Circular Arcs as Primitives for Vector Textures

Zheng Qin, Craig Kaplan, and Michael McCool
University of Waterloo

Abstract. Because of the resolution independent nature of vector graphics images, it is useful to consider using them directly as textures in 3D rendering. Spline curves are used to represent boundaries in standard vector graphics representations. Antialiased rendering of such content can be obtained by soft thresholding an implicit representation of these boundary features. The distance function is an especially useful implicit representation, and an accurate distance can also be used for special effects such as embossing and stroke texturing. Unfortunately, computation of the true distance to a spline curve is expensive. Therefore, normally either the distance is approximated or the spline curves are approximated with simpler primitives for which an accurate distance can be computed. Approximation with line segments gives only C^0 continuity. We approximate spline curves using circular arcs instead. This approximation has C^1 continuity, and computing the distance to a circular arc is nearly equivalent in expense to computing the distance to a line segment.

1. Introduction

Vector graphics uses an explicit representation of region boundaries and strokes to represent images. Because of their resolution-independent nature and the advent of more programmable GPUs, the use of vector graphics image representations as a replacement for raster images in texture maps has become possible [Sen 04, Qin et al. 06, Nehab and Hoppe 07]. Vector graphics images have several advantages, particularly for image content where sharp boundaries are desired no matter how much textures are magnified.

In order to exploit standard tools for content creation, it is useful to design efficient mechanisms for rendering content expressed in a standard vector graphics format, such as SVG or PDF files. In order to support texture mapping the representation must also support efficient random access and antialiasing. Special effects such as drop shadows, embossing, and stroke textures would also be useful.

The distance to the closest boundary features can be used for antialiasing [Frisken et al. 00]. For anisotropic antialiasing, the gradient of this function is also needed [Qin et al. 06]. Close to curves, the distance function can be approximated by taking any other implicit representation and dividing by the magnitude of its gradient [Loop and Blinn 05, Nehab and Hoppe 07], but for many applications, such as embossing and stroke textures, the true distance function is required and is generally more robust.

The boundaries in standard vector graphics file formats may include points, line segments, elliptical arcs, and spline curves. Of these, spline curves are the most general. Computing the exact distance to a higher-order parametric spline directly is expensive. Even computing the distance to a quadratic spline requires finding the zeros of a cubic polynomial and a case analysis. To simplify this problem, we can approximate the curve with more primitive features for which it is possible to calculate a distance field easily and exactly. For example, line segments can be used to approximate spline curves, since is easy to compute the distance to a line segment [Qin et al. 06]. The disadvantage of line segment approximation is that it only has C^0 continuity. If a static approximation is used when the textures are magnified visible creases will be observed, reducing the resolution-independent advantage of vector textures.

In this paper, we present the use of circular arcs as an approximation primitive for vector texture representations. Computing the distance to a circular arc is nearly as inexpensive as computing the distance to a line segment, the storage increase is small (20% at most per primitive), and the resulting approximations have C^1 continuity.

2. Arc Splines

A curve represented with a sequence of arcs is called an *arc spline*. A simple procedure makes it possible to approximate any polynomial spline with an arc spline to any desired accuracy [Yang 02]. Start by sampling the polynomial spline finely using an adaptive parameter step. Then use two arcs joined with tangent continuity (a *biarc*) to approximate each segment. Note that as with a cubic spline segment, in a biarc the tangents of the endpoints can be chosen independently. If any segment cannot be approximated within a given error threshold, subdivide it. Once the whole spline is approximated with biarcs, apply a merging step to combine three arcs into two arcs whenever it the

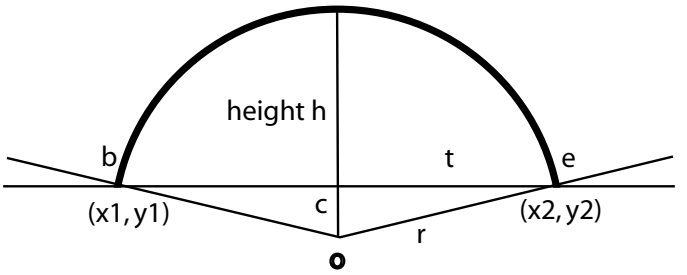

Figure 1. An arc feature can be parameterized by its endpoints and midpoint height relative to its chord.

result would satisfy the error bound.

