

# Testing a Rotation Axis to Drain a 3D Workpiece

Yusuke Yasui, Sara McMains

*University of California, Berkeley*

---

## Abstract

Given a triangular mesh defining the geometry of a 3D workpiece filled with water, we propose an algorithm to test whether, for an arbitrary given axis, the workpiece will be completely drained under gravity when the rotation axis is set parallel to the ground and the workpiece is rotated around the axis. Observing that all water traps contain a concave vertex, we solve our problem by constructing and analyzing a directed “*draining graph*” whose nodes correspond to concave vertices of the geometry and whose edges are set according to the transition of trapped water when we rotate the workpiece around the given axis. Our algorithm to test whether or not a given rotation axis drains the workpiece outputs a result in about a second for models with more than 100,000 triangles after a few seconds of preprocessing.

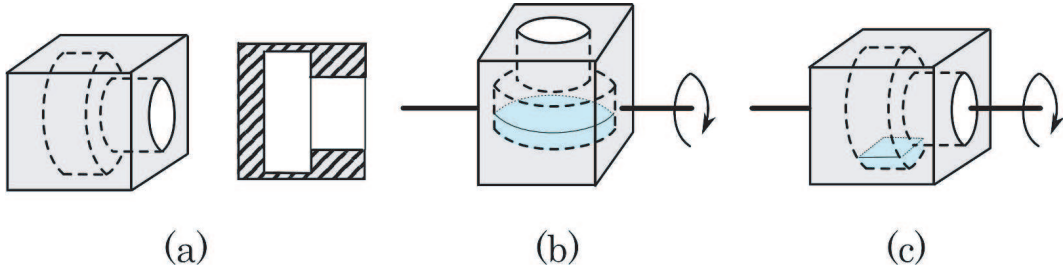
*Keywords:*

draining, rotation axis, directed graph, water traps, cleanability, manufacturing

---

## 1. Introduction

Cleaning engine components to remove hard particle contaminants introduced during the manufacturing process is becoming a significant issue for industry [1, 2, 3]. Manufacturing byproducts such as chips from machining and sand from casting are commonly cleaned off the surfaces of workpieces using high pressure water-jets. However, if the workpiece has complicated concave regions, the cleaning water may not easily drain from the workpiece. In order to minimize the subsequent draining time, our industrial partner first mounts workpieces on a slowly rotating carrier so that gravity can drain out as much water as possible. Their current set-up rotates in one direction (either clockwise or counterclockwise) around a single axis oriented parallel to the ground.



**Figure 1:** *The choice of a rotation axis matters to drain trapped water. (a) A workpiece geometry and its cross section (b) A rotation axis relative to the workpiece that can drain the trapped water. (c) A rotation axis relative to the workpiece that cannot drain the trapped water.*

The choice of a rotation axis determines whether trapped water will drain. Figure 1 illustrates an example. Given a workpiece geometry we would like to drain (Figure 1 (a)) and a rotation axis set parallel to the ground, if we initially choose a rotation axis relative to the workpiece as shown in Figure 1 (b), we can drain the trapped water. On the other hand, if we choose a rotation axis relative to the workpiece as shown in Figure 1 (c), we cannot drain the trapped water.

Our ultimate goal is to find a rotation axis for a given workpiece geometry such that when the workpiece is first oriented with this axis parallel to the ground and then rotated slowly around the axis, all water drains from all voids of the workpiece. As a first step toward this goal, we propose an algorithm to test whether or not clockwise or counterclockwise rotation around a given axis in 3D space can drain all trapped water from a workpiece whose geometry is represented as a triangulated mesh.

### 1.1. Related Work

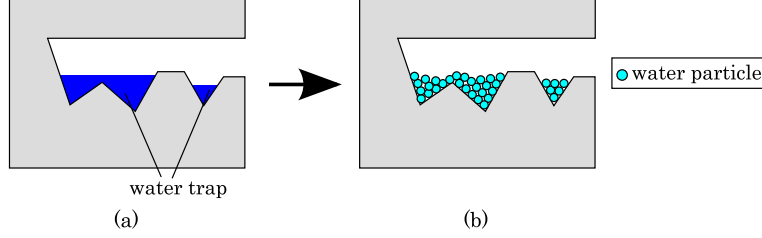
The most straightforward approach to solve this problem might be using a general-purpose physics based approach such as computational fluid dynamics (CFD). Although the power of computers is increasing every year, the computational cost of such a physics-based approach is still too expensive to be suitable for applications that require interactivity. Since we would like to provide interactive feedback to designers, we need a real-time algorithm that does not rely on a computationally expensive method that can take hours to converge.

