Preprint: To appear in *Computing and Mathematics: The Use of Computers in Undergraduate Mathematics Instruction.* The Mathematical Association of America.

Geometry and Computers

by
James R. King
Department of Mathematics
University of Washington
Seattle, WA 98195

Oversize figures for reproduction

March 19, 1987

Geometry and Computers

James R. King

A computer is a flexible and powerful drawing tool. It can also be a tool for thinking. Therefore, computers have great potential for teaching geometry. However, when one sets out to teach a geometry course using computers, there is no well-trod path to follow. There are many choices to make, both about the mathematics to include and about how the computers will be used. This paper is a discussion of some topics in geometry that lend themselves to computer exploration. It is based on my experience teaching a course in which the computer is the central tool, with students doing their own programming in Logo.

My view is that, while there is no separate subject of "computer geometry," the use of a computer can illuminate geometric ideas that are not as tractable by hand. We already see this phenomenon in the experience that some geometric ideas seem clearer using synthetic methods, while some are easier to think about using analytic methods. It is also true that using both approaches may show more than one face of the same idea. The computer gives a somewhat different view of geometry, which sometimes may not add much, but other times may lead to a very different way of thinking about a geometric situation. Sometimes the contribution of the computer is not so much a new approach as the facility to generate rapidly a host of complex examples. However, this facility can give a subject a whole new flavor. A rich stock of examples leads to better understanding. When one can generate families of lines as easily as one could formerly generate individual triangles, one can take a more experimental approach to subjects that were formerly accessible only to formal argument.

The Method of Turtle Geometry

The drawing tools of the ancient Greeks were compass and straightedge, and the use of these tools shaped the thought in their geometry. With a computer one can draw simply by plotting the (x,y) coordinates of analytic geometry, but while this can be a valuable approach on occasion, it loses the feel of drawing that is present in classical Euclidean constructions, and it may provide little geometric insight. There is, however, a flexible and truly geometric drawing tool for a computer; it is called, curiously, a "turtle." The geometry based on this tool is called *turtle geometry*.

An element of plane turtle geometry is not just a point, it is a *turtle state*, which is a pair consisting of a position (a point in the plane) and a heading (which can be represented by an angle or a unit vector). The turtle drawing tool appears on the computer screen as an arrow located at the position and pointed in the direction of the heading. The drawing commands are a forward command, which moves the turtle in the direction of the heading and draws a line from the old position to the new one, and a turn command which changes the heading. Figures are drawn by a succession of these commands, using the logical organization afforded by the presence of repetition, logical branching, and perhaps recursion in the computer language. Originally, the turtle was a robot which crawled along the floor, drawing with a pen; it looked something like a turtle, hence the name turtle geometry. The turtle in a computer representation can also (metaphorically) raise and lower its pen, depending on whether one wants it to draw when it moves or not.

To use the computer turtle, one gives it commands. The command FORWARD 100 causes the turtle to move its position 100 units in the direction of the heading, while leaving the direction unchanged. The command LEFT 90 causes the heading to be rotated counterclockwise 90 degrees, leaving the position unchanged. Thus a sequence of four commands FORWARD 100 LEFT 90 draws a square, while a sequence of three commands FORWARD 100 LEFT 120 draws an equilateral triangle. Also, in Logo and other suitable languages it is possible to write a procedure, e.g., SQUARE or TRIANGLE, which incorporates the drawing commands for a figure and can be invoked as a new command.

Like most powerful ideas, this is a fascinating mixture of the simple and the profound. Turtle geometry is simple in the sense that ruler and compass constructions are simple; it is accessible to our intuition and builds on our geometric experience. Learning to draw with the turtle is easy enough that young children can learn to draw complex figures. Giving instructions for drawing a figure is very much akin to giving directions to someone driving a car ("you can't miss it; just go straight ahead a quarter mile, then turn left . . ."). The profound part is that this geometry is an intrinsic coordinate-free geometry that unites classical polygons with ideas from modern differential geometry. It turns

out to be well-suited for all sorts of geometric investigations, as we shall see.

An elementary consequence of the intrinsic nature of turtle geometry is that one can draw the pinwheel in Figure 1 without computing any (x,y) coordinates. Just repeat the commands SQUARE RIGHT 36 ten times. Other examples of the value of an intrinsic approach to geometry will appear later.

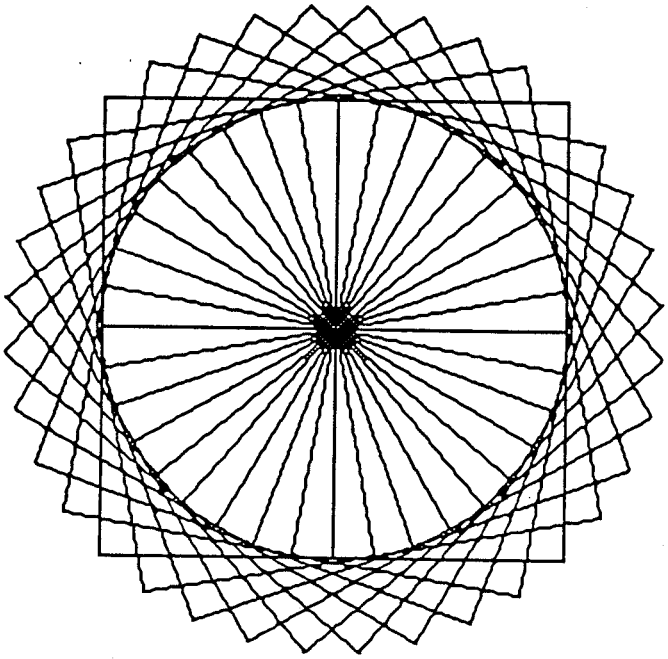

****Figure 1***

Turtle geometry is often associated with the Logo programming language, for it was introduced and advocated by the group at MIT around Seymour Papert which also developed Logo. The merits of Logo as a language for learning mathematics are presented eloquently by Papert in *Mindstorms* [11]; however, the turtle approach to geometry can be implemented in virtually any computer language, although with some limitations. This is explained in the fundamental reference and text by Abelson and diSessa [1].

Foundations of Turtle Geometry

Before proceeding with turtle geometry on computers, I digress to say a few words about its mathematical foundations. Mathematically, an element of the turtle geometry of the plane is a turtle state, a pair consisting of a point in the plane (the position) and a direction (the heading), which we represent by a unit vector. Since the set of unit vectors in the plane forms a circle, the space of all turtle states is a three-dimensional space, the cartesian product of a plane and a circle. There are two natural one-parameter groups of transformations on this space, the forward transformations F_t and the turns L_u.

Let F_t be the transformation which carries any state (P,V) to the state (P+tV,V). Here P is a point, V is a unit vector, t is a real scalar and addition is vector addition. Let R_u be counterclockwise rotation by angle u about the origin. Then let L_u be the transformation which maps (P,V) to (P,R_u(V)). These transformations correspond to the Logo commands FORWARD and LEFT. A general turtle geometry transformation is the composition of these simple transformations. It is not difficult to show that a general turtle transformation T is given by T(P,V) = (P+MV,RV), where R is a rotation R_u and M is a linear transformation of the form M(x,y) = (ax - by, bx + ay), i.e., M is a scalar multiple of a rotation; if the scalar is not 0, this is a similarity transformation. Such transformations can also be represented by complex numbers. Thus turtle geometry can be studied by means of its group of transformations, in the spirit of Felix

Klein's approach to geometry. This group of turtle transformations is a 3–dimensional group.

