

Swept Volume Computation of Malformed Objects in PLM*

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

Abstract— We present a fast 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, previous algorithms 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 nor deforming offsets. 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.

Note to Practitioners— Swept volumes are useful for collision detection, ergonomics, mechanical assembly, and many other fields. We successfully tested our method on actual industrial models, including a digital actor. Therefore, no additional work is required in order to put our technique into practice for polygon soup models. Additionally, the basic graphics card functionality used is accessible without programming that card itself.

Index Terms— Visualization, CAD/CAM, Motion analysis, Planning, Assembly

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

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).

* A shorter version of this paper appeared at IEEE Int. Conf. on Robotics and Automation, Rome, 2007.

modeling [3], collision detection [4], mechanical assembly [5], and ergonomic studies [6]. We are interested in its applications within Product Lifecycle Management, a business strategy and a technology solution to manage the entire life span of a product, from cradle to grave [7].

Two commonly encountered industrial cases in PLM that can greatly benefit from efficient SV calculation are mechanical assembly and ergonomic studies. Given a trajectory for extracting a part, engineers often desire to keep it collision-free, easing disassembly and maintenance tasks. Likewise, the SV can represent the volume of one or more human motions, 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.) [8]. Many geometric algorithms require watertight volumes to give meaningful results (by watertight we mean closed 2-manifolds, where each edge connects exactly 2 polygons, and polygons only touch on their edges). Although malformed models can theoretically be transformed into watertight 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. Our algorithm stays true to the original volume, and is capable of generating both volumes and flat surfaces where appropriate.

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.* [9].

Modern analytical approaches include envelope theory [10], singularity theory (a.k.a. Manifold Stratification or Jacobian rank deficiency method) [1, 11, 12], and Sweep Differential Equations [10, 13]. However, the type of data that we are treating does not lend itself easily to mathematical analysis.

1) Implicit Surfaces and Distance Fields

Schroeder *et al.* [14] 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. This value is positive for grid points outside the volume, and negative for those inside. The workspace (a volume bounding the entire sweep) is imposed a grid as well. 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, and the workspace values are updated in consequence. Finally, the boundary of the SV can be approximated as an isosurface where the distance equals zero. Marching Cubes is used to extract and triangulate this surface.

A consequence of this approach is that a volume must cover at least one grid point in order for it to be detected. In addition, the inside and outside of the input model must be clearly distinguishable. It can neither detect nor generate flat surfaces, and volumes with “holes” are considered invalid as input.

Nevertheless, it appears that the work of Schroeder *et al* [14] has been used in commercial products to generate swept volumes of polygon soup models [5]. Although not explicitly discussed in the literature, we can hypothesize that this feature is accomplished by extracting the isosurface at a positive value. This is equivalent to adding an offset d around all surfaces, essentially “growing” the volumes. In this case, surfaces become volumes. If d is high enough, any point on the polygon soup will grow into a volume sufficiently large to cover at least one grid point, making these points detectable, at the cost of deforming the resulting volume. For our work, we insisted on generating surfaces without offsetting.

The distance fields used in this algorithm can be generated through regular sampling of a bounding volume [16]. 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) [17].

2) GPU-based Directed Distances

Kim *et al.* [14] extend the implicit surface approach 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 the SV as well.

To remove the interior surfaces, the workspace is sliced along each axis, and a 2D grid is imposed onto each slice (Fig. 1). Using the GPU, distance fields (sets of distances from grid points to the object) are found along the edges of the grid. These fields are directed, meaning that they give the distances to the object along the 3 major axes, rather than the unidirectional scalar values used by Schroeder *et al.*

The grid points are then classified as outside or inside the SV using a propagating front level set method. Next, the surface of the SV is extracted using the Extended Marching Cubes (EMC) algorithm [15], 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 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 distance-fields) 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 contribution of our paper.

