Computing Surface Offsets and Bisectors Using a Sampled Constraint Solver

David E. Johnson* Elaine Cohen†

School of Computing
University of Utah

ABSTRACT

This paper describes SCSolver, a geometric constraint solver based on adaptive sampling of an underlying constraint space. The solver is demonstrated on the computation of the offset to a surface as well as the computation of the bisector between two surfaces. The adaptive constraint sampling generates a solution manifold through a generalized dual-contouring approach appropriate for higher-dimensional problems. Experimental results show that the SCSolver approach can compute solutions for complex input geometry at interactive rates for each example application.

Index Terms: I.3.5 [COMPUTER GRAPHICS]: Computational Geometry and Object Modeling—Geometric Algorithms

1 INTRODUCTION

Surface offsets and bisectors are examples of an important class of geometric problems that can be solved by finding the solutions to a set of equations representing constraints on the space of possibilities. This solution paradigm is followed by constraint solvers, which given some input data and a set of equations, generate the solution set.

Recent work by Elber and Kim [4, 11] and others [21, 19, 2] has extended earlier work [22, 16] in developing a general purpose constraint solver based on representing non-linear constraint equations as multi-dimensional parametric hyper-volumes. Symbolic operators on smooth models can build up exact, explicit representations of these constraints, and adaptive refinement combined with numerical methods can extract the zero sets. However, while this explicit representation provides the basis for robust operation, the size of the constraint representations also limits the complexity of problems that can be solved and problem solution speed.

In response to these issues, this paper demonstrates a new approach for constraint solvers based on adaptive sampling of the underlying constraint equations. This paper will refer to this approach as a sampled constraint solver, or SCSolver. Sampling the set of constraints, rather than explicitly building them, provides several advantages over current approaches. By avoiding an explicit representation of the constraint equations, problem setup time and the memory footprint of the solver is reduced. Additionally, as long as the problem input representation supports the necessary queries, it can be used in the SCSolver framework, expanding the applicability of the approach to different model representations. Finally, the SCSolver generates a set of solution points that meet the constraints within numerical tolerance and reconstructs a piecewise linear surface from those points. Other discrete approaches, such as GPU-based gridded sampling [24], require very high resolutions to meet those tolerances.

*e-mail: dejohnso@cs.utah.edu
†e-mail: cohen@cs.utah.edu

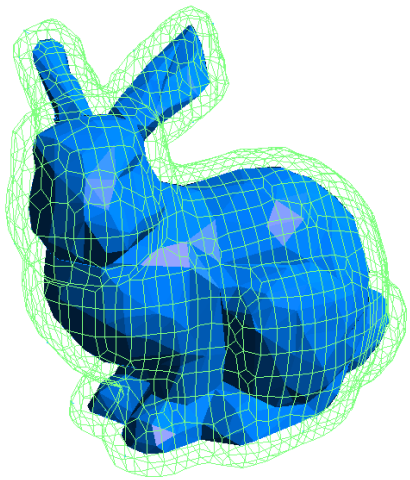

Figure 1: An example offset surface generated by the SCSolver approach. The offset distance can be adjusted interactively and smoothly animated.

This SCSolver approach is demonstrated on two classical geometric computing problems: finding the offset surface of a model (see Figure 1) and computing the bisector surface between two models. Each of these problems is an important application and research topic on its own. Offset surfaces are used in surface reconstruction [23], model simplification [3], CNC machining [15], and geometric modeling [25]. Bisector surfaces have widespread application in modeling and engineering, in robot motion planning [6], and in computing the Voronoi diagram [18]. Other symbolic problems are solvable within the system as long as an appropriate distance metric can be posed within the parameter space. The SCSolver approach generates high-resolution piecewise linear approximations of solutions to these problems at interactive rates.

The contributions of this paper are:

- A constraint solver system for geometric problems based on efficient, adaptive sampling of the constraint space.
- A generalization of the dual-contouring isosurfacing approach suitable for extracting different dimensional manifolds from arbitrary dimensional implicit volumes.
- New, interactive solutions to computing offset surfaces and bisector surfaces for polygonal data.

2 RELATED WORK

The SCSolver approach draws upon several current research directions. Its adaptive sampling of the constraint space is based on a generalized octree data structure. The overall constraint approach shares common goals with other constraint solvers. The solution

manifold reconstruction is inspired by dual-contouring methods that been successfully applied to implicit surface polygonalization, but is generalized here to higher-dimensional problems. These different areas will be summarized below, as well as other approaches to offset and bisector computations.