It is interesting to note that the proper Euclidean group on the plane also defines a 3–dimensional group on the state space, but it is a different group. The transformation S(P) = AP + B, where A is a rotation matrix and B is a position vector, induces the transformation which maps (P,V) to (AP + B,AV). These two groups commute with each other and have only the identity in common. The action of each group is transitive and free on the state space. In other words, if two states are given, there is exactly one transformation in each group which carries the first to the second. This means that the transformation group can be identified with the state space, provided a reference, or "home," state is chosen. The best reference for the foundations of turtle geometry and turtle transformations is [1], although the turtle transformation group itself is not discussed as such there.

Symmetry and Total Curvature

The topic most accessible to the beginning turtle geometer is rotational symmetry. It is easy to draw a rotationally symmetric figure; just repeat any sequence of turtle transformations over and over until the turtle returns to its original state.(This is a slight over-simplification; the correct statement is given below.)

As an example, we consider the simplest turtle geometry procedure, called POLY in the Logo literature. POLY commands the turtle to go FORWARD a distance of S units and turn LEFT A degrees, where S and A are inputs, and then to repeat indefinitely. If A is 360/N, the procedure draws a regular N-gon. For A a rational multiple of 360, say A = (p/q)360, with p/q reduced to lowest terms and q>1, POLY draws a star polygon with q vertices.

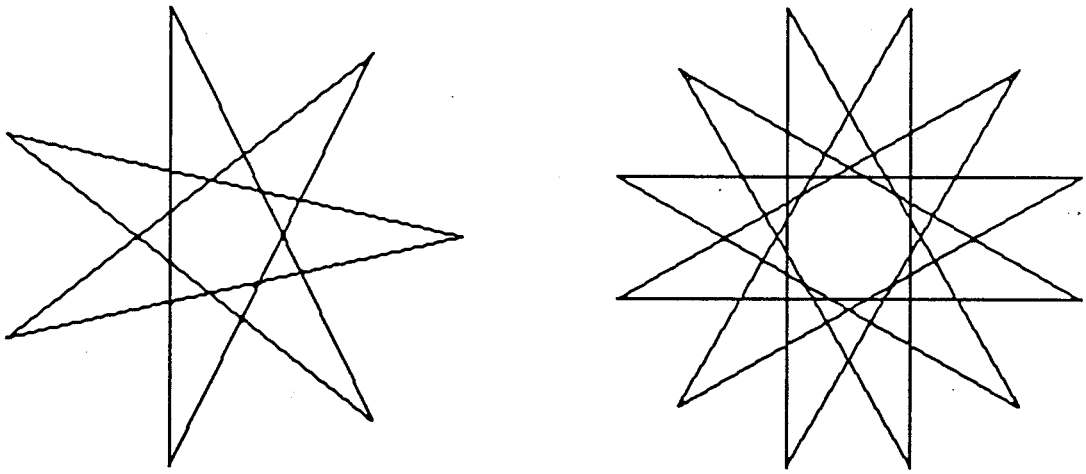

****Figure 2***

The conventional approach to rotational symmetry is to use the center of symmetry Q. The rotational symmetry of the figure is described by the group of rotations with center Q which leave the figure invariant. To draw a regular polygon or star polygon, one would mark off q equally spaced vertices on a circle centered at Q. By connecting consecutive points on the circle, one draws a regular polygon; by connecting every p-th point, one draws star polygons. The turtle geometry approach is to observe that in this construction the direction vectors from one vertex to the next also are equally spaced around the circle of unit vectors, so one draws the figure by turning the turtle in these directions by a sequence of equal turns. (See Figure 3.) One feature that is lost with this approach is that the center of rotational symmetry of the figure must be computed independently, but a very important gain is that it is possible to create very complex rotationally symmetric figures with ease.

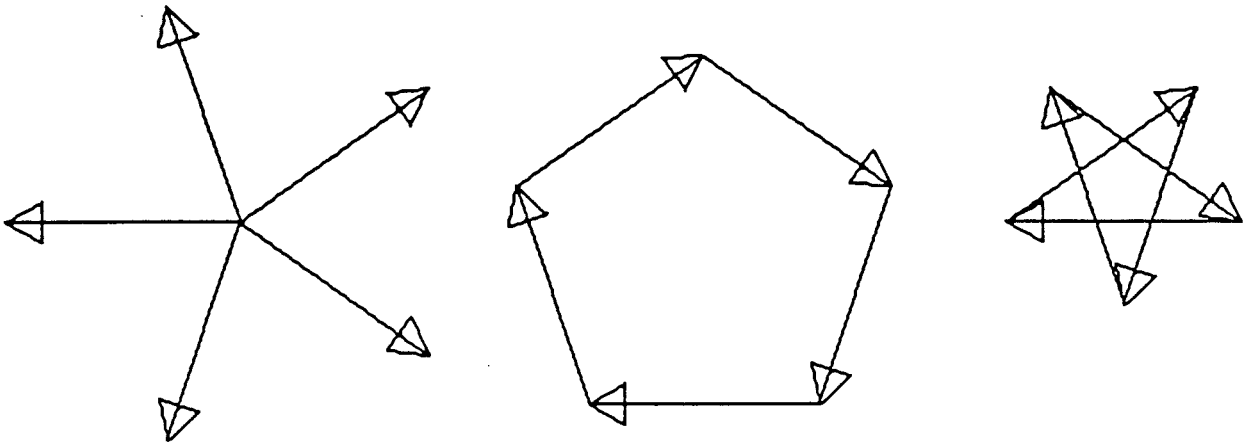

****Figure 3***

The tool that makes this possible is the *Total Turning Theorem* and the concept of *total turn* of a turtle path. The total turn of a turtle path is simply the sum of the degrees of all the left turns in the path (where a right turn by A degrees is viewed as a left turn by -A degrees). The Total Turning Theorem states that for a closed turtle geometry path the total turn is an integer multiple of 360. A companion theorem says that for any simple closed path, i.e., a path without self-intersections, the total turn is ±360; the integer multiple is ±1.

This can be used to analyze figures. For example, in Figure 4, the 7-pointed star traced in a counterclockwise direction has a total turn of 360. If the interior angle at one of the outer vertices is 30, then the turn at this vertex is 180 - 30 = 150. But the figure is made up of 7 congruent parts, each consisting of two segments and two turns. The total turn of each part must be 360/7, as the sum is 360. Thus the turn at the inner, reentrant, vertices must be (360/7) - 150.

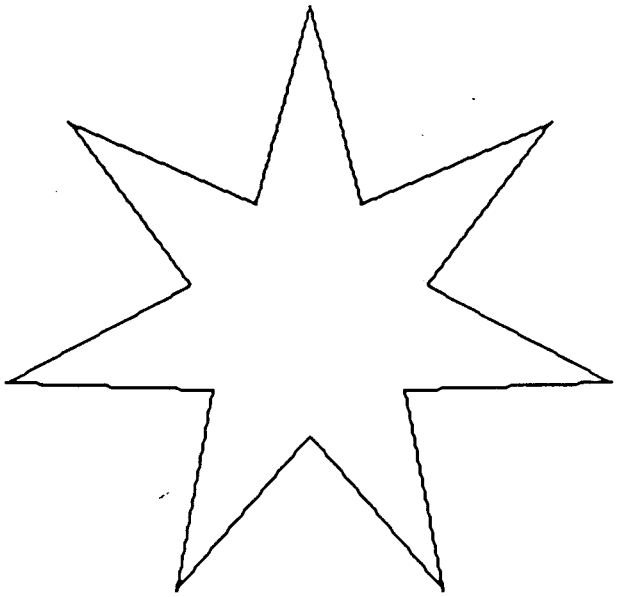

****Figure 4***

The flower in Figure 5 has five petals, each of which is tangent to its neighbors. Since they are tangent, the interior vertex angle of each petal is 360/5 = 72. On the other hand, each petal has a 2-fold rotational symmetry, i.e., rotation by 180 degrees, so each of the two congruent pieces consisting of an arc and a turn must have a total turn of 180. Since the turn is 180 - 72 = 108, the arcs in the petals must have central angle 72 degrees.

