Lecture 2: Fractals from Recursive Turtle Programs

use not vain repetitions... Matthew 6: 7

1. Fractals

Pictured in Figure 1 are four fractal curves. What makes these shapes different from most of the curves we encountered in the previous lecture is their amazing amount of fine detail. In fact, if we were to magnify a small region of a fractal curve, what we would typically see is the entire fractal in the large. In Lecture 1, we showed how to generate complex shapes like the rosette by applying iteration to repeat over and over again a simple sequence of turtle commands. Fractals, however, by their very nature cannot be generated simply by repeating even an arbitrarily complicated sequence of turtle commands. This observation is a consequence of the Looping Lemmas for Turtle Graphics.

![Sierpinski Triangle](image1.png) ![Fractal Swiss Flag](image2.png) ![Koch Snowflake](image3.png) ![C-Curve](image4.png)

Figure 1: Four fractal curves.

2. The Looping Lemmas

Two of the simplest, most basic turtle programs are the iterative procedures in Table 1 for generating polygons and spirals. The looping lemmas assert that all iterative turtle programs, no matter how many turtle commands appear inside the loop, generate shapes with the same general symmetries as these basic programs. (There is one caveat here, that the iterating index is not used inside the loop; otherwise any turtle program can be simulated by iteration.)

<table>
<thead>
<tr>
<th>POLY (Length, Angle)</th>
<th>SPIRAL (Length, Angle, Scalefactor)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Repeat Forever</td>
<td>Repeat Forever</td>
</tr>
<tr>
<td>FORWARD Length</td>
<td>FORWARD Length</td>
</tr>
<tr>
<td>TURN Angle</td>
<td>TURN Angle</td>
</tr>
<tr>
<td></td>
<td>RESIZE Scalefactor</td>
</tr>
</tbody>
</table>

Table 1: Basic procedures for generating polygons and spirals via iteration.
Circle Looping Lemma
Any procedure that is a repetition of the same collection of FORWARD and TURN commands has the structure of POLY(Length, Angle), where

Angle = Total Turtle Turning within the Loop
Length = Distance from Turtle's Initial Position to Turtle's Final Position within the Loop

That is, the two programs have the same boundedness, closing, and symmetry. In particular, if Angle ≠ 2πk for some integer k, then all the vertices generated by the same FORWARD command inside the loop lie on a common circle.

Spiral Looping Lemma
Any procedure that is a repetition of the same collection of FORWARD, TURN, and RESIZE commands has the structure of SPIRAL(Length, Angle, Scalefactor), where

Angle = Total Turtle Turning within the Loop
Length = Distance from Turtle’s Initial Position to Turtle’s Final Position during the First Iteration of the Loop
Scalefactor = Total Turtle Scaling within the Loop

That is, the two programs have the same boundedness and symmetry. In particular, if Angle ≠ 2πk for some integer k, and Scalefactor ≠ ±1, then all the vertices generated by the same FORWARD command inside the loop lie on a common spiral.

To prove the Circle Looping Lemma, follow the turtle through two iterations of the loop and observe that the initial turtle state undergoes exactly the same change as the turtle traversing two iterations of the POLY procedure (see Figure 2 below). Since we are iterating through exactly the same loop over and over again, after every iteration of the loop the looping turtle must end up in exactly the same state as the turtle executing the POLY procedure. This observation is the essence of the Circle Looping Lemma. The Spiral Looping Lemma can be proved in an exactly analogous manner.

Figure 2: The turtle traversing through two iterations of a loop.
**Examples:** The WALK procedure and the accompanying turtle path (Figure 3, left) provide typical embodiments of the Circle Looping Lemma. Notice that vertices generated by the same FORWARD command inside the loop lie on a common circle. Similarly, the PENTARAL procedure and the accompanying turtle path (Figure 3, right) are embodiments of the Spiral Looping Lemma. Notice that in the PENTARAL curve vertices generated by the same FORWARD command inside the outer loop of the program lie on a common spiral.

