AN ABSTRACT OF THE THESIS OF

Benjamin M. Weiss for the degree of Honors Baccalaureate of Science in Mechanical Engineering presented May 19, 2011. Title: A Graphical Method of Computing Offset Curves.

Abstract approved:

_____

Mike Bailey

Computation of offset curves is an operation critical to many computer-aided design and manufacturing (CAD/CAM) applications. Though simple on the surface, differences between the straightforward mathematical definition and the demands of CAD/CAM environment in the formulation and expression of an offset curve create a problem for which only complicated, approximate solutions are presently available. This thesis explores one of the newest methods of offset curve computation, using graphics hardware to directly compute the offset curve for arbitrary input geometry. Linear segments of the input curve are represented as meshes in 3D space, and the rendering process is used to create a field of depth values from which the offset curve is extracted as an isoline. This results in significant performance enhancements over previous, similar methods. Combined with a quantification of the errors involved in a graphical approach, these advances bring the technique closer to industrial readiness. Algorithm performance is shown to be linear with respect to geometric complexity of the input curve.

Key Words: Offset curve, General-purpose GPU programming, Loop elimination

Corresponding e-mail address: bweiss.ben@gmail.com

A Graphical Method of Computing Offset Curves

by

Benjamin M. Weiss

A PROJECT

submitted to

Oregon State University

University Honors College

in partial fulfillment of
the requirements for the
degree of

Honors Baccalaureate of Science in Mechanical Engineering

Presented May 19, 2011
Commencement June 2011

Honors Baccalaureate of Science in Mechanical Engineering project of Benjamin M. Weiss presented on May 19, 2011.

APPROVED:

_____

Mentor, representing Computer Science

_____

Committee Member, representing Mechanical Engineering

_____

Committee Member, representing Mechanical Engineering

_____

Dean, University Honors College

I understand that my project will become part of the permanent collection of Oregon State University, University Honors College. My signature below authorizes release of my project to any reader upon request.

_____

Benjamin M. Weiss, Author

ACKNOWLEDGEMENT

The author would like to gratefully acknowledge the incredible investment, encouragement, and support of my parents over the last twenty-three years. Without your patient nourishing of the gifts and talents God has given me, none of this would be. Additionally, my thanks go to Mike's consistent and fruitful mentorship in my life. The glory goes to God alone!

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

A Graphical Method of Computing Offset Curves

INTRODUCTION

Drawing offset curves is a task so visually intuitive that children have been doing it for millennia. Take a shape, then draw a line around it, a constant distance away from the original line. Repeat a few times, like in Figure 1, and the curve starts to look rounder and less sharp; continue still further and the shape slowly becomes a circle. The visual intuition present in six-year-olds can solve a problem that, though easy to formulate mathematically, has eluded a concise, clean algorithmic implementation for engineering applications.

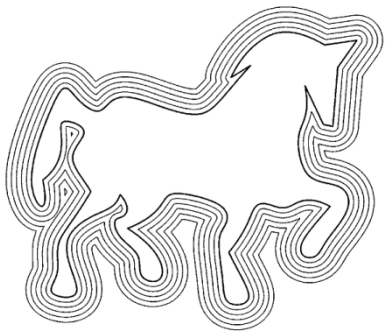


Figure 1. An example of several offset curves. (Lee, Kim, and Elber, 1998)

In the following pages, we explore a graphical technique for offsetting curves. By utilizing the ability of 3D graphics to trick our eyes into believing an illusion, we can effectively and quickly compute good approximations for offset curves. In order to do this, the problem must first be fully defined, then explored in the context of existing published solutions. Next, our approach is described in detail, including an assessment of the errors involved. The results and performance are presented, verified, and discussed, with an eye towards eliminating the remaining barriers that keep a graphical solution from being widely used in commercial and industrial offset curve approximation applications.

PROBLEM DESCRIPTION AND APPLICATIONS

Offset curve creation is a deceptively simple geometry problem. A straightforward mathematical definition for the construct exists, but the results are not useful to engineering applications. In this chapter, we will first explore the mathematical definition of exact offset curves, then present a series of examples from engineering applications which demonstrate both the utility of offset curves and the inadequacy of the purely mathematical definition, leading to the definition of the "global" offset curves described in subsequent sections.

An exact offset curve (or parallel curve) consists of the curve obtained when each point on the original (or progenitor) curve is moved a fixed distance along its normal (Figure 2). This definition is described mathematically for a regular parametric curve as:

Figure 2. Source curve (black) and an offset curve (gray).

$$\mathbf{C_d}(t) = \mathbf{C}(t) + d\widehat{\mathbf{N}}(t) \quad (1)$$

where *d* represents the offset distance and $\widehat{\mathbf{N}}(t)$ represents the unit $asdf$ normal vector, defined in the plane as:

$$\widehat{\mathbf{N}}(t) = \frac{y'(t)\hat{\mathbf{i}} - x\prime(t)\hat{\mathbf{j}}}{\sqrt{x'(t)^2 + y'(t)^2}} \quad (2)$$

where $x(t)$ and $y(t)$ are the *x* and *y* components of $\mathbf{C}(t)$ (Elber 1997). Mathematically, this presents no difficulties for regular parametric curves, where the regularity condition requires that $\widehat{\mathbf{N}}(t)$ be well-defined on the curve and nowhere is $\big(x'(t), y'(t)\big) = (0,0)$. In practice, complications arise because a polynomial $\mathbf{C}(t)$ gives rise to a parallel curve that is not in general

polynomial or even rational. Thus, an offset curve is significantly more mathematically complex than its progenitor curve.

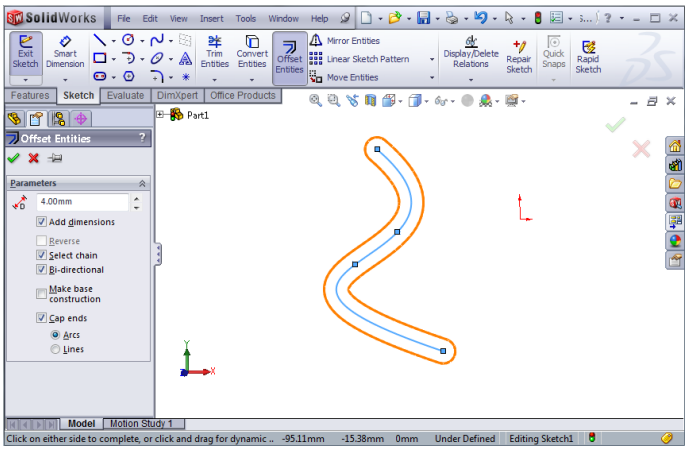


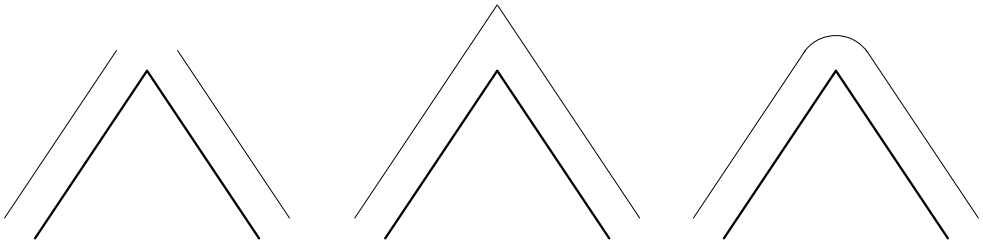
Figure 3. SolidWorks 2009 2D Offset function generates offset curves (the SolidWorks program and trademark are owned by Dassault systems)

The applications of offset curves in engineering practice point out several additional limitations of this definition. Applications in the mechanical engineering world, including CAD, CAM and rapid prototype manufacturing, require an offset curve definition not quite as mathematically straightforward as given above and motivate the definition of global offset curves.

In the world of computer-aided design (CAD), offset curves form a valuable tool when constructing 2D sketches; their three-dimensional cousins, offset surfaces, are frequently used in thickening and shelling operations. Figure 3 shows a form of offset curve that many CAD applications find valuable. Instead of a single output curve, the offset is bidirectional, meaning both the mathematical offset d and the complementary offset $-d$ are computed, and caps have been applied to the endpoints to close the curve.

In the more general case of a polyline curve, segments that begin or end without continuous first derivatives produce a cusp. When offset, the resulting curves do not intersect and a post-processing step must re-connect the results (Figure 4). Some methods, such as Liu, et al. (2007) do this by extending or trimming the lines, as in Figure 4b. While this is effective, it creates geometry in the offset that is more than d away from the progenitor curve, deviating significantly from the mathematical definition. An alternative is to add a circular arc of radius d,

centered at the intersection of the source curves, that smoothly connects the two exact offset curves. By doing this, we maintain the mathematical definition, with the direction of $\widehat{\mathbf{N}}(\boldsymbol{t})$ being ambiguous at the cusp point.



(a) ***(b)*** ***(c)***

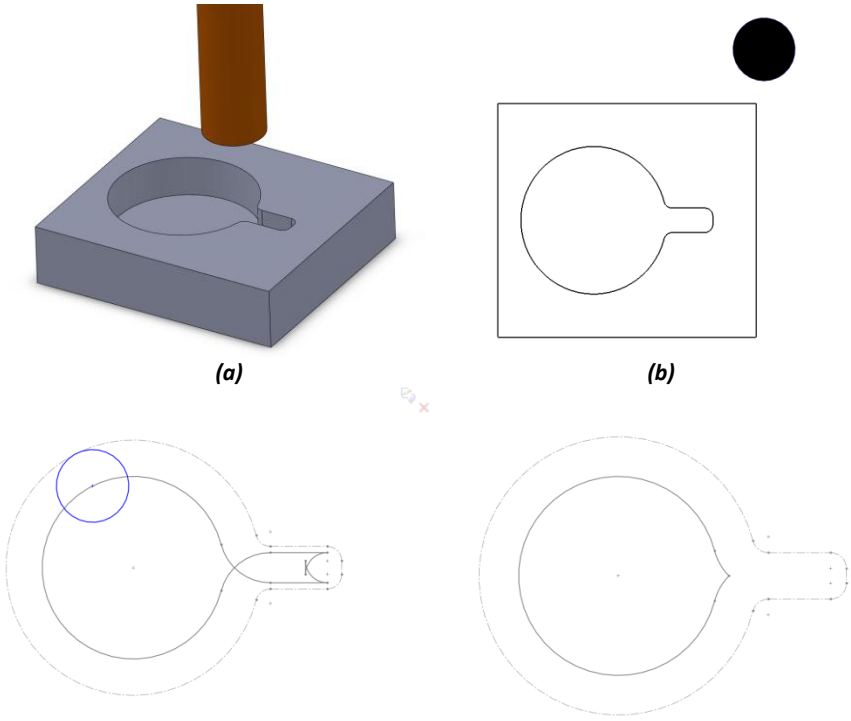
Figure 4. Two segments of a polyline produce a cusp. *(a)* Exact offset without any trimming. *(b)* Offset adjusted by extending source lines. *(c)* Offset adjusted by inserting an arc.

Computer-aided manufacturing (CAM), which converts CAD models into code that can be run on numerical control (NC) machines to fabricate parts, also finds applications for offset curves. In the case of an NC mill machining a pocket out of a block of material, the CAM software translates the desired solid geometry into a series of tool paths. The NC mill follows the tool paths to produce the desired part. In this case, the profile of the pocket defines the source curve, and the path the tool with cutter radius *d* must follow is the one-sided offset curve of the pocket (Figure 5).

Figure 5. NC milling machine cutting a pocket. The tool follows a path that is the offset of the desired pocket contour (Morrow, 2008).

This application points out another deviation of an offset curve from its mathematical definition. Consider a pocket which is to be roughed out by a cutter with a radius larger than the keyway, Figure 6a. The engineering intent is clearly to cut only those areas of the final pocket that can be accessed with the blunt, fast tool, then to return later with a smaller tool and cut the fine features. However, the mathematical offset of the pocket gives a curve that is clearly

undesirable (Figure 6c). Such a tool path would gouge other parts of the pocket during cutting, deforming the final geometry. Instead, the desired result is an offset curve that eliminates the loops caused by small features, as in Figure 6d.



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

Figure 6. A pocket machining example. *(a)* The pocket to be machined and roughing tool. *(b)* Plan view of the pocket. *(c)* The mathematically correct offset curve defining the tool path; cutter geometry is overlaid in blue. *(d)* The actual desired tool path.

The cause of the undesired result in the example above is present in parametric curves as well. In this case, a loop appears in the mathematical offset whenever the instantaneous radius of curvature of the curve is less than the offset distance (Figure 7). This phenomenon is referred to as a *local intersection*.

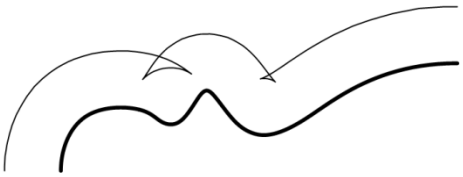
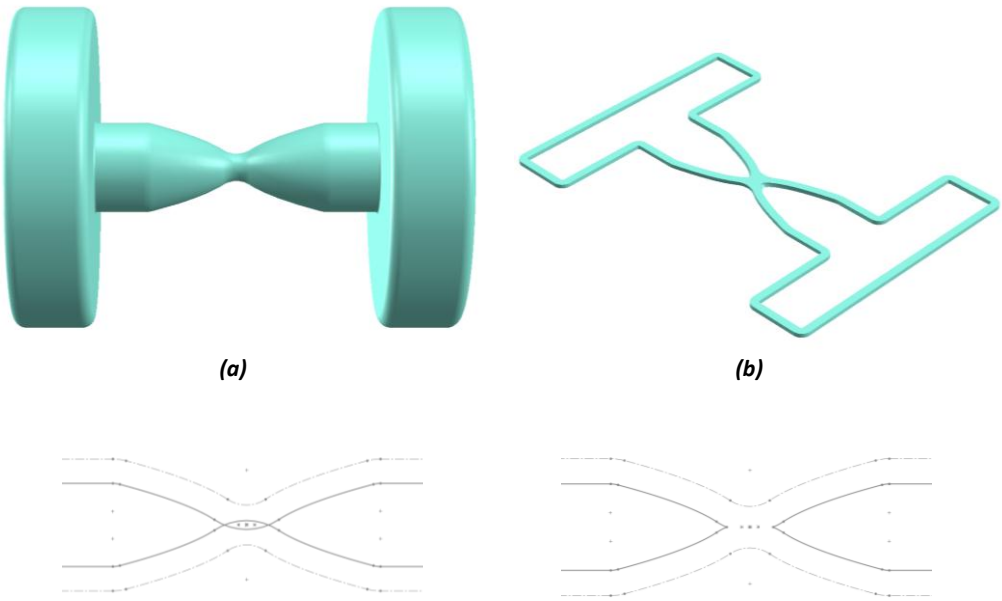


Figure 7. A cubic B-Spline with curvature less than the offset distance (dark curve) and mathematically correct offset (Elber 1997)

A third application highlights one additional deviation from the mathematical definition that most engineering applications of offset curves demand. In the process of rapid prototyping,

be it by lithography, deposition, or 3D printing techniques, layers of material are deposited or hardened one after another until the final part is formed. Because the material deposited is expensive and it takes time to harden/deposit it, it is common practice to hollow the model in the pre-processing step prior to fabrication, reducing the total volume of plastic used.