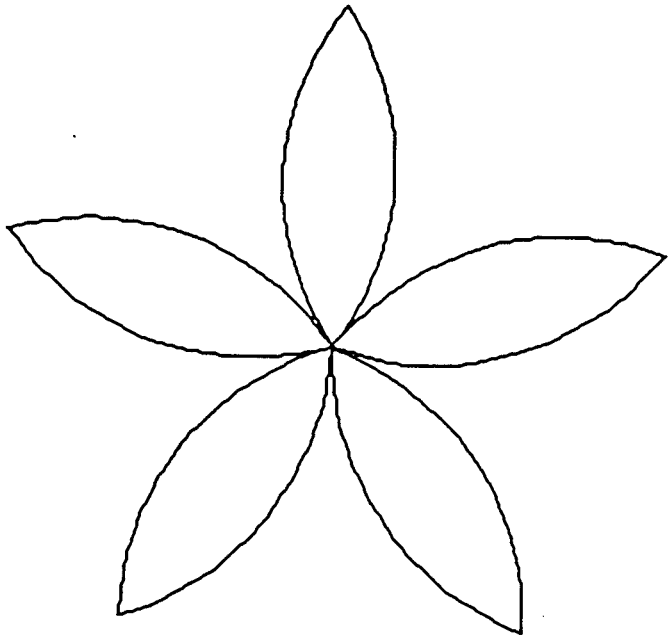

****Figure 5***

The careful reader may object at this point that arcs are not polygons, and this figure is not covered by the Total Turning Theorem. There are two ways to answer this objection. From the computer point of view, arcs are drawn as parts on regular n-gons for large n; for example, this "Logo arc" could be (and was) drawn as 72 sides of a 360-gon. However, what makes this more interesting is the second point of view, that the theorem can be extended to arcs and in fact to piecewise smooth curves in general, where the total turning is replaced by the *total curvature*, the integral of the curvature of the smooth curves with respect to arc length, plus the sum of the turns where the curve is not smooth. It is a theorem of differential geometry that the total curvature of a closed curve is a multiple of 360 degrees (although it is usually expressed in radians). For an arc of a genuine circle, the total curvature is precisely the central angle subtended by the arc, and it is approximately equal to the total turning of a polygon used to approximate the arc. We note also that such piecewise smooth curves define curves in the state space: If X(s) is a curve parametrized by arc length, the curve (X(s),X'(s)) defines a curve in state space; this is a standard technique in differential geometry. By adding turn commands at the corners of the piecewise smooth curve, one can define a continuous curve in the state space and, by projection, a continuous map to the unit circle. Then the Total Turning Theorem becomes a statement about the topological properties of such maps, which are classified by an integer called the *degree*, which in this case is the total curvature divided by 360 degrees.

However, the use of the total turn in this way is not confined to the analysis of existing figures. Suppose that a figure is drawn by repeating a basic sequence of turtle commands indefinitely. Suppose that the total turn of the basic figure is (p/q)360 degrees, q>1. Then q repetitions of the basic sequence of commands draws a closed turtle figure. In other words, if the total turn after repeating the basic figure a certain number of times is an integer multiple of 360, then the figure has closed up. Thus for such looping figures, the converse of the Total Turning Theorem is true.

This converse means that figures can be created by taking the total turn into account. Thus, the analysis of the two figures above is sufficient to create the figures as well. But other figures can result from similar analysis. for

example, the star in Figure 4 has equal sides defining each vertex. If one repeats instead sequence of commands FORWARD S LEFT 150 FORWARD T LEFT 360/7 - 150, then one gets a star which is no longer symmetric with respect to reflection in a line. (See Figure 6.)

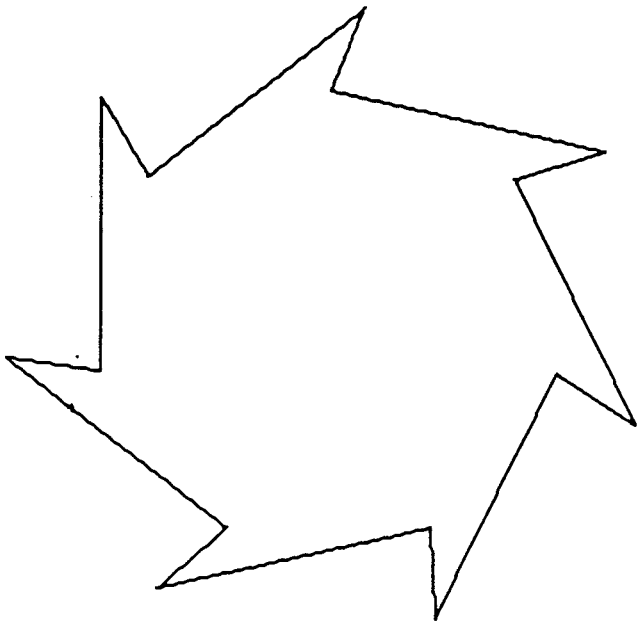

****Figure 6***

One can also create sometimes beautiful figures by simply repeating some random jotting until it closes up. In Figure 7 the doodle on the left is repeated to form the symmetric figure on the right.

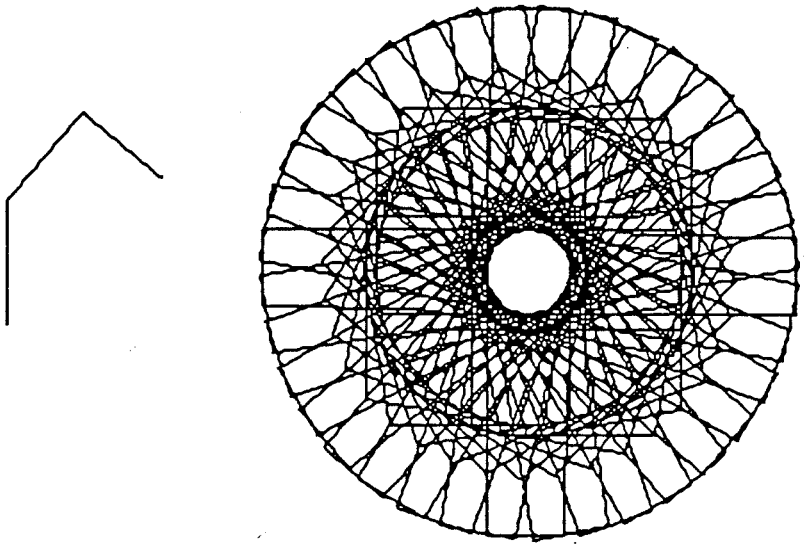

****Figure 7***

In a course one can explore a great deal of interesting geometry with this approach. For example, one can:

- Create and analyze figures made up of lines and arcs and describe their rotational symmetry.
- Write an algebra procedure to reduce fractions to lowest terms, using the Euclidean algorithm for the greatest common divisor, and use this to determine the number of repetitions in looping programs.
- Prove all the standard theorems of high-school geometry about angles and secants and chords of circles by taking turtle trips around the figures.
- Compute the centers of figures; inscribe polygons by connecting corresponding points of looping figures.
- Use recursion to create more complex forms of repetition (e.g., the procedure INSPI, which changes the angle of the turn in a periodic way [1]).

According to one's preferences, all this work can be done as informal but challenging geometry using computer programming alone or it can be accompanied by a traditional treatment of symmetry with the degree of rigor that seems appropriate. The most complete reference for the material of this section, and the only one with proofs, is [1], but almost every Logo book discusses these ideas on some level.

