2007 IEEE International Conference on
Robotics and Automation
Roma, Italy, 10-14 April 2007

Swept Volume approximation of polygon soups

Jesse C. Himmelstein, *Member, IEEE*, Etienne Ferré, and Jean-Paul Laumond, *Fellow, IEEE*

***Abstract*— We present a fast GPU-based algorithm to approximate the Swept Volume (SV) boundary of arbitrary polygon soup models. Despite the extensive research on calculating the volume swept by an object along a trajectory, the efficient algorithms described have imposed constraints on both the trajectories and geometric models. By proposing a general algorithm that handles flat surfaces as well as volumes and disconnected objects, we allow SV calculation without resorting to pre-processing mesh repair. This is of particular interest in the domain of Product Lifecycle Management (PLM), which deals with industrial Computer Aided Design (CAD) models that are malformed more often than not. We incorporate the bounded distance operator used in path planning to efficiently sample the trajectory while controlling the total error. We develop a triangulation scheme that draws on the unique data set created by an advancing front level-set method to tessellate the SV boundary in linear time. We analyze its performance, and demonstrate its effectiveness both theoretically and on real cases taken from PLM.**

I. Introduction

Although Swept Volumes have been studied for quite some time, their applications within the industrial world of Product Lifecycle Management require a specialized algorithm.

A. Motivation

A Swept Volume (SV) is defined as the totality of points touched by a geometric entity while in motion. Since their introduction in the 1960's, SVs have proved useful in many different areas, including numerically controlled machining verification [1], robot workspace analysis [2], geometric modeling [3], collision detection [4], mechanical assembly [5], and ergonomic studies [6]. We are interested in its applications within Product Lifecycle Management.

Product Lifecycle Management (PLM) is both a business strategy and a technology solution to manage the entire life span of a product, from cradle to grave. Although Computer Aided Design (CAD) and Product Data Management (PDM) systems have been available since the 1980's, the concept of an overarching information management solution encompassing all knowledge about a product and which is intended for all parts of a company (including marketing, sales, and support) appeared only late in 1990's, and is still today gathering speed [7].

PLM both draws on previous research in robotics and poses new challenges [8]. There are two commonly encountered industrial cases in PLM that can greatly benefit from efficient SV calculation:

1) Mechanical assembly and disassembly— the sequence of parts to remove and their corresponding paths can now be calculated entirely within software, without needing to build a physical mockup [5].

2) Ergonomic studies— designs for workspaces are often designed to "fit" their human users. Through analysis of kinematic modeling of the human limbs [6] (otherwise known as "reach envelopes") engineers can test the feasibility of an operational task and create one or more corresponding reaching paths.

Once the paths are generated, engineers often desire to keep them collision-free, easing disassembly/maintenance tasks. The SV can represent the volume of one or more paths, and further placement of parts can be efficiently checked for collisions against it.

PLM designs are often based around CAD data, and these models are infamously malformed (containing degeneracies such as cracks, intersections, wrongly oriented polygons, etc.) [9]. Many geometric algorithms require closed 2-manifold volumes to give meaningful results. Although malformed models can theoretically be transformed into proper volumes, in practice this is both a difficult and time-consuming pre-processing step. In addition, certain models contain flat surfaces surrounding no volume whatsoever. Such models may be dealt with more straightforwardly as polygon soups– unordered sets of triangles with no enforced connectivity constraints.

For this reason, we chose to adapt a state-of-the-art SV approximation algorithm to handle pure polygon soups.

B. Related Work

SV calculation dates back to the 1960s, originally in a 2D context. The problem of calculating the volume is often simplified to finding the boundary of the volume. Even so, the mathematics can be very complex, including self-intersections of the SV. Due to the sheer volume of the work on the subject, we refer the interested reader to the survey by Abdel-Malek *et al.* [10].

Modern analytical approaches include envelope theory [11], singularity theory (a.k.a. Manifold Stratification or Jacobian rank deficiency method) [1, 12, 13], and Sweep Differential Equations [11, 14]. However, the type of data that we are treating does not lend itself easily to

Jesse Himmelstein is a PhD student at LAAS-CRNS under a joint grant with Kineo CAM, 31312 Labège FRANCE (phone: +33 (0)5 61 00 90 60; fax: +33 (0)5 61 00 90 61; e-mail: jhimmel@laas.fr).

Etienne Ferré is with Kineo CAM, 31312 Labège FRANCE (e-mail: ef@kineocam.com).

