High Quality Surface Mesh Generation for Swept Volumes*

Andreas von Dziegielewski † Michael Hemmer‡

Abstract

We present a novel and efficient technique to generate a high quality mesh that approximates the outer boundary of a swept volume (SV). Our approach comes with two guarantees. First, the approximation is conservative, i.e. the swept volume is enclosed by the output mesh. Second, the one-sided Hausdorff distance of the generated mesh to the swept volume is upper bounded by a user defined tolerance. Exploiting this tolerance our method produces an anisotropic mesh which nicely adapts to the local complexity of the approximated swept volume boundary. The algorithm is two phased: a initialization phase that generates a conservative voxelization of the swept volume, and the actual mesh generation which is based on CGAL's Delaunay refinement implementation.

1 Introduction

The swept volume is defined as the entity of all points touched by a solid (the generator) under the transformations of either a continuous or discrete trajectory. In the context of this paper, we assume a discrete trajectory and define $\mathcal{SV}$ as the entity obtained by linear interpolating the generator geometry between consecutive time steps, as proposed in [2].

The swept volume plays an important role in computer aided design (CAD), numerically controlled (NC) machining verification, robot analysis and graphical modeling. For safety reasons most applications demand for a conservative approximation of $\mathcal{SV}$, that is, the result must contain $\mathcal{SV}$. At the same time, the result should approximate $\mathcal{SV}$ as close as possible while keeping the complexity of the output low, the latter becoming more and more relevant due to increasing model complexity and high demands concerning error tolerance. Moreover, a practical algorithm should be tolerant to topologically inconsistent input data and should preferably impose no restrictions on the input models.

*This work has been supported in part by the 7th Framework Programme for Research of the European Commission, under FET-Open grant number 255827, CGL, Computational Goemetry Learning.

†Institut für Informatik, Johannes Gutenberg Universität Mainz, dziegiel@uni-mainz.de

‡School for Computer Science, Tel Aviv University, mhsaar@google.mail.com

1.1 Previous and related work

Mathematical formulations describing the swept volume include Jacobian rank deficiency methods, sweep differential equations and envelope theory, for a survey see [1].
Practical approaches, most relying on volumetric SV representations cannot give geometrical guarantees because sharp features can be missed ([7, 6]). In [11] the authors claim to be able to archieve arbitrarily tight error-bounds (although do not regard conservativeness) but unfortunately no timings for a priori error bounds are given. They all produce a highly overtessallated mesh and do not propose any error bound mesh simplification method. Applying error bound simplification (e.g. [10]) to these meshes, a posteriori, will always be limited by the storage needed for the mass of triangles of the original mesh.

Recent approaches [9, 3] are able to produce a conservative approximation. In [3] local culling criteria are applied to generate a superset of the SV boundary that is then blended using a BSP tree. Their output mesh is non-adaptive and possibly highly overtessellated, and further simplification in convex regions would violate conservativeness. The method proposed in [9] relies on special conservative depth buffer voxelization and an error bound mesh simplification phase. The output mesh is adaptive and almost everywhere manifold. It is conservative, but error bounds can not be given for all concave parts of the SV.

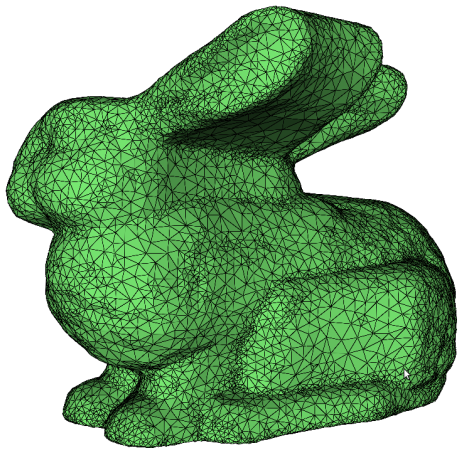

Figure 1: Bunny swept along a trajectory consisting of 12 transformations.

1.2 Our contribution

In contrast to these previous approaches our mesh is build top down, i.e. we start from a coarse mesh and apply local refinement until we are guaranteed a conservative and error bound approximation of the outer $\mathcal{SV}$ boundary. More precisely, we guarantee that the one-sided Hausdorff distance to $\mathcal{SV}$ does not exceed the user defined tolerance δ. In addition the resulting anisotropic mesh is of high quality while keeping the complexity relatively low, that is: produced triangles obey quality constrains such as lower bounds on smallest angles while mesh density adapts to the local complexity of $\mathcal{SV}$ as far as it is required by δ. The algorithm is two phased: a initialization phase that generates a conservative voxelization of the swept volume, and the actual mesh generation which is based on CGAL's Delaunay refinement implementation [8].

The Delaunay refiment and the general scheme of the approach are discussed in Section 2. More details of the initialization phase, are then discussed in Section 3. Implementation details and some preliminary results are presented in Section 4.