2.1 Octree Data Structures

Octrees were introduced to computer graphics and image processing in response to the large memory requirements used by regular, gridded sampling of data needed to produce high-fidelity results [8]. More recent work by Frisken [7] adaptively sampled the distance fields around models using an octree data structure to use as a generalized model representation. That work also showed how to perform various modeling operations on the octree and how to produce surface reconstructions from the volumetric implicit function defined by the distance field.

The SCSolver uses a generalized octree structure that only splits a parent along the longest dimension of the cell, an approach shared by [4]. This binary splitting scales naturally with problem dimension, as it just makes the tree of cells taller, not exponentially wider.

2.1.1 Constraint Solvers

The solution to some important geometric problems can be reformulated to be the solution to an equivalent set of nonlinear equations. For example, suppose $F(u,v) = (x_f(u,v), y_f(u,v), z_f(u,v))$ and $G(r,s) = (x_g(r,s), y_g(r,s), z_g(r,s))$ are two parametric surfaces. Then, their intersection occurs at the set of points $\{(x,y,z) \mid F(u,v) = (x,y,z) = G(r,s)\}$, which can be explicitly written as three nonlinear equations in four unknowns:

$$\begin{aligned} x_f(u,v) - x_g(r,s) &= 0, \\ y_f(u,v) - y_g(r,s) &= 0, \\ z_f(u,v) - z_g(r,s) &= 0. \end{aligned}$$

These three constraints reduce the dimensionality of the solution to be a one-manifold in the four-dimensional *uvrs*-parameter space. This zero-set manifold can be evaluated to form corresponding intersection curves in the *xyz* space. B-splines support a variety of mathematical operations, such as addition, dot product, cross product and derivatives, that allow these constraints to be reformulated as a system of multivariate B-splines, given B-spline input models. Adaptive parameter space techniques [16, 4] can then adaptively prune the large constraint hyper-surfaces until a leaf refinement level is reached. Numerical methods improve the precision of the solution by finding a zero in each leaf cell. When the solutions are higher-dimensional entities, such as surfaces, a modified dual-contouring method is used to extract a linear approximation to the continuous zero set manifold.

While these approaches are quite robust and general, they scale poorly with problem complexity and dimensionality. Furthermore, symbolic computation of the constraint equations can be costly, and these computations must be redone for each new query. The cost of building the constraint equations, only to prune much of it away without detailed examination, motivates this paper's approach.

2.2 Manifold Solution Reconstruction

The extraction of a linear approximation to the continuous zero set solution can be thought of as surface reconstruction from an implicitly defined volume, where the value of the constraint equations define the implicit. This is a heavily studied problem with application to surface reconstruction from volumetric medical data or from point cloud models, implicit modeling, and volume visualization. A classic surface reconstruction algorithm is Marching Cubes [13], which examines the value of the implicit volume at the vertices of cells.

More recently, dual contouring [10] has been used to preserve sharp features in the data and to more easily work with adaptive structures such as octrees. Dual contouring places a solution point in each cell that should have a solution. Quadrilaterals are formed around edges that contain a zero crossing, since the solution surface must be pierced by these edges.

2.3 Offsets and Bisectors

Offset surfaces are formed by projecting points on the defining model out along their surface normals for a specified distance. The resulting offset is everywhere a fixed distance from the original model. These two views of an offset also provide the basis for two main approaches for computing the offset. Projection-based approaches move defining elements of the model along their normals. Mathematically, this can be described for a parametric curve $C(t)$ and offset d as

$$O_d(t) = C(t) + dN(t) \tag{1}$$

where $N(t)$ is the unit normal along the curve. For non-parametric models, this explicit offset definition can be approximated by geometric projection operations. A good example of this approach is by Breen et al. [1], which swept triangles out from the surface. A difficulty of this approach is the need to trim these swept volumes against each other to prevent "swallowtails", which are locally correct, yet globally incorrect solutions to the offset.

A B-spline constraint solver approach was successfully used to trim swallowtails from curve offsets in [20]. Higher-dimensional critical point analysis was used to more accurately place the solutions at offset cusps.

