

Recursive Turtle Programs and Iterated Affine Transformations

Tao Ju*, Scott Schaefer, Ron Goldman

Department of Computer Science, Rice University, Houston, TX 77005

Abstract

We provide a formal proof of equivalence between the class of fractals created by Recursive Turtle Programs (RTP) and Iterated Affine Transformations (IAT). We begin by reviewing RTP (a geometric interpretation of non-bracketed L-systems with a single production rule) and IAT (Iterated Function Systems restricted to affine transformations). Next, we provide a simple extension to RTP that generalizes RTP from conformal transformations to arbitrary affine transformations. We then present constructive proofs of equivalence between the fractal geometry generated by RTP and IAT that yield conversion algorithms between these two methods. We conclude with possible extensions and a few open questions for future research.

Key words: Fractal, affine transformation, turtle graphics

1 Introduction

In computer graphics, there are several different methods for generating 2D geometry based on points and lines. One simple model is to use a turtle, as

* Corresponding author. Email: jutao@rice.edu Fax: (713)348-5930

in LOGO-like languages. A turtle is represented by a state such as a position and a direction vector, and the turtle moves in response to commands such as FORWARD and TURN. Each command induces a transformation of the turtle state in its *local* coordinate frame. The trail that the turtle leaves behind forms the geometry which we will refer to as Turtle Geometry. A seemingly unrelated model, Affine Geometry, generates shapes by performing affine transformations (e.g., translation, rotation, scaling and shearing) on a set of initial shapes in a *global* coordinate system.

Both models are capable of composing complicated and beautiful graphics [1,2], and both are popular tools for generating self-similar fractals [3], such as the tree-like fractal and its variations in figure 1. These fractals consist of components that are affinely transformed versions of themselves.

One way to generate these fractals is to iteratively apply contractive affine transformations to some initial shape [1]; the fractal is the limit shape (or the attractor) of these transformations. Another way is to write a turtle program with turtle commands and recursive calls in a LOGO-like environment [2]. The fractal is the shape drawn at the limit of the recursion. A natural question to pose is: which model, iterated affine transformations (**IAT**) or recursive turtle programs (**RTP**), is more powerful for generating fractals?

In modern computer graphics, people have studied the relationship between L-systems, a string rewriting scheme with a turtle interpretation[4], and Iterated Function Systems (IFS), a superset of IAT containing arbitrary contractive transformations. Prusinkiewicz *et al.* [5] first mentioned a procedure for converting L-systems that result in constant branching angles and scalings to IFS with condensation sets. Culik *et al.* [6] formalize the notion of “uniform

growth” and generalize the preceding results to provide conversion from uniformly growing L-systems to mutually recursive IFS. In the other direction, Prusinkiewicz *et al.* [7] consider language-restricted IFS in which sequences of transformations can be represented by formal languages, and provide a conversion method to an equivalent L-system with a turtle interpretation. In a different approach, Hart [8] introduces the object instancing paradigm and provides methods for converting both L-systems and IFS into this new model for generating fractals.

Unfortunately, due to the complexity and diversity of L-systems and IFS, these conversion methods are often incomplete or lack formal rigor. For example, previous work is restricted only to IFS containing affine transformations, and, in particular, *oriented conformal maps* (although not explicitly stated). Discussions of L-systems, on the other hand, often rely on bracketed branching production rules to avoid accumulated changes in the turtle state. Moreover, no procedural algorithm exists for converting the transformations in an IFS into the symbols in an L-system.

In our work, we will study both L-systems and IFS in their simplest settings. We consider first a simple RTP, which corresponds to a non-bracketed L-system with a single production rule, and an IAT containing oriented conformal maps. We will show that these two methods are equivalent in their expressive power, and we will provide formal conversion algorithms in both directions. We then consider IAT containing arbitrary non-singular affine transformations, and an augmented RTP with a new syntax of turtle commands. As we will see, these two extended methods for generating fractals are again equivalently powerful, and the conversion algorithms are simple adaptations of those in the simpler setting. By establishing these equivalences rigorously in

these basic settings, it is then possible to extend these techniques to formally prove equivalences between more complex RTPs (or L-systems) and IATs (or IFSs).

We start by introducing in detail these two methods for generating fractals, and comparing several RTPs and IATs that generate the same fractals. The conversion algorithms are presented next together with proofs of equivalence. We conclude with a discussion of related issues and possible extensions of our results, while posing some open questions for further research.

2 Fractal Generating Methods

2.1 Recursive Turtle Programs (RTP)

In 2D turtle geometry, the turtle is represented by a set of 2D vectors embodying its *state*, and its movement is governed by a set of *commands* that induce translation, scaling or rotation of the state vectors. We will examine two types of turtle frameworks that differ in their set of state vectors and commands.

2.1.1 Classical Turtle

Traditionally, a turtle state is described by the turtle's location in the plane ($P = \{p_1, p_2\}$), and a vector ($u = \{u_1, u_2\}$) denoting both the direction in which it is facing and its step size. (Note that this model differs slightly from standard Logo, but has exactly the same expressive power; see page 136 in "Turtle Geometry" by Abelson and Disessa [2].) There are four turtle commands, FORWARD, MOVE, TURN and RESIZE. Each command takes a

single scalar value as its parameter and alters the turtle state as follows:

- (1) FORWARD(d): move forward d turtle steps, and draw a straight line,

$$P_{new} = P + d * u, \quad u_{new} = u$$

- (2) MOVE(d): same effect as FORWARD(d), but without drawing.

- (3) TURN(a): rotate heading by a ,

$$P_{new} = P,$$

$$u_{new} = \{u_1 \cos(a) - u_2 \sin(a), u_1 \sin(a) + u_2 \cos(a)\}$$

- (4) RESIZE(s): alter step size by a factor of s ,

$$P_{new} = P, \quad u_{new} = s * u$$

Observe that these commands are really transforming the turtle with respect to the turtle's current state. For example, FORWARD translates the turtle along the current forward direction, and TURN rotates the turtle around the current position. Indeed, the current turtle state describes a local coordinate system S in which the next turtle command takes effect. Each turtle command can be interpreted as a transformation matrix L that converts the turtle's current coordinate system S to the turtle's next coordinate system S_{new} via

$$S_{new} = L * S \tag{1}$$

Note that the transformations induced by the commands for this classical turtle are oriented angle-preserving transformations (i.e., *oriented conformal maps*). Hence S is always orthogonal and uniform, so S can be expressed in terms of the turtle state vectors plus an orthogonal vector u' . Using homoge-

nous coordinates,

$$S = \begin{Bmatrix} u \\ u' \\ P \end{Bmatrix} = \begin{Bmatrix} u_1 & u_2 & 0 \\ -u_2 & u_1 & 0 \\ p_1 & p_2 & 1 \end{Bmatrix} \quad (2)$$

From (1) and (2), we derive in table 1 the transformation matrix corresponding to each turtle command as .

2.1.2 Augmented Turtle

To allow arbitrary affine transformations on the turtle's local coordinate system (which is important for our later discussion of the equivalence of RTP and IAT), we can let the turtle hold two direction vectors. This augmented turtle is represented by a position ($P = \{p_1, p_2\}$) in the plane, a forward vector ($u = \{u_1, u_2\}$) pointing ahead, and a left-hand vector ($v = \{v_1, v_2\}$) pointing to its left. The reason to have this second direction vector v is that now we can fully describe an arbitrary turtle coordinate system as:

$$S = \begin{Bmatrix} u \\ v \\ P \end{Bmatrix} = \begin{Bmatrix} u_1 & u_2 & 0 \\ v_1 & v_2 & 0 \\ p_1 & p_2 & 1 \end{Bmatrix} \quad (3)$$

Accordingly, our advanced turtle can accept commands that perform non-conformal transformations. To make the extension simple, we inherit the same commands from the classical framework, but augment new syntax and trans-

formation matrices as in table 2.

