# Lecture 8: Recursive Turtle Programs and Conformal Iterated Function Systems

*convert, and be healed.* Isaiah 6:10

## 1. Motivating Questions

We have studied two methods for generating fractals: recursive turtle programs and iterated function systems. We have also examined many fractal curves that can be generated by both methods. Whenever there are two seemingly dissimilar approaches to accomplishing the same task, questions naturally arise concerning how the two techniques are related. For fractals we ask:

1. Is it always true that a fractal generated by one method can be generated by the other technique?
2. Do there exist algorithms that convert recursive turtle programs into iterated functions systems?
3. Do there exist algorithms that convert iterated function systems into recursive turtle programs?

The turtle commands FORWARD, MOVE, TURN, and RESIZE embody the conformal transformations TRANSLATE, ROTATE, and SCALE. Since there exist affine transformations such as non-uniform scaling which are not conformal, there exist fractals that can be generated by iterated function systems that cannot be generated by recursive turtle programs. But as long as we restrict our attention to iterated function systems consisting solely of conformal transformations, then the answer to all three questions is yes.

The purpose of this lecture is to establish these results by investigating the connections between recursive turtle programs and conformal iterated function systems. Using the equivalence of recursive turtle programs and conformal iterated function systems, we shall also establish that the fractals generated by recursive turtle programs are independent of the base case.

## 2. The Effect of Changing the Turtle's Initial State

A (non-recursive) turtle program is simply a finite sequence of FORWARD, MOVE, TURN, and RESIZE commands. In this section we are going to study the effect of changing the turtle's initial state on the geometry generated by a turtle program. The insights we shall develop here are actually the keys to understanding the relationship between recursive turtle programs and conformal iterated function systems.

The turtle knows only her state $(P,v)$, where $P$ is her position and $v$ her direction vector. Besides sometimes drawing a line, all that the turtle commands really do is change the state of the turtle. Now consider two possible turtle states: $S_1 = (P_1, v_1)$ and $S_2 = (P_2, v_2)$. Is it always possible for the turtle to get from one state to the other? Sure. In fact, we can see from Figure 1 that there is always a sequence of four turtle commands --

*TURN(A), MOVE(D), TURN($-B$), RESIZE(R)*

-- that maps the turtle from $S_1$ to $S_2$. Similarly, there is a sequence of three conformal transformations consisting of translation, rotation, and uniform scaling that maps $S_1$ to $S_2$.

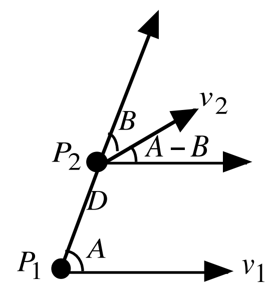

**Figure 1:** A pair of turtle states: $S_1 = (P_1, v_1)$ and $S_2 = (P_2, v_2)$. The turtle can get from $S_1$ to $S_2$ by executing the commands: *TURN(A), MOVE(D), TURN($-B$), RESIZE(R),* where $R = length(v_2)/length(v_1)$. Similarly, executing consecutively the three conformal transformations $TRANS(P_2 - P_1)$, $ROT(A-B)$, $SCALE(R)$ maps $S_1$ to $S_2$.

Now we are going to make two key observations that will help guide us to an understanding of the relationships between recursive turtle programs and conformal iterated function systems.

**Lemma 1:** *Suppose that*

$S_1, S_2$ = *two turtle states*

$T_1, T_2$ = *two turtle programs that differ only by their initial states* $S_1, S_2$

$G_1, G_2$ = *geometry generated by* $T_1, T_2$

$M$ = *conformal transformation that maps* $S_1$ *to* $S_2$

*Then*

$G_2 = M(G_1)$

Proof: Clearly if we change the turtle's initial state by, for example, a TURN($A$) command, the turtle will still draw exactly the same shape, but rotated by $A$ radians. Similarly, MOVE($D$) will translate the shape and RESIZE($R$) will scale the shape. But we have just seen that we can get from any turtle state to any other turtle state by a sequence of four such turtle commands. Thus if $M$ is the affine transformation corresponding to these four turtle commands and $G_1, G_2$ are the geometry generated by the turtle programs $T_1, T_2$, then $G_2 = M(G_1)$.

The effects of the turtle programs SHIFT, SPIN, and SCALE are examples of Lemma 1 in action.

