Annales Mathematicae et Informaticae
48 (2018) pp. 83–91
http://ami.uni-eszterhazy.hu

Generating Distance Fields from Parametric Plane Curves

Gábor Valasek

Eötvös Loránd University
valasek@inf.elte.hu

Submitted March 5, 2018 — Accepted September 13, 2018

Abstract

Distance fields have been presented as a general representation for both curves and surfaces [4]. Using space partitioning, adaptive distance fields (ADF) found their way into various applications, such as real-time font rendering [5].

Computing approximate distance fields for implicit representations and mesh objects received much attention. Parametric curves and surfaces, however, are usually not part of the discussion directly. There are several algorithms that can be used for their conversion into distance fields. However, most of these are based converting parametric representations to piecewise linear approximations [7].

This paper presents two algorithms to directly compute distance fields from arbitrary parametric plane curves. One method is based on the rasterization of general parametric curves, followed by a distance propagation using fast marching. The second proposed algorithm uses the differential geometric properties of the curve to generate simple geometric proxies, segments of osculating circles, that are used to approximate the distance from the original curve.

Keywords: Computer Graphics, Distance Fields, Geometric Modeling

MSC: 65D17, 65D18, 68U07

1. Introduction

Distance fields are samples of the signed or unsigned distance function to a geometric object. They are a versatile representation of curves and surfaces and have been successfully applied in many areas of geometric modeling and computer graphics. Distance fields can simplify otherwise sophisticated operations, such as morphing between geometries with different topologies [3], and also speed up costly queries like collision detection.

Computation of distance fields from implicit representations has been extensively studied in the literature [7], but direct conversion of parametric curves or surfaces is less often exposed.

This paper presents two methods to convert parametric plane curves to sampled distance fields over a uniform grid.

The first method discretizes the parametric curve to grid cells. These cells are then used to form the boundary condition of the eikonal equation that defines the signed distance function of the parametric curve. Solving this equation using fast marching results in a highly efficient, approximate solution to the distance field.

The second method uses osculating circle segments to create a piecewise circular approximation to the original curve and extracts gridpoint distances using these circle segments as proxies for the original curve. This approach provides simple means to control the trade-off between accuracy and speed of execution.

These two methods are related in that they consist of a sampling and an approximation step. In rasterization, we sample the continouos problem first and then compute an approximate solution. In the osculating circle scenario, first we approximate the original curve with continuous proxies and then sample grid distances from these entities.

2. Rasterization-based distance field generation

One of the key components in real-time computer graphics is the high speed rasterization of geometric primitives. There are highly efficient algorithms to rasterize line segments [1], conic sections, and even higher order algebraic curves [6]. Nevertheless, research on the rasterization of general parametric curves is less prominent in the literature and they are usually discussed only in terms of lower degree polynomials, e.g. cubic Bezier curves [2].

The first method proposed in this paper to compute the distance field of a parametric curve consists of two steps:

1. Rasterize the input curve onto the distance field grid

2. Propagate distances over the grid from the rasterized cells

Step 1 requires a robust, gap-free, as in 8-connected, discretization of the input curve $\mathbf{r}(t) : [0, 1] \to \mathbb{E}^3$. This operation is inherently representation-aware, that is, it needs to know the particular basis and control data by which $\mathbf{r}(t)$ is represented.

For the sake of simplicity, we assume we are given a uniform grid, that is, a collection of rectangles of equal width and heigh. This way, the grid can be considered as collection of cells accessed by two dimensional integer indices, i.e. $Grid = [0, 1, .., W - 1] \times [0, 1, .., H - 1]$, where $W, H \in \mathbb{N}$ are the number of colums and rows, respectively. Rasterization should generate a gap-free subset $Rasters \subset Grid$.

Step 2 starts with the cells in *Rasters* and sets the distances to zero for these. Then it proceeds by using the fast marching method to compute the distance of every cell from *Rasters*. It does not rely on any prior information about the original representation of the curve, all it requires is the set of rasterized cells and the grid onto which the distances should be propagated.