Note that the commands FORWARD and MOVE preserve the same meaning as in the classical framework. The RESIZE command now can take two arguments that change the length of the forward vector and the left-hand vector independently. The TURN command also accepts two parameters that rotate the forward vector and the left-hand vector independently **in the turtle's coordinate system**. From the turtle's point of view, the turtle's two vectors are both of unit length and orthogonal to each other, and the TURN command rotates each vector relative to this orthonormal basis. An equivalent definition of $\text{TURN}(\alpha_1, \alpha_2)$ is to first map the turtle's coordinate system S to a fixed global coordinate system, then rotate the x-axis vector by α_1 and the y-axis vector by α_2 , and subsequently map the global coordinate system back to S by applying the inverse of the first map. For example, applying $\text{TURN}(\frac{1}{2}\pi, \frac{1}{2}\pi)$ to the turtle's coordinate system on the left of figure 2 would modify the turtle's two direction vectors as on the right of figure 2 (note that the length of each vector changes accordingly). Figure 3 illustrates the effect of the modified TURN and RESIZE commands on the shapes drawn by the augmented turtle. Both TURN and RESIZE can still take a single parameter. To be consistent with our previous definitions, $\text{TURN}(\alpha)$ is equivalent to $\text{TURN}(\alpha, \alpha)$, and $\text{RESIZE}(s)$ is equivalent to $\text{RESIZE}(s, s)$.

The classical turtle is a special case of the augmented turtle whose left-hand vector is always orthogonal to and has the same length as the forward vector, and which only accepts TURN or RESIZE commands that affect both axes uniformly. The augmented turtle, on the other hand, carries a more flexible coordinate system that gives the turtle programmer more control over shape generation.

2.1.3 Fractals from RTP

A turtle program is a finite sequence of turtle commands, and draws points and lines generated by the turtle's path. To generate fractal shapes, which may contain an infinite number of points and line segments, we resort to recursion. A recursive turtle program (RTP) is composed of:

- (1) Base case: a finite set of turtle commands that draw some shape at the bottom level of recursion,
- (2) Recursion body: a finite set of turtle commands as well as recursive calls to the program itself. (Note that here we do not allow mutually recursive turtle programs, which will be a topic of discussion in section 4.2.)

The fractal is the limit shape generated by the RTP as the depth of recursion goes to infinity. Note that the recursion body corresponds to the production rule in an L-system, and recursive calls correspond to production symbols. For simplicity, we will study RTPs containing a single recursion body and no state stack. However, the techniques we shall develop can be extended for RTPs with multiple recursion bodies and stack commands (see section 4), which correspond to L-systems with multiple production rules and brackets.

In practice, we only need to recur to a finite level for visualization. An example RTP is shown in figure 4 left using classical turtle commands, which generate the tree-like fractal in figure 5 top with increasing depth of recursion (assuming the initial turtle state is at the origin with forward vector $\{1, 0\}$). On the right of figure 4 is another example RTP but using the augmented turtle commands. The only essential difference between these two programs lies in the two RESIZE commands. When the variables $\{\alpha, \beta\}$ assume values other than $\{1, 1\}$, the program on the right generates variations of the same fractal

as its classical counterpart. For example, the middle and right fractals from figure 1 can be produced as shown in the middle and bottom of figure 5, where $\{\alpha, \beta\}$ are set to $\{0.75, 1\}$ and $\{1, 0.75\}$ respectively.

Observe from figure 5 that the drawing commands in the base case of an RTP generate a different portion of the fractal than the drawing commands in the recursion body. The outer “leaves” (shown as dark lines) of the tree are drawn by the FORWARD command in the base case, and the tree “branches” (shown as gray lines) are drawn by the FORWARD command in the recursion body.

In both RTPs, the base case and the recursion body introduce no net change in the turtle state. Any overall changes in the position or the heading of the turtle in the base case or in a recursion body may have an unsettling impact on the convergence of the turtle state, which will be discussed briefly in section 4. For now we assume that the RTPs in our discussion involve no overall change in the turtle state either in their base case or in their recursion body.

2.2 Iterated Affine Transformations (IAT)

In affine geometry, complex shapes are generated from simple shapes by performing affine transformations in a global coordinate system. The class of shapes that can be generated is determined by the class of transformations allowed. We will examine two groups of affine transformations: oriented conformal maps and arbitrary non-singular affine transformations.

2.2.1 *Oriented Conformal Maps*

Oriented conformal maps, or angle-preserving affine transformations, are products of elementary transformations such as translations, rotations, and uniform scalings. The corresponding matrices are summarized in the first three lines of table 3. As we have seen in the classical turtle framework, such transformations always preserve the orthogonality and the uniformity of the coordinate system.

2.2.2 *Arbitrary non-singular affine transformations*

An arbitrary affine transformation is any transformation that preserves collinearity (i.e., points lying on a line still lie on a line after transformation) and ratios of distances (e.g., the midpoint of a line segment remains the midpoint after transformation). Such affine transformations are products of a larger group of elementary transformations, including non-uniform scaling and shearing. The corresponding transformation matrices are shown in table 3.

2.2.3 *Fractals from IAT*

To generate fractal shapes, we can iteratively apply a set of affine transformations to some initial shapes. Specifically, an iterated affine transformation (IAT) [1] is composed of:

- (1) A set of affine transformation matrices: $\{A_1, \dots, A_n\}$,
- (2) A condensation set C : either the empty set or a finite collection of points and lines, and
- (3) An initial geometry F_0 : either the empty set or an initial finite collection

of points and lines.

The geometry at the $(p + 1)$ st iteration is defined inductively as

$$F_{p+1} = A_1(F_p) \cup \dots \cup A_n(F_p) \cup C \quad (4)$$

The fractal is the limit shape of the IAT as the depth of iteration goes to infinity. To generate the same tree-like fractal of which the first few iterations are shown in figure 5 top, we can construct an IAT using the conformal maps (we will show how one can compute these matrices using turtle programs in Section 3.3)

$$A_1 = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ -\frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 1 & 1 \end{pmatrix}, A_2 = \begin{pmatrix} \frac{1}{2} & -\frac{1}{2} & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 1 & 1 \end{pmatrix},$$

along with the condensation set and initial geometry

$$C = \text{Line}(\{0, 0\}, \{0, 1\}), F_0 = \text{Line}(\{-1, 1\}, \{1, 1\}).$$

We can also define an IAT that generates the fractals shown in the middle and bottom of figure 5 using the non-conformal maps

$$A_1 = \begin{pmatrix} \frac{\beta}{2} & \frac{\alpha}{2} & 0 \\ -\frac{\beta}{2} & \frac{\alpha}{2} & 0 \\ 0 & 1 & 1 \end{pmatrix}, A_2 = \begin{pmatrix} \frac{\beta}{2} & -\frac{\alpha}{2} & 0 \\ \frac{\beta}{2} & \frac{\alpha}{2} & 0 \\ 0 & 1 & 1 \end{pmatrix},$$

along with the condensation set and initial geometry

$$C = \text{Line}(\{0, 0\}, \{0, 1\}), F_0 = \text{Line}(\{-1, 1\}, \{1, 1\})$$

where $\{\alpha, \beta\}$ are the same scaling parameters as in the augmented turtle RTP in figure 4.

Note that the condensation set C is necessary in order to produce the complete fractal. For example, the previous IATs would generate only the outer “leaves” of the tree-like fractal in figure 5 without the condensation set, which is necessary to construct the “branches” of the tree.

