Conservative Swept Volume Boundary Approximation

A. von Dziegielewski
Johannes
Gutenberg-Universität Mainz
Staudinger Weg 9
Mainz, Germany
dziegiel@uni-mainz.de

R. Erbes
Johannes
Gutenberg-Universität Mainz
Staudinger Weg 9
Mainz, Germany
erbes@uni-mainz.de

E. Schömer
Johannes
Gutenberg-Universität Mainz
Staudinger Weg 9
Mainz, Germany
schoemer@uni-mainz.de

ABSTRACT

We present a novel technique for approximating the boundary of a swept volume. The generator given by an input triangle mesh is rendered under all rigid transformations of a discrete trajectory. We use a special shader program that creates offset geometry of each triangle on the fly, thus guaranteeing a conservative rasterization and correct depth values. Utilizing rasterization mechanisms and the depth buffer we then get a conservative voxelization of the swept volume (SV) and can extract a triangle mesh from its surface. This mesh is simplified maintaining conservativeness as well as an error bound measured in terms of the one-sided Hausdorff distance. For this we introduce a new technique for tolerance volume computation. The tolerance volume is implicitly given through six 2D-textures residing in texture memory and is evaluated in a special shader program only when needed.

Categories and Subject Descriptors

I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling

General Terms

Algorithms, Design

1. INTRODUCTION

The swept volume (SV) is the entity of all points touched by a solid (the generator) under the transformations of either a continuous or discrete trajectory. It plays an important role in NC machining verification, robot analysis and graphical modeling. Car manufacturers are often interested in swept volumes, e.g. for distance computations and clearance checks between vibrating engine parts, maintainability analysis or motion path planning. Due to increasing model complexity and high demands concerning error tolerance, algorithms, in the fields of computer aided design (CAD) and digital mock up (DMU) have to process masses of data. Since the input models often lack topological qualities, a practical algorithm has to be stable and preferably imposes no restrictions on the input models. For safety reasons it is furthermore important, that an algorithm never underestimates the actual SV, i.e. the approximation has to be conservative.

2. PREVIOUS WORK

Mathematical formulations describing the swept volume include Jacobian rank deficiency methods (singularity theory) [3, 2], sweep differential equations [5, 9] and envelope theory [19, 25], for a survey see [1]. Due to the high complexity of our input data and the often topologically malformed input meshes, these approaches are not easily applicable.
The authors in [20, 8] classify points belonging to the swept volume boundary by testing intersections of a point swept along the inverted trajectory with the generator in its initial position, but they impose restrictions on the trajectory. Numerical approximation of swept volume boundaries utilizing implicit modeling was introduced in [21]. The minimal distance to the swept generator is stored using a signed distance field, from which an approximation to the SV can be extracted as an isosurface. In [16, 11] this implicit approach is extended to computing a directed distance field relative to the tessellated ruled and developable surfaces arising from sweeping polyhedral surface primitives on a parametric trajectory, using the GPU. The authors compute this directed distance field utilizing the GPU and extract the SV via the extended marching cubes algorithm [17]. In a recent work [27], the authors propose a less memory consuming adaptive signed distance field. They can guarantee that the output mesh has the same topology as the exact SV and can give geometric bounds in terms of the (two-sided) Hausdorff-distance. However, since our output meshes serve for clearance and distance checks, the one-sided Hausdorff-distance is the appropriate (less restrictive) choice. Another approach for reducing memory consumption is to store only 2D images of the sweep and reconstruct an approximation of the SV from them. These image based SV-approximations, that often utilize the depth buffer, were first introduced in [23]. They were used for displaying purposes [14, 13, 24] as well as for 3D mesh extraction [4]. In [4] the authors merge depth buffer images from arbitrarily positioned cameras to generate a point cloud of the swept volume boundary from which a mesh can be extracted.
Our method can be seen as an extension of the latter work, yielding an easy to implement practical method for conservative error bounded SV approximation that poses no restrictions on the input triangle meshes and the trajectories

and is absolutely stable. We believe that it is one of a few practical algorithm that is able to handle mass input data.

3. PROBLEM DEFINITION, OVERVIEW

A sequence of rigid body transformations $(\mathcal{R}_i)_{i\in\mathcal{I}}$ is called a **trajectory**, with $\mathcal{I}$ being some index set. Let $\mathcal{G} \subset \mathbb{R}^3$ be a compact set. We define the **swept volume** $SV(\mathcal{G}, (\mathcal{R}_i)_{i\in\mathcal{I}})$ of the **generator** $\mathcal{G}$ to be the entity of all points in $\mathbb{R}^3$ that are touched by $\mathcal{G}$ during the sweep:

$$\begin{aligned} SV(\mathcal{G}, (\mathcal{R}_i)_{i\in\mathcal{I}}) &= \left\{ \tilde{a} \in \mathbb{R}^3 \,\middle|\, \exists i \in \mathcal{I},\, \exists a \in \mathcal{G},\, \tilde{a} = \mathcal{R}_i(a) \right\} \\ &= \bigcup_{i\in\mathcal{I}} \mathcal{R}_i(\mathcal{G}). \end{aligned}$$

Although usually in literature a trajectory is defined to represent a continuous motion, the reason for our discrete definition is that most trajectories we use are actually real world measured data from a car manufacturer. One can think of a continuous motion, densely sampled at uniform time steps.