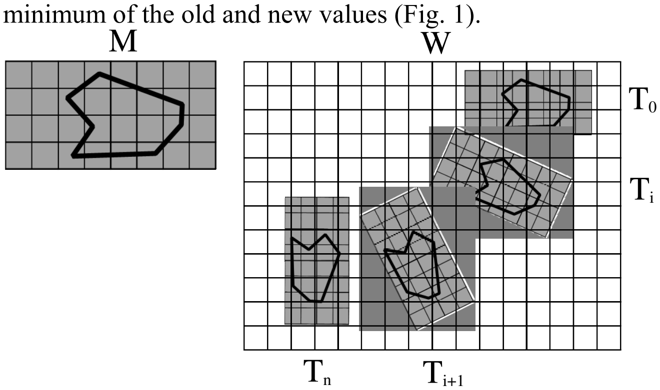
Jean-Paul Laumond is with LAAS-CRNS, 31077 Toulouse FRANCE (e-mail: jpl@laas.fr).

1-4244-0602-1/07/$20.00 ©2007 IEEE.

mathematical analysis.

1) Implicit Surfaces and Distance Fields

Schroeder *et al* [15] introduced another type of method-manipulating numerical approximations of implicit surfaces. They first impose a grid on the model space and assign each grid point a value equal to its distance from the model surface (a set of these values is called a distance field). The workspace (a volume bounding the entire sweep) is imposed a grid as well, with initial distance values of infinity. As the object is swept along its trajectory, the inverse transform of each workspace point is calculated, to find the nearest neighbor points in model space. A new distance value is evaluated as the trilinear interpolation of those model space distance values. The workspace distance value is the minimum of the old and new values (Fig. 1).



$$f(\mathrm{p_W}) = \min(\, f(\mathrm{T_0^{-1} p_W}), \ldots, f(\mathrm{T_n^{-1} p_W})\,)$$

Fig. 1 The inverse method of implicit distance calculation used by Schroeder *et al.* As the model **M** is swept through the workspace **W**, a transformation **T** is applied at each step. For each point $\mathbf{p_W}$ in the workspace, the implicit function $f(\mathbf{p_W})$ is assigned the minimum of all the original distance values transformed there. Since a transformed $\mathbf{p_W}$ will rarely line up exactly on a $\mathbf{p_M}$, trilinear interpolation is used between the closest values.

It is important to note that the distance values for grid points lying inside the volume are given negative values. Thus, the boundary of the SV can be approximated as an isosurface where the distance equals zero. Finally, the Marching Cubes algorithm [16] is used to extract this isosurface at a certain distance value (distances other than zero generate offset surfaces).

These distance fields can be generated through regular sampling of a bounding volume [17]. Such sampling lends itself naturally to the powerful parallel processing capabilities of a graphics card, now commonly referred to as a Graphics Processing Unit (GPU) [18].

2) GPU-based Directed Distances

Kim *et al.* [19] combine and extend these approaches to quickly find the SV boundary using the GPU. They begin with a triangulated mesh and a trajectory composed of rigid motions. The edges and faces of the mesh are treated as ruled and developable surfaces, and triangulated along the trajectory within a certain error threshold. The new object includes the SV boundary, but contains surfaces on the interior of SV as well.

To remove the interior surfaces, the object is split into slices, and a 2D grid is imposed onto each slice (Fig. 2). Using the GPU, distance fields are found along the edges between neighboring grid points. The distance fields are directed (along the 3 major axes), rather than the scalar values as in Schroeder *et al.* They are also unsigned, as there is not yet a notion of interior and exterior.

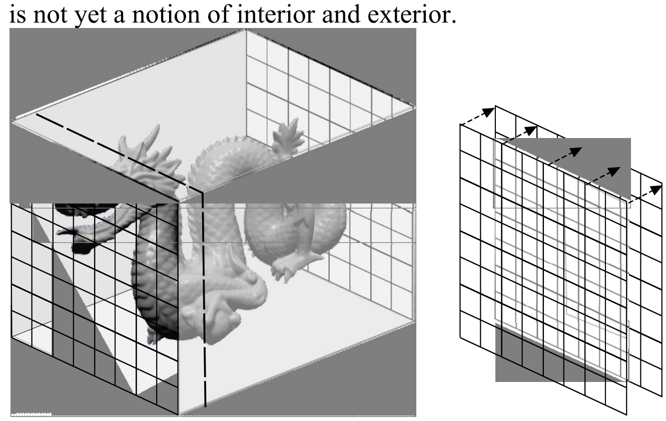


Fig. 2 The workspace is split into slices, and depth measurements performed for each slice. On the left, the dragon model is shown inside a bounding box. That box is split (right), and for each slice (such as from the front to the dashed line), distance fields are taken along the split direction.

The grid points are then classified as outside or inside the SV using a propagating front level set method. Then the surface of the SV is extracted using the Extended Marching Cubes (EMC) algorithm [20], which exploits the directed distance fields and triangle normals to provide a more faithful triangulation than traditional Marching Cubes. The final step is a topological check. If the surface is not evaluated as closed and watertight, the spatial grid is refined and the algorithm executed again.

Their algorithm represents an advancement from that of Schroeder *et al.* both in performance (thanks to the GPU) and quality (through EMC and the absence of interpolation). However, the range of acceptable input is limited; only a single watertight 2-manifold is allowed. This restriction is imposed by the tessellation method and the final topological check. To handle real cases in PLM, critical modifications need to be made. Such modifications constitute the main contributions of our paper.