Another approach searches for the set of points that satisfy an implicit definition of the offset to a model M, as in

$$O_d = \{P_{x,y,z} | P_{x,y,z} \in D \wedge dist(P_{x,y,z}, M) - d = 0\}. \tag{2}$$

In this case, D is the search domain of interest. This approach is typically implemented and approximated through computation of the distance field around the model and extracting the appropriate iso-surface corresponding to the distance d. A fast graphics card approach is demonstrated in [24] and a point-sampled volume approach is used for CAD models in [14]. Interval arithmetic was used to adaptively search for planar curve offsets and bisectors in [9].

A more recent approach [17] has some similarities to the approach proposed in this paper. It uses distance bounds on an adaptively sampled octree with solution reconstruction to create high-quality offsets. However, it uses a feature-based implicit distance field and uses bounds on cell distance to these volumetric primitives, rather then the underlying mesh. The result is an adaptively-sampled, high-quality solution, however, the system runs in 10's of seconds, rather than interactively like the system presented here.

3 Research Approach

The SCSolver development has been motivated by the lack of interactive constraint solvers. Interactivity can play a critical role in the usefulness of a tool. Another factor guiding this development is the need to include geometric data that cannot directly generate B-spline constraints, such as commonly used polygonal data. The sampling operation allows any model type that supports the sampling queries to be used in the solver.

First, this paper will describe the general SCSolver algorithm and then additional sections will describe some initial applications.

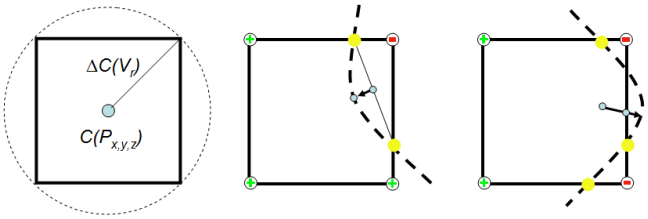

Figure 2: (a) A 2D illustration of a cell being tested for a possible solution using the cell radius as a bound on changes to the constraint value. (b) A solution point is found by finding zero crossing (in yellow) along edges. The average of these points provides an initial guess which then moves onto the solution set. (c) Some mass points provide poor initial guesses, and the solution point is clipped to remain in the cell bounds.

3.1 The Sampled Constraint Solver

The first task in running the solver is to define a domain of interest for the problem. This initial domain becomes the top node of the generalized octree data structure used to efficiently search the constraint space. Given this node, the SCSolver does a recursive, adaptive search for nodes below some size or error tolerance that contain a point on the solution manifold. The cells of the generalized octree are nodes of the tree. Cells containing solutions points are leaves of the tree.

The adaptive search is more precisely described with some pseudocode. The following example is actually very close to the actual code at the heart of the solver.

```
function SCSolver( cell )
{
  if cell.MayHaveSolution()
  {
     if (cell.Leaf( cell ))
     {
         cell.FindPointSolution
         cell.AddPointToSolution
     }
     else
         SCSolver( cell.Subdivide )
  }
}
```

Once the SCSolver is run, the solution manifold is extracted from the set of octree leaf nodes. Each component of the algorithm is described in the following subsections.

3.1.1 Testing for Possible Cell Solutions

Each cell may or may not have a portion of the zero set solution within the cell domain. Often in gridded sampling of volume data, the corners of each cell are evaluated. Changes in sign between the corners of the cell indicate a solution lies within. However, this approach misses solutions completely contained within the cell, as well as solutions that penetrate the faces of the cell rather then the edges.

Instead, the SCSolver evaluates the set of constraint equations at the center of a cell. If the constraint equations cannot change enough to generate a zero solution while moving within the cell domain, then a solution cannot exist there. The possible change of value within a cell is conservatively bounded by a cell radius, which is the distance from the cell center to a corner (Figure 2(a)). For a constraint $C(P_{x,y,z}$, the test for a possible zero within a cell volume V and cell radius V_r is

$$\begin{aligned} possibleZero \quad = \quad & C(P_{x,y,z}) + \Delta C(V_r) \geq 0 \text{ and} \\ & C(P_{x,y,z}) - \Delta C(V_r) \leq 0 \end{aligned}$$

In general, it is difficult to create the metric ΔC relating change in position within a cell to change in constraint value, although it is trivial for offsets and bisectors. For more complex cases, the robotics community has some approaches for computing these metrics by providing conservative bounds, for example in the case of moving linkages with revolute joints [5]. That general bounding approach seems applicable to a variety of symbolic operations, however, efficient computation of general metrics remains an area of future research.

