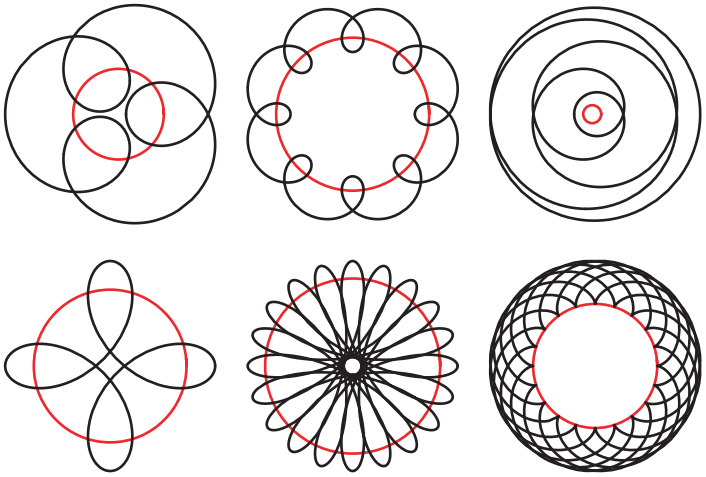
Around and around

Andrew Glassner

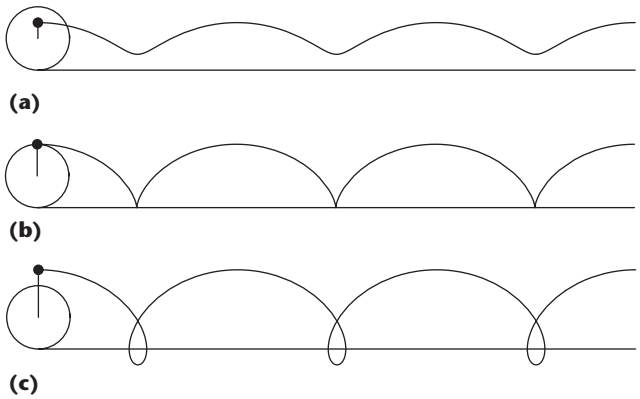
Everybody loves making pictures with a Spirograph. This wonderful toy was introduced in 1966 by Kenner Products and is now manufactured and sold by Hasbro.

The basic idea is simplicity itself. The box contains a collection of plastic gears of different sizes. Every gear has several holes drilled into it, each big enough to accommodate a pen tip. The box also contains some rings that have gear teeth on both their inner and outer edges. To make a picture, you select a gear and set it snugly against one of the rings (either inside or outside) so that the teeth are engaged. Put a pen into one of the holes, and start going around and around. The result is a pretty, swirly design, like the pictures in Figure 1.

I got to thinking about this toy recently, and wondered what might happen if we used other shapes for the pieces, rather than circles. I wrote a program that produces Spirograph-like patterns using shapes built out of Bezier curves. I'll describe that later on, but let's start by looking at traditional Spirograph patterns.



1 Several Spirograph-style roulettes.



2 Three cycloids. In this figure, the wheel radius $r = 1$, and the pen is at a distance h from the wheel's center. (a) $h = 0.5$, (b) $h = 1.0$, and (c) $h = 1.5$.

Roulettes

Spirograph produces planar curves that are known as *roulettes*. A roulette is defined by Lawrence this way: "If a curve C_1 rolls, without slipping, along another fixed curve C_2, any fixed point P attached to C_1 describes a roulette" (see the "Further Reading" sidebar for this and other references). The word *trochoid* is a synonym for roulette. From here on, I'll refer to C_1 as the *wheel* and C_2 as the *frame*, even when the shapes aren't circular.

The simplest way to create a roulette is probably to use a circle for the wheel and a straight line for the frame. This results in a figure that has the special name *cycloid*. Where we choose to put our pen tip results in three different types of curves, as we can see from Figure 2.

Spirograph only has a few curve types available—as shown in Figures 1 and 2a—where the pen is inside the circle. To make other types of curves, it might be useful to attach a rigid arm to the center of the disk. As the disk rotates, the arm spins with it.

Determining whether the pen tip is inside the rolling shape, right on its edge, or outside of it, is a useful tool for distinguishing among different roulette types. Figure 3 shows the mathematical names for the different types of roulettes. The first distinction is whether the wheel is

 Published by the IEEE Computer Society

Further Reading

I found a bit of history on Spirograph at the Kenner Toys Web site at http://www.kennertoys.com/history.html.

There's a ton of information on roulettes available on the Web; just go to Google and type in "roulette." A great reference for all kinds of curves is J. Dennis Lawrence's paperback book, *A Catalog of Special Plane Curves* (Dover Publications, 1972).

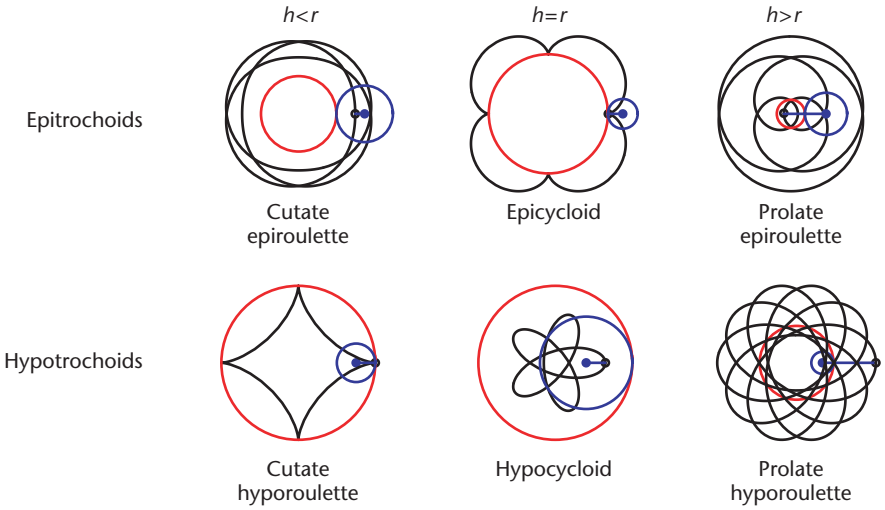
You can find lots of the mathematical details behind epitrochoids at http://www.math.hmc.edu/faculty/gu/curves_and_ surfaces/curves/epicycloid.html and similar information for hypotrochoids at http://www.math.hmc.edu/faculty/gu/curves_and_surfaces/curves/hypocycloid.html, as well as at http://mathworld.wolfram.com/Epitrochoid.html.

Two nice galleries of pretty roulettes are available at http://aleph0.clarku.edu/~djoyce/roulettes/roulettes.html and http://www.xahlee.org/SpecialPlaneCurves_dir/EpiHypocycloid_dir/epiHypocycloid.html.

My information on Nasir al-Din as-Tusi in the "Some History" sidebar came from http://mathworld.wolfram.com/TusiCouple.html.

inside or outside of the frame. If it's outside, we call it an *epitrochoid*; otherwise it's a *hypotrochoid*. If the pen is right on the edge of the wheel, then it's an *epicycloid* or *hypocycloid*. If the pen is not on the edge, then the curve is an *epiroulette* or *hyporoulette*. There are two forms of each of these. If the pen is inside the wheel, then we have a *cutate* epiroulette or hyporoulette, otherwise it's a *prolate* epiroulette or hyporoulette.

