Horizontal Decomposition of Triangulated Solids for the Simulation of Dip-coating Processes

B. Strodthoff [a] M. Schifko [b] B. Jüttler [a]

[a] Johannes Kepler University, Institute of Applied Geometry, Linz, Austria
[b] Magna Powertrain Engineering Center Steyr GmbH & Co.KG, St. Valentin, Austria

Abstract

In dip-coating processes a three-dimensional object, e.g. an entire car body, is dipped into a liquid bath. In order to simulate such processes, the space surrounding the object is decomposed into the so-called *flow volumes*, for which each intersection with a horizontal plane is connected. At any time the liquid's surface then has a unique level within such a flow volume, which greatly simplifies the simulation of the liquid. The decomposition into flow volumes corresponds to the Reeb graph of the object's exterior (considered as 3-manifold with boundary) with respect to the height function. This article presents an algorithm which computes this decomposition for an object represented as oriented triangular boundary mesh. First critical vertices of the surface are identified, which include the upper and lower ends of flow volumes. Using local information about horizontal intersection planes near the critical points, a sweep plane algorithm then constructs the volume decomposition in a second step. It is shown that the method can deal with realistic data.

1 Introduction

Car bodies or body parts in the automotive industry are covered by several coatings, e.g. to prevent corrosion. Those are often applied by electrophoretic dip coating [18], where the objects are successively moved through tanks of different coatings. Air bubbles may be created while dipping an object into the liquid, and puddles of liquid may remain when removing the object from the bath. It is desirable, however, to achieve a uniform and complete coverage of the surface. To ensure this, it is currently the state of the art to build prototypes and perform the dip coating. Afterward, the prototype is cut into slices to analyze the coating's quality, also in hidden areas. If surface parts are detected where the dip coating has failed, the car body has to be modified accordingly, e.g. by inserting a small hole to release an air bubble.

Since this approach uses prototypes, tests cannot be performed until a very late stage of the design process. Each change of the car body design at this stage is expensive and exceedingly delays the development process. Therefore a reliable simulation of the dip coating is desirable, as it allows not only optimizations of the diving path, but also adaptations of the geometry in an early stage of the design process.

This paper describes the *geometric kernel* of a dip-coating simulation procedure, which analyzes specific properties of an object's surface to decompose the computational domain. This has to be done frequently if the mesh is rotated, therefore it is essential to use an efficient algorithm. The decomposition serves as input data for a *simulation kernel*, which computes the distribution of fluid among the volume parts.

Objects in manufacturing processes are often described by triangular surface meshes, mainly due to the needs of data exchange. Indeed, triangular meshes are suitable for merging components which are provided by different suppliers, as they can be exported from any CAD-program. The triangular boundary representation of a car consists of up to 10^{10} triangles. We assume this *triangulated solid* to be arranged in a halfedge data structure, where every triangle can access its adjacent triangles, edges and vertices in constant time. In the implementation, these halfedge data structures are constructed from STL files exported from CAD-programs.

The dip coating was so far mainly simulated using computational fluid dynamics (CFD [37]) with the so-called two-phase description. Theoretically, almost any detailed description of the time-dependent diving process can be calculated by CFD. However, according to experience in industry, this method is not optimal, as it requires very long computation times. Additionally, CFD simulations are highly sensitive to the choice of boundary conditions. This suggests considering alternative approaches to the simulation.

In CFD and Finite Element Analysis, the computational domain is usually divided into a large number of small primitive volume parts, like tetrahedra or hexahedra (see e.g. [19, 29, 31, 12, 40, 34, 28]). If the parts are sufficiently small, computations can be simplified inside each part without introducing substantial errors. In our application, the main idea for improving computation times is the use of fewer, larger volume parts. In order to preserve accuracy, computations within these large volume parts should no longer be approximated. Thus the simulation of fluid inside a volume part needs to be relatively simple.

One approach ensuring this is a *convex decomposition* of the computational domain: Within a convex volume part a liquid's surface is exposed to uniform pressure conditions, such that the liquid will always be able to balance.

Therefore a unique filling height can be computed for a volume part given the amount of contained liquid, taking into account only information about the volume part's geometry. Convex decompositions are computed e.g. by sweep plane algorithms [10] comparable to the algorithm presented here, by flooding algorithms where adjacent surface triangles are collected for each patch until convexity is violated [10, 17], or by repeatedly splitting the object in areas of concavity [11, 3, 23, 21]. Such decompositions consist of a large number of volume parts, which can be reduced by allowing small concavities in the parts [41, 23]. However, these nearly complex decompositions still contain more parts than necessary for our purpose.

Concave boundary parts or holes in a spatial area impose many convex parts in the decomposition, often many more than needed to achieve uniform physical conditions as mentioned above. In an *Alternating Sum of Volumes* [38, 39] decomposition, a spatial area is described by the set-union *and* set-difference of convex volume parts. A spatial area containing a hole thus is not segmented unnecessarily, as the hole is represented separately. In certain cases, however, concave boundary parts lead to violations of the uniform physical conditions of a convex volume part, which makes this approach inappropriate for our purpose.

In this paper, a new kind of decomposition of the computational domain is introduced, which combines the previously described characteristics: Horizontal slices of the volume parts are prescribed to be connected, which induces the homogeneous physical conditions, avoiding oversegmentation. Additionally the new volume parts, called *flow volumes*, are defined such that they meet only along horizontal boundary patches, which simplifies the interactions between adjacent flow volumes.

These flow volumes are in some way monotonic: In the planar case, a polygon is called monotonic with respect to a straight line L if the intersection of the polygon with any line orthogonal to L is connected [5]. As a generalization, we could call a spatial area (like the interior of a polyhedron) *monotonic* with respect to a straight line L if every intersection of the area with a plane orthogonal to L is connected. In this sense, flow volumes are monotonic with respect to the z-axis. This form of monotonicity, however, is even weaker than the concept of weak monotonicity of polyhedra as described e.g. in [20], which requires the intersection to be bounded by a simple polygon.

Reeb graphs are an important concept in Morse theory for gathering topological information. In Morse theory, the topology of a manifold is analyzed using critical points of a differentiable function f over the surface, the *Morse function* [24, 25, 4]. A Reeb graph's vertices represent these critical points, its edges correspond to parts of the manifold of which all points with the same f value are connected (see [7] for an introduction). Applications of Reeb graphs include shape abstraction [2, 6, 22] and recognition [8, 36].

The flow volume decomposition motivated by the application of dip-coating simulations corresponds to the *Reeb graph* of the computational domain, considered as 3-manifold with boundary and using the height function as Morse function. Each edge of this Reeb graph represents one flow volume. There exist several algorithms for computing Reeb graphs (or the closely related *contour trees*) for 3-manifolds. These algorithms use volumetric descriptions of the object, e.g. tetrahedral meshes [9, 26, 15, 16, 35], or voxel based representations [30], and they are able to deal with general Morse functions. In the case of flow volumes, where the Morse function is the height function, using a boundary representation is sufficient and simplifies the computation. This is demonstrated in our paper.

There exist also algorithms operating on surfaces. These compute the Reeb graph of the surface, considered as 2-manifold without boundary. As shown in Figure 1, the Reeb graphs of the surface, the interior or exterior volume are in general different. Reeb graphs (or contour trees) of a surface with respect to the height function are typically computed by tracking intersection components while sweeping the surface with a horizontal intersection plane [32, 14, 6]. Similarly the surface can be dissected into horizontal slices of a certain thickness, using refinements until no topologically important features are lost [2]. However, these setups typically use many horizontal intersections. Other approaches [22, 9, 13, 16, 27] first identify critical points, and then find connections between them, which dramatically decreases the required number of intersections. They compute only incomplete horizontal intersections, if any.