2 Overview

In order to generate the output mesh we apply Delaunay refinement which is known to be one of the most powerful techniques in the field of mesh generation and surface approximation. More precisely, our algorithm utilizes CGAL's frame work for 3D mesh generation [8] which we briefly review next.

Starting from an initial points set on the surface of the to be meshed domain $\mathcal{D}$, the process maintains a Delaunay triangulation of this point set. Thereby, it classifies each tetrahedron as interior or exterior according to the position of the center of its circumscribing ball. A triangle is said to belong to the surface if the classification of the two neighboring tetrahedra differ. Such a surface facet f can be refined by inducing the intersection point of its Voronoi edge (the dual of f) with $\partial\mathcal{D}$.[1] The refinement process now successively refines those boundary facets that are classified as bad facets, for instance, triangles that are considered two large or whose minimal angle is too small. The latter criterion ensures well formed triangles in the resulting mesh.

The main idea of our approach is to add another criterion that classifies a boundary facet as bad if: (a) the facet intersects $\mathcal{SV}$ or (b) the one-sided Hausdorff-distance of the facet to $\mathcal{SV}$ is too large. Note that part (a) implies that we can not place intersection points directly on $\partial\mathcal{SV}$ since boundary facets would always intersect $\mathcal{SV}$ in regions where $\mathcal{SV}$ is locally convex. Thus the idea is to place intersections points on some offset of $\mathcal{SV}$ at about half of the user defined tolerance. This way it is guaranteed that boundary facets eventually fulfill both conditions which in turn guarantees termination of the generation process.

[1] On the duality of the Delaunay triangulation and the Voronoi diagram see for instance [4]

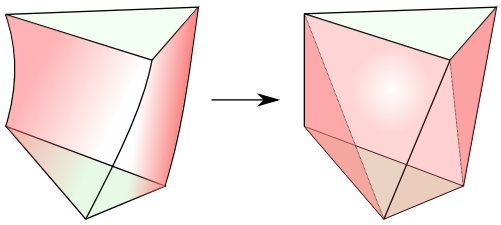

Figure 2: Sweeping a triangle yields a prism with three ruled surfaces, each of which is approximated and tessellated by inserting the diagonal with the lower dihedral angle.

The approach is two phased: In a initialization phase we generate a conservative and sufficiently precise voxelization $\mathcal{V}_0$ of $\mathcal{SV}$. In addition we compute two offsets $\mathcal{V}_1$ and $\mathcal{V}_2$ by successively adding an additional layer of voxels to $\mathcal{V}_0$. In the second phase, the mesh generation phase, we set $\mathcal{D} = \mathcal{V}_1$. Intersection points of Voronoi edges with $\partial\mathcal{V}_1$ are computed using bisection. In order to classify a facet, the facet is voxelized. The facet is considered as bad if one of its voxels is contained in $\mathcal{V}_0$ (ensuring (a)) or outside of $\mathcal{V}_2$ (ensuring (b)).

Thus the core of our approach is an efficient generation of $\mathcal{V}_0$ and its offsets, which is discussed next.

3 Initialization Phase

In order to obey the user defined tolerance δ it is of course necessary to chose the size of each voxel ε smaller than δ. Taking into account that the diagonal of a voxel is $\varepsilon\sqrt{3}$, the two layers for $\mathcal{V}_2$ as well as another layer for the conservative voxelization of $\mathcal{V}_0$ we have to chose $\varepsilon < \delta/3\sqrt{3}$. That is, for a reasonable scenario with an area of interest of about $1m^3$ and a user defined tolerance of $1mm$ this would result in about $5200^3 \simeq 140 * 10^9$ voxel, which brings memory consumption onto the table. In order to decrease the memory usage we store a voxelization as an octree which we, similar to [5], internally represented by a hash set. A node is marked as occupied by its pure existence in the hash set. In case all 8 children of a node are marked the node is marked and the children can be deleted. A voxel is not covered if its leaf and non of its ancestors exists in the hash set. Testing containment as well as insertion is in $O(\log 1/\varepsilon)$ However, the most important property is that for a reasonable $\mathcal{SV}$ one can expect a memory consumption that is proportional to $\partial\mathcal{SV}$, i.e., quadratic in $1/\varepsilon$.

Assuming a linear interpolation every triangle gives rise to a deformed prism between two trajectory positions, see also Figure 2. Thus inserting all voxels of

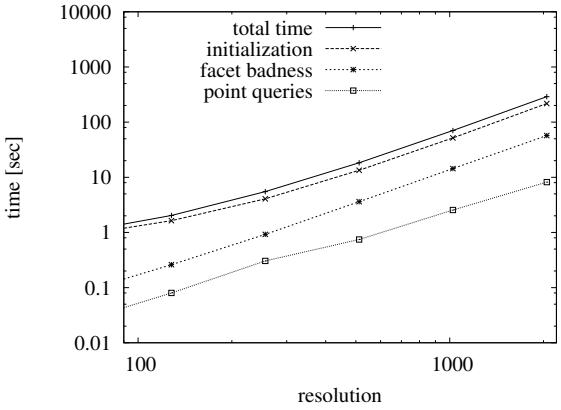

Figure 3: Total time in relation to main functions.