There are some special cases of these curves that mathematicians have studied over the years. I'll mention these briefly, without going into detail. (You can learn more about the background behind these curves in the sidebar "Some History.") In each of these



3 Distinguishing among the roulettes. The upper row shows epitrochoids, because the wheel is outside of the frame. The lower row shows hypotrochoids. The wheel has radius *b* and the pen is a distance *h* from its center.

Some History

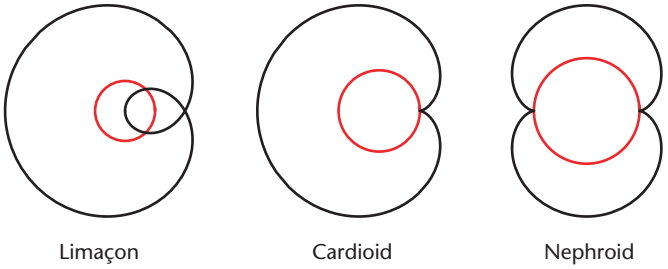
Although the Spirograph is a wonderful toy for exploring roulettes, mathematicians started studying them long before the 1960s.

In 200 BC, the astronomer Apollonius of Perga described the motion of celestial objects with combinations of circles. In 150 BC, Hipparchos of Nicaea followed up on this idea and worked out the apparent motions of the planets and the sun using circles. Ptolemy popularized this model in 150 AD, and his name became associated with the idea of a solar system that had the Earth in the center, with everything else spinning around the Earth in orbits that were described by combinations of circles. As these planets rotated around the Earth, and rotated themselves, they traced out epicycloids.

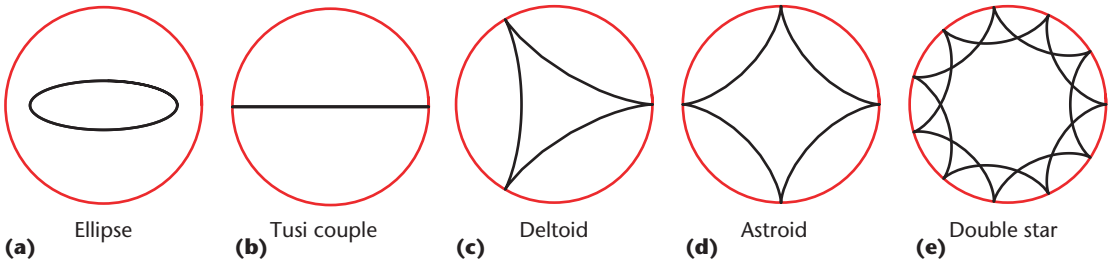
In 1525, Albrecht Dürer wrote about epicycloids, calling them "spider lines" in his book *Instruction in Measurement with Compass and Straight Edge.* Since then, epicycloids have been studied by a host of famous mathematicians, including Desargues, Huygens, Leibniz, Newton, de L'Hôpital, Jakob Bernoulli, la Hire, Johann Bernoulli, Daniel Bernoulli, and Euler.

Many roulettes have special names, given to them by mathematicians who studied their properties. Among the epitrochoids, R.A. Proctor named the nephroid in 1878 in his book *The Geometry of Cycloids*. The full name of the limaçon is the Limaçon of Pascal (limaçon means "snail"). The Pascal here is not the famous Blaise Pascal, but his father, Etienne. De Castillon named the cardioid in a 1741 paper.

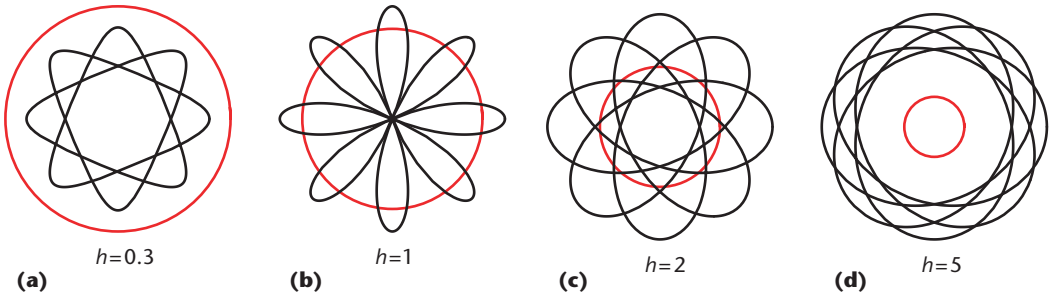
Among the hypotrochoids, the ellipse was probably first studied in 350 BC by Menaechmus, and named in 200 BC by Apollonius. Euler studied the deltoid (or tricuspid) in 1745. Roem first studied the astroid (or tetracuspid) in 1674. The name of the straight line called the Tusi couple comes from the Persian astronomer and mathematician Nasir al-Din al-Tusi, who studied this shape in the late 13th century.



4 Three special epitrochoids: (a) A limaçon ($r_f = r_w$), (b) a cardioid (an epicycloid with $r_f = r_w$), and (c) a nephroid (an epicycloid with $r_f = 2r_w$).



5 Special hypotrochoids. At the far left is (a) an ellipse ($r_f = 2r_w$). The other figures are all hypocycloids (so $h = r_w$). (b) A Tusi couple ($r_f = 2r_w$), (c) a deltoid ($r_f = 3r_w$), (d) an astroid ($r_f = 4r_w$), and (e) a 10-pointed star.



6 Rosettes are hypocycloids ($h = r_w$) generated by the formula given in the text. The four curves here are generated with $n = 4$, using different values of h. From left to right, (a) $h = 0.3$, (b) $h = 1$, (c) $h = 2$, and (d) $h = 5$.

special cases, both the wheel and the frame are circles. I'll call the frame circle C_f with radius r_f, and the wheel C_w with radius r_w. The distance of the pen tip from the center of C_w is h.

Figure 4 shows three special epitrochoids. A *limaçon* is created when $r_f = r_w$. The *cardioid* is the epicycloid case of the limaçon, created when $h = r_w$ and $r_f = r_w$. The *nephroid* is another epicycloid, resulting from $r_f = 2r_w$.

Figure 5 shows five special hypotrochoids. You get an *ellipse* when $r_f = 2r_w$. The other examples are all hypocycloids, so $h = r_w$. The *Tusi couple* is what you get when drawing an ellipse with $r_f = 2r_w$: It's a straight line. The three-pointed star called the *deltoid* appears when $r_f = 3r_w$, and the four-pointed star called the *astroid* is created when $r_f = 4r_w$. You probably get the general idea that you can make an n-pointed star with $r_f = nr_w$. Figure 5 also shows a 10-pointed star.

Figure 6b shows a special hypocycloid called a *rose* or *rosette*, sometimes also called *rhodonea*. These are hypocycloids where the center of the flower passes through the center of the frame. You can generate rosettes with different numbers of petals with this formula:

$$r_f = \frac{2nh}{n+1}$$
$$r_w = \frac{(n-1)/h}{n+1}$$

The flower in the figure uses $n = 4$, and a variety of values of h.