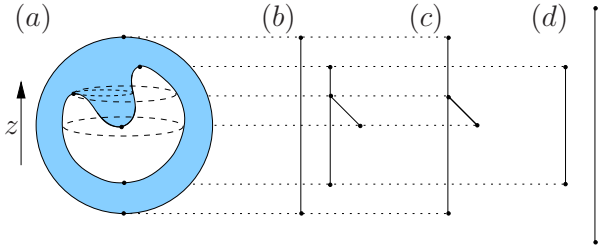

Figure 1: Reeb graphs with respect to the height function. (a) Sample object: sphere with a bowl-shaped hole. (b) Reeb graph of the surface. (c) Reeb graph of the solid's interior (shaded area). (d) Reeb graph of the solid's exterior. In this example the three Reeb graphs are different. The flow volumes correspond to the edges in (d).

For computing flow volumes for an object represented by a boundary surface mesh, we adapt algorithms for surface Reeb graphs. Additional information about the relative spatial position of intersection polygons is required in order to determine intersection polygons' connections through the computational domain. To this end, the object is swept with a horizontal plane, computing global intersections, but only in critical points where they are needed for the flow volume computation. Between those, intersection components are traced by monotone paths similar to [16, 13].

2 Problem description

The geometric kernel of the dip-coating simulation decomposes the *free volume*, i.e. the volume outside the triangulated solid, into the so-called *flow volumes*:

Definition 2.1. A connected part F of the three-dimensional space is called *flow volume* for a given triangulated solid if

- The boundary of F consists of triangles of the triangulated solid and parts of horizontal planes on top and bottom.
- F does not intersect the interior of the solid.
- Every horizontal slice of F is connected.
- F is maximal among all volumes satisfying the previous conditions.

Due to the third property in Definition 2.1 there can be only one filling level within a flow volume without violating physical laws. For example the inside and outside of a bowl cannot belong to the same flow volume, because the filling level inside the bowl might be different than the filling level on the outside. On the other hand the inside of a simple bowl is contained in one flow volume, as any liquid in the bowl will balance such that a unique filling level is achieved.

The last property makes sure that the computed volume parts are not too small, as otherwise arbitrarily small volume slices would be allowed. Figure 2 shows the volume decomposition for a simple object. Note that flow volumes need not be topologically simple, as they may contain holes.

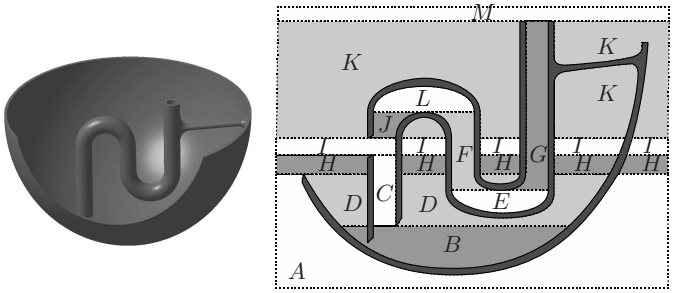

Figure 2: Decomposition of free volume. Left: A given object. Right: Vertical cut through the object with the flow volumes marked by letters. Flow volume H is an example for a flow volume with holes (one hole for each pipe)

The decomposition of free volume into flow volumes is unique and always possible. To visualize this, consider a horizontal plane sweeping through the object from bottom to top. Each connected area where the plane intersects free volume corresponds to one flow volume. Whenever such an area appears in the intersection plane, a new flow volume is started, the disappearance of such an area from the intersection plane marks the end of a flow volume. When such areas are joined or separated, the corresponding flow volumes are closed and new ones are started, as otherwise the horizontal intersections would no longer be connected.

3 Overview of the algorithm

For the computation of flow volumes, a plane sweep algorithm is applied. A horizontal plane is moved through the object in direction of increasing z coordinates. Once a spatial region has been passed by this *sweep plane*, the decomposition of this region is completed. In order to achieve this, the sweep plane carries a certain *status*, in which it keeps track of intersection components, i.e. of parts of the triangulated solid which intersect the plane in one connected intersection polygon, and remembers which flow volumes they belong to. This status does not change continuously, but only in the so-called *events*.

The naive approach to the selection of events would be to use each vertex of the mesh as event. In this setup every change of horizontal intersections is detected immediately, however at the price of considering a huge number of intersections. It is rather obvious that not every vertex needs to be considered, as a much smaller set of *critical* points can be identified in which the only major changes in the structure of horizontal intersections occur. These critical points can be determined before starting the plane sweep algorithm, and some local information about the triangulated solid is collected for each of them, which is valuable for the event handler. An *update routine* takes care of the minor changes in horizontal intersections between successive events.

The sweep plane algorithm is specified as follows:

- *Input:* A list of events, sorted according to z-coordinates.
- *Output:* The flow volumes, each containing sufficient information for gathering its complete boundary.
- *Status:* A list of edges of the triangulated solid intersecting the sweep plane, one edge for each intersection component, which are called *representative edges* (REs). The edges for intersection components contained in a common flow volume form an *edge group*, and each edge group contains a reference to the associated flow volume. The status is initialized by an empty edge group.
- *Events:* The critical points of the triangulated solid, equipped with additional local information.
- *Event handler:* adapts edge groups in the status and generates part of the output in appropriate events.
- *RE-update:* The REs are updated by climbing along the triangulated solid to the z-level of the next event.

It is summarized in the pseudocode shown in Algorithm 1.

Algorithm 1 computeVolumes(surface mesh M)

```
status ← empty List
eventList ← GenerateEvents(M)      //see Algorithm 2
Sort eventList by increasing z-coordinates
for all events e in eventList do
   Climb to the z-level of e
   associateGroups(status,e)       //see Algorithm 3
   adaptStatus(status,e)           //see Algorithm 4
end for
```

Using results from the following sections, the correctness of the overall algorithm can be proved.

Theorem 3.1. *The sweep plane algorithm computes the correct flow volume decomposition.*

The proof of will be presented in Section 6.

The remainder of this paper is structured as follows: Section 4 introduces the critical points and describes additional information which is collected for each of them to form events. Section 5 thereafter explains how these events are computed. In Section 6 the event handler is discussed and Section 7 shows some examples. Runtime considerations are presented in Section 8. We conclude with some remarks on the implementation and further tasks in Sections 9 and 10.

4 Definitions and events

This section considers the critical points of the height function on the surface of the triangulated solid, which will turn out to be a good choice for the events of the sweep plane algorithm. For ease of presentation, we exclude meshes with horizontal edges. Later, we will address this case in Section 9.

Definition 4.1. A vertex v of the mesh is a *critical point* if

- v is a local extremum, i.e. a local minimum or maximum.
- v is a saddle point, i.e. the boundary polygon of the vertex star of v intersects the horizontal plane through v more than twice.

4.1 Subdivision of an intersection plane

A plane traversing the triangulated solid can be subdivided into parts where it intersects the solid's interior, and parts where it intersects free volume:

Definition 4.2. Assume that all polygons forming the intersection of the triangulated solid with a horizontal plane p at height h are simple and without contact. These polygons decompose p into connected areas called *regions*, which form the *subdivision* $S(h)$. A region is called *filled* if p intersects the solid's interior in this area, or *empty* if p traverses free volume there.

When a horizontal plane sweeps the solid, its decomposition changes mainly when passing critical points:

Lemma 4.3. *Let p denote a horizontal plane at height h_p.*

- *$S(h_p)$ is well defined if and only if p does not contain a critical point.*
- *$S(h_p)$ changes only when p passes a critical point.*

Proof. Horizontal intersection polygons meet exactly in saddle points of the surface: If intersection polygons touch in a point x, in total at least 4 polygon segments contain x as endpoint. Each of them forms the boundary between surface parts above and below p. Thus the boundary polygon of the vertex star of x intersects p at least 4 times, and x is a saddle point. The other implication follows analogously.

On the other hand an intersection polygon collapses to a point exactly if this point is a local extremum. This is true if the triangulated solid fulfills some basic assumptions, for example that points must not lie on more than one surface. By splitting such points into several vertices of the halfedge data structure, this condition can be eliminated.