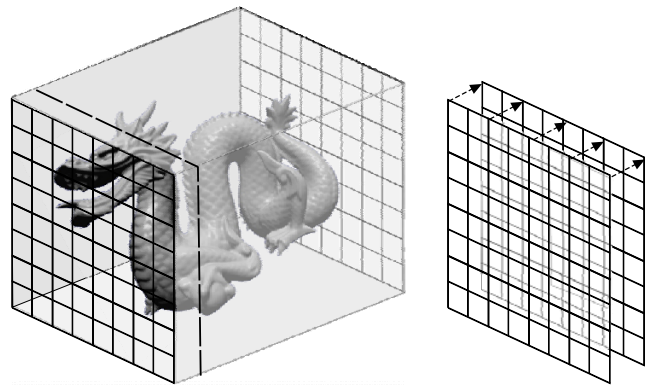


Fig. 1 The workspace is split into slices, and depth measurements performed for each slice. On the left, the dragon model is shown inside a bounding box. That box is split (right), and for each slice (such as from the front to the dashed line), distance fields are taken along the split direction.

C. Contributions

Our main contribution is devising a fast SV approximation algorithm to accept arbitrary polygon soups as input, and generate watertight volumes or flat surfaces as output, depending on the result.

In addition, we create a specialized tessellation algorithm for directed distance fields that generates meshes in time linear to the number of output vertices. Finally, we define a unified error bound of both slice width and trajectory sampling based upon the bounded move operator.

D. Outline

The rest of the paper is organized into six sections. In Section II, we discuss how the trajectory can be sampled while bounding possible error. In Section III, we develop an alternative level-set method to detect the SV surface. Section IV introduces our specialized triangulation algorithm and its theoretical error bounds. Section V presents our results, and we conclude with ideas for future work in Section VI.

II. TRAJECTORY SAMPLING

Rather than creating a swept mesh through ruled and developable surfaces as with Kim *et al* [18], we sample the surface at a certain number of intervals. Using discrete samples introduces the possibility of generalizing to

discontinuous motions, which are used to study mechanical vibration cases.

Sampling the trajectory along regular intervals would lead to situations in which 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 smaller movements. Additionally, determining the error bound implied by a given number of samples is not trivial (Fig. 2).

A response to this problem lies in [16], where the authors define a *bounded distance* operator on robot paths in the spirit of [17]. Given the set of points in the model A , and a continuous function of configurations $q(s)$, then we can state that the distance between two configurations $q(s_i)$ and $q(s_{i+1})$ is bounded by distance Δ if

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

Here, $P(q(s))$ signifies the position of a point in the configuration $q(s)$. In other words, the trajectory is bounded by distance Δ if no point in the model moves more than Δ between two successive configurations. The bounded distance operator provides a robust and convenient way to control error while minimizing the number of required samples.

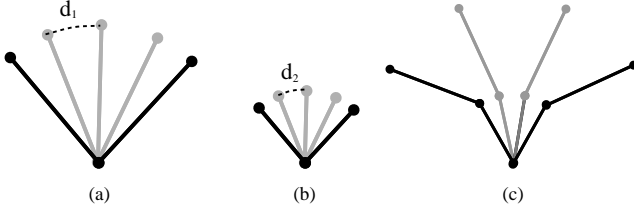


Fig. 2 Problems with regular path sampling. In (a) and (b), a simple straight arm is rotated about one end. Sampling this trajectory uniformly would result in very different error bounds for d_1 in (a) and d_2 in (b). This problem is only aggravated further when kinematic chains, such as (c), are introduced. The bounded distance operator aims at controlling the maximum error.

III. GRID COMPUTATION

When dealing with surfaces that do not contain any volume, the notions of inside and outside lose their meaning. In these cases, isosurface extraction methods such as Marching Cubes fail to find the contour (Fig. 3).

To properly detect these surfaces, we modified the fast marching level-set method presented in [14] (and in turn based upon [18]). Whereas they use it to simply classify grid points as inside or outside, we employ it to tag grid edges that cross the surface. These edges can later be used by our specialized triangulation algorithm (Fig. 4).

The method is based on an advancing front that starts from the extremities of the grid and slowly moves inwards until it hits the SV surface, at which point it stops. In order to calculate the front advancement in a sequential manner, we use a queue that contains all grid points currently participating in the front. So as to avoid adding the same

grid point to the queue multiple times, grid points are tagged with a *state* of 3 possible values. The state is initially *Far*, meaning that the grid point 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*.