In the computer graphics community, several efforts have been made to accelerate algorithms borrowed from computational sciences while maintaining plausibility [4]. The first real-time GPU (Graphics Processing Unit) implementation of fluid simulation using a regular grid of cubical cells was reported in [5]. Unfortunately, because the algorithm accuracy is dependent on the 3D grid resolution, it is not appropriate for our complex target geometries since we would be required to split space into a tremendous number of grid cells to perform the simulation reliably. To avoid this issue, particle-based approaches using smoothed particle hydrodynamics (SPH) are popular for real-time simulations since they do not require a grid throughout the whole domain [6, 7]. Although particle systems can produce attractive visual results, they cannot match the accuracy of fluid simulation unless the number of the simulated particles is very high, but when the number of particles increase, the performance suffers.

Since performing physics-based simulation in realtime on complicated geometry is still challenging, and we do not care about the full details of the fluid flow, only whether or not the workpiece drains completely, we are motivated to devise an algorithm to solve our problem geometrically to reduce computational cost. It combines analysis of (free) fluid flow and accessibility from a geometric perspective.

In the case of fluid flows inside complex geometric models, a similar problem, considering the problem of draining water (a single particle) out of a closed polygon by rotating the shape in 2D space, was introduced by Aloupis et al. [8]. Given a closed polygon and a trapped water particle inside of it, they proposed an algorithm to find how many holes must be punctured to drain them. Letting  $n$  be the number of vertices of a given polygon, they showed that  $\lfloor n/6 \rfloor$  holes are sometimes necessary and  $\lceil n/4 \rceil$  holes are sufficient to drain any polygon. Then, they proposed an  $O(n^2 \log n)$  algorithm to find the minimum number of holes needed to drain.

Geometric analysis has been developed to study flow of liquid in a mold as well. Bradley and Heinemann [9] proposed geometric analysis of the mold to develop shape factors indicating the effect of the geometry of the mold on fluid flow under gravity. Bose et al. [10] considered the problem of filling a mold from a purely geometric perspective in 3D space such that, when it is filled, no air pockets and ensuing surface defects arise. They proposed a linear time algorithm to check whether a given polyhedron can be completely filled without forming air pockets in a fixed orientation. They also proposed an  $O(n^2)$  algorithm to find the most favorable orientation for a given polyhedron.



**Figure 2:** We assume that we can approximate a volume of water (shown in (a) for a 2D example) by a set of water particles (shown in (b)). A water trap is a set of water particles directly or indirectly touching each other and some of which are touching the input geometry. In this example, two water traps are formed.

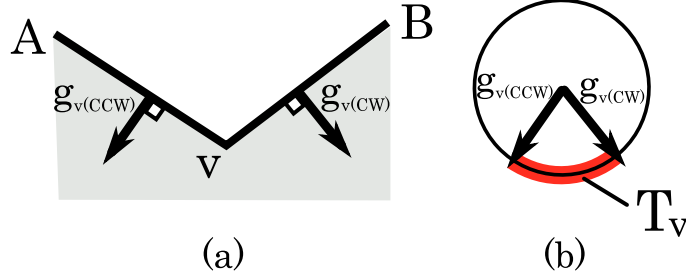
A similar problem to ours arises in planning for 4-axis NC machining, how to find a rotation axis that maximizes “visible” surface in a single setup, which was investigated by Tang et al. [11]. Generally speaking, to manufacture a desired shape, multiple setups are required; however, the setup is time-consuming and therefore the number of setups should be minimized. To consider this problem, visibility plays a vital role. Woo developed the concept of visibility maps to represent and compute accessibility [12].

### 1.2. Assumptions and a Key Observation

We assume that we can approximate a volume of water by a set of water particles whose viscosity is negligibly small. We also assume that the rotation is slow enough so that the water particles reach equilibrium for each orientation through which we rotate, and that the particles move only under the effect of gravity (assuming that any other phenomena such as friction or centrifugal force are negligible).

We define a *water trap* in a particular orientation as a connected volume of undrained water, which we approximate by a set of water particles directly or indirectly touching each other (Figure 2).

The key observation for our problem is that, for each water trap, there is always at least one concave vertex of the input mesh such that some of its incident edges and faces are touching water particles constituting the water trap. Based on this observation, our goal is to drain all the concave vertices of an input mesh since this is equivalent to draining all water traps from all voids of the workpiece.



**Figure 3:** (a): Concave vertex  $v$  (b): The diagram showing  $T_v$  of  $v$ .

In the next section, we describe an overview of our approach, going through a 2D example to introduce our directed *draining graph* method. Then, in sections 3 and 4, we describe how we actually construct and analyze the draining graph for a 3D geometry and arbitrary 3D rotation axis. In sections 5-8, we present results, complexity analysis, discussion and future work, and conclusions, respectively.

## 2. Approach and Theory

To begin, we discuss a simplified case using a 2D example.

### 2.1. Simplified case: each water trap is represented by a single water particle

First, we consider the case that each water trap consists of only one water particle. Recall that a water trap can only be formed at a concave vertex.