This proves the first statement. Due to continuity of the triangulated solid, the second statement follows directly from the first one. □

4.2 Marginal subdivisions

According to Lemma 4.3, the subdivision of a plane containing a critical point is not defined, as intersection polygons (and therefore also regions) may collide or collapse to a point, and polygons need no longer be simple. Therefore two slightly perturbed intersections are considered in this case; see Figure 3.

Definition 4.4. For a height h the *marginal* subdivisions are defined as follows:
$S^*(h) := \lim_{\epsilon\to 0}\{S(h+\epsilon)\}$ is the set of *upper regions* at h, and
$S_*(h) := \lim_{\epsilon\to 0}\{S(h-\epsilon)\}$ is the set of *lower regions* at h.

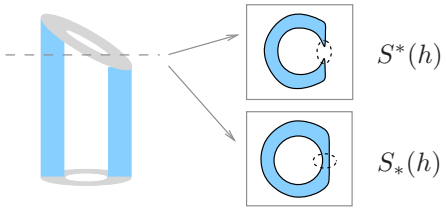

Figure 3: Left: Vertical cut through a pipe with marked intersection height h. Right: Horizontal cut directly above and below h; Empty regions are white, filled regions are shaded, black lines show intersection polygons. The dotted lines mark points on the polygons which collapse to the same physical point at height h. Upper subdivision consists of one empty and one filled region, lower subdivision contains two empty (inside and outside pipe) and one filled region.

4.3 Local subdivisions

Consider the subdivision S of the horizontal plane through a given vertex v. Only those intersection polygons in S which contain v can be gathered conveniently by starting polygons in v and tracing the triangulated solid. However, the sweep plane algorithm does not need more than this information for a critical point v, as only the regions touching v change while the sweep plane passes v. The following operator describes the transition from the subdivision of an intersection plane to a local subdivision, which requires only this easily accessible information; see Figure 4.

Definition 4.5. For a vertex v of the triangulated solid, let $\mathcal{S}_v$ denote the set of all subdivisions of a horizontal plane through v. The operator $\mathcal{L}_v : \mathcal{S}_v \to \mathcal{S}_v$ maps a subdivision to a *local subdivision* obtained by

- removing polygons which are not connected to v,
- deleting regions which neither contain v on the boundary, nor on the inside, and
- extending the remaining regions such that a valid subdivision is obtained again.

In order to distinguish it from its local version, the original subdivision is referred to as *global* subdivision.

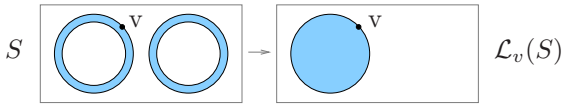

Figure 4: Left: Global subdivision S of a horizontal plane through a vertex v, three empty regions (two pipes and outside) and two filled regions. Right: Local subdivision around v, One empty and one filled region.

This function can be applied to the subdivisions defined in Definitions 4.2 and 4.4:

Definition 4.6. Consider a vertex v at height h_v. The *local (upper or lower) subdivisions* around v are defined as:

$$L(v) := \mathcal{L}_v(S(h_v))$$
$$L^*(v) := \mathcal{L}_v(S^*(h_v))$$
$$L_*(v) := \mathcal{L}_v(S_*(h_v))$$

4.4 Region Sets

In critical points, only marginal subdivisions are defined. For efficiency reasons, only local versions of them are computed, which serve as additional data for the sweep plane algorithm.

Definition 4.7. The *region sets* of a critical point v are the local marginal subdivisions $L^*(v)$ and $L_*(v)$.

For a maximum or minimum v, one region set consists of one empty and one filled region, the other region set contains only one filled or empty region. In the case of saddle points, however, both region sets contain several regions; see Figure 5.

4.5 Important critical points

Each empty region in an intersection plane corresponds to a flow volume hit by this plane. Therefore critical points with changes in the empty regions are distinguished from critical points in which only filled regions change.

Definition 4.8. A critical point v is called *important* if the number or connectivity of empty regions in the decomposition of an intersection plane changes while it passes the point. Otherwise v is called *unimportant*.

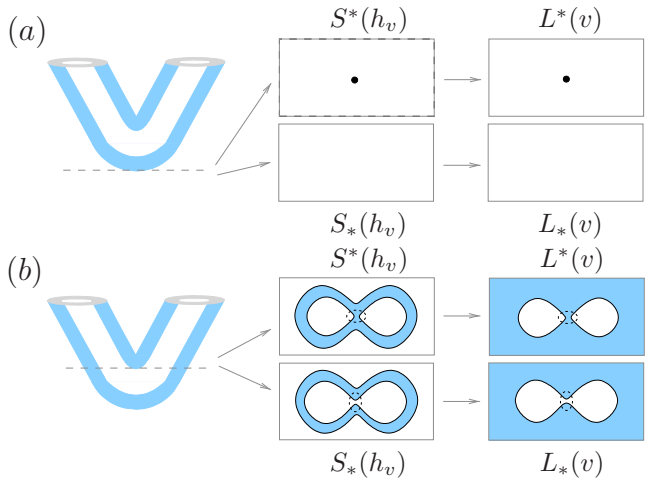

Figure 5: Region sets of critical points v at height v_h on a bent pipe. Vertical cut with marked intersection height, marginal upper and lower subdivisions $S^*(h_v)$ and $S_*(h_v)$ and region sets $L^*(v)$, $L_*(v)$. (a) A minimum below the solid's interior: one empty lower region, one empty upper region and a filled upper region consisting of only one point. (b) A saddle point where one empty region splits up while the intersection plane passes the saddle point.

In Figure 5, the minimum presented in (a) is unimportant, while the saddle point in (b) is important. The region sets of a critical point determine whether it is important or not:

Lemma 4.9. *A critical point v is unimportant if its region sets contain exactly one upper and one lower empty region. Otherwise v is important.*

Proof. If a region set of a critical point v contains more than one empty region, each of them is merged with another empty region of the same region set. Otherwise the surface parts bounding the associated free volumes would touch only in the point v, which is excluded by similar argumentations as in the proof of Lemma 4.3. Thus only region sets of important critical points may contain several empty regions. If, on the other hand, one region set of v contains no empty regions, then v is an important minimum or maximum. So each region set of an unimportant critical point contains exactly one empty region.

The other implication follows directly from the definition of important critical points, as the number or connectivity of empty regions in a horizontal plane cannot change in a critical point with exactly one upper and lower region. □

The important critical points are helpful for constructing the volume decomposition:

Lemma 4.10. *A connected volume part fulfilling the first two properties of Definition 2.1 is a flow volume if and only if it contains important critical points exactly on its top and bottom end.*

Proof. Consider a volume part V that fulfills the assumptions of this lemma.

The third property (connected horizontal slices) of Definition 2.1 is equivalent to the absence of important critical points inside V: The region sets of an important saddle point inside V contain several upper or several lower empty regions. Between its upper and lower end, V is only bounded by the triangulated solid. So all these empty regions would have to be contained in a slightly shifted slice of V, which thus would not be connected. For an important extremum, one region set contains exactly one region, which is filled. As V is assumed to be connected, a slice of V at this height has to contain another empty region, or V would not extend beyond this height. So again there is a slice of V slightly above or below the important extremum which is not connected. Thus a flow volume cannot contain an important critical point. For the other implication, assume V has a slice which consists of several empty regions. As V is spatially connected, there has to be an important saddle point in V where these empty regions are merged.