Turtle Geometry and Spirals

Spirals are another very satisfying topic to explore with turtle geometry, since they are easy to generate and create spectacular designs. They are created very much in the style of POLY. One common form of spiral is called POLYSPI in the Logo literature. In addition to inputs of the side length S and the angle A, the procedure has an increment input I. A recursive definition of the procedure is to go FORWARD S, turn LEFT A, then run the procedure again with the side length changed to S + I and the other inputs the same. Thus the figure is composed of turns by a constant angle followed by sides whose lengths increase at a constant rate. (See Figure 8.)

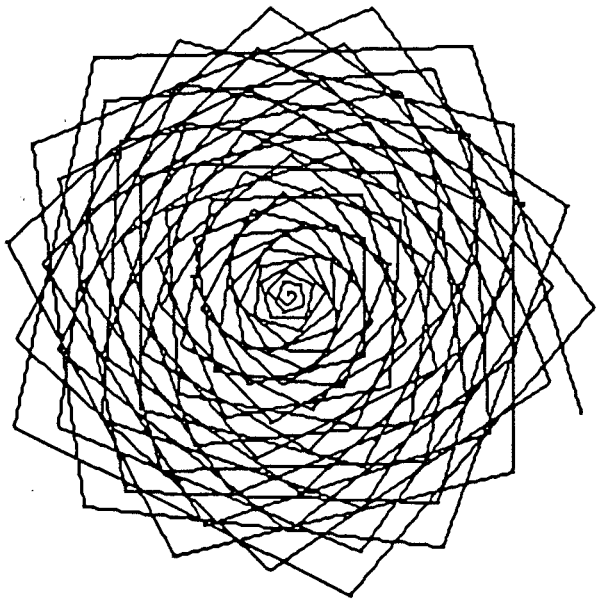

****Figure 8***

One interesting effect is created by choosing the turn angle to be a rational multiple of 360 with small denominator. Then the spiral appears as a growing polygon. On the other hand, if the angle is chosen slightly different from such an angle, a precession phenomenon is obtained. One can experiment to obtain optical illusions and moiré patterns.

The POLYSPI spiral does not have any symmetry transformations which are similarity transformations of the plane. A spiral which does is the equiangular or logarithmic spiral. This is obtained in turtle geometry by making the new

side length a constant multiple of the preceding one. This spiral appears in a number of biological models: By building a chain of similar figures (e.g., triangles or quadrilaterals), one obtains rather natural-looking shells or horns, and the procedure for creating them mimics the process that occurs in nature. (See Figure 9.)

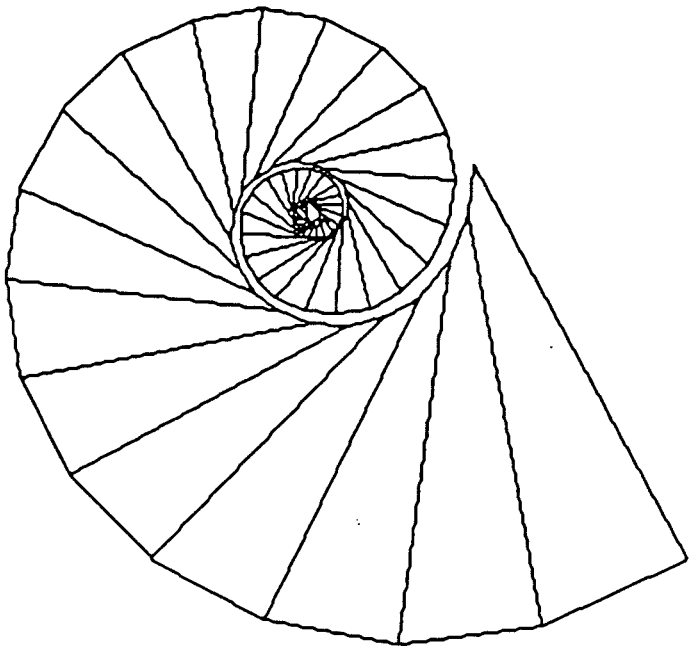

****Figure 9***

One strength of turtle geometry in this kind of investigation of complicated figures is that is quite easy to study how the figures change if parameters such as angle or ratio are changed. This leads to thinking about deformation, stability, and other "dynamic" aspects of geometry [1], [15].

Turtle Geometry and Fractals

Fractal geometry is an excellent topic for a computer geometry course. It is exciting because it is surprising for students and because it is an active area of contemporary mathematical research. Even better, a computer makes this sophisticated mathematics accessible for experimentation by undergraduates.

Since turtles draw curves, the part of fractal theory that works best with turtle geometry is the theory of fractal curves. These are curves topologically, but metrically their dimension is greater than one. An extreme example is the Peano curve which fills a square region, but examples that fill less space can have quite beautiful geometry. For example, the Koch snowflake is obtained as the limit of a sequence of curves. The first curve is a triangle and the second is obtained by replacing the straight line segments in the triangle by congruent "lumps" consisting of four segments. At each successive stage, the next curve is obtained by replacing every segment in the previous curve with lumps similar to the first ones. The limiting curve is a curve of Hausdorff dimension great than 1. On the computer one must stop before infinity, but after a number of stages a curve of great beauty and complexity is obtained. (See Figure 10.)

Using this pattern of taking a figure and a model lump and recursively replacing every segment in the figure by a lump similar to the model, it is possible to obtain an extremely varied collection of curves of mind-boggling complexity. The process of actually writing a computer program to draw such a curve requires the student to really understand the recursive construction, and the feedback from trying out the program makes it possible to correct possible misconceptions. Turtle geometry is almost essential to the writing of this sort of program, for it is simple to describe the lump with turtle commands and then draw it at any size and at any location in the plane, whereas the use of Cartesian coordinates leads to very difficult and opaque computations.

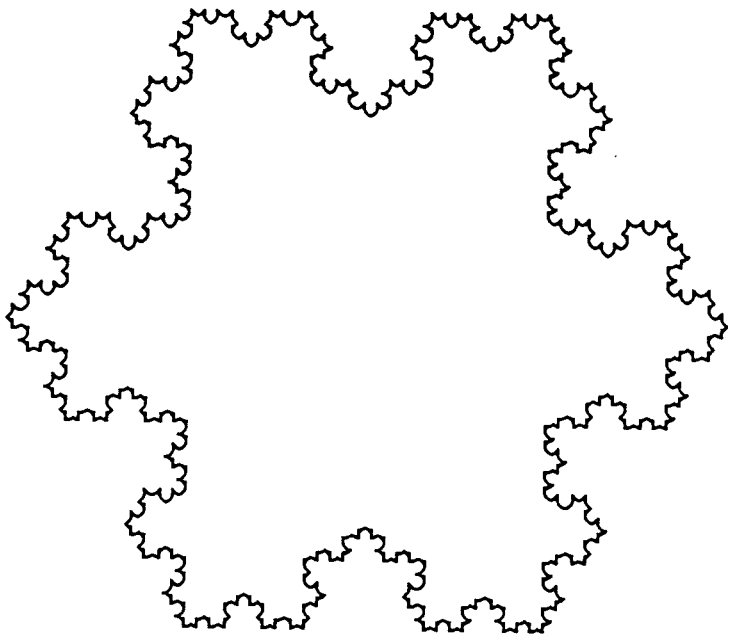

****Figure 10***

Examples of fractal curves can be found in [1] and [15], but anyone interested in fractals must refer to the fundamental book [10] for the explanation of the theory of fractals, for the magnificent collection of pictures (which can serve as a source of countless experiments), and for its bibliography. The examples of curves which are limits of polygonal paths are the most accessible to turtle geometry, but using a computer with fast graphics commands to draw arcs and fill shapes, it is possible to approach some of the other examples as well.

