ELLIPTICAL SHAPE COMPUTATION AND SHAPE MACHINE

Aryan Prodduturi*, Ethan Kahn†, Xavier Evans‡

How do we represent more complex designs for vector-based shape rewrite operations? What computational barriers exist for the shape grammar interaction of elliptical arcs? The paper discusses the expanded functionality of elliptical shape grammars within Shape Machine, a shape-rewrite computational system that features shape-based operations integrated with the CAD software Rhino. The vector parameterization and computational methods are briefly discussed to display the advancement of shape grammar technology used in Shape Machine.

INTRODUCTION

A shape grammar is a rule-based system for generating and transforming designs using geometric shapes instead of symbols. The concept was introduced by George Stiny and James Gips in the early 1970s as a way to formalize visual design processes.

In a shape grammar, a design begins with an initial shape and evolves through the repeated application of transformation rules. Each rule specifies how a particular configuration of shapes can be replaced or modified to produce new forms. By applying these rules iteratively, a wide variety of designs can be generated while still maintaining a consistent underlying style or structure.

Shape grammars are commonly used in fields such as architecture, industrial design, and computational design, where they help model design languages, analyze stylistic patterns, and automatically generate new design variations.

Through clever use of shape grammars, it is possible to solve a variety of mathematical and computational problems in a visual way, as opposed to the theoretical processes commonly used in those fields.

Shape Machine

Shape Machine is a software/framework that can apply shape grammar rules into documents. It applies a shape rule by replacing the left-hand side (LHS) shape into a new one on the right-hand side (RHS). The shape machine software is split into a few parts.

*Undergraduate Student, College of Computing, Georgia Institute of Technology, Atlanta, GA USA
†Undergraduate Student, College of Computing, Georgia Institute of Technology, Atlanta, GA, USA
‡Undergraduate Student, School of Mathematics, Georgia Institute of Technology, Atlanta, GA, USA

First, there is a backend. This backend can take esoteric inputs of shapes, lines, etc as well as a representation of the set of rules that should be applied. It will then output the new set of shapes, lines, etc that exist after the application of those rules.

Second, there are plugin layers built over the backend. Currently, the most functional of these plugins is the one for the Rhino architecture software. There are also plugins for a web interface and Grasshopper.

Our work mostly focuses on extending the backend part of Shape Machine. Prior to our extension, the backend could only apply rules to lines, circles, and circular arcs. We wanted to improve this functionality by also making it accommodate ellipses and elliptical arcs. We will simultaneously be deprecating the circular arcs from Shape Machine. This will not reduce any functionality because any circle or circular arc is also covered by a special case of the ellipse and ellipse arc.

This feature will be most useful for those in architecture or other artistic fields by allowing them to apply shape grammars to more of their existing work as well as removing the restriction of not using elliptical arcs that were placed on them by previous iterations of this software.

Carriers and Arcs

In the previous sections, we discussed full shapes (like circles and ellipses) in conjunction with their arc (circular arcs and elliptical arcs). This is one of the fundamental concepts of Shape Machine. Before applying shape grammars, we first calculate the carrier of each of the curves. This means finding the full circle/ellipse behind every arc, and even the infinite line behind every line segment. We can then use the intersections between these carriers to apply the rules.

Any shape is a subset of a carrier. For our circles and ellipses, that just means the start and stop angle of the arc. Whatever transformations and other manipulations that the rule says to apply with be applied to the actual shape and not the carrier, the carrier is only used to abstract the computation.

Shape Signatures

When we apply a shape rule, we will need to find instances of the rule's LHS inside the larger, more complex design. In Shape Machine, we have the option to choose how a shape may be transformed to match the LHS of the rule. This is a generally tedious process, due to the numerous possibilities a simple rule can be expressed under transformations such as rotations, translations, scaling, shear, and their compositions. Shape signatures act as the solution to improve the speed of this process.

As Figure 1 shows, a shape signature is created from the intersection points between all of the carriers in the shape. Additionally we will also add a point for the center any arc carrier in the shape as well. The shape signature functions as a hash for the a given shape. We will denote these points as registration points. A signature is simply a set of registration points constructed via the interactions of elements in the shape. Because it is

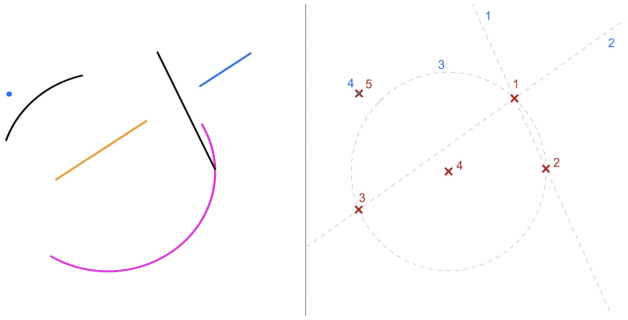

Figure 1: Current implementation of the shape signature system

a set of points, it is much easier to find transformations of those points that are subsets of others. Registration points have some accompanying information that we also need to check for matches.