(a) ***(b)***

(c) ***(d)***

Figure 8. A deposition modeling example. *(a)* The model to be fabricated. *(b)* One slice of the model, hollowed to conserve plastic. *(c)* The hollowing operation, implemented as a mathematically correct offset curve of the profile. *(d)* The correct result.

Consider the model in Figure 8a. As described by Ganesan and Fadel (1994), in the process of fabricating it, the geometry is sliced, then the slices are hollowed with an offset curve algorithm. For the center section, the mathematical offset curve gives the result shown in Figure 8c. At best, this error is ignored by the shelling operation; at worst, an error occurs (SolidWorks 2009 refuses to directly shell a model like this). The desired output is instead what is shown in Figure 8d. As in the CAM example above, the result is a deviation from the purely mathematical formulation due to a visually obvious defect in the result, splitting the resulting curve in two and producing two cusp points. This kind of defect appears in curves whenever they intersect or get

within 2*d* of themselves. Unlike in local intersection, these situations, referred to as *global intersections*, resolving this error creates two offset curves as the result, significantly increasing the complexity of the output.

This technique is expanded upon by Park (2005) to create uniform-thickness geometry even in the presence of sloping walls. In doing so, he develops a theory similar to that described below, then reduces the problem back to a series of 2D offset computations, the specific algorithm for which he does not explore.

Engineering applications frequently need three modifications on the exact offset curve for their applications. Offset curves should be augmented to extend around the ends of the curve, such that cusps on polyline curves are handled in an intuitive manner. Loops produced in the offset curve due to areas of the source with curvature less than the offset distance need to be eliminated. Finally, intersections in the offset curve due to necking regions in the source model are to be removed, splitting the offset into two output curves. In the following chapter, we explore the ways this problem of generating these global offset curves has been solved in the literature.

PREVIOUS WORK

Offset Curves

The algebraic complexity of the mathematical offset, as well as the need for many additional tweaks and manipulations, makes directly implementing this approach impractical. A wide variety of alternative methods have been developed to approximate the offset curve while maintaining the desired engineering characteristics. Generally, this process is broken down into offset computation, which deals with defining new polynomial parametric functions that closely mimic the exact offset curve, and curve trimming, which eliminates the unwanted loops and splits the resulting curve appropriately for engineering applications. This section will provide a brief overview of the previous work in offset curve approximation using this approach, as well several methods for directly generating the global (fully-trimmed) offset curve directly.

Table 1. Selected offset computation methods compared.

Source	Input	Output	Method	Notes
Parametric Manipulation Methods				
Cobb, 1984	B-Spline	B-Spline	Shift the control knots of the B-Spline by the offset distance	Always under-estimates the offset
Coquillart, 1987	B-Spline	B-Spline	Shift the control knots based on normal of the closest point and curvature	Expands Cobb's method; exactly offsets linear and circular segments
Tiller and Hanson, 1984	B-Spline	B-Spline	Shift the edges of the control polygon by *d*	Remains one of the most efficient algorithms (control points/tolerance; Elber, 1997)
Elber and Cohen, 1991	B-Spline	B-Spline (more complex)	Based upon one of the above; locally estimates error and subdivides accordingly	Exactly resolves offsets of circles

Source	Input	Output	Method	Notes
Hoscheck and Wissel, 1988; Hoscheck, 1988	Spline	Spline of arbitrary degree	Use parameter transformations; nonlinear optimization to set control points on output curve	Least-squares optimization bridges parametric and interpolation approaches
Interpolation Methods				
Klass, 1983	Cubic Spline	Cubic Spline	Interpolate curvatures and tangents of the exact offset curve at the endpoints with a cubic Hermite spline	Stability issues
Pham, 1983	Arbitrary	B-Spline	Use curve fitting and sample offset points to approximate the offset	Restricted to the information contained in the sample points
Piegl and Tiller, 1998	Arbitrary	NURBS	Generate sample points based on curvature, interpolate to a NURBS curve of arbitrary order, then remove knots until tolerance is exceeded	Better guarantees of linearity than Pham, 1983
Other Methods				
Lee, Kim, and Elber, 1996	Para-metric	Para-metric (higher degree)	Convolve the source curve with a polynomial approximation of a circle, producing a new polynomial curve with order between two and five times greater	Significant increase in order of the output curve; error introduced from parameterization of a circle.
Liu, et al., 2006	Polyline	Lines and arcs	Offset each component, patching the results together again by extending or adding elements	Very common in commercial CAD/CAM packages; complex branching in algorithm required to cover all the cases. Produces a trimmed, global offset
Chaing, 1991	Arbitrary	Polyline curve	Map the input curve onto a 2D, fixed grid, then compute the distance field iteratively over the entire grid. Extract the offset curve by finding *d* in the grid	Requires large amounts of memory and computation time; must discretize original curve onto grid, adding error. Produces a trimmed, global offset

Source	Input	Output	Method	Notes
Kimmel and Bruckstein, 1993	Arbitrary	Polyline curve	Initialize a function φ on a 2D grid such that $\varphi(x,y,0) = 0$ along the input curve, then update it in the grid using techniques from computational fluid dynamics. Extract the result as an isoline of $\varphi(x,y,d) = 0$	Requires greater computation time; produces a trimmed, global offset

Traditional methods used to approximate offset curves fall roughly into two main categories, those that use the geometry and topology of the base curve to manipulate the control points of parametric curves (B-Splines are especially popular) to produce an offset approximation, and those that use sample points from the exact offset curve to feed an approximation method that fits a curve to the offset. Table 1 summarizes a few representative methods, for a more complete survey and analysis the reader is referred to Maekawa (1999), Elber, Lee, and Kim (1997), and Pham (1992). The 1980's saw a great deal of progress in both these areas. Parametric manipulation is most simply described by Cobb (1984), whose algorithm simply shifts the control knots of the source curve by *d* in a direction normal to the curve, as shown in Figure 9. This simplistic manipulation results in a consistent under-estimation of the offset, creating significant errors. Over the following few years, several papers addressed these shortcomings, with Tiller and Hanson's (1984) still relatively simple method remaining one of the most efficient in terms of total control knots needed to achieve an offset of a given tolerance in many situations (Elber, 1997).

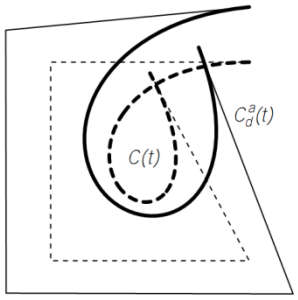


Figure 9. Cobb's method illustrated; dashed curve is the generator curve (Elber, 1997)

Interpolation methods, instead of relying on the control knots of the progenitor curve, sample two or more points on the actual offset curve (computed using Equation (1), then fits a parametric function to them to obtain the offset approximation. A simple case is found in the work of Klass (1983), which samples the source curve at the end points, obtaining location, slope, and curvature information of the offset curve, then uses this data to fit a cubic spline to estimate the actual offset. Error is again a problem, and over the succeeding years the method was refined to produce solutions that are both accurate to an arbitrary tolerance and efficient in the number of knots and curves produced. Elber concluded that least-squares-based interpolations of sample points in the offset curve in general provide the most efficient fit for a given tolerance.

Industry CAD/CAM systems have for many years simply broken down the input geometry into lines and arcs, offset each piece individually, and then patched the results back together again. Though not as general as methods that use arbitrary parametric or spline input curves, the majority of industry-driven design consists only of simple combinations of arcs and lines.

The fundamental shortfall in all of these approaches lies in their inability to handle local or global intersections and eliminate loops appropriately. The literature approaches this problem either by trimming the offset curves produced by one of the methods above or by seeking entirely different representations of the problem that do not give rise to the undesirable features. Newton's approximation allows intersections to be located to within a specified error tolerance, but requires good initial estimates of the intersection location. Beyond this, even local trimming methods are difficult to implement; just one offset curve can interfere with itself in any of seven distinct ways, meaning most algorithms must individually handle many branches (Wallner, 2001). Elber and Cohen's (1991) approach is perhaps the most elegant: local

intersections are detected by watching the sign of the tangent vector of the offset curve, consistently detecting most loops created through parametric manipulation. No attempt is made to handle global intersections, however. More mathematically rigorous approaches are taken by Maekawa, Cho and Patrikalakis (1997) and Seong, Elber, and Kim (2006), both of which correctly find intersection points and removing the undesired geometry at the cost of increased complexity.

Elimination of loops, both globally and locally, has been well-studied in the context of industrial applications, in which the input is restricted to lines and arcs. In general, trimming of these curves occurs either by use of a Voronoi diagram (Held, 1991; Lai, 2006) or some form of search algorithm (Hansen and Arbab, 1992; Choi and Park, 1999). Both work well enough for most straightforward engineering applications, but scale poorly with geometric complexity and require messy algorithms to implement.

As an alternative to these approaches, a variety of methods that forsake the intermediate step of an exact offset curve and attempt to directly generate the global offset have been proposed. Chaing, Hoffmann, and Lynch (1991) propose mapping the input curve to a finite, rectangular 2D grid, then propagating the distance function across the grid iteratively until the contents converge to the shortest distance from each point to the curve. By walking along the cells containing *d*, the offset curve is produced. In a related method, Gurbuz and Zeid (1995) estimated the offset by estimating the combined profile of a closed ball run along the curve's length, also in a discrete, ordered space. Kimmel and Bruckstein (1993) also use a discrete grid, instead borrowing tools from fluid dynamics to model the distance function as a steady-state property in the 2D domain, whose distribution is described by a differential equation such that the resulting property field describes the minimum distance to the progenitor curve. Walking an isoline along the resulting "level set" function extracts the offset. A

disadvantage of these methods, as well as the technique described below, while they automatically produce global offset curves, lies in the large amounts of data required to represent the resulting curve.

In summary, a great deal of work has been done in the area of offset curve approximation. Good estimates exist for offsetting arbitrary parametric input curves in any of several ways. Computing global offset curves which correctly handle both local and global offset intersections proves much more difficult, giving rise to several discrete approaches that directly compute the global offset. We note that a method for giving good initial estimates of intersection locations to a Newton's Method-type search tool would significantly ease the difficulty of the problem.

Graphical Approaches

The advent of the modern graphics processor, driven mainly by the demand of the consumer gaming industry, has created a new platform for computation and the solution of many non-gaming problems. The incredible power presented by the platform, with dedicated hardware designed specifically to perform highly-parallel functions on vertexes and pixels on their way to the screen, has long been recognized as an interesting new way to approach old problems. The rise of general-purpose graphics processor programming (GPGPU), spurred by the introduction of non-graphics interfaces such as NVIDIA's CUDA and Khronos Group's OpenCL, has moved from a sidelined science practiced by a select few in their basement to a mainstream

2D primitive	Shape of Distance Function
Point	Right circular cone
Line segment	"Tent"
Curve	Polyline of cones and tents
Polygon	Cones and tents

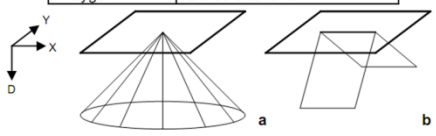


Figure 10. Voronoi Diagram primitives and associated distance meshes. (Figure modified from Hoff et al. 1999).

approach to problem solving, gaining traction in the corporate and research worlds.

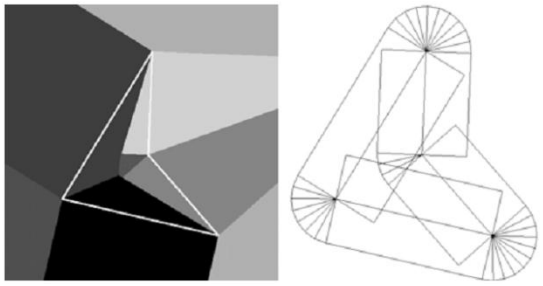


Figure 11. Voronoi diagram and its associated distance mesh (viewed along the +*D* axis), Hoff, et al. (1999)

Voronoi diagrams, used in loop-elimination approaches (above) as well as in many other environments, have taken advantage of this new approach. Hoff, et al. (1999) describes a "distance mesh" method for computing Voronoi diagram of a set of geometric primitives in 2D. Each component is represented by a simple 3D mesh such that the out of plane direction, labeled *D* in Figure 10, grows with the distance away from the primitive. Looking down the +*D* axis, two meshes will intersect when they are equidistant from their respective primitives, defining the new edge on the Voronoi diagram. By coloring the different meshes differently, we can obtain a diagram that estimates the precise Voronoi diagram to within an arbitrary precision (Figure 11), given appropriate selections of mesh density and screen resolution.

The graphics hardware and rendering process allow solutions like this to take advantage of the visual intuition that video games and other 3D applications require to trick our eyes into believing depth exists on the screen. To make a realistic 3D scene, the rasterizer must be able to rapidly interpolate quantities across the pixels covered by each of many millions of triangles, deceiving our eyes into thinking that bodies are single, solid objects rather than many small pixels. Additionally, the depth buffer allows only those objects closest to the viewer to be shown in the scene, catering to our physical intuition regarding occlusion. Hoff's work takes advantage of both of these hardware-implemented abilities to interpolate depth values across the polygons of the distance mesh as well as to occlude pixels associated with the parts of the mesh that fall behind (i.e. at greater distance from) the eye as others.

This visual intuition has been the basis for constructing offset curves by hand for centuries. The oldest reference to the subject found was an 1897 manuscript from Wilson's *Theoretical and Practical Graphics*, which describes the manual construction of a "parallel curve" using a compass. A series of arcs with radius *d* are constructed from successive points on the base curve. The offset curve lies tangent to all these arcs, and global and local loops are removed through visual intuition.

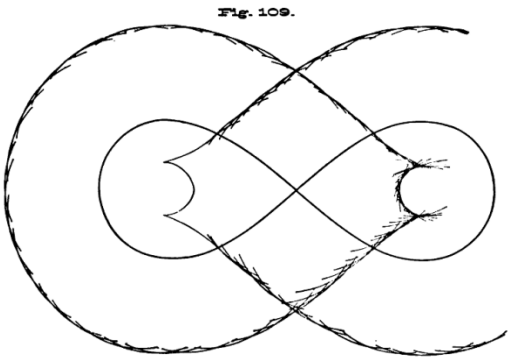
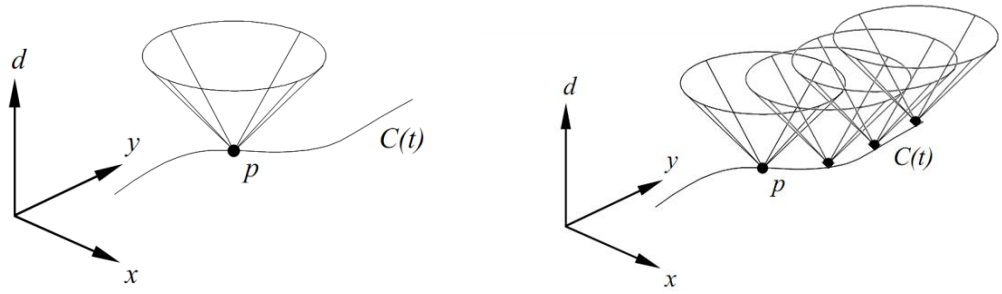


Figure 12. Parallel curve construction of a Lemniscate (light curve) with a compass and straight edge, from Wilson (1897).