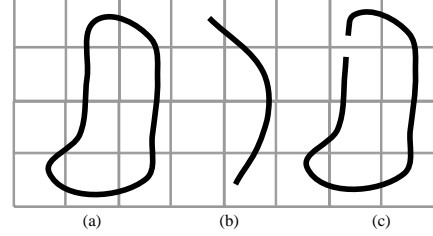


Fig. 3 These three surfaces are treated differently by traditional isosurface extraction. The volume in (a) will be found correctly, but since the surface in (b) does not contain any volume, it will be ignored, as will the volume with hole in (c).

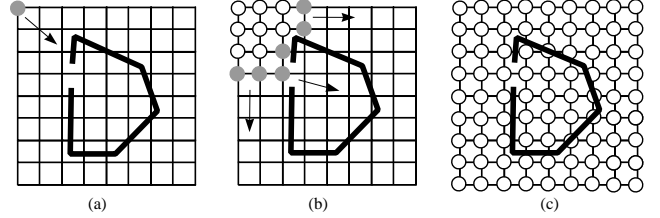


Fig. 4 The advancing front enters holes, rendering Marching Cubes useless. The front, represented by grey circles, starts at the upper left corner in (a), moving down and to the right. In (b), the front has advanced up to the hole in the volume. By the time the front is exhausted, in (c), it has completely filled the space. By labeling all the visited points as outside, it is impossible to recover the correct surface. Although the hole is exaggerated here for the purposes of example, the same phenomenon occurs with the smallest of imperfections in common PLM models.

```

1. while front ≠ ∅:
2.   p ← pop(front)
3.   state(p) ← Known
4.   for each neighbor q of p:
5.     if crosses_surface(pq):
6.       color(pq) ← Black
7.     else if state(q) = Far:
8.       state(q) ← Trial
9.       push(front, q)
10.    end if
11.  end for each
12. end while

```

Algorithm 1 Fast Marching Method adapted to recognize surface points. The algorithm works upon a queue of grid points called *front*. At each iteration, it takes a new grid point from the queue, and examines the edges with its neighbors, coloring them black if they cross the surface.

In addition, grid edges are associated with a color, initially white. If a grid edge is reached by the front, it is colored black if it crosses the wrapping surface. Once the queue is exhausted, these black edges are precisely those that cross the SV boundary.

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

IV. TRIANGULATION

From the distance fields gathered earlier, we know where each grid edge crosses the SV surface. We are interested in the 3D coordinates of these crossings, which we will call *detected surface points*, since they form the wrapping surface. Once the detected surface points are identified (i.e. those edges colored black by Algorithm 1), the next step is to triangulate them.

A large number of published tessellation algorithms could be used for this task. In particular, algorithms that tessellate point clouds would be appropriate [19, 20]. However, rather than dealing with the detected surface points as an unorganized set of points, we can exploit the known structure of the data returned by the level-set method to achieve linear-time triangulation.

A. Graph Building and Tessellation

Intuitively, we link each detected surface point with its neighbors, and then triangulate the result. In order to find a neighbor for a detected surface point, we start at the grid edge in question and proceed to walk along the grid in a closed curve until we cross the surface again. Four such walks will yield the neighbors to triangulate with.

In order to keep track of this information, we transpose the detected surface points onto an undirected graph. Each node of the graph represents a detected surface point, and the edges between nodes provide adjacency information. Intuitively, each point should be linked with its neighbors in the final triangulation. In terms of the graph, this means that each point participates in an elementary cycle with its neighbors.

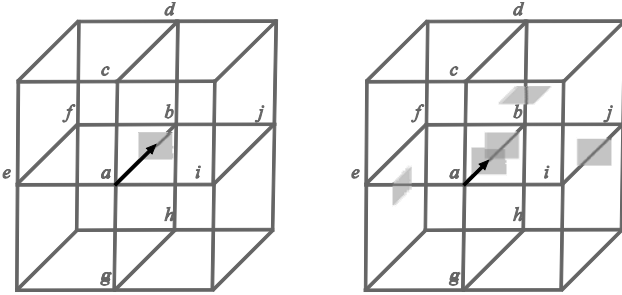


Fig. 5 Walking along the grid. In order to triangulate the detected surface points, we first need to determine which pairs of surface points are “neighbors”. To do so, we start at a single detected surface point and take 4 pre-determined walks along the grid. Along each walk, we check if certain grid edges cross the surface, and stop if that is the case. Each walk touches 4 edges maximum and is guaranteed to reach a neighboring surface point.