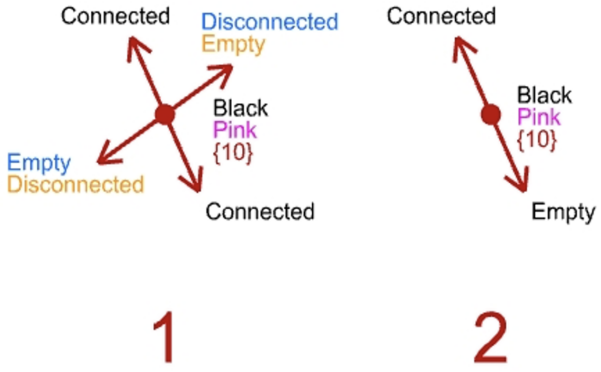

Figure 2: Current implementation of the registration point system

Registration points also store information as shown in Figure 2. In the current implementation these are spokes in the direction of each line carrier intersection and the radius of any corresponding arc carriers at an intersection. Finally we will add some extra attributes, shown by the colors of the lines being intersected, and whether the carrier intersection is filled or unfilled in regards to the shape embedded by the carrier.

Two signatures overlapping means that there is likely a way to transform the LHS to fit into the design but does not guarantee it. We can simply check if one signature is a subset of another. That is, we do a thorough check of the actual shapes under the transformation to verify the suggested transformation from before.

The shape signature and registration point system currently cannot accommodate ellipses in a reasonable way. The prior implementation can handle arcs but we cannot modify that previous system to handle conics because the system takes advantage of the the radial symmetry of a circle. This process is much faster than always transforming the real shape objects, but it needs significant changes for our needs.

Rhino Plugin

The main way to access Shape Machine is currently through Rhino. So, our update to the backend isn't fully complete until we also update the Rhino plugin to accommodate our new features. This has two main parts: changing how the plugin transmits the information about shapes to the Shape Machine backend and changing how the plugin uses the output from Shape Machine to edit the design in the file.

ELLIPTICAL TRANSFORMATIONS

Within Shape Machine, a generic Shape class is used in the code base with a few key methods for interactions between shapes. To implement complete functionality with ellipses, we have methods for construction, intersection, and transformation required.

Ellipse from Five Points

From the definition of generalized conics, all conic sections have five degrees of freedom. This is seen from the various expressions of ellipses:

- its center x and y, its major axis, its minor axis, and its rotation
- represented as the coefficients of

$$Ax^2 + Bxy + Cy^2 + Dx + Ey = 1 \tag{1}$$

This last method is the one we will utilize for our construction.

To create our representation of an ellipse, we use five points to fill in the constraints. It is not as simple as solving the linear model; there is inherent rounding due to floating point representations and possibly other sources of error. Our code accommodates that by adjusting for an error factor and can find ellipses even when the five given points are perturbed. Even so, the range of error that we accommodate is small since we want to keep Shape Machine as accurate as possible. Additionally, selecting false positives (i.e. thinking non-elliptical arcs are indeed elliptical) would significantly increase the computation needed and may also result in nonsensical values. For example, a slightly curved line could end up being recognized as part of an ellipse magnitudes larger than the size of the working document.

Given five points of the form $(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)(x_5, y_5)$, we know that if they fall in a conic, they will abide by (1). This can be represented as:

$$M = \begin{bmatrix} x_1^2 & x_1y_1 & y_1^2 & x_1 & y_1 \\ x_2^2 & x_2y_2 & y_2^2 & x_2 & y_2 \\ x_3^2 & x_3y_3 & y_3^2 & x_2 & y_3 \\ x_4^2 & x_4y_4 & y_4^2 & x_3 & y_4 \\ x_5^2 & x_5y_5 & y_5^2 & x_5 & y_5 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \\ D \\ E \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}.$$

With this system, we can solve it using standard linear algebra strategies. If no solution is found, we conclude early that our points are not in a conic form. However, knowing the set of points is conic is not sufficient: it could still be parabolic or hyperbolic. This property is decided by the sign of $B^2 - 4AC$, where a negative output corresponds with an elliptical (or circular) shape. From here, we use known properties of conics to solve for the center, the major axis, the minor axis, and the angle based on A, B, C, D, and E to create our representation of our conic.