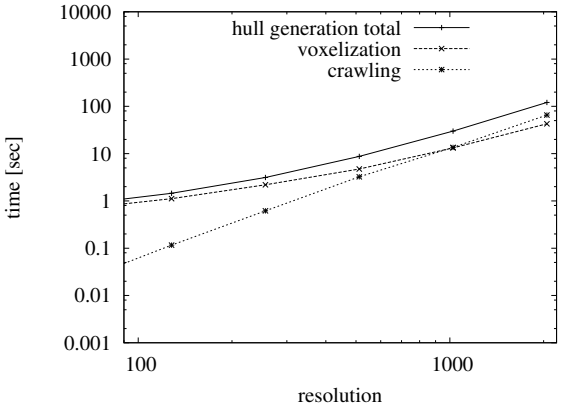

Figure 4: Times for initialization phase.

a conservative voxelization of each such prism would directly yield $\mathcal{V}_0$. Since generating all these voxels is to expensive we follow a culling heuristic by [3] that tries to rule out those triangles (of the prisms) that do not contribute to the boundary. This is based on local criteria of the mesh in relation to the current direction of the motion. Only the remaining triangles are then voxelized. The voxelization is based on a simple subdivision scheme. Starting from the initial cube, the boxes that still intersect the triangle are subdivided until the required resolution is reached. The intersection test is based on the separating axes theorem. The result $\partial\tilde{\mathcal{V}}_0$ is a subset of $\mathcal{V}_0$ and contains $\partial\mathcal{V}_0$. In a next step we generate $\partial\mathcal{V}_1 = \mathcal{V}_1 \setminus \mathcal{V}_0$ by crawling along the outer boundary of $\partial\tilde{\mathcal{V}}_0$. We then throw away $\partial\tilde{\mathcal{V}}_0$ and generate $\mathcal{V}_0$ by filling $\partial\mathcal{V}_1$. $\mathcal{V}_1$ is then generated by simply adding $\partial\mathcal{V}_1$ to a copy of $\mathcal{V}_0$. $\mathcal{V}_2$ is created in a similar fashion. Since $\partial\tilde{\mathcal{V}}_0$ and $\partial\mathcal{V}_1$ are stored in a plain hash set the crawling is, for reasonable volumes, in $O((1/\varepsilon)^2)$. Note that the filling is not cubic since it uses an hierarchic scheme, which we can no discuss here due to limited space.

4 Preliminary Results

All benchmarks were measured on a Intel(R) Core(TM) i5 CPU M 450 with 2.40GHz with 512 kB cache under Ubuntu Linux and the GNU C++ compiler v4.4 with optimizations (`-O2`). However, we only use one CPU since we did not parallelize any part of the approach yet.

The swept Stanford Bunny, Figures 1 and 5, were generated from an initial mesh with 8100 triangles and trajectories of size 12/50. The resolution was set to $2^{10} = 1024$ which corresponds to $\delta \simeq 0.005$. One can see that the mesh nicely adapts to the local complexity of $\mathcal{SV}$, with sparse areas in well behaved regions. The resulting mesh has only about 20k/10k boundary facets. Figure 3 shows the obtained times for the Stanford Bunny shown in Figure 1. Note that times and resolution are given in a logarithmic scale. The graphs show the total time in relation to three major components: the time for the initialization phase, time spend in point classification and facet classification. The sum of the later two is essentially the time spend in the mesh generation phase. Thus already for such a small example the initialization phase clearly dominates the time used overall.

It turns out that most of the time of the initialization phase is currently spend in the voxelization of all triangles that where not culled and the computation of the hull surrounding it. All other steps such as the hierarchical filling of $\mathcal{V}_0$ and the offset computation to compute $\mathcal{V}_1$ and $\mathcal{V}_2$ from $\mathcal{V}_0$ are not relevant. Figure 4, shows a more detailed plot of the time spend in the hull computation. Initially, the voxelization is clearly the dominating part, but for high resolution the hull computation eventually takes over. This is due to the fact that the offset computation in this part is more expensive since the volume is not filled and thus the size of the octree is much larger. However, for larger models and larger trajectories the first part would currently be the most dominating time factor.

5 Further Work

Though our results are already very promising we already see several, partially obvious, optimizations that where not yet applied due to lack of time.