The overall idea of our method is as follows: For every transformation of the trajectory we render the input generator triangle mesh to six depth buffers with a special shader program applied that achieves a conservative rasterization, without clearing the depth buffers. From these six depth buffer images we can extract a voxelization that overestimates the actual SV. We can give bounds on this overestimation in convex regions of the SV and in all concavities seen from at least one of the six directions. The triangle mesh extracted from the voxelization boundary is highly over-tessellated. We therefore define a conservative tolerance volume that the mesh must stay inside during a following simplification algorithm.
Given a set $\mathcal{A} \subset \mathbb{R}^3$ and $\delta > 0$ we define an **offset** $\mathcal{O}_\delta(\mathcal{A})$ of $\mathcal{A}$ to be the Minkowski sum of $\mathcal{A}$ and the solid ball $\mathcal{B}_\delta(0)$ centered at the origin,

$$\mathcal{O}_\delta(\mathcal{A}) = \mathcal{A} \oplus \mathcal{B}_\delta(0) = \left\{ x \,\middle|\, \exists a \in \mathcal{A} \;: \|x - a\| \leq \delta \right\}$$

and an **offset surface** of $\mathcal{A}$ to be its boundary $\mathcal{S}_\delta(\mathcal{A}) = \partial\mathcal{O}_\delta(\mathcal{A})$. We call δ **offset radius**. Given $\mathcal{A} \subset [0,1]^3$, closed, a **grid resolution** $n \in \mathbb{N}$ and a uniform axis-oriented grid $G_n = \left\{\frac{1}{2n}, \frac{3}{2n}, \ldots, \frac{2n-1}{2n}\right\}^3$, we call a voxel-set $\mathcal{V}_I$ on G_n (the midpoint of a voxel V_{g_i} coincides with $g_i \in G_n$) a **voxelization of** $\mathcal{A}$ if for all $a \in \mathcal{A} \cap G_n$ there exists an index $i \in I$ such that $a \in V_{g_i}$. We say a voxelization $\mathcal{V}_I$ is **minimal** if there does not exist a voxelization of $\mathcal{A}$, that is a proper subset of $\mathcal{V}_I$. We further call a voxelization $\mathcal{V}_I$ **conservative** if for all $a \in \mathcal{A}$ there exists an index $i \in I$ such that $a \in V_{g_i}$. Finally, a conservative voxelization $\mathcal{V}_I$ is said to be **minimal-conservative** if no voxels can be removed from $\mathcal{V}_I$ without violating the conservative-condition. It is easy to see that given a closed set $\mathcal{A} \subset [0,1]^3$, a uniform axis-oriented grid G_n and an offset radius $\delta \in \mathbb{R}$, greater than half the length of a voxel-diagonal $\delta > \frac{\sqrt{3}}{2n}$, the minimal voxelization $\mathcal{V}_I$ of $\mathcal{O}_\delta(\mathcal{A})$ is a conservative voxelization of $\mathcal{A}$. However, in general $\mathcal{V}_I$ ist not the minimal-conservative voxelization of $\mathcal{A}$, for a counterexample see [12]. Given a set $\mathcal{A}$ we want to define a voxelized volume around $\mathcal{A}$ that neither deviates too far from, nor intersects it. This will be a tolerance volume that our output mesh must stay inside throughout the following simplification process.

Let $\mathcal{A} \subset [0,1]^3$ be a closed set and let $\delta > \epsilon > 0$. We define a **conservative tolerance volume** of $\mathcal{A}$ to be the set $\mathcal{O}_\epsilon(\mathcal{S}_\delta(\mathcal{A}))$. One can show that the volume satisfies the demands stated above:

THEOREM 1. *Let $\mathcal{A} \subset [0,1]^3$ be a closed set and $\mathcal{M} \subset \mathcal{O}_\epsilon(\mathcal{S}_\delta(\mathcal{A}))$, with $\delta > \epsilon > 0$. The following assertions hold:*