In order to generate convergent geometry, we require that the affine transformations A_i are *contractive maps*. That is, the distance between any two distinct points in the plane always contracts after the transformation. Due to the Contractive Mapping Theorem [1], F_p always converges to the same limit shape no matter what F_0 we choose to start with (provided that either $F_0 \neq \emptyset$ or $C \neq \emptyset$). For example, if we choose our initial geometry F_0 to be either a triangle or a quadrilateral in the tree-like fractal example, the geometry generated by the IAT for these two initial shapes look more alike as the number of iterations increases (see figure 6). In the limit, we obtain the same fractal shapes, since the tiny copies of F_0 all shrink to points.

3 Equivalence and conversion algorithms

3.1 Observations

Comparing the fractal trees generated by RTP and IAT in the previous section along with many other examples leads to following observations:

- (1) The base case of an RTP corresponds to the initial geometry F_0 in an IAT.
(They are both responsible for generating the dark lines in the fractals

in figure 5).

- (2) The turtle commands in the recursion body of an RTP correspond to the condensation set C in an IAT. (They are both responsible for generating the gray lines in the fractals in figure 5).
- (3) The RTP has the same number of recursive calls as the number of affine transformations in the corresponding IAT.

These observations suggest that we can express the shape generated by an RTP by a recurrence relation similar to (4) for an IAT.

3.2 Recurrence relation for RTP

Our construction of this recurrence relation is based on the following proposition (Similar results are also obtained by Prusinkiewicz [7]):

Proposition 3.1 *Suppose L_1, \dots, L_n are the transformation matrices corresponding to the turtle commands c_1, \dots, c_n , and let S_k be the turtle's coordinate system after the k th turtle command c_k is executed. Then $L = L_n * \dots * L_1$ is the transformation matrix that transforms S_0 to S_n .*

Proof: From (1) we have $S_k = L_k * S_{k-1}$, therefore $S_n = L_n * S_{n-1} = L_n * L_{n-1} * S_{n-2} = \dots = L_n * \dots * L_1 * S_0$.

Since the shape drawn by the turtle consists of geometries that are generated in the turtle's coordinate system, we have the following corollary:

Corollary 3.2 *Suppose $L_1, \dots, L_k, \dots, L_n$ are the transformation matrices corresponding to the turtle commands $c_1, \dots, c_k, \dots, c_n$. Let P_k be the shape drawn by the turtle program $\{c_1, \dots, c_k\}$, and let Q_k be the shape drawn by the*

turtle program $\{c_{k+1}, \dots, c_n\}$. Then $P_n = Q_0 = P_k \cup L(Q_k)$ for $k = 1, \dots, n-1$, where $L = L_k * \dots * L_1$.

For an RTP with n recursive calls, we can compose transformation matrices $N_i (i = 1, \dots, n+1)$ as the product of the transformation matrices corresponding to the turtle commands in between the recursive calls *in reverse order*:

$$N_i = L_{n_i}^i * \dots * L_1^i \quad (5)$$

where L_k^i represents the transformation matrix corresponding to the k th turtle command after the $(i-1)$ th recursive call (or since the beginning of the recursion body if $i = 1$), and n_i is the total number of such commands before the i th recursive call (or before the end of the recursion body if $i = n+1$).

Once we have N_i we can formulate matrices M_i that transform the canonical orthogonal basis (i.e.; the initial turtle coordinate system) to the turtle's coordinate system at the beginning of the i th recursive call as follows:

$$M_1 = N_1, \quad M_i = N_i * M_{i-1} (2 \leq i \leq n) \quad (6)$$

Note that (6) holds if and only if each recursive call introduces no net change in the turtle state (a more complicated formulation results if net change is introduced, as will be discussed in section 4.1).

Denote by G_0 the shape generated by the drawing commands in the base case, and by R the shape generated by the drawing commands in the recursion body. Due to Corollary 3.2, the shape G_{p+1} produced by the RTP with recursion depth $p+1$ satisfies the following recurrence relation:

$$G_{p+1} = M_1(G_p) \cup \dots \cup M_n(G_p) \cup R \quad (7)$$

Observe the close resemblance between (4) and (7), which matches our prior observations. We can now study equivalences and conversion algorithms between RTPs and IATs based on the common structure of their recurrences.

3.3 Convert RTP to IAT

The conversion algorithm is given by the following two propositions:

Proposition 3.3 *The geometry generated by an RTP using the classical turtle commands and states can be reproduced by an equivalent IAT using oriented conformal maps.*

Proof: Since each L_k^i in equation (5) is an oriented conformal map, their products, N_i and M_i , as defined in (5) and (6), are oriented conformal maps as well. Therefore we can construct the following conformal IAT:

$$F_0 = G_0, C = R, A_i = M_i (i = 1, \dots, n).$$

where G_0 and R are the shapes that the RTP draws in the base case and in the recursion body. Now compare the geometries F_p and G_p defined respectively by recurrence relations (4) and (7). They have the same base case and satisfy the same recurrence; therefore, by induction, $F_p = G_p$ for all $p \geq 0$.

Proposition 3.4 *The geometry generated by an RTP using the augmented turtle commands and states can be reproduced by an equivalent IAT using arbitrary affine transformations.*

Proof: Since each L_k^i in equation (5) is a non-singular affine transformation, their products, N_i and M_i , as defined in (5) and (6), are also non-singular affine transformations. The rest of the proof is identical to that of Proposition

3.3.

The proofs of Proposition 3.3 and Proposition 3.4 provide a conversion procedure from an RTP to an equivalent IAT. All we need to do is to construct F_0 and C from the turtle commands, and to build the matrices A_i from the turtle transformation matrices given in (5) and (6).

To see this process in action, we will now convert the classical RTP on the left of figure 4 to an IAT using oriented conformal maps. The conversion from the augmented RTP on the right of figure 4 to an IAT using arbitrary affine transformations follows closely to the form of this procedure. We proceed in the following fashion:

- (1) Construct initial geometry F_0 : The base case draws a line between $\{-1, 1\}$ and $\{1, 1\}$. Hence $F_0 = \text{Line}(\{-1, 1\}, \{1, 1\})$.
- (2) Construct condensation set C : The recursion body draws a line between $\{0, 0\}$ and $\{0, 1\}$. Hence $C = \text{Line}(\{0, 0\}, \{0, 1\})$.
- (3) Construct matrices A_i : First we build the products N_i : $N_1 = L_R(-\frac{1}{4}\pi) * L_S(\frac{\sqrt{2}}{2}) * L_T(1) * L_R(\frac{1}{2}\pi)$, $N_2 = L_R(-\frac{1}{2}\pi)$, and $N_3 = L_R(\frac{1}{2}\pi) * L_T(1) * L_S(\sqrt{2}) * L_R(-\frac{1}{4}\pi)$. Next we compute the transformations A_i (M_i) from (6):

$$A_1 = \begin{Bmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ -\frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 1 & 1 \end{Bmatrix}, A_2 = \begin{Bmatrix} \frac{1}{2} & -\frac{1}{2} & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 1 & 1 \end{Bmatrix}$$

Observe that A_i , C and F_0 are identical to the first IAT of section 2.2.3 which produces the same tree-like fractal as our RTP.

The conversion procedure is useful in several ways. Because a turtle always walks and turns in its local coordinate system, it is often difficult to understand the “big picture” of the resulting geometry from the turtle commands. On the other hand, the equivalent IAT captures the self-similarity of the fractal precisely through global transformation matrices. Hence the conversion from RTP to IAT provides some insight into the geometry of the fractal corresponding to the turtle program.

Moreover, the convergence properties of the geometry generated by iterated affine transformations are well-known. By converting an RTP to an equivalent IAT, the convergence of the resulting fractal shape can be studied directly by examining the contractiveness of the transformation matrices.

It also follows from this conversion procedure that, in analogy to the observation that the fractal shape of an IAT does not depend on the initial shape F_0 , what the turtle draws in the base case (without introducing net change in the turtle state) does not affect the limit shape. As we will see in section 4.1, this property still holds if both the base case and the recursion body introduce the same net change in the turtle state.