C. Contributions

Our essential contributions are the following:

- Devising a fast GPU-based SV approximation algorithm to accept arbitrary polygon soups as input.
- Creating a specialized tessellation algorithm to generate meshes in linear time.
- Defining a unified error bound of both mesh size and trajectory sampling based upon the bounded move operator.

D. Outline

The rest of the paper is organized into 7 sections. In Section II, we discuss how the trajectory can be sampled to tightly control error. In Section III, we develop an alternative level-set method to detect the SV surface. Section IV introduces our specialized triangulation algorithm. We analyze the error and complexity of the complete procedure in Section V. Section VI presents our results, and we conclude with ideas for future work in Section VII.

II. Trajectory Sampling

Rather than creating a swept mesh through ruled and developable surfaces as with Kim *et al* [19], we decided to sample the surface at a certain number of intervals. A naïve approach would be to sample the trajectory along regular intervals. However, arbitrary motions can lead to situations where certain points sweep large paths while others barely move. To limit error in this case, it would be necessary to impose a large number of intervals, many of which would be wasted on more constant movements. Additionally, determining the error bound implied by a given number of samples is not trivial (Fig. 3).

A response to this problem lies in [21], where the authors define a *bounded distance* operator on robot paths in the spirit of [22]. Given the set of points in the model **A**, and a continuous function of configurations $\mathbf{q}(s)$, then we can state that the distance between two configurations $\mathbf{q}(s_i)$ and $\mathbf{q}(s_{i+1})$ is bounded by distance ε if:

$$\forall P \in \mathrm{A} \quad / \quad \int_{s_i}^{s_{i+1}} \left\| \frac{dP}{ds}\big(\mathbf{q}(s)\big) \right\| ds < \Delta$$

In other words, the trajectory is bounded by distance Δ if no point in the model moves more than Δ between two successive configurations.

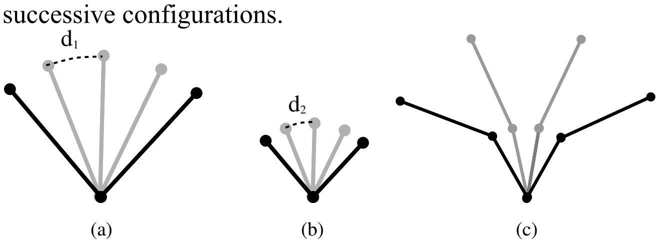


Fig. 3 Problems with regular path sampling. In (a) and (b), a simple straight arm is rotated about one end. Sampling this trajectory uniformly would result in very different error bounds for $\mathbf{d_1}$ in (a) and $\mathbf{d_2}$ in (b). This problem is only aggravated further when kinematic chains, such as (c), are introduced. The bounded distance operator aims at controlling the maximum error.

The bounded distance operator provides a robust and convenient way to control error while minimizing the number of required samples, even with multiple complex objects of arbitrary geometry, such as polygon soups and kinematic chains.

III. Grid Computation

In implicit modeling, a field function $f(\mathbf{p})$ defines a value for each point $\mathbf{p}$ in space. Surfaces are therefore equivalent to contours sharing a field value. In our case, $f(\mathbf{p})$ returns the distance to one of the polygons drawn by the GPU. However, $f(\mathbf{p})$ will be zero for surfaces lying within the SV as well as along the boundary.

When dealing with volumes, it suffices to negate $f(\mathbf{p})$ for all p within the SV. As long as at least one grid point falls within the volume, it creates a difference that can be detected by the discrete surface extraction procedure. However, when dealing with surfaces that do not contain any volume (Fig. 4) this method fails to find the contour.

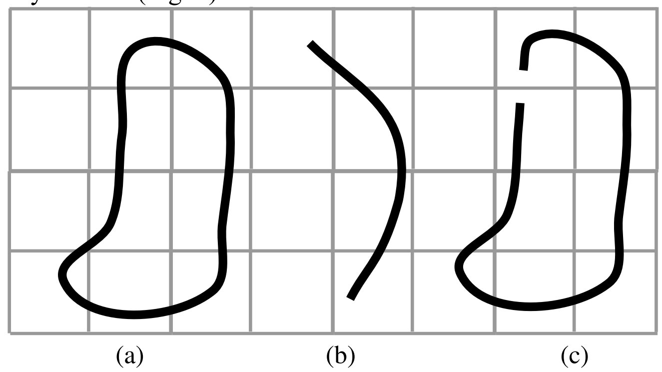


Fig. 4 These three surfaces are treated differently by traditional isosurface extraction. The volume in (a) will be found correctly, but since the surface in (b) does not contain any volume, it will be ignored, as will the volume with a hole found in (c).

To properly detect these surfaces, we modified the fast marching level-set method presented by Lin *et al.* (and in turn based upon [23]). Whereas they use it to simply classify grid points as inside or outside, we employ it to detect the surface itself, and to generate 3D points that are used later by our specialized triangulation algorithm.