Tilings

The study of tilings of the plane is much more accessible to students using the traditional tools of geometry than is the study of fractals, but it can also benefit from the introduction of computer drawing, because reproducing a tile over and over is so labor-intensive, and because the patterns in the logical structure of the program echo the mathematical pattern of the tiling. With a computer one can complement the proof of the theorem about which regular n-gons tile a neighborhood of a point in the plane by writing a program which actually draws the successful and unsuccessful cases on the screen. Understanding how to determine which n-gons to try in the program and when all the possibilities are exhausted is very close to understanding the rigorous proof.

One can write programs that exhibit the standard tilings of the plane, then modify the tiles in some artistic way to get patterns inspired by the drawings of Maurits Escher or by the designs of the Moors.

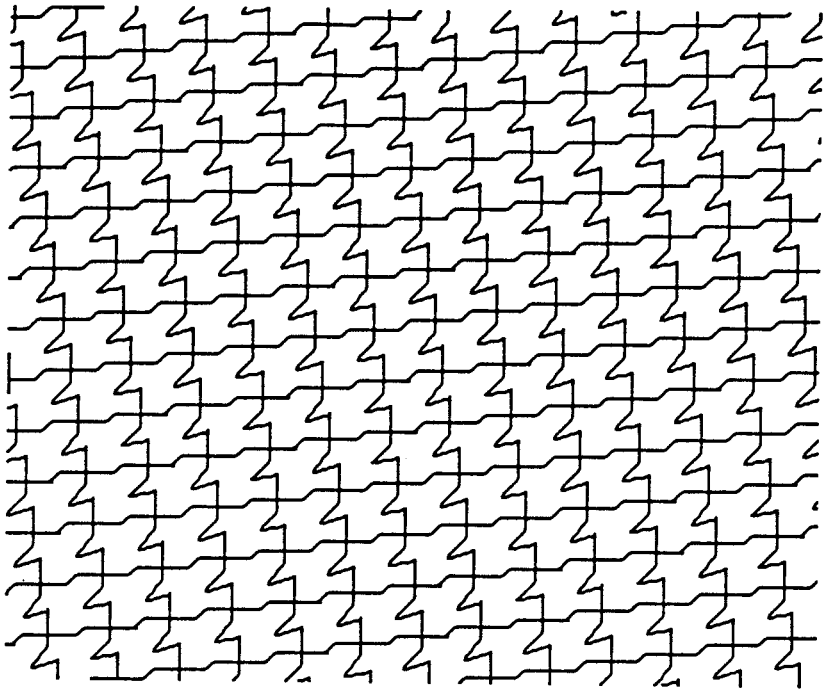

****Figure 11 ***

There are examples of this in [1] and [15], but richer sources of ideas are [6] and [14]. There are also some books, such as [2], intended for younger students with exercises for pencil and paper which can be converted to computer programs.

Line Conics and String Figures

There are numerous curves of classical geometry and modern geometry that are defined as the envelope, or outline, of a family of lines. One example is the construction of conics as line conics. The simplest such constructions are done at the kindergarten level with string. A child creates a string figure on some pattern, say the two sides of an angle drawn on cardboard, by punching holes regularly along the pattern and then connecting the n–th hole of the first side to the n–th hole of the second side by sewing with string or yarn. This makes a pretty pattern. The outline, or envelope, of the strings is a parabola. (This is a special case of a theorem of projective geometry that states that if the points of two lines are related by a projective transformation, the lines connecting corresponding points form a line conic.)

It is easy to simulate this child's play with turtle geometry. One just moves the turtle along each line and saves a list of points for each. Then one draws a line through each pair of corresponding points. Now with essentially the same program, one can draw other string figures which produce ellipses and hyperbolas. We run the turtle around two circles, marking points and connecting them as before. Depending on the relative positions of the circles and the starting points, conics of varying shapes are obtained.

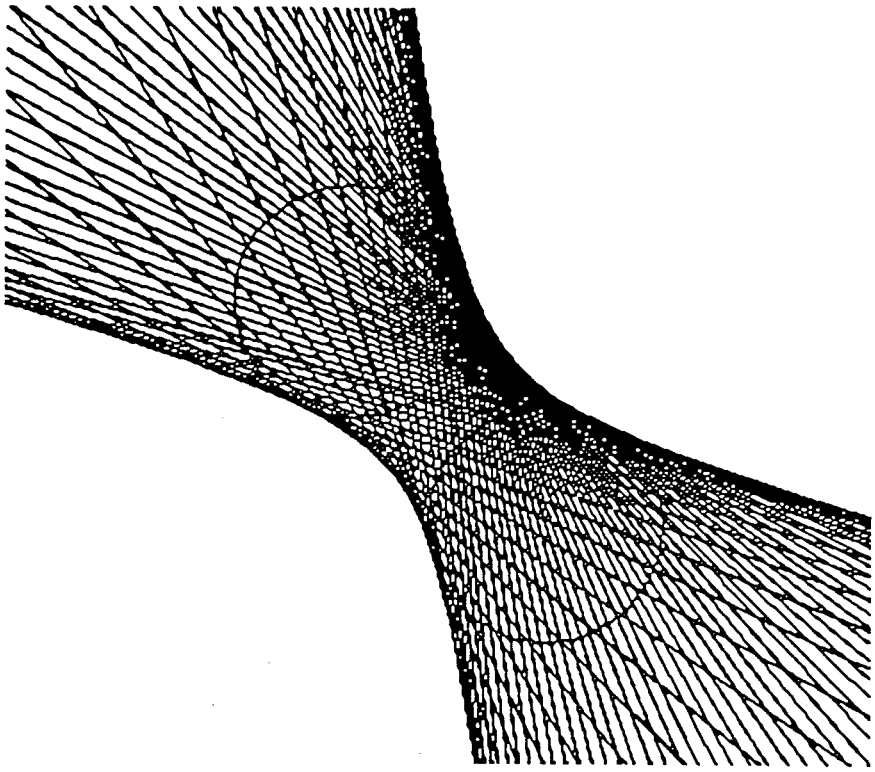

****Figure 12A***

Now we change the rules a bit. For example, we use one circle for marking the points, but the second time we let the turtle go twice as far for each step (and so twice around the circle). The envelope of the lines connecting these points is a cardioid. If the ratio of steps is 3 to 1, a nephroid results; if 4 to 1, a three-cusped epicycloid. There are other classical constructions of line conics which are interesting to draw with computers. For instance, we choose a circle and a point P not on the circle. For every point Q on the circle, we draw the line through Q perpendicular to the segment from P to Q. Then the envelope of this family of lines is a conic. For P inside the circle an ellipse results; for P outside, a hyperbola. For more details about these envelopes, see [12]. Another book with a treatment of curves easily adapted to computers is [9].