1. *(Conservativeness) The conservative tolerance volume does not intersects the set itself,*
$$\mathcal{M} \cap \mathcal{A} = \emptyset.$$
2. *No point in the conservative tolerance volume deviates from the set more than $\epsilon + \delta$,*
$$h(\mathcal{M}, \mathcal{A}) \leq \epsilon + \delta.$$

The proof is omitted for the sake of brevity. It further holds:

COROLLARY 2. *Given a closed set $\mathcal{A} \subset [0,1]^3$, a uniform axis-oriented grid G_n and $\delta > \epsilon > \frac{\sqrt{3}}{2n}$. Further let $\mathcal{V}_I$ be the minimal voxelization of $\mathcal{O}_\delta(\mathcal{A})$ Then it holds that*
$$\partial\mathcal{V}_I \subset \mathcal{O}_\epsilon(\mathcal{S}_\delta(\mathcal{A})).$$

Corollary 2 justifies the use of the tessellated voxelization-surface as a starting mesh for further simplification steps. As long as we make sure, that during the simplification we stay inside the conservative tolerance volume, Theorem 1 guarantees conservativeness and the desired error bound on the output mesh. We want to emphasize that in the following, $\mathcal{A}$ must not be mistaken for the actual SV. It is rather a superset of the SV that misses concavities not seen from any of the six depth buffer directions.

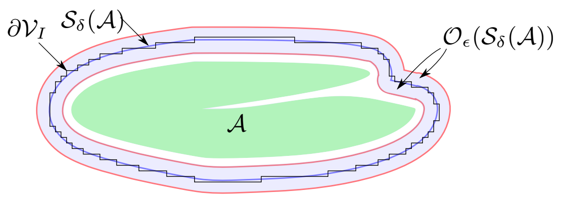
$\partial\mathcal{V}_I$
$\mathcal{S}_\delta(\mathcal{A})$
$\mathcal{O}_\epsilon(\mathcal{S}_\delta(\mathcal{A}))$
$\mathcal{A}$

Figure 1: Compact set $\mathcal{A}$ with an offset surface $\mathcal{S}_\delta(\mathcal{A})$, the voxelization of the offset surface $\partial\mathcal{V}_I$ and a conservative tolerance volume $\mathcal{O}_\epsilon(\mathcal{S}_\delta(\mathcal{A}))$.

4. CONSERVATIVE SV-BOUNDARY VOXELIZATION

Karabassi et al. [15] describe an efficient depth buffer based voxelization algorithm that relies on the rasterization mechanism of graphics hardware. The main idea behind their approach is to generate a depth buffer image of the object for each of the six normal directions of the cube's faces and then extract those voxels that are inside the bounds given through the six images.
Assuming the object to be voxelized fits into the unit cube $[0,1]^3$, we render the object with orthographic projection from each of the three principal axes twice, with an $n \times n$-viewport set up to match the faces of the cube. The depth

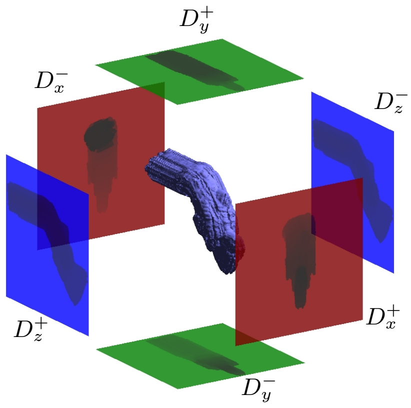

Figure 2: Three pairs of sweep depth buffers images.

buffer images $\{D_x^+, D_x^-, D_y^+, D_y^-, D_z^+, D_z^-\}$ now hold information on the regularly sampled directed distance values from each unit cube-surface to the object's surface in the respective direction. A grid point $g_{(i,j,k)} \in G_n$ lies within the (solid) voxelization if the following three conditions hold:

$$\begin{aligned} D_x^-(k, j) \leq (g_{(i,j,k)})_x \leq D_x^+(k, j) \\ D_y^-(i, k) \leq (g_{(i,j,k)})_y \leq D_y^+(i, k) \\ D_z^-(i, j) \leq (g_{(i,j,k)})_z \leq D_z^+(i, j). \end{aligned} \qquad (\star)$$