The fourth property (maximality) of Definition 2.1 corresponds to the presence of important critical points on the volume part's ends: If horizontal slices of V are connected, V corresponds to one empty region r in an intersection plane. Maximality of V is equivalent to V starting and ending at heights where r disappears, splits or is merged with other empty regions. Due to Lemma 4.3, this happens in critical points, and those critical points in which empty regions change are per definition important. □

Remark. The important critical points defined here correspond to component-critical points (as used in [13, 7]) of the object's exterior, seen as 3-manifold with boundary. We distinguish important and unimportant critical points with respect to the height function using only the surface of the object (see Section 5). In the case of a general Morse function, volumetric representations (e.g. tetrahedral meshes) are required in the definition and identification of component-critical points.

4.6 Events

Every important or unimportant critical point induces an event:

Definition 4.11. An *event* for the sweep plane algorithm is a critical point v with its region sets $L^*(v)$ and $L_*(v)$.

The list of events is sorted according to z coordinates. Critical points at the same z coordinate form a single event if they are connected by intersection polygons, otherwise they can be dealt with in an arbitrary order.

5 Computation of events

The critical points of the triangulated solid can easily be identified by considering their incident halfedges. Now the aim is to compute the region sets for these critical points.

5.1 Oriented intersection polygons

The polygons inducing the local decomposition of a plane through a critical vertex v are computed by tracing the intersections along the surface, starting in v. For determining whether a region is filled or empty, the intersection polygons are computed with a certain orientation.

Definition 5.1. An intersection polygon is *oriented* such that in the intersection plane an empty region touches the polygon on the left, and a filled region on the right-hand side (if viewed from above).

Polygons are oriented using the orientation of the solid's triangles: starting in a given vertex v, an oriented intersection polygon is started through each incident triangle in which the predecessor of v lies below the successor of v; see Figure 6.

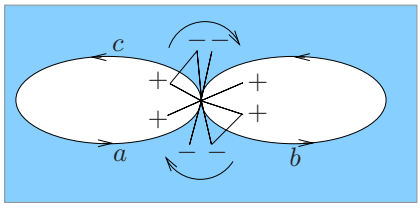

Figure 6: Oriented intersection polygons in a horizontal plane through the saddle point in Figure 5.b. In the critical point in the middle, incident edges above the intersection plane are marked by +, those below by −. The arrows indicate the orientation of the incident triangles, so here the face normal points away from the viewer. The pictured triangles are those in which oriented intersection polygons are started.

In order to determine the regions of the subdivision, the oriented polygons are sorted such that each polygon knows its surrounding and surrounded polygons. Then empty regions are located inside each counter-clockwise oriented polygon, and outside the outermost polygon if it is clockwise oriented.

5.2 Marginal intersection polygons

The region sets of a critical vertex contain marginal regions, so a similar concept is needed for intersection polygons:

Definition 5.2. Intersection polygons forming boundaries of marginal regions are called *marginal polygons*. As for regions we distinguish between *upper* and *lower* polygons.

In a saddle point like in Figure 5.b, the sets of upper and lower polygons contain the same points. However, all points are united in one lower polygon (which is no longer simple), while the set of upper polygons consists of two polygons forming the boundaries of the two upper empty regions.

Marginal intersection polygons are computed without applying limiting processes: When a polygon q meets a

saddle point, it has several possibilities where to continue. Due to the orientation of q, the surface goes up on the right-hand side, and down on the left-hand side of q (viewed from the direction of face normals). So a lower polygon chooses the leftmost valid possibility in order to remain on the same lower region, whereas an upper polygon chooses the rightmost valid possibility.

In Figure 6, for example, when a lower intersection polygon meets its start point (the critical vertex) coming from a, it continues toward b along the large lower empty region. On the other hand an upper polygon will try to continue along the upper empty region in direction c where it was started, and thus terminate there. In the figure the rising incident edges appear on the right-hand side of the polygons as the intersection plane is viewed from above, while the face normal vectors point down.

5.3 Summary and algorithm

Summing up, we distinguish important and unimportant critical points v using their region sets, i.e. by analyzing the horizontal intersection polygons which contain v; see Algorithm 2.

Algorithm 2 collectEvents(surface mesh M)

```
eventList ← empty list
for all vertices v ∈ M do
  if v is critical then
    e ← new event induced by v
    trace horizontal polygons starting in v
    determine region sets and store them in e
    push e to eventList
  end if
end for
```

6 The event handler

For a given event, i.e. a critical point v with its region sets, the event handler first marks those edge groups in the sweep plane's status whose associated flow volumes contain v, and associates them with empty regions in $L_*(v)$. If v is unimportant, the unique (according to Lemma 4.9) marked edge group is adapted. Otherwise, the flow volumes corresponding to the marked edge groups are closed and reported as output, and a new flow volume is created for each empty region in $L^*(v)$. Additionally the marked edge groups are replaced by new groups corresponding to the new flow volumes. In the following two sections we will consider these operations in more detail.

6.1 Associating lower regions with edge groups

Each lower empty region r is associated with the edge group which represents the corresponding global lower region in the status.

The region sets of a minimum v contain exactly one lower region r. If r is empty, the corresponding RE-group has to be identified. An empty region is constructed for each RE-group by computing oriented intersection polygons starting in the edges of the group. If such a region contains the minimum v, the edge group is associated with the lower region of v.

A saddle point or maximum v has to be hit by an intersection component represented by an edge in the sweep plane's status. So here an edge group is associated with a lower empty region r of v if at least one edge of the group intersects a polygon bounding r. A slightly more elaborate procedure, which exploits the intersection polygons' orientation, has to be applied if an edge of the status directly hits the critical vertex itself, as otherwise all lower regions of v might be identified in this case.

For an event e the algorithm is summarized in Algorithm 3.

Algorithm 3 associateGroups(status,e)

```
if e is no minimum then
  for all lower empty regions r ∈ L*(e) do
    for all edge groups g ∈ status do
      if r is hit by an edge in g then
        associate r with g
        continue with next region
      end if
    end for
  end for
else if e is an unimportant minimum then
  r ← the unique lower empty region of e
  for all edge groups g ∈ status do
    r_g ← empty region computed from g
    if r_g contains e then
      associate r with g
      do not check further edge groups
    end if
  end for
end if
```

Its functionality is described in the following Lemma.

Lemma 6.1. *Algorithm 3 correctly associates edge groups with lower regions of an event.*

Proof. The global lower subdivision $S_*(h)$ forms a decomposition of the intersection plane. A *minimum* v at height h is not reached by lower intersection components, so v has to be contained on the inside of exactly one global lower region r_g. If r_g is empty, v is an unimportant minimum (Fig. 7.b). By constructing global empty regions from the edge groups in the status, the event handler correctly identifies r_g. Otherwise, i.e. if r_g is filled, no associations have to be established. This is the case if also the unique lower region of v is filled, i.e. if v is an important minimum (Fig. 7.a)

The boundary of the lower empty regions of a *saddle point* (Fig. 7.c) consists of lower polygons through v. The status contains an edge through every lower intersection polygon, so each lower region is hit by an edge of the status. On the other hand, since each edge group represents

one flow volume, i.e. one empty region in a lower intersection, every lower region is hit by exactly one edge group. Thus the event handler marks exactly one edge group for each lower region. □

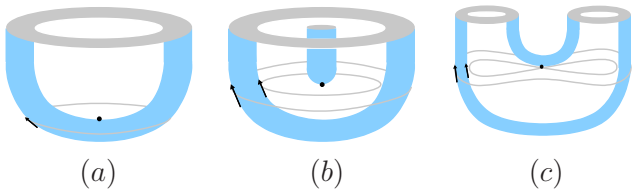

Figure 7: Different critical points (black dots), the lower marginal polygons and possible representative edges of the status at the respective height (black arrows). Saddle point (c) is hit by a lower polygon, important (a) and unimportant (b) minima are not.

6.2 Update of the sweep plane's status

When adapting the sweep plane's status, important and unimportant critical points are handled differently.