![WALK](image1)

![PENTARAL](image2)

**Figure 3:** The turtle paths corresponding to the WALK procedure (left) and the PENTARAL procedure (right). Notice that corresponding vertices in the WALK curve lie on a common circle and that corresponding vertices in the PENTARAL curve lie on a common spiral.

When the POLY procedure generates either a regular polygon or star, the vertices clearly lie on a common circle. But for angles $A$ that are incommensurable with $2\pi$ -- that is, for angles $A$ for which there are no integers $m,n$ such that $mA = 2n\pi$ -- the curve does not close and it is not so obvious that the vertices generated by the POLY procedure must necessarily lie on a circle. Yet, as we can see from Figure 4, even when $2\pi/A$ is not rational, the vertices do nevertheless seem to lie on a common circle. We shall now give a simple proof of this result based on standard arguments from Euclidean geometry.
Figure 4: The output of the POLY procedure for different input angles $A$. From left to right: a regular ten sided polygon ($A = \pi / 5$), a seven pointed star ($A = 6\pi / 7$), and a figure that never closes ($A = \pi \sqrt{2} / 2$). Notice that in each case, the vertices lie on a common circle.

Proposition 1: The vertices generated by the procedure POLY(Length, Angle) lie on a common circle for any angle $A \neq 2\pi k$, for some integer $k$, and any length $L \neq 0$.

Proof: Consider the circle generated by the first three vertices $D, E, F$ visited by the turtle while executing the POLY procedure (see Figure 5). We need to show that the next vertex $G$ visited by the turtle lies on the same circle -- that is, we need to show that $x = CG = R$, where $R$ is the radius of the circle circumscribed about $\Delta DEF$ and $C$ is the center of this circle. But

$\Delta CDE \cong \Delta CEF$

since by construction the lengths of the three corresponding sides are equal (SSS). Now

$A + \beta + \alpha = A + \alpha + \alpha \Rightarrow \beta = \alpha$

Therefore

$\Delta CEF \cong \Delta CFG$

because these triangles agree in two sides and an included angle (SAS). Hence, since $\Delta CEF$ is isosceles, $\Delta CFG$ is also isosceles, so

$x = CG = R$.

Figure 5: The circle generated by the first three vertices $D, E, F$ visited by the turtle while executing the POLY procedure.

The vertices of the fractals in Figure 1 do not lie on common circles nor do they lie on common spirals. Thus, as a consequence of the Looping Lemmas, these fractals cannot be generated by iterative turtle procedures. To generate these fractal curves, we must resort instead to recursive turtle programs.
3. Fractal Curves and Recursive Turtle Programs

There are many different types of fractal curves, including gaskets, snowflakes, terrain, and even plants. Here we are going to concentrate on two representative types of fractals: fractal gaskets and bump fractals. Both of these types of fractal curves can be generated easily by simple recursive turtle programs.

3.1 Fractal Gaskets. The Sierpinski triangle in Figure 1 is an example of a fractal gasket. Because of the Looping Lemmas, we cannot use iteration to generate fractals, so we shall develop a recursive turtle program to generate this gasket. The key observation for building such a recursive program is that the big Sierpinski gasket is made up of three identical scaled down copies of the entire gasket. Thus, a Sierpinski gasket is just three smaller Sierpinski gaskets joined together. This definition would be circular, if we did not have a base case at which to stop the recursion. We shall say, therefore, that a very small Sierpinski gasket is just a very small triangle.

Now the structure of the recursive portion of the turtle program for the Sierpinski gasket must be something like the following: To make a large Sierpinski gasket:

- Make a smaller Sierpinski gasket at one of the corners.
- Then move to another corner and make another small Sierpinski gasket.
- Then move to another corner and make another small Sierpinski gasket.
- Then return the turtle to her initial position and heading.