**Lemma 2:** *Suppose that*

$S_1$ = *initial turtle state*

$T_1$ = *turtle program*

$P$ = *sequence of MOVE, TURN, and RESIZE commands (Turtle Program)*

$T_2 = (P, T_1)$ *(do P first, then do* $T_1$*)*

$G_i$ = *geometry generated by* $T_i$ $\quad i = 1,2$

*If*

$P{:}S_1 \to S_2$

$M{:}S_1 \to S_2$ *(conformal transformation)*

*then*

$G_2 = M(G_1)$

Proof: The only effect of the turtle program $P$ is to change the state of the turtle from state $S_1$ to state $S_2$. Therefore this result is a consequence of Lemma 1.

## 3. Equivalence Theorems

We are now ready to establish our main results concerning the equivalence of recursive turtle programs and conformal iterated functions systems.

**Theorem 1:** *For every recursive turtle program (RTP), there exists a conformal iterated function system (CIFS) such that starting with the same base case, level for level the RTP and the CIFS generate exactly the same geometry.*

Proof: Consider a recursive turtle program:

- Base Case: Turtle Program ($T_B$)
- Recursion:

  $Turtle\ Commands(T_1), Turtle\ Recursion$

  $\vdots$

  $Turtle\ Commands(T_p), Turtle\ Recursion$

  $Turtle\ Commands(T_{p+1})$

Let $G_R$ be the geometry drawn by the recursion without the recursive calls -- that is, $G_R$ is the geometry drawn by the turtle program $T = T_1, \ldots, T_{p+1}$. Suppose that

$S_0$ = the initial turtle state;

$S_k$ = the turtle state immediately after executing the commands $T_k$.

Let $M_k$ be the conformal transformation that maps $S_0$ to $S_k$ -- that is,

$$M_k\colon\ S_0 \to S_k.$$

Define a conformal iterated function system by setting

$$CIFS = \{M_1,\ldots,M_p\}$$

$$\text{Condensation Set} = G_R.$$

We shall show that starting with the same base case, level for level the RTP and this CIFS generate exactly the same geometry.

Let $G_B$ be the geometry drawn in base case -- that is, the geometry drawn by $T_B$ -- and let $L_n$ be the geometry drawn by the *nth* level of the recursive turtle program. The key insight is that at the $(n+1)^{st}$ level, every recursive call draws the geometry $L_n$, but starting from a different initial state. Thus by Lemma 2

$$L_0 = G_B$$

$$L_{n+1} = M_1(L_n) \cup \cdots \cup M_p(L_n) \cup G_R$$

On the other hand, let $F_n$ be the geometry drawn by the *nth* iteration of the iterated function system. If we start the iteration with the base case $G_B$, then by the definition of an IFS with a condensation set,

$$F_0 = G_B$$

$$F_{n+1} = M_1(F_n) \cup \cdots \cup M_p(F_n) \cup G_R$$

Therefore, by induction, level for level the recursive turtle program and the conformal iterated function system generate exactly the same geometry.

**Corollary 1:** *Every fractal generated by a recursive turtle program can be generated by an iterated function system consisting solely of conformal transformations. Moreover, the number of conformal transformations in the CIFS is equal to the number of recursive calls in the RTP.*

Proof: This result is an immediate consequence of the statement and proof of Theorem 1.

**Corollary 2:** *The fractal drawn by a recursive turtle program is independent of the geometry drawn in the base case.*

Proof: This result follows from Theorem 1 because by the Fractal Theorem in Lecture 7, the fractal generated by an IFS is independent of the geometry in the base case.

**Theorem 2:** *For every iterated function system consisting solely of conformal transformations (CIFS), there exists a recursive turtle program (RTP) such that starting with the same base case, level for level the CIFS and the RTP generate exactly the same geometry.*

Proof: Given a conformal iterated function system

$$CIFS = \{M_1, \dots, M_p\}$$

$$\text{Condensation Set} = G_R$$

we seek a recursive turtle program that generates, level for level, the same curves as this IFS. Fix an initial turtle state $S_0$, and let $S_k$ be the turtle state into which $S_0$ is mapped by the conformal transformation $M_k$. From our observations in Section 2, we know that for every $k$ there is a sequence of turtle commands $T_k$ consisting of MOVE, TURN, and RESIZE commands that map $S_{k-1}$ to $S_k$. We will use these turtle commands to transition between recursive calls. To finish the recursive turtle program, let $T_{p+1}$ be the turtle commands that draw the condensation set $G_R$ and return the turtle to her initial state. For the base case of the recursive turtle program, we can choose any turtle program $T_B$ that draws some geometry $G_B$ and returns the turtle to her initial state $S_0$. We claim that the following recursive turtle program generates, level for level, the same geometry as the given CIFS:

- Base Case: Turtle Program ($T_B$)
- Recursion:

$$\begin{aligned}
&\textit{Turtle Commands}(T_1), \textit{Turtle Recursion}\\
&\qquad\qquad\vdots\\
&\textit{Turtle Commands}(T_p), \textit{Turtle Recursion}\\
&\textit{Turtle Commands}(T_{p+1})
\end{aligned}$$

Let $L_n$ be the geometry drawn by the *nth* level of this recursive turtle program. Again the key insight is that at the $(n+1)^{st}$ level, the *kth* recursive call draws the geometry $L_n$, but starting from the initial state $S_k$. Since $M_k$: $S_0 \to S_k$, it follows by Lemma 2 that

$$L_0 = G_B$$

$$L_{n+1} = M_1(L_n) \cup \cdots \cup M_p(L_n) \cup G_R$$

On the other hand, let $F_n$ be the geometry drawn by the *nth* iteration of the conformal iterated function system. If we start the iteration with the base case $G_B$, then by the definition of an IFS,

$$F_0 = G_B$$

$$F_{n+1} = M_1(F_n) \cup \cdots \cup M_p(F_n) \cup G_R$$

Therefore, by induction, level for level, the recursive turtle program and the conformal iterated function system generate exactly the same geometry.

**Corollary 3:** *Every fractal generated by a conformal iterated function system can be generated by a recursive turtle program whose initial and final states are identical both in the base case and in the recursion.*
Proof: This result is an immediate consequence of the statement and proof of Theorem 2.

**Corollary 4:** *Every fractal generated by a recursive turtle program can be generated by a recursive turtle program whose initial and final states are identical both in the base case and in the recursion.*
Proof: By Corollary 1 we know that every fractal generated by a recursive turtle program can be generated by a conformal iterated function system, and by Corollary 3 we know that every fractal generated by a conformal iterated function system can be generated by a recursive turtle program whose initial and final states are identical.

## 4. Conversion Algorithms

Although we have just proved that every recursive turtle program (RTP) is equivalent to a conformal iterated function system (CIFS), we would like to have an explicit algorithm for finding this CIFS from a given RTP. Below we present two such explicit conversion algorithms: Ron's algorithm and Tao's algorithm.

Ron's Algorithm takes an exocentric point of view, a perspective from outside the turtle's domain. Suppose that the turtle is in some state $S=(P,v)$. Viewed from outside the turtle, each turtle command corresponds to a conformal transformation:

$$FORWARD\ D \leftrightarrow TRANS(v * SCALE(D))$$

$$MOVE\ D \leftrightarrow TRANS(v * SCALE(D))$$

$$TURN\ A \leftrightarrow ROT(P,A)$$

$$RESIZE\ S \leftrightarrow SCALE(P,S).$$

Ron's algorithm proceeds in the following fashion.

**<u>Ron's Algorithm: RTP → CIFS</u>**

1. *Preprocessing*
   a. For each turtle command, compute the corresponding conformal transformation.
   b. If the turtle changes state during the recursion, compute the conformal transformation corresponding to the state change caused by the recursion.

2. *Finding the CIFS*
   a. $N_k$ = product of the transformations between the $(k-1)^{st}$ and $k^{th}$ recursive call.
   b. $M_1 = N_1$ = product of the transformations <u>before</u> the first recursive call.
   c. $M_k = M_{k-1} * N_k$
   d. $CIFS = \{M_1, \ldots M_p\}$ $\quad$ $\{p$ = number of recursive calls$\}$

**Example 1: Sierpinski Gasket -- Ron's Method**

*Outer Triangle*

Vertices = $(P_1, P_2, P_3)$

Edges = $(w_1, w_2, w_3)$

```
Recursive Turtle Program                  Transformations

BASE CASE:  TRIANGLE
RECURSION:
    REPEAT 3 TIMES
        RESIZE 1/2          <->           $SCALE(P_k, 1/2)$
        RECUR
        RESIZE 2            <->           $SCALE(P_k, 2)$
        MOVE 1              <->           $TRANS(w_k)$
        TURN $2\pi/3$       <->           $ROT(P_{k+1}, 2\pi/3)$
```