2.1. Rasterization

Let $\mathbf{r}(t) : [0,1] \to \mathbb{E}^2$ be a regular parametric curve, i.e. $|\mathbf{r}'(t)| \neq 0, t \in [0,1]$. Let $D > 0$ denote the common width and height of all cells, measured in the units of the space in which the curve is embeded in.

The proposed rasterization method relies on a naive, curve traversal approach: starting from the $t = 0$ parameter, let us increase the parameter value by some $\Delta t > 0$ such that we travel approximately D units along the curve, i.e. the arc length between t and $t + \Delta t$ is $\approx D$. Evaluate the curve at this new point, find the closest grid cell, and if it is connected to the last cell in *Rasters*, add its index to it as well. If the connection is broken, we have to decrease Δt until the new point on the curve is rasterized onto a cell that is connected to *Rasters*.

To find the required $\Delta t > 0$ change of parameter, we have to compute the inverse of the arc-length function. The arc-length function is defined as $s(t) = \int_0^t |\mathbf{r}'(x)|dx : [0,1] \to [0,L]$ and since $\mathbf{r}(t)$ is regular, $s(t)$ is strictly increasing, i.e. it has an inverse denoted by $t(l) = s^{-1}(l) : [0,L] \to [0,1]$.

The n-th derivative of the arc-length function can be expressed in terms of the derivatives of the curve up to n. More precisely, in the case of the first three derivatives, we have

$$s'(t) = |\mathbf{r}'(t)| = (\mathbf{r}'(t) \cdot \mathbf{r}'(t))^{\frac{1}{2}}$$

$$s''(t) = \left((\mathbf{r}'(t) \cdot \mathbf{r}'(t))^{\frac{1}{2}}\right)' = \frac{\mathbf{r}'(t) \cdot \mathbf{r}''(t)}{|\mathbf{r}'(t)|}$$

$$s'''(t) = \left(\frac{\mathbf{r}'(t) \cdot \mathbf{r}''(t)}{|\mathbf{r}'(t)|}\right)' = \frac{\mathbf{r}''(t) \cdot \mathbf{r}''(t) + \mathbf{r}'(t) \cdot \mathbf{r}'''(t)}{|\mathbf{r}'(t)|} - \frac{(\mathbf{r}'(t) \cdot \mathbf{r}''(t))^2}{|\mathbf{r}'(t)|^3}$$

The above expressions can be simplified by using the Frenet coordinates of the derivatives: let $\mathbf{t}, \mathbf{n} : [0,1] \to \mathbb{R}^2$ denote the unit tangent and normal vectors of the curve and let $x_i(t) = \mathbf{r}^{(i)}(t) \cdot \mathbf{t}(t)$ and $y_i(t) = \mathbf{r}^{(i)}(t) \cdot \mathbf{n}(\mathrm{t})$. It can be easily shown that

$$s'(t) = x_1(t) \ , \quad s''(t) = x_2(t) \ , \quad s'''(t) = \frac{y_2^2(t)}{x_1(t)} + x_3(t) = \kappa^2(t)x_1(t)^3 + x_3(t) \ ,$$

where $\kappa(t)$ denotes the curvature function of the curve.

Let us now omit the parameter of evaluation from the formulas. Using that the derivative of the inverse function can be written as

$$t' = (s^{-1})' = \frac{1}{(s' \circ s^{-1})}$$

$$t'' = (s^{-1})'' = -\frac{s'' \circ s^{-1}}{(s' \circ s^{-1})^3}$$

$$t''' = (s^{-1})''' = -\frac{s''' \circ s^{-1} \cdot (s' \circ s^{-1}) + 3(s'' \circ s^{-1})^2}{(s' \circ s^{-1})^5}$$

the derivatives of the inverse of the arc-length function can be expressed as

$$t' = \frac{1}{x_1}\ ,\ t'' = -\frac{x_2}{x_1^3}\ ,\ t''' = -\frac{(\kappa^2 x_1^3 + x_3)x_1 - 3x_2^2}{x_1^5} = -\frac{y_2^2 + x_1 x_3 - 3x_2^2}{x_1^5}$$

that is

$$t(l + \Delta l) \approx t(l) + \Delta l \frac{1}{x_1} - \frac{\Delta l^2}{2}\frac{x_2}{x_1^3} - \frac{\Delta l^3}{6}\frac{y_2^2 + x_3 x_1 - 3x_2^2}{x_1^5} - \ldots$$

is the Taylor expansion of $t = s^{-1}$ about $l \in [0, L]$.

