Automated Edge Grid Generation Based on Arc-Length Optimization

David McLaurin[1] and Suzanne M. Shontz[2]

[1] High Performance Computing Collaboratory, Center for Advanced Vehicular Systems, Graduate Program in Computational Engineering, Mississippi State University, Mississippi State, MS, U.S.A., d.mclaurin@msstate.edu

[2] Department of Mathematics and Statistics, Department of Computer Science and Engineering, Center for Computational Sciences, Graduate Program in Computational Engineering, Mississippi State University, Mississippi State, MS, U.S.A., sshontz@math.msstate.edu

Summary. Computational design and analysis has become a fundamental part of industry and academia for use in research, development, and manufacturing. In general, the accuracy of a computational analysis depends heavily on the fidelity of the computational representation of a real-world object or phenomenon. Most mesh generation strategies focus on element quality–with the justification being that downstream applications require high quality geometries in order to achieve a desired level of accuracy. However, element quality should be secondary to accurately representing the underlying physical object or phenomenon. This work seeks to improve the process of creating a computational model of an object of interest by accelerating the process of mesh generation by reducing the need for (often) manual intervention. This acceleration will be accomplished by automatically generating *optimal* discretizations of curves by minimizing the *arc-length deficit*. We propose a method for generating optimal discretizations through local optimization of the arc length. Our results demonstrate the robustness and accuracy of our optimal discretization technique. We also discuss how to incorporate our edge grid generator into existing mesh generation software.

Key words: mesh, grid, adaptive, refinement, optimal, edge

1 Introduction

Computational design and analysis has become a fundamental part of industry and academia for use in research, development, and manufacturing. In general, the accuracy of a computational analysis depends heavily on the fidelity of the computational representation of a real-world object or phenomenon. However, the task of creating high fidelity models of an actual geometry can be time-consuming–sometimes consuming up to seventy-five percent of the time

required to produce a solution [1]. This work seeks to improve the process of creating a computational model of an object of interest by accelerating the process of mesh generation. In general, a valid volume grid (three-dimensional) is bounded by surface grids (two-dimensional); surface grids are bounded by edge grids (one-dimensional). At the start of the grid generation hierarchy are point spacing values at the end points of analytical or parametric curves which bound the edge grids. Once the bounding surface grid is generated, volume grid generation is, in most cases, a highly automated process. The same generalization can be made for surface grids and edge grids. Algorithms that use automated point creation/insertion for mesh generation of one-, two-, and three-dimensional geometries are ubiquitous [2, 3, 4]. The work here seeks to build on [5] by formalizing the problem description, developing error bounds, and improving the robustness and accuracy of the previously developed algorithms and concepts.

The computation involved with edge grid generation is trivial when compared to volume grid generation – even with high-order NURBS curves. However, the point spacing values at the end points of curves have to be set manually in order to satisfy a desired length scale. This manual process is time consuming. If geometry repair (gluing, trimming, etc) is not considered, the amount of user input required to generate a volume grid can be concentrated on the lowest levels of the grid generation hierarchy – i.e., edge grid generation. In addition, if the edge grids are not generated appropriately then the errors present there, such as overly dense or sparse spacing, will be propagated up the grid generation hierarchy and be present in each subsequent higher-dimensional entity.

The proposed algorithm is a general-use method that can be applied to any "digital curve" regardless of its representation. This is due to every step being developed *without* the use of derivatives. Most other methods operate on a specific type of curve, such as NURBS or B-splines, and use the specific information available for the type of curve in use. NURBS curves are the de-facto standard in CAD; however, in other fields, such as pattern-recognition, other types of digital curves, such as parametric, are more common [6]. T-splines are also becoming more popular in isogeometric analysis, for example [7].

The justification for the development of these methods lies in the need for an automated way of setting point spacing values on curves. Therefore, a general algorithm that does not require derivative information to generate a suitable edge grid has been developed. A result of not using derivatives is that each step in the algorithm is robust to large changes in derivatives or curves that are not "well behaved", e.g., they were highly oscillatory. This process can only be automated if some way of judging "how well" an edge grid represents a curve is present. To this end, a method of generating edge grids through constrained optimization is detailed below. Further discussion of element quality, robustness, and a framework for implementing the information associated with an optimal edge grid into an existing grid generator is also presented. Generating edge grids in a more automated fashion accelerates the

process of surface grid generation—and ultimately volume grid generation. Using our algorithm, or another automated method for setting point spacings does not change the number of steps required for grid generation. However, it does reduce the number of manual steps involved in starting the process.

2 Related Work

In general, grid generation is a name for any process that creates a grid. For example, the advancing-front algorithm advances boundaries into space to generate a grid [8]. Other methods generate grids from iterative refinement or enrichment from initial, coarse configurations [9, 10]. Usually the benchmark for separating the two methods, generation and refinement, are the prioritization of grid quality and grid accuracy (both of these issues will be addressed later). From a standard text [11]: One dimensional grids, or edge grids,

> "...are created using a one-dimensional version of the standard grid generation procedure. This ensures that point distribution and growth rates are fully compatible for optimal final grid quality. For each edge or segment the point spacing is specified at both ends... Edge grid generation is then used to produce the point distribution..."