The moves can be made with MOVE and TURN commands; the rescaling is achieved with RESIZE commands. To make the three smaller gaskets requires three recursive calls. Form follows function! By the way, why do you think you have to return the turtle to her initial state?

Below is the complete program for generating the Sierpinski gasket. It’s short and simple. Different levels of the gasket generated by the SIERPINSKI program are illustrated in Figure 6. Notice that all the drawing done by this program is performed exclusively by the FORWARD commands inside the POLY procedure in the base case:

\[
\text{SIERPINSKI} \text{(Level)} \\
\begin{align*}
\text{IF Level} &= \text{0, POLY (1, } \frac{2\pi}{3} \text{)} \\
\text{OTHERWISE} \\
\text{REPEAT 3 TIMES} \\
& \text{RESIZE } \frac{1}{2} \\
& \text{SIERPINSKI (Level - 1)} \\
& \text{RESIZE 2} \\
& \text{MOVE 1} \\
& \text{TURN } \frac{2\pi}{3}
\end{align*}
\]
Examining the syntax of the SIERPINSKI program, we see that the recursive body has a structure very similar to the program for generating a triangle. Applying this observation, we can easily generalize this program for the Sierpinski triangle to the following program that generates gaskets for arbitrary polygons. Different levels of the pentagonal gasket are illustrated in Figure 7. Some additional fractal gaskets are illustrated in Figure 8.

\[
\text{POLYGASKET} \ (N, \text{Level})
\]

\[
\text{IF Level} = 0, \ \text{POLY} \ (1, \ 2\pi / N) \\
\text{OTHERWISE}
\]

\[
\text{REPEAT} \ N \ \text{TIMES}
\]

\[
\text{RESIZE} \ 1/2 \\
\text{POLYGASKET} \ (N, \text{Level - 1}) \\
\text{RESIZE} \ 2 \\
\text{MOVE} \ 1 \\
\text{TURN} \ 2\pi / N
\]

Figure 6: Levels 1, 3, 6 of the Sierpinski gasket.

Figure 7: Levels 1, 3, 5 of a pentagonal gasket.
Figure 8: More fractal gaskets. We leave it as an exercise to the reader to develop recursive turtle programs to generate each of these fractal gaskets (see Exercise 4).

3.2 Bump Fractals. A bump curve is just a pulse along a straight line. Figure 9 illustrates three examples of bump curves. A bump fractal is a fractal curve generated by recursively replacing each straight line of a bump curve by a scaled version of the bump. The C-curve in Figure 1 is the bump fractal corresponding the right most bump curve in Figure 9.

Figure 9: Some examples of bump curves.

It is actually quite easy to write a recursive turtle program to generate any arbitrary bump fractal. The base case is a straight line, so the code for the base case consists of the single turtle command FORWARD 1. To generate the recursive body of the program, proceed in two steps:

1. Write a turtle program to generate the corresponding bump curve. In this turtle program, the FORWARD commands should all take the parameter 1; the actual distance traveled should be adjusted by using the RESIZE command.
2. Replace all the FORWARD commands in step 1 by recursive calls.

In the recursive turtle program for a bump fractal, the base case draws a straight line. Therefore, since the FORWARD commands in the turtle program for the bump curve are replaced by recursive calls, level 1 of the recursion reproduces the original bump curve, and level $n$ replaces each line on level $n-1$ by a scaled down version of the original bump. Thus, given an arbitrary bump curve, a bump fractal is generated recursively by placing a bump fractal along each straight line in the original bump curve.

We illustrate this approach to generating recursive turtle programs for bump fractals in Table 2 for the leftmost triangular bump curve in Figure 9. The corresponding bump fractal is called the Koch curve. Different levels of the Koch curve are depicted in Figure 10. The Koch Snowflake in Figure 1 can be generated by spinning the Koch curve three times through the angle $2\pi / 3$.  

