Distance Calculation between a Point and a Subdivision Surface

Torsten Ullrich, Volker Settgast, Ulrich Krispel, Christoph Fünfzig, and Dieter W. Fellner

Institute of Computer Graphics and Knowledge Visualization (CGV), TU Graz, Austria
Graphisch-Interaktive Systeme (GRIS), TU Darmstadt, Germany
PRISM Lab, Arizona State University, Tempe, USA
Contact Email: t.ullrich@cgv.tugraz.at

Abstract

This article focuses on algorithms for fast computation of the Euclidean distance between a query point and a subdivision surface. The analyzed algorithms include uniform tessellation approaches, an adaptive evalution technique, and an algorithm using Bézier conversions. These methods are combined with a grid hashing structure for space partitioning to speed up their runtime.

The results show that a pretessellated surface is sufficient for small models. Considering the runtime, accuracy and memory usage an adaptive on-the-fly evaluation of the surface turns out to be the best choise.

1 Introduction

The problem to determine the Euclidean distance between an arbitrary point in 3D and a free-form subdivision surface is fundamental in many different communities including computer-aided geometric design, robotics, computer graphics, and computational geometry.

A lot of algorithms in the context of physical simulation, path planning, etc. have to determine this distance: an exemplary algorithm is the shape fitting approach by TORSTEN ULLRICH. It evaluates distances between a point cloud and some subdivision surfaces in order to fit a parametric model [1]. As query time is always an issue, the goal is to choose the best combination for the application at hand.

Subdivision surfaces are based on polygonal meshes, and they can be subdivided into triangle meshes. So is it suitable to preprocess the object into a triangle mesh and compute distances to the object just by searching the closest triangle? Should the search be performed on the subdivision surface patches? This article discusses accuracy, runtime and memory usage of various approaches for searching strategy, surface primitives used, and calculation of the primitive's minimum distance.

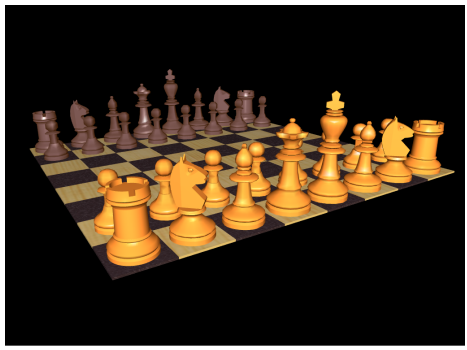

Figure 1: The test objects are chess figures modeled with subdivision surfaces. Each initial mesh has between 70 ("pawn") and 1 454 ("rook") polygons. The Figure shows all test pieces in their initial chess position.

2 Related Work

Subdivision surfaces are part and parcel of this article. They define an object through recursive subdivision starting from an initial control mesh. A variety of schemes with different subdivision rules exist for geometric design and graphics applications. An overview on subdivision surface modeling in the context of computer-aided design has been presented, e.g., by WEIYIN MA [2].

2.1 Subdivision Surfaces

A subdivision surface is defined by an infinite subdivision process. In contrast to parametric surfaces which provide a finite evaluation algorithm, a subdivision surface does not come with a direct evaluation method at arbitrary parameter values. Currently, it can be evaluated via

- **Uniform subdivision:** If the subdivision rules are applied sufficiently often, the resulting mesh will be a tight approximation of the limit surface [3]. For non-interpolating subdivision schemes, e.g., Catmull-Clark, the resulting mesh points will not lie on the limit surface in general. In order to decrease the deviation, limit point rules can calculate the point on the limit surface for a subdivision mesh point [4].
- **Adaptive subdivision:** Due to the exponential need of memory it is a good strategy to subdivide the mesh adaptively. This results in a subdivision process with varying subdivision depth but constant overall accuracy [5]. The use of limit point rules is essential for the connection of mesh parts with different subdivision depths.
- **Exact evaluation & conversion:** Stationary subdivision schemes, e.g., Catmull-Clark, allow an exact evaluation at arbitrary parameter values [6]. JOS STAM makes use of the property that regular patches (control mesh faces with all vertices of valence 4) can be evaluated as uniform, bicubic b-spline patches. The region around irregular points (non-valence 4) shrinks successively when subdividing the irregular patches, and the eigen-structure of the subdivision matrix is used to determine the limit there. Two alternative parameterizations for irregular patches were proposed in [7], which ensure non-degenerate derivatives of the parameterization. For Catmull-Clark subdivision, a regular quad patch can even be represented as a single bicubic Bézier patch.