Now that we've looked at a bunch of particularly special epitrochoids and hypotrochoids, let's gather up the families for some group portraits and see what they look like together. Figure 7 shows the epitrochoids, and Figure 8 shows the hypotrochoids. We can use Spirograph to make all the curves in Figure 7, and the upper-left quadrant of Figure 8. In the right side of Figure 8, we'd need an extension arm to get the pen tip outside of the inner gear, and on the bottom, the gear is too big for the inner ring and would need to pass through it.

The geometry

I didn't draw Figures 7 and 8 by hand. It turns out that it's easy to derive a simple formula for the two different roulette types. Implementing these formulas is easy in

just about any modern programming language.

The easiest way to set up the geometry is in Figure 9 (next page). For the illustrations, I'll use a wheel that's smaller than the frame. I'll never make use of that condition in the math, so the geometry works no matter what the relative sizes are between the wheel and frame.

We'll put the larger circle (the frame), with radius R, at the origin (which I'm calling point A), and the smaller circle (the wheel), with radius r, on the x-axis to the right of the origin, just touching the frame. So the center of the wheel is $B = (R + r, 0)$. The point P representing the pen tip is at a distance h to the right of B, so $P = (R + r + h, 0)$. At the start, $P_0 = B_0 + h$.

The parameter t rotates the wheel around the frame counterclockwise by t radians. So for any given value of t, the center of the wheel is at $B_t = (R + r)(\cos(t), \sin(t))$. As the wheel rolls around the frame, it too rotates counterclockwise, as Figures 9b and 9c show. If we can find out how much the wheel has rotated for a given value of t, we can rotate P about B_t by that amount, and we'll have our pen tip location P_t.

Let's label everything as in Figure 9d. The angle we've rotated by is t, shown at the center of both the frame circle and the wheel. The wheel has rotated by an angle v. From the drawing, we can see $v = u + t$, so now we need to find the angle u.

The trick to finding u is to remember that since the wheel isn't slipping as it rotates, the arc EF around its perimeter has the same length as the arc DE around the perimeter of the frame. Writing $|EF|$ for the length of arc EF, we know that $|EF| = ur$, and similarly, $|DE| = tR$. So $|DE| = |EF|$ means $tR = ur$, which tells us $u = t(R/r)$. Now we can plug this back into our expression for $v = u + t = t + [t(R/r)] = t[1 + (R/r)]$. Let's write this as $v = ct$ where $c = (R + r)/r$.

So now we can find P_t by taking B_t and adding $h(\cos(ct), \sin(ct))$:

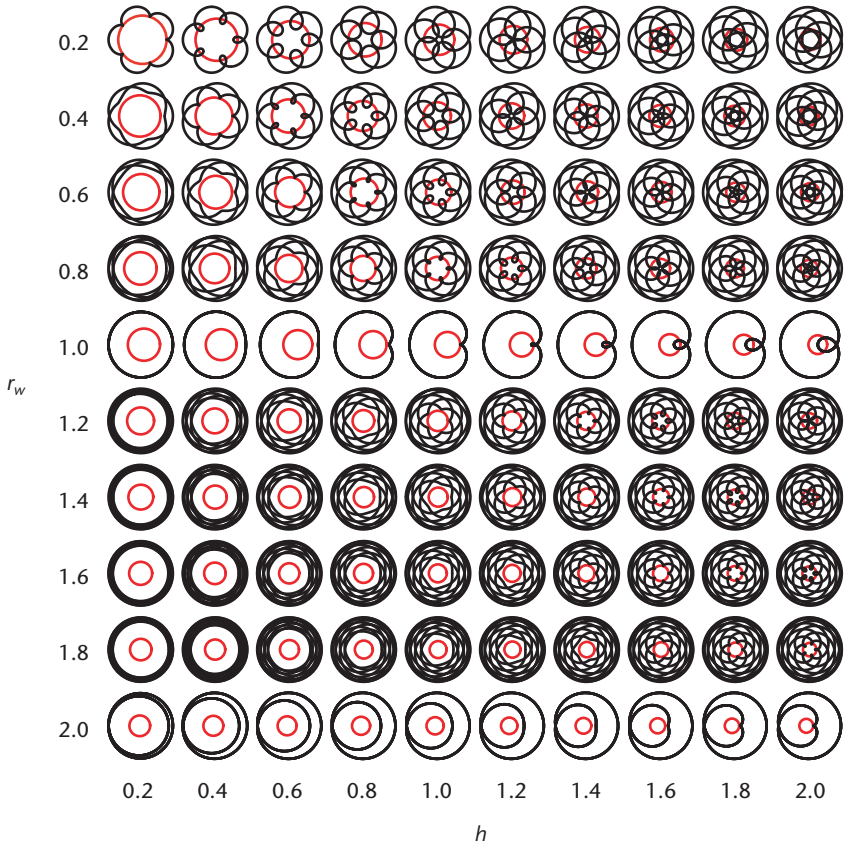
$$P_t = (R + r)(\cos(t), \sin(t)) + h(\cos(ct), \sin(ct))$$

where $c = (R + r)/r$.

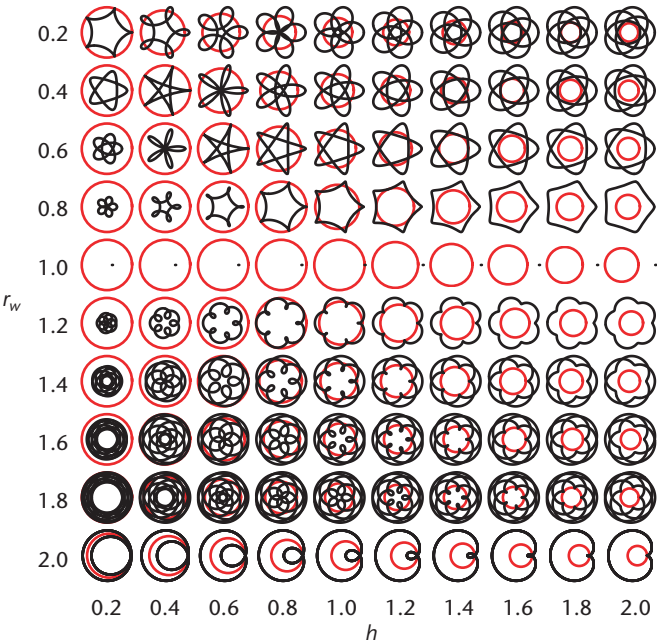
The explicit parametric form of the epicycloid is thus