Traditionally, edge grid generation processes produce good quality grids from the combination of geometric growth rates and smoothing. However, the process requires input: point spacing values. If the point spacing values are not appropriate, then the geometry can be under and/or over sampled for the intended use. That fact is not an indictment of the grid generation process, but instead implies that the final grid is heavily dependent on the inputs. In addition, if some way of controlling the point spacing in the middle of a curve is not present, then more points could be wasted/omitted in an attempt to accurately represent geometry.

Instead of setting appropriate point spacing values and using common grid generation techniques, other efforts have gone into creating a locally or globally "optimal" edge grid. Many names have been assigned to this particular task, but the underlying goal is very similar – represent a curve as accurately as possible – whatever that means for each application. For example, [12] first linearized the interface between curves in order to simplify the process of generating edge grids and surface grids on topologically adjacent patches. Other "geometry aware" or "curvature based" approaches have been developed. One such application is for discretizing curves for use in level set methods [13]. Others include energy minimization [14], curvature minimization [15], and angle minimization [16]. Most need, or are designed to include, the topologically adjacent geometry [17, 18, 19], or only can be applied to a certain class of curves [20].

3 Discretization Error

The accuracy, or discretization error, of a piecewise, linear representation (*discretization*) of an analytical curve (*curve*) in R^3 can be defined in many ways depending on the intended application. The error associated with the *discretization* is discussed in terms of the "deviation" from the *curve* – most often quantified by calculating or approximating the distance from the *curve* for each linear segment in the *discretization*, or the area of the ruled surface between a curve-piece and the segment representing that part of the *curve*. Another way of quantifying the error associated with a *discretization* would be to consider how well it approximates the arc length of the *curve* it represents. In general the arc length is not known *a priori*, but depending on the underlying representation it can be calculated exactly (parametric or analytical) or can be estimated (Bezier). One of the goals of this method was to be "general" in that it should be independent of the underlying geometric representation. Therefore, a method that requires the arc length of the underlying geometry violates the aforementioned concept of "generality" and restricts the applications for which the proposed method could be applied. Some other way of determining/generating an edge grid based on arc length is needed. This process will be detailed later.

Arc-length convergence of a *discretization* is a sufficient condition for other schemes of edge grid generation/refinement. That is: if the difference between the arc length of the *curve* and the sum of the segments in the *discretization* approaches zero then that is sufficient to conclude that the distance between the *discretization* and the *curve* is also approaching zero, also the angles between segments approaches 180 degrees. However, the converse of that statement is not true. The pathological case of a highly oscillatory, low amplitude *curve* approximated by two straight lines (sine-wave approximated by straight lines) shows that a *discretization* of a *curve* can have a small "deviation" or angles between segments but be a poor estimate for arc length. Another pathological case is a "nonconvex" *curve* where the parameterization goes well "outside" of the segment.

4 Discrete Curvature Approximation

The concept of "deviation" as defined above is relatively straightforward and intuitive. However, another related way of describing "how well" a *discretization* represents a *curve* is the degree to which the discrete representation approximates curvature – where curvature is defined as the amount of "bend" in a *curve* or surface, or "how much" a *curve* or surface "differs" from a straight line or plane (words in quotes are subject to gradation). First, however, curvature must be defined in such a way that a discrete approximation is meaningful and appropriate. In relevant literature, there are many ways to estimate curvature [21]. Some of it bears repeating, because it is germane to

what is being discussed here: Consider the following planar *curve*, C, at point P. At a given point P there exists an osculating circle, O, of radius r such that the circle has the same tangent as the *curve* C as well as the same radius of curvature [22].

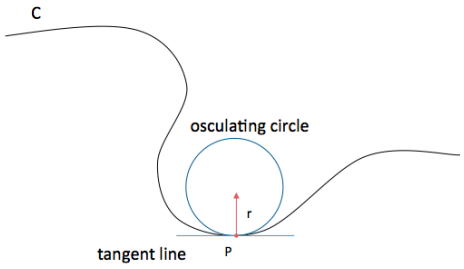

Fig. 1. Osculating Circle of a Planar Curve

Just as the tangent line is the line best approximating a *curve* at a point, the osculating circle is the best circle that approximates the *curve* a point. Ignoring degenerate *curve*s such as straight lines, the osculating circle of a given *curve* at a given point is unique [22]. The radius, r, of the osculating circle at a given point on a *curve* is equal to the radius of curvature, R, which is the reciprocal of curvature, κ–sometimes called the "first curvature" [23]. For a two-dimensional *curve* of the form $y = f(x)$, the curvature equation is:

$$R = \frac{1}{\kappa}, \text{ where } \kappa = \frac{\frac{d^2y}{dx^2}}{[1 + \frac{dy}{dx}^2]^{\frac{3}{2}}}.$$

This quantity, κ, necessarily includes the calculation of derivatives which, depending on the representation of the underlying geometrical description, could be relatively costly. Therefore, this is avoided by defining this radius of curvature on a segment or at a point in the *discretization* without the use of derivatives. This is discussed in the following paragraphs.

A value of curvature can be calculated for each edge in the *discretization* by considering the corresponding osculating circle on a given edge. The osculating circle here (circle, Figure 1) can be approximated by considering the circumscribed circle (circumcircle) [24] defined by the two end points of the edge, P_0 and P_1, and a point, P, between them in the *curve* parameterization (Figure 2) – the radius of the circumcircle will be referred to as the discrete radius of curvature.