In an *important* critical point v, flow volumes are started or closed, according to Lemma 4.10. Since the generated flow volumes are required to know the global empty regions forming their upper and lower end, it is necessary to add the bounding polygons of the global regions that do not touch v to the localized subdivision: Some edges of the associated edge group of a lower empty region r might not hit bounding polygons of r. Starting from those edges, oriented intersection polygons are computed and added to r. Then each polygon added to a lower empty region of v is also added to the upper region containing it. Finally, edge groups associated with lower regions of v are removed from the status and their corresponding flow volumes are closed. For each upper empty region r of v, an edge group g is added to the status. The group g contains an edge passing through each bounding polygon of r and it is associated with a new flow volume f.

In an *unimportant* critical point v, no re-grouping of edges in the status is necessary, merely the edge group g associated with the unique lower empty region of v is adapted: Edges hitting lower polygons through v are removed from g and replaced by one edge for each polygon bounding the unique upper empty region of v.

The algorithm for the status update in an event e is summarized in Algorithm 4.

The following Lemma summarizes the functionality of the event handler.

Theorem 6.2. *The event handler updates the status correctly and generates the desired output.*

Proof. According to Lemma 6.1, the event handler correctly associates lower regions with edge groups. Edge groups whose corresponding flow volumes do not touch the critical vertex are not associated with any lower regions, and therefore left unchanged. In the other edge groups, those edges intersecting lower polygons through v are replaced by edges through upper polygons through v, which accounts for the changes in the intersection components. The remaining edges are left unchanged in the case of unimportant critical points, or they are moved to the correct new edge groups when treating important critical points, so no intersection components are lost.

The correctness of the output follows from Lemma 4.10: The idea of tracing empty regions while moving the sweep plane makes sure that the computed volume parts fulfill the first two properties in Definition 2.1. Since the computed volume parts additionally start and end in important critical points, they are flow volumes due to Lemma 4.10. □

Algorithm 4 adaptStatus(status,e)

```
if e is unimportant then
  r1 ← the lower empty region of e
  g ← the associated edge group of r1
  r2 ← the upper empty region of e
  adapt edge group g as described in Section 6.2
else
  for all lower empty regions r ∈ L*(e) do
    g ← the associated edge group of r
    add polygons to r starting from edges in g
        //see Section 6.2
    f ← the associated flow volume of g
    set upper end of f to r
    report f
    remove g from status
  end for
  for all upper empty regions r ∈ L^*(e) do
    add polygons from lower regions to r
        //see Section 6.2
    f ← new flow volume starting in r
    g ← new edge group with edges through r
    associate g with volume f
    insert g in status
  end for
end if
```

We are now ready to prove the main result of our work.

Proof of Theorem 3.1. The lowest vertex of the triangulated solid is always a minimum, representing the lowest event. So the sweep plane's status is properly initialized by an empty edge group, representing the free volume below the object.

According to Lemma 4.3, intersection components change only in critical points. Thus it is always possible to climb along intersection components to the height of the next event, as no component disappears between successive events. Furthermore an edge is found on every intersection component slightly below the next event, as intersection components do not appear or split between successive events. So the update routine does not destroy the validity of the sweep plane's status.

Finally, according to Theorem 6.2, the event handler correctly adapts the status and generates the desired output. □

7 Results

The presented algorithm has been implemented in C++ using the computational geometry algorithms library (CGAL) [1] and tested on a PC with 2.66GHz and 8GB memory.

We decomposed a watertight surface mesh of the Stanford bunny in Figure 8(a), consisting of 71 000 triangles. Since our algorithm decomposes the exterior of an oriented surface mesh, it was applied to a mold of the bunny as shown in Figure 8(b). The computed decomposition consists of 307 flow volumes. Especially on the base of the bunny many small flow volumes arise due to oscillations of the surface mesh resulting from closing the holes there. The significant flow volumes are displayed in Figure 8(c). The computation took less than one second.

As second example, the algorithm was applied to the watertight Rolling Stage model with a resolution of 40 000 triangles. The object's exterior, which would be relevant for dip-coating simulation, was decomposed into 19 flow volumes in less than half a second; see Figure 9.

The shape presented in Figure 2 consisting of 28 000 triangles was processed in 0.06 seconds. The object in Figure 10 consists of 18 copies of a variation of this shape, arranged horizontally with intersections. This new object was triangulated in different resolutions ranging from 30 000 to 3 000 000 triangles. The resulting meshes were decomposed into 310 flow volumes within 0.5 to 5 seconds; see also Section 8.2. Thus, our algorithm compares favorably with existing algorithms for the computation of Reeb graphs of 3-manifolds with boundary, although the algorithms are not directly comparable since they use different input (surface/volumetric mesh) and produce different output (Reeb graph of interior/exterior of the solid): We process the mesh of 3 million triangles within 5 seconds, while the state-of-the-art algorithm presented in [35] processes an object of similar complexity, given as mesh of 3.5 million tetrahedra, in 7.8 seconds.

Finally, our algorithm has been tested successfully on several real geometries, including two complete car bodies (see Section 8.3).

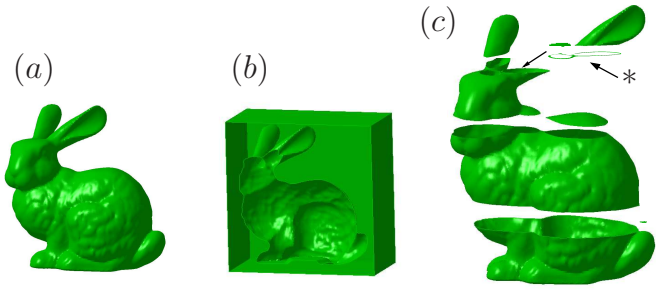

Figure 8: (a) Stanford bunny. (b) Mold of the bunny. (c) Computed flow volumes when applying our algorithm to the mold. The flat volume marked by an asterisk (*) is due to the fact that the upper part of the bunny splits into three parts: the second ear splits from the head slightly above the first, thereby creating one very flat flow volume in between. A similar phenomenon causes the small volume at the tail.

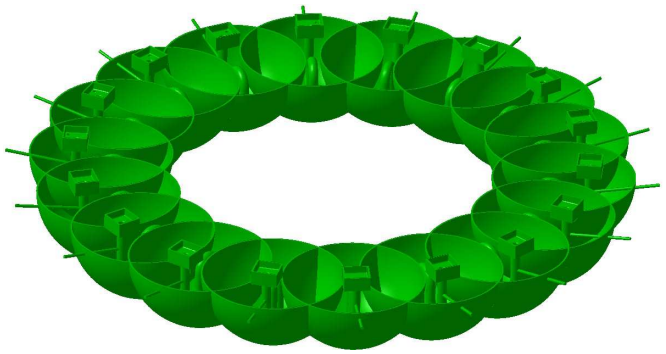

Figure 10: Test object, 455 critical points and 310 flow volumes are found.

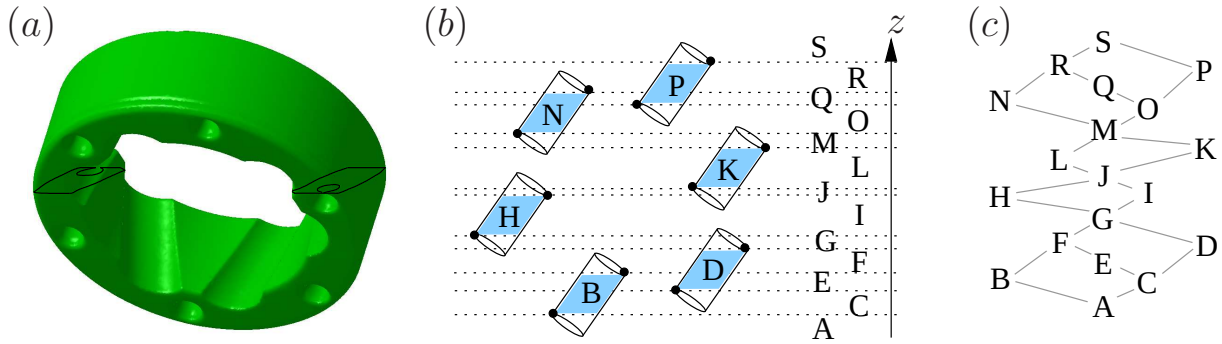

Figure 9: (a) Rolling Stage model taken from the AIM@SHAPE Shape Repository, displayed with a representative horizontal cut. (b) Schematic view of the volume decomposition, with letters denoting the flow volumes. Two flow volumes are created at the lower end of each hole, one inside the hole (shaded) and one on the unbounded exterior of the object. This is the case e.g. at the saddle point on the right-hand component of the displayed horizontal cut in (a), where flow volumes K and L are created. At the upper end of each hole, one unbounded volume is created, like at the saddle point near the left-hand component of the displayed cut, where volume J starts. (c) Volume graph visualizing the connectivity between flow volumes.