3.4 Convert IAT to RTP

Conversely, we can construct an equivalent RTP for any given IAT. Our construction is based on the following propositions:

Proposition 3.5 *Every oriented conformal map can be represented as the product of four classical turtle transformation matrices.*

Proof: Since a conformal map A transforms one orthogonal basis to another, using the notation in table 3, A can be decomposed into the following product:

$$A = A_S(s) * A_R(\alpha) * A_T(dx, dy)$$

where s is the scaling factor, α measures the rotation angle, and $\{dx, dy\}$ is the translation vector. We can rewrite this product as the product of four classical turtle transformation matrices shown in table 1:

$$A = L_S(s) * L_R(\alpha - \beta) * L_T(d) * L_R(\beta) \quad (8)$$

where $d = \sqrt{dx^2 + dy^2}$ and β is the angle between the translation vector $\{dx, dy\}$ and the vector $\{1, 0\}$.

Proposition 3.6 *Every non-singular affine transformation can be represented as the product of four augmented turtle transformation matrices.*

Proof: Any non-singular affine transformation A can be written using homogeneous coordinates in the following form:

$$A = \begin{pmatrix} a & b & 0 \\ c & d & 0 \\ e & f & 1 \end{pmatrix}$$

where $ad \neq bc$. Similar to (8), we can decompose A into the product of the following four augmented turtle transformation matrices shown in table 2:

$$A = L_{NS}(s_1, s_2) * L_{NR}(\alpha_1 - \beta, \alpha_2 - \beta) * L_T(\delta) * L_R(\beta) \quad (9)$$

where $s_1 = \sqrt{a^2 + b^2}$, $s_2 = \sqrt{c^2 + d^2}$, $\delta = \sqrt{e^2 + f^2}$, α_1 is the angle between the vector $\{a, b\}$ and $\{1, 0\}$, α_2 is the angle between the vector $\{c, d\}$ and $\{0, 1\}$, and β is the angle between the vector $\{e, f\}$ and $\{1, 0\}$.

Proposition 3.5 and Proposition 3.6 suggest that the effect of applying any oriented conformal map (or arbitrary non-singular affine transformation) can be reproduced using a sequence of four classical (or augmented) turtle commands: TURN, MOVE, TURN, and SCALE. (Note that the order is reversed, according to Proposition 3.1.) The parameters for each command can be computed by measuring the angles and distances in the canonical coordinate system as shown in figure 7.

Now we can complete the proof of equivalence between RTP and IAT by proving the following two propositions:

Proposition 3.7 *The geometry generated by an IAT using oriented conformal maps can be reproduced by an equivalent RTP using classical turtle commands and states.*

Proof: Let $A_i (i = 1, \dots, n)$ be the affine transformations in the given IAT. Construct the following $n + 1$ matrices:

$$N_1 = A_1, N_i = A_i * A_{i-1}^{-1} (2 \leq i \leq n), N_{n+1} = A_n^{-1}. \quad (10)$$

Since A_i are oriented and conformal, N_i are oriented and conformal. Due to Proposition 3.5, there exists a recursive turtle program P whose recursion body consists of classical turtle commands and whose corresponding matrices satisfy relation (5). Let the base case of P draw the same shape G_0 as the initial geometry F_0 in the IAT, and return to the initial turtle state. Augment

turtle commands to the recursion body, if necessary, to draw the same shape R as the condensation set C in the IAT, without making an overall change to the initial turtle state. Since $N_{n+1} * \dots * N_1 = I$ (where I is the identity matrix), no net change in the turtle state is introduced in each recursive call. Hence we can substitute (10) into (6) and get:

$$M_i = A_i (1 \leq i \leq n).$$

Now compare the geometry F_p and G_p defined respectively by recurrence relations (4) and (7). They have the same base case and recurrence structure; therefore, by induction, $F_p = G_p$ for all $p \geq 0$.

Proposition 3.8 *The geometry generated by an IAT using arbitrary non-singular affine transformations can be reproduced by an equivalent RTP using augmented turtle commands and states.*

Proof: Let $A_i (i = 1, \dots, n)$ denote the affine transformations in the given IAT. Construct $n + 1$ matrices N_i as in (10). Since A_i express non-singular affine transformations, N_i denote non-singular affine maps. Due to Proposition 3.6, there exists a recursive turtle program P whose recursion body consists of augmented turtle commands and whose corresponding matrices satisfy relation (5). The rest of the proof is identical to that of Proposition 3.7.

To construct an equivalent RTP from a given IAT, the key is to decompose the matrices N_i , defined in (10), into transformation matrices corresponding to the classical (or augmented) turtle commands. As an example, we will construct an RTP equivalent to the IAT with conformal maps in section 2.2.3. Here we only demonstrate how to formulate the non-drawing commands in the recursion body, since the base case and the drawing commands in the

recursion body can easily be derived from the initial geometry F_0 and the condensation set C of the IAT. Note that the conversion of an IAT with arbitrary non-singular affine transformations follows closely to the form of the following procedure.

- (1) Construct N_i from the transformations A_i of the IAT as

$$N_1 = A_1, N_2 = A_2 * A_1^{-1}, N_3 = A_2^{-1}$$

- (2) Decompose N_i into turtle commands:

$$N_1: \text{TURN}(\frac{1}{2}\pi), \text{MOVE}(1), \text{TURN}(-\frac{1}{4}\pi), \text{RESIZE}(\frac{\sqrt{2}}{2}).$$

$$N_2: \text{TURN}(-\frac{1}{2}\pi).$$

$$N_3: \text{TURN}(-\frac{1}{4}\pi), \text{MOVE}(\sqrt{2}), \text{TURN}(\frac{1}{2}\pi), \text{RESIZE}(\sqrt{2}).$$

- (3) The recursion body (without the drawing commands) is the conjunction of the turtle commands above and two recursive calls, one after the decomposed commands of N_1 and the other after the decomposed commands of N_2 .

Observe that the resulting RTP differs slightly with the RTP on the right of figure 4. This is because the decomposition of the N_i is not unique. In fact, there are infinitely many ways to decompose N_i into turtle commands, resulting in infinitely many RTPs that are equivalent to a given IAT. Hence the mapping between the set of RTPs and IATs is not one-to-one.

The conversion from an IAT to an equivalent RTP is useful for composing a turtle program that generates a given fractal. It is sometimes difficult to construct the corresponding RTP directly from a given fractal, especially those involving non-conformal transformations. Often, we can compose the equiv-

alent IAT more easily by examining the self-similarity of the fractal shape. Then, using the conversion algorithm presented here, an RTP can be generated automatically from the IAT.

As an example, we will construct an RTP that draws a quadratic Bezier curve, such as the one shown on the right of figure 8. Using De Casteljau's subdivision algorithm, this quadratic Bezier curve can be generated using an IAT that transforms the original control triangle with vertices $(\{0, 0\}, \{1, 0\}, \{0.3, 1\})$ to the control triangles of the first and second halves of the curve. The transformation matrices are

$$A_1 = \begin{pmatrix} 0.4 & 0.5 & 0 \\ 0.03 & 0.35 & 0 \\ 0 & 0 & 1 \end{pmatrix}, A_2 = \begin{pmatrix} 0.6 & -0.5 & 0 \\ 0.07 & 0.15 & 0 \\ 0.4 & 0.5 & 1 \end{pmatrix},$$

along with the condensation set and initial geometry

$$C = \emptyset, F_0 = \text{Line}(\{0, 0\}, \{1, 0\}).$$

The equivalent RTP consists of the augmented turtle commands shown in figure 9. The shapes drawn by this RTP at the first few recursion levels are shown in figure 8 left. Observe that this RTP would be very hard to construct without the help of the corresponding IAT.