Consider a circular segment, which represents the *curve* s. The corresponding chord, which represents a segment in the *discretization*–a, and saggitta–which represents the "deviation" of the segment away from the *curve* h is shown in Figure 3.

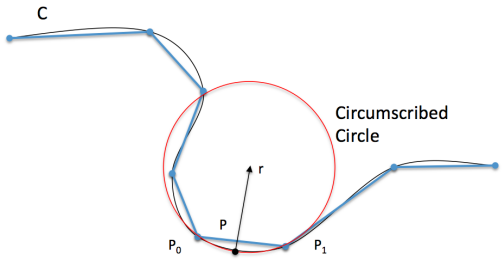

Fig. 2. Osculating Circle of Discrete Edge Grid

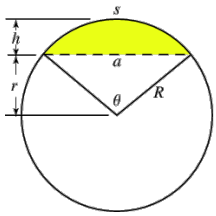

Fig. 3. Circular Segment and Related Geometry [25]

Theorem 1. *As the length of a approaches the length of s, the length of h goes to zero, therefore the radius of the circle, R, goes to infinity.*

Proof. First,

$$a = 2 * \sqrt{R^2 - r^2} = 2 * \sqrt{(h * (2 * R - h)}. \tag{1}$$

Upon rearranging, we obtain

$$R = \frac{(\frac{a}{2})^2 * \frac{1}{h} + h}{2} = \frac{a^2 + 4 * h^2}{8 * h} = \frac{a^2}{8 * h} + \frac{h}{2}. \tag{2}$$

Finally,

$$\lim_{h \to 0} (\frac{a^2}{(8 * h)} + \frac{h}{2}) = \infty. \tag{3}$$

Given the aforementioned proof, if the discrete curvature radius is divided by the length of the corresponding segment on the *curve*, then its value approaches zero. This is a scale-independent measure that converges to a computer representable number as the *discretization* approaches the length of the *curve*. The parameter is known as the curvature ratio. It is an intuitive measure that relates "how far" the *curve* deviates from the segment that is representing it as the ratio of those lengths. Consider a point on a *curve* between two endpoints of a segment of a *discretization*, as seen in Figure 4. The length of the segment, L_i, is the distance from P_0 to P_1. The perpendicular distance between the point on the *curve* between the end points and the segment is L_c. The three points define a curvature ratio through the ratio of L_c to L_i.

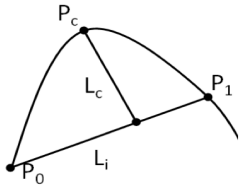

Fig. 4. The definition of *curvature ratio* : $\frac{L_c}{L_i}$ [26]

As shown in [5], deviation-based methods are intuitive and straightforward to implement. However, drawbacks include the fundamental lack of consistently being able to indicate discretization accuracy for curves that are not "well-behaved" between discrete segments.

5 Refinement via Arc-Length Deficit

Mentioned above, a way of determining "how well" a discretization approximates a curve is to consider the difference between arc lengths. This is another way to determine "how well" a *discretization* captures curvature. Locally, it is important for each segment in the discretization to represent the local geometry present in the curve. If a segment is to be subdivided in order to improve the discretization, then it should be subdivided effectively/efficient locally. Also, if the purpose of the refinement process is to minimize the actual arc length minus the discrete arc length, then an optimization problem can be formed where an objective function is minimized as the combined length of the segments in the discretization approaches that of the curve.

Let $C(u)$ be a parameterized curve, and D be a discretization of the curve comprised of n_t points, $P_i : i \in \{1, ..., n_t\}$, and segments, $S_j : j \in \{1, ..., (n_t - 1)\}$. Segment S_j is defined by two successive parametrization values, u_j and u_{j+1}. If $L(S)$ is a function that calculates the length of a segment in (x, y) space then the optimization problem can be stated as:

$$\begin{aligned} \underset{u_i}{\text{minimize}}\ O &= -\textstyle\sum_{j=1}^{n_t-1} L_j \\ \text{subject to } u_1 &= a \\ u_1 &< u_2, \\ u_2 &< u_3, \\ &\vdots \\ u_{n_t-1} &< u_{n_t}, \\ u_{n_t} &= b. \end{aligned}$$

The resulting optimization problem is a mixed integer linear programming problem if both the parameterization values and number of interior points on the curve are unknown. Mixed integer linear programming problems can be

solved by a variety of standard techniques (e.g., [27], [28], or [29]). However, such problems are, in general, NP-hard. Although the analysis and computation are straightforward for a fixed number of interior points, one would need to specify *a priori* this number, which is impractical.