This algorithm guarantees that the tangents at the end points of the approximation are exactly the same as those in the original segment. The tangents at the joints between segment will be consistent so the entire resulting curve has C^1 continuity if the original curve was at least C^1 continuous. Elliptical arcs (and, in fact, any kind of C^1 curve) can be approximated with arc splines similarly.

3. Feature Representations

Five parameters are required to represent circular arcs. We use four parameters to represent the coordinates of the two end points: $\mathbf{b} = (x_1, y_1)$ and $\mathbf{e} = (x_2, y_2)$. By representing endpoints explicitly, our representation is robust under quantization of these endpoints, since endpoints of different arcs will be quantized in exactly the same way. A fifth parameter is used to record h, the distance from the highest point on the arc to the chord formed by the two end points (we call it the *height*). See Figure 1.

Based on these five parameters, we can compute the center point (origin) of the circle containing the arc, and use this to compute the closest distance from any query point to the arc. Let c be the distance from the origin to the chord. Then by the Pythagorean Theorem and using the radius $r = h + c$ we have

$$(h+c)^2 = c^2 + t^2, \tag{1}$$

$$h^2 + 2hc + c^2 = c^2 + t^2, \tag{2}$$

$$2hc = t^2 - h^2. \tag{3}$$

where t is half the length of the chord, or half the distance between $\mathbf{b}$ and $\mathbf{e}$.

We can then solve for c, and use this value to find the origin point $\mathbf{o}$:

$$
\begin{aligned}
c &= \frac{t^2 - h^2}{2h}, && (4)\\
\vec{\mathbf{v}} &= \mathbf{e} - \mathbf{b} && (5)\\
&= (x_v, y_v), && (6)\\
\vec{\mathbf{w}} &= (y_v, -x_v)/2t, && (7)\\
\mathbf{o} &= \mathbf{b} + \vec{\mathbf{v}}/2 + c\vec{\mathbf{w}}. && (8)
\end{aligned}
$$

Within the wedge formed by $\mathbf{ob}$ and $\mathbf{oe}$, the distance to the arc from a query point $\mathbf{q}$ is then $d_i = |\|\mathbf{q} - \mathbf{o}\| - r|$ for an arc of radius r.

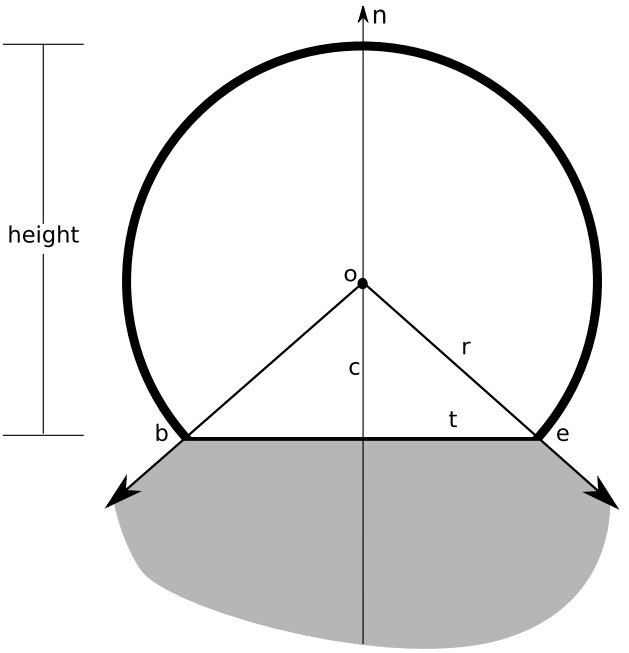

Figure 2. Arcs with large heights can also be computed by the above equations.

Outside the wedge bounded by the endpoints of the arc and the origin, we have to use the distance to the endpoints of the arc instead. We can test if the query point is inside/outside of the wedge using plane equations generated by the lines $(\mathbf{o}, \mathbf{b})$ and $(\mathbf{o}, \mathbf{e})$. The vector $\vec{\mathbf{w}}$ is perpendicular to the line $\mathbf{be}$ and when combined with the origin $\mathbf{o}$ generates the edge $\mathbf{on}$ which bisects the shaded region in Figure 2. To determine if a query point $\mathbf{q}$ falls into the shaded region and determine which end point is the closer one, we use the following tests:

// test if **q** *is in shaded area*
bool α = (right(**q**,**on**) && right(**q**,**oe**)
bool $\beta = \alpha$ || (left(**q**,**on**) && left(**q**,**ob**))
// select the closer endpoint
$\mathbf{d} = \alpha$? **e** : **b**,
d_o = distance(**d**,**q**)
// merge the inside/outside cases and compute the distance
$d = \beta$? d_o : d_i.