A voxelization $\mathcal{V}_I$ of the object on the uniform axis-oriented grid G_n with n^3 grid points can now be extracted [15]. The extension of the voxelization algorithm for the computation of swept volumes is straightforward. The key is to render the object to all six depth buffers for each transformation without clearing the depth buffers (Figure 2). The voxels extracted according to ($\star$) from these sweep depth buffer images form a voxelization of a superset of the volume swept by the generator:

$$\bigcup_{i \in \mathcal{I}} \mathcal{R}_i(\mathcal{G}).$$

The method has two major drawbacks:
First, only convex parts of the SV and concavities seen from at least one direction are voxelized correctly. Attempts to overcome this problem have been made [4, 18]. They are primarily based on the use of more depth buffer images from other, arbitrary viewing directions and adding more depth buffer images for each direction to allow for a higher depth complexity. Unfortunately it causes the algorithmic complexity to become proportional to the depth complexity of the object rendering this approach impractical for our applications with highly complex meshes. Since our primary application of the method lies in the computation of swept volumes of vibrating engine parts for later clearance checks, we do not consider this to be crucial and leave it to the user to ensure that the critical parts are visible to at least one of the depth buffer directions.
The second drawback is that like many voxelization algorithms utilizing graphics hardware, this algorithm heavily relies on the rasterization mechanism which is not conservative [10], hence we cannot expect the resulting voxelization to be conservative. According to the OpenGL specifications [22] a pixel is rasterized if its midpoint is occluded by the geometry, therefore the resulting voxelization is a minimal voxelization of a superset of the rendered model, that misses all concavities not visible to at least one depth buffer image.

From Section 3 we know that a conservative voxelization of a set $\mathcal{A}$ is given by the minimal voxelization of the offset $\mathcal{O}_\delta(\mathcal{A})$. So if we render not the object itself to the depth buffers, but an adequate offset, we can ensure to have a conservative voxelization.
An attempt to overcome the problem of nonconservative rasterization that is similar to our approach can be found in [10]. The authors use a combination of a geometry and a fragment shader to achieve conservativeness, but their method only applies rather coarse depth values, and their computation highly overestimates depth values for triangles with normals perpendicular to the viewing plane. Further their method cannot render arbitrary offsets. In the following we present a new rendering technique, that overcomes this inaccurate depth value estimation and therefore we can give proper bounds on the deviation from each produced fragment to the original geometry.
Given $a, b, c \in \mathbb{R}^3$ we denote $T_{(a,b,c)}$ to be the **triangle** with its **vertices** a, b, c and **edges** e_{ab}, e_{bc}, e_{ac}. For an edge e_{ab} and $\delta > 0$ we define the **solid cylinder** $C_\delta(e_{ab})$ with axis e_{ab} and radius δ to be the set

$$C_\delta(e_{ab}) = \Big\{ x \in \mathbb{R}^3 \Big| \quad \Big\|(x-a) \times \tfrac{b-a}{\|b-a\|}\Big\|^2 \leq \delta^2) \\ \wedge \quad 0 \leq \tfrac{(x-a)\cdot(b-a)}{\|b-a\|^2} \leq 1 \Big\}.$$

For a given triangle $T_{(a,b,c)}$ with **normal** $n = \frac{(b-a)\times(c-a)}{\|(b-a)\times(c-a)\|}$ we define an **offset-prism** of $T_{(a,b,c)}$ to be the convex hull of the two shifted triangles $T_{(a+\delta n, b+\delta n, c+\delta n)}$ and $T_{(a-\delta n, b-\delta n, c-\delta n)}$:

$$P_\delta(T_{(a,b,c)}) = \mathrm{CH}\{ \quad a+\delta n,\, b+\delta n,\, c+\delta n, \\ a-\delta n,\, b-\delta n,\, c-\delta n\}.$$

The offset to a triangle $T_{(a,b,c)}$ equals the union of three spheres, three cylinders and a prism:

$$\mathcal{O}_\delta(T_{(a,b,c)}) = \quad P_\delta(T_{(a,b,c)}) \cup C_\delta(e_{ab}) \cup C_\delta(e_{bc}) \\ \cup C_\delta(e_{ac}) \cup \mathcal{B}_\delta(a) \cup \mathcal{B}_\delta(b) \cup \mathcal{B}_\delta(c).$$

The straightforward method to compute and tessellate this geometry for every triangle, is not practical in the case of a high triangle count. We therefore present a new approach to efficiently compute and render the exact offset geometry up to pixel resolution. The original mesh can easily be compiled in a display list and resides in graphics memory. A combination of a geometry shader and fragment shader computes the depth-values of fragments of the offset boundary on the fly, hence no extra storage is needed.
We process every triangle with the following pipeline: First the vertex shader passes on the vertices of the triangle to the geometry shader stage without changing anything. The geometry shader creates a patch for each edge of the triangle in the xy-plane and also emits the two triangles of the offset prism in 3D. The patches of the edges however still have a depth value according to $z = 0$, hence we calculate the correct depth value for each patch-fragment with a ray casting method. We now give a detailed description of the shaders:
The geometry shader is able to create new primitives (in our case triangles) that are passed on to the rasterization stage just as primitives directly rendered by OpenGL. In the general case we emit eight triangles per input triangle: two for each edge-patch and two triangles for the offset prism. For each edge e_{ab} of the triangle we compute a rectangular

