(u, v) = \delta(\gamma(u), v).$$

An advantage of this scheme is that the generative modeling representation is closed under the composition operations above, resulting in a powerful algorithmic model.

4.2 Functional Based Models

Functional based models describe objects through functions of space. A basic shape is defined by one function and may be altered by the application of other functions. In this type of model, primitives and transformations are combined using functional composition.

4.2.1 Textures

A texture is a map $t : U \subset \mathbb{R}^m \rightarrow \mathbb{R}^n$, where $\mathbb{R}^n$ is usually identified with a color space and $\mathbb{R}^m$ is the support space of the texture (texture space). Given a function $f : V \rightarrow U$, where V is a subset of an object space, we call *texture mapping* the composition $f \circ t : V \rightarrow U$, that associates to every point x of the object with an element of the vector space $\mathbb{R}^n$. See Figure 4.1. When U is a subset of $\mathbb{R}^2$, t is a 2D texture. When U is a subset of $\mathbb{R}^3$, t is a 3D texture.

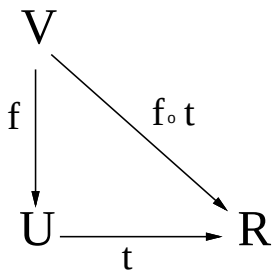

FIGURE 4.1: Texture Mapping

Textures can be used to define various attributes, including the microgeometry, of an object. In the following, we give some examples of procedural texture mapping in algorithmic modeling.

One example is the use of Fourier synthesis to produce textures. This method can be applied in the modeling of clouds and terrain, (Gardner, 1985), (Gardner, 1984). In the first case, the texture is interpreted as a density function. In the second case, it is a height field.

Another example of procedural texture involves a *noise function*. One way to construct this function is to define a random variable over an integer lattice of space; the function value on other points is computed by interpolation. The noise function is used as a basis in the creation of other functions through a composition process. Note that it plays a similar role as the sinusoidal function in a Fourier series. The textures generated by functional composition of noise have been used very successfully to model the appearance of marble, wood and other materials, (Perlin, 1985).

Mapping can also be used to define the microgeometry of surfaces. This type of model, called *bump mapping*, simulates the appearance of surface irregularities. The effect is obtained by a perturbation of the surface normal during the computation of the illumination function. In this way, the model behaves as if the original surface was slightly deformed to be compatible with those normal vectors, (Blinn, 1978). An alternative to bump mapping is the *displacement mapping* which actually modifies the surface geometry, (Cook, 1984).

4.2.2 Hypertexture

Hypertexture is based on solid texture functions. These functions are used directly to define the geometry of the object, rather than to define shape attributes as in texture mapping, (Perlin and Hoffert, 1989).

This algorithmic model works with two kinds of functions: *object density function*, D, which describes the density of a shape over space; and *density modulation functions*, f_i, which are used to control various aspects of the object's spatial characteristics. Hypertexture is created by successive application of modulation functions f_i to a shape density function $D(x)$:

$$H(D, x) = f_n(\ldots f_1(f_0(D(x)))).$$

Note that, although $H(D, x) = c$ seems equivalent to the implicit description of a surface, we cannot guarantee that H always defines a valid surface. Therefore, hypertexture cannot be considered strictly a geometric modeling scheme.

The hypertexture method have been used very effectively to model objects whose local geometry

is so complex that cannot be represented by a bidimensional surface. Typical examples include fur and hair.

4.3 Grammar Based Models

Grammar based models employ a language to define the structure of an object. The elements of this type of model constitute an alphabet with which valid representations (words) are generated.

In computer graphics two kinds of formal grammars are used: geometric and topological. The words in a geometric grammar contain all necessary information to describe the shape of an object. The words in a topological grammar contain only connectivity information and must be augmented with geometric meaning in order to fully describe a shape.

This section studies applications of both of these grammars.

4.3.1 Geometric Grammars

Geometric grammars are "shape grammars". The alphabet, in a formal geometric grammar, is composed of basic shapes that are combined recursively to produce arbitrarily complex shapes. This model was developed in connection with fractal objects. Fractals can be classified into deterministic and random.

Deterministic fractals are naturally defined by a shape grammar. Their construction starts with an *initiator* that is transformed by *generators*. In this scheme, the initiator is the start symbol and the generators are the production rules of the grammar.