In order to link a node of the graph with its neighbors, we follow the underlying grid structure. Starting from the detected surface point of the node, we engage on four pre-determined walks along the edges of the grid, identical except for their starting directions. We continue each walk until we cross the surface again. The node corresponding to this detected surface point is our neighbor, and an edge is added to the graph between these two nodes. Since the final edge brings us back to the grid point from which the surface was initially detected (i.e. the walk traces a closed curve),

we are guaranteed to hit the surface during the walk.

In the example on the left of Fig. 5, the surface was detected along the edge \overline{ab} . The first walk leaves upwards from a along edge \overline{ac} , then forwards along \overline{cd} , downwards along \overline{db} , and finally back again along \overline{ba} . Similarly the walk moving left from a involves the sequence of edges $\{\overline{ae}, \overline{ef}, \overline{fb}, \overline{ba}\}$, the one moving down $\{\overline{ag}, \overline{gh}, \overline{hb}, \overline{ba}\}$, and the one moving right $\{\overline{ai}, \overline{ij}, \overline{jb}, \overline{ba}\}$. On the right, we include neighboring detected surface points. The upwards walk would therefore touch and stop on \overline{db} , the leftwards walk would touch \overline{ae} , the rightwards one \overline{ij} , and finally the downwards walk would loop around to touch the opposing surface point on \overline{ba} .

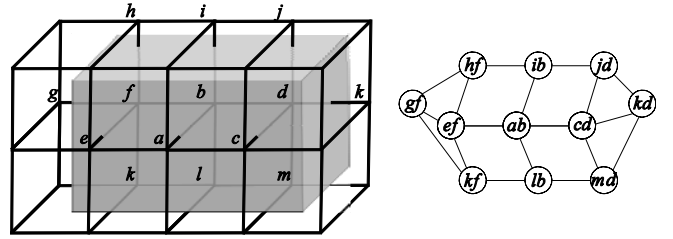


Fig. 6: Graph building. When the box on the left is placed in the grid, 11 surface points are detected at the intersections. They are transposed onto the graph on the right, by connecting each detected surface point (described by the corresponding grid edge) to its neighbors. Note that the letters on the left designate the grid points, whereas the nodes on the right reference grid edges by a pair of grid points.

B. Error Analysis

Both the distance field generation and trajectory sampling steps are potential sources of error in our approximation algorithm. Each one is controlled by a single parameter, and their combination defines the error bound as well as the time and memory complexity of the algorithm.

Let S equal the slice width. Since no measurements are taken between grid lines, the approximated surface could be off by up to the diagonal length of the grid cell, or $\sqrt{3}S$. Between two sampled configurations, we can guarantee that no point on the object traces a path longer than Δ . Assuming that the point is detected at both configurations, the farthest it could move from the detected surface is $\Delta/2$. The resulting net error is the sum of the two: $\epsilon \leq \sqrt{3}S + \Delta/2$.

V. EXPERIMENTAL RESULTS

We implemented the SV algorithm in C++, with graphic routines in OpenGL. Since we use only very basic graphic card functionality, any 2nd generation GPU supporting z-buffer and frame-buffer readback suffices.

All tests are conducted on an Intel Core 2 Duo running at 2.66 GHz with 4GB RAM (the work is only done on one core, however). The GPU is an nVidia Quadro FX 4600 with 768 MB dedicated memory. The operating system is 64-bit Windows Vista.

We used 3 models for testing: a car seat, a car exhaust, and a virtual human actor. All came from actual PLM cases

encountered by Kineo CAMTM. The path being swept is the result of a path planning process. Table I lists the number of triangles for each model, as well as the size of the workspace surrounding the swept path.