$$x_e(t) = (R + r)\cos(t) + h\cos(ct)$$
$$y_e(t) = (R + r)\sin(t) + h\sin(ct)$$

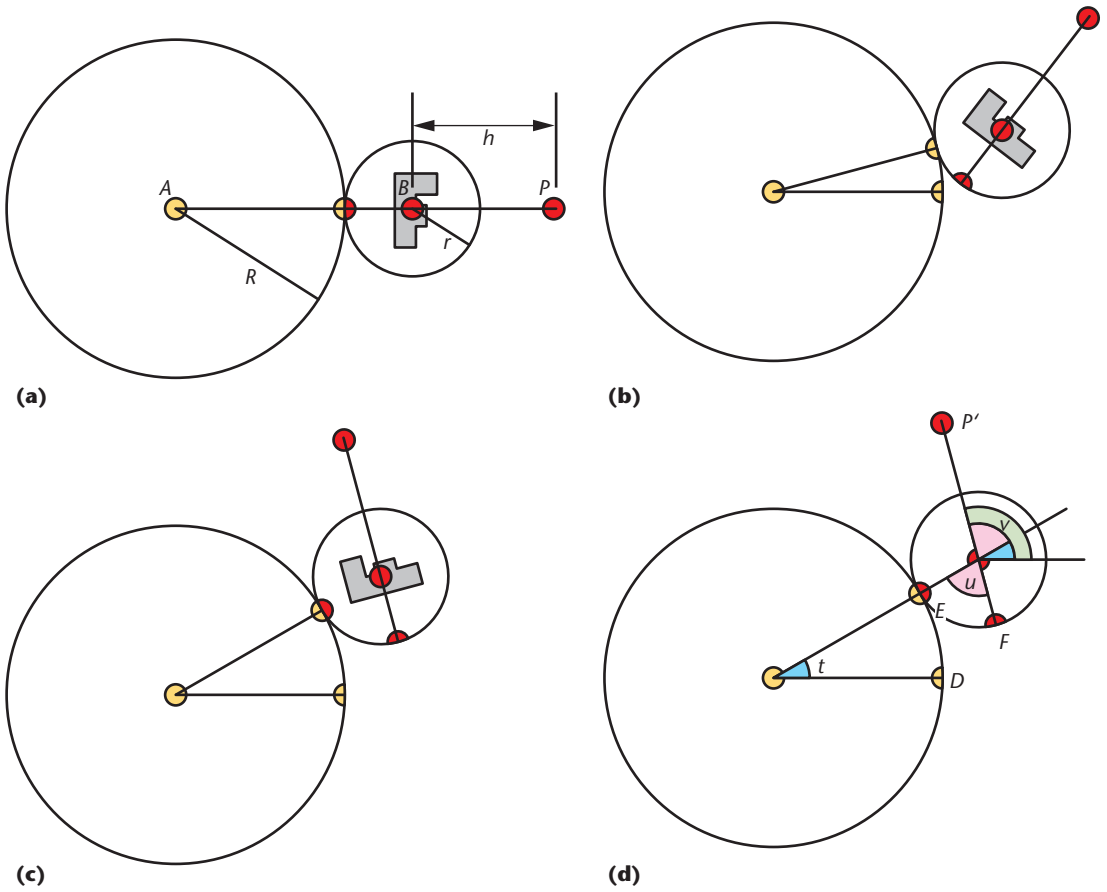
The hypocycloid follows the same reasoning. As Figure 10 shows, we put the wheel inside the frame this



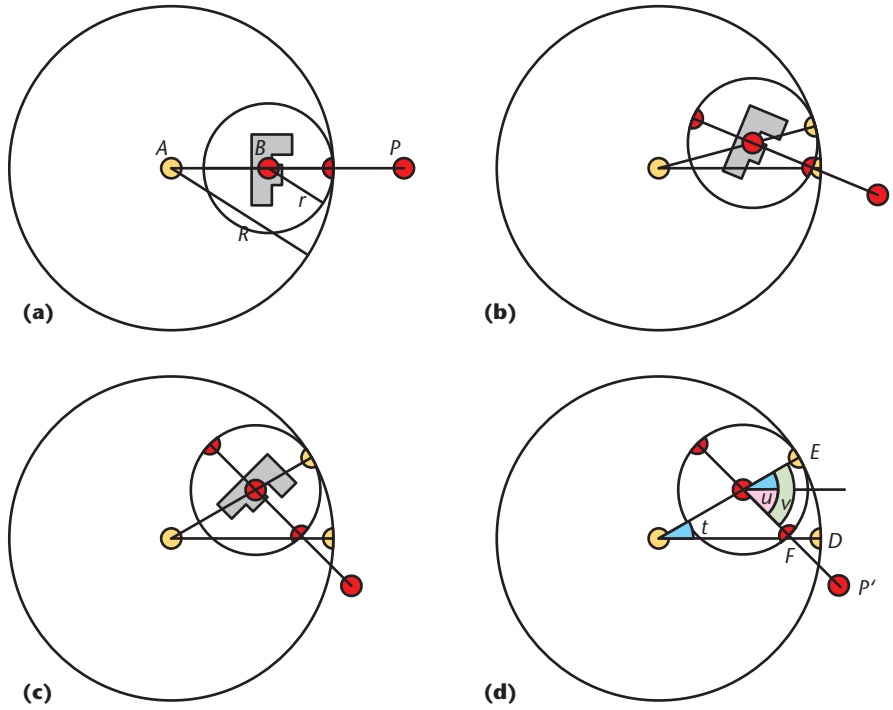
7 Grid of epitrochoids. The horizontal axis shows different values of h, with a vertical axis r_w. Throughout, $r_f = 1$.



8 Grid of hypotrochoids. The horizontal axis shows different values of h, with a vertical axis r_w. Throughout, $r_f = 1$.



9 The geometry for the epitrochoids. (a) The wheel in its starting position. (b) The wheel after a little bit of rotation. (c) The wheel after more rotation. (d) The rotated wheel's geometry.



10 Geometry for the hypotrochoids. (a) The wheel in its starting position. (b) The wheel after a little bit of rotation. (c) The wheel after more rotation. (d) The rotated wheel's geometry.

time. Figures 10b and 10c show that as the wheel rotates it turns clockwise.

From the figure we see that $|DE| = |EF|$, or $tR = vr$, giving us $v = t(R/r)$. Observing that $v = t + u$, we isolate $u = v - t = t(R/r) - t = t[(R/r) - 1]$, or $u = dt$, where $d = (R - r)/r$. We want to rotate P not by u but by $-u$,

because the wheel is turning clockwise. Write this out as

$$P_t = (R-r)(\cos(t), \sin(t)) + h(\cos(-dt), \sin(-dt))$$
$$= (R-r)(\cos(t), \sin(t)) + h(\cos(dt), -\sin(dt))$$

where $d = (R-r)/r$, and we noted that $\cos(-x) = \cos(x)$ and $\sin(-x) = -\sin(x)$ for any x.

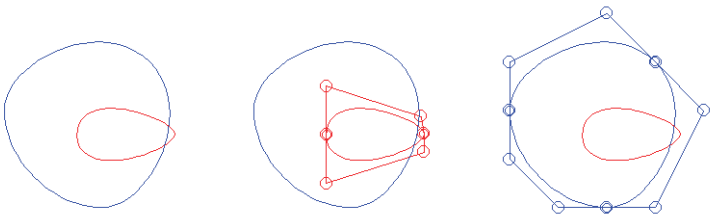
The explicit parametric form of the hypocycloid is thus

$$x_h(t) = (R-r)\cos(t) + h\cos(dt)$$
$$y_h(t) = (R-r)\sin(t) - h\sin(dt)$$

From circles to Beziers

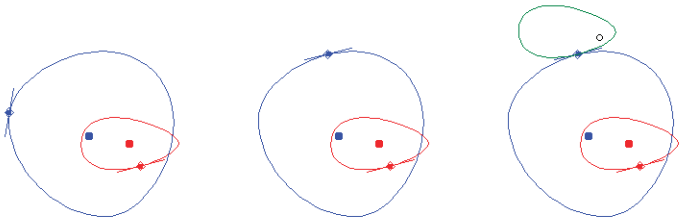
To write a more general curve-based Spirograph program, I decided to use Bezier curves. There are many curve representations out there, but Beziers are very simple to program, numerically stable, and easily controllable. Maintaining smooth continuity across Bezier segments is also easy. For a little project like this, they're just about perfect.