All grid points are tagged with a state and a membership, each of which having 3 possible values. The state is initially *Far*, meaning that it has not been reached by the advancing front. A point in the front is in the *Trial* state, and once a point has been analyzed it is *Known*. The membership of untreated points is *Inside*. Once a point is reached by the front, it might be *Superficial* if it lies next to the surface, or *Outside* otherwise.

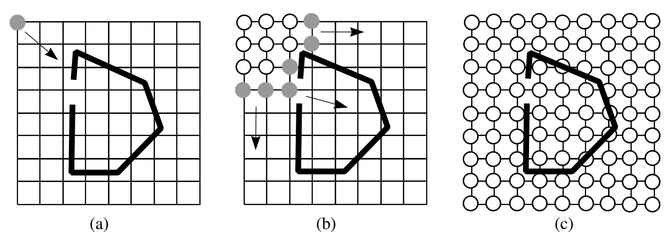


Fig. 5 The advancing front enters holes, rendering Marching Cubes useless. The front, represented by grey circles, starts at the upper left corner in (a), moving down and to the right. In (b), the front has advanced up to the hole in the volume. By the time the front is exhausted, in (c), it has completely filled the space. By labeling all the visited points *Outside*, it is impossible to recover the correct surface. Although the hole is exaggerated here for the purposes of example, the same phenomenon is found with the smallest of imperfections in common PLM models.

In addition, all points have a set of detected surface samples, each of which contains 2 pieces of information: their coordinates in the workspace, and the direction in which they were detected by the advancing front. The members of the starting front (e.g. the limits of the bounding volume) are inserted into a queue and set to *Trial* and *Outside*.

Algorithm 1 describes the iterative procedure that advances the front, labeling grid points as *Outside* or

Superficial as needed. For *Superficial* points, a set of corresponding 3D points is generated, one for each axis upon which the surface was detected. This set is attached to the grid point for use during the triangulation process.

```
while front is not empty
    P = pop(front)
    state(P) = Known
    for each neighbor Q of P:
      u = state(Q) is not Known
      s = membership(Q) is Superficial
      if u or s then
        dir = direction from P to Q
        dist = edge_length(P, Q)
        if slice_width(dir) > dist then
          membership(Q) = Outside
          state(Q) = Trial
          push(front, Q)
        else
          p = generate_3D_point(P, dir, dist)
          push(detected_surfaces(P), p)
          membership(P) = Superifical
        end if
      end if
    end foreach
end while
```

Algorithm 1 Fast Marching Method adapted to recognize surface points

Note that this surface detection algorithm refuses to enter closed volumes, but also recognizes non-volumic surfaces, and is therefore appropriate for any kind of geometrical model, polygon soup or otherwise.

IV. Triangulation

Once the surface points are detected, they can be triangulated. Tessellation has been wildly studied, and there exists many algorithms in the literature that could be used. In particular, algorithms that tessellate point clouds would be appropriate [24, 25]. However, rather than dealing with the detected surface points as a point cloud (Fig. 6), we can exploit the known structure of data returned by the level-set method to achieve linear-time triangulation.

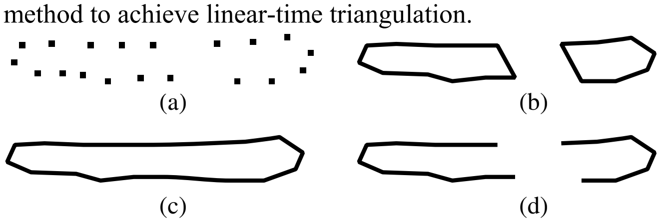


Fig. 6 Different interpretations of a point cloud. The unorganized point cloud in (a) could be interpreted in different ways. In (b) it represents two different objects, whereas in (c) it is only one. The cloud could even be two non-volumic surfaces, as in (d).

A. Building a Graph

For each surface point, we know the grid point from which the surface was detected by the advancing front, and in which direction. Given these two pieces of information, we can determine the neighboring surface points that the point under consideration should be connected to. By connecting each point to all its neighbors, we construct an undirected graph (Fig. 7).

The graph representation defines a valid topological relationship between points in the cloud. Within the graph, each detected surface point is represented by a node. Therefore, a flat surface detected in multiple directions will have multiple nodes corresponding to the same coordinates in the workspace. In other words, the graph always represents a closed 2-manifold volume, rather than a polygon soup.