Instead, in order to derive a practical algorithm that controls the number of interior points in the discretization, we add user-defined bounds on the distance between points in the discretization. This turns the optimization problem into a linear programming problem. To this end, let e and m represent user-defined lower and upper bounds on the distance between points in the discretization, respectively. The bounds are easily worked into the set of constraints, where P_i represents a point in non-parametrized, (x, y)-space as follows:

$$\begin{aligned} P_1 &= \alpha, \\ e \leq L_1 &\leq m \\ e \leq L_2 &\leq m \\ &\vdots \\ e \leq L_{n_t-1} &\leq m \\ P_{n_t} &= \beta. \end{aligned}$$

The definition of lower and upper bounds for the distance between points implicitly defines an upper and lower bound for the number of interior points. The implicit definition would be in the form of an over-constrained problem where solutions did not exist for too many or too few points. For instance, too many points could not satisfy the minimum-distance set of constraints, and too few points could not satisfy the maximum-distance set of constraints. However, explicitly determining these bounds for n_t would prove difficult. For example, it could involve repeatedly sampling the curve to determine the maximum number of e-length segments and the minimum number of m-length segments. This would be possible but is inefficient. Another option is to estimate the number of points needed [20]. However, if n_t is to be estimated, then the discretization is not guaranteed to be globally optimal. Therefore, a global optimization problem, while possible, is not very practical in this case. One of the aims of this work is to accelerate the generation of suitable grids for simulation; moving the bottleneck for grid generation to the lowest level in the grid generation hierarchy just increases the amount of time required to generate a grid. The above method does, however, represent a solution to the problem of generating automated, optimal edge grids.

Others have attempted dynamic programming methods for generating "optimal" discretizations for digital curves [30]. However, in general this should prove no more effective than any of the approaches mentioned above. It is true that the problem of generating a discretization to accurately represent a curve exhibits optimal substructure, which is defined where "...an optimal solution can be constructed efficiently from optimal solutions to its subproblems" [31]. However, the number of distinct subproblems available that represent an optimal solution at a defined error bound can be infinite. Therefore,

instead of trying to find an optimal number of nodes required for an optimal discretization (which seems very inefficient), the proposed algorithm will use a divide-and-conquer (recursive) approach to generating an ideal discretization relative to a given tolerance. The combination of the optimized segments represents an optimal discretization for the entire curve.

This would generate an optimal solution using two segments to represent the entire curve – which can be stated another way as maximizing the perimeter of the triangle formed by the existing segment and the two new segments. The above optimization problem could then be applied recursively to each new segment with $n_t = 3$. This process breaks the task of optimizing a discretization for an entire curve into optimizing a simple discretization for smaller section of the curve with the following algorithm. The optimization algorithm described above with $n_t = 3$ for a given segment is:

Algorithm 1 Optimization Algorithm with $n_t = 3$

$n_t = 3$
$u_i : i \in \{1, 2, 3\}$
u_1 and u_3 define the segment $S_{1,3}$
procedure LOCAL OPTIMIZATION($S_{1,3}$)
    $L(S_{1,3}) =$ length of segment
    Place interior point u_2 to maximize $L(S_{1,2}) + L(S_{2,3})$ within *tolerance*
end procedure

Algorithm 2 Optimization Algorithm for Discretization

$D(S_j) : j \in 1$
push S_1 into *list* ▷ *list* is *queue* if breadth-first, *stack* if depth-first
while *list* is not empty **do**
    pop S_j from *list*
    if S_i is optimal **then**
        do nothing
    else
        optimize S_j with Algorithm 1
        push $S_{i,i+\frac{1}{2}}$ into *list*
        push $S_{i+\frac{1}{2},i+1}$ into *list*
    end if
end while

Algorithm 2, often referred to as adaptive refinement or enrichment (see above), would be applied for each segment in the discretization. Also, since the discretization of the curve exhibits optimal substructure, the starting point to the optimization algorithm and the method of refinement or enrichment are irrelevant to the extent to which they prevent an optimal solution from

being generated. However, they obviously contribute to the efficiency of the algorithm.

Now to define the optimization part of the above algorithm: Divide and conquer can be considered to be based on multi-branched recursion. The objects to be constructed at the end of the recursion are the smallest set of segments that approximates the arc length of the curve to a defined precision. This is not quantifiable without *a priori* knowledge of the arc length of the curve. As discussed above, calculating the length of a curve for this application is impractical. So how can the "goodness" of a discretization be measured? At each step, the discretization will be refined on each segment by locally optimizing an objective function analogous to the one developed above. In order to minimize each segment's objective function, arc-length deficit (ALD), then a point P has to be placed on the curve on the segment such that the new sum of the arc lengths is changed maximally. Since this entire project is to be done without calculating derivatives (see reasons above), the optimization scheme chosen here is not given access to derivative information either. Since the objective function is a non-negative planar curve, $(O : C \rightarrow ALD)$, any line search method of optimization could be used. However, the method cannot have any requirements on differentiability due to the possibility that the derivative of O could be discontinuous.

The golden section search method is implemented here, since, unlike the bisection method, it meets all of the above criteria and has the possibility to converge superlinearly [32]. Alternatively, a pattern search [33], simplex [34, 35], or interior point [36] method could be used. If the length of each segment locally approaches the portion of the curve it represents (i.e., the local objective function is minimized), then the global length of the discretization approaches the global length of the curve (a restatement of the property of optimal substructure). Also, since the optimization algorithm for each segment is only concerned about the portion of the curve it represents, then this method exhibits scale-independence, which was one of our requirements.