2.2 Distance Calculations

Distance fields are a representation, where at each point within the field, the distance from that point to the closest point on a fixed object within the domain is known. In addition to distance, other properties (direction to the surface, etc.) may be derived from the distance field. A survey of methods for the precomputation and representation of distance fields can be found in “3D Distance Fields: A Survey of Techniques and Applications” [8]. To speed up the search in a domain space-partitioning data structures allow to access object parts by spatial proximity and other properties. Data structures used for this, like tree structures (e.g., kd-tree), grid structures (e.g., 3D regular grid) and cell structures (e.g., Voronoi diagram) are described in “Geometric Data Structures for Computer Graphics” [9].

For applications where many distance queries are performed and the object is fixed within the domain, the distance field can be precomputed and represented in a scalar data structure for the domain, like a regular grid or a compressed regular grid. With this *distance field data structure*, the distance from a domain point to the closest object point can be retrieved from the data structure. In the case, where the object is deforming, it is necessary to update this derived, scalar field. In any case, where object information is stored about the location within the domain, this has to be updated. So for deforming and changing objects it is beneficial to keep this amount as small as possible, at best without any domain data structure. This setting without preprocessing is called *online distance evaluation*.

For the problem of distance computation to subdivision surfaces, we propose the following classification of approaches:

- **Approaches based on the distance field:** A separate scalar data structure reconstructs the (signed) distance to the closest point on the object [8]. We also consider to this group GPU approaches like [10], which compute and evolve the distance field in a small narrow band around the object.
- **Searching of surface primitives of the original object representation:** The curved surface patches, which correspond to a face of the control mesh, are organized in a spatial data structure for the domain based on their bounding volume. Only this data structure has to be updated after model deformations. The spatial data structure is then traversed in increasing minimum distance to the query point, and the primitive’s minimum distance is computed as a subproblem. A termination condition is necessary to stop the search with the correct distance value.

- **Searching of surface primitives derived from the original object representation:** Instead of using the surface primitives of the original object representation immediately, one derives a small set of simpler primitives from the original surface primitives. The reason could be that they offer a simpler minimum distance algorithm. In the case of subdivision surfaces, the surface's triangulation is often available also from other tasks.

3 Exposition of Methods

In this work, we chose three kinds of algorithms to determine the distance between a query point and a subdivision surface. The first group consists of three algorithms which use the triangulation of a subdivision surface. The next approach evaluates the subdivision surface on-the-fly. And the last algorithm converts it into Bézier patches. In this case distance queries are answered by a numerical minimization routine.

3.1 Uniform Triangulation

The most simple approach uses an uniform tesselation of the subdivision surface at a fixed depth to create a triangle mesh. For a tight approximation of the limit surface, the limit points of the control vertices have been used. For each query point the distance to each triangle is calculated [11], and the minimum is selected. This approach does naive search without any spatial data structure.

pros The calculation is robust and its correctness can be verified easily.

cons As runtime and memory footprint of a single distance query are linear in the number of triangles and exponential in the subdivision depth, this algorithm is not useful for real world applications. The implementation has been used to verify the results of the following algorithms, but it is not considered to be a practical solution.

3.2 Hashed Triangulation

A significant speed-up can be achieved if the triangulation is stored in a space-partitioning data structure. The hashed triangulation approach is a space-efficient implementation of a 3D regular grid by using spatial hashing [12]. In this way, the storage requirements can be restricted arbitrarily, e.g., linear in the number of model triangles.