8 Computation times

The analysis uses the following characteristics of input data:

- n ... Number of triangles in the triangulated solid
- m ... Size of longest horizontal intersection polygon
- p ... Maximal length of a path
- e ... Number of events
- k ... Maximal number of polygons in an intersection
- l ... Maximal number of horizontal polygons passing a critical point

All these quantities depend on the geometry of the solid, but only n, m and p depend on the triangulation.

8.1 Theoretical results

The presented algorithm extensively uses planar polygons. The computational complexity of the construction of polygons, as well as some basic computations (like testing the position of a point relative to a polygon), is proportional to the size of the polygon.

Computation of events

Critical points can be identified in $O(n)$ time, as a small, constant amount of work arises for each vertex, and the number of vertices is proportional to the number of triangles in the mesh. For computing region sets of a critical vertex, at most l intersection polygons are computed, taking $O(lm)$ time. Sorting those takes $O(l \log l)$ comparisons of complexity m. Finally the events are sorted according to z-coordinates in $O(e \log e)$ time. So the total time needed for the computation of events is $O(n + elm + elm \log l + e \log e)$.

Event handler: Associate lower regions with edge groups

The lower polygons of a saddle point may contain l polygons, which are tested for intersections with at most k REs in $O(klm)$ time. When handling an unimportant minimum, at most k intersection polygons are computed from REs, and the position of the critical point relative to these polygons is determined in $O(km)$ time. An important minimum on the other hand causes no effort in this step. So in total, for all events, the computational complexity of the first step of the event hander is $O(eklm)$.

Event handler: Adapt status

In an unimportant event, merely at most l REs are removed from or added to the status. In an important event, at most k polygons are built for globalizing the critical point's lower regions. Each of them is tested against the at most l upper polygons to determine which upper region it belongs to. So this step of the event handler in total also takes $O(eklm)$ time.

RE-update

After each event, at most k REs are updated, which takes $O(p)$ time each. So the RE-update is in total proportional to ekp. Adding up, the sweep plane algorithm takes $O(eklm + ekp)$ time.

Heuristic estimates

For simplifying practical analysis, the previous estimates can be slightly simplified by heuristic considerations. In the complexity estimate for the computation of events, the term $\log e$ is typically much smaller than the maximal polygon size m, simplifying the estimate to $O(n + elm \log l)$. In the estimate for the sweep plane algorithm, the maximal path length p typically will not outweigh the maximal polygon size m, reducing the estimate to $O(eklm)$. For the total computation time, $\log l$ can obviously be estimated by k, leading to an overall complexity of $O(n + eklm)$. Section 8.3 indicates that neglecting the $O(n)$-term still leads to reasonable estimates in general cases.

8.2 Dependence of computation time on triangulation

Only three of the quantities considered in the last section depend on the triangulation. One of them, the maximal path length, does not appear in the heuristic complexity estimate. The influence of the remaining two quantities, namely n and m, on the computation time, can be visualized by considering the same object in different triangulations. This has been done for the object shown in Figure 10 with triangulations using between 30 000 and 3 000 000 triangles.

Obviously the total number of triangles n and the maximal polygon length m are correlated. Since m is the complexity of a (one-dimensional) line on a (two-dimensional) surface of complexity n, a dependence similar to $m \sim \sqrt{n}$ can be expected. Figure 11 confirms this estimate for the test case.

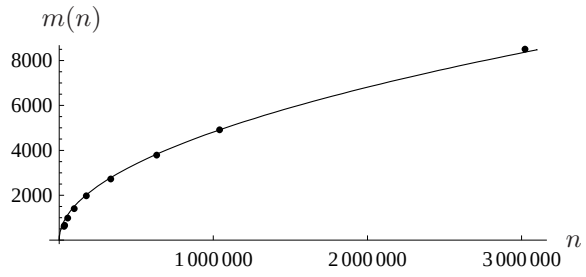

Figure 11: Dependence of polygon length m on triangulation size n in test example. The plotted best approximate function is given by $m(n) = 4.8\sqrt{m}$.

Now consider the influence of n and m on the computation time for determining events, t_e, and for the sweep plane algorithm, t_s. According to theoretical analysis, $t_e(n, m) = c_1 n + c_2 m$ and $t_s(m) = c_3 m$ with constants c_1, c_2, c_3. Since m can be expressed by n with satisfying accuracy in this example, the best approximate functions for t_e and t_s can be expressed as functions of n for simplifying visualization, i.e. $t_e(n) := t_e(n, m(n))$ and $t_s(n) := t_s(m(n))$, using the previously fitted function $m(n)$. According to Figure 12 the data obtained for t_e and t_s complies with the theoretic predicates.

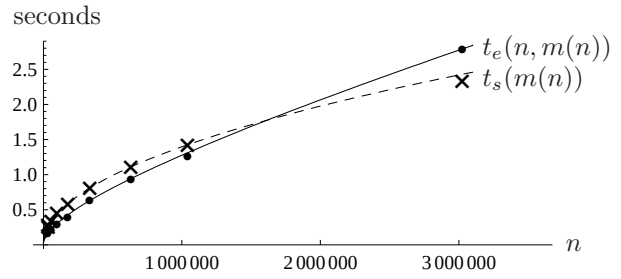

Figure 12: Computation times depending on triangulation size n. For small n, the $O(m) = O(\sqrt{n})$-term in t_e and t_s dominates the computation costs, for larger n it is outweighed by the linear $O(n)$ term in t_e.

8.3 Dependence of computation time on geometry

The algorithm has been tested on 12 real geometries of different size and complexity, including various parts of car bodies and two complete car bodies, see Table 1. In order to protect the commercial interests of ECS' customers, pictures of the car bodies and body-parts cannot be included here.

When considering varying geometries, all input quantities have to be taken into account. To validate the (heuristic) theoretical results, an at most linear dependence of the computation times t_e and t_s on the variables

$$y_e := e\, l\, m \log l$$
$$y_s := e\, k\, l\, m$$

has to be shown. As hinted in Section 8.1 the $O(n)$ term is omitted, as it turns out not to exhibit significant influence. Figure 13 shows doubly logarithmic plots for these cases.

9 Implementation remarks

We conclude with some comments on the implementation of the algorithm.

Shortened intersection polygons

To avoid computing and storing all intersection points twice, for upper and for lower polygons, one could terminate polygons when they meet their start point for the first time. These shorter polygons can then be lower and upper polygons at the same time. However, the full marginal polygons are needed for the construction of region sets, such that connections between the short polygons have to be determined. Especially in cases where intersection polygons contain several critical points, this leads to some inconvenient cases. Anyhow these shorter polygons are used in the implementation.

Horizontal areas in the triangulated solid

In the definition of events, all vertices have been considered separately. However, the triangulated solid may contain horizontal surface patches, which may be critical without containing any critical vertices. These horizontal areas require special treatment at different stages of the algorithm, like in the definition and computation of events and horizontal intersection polygons, or when climbing along the surface during the RE-update. It would be possible to eliminate horizontal edges and areas by applying a small rotation to the object. In the context of dip-coating simulations, however, it is highly desirable to obtain precise information about them in order to correctly identify the connections between flow volumes. Therefore we fully implemented handling the horizontal areas. The details, which are omitted here, can be found in [33].

