HODOGRAPH TURTLES

Tao Ju
Computer Science Department
Rice University
Houston, TX, USA
e-mail: jutao@rice.edu

Ron Goldman
Computer Science Department
Rice University
Houston, TX, USA
e-mail: rng@rice.edu

ABSTRACT
In classical turtle graphics a line is drawn to connect the turtle’s position vector before and after executing each FORWARD command. A hodograph turtle shadows the classical turtle and draws a line connecting the classical turtle’s direction vector before and after executing each TURN command. Here we study examples of the hodograph turtle in action along with several extensions. We show that some shapes are easier to generate using a hodograph turtle instead of the classical turtle. More importantly, we can extract information about the program of the classical turtle from the geometry generated by a hodograph turtle. We shall see that especially for complicated fractals, there is often more than one natural way to generate the same shape using a turtle. The information about the turtle commands is lost once the shape is drawn, and the simplicity of the turtle program is clouded by the complexity of the turtle geometry.

Here we are going to introduce a new class of turtles called hodograph turtles. The classical turtle in standard LOGO can be modelled by a position vector and a direction vector, and the turtle geometry is formed by plotting the trace of the position vector as the turtle moves in the plane. (Note that this model differs slightly from standard LOGO where the turtle stores an angle instead of a direction vector, but this model has exactly the same expressive power as the standard model; see [3], page 136.) The hodograph turtle, in the simplest setting, draws the trace of the classical turtle’s unit direction vector, connecting the head of the direction vector before and after executing each TURN command. Turtle geometry is local and coordinate free, but the path of the simple hodograph turtle is coordinate dependent, inscribed in a unit circle centered at the origin. This simple hodograph creature can be extended to plot shapes on concentric circles around the origin, or even to wander off the origin and plot shapes inscribed in circles centered at arbitrary locations on the plane.

The simple hodograph turtle and its extensions are worth studying for several reasons. First, since the direction vector is not affected by the FORWARD commands, programming this hodograph turtle is much simpler than programming the classical turtle. Thus it is often easier to generate shapes inscribed in circles using a hodograph turtle rather than the classical turtle. Second, as a companion of the classical turtle, the hodograph turtle reveals how the classical turtle accomplishes the drawing, which is not always evident from the actual geometry generated by the classical turtle. We shall see that the paths of the hodograph turtles are often much simpler than the paths of the classical turtles, especially for complicated shapes such as fractals. Third, the hodograph turtles themselves can generate interesting shapes. In particular, hodograph turtles open up a new way of generating novel fractal shapes associated with standard turtle fractals.

We will start with a brief description of the classi-
Figure 1. Comparison of paths (in thickened lines) drawn by the classical turtle (top) and the hodograph turtle (bottom) using the turtle programs in Table 1. Note that the paths drawn by the classical turtle are rendered at a different scale from the paths drawn by the hodograph turtle. Thus, for example, the circles drawn by the two turtles in (c) are not actually the same size.

<table>
<thead>
<tr>
<th>POLYGON N</th>
<th>STAR N</th>
<th>CIRCLE S</th>
<th>ROTATE_STAR N, M, A</th>
</tr>
</thead>
<tbody>
<tr>
<td>REPEAT N TIMES</td>
<td>REPEAT N TIMES</td>
<td>REPEAT 360 TIMES</td>
<td>REPEAT M TIMES</td>
</tr>
<tr>
<td>FORWARD 1</td>
<td>FORWARD 1</td>
<td>FORWARD S</td>
<td>STAR N</td>
</tr>
<tr>
<td>TURN 2π/N</td>
<td>TURN 4π/N</td>
<td>TURN π/180</td>
<td>TURN A</td>
</tr>
</tbody>
</table>

Table 1. Turtle programs that draw a regular polygon, a star, a circle and a sequence of rotated stars.