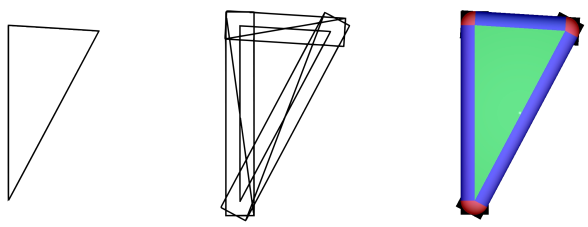

Figure 3: For each edge of the triangle projected to the xy-plane, a patch is constructed in the geometry shader stage and the appropriate depth values are computed in the fragment shader (blue, red). Together with the shifted triangle (green), the final offset surface is rendered, and dispensable fragments (black) are clipped.

patch that is a bounding box of the orthogonal projection of the δ-balls $\mathcal{B}_\delta(a)$, $\mathcal{B}_\delta(b)$ and the δ-cylinder $C_\delta(e_{ab})$ onto the viewing plane. The geometry shader computes and emits the triangles of the tessellated patch. The triangles of the δ-offset prism can be computed by shifting the vertices a, b, c by δ once in normal direction and once in the opposite direction. The rectangular faces of the prism are not needed, since they are completely occluded by the cylinders.

The fragment shader has to discard fragments that are outside the offset and apply correct depth values to all others. In the case of a fragment belonging to a triangle of the prism, the z coordinate of the built-in fragment coordinates is correctly interpolated by hardware and can be mapped directly to the output depth value just like in the fixed function pipeline. For the computation of the correct depth value for a patch-fragment f we define the line $\tilde{e}_{ab}$ as the edge e_{ab} extended to infinity and the endless cylinder $C_\delta(\tilde{e}_{ab}) = \mathcal{O}_\delta(\tilde{e}_{ab})$, given in implicit form:

$$C_\delta(\tilde{e}_{ab}) = \left\{ p \in \mathbb{R} \;\middle|\; \left\|(p-a)\times(b-a)\right\|^2 \leq \delta^2 \|b-a\|^2 \right\}.$$

The intersection of the ray

$$r(\lambda) = \begin{pmatrix} f_x \\ f_y \\ 0 \end{pmatrix} + \lambda \begin{pmatrix} 0 \\ 0 \\ -1 \end{pmatrix}, \quad \lambda > 0$$

with $\partial C_\delta(\tilde{e}_{ab})$ yields the quadratic equation

$$\left\|(f - a - \lambda e_z) \times (b-a)\right\|^2 = \delta^2 \|b-a\|^2$$

that we can solve for λ.

So far we have only considered an endless cylinder with axis e_{ab}. To get the correct offset $\mathcal{S}_\delta(e_{ab})$, we need to clip the cylinder at a and b and intersect $r(\lambda)$ with the balls $\mathcal{B}_\delta(a)$ or $\mathcal{B}_\delta(b)$ respectively. If the ray neither intersects the (clipped) cylinder, nor the balls, we discard the fragment. This only happens in the corners of the patch, as shown in Figure 3.

Using this shader programm we are able to render offset surfaces for highly complex meshes at interactive rates on consumer graphics cards (GeForce GTX 280). A comparable non-GPU-approach, to calculate and tessellate offset spheres, cylinders and prisms at a preprocessing stage for later rendering, is not only significantly slower and less accurate (due to tessellation errors) but would produce an exorbitant number of triangles and therefore exceed the rendering capacity of even modern graphics hardware.

For a conservative swept volume voxelization using the aforementioned algorithm, we need to compute and render the offset

$$\mathcal{O}_\delta(SV(\mathcal{G}, (\mathcal{R}_i)_{i\in\mathcal{I}})),$$

which is problematic, since

$$SV(\mathcal{G}, (\mathcal{R}_i)_{i\in\mathcal{I}})$$

is implicitly described, hence not given through a triangle mesh. The solution lies in the definition of the offset and the distributivity of the Minkowski sum:

$$\mathcal{O}_\delta(SV(\mathcal{G}, (\mathcal{R}_i)_{i\in\mathcal{I}})) = \bigcup_{i\in\mathcal{I}} \Big(\mathcal{O}_\delta(\mathcal{R}_i(\mathcal{G}))\Big).$$

We see that it makes no difference if we voxelize the offset of a swept generator or voxelize the swept offset of the generator.

5. MESH EXTRACTION AND SIMPLIFICATION

We can easily extract a triangle mesh from the surface of a voxelization, as computed in the last section, since it consists of rectangular voxel-faces. But it is highly over-tessellated for fine voxelizations and therefore impractical for later usage, hence a rigorous triangle decimation is inevitable. We present in this section a simplification method that retains a one-sided Hausdorff distance error bound and fits perfectly into our framework, as it uses the six depth buffer images to compute the tolerance volume.