*<u>Conformal Iterated Function System</u>*

$$M_1 = SCALE(P_1, 1/2)$$

$$\begin{aligned} M_2 &= M_1 * SCALE(P_1, 2) * TRANS(w_1) * ROT(P_2, 2\pi/3) * SCALE(P_2, 1/2) \\ &= TRANS(w_1) * ROT(P_2, 2\pi/3) * SCALE(P_2, 1/2) \end{aligned}$$

$$\begin{aligned} M_3 &= M_2 * SCALE(P_2, 2) * TRANS(w_2) * ROT(P_3, 2\pi/3) * SCALE(P_3, 1/2) \\ &= TRANS(w_1) * ROT(P_2, 2\pi/3) * TRANS(w_2) * ROT(P_3, 2\pi/3) * SCALE(P_3, 1/2) \end{aligned}$$

Notice that in Ron's algorithm, the transformations depend on the turtle's current state. Thus Ron's algorithm has the annoying property that it must keep track of the turtle's state after each turtle command. For the Sierpinski gasket, keeping track of the turtle's state is easy, but for many other fractals it is not so straightforward to keep track of the turtle's state.

Tao's algorithm takes a turtle-centric point of view, a perspective from inside the turtle's domain. From the turtle's viewpoint, the turtle is always at the center of the universe (the origin) facing in the direction of the *x*-axis (*ivec*), so there is no need to keep track of the turtle's state. Therefore, viewed from inside the turtle, the turtle commands correspond to the following conformal transformations:

$$FORWARD\ \ D \ \leftrightarrow\ TRANS\big(ivec * SCALE(D)\big)$$
$$MOVE\ \ D \ \leftrightarrow\ TRANS\big(ivec * SCALE(D)\big)$$
$$TURN\ \ A \ \leftrightarrow\ ROT(Origin,\ A) = ROT(A)$$
$$RESIZE\ \ s \ \leftrightarrow\ SCALE(Origin,\ s) = SCALE(s).$$

There is one novel detail here. When we apply transformation matrices to compute the new coordinates of a point or vector, we multiply on the right: $Q_{new} = Q_{old} * M$. But when we apply these transformations to change the state of the turtle -- the turtle's local coordinate system -- then we must multiply on the left. For suppose that the turtle is in the state $S = (P, v)$. Then

$$\begin{matrix} FORWARD\ \ D \\ MOVE\ \ D \end{matrix} \ \Leftrightarrow\ \begin{matrix} v_{new} = v \\ v_{new}^{\perp} = v^{\perp} \\ P_{new} = P + D\,v \end{matrix} \ \Leftrightarrow\ \begin{pmatrix} v_{new} \\ v_{new}^{\perp} \\ P_{new} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ D & 0 & 1 \end{pmatrix} * \begin{pmatrix} v \\ v^{\perp} \\ P \end{pmatrix}$$

$$RESIZE\ \ s \ \Leftrightarrow\ \begin{matrix} v_{new} = s\,v \\ v_{new}^{\perp} = s\,v^{\perp} \\ P_{new} = P \end{matrix} \ \Leftrightarrow\ \begin{pmatrix} v_{new} \\ v_{new}^{\perp} \\ P_{new} \end{pmatrix} = \begin{pmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} v \\ v^{\perp} \\ P \end{pmatrix}$$

$$TURN\ \ A \ \Leftrightarrow\ \begin{matrix} v_{new} = \cos(A)\,v + \sin(A)\,v^{\perp} \\ v_{new}^{\perp} = -\sin(A)\,v + \cos(A)\,v^{\perp} \\ P_{new} = P \end{matrix} \ \Leftrightarrow\ \begin{pmatrix} v_{new} \\ v_{new}^{\perp} \\ P_{new} \end{pmatrix} = \begin{pmatrix} \cos(A) & \sin(A) & 0 \\ -\sin(A) & \cos(A) & 0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} v \\ v^{\perp} \\ P \end{pmatrix}$$

These observations lead to the following result.

**Proposition:** *If $L_1,\ldots,L_n$ are the transformation matrices corresponding to the turtle commands $C_1,\ldots,C_n$, then $L = L_n * \cdots * L_1$ is the transformation matrix corresponding to the turtle program $C_1,\ldots,C_n$.*

Proof: Let $S_k$ denote the state of the turtle after executing the command $C_k$. Since $L_k * S_{k-1} = S_k$, it follows that $L * S_0 = (L_n * \cdots * L_1) * S_0 = S_n$.