In Figure 2, when the height h is larger than the radius r. The value of c will be negative. In this case the origin **o** will be on the side of the chord in the opposite direction of $\vec{\mathbf{w}}$. Our equations above compute this case correctly.

Line segments can also be represented by the same parameters. As before, the first four parameters record the coordinates of the two end points. By checking if the height h equals zero, we can tell if a feature is a line segment or not. If a feature is a line segment, we need to compute the distance to it using an alternative code path, since the above computation would result in a division by zero.

Alternatively, the parameter h can be expressed relative to the length t so only the ratio h/t is stored, assuming t does not equal zero. This value can then be quantized over a finite range. If we limit h/t to the range $[-1, 1]$, then only arcs that sweep out a maximum angle of 180° can be stored, although they can be convex or concave. However, arcs that sweep out larger angles can always be broken into shorter ones with no loss in fidelity.

4. Results

We incorporated the arc representation into a system that imports SVG files, approximates them with arc splines, and creates a representation that allows their use as texture maps in real-time 3D rendering. In the shader that interprets the vector texture representation, we used control flow to separate the computations for line segments and arc splines. If the feature is line segment, the distance to a line segment is used, otherwise, the distance to an arc is used.

Our test system used an NVIDIA 8800 GTX GPU. With this GPU we found that for a vector image which contains both line segments and spline curves the control flow significantly slows down the rendering speed. However, when we approximated all line segments with shallow arcs, the speed was increased by 4% to 22%.

Remember when we used the same representation for line segments and arcs, the height was set to 0 for line segments. We cannot use the equations for arcs directly for distances to line segments, because equation (4) will have a

Figure 3. The lion with all the line segments approximated with shallow arcs.

division by 0. To solve this problem, use shallow arcs instead of line segments, setting the height to a small but non-zero number. In our experiments, if the ratio between the height and the length of the chord is 1/1000, no bending can be observed. In Figure 3, rendered by our system using this approximation, the input image was composed of line segments only.

In order to determine how much slower the arc spline distance computation was compared with the line segment distance computation, we applied both approaches to a number of examples (see Figure 4), setting up the line segment and biarc approximations to generate the same number of features. In general, the speed of arc spline computation was $70\% - 78\%$ of the speed of a line segment computation. However, it should be noted that often fewer arcs than line segments are needed to approximate curves to the same accuracy, although this depends on the input data.

References

[Frisken et al. 00] Sarah F. Frisken, Ronald N. Perry, Alyn P. Rockwood, and Thouis R. Jones. "Adaptively sampled distance fields: a general representation of shape for computer graphics." *SIGGRAPH*, pp. 249–254.

[Loop and Blinn 05] Charles Loop and Jim Blinn. "Resolution Independent Curve Rendering using Programmable Graphics Hardware." In *Proceedings of ACM SIGGRAPH 2005*, pp. 1000–1010, 2005.

[Nehab and Hoppe 07] Diego Nehab and Hugues Hoppe. "Texel Programs for Random-Access Antialiased Vector Graphics." In *Microsoft Research Technical Report MSR-TR-2007-95, July*, pp. 1–9, 2007.

[Qin et al. 06] Z. Qin, M. D. McCool, and C. S. Kaplan. "Real-Time Texture-Mapped Vector Glyphs." In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, edited by C. Séquin and M. Olano, pp. 125–132. ACM SIGGRAPH, 2006.

[Sen 04] Pradeep Sen. "Silhouette Maps for Improved Texture Magnification." In *Proc. Graphics Hardware*, pp. 65–74,147, 2004.

[Yang 02] X. Yang. "Efficient Circular Arc Interpolation Based on Active Tolerance Control." *Computer Aided Design* 23 (2002), 1037–1046.

Figure 4. Several examples. The apple image uses a linear gradient in the background.