cal turtle commands and turtle programs. The hodograph turtles are introduced next, in the order of their increasing ability of maneuver. The simplest hodograph turtle, introduced in Section 3, plots paths whose vertices lie only on the unit circle by ignoring the FORWARD commands and executing only the TURN commands. By introducing the RESIZE command in Section 4, we create a more advanced hodograph turtle who can draw shapes inscribed in concentric circles. The augmented hodograph turtle, presented in Section 5, is able to draw arbitrary shapes as general as those drawn by the classical turtle by accepting a new pair of commands called ANCHOR-UP and ANCHOR-DOWN and executing the FORWARD command whenever the anchor is up. We close in Section 6 with a summary of our work along with a few open questions for future research.

2 Classical Turtle and Turtle Programming

The classical turtle’s state is described by the turtle’s position in the plane \( P = (p_1, p_2) \) and a vector \( w = (w_1, w_2) \) capturing both the direction in which it is facing and its step size. For drawing, the turtle is also holding a pen at its position \( P \). In the classical setting, there are four turtle commands, FORWARD, TURN, PEN-UP and PEN-DOWN. FORWARD and TURN each take a single scalar parameter and modify the turtle’s state as shown in Table 2. The PEN-UP and PEN-DOWN commands specify whether the turtle’s pen is in the up or down position. When the pen is down, the turtle draws a line connecting the position vector \( P \) before and after executing each FORWARD command. Otherwise, the turtle moves without leaving a trail. Throughout the paper, we assume that the turtle starts with \( P = (0, 0) \), \( w = (1, 0) \), and the pen in the down position.

<table>
<thead>
<tr>
<th>Command</th>
<th>( P )</th>
<th>( w )</th>
</tr>
</thead>
<tbody>
<tr>
<td>FORWARD</td>
<td>( P^{\text{new}} = P + dw )</td>
<td>( w^{\text{new}} = w )</td>
</tr>
<tr>
<td>TURN a</td>
<td>( P^{\text{new}} = P )</td>
<td>( w_1^{\text{new}} = w_1 \cos(a) - w_2 \sin(a) )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( w_2^{\text{new}} = w_1 \sin(a) + w_2 \cos(a) )</td>
</tr>
</tbody>
</table>

Table 2. Turtle commands and how they modify the position vector \( P \) and the direction vector \( w \).

A turtle program is a finite sequence of turtle commands. Note that the turtle commands transform the turtle relative to the turtle’s current state. In other words, the classical turtle always draws in its own local frame.
3 Hodograph Turtle

A hodograph turtle can always be associated with a classical turtle. In the basic setting (which will be extended in Sections 4 and 5), the state of the hodograph turtle is described by the direction vector \( w \) of the associated classical turtle. As seen in Table 2, the direction vector \( w \) is unaffected by the FORWARD command. Hence the hodograph turtle responds only to the TURN commands. When a turtle program is executed, the classical turtle draws a line connecting the head of the position vector \( P \) before and after executing each FORWARD command (when the pen is down), while the hodograph turtle draws a line connecting the head of the direction vector \( w \) before and after executing each TURN command (when the pen is down).

The term hodograph was first introduced by Sir William Rowan Hamilton to describe the plot of the velocity of a particle as a function of time. Just as the velocity is tangent to the trajectory of particle, so too is the direction vector \( w \) tangent to the path of the classical turtle; hence the name hodograph turtle.

In Figure 1, we compare the shapes generated by the classical turtle and the corresponding hodograph turtle using the turtle programs in Table 1. Unlike the classical turtle with its local coordinate frame, the hodograph turtle lives in a fixed global coordinate system, since the tail of the direction vector \( w \) is always at the origin. Thus while the classical turtle turns around the turtle’s current location, the hodograph turtle turns around a fixed origin (Figure 1 d). Accordingly, whereas the angle in each TURN command represents the external angle formed by consecutive line segments in the path of the classical turtle, this TURN angle represents the central angle subtending a line segment in the path of the hodograph turtle.