7
TriangularBumpFractal (Level)
If Level = 0 FORWARD 1
Otherwise
RESIZE 1/3
RESIZE 1/3
FORWARD 1
TriangularBumpFractal (Level-1)
TURN $\pi / 3$
TURN $\pi / 3$
FORWARD 1
TriangularBumpFractal (Level-1)
TURN $-2\pi / 3$
TURN $-2\pi / 3$
FORWARD 1
TriangularBumpFractal (Level-1)
TURN $\pi / 3$
TURN $\pi / 3$
FORWARD 1
TriangularBumpFractal (Level-1)
RESIZE 3
RESIZE 3

Table 2: The turtle program for a triangular bump, and the recursive turtle program for the corresponding bump fractal (the Koch curve).

<table>
<thead>
<tr>
<th>TriangularBump</th>
<th>TriangularBumpFractal (Level-1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RESIZE 1/3</td>
<td>FORWARD 1</td>
</tr>
<tr>
<td>TURN $\pi / 3$</td>
<td>TURN $\pi / 3$</td>
</tr>
<tr>
<td>FORWARD 1</td>
<td>TURN $-2\pi / 3$</td>
</tr>
<tr>
<td>TURN $-2\pi / 3$</td>
<td>FORWARD 1</td>
</tr>
<tr>
<td>TriangularBumpFractal (Level-1)</td>
<td>RESIZE 3</td>
</tr>
<tr>
<td>TURN $\pi / 3$</td>
<td></td>
</tr>
<tr>
<td>FORWARD 1</td>
<td></td>
</tr>
<tr>
<td>TriangularBumpFractal (Level-1)</td>
<td></td>
</tr>
</tbody>
</table>

Figure 10: Levels 1, 2, 5 of the Koch curve corresponding to the recursive turtle program in Table 2.

Notice that the recursive turtle program for the Koch curve restores the original step size of the turtle, but unlike the turtle program for the Sierpinski gasket, the turtle program for the Koch curve does not restore the turtle to her original position. Be sure that you understand why it is not necessary to restore the turtle to her initial position in order to generate the Koch curve or, in fact, to generate any other bump fractal.

4. Summary: Fractals -- Recursion Made Visible

We never actually defined the term fractal. Nevertheless, from the turtle’s point of view, fractals are just recursion made visible.

In this lecture we have studied two kinds of fractals: fractal gaskets and bump fractals. But pretty much any recursive turtle program generates a fractal curve. Conversely, many different fractal curves can be generated by recursive turtle programs.

Consider, for example, the fractal curves in Figure 11. Each of these fractals is made up of
several identical smaller scaled copies of the entire fractal. Curves that consist of several identical scaled copies of themselves are said to be *self-similar*. In the recursive turtle programs that generate self-similar fractals, there must be as many recursive calls as there are equivalent scaled copies. Once again, *form follows function.*

**Figure 11:** A few more self-similar fractal curves: a fractal hangman, a fractal bush, and a fractal tree. We leave it as a challenge to the reader to develop recursive turtle programs to generate these fractals, using the techniques described in this lecture (see Exercise 11).

When writing a recursive turtle program to generate a specific fractal, you should always keep in mind the answers to following questions:

1. Where does the turtle start?
2. Where does the turtle finish?
3. How many recursive calls must the turtle make?
4. What are the scale factors?
5. What are the turtle transitions (MOVE and TURN commands) between recursive calls?

*The most common mistake students make in writing recursive turtle programs to generate specific fractal curves is failing to return the turtle to her original state.* Not all fractals require that the turtle must return to her original state, but if the curve resulting from your recursive turtle program does not produce the desired result, the first thing you should check is whether you have restored your turtle to the proper state at the end of your recursive procedure.