Figure 11 shows the basic idea using screen shots from my system. I have two curves: the wheel (in red) and the frame (in blue). I can toggle on and off the display of control points for each curve independently. When the knots are displayed, I can simply click in them and drag them where I want.



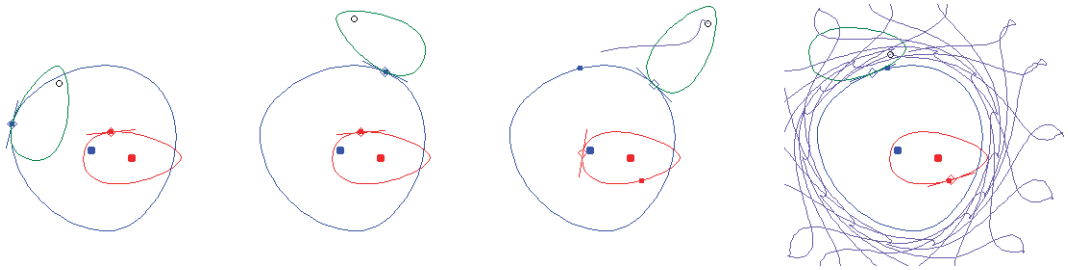
(a) (b) (c)

11 Creating shapes with Bezier curves. (a) The curves. (b) The Bezier polygon for the wheel. (c) The Bezier polygon for the frame.



(a) (b) (c)

12 Setting the starting points. (a) Choosing a pair of zero points. The solid dots show the curve's center, the diamond on the curve is the zero point, and the local tangent is also drawn. (b) A different pair of points. (c) The wheel translated and rotated into position, shown in green.

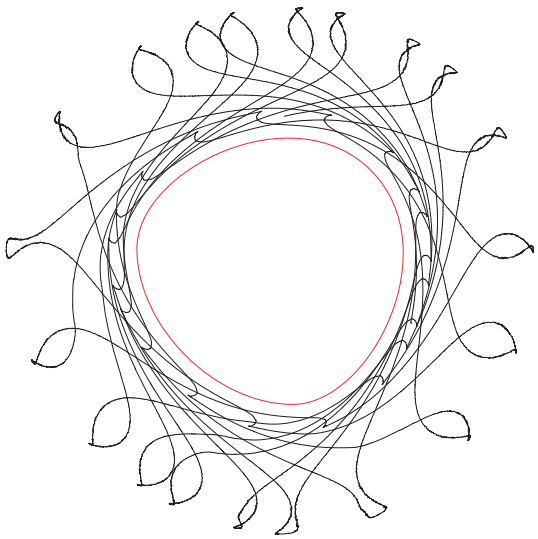


(a) (b) (c) (d)

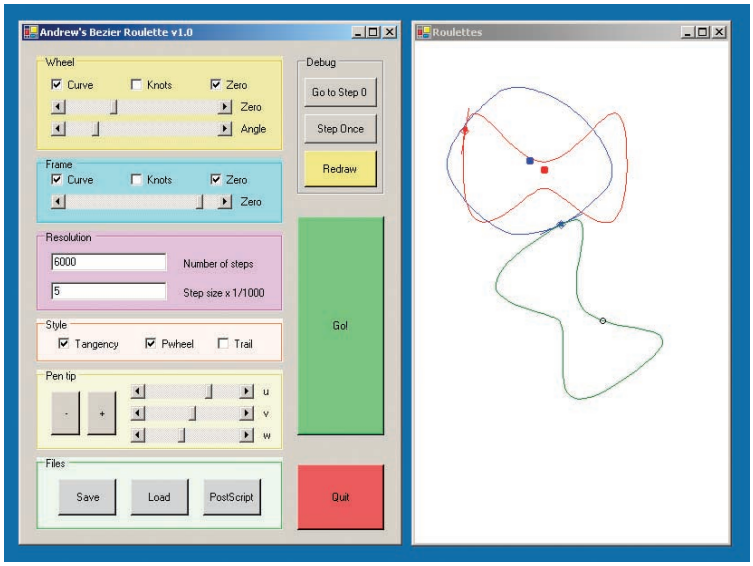
13 Starting off a drawing. The green version of the wheel is the moving copy. (a) Starting the wheel inside the frame. (b) Starting the wheel outside the frame. (c) Figure 13b after a few steps; the wheel has rotated as it moved around the frame. (d) After many more steps.

Once I have the curves shaped the way I like them, I assign what I call *zero points*. These are the initial points of contact between the two curves. I have a slider in the interface that lets me pick the zero point anywhere on each curve, as Figure 12 shows. Once I've identified the zero point on each curve, I snap them together. The wheel moves to the frame so that the zero points are coincident, and then it rotates so that its local tangent is parallel to the frame's tangent.

It's important to specify whether the wheel is inside or outside the frame. Figure 13 shows the same curves, but starting inside and outside. Figure 13c shows the outside version after a few steps. You can start to see the trail left behind by the pen tip. Figure 13d shows the result after many more steps. Figure 14 shows the curve resulting from this pair of shapes.



14 Curve created by the initial setup in Figure 13b.



15 My Bezier roulette program in action. The control panel is on the left, and the graphical design window is on the right.

Programming

Figure 15 shows a picture of my system in action. The control panel is on the left, and the design window is on the right. I'll discuss some of the programming ideas as we walk through the controls.

At the top of the control panel is a cluster of controls in a pale yellow box. These relate to the wheel. Using the checkboxes, you can enable or disable the display of the curve itself, its Bezier polygon, and the zero point. The upper horizontal scrollbar lets you specify where the zero point is: If the zero checkbox is on, then as you move this scrollbar from the left to the right you see the zero point make a full orbit around the wheel. The lower scrollbar lets you rotate the curve into any orientation.

When you move the upper scrollbar to set the zero point, the system asks the wheel for its arc length, s. Then the value of the scrollbar is used to select a point in the interval $[0, s]$, and the zero point is moved there. So the first job is to compute the arc length of the wheel. I do this by summing up the arc lengths of each Bezier segment. These I find by a simple numerical evaluation. Bezier curves are parameterized by a single value that sweeps along their length. So I simply take many tiny steps and add up the length of each step.

Since my design window initially represents a unit square, most of my curves have a length of around 2 or 3 units. A little experimenting with these curves showed that 3,000 steps was way more than necessary to find an accurate arc length, but it was quick to compute and gave me lots of headroom for accurately computing the lengths of larger curves, so I left it there. Thus the parameter u along each Bezier runs from 0 to 1 in steps of 1/3,000. I save the length of each Bezier curve along with that segment. To find the arc length of the total curve, I ask each segment for its length and simply sum them up. I then save that total with the curve.