This method of constructing offset curves was brought into the 21st Century and GP-GPU computing by a paper Li, Zhou, and Chan (2009), which describes a method of reformulating the problem using graphics hardware. As with Hoff, et al. (1999), a point in the plane is drawn as the tip of a cone that extends upwards into the distance direction, as shown in Figure 13.



(a) ***(b)***

Figure 13. *(a)* Distance function representation of a point, as described in Li, et al. (2009). *(b)* A curve represented as a sequence of points in *x-y-d* space. Figure modified from Li, et al. (2009).

Arbitrary input lines are discretized, and then a cone is drawn at each vertex. They adjust the distance of the offset by manipulating the front and back clipping planes, removing geometry with a depth greater than the offset, as in Figure 14. By changing the depth of the clipping plane, offset curves at varying distances from the camera can be produced.

Li, et al. go on to describe a method of offseting curves lying on an arbitrary surface by mapping the *u*-*v* coordinates of several sample offset curves on the mesh onto a new surface that exists in the *u*-*v*-*d* space, then interpolating the surface from the sampled curves. An offset is constructed by intersecting the plane with a plane horizontal to *u*-*v* at the height of the desired offset.

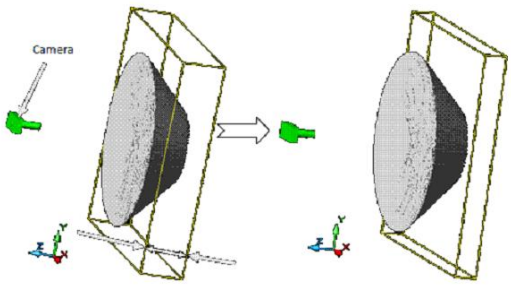


Figure 14. Figure from Li, Zhou, and Chan (2009) showing how varying the extents of the viewing volume (front and back clipping planes) changes the perceived offset distance.

The advantage of this approach is the automatic removal of all loops. In a manner similar to Chaing, Hoffmann, and Lynch (1991), it maps the curve and solution onto a discrete grid (the screen), and because the cones just overlap when the distance between any two line segments is less than 2*d*, no edge is extracted in that region and the loop is automatically removed. Adjusting the offset distance becomes trivially easy, not even requiring the re-formulation of the distance mesh, so generating families of offset curves is very straightforward. Because it utilizes the graphics pipeline for computation, the algorithm can be run in real time for most systems.

Disadvantages of this approach also exist, however. Because the resulting curve is a collection of closely-spaced points, it is less than ideal for engineering applications where data proliferation is an issue. If many offsets are present in a model, the amount of information required to fully define it grows very quikly to intractable levels (Lee, Kim and Elber, 1998). Additionally, no error metric is available for this method; in order to apply it to a real-world situation, a relationship must be established between error and computation parameters. Neither of these shortcomings are addressed by Li, et al..

So, although graphical approaches have been suggested for computing offset curves along with a variety of related problems, the current state of the field leaves significant gaps in our understanding of these methods. An opportunity exists to address some of the shortcomings in Li, et al. while applying some of the techniques developed in other similar GPGPU applications to this situation.

OUR APPROACH

The body of research on global offset curve approximations has grown quite extensive over the last three decades. In this work, we seek to extend the state of the art by implementing and exploring our own graphical offset approximation. This chapter describes the motivation for this project, continuing with a thorough description of the methodology employed to obtain our results. Finally, an error estimate will be presented for the approximation method.

Stemming ultimately from Wilson's 1897 work, we saw opportunities to combine the ideas of Hoff, et al. (1999) in their graphical approach to Voronoi diagrams, which utilize the mathematical intuition developed by tricking our eyes into believing two-dimensional objects have depth. The methods of Hoff, et al. and Li, et al. effectively use the rendering process to automatically detect and eliminate loops, we wished to extend their approach not only to eliminate loops but to increase the accuracy of the offset. In other CAM applications involving graphics processing, such as Inui and Ohta (2007), the depth buffer data is queried directly to obtain the cutting surface needed for surface machining. We believed the depth buffer, which modern hardware allows to be both large and high-precision, could be queried in our application as well.

The data from the buffer could be processed in a manner similar to the method proposed by Kimmel and Bruckstein (1993), which traces an isoline through a level set function evaluated over a rectangular grid to obtain the offset curve, using the depth buffer as the height field to be traced. The height field is effectively a discrete representation of an implicit parametric equation that represents the shortest distance between each vertex and the line segment. Extracting data from such a field is very similar to calculating the set of points needed to define a level set function in a more traditional sense. The difference lies in how the field of

points is generated: instead of relying on mathematics or iterative approaches like Kimmel and Buckstein, we use the graphics hardware.

Although developed independently, the method we developed is similar to that of Li, et al. (2009), with several important exceptions. First, our use of depth buffer data to interpolate the resulting curve allows much higher accuracies in the output than their approach. Secondly, we extend the concept by adding "tents" similar to Hoff, et al. (1997; Figure 10) between the points along the curve. Finally, we present an analysis of the error sources involved in the offsetting operation.

Any graphical offset algorithm can be cleanly broken down into three steps: preparation, rendering, and data extraction. The following sections will define and explore each of these steps as implemented in our method. This chapter will conclude with discussions on the error sources present in this method and approaches used to mitigate those errors.

Preparation

The preparation stage of the offsetting process revolves mainly around importing and discretizing the geometry. Discretization of the geometry to be processed into finite line segments is a critical aspect of the approach; error is inevitably introduced in this step. One of the beauties of graphical approximation, however, is that error-bounded discretization schemes exist which can display almost any kind of geometry. It doesn't matter if the input is a series of lines, a spline curve, or even an implicit mathematical construct, if it's possible to find a set of points that describe the geometry to an arbitrary precision, it can be used. This is in contrast to most discretization schemes, which are designed to work primarily with one kind of input curve. As mentioned above, commercial CAM systems frequently handle only lines and circular arcs,

converting other geometry into this representation before processing. Spline approximation methods such as Tiller and Hansen (1984) require the input to be a parametric plane curve, requiring other approximation methods to be utilized when different curve types are encountered or converting other geometry into a spline approximation. For simplicity, our test application is designed to offset only a single cubic Bézier curve.

Converting all input geometry into linear segments tends to balloon the amount of information required to represent a shape. It is important, therefore, that the algorithm be able to handle very large input datasets without significant computational cost. Because the offsetting and loop elimination operations are implemented as graphics rendering problems, the entire process occurs in linear time. Thanks to the gaming industry, the graphics pipeline routinely accepts hundreds of thousands of polygons per frame, making it robust for most industry problems, especially if real time performance is not required.

Rendering

The rendering step of the process entails drawing the geometry to the screen. The viewing volume is configured to be orthographic, viewing down the –*z* axis, and defined to be large enough in model coordinates to view the entire expected offset. The viewing volume should be just slightly larger than the expected offset curve; any larger and unnecessary time is expended analyzing irrelevant pixels. Determining the bounding box of most reasonable 2D geometry is straightforward, and expanding this bounding box by a bit more than the offset distance provides a good guarantee that the result will optimally fill the viewing volume. In the *z* direction, the viewing volume is restricted to a region one unit high, spanning the interval [-0.5, 0.5].

Geometry is placed into the scene in the form of a cone at every vertex. As in Hoff, et al. (1999), the cone's vertex lies coincident with the source curve in the *x*-*y* direction, with sides that slope down at a constant angle. The cone is drawn with its vertex at $z = 0.5$ and with a height of 1.0, and a base radius of 2.0. The cone is then scaled by 2*d* in all three dimensions in the model transformation matrix (Figure 15 a, b). In this way, were the image sliced by a plane at $z = 0$, the union of the intersected curves would give an approximation of the offset of distance *d* (Figure 15c).

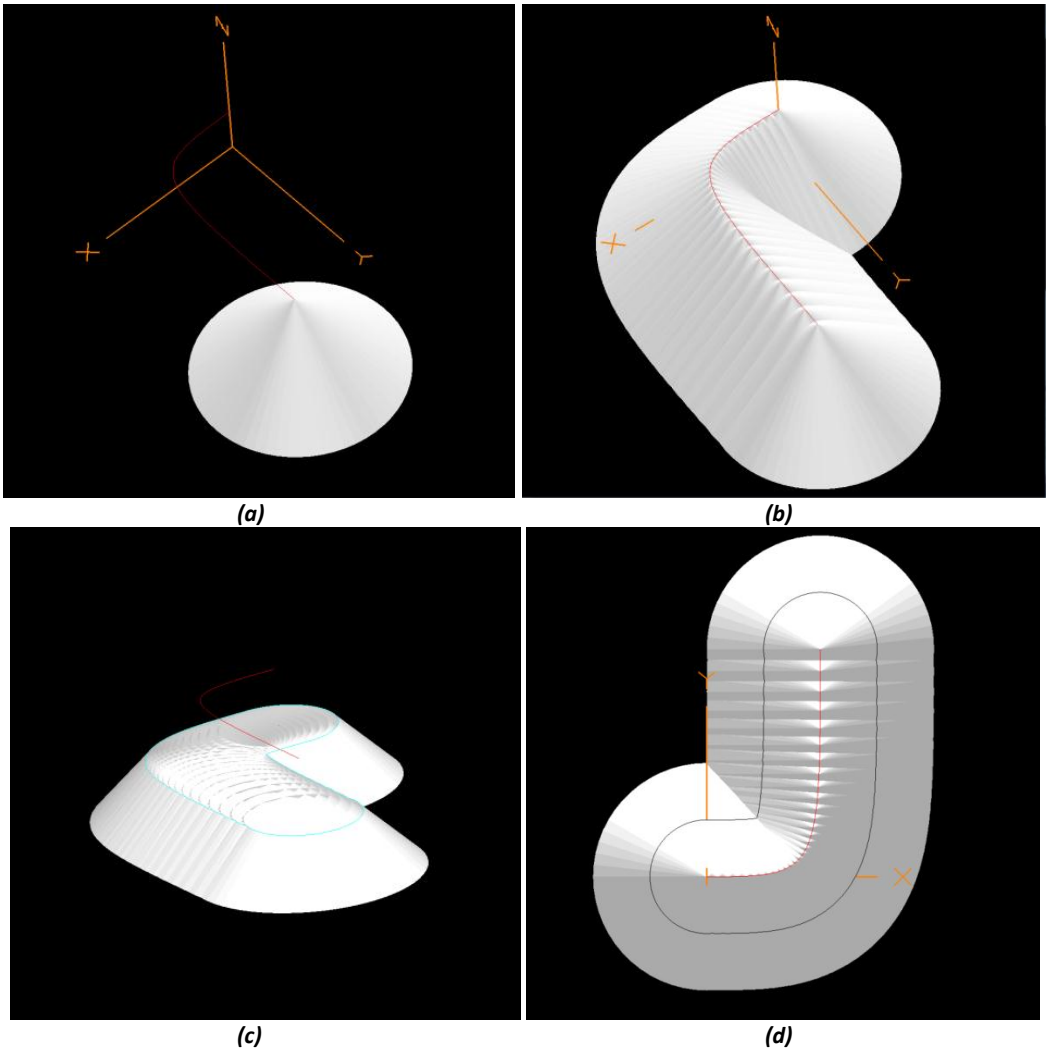
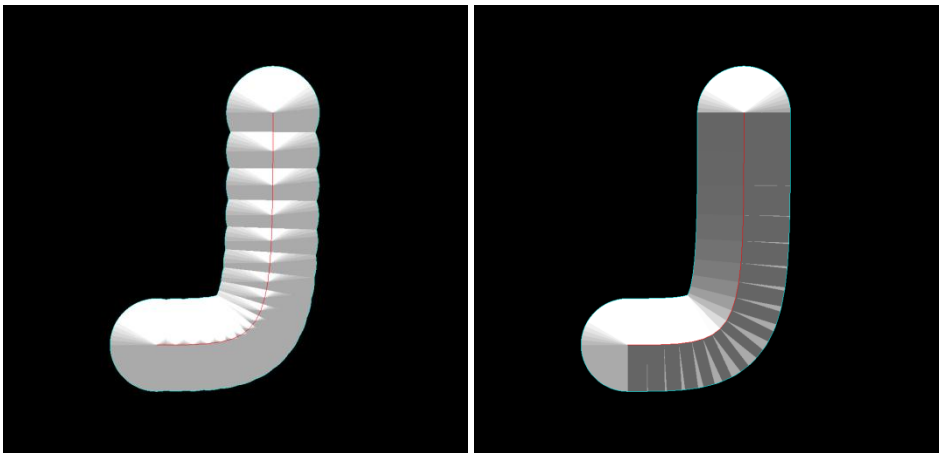


Figure 15. ***(a)*** **A single cone drawn on the input curve. It has a height and base diameter of 2*d*, extending *d* above and below the *xy* plane.** ***(b)*** **A series of cones drawn on the input curve.** ***(c)*** **The cones cut at *z*=0.** ***(d)*** **The extracted offset viewed from the top.**

One significant drawback of this method can be seen in Figure 16. When the input curve is not discretized with sufficiently small granularity, the cones provide a poor estimate of the offset in the valleys between them, creating a significant error. To combat this, we again follow the lead of Hoff, et al. (1999) by implementing "tents" between the discrete vertices. In this way, the method of Li, et al. (2009) is fundamentally extended. Instead of estimating the offset by creating the offset of a series of points, we estimate the offset of a series of straight lines, allowing straight segments to be drawn with far fewer polygons and providing significant advantages in the error analysis presented below. A schematic of the change and its consequences in output geometry is presented in Figure 17.



(a) ***(b)***

Figure 16. A graphical offset drawn *(a)* just with cones and *(b)* with cones and tents.

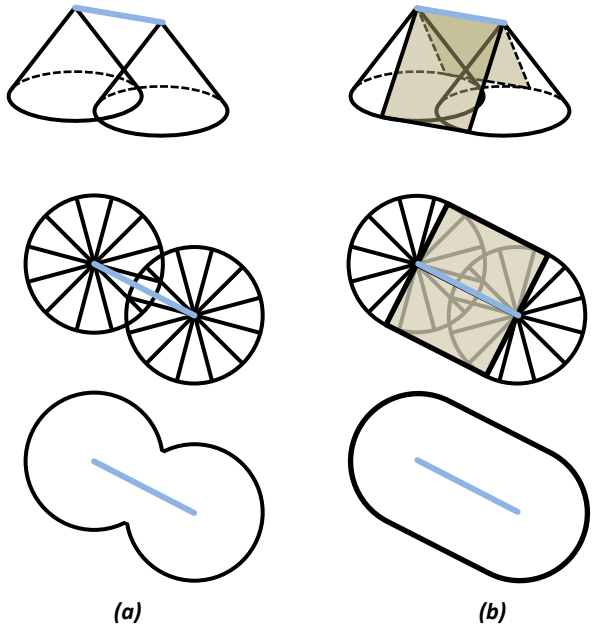


Figure 17. A schematic of the geometry rendered and results obtained for a single line segment *(a)* without tents and *(b)* with tents.