**<u>Tao's Algorithm: RTP → CIFS</u>**

1. *Preprocessing*
   a. For each turtle command, compute the corresponding conformal transformation.
   b. If the turtle changes state during the recursion, compute the conformal transformation corresponding to the state change caused by the recursion.
2. *Finding the IFS*
   *a*. $N_k$ = product of transformations between the $(k-1)^{st}$ and $k^{th}$ recursive call in reverse order (right to left, not left to right)
   b. $M_1 = N_1$ = product of transformations <u>before</u> the first recursive call
   c. $M_k = N_k * M_{k-1}$
   d. $CIFS = \{M_1, \ldots M_p\}$ $\{p$ = number of recursive calls$\}$

Note that all rotation and scaling is about the origin and all translation is in the *x*-direction.

**Example 2: Sierpinski Gasket -- Tao's Method**

| <u>*Recursive Turtle Program*</u> | | <u>*Transformations*</u> |
|---|---|---|
| BASE CASE: TRIANGLE | | |
| RECURSION: | | |
| REPEAT 3 TIMES | | |
| RESIZE 1/2 | ↔ | $SCALE(1/2)$ |
| RECUR | | |
| RESIZE 2 | ↔ | $SCALE(2)$ |
| MOVE 1 | ↔ | $TRANS(ivec)$ |
| TURN $2\pi/3$ | ↔ | $ROT(2\pi/3)$ |

<u>*Iterated Function System*</u>

$$M_1 = SCALE(1/2)$$

$$M_2 = SCALE(1/2) * ROT(2\pi/3) * TRANS(ivec) * SCALE(2) * M_1$$
$$= SCALE(1/2) * ROT(2\pi/3) * TRANS(ivec)$$

$$M_3 = SCALE(1/2) * ROT(2\pi/3) * TRANS(ivec) * SCALE(2) * M_2$$
$$= SCALE(1/2) * ROT(2\pi/3) * TRANS(ivec) * ROT(2\pi/3) * TRANS(ivec)$$

Note that in Tao's method, unlike Ron's method, there is no need to keep track of the turtle's state! Nevertheless both Tao's method and Ron's method generate the identical IFS.

## 5. Bump Fractals

The easiest way to describe a bump is with a turtle program. When we studied bump fractals, we observed that we could generate the recursive turtle program for a bump fractal directly from the turtle program for the bump. We would like to do the same for iterated function systems: that is, we would like to extract the IFS for the bump fractal directly from a simple turtle description of the bump. Since we know how to get from a turtle description of a bump to a recursive turtle program for the bump fractal and since we know how to convert from a recursive turtle program to an iterated function system, all we need to do to get from a turtle description of a bump to an iterated function system for the bump fractal is to combine these two procedures. We exhibit this algorithm in Table 1. In order to avoid keeping track of the turtle's state, we use Tao's algorithm to convert from a recursive turtle program to an iterated function system.

| Bump | | Fractal (*Level*) | | CIFS |
|---|---|---|---|---|
| | | *Base Case:* FORWARD 1 | | |
| | | *Recursion:* | | |
| TURN $A_1$ | $\rightarrow$ | TURN $A_1$ | $\rightarrow$ | $ROT(A_1)$ |
| RESIZE $S_1$ | $\rightarrow$ | RESIZE $S_1$ | $\rightarrow$ | $SCALE(S_1)$ |
| FORWARD 1 | $\rightarrow$ | Fractal $(Level-1)$ | $\rightarrow$ | $TRANS(ivec)$ |
| TURN $A_2$ | $\rightarrow$ | TURN $A_2$ | $\rightarrow$ | $ROT(A_2)$ |
| RESIZE $S_2$ | $\rightarrow$ | RESIZE $S_2$ | $\rightarrow$ | $SCALE(S_2)$ |
| FORWARD 1 | $\rightarrow$ | Fractal $(Level-1)$ | $\rightarrow$ | $TRANS(ivec)$ |
| ⋮ | | ⋮ | | |
| TURN $A_{n+1}$ | $\rightarrow$ | TURN $A_n$ | $\rightarrow$ | $ROT(A_n)$ |
| RESIZE $S_{n+1}$ | $\rightarrow$ | RESIZE $S_n$ | $\rightarrow$ | $SCALE(S_n)$ |
| FORWARD 1 | $\rightarrow$ | Fractal $(Level-1)$ | | |
| TURN $A_{n+1}$ | $\rightarrow$ | TURN $A_{n+1}$ | | |
| RESIZE $S_{n+1}$ | $\rightarrow$ | RESIZE $S_{n+1}$ | | |