3.1.2 A Cell Leaf Test

In general, a cell is declared a leaf when the dimensions of the cell are considered "small enough". This can be a simple, user-provided scalar value, or based on some criteria from the problem. A user-provided scalar gives control over the polygon count of the resulting solution.

3.1.3 Finding the Cell Solution Point

Given a cell containing a possible solution, it is necessary for the solver to compute a representative point to use later with the dual-contouring solution reconstruction. Dual-contouring approaches have illuminated several potential problems that can occur while solving for this representative point. A typical approach may be to start with the cell center as an initial guess and to project onto the zero set through numerical or iterative methods, but the nearest portion of the solution manifold may lie out of the cell. Solution points that exit the cell can cause flips and concavities in the resulting solution surface reconstruction.

Instead, the concept of the *mass point* is adapted from dual-contouring. Each edge with a vertex sign change computes an approximate edge point through linear interpolation of the vertex scores. The normalized sum of these edge points is the mass point. The mass point must lie within the cell and should be near the unknown constraint solution (Figure 2(b)). The mass point is then used to initialize an iterative error minimization method that moves the point onto the zero level-set. The method used is particular to the constraint system, although general purpose numerical methods are appropriate as well. A problem is that at coarse resolutions, the cell may poorly approximate the underlying zero set, and the mass point does not provide an appropriate initial point. In this case, the solution point is clipped against the cell boundaries, introducing some small, bounded error, but preserving the topology of the solution surface (Figure 2(c)).

3.1.4 Adding the Point to the Solution Set

Points that are accepted as part of the solution set get inserted into a vertex-edge-quad data structure. The solution point becomes a vertex in the solution manifold. These vertices point back into the octree data structure to preserve information about where the solution manifold crosses into neighboring cells.

3.1.5 Cell Subdivision

The SCSolver uses a binary split approach for generating children nodes of a cell. The lengths of the cell are found in all dimensions. The longest axis is split in half, producing two children nodes. This differs from the octree data structure used in standard dual-contouring. The next section discusses approaches for extracting a zero-set surface from this generalized octree.

3.1.6 Solution Manifold Extraction

Given a set of solution points and their containing cells, the SCSolver uses a modified dual-contouring approach to extract out surface quadrilaterals. In standard dual-contouring, a recursive approach finds neighboring cells until four cells surrounding an edge are found. The solution points within these four cells generate a quadrilateral.

A design goal for the SCSolver's manifold extraction is to generalize to higher-dimensional constraint spaces and to be able to extract out solutions of differing dimensions, such as volume solutions in a four-dimensional constraint space. Providing dual-contouring approaches specific to each possible case would require an enormous amount of specialized code. Instead, this system first searches for edges between neighboring cells. This is actually far simpler to code than the dual-contouring quadrilateral search. Given a set of connected vertices and edges, different dimensional solutions can be extracted. For example, in the surface quadrilateral solution case, first the edges are extracted, then a graph search combines the edges into the desired quadrilaterals. If a volume solution is needed, the edges could be combined into cubes, and so on.

3.1.7 Implementation Details

The SCSolver system takes advantage of many hash tables to maintain coherency between different parts of the system. For example, the corners of the cells are shared between many different nodes of the tree. Yet, due to the binary split of the cells and the goal of allowing adaptive refinement, these corners are difficult to store in an explicit grid of points. Instead, a hash table on the point coordinates is used to store expensive constraint evaluations for later use. In the experimental results shown later in the paper, 50 to 75 percent of the constraint evaluations were reused.

Another use of hash tables is during the manifold reconstruction. The graph search for quadrilaterals from connected edges can produce redundant solutions. A hash of the vertex indices is used to detect redundant solutions and prevent them from being added to the solution manifold.

The overall solver is implemented as a C++ base class with virtual functions. Specific constraint problems then implement a derived class with its functions, such as constraint evaluation, overriding the virtual function in the base class.

3.2 Computing Offset Surfaces

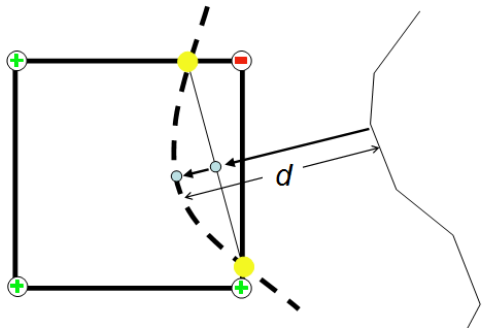

Figure 3: Points on the offset manifold are found by computing the closest point on the surface and using the surface normal at that point to move the cell's mass point onto the constraint zero set.