Data Extraction

The third, and most difficult, step of the offsetting process requires taking the height field produced by the rendering process above and extract the offset curve. To do this, the data is first retrieved from the framebuffer as an array of 32-bit float depth buffer values, scaled between the near and far clipping planes. The desired offset surface is the set of points for which depth is equal to zero, and extracting a set of points that lie along this transition can be achieved through the use of a marching squares isoline generation algorithm. This widely-used approach is the two-dimensional recasting of the marching cubes algorithm. The subsequent paragraphs give a brief overview of this process.

Extracting the resulting geometry in this way, we effectively use the depth buffer as the discrete representation of an implicit function for distance from the source curve. Extracting a level set out of this implicit function creates the offset curve. Unlike other level set offset

approaches, however, the level set function comes from the rendering pipeline instead of continuous mathematical manipulations of the source geometry.

In this approach, the height field is broken into squares, with known values at each corner. For each squares, edges are evaluated to detect sign changes (i.e. does an isoline of height 0 pass through this edge?). Whenever a sign change is detected, linear interpolation between the vertices determines the location of a point on the isoline, see Figure 18a.

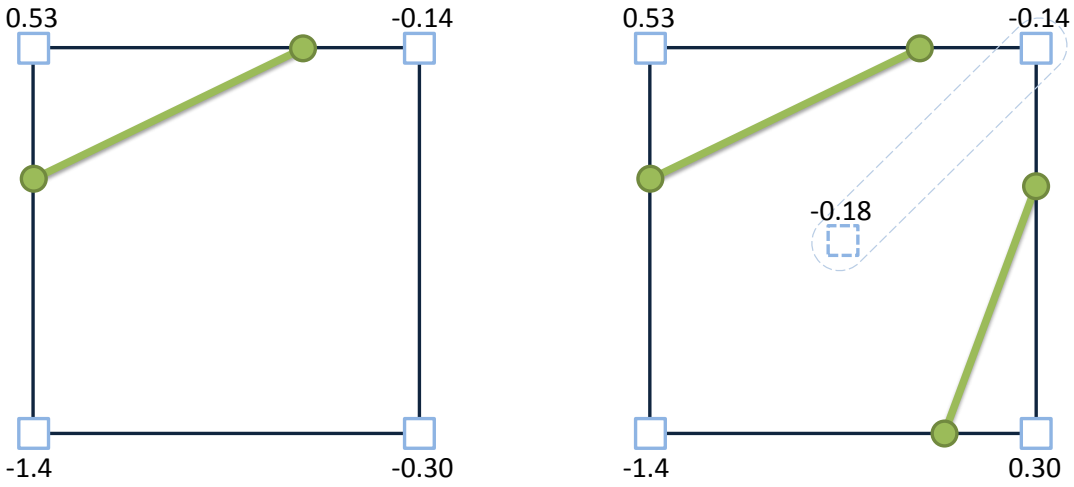


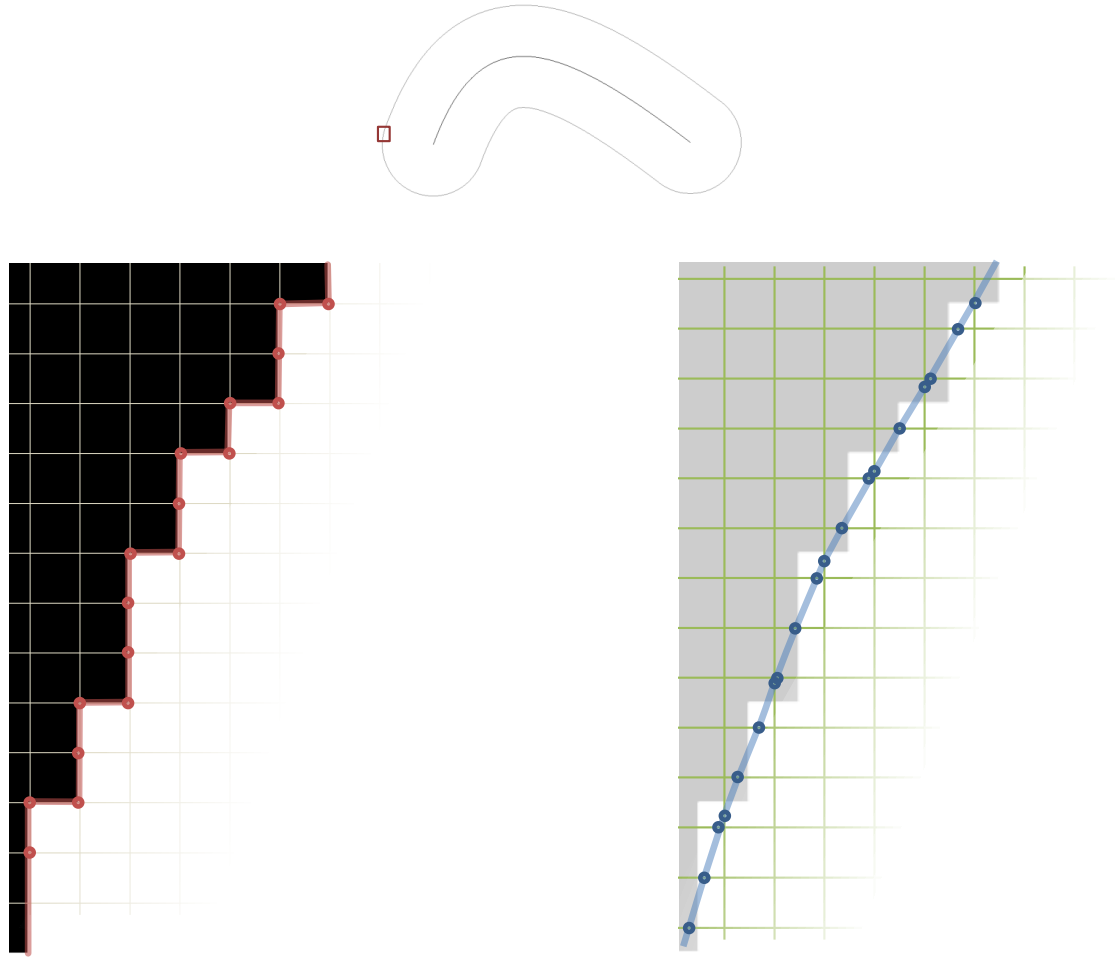
Figure 18. Marching squares examples. *(a)* A simple example. Two edges exhibit a sign change; the two points are linearly interpolated to form a segment of the offset curve. *(b)* A more complex case. Four intersections are detected; the interpolated value of the center point is used to determine which pairs to connect.

For a continuous dataset, the isoline must enter and leave the squares either not at all, twice, or four times. In the latter case, two parts of the isoline must pass through the squares, and the four points are connected in two lines. To determine which pairs are connected, the estimated value at the center of the cell is determined through bilinear interpolation, then compared to one of the vertices. If the sign of the center and the vertex are the same, they both lie on the same side of the isoline and the pairing should be made so as to not divide them, as shown in Figure 18b.

Marching squares by itself returns a long list of line segments, but the application demands a list of *contiguous* line segments. Thus, we need to order the offset segments into

loops, with each line adjacent to its neighbors. To accomplish this, the marching squares algorithm is modified to first generate all the grid intersection points on the offset curve, storing them in an array that is overlaid on the original geometry. Then, the space is iterated over a second time, and each time an edge is detected, it is recursively followed all the way around the loop, following the path from one square to the next until the entire loop is extracted. In each square, the new offset line is constructed from the interpolated vertices as described above, then added to a list of output lines and flagged “ignore” to avoid further processing.

The result is a list of contiguous line segments that reflect the exact offset curve to a level of precision much better than the size of a pixel. This contrasts strongly with the approach taken by Li, et al. (2009), which looks only at color change at the edge of the drawn object, providing an accuracy on the order of the pixel size and requiring proportionately more memory to obtain a similar error bounds. Figure 19 illustrates this tradeoff; note that the grid used to evaluate the offset curve is shifted by 0.5 pixel both horizontally and vertically to match the “vertex at the corner” layout required by marching squares.



(a) *(b)*

Figure 19. Offset curve extraction from *(a)* color data and *(b)* height data, zoomed in. In *(b)*, grey pixels are geometry with a depth value less than zero.

Error Analysis

One of the biggest gaps that needs to be closed before this kind of graphical approximation to offsets can find a place in industrial applications is the analysis of the error sources inherent in the algorithm. The ultimate goal is to have a realistic, algebraic expression for the upper bound on the error given a simulation environment. This can then be inverted to specify an appropriate value of the relevant variables while ensuring the entire process maintains an error less than the allowable process error.

Five major sources of error are present in this approach. In the following subsections, each of these sources of error will be addressed and quantified where possible. For the purposes of illustrating and understanding the errors involved, this section assumes the progenitor curve is any smooth, continuous parametric plane curve, though these principles can be easily extended to non-smooth input curves. Finally, a brief description of the numeric error estimator used to validate the results will be presented.

Error Source 1: Discretization of Input Geometry

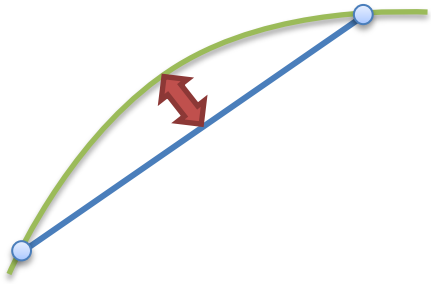


Figure 20. Discretization error. Original (green) curve segment is approximated by linear (blue) segment.

Because our approach reduces all input geometry to line segments, any non-linear input curve will create error in the discretization step. Whenever a linear approximation for a function is used, higher-order derivatives are truncated and curvature information is lost, introducing error. Thankfully, discretization into linear segments is a popular operation, and most geometries possess good, error-bounded subdivision schemes. In the case of Bézier curves, de Casteljau's method provides a fast, recursive, error-bounded discretization.

The effect this approximation has on the offset curve is the real quantity of interest, however. A little geometry can demonstrate that the error in the discretization described above is identical to the expected error in the offset, all other things being equal. Allowing only that the source curve is continuous and smooth in the first derivative, without loops in the discretization area, we can draw a picture of a linear approximation of a short section of curve like that in Figure 21a. The Mean Value Theorem guarantees that at least one point on the original line has the same slope as the linear approximation, and calculus tells us that the maximum deviation from the flat line must occur at such a point. When the offset operation is

applied (Figure 21b), both curves shift a distance *d* along their normals. However, since at the maximum point, the normal to the curve and the normal to the approximation are aligned, the offset distance and direction is the same in both cases. Since the distance between the two points remains unchanged, the discretization error of the offset is the same as the discretization error of the source curve (Figure 21c). In this way, the error metric used in the original discretization process can be directly applied to the error introduced in the output curve due to discretization.

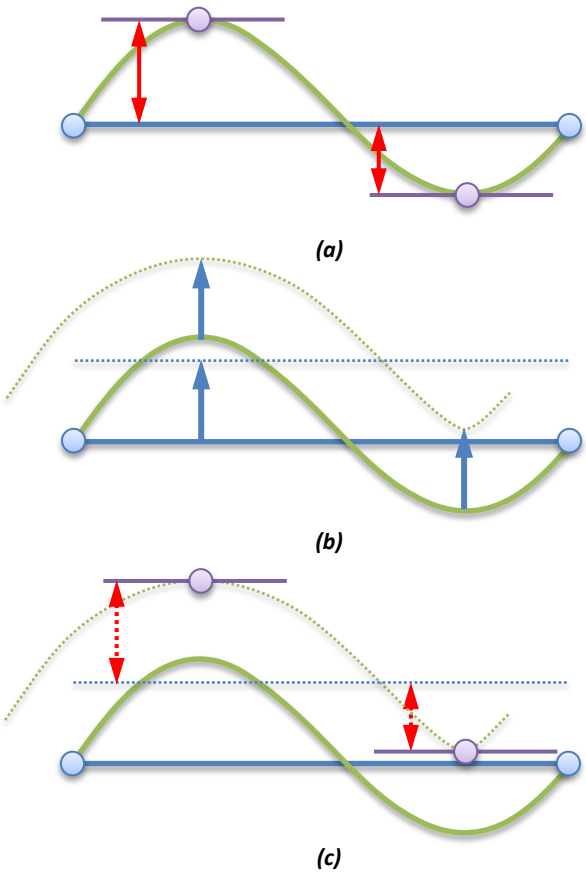


Figure 21. Error introduced by offsets of linear approximation of more complex functions over short areas. *(a)* Error before offsetting. *(b)* Offsetting operation. *(c)* Error after offsetting.

Error Source 2: Cone Approximation Error

Because the graphics pipeline is only capable of drawing simple geometric primitives (points, lines, and triangles), more complex objects such as cones are approximated by a set of triangular primitives, and this approximation produces error. This error has been treated in Hoff, et al. (1999). His error estimate can be stated as

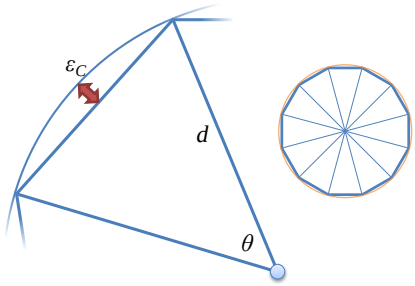


Figure 22. Cone approximation error illustrated, viewed from above

$$\varepsilon_C = d\left(1 - \cos\left(\frac{\theta}{2}\right)\right) \quad (3)$$

where ε_C is the maximum error, *d* is the circle's radius (in this case, the offset distance) and θ is the angle subtended by a single triangular primitive. Alternatively, the error estimate in (3) can be expanded to relate to *N*, the number of primitives used to represent a single cone:

$$\varepsilon_C = d\left(1 - \cos\left(\frac{\pi}{N}\right)\right) \quad (4)$$

Error Source 3: Valleys between Cones

A third source of error occurs in the region of the offset between cones when tents are not used. Illustrated in Figure 23, this error magnitude ε_V can be evaluated in terms of the offset distance *d* and the distance between the two cone centers *s*, and is found to be

$$\varepsilon_V = c - \sqrt{c^2 - \left(\frac{s}{2}\right)^2} \quad (5)$$

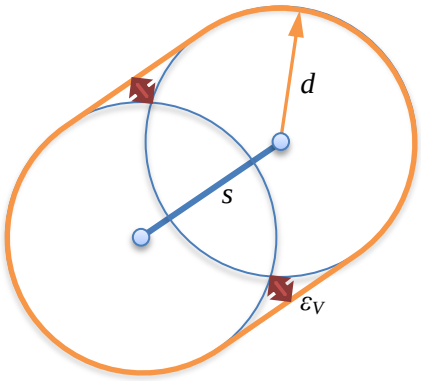


Figure 23. Valley error illustrated as the difference between orange curve (desired result) and blue curves (rendered geometry).