The vertices on the path of the hodograph turtle lie on the unit circle centered at the origin, since the magnitude of the turtle’s direction vector \( w \) is always 1 (see Figure 1). This observation leads to simpler turtle programs for generating shapes inscribed in a circle by invoking the hodograph turtle instead of the classical turtle.

3.1 Drawing Shapes Inscribed in a Circle

Regular polygons and stars are among the simplest shapes generated by the classical turtle. Since their vertices lie on a circle, regular polygons and stars can also be generated easily by a hodograph turtle. In fact, writing a turtle program so that the hodograph turtle draws the desired shape is often much easier than writing a turtle program in which the classical turtle draws the same shape.

Consider the programs POLYGON and STAR in Table 1. The parameters in the FORWARD commands affect the shape drawn by the classical turtle. If each edge of the polygon is drawn by a FORWARD command with a different parameter, the resulting path of the classical turtle will not be a regular polygon. The hodograph turtle, on the other hand, is insensitive to these parameters. In fact, the FORWARD commands can be removed from the program and the hodograph turtle will still plot the same shape.

The advantage of the hodograph turtle becomes more apparent when pre-computing the FORWARD parameter is difficult. For example, to draw a rosette (i.e., a polygon with all its diagonals), we can either run the program ROSETTE_1 on the left of Table 3 and plot the path of the classical turtle (top-left of Figure 2), or we can run the program ROSETTE_2 on the right of Table 3 and plot the path of the hodograph turtle (bottom-right of Figure 2). Note that ROSETTE_1 uses FORWARD commands whose parameter values involve trigonometric functions, whereas ROSETTE_2 uses only TURN commands with simple parameters (an auxiliary FORWARD command has been inserted between the two TURN commands in the inner loop so that the classical turtle also plots an interesting path, shown at the top-right of Figure 2).

![Figure 2. Paths of the classic turtle (top) and the hodograph turtle (bottom) executing the ROSETTE_1 program (left) and the ROSETTE_2 program (right).](image-url)

<table>
<thead>
<tr>
<th>ROSETTE_1</th>
<th>ROSETTE_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{REPEAT} N \text{ TIMES}</td>
<td>\text{REPEAT} N \text{ TIMES}</td>
</tr>
<tr>
<td>\text{FOR} i = 2 \text{ TO } N - 2</td>
<td>\text{FOR} i = 2 \text{ TO } N - 2</td>
</tr>
<tr>
<td>\text{TURN } \pi / N</td>
<td>\text{TURN } 2i\pi / N</td>
</tr>
<tr>
<td>\text{FORWARD } 2\sin(i\pi / N)</td>
<td>\text{FORWARD } 1</td>
</tr>
<tr>
<td>\text{FORWARD } -(N - 3)\pi / N</td>
<td>\text{TURN } -2i\pi / N</td>
</tr>
<tr>
<td>\text{FORWARD } 2\sin(\pi / N)</td>
<td>\text{TURN } 2\pi / N</td>
</tr>
<tr>
<td>\text{TURN } 2\pi / N</td>
<td></td>
</tr>
</tbody>
</table>

Table 3. Turtle programs that draw a rosette using the classical turtle (left) and using the hodograph turtle (right).
Another good example is simply to draw a circle together with an inscribed star (or polygon). To accomplish this task, the classical turtle programmer has to compute the appropriate parameter to the procedure CIRCLE as well as a suitable turning angle so that the vertices of the star lie exactly on the circle (see program INSCRIBED\_1 on the left of Table 4). On the other hand, the hodograph turtle draws the same shape simply by executing STAR and CIRCLE (with any parameter) in sequence (see program INSCRIBED\_2 on the right of Table 4). The resulting shapes produced by the classical turtle as well as the hodograph turtle are plotted in Figure 3. Observe that the classical turtle draws a star and a circle with incorrect aspect ratios and locations using the simpler program INSCRIBED\_2.