For a given query point, the hashed triangulation approach determines which grid cells may potentially contain the nearest triangle. Within the grid cells in question, the registered triangles are checked. According to our classification, it is based on *searching of surface primitives derived from the original object representation*.

pros The technique is easy to implement, and a well chosen grid cell size gives good query times.

cons The memory footprint is exponential in the subdivision depth which disqualifies it for many applications. Another problem is the algorithm's dependency on the choice of the grid cell size. A reasonable size takes into account the model's bounding volume as well as its face distribution within the domain. This problem is discussed in detail in Section 4.2.

3.3 Hashed Triangulation – First Hit

A further speed-up is possible, if only the distance value (not the corresponding perpendicular point) is needed, and if a small error is acceptable. In this case, only the nearest non-empty cell is checked. If no other cell is checked, the returned value may have an error up to the length of the cell's diagonal.

pros Same as 3.2 Hashed Triangulation.

cons Same as 3.2 Hashed Triangulation. The returned distance value is only a rough approximation.

3.4 Adaptive Subdivision

The triangulation-based distance calculations described before have large memory requirements in common. If the subdivision control mesh has to remain in memory, for any reason, the triangulation-based methods are not suitable due to their large memory requirements. An approach which does the refinement of the subdivision mesh on-the-fly has always smaller memory requirements. Our implementation of the adaptive subdivision algorithm uses a hashed 3D regular grid structure to identify relevant subdivision patches. These patches are subdivided using slates [13] as needed. According to our classification, it uses *searching of surface primitives of the original object representation*.

pros The memory footprint is only linear in the size of the subdivision mesh due to the 3D hash table. The additional overhead during a patch evaluation is of small, fixed size and can be neglected.

Only a small preprocessing is needed. In contrast to triangulation-based approaches, this allows to modify the maximum subdivision depth and therefore adapt the accuracy of the distance calculation as needed.

cons The algorithm requires a substantial implementation.

3.5 Bézier Representation & Numerical Optimization

Some subdivision schemes, e.g. Catmull-Clark subdivision [6], allow direct evaluation at arbitrary parameter values. This property can be used to formulate a distance calculation algorithm. Having identified relevant subdivision patches, the algorithm converts them into Bézier patches. For regular patches this can be done exactly. Irregular patches have to be approximated. Using a parameterization as a Bézier patch, the distance calculation can be formulated as a minimization problem in parameter space [14–16]. For the resulting nonlinear minimization problem, Newton-type techniques [17], [18] can be used with suitable start values in parameter space.

pros The memory requirements are comparable to the adaptive subdivision algorithm. As the distance calculation is reduced to a standard problem of numerical optimization, highly-optimized numerical libraries can be used.

cons The Bézier approximation has some additional runtime overhead, but can be cached with the subdivision mesh. The following distance minimization requires considerable tuning of the step sizes. The choice of the start parameter of the Newton-like iteration has more influence on the runtime than the size of the model.

Furthermore the conversion of Catmull-Clark subdivision to bicubic Bézier patches is patent-registered ("Approximation of Catmull-Clark subdivision surfaces by Bézier patches", United States Patent No. 6950099).

4 Implementation

In order to allow a thorough comparison of the chosen algorithms some implementation issues are discussed in detail.

4.1 Evaluation Errors

The triangulation-based methods use a fixed, uniform subdivision depth of three subdivisions. Note that the use of limit points improves the approximation error, which can be bounded by a factor times the maximum of the triangle's side lengths, where the factor depends on the model. The limit points lie in the convex hull of the Bézier control mesh instead of the convex hull of the corresponding face's 1-ring in the Catmull-Clark mesh. This error has been used to set the termination condition of the adaptive subdivision algorithm. Therefore, the adaptive version has a maximum subdivision depth of three, but it is allowed to terminate earlier, if the resulting maximum error is of same size.

The Bézier surface patches resulting from the conversion have a deviation from the Catmull-Clark surface patches only in irregular patches. But the subsequent parameter search, which works with the Bézier representation, produces an error by itself. With the termination condition in parameter space it is difficult to control the distance error because the threshold in parameter space depends on the curvature near a minimum point's parameter. In our experiments we used only a fixed threshold.