4 Discussion

4.1 Net change in turtle state

The conversion algorithm from an RTP to an IAT presented in the previous section is based on the assumption that no net change in the turtle state is introduced in the base case or recursive calls. However, many well-known RTPs, such as the one on the right of figure 11 which produces the C-curve in figure 10, do involve overall change in the turtle state. In fact, as we shall see, this assumption is not absolutely necessary for an RTP to be convertible to an equivalent IAT.

Suppose the base case of an RTP results in an overall transformation O_0 of the turtle's coordinate system. Due to Proposition 3.1, the accumulated transformation O_p of the turtle's coordinate system at the end of the RTP with recursion depth p is computed as

$$O_p = N_{n+1} * O_{p-1} * \dots * N_2 * O_{p-1} * N_1 \quad (11)$$

where n is the number of recursive calls, and N_i are defined as in (5). Hence (6) is reformulated as:

$$M_1^{p+1} = N_1, M_i^{p+1} = N_i * O_p * M_{i-1}^{p+1} (2 \leq i \leq n) \quad (12)$$

Note that the superscript $p + 1$ is added to the transformation matrices M_i in (6) because the transformation of the turtle's coordinate system before each recursive call now depends on the overall changes in the turtle state O_p introduced by the preceding recursive calls, which may vary with the recursion

depth p . Consequently, the recurrence relation for the shape G_{p+1} produced by this RTP is expressed as

$$G_{p+1} = M_1^{p+1}(G_p) \cup \dots \cup M_n^{p+1}(G_p) \cup R \quad (13)$$

Note that, in general, this RTP cannot be converted to an IAT with step-by-step equivalence using the algorithm presented in section 3, because M_i^{p+1} depends on the recursion depth p . However, we have the following observation:

Proposition 4.1 *If $O_1 = O_0$ in (11), then $O_p = O_0$ for every $p > 0$.*

Proof: By induction on p . The statement holds in the case where $p = 1$. Suppose $O_p = O_0$ for all $p \leq k$. From (11) we have:

$$\begin{aligned} O_{k+1} &= N_{n+1} * O_k * \dots * N_2 * O_k * N_1 \\ &= N_{n+1} * O_0 * \dots * N_2 * O_0 * N_1 \\ &= O_1 = O_0 \end{aligned}$$

Hence $O_p = O_0$ for all $p > 0$.

Now observe that in this case ($O_1 = O_0$) M_i^p is independent of p , so we can use the conversion algorithm in section 3 based on equation (13) and draw the following conclusion:

Corollary 4.2 *If a classical (augmented) RTP introduces the same overall change in the turtle state in the base case as in the first level of recursion, the resulting geometry can be reproduced by an IAT using conformal (arbitrary) affine transformations.*

Furthermore, we can show that,

Corollary 4.3 *If $O_1 = O_0$ in (11) for an RTP P , then there exists a different RTP P' which produces the same geometry as P , yet no net change is introduced in either the base case or any recursive call of P' .*

Proof: Due to Corollary 4.2, we can find an IAT equivalent to the RTP P . Then, using the conversion algorithm presented in section 3, an RTP P' can be constructed from this IAT that does not involve any net change in the turtle state.

In fact, we can construct P' from a given P explicitly as follows: Let B denote the sequence of commands in the base case of P , and let B^{-1} denote the turtle commands that return the turtle to its initial state (without drawing) after the commands in B are executed. We can modify P as follows:

- (1) Append to the base case the commands B^{-1} .
- (2) Let \hat{B} be the sequence of commands in B with FORWARDSs replaced by MOVEs. In the recursion body, insert \hat{B} after each recursive call, and append B^{-1} to the end.

Denote the modified turtle program by P' , and let O'_p be the accumulated change in the turtle state in P' with recursion depth p . By construction, we have $O'_0 = O_0^{-1} * O_0 = I$ and

$$\begin{aligned}
 O'_1 &= O_0^{-1} * N_{n+1} * O_0 * O'_0 * \dots * N_2 * O_0 * O'_0 * N_1 \\
 &= O_0^{-1} * N_{n+1} * O_0 * \dots * N_2 * O_0 * N_1 \\
 &= O_0^{-1} * O_1 = I
 \end{aligned}$$

Due to Proposition 4.1, P' introduces no net change in the base case or in any recursive call. Hence the shape generated by P' satisfies the recurrence

relation (7), in which the transformation matrices M_i are defined by:

$$M_1 = N_1, M_i = N_i * O_0 * M_{i-1} (2 \leq i \leq n)$$

Since $O_p = O_0$ for all $p > 0$, $M_i = M_i^p$ for all $p > 0$. Since P' does not alter the geometry drawn in P by the drawing commands in the base case or in the recursion body, and since P and P' satisfy the same recurrence relation, the two programs generate the same shape.

As an example, the RTP on the right of figure 11 produces the same fractal as the RTP to its left, while introducing no net change in the turtle state in the base case or the recursion body. Observe that Proposition 4.1 holds for both classical RTP and augmented RTP. From the previous discussion of the conversion algorithms, we draw the following conclusion:

If $O_1 \neq O_0$, complications arise. Generally, there is no corresponding IAT that generates the same shape as the RTP at every depth p . However, if O_p converges to some fixed transformation O_∞ in the limit, an IAT can be constructed based on the recurrence relation (7) where $M_1 = N_1$ and $M_i = N_i * O_\infty * M_{i-1}$. This IAT defines the same fractal as the RTP in the limit.

If O_p diverges, the geometry generated by the RTP may also diverge, so there may not exist an IAT that is equivalent to the turtle program in the limit. Therefore, we would like to be able to determine the convergence of O_p from (11). The answer to this question will lead to a fully automatic conversion algorithm from an arbitrary RTP to an equivalent IAT.

4.2 *State stack in RTP*

To interpret brackets in an L-System, we can augment an RTP with stack commands such as PUSH and POP. Assuming that the stack commands are well-behaved (i.e., each PUSH is paired with an POP in the same recursion body), adding PUSH and POP does not increase the expressive power of the RTP. These commands do, however, ease the conversion from an IAT because no commands have to be generated to return the turtle to its initial state at the end of the base case or the recursion body. Conversion from an RTP with a state stack to an equivalent IAT proceeds in a similar manner to the algorithm in section 3.3, the only exception is that in (5) and (6), the turtle commands within paired PUSH and POP commands do not contribute to the overall change in the turtle state.

5 **Open Questions**

5.1 *Mutually Recursive Turtle Programs and LR-IAT*

A turtle program may involve multiple recursion bodies and mutually recursive calls, which enables the RTP to interpret a wider range of L-Systems that incorporate multiple production rules. Correspondingly, an IAT (or IFS) can be extended so that different sets of transformations are applied in a well-defined sequence. Culik [6] studied a mutually recursive version of IFS, which are generalized still further by Prusinkiewicz [7] to language-restricted IFS. The composition/decomposition of turtle transformation matrices presented here is well suited for providing conversion algorithms between language-restricted

IATs (LR-IAT) on conformal or arbitrary non-singular affine transformations and mutually recursive turtle programs with classical or augmented turtle states and commands.