Angle To find the rotation angle of the ellipse, we seek the angle θ that eliminates the xy cross-term when the conic is expressed in a rotated coordinate frame. Applying the substitution $x = x' \cos\theta - y' \sin\theta$, $y = x' \sin\theta + y' \cos\theta$ and requiring the coefficient of $x'y'$ to vanish yields the condition: $\cot(2\theta) = \frac{A-C}{B}$. Equivalently, this can be written as:

$$\theta = \frac{\pi}{2} + \arctan\left(\frac{C - A - \sqrt{B^2 + (A-C)^2}}{B}\right)$$

When $|B|$ is negligibly small (i.e., the axes are already aligned), the cross-term vanishes and the axes are aligned with the coordinate frame: $\theta = \frac{\pi}{2}$ if $A < C$, and $\theta = 0$ if $A > C$.

Center To find the center (x_0, y_0), we note that the center of a conic $Ax^2 + Bxy + Cy^2 + Dx + Ey = 1$ is the point at which the gradient of the left-hand side vanishes, giving the system:

$$\frac{\partial}{\partial x}(Ax^2 + Bxy + Cy^2 + Dx + Ey) = 2Ax + By + D = 0$$
$$\frac{\partial}{\partial y}(Ax^2 + Bxy + Cy^2 + Dx + Ey) = Bx + 2Cy + E = 0.$$

Solving this 2×2 linear system via Cramer's rule, with determinant $d = B^2 - 4AC < 0$ (from the fact that we ensured that it is a conic):

$$x_0 = \frac{2CD - BE}{d}$$
$$y_0 = \frac{2AE - BD}{d}.$$

Major & Minor Axis To find the semi-axis lengths, we translate the conic to the center (x_0, y_0) and write it in the form $\lambda_1 \hat{x}^2 + \lambda_2 \hat{y}^2 = 1$, where λ_1, λ_2 are the eigenvalues of the matrix

$$Q = \begin{bmatrix} A & B/2 \\ B/2 & C \end{bmatrix}.$$

The eigenvalues of Q are:

$$\lambda_{1,2} = \frac{(A+C) \pm \sqrt{B^2 + (A-C)^2}}{2}$$

After translating to the center, the constant on the right-hand side becomes $\hat{F} = 1 + x_0 D + y_0 E$, which simplifies to:

$$\begin{aligned} F' =& \frac{CD \cdot D + AE \cdot E + 2dF}{2d^2} \cdot 2d \\ =& \frac{x_0 D + y_0 E - 1}{d} \cdot d \end{aligned}$$

More concisely, letting $K = CD \cdot D + AE \cdot E + 2\delta F$ (where $F = -1$), the semi-axis lengths are:

$$a = -\sqrt{\frac{K(A + C - \sqrt{B^2 + (A-C)^2})}{d^2}},$$
$$b = -\sqrt{\frac{K(A + C + \sqrt{B^2 + (A-C)^2})}{d^2}}$$

where $a \geq b > 0$ are the semi-major and semi-minor axes respectively. The expressions under the square roots must be negative (which is guaranteed when $d < 0$ and the points form a valid ellipse); if either is non-negative, the configuration does not yield a real ellipse and is discarded.

Ellipse to Ellipse Intersections

The intersection between two ellipses is a complicated problem to solve. As ellipses can be layered on top of each other in different ways, we can have a multitude of different intersection scenarios. Below are some examples:

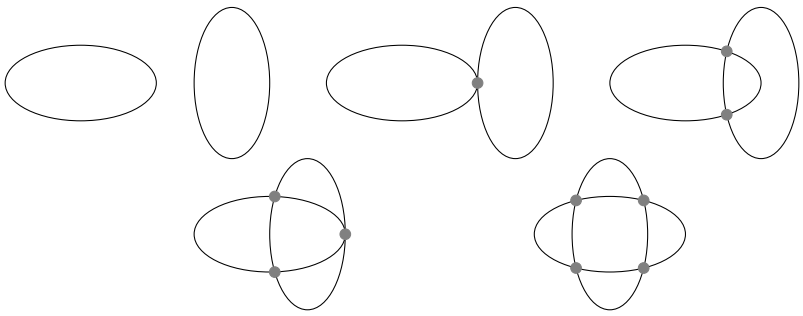

Figure 3: Various elliptical intersections patterns. From left to right: 0 intersection, 1 intersection, 2 intersections, 3 intersections, 4 intersections.