**Table 1:** Bump $\rightarrow$ Recursive Turtle Program $\rightarrow$ CIFS. All the rotation and scaling commands are around the origin. The last three bump commands -- FORWARD 1, TURN $A_{n+1}$, and RESIZE $S_{n+1}$ -- are ignored by the iterated function system, since the only purpose of these commands in the bump program is to conduct the turtle to her final state. To read off the IFS from the transformations in the last column, simply write down the transformations in reverse order and group them between successive appearances of $Trans(ivec)$ (see Example 3).

**Example 3: Koch Curve**

| Koch IFS | | Koch Bump | | Koch (*Level*) |
|---|---|---|---|---|
| | | | | *Base Case:* FORWARD 1 |
| | | | | *Recursion:* |
| $SCALE(1/3)$ | $\leftarrow$ | RESIZE 1/3 | $\rightarrow$ | RESIZE 1/3 |
| $TRANS(ivec)$ | $\leftarrow$ | FORWARD 1 | $\rightarrow$ | Koch (*Level-1)* |
| $ROT(\pi/3)$ | $\leftarrow$ | TURN $\pi/3$ | $\rightarrow$ | TURN $\pi/3$ |
| $TRANS(ivec)$ | $\leftarrow$ | FORWARD 1 | $\rightarrow$ | Koch (*Level-1)* |
| $ROT(-2\pi/3)$ | $\leftarrow$ | TURN $-2\pi/3$ | $\rightarrow$ | TURN $-2\pi/3$ |
| $TRANS(ivec)$ | $\leftarrow$ | FORWARD 1 | $\rightarrow$ | Koch (*Level-1)* |
| $ROT(\pi/3)$ | $\leftarrow$ | TURN $\pi/3$ | $\rightarrow$ | TURN $\pi/3$ |
| | | FORWARD 1 | $\rightarrow$ | Koch (*Level-1)* |
| | | RESIZE 3 | $\rightarrow$ | RESIZE 3 |

$$\underbrace{Rot(\pi/3)*Trans(ivec)*\underbrace{Rot(-2\pi/3)*Trans(ivec)*\underbrace{Rot(\pi/3)*Trans(ivec)*\underbrace{Scale(1/3)}_{M_1}}_{M_2}}_{M_3}}_{M_4}$$

A CIFS for the Koch curve is given by the transformations $\{M_1, M_2, M_3, M_4\}$. To find these transformations, we list in column 1 the transformations corresponding to the turtle program in column 2 for the Koch bump. We then write these transformations horizontally in reverse order and group these products between successive appearance of $Trans(ivec)$.

## 6. Summary

The main result of this lecture is the equivalence between recursive turtle programs (RTP) and conformal iterated function systems (CIFS). The conformal transformations in the CIFS correspond to changes of state between successive recursive calls in the RTP; the condensation set of the CIFS is the geometry drawn during highest level of the recursive phase of the RTP. These and other explicit correspondences between recursive turtle programs and conformal iterated functions systems are summarized in Table 2. Since, by the trivial fixed point theorem, the fractal generated by an iterated function system is independent of the base case, it follows from our equivalence theorems that the fractal generated by a recursive turtle program is also independent of the base case.

| *CIFS* | *RTP* |
|---|---|
| Number of Transformations | Number of Recursive Calls |
| Transformations | Changes of State between Successive Recursive Calls |
| Base Case | Geometry Drawn in the Base Case |
| Condensation Set | Geometry Drawn at the Highest Level of Recursion |

**Table 2:** Correspondences between the CIFS and the RTP that generate the same fractal.

We presented two explicit algorithms to convert from a recursive turtle program to an equivalent conformal iterated function system: Ron's algorithm and Tao's algorithm. Tao's algorithm was used to generate the iterated function system for a bump fractal directly from a turtle program for the bump. Below we compare and contrast the main features of these two algorithms.