Any further requests for the curve's length, or the length of any Bezier segment, are easily served just by returning the saved value. If at any time I add or delete a Bezier segment, or move any of the control points, I invalidate all of the arc lengths associated with that curve, so the next request triggers a recomputation. That new result is saved again so further requests are once again nice and quick.

Armed with these arc lengths, I can find the point at arc length s by basically going the other way around. I step through the segments of the curve until I find the one that holds this value, and then I step through that segment—again in steps of 1/3,000—until I reach that arc length, or have a couple of steps that contain it. In the former case I simply return that point, and in the latter case I interpolate between the two points.

I also compute the tangent vector at the point at arc length s by finding the points at arc length $s - \delta$ and $s + \delta$, using a technique I'll describe later.

Each time I move the slider, I find the point corresponding to the arc length at that value of the slider, and save both the arc length and the position of the point. If the zero checkbox is turned on, I draw a little diamond at that point, and a short line to indicate the tangent vector.

The pale blue box just beneath the wheel controls holds an identical set for the frame. Beneath this is a purple box that controls the resolution of the roulette that's generated by the wheel and frame. To see what these numbers do, let's look at how the roulette gets generated.

To create the roulette, I march along the wheel's perimeter, rolling it along the frame. I find a point on the wheel, move it to the frame, rotate the wheel so that it's tangent to the frame, compute the location of the pen tip, and draw a little line to that point from the last pen tip. Then I do it again. Let's see this in more detail.

To start the process, I first find the arc length of the zero point on the wheel. For reasons that will become clear, let's call this arc length c_w. I can use this value to find the corresponding point on the wheel, $W(c_w)$. Of course, at the start of the process that's just the wheel's zero point.

I now want to find the tangent at this point. I find two points that surround the current point: $\mathbf{T}_0 = W(c_w - t)$ and $T_1 = W(c_w + t)$. The value of t tells me how far ahead and back to move along the curve. To match my arc length computation above, I use an initial value of 1/3,000. This gives me the vector $\mathbf{T}_w = T_1 - T_0$, which is the local approximation to the tangent. Just in case I'm at a part of the curve where the parameter is changing slowly, I compute the length of the vector $\mathbf{T}_w$. If it's smaller than some threshold (I use 4/3,000), then I double the value of t and compute T_0 and T_1 again. I repeat this until $\mathbf{T}_w$ becomes large enough, or I've repeated it 10 times (in practice, one or two loops is all that's usually needed). I normalize the vector $\mathbf{T}_w$ and save it with the wheel.

I then repeat the whole process for the frame. I find the arc length c_f for the zero point of the frame, and save the point $F(c_f)$. I compute the tangent $\mathbf{T}_f$ and save it.

Now I move the wheel by the vector $\mathbf{W}(c_w) - \mathbf{F}(c_f)$, so that the two curves come into contact at their respective points (the letter c in the arc length values c_w and c_f stands for contact). I compute the angles θ_w and θ_f corresponding to the tangent vectors for the two curves. I then find the smallest rotation that I can apply to the wheel's most recent orientation so that the tangent vectors are parallel. When we're starting, the wheel's current orientation is whatever I set it to by using the rotation scrollbar in the interface.

Now that I've got the wheel in the right position and orientation, it's time to locate the pen tip. Five numbers represent the pen tip: the Bezier curve number b, the first-knot number k, and three weights u, v, and w. Figure 16 illustrates the idea. I imagine a triangle starting from the center of the wheel (this is just the average of all the vertices in the Bezier polygon) to one edge of the Bezier polygon. The Bezier curve number b tells me which Bezier curve to use, and the first-knot number k tells me which knot forms one vertex of the triangle; the next knot forms the next vertex. So if $b = 2$ and $k = 1$, as in the figure, then I know the triangle is made up of the wheel's center (call that point C), and the second and third control points in the third Bezier curve (call these K_0 and K_1). I compute the center of this triangle, called point T, by averaging these three points together. Using these four points, I use the weights u, v, and w to compute the pen tip P this way:

$$P = T + u(T - C) + v(T - K_0) + w(T - K_1)$$

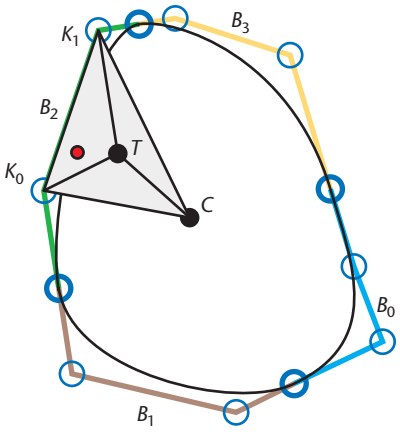
I let the weights u, v, and w range from about –5 to 5, which lets me move the point pretty far from the wheel if I want. Figure 17 shows the triangle used for determining the pen tip during the drawing process.

Once I have the pen tip, I check to see if this is the first point of the roulette. If it's not, I draw a line from the last pen tip location to the current one. Then I save this location for use in the next line.

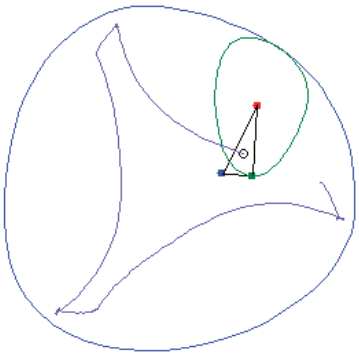
To generate the next point on the roulette, I need to first move the wheel. I retrieve the contact arc length c_w, and bump it to get the new contact arc length: $c'_w = cw + \Delta s$ (Δs is derived from one of the numbers you can set from the interface). From this, I get a new point on the wheel $W(c'_w)$, and of course a new tangent as well. I do the same thing with the frame, getting a new contact arc length c'_f, a new contact point $F(c'_f)$, and a new tangent.

It's worth a moment to consider an important subtlety: The increment along the two curves is the same because we're dealing with arc length, not the curve's intrinsic parameterization. Remember from our discussions of the geometry using Figures 9 and 10 that because the wheel doesn't slip, the arc lengths along the wheel and the frame at each step are the same. So it's important that the distance $|F(c'_f) - F(c_f)| = |W(c'_w) - W(c_w)|$.

Now that I have new contact points, I move the wheel so that $W(c'_w)$ sits on top of $F(c'_f)$, and then rotate it by the smallest angle that I can so that it's tangent to the frame. I locate the pen tip, draw a little line to there from the last location, and then repeat the whole process.