Using only one offset would significantly improve memory usage but would also improve ε to $\delta/4$. It only requires a slightly modified point generation function. On the other hand, the new scheme would only leave a corridor of width about $\delta/2$ to place triangles. This would probably have a negative impact on the size of the resulting mesh. For the current scheme the width is about $2\delta/3$.

It is clear that the voxel generation can be easily parallelized. For instance, each CPU may be ded-

icated to a certain volume in space. The resulting octrees may be merged afterwards. That is, it is easy to parallelize the currently most time consuming part of the algorithm. An obvious alternative is to move the voxel generation to the GPU.

So far our implementation can not handle input meshes that are not manifolds. However, this is just due to missing case distinction in the current code handling the culling. In principal the approach is able to easly handle meshes that are not watertight or have other topological inconsistencies. For instance, the voxelization would fill the gap if it is smaller than ε. Note that this also means that the approach ignores narrow tunnels to maybe rather large caves inside the model. In case the tunnel has diameter around delta the hull computation explores the cave, but it may appear as a closed void since the tunnel is closed due to the subsequent offset computations. Note that this does not contradict our guarantees since we are only interested in the one-sided Hausdorff-distance.

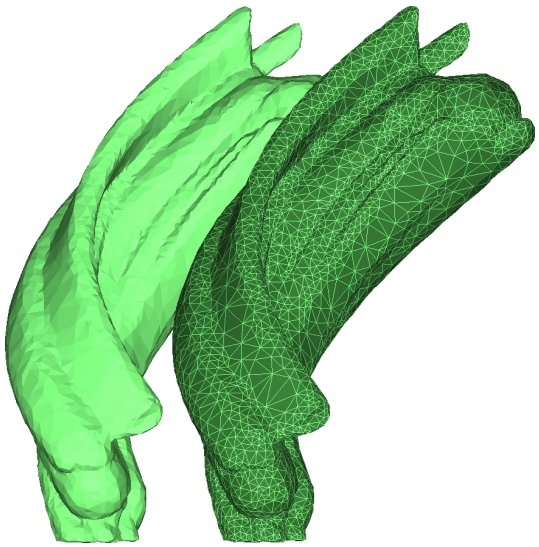

Figure 5: Bunny swept along a trajectory consisting of 50 transformations.

6 Conclusion

To the best of our knowledge this is the first conservative approache for swept volume mesh genearation that also guarantees an approximation quality in terms of the one-sided Hausdorff distance. The resultings meshes are of high quality and very reasonable size, which makes them ideal for further processing in industrial applications. Moreover, our prelimiary benchmark results indicate that the approach should even applicalbe for very large inputs, in particular, once the parallization of the initialization phase is in place.

For recent improvements, more examples and other supplementary material we refer to our website: `http://acg.cs.tau.ac.il/projects/internal-projects/swept-volume/`.

Acknowledgments

We thank R. Erbes for supplying us with the voxelization algorithm for triangles.

References

[1] K. Abdel-Malek, D. Blackmore, and K. Joy. Swept volumes: Foundations, perspectives, and applications. *International Journal of Shape Modeling*, 23(5):1–25, 2004.

[2] S. Abrams and P. K. Allen. Computing swept volumes. *Journal of Visualization and Computer Animation*, 11:69–82, 2000.

[3] M. Campen and L. Kobbelt. Polygonal boundary evaluation of minkowski sums and swept volumes. In *Eurographics Symposium on Geometry Processing (SGP 2010)*, 2010.

[4] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 3rd edition, 2008.

[5] S. F. Frisken and R. N. Perry. Simple and efficient traversal methods for quadtrees and octrees. *Journal of Graphics Tools*, 7(3):1–12, 2002.

[6] J. C. Himmelstein, E. Ferre, and J.-P. Laumond. Swept volume approximation of polygon soups. In *ICRA*, pages 4854–4860, 2007.

[7] Y. J. Kim, G. Varadhan, M. C. Lin, and D. Manocha. Fast swept volume approximation of complex polyhedral models. In *SM '03: Proceedings of the eighth ACM symposium on Solid modeling and applications*, pages 11–22, New York, NY, USA, 2003. ACM.

[8] L. Rineau and M. Yvinec. *3D Surface Mesher*, 3.2 edition, 2006. CGAL User and Reference Manual.

[9] A. von Dziegielewski, R. Erbes, and E. Schömer. Conservative swept volume boundary approximation. In *SPM '10: Proceedings of the 14th ACM Symposium on Solid and Physical Modeling*, pages 171–176, New York, NY, USA, 2010. ACM.

[10] S. Zelinka and M. Garland. Permission grids: Practical, error-bounded simplification. *ACM Transactions on Graphics*, 21:2002, 2002.

[11] X. Zhang, Y. J. Kim, and D. Manocha. Reliable sweeps. In *SPM '09: 2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling*, pages 373–378, New York, NY, USA, 2009. ACM.