The turtle programs POLYGON, STAR and CIRCLE in Table 1 are special in that both the classical turtle and the hodograph turtle produce the same geometry (ignoring difference due to rotation and scaling). However, this equivalence does not always hold, even for other shapes inscribed in a circle, as illustrated by the ROSETTE and INSCRIBED programs presented above.

---

### 4 RESIZE Command

The direction vector of the classical turtle always has unit length, so the hodograph turtle can only plot shapes that are inscribed in a unit circle. By introducing a new turtle command, RESIZE, the hodograph turtle acquires the ability to draw shapes on circles of different radii. This concentric nature of the path of the hodograph turtle will help us to better understand the recursive structure of the turtle programs that generate complex fractal geometry using the classical turtle.

#### 4.1 RESIZE

The RESIZE command takes a single parameter $s$ and scales the turtle’s direction vector $w$ so that $w_{\text{new}} = sw$. RESIZE is similar to TURN in that it modifies the direction vector without changing the turtle’s position. Although the classical turtle does not plot anything when a RESIZE command is executed, the hodograph turtle draws a line connecting the head of the direction vector $w$ before and after executing the RESIZE command (when the pen is down). RESIZE commands enable the hodograph turtle to plot shapes on concentric circles whose radii correspond to the different magnitudes of the direction vector $w$. Two examples of turtle programs involving RESIZE commands are shown in Table 5, and the resulting shapes generated by the classical turtle and the hodograph turtle are shown in Figure 4. Notice that the classical turtle scales from a point on its path, whereas the hodograph turtle scales from the origin.

#### 4.2 Fractals

The most interesting and exciting geometry that can be generated by the turtle are fractals. Fractals often consist of components that are transformed versions of themselves [6]. For example, the Sierpenski Triangle in Figure 5 consists of three scaled-down copies of itself. To generate fractals using turtles, we must resort to recursion.

#### 4.2.1 Classical turtle and fractals

Fractals are generated by the classical turtle using recursive turtle programs (RTPs). The simplest RTP is composed of:
Figure 4. Paths of the classic turtle (top) and the hodograph turtle (bottom) executing the SCALE_POLYGON program (left) and the SPIRAL program (right) in Table 5.

1. Base case: a finite sequence of turtle commands that draw some shape at the bottom level of recursion,

2. Recursion body: a finite sequence of turtle commands as well as recursive calls to the program itself (denoted by RECUR in the pseudo code).

The fractal is the limit shape generated by the RTP as the depth of recursion goes to infinity. In practice, however, we need to recur only to a finite level for visualization. Two sample RTPs are shown in Table 6, both generating the Sierpenski Triangle in Figure 5 using the classical turtle with 5 levels of recursion. Note that the RESIZE commands play a critical role in producing scaled copies of the triangles that form the fractal.

Although the classical turtle traverses different local paths when executing the two programs in Table 6, the final plots are identical. By looking at the fractal, we know nothing about how the shape is drawn. More importantly, although the structure of the turtle programs is simple, the path of the classical turtle is complex and difficult to understand.

### 4.2.2 Hodograph turtle and fractals

Typically, when the classical turtle draws a fractal shape, so does the hodograph turtle. Moreover, the path of the hodograph turtle provides us with some understanding of the underlying turtle program as well as the actual path taken by the classical turtle.

![Figure 5. The Sierpenski Triangle](image)

![Figure 6. Paths of the hodograph turtle](image)

Table 6. Two recursive turtle programs that generate a Sierpenski Triangle.