The performance results are shown in Table II. Swept volumes are generated with two different pairs of parameters: the slice width S and the bounding distance Δ . The error ϵ is directly determined by these two parameters. The chosen bounding distance Δ leads to a certain number of samples along the path. To gather distance fields, the object is drawn at each of these samples for each slice of the workspace, leading to a large number of drawn triangles. The grid size relates the number of grid points along each of the workspace axes, a result of the chosen slice width S and the size of the workspace. The number of grid points is simply the product of the grid size

To get a grasp on the effect of changing the input parameters, we varied the slice width for the *Exhaust* model and placed the results in Table III. Reducing the slice width has a dramatic effect on all performance indicators, since it raises the number of grid points, detected surface points, and slices for which to draw the model. On the other hand, the slice width is the dominating contributor to the total error.

It is difficult to directly compare with the experimental results of Kim *et al* [14], since we use different models and experimental setup, but our results are of the same order of magnitude, while solving a more difficult problem. Furthermore, the high visual quality of the produced SV is suitable for PLM purposes (Fig. 7).

TABLE I
TEST MODELS

Model	# Triangles	Workspace Size (mm)
Seat	30765	2027 x 1578 x 1193
Exhaust	32641	1940 x 597 x 666
Human	55632	1667 x 1102 x 805

Despite appearances, none of the test models are watertight meshes. To further demonstrate the applicability of our algorithm to non-volumetric geometry, we sweep a simple flat surface in Fig. 8. Videos corresponding to these examples are available at <http://www.laas.fr/gepetto>.

VI. 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 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 volumetric 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.

Finally, our use of the GPU is quite limited—drawing polygons and reading from the z-buffer to obtain the directed distance fields. It would be interesting to explore how applying modern GPGPU techniques could both accelerate the performance and add new capabilities.

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," in *Proceedings of Pacific Graphics*, 2000, pp. 272-280.
- [4] A. Foisy and V. Hayward, "A safe swept volume method for robust collision detection," in *Proceedings of 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," in *Proceedings of 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] T. M. Murali and T. A. Funkhouser, "Consistent solid and boundary representations from arbitrary polygonal data," in *Proceedings of SIGGRAPH Symposium on Interactive 3D Graphics*, 1997.
- [9] K. Abdel-Malek, D. Blackmore, and K. Joy, "Swept Volumes: Foundations, Perspectives, and Applications," *International Journal of Shape Modeling*, 2002.
- [10] R. R. Martin and P. C. Stephenson, "Sweeping of Three-dimensional Objects," *Computer Aided Design*, vol. 22, pp. 223-234, 1990.
- [11] 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.
- [12] 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.
- [13] D. Blackmore and M. C. Leu, "A differential equation approach to swept volumes," in *Proceedings of Rensselaer's 2nd International Conference on Computer Integrated Manufacturing*, 1990, pp. 143-149.
- [14] 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.
- [15] L. P. Kobbelt, M. Botsch, U. Schwanecke, and H.-P. Seidel, "Feature Sensitive Surface Extraction from Volume Data," in *Proceedings of SIGGRAPH*, 2001, pp. 57-66.
- [16] E. Ferré and J.-P. Laumond, "An iterative diffusion algorithm for part disassembly," in *Proceedings of International Conference on Robotics and Automation*, 2004, pp. 3149-3154.
- [17] F. Schwarzler, 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.
- [18] 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.
- [19] J. R. Sack and J. Urrutia, *Handbook of Computational Geometry*. North Holland: Elsevier, 2000.
- [20] J.-D. Boissonnat and M. Yvinec, *Algorithmic geometry*. Cambridge University Press, 1998.