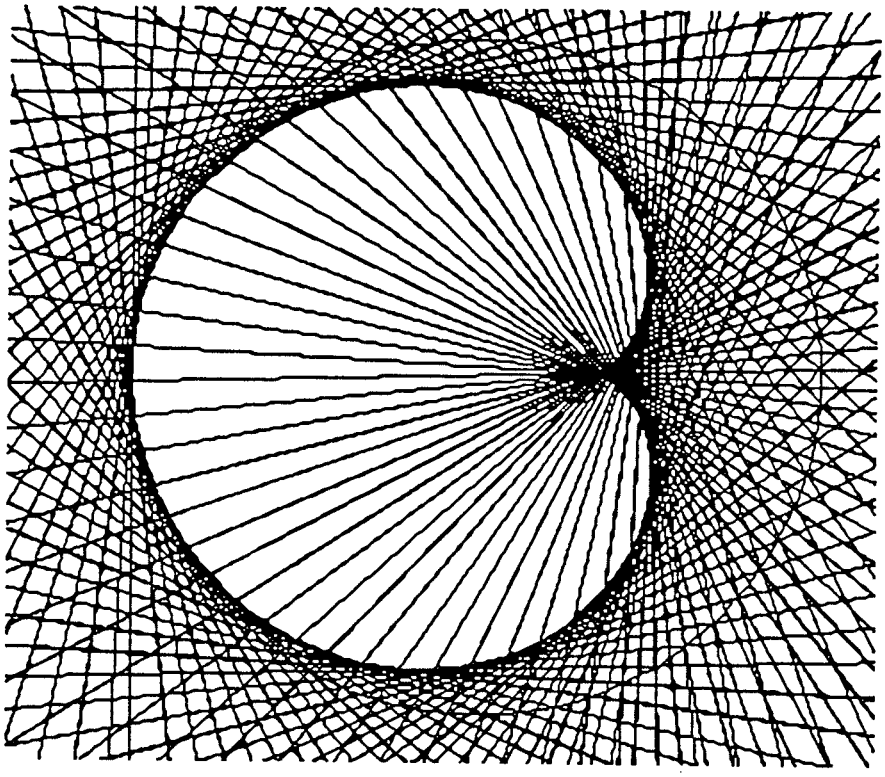

****Figure 12B***

Another interesting way to produce families of lines is to draw a curve (e.g., a spiral) made up of turtle steps and turns. After each step draw the normal line to the curve, i.e., the line normal to the direction of the turtle. For a smooth curve, the envelope of the normals is called the evolute. While the evolute of some smooth curves is also smooth (e.g., the evolute of a logarithmic spiral is a similar spiral), usually the evolute has singularities. The evolute of an ellipse is an interesting example. (In the section on affine turtle geometry below we discuss one method for producing ellipses.) For a general discussion of evolutes, see [7].

When lines are interpreted as light rays, the envelope is a caustic, the locus where the light is focused. Look at the bright curve of light in your coffee cup; it is possible to simulate this curve by choosing a circle (more generally an ellipse) and a point P not on the circle. For each point Q on the circle, draw the line which is the reflection of the line PQ across the tangent line of the circle at Q. (In other words, reflect the point P across the tangent line to get P' and draw the line P'Q.) If P is at the center of the circle (or a focus of the ellipse), the lines all pass through a single point, but otherwise they define an envelope which is the bright curve on the surface of the coffee. For the details of this, see [3]. A very thorough and accessible discussion of envelopes and related ideas is in [4].

Cycloids and Mechanical Devices

Another classical method for describing curves is illustrated by the definition of a cycloid as the locus of a point on a circle which rolls along a line. In other words, if a light is attached to the rim of a rolling wheel and this is photographed by a time exposure, then the trace of the light in the picture is a cycloid. Similar definitions are given for epicycloids and hypocycloids, which are the loci of points on circles rolling outside or inside other circles. It is not difficult to simulate this type of definition with a computer. One can simply plot the position of the point using analytic geometry, but one can see the nature of the geometric construction better by having one turtle go around the first circle and another turtle go around the second circle, and then taking the vector sum of the position vectors relative to the centers of the circles. (A more difficult feat would be to create an animated simulation of the rolling circle.)

This opens up another vast area of classical geometry wherein curves are defined as loci of points on various mechanical devices. Another example is the locus of a point on a ladder resting on a floor as the ladder slides down a wall (an ellipse). In this case, one can also look at the envelope defined by the segments representing the ladder (an astroid). In fact, one can recall the string constructions connecting pairs of points and plot the locus of the midpoint of each segment or the point dividing the segment in some fixed ratio. Examples of these constructions are found in [9], [12], [16]. These constructions are elementary to program. A more difficult project along the same lines would be to simulate one or more of the classical devices for performing constructions such as inversions in a circle, e.g., Peaucellier's linkage [5].

Affine Geometry and Home-brew Turtles

Affine geometry is the geometry of lines and parallelism without the introduction of angle and distance. Plane affine geometry is the geometry that is preserved under orthogonal projection from one plane in three-space to another. Thus circles and ellipses are indistinguishable in affine geometry, and all triangles are equivalent. But the notion of parallelogram makes sense in affine geometry.

It is possible to modify the commands of turtle geometry so that drawing an ellipse is as easy as drawing a circle. The idea is to slant the plane of the turtle so that what appears on the screen is an ellipse, although the turtle "thinks" it is drawing a circle.

In the section on the foundations of turtle geometry, we said a turtle state was a position and a heading; an equivalent definition would be a position and a moving coordinate system, since in the oriented Euclidean plane, a coordinate system is determined by the unit vector along the first axis. We chose the unit vector on the first axis to be the turtle heading and the unit vector on the second axis by rotating the heading counterclockwise by 90 degrees. Let a state be given by (P,V,V'), where V' = R(90)V. Then the FORWARD command is given as before, by $F_t(P,V,V') = (P + tV,V,V')$, but the LEFT command is defined to be $L_u(P,V,V') = (P,W,W')$, where

$$W = (\cos u)V + (\sin u)V', \text{ and}$$
$$W' = (-\sin u)V + (\cos u)V',$$

according to the usual formulas for rotation. Now to slant the plane, we "deceive" the turtle by letting the turtle state be (P,H,L), where H and L are two independent vectors which need not be orthogonal or of unit length. These vectors define a moving coordinate system which is affine but need not be Euclidean. Then if L_u is defined by the same formulas, $H' = (\cos u)H + (\sin u)L$ and $L' = (-\sin u)H + (\cos u)L$, a sequence of turtle commands will draw a figure as it would be projected on a plane in such a way that the original Euclidean coordinate vectors would be projected to H and L. The vectors H and L define the affine moving frame of the turtle.

Various commands other than L_u can be devised to change one such affine frame to another. The simplest would be a SETFRAME command which takes a new frame as input and changes the frame to the new one. Other useful intrinsic commands would be ones to rescale one or the other of the frame vectors by multiplying it by a scalar. It is also possible to define a FLIP command as a special case of this, -1 as the scalar. Notice that the L vector (which is the "left lateral" vector by convention in the definition of the left turn L_u) can actually be on the right. In this case, all left turns will appear on the screen as right turns.

These tools can be used with a CIRCLE procedure to draw ellipses. (See Figure 13.) Similarly, they can be used with SQUARE to draw parallelograms or with TRIANGLE to draw arbitrary triangles. Mirror images can be drawn by using the same procedure twice, once before and once after the FLIP command, and shadows can be constructed with SETFRAME. The same tools can be used to illustrate which properties of Euclidean geometry are affine invariants and which are not (e.g., angle bisectors are not, medians are).

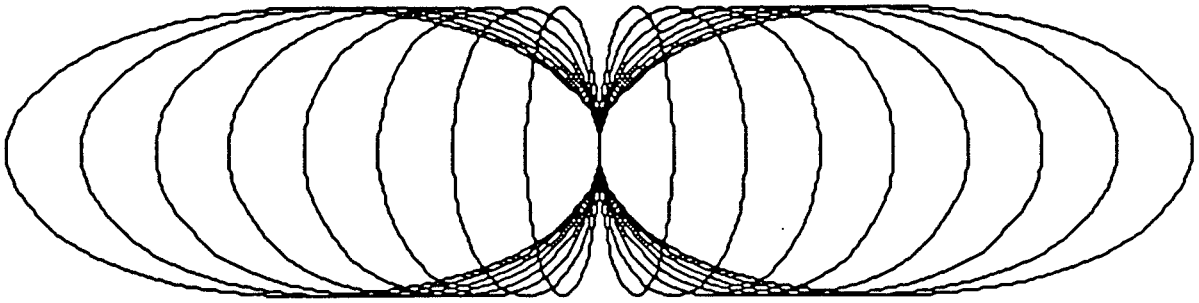

****Figure 13***

To create such a "home-brew" turtle in Logo or some other language, one must do two things. The first is to create the "logical turtle." This means that some variable or variables are used to store the information about the turtle state, and then some procedures are defined to alter this state by the rules of F_t and L_u. As a practical matter it may be necessary to give these commands slightly altered names so as not to interfere with the usual turtle commands (e.g., AFORWARD or AFD and ALEFT or ALT). The second stage is to use the standard turtle as a drawing tool to represent the home-brew turtle on the screen, the "screen turtle." This means that after each command to the logical turtle, the screen turtle is moved to a new position and/or heading depending on the new state of the logical turtle.