Recursive algorithms require stopping criteria. In this case the stopping criteria should not permit the method to infinitely subdivide the curve. For instance, the aforementioned minimum and maximum segment lengths can be used (and were implemented here). Even though using a minimum edge length would prevent the infinite subdivision of the curve, another criteria is needed such that the minimum segment length is not needed to satisfy the criteria. This stopping criterion could be in the form of a delta-segment length. That is, if the new segments' combined length is below a defined fraction larger than the existing segment then it should not be subdivided. This is a "pure-greedy" method of subdivision, in that it does not consider the rest of the "solution" when deciding to stop. One problem with this set of stopping criterion is immediately apparent: the "large" segments could potentially not be subdivided because locally it is not justified–even if the subdivision of the large segment would cause a global change in the length of the curve that is significant. This value, global delta-segment, would have to be smaller than

the one used locally for each segment; otherwise, it would have no effect. Therefore, an additional criterion is needed to determine if a segment should be subdivided: if the total change in length of the discretization would be changed by a defined fraction then it should be subdivided. The addition of this last subdivision criterion makes the method "less-greedy". This set, minimum segment length, maximum segment length, local delta-segment, and global delta-segment define a robust, minimum set of criterion needed for generating an optimum solution to the problem of representing a curve via arc-length deficit.

6 Error Bounds

Since the optimization function for each curve is not given access to derivative information, it is conceivable that it would not find the optimal value and instead converge to a local minimum. However, the problem of escaping local minima is common to all optimization problems. The addition of the conditional that states: "refine a segment if the refinement changes the length of the entire discretization by more than an epsilon" was deliberately included to lessen the chance that a segment would not be refined when it was prudent to do so. If the method succeeds in finding the global minimum for each segment, then the error bound will be on the order of the segment length. However, when the method fails to do so, or chooses a local minimum instead, there is no formal way to express the error as a function of arc-length-deficit–since there is no information about what the global minimum might be (without explicitly calculating the length of the curve/segment). Therefore, the error can only be quantified for when the method has succeeded in finding the minimum for each segment.

Arc-length deficit is a single-valued function on the curve. The obvious problem with this single-valued function is that the actual arc length of the curve is never known and can therefore not be compared to the arc length of the segments. How then can error be quantified? The error bounds could be detailed for unimodal pieces of the curve–those where the ALD function has one peak. However, for segments that do not have a unimodal distribution of the ALD function on the local curve segment, the error estimation is not straightforward. In fact, it is no longer possible to determine what the bound for the arc-length deficit error is. However, we can state some observations about the geometry related to these configurations:

Assume that the optimization function on a general segment found the global minimum, i.e., maximized the change in edge length for the new combined segments, for the segment and corresponding curve piece. With the given geometry, an ellipsoid can be formed with the endpoints of the segment, F_1 and F_2, as the foci and the semi-major and semi-minor axes are defined implicitly by the new segments connecting the new point with the endpoints of the current segment, r_1 and r_2. "An ellipse is a curve that is the locus of

all points in the plane the sum of whose distances r_1 and r_2 from two fixed points F_1 and F_2 (the foci) separated by a distance of $2c$ is a given positive constant $2a$" [37].

Theorem 2. *With the above assumption and definition, the entirety of the curve represented by the segment must lie within the prolate spheroid formed by the geometry below in Figure 5.*

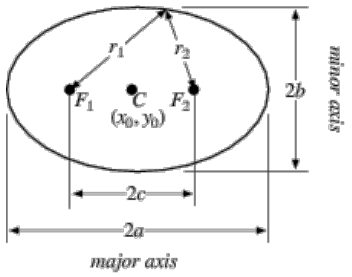

Fig. 5. Ellipse Geometry [25]

Proof. If the perimeter is maximized keeping $2c$ a constant, $(r_1 + r_2)$ is maximized *because* $2c$ is a constant. Maximizing $(r_1 + r_2)$ maximizes a from the following relation with $2c$ being constant: $r_1 + r_2 = 2a$. Combining the following relation, $b^2 = a^2 + c^2$, with the area of the ellipse, $A = \pi * a * b$, yields $A = \pi^2 * a^2(a^2 - c^2)$. If c is a constant and a is maximized, then the maximal area is obtained from maximal $(r_1 + r_2)$. Which means the curve must be inside of the spheroid defined by the ellipse. If the curve is not inside the spheroid then there is a point on the curve such that $(r_1 + r_2)$ is larger and therefore the area is larger and therefore the volume is larger which means that $(r_1 + r_2)$ was not maximized.

Observe that there was no mention of the length of the curve inside of the spheroid, or the ruled area that the segment and curve could define. This is because there is no way this information can be known (without explicitly calculating the length of the curve/segment). It is unfortunate that there is no way to quantify the discretization error of a curve except in terms of volume of the spheroids defined by each segment. This is nonintuitive, but no more specificity is possible. The volumes would also be scale-dependent and offer no insight into how well the discretization approximates the curve – without some context.

7 Growth Ratio

Our goal was to develop an automated edge grid generator that accurately represents the underlying geometry. For this expressed purpose of representing

the curve to a desired tolerance, the grid quality is not a concern. However, for most applications, the grid quality directly affects downstream analyses.

Therefore, we explain how to modify our optimization problem in order to yield an edge grid that both accurately represents the underlying geometry and is of sufficient quality. Edge grid quality is typically defined in terms of the growth ratio, i.e., the length of a segment divided by the length of a topologically adjacent segment. If the growth ratio strongly deviates from unity, the sizes of neighboring elements are not similar which leads to a large disparity in the sizes of the surface and volume elements which are generated. Such disparate length scales can cause problems for numerical partial differential equation methods.