<table>
<thead>
<tr>
<th>SIERPENSKI1</th>
<th>SIERPENSKI2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Base case:</strong></td>
<td><strong>Base case:</strong></td>
</tr>
<tr>
<td>POLYGON 3</td>
<td>POLYGON 3</td>
</tr>
<tr>
<td><strong>Recursion body:</strong></td>
<td><strong>Recursion body:</strong></td>
</tr>
<tr>
<td>REPEAT 3 TIMES</td>
<td>RESIZE 1/2</td>
</tr>
<tr>
<td>RESIZE 1/2</td>
<td>RESIZE 2</td>
</tr>
<tr>
<td>RECUR</td>
<td>FORWARD 1/2</td>
</tr>
<tr>
<td>RESIZE 2</td>
<td>RESIZE 1/2</td>
</tr>
<tr>
<td>FORWARD 1</td>
<td>RECUR</td>
</tr>
<tr>
<td>TURN 2π/3</td>
<td>RESIZE 2</td>
</tr>
<tr>
<td>TURN −2π/3</td>
<td>FORWARD 1/2</td>
</tr>
<tr>
<td>TURN 2π/3</td>
<td>TURN −2π/3</td>
</tr>
<tr>
<td>FORWARD 1/2</td>
<td>FORWARD 1/2</td>
</tr>
<tr>
<td>TURN 2π/3</td>
<td></td>
</tr>
</tbody>
</table>

In Figure 6, the paths of the hodograph turtle are shown for the two RTPs in Table 6. Unlike the classical turtle, the hodograph turtle draws different paths for the two RTPs. The TURN and RESIZE commands that are not reflected in the geometry of the Sierpenski Triangle are evident in the path of the hodograph turtle. In particular, the RESIZE commands before and after each recursive call are drawn by the hodograph turtle as lines that connect segments of paths lying on circles of different radii. In Figure 6 left, there are three such lines between every two neighboring concentric circles, whereas there is only one line visible in Figure 6 right. These lines reveal that before each recursive call in the first RTP, the classical turtle holds one of the three possible direction vectors \( \{1, 0\}, \{-1/2, \sqrt{3}/2\}, \{-1/2, -\sqrt{3}/2\} \), whereas for each recursive call in the second RTP the turtle starts with the same direction vec-
tor (\{1, 0\}). This information cannot be retrieved from the geometry drawn by the classical turtle.

The paths plotted by the hodograph turtle also look much simpler than the original fractal. In fact, at a fixed recursion level, the direction vector of the classical turtle has the same magnitude regardless of the position on the turtle in the Sierpinski Triangle. Hence the paths of the hodograph turtle lie on concentric circles, each of which corresponds to a level of recursion. The simplicity of the hodograph reflects the simple recursive structure of the underlying RTP, a simplicity hidden in the complex geometry produced by the classical turtle.

Plotting the path of the hodograph turtle also provides a new way of generating novel fractal shapes from existing fractal programs. Figure 7 shows two engaging examples of new fractals generated by the classical turtle, corresponding to a fractal C curve and a Koch Snowflake generated by the classical turtle. The corresponding RTPs for generating these fractals are presented in Table 7.

![Figure 6](image6.png)

**Figure 6.** The paths drawn by the hodograph turtle using the two RTPs in Table 6.

![Figure 7](image7.png)

**Figure 7.** A fractal C curve (top left) and the Koch Snowflake (top right) drawn by the classical turtle, and the paths of their associated hodograph turtle (bottom left and bottom right).

### 4.3 Fractals and Iterated Affine Transformations

Besides using turtle programs, another popular method for generating fractal geometry is by applying iterated affine transformations (IAT). In this section, we will discuss how IATs are related to RTPs, and how IATs help us better understand the structure of the fractal shapes generated from RTPs by the classical turtle as well as by the hodograph turtle.

An IAT consists of an initial geometry $F_0$, a condensation set $C$, and a set of contractive affine transformations \{A_1, \ldots, A_n\} [6]. The geometry at the $(p + 1)$st iteration is defined inductively by:

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

Since the affine transformations $A_i$ are contractive maps, due to the Contractive Mapping Theorem [6], iterated application of the $A_i$ always converges to the same fractal shape in the limit no matter what initial geometry $F_0$ ($F_0 \neq \emptyset$) we choose to start with. The transformations $A_i$ precisely describe the relationships between the fractal shape and its self-similar components.