We note that the maximum number of intersections distinct ellipses can have is four. Since five points define a conic, having at least five shared points would imply they are the same. We are therefore motivated to create an equation with up to four solutions.

We proceed by defining our ellipses. Let our first ellipse have center (H_1, K_1), major axis a_1, minor axis b_1, and rotation A_1. Similarly, let our second ellipse have a similar schema: (H_2, K_2) center, major axis a_2, minor axis b_2, and rotation A_2. We can define our first ellipse parametrically as a function of θ:

$$x = H_1 + a_1 \cos(A_1) \cos(\theta) - b_1 \sin(A_1) \sin(\theta),$$
$$y = K_1 + a_1 \sin(A_1) \cos(\theta) + b_1 \cos(A_1) \sin(\theta).$$

This gives us expressions of trigonometric functions, which do not provide straightforward ways of solving for intersections. To remedy this, we will employ the Weierstrass substitution: let $t = \tan\left(\frac{\theta}{2}\right)$. This implies that $\cos(\theta) = \frac{1-t^2}{1+t^2}$ and $\sin(\theta) = \frac{2t}{1+t^2}$, which when substituted gives us

$$x = H_1 + a_1 \cos(A_1) \frac{1-t^2}{1+t^2} - b_1 \sin(A_1) \frac{2t}{1+t^2},$$
$$y = K_1 + a_1 \sin(A_1) \frac{1-t^2}{1+t^2} + b_1 \cos(A_1) \frac{2t}{1+t^2}.$$

We now have a way to represent our first ellipse using two rational functions of t. If we define our second ellipse in the form

$$\frac{((x-H_2)\cos(A_2) + (y-K_2)\sin(A_2))^2}{a_2^2} + \frac{((x-H_2)\sin(A_2) - (y-K_2)\cos(A_2))^2}{b_2^2} = 1,$$

we can substitute our definitions for x and y from the first ellipse to properly apply the constraints of both of our ellipses. While algebraically complex, this will reduce down to a quartic of t, which can be solved using numerous ways, including the quartic equation. The roots of this equation correspond to our intersection points, with imaginary roots and repeated roots implying missing intersections (meaning less than four).

Ellipse to Line Intersections

Solving intersections between ellipses and lines is much simpler than the algorithm previously explained. There can be only 0, 1 or 2 intersection points, and using a parametric equation for the lines, we can create a simple 2nd degree polynomial to solve for the positions of the intersections.

In the same vein as the previous algorithm, this is not an analytical solution, but rather an exact solution depending on the parameters of the ellipse and line.

ELLIPTICAL EXTENSION OF REGISTRATION POINTS

To adapt the preexisting registration points to the needs of elliptical arcs, we would have to find a solution that could securely keep most essential information of the particular ellipse used at the singular location. The carrier of the complete ellipse is unwieldy for quick transformations of indeterminate types, so another solution was needed.

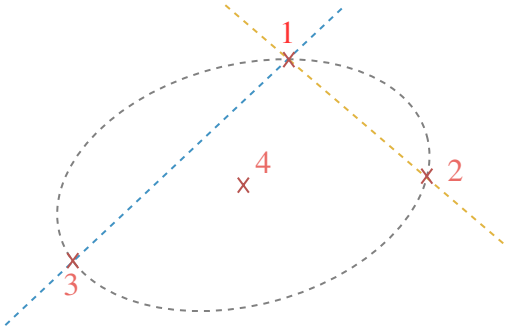

Figure 4: Signature of a composite shape

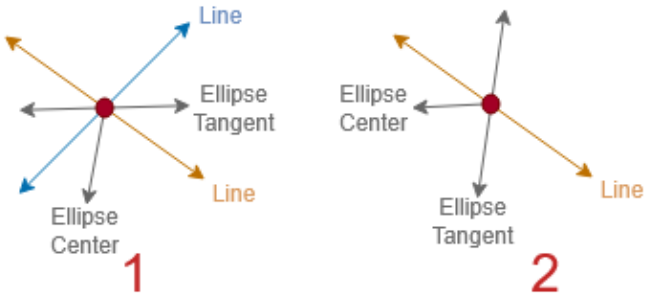

Figure 5: Detailed look at registration points one and two

As Figure 4 demonstrates, the algorithm for creating registration points is much the same, but using ellipses instead. The difference from the original version is show in Figure 5; our final implementation involved tracking the slopes of the tangent line and the radial line at a given registration point that happened to be on the surface of the ellipse. This particular solution projects the five dimensional ellipse space onto a four dimensional registration point space. This leaves one free degree of freedom but significantly improves computational speed. Because this new solution entirely works off of the spoke system used for registration points, there is no use for radius to be stored with any registration points; therefore, we have deprecated the use of radii in any registration point calculation. Each registration point has plenty of information that can be used to distinguish it from other registration points.