This error metric presents difficulties because depending on the discretization scheme used on the original curve, the distance between successive points (*s*) may not have a known upper bound. This makes it much harder to predict, for arbitrary input geometry, what the error will be when using a given discretization scheme.

To resolve this issue, the concept of tents was introduced, which offsets the line segment itself instead of just the end points and removes all valley errors entirely.

Error Source 4: Resolution of the Pixel Grid

A fourth source of error, one which is unique to a rendering approach to the offset problem, occurs due to the resolution of the pixel grid. The grid resolution affects the maximum length of the output lines, creating a new discretization error, as well as the minimum size of features that can be resolved. Each of these error sources will be treated in the following paragraphs.

For a grid with spacing *w*, the worst case error that does not leave the grid square is shown in Figure 24. Without leaving the cell and changing the resulting topology, the exact offset curve can deviate by no more than $w\sqrt{2}$.

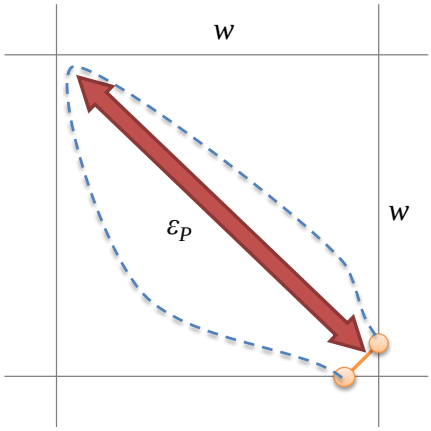


Figure 24. Worst case discretization error visualized. Grey lines represent grid lines travelling through pixel centers. Orange line is the discretized output curve, and blue represents a cusp condition.

Due to the nature of the marching squares algorithm, it is possible for the offset to exit and re-enter the square on the same side without being detected. Thus, only features that have a minimum dimension larger than *w* are guaranteed to be resolved. For features with minimum size less than *w*, one of three cases can occur, as illustrated in Figure 25. In Figure 25a, the feature overlaps with a gridline along its entire length, and the curve is fully resolved. This is the best case. Figure 25b shows a marginal feature that falls entirely between horizontal grid lines. Because the marching squares algorithm assumes a linear function between datapoints, it

cannot detect the double sign change in the two vertical edges and the entire feature is not resolved.

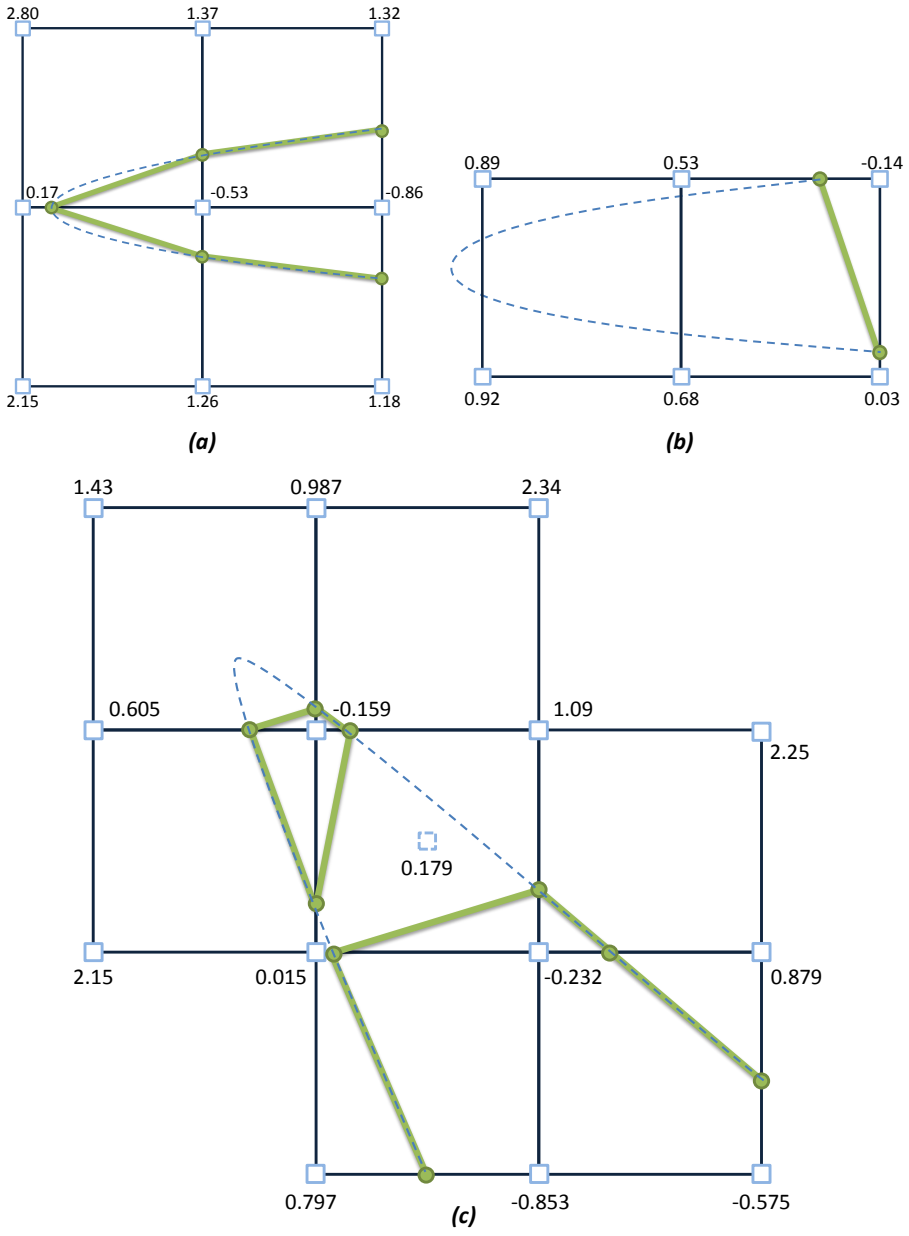


Figure 25. Marginal features (those with minimum dimension less than the grid spacing) can be either *(a)* fully resolved, *(b)* unresolved, or *(c)* partially resolved, depending on grid placement. Blue dotted lines are the exact offset curves, green lines represent extracted offset, and blue squares are pixels with associated depth values.

An example of the third alternative is shown in Figure 25c. In this case, the marginal feature lies diagonally on the grid, but the change in the depth function is again too rapid for the linear approximation assumed by the marching squares algorithm to correctly handle. As a

result, when the center square is processed, the interpolated midpoint depth inconsistently predicts the actual depth value and causes the two lines to connect the wrong pair of vertices. The result obtained contains aspects of the intended geometry, however an additional, erroneous output curve has been created. This *phantom curve* is topologically correct except for one edge, which may or may not share the same cell as the associated erroneous edge on the main result curve. Note also that phantom curves can surround more than one pixel, and more than one phantom curve can be present per cusp.

Because the exact location of marginal features with respect to screen pixels makes it possible to entirely miss them, the magnitude of this error can be bounded in only one axis. The algorithm may omit any feature of minimum size less than *w*, but the length of this feature is, under the right circumstances, theoretically unbounded.

Error Source 5: Floating Point Roundoff

The final error source present in this algorithm occurs due to the floating point storage used for geometry and depth data. Because the graphics pipeline operates on 32-bit floating point values for locating geometry, interpolating depth values between fragments, and storing values in the depth buffer, the results of any graphical offsetting algorithm will be limited by the resolution of a 32-bit float.

Numeric Error Estimation

In order to estimate and visualize the error produced by the graphical offsetting process, a numeric error approximation was developed. This metric does not provide an exhaustive or guaranteed accurate error analysis, but does provide a good estimate in most cases and allows easy visualization of the relative error between cases. An example result is shown in Figure 26.

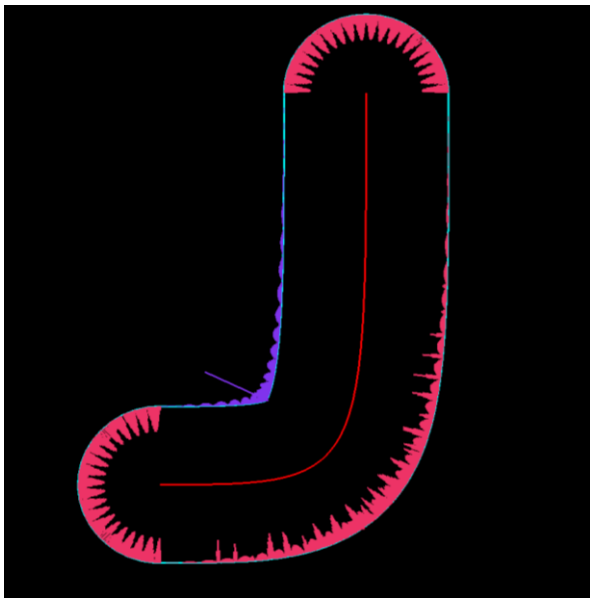


Figure 26. Visualization of offset error. Pink represents gouging error; blue represents undercut. Errors are magnified 100X.

In order to estimate error, the graphical offset curve was obtained, then each vertex and the center point of each line was checked for error by evaluating the distance between the point and the nearest point on the progenitor curve. If the distance is less than the offset distance, gouging error occurred (shown in pink in Figure 26); if it is greater, undercut error occurred (shown in blue in Figure 26). The error was then plotted as a line originating at the point tested and travelling along the line between the point and the closest point on the original Bézier curve, scaled by a user-defined factor for visibility.

The process of determining the smallest distance between a point and a Bézier curve, used at each test point described above, was accomplished in two steps. First, the discretized curve was searched for changes in sign of the derivative of the distance function, seeking locations along the curve closest to the point in question. For each sign change, the *t* value for the corresponding spot on the original Bézier curve was obtained, then used as an initial guess in a Newton's method estimation of the nearest zero crossing of the function

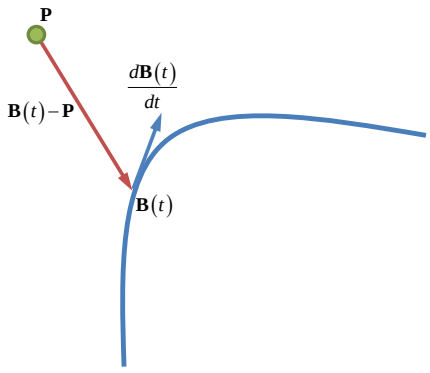


Figure 27. The shortest distance between a point and a Bézier curve.

$$\frac{d\mathbf{B}}{dt} \cdot (\mathbf{B}(t) - \mathbf{P}) = 0 \qquad (6)$$

where $\mathbf{B}(t)$is the parametric function defining the Bézier curve and $\mathbf{P}$ is the offset point (see Figure 27). $\|\mathbf{B}(t) - \mathbf{P}\|$ is at an extreme when it is perpendicular to $\frac{d\mathbf{B}}{dt}$**,** the tangent to the Bézier curve at that point. When the two vectors are perpendicular, their dot product is zero, so utilizing Newton's method to zero Equation (6) gave the value of *t* for which $\|\mathbf{B}(t) - \mathbf{P}\|$ gave a maximum or minimum distance between the curve and the point $\mathbf{P}$. The output distance was obtained by taking the minimum of $\|\mathbf{B}(t) - \mathbf{P}\|$ for each change in slope of the distance function in the discretized curve, the value of $\|\mathbf{B}(t) - \mathbf{P}\|$ obtained when the Newton's method approximation is seeded with the end points, and the distance between the end points themselves and $\mathbf{P}$. In all cases, *t* values outside the range used by the parametric curve were culled.

The result of this algorithm was an error estimate that related a large number of test points on the final output curve to the original, mathematically-defined Bézier progenitor curve. In this way, error induced by all error sources, from curve discretization to screen resolution, appear in the error estimate.

RESULTS AND DISCUSSION

In this chapter, the results and performance of our graphical approximation to offset curves is presented, accompanied by appropriate discussion. First, results are presented demonstrating basic offsetting and loop removal capabilities, followed by a series of error plots describing the experimental error obtained for various simulation scenarios. The next section describes the experimental algorithmic performance of our method. Following these results, advantages and disadvantages of this approach are discussed, and the chapter concludes with a discussion of unexpected results obtained and lessons learned.

Offset Curve Results

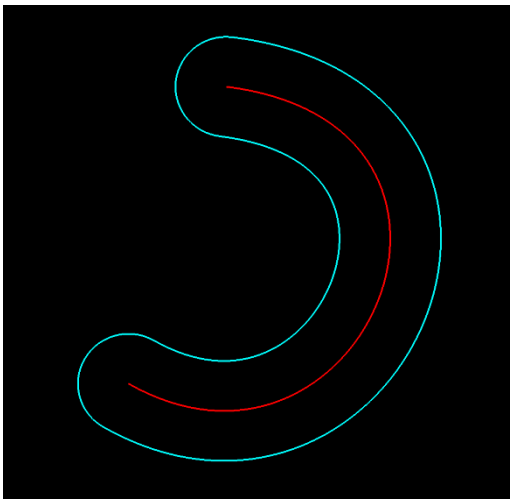


Figure 28. An offset (teal) of a Bézier curve (red)

We began with a simple offset of a Bézier curve, Figure 28. As expected, the offset curve tracked nicely with the original, producing both interior and exterior offsets, as well as circular caps around the end points. This behavior is the most easily obtained result for a graphical approach, and separating the caps, interior, and exterior curves in post-processing should be straightforward.

Local loop elimination functionality is shown in Figure 29, in which the Bézier was given a curvature much smaller than the offset distance. This produces a cusp where the two exact offset curves intersect (see Figure 7). As predicted, the loop is correctly eliminated, though the

cusp is not fully resolved once it drops below the resolution limit. Elsewhere, however, the output curve traces the exact offset with a precision much higher than the rendering resolution.

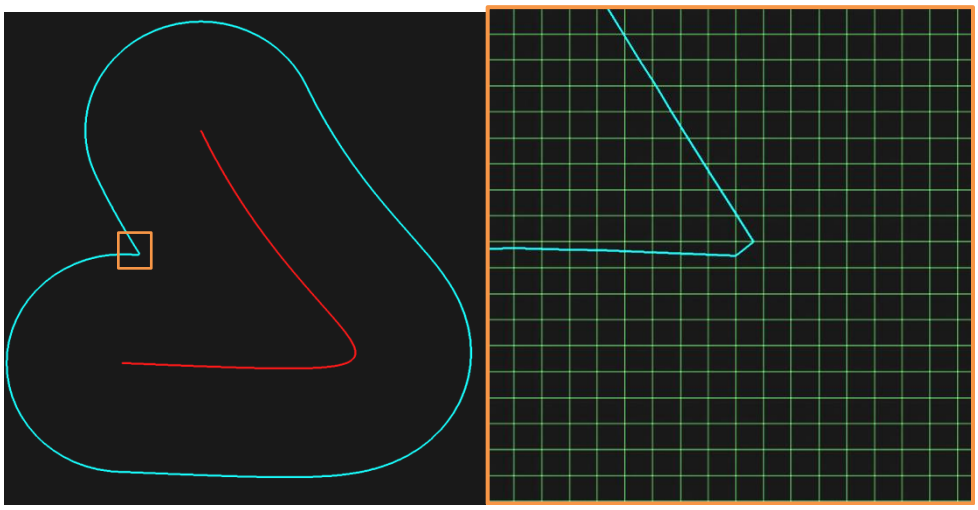


Figure 29. Local loop elimination.