Random fractals can be generated by a displacement subdivision algorithm, (Fournier, Fussell and Carpenter, 1982). The computational process for this method can be specified by a shape grammar with the addition of a probabilistic component. This type of algorithm introduces random perturbations with the right statistical properties at increasingly smaller scales of the object. The fractal dimension determines the relative magnitude of the perturbation in each scale.

An important theoretical issue concerns the conjecture that all fractals can be generated by a shape grammar, (Smith, 1984a).

4.3.2 Topological Grammars

Topological grammars are "graph grammars". The alphabet, in a formal topological grammar, is composed of link elements that are arranged to define the connectivity of an object. This structure must then be interpreted in a geometry. The models generated by topological grammars are also called *graftals*, (Smith, 1984b).

A shape is generated in two steps by this method. The first step corresponds to the application of the grammar to produce the topological structure of the object. The second step consists in the interpretation of this structure to create the geometry of the object. This interpretation step requires the use of: *verisimilitude rules* which describe the intrinsic geometric characteristics of a class of objects and also *tropism rules* which regulate the influence of environmental factors to the shape of the object.

Graftals have been applied mainly to the modeling of biological systems, such as trees and other plants. The formal languages known as "L-Systems" were developed for this purpose, (Prusinkiewicz, Lindenmayer and Hanan, 1988). Languages of this class are based on parallel graph grammars. Type $0L$ languages are context-free. Types $1L$ and $2L$ are sensitive to a neighborhood of, respectively, one and two symbols. These languages may also include parenthesis ("Bracketed L-Systems") that incorporate branches in the topological structures.

4.4 Physics Based Models

Physics based models use the methods of mathematical physics to describe the shape and motion of objects. In this context, the main parameters in the definition of the models have a physical nature, like forces and torques. Moreover, the models are time dependent because, even if we are only interested in static configurations, the equations expressing the laws of the physical system must be integrated over time, such that forces produce the desired results.

Physics based models employ a simulation mechanism in order to compute the differential equations defining the system. In this respect, the problem or the solution must be discretized to allow for the numerical computation. The discretization process results in a system of linear equations that should be solved.

The techniques behind this type of models are well known: Optimization, Differential Equations, Numerical Analysis.

4.4.1 Particle Systems

A particle system consists of point masses that move under the influence of forces. This type of model is used to represent complex objects and phenomena which can be decomposed into (or governed by) a set of particles. In this way, fuzzy objects, such as fire, rain, foliage, may be described by a clump of primitive elements, (Reeves, 1983). Also, non-rigid objects, such as cloth, may have their shapes and motion determined by particles which act as control points, (House, Breen and Getto, 1992).

Particle systems may be classified according to the type of interactions between particles: In *uncoupled systems*, the forces acting on a given particle are independent of other particles in the system. In *coupled systems*, internal forces are the result of interaction between particles. Coupled systems may be further subdivided into systems with *fixed* and *dynamic* coupling.

In a particle system, particles are created, exist during a certain period of time, and are exterminated. The attributes of a particle, including its shape, may change over time according to the evolution of the system.

This type of algorithmic model is composed of three basic elements: the start values, the equations of motion, and the simulation mechanism. The start values define the initial configuration of the system, they are the seed data for the model. The equations of motion describe the evolution laws of the system. The simulation mechanism corresponds to the algorithmic substrate of the model. It is responsible for computing the state of the system over time. This process is executed repeatedly for each time step according to the granularity of the simulation.

A characteristic of particles systems is the use of stochastic techniques. This is a way to simulate complex behavior based on simple rules. Random processes may influence particle attributes determined by statistical properties of the object.

4.4.2 Deformable Models

Deformable models describe continuous non-rigid objects which have their shapes modified by the action of forces, (Terzopoulos et al., 1987). The physical model can simulate perfectly *elastic* materials, as well as, deformable *inelastic* materials. In this last case, material properties, such as viscoelasticity, plasticity and fracture, must be considered, (Terzopoulos and Fleischer, 1988).

The algorithmic model, usually, involves the computation of minimum energy configurations that may be solved using variational or optimization techniques.

4.4.3 Constraint Systems

Constraint systems are very general and applicable to a variety of modeling problems They are particularly useful to describe compound objects structured by mechanical linkages, (Barzel and Barr, 1988). As an example, we could mention articulated objects, such as robots, in which constraints are used to govern the work of different types of joints, (Wilhelms and Barsky, 1985). Another interesting application is the construction of self assembly structures, in which constraints make sure that parts are in the right places.

Note: Autonomous Models

Autonomous models go beyond the scope of algorithmic modeling. They use methods of Artificial Intelligence and would be better understood in the context of behavioral modeling.