To compute an offset surface, the SCSolver searches for a set of points that lie at a distance d from the original model. Evaluating the constraint at cell centers and vertices is then equivalent to finding the minimum distance from the sample point in question to the model. A modified PQP package [12] with added point-model minimum distance functionality was used to evaluate the distance constraint.

Given a initial mass point within a cell, a point on the solution zero-set was found with an explicit geometric computation. The distance from the mass point to the model is evaluated, and the supporting closest point on the surface found. This closest point is an orthogonal projection of the mass point onto the surface, thus the vector between the two is in the direction of the surface normal at that closest point. The mass point can then be moved along the normal to lie at the appropriate distance d from the model, and is exactly on the zero set of the constraint (Figure 3).

3.2.1 Examples

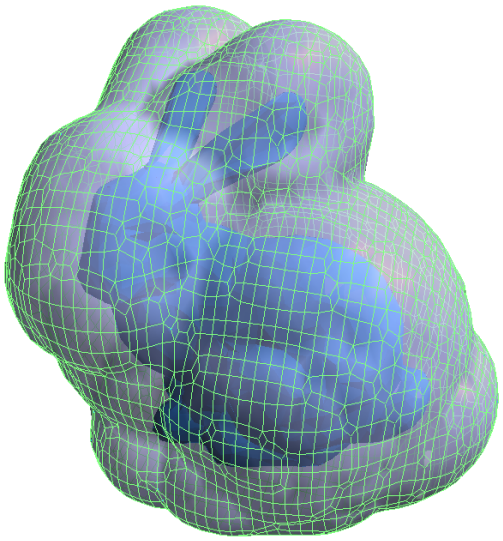

Figure 4: A representative offset surface (translucent) for a bunny model.

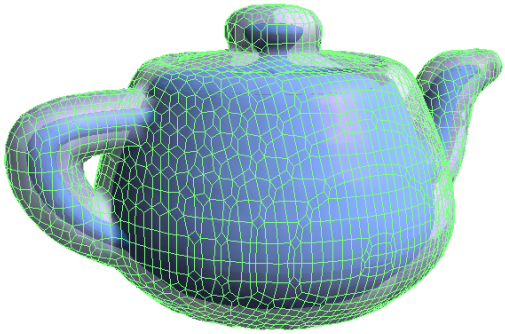

Figure 5: A representative offset surface (translucent) for a teapot model.

In this section, several example offsets are shown. The input models are a bunny model with 2204 triangles (Figure 4) and a teapot model with 8488 triangles (Figure 5). The distance calls used to evaluate the distance constraint scales as $O(log_2(n))$ with the number of triangles n in the model, so model complexity is not a bottleneck in the computation.

3.2.2 Timings

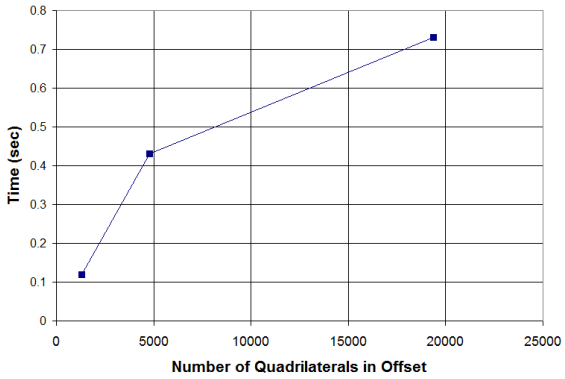

Figure 6: The time to generate the offset surface vs. the number of quadrilaterals in the offset surface.

Figure 6 shows some sample timings for the bunny model. At low solution resolutions, the offset value can be changed interactively. At higher solution resolutions, the algorithm shows good scaling performance. To help judge the performance, note that the bunny model had 2204 triangles, and was a reasonably detailed model. The equivalent solution resolution for number of triangles is between the first and second data points, so the offset also captures model detail. Furthermore, offset surfaces are a smoothing operation, so fewer triangles are acceptable.