Figure 30 shows an example of global loop elimination. Graphically, this problem is solved in a manner identical to local loop elimination, except the output geometry includes two curves instead of one.

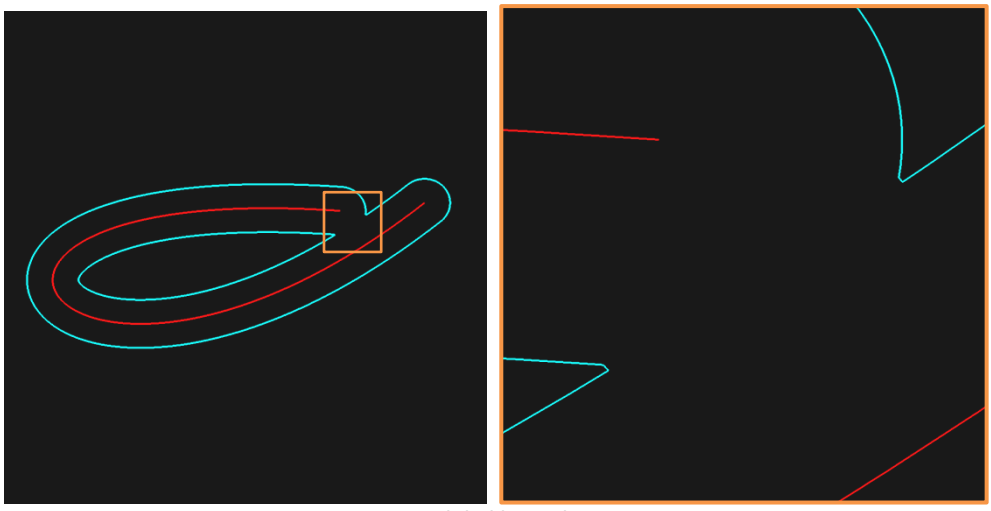
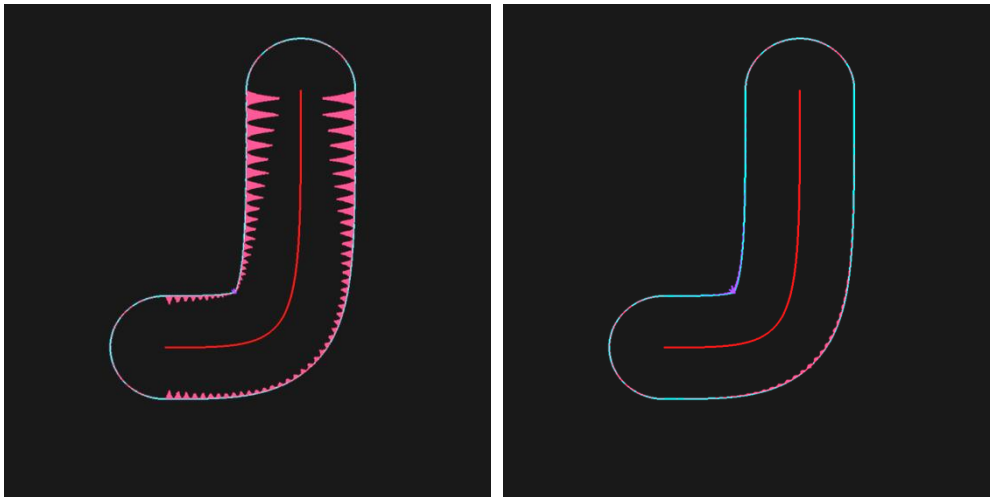


Figure 30. Global loop elimination.

Error Visualization

The ability to visualize the error changes that occur when varying the computation parameters was key to our understanding of error sources. Enabling tents, for example, resulted in a profound reduction in error, as shown in Figure 31. It is clear in places of wider discretization that the valley between the cones is producing a comparatively large amount of error. The caps at the ends show very little error due to a large number of polygons making up the cones. In the crook of the curve, a small amount of undercutting error occurs in the region where the near-cusp drops below the resolvability limit.

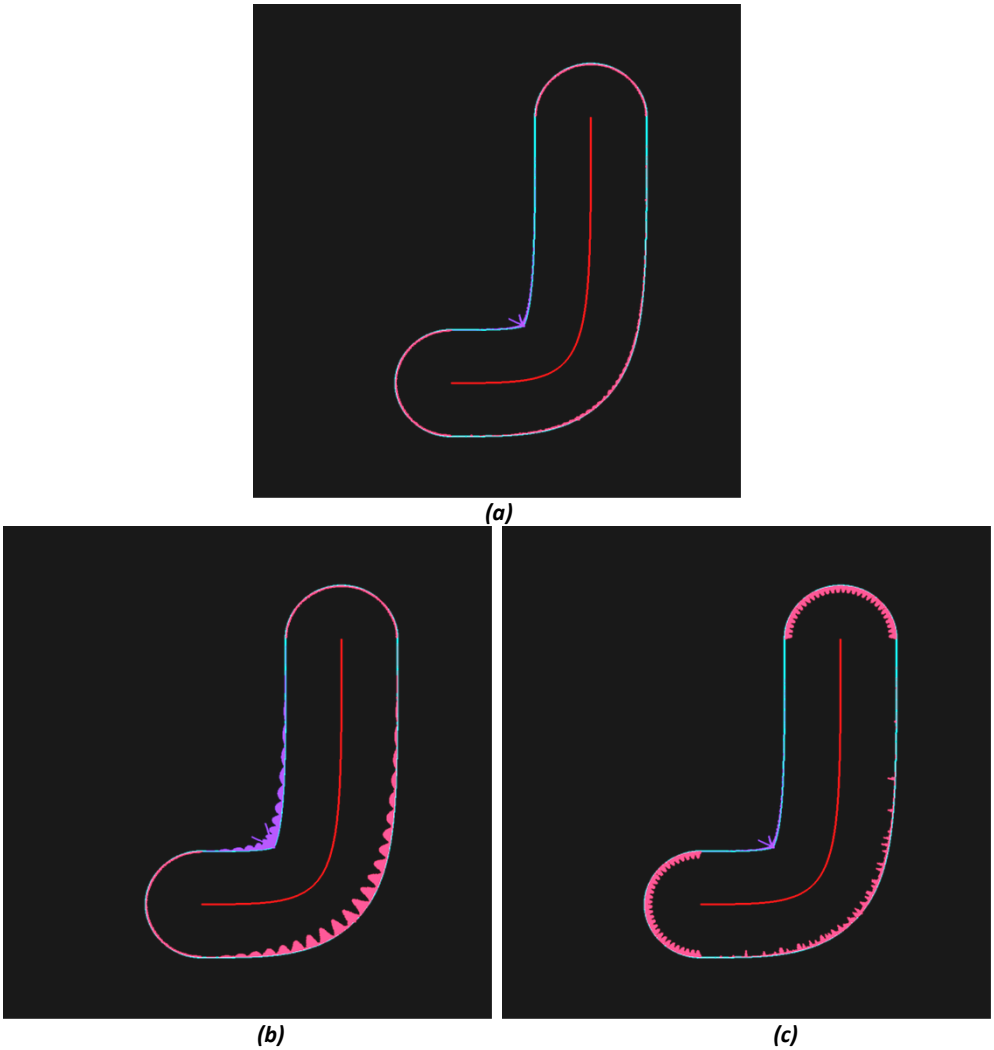


(a) ***(b)***

Figure 31. Error, magnified 50 times, of a curve offset *(a)* before and *(b)* after enabling tents.

Figure 32a shows the error plot produced by relatively high quality settings. For this image, the curve was discretized into 60 linear segments, cones were represented with 125 triangles each, and the window used in rendering the offset geometry was 600x600. Tents were enabled for all three images. By decreasing the number of progenitor curve line segments to 25, the error increased as shown in Figure 32b. Note that as predicted, the discretization error shows itself most prominently when the curvature per line segment is comparatively high (these curves are discretized by stepping a constant Δt). Also, the direction of the error tracks with the

direction of curvature. In concave regions, the discretization causes undercut in the output geometry, while in convex regions, the opposite occurs. If instead the number of triangles per cone was decreased to 60, the plot changed to look like Figure 32c. In this case, the error appears on the caps, and convex portions of the offset curve, anywhere cones dominate the visible geometry.



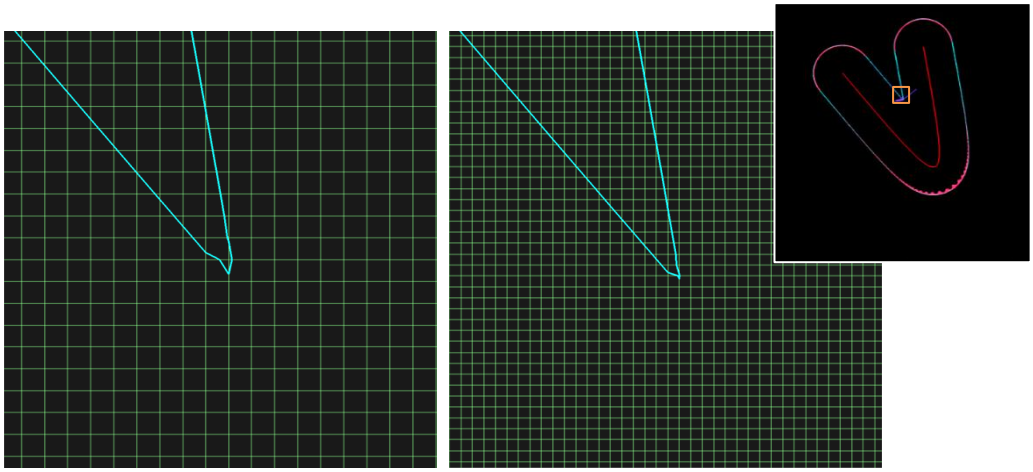
(a)

(b) ***(c)***

Figure 32. Error plotted (magnified 100X) in *(a)* the base case, *(b)* with fewer input curve segments, and *(c)* with fewer polygons per cone.

Varying the resolution of the computation buffer has relatively little effect on the bulk error but directly affects the resolution of marginal features. Figure 33 shows the same cusp

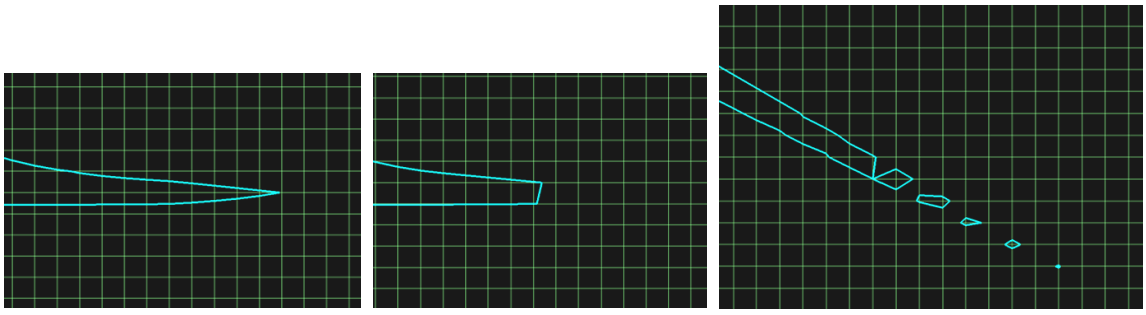
extracted at two different resolutions. As the resolution increases, the larger features of the offset remain unchanged, but cusp, which near the tip drops below the grid resolution in size, is much better resolved at higher resolutions.



(a) *(b)*

Figure 33. Two frames of the same portion of a cusp at different resolutions. *(a)* 300x300 pixels; *(b)* 600x600 pixels

As a result of this inability to resolve cusp regions exactly, the marching curves algorithm produces a variety of interesting, albeit wrong, results. Figure 34 shows that the result can vary drastically depending on the interaction between the grid and the curve itself. Outside of these marginal areas, however, the solution is remarkably insensitive to grid resolution; low-error offset approximations can be obtained from buffer sizes as small as 300x300 while maintaining accuracies on the order of 0.1% (error / offset distance).



(a) *(b)* *(c)*

Figure 34. Results for a marginal feature (minimum size less than $w\sqrt{2}$). *(a)* Nearly correct. *(b)* Badly truncated. *(c)* Several phantom curves. Grid lines represent edges of squares used for isoline extraction.

Algorithmic Performance

In order to gauge the effectiveness of the algorithm in scaling to larger systems, a series of three tests was performed, each one scaling a simulation parameter about a base case. The density of points on the discretized Bézier curve, the number of polygons in the cone approximation, and the resolution of the pixel grid were all evaluated. To emphasize trends in the scaling, the base case was chosen to be a comparatively rough set of conditions (20 curve points, 50 triangles/cone, and 200x200 resolution). Figure 35 shows the test geometry used, which contains a single marginal feature in the "crook" of the elbow, along with the error plot in the base case. By changing one parameter at a time, we obtain a quantitative measure both of the performance of the algorithm as well as the effect that variable has on error. The following pages will explore the results of this evaluation.

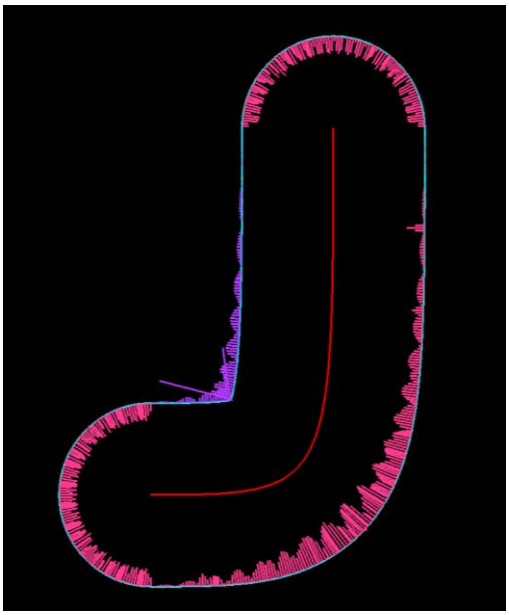


Figure 35. Error plot (magnified 100X) for the base case used to evaluate scaling performance.