The mesh extracted from the voxelization does not self intersect, has correctly oriented triangles and has neither holes nor T-junctions. However, in general it is not 2-manifold, since it lacks two topological properties: A vertex can be adjacent to more than one triangle-fan and an edge can be adjacent to more than two triangles. Since in practice these degeneracies have shown to occur rarely, we call the resulting mesh *mostly manifold* according to [7].

As we are primarily interested in constructing a mesh for further clearance or distance checks we do not require the mesh to have any topological qualities other than being watertight.

We need a mesh-simplification method that is able to cope with the two degeneracies mentioned above and does not rip holes in the mostly manifold mesh. According to [7] the edge collapse operator fits adequately. During an edge collapse two adjacent vertices are merged and the two (or more in the case of degeneracies) triangles bounded by the edge are deleted. However, due to the locality of the operation, maintaining a global error bound during multiple edge collapses is challenging.

We follow an approach similar to [26, 6]. The authors propose to generate a tolerance volume around the input mesh in a preprocessing step that does not deviate from the surface more than ϵ. They check after each candidate simplification step, if the affected triangles are still inside the volume, i.e. if the candidate edge collapse is valid. If not, the simplification step is reversed.

This ensures, that the directed Hausdorff distance from all triangles produced during the simplification process to the initial mesh is bounded by ϵ.

In [6] the check is done utilizing the GPU: The tolerance

volume is implicitly given by a precomputed distance field, that resides in GPU memory in form of a 3D texture. All triangles affected by a candidate simplification step are rendered and a special fragment shader maps the corresponding distance value to each fragment, i.e. the distance field is evaluated on the triangles. Only the fragments outside the distance bound ϵ are written to the framebuffer and detected using occlusion queries. A simplification step is valid, if no fragment passes the occlusion test.
A problem arises in the case of sharp features (smaller than the sampling density $\ll \frac{\sqrt{3}}{2n}$) that can be missed, since the error is only measured in terms of the directed Hausdorff distance. However, this is not a problem in our case as we are only voxelizing offset surfaces (with $\delta > \frac{\sqrt{3}}{2n}$), that have no sharp features by definition.
Still, this approach has two drawbacks: Both, computational time and storage for the distance field, grow as n^3 with n being the grid resolution.
We can circumvent these two drawbacks, since the inside-outside condition ($\star$) partitions $[0,1]^3$ into the interior of the approximate SV and the exterior. To decide if an inner (outer) point p lies inside the conservative tolerance volume, we check whether the ball $\mathcal{B}_\epsilon(p)$ contains at least one point from the outer (inner).
This check is done in the fragment shader by testing uniformly distributed sample points in $\mathcal{B}_\epsilon(p)$. This way it can happen that a fragment is falsely classified to be outside the tolerance volume, but we will never falsely consider a fragment to be inside the tolerance volume. Therefore we can assure that the simplified mesh stays within the tolerance volume. The six depth buffer images from Section 4 are copied to GPU texture memory and accessed by each fragment independently via OpenGL texture sampling.
Unfortunately, sampling of textures introduces inaccuracies, since we have only the correct depth values for the texel-midpoints. An inner (outer) sample point can be falsely classified to be outside (inside), because the midpoint of its containing voxel (given through the 6 texels) is flagged outside. Therefore the maximum error introduced by texture-sampling is $\sigma_1 = \frac{\sqrt{3}}{2n}$, with n being the grid resolution.
To guarantee a uniform rasterization of the triangles for the fragment generation, each triangle is mapped to the standard triangle scaled with the length of the longest edge of the triangle. The texture coordinates are set accordingly to the actual positions of the triangle's vertices in $[0,1]^3$ and passed on to the shaders. The sampling of the scaled standard triangle introduces further inaccuracies: the maximum spatial deviation from a point on the a triangle to its nearest sample-point is at most $\frac{\sqrt{2}}{2n}$. Altogether the overall maximum sampling error turns out to be $\sigma = \sigma_1 + \sigma_2 = \frac{\sqrt{2}+\sqrt{3}}{2n}$.