16 Geometry for finding the pen tip.



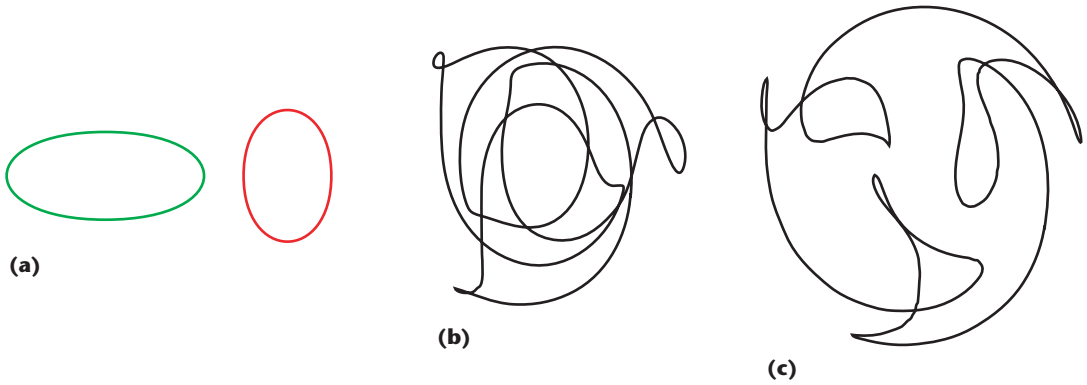
17 Showing the pen tip calculation in action.

Returning to the purple box in the control panel, the first number determines how many of these steps I should take to make the roulette. If the roulette is closed and matches up with itself, taking too many steps will simply cause it to repeat on top of itself. But generally with these oddball shapes the roulette doesn't close up, and so this becomes a matter of choosing an endpoint based on aesthetic and time considerations.

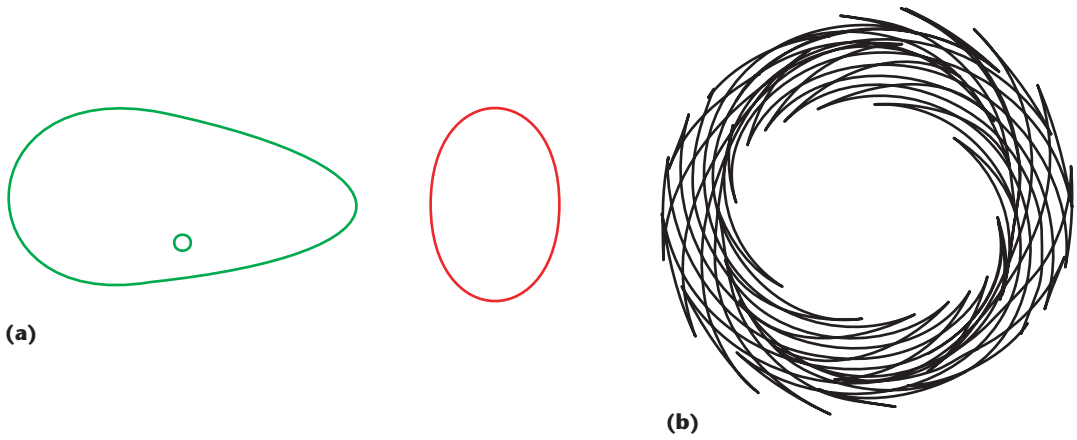
The other number controls the precision of each step. Recall that the arc lengths are bumped by Δs. The number in the second purple box sets this in multiples of 1/1,000. So if the number is 5, then the arc length is incremented by 5/1,000 on each step. As I mentioned, I start with a drawing space in the unit square, so curves tend to have lengths of around 2 or 3 units. A smaller number in this box means that the steps are smaller, resulting in more precision at a cost of slower drawing time and a bigger output file.

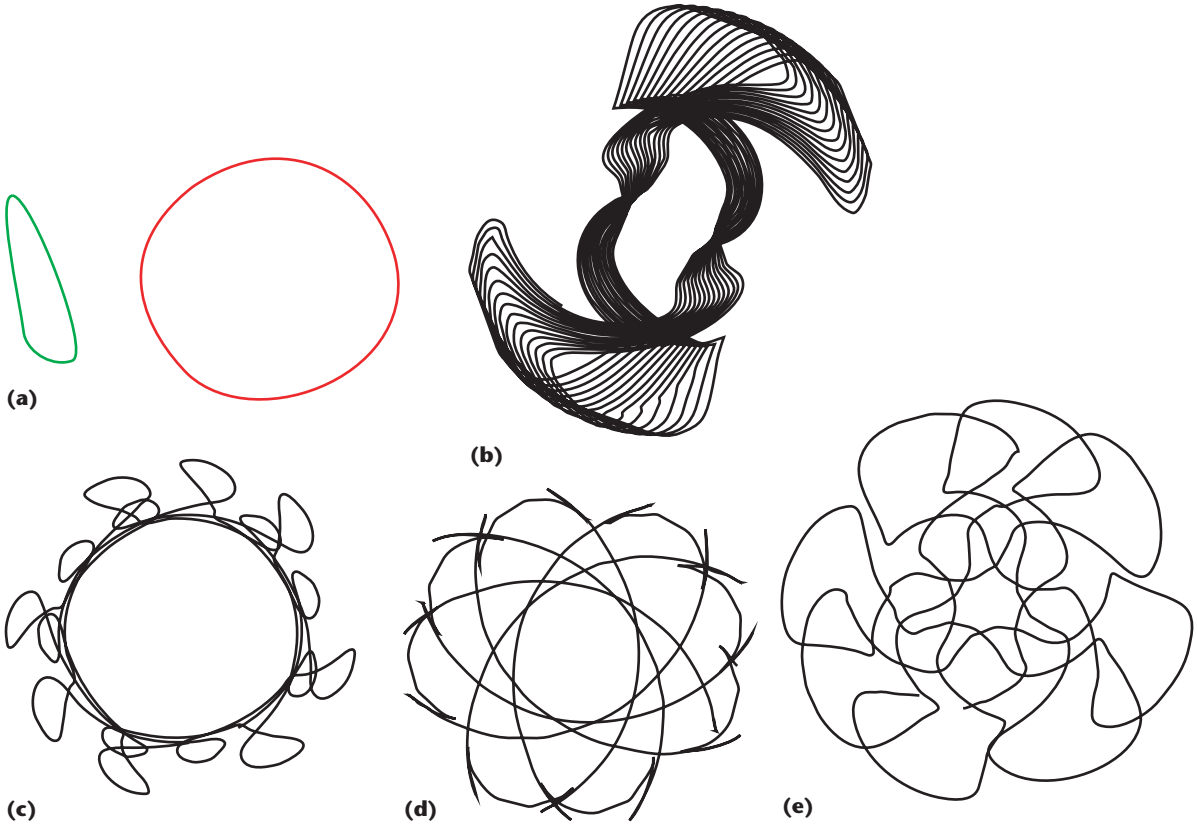
The orange box contains three checkboxes. The first toggles on and off the tangency constraint. Just for fun, I thought I'd give myself the option of turning off the rotation at each step that causes the wheel to become tangent to the frame. If you turn off this box, the wheel still moves around the frame, and the point of contact still moves around the wheel, but the wheel doesn't rotate. The other two boxes simply turn on and off the display of the transformed wheel as it moves, and the trail it leaves behind. Turning these off during the calculation, and then turning them on again when it's over, can save some time.

18 Generating roulettes with ellipses. (a) The wheel is in green, the frame in red. (b) A roulette generated by this pair. (c) Another roulette.



19 (a) The wheel is now an egg shape. (b) The roulette is quite different from Figure 18.





20 (a) The frame is roughly circular, but the wheel is a thin, bent egg. (b) and (c) A variety of roulettes generated by moving the pen tip, while the wheel rotates outside the frame. (d) and (e) The wheel is rotating inside the frame.