Animated Groups

Animated groups are a set of individual objects that together act as coherent whole. The algorithmic model associated with this type of system concerns the description of behavior and interaction among group members, (Reynolds, 1987)

Some of the main characteristics of animated groups are:

- Complex geometry of the members.
- Actions of the members are specified by a behavior model.
- The interaction between members define the action of the group.

This type of model makes heavy usage of communication and synchronization mechanisms. It also requires the use of spatial searching techniques.

Chapter 5

Examples

This chapter gives a few examples of algorithmic models. In spite of their extreme simplicity, we will be able to identify most of the concepts introduced in this part.

The procedural description of the objects will be given in two forms: one using a language for a virtual machine specific of each object's class; and another using a general programming language associated with a universal machine. These two descriptions are totally equivalent and a simple transformation converts one into the other. We hope, this will make explicit the algorithmic nature of the models and, at the same time, will clearly draw the fine line separating the notions of data and programs.

The examples roughly correspond to the classes of algorithmic models analyzed in the previous chapter. The first one is a geometric object, a circle. The second one is a texture object, a firewall. The third one is a fractal object, the Koch's curve. The last one is a particle system, a starfield.

These objects are all two dimensional and the procedural descriptions given below are intended to generate a picture of them. This is how algorithmic models are normally used in a graphics system – they interface directly with the rendering routines to produce an image. Note that, if a geometric representation is required, the rendering routines can be replaced with functions to create the appropriate data structures.

This use of procedural objects is becoming a standard practice in the world of 2-D graphics. Several drawing packages employ an algorithm model (Postscript) as their external representation. As a result, the differences between data and programs can get even fuzzier, e.g. data is used to generate a program and programs are interpreted as data.

5.1 Circle

A circle is a simple geometric object. It can be represented as a primitive in parametric or implicit form. The parametric representation will be used in the example.

5.1.1 Geometric Representation

A circle is naturally specified by its center and radius. Therefore, its representation as a geometric primitive is given by the 3-vector (x, y, r), which in a programming language may be coded as:

```
def circle = {double x, y, r;}
```

This pure geometric representation scheme requires that the knowledge to manipulate the mathematical model of the circle is encoded somewhere in the graphics system. For example, the parametric form can be used to draw it.

```
draw_circle(c, n)
{
   moveto(c.x + c.r, c.y);
   for (t = 0; t <= 2PI; t += 2PI/n)
      drawto(c.x + cos(t) * c.r, c.y + sin(t) * c.r);
}
```

5.1.2 Object Representation

The representation of the circle as a data structure together with the procedural substrate required to manipulate it can be encapsulated in a computational description of this family of objects.

```
(class circle
   (state x y r)
   (method draw (n)
      ((send graphics moveto (+ x r) y)
       (repeat (send graphics drawto (+ x (* (cos t) r) (+ y (* (sin t) r)))
               (t 0 2PI (/ 2PI n))))))
```

The virtual machine of an object-oriented system has the following structure:

```
m-object_sys()
{
   loop {
      switch (read(input)) {
      case DEF_CLASS:
         if (compile(class_def))
            install_dict(class);
      case NEW_OBJ:
         if (lookup(class_name))
            instantiate(obj, class);
      case SEND_MSG:
         if (valid(msg, obj))
            evaluate(method(msg,obj));
      default:
         error();
      }
   }
}
```

To draw a circle using this scheme, first an instance of a circle object is created (`(new circle 1 2 3)`), then the message `draw` is sent to the object.

```
(send (new circle 1 2 3) draw 36)
```

5.1.3 Universal Machine Representation

The algorithmic model of the circle primitive consists of the set of procedures used to manipulate the geometric representation.

In order to generate the drawing we execute the program:

```
draw_circle(1, 2, 3, 36);
```

5.2 Fire

Fire has a fuzzy geometry that can be modeled as a density function. Its algorithmic description uses a procedural texture. This functional based model can be implemented by a stack machine.

5.2.1 Stack Machine Representation

The procedural representation is in a form of a posfix expression language similar to Postscript (Adobe Systems, 1986).

The code for a fire texture is

```
dup 3 1 roll turbulence add colormap
```

where `dup` and `roll` are stack operators, `add` is an arithmetic operator and `colormap` and `turbulence` are texture operators. `turbulence` is based on the noise primitive (Perlin, 1985).