We can compute the required Δt change of parameter to advance approximately D units along the curve by evaluating the Taylor expansion with D

$$\Delta t = t(D) = D\frac{1}{x_1} - \frac{D^2}{2}\frac{x_2}{x_1^3} - \frac{D^3}{6}\frac{y_2^2 + x_3 x_1 - 3x_2^2}{x_1^5} - \ldots$$

Note that this means that this method is only rasterizing grid cells through which the curve passes through at least D units long consequitively sans approximation error. As a result, generally, we do not rasterize every cell that the curve touches, but the total arc-length of the curve within the omitted cells is small, provided the inverse arc-length approximation is precise enough and D is selected properly.

In practice, we have to keep track of whether the truncation introduced such an error that creates a gap during rasterization. This can be verified by comparing the integer grid coordinates of the previous and the new rasters. Should a gap occur, we can use a simple binary search between t and $t + \Delta t$ to find the parameter that corresponds to a curve point that rasterizes onto a grid cell that does not break the connectivity of the rasters or, better yet, until the arc-length between the two points is indeed closer to D. Using the latter as a stop criteria may or may not be possible, depending on our computational and time constraints.

2.2. Distance propogation

The second step of the algorithm can be formulated as an eikonal equation: the rasterized cells, denoted by *Rasters*, form the zero set of the unknown signed distance function $f(x, y)$ of the curve $\mathbf{r}(t)$, and we are looking for a solution over the entire *Grid* such that

$$\begin{aligned} |\nabla f(\mathbf{x})| &= 1 \text{ for } \mathbf{x} \in Grid \\ f(\mathbf{x}) &= 0 \text{ for } \mathbf{x} \in Rasters \end{aligned}$$

This can be solved on the grid using fast marching in $O(N \log N)$ time [8], where N denotes the number of grids, but there are linear solvers for the problem [9].

3. Osculating cirlce-based distance field generation

The rasterization based method proves to be an efficient sequential algorithm, however, its accuracy, as shown in the test results section, may be lacking for certain applications. The method proposed in this section aims to improve on accuracy and provides a user-controllable parameter to balance between speed and precision.

The method constitutes of the following steps:

1. Adaptively sample the curve and compute osculating circles at samples
2. Select segments of the osculating circles to approximate the curve
3. Compute cell distances from osculating circle segments

Step 1 and 2 do a pre-processing of the input curve. The goal is to create a collection of circular segments as the approximation of the original curve from the osculating curcles. The number of osculating circles can be set by the user with or without taking into account the particular geometric complexity of the input curve. The more osculating circles there are, the smaller the mean and maximum error becomes, but the execution time is also increasing.

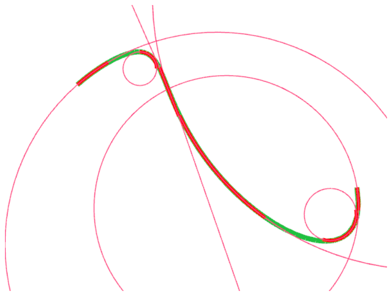
Step 3 speeds up distance computations by taking advantage of the piecewise circular structure of the curve approximant.

The following subsections discuss these three steps in detail.

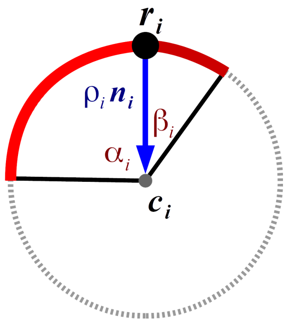
3.1. Sampling the curve

In order to adapt the number of segments to the particular curve in question, one has to find a measure of complexity for parametric curves. Initially, we used the curvature function plot to determine this complexity scalar by counting the number of local extrema of the curvature function. The user could select the number of osculating circles we sample between these extrema.