In the case of the classical turtle, it is known that any simple recursive turtle program (whose base case introduces the same state change as the recursion body) is equivalent to an IAT that generates the identical fractal [7].

![Table 7](image8.png)

**Table 7.** Turtle programs that generate a fractal C curve and a Koch Snowflake using the classical turtle.
What the turtle draws in the base case of an RTP forms the initial geometry $F_0$ in the corresponding IAT; what the turtle draws while executing the FORWARD commands in the recursion body of the RTP forms the condensation set $C$ in the IAT; and the products of the transformation matrices corresponding to the turtle commands in the recursion body of the RTP form the transformation matrices $A_i$ in the IAT.

Similarly, we can utilize IATs to understand the self-similarity of the fractal shapes generated by the hodograph turtle. Using a method similar to the technique described in [7], we can convert an RTP into an IAT that generates the same fractal as the hodograph turtle. The shape drawn by the hodograph turtle executing the TURN and RESIZE commands in the base case of an RTP forms the initial geometry $F_0$ in the corresponding IAT; the shape drawn by the hodograph turtle executing the TURN and RESIZE commands in the recursion body of the RTP forms the condensation set $C$ in the IAT; and finally, the transformation matrices $A_i$ in the IAT are the products of the transformation matrices corresponding to the turtle commands to the hodograph turtle in the IAT; and the products of the transformation matrices $A_i$ in the IAT are all compositions of rotations and scalings. These specialized IATs account for the revolving and contracting structure of the fractals produced by the hodograph turtle, as witnessed in Figures 6 and 7.

5 Anchor Commands

The fractals generated by the hodograph turtle are rather restricted; their self-similar components are all related by rotations and scalings about a fixed origin. The hodograph turtle cannot produce fractals consisting of translated copies, such as the Sierpenski Triangle. To extend the capability of the hodograph turtle to plot a wider range of fractals requires a more advanced model, a model that frees the hodograph turtle from being tethered to the origin of the coordinate system. By introducing two new commands, ANCHOR-UP and ANCHOR-DOWN, we will enable the hodograph turtle to draw the same class of fractals as the classical turtle.

5.1 ANCHOR-UP and ANCHOR-DOWN

The hodograph turtle is represented by a direction vector $w$ whose tail is always fixed at the origin. We shall now augment the hodograph turtle with a position vector $P'$ that specifies the location of the tail of the direction vector $w$. Instead of connecting the head of $w$ before and after executing every TURN or RESIZE command (when the pen is down), the augmented hodograph turtle draws a line connecting the head of the composite vector $P' + w$ before and after executing each turtle command.

The new position vector $P'$ can be thought of as an anchor for the hodograph turtle. When the anchor is weighed (or up), $P'$ is modified by turtle commands in the same way as the position vector $P$ of the classical turtle. When the anchor is dropped (or down), $P'$ stays fixed and is unaffected by any subsequent turtle commands. To control the state of the anchor, we introduce two new turtle commands, ANCHOR-UP and ANCHOR-DOWN. These two commands weigh or drop the anchor of the hodograph turtle, and have no effect on the classical turtle. To be compatible with our previous definitions, we assume that initially the anchor is down and $P' = \{0, 0\}$. Figure 8 shows the pentagon and star shape generated by the augmented hodograph turtle using the programs in Table 8. Note that due to the introduction of ANCHOR-UP commands, the vertices of the shapes generated by the augmented hodograph turtle need no longer lie on concentric circles centered around the origin. The classical turtle draws the same pentagon and star shapes as in Figure 1, since classical turtles are unaffected by the new anchor commands.

5.2 Fractals

The augmented hodograph turtle can generate a much larger class of shapes than the un-augmented hodograph turtle. For fractals, we have the following result.