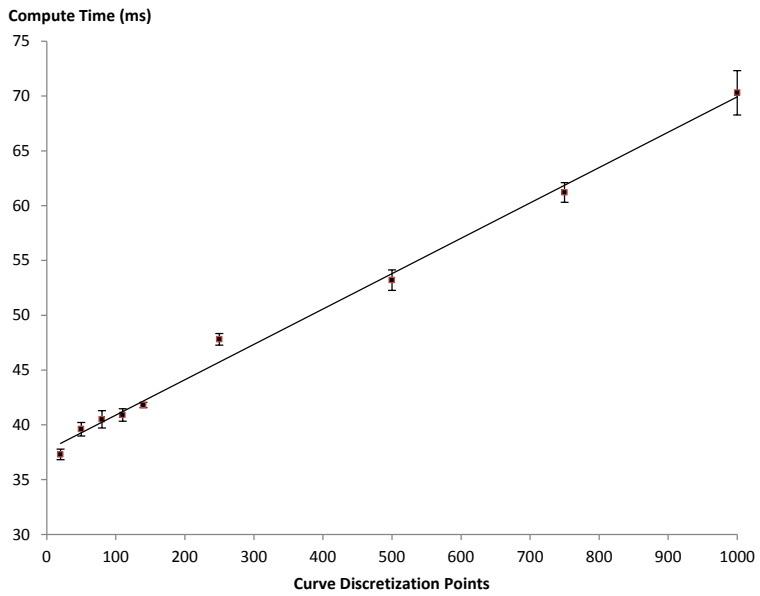
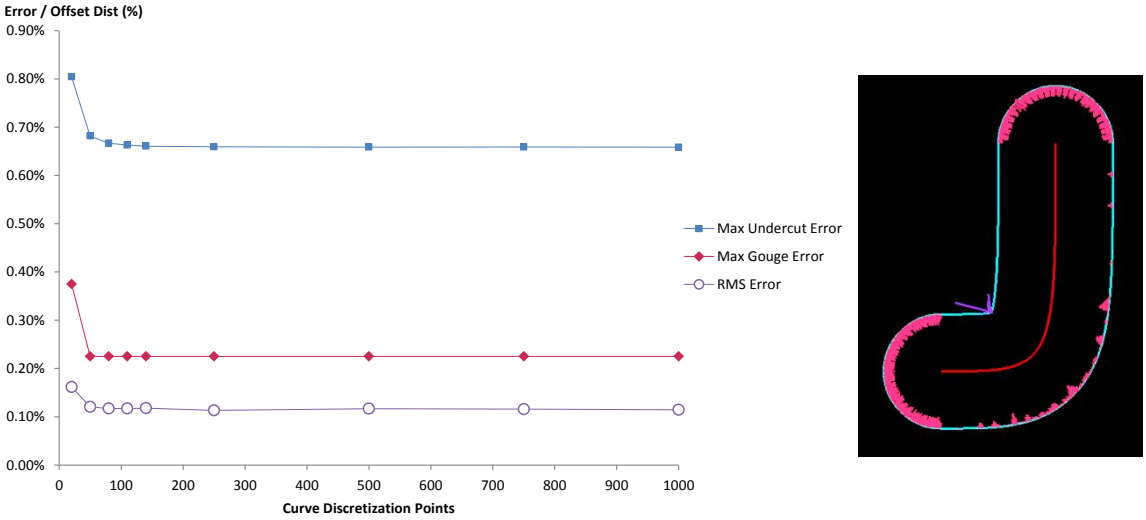


Figure 36. Algorithm speed vs. the number of curve input points; linear fit ($R^2 = 0.9953$) overlaid. Error bars represent a 95% confidence interval.

Figure 36 shows the relationship between execution time and the number of points placed in the Bézier curve discretization. Overlaid on the graph is a second-degree polynomial fit to the curve. Because the coefficient on the n^2 term is negligible, we can confidently conclude that the algorithm scales linearly with input geometry complexity. Not only is the relationship $O(n)$, the coefficient is quite small: increasing the number of vertices by an order of magnitude from 100 to 1000 just doubles the computation time. Since most 3D applications routinely throw hundreds of thousands of polygons at the graphics card and expect real-time performance, rendering operations have grown to scale very well with geometric complexity.

In addition to tracking speed, Figure 37 shows the effect changing the number of discretization points has on error. Error is plotted as the absolute positional error divided by the offset radius, expressed as a percentage, for maximum gouging error, maximum undercut error, and root-mean-square (RMS) average error. As is clearly visible in the graph, other error sources take over quite quickly (due to the low quality settings of the base case), and discretization affects all three error types (undercut, gouging, and mean error) to a similar degree.



(a) *(b)*

Figure 37. ***(a)*** **Offset error plotted against curve discretization points.** ***(b)*** **Error plot (magnified 100X) of the offset at 1000 pts**

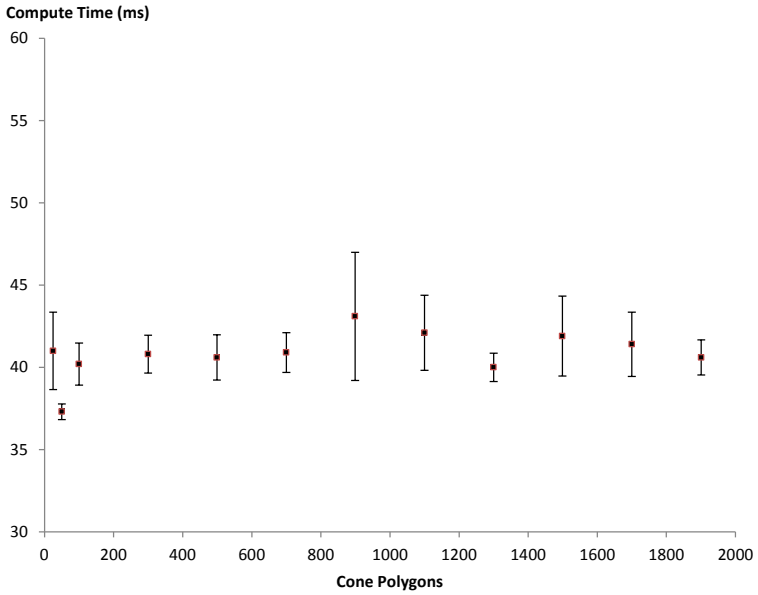
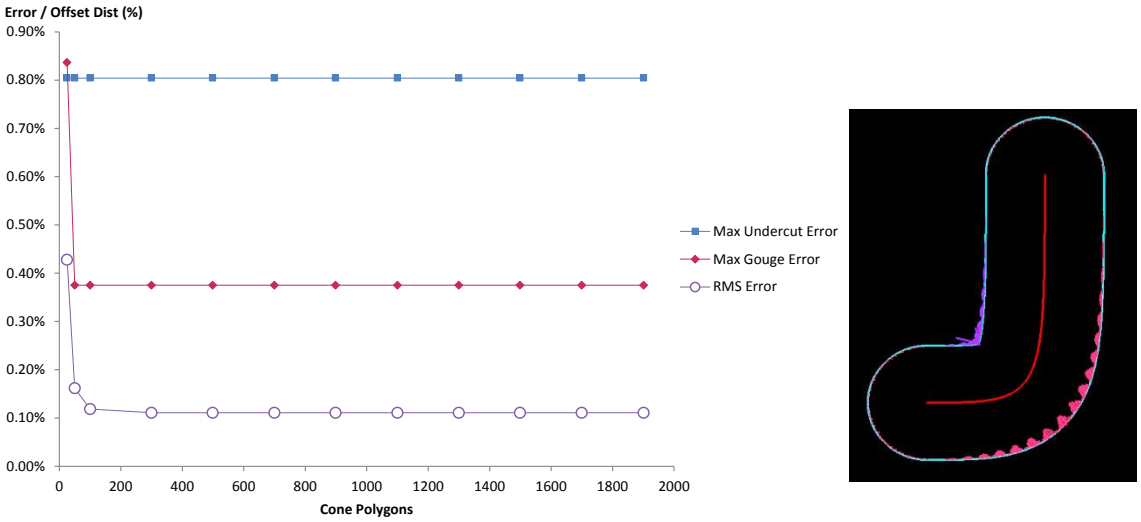


Figure 38. Algorithm speed vs. the number of polygons in each cone. Error bars represent a 95% confidence interval.

Varying the complexity of the cone model is also predicted to affect performance. As shown in Figure 38, however, no clear trend is visible, even for very large polygon counts. It is possible that the relatively small number of cones created so little additional geometry that the

rendering pipeline never saw enough load to create a performance difference. Rendering cones with moderately high polygon counts should not negatively affect performance.



(a) ***(b)***

Figure 39. ***(a)*** **Offset error plotted against number of cone polygons.** ***(b)*** **Error plot (magnified 100X) of the error at 2000 polygons per cone.**

The cone polygon relationship with error present in Figure 39 resembles the trend seen in Figure 37: the error contribution is only significant at relatively small polygon counts. It is notable that the polygon count has absolutely no effect on the undercut error term, which occurs in the concave “elbow” region of the offset curve. In this area, the tents cover all cone geometry and the density of polygons on the cones does not affect any of the final offset geometry in that area.

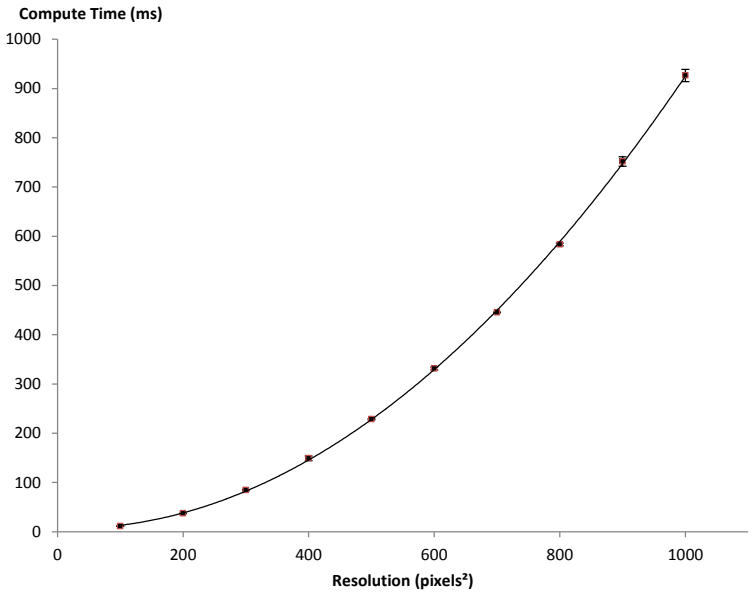
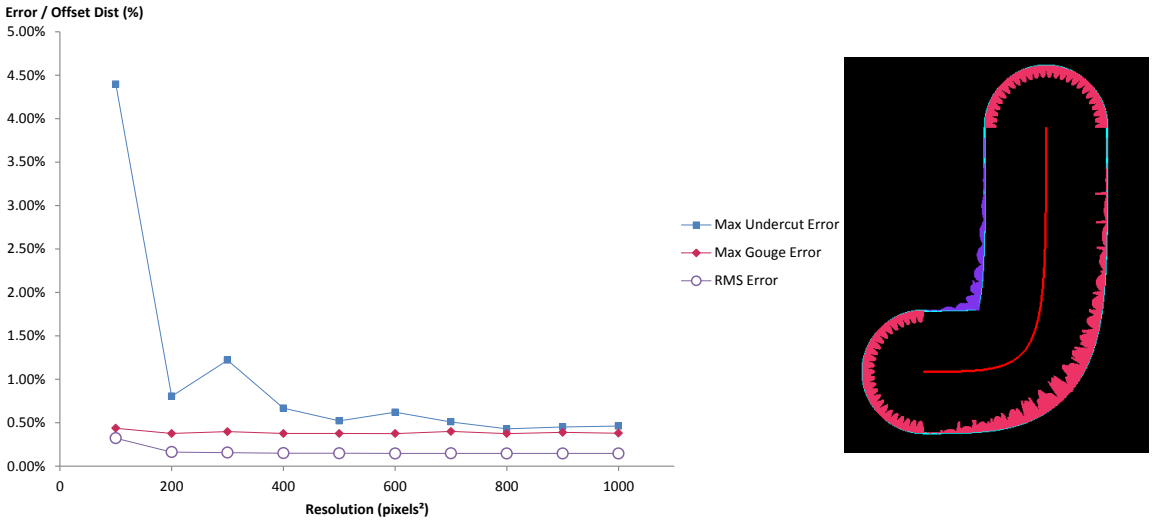


Figure 40. Algorithm speed vs. buffer resolution; quadratic trend line overlaid. Error bars represent a 95% confidence interval.

Figure 40 reports the relationship between the screen resolution and algorithm performance. Due to the fact that the curve extraction algorithm requires iterating over every pixel in the output buffer, we anticipate an $O(n^2)$ relationship with respect to resolution, one which is clearly seen in Figure 40.



(a) ***(b)***

Figure 41. *(a)* Relationship between buffer resolution and offset error. *(b)* Error plot (magnified 100X) of the error at 1000x1000 buffer.

Resolution has a profound impact on the maximum undercut error (Figure 41a). Because the maximum undercut error occurs at the marginal feature in the crook of the elbow, as the resolution changes, the grid intersects the corner in various ways, creating a highly-unpredictable relationship. As resolution continues to go up, the feature becomes fully resolved and error converges to a single value. Note that the gouging and RMS errors are both almost totally unaffected by resolution. Thus, if a better method of detecting marginal features were present, reducing the maximum undercut error significantly, very low resolutions would be easily attainable. One could then zoom in only on the regions containing marginal features, re-render a small part of the original window, and maintain strong performance.

Discussion

Using a graphical approximation such as the one presented here has some significant advantages over the alternatives presented in the literature. Instead of requiring complex logic and expensive search operations to find and eliminate local and global loops, this approach allows direct and automatic elimination of unwanted offset geometry. Additionally, it is fast, taking advantage of the parallel processing capabilities of the GPU for generation of the depth field used to create the offset.

A graphical approach also has several significant limitations that must still be overcome before it can be a viable option for industrial applications. Prior to this work, the error analysis available for this method has been almost totally lacking in the literature, and further refinements of the method will still be required before a full understanding of the error involved in marginal features is finalized. This is critical; NC pocket machining applications frequently involve pockets sized just large enough to accommodate the tool cutting the pocket. Were the

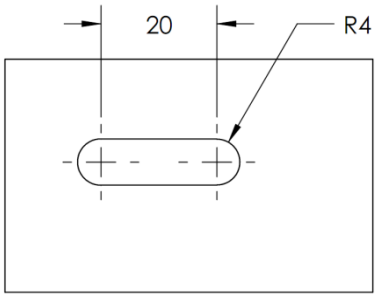


Figure 42. A simple slot creates difficulties for the graphical approach.

engineer to wish to cut the slot in Figure 42 with an 8 mm diameter endmill, the tool path would be a single, straight line. A straight line is a loop of infinitely small thickness; the current graphical approach would entirely miss this solution because it is well below the resolution threshold.

A second shortcoming that must be overcome is the separation of the output curve into offsets on each side and caps on the end. Although NC applications need the entire offset in many cases, most CAD systems want the user to be able to choose how much of the offset chain he gets.