All timings were done on a quad-core Q6600 Core 2 Duo computer with 8 gigabytes of memory, although the application is single-threaded. In our experience, only a minor dropoff in speed was shown while running on a modern laptop system.

3.3 Computing Bisector Surfaces

The bisector of two models is the set of points equidistant to both models. Mathematically, the can be expressed implicitly as

$$B_{M_1,M_2} = \{P_{x,y,z} | P_{x,y,z} \in D \wedge d(P_{x,y,z},M_1) = d(P_{x,y,z},M_2)\}. \quad (3)$$

In this case the constraint space represents the difference between distances from a point in the constraint space to each model. The zero set of this constraint space is where points are equidistant. The space is sampled as before, with cells evaluated at their centers to detect possible solutions and at cell corners to determine the solution point and solution topology.

There is no direct algorithmic approach for moving a mass point onto the solution manifold, as there was for offset surfaces. One possibility would be to numerically approximate the constraint space gradient at the mass point, and use numerical methods to move onto the solution manifold. In the spirit of the geometric approach used in the offset solver, instead, a novel geometric approach is used to move the mass point onto the bisector.

3.3.1 Improving the Solution Point

The bisector of two points is a straight line. The bisector of two general shapes can be approximated by the central shape of the Voronoi diagram of a set of points from each model. This central shape is composed of short segments made up of bisectors between points on one model and points on the other.

The closest points on the models to the mass point can be thought of as a sampling of those approximating points for the Voronoi diagram. Thus, the bisector between those points is an approximation of the full bisector surface. Iteratively projecting on that linear bisector, then using the projected point to find new closest points on the model, moves the test point onto the true bisector manifold. In practice, only a few iterations of this process are needed to converge to a small epsilon of error.

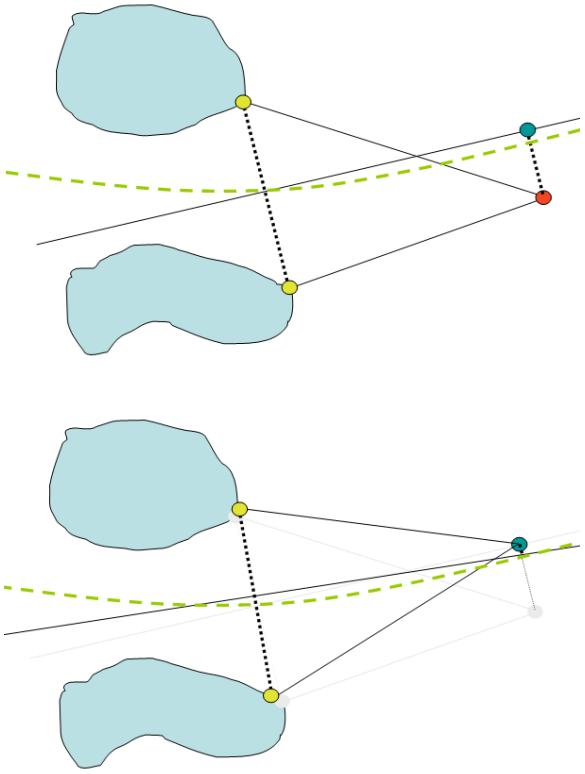

Figure 7: Projecting the mass point onto the linear bisector of the two closest points on the models geometrically converges the mass point onto the true bisector manifold. The true bisector manifold is shown as a dashed line.

3.3.2 Examples

In this section, several bisector surfaces are shown (see Figures 8 and 9). The models are able to be moved while the bisector surface updates interactively.

3.3.3 Timings

Figure 10 shows performance data for computing the bisector surfaces. The data shows remarkably linear behavior with respect to bisector surface resolution. At this time, the reason for different scaling behaviors between the offset and bisector solvers is unexplained, but the different approaches for finding the solution point may have some impact. The bisector surface is generally slower than the offset surface. Evaluating the constraint system takes two distance calls for bisectors versus one for offsets. In addition, the offset mass point is moved onto the solution manifold in a single step, whereas the bisector takes an iterative procedure with multiple distance calls.