The accuracy of the First-Hit algorithm is determined by the triangulation error plus $\sqrt{3}$ times the grid cell size.

4.2 Grid Size Problems

The grid cell size is not only responsible for the algorithm's accuracy. The choice of a reasonable value affects the algorithm's performance significantly. Unfortunately, the value depends on the distribution of the cached geometric primitives (triangles, Bézier patches, etc.) within space. Without additional knowledge only some heuristics are at hand. Let d be the bounding volume's diagonal length, and p be the number of geometric primitives to hash. If all objects are distributed uniformly in their bounding volume, a grid cell size of $d/\sqrt[3]{p}$ is a reasonable choice. If the surface of a geometric object is not distributed uniformly in space, the

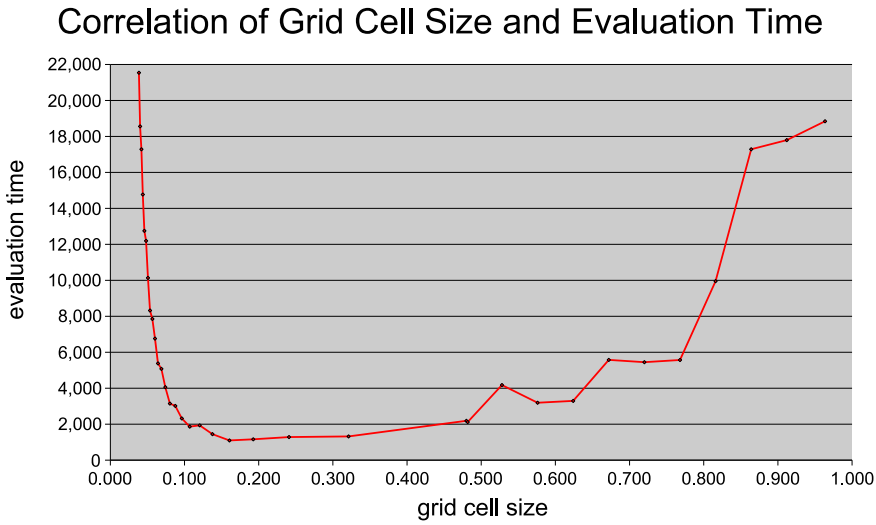

Figure 2: This Figure demonstrates the correlation between grid cell size and runtimes of hashing-based algorithms. The used test object "pawn" has been triangulated (8 862 triangles). All triangles reside inside the axis-aligned bounding box whose diagonal has a length of 3.94. According to the heuristics in Equation 1 the cell size should be between $3.94/\sqrt[3]{8862} \approx 0.19$ and $3.94/\sqrt[5]{8862} \approx 0.65$. The needed time in milliseconds to calculate the distance of 10 000 arbitrary points to the triangle mesh using the First-Hit algorithm is plotted against the used grid cell size.

grid should be coarsened. In our implementation the grid cell size had been chosen to

$$s := \frac{d}{\sqrt[n]{p}} \tag{1}$$

with $3 \leq n \leq 5$, which has led to feasible runtimes. An illustrative example in Figure 2 shows the correlation of cell grid size and evaluation time for a test object.

4.3 Hashing

All presented algorithms use grid-based hashing. We used the hashing function presented by MATTHIAS TESCHNER [12]. It takes the indices (x, y, z) of a grid cell and returns the hash value

$$hash(x, y, z) = (x\, p_1 \textbf{ xor } y\, p_2 \textbf{ xor } z\, p_3) \textbf{ mod } n \tag{2}$$

using the prime numbers $p_1 = 73\ 856\ 093$, $p_2 = 19\ 349\ 669$, $p_3 = 83\ 492\ 791$. The function can be evaluated very efficiently and produces a comparatively small number of hash collisions for small hash tables of size n. The traversal within the grid structure is illustrated in Figure 3.

4.4 Slates for Subdivision Surfaces

The adaptive subdivision algorithm does not modify the base mesh. Instead a separate data structure is used consisting of two so-called *slates*.