5.2 Equivalence between IATs (RTPs)

One question to which we are still seeking an answer, is how to determine the equivalence between two given IATs (or two RTPs). The same fractal can often be generated using IATs with different sets of affine transformations or RTPs with different turtle commands. For example, the Sierpinski Triangle on the left of figure 12 can be generated using an IAT with the three conformal maps:

$$A_1 = A_S(\frac{1}{2}),$$

$$A_2 = A_S(\frac{1}{2}) * A_T(\frac{1}{2}, 0),$$

$$A_3 = A_S(\frac{1}{2}) * A_T(\frac{1}{4}, \frac{\sqrt{3}}{4}).$$

Alternatively, these following transformations produce the same fractal:

$$A_1 = A_S(\frac{1}{2}),$$

$$A_2 = A_S(\frac{1}{2}) * A_R(\frac{2}{3}\pi) * A_T(1, 0),$$

$$A_3 = A_S(\frac{1}{2}) * A_R(\frac{4}{3}\pi) * A_T(\frac{1}{2}, \frac{\sqrt{3}}{2}).$$

So far we can only determine the equivalence between two IATs or two RTPs by rendering the shapes they generate, for example by running the two RTPs. Is it possible to establish the equivalence between two IATs just from examining their affine transformations, or two RTPs by examining their turtle commands?

5.3 Minimal number of transformations (recursive calls)

Fractals can be generated using IATs with different numbers of transformations. For example, the Koch fractal on the right of figure 12 can be generated either using the 4 transformations:

$$\begin{aligned}A_1 &= A_S\left(\frac{1}{3}\right), & A_2 &= A_S\left(\frac{1}{3}\right) * A_R\left(\frac{1}{3}\pi\right) * A_T(1, 0), \\A_3 &= A_S\left(\frac{1}{3}\right) * A_T(2, 0), & A_4 &= A_S\left(\frac{1}{3}\right) * A_R\left(-\frac{1}{3}\pi\right) * A_T\left(\frac{3}{2}, \frac{\sqrt{3}}{2}\right).\end{aligned}$$

or using the 2 transformations:

$$\begin{aligned}A_1 &= A_S\left(\frac{\sqrt{3}}{3}\right) * A_R\left(\frac{7}{6}\pi\right) * A_T\left(\frac{3}{2}, \frac{\sqrt{3}}{2}\right), \\A_2 &= A_S\left(\frac{\sqrt{3}}{3}\right) * A_R\left(\frac{5}{6}\pi\right) * A_T(3, 0).\end{aligned}$$

For a given fractal, how can we find the IAT with the minimum number of transformations or equivalently the RTP with the minimal number of recursive calls?

5.4 IFS and RTP

So far we have considered only IFS with affine transformations. Can we extend the turtle to express fractals generated by an IFS with non-linear complex functions, such as the Julia sets? To accomplish this feat, the turtle programmer would need to know the turtle's location in a global coordinate system to compute the transformation to its next state. In the classical Logo environment, however, the programmer that controls the turtle does not have access to the global position of the turtle. The same turtle commands transform the turtle states in the same way regardless of the turtle's location in the plane.

6 Conclusion

This paper presents rigorous proofs of equivalence and constructs formal conversion algorithms between RTPs composed of classical turtle commands and IATs using oriented conformal maps, as well as between RTPs composed of augmented turtle commands and IATs using arbitrary non-singular affine transformations. The conversion algorithms are useful for the explicit construction of RTPs from given fractal shapes, as well as for the study of the limit shape of an RTP using an equivalent IAT. Our results can be extended to explore equivalences between more complex RTPs (or L-Systems) and IATs (or IFSs). This study also raises some interesting questions to which we are still seeking answers, such as finding necessary and sufficient conditions for the turtle state to converge, or for two IATs or RTPs to generate the same fractal.

References

- [1] M. F. Barnsley, *Fractals Everywhere*, Morgan Kaufmann, 1993.
- [2] H. Abelson, A. A. Disessa, *Turtle Geometry*, MIT Press, 1986.
- [3] B. B. Mandelbrot, *The Fractal Geometry of Nature*, W. H. Freeman, San Francisco, 1982.
- [4] P. Prusinkiewicz, Graphical applications of l-systems, *Proceedings of Graphics Interface 86 and Vision Interface 86 (1986)* 247–253.
- [5] P. Prusinkiewicz, A. Lindenmayer, *The Algorithmic Beauty of Plants*, Springer-Verlag, New York, 1992.

- [6] K. C. II, S. Dube, L-systems and mutually recursive function systems, *Acta Informat.* 30 (1993) 279–302.
- [7] P. Prusinkiewicz, M. Hammel, Language-restricted iterated function systems, koch constructions, and l-systems, *SIGGRAPH 94 Course Notes* .
- [8] J. C. Hart, The object instancing paradigm for linear fractal modeling, *Proceedings of Graphics Interface 92* (1992) 224–231.

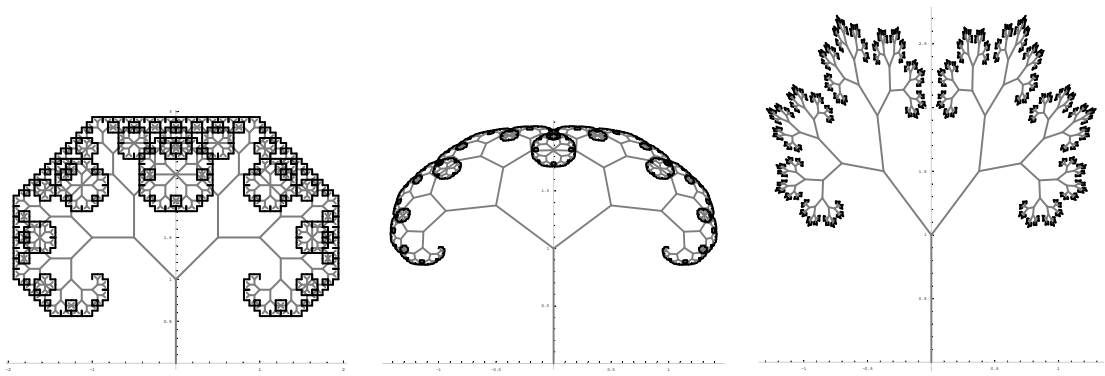


Fig. 1. Self-similar fractals: a fractal tree (left) and two variations.

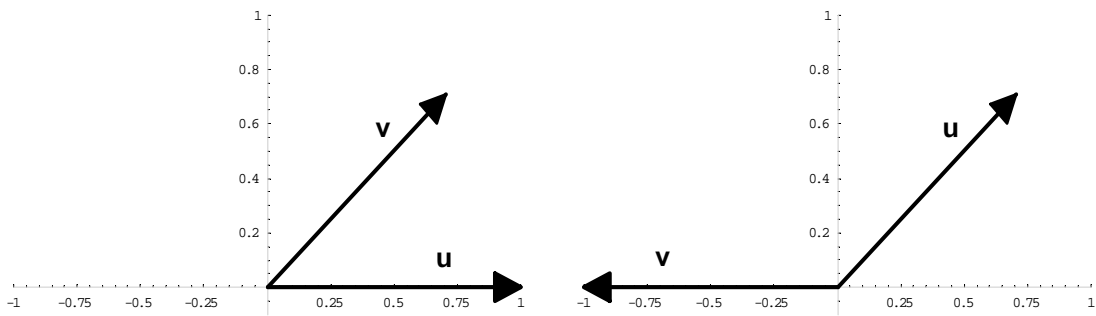


Fig. 2. Applying $\text{TURN}(\frac{1}{2}\pi, \frac{1}{2}\pi)$ to an augmented turtle with a non-orthogonal coordinate system: before (left) and after (right).