However, our tests have shown that for the case of distance field computation, an equidistant parameter sampling of the curve generates roughly the same quality distance approximation as sampling the curve adapted to the curvature extrema. As a result, we suggest the use of equidistant parameter sampling in time critical situations. Figure 1a illustrates the osculating circles obtained by this process.



(a) Osculating circles.



(b) Circular segment.

Figure 1: Osculating circles sampled along the curve (left). The representation of a single circular segment consists of $\alpha, \beta > 0$ angles, associatied with the osculating curve at sample $\mathbf{r}_i$ (right).

3.2. Determining the piecewise circular curve approximant

If all osculating circles are determined, we have to select segments on each, such that they put together form at least a globally G^0 approximation to the curve.

Let $t_0 = 0 < t_1 < t_2 < ... < t_N = 1$ denote the selected sampling parameters along the curve. Then $\mathbf{r}_i = \mathbf{r}(t_i)$ are the curve points and the unit tangent and normal vectors are $\mathbf{t}_i = [\mathbf{r}'(t_i)]_0$ and $\mathbf{n}_i = [(\mathbf{r}'(t_i) \times \mathbf{r}''(t_i)) \times \mathbf{r}'(t_i)]_0$, respectively, where $[\mathbf{a}]_0$ denotes normalization, i.e. $[\mathbf{a}]_0 = \frac{\mathbf{a}}{|\mathbf{a}|}$ if $\mathbf{a} \neq \mathbf{0}$.

Curvature at t_i is denoted by κ_i, their recipocals, i.e. the radii of the osculating circles are $\rho_i = \frac{1}{\kappa_i}$, and the centers of the osculating circles are $\mathbf{c}_i = \mathbf{r}_i + \rho_i \mathbf{n}_i$.

Since the circular segments have to include $\mathbf{r}_i$, we can define the segments used for approximation as two angles measured away from $\mathbf{n}_i$, the left angle α_i and the right angle β_i, see Figure 1b.

At the two endpoints we do not want to extend beyond the original curve, i.e. $\alpha_0 = 0$ and $\beta_N = 0$.

In case of a general point, consider the three osculating circles at t_{i-1}, t_i, t_{i+1}. If the $(i-1)$-th and i-th osculating circles intersect, let α_i be the angle that corresponds to the intersection point that is closer to $\mathbf{r}_i$ and still lies on the left of it, i.e. on the opposite direction of $\mathbf{t}_i$. If the circles are non-intersecting, or the intersection lies in the opposite direction, let us pick α_i such that it points to the point that is the closest to $\frac{\mathbf{r}_{i-1}+\mathbf{r}_i}{2}$. The β_i angle is determined similarly.

3.3. Distance computation

The reason for the gain in execution time lies in the fact that distance computation from a circular arc can be resolved by three point-point distance checks. We can use the distance of the point from the circular segment in the cone determined by $\mathbf{c}_i$ and the α_i, β_i spans, which is either $|\mathbf{x}-\mathbf{c}|-\rho_i$ or $|\mathbf{x}-\mathbf{c}|+\rho_i$, depending whether

$\mathbf{x} - \mathbf{c}$ forms and angle less than 90 degrees with $\mathbf{n}_i$. For points that are outside of this cone, we can check against the endpoints of the circular segments.

4. Test results

We benchmarked the execution time of both proposed algorithms and used a brute force, geometric Newton-Raphson based distance field as the ground truth for accuracy tests. A pure Python, sequential CPU implementation was used to compare the run times. After running a batch of 1000 random curves, we obtained the runtime and accuracy statistics for the distance field generation methods. The osculating circle based method used 7 osculating circles to approximate the curve.

	execution time (seconds)		error		
method	mean	std. dev.	max	mean	std. dev.
brute force	0.36	0.03	0	0	0
raster	**0.014**	**0.0029**	30.64%	21.53%	24.65%
osculating	0.16	0.036	**7.28%**	**0.46%**	**4.23%**

The errors are relative errors with respect to the distance values obtained from the brute force method. Both tests were conducted on 1000 random quntic Bézier curves and the distance transform was carried out on a 16×16 grid.