4 Discussion and Conclusion

This paper has described and demonstrated a constraint solver based on sampling the system of constraints and adaptively searching that constraint space. The given example constraint problems, surface offsets and bisector surfaces, are fairly straightforward to implement in the SCSolver system. Other symbolic problems that

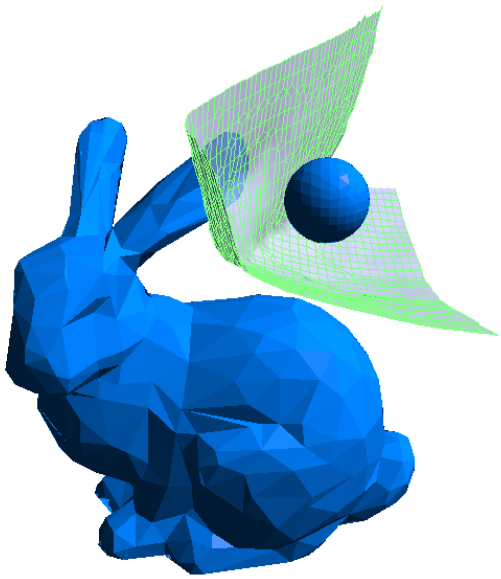

Figure 8: A representative bisector surface (translucent) between bunny and sphere models.

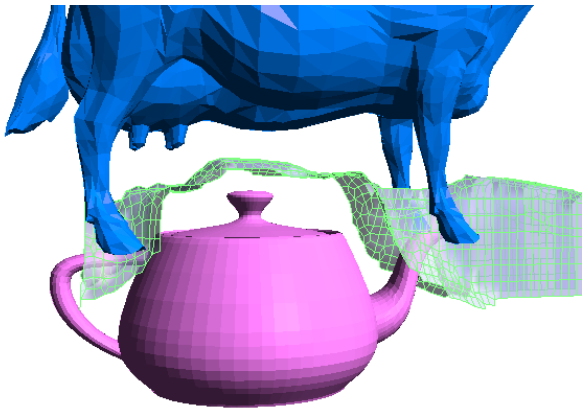

Figure 9: A representative bisector surface (translucent) between cow and teapot models. The z width of the solution domain has been reduced to improve visibility.

are defined using distance measures, such as the medial axis, also should be aaptable to this approach. Other constraints that are not so directly linked to distance measures will require additional work to formulate appropriate bounding metrics on cells of the adaptive octree search. Some initial experiments on higher-dimensional constraint spaces show promise, so the design goals of being a general purpose constraint solver have some validation.

While the search is hierarchical in the constraint space, leaf node size selection is based on a user parameter. Further computational gains may be realized by making the solution surface resolution also adaptive, so that higher resolutions are used in areas of higher solution surface curvature.

The efficiency of the SCSolver approach compares well with current approaches, including graphics card accelerated systems such as [24]. The algorithm should be adaptable to multi-core systems, since the evaluation of the constraint space is the main bottleneck and the evaluations at a cell are largely independent of those in other cells.

In conclusion, the SCSolver approach shows the utility of avoiding a costly explicit constraint space construction phase by sampling the constraint values and providing bounds on changes to the constraints values within a region of the constraint space. The resulting system allows interactive computation of two important geometric problems, offset surfaces and bisectors of models.

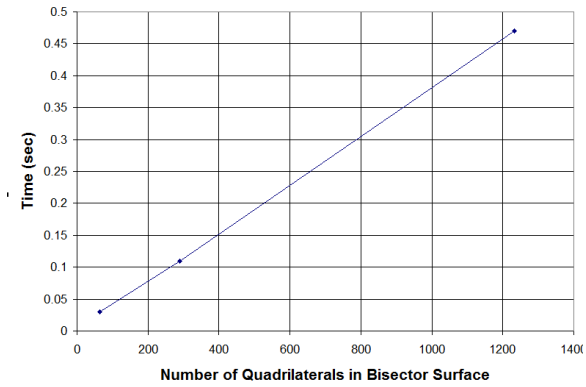

Figure 10: The time to generate the bisector surface vs. the number of quadrilaterals in the bisector surface.

Acknowledgements

This work was supported in part by NSF (CCF0541402). All opinions, findings, conclusions, or recommendations expressed in this document are those of the authors and do not necessarily reflect the views of the sponsoring agencies.

References

[1] D. Breen, S. Mauch, and R. Whitaker. 3d scan conversion of csg models into distance volumes. In *Proceedings of the 1998 Symposium on Volume Visualization*, pages 7–14. ACM SIGGRAPH, 1998.