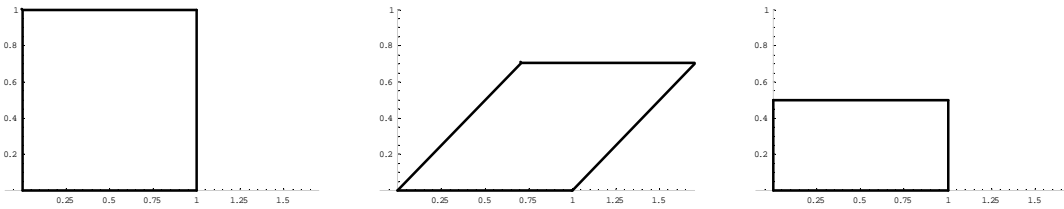
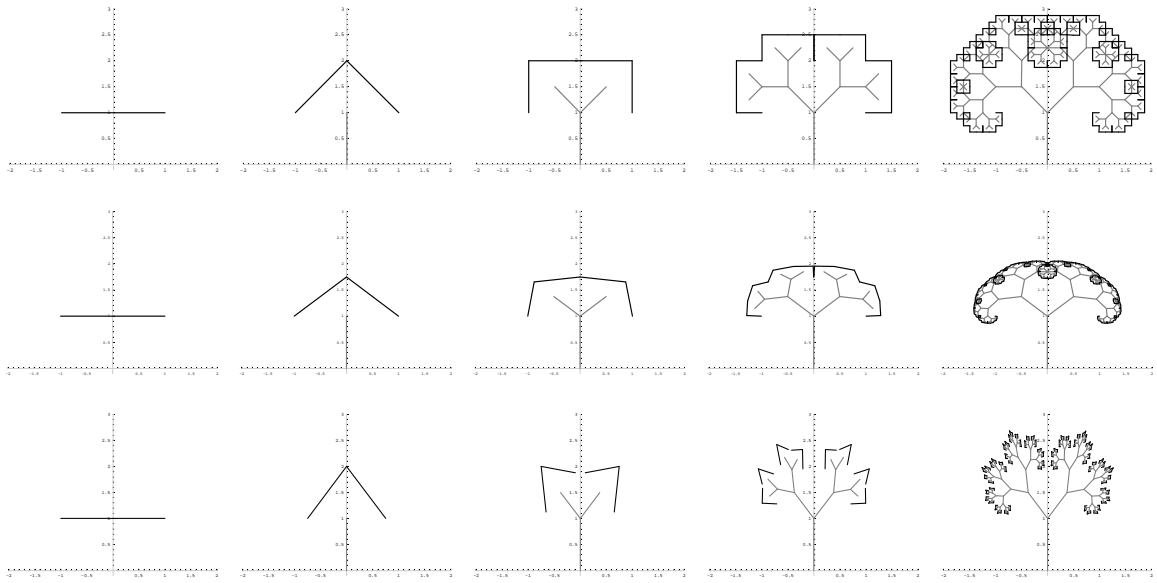


Fig. 3. A square drawn by the classical turtle by repeating the turtle commands FORWARD(1) and TURN($\frac{1}{2}\pi$) four times (left), and the shapes drawn by the augmented turtle by executing the same sequence of commands after a TURN($0, -\frac{1}{4}\pi$) (middle) or a RESIZE(1, 1/2) (right).

Classical Turtle	Augmented Turtle
<i>Base Case:</i>	<i>Base Case:</i>
TURN($\frac{1}{4}\pi$),	TURN($\frac{1}{4}\pi$),
MOVE($\sqrt{2}$),	MOVE($\sqrt{2}$),
TURN($\frac{3}{4}\pi$),	TURN($\frac{3}{4}\pi$),
FORWARD(2),	FORWARD(2),
TURN($\frac{3}{4}\pi$),	TURN($\frac{3}{4}\pi$),
MOVE($\sqrt{2}$),	MOVE($\sqrt{2}$),
TURN($\frac{1}{4}\pi$).	TURN($\frac{1}{4}\pi$).
<i>Recursion Body:</i>	<i>Recursion Body:</i>
TURN($\frac{1}{2}\pi$),	TURN($\frac{1}{2}\pi$),
FORWARD(1),	FORWARD(1),
<u>RESIZE($\frac{\sqrt{2}}{2}$)</u> ,	<u>RESIZE($\alpha\frac{\sqrt{2}}{2}, \beta\frac{\sqrt{2}}{2}$)</u> ,
TURN($-\frac{1}{4}\pi$),	TURN($-\frac{1}{4}\pi$),
Recur,	Recur,
TURN($-\frac{1}{2}\pi$),	TURN($-\frac{1}{2}\pi$),
Recur,	Recur,
TURN($-\frac{1}{4}\pi$),	TURN($-\frac{1}{4}\pi$),
<u>RESIZE($\sqrt{2}$)</u>	<u>RESIZE($\frac{\sqrt{2}}{\alpha}, \frac{\sqrt{2}}{\beta}$)</u>
MOVE(1),	35 MOVE(1),
TURN($\frac{1}{2}\pi$).	TURN($\frac{1}{2}\pi$).

Fig. 4. RTPs that generate different tree-like fractals using the classical turtle (left) and the augmented turtle (right).



level = 0 level = 1 level = 2 level = 4 level = 8

Fig. 5. Tree-like fractals: generated by classical turtle or conformal affine transformations (top), generated by augmented turtle or non-conformal affine transformations with $\alpha = 0.75$ (middle) and $\beta = 0.75$ (bottom).

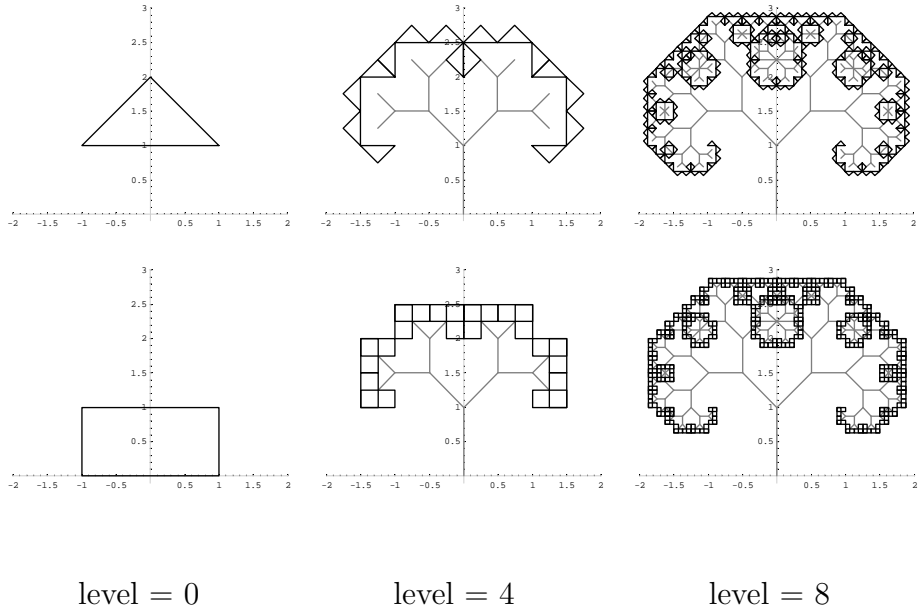


Fig. 6. Same IAT with different initial geometry F_0 : triangle (top) and quadrilateral (bottom).

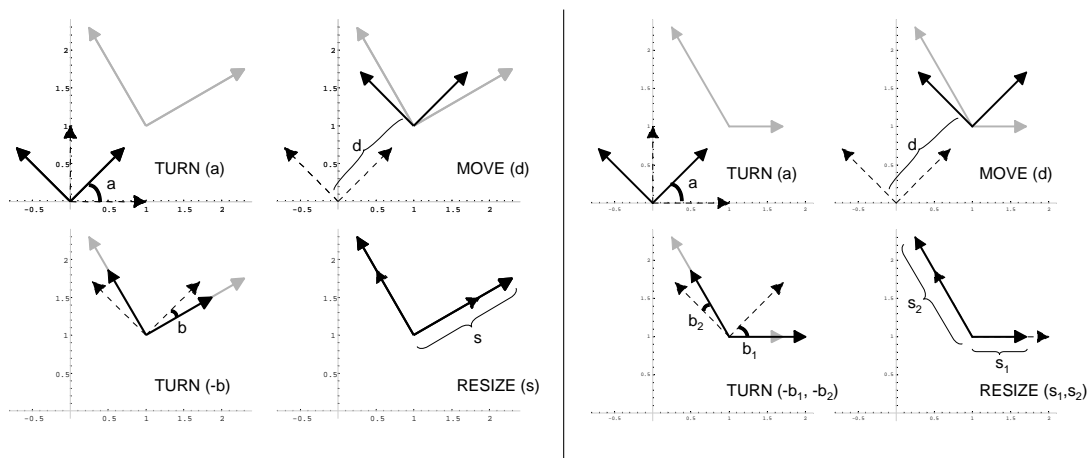


Fig. 7. Reproduce an affine map that transforms the canonical coordinate system to a new coordinate system (colored gray) with turtle commands: oriented conformal map with four classical turtle commands (left) and arbitrary non-singular affine map with four augmented turtle commands (right).