A slate is composed of a two-dimensional array table of size

$$(2^{sd} + 3)^2$$

and four one-dimensional corner arrays of size

$$(val - 4) \cdot 2,$$

where sd is the maximum subdivision depth and val the maximum valence. For performance reasons, the slates are allocated statically as they can be reused for each face to be tessellated.

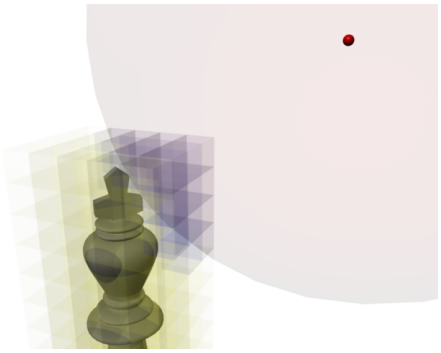

Figure 3: The storage of a model in a regular grid allows a fast preselection of relevant patches/triangles, which are near the query point (red). In combination with a good hash function the memory footprint is proportional to the number of model primitives.

The subdivision process firstly collects the 1-neighborhood of the considered face f and stores it in the first slate. The vertices of f and the vertices of its edge neighbor faces are stored in the table. If one of the vertices of f has valence greater than four, the remaining vertices are stored in the dedicated corner arrays. Figure 4 illustrates this storage scheme for a quad. Other configurations and further details on slates can be found in "Adaptive Tessellation of Subdivision Surfaces" [13].

The subdivision algorithm processes the vertices row by row and stores the result of one subdivision step in the second slate.

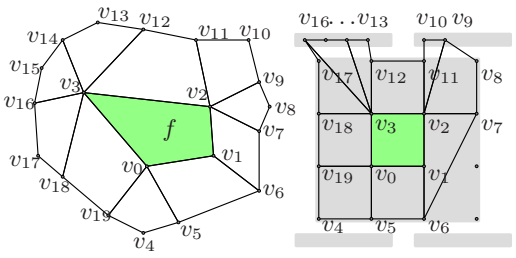

Figure 4: The adaptive subdivision algorithm stores the collected 1-neighborhood of a face f from the base mesh (left) in a data structure called slate (right).

For the next step, source and destination slates are swapped. After two subdivision steps, the algorithm starts calculating distances from the corresponding limit points of the 25 (5×5) vertices to the query point. For the following subdivision steps, only a subpart of 9 (3×3) vertices of the table array is used, see Figure 5. The subpart is chosen depending on the results of the distance calculations. The process is repeated until the difference of the minimal distance for the current and the last iteration is below a user defined threshold.

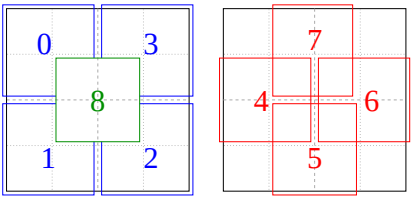

Figure 5: After the second subdivision step each face of the control mesh consists of 5×5 vertices. During the distance calculation only relevant subparts out of nine possibilities are processed further on. Five possible sectors are illustrated on the left, four on the right.

5 Benchmarks

The test scenario is made of six subdivision surface models.

1. **Pawn** This object consists of 70 patches. Its triangulation at subdivision level 3 has 8 862 triangles.
2. **Rook** Within the test scenario this object is the most complex one. It is composed of 1 454 subdivision surface patches, respectively 185 328 triangles.
3. **Knight** The control mesh of this model has 78 faces. Triangulated after three successive subdivisions it consists of 9 356 triangles.
4. **Bishop** The bishop is modeled using 130 patches. In this case the triangulation-based algorithm have to handle 16 542 triangles.
5. **Queen** This model has 387 subdivision surface patches which results in a triangulation with 49 508 elements.
6. **King** The king consists of a subdivision mesh with 175 faces. Its tessellation with 19 560 triangles ranges in the midfield of the test scenario.