**Proposition 1** Consider a recursive turtle program with FORWARD, TURN, RESIZE, PEN-UP, PEN-DOWN, ANCHOR-UP and ANCHOR-DOWN commands. The hodograph turtle generates the same fractal in the limit as
the classical turtle if the following two conditions are satisfied:

1. Both the pen and the anchor are up in the recursion body.

2. In the base case, the pen is down and either
   
   (a) The anchor is up, or
   
   (b) The anchor is down and the turtle commands introduce no net change in the classical turtle’s position vector \( \mathbf{P} \).

Proof: Using the conversion methods presented in Section 4.3, we will show that the hodograph turtle and classical turtle generate the same fractal using an RTP satisfying the two conditions listed above by examining their equivalent IATs. Since by Condition 1 the pen is up in the recursion body, the condensation set \( C \) is empty in the equivalent IATs of the two turtles. Conditions 2(a) and 2(b) guarantee that the states of both the hodograph turtle and the classical turtle are transformed by the same amount in the base case, although the shapes the two turtles draw in the base case can still be quite different. Since by Condition 1 the anchor is up in the recursion body, the turtle commands apply the same transformations to both turtles, and hence the equivalent IATs of the two turtles contain the same transformation matrices \( A_i \). Finally, since iterated application of the \( A_i \) always converges to the same fractal no matter what initial geometry \( F_0 \) we choose to start with (or what the turtle draws in the base case), in the limit both the hodograph turtle and the classical turtle generate the same fractal shape.

Figures 9 and 10 compare the fractal shapes (i.e., the Sierpinski Triangle and the fractal C curve) generated by the hodograph turtle and the classical turtle using the RTPs in Table 9. Note that the program SIERPENSKI_3 on the left of Table 9 satisfies conditions 1 and 2(b) in Proposition 1, while the program C_CURVE_2 to its right satisfies conditions 1 and 2(a). Observe in Figures 9 and 10 that although the hodograph turtle generates different shapes than the classical turtle for the first few iterations, the drawings generated by the two turtles look more and more alike as the number of iterations increases and in the limit these shape will indeed converge to the same fractal.

### 6 Conclusion and Open Questions

We have introduced a new class of turtles for generating 2D geometry, and we have shown that for drawing many simple 2D shapes these hodograph turtles are easier to program than classical turtles. Hodograph turtles also help us to understand how classical turtles generate complicated fractal geometry. By introducing the ANCHOR commands, we have enabled the hodograph turtle to draw the same class of fractals as the classical turtle.

Many questions remain open for further research. Given the close association between the classical turtle and
the hodograph turtle, we would like to know whether theorems such as the Looping Lemma [3] for the classical turtle remain valid for the different variants of the hodograph turtle. Classical turtles have been used to investigate the geometry of non-planar manifolds as well as the warping of space-time under strong gravitational fields [3]. We expect that it would also be worthwhile to investigate how hodograph turtles behave on non-planar manifolds or in strong gravitational fields. Classical turtles that carry a left-hand vector along with a forward vector are studied in [7] in order to simulate arbitrary affine transformations along with the standard conformal maps. The definition of the hodograph turtle could also be extended to include these left-hand vectors. How would these new hodograph turtles behave? What new commands would they require?

Finally, we would be interested to know if the hodograph turtle is easier for young children to learn than the classical turtle. We have seen that in some ways and in certain situations the hodograph turtle is easier to program than the classical turtle. The basic hodograph turtle responds to only one command, TURN, whereas the classical turtle responds to two commands, FORWARD and TURN. Thus to learn to control the hodograph turtle, children can concentrate initially on a single transformation, rotation, rather than needing to understand both rotation and translation as well as how these two transformations interact. New commands such as RESIZE and FORWARD (with ANCHOR-UP) can be presented to children one at a time, introducing children to scaling and translation sequentially instead of all at once. We hope that someone takes up the challenge of teaching the hodograph turtle to young children; we would be very interested in the results.

References