In addition to defining an affine turtle as described, home-brew turtles can be used to define multiple turtles for language implementations that only come with one turtle (or none). Multiple turtles are very convenient for constructions like the string figures, which naturally have two or more loci of points. A reference for defining turtles is [1]. A Logo implementation of the affine turtle is described in [8].

Three-dimensional Geometry

A few implementations of turtle graphics have three-dimensional geometry built in, but most do not. However, home-brew three-dimensional turtles can be defined roughly as the affine turtles were defined. The main difference is that they carry a frame of three vectors. (For Euclidean geometry, the vectors are orthonormal.) There are also three turning commands, one for each coordinate plane. Commanding this turtle is more like flying an airplane than driving a car.

Another difference is that the position of the logical turtle is in three-space but the position of the screen turtle is in the plane of the screen. After each change in turtle state, the new position is projected to a new planar position for the screen turtle. This is one of the more interesting aspects of these turtles, for one has a choice of how to map three-space onto the plane of the screen. Two natural choices are orthogonal projection and central projection. It is very interesting for students to see how a figure in three-space that they have created changes with different choices of projection. How to move the figure or change the projection so that the figure looks the expected way becomes an important practical problem, and solving it provides some much needed practice with the geometry of space.

There are many things to do with this space-turtle geometry. It is interesting to draw simple figures like cubes and prisms. By drawing a family of line segments in space, one can generate ruled surfaces such as cylinders, cones and hyperboloids. One can then point out to the students that the pictures on the screen are identical to some of the string figures and that there is a relationship between envelopes and projections of surfaces in space.

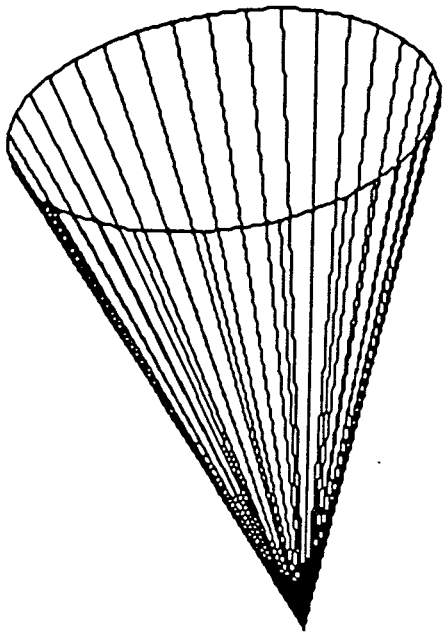

****Figure 14***

It is an interesting exercise to have students construct a three-dimensional scene with buildings and roads and then project it so that it looks like a typical perspective drawing with a vanishing line "at infinity" near the center of the screen and with the roads coming together at this infinity. This is harder than it looks, because the student has to understand how planes in the construction must be related to the projection. It gives some insight into the rudiments of projective geometry.

Some of my students have constructed the Platonic solids on the screen, with the duals in contrasting colors. Others have created aerial views of cities, three–dimensional tilings, and three–dimensional fractals.

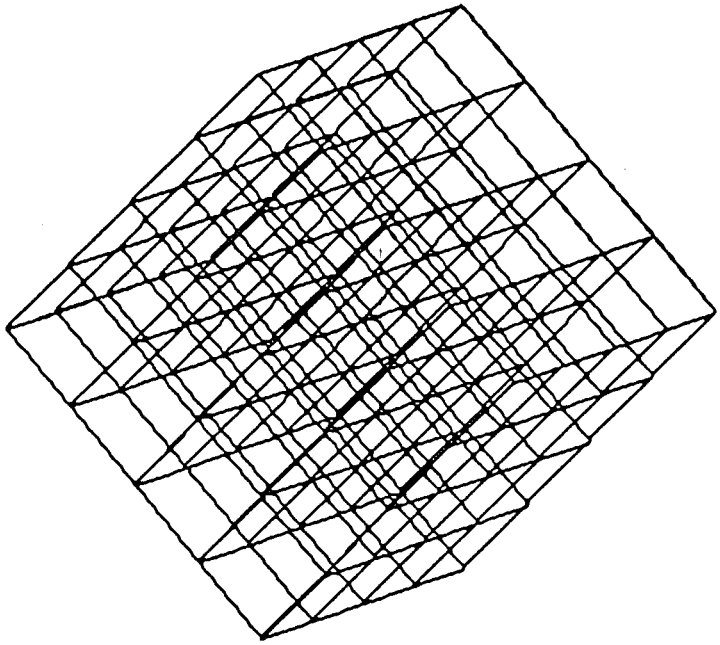

****Figure 15***

Although the geometry of space is very rewarding, it is definitely more difficult to explore with turtle geometry than plane geometry. First, the geometry itself is harder. Second, it is harder to represent the objects of the geometry; for example, the wire-frame representation of surfaces has many drawbacks. Finally, home-brew three-dimensional turtle geometry is much slower than plane geometry, at least in Logo, although on the newer and more powerful machines this is less excruciating. References for three-dimensional turtle geometry are [1] and [13].

Algebra and Geometry

It is important to understand the various ways of representing mathematical objects. For example, a rotation can be represented by a vector formula, a matrix, or a complex number. It is possible to explore this circle of ideas geometrically by first defining various algebraic operations such as complex multiplication and matrix multiplication in a language such as Logo. Then one can plot the points generated by recursively multiplying the current turtle position (viewed as a complex number) by another fixed complex constant and view the results. If the modulus of the constant is one, then one gets a polygon; if it is not, a logarithmic spiral results. Thus one obtains the same figures studied earlier by turtle geometry, and the student can be asked to understand the relationship between the two approaches.

Students can also be asked to create procedures that perform the operations of vector algebra, such as vector sum and dot product. Then they can use these operations to write a procedure which will project one vector onto another and draw the picture at the same time. Getting the picture to match the algebra seems to be a useful exercise leading to better comprehension of what these vector operations really mean.

The equation of a line, as elementary as it is, can be used for a number of drawing experiments. First consider a procedure that simply draws the line on the screen. This actually has a few ideas in it, for one must decide in advance what kind of data about the line will be given. Students sometimes find it a challenge to take a procedure that they have written to draw a line given two points and use it to define a procedure which draws a line given the equation (where are the points?). This forces one to think about the various ways of representing lines and how one can systematically pass from one to another. This illustrates an important idea that can be difficult for students: When reasoning about an object, it is very important to take into account the representation of the object.

Some of this is a little austere for the typical student, so it is important to ask geometric questions which require him or her to confront these issues. For example, if a line is determined by its normal direction and a point P on the line, what happens to the line if the normal direction is rotated, and simultaneously the point P is rotated about the origin perhaps at a different rate. This is closely connected to the string figure geometry described above, and the rotational symmetry of the envelope is based on the same reasoning about angles as was used above in our discussion of symmetry.