**Exercises:**

1. Prove the Spiral Looping Lemma.

2. Notice that in Figure 4 not only do all the vertices generated by the POLY procedure lie on a circle, but the all edges too seem to be tangent to a common circle, even for angles $\theta$ incommensurate with $2\pi$. 
a. Prove that for any angle $A \neq 2\pi k$, for some integer $k$, and any length $L \neq 0$, the line segments of the curve generated by the turtle program $POLY(L,A)$ are tangent to a common circle. (Hint: In Figure 12, prove that $\beta = \alpha$. Conclude that $\triangle CHG \cong \triangle CFG$, so $x = R$ and $\angle CHG = \pi/2$.)

b. How does the radius of the inner circle vary as the angle $A$ is increased?

\[ \text{Figure 12: The circle tangent to the first two edges generated by the turtle while executing the POLY procedure.} \]

3. Suppose that in either of the Looping Lemmas the total turning angle inside the loop is $2\pi k$ for some integer $k$. On what curve do all the vertices generated by the same FORWARD command inside the loop necessarily lie? Why? Give some examples to illustrate your answer.

4. Write turtle programs to generate each of the fractal gaskets depicted in Figure 8.

5. Generate new fractal gaskets by changing the scaling parameter inside the body of the recursion for gaskets you have already generated.

6. Write turtle programs to generate each of the bump fractals corresponding to the bump curves depicted in Figure 9.

7. Write turtle programs to generate each of the bump fractals corresponding to the bump curves depicted in Figure 13.

\[ \text{Figure 13. A triangular pulse (left) and a rectangular pulse (right).} \]
8. Write turtle programs to generate each of the bump fractals corresponding to the bump curves depicted in Figure 14.

![Figure 14](image1)

**Figure 14.** Polygonal bumps: a pentagonal bump (left), a hexagonal bump (middle), and a polygonal bump with 20 sides (right).

9. Write turtle programs to generate each of the bump fractals corresponding to the bump curves depicted in Figure 15.

![Figure 15](image2)

**Figure 15.** More polygonal bumps. To generate these bumps, start at the left end point of the line at the base, proceed counterclockwise around the polygon, and finish at the right end point of the line.

10. Generate new fractals by applying the SPIN and SCALE procedure to fractals that you have already generated.

11. Write turtle programs to generate each of the fractal curves depicted in Figure 11.

12. Write turtle programs to generate each of the fractal curves depicted in Figure 16.

![Figure 16](image3)

**Figure 16:** More fractal curves: a fractal star, a fractal flower, and a fractal leaf.
13. In classical LOGO there is no RESIZE command.
   a. Show that the two commands RESIZE $S$, FORWARD $D$ are equivalent to the
      single command FORWARD $S\ D$.
   b. Conclude from part $a$ that the result of any turtle program that includes RESIZE
      commands can be simulated by a turtle program without any RESIZE commands.
   c. Explain how to generate bump fractals without using the RESIZE command.

Programming Projects:

1. **The Classical Turtle**
   Implement LOGO in your favorite programming language using your favorite API.
   a. Write LOGO programs for the curves discussed in the text and in the exercises for
      Lectures 1 and 2.
   b. Write a LOGO program to design a novel flag for a new country.
   c. Write recursive LOGO programs to create your own novel fractals.
   d. In the text we generated fractals using simple recursive turtle programs. But fractals can
      also be generated using mutually recursive turtle programs -- two or more recursive turtle
      programs that call one another.
      i. Build some novel fractals using mutually recursive turtle programs.
      ii. The Hilbert curve is a space filling fractal curve (see Figure 17). Write two mutually
          recursive turtle programs to generate the Hilbert curve.

   ![Figure 17: The first four levels of the Hilbert curve.](image)

2. **The Hodograph Turtle**
   The classical turtle draws a line from her initial position to her new position after each turtle
   command. The hodograph turtle draws a line from the initial position of the tip of her direction
   vector to the new position of the tip of her direction vector after each turtle command.
   a. Implement the hodograph turtle in your favorite programming language using your
      favorite API.
      i. Implement the turtle commands PENUP and PENDOWN, so that, if necessary, the
         hodograph turtle can TURN and RESIZE without drawing a line.
      ii. Implement two new turtle command: ANCHORUP and ANCHORDOWN.
When the anchor is down, the hodograph turtle ignores the FORWARD and MOVE commands.