6. ERROR ANALYSIS

Due to the additional sampling error σ, we choose an offset radius δ used for the conservative voxelization of the swept volume, and a tolerance ϵ, such that

$$\delta > \epsilon + \sigma \text{ and } \epsilon > \frac{\sqrt{3}}{2n}.$$

with n being the resolution of the voxelization grid and δ being the offset radius.
Following Corollary 2 we chose the starting mesh to be the tessellated boundary of the voxelization $\mathcal{M}_S = \partial\mathcal{V}_I$ that lies inside the conservative tolerance volume $\mathcal{M}_S \subset \mathcal{O}_\epsilon(\mathcal{S}_\delta(\mathcal{A}))$. It holds the conditions from Theorem 1, i.e. the starting mesh $\mathcal{M}_S$ does not intersect the swept volume $\mathcal{M}_S \cap \mathcal{A} = \emptyset$ and no point of $\mathcal{M}_S$ deviates from the swept volume more than $\epsilon + \delta$

$$h(\mathcal{M}_S, \mathcal{A}) \leq \epsilon + \delta.$$

We then apply the simplification method on $\mathcal{M}$ described in Section 5. We call $(\mathcal{M}_j)$ the sequence of meshes emerging in the course of the simplification procedure. Since we took care of the additional maximum inaccuracy σ when choosing the offset radius, the mesh simplification algorithm maintains conservativeness for all $\mathcal{M}_j$. We see this by replacing ϵ with $\epsilon + \sigma$ in Theorem 1. Unfortunately this also means that we have

$$h(\mathcal{M}_j, \mathcal{A}) \leq \epsilon + \delta + \sigma, \text{ for all } j.$$

Hence, when setting the parameters ϵ and δ appropriately, our method guarantees the resulting mesh to be a conservative approximation of the swept volume boundary, that does not deviate more than $\epsilon + \delta + \sigma$ from the swept volume at all convex areas and visible concavities.
In practical applications we choose tight bounds, i.e. $\epsilon = \frac{\sqrt{3}}{2n}$ and $\delta = \frac{2\sqrt{3}+\sqrt{2}}{2n}$. We therefore get a global error

$$\epsilon_{global} = \frac{2\sqrt{3}+\sqrt{2}}{n}.$$

Setting ϵ_{global} as an a priori error bound yields a sufficient grid resolution of

$$n = \frac{2\sqrt{3}+\sqrt{2}}{\epsilon_{global}}.$$

7. PRACTICAL RESULTS

We demonstrate the capabilities of the method with real world scenarios. We used a 3.0 GHz CPU with 4 GB of RAM and a GeForce GTX 280 with 1 GB.
In the first scenario, the movement of a steering gear relative to a neighboring engine part unit was recorded during a test-drive, in the second the vibration of an engine part (oil pan) was recorded and the third is an engine assembly path that was planned with DMU. The results are shown in Figure 5. The trajectories consist of N_T transformations and models are given as triangle meshes consisting of T_{model} triangles. For the voxelization we used different grid resolutions n. The overall time for our method t_{total} is the sum of the time needed for voxelizing the swept volume t_{vox} and for the subsequent simplification process t_{simp}. The starting mesh consisted of T_{start} triangles and the final output mesh (as shown in Figure 4) of T_{final} triangles. The global error ϵ_{gobal} has to be interpreted as a fraction of the SV's bounding cube edge length.

8. CONCLUSIONS

We have presented a novel technique for approximating the boundary of a swept volume. The algorithm is stable and can handle any kinds of malformed input triangle meshes.
We have presented a new rendering technique for conservative rasterization with correct depth values and a mesh simplification method utilizing a low memory consuming technique.

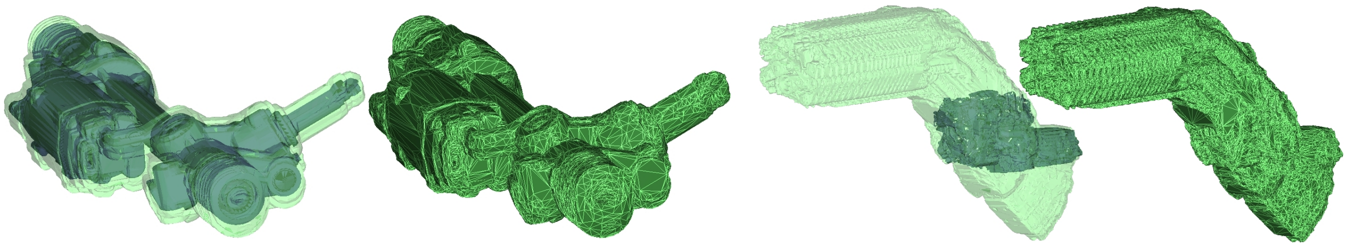

Figure 4: Swept volume approximations of steering gear unit and assembly path ($n = 1500$).

model		trajectory	swept volume boundary computation						
name	T_{model}	N_T	n	$t_{vox}(min)$	$t_{simp}(min)$	$t_{total}(min)$	T_{start}	T_{final}	ϵ_{gobal}
Steering gear	261,807	27,442	1,500	231	37	268	3,478,888	29,916	0.00325
			1,000	168	4	172	1,533,728	10,336	0.00488
			500	167	0.5	168	378,044	3,780	0.00976
Oil pan	102,513	9,112	1,200	29	49	78	9,053,512	26,390	0.00407
			800	27	19	46	5,074,452	19,966	0.00610
Engine assembly	328,417	194	1,500	2	66	68	3,941,092	104,880	0.00325

Figure 5: Results for real world scenarios used for testing our method.