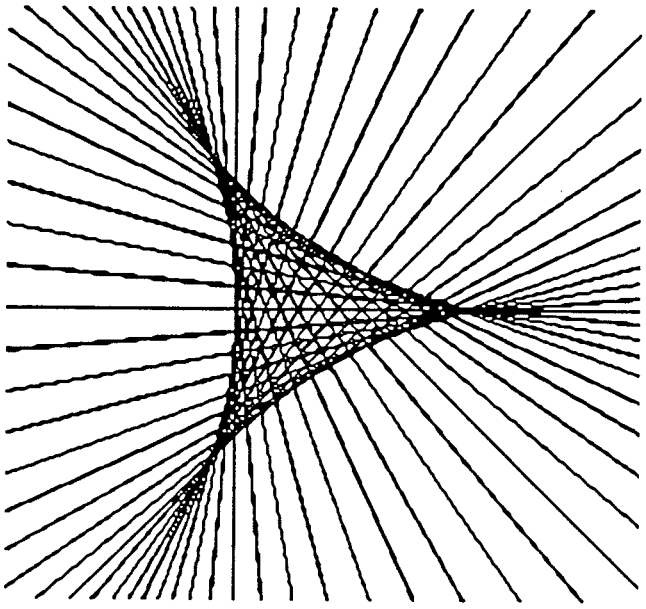

****Figure 16***

Once one has a representation for lines and a means to draw them, then one can ask how they interact with turtles. For example, given a line, we may write a procedure that will move the turtle forward until it meets the line. (What to do when the turtle must move backward to meet the line, or will not meet it at all, depends on what we plan to do with this procedure.) This is a good exercise in the use of the dot product. Then we may write a procedure which will bounce the turtle off the line like a billiard ball. After this it is possible to simulate the trajectory of a billiard ball on a table of any shape.

One interesting aspect of the projects in the preceding paragraph is that they will will work unaltered for three-dimensional geometry as well, provided they are written in terms of vector operations such as dot product and not explicitly in terms of coordinates. Thus the billiard table program becomes a racquetball program, since the equations for planes in space are the same as the equations for lines in the plane.

Other Topics

There are many interesting questions that one can raise about convexity and connectivity of regions. For example, given a finite set of points in the plane, we may write a program that will draw the convex hull of the points. Or, given a simple closed polygonal curve, we can determine whether a given point is inside or outside the curve, or create a procedure that will fill the interior with cross–hatching. This sort of activity borders on the questions that are studied in linear programming and in the computational geometry used in computer graphics.

Inversive geometry, the geometry of inversion in circles, benefits greatly from computers, since examples are generated more quickly. Also, on computers with very fast arc-drawing routines, such as the Macintosh, it is possible to build a complete turtle geometry of the Poincaré disk model of hyperbolic non-Euclidean geometry that works fairly briskly. One can also consider envelopes of families of curves that are not lines, such as circles or parabolas.

Projective geometry is a very tempting subject, but it would require a careful approach to make it accessible to the beginning programmer, since figures always run off the screen. Some of the synthetic constructions of conics would be similar to the constructions of string figures and loci described above.

An ambitious but beautiful topic is the study of the curvature of surfaces and of space, which is the topic of the last half of [1]. It includes both the geometry of the sphere and the (metric) geometry of the cube. The former has constant positive curvature while the latter has curvature concentrated only at the corners. This is explained by defining turtle geometry on these spaces and using the Gauss-Bonnet theorem to define curvature.

Conclusion

I hope this sampling will generate some ideas about what can be done in a geometry course in which the students do extensive programming. Based on my experience, if one wants to teach such a course, the important thing is to learn a modicum of programming, read a couple of books and then jump in. Some mistakes are inevitable, but once you get some experience with looking at geometry from the programming point of view, all sorts of ideas will present themselves. Look in old tomes for partially forgotten mathematics and look in new books such as [6], [10], [14] always asking the question "would this be interesting on computers?"

Also, look at classical geometry theorems with new eyes. For example, there is a theorem about hyperbolas that says that if a point P is on a hyperbola, the rays from P to the foci are on opposite sides of the tangent line and make equal angles with this line. Thinking about turtles, we say this: Suppose A and B are two points and a turtle moves so that A is to the left and B is to the right, with the rays toward the two points always making the same angle with the heading. Then the turtle moves along a hyperbola. This is actually a recipe for drawing a hyperbola with a turtle. (The actual drawing will be a discrete approximation to the correct curve, which is described as a solution to the differential equation that appears implicitly in this formulation of the theorem.)

This account has concentrated on what one can teach with computers and has begged the question of why one should want to do so. There is too much to say about this to really address the question in this article, but I would like to conclude with some brief observations based on my experience. The geometry seems more immediate and alive to students; one can bring in modeling problems that show the geometry of growth in nature. The disciplined creative act of writing a program is closely akin to writing a proof, but a geometric program has the advantage that it will immediately draw a picture that will tell you whether you have the ideas right. This immediate feedback is like having a Socratic tutor on call at all times. Students can think more like mathematicians in that they can generate their own geometric questions and then perhaps solve them; they can also begin with simple cases and work their way up to general understanding. This is the only undergraduate mathematics course I have ever taught in which the students do original projects. Also, it is easier for students to work at various levels of competence and expertise and still learn something and experience some success with mathematics; conventional courses are very prone to baffle the weaker students and bore the more able ones. There are the factors of surprise and serendipity: Perhaps one does something with squares and sees spirals on the screen which reveal hitherto unnoticed relationships. Finally, one additional reward for the teacher is that this geometry is new and interesting for him or her, too, and this different point of view generates fresh pleasures from geometry.

REFERENCES

1. Abelson, H., and diSessa, A., *Turtle Geometry*, MIT Press, Cambridge, 1980.

2. Bezuska, S., Kenney, M., Silvey, L., *Tesselations: The Geometry of Patterns*, Creative Publications, Palo Alto, 1974.

3. Bruce, J. W., Giblin, P. J ., and Gibson, C. G., Caustics Through the Looking Glass, The Mathematical Intelligencer, Vol. 6, No. 1 (1984), 47-58.

4. Bruce, J. W., and Giblin, P. J., *Curves and Singularities*, Cambridge Univ. Press, Cambridge (UK), 1984.

5. Coxeter, H. S. M., Introduction to Geometry, Wiley, New York, 1961.

6. Grünbaum, B., and Shephard, G. C., *Tilings and Patterns*, Freeman, New York, 1986.

7. Guggenheimer, H., *Differential Geometry*, Dover, New York, 1977. (originally, McGraw-Hill, New York, 1963)

8. King, J., The Affine Turtle: A Turtle That Can Draw Shadows, Logo Exchange, Vol. 5, No. 7 (1987), pp.15-16.

9. Leapfrogs, *Curves*, Leapfrogs, Cambridge, 1982. (ISBN 0 905531 29 9)

10. Mandelbrot, B., *The Fractal Geometry of Nature*, W. H. Freeman, 1982.

11. Papert, S., *Mindstorms: Children, Computers, and Powerful Ideas*, Basic Books, 1980.

12. Pedoe, D., *Geometry and the Visual Arts*, Dover, New York, 1983. (originally *Geometry and the Liberal Arts*, St. Martin's Press, 1976)

13. Reggini, H., Towards an Artisanal Use of Computers, Logo 86, The Third International Logo Conference, MIT, Cambridge, 1986.

14. Stevens, P. S., *A Handbook of Regular Patterns: An Introduction to Symmetry in Two Dimensions*, MIT Press, Cambridge, 1981.

15. Thornburg, D., *Discovering Apple Logo: An Invitation to the Art and Pattern of Nature*, Addison-Wesley, 1983.

16. Yates, R., *Curves and their Properties*, National Council of Teachers of Mathematics, 1974.