Traditionally edge grids are generated with an *a priori* defined growth ratio along with other parameters that ensure that the grid has good quality. Upper and lower bounds can be included for the grid quality (growth ratio) as nonlinear constraints in the optimization problem. However, in this case, the number of constraints grows very quickly, albeit linear in the number of grid points. Alternatively, minimum and maximum growth rates could be enforced by the optimization procedure by splitting an edge if the growth rate is too large or small. *A posteriori* methods for quality control could include some type of smoothing or optimization [38, 39, 40, 41].

One final method is to use the output from the edge grid generator, the "optimal" grid, as input for a grid generator, which presumably has strict quality control measures in place. This would be accomplished by using the point spacing values present at the end points of the discretization at the end points of the curve. The resulting edge grid from the grid generator could then be analyzed for the purpose of determining how far it deviates from "optimality" in the interior of the curve. If the deviation is too large, a point spacing source could be inserted to adjust and control the point spacing as desired during grid generation.

8 Experimental Results

Three curves were chosen to demonstrate the aspects of the developed methods. The first is a family of curves, *Lissajous* curves (Figure 6), which are a combination of two perpendicular harmonic oscillations. This curve was chosen due to the sharp changes in curvature and self-intersecting nature, which are present in real-world applications. The second curve, a tricuspoid (Figure 7) was chosen for the sharp, discontinuous features, which are also present in real-world applications. In Figures 6 to 8 the curve is shown in red, and the discretization is shown in black with vertices highlighted by circles indicating their position. Each curve was scaled so that the parametrization, t, was normalized between zero and unity. In each case the curve was originally discretized using one segment corresponding to a vertex located at $t = 0$ and

$t = 1$. Once the original discretization is created, the discretizations are refined using Algorithm 2. Further results can be found in Table 1. In this table, the true lengths of the curves can be found. The combined length of the segments in each discretization are also given with the actual arc-length deficit. As stated earlier the chief goal of the developed algorithms was to accelerate the grid generation process. The presented results were generated in no longer then 0.004 seconds for any curve or convergence criteria.

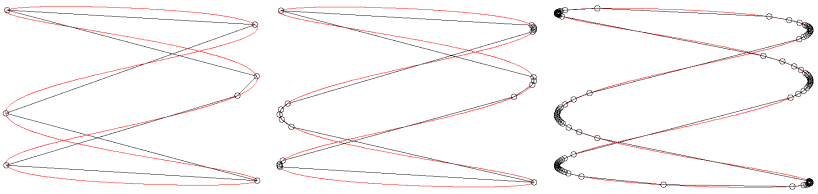

Fig. 6. Lissajous Curves: 10% deficit (left), 1% deficit (middle), 0.1% deficit (right); $x(t) = a * \sin(n * t + c)$, $y(t) = b * \sin(t)$, $a = b = c = 1$, $n = 3$, $0 < t < 2\pi$

For the Lissajous curve, the optimization can be seen to be effective and efficient with regards to only generating vertices where the curvature is high and not wasting vertices on the relatively "straight" portions of the curve. In addition, the self-intersection present on this curve did not impede the generation of an optimal edge grid.

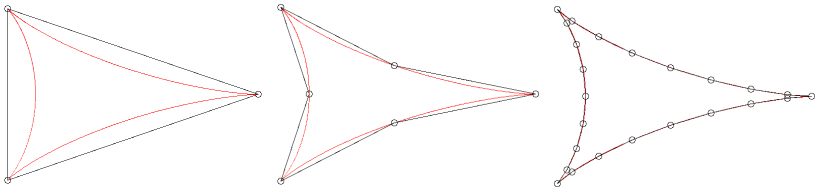

Fig. 7. Tricuspoid Curve: 10% deficit (left), 1% deficit (middle), 0.1% deficit (right); $x(t) = a * (2 * \cos(t) + \cos(2 * t))$, $y(t) = a * (2 * \sin(t) - \sin(2t))$, $a = 1$, $0 < t < 2 * \pi$

For the Tricuspoid curve, the algorithm for optimal point placement can be seen to be accurate with regards to placing a vertex at the discontinuities. Placing a vertex at the discontinuity is efficient since no further nodes are required to capture that feature of the curve. It can be seen that further refinements are placed elsewhere in order to capture curvature.

For the Cochleoid curve, an interesting feature stands out: when using 10% as the ALD, a self-intersecting discretization was generated where the curve does not exhibit self-intersection. This is due to the rapid change in curvature near the center of the spirals. In general, this cannot be avoided since *a priori*

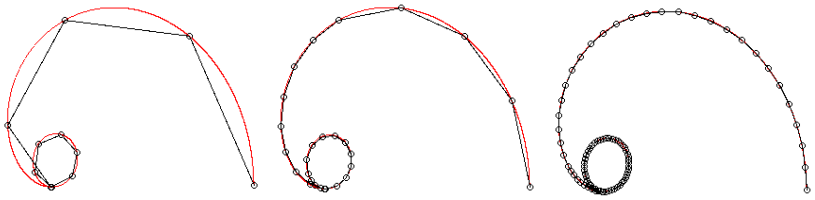

Fig. 8. Cocheloid Curve: 10% deficit (left), 1% deficit (middle), 0.1% deficit (right); $x(t) = 2 * t + 3 * \sin(7 * t)$, $y(t) = t + 8 * \cos(3 * t)$, $0 < t < 1$

knowledge of where the curve is self-intersecting would be needed to refine the discretization where appropriate. When refined the result is valid. The results from using 1% and 0.1% ALD can be seen to accurate and increase in resolution where the curve exhibits changes in curvature. On the outer portions of the spiral, the discretization is less refined than near the center of the spirals.