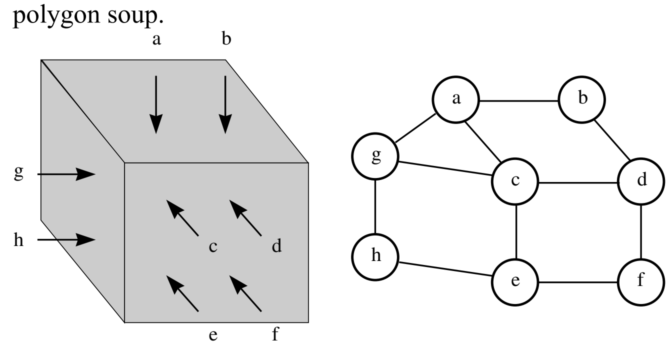


Fig. 7 Graph building. There are 8 points detected on the surface of the box on the left. They are transposed onto the graph on the right, by connecting each point to its neighbors in the grid. By respecting the orientation of undirected edges, this graph completely describes the SV boundary. Note that a complete graph would surround a volume and not have boundary edges like in this example.

B. Tessellation

From here it is fairly easy to tessellate the graph. By following the elementary cycles that do not contain any others, we construct a simple convex loop (of up to 6 points) that can be easily triangulated. For example, in Fig. 7, nodes **c, d, f, and e** form such a cycle. The algorithm terminates when there are no such cycles left.

With this algorithm, all objects (including those detected to have strictly planar sections) are implicitly tessellated as closed polyhedra (although duplicate polygons could be filtered to preserve single-sided planar geometry). In addition, all triangles are oriented to face out of the object.

C. Degeneracies

There are two possible degeneracies that can be created by this process. First, a node that is connected to a neighbor by 2 opposing directions corresponds to a non-manifold edge. Secondly, a node that is connected to a neighbor by all directions corresponds to an isolated line segment or single point.

Since these degeneracies are so easily detected, they can be removed and the edges re-worked around them before the triangulation process is launched.

V. Analysis of Algorithm

To ease the following discussion, we will assume that a cubic bounding volume of size L^3 is divided into M grid points along each axis and that there are n surface points detected. Moreover, there is a total of p triangular primitives, the product of the number of samples and the number of triangles in the object.

A. Computational Complexity

1) Constructing Bounding Volume

To create the bounding volume, we must pass by each primitive three times, a complexity of $O(p)$.

2) Gathering Distance Fields

Like Kim *et al.* [19], we will analyze the performance of distance field generation as proportional to the number of primitives sent to the GPU for rendering. We draw the primitives for each of 6 slicing directions and each of M slices. The size of the grid has an additional effect on both rendering and read-back performance. The complexity is thus $O(M^3 + Mp)$.

3) Advancing Front

The level-set method shows performance proportional to the size of the grid, or $O(M^3)$.

4) Tessellation

Although, for simplicity, we presented the tessellation process in two steps– first building the graph, then enumerating the cycles– it is possible to combine the two. Starting at one detected surface point, we triangulate all of the cycles that it corresponds to. By marking the grid points that we pass by so as to avoid regenerating the same cycles more than once, we can demonstrate algorithmic complexity proportional to the number of primitives. Since a cycle can link no more than a small number of points, triangulating can be done in constant time. The complexity is therefore $O(n)$.

5) Total Complexity

Taking all the steps into account, the total computational complexity of our SV algorithm is $O(M^3 + Mp + n)$. Since n is upper bounded by $6M^3$, we can further simplify the complexity relation to $O(M^3 + Mp)$.

For comparison, the complexity of the method proposed by Kim *et al.* [19] can be expressed as $O(M^3 + g + T)$, where g is the number of triangles in the ruled and developable surface tessellation, and T is the number of rendered triangles. Since they cull triangles so as to render them only for the slices that they occupy, T is likely to be much smaller than Mp in the average case. On the other hand, g is dependant on the given error bound, and thus difficult to predict and compare to our method.

B. Error

There are two potential sources of sampling error in our approximation algorithm: the distance fields and the trajectory. We are able to control both of them with the same parameter. There is an additional error related to hardware, since the GPU depth buffer is used to calculate the distance fields, and so precision is limited by the width of the buffer (modern GPUs have at least 32 bits).

1) Distance Field Error

Consider sampling an object that doesn't move. Given that a measurement is taken M times along each axis of length L., let $S = L / M$, i.e. the length of one side of one cubic cell within the grid.

Between the grid lines, no measurements are taken. Therefore the error is limited to the possible distance the surface could move within the grid cell, which is equal to S (Fig. 8).

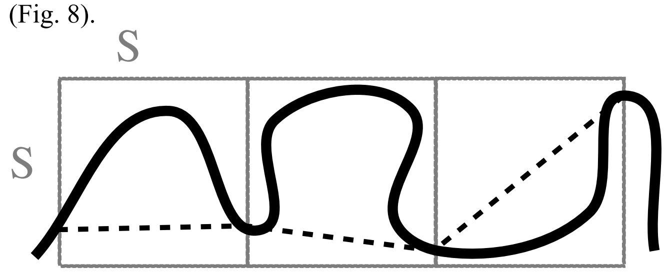


Fig. 8 Distance field sampling error. In the worst case, the actual SV boundary (heavy line) could fluctuate wildly between the grid points. The inaccuracy of the resulting approximation (dashed line) is limited by the size of the grid cell, S.