As we took special care of the demands from a car manufacturer, we believe that our method has the potential to be integrated into existing CAD programs.

9. 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] K. Abdel-Malek and H.-J. Yeh. Geometric representation of the swept volume using jacobian rank-deficiency conditions. *Computer-Aided Design*, 29(6):457–468, 1997.

[3] K. Abdel-Malek and H.-J. Yeh. On the determination of starting points for parametric surface intersections. *Computer-Aided Design*, 29(1):21–35, 1997.

[4] J. Ahn and S. J. Hong. Approximating 3d general sweep boundary using depth-buffer. In *ICCSA (3)*, pages 508–517, 2003.

[5] D. Blackmore and M. C. Leu. Analysis of swept volume via lie groups and differential equations. *Int. J. Rob. Res.*, 11(6):516–537, 1992.

[6] M. Botsch, D. Bommes, C. Vogel, and L. Kobbelt. GPU-based tolerance volumes for mesh processing. In *Pacific Conference on Computer Graphics and Applications*, pages 237–243, 2004.

[7] J. D. Cohen. Concepts and algorithms for polygonal simplification. *SIGGRAPH 99 Course Tutorial, Interactive Walkthroughs of Large Geometric Datasets*, 20:C1–C34, 1999.

[8] H. Erdim and H. T. Ilieş. Classifying points for sweeping solids. *Comput. Aided Des.*, 40(9):987–998, 2008.

[9] W. Guoping, S. Jiaguang, and H. Xuanji. The sweep-envelope differential equation algorithm for general deformed swept volumes. *Computer Aided Geometric Design*, 17(5):399 – 418, 2000.

[10] J. Hasselgren, T. Akenine-Möller, and L. Ohlsson. *GPU Gems 2*, chapter Conservative Rasterization, pages 677–690. Addison-Wesley Professional, 2005.

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

[12] J. Huang, R. Yagel, V. Filippov, and Y. Kurzion. An accurate method for voxelizing polygon meshes. In *VVS '98: Proceedings of the 1998 IEEE symposium on Volume visualization*, pages 119–126, New York, NY, USA, 1998. ACM.

[13] Y. Huang and J. H. Oliver. NC milling error assessment and tool path correction. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 287–294, New York, NY, USA, 1994. ACM.

[14] K. Hui. Solid sweeping in image space application in NC simulation. *The Visual Computer*, 10(6):306–316, 1994.

[15] E.-A. Karabassi, G. Papaioannou, and T. Theoharis. A fast depth-buffer-based voxelization algorithm. *J. Graph. Tools*, 4(4):5–10, 1999.

[16] 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.

[17] L. P. Kobbelt, M. Botsch, U. Schwanecke, and H.-P. Seidel. Feature sensitive surface extraction from volume data. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 57–66, New York, NY, USA, 2001. ACM.

[18] G. Passalis, I. Kakadiaris, and T. Theoharis. Efficient hardware voxelization. *Computer Graphics International Conference*, 0:374–377, 2004.

[19] M. Peternell, H. Pottmann, T. Steiner, and H. Zhao. Swept volumes. *Computer-Aided Design Appl.*, 2:599–608, 2005.

[20] J. Rossignac, J. J. Kim, S. C. Song, K. C. Suh, and C. B. Joung. Boundary of the volume swept by a free-form solid in screw motion. *Comput. Aided Des.*, 39(9):745–755, 2007.

[21] W. J. Schroeder, W. E. Lorensen, and S. Linthicum. Implicit modeling of swept surfaces and volumes. In *VIS '94: Proceedings of the conference on Visualization '94*, pages 40–45, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

[22] M. Segal and K. Akeley. *The OpenGL Graphics System: A Specification (Version 2.1)*, chapter 3: Rasterization, page 122 ff. 2006.

[23] T. Van Hook. Real-time shaded nc milling display. *SIGGRAPH Comput. Graph.*, 20(4):15–20, 1986.

[24] A. S. Winter and M. Chen. Image-swept volumes. *Comput. Graph. Forum*, 21(3), 2002.

[25] Z. Xu, Z. Chen, X. Ye, and S. Zhang. Approximate the swept volume of revolutions along curved trajectories. In *SPM '07: Proceedings of the 2007 ACM symposium on Solid and physical modeling*, pages 309–314, New York, NY, USA, 2007. ACM.

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

[27] 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.