10 Further tasks and future work

Besides the construction of the actual flow volume decomposition, some additional information is required for the simulation of dip coating processes.

Capacity of flow volumes

For the simulation of dip coating processes the capacity of each flow volume is required. Using Gauss' theorem the capacity of a flow volume can be computed by evaluating a simple formula for each surface triangle bounding the volume. The computation of volume capacities is therefore linear in the number of triangles of the mesh. For determining the bounding triangles, each flow volume remembers all edges which have, at some time, belonged to the associated edge group in the sweep plane's status. As there are representative edges on every intersection component inside a flow volume, all triangles on the flow volume's boundary can be identified by considering adjacent triangles of those REs within the volume's z-bounds.

Construction of a volume graph

The sweep plane algorithm constructs the flow volumes forming the decomposition of free volume. These flow volumes need to be connected to a geometry graph, which links each flow volume v to the volumes directly above and below, which are only separated from v by a horizontal plane. To this end every important critical point stores the volumes reaching it from below and from above. Usually each lower volume is connected to each upper volume of a critical point. If a critical point has several lower and several upper volumes, it is however possible that some lower volumes are only connected to certain upper volumes and vice versa.

Flow paths

So far, a volume graph was constructed, where each volume knows its direct neighbors on top and bottom. However, another important question arises in the simulation of a dip-coating process: If a certain flow volume is filled up, which other flow volumes will be filled next? Here often the liquid will flow or drop through several other flow volumes until it reaches a local minimum. Which flow volumes exactly are influenced if one flow volume starts overflowing cannot be determined from the volume graph; see Figure 14.

Thus, in order to find these flow paths, liquid leaking from an overflowing volume has to be followed along the triangulated solid in gradient direction. If a saddle point is met which allows several further flow paths, all of them

model	n	m	k	l	e	v	y_e	y_s	t_e	t_s
part 1	15 416	501	10	10	25	34	$2.88\ 10^5$	$1.25\ 10^6$	$0.07s$	$0.01s$
part 2	15 894	522	8	6	68	11	$3.82\ 10^5$	$1.70\ 10^6$	$0.15s$	$0.01s$
part 3	26 282	761	37	23	14	54	$7.68\ 10^5$	$9.07\ 10^6$	$0.10s$	$0.03s$
part 4	29 906	761	7	7	35	24	$3.63\ 10^5$	$1.31\ 10^6$	$0.10s$	$0.01s$
part 5	47 514	699	13	10	120	65	$1.93\ 10^6$	$1.09\ 10^7$	$0.20s$	$0.07s$
part 6	47 582	2009	32	28	37	37	$6.94\ 10^6$	$6.67\ 10^7$	$0.49s$	$0.06s$
part 7	89 396	535	3	4	38	1	$1.13\ 10^5$	$2.44\ 10^5$	$0.08s$	$0.01s$
part 8	125 050	630	6	4	58	7	$2.03\ 10^5$	$8.77\ 10^5$	$0.11s$	$0.01s$
part 9	145 210	872	16	12	243	75	$6.32\ 10^6$	$4.07\ 10^7$	$0.50s$	$0.15s$
part 10	177 140	1983	54	6	1614	784	$3.44\ 10^7$	$1.04\ 10^9$	$2.75s$	$5.14s$
car 1	1 080 814	6369	116	9	11 247	6 455	$1.42\ 10^9$	$7.49\ 10^{10}$	$39.19s$	$114.13s$
car 2	1 300 202	4501	148	17	14 361	8 064	$3.11\ 10^9$	$3.62\ 10^{11}$	$43.84s$	$175.64s$

Table 1: Computation time results for different real examples, with v denoting the number of computed flow volumes, and the other variables defined as before. The last two examples are complete car bodies. In part 7, l appears to be greater than k, which is caused by different ways of storing and counting the polygons; see Section 9. Only in the large examples there is a significant difference between l and k.

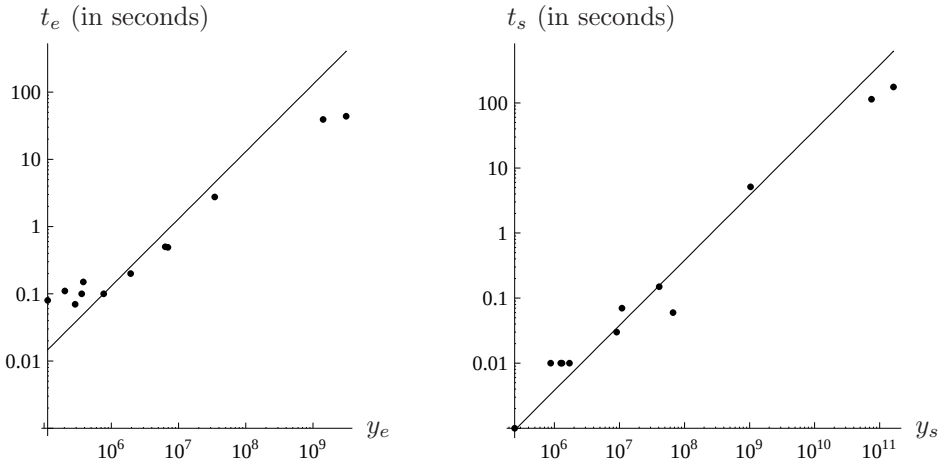

Figure 13: Doubly logarithmic plots of computation time for determining events and for the sweep plane algorithm, depending on compound input variables y_e and y_s. In both cases the points (y, t) lie near a line with slope 1, which confirms linear dependence of the computation times on the estimated variables. Since the estimates y are rather pessimistic, the slopes of the computation times are actually smaller than 1.

need to be considered. If a point is reached where the liquid detaches from the surface and starts falling through free volume, the surface triangles of the respective flow volumes have to be searched to find the point where the liquid hits the surface again. These procedures are not complicated in principle, but require careful implementation.

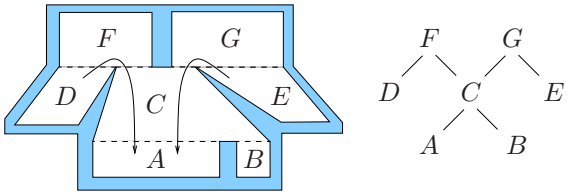

Figure 14: Visualization of flow paths: the volume graph (right) is symmetric, it does not reveal where liquid goes after overflowing volumes D or E.

Rotating mesh

This paper considers the volume decomposition and dip-coating simulation for a stationary triangulated solid. The simulation, however, is supposed to be able to rotate the object while moving it through the different baths. This rotation is approximated by discrete angular steps. Each of them can be handled like the stationary case, i.e. one can compute a volume graph and afterward balance liquid surfaces by the simulation kernel consecutively for each angular step. For more advanced versions of the simulation, it would, however, be preferable not to compute the complete volume graph for each angular step separately, but to find a way of reusing information of one volume graph for the construction of the next graph, at least in areas of the surface where no major changes occur in the structure of the graph.

In any case the filling levels derived for one angular step have to be carried over to the flow volumes of the next angular step. One approach to this is to freeze the liquid during the rotation, i.e. to mark positions of the surface where it is touched by liquid before rotation, and rotate

these positions with the mesh. After the rotation these blocks of frozen liquid are distributed to the new flow volumes containing the respective surface triangles. When they are defrosted, liquids should not simply be flowed to the bottoms of their new containing flow volumes, but rather something like flow paths has to be used in order to allow liquids to move appropriately through the new volume graph.