2) Trajectory Sampling Error

As described in Section II, we can guarantee that no point on the object traces a path longer than Δ between two sampled configurations. Assuming that the point is detected at both configurations, the farthest it could move from the detected surface is $\Delta / 2$.

3) Total Sampling Error

We now take both sources of error into account. In the worst case, the surface will be sampled at a distance S from its real location. In addition, between the two sampled configurations, the surface will have moved $\Delta / 2$. The resulting effect is the sum of the two: $\varepsilon = S + \Delta / 2$.

With this simple relation, the user has the possibility to adjust the speed and memory use of the algorithm while staying within a global error bound.

VI. Experimental Results

A. Implementation

We implemented the SV algorithm in C++, with graphic routines in OpenGL. It is designed as a module for Kineo Path Planner™, and benefits from its stable implementation of the bounded move operator. Since we use only very basic graphic card functionality, any 2nd generation GPU supporting z-buffer and frame-buffer readback suffices.

B. Results

All tests are conducted on a Intel Pentium 4 PC running at 2 GHz with 1GB RAM. The GPU is a nVidia Quadro FX 500 with 128 MB memory. The operating system is Windows XP SP2.

We used 3 models for testing (Table I). All came from actual PLM cases encountered by Kineo CAM™. The performance results are shown in Table II. To illustrate the interplay between the two user-definable parameters, Δ and ε, the *Exhaust* Model is expanded across multiple resolutions and bounded distances.

The visual quality of the SV mesh is quite high, and suitable for PLM purposes (Fig. 9). However, the triangle count of the SV is also important, and so in our application we pass the generated SVs directly through a simplification procedure, such as the vertex decimation algorithm presented in [26].

TABLE I
TEST MODELS

Model	# Triangles
Seat	30765
Exhaust	32641
Human	55632

Despite their appearance, none of the test models are watertight 2-manifold meshes. To further demonstrate the applicability of our algorithm to non-volumic geometry, we sweep a simple flat surface in Fig. 10.

Videos corresponding to these examples are available at http://www.laas.fr/gepetto

VII. Conclusion and Future Work

Our SV approximation algorithm successfully deals with real challenges posed by PLM, including disassembly and ergonomic studies. Its fast execution allows for rapid analysis of the given paths and for subsequent collision detection and path-planning requirements. By relaxing the requirements of watertight 2-manifold geometry, no pre-processing is needed to handle arbitrary CAD models.

There are several areas for future work. The intermediate graph data structure, representing a volumic mesh, could have potential for manipulating the object before it takes on polygon soup form. Even algorithms that require closed watertight geometry could be run at this point.

In addition, we would like to consider introducing sub-sampled points when feature geometry is detected, as is done by algorithms such as EMC. By clever use of surface normals and local grid refinement, we could extend the algorithm to achieve lower error bounds without resorting to global refinement of the grid.

Acknowledgment

We would like to thank the development team at Kineo CAM [27] for their excellent ideas and patient support.

References

[1] K. Abdel-Malek, W. Seaman, and H.-J. Yeh, "NC Verification of up to 5 Axis Machining Processes Using Manifold Stratification," *ASME Journal of Manufacturing Science and Engineering*, vol. 122, pp. 1-11, 2000.

[2] S. Abrams, P. K. Allen, and K. Tarabanis, "Computing Camera Viewpoints in an Active Robot Work Cell," *International Journal of Robotics Research*, vol. 18, pp. 267-285, 1999.

[3] J. Conkey and K. I. Joy, "Using Isosurface Methods for Visualizing the Envelope of a Swept Trivariate Solid," presented at Pacific Graphics, Hong Kong, 2000.

[4] A. Foisy and V. Hayward, "A safe swept volume method for robust collision detection," presented at Robotics Research, Sixth International Symposium, 1994.

[5] C. C. Law, Lisa S Avila, and W. J. Schroeder, "Application of Path Planning and Visualization for Industrial Design and Maintainability Analysis," presented at Reliability and Maintainability Symposium, 1998.

[6] K. Abdel-Malek, J. Yang, R. Brand, and E. Tanbour, "Towards Understanding the Workspace of Human Limbs," *Ergonomics*, vol. 47, pp. 1386-1406, 2004.

[7] F. Ameri and D. Dutta, "Product Lifecycle Management: Closing the knowledge loops," *Computer-Aided Design and Applications*, vol. 2, pp. 577-590, 2005.

[8] J.-P. Laumond, "Motion planning for PLM: state of the art and perspectives," *International Journal of Product Lifecycle Management*, vol. 1, pp. 129-142, 2006.

[9] T. M. Murali and T. A. Funkhouser, "Consistent solid and boundary representations from arbitrary polygonal data," presented at SIGGRAPH Symposium on Interactive 3D Graphics, 1997.

[10] K. Abdel-Malek, D. Blackmore, and K. Joy, "Swept Volumes: Foundations, Perspectives, and Applications," *International Journal of Shape Modeling*, 2002.