The stack virtual machine has the following structure:

```
m-stack(code, args)
{
   push(args);
   while (t = get_token(code)) {
      if (t.type == NUMBER)
         push(t.value);
      else if (t.type == OPERATOR)
         exec(t.code);
      else
         error();
   }
   return pop();
}
```

The implementation of operators usually involves manipulating the stack, where the values of operands are stored. An example is the `add` operator

```
add()
{
   push(pop() + pop());
}
```

5.2.2 Universal Machine Representation

The representation of the texture function for a universal machine is:

```
firewall(x,y)
{
   return colormap(y + turbulence(x,y));
}
```

5.2.3 Texture Generation

In both cases, the density array is generated evaluating the texture function at a set of points of its domain $[0,1] \times [0,1]$.

```
texture(f)
{
   for (u = i = 0; u <= 1; u += uinc, i++)
      for (v = j = 0; v <= 1; v += vinc, j++)
         t[i][j] = f(u,v);
}
```

where `f` is `m-stack(firecode, (u,v))` in the first case and `firewall(u,v)` in the second case.

5.3 Koch's Curve

Koch's curve is a self-similar deterministic fractal. Its algorithmic description uses a formal geometric grammar. The associated virtual machine is a rewriting system which can be implemented by a recursive procedure.

5.3.1 Rewriting System Representation

The grammar representation of the Koch's curve is:

```
I := X
R := X -> X + X - X + X
```

where `I` is the initiator (start symbol), and `R` is the generator (production) [1].

[1] Note that we could have more than one production

The virtual machine has the following structure:

```
m_rewriting(word, productions, n)
{
   foreach (c in word) {
      foreach (r in production) {
         if (c == r.right)
            append(r.left, new_word);
         else
            copy(c, new_word);
      }
   }
   if (n++ < MAX_RECURSION)
      m_rewriting(new_word, productions, n);
   else
      draw_word(new_word, n);
}
```

To draw an approximation of the fractal curve we use a procedure similar to the one adopted by the LOGO language. The final string is interpreted as follows: symbol `X` corresponds to a line segment of length $1/n$; symbol `+` corresponds to a positive rotation of 60 degrees; symbol `-` corresponds to a negative rotation of 120 degrees. The drawing routine is:

```
draw_word(word, n)
{
   foreach (c in word) {
      switch (c) {
      case X: draw_line(1/n);
      case +: turn(60);
      case -: turn(-120);
      }
   }
}
```

To generate the model, we invoke the rewriting machine with the representation of Koch's curve:

```
m_rewriting((X), (X -> X + X - X + X), 0)
```

5.3.2 Universal Machine Representation

The algorithmic model of Koch's curve can be represented in a universal machine by the following program:

```
x_code(angle, n)
{
   if (n++ < MAX_RECURSION) {
      x_code(60, n);
      x_code(-120, n);
      x_code(60, n);
      x_code(0, n);
   } else {
      draw_line(1/n);
      turn(angle);
   }
}
```

In order to generate the drawing we execute the program:

```
x_code(0, 0);
```

5.4 Starfield

A starfield is composed of bodies that move independently according to laws of motion. Its algorithmic description is that of a particle system. The virtual machine is an event-based simulation system which can be implemented using a iterative structure. The system also incorporates a stochastic component.

5.4.1 Simulation System Representation

The description of the starfield for the simulation system is:

```
MODEL: starfield
POPULATION: num_particles
RUN_TIME: num_frames
EVENT: out_of_frame
   kill_particle;
EVENT: new_frame
   clear_image := clear();
   draw_particle := plot(p.pos);
   update_particle := p.pos += p.vel;
RANDOM:
   p.pos := (p_mean, p_deviation);
   p.vel := (v_mean, v_deviation);
```

The virtual machine for the simulation system has the following structure:

```
m-simulation(commands)
{
   parse commands;
   generate population;

   do {
      until ((e = event_list()) != NULL)
         process(e);
   } while (runtime < num_frames);
}
```

To generate the starfield we invoke the simulation machine with the commands describing the particle system.

```
m_simulation(starfield)
```

5.4.2 Universal Machine Representation

The model of a starfield can be represented in a universal machine by the following programs:

```
particle_system(num_particles, num_frames)
{
   parallel (num_particles) {
      particle(random(p_mean, p_deviation)
               , random(v_mean, v_deviation));
   }
   for (num_frames) {
      clear();
      signal(new_frame);
   }
}


particle(pos, vel)
{
   while (wait(new_frame)) {
      if (out_of_image) {
         exit();
      } else {
         plot(pos);
         pos += vel
      }
   }
}
```

In order to generate the starfield we start the particle system.

```
particle_system(num_part, nun_frames);
```

5.5 References

Adobe Systems, I. (1986). *Postscript Language – Reference Manual.* Addison-Wesley.