Conclusion

Motivated by an application to the simulation of the dip-coating process of automotive parts, we described an algorithm for automatically decomposing triangulated solids into flow volumes. These volumes correspond to the edges of the Reeb graph of the volume's exterior (considered as manifold with boundary) with respect to the height function. Since no inner extrema and saddle points are present, it is possible to generate this segmentation solely by analyzing the boundary of the triangulated solid. We designed a sweep plane algorithm which uses local information about horizontal intersections near critical points of the height function to construct the volume decomposition. As demonstrated by the examples, this algorithm is suitable for realistic data in automotive applications.

Acknowledgements This work has been supported by MAGNA POWERTRAIN Engineering Center Steyr GmbH & Co.KG. The authors would like to thank the reviewers for their comments which have helped to improve this paper.

References

[1] CGAL, Computational Geometry Algorithms Library, http://www.cgal.org.

[2] M. Attene, S. Biasotti, and M. Spagnuolo. Shape understanding by contour-driven retiling. *The Visual Computer*, 19:127–138, 2003.

[3] C. Bajaj and T. Dey. Convex decomposition of polyhedra and robustness. *SIAM J. Comput.*, 21(2):339–364, 1992.

[4] T. F. Banchoff. Critical points and curvature for embedded polyhedral surfaces. *Amer. Math. Monthly*, 77(5):pp. 475–485, 1970.

[5] M. Berg, O. Cheong, M. Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 2008.

[6] S. Berretti, A. Del Bimbo, and P. Pala. Partitioning of 3D meshes using Reeb graphs. In *Proc. Conf. on Pattern Recognition*, pages 19–22. IEEE Computer Society, 2006.

[7] S. Biasotti, D. Giorgi, M. Spagnuolo, and B. Falcidieno. Reeb graphs for shape analysis and applications. *Theoretical Computer Science*, 392(1-3):5–22, 2008.

[8] S. Biasotti, S. Marini, M. Spagnuolo, and B. Falcidieno. Sub-part correspondence by structural descriptors of 3D shapes. *Computer-Aided Design*, 38(9):1002–1019, 2006.

[9] H. Carr, J. Snoeyink, and U. Axen. Computing contour trees in all dimensions. In *Proc. Sympos. on Discrete Algorithms*, pages 918–926. Society for Industrial and Applied Mathematics, 2000.

[10] B. Chazelle, D. Dobkin, N. Shouraboura, and A. Tal. Strategies for polyhedral surface decomposition: An experimental study. *Computational Geometry*, 7(5-6):327–342, 1997.

[11] B. Chazelle and L. Palios. Decomposing the boundary of a nonconvex polyhedron. *Algorithmica*, 17(3):245–265, 1997.

[12] S.-W. Cheng, T. K. Dey, E. A. Ramos, and T. Ray. Quality meshing for polyhedra with small angles. In *Proc. Sympos. on Comput. Geom.*, pages 290–299. ACM, 2004.

[13] Y.-J. Chiang, T. Lenz, X. Lu, and G. Rote. Simple and optimal output-sensitive construction of contour trees using monotone paths. *Computational Geometry*, 30(2):165–195, 2005.

[14] K. Cole-McLaughlin, H. Edelsbrunner, J. Harer, V. Natarajan, and V. Pascucci. Loops in Reeb graphs of 2-manifolds. In *Proc. Sympos. on Comput. Geom.*, pages 344–350. ACM, 2003.

[15] H. Doraiswamy and V. Natarajan. Efficient output-sensitive construction of Reeb graphs. In *Proc. Sympos. on Algorithms and Computation*, pages 556–567. Springer-Verlag, 2008.

[16] H. Doraiswamy and V. Natarajan. Efficient algorithms for computing Reeb graphs. *Computational Geometry*, 42(6-7):606–616, 2009.

[17] S. Ehmann and M. Lin. Accurate and fast proximity queries between polyhedra using convex surface decomposition. *Computer Graphics Forum*, 20(3):500–511, 2001.

[18] Electrocoat Association. *Electrocoating: A Guidebook for Finishers*. Hanser Gardner Pubns, 2002.

[19] P. J. Frey and P.-L. George. *Mesh Generation. Application to Finite Elements (Second Edition)*. Hermes Science Europe, 2000.

[20] J. Ha, K. Yoo, and J. Hahn. Characterization of polyhedron monotonicity. *Computer-Aided Design*, 38(1):48–54, 2006.

[21] P. Hachenberger. Exact Minkowski sums of polyhedra and exact and efficient decomposition of polyhedra into convex pieces. *Algorithmica*, 55(2):329–345, 2009.

[22] F. Lazarus and A. Verroust. Level set diagrams of polyhedral objects. In *Proc. Sympos. on Solid Modeling and Appl.*, pages 130–140. ACM, 1999.

[23] J.-M. Lien and N. M. Amato. Approximate convex decomposition of polyhedra and its applications. *Comput. Aided Geom. Des.*, 25:503–522, 2008.

[24] Y. Matsumoto. *An introduction to Morse theory.* American Mathematical Society, 2002.

[25] J. Milnor. *Morse theory.* Princeton University Press, 1963.

[26] V. Pascucci, G. Scorzelli, P.-T. Bremer, and A. Mascarenhas. Robust on-line computation of Reeb graphs: simplicity and speed. *ACM Trans. Graph.*, 26:58.1–58.9, 2007.

[27] G. Patané, M. Spagnuolo, and B. Falcidieno. A minimal contouring approach to the computation of the Reeb graph. *IEEE Transactions on Visualization and Computer Graphics*, 15, 2009.

[28] D. Rypl. Sweeping of unstructured meshes over generalized extruded volumes. *Finite Elements in Analysis and Design*, 46(1-2):203–215, 2010.

[29] J. Schöberl. NETGEN an advancing front 2D/3D-mesh generator based on abstract rules. *Computing and Visualization in Science*, 1:41–52, 1997.

[30] D. Shattuck and R. Leahy. Automated graph-based analysis and correction of cortical volume topology. *IEEE Transactions on Medical Imaging*, 20(11):1167–1177, 2001.

[31] J. R. Shewchuk. Tetrahedral mesh generation by Delaunay refinement. In *Proc. Sympos. on Comput. Geom.*, pages 86–95. ACM, 1998.

[32] Y. Shinagawa and T. Kunii. Constructing a Reeb graph automatically from cross sections. *IEEE Computer Graphics and Appl.*, 11(6):44–51, 1991.

[33] B. Strodthoff. Erstellen eines Geometriebaums zur Simulation von Tauchlackierprozessen. Master's thesis, Johannes Kepler Universität, 2010.

[34] K. Sugihara. Sliver-free perturbation for the Delaunay tetrahedrization. *Computer-Aided Design*, 39(2):87–94, 2007.

[35] J. Tierny, A. Gyulassy, E. Simon, and V. Pascucci. Loop surgery for volumetric meshes: Reeb graphs reduced to contour trees. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1177–1184, 2009.

[36] T. Tung and F. Schmitt. Augmented Reeb graphs for content-based retrieval of 3D mesh models. In *Proc. of the Shape Modeling International*, pages 157–166. IEEE Computer Society, 2004.

[37] H. Versteeg and W. Malalasekra. *An Introduction to Computational Fluid Dynamics: The Finite Volume Method (Second Edition).* Prentice Hall, 2007.

[38] E. Wang and Y. S. Kim. Form feature recognition using convex decomposition: results presented at the 1997 ASME CIE feature panel session. *Computer-Aided Design*, 30(13):983–989, 1998.

[39] E. Wang and Y. S. Kim. Feature-based assembly mating reasoning. *Journal of Manufacturing Systems*, 18(3):187–202, 1999.

[40] H. Zhang, G. Zhao, and X. Ma. Adaptive generation of hexahedral element mesh using an improved grid-based method. *Computer-Aided Design*, 39(10):914–928, 2007.

[41] E. Zuckerberger, A. Tal, and S. Shlafman. Polyhedral surface decomposition with applications. *Computers & Graphics*, 26(5):733–743, 2002.