b. Write turtle programs to generate simple curves like polygons, stars, and spirals using the hodograph turtle.

c. Draw the curves in Figure 18 using
   i. the classical turtle
   ii. the hodograph turtle
      Which is easier?

d. Compare and contrast the curves generated by the classical turtle and the hodograph turtle for the same turtle programs when the anchor is down. For example:
   i. What curve does the hodograph turtle draw, when the classical turtle draws a polygon, a star, or a rosette?
   ii. What curve does the hodograph turtle draw, when the classical turtle draws the Sierpinski gasket or the Koch curve?

e. Compare the fractals generated by the classical turtle and the hodograph turtle for the same recursive turtle program. Form a conjecture concerning when the two turtles generate the same fractal curve.

![Figure 18](image)

**Figure 18.** A pentagon inscribed in a circle (left), a pentagon circumscribed about a circle (center), and a collection of concentric circles (right). Are these curves easier to generate with the classical turtle or with the hodograph turtle?

3. *The Classical Turtle on a Bounded Domain*

   The classical turtle lives on an infinite plane. Suppose, however, that the turtle is restricted to a finite domain bounded by walls. When the turtle hits a wall, she bounces off the wall so that her angle of incidence is equal to her angle of reflection, and then she continues on her way. Thus the turtle behaves like a billiard ball on a table with no friction or like a light beam surrounded by mirrors. To implement the turtle on a bounded domain, the FORWARD command must be replaced by:
NEWFORWARD \( D \)

\[ D_1 = \text{Distance from Turtle to Wall in Direction of Turtle Heading} \]

IF \( D < D_1 \), FORWARD \( D \)

OTHERWISE

FORWARD \( D_1 \)

TURN \( A \) \( \{ A = \text{angle of reflection} \} \)

NEWFORWARD \( D - D_1 \)

To compute the angle of reflection \( A \), let \( \angle(w,N) \) denote the angle between the turtle direction vector \( w \) and the normal \( N \) to the wall at the point where the turtle hits the wall. By a straightforward geometric argument, you should be able to show that: \( A = \pi - 2\angle(w,N) \).

a. Implement LOGO on a bounded domain, where the walls form:
   i. a rectangle
   ii. a circle
   iii. an ellipse

b. Consider the turtle program consisting of the single command

NEWFORWARD \( D \)

Investigate the curves generated by this program for different shape walls, different values of \( D \), and different initial positions and headings for the turtle.

c. Investigate how curves drawn by the walled in turtle differ from curves drawn by the turtle on an infinite plane using the same turtle programs, where FORWARD is replaced by NEWFORWARD. In particular, investigate the curves generated by POLY, SPIRAL, and different recursive turtle procedures for generating fractals.

4. The Left-Handed Turtle

The state of the classical turtle is specified by her position and one vector, her forward facing direction vector. The state of the left-handed turtle is specified by her position and two vectors: one vector specifies her forward facing direction, the other vector specifies the direction of her left hand. Thus the left-handed turtle carries around her own local coordinate system. To manipulate the left hand vector, the turtle commands TURN and RESIZE are redefined to be functions of two parameters: the first parameter refers to the forward vector, the second parameter refers to the left hand vector. The RESIZE command is easy to define; just rescale each vector by the specified amount. The TURN command, however, is more subtle. The left-handed turtle measures all angles relative to her own coordinate system, which she assumes is a rectangular coordinate system. Thus when she executes the TURN command, she rotates the forward facing vector and the left-hand vector relative to her own coordinate system rather than relative to a global coordinate system. Let \( (P,w,w^*) \) denote the current state of the turtle, where \( w \) is the forward facing direction vector and \( w^* \) is the vector that specifies the direction of the turtle’s left hand. Then the turtle commands have the following effect on the turtle’s state:
• **FORWARD** $D$, **MOVE** $D$
  \[ P_{\text{new}} = P + Dw \]