Barnsley, M. (1988). *Fractals everywhere.* Academic Press.

Barzel, R. and Barr, A. H. (1988). A modeling system based on dynamic constraints. *Computer Graphics (SIGGRAPH '88 Proceedings)*, 22(4):179–188.

Blanchard, P. (1984). Complex analytic dynamics of the riemman sphere. *Bulletin of the American Mathematical Society*, 11:85–141.

Blinn, J. F. (1978). Simulation of wrinkled surfaces. *Computer Graphics (SIGGRAPH '78 Proceedings)*, 12(3):286–292.

Blum, L., Shubb, M., and Smale, S. (1989). On a theory of computation and complexity over the real numbers: *np*-completness, recursive functions and universal machines. *Bulletin of the American Mathematical Society*, 21(1):1–46.

Blum, L. and Smale, S. (1992). The godel incompleteness theorem and decidability over a ring. preprint.

Cook, R. L. (1984). Shade trees. *Computer Graphics (SIGGRAPH '84 Proceedings)*, 18(3):223–231.

Fournier, A., Fussell, D., and Carpenter, L. (1982). Computer rendering of stochastic models. *Communications of the ACM*, 25(6):371–384.

Gardner, G. Y. (1984). Simulation of natural scenes using textured quadric surfaces. *Computer Graphics (SIGGRAPH '84 Proceedings)*, 18(3):11–20.

Gardner, G. Y. (1985). Visual simulation of clouds. *Computer Graphics (SIGGRAPH '85 Proceedings)*, 19(3):297–303.

House, D. H., Breen, D. E., and Getto, P. H. (1992). On the dynamic simulation of physically-based particle-system models.

Mandelbrot, B. (1977). *Fractals: Form, Chance and Dimension.* W. H. Freeman, San Francisco, California.

Milner, R. (1989). *Communication and Concurrency.* Prentice Hall, Englewood Cliffs, N.J.

Peitgen, H. O. and Saupe, D. (1988). *The Science of Fractal Images.* Springer-Verlag, New York.

Perlin, K. (1985). An image synthesizer. *Computer Graphics (SIGGRAPH '85 Proceedings)*, 19(3):287–296.

Perlin, K. and Hoffert, E. M. (1989). Hypertexture. *Computer Graphics (SIGGRAPH '89 Proceedings)*, 23(3):253–262.

Prusinkiewicz, P., Lindenmayer, A., and Hanan, J. (1988). Developmental models of herbaceous plants for computer imagery purposes. *Computer Graphics (SIGGRAPH '88 Proceedings)*, 22(4):141–150.

Reeves, W. T. (1983). Particle systems – a technique for modeling a class of fuzzy objects. *ACM Trans. Graphics*, 2:91–108.

Reynolds, C. W. (1987). Flocks, herds, and schools: A distributed behavioral model. *Computer Graphics (SIGGRAPH '87 Proceedings)*, 21(4):25–34.

Smale, S. (1990). Some remarks on the foundations of numerical analysis. *SIAM Review*, 32(2):211–220.

Smith, A. R. (1984a). Graftal formalism notes. Technical Memo no. 4. Pixar.

Smith, A. R. (1984b). Plants, fractals and formal languages. *Computer Graphics (SIGGRAPH '84 Proceedings)*, 18(3):1–10.

Snyder, J. M. and Kajiya, J. T. (1992). Generative modeling: A symbolic system for geometric modeling. *Computer Graphics (SIGGRAPH '92 Proceedings)*, 26(2):369–378.

Terzopoulos, D. and Fleischer, K. (1988). Modeling inelastic deformation: Viscoelasticity, plasticity, fracture. *Computer Graphics (SIGGRAPH '88 Proceedings)*, 22(4):269–278.

Terzopoulos, D., Platt, J., Barr, A., and Fleischer, K. (1987). Elastically deformable models. *Computer Graphics (SIGGRAPH '87 Proceedings)*, 21(4):205–214.

Wilhelms, J. and Barsky, B. A. (1985). Using dynamic analysis to animate articulated bodies such as humans and robots. In Wein, M. and Kidd, E. M., editors, *Graphics Interface '85 Proceedings*, pages 97–104. Canadian Inf. Process. Soc.