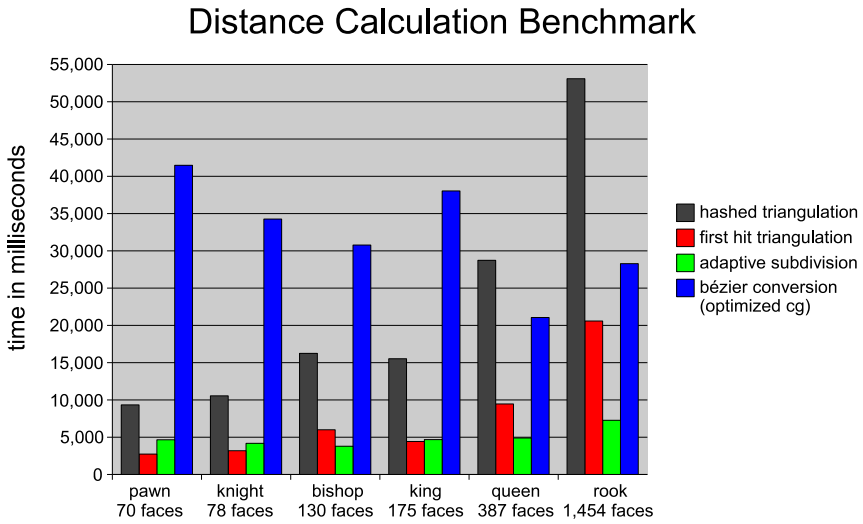

Figure 6: The test objects are Catmull-Clark surfaces. The smallest object – the Pawn – has a subdivision control mesh which consists of 70 faces. The most complex model is the Rook with 1 454 patches. Its triangulation at subdivision level 3 has 185 328 triangles. The triangulation-based algorithms as well as the adaptive subdivision one correlates with the model's complexity in contrast to the approach using Bézier conversion and numerical optimization. This algorithm is rather determined by internal parameters (initial optimization values, etc.) than by model complexity.

During each test an algorithm has to calculate the distance between the test object and 10 000 arbitrary query points. The query points are uniformly distributed within a box whose volume is twice as large as the test object's axis-aligned bounding box (AABB) volume. Each test model has a closed 2-manifold boundary and the query points may be located inside and outside of it; whereas the returned distance has no sign and does not distinguish between interior and exterior.

The runtimes of these tests are shown in Figure 6. The results indicate some interesting facts. Both the adaptive subdivision technique and the Bézier conversion approach use the same 3D hashed grid structure to identify relevant patches with a grid cell size of $d/\sqrt[3]{p}$, whereas d denotes the AABB diagonal and p the number of patches in the base mesh. The adaptive subdivision depends on the number of relevant patches which correlates with the model's complexity. But the Bézier conversion is rather determined by internal parameters (start values for numerical iterations, etc.) than by model complexity. This calculation overhead is almost independent from the input data and surmounts the time needed by the adaptive subdivision approach several times.

Another interesting point which can be seen in the diagram is the speed-up factor of the first-hit algorithm. Compared with the variant which checks additional grid cells in order to return the exact distance instead of an approximation the first-hit version is three times faster ($\phi \approx 3.09$). Of course, both algorithms use the same grid size. The number of grid cells is proportional to the number of triangles in the tessellation.

While it is normally not recommended to triangulate a subdivision surface ahead of time, the first hit version has similar timings as the adaptive evaluation technique, at least for small- and medium-sized models.

6 Conclusion

According to the benchmarks presented above, the distance between an arbitrary point and a subdivision surface should be determined using an efficient space partitioning technique such as hashed, regular 3D grid and an on-the-fly subdivision surface evaluation algorithm. The result is a distance calculation which

- needs considerably less memory than triangulation based approaches, and
- is the fastest method in most cases.

The only negative point of the adaptive subdivision method is its complex implementation. The conversion method may use numerical libraries and the triangulation methods can use wide-spread, standard techniques, whereas an efficient, on-the-fly evaluation of subdivision surfaces must be implemented efficiently for the mesh structure used.