The next box down controls the pen tip's location. The two buttons select which triangle is being used as the reference for the tip. Pressing the button with a minus sign in it moves backward along the curve by one knot, backing up to a previous curve if necessary. Similarly, the button with a plus sign moves forward one knot. For example, pressing the minus button would move the pen tip in Figure 16 into the triangle defined by C, the knot labeled K_0, and the knot just below it. The three scrollbars let you set the value of the three weights that set the location of the pen tip.

The bottom set of buttons let me save and load files that contain the curve coordinates and all the control settings. I can also save the current wheel and frame and the computed roulette to PostScript, which I used to save the examples in the next section.

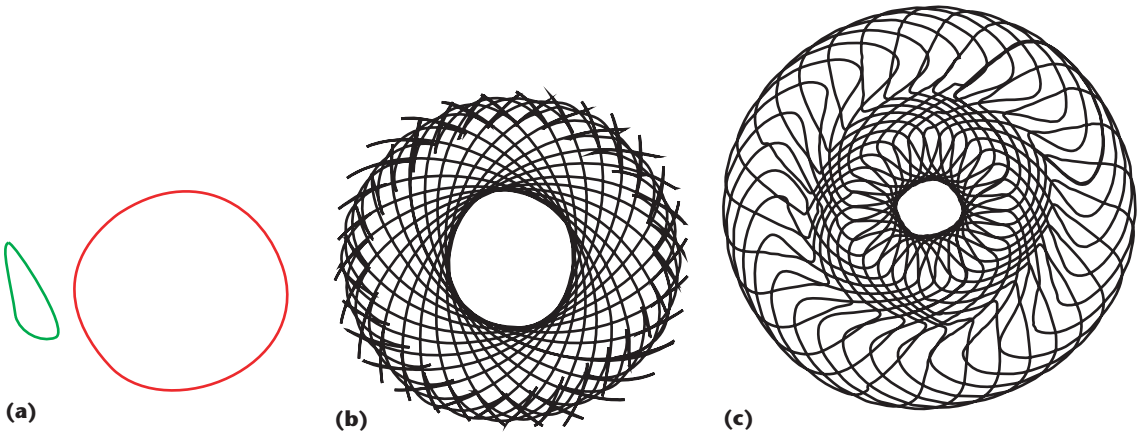
The buttons in the upper right are for debugging. The big green button on the right starts the roulette computation. The button in the lower right causes the system to pack up and quit.

Examples

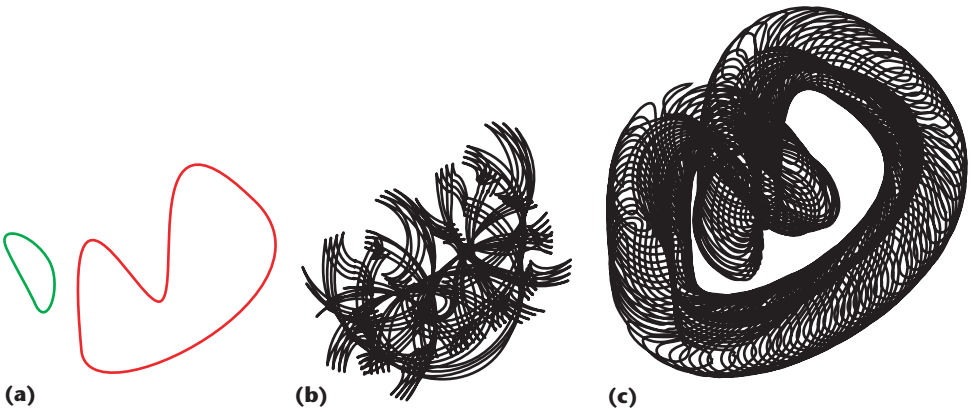
Let's look at some roulettes that I made with Bezier curves for the wheel and frame. Figures 18 through 25 show a variety of roulettes generated by a bunch of different shapes (Figures 23 through 25 are on the next page). I don't show the location of the pen tip with the shapes, because for many of the most interesting roulettes the pen was far away from the wheel, which would have meant leaving a lot of blank space on the page. If you're interested in reproducing these roulettes, fooling around with the shapes and the pen tip is enough fun to be worth the effort.

Because I calculate my roulettes by taking many small steps, the PostScript files that I generate tend to be huge. To prepare the figures for this column I opened my roulettes in Adobe Illustrator and used its built-in simplify command to reduce the number of points in the curve. This had the effect of eliminating some tiny jiggles in the curves that resulted from the finite precision of my calculations. These mostly occurred in places where both the wheel and the frame were nearly flat, and the approximated tangent vector wiggled by a very small angle from one contact point to the next. ■

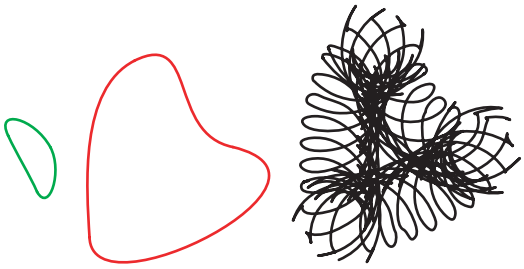
Readers may contact Andrew Glassner at andrew@glassner.com.



21 (a) The same frame as in Figure 20, but the wheel is more asymmetrical. (b) Rotating the wheel inside the frame. (c) Rotating the wheel outside the frame.

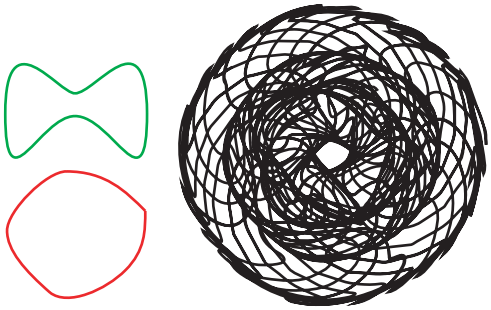


22 (a) A camel's-hump frame and a blunt wheel. (b) The wheel is inside the frame. (c) The wheel is outside the frame.



(a) (b)

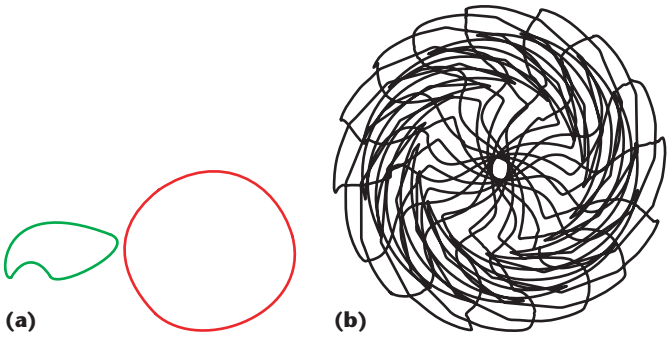
23 (a) The same wheel as in Figure 22, but a softened frame. (b) The roulette.



(a) (b)

24 (a) A bowtie-shaped wheel and a roundish frame. (b) The roulette.

25 (a) The same roundish frame as in Figure 20, but with a strange wheel. (b) The resulting roulette.



(a) (b)