[11] R. R. Martin and P. C. Stephenson, "Sweeping of Three-dimensional Objects," *Computer Aided Design*, vol. 22, pp. 223-234, 1990.

[12] K. Abdel-Malek and S. Othman, "Multiple sweeping using the Denavit-Hartenber representation method," *Computer-Aided Design and Applications*, vol. 31, pp. 567-583, 1999.

[13] K. Abdel-Malek and H.-J. Yeh, "On the Determination of Starting Points for Parametric Surface Intersections," *Computer Aided Design*, vol. 29, pp. 21-35, 1997.

[14] D. Blackmore and M. C. Leu, "A differential equation approach to swept volumes," presented at Rensselaer's 2nd International Conference on Computer Integrated Manufacturing, 1990.

[15] W. J. Schroeder, W. E. Lorensen, and S. Linthicum, "Implicit modeling of swept surfaces and volumes," presented at IEEE Visualization, 1994.

[16] W. E. Lorensen and H. E. Cline, "Marching Cubes: A high resolution 3D surface construction algorithm," presented at SIGGRAPH, 1987.

[17] J. Bloomenthal, "Polygonization of Implicit Surfaces," *Computer Aided Geometric Design*, vol. 5, pp. 34 -355, 1988.

[18] I. Kenneth E. Hoff, T. Culver, J. Keyser, M. Lin, and D. Manocha, "Fast Computation of Generalized Voronoi Diagrams Using Graphics Hardware," presented at SIGGRAPH, 1999.

[19] Y. J. Kim, G. Varadhan, M. C. Lin, and D. Manocha, "Fast swept volume approximation of complex polyhedral models," *ACM Symposium on Solid and Physical Modeling*, pp. 11-22, 2003.

[20] L. P. Kobbelt, M. Botsch, U. Schwanecke, and H.-P. Seidel, "Feature Sensitive Surface Extraction from Volume Data," presented at SIGGRAPH, 2001.

[21] E. Ferré and J.-P. Laumond, "An iterative diffusion algorithm for part disassembly," presented at International Conference on Robotics and Automation, New Orleans (USA), 2004.

[22] F. Schwarzer, M. Saha, and J.-C. Latombe, "Exact Collision Checking of Robot Paths," in *Algorithmic Foundations of Robotics*, J. D. Boissonnat, J. Burdick, K. Goldberg, and S. Hutchinson, Eds.: Springer 2004, pp. 25-41.

[23] J. A. Sethian, "A Fast Marching Level Set Method for Monotonically Advancing Fronts," *Proceedings of the National Academy of Sciences (USA)*, vol. 93, pp. 1591-1595, 1996.

[24] J. R. Sack and J. Urrutia, *Handbook of Computational Geometry*. North Holland: Elsevier, 2000.

[25] J.-D. Boissonnat and M. Yvinec, *Algorithmic geometry*: Cambridge University Press, 1998.

[26] W. J. Schroeder, J. A. Zarge, and W. E. Lorensen, "Decimation of Triangle Meshes," presented at International Conference on Computer Graphics and Interactive Techniques, 1992.

[27] J.-P. Laumond, "Kineo CAM: a success story of motion planning algorithms," in *IEEE Robotics & Automation Magazine*, vol. 13, 2006, pp. 90-93.

TABLE II
PERFORMANCE AND ERROR OF TESTS

Model	Parameters		ε	Performance (s)				
	M	Δ		Sampling	Distance Field	Front	Tessellation	TOTAL
Exhaust	64	50	38.455	0.030	17.328	3.266	3.500	24.125
Exhaust	64	25	25.955	0.630	34.875	2.921	2.766	41.192
Exhaust	128	50	31.727	0.032	35.249	22.547	17.328	75.156
Exhaust	128	25	19.227	0.062	68.358	22.61	12.922	103.952
Exhaust	256	5	5.864	0.297	669.664	1016.33	70.874	1757.165
Seat	128	12	13.915	0.469	143.109	18.218	15.297	177.093
Human	128	13	13.254	1.295	170.469	59.765	9.875	241.404