Therefore, the triangulation-based approach with the first-hit termination might be considered for small model sizes, if the perpendicular point is not needed and if an approximation of the distance is enough. In all other cases the adaptive subdivision technique is the best choice.

7 Acknowledgment

The authors would like to thank RENÉ BERNDT. He created and provided the chess models which have been used for benchmarking.

References

[1] T. Ullrich and D. W. Fellner, "Robust shape fitting and semantic enrichment," *Proceedings of the International Symposium of the International Committee for Architectural Photogrammetry (CIPA) 2007*, vol. 21, p. to appear, 2007.

[2] W. Ma, "Subdivision surfaces for CAD - an overview," *Computer-Aided Design*, vol. 37, no. 7, pp. 693–709, 2005.

[3] E. Catmull and J. Clark, "Recursively generated B-spline surfaces on arbitrary topological meshes," *Computer-Aided Design*, vol. 10, pp. 350–355, 1978.

[4] D. Zorin, P. Schröder, T. DeRose, L. Kobbelt, A. Levin, and W. Sweldens, "Subdivision for Modeling and Animation," *SIGGRAPH 2000 Course Notes*, vol. 1, pp. 1–116, 2000.

[5] K. Müller and S. Havemann, "Subdivision Surface Tesselation on the Fly using a versatile Mesh Data Structure," *Computer Graphics Forum*, vol. 19, no. 3, pp. 151–159, 2000.

[6] J. Stam, "Exact evaluation of Catmull-Clark subdivision surfaces at arbitrary parameter values," *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, vol. 1, pp. 395 – 404, 1998.

[7] I. Boier-Martin and D. Zorin, "Differentiable Parameterization of Catmull-Clark Subdivision Surfaces," *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, vol. 71, pp. 155 – 164, 2004.

[8] M. W. Jones, A. J. Baerentzen, and M. Sramek, "3D Distance Fields: A Survey of Techniques and Applications," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 4, pp. 581– 599, 2006.

[9] G. Zachmann and E. Langetepe, "Geometric Data Structures for Computer Graphics," Course notes, EG tutorial, 2002.

[10] M. Botsch, D. Bommes, C. Vogel, and L. Kobbelt, "Gpu-based tolerance volumes for mesh processing," *Proceedings of 12th Pacific Conference on Computer Graphics and Applications*, vol. 12, pp. 237– 243, 2004.

[11] M. W. Jones, "3D Distance from a Point to a Triangle," *Technical Report, Department of Computer Science, University of Wales Swansea*, vol. 5, pp. 1–5, 1995.

[12] M. Teschner, B. Heidelberger, M. Mueller, D. Pomeranets, and M. Gross, "Optimized Spatial Hashing for Collision Detection of Deformable Objects," *Proceedings of Vision, Modeling, and Visualization*, vol. 1, pp. 47–54, 2003.

[13] V. Settgast, K. Müller, C. Fünfzig, and D. W. Fellner, "Adaptive Tesselation of Subdivision Surfaces," *Computers and Graphics*, vol. 28, no. 1, pp. 73 – 78, 2004.

[14] Y. L. Ma and W. T. Hewitt, "Point inversion and projection for nurbs curve and surface: control polygon approach," *Computer Aided Geometric Design*, vol. 20, no. 2, pp. 79–99, 2003.

[15] M. Marinov and L. Kobbelt, "Optimization methods for scattered data approximation with subdivision surfaces," *Graphical Models*, vol. 67, no. 5, pp. 452 – 473, 2005.

[16] I. Selimovic, "Improved algorithms for the projection of points on nurbs curves and surfaces," *Computer Aided Geometric Design*, vol. 23, no. 5, pp. 439–445, 2006.

[17] R. Fletcher and C. M. Reeves, "Function minimization by conjugate gradients," *The Computer Journal*, vol. 7, pp. 149–154, 1964.

[18] C. Kelley, *Iterative Methods for Optimization*, ser. Frontiers in Applied Mathematics, C. Kelley, Ed. Society for Industrial and Applied Mathematics (SIAM), 1999, vol. 18.