Finally, commercial CAD packages are concerned about data proliferation: a graphical approach such as this produces a great many output points, even for simple input geometry. Storing and further processing such large amounts of data is prohibitively resource-intensive. This graphical approach should be combined with an interpolation algorithm such as Piegel and Tiller's (1998) to reduce the quantity of data returned by fitting higher order curves to the offset points.

Unexpected Results and Lessons Learned

Implementing and exploring a new graphical approximation to the offset curve problem has proven a challenging and at times daunting task. I have learned much, from algorithm design and debugging to effective literature search techniques to explaining the project in a way a first grade teacher can understand. Along the way, several of the results genuinely surprised me, observations that undid misconceptions and caused a great deal of consternation until fully understood.

In one instance, the numeric error analysis function was reporting unexplained errors that shifted with location on the screen (Figure 43). After several days of consternation and a great deal of debugging, I discovered that the root cause was an incorrect assumption regarding the exact size of the viewing volume. The working, unjustified assumption, upon which the offset extraction algorithm had been built, was that the viewing volume started at the center of the first pixel on each edge. Instead, a little math and a few code tweaks demonstrated that the unexplained error went away entirely when the viewing volume started at the *edge* of the first pixel instead of the center. The further away the curve moved from the origin, the more that 0.5 pixel error influenced the result, creating location-dependent error variations. This revelation reinforced one of the key lessons learned in the project: when using the GPU for more than just graphics, details matter. A 0.5 pixel error may not ruin a video game, but the moment one begins to rely on the numeric accuracy of the rendering transformation, “close enough” isn’t sufficient.

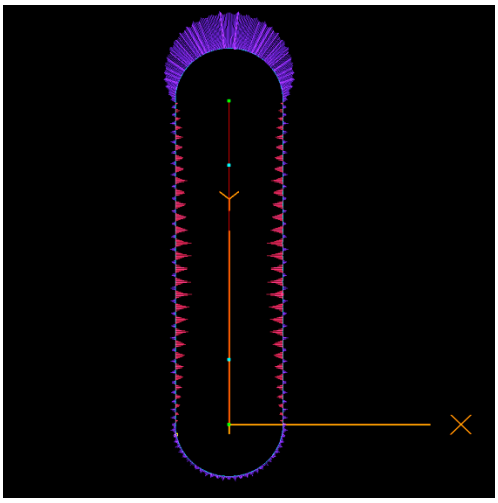


Figure 43. Initially unexplained error bars that varied with position.

A second unexpected result reinforced one of the key complexities with extracting useful information from a discrete field such as the depth buffer. The phantom loops phenomenon was a highly unexpected result, one that took several months to fully understand and explain. It was a very long trail of debugging and failed hypotheses that led to the explanation presented above; in part because the author assumed an algorithm with such wide application as marching squares should not be the cause of such a blatantly erroneous result.

The results presented here, both good and bad, represent only a small fraction of the test cases examined. The above discussion attempts to place these results in the context of the broader application for an algorithm such as this, as well as explore its limitations. The next chapter further expands on several ways of mitigating these drawbacks and bringing this method closer to industry utility.

FUTURE WORK

The scope of this project precludes a great deal of additional avenues of exploration to further understand and optimize this algorithm. In this chapter, a brief overview of several avenues of future pursuit will be given. The future work on this problem falls mainly into two arenas: refinements to the existing method to improve performance and expansions on the technique to address new problems.

Refinements and Improvements

As mentioned above, several refinements are needed before this approach can be utilized in any practical sense. The three most necessary refinements, in our opinion, are marginal feature detection and refinement, curve tracing and parameterization, and curve subdivision.

Detecting marginal features remains a major challenge. Several clues have been observed that could possibly lead to effective methods for locating features that are not fully resolved. We note that small changes in grid resolution have significant effects on the resulting topology in regions of marginal features, while the vast majority of the curve remains unaffected. If these changes could be detected, two or three closely related resolutions could be attempted and the marginal features located. After locating them, portions of the scene could be re-rendered, dynamically subdividing the output curve until the desired precision is met.

An alternative approach to marginal features is to revise the marching squares approach. Instead of using a strictly linear approximation, if both depth and slope information were obtained from the rendering process, perhaps with the use of a fragment shader, the

linear interpolations present in marching squares could be replaced with quadratic estimators that would more correctly discern the sub-pixel variations present in the output.

Curve tracing, or fitting a higher-order curve to the points obtained from the graphical offset operation is needed to reduce data proliferation. Several algorithms using a smaller number of test points have been proposed (see Interpolation Methods in Table 1); one should be selected and adapted for this application. Alternatively, a variety of other methods provide algorithms specifically designed to fit functions to large datasets of points, see for example Goshtasby (2000). An important aspect of curve fitting is cusp detection – a good parameterization will not try to fit a smooth curve over a cusp point; several aspects of the graphical approach make locating cusp points relatively straightforward. Two possible avenues for exploration include finding cusps in the same process that detects marginal features (most cusps appear as marginal features when rendered), and taking the offset of the offset, then searching for pixels covered by the original curve but not the offset. A cusp point will occur on the nearest point in the offset curve.

The final important area for further investigation involves trimming the offset curve into caps, inner, and outer offsets. This problem might be approached by determining the progenitor curve's normal at the beginning and end of the line, then slicing the offset curve in that region by intersecting the normal at the end points. An alternative, more graphical approach is to color the cones drawn on the ends of the curve differently, then instruct the curve extraction algorithm to break the line when the color changes.

New Avenues of Investigation

The fundamental concept employed in this problem to obtain offset curves has possibilities in a wide variety of related fields. Three areas of expansion present themselves. Doubtless, more abound, but these examples show the flexibility of the rendering process for solving more complex problems.

In the first, one of the beauties of the graphical approximation used here is the spatial flexibility of the rendered geometry. In a traditional offset, the size of the cones rendered remains constant everywhere, but this need not be the case. A variable-radius offset could be easily obtained by changing the size of the cones and tents along the input line's length. Such a feature could have applications in variable-radius fillets in CAD, slope-controlled offset distance to obtain uniform wall thickness in rapid prototype hollowing operations, and elsewhere.

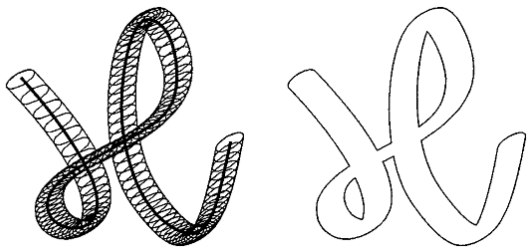


Figure 44. The Minkowski sum boundary of an ellipse and a cursive "H". (Lee, Kim, and Elber 1998).

A similarly easy tweak to the graphical approximation allows the computation of Minkowski sum boundaries, the superset of geometric functions to which offset curves belong. The Minkowski sum boundary of two shapes is the curve obtained when loops are eliminated from their convolution, see Figure 44. An offset curve is a Minkowski sum boundary of the input

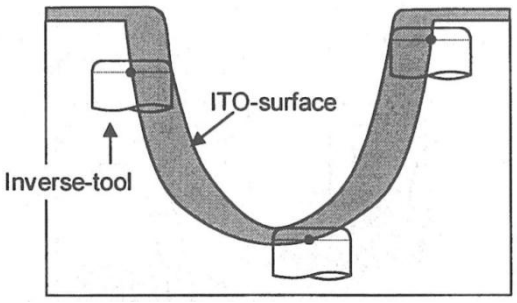


Figure 45. The vertical path taken by a cutter in a pocket is described by the Minkowski sum boundary of the pocket and the inverse-tool. (Choi, 1998)

curve and a circle of radius *d*. In NC machining, the Minkowski sum boundary between the tool profile and the pocket elevation defines the curve the cutter should walk vertically in the machining process.

A third direction for further expansion involves correlating these methods with the more complex 3D offset surface problem. It is possible to directly expand the techniques discussed here to a third dimension, however, rendering it on GPU hardware directly becomes much more difficult. Nevertheless, a software, 4D rendering environment could be easily set up, and the method adapted to offset surfaces. In the world of sculped sufrace machining, offset surfaces are a critical player in determining tool paths.

CONCLUSIONS

This project has sought to explore the applications of graphics hardware to generating offset curve approximations in a way that obtains results desirable to many practical applications without requiring the complex algorithms and limitations found in most existing methods. A graphical method was reviewed in the context of the larger body of research, both in offset curve computation by other means, and in similar applications of graphics processing in related problems. Approaches utilized in several other papers were developed and synthesized into a method that expands directly on the work of Li, et al. (2009). The new graphical method was described in great detail, and an attempt was made to quantify the error sources present. The results of the method were described and discussed, with special emphasis on the remaining areas of exploration needed to turn this approach into a reality feasible for use in commercial systems. Graphical approximation methods for offset curves are relatively new to an ancient field but have the potential to become an elegant, fast, error-bounded method for computation of offset curves in a variety of applications.

BIBLIOGRAPHY

Chiang, C.-S., Hoffmann, C., and Lynch, R. *How to compute offsets without self-intersection*. Purdue University, Dept. of Computer Science: West Lafayette, Ind., 1991.

Choi B. and Park, S. "A pair-wise offset algorithm for 2D point-sequence curve," *Computer-Aided Design*, vol. 31, no. 12, pp. 735-745, 1999.

Choi, B. *Sculptured surface machining : theory and applications*. Dordrecht ;;London: Kluwer Academic, 1998.

Cobb, E. "Design of sculptured surfaces using the B-spline representation," Ph. D., University of Utah, 1984.

Coquillart, S. "Computing offsets of B-spline curves," *Computer-Aided Design*, vol. 19, no. 6, pp. 305-309, 1987.

Elber, G. and Cohen, E. "Error bounded variable distance offset operator for free form curves and surfaces," *International Journal of Computational Geometry and Applications*, vol. 1, no. 1, pp. 67-78, 1991.

Elber, G., In-Kwon, L., and Kim, M.-S. "Comparing offset curve approximation methods," *Computer Graphics and Applications, IEEE*, vol. 17, no. 3, pp. 62-71, Jun. 1997.

Ganesan, M. and Fadel, G. M. "Hollowing rapid prototyping parts using offsetting techniques," *Proceedings of the Fifth International Conference on Rapid Prototyping. Dayton, OH*, 1994.

Goshtasby, A. A. "Grouping and parameterizing irregularly spaced points for curve fitting," *ACM Transactions on Graphics*, vol. 19, no. 3, pp. 185-203, 2000.

Gurbuz, A. "Offsetting operations via closed ball approximation," *Computer-Aided Design*, vol. 27, no. 11, pp. 805-810, 1995.

Hansen, A. and Arbab, F. "An algorithm for generating NC tool paths for arbitrarily shaped pockets with islands," *ACM Transactions on Graphics*, vol. 11, no. 2, pp. 152-182, 1992.

Held, M. "A geometry-based investigation of the tool path generation for zigzag pocket machining," *The Visual Computer*, vol. 7, no. 5-6, pp. 296-308, 1991.

Hoff, K. E., Keyser, J., Lin, M., Manocha, D., and Culver, T., "Fast computation of generalized Voronoi diagrams using graphics hardware," in *Proceedings of the 26th annual conference on Computer graphics and interactive techniques - SIGGRAPH '99*, Not Known, 1999, pp. 277-286.

Hoschek, J. "Spline approximation of offset curves," *Computer Aided Geometric Design*, vol. 5, no. 1, pp. 33-40, 1988.

Hoschek, J. and Wissel, N., "Optimal approximate conversion of spline curves and spline approximation of offset curves," *Computer-Aided Design*, vol. 20, no. 8, pp. 475-483, 1988.

Inui, M. and Ohta, A. "Using a GPU to Accelerate Die and Mold Fabrication," *IEEE Computer Graphics and Applications*, vol. 27, no. 1, pp. 82-88, 2007.

Kimmel, R. and Bruckstein, A. "Shape offsets via level sets," *Computer-Aided Design*, vol. 25, no. 3, pp. 154-162, 1993.

Klass, R. "An offset spline approximation for plane cubic splines," *Computer-Aided Design*, vol. 15, no. 5, pp. 297-299, 1983.

Lai, Y.-L., Wu, J. S.-S., Hung, J.-P., and Chen, J.-H., "A simple method for invalid loops removal of planar offset curves," *The International Journal of Advanced Manufacturing Technology*, vol. 27, no. 11-12, pp. 1153-1162, 2005.

Lee, I., Kim, M., and Elber, G. "Polynomial/Rational Approximation of Minkowski Sum Boundary Curves," *Graphical Models and Image Processing*, vol. 60, no. 2, pp. 136-165, 1998.

Lee, I., Kim, M., and Elber, G., "Planar curve offset based on circle approximation," *Computer-Aided Design*, vol. 28, no. 8, pp. 617-630, 1996.

Li, C. L., Zhou, G., and Chan, C. W. "A Graphical Approach to Approximate Offset Computation," in *2009 Sixth International Conference on Computer Graphics, Imaging and Visualization*, Tianjin, China, 2009, pp. 217-221.

Liu, X., Yong, J., Zheng, G., and Sun J., "An offset algorithm for polyline curves," *Computers in Industry*, vol. 58, no. 3, pp. 240-254, 2007.

Maekawa, T. "An overview of offset curves and surfaces," *Computer-Aided Design*, vol. 31, no. 3, pp. 165-173, 1999.

Maekawa, T., Cho, W., and Patrikalakis, N. M. "Computation of Self-Intersections of Offsets of Bézier Surface Patches," *Journal of Mechanical Design*, vol. 119, no. 2, p. 275, 1997.

Morrow, D. "The 220 Clock," 25-Aug-2008. [Online]. Available: http://www.ldrider.ca/cnc/clock08/clock08.htm. [Accessed: 12-May-2011].

Park, S. "Uncut free pocketing tool-paths generation using pair-wise offset algorithm," *Computer-Aided Design*, vol. 33, no. 10, pp. 739-746, 2001.

Pham, B. "Offset approximation of uniform B-splines," *Computer-Aided Design*, vol. 20, no. 8, pp. 471-474, 1988.

Pham, B. "Offset curves and surfaces: a brief survey," *Computer-Aided Design*, vol. 24, no. 4, pp. 223-229, 1992.

Piegl, L. A. and Tiller, W., "Computing offsets of NURBS curves and surfaces1," *Computer-Aided Design*, vol. 31, no. 2, pp. 147-156, 1999.

Seong, J., Elber, G., and Kim, M. "Trimming local and global self-intersections in offset curves/surfaces using distance maps," *Computer-Aided Design*, vol. 38, no. 3, pp. 183-193, 2006.

Tiller, W. and Hanson, E., "Offsets of Two-Dimensional Profiles," *IEEE Computer Graphics and Applications*, vol. 4, no. 9, pp. 36-46, 1984.

Wallner, J. "Self-intersections of offset curves and surfaces," *International Journal of Shape Modeling*, vol. 7, no. 1, p. 1, 2001.

Willson, F. N. *Theoretical and practical graphics: an educational course on the theory and practical applications of descriptive geometry and mechanical drawing.* The author, 1897.