Table 1 and Table 2 summarize the results from discretizing the three curves. Each row in Table 1 shows the results from a particular percentage change in edge length that was used as the refinement constraint. In each row the discretization length of each curve is shown along with the arc-length deficit relative to the true length of the curve and number of segments in parenthesis. Each result is less than the desired arc-length deficit for the entire curve – except for 1% result for the *Lissajous* curve which is slightly higher. Each row in Table 2 shows the number of segments corresponding to each discretization and the number of function evaluations. Other test results run by the authors showed similar results in accuracy and robustness. It should be noted that no effort was made to prematurely optimize the number of function evaluations, e.g., caching. Algorithmic optimization proved to not be needed due to the extremely low computational cost of the existing implementation.

Table 1. Discretization length with respect to true curve length, arc-length deficit

	Lissajous	Tricuspoid	Cochleoid
True Length	13.0653	16.0	2.94
10% deficit	12.7123 (2.7%)	15.5885 (2.5%)	2.7916 (5.07%)
1% deficit	12.884 (1.4%)	15.87 (0.78%)	2.916 (0.0842%)
0.1% deficit	13.04 (0.166%)	15.99 (0.058%)	2.939 (0.076%)

9 Conclusions and Future Work

An algorithm for edge grid discretization through local optimization was developed. In an effort to accelerate the process of grid generation, minimal

Table 2. Number of segments in final discretization, function evaluations

	Lissajous	Tricuspoid	Cochleoid
10% deficit	8, 1599	4, 615	11, 2337
1% deficit	21, 4797	7, 1353	27, 6273
0.1% deficit	121, 29397	25, 5781	86, 20787

user input is required for the developed method: a single parameter which is used as a limit for local refinement. The results show that the generated edge grid is optimal with respect to arc-length deficit. In addition, the process was shown to be robust to discontinuities, abrupt changes in curvature, and self-intersections. Results were shown here in two dimensions for ease of presentation. The developed algorithm is easily abstracted to three dimensional curves through a change in the kernel for edge length calculation.

Future work will include a comparison to a global optimization problem formulated with the presented constraints and an additional constraint of a given number of edge grid points. Grid quality measures will also be included in the optimization problem via *a priori* quality constraints.

More work will also be done to abstract the problem from strictly one-dimensional simplices (edge grids) to two-dimensional simplices (triangles). While it was straightforward to determine which part of a curve an edge grid represents, it is non-trivial to determine which part of a surface a triangle represents. The development of a map between the planar elements representing a surface and the underlying geometry would be one of the chief tasks moving forward. Additionally, the edges, as well as the triangles, in the discretization must considered when optimizing the surface grid.

Finally, we will apply our edge and surface grid generation routines on problems stemming from real-world applications, including those from mechanical engineering and medicine. One challenge that will need to be faced is the development of a surface grid generator which can develop an optimal representation of a surface from noisy data, e.g., medical imaging. An engineering application would be to accelerate the mesh generation process for fluids simulations by automatically generating surface meshes that capture local geometry.

10 Acknowledgments

The work of the second author is supported in part by NSF CAREER Award ACI-1330056 (formerly OCI-1054459).