To calculate the partial slopes for an ellipse described by its major axis (a), minor axis (b), center (c_x, c_y), and rotation angle (θ); we can find the slopes (stored as a direction vector) of the tangent line and radial line to the registration point (p_x, p_y) on the perimeter of the ellipse.

First, we need the angle ϕ relative to the center of the ellipse, calculated as

$$\phi = 2\pi - \arctan\left(\frac{p_y - c_y}{p_x - c_x}\right).$$

Then with this, we could get the point (q_x, q_y) which is the location of the point (p_x, p_y)

if the ellipse were to be rotated and translated such that $\theta = 0$ and $(c_x, c_y) = (0, 0)$. This point is found with the following:

$$r = \frac{ab}{\sqrt{a^2 \sin^2(\theta + \phi) + b^2 \cos^2(\theta + \phi)}}$$
$$q_x = r \cos(\theta) \cos(\phi - \theta) - r \sin(\theta) \sin(\phi - \theta)$$
$$q_y = -r \sin(\theta) \cos(\phi - \theta) - r \cos(\theta) \sin(\phi - \theta).$$

We used r as the distance from the center of the ellipse to the point (p_x, p_y). This can give the slope s of the tangent point to the transformed ellipse to be

$$s = \frac{b^2 q_x}{a^2 q_y}$$

such that our final tangential slope can be represented as the direction vector

$$(\sin(\arctan(1/s) - \theta),\ \cos(\arctan(1/s) - \theta)).$$

Similarly, we have radial direction vector

$$(\cos(\phi), -\sin(\phi))$$

to connect from the center (c_x, c_y) to the point (p_x, p_y).

RHINO PLUGIN

Inputs

Currently the plugin assumes all selected arcs are circular arcs, but this should no longer be the case. When the plugin recognizes an arc, we will first need to query 5 points off that arc to test if the arc is coming from an ellipse. Then we can send it to Shape Machine as an elliptical arc in our design. While we can pick any 5 points, we pick 5 evenly spaced points in order to reduce the change of errors coming from rounding and floating point. After this step, the Shape Machine backend can take the inputs and work with them as expected.

Outputs

Drawing elliptical arcs in Rhino is a bit difficult. While there is good support for drawing full ellipses, the support in basic Rhino for drawing partial ellipse arcs is quite lacking. For this, we had to...

CONCLUSION

This paper presented the extension of Shape Machine to support elliptical arcs and ellipses. We introduced methods for ellipse construction from five points, ellipse-to-ellipse intersection via substitution and solving a quartic, and ellipse-to-line intersection via parametric reduction. Additionally, we described the adaptation of the registration point system

to accommodate elliptical arcs through tangent and radial direction vectors, replacing the radius-based approach that relied on circular symmetry. Finally, we outlined the updates we made to the Rhino plugin to transmit and receive elliptical arc data through the Shape Machine backend.

As a consequence of this work, circular arcs have been deprecated from Shape Machine without any reduction in expressive capability, as circles and circular arcs are consumed by the elliptical case. This simplifies the internal representation of Shape Machine while strictly expanding the set of designs it can operate on.

Several directions remain open for future work. The current intersection algorithms prioritize precision through closed-form solutions; future work could explore performance optimizations for designs with large numbers of elliptical elements. Additionally, the web and Grasshopper plugin layers, while outside the scope of this paper, would benefit from similar extensions to fully expose elliptical functionality across all interfaces of Shape Machine.

The ellipse class can be expanded to accommodate all conic curves, with much of the same code. This will allow the use of hyperbolas and parabolas in shape machine designs. Our new registration point and shape signature system already works well enough to handle these new kinds of curves without having to be overhauled again. The biggest work here would have to be done in Rhino or any other plugin that is interfacing with the backend because partial hyperbolic and parabolic arc support is likely even more lacking than elliptical arc support.