• **TURN** $A_1, A_2$
  \[ w_{\text{new}} = w \cos(A_1) + w^* \sin(A_1) \]
  \[ w^*_{\text{new}} = w^* \cos(A_2) - w \sin(A_2) \]

• **RESIZE** $S_1, S_2$
  \[ w_{\text{new}} = S_1 u \]
  \[ w^*_{\text{new}} = S_2 v \]

In order that the our old LOGO programs will still work for the left-handed turtle, we define

\[ \text{TURN } A = \text{TURN } A,A \]
\[ \text{RESIZE } S = \text{RESIZE } S,S \]

a. Implement LOGO for the left-handed turtle in your favorite programming language using your favorite API.

b. Draw an ellipse using the left-handed turtle and then spin the ellipse to generate the pattern in Figure 19.

c. Study the shapes generated by the following turtle programs:

- **NEWPOLY**(Length, Angle1, Angle2)
- **NEWSPIRAL**(Length, Angle, Scale1, Scale2)

\[ \text{Repeat Forever} \]
\[ \text{FORWARD} \text{ Length} \]
\[ \text{TURN} \text{ Angle1, Angle2} \]
\[ \text{RESIZE} \text{ Scale1, Scale2} \]

\[ \text{Repeat Forever} \]
\[ \text{FORWARD} \text{ Length} \]
\[ \text{TURN} \text{ Angle} \]
\[ \text{RESIZE} \text{ Scale1, Scale2} \]

d. What are the analogues of the Looping Lemmas for the left-handed turtle?

e. Write recursive programs for the left-handed turtle to create your own novel fractals.

---

**Figure 19:** An ellipse (left) and the spin operator applied to an ellipse (right).
5. The Turtle on 2-Dimensional Manifolds

The classical turtle lives on a flat plane. Here we shall investigate turtles that live on curved 2-dimensional manifolds, including cylinders, mobius strips, tori, klein bottles, and projective planes. These surfaces can be modeled by rectangles, where one or more pairs of opposite sides are glued together, possibly with a twist (see Figure 20). (Notice that by gluing together the appropriate sides, the cylinder, the mobius strip, and the torus can be embedded in 3-dimensions. However, the klein bottle and the projective plane cannot be embedded in 3-dimensions due to the twists; these manifolds live naturally in 4-dimensions, even though they are 2-dimensional surfaces.) If a pair of opposite sides are not glued together, then these sides are treated like a wall just as with the classical turtle on a bounded domain (see Project 3). If, however, a turtle encounters a wall where opposite sides are identified, then she does not bounce off the wall, but rather emerges on the opposite side of the rectangle heading in the same direction relative to the new wall in which she hit the opposing wall.

a. Implement LOGO on one or more of the following 2-manifolds:
   i. cylinder
   ii. mobius strip
   iii. torus
   iv. klein bottle
   v. projective plane

   Note: For each of these manifolds, you will need to define a command MFORWARD to replace of the command FORWARD in classical LOGO. (See Project 3.)

b. Consider the turtle program consisting of the single command

   \begin{align*}
   \text{MFORWARD } D
   \end{align*}

   Investigate the curves generated by this program for different manifolds, different values of $D$, and different initial positions and headings for the turtle.

c. Investigate how curves drawn by turtles on these 2-dimensional manifolds differ from curves drawn by the turtle on an infinite plane using the same turtle programs, where FORWARD is replaced by MFORWARD. In particular, investigate the curves generated by POLY, SPIRAL, and different recursive turtle procedures for generating fractals.

\begin{center}
\begin{tabular}{ccccc}
\textbf{Cylinder} & \textbf{Mobius Strip} & \textbf{Torus} & \textbf{Klein Bottle} & \textbf{Projective Plane} \\
\end{tabular}
\end{center}

\textbf{Figure 20:} Rectangular representations of five different 2-dimensional manifolds.