*Ron's Algorithm (Exo-Centric)*

- Matrix products are taken in order from left to right.
- Needs to keep track of the turtle's state.
- All transformations are rooted at the turtle's current state.

*Tao's Algorithm (Turtle-Centric)*

- Matrix products are taken in reverse order from right to left.
- Does not need to keep track of the turtle's state.
- All transformation are rooted at the origin and along the $x$-axis.

**Exercises:**

1. Develop an explicit algorithm to convert a conformal iterated function system to an equivalent recursive turtle program.

2. The text presents two algorithms to convert a recursive turtle program to an equivalent conformal iterated function system: Ron's algorithm and Tao's algorithm.
   a. Verify that Ron's method and Tao's method generate the same CIFS for the Sierpinski gasket.
   b. Does Ron's method always give the same CIFS as Tao's method? Prove or give a counterexample.

3. Is the CIFS for the Koch curve presented in Example 3 the same as the CIFS for the Koch curve presented in Lecture 6, Exercise 6?

4. Consider the bumps in Lecture 2, Figure 9.
    a. Write a turtle program to generate each of these bumps.
    b. Write a recursive turtle program to generate the bump fractals corresponding to each of these bumps.
    c. Describe the transformations in the conformal iterated function systems that generate each of these bump fractals.

5. Consider the bumps in Lecture 2, Figure 13.
    a. Write a turtle program to generate each of these bumps.
    b. Write a recursive turtle program to generate the bump fractals corresponding to each of these bumps.
    c. Describe the transformations in the conformal iterated function systems that generate each of these bump fractals.

6. Consider the bumps in Lecture 2, Figure 14.
    a. Write a turtle program to generate each of these bumps.
    b. Write a recursive turtle program to generate the bump fractals corresponding to each of these bumps.
    c. Describe the transformations in the conformal iterated function systems that generate each of these bump fractals.

7. Consider the bumps in Lecture 2, Figure 15.
    a. Write a turtle program to generate each of these bumps.
    b. Write a recursive turtle program to generate the bump fractals corresponding to each of these bumps.
    c. Describe the transformations in the conformal iterated function systems that generate each of these bump fractals.

8. Consider the iterated function systems for the Sierpinski gasket generated from the standard recursive turtle program for the Sierpinski gasket by Ron's Algorithm (Example 1) and by Tao's Algorithm (Example 2).
    a. Show that these two iterated function systems are identical.
    b. Show that these two iterated function systems are not identical to the standard iterated function system for the Sierpinski gasket given by the three scaling transformations $Scale(P_1,1/2)$, $Scale(P_2,1/2)$, $Scale(P_3,1/2)$.
    c. Conclude from part *b* that the IFS for generating a fractal is not unique.
    d. Construct a recursive turtle program that corresponds to the standard iterated function system for the Sierpinski gasket given in part *b*.

9. Prove that the classical turtle and the hodograph turtle (see Lecture 2, Project 2) generate the same fractal when both of the following conditions are satisfied:

   i. the pen is down and the anchor is up during the base case;
   ii. both the pen and the anchor are up during the recursive body of the program.

10. Consider the left-handed turtle described in Lecture 2, Project 4.
    a. Show that if we replace the classical turtle by the left-handed turtle and conformal transformations by arbitrary affine transformations, then
       i. Lemma 1 and Lemma 2 remain valid.
       ii. Theorems 1, 2 and Corollaries 1-4 remain valid.
    b. Explain how to extend Tao's algorithm to convert recursive turtle programs for the left-handed turtle to iterated functions systems consisting of arbitrary affine transformations.

11. Using an iterated function system with affine transformations:
    a. Build a fractal that cannot be generated by a recursive turtle program for the classical turtle.
    b. Construct a recursive turtle program for the left-handed turtle that generates the fractal in part *a*.

**Programming Projects:**

1. *Parsers*
   a. Write a parser to convert recursive programs for the classical turtle into conformal iterated function systems that generate the same fractal.
   b. Write a parser to convert recursive turtle programs for the left-handed turtle into iterated function systems that generate the same fractal (see Exercise 10).

2. *Open Problem*s (See Exercise 8)
   a. Develop an algorithm that given two arbitrary iterated function systems determines whether the two systems generate the same fractal or show that no such algorithm exists.
   b. Develop an algorithm that given two arbitrary recursive turtle programs determines whether the two programs generate the same fractal or show that no such algorithm exists.

   Note that by the results in this chapter, the preceding two open problems are equivalent.