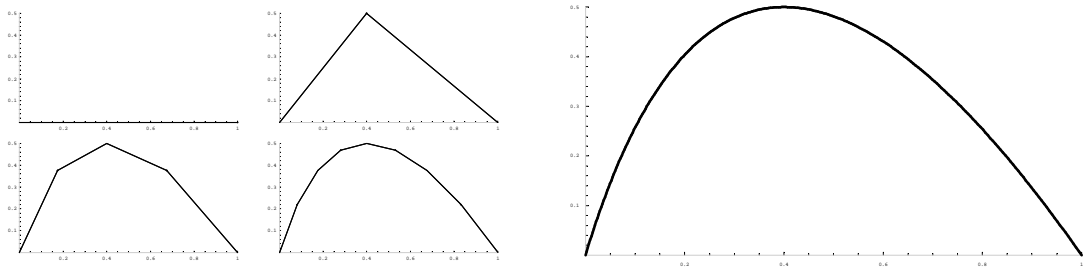


Fig. 8. A quadratic Bezier curve generated by an RTP: recursion level 0 to 3(left), recursion level 8(right).

Base Case	Recursion Body
FORWARD(1).	TURN(0.896055, -0.0855053),
TURN(π),	RESIZE(0.640312, 0.351283),
MOVE(1),	Recur,
TURN(π),	MOVE(1),
	TURN(-1.14794, -0.674741),
	RESIZE(4.38634, 0.256125),
	Recur,
	TURN(-1.62075),
	MOVE(4.005),
	TURN(2.90009, 1.7369),
	RESIZE(4.17612, 4.83256),

Fig. 9. RTP that generates a quadratic Bezier curve.

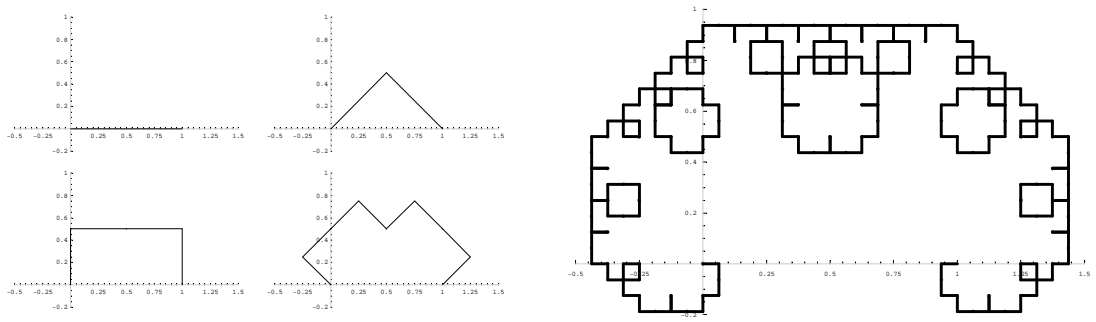


Fig. 10. A fractal C-curve generated by an RTP: recursion level 0 to 3(left), recursion level 8(right).

With net change	No net change
-----------------	---------------

Base Case:

FORWARD(1).

Base Case:

FORWARD(1),

TURN(π),

MOVE(1),

TURN(π).

Recursion Body:

TURN($\frac{1}{4}\pi$),

RESIZE($\frac{\sqrt{2}}{2}$),

Recur,

TURN($-\frac{1}{4}\pi$),

Recur,

TURN($\frac{1}{4}\pi$),

RESIZE($\sqrt{2}$).

Recursion Body:

TURN($\frac{1}{4}\pi$),

RESIZE($\frac{\sqrt{2}}{2}$),

Recur,

MOVE(1),

TURN($-\frac{1}{4}\pi$),

Recur,

MOVE(1),

TURN($\frac{1}{4}\pi$),

RESIZE($\sqrt{2}$),

TURN(π),

MOVE(1),

42 TURN(π).

Fig. 11. RTP that generates the C-curve in figure 10 (left), and the equivalent RTP introducing no net change in the turtle state (right).

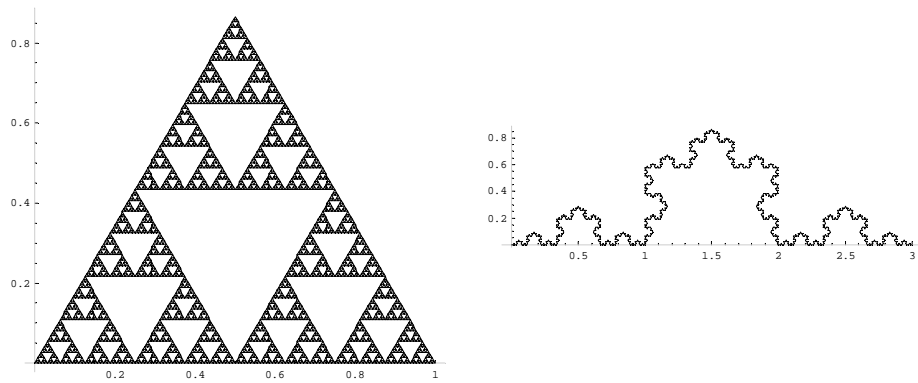


Fig. 12. Fractals with multiple IATs: the Sierpinski triangle(left) and the Koch bump(right).

FORWARD(d), MOVE(d)	$L_T(d) = \begin{Bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ d & 0 & 1 \end{Bmatrix}$
TURN(α)	$L_R(\alpha) = \begin{Bmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{Bmatrix}$
RESIZE(s)	$L_S(s) = \begin{Bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{Bmatrix}$

Table 1

Transformation matrices for classical turtle commands

TURN(α_1, α_2)	$L_{NR}(\alpha_1, \alpha_2) =$	$\begin{Bmatrix} \cos(\alpha_1) & \sin(\alpha_1) & 0 \\ -\sin(\alpha_2) & \cos(\alpha_2) & 0 \\ 0 & 0 & 1 \end{Bmatrix}$
RESIZE(s_1, s_2)	$L_{NS}(s_1, s_2) =$	$\begin{Bmatrix} s_1 & 0 & 0 \\ 0 & s_2 & 0 \\ 0 & 0 & 1 \end{Bmatrix}$

Table 2

Augmented transformation matrices for augmented turtle commands

Translation	$A_T(dx, dy) = \begin{Bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ dx & dy & 1 \end{Bmatrix}$
Rotation	$A_R(\alpha) = \begin{Bmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{Bmatrix}$
Uniform Scaling	$A_S(s) = \begin{Bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{Bmatrix}$
Non-Uniform Scaling	$A_{NS}(sx, sy) = \begin{Bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{Bmatrix}$
Shearing	$A_H(hx, hy) = \begin{Bmatrix} 1 & hy & 0 \\ hx & 1 & 0 \\ 0 & 0 & 1 \end{Bmatrix}$

Table 3

Transformation matrices for affine transformations.