References

1. S. Bischoff and L. Kobbelt, "Structure preserving CAD model repair," *Eurographics*, vol. 24, no. 3, 2005.

2. "CUBIT: Geometry and mesh generation toolkit." `http://cubit.sandia.gov`.
3. S. Cheng, T. Dey, and J. Shewchuk, *Delaunay Mesh Generation*. CRC Press, 2012.
4. D. Marcum, "Advancing-front/local-reconnection (ALFR) unstructured grid generation," in *Computational Fluid Dynamics Review*, p. 140, World Scientific-Singapore, 1998.
5. D. McLaurin, "Automated, curvature based edge-grid generation," in *Proc. of AlaSim 2012*, (Huntsville, AL), Alabama Modeling and Simulation Council, 2012.
6. M. Sarfraz, *Interactive Curve Modeling: With Applications to Computer Graphics, Vision, and Image Processing*. Springer-Verlag, 2008.
7. T. Hughes, J. Cottrell, and Y. Bazilevs, "Isogeometric analysis: CAD, finite elements, NURBS, exact geometry, and mesh refinement," *Comput. Methods Appl. Mech. Engrg.*, vol. 194, pp. 4135–4195, 2005.
8. R. Löhner and P. Parikh, "Generation of three-dimentsional unstructured grids by the advancing-front method," *Int. J. Numer. Meth. Fl.*, vol. 8, pp. 1135–1149, 1988.
9. J. Shewchuk, "Tetrahedral mesh generation by delaunay refinement," in *Proc. of the* 14^{th} *Annual Symposium on Computational Geometry*, pp. 86–95, 1998.
10. J. Shewchuk, "Delaunay refinement algorithms for triangular mesh generation," *Comput. Geom. Theory Appl.*, vol. 22, pp. 21–74, 2002.
11. J. Thompson, B. Soni, and N. Weatherill, eds., *Handbook of Grid Generation*. CRC Press, 1998.
12. P. Laug and H. Borouchaki, "Curve linearization and discretization for meshing composite parametric surfaces," *Commun. Numer. Meth. En.*, vol. 20, pp. 869–876, 2004.
13. P. Macklin and J. Lowengrub, "An improved geometry-aware curvature discretization for level set method: Applications to tumor growth," *J. Comput. Phys.*, vol. 215, pp. 392–401, 2006.
14. M. Hofer and H. Pottmann, "Energy-minimizing splines in manifold," *ACM Transactions on Graphics (TOG) - Proc. of ACM SIGGRAPH 2004*, vol. 23, pp. 284–293, 2004.
15. N. El-Zehiry and L. Grady, "Fast global optimization of curvature," in *Computer Vision and Pattern Recognition, IEEE*, (Princeton, NJ), pp. 3257–3264, 2010.
16. M. Ebeida, R. Davis, and R. Fruend, "A new fast hybrid adaptive grid generation technique for arbitrary dimensional domains," *Int. J. Numer. Meth. Eng.*, vol. 84, pp. 305–329, 2010.
17. W. Quadros, S. Owen, M. Brewer, and K. Shimada, "Finite element mesh sizing for surfaces using skeleton," in *Proc. of the* 13^{th} *International Meshing Roundtable*, 2004.
18. J. Cabello, "Towards quality surface meshing," in *Proc. of the* 12^{th} *International Meshing Roundtable*, 2003.
19. A. Cunha, S. Canann, and S. Saigal, "Automatic boundary sizing for 2d and 3d meshes," in *Proc. of the* 6^{th} *International Meshing Roundtable*, 1997.
20. J. Cuilliere, "A direct method for the automatic discretization of 3D parametric curves," *Comput.-Aided Des.*, vol. 29, pp. 639–647, 1997.
21. S. Hermann and R. Klette, "A comparative study on 2D curvature estimators," in *Proc. of the International Conference on Computing: Theory and Applications (ICCTA)*, pp. 584–589, 2007.

22. A. Gray, E. Abbena, and S. Salamon, *Modern Differential Geometry of Curves and Surfaces with Mathematica*, pp. 111–115. Boca Raton, FL: CRC Press, 1997.
23. E. Kreyszig, *Differential Geometry*, pp. 34–36. Dover, NY: Dover Books on Methematics, 1st ed., 1991.
24. J. Casey, ed., *A Sequel to the First Six Books of the Elements of Euclid, Containing and Easy Introduction to Modern Geometry with Numerous Examples*. Dublin: Hodges, Figgis, & Co., 5^{th} ed., 1888.
25. E. Weisstein, "Ellipse." `http://mathworld.wolfram.com/Sagitta.html`. Online Knowledge Base.
26. D. McLaurin, "Automated point spacing." `http://www.simcenter.msstate.edu/docs/solidmesh/automatedpointspacingreport.pdf`, 2010. Online Tutorial.
27. G. Nemhauser, M. Savelsbergh, and G. Sigismondi, "MINTO, a Mixed INTeger Optimizer," *Oper. Res. Letters*, vol. 15, pp. 47–58, 1994.
28. "MOSEK homepage." `www.mosek.com/index.php`.
29. T. Ralphs and M. Güzelsoy, "The Symphony callable library for mixed integer programming," in *The Next Wave in Computing, Optimization, and Decision Technologies, Operations Research/Computer Science Interfaces Series*, vol. 29, pp. 61–76, 2005.
30. J. Horng and J. Li, "An automatic and efficient dynamic programming algorithm for polygonal approximation of digital curves," *Pattern Recogn. Lett.*, vol. 23, pp. 171–182, 2002.
31. T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*. MIT Press, 2nd ed., 2001.
32. R. Brent, *Algorithms for Minimization Without Derivatives*. Englewood Cliffs, NJ: Prentice-Hall, 1973.
33. "Hybrid optimization parallel search PACKage home page." `https://software.sandia.gov/trac/hopspack/wiki`.
34. G. Dantzig and M. Thapa, *Linear Programming 1: Introduction*. Springer-Verlag, 1997.
35. G. Dantzig and M. Thapa, *Linear Programming 2: Theory and Extensions*. Springer-Verlag, 2003.
36. N. Karmarkar, "A new polynomial time algorithm for linear programming," *Combinatorica*, vol. 4, no. 4, pp. 373–395, 1984.
37. E. Weisstein, "Ellipse." `http://mathworld.wolfram.com/Ellipse.html`. Online Knowledge Base.
38. L. Freitag and P. Knupp, "Tetrahedral mesh improvement via optimization of the element condition number," *Int. J. Numer. Meth. Eng.*, vol. 53, pp. 1377–1391, 2001.
39. T. Munson, "Mesh shape-quality optimization using the inverse mean-ratio metric," *Math. Program.*, vol. 110, pp. 561–590, 2007.
40. J. Kim, T. Panitanarak, and S. Shontz, "A multiobjective mesh optimization framework for mesh quality improvement and mesh untangling," *Int. J. Numer. Meth. Eng.*, vol. 94, pp. 20–42, 2013.
41. S. Sastry, S. Shontz, and S. Vavasis, "A log-barrier method for mesh quality improvement and untangling," *Eng. Comput.*, 2012. Published online ahead of print.