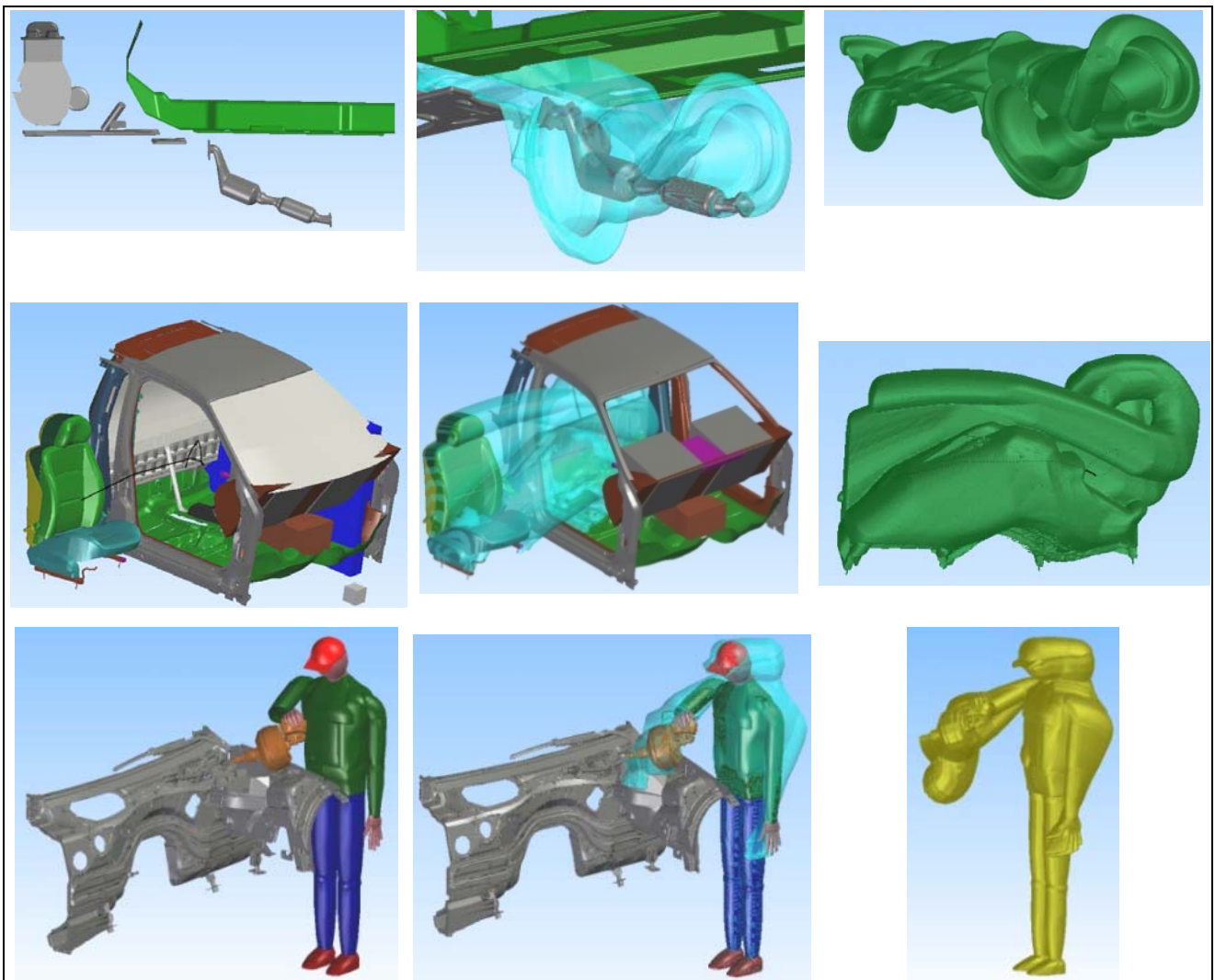


Fig. 7 SV calculations on PLM extraction scenarios. The *Exhaust* scenario is shown in the top row, followed by the *Seat* and *Human* test cases. The first column shows the object in its environment without the corresponding SV. The second column includes the SV as a blue transparent mesh, and the third displays the SV by itself. Videos corresponding to these examples are available at <http://www.laas.fr/gepetto>

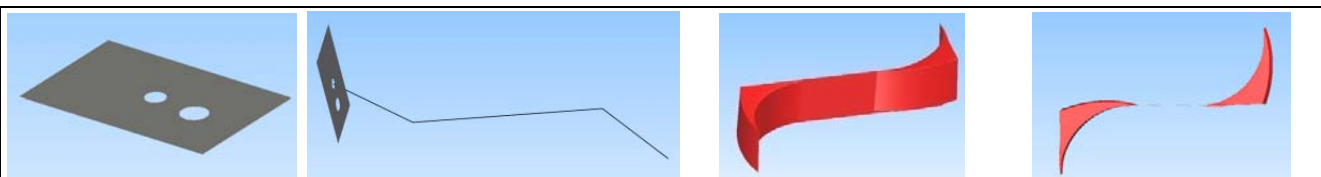


Fig. 8 Flat surface example. The surface with two holes (first) is swept along an S-shaped curve. (second). The resulting SV (viewed from an angle in third, side in fourth) displays the flat surface in the middle as well as the volumes on the ends. The flat surface would be unobtainable with previously published algorithms.