[2] X. Chen, R. F. Riesenfeld, E. Cohen, and J. N. Damon. Theoretically based algorithms for robust tracking of intersection curves of deforming parametric surfaces. *Computer Aided Design*, 2007.

[3] J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. Brooks, and W. Wright. Simplification envelopes. pages 119–128, 1996.

[4] G. Elber and M. S. Kim. Geometric constraint solver using multivariate rational spline functions. In *Proc. of International Conference on Shape Modeling and Applications*, pages 216–225. MIT, USA, 2005.

[5] J.-c. L. Fabian Schwarzer, Mitul Saha. Exact collision checking of robot paths, 2002.

[6] M. Foskey, M. Garber, M. Lin, and D. Manocha. A voronoi-based hybrid motion planner for rigid bodies, 2001.

[7] S. Frisken, R. Perry, A. Rockwood, and T. Jones. Adaptively sampled distance fields: A general representation of shape for computer graphics. In *Proc. of SIGGRAPH 2000*, pages 249–254, 2000.

[8] G. M. Hunter. *Efficient computation and data structures for graphics.* PhD thesis, Princeton, NJ, USA, 1978.

[9] a. B. S. d. O. Jo and L. H. de Figueiredo. Robust approximation of offsets and bisectors of plane curves. In *SIBGRAPI '00: Proceedings of the 13th Brazilian Symposium on Computer Graphics and Image Processing*, pages 139–145, Washington, DC, USA, 2000. IEEE Computer Society.

[10] T. Ju, F. Losasso, S. Schaefer, and J. Warren. Dual contouring of hermite data. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 339–346, New York, NY, USA, 2002. ACM.

[11] M. S. Kim and G. Elber. Problem reduction to parameter space. In *The Mathematics of Surface IX (Proc. of the Ninth IMA Conference)*, pages 82–98. London, 2000.

[12] E. Larsen, S. Gottschalk, M. Lin, and D. Manocha. Fast distance queries with rectangular swept sphere volumes. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 24–48, 2000.

[13] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169, 1987.

[14] K. Museth, D. Breen, R. Whitaker, S. Mauch, and D. Johnson. Algorithms for interactive editing of level set models. *Computer Graphics Forum*, 24(4):1–22, 2005.

[15] S. L. Omirou and A. C. Nearchou. A cnc machine tool interpolator for surfaces of cross-sectional design. *Robot. Comput.-Integr. Manuf.*, 23(2):257–264, 2007.

[16] N. Patrikalakis and T. Maekawa. *Shape Interrogation for Computer Aided Design and Manufacturing*. Springer Verlag, 2002.

[17] D. Pavic and L. Kobbelt. High-resolution volumetric computation of offset surfaces with feature preservation. *Computer Graphics Forum*, 27(2):165–174, April 2008.

[18] J.-K. Seong, E. Cohen, and G. Elber. Voronoi diagram computations for planar nurbs curves. In *SPM '08: Proceedings of the 2008 ACM symposium on Solid and physical modeling*, pages 67–77, New York, NY, USA, 2008. ACM.

[19] J.-K. Seong, G. Elber, and E. Cohen. Simultaneous precise solutions to the visibility problem of sculptured models. In *GMP*, pages 451–464, 2006.

[20] J.-K. Seong, G. Elber, and M.-S. Kim. Trimming local and global self-intersections in offset curves/surfaces using distance maps. *Computer Aided Design*, 38(3):183–193, 2006.

[21] J.-K. Seong, D. E. Johnson, and E. Cohen. A higher dimensional formulation for robust and interactive distance queries. In *SPM '06: Proceedings of the 2006 ACM symposium on Solid and physical modeling*, pages 197–205, New York, NY, USA, 2006. ACM Press.

[22] E. Sherbrooke and N. Patrikalakis. Computation of the solutions of nonlinear polynomial systems. *ComputerAided Geometric Design*, 10(5):379–405, 1993.

[23] H. Siegfried. Surface reconstruction and variable offset. In *CAD Systems Development: Tools and Methods [Dagstuhl Seminar, 1995]*, pages 199–206, London, UK, 1997. Springer-Verlag.

[24] A. Sud, M. Otaduy, and D. Manocha. Difi: Fast 3d distance field computation using graphics hardware. *Computer Graphics Forum*, 23(3):557–566, 2004.

[25] K. Wentland and D. Dutta. Method for offset-curve generation for sheet-metal design. *Computer-Aided Design*, 25(10):662–670, 1993.