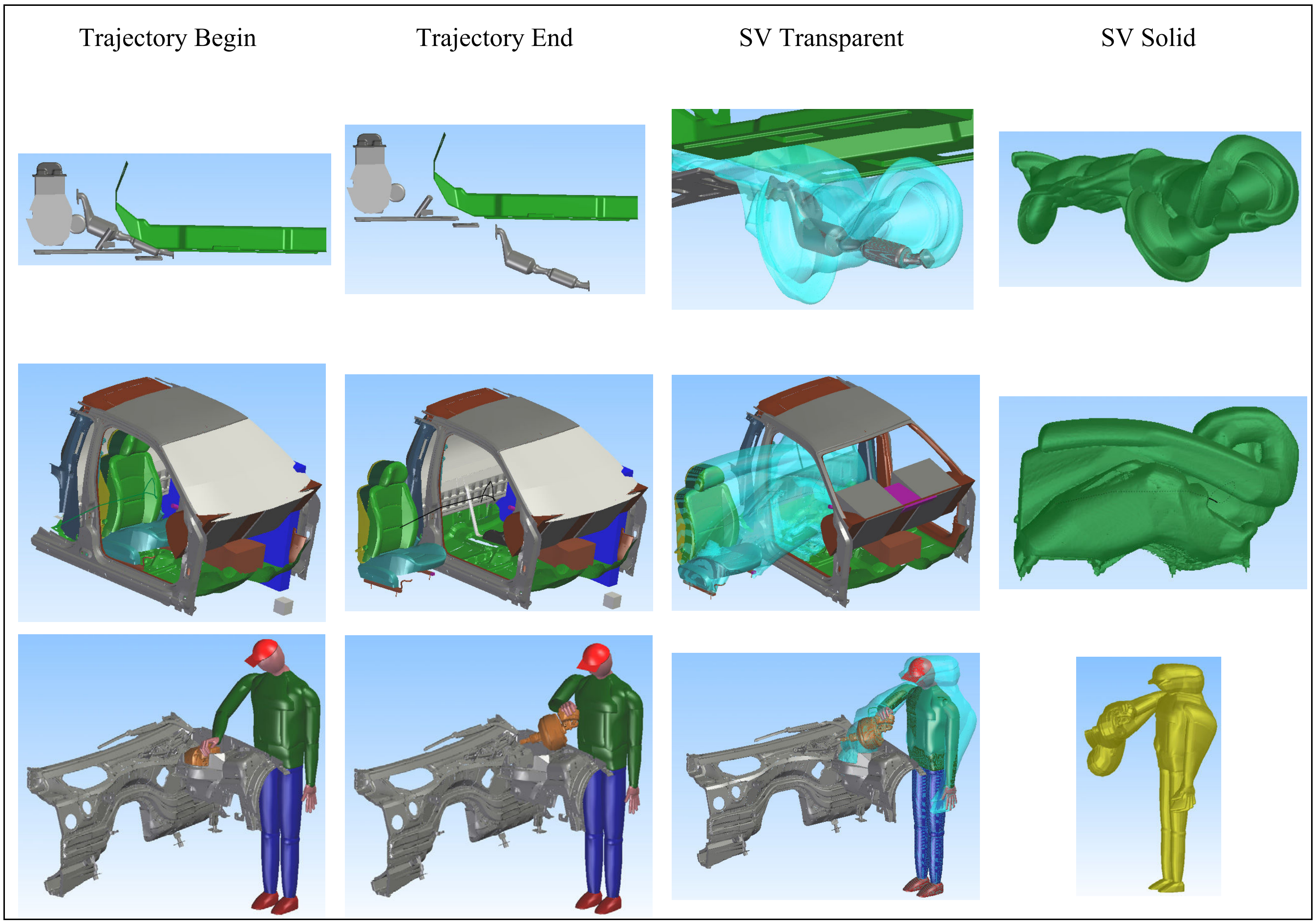


Fig. 9 Visual quality of SV calculations on PLM extraction scenarios. For each of the three benchmark models (*Exhaust*, *Seat*, and *Human*) we display the result of our SV calculation. The first two columns illustrate the beginning and ending of the trajectory, which has been generated by a path planning procedure to extract a certain part from an assembly. The third column shows the SV as a blue transparent mesh, containing the swept model inside it (note the in the *Human* case the SV includes the virtual mannequin itself). The SV is shown by itself as a solid mesh in the last column. Videos corresponding to these examples are available at http://www.laas.fr/gepetto

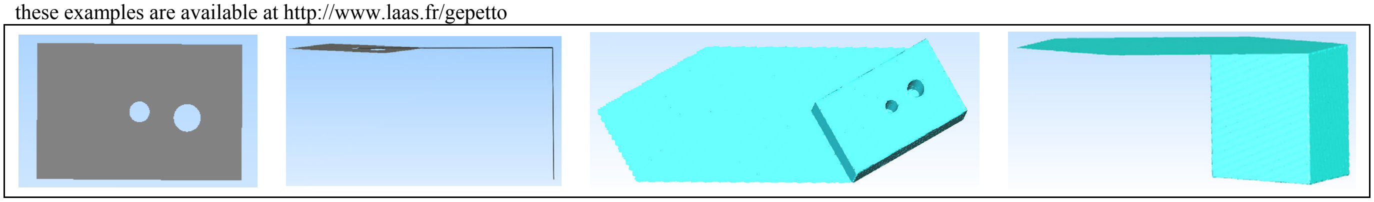


Fig. 10 Flat surface example. The surface with two holes (first) is first swept along its own plane, and then orthogonally (second). The resulting SV (viewed from bottom in third, side in fourth) displays the volume generated by the vertical motion as well as the flat surface generated by the horizontal movement.