TABLE II
EXPERIMENTAL RESULTS

Model	Parameters		Statistics					Performance		
	S (mm)	Δ (mm)	ε (mm)	# Samples	# Drawn Triangles	Grid Size	# Grid Points	Time (s)	Memory (MB)	# Produced Triangles
Exhaust	15	10	31	582	15.1G	130x47x25	275k	10.5	20	99.8k
Exhaust	7	6	15.1	970	48.1G	228x100x96	2.2M	52.8	146.3	181.9k
Seat	15	10	31	195	7.3G	136x106x80	1.2M	18.2	56.4	91.7k
Seat	10	6	20.3	325	18.3G	203x158x120	3.8M	52.7	201.5	451.7k
Human	15	10	31	693	24.1G	139x92x67	857k	26.4	28.5	55k
Human	10	6	20.3	1155	48.6G	167x111x81	1.5M	65.9	85.1	124.9k

Swept volumes are generated with two parameters: the slice width S and the bounding distance Δ .

TABLE III
VARYING SLICE WIDTH

Model	Parameters		Statistics					Performance		
	S (mm)	Δ (mm)	ε (mm)	# Samples	# Drawn Triangles	Grid Size	# Grid Points	Time (s)	Memory (MB)	# Produced Triangles
Exhaust	5	6	11.7	970	42G	389x140x134	7.3M	110.1	398.2	362.9k
Exhaust	6	6	13.4	970	34.9G	323x117x111	4.2M	73.2	239.5	249.7k
Exhaust	7	6	15.1	970	48.1G	228x100x96	2.2M	52.8	146.3	181.9k
Exhaust	8	6	16.9	970	26.2G	243x88x84	1.8M	42.4	112.3	138.5k
Exhaust	9	6	18.6	970	23.3G	216x78x74	1.2M	33.8	83.8	109.1k



Jesse C. Himmelstein (M'06) is a PhD student at the LAAS-CNRS under a joint grant with Kineo CAM. He received his B.S. in computer science from Johns Hopkins University (Baltimore, Maryland, USA) in 2002, and his M.S. at INSA (Toulouse, France) in 2005. His research interests include motion planning, collision detection, and geometric deformation. He is a student member of IEEE.



Jean-Paul Laumond (M'95–SM'04–F'07) received the M.S. degree in mathematics, the Ph.D. degree in robotics, and the Habilitation degree in robotics from the University Paul Sabatier, Toulouse, France, in 1976, 1984, and 1989, respectively. In Fall 1990, he had been an Invited Senior Scientist at Stanford University, Stanford, CA. From 1991 to 1995, he was a member of the French Comité National de la Recherche Scientifique. He has been a Coordinator for two European Esprit projects, PROMotion during 1992–1995 and MOLOG during 1999–2002, both dedicated to robot motion planning technology. During 2001–2002, he created and managed Kineo CAM, a spin-off company from the LAAS-Centre National de la Recherche Scientifique (CNRS), Toulouse, to develop and market motion planning technology. He teaches robotics at the ENSTA and Ecole Normale Supérieure in Paris. Currently, he is a Directeur de Recherche at the LAAS-CNRS, Toulouse, where he is also associated with JRL-France, as its Co-Director. He is the author or coauthor of more than 100 papers published in international journals and conferences in computer science, automatic control, robotics and neuroscience, and is the Editor of three books. His current research interests include human motion studies along three perspectives: artificial motion for humanoid robots, virtual motion for digital actors and mannequins, and natural motions of human beings. Dr. Laumond is an IEEE Fellow. He is currently an IEEE Distinguished Lecturer and a member of the IEEE RAS AdCom.



Etienne Ferré was graduated from the Ecole Polytechnique in 1997 with a specialized graduation at ENSTA in 2001. He joined Kineo CAM in 2001 after the creation and became CTO in 2003.