In a sequential environment, the rasterization based approach proved to be superior, however, the accuracy suffered greatly from the initial quantization of distance values, i.e. the setting of distances to zero for rasterized cells. This, however, is necessary for most fast marching algorithms.

The accuracy (measured as mean error) of the osculating circle based approach is two orders of magnitude better than that of the rasterization method.

Another focus of our tests was to determine how does the number of osculating circles correlate with the error statistics. The following table shows these as we increase the number of equidistantly sampled osculating circles. Interestingly, replacing the equidistant sampling strategy did not alter the error significantly, however, it is subject to future research to find computationally viable sample parameter selection methods that could increase distance field accuracy.

no. of circles	**mean error**	**std. dev.**
5	10.54%	88.86%
10	1.2%	11.95%
15	0.26%	1.79%
25	0.15%	1.4%
35	0.11%	1.08%

It is also important to note that creating a massively parallel implementation of the osculating circle based method is trivial, whereas the rasterization method suffers from the difficulties that arise when migrating fast marching to this setting, making it much harder to implement on e.g. GPUs.

Simple parallelization of the brute force method is also possible, however, one advantage of the osculating circle based method is that the actual distance transform part, where we measure the cell distances from the curve approximation, does not depend on the original representation of the curve, while the brute force algorithm has to rely on it throughout its execution.

5. Conclusion

This paper proposed two algorithms for the problem of direct conversion of parametric plane curves to distance fields. We compared these methods with each other and to a brute force distance field generation algorithm.

In sequential environments, the rasterization based method proved to be the fastest. However, if accuracy is required, the osculating circle based approach offers a superior solution. It relies on the prior knowledge of the representation of the curve, even though to a lesser extent than the brute force approach, which has to be taken into account when it comes to implementation.

References

[1] J. E. Bresenham: Algorithm for computer control of a digital plotter, IBM Systems Journal, Volume 4 Issue 1, March 1965, Pages 25-30

[2] S.-L. Chang, M. S. R. Rocchetti: Rendering cubic curves and surfaces with integer adaptive forward differencing, SIGGRAPH '89 Proceedings of the 16th annual conference on Computer graphics and interactive techniques, Pages 157 - 166, ISBN:0-89791-312-4

[3] D. Cohen-Or , D. Levin , A. Solomovici: Three-Dimensional Distance Field Metamorphosis, *ACM TRANSACTIONS ON GRAPHICS* 1998, Volume 17, Number 2, Pages 116–141

[4] Sarah F. Frisken, Ronald N. Perry, Alyn P. Rockwood, Thouis R. Jones: Adaptively sampled distance fields: a general representation of shape for computer graphics, *Proceeding SIGGRAPH '00 Proceedings of the 27th annual conference on Computer graphics and interactive techniques* 2000, Pages 249-254

[5] C. Green: Improved Alpha-Tested Magnification for Vector Textures and Special Effects, *Proceeding SIGGRAPH '07 ACM SIGGRAPH 2007 courses*, 2007, pages 9–18.

[6] J. D. Hobby: Rasterization of nonparametric curves, ACM Transactions on Graphics, Volume 9 Issue 3, July 1990, Pages 262-277

[7] Mark W. Jones, J. Andreas Baerentzen, Milos Sramek: 3D Distance Fields: A Survey of Techniques and Applications, *IEEE Transactions on Visualization and Computer Graphics* , Volume 12 Issue 4, July 2006, Page 581-599

[8] J.A. Sethian: A Fast Marching Level Set Method for Monotonically Advancing Fronts, Proc. Nat. Acad. Sci., 93, 4, pp.1591-1595, 1996.

[9] L. Yatziv, A. Bartesaghi, G. Sapiro: O(N) implementation of the fast marching algorithm, Journal of Computational Physics, Volume 212, Issue 2, 1 March 2006, Pages 393-399