For each concave vertex  $v$ , we consider gravity directions such that, if a water particle is at  $v$ , it will be trapped. In the 2D case, any gravity direction can be described as a point on the *Gaussian circle* (a circle whose radius is one and center is at the origin). When we rotate a workpiece, the gravity direction moves relative to the workpiece along the Gaussian circle. For each concave vertex  $v$ , we define a space  $T_v$  on the Gaussian circle consisting of gravity directions such that, if a water particle is at  $v$ , it will be trapped. Figure 3 shows a specific example. In the 2D case, each of the two gravity directions  $g_{v(CCW)}$  and  $g_{v(CW)}$  bounding  $T_v$  are orthogonal to the two edges incident to  $v$ . We define  $g_{v(CCW)}^*$  as a point on the Gaussian circle that is not in  $T_v$  and is closest to  $g_{v(CCW)}$ . In a similar manner, we define  $g_{v(CW)}^*$  as a point on the Gaussian circle that is not in  $T_v$  and is closest to  $g_{v(CW)}$ . For a workpiece orientation with corresponding gravity direction

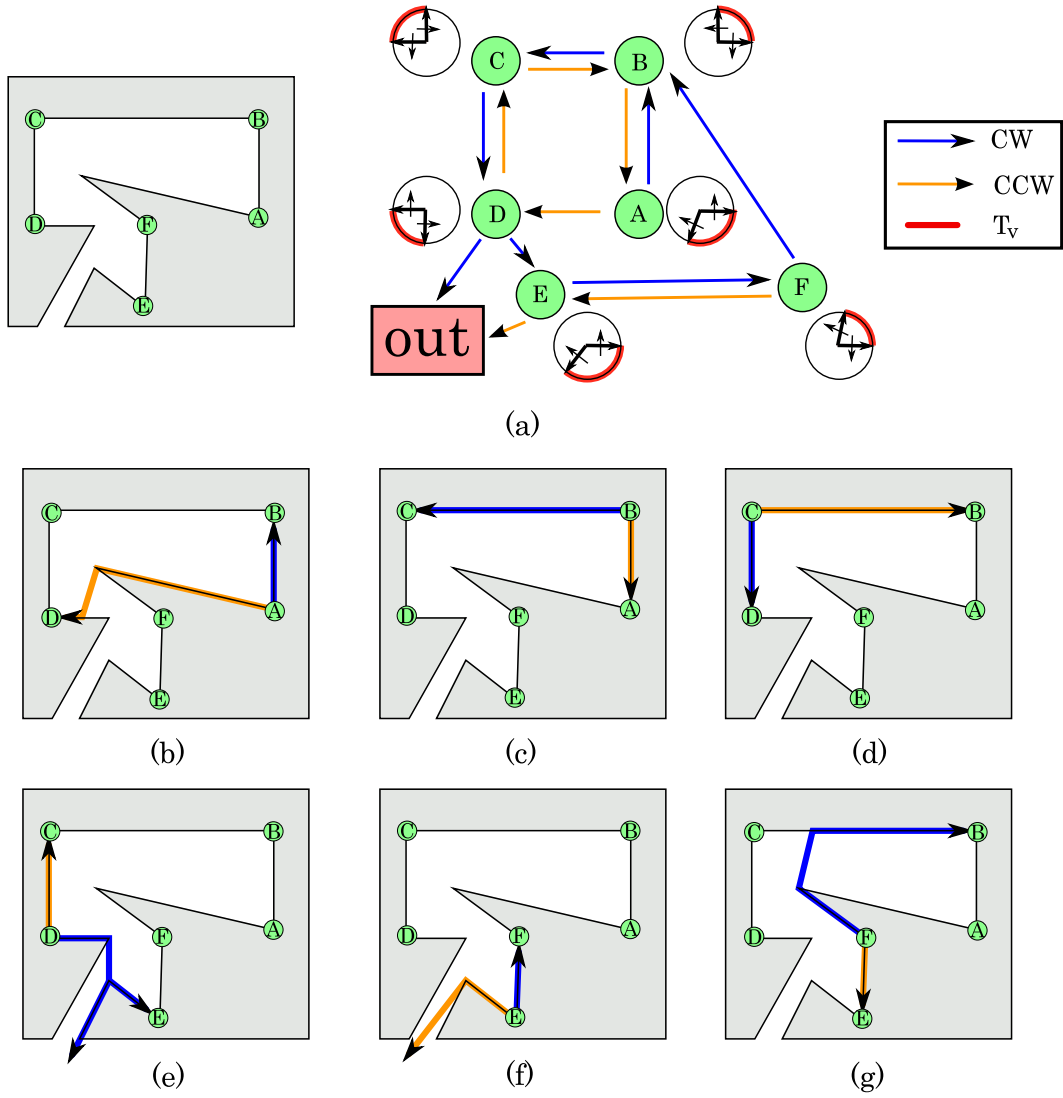
relative to the workpiece currently in  $T_v$ , when the workpiece rotates far enough that the gravity direction relative to the workpiece coincides with  $g_{v(CCW)}^*$  (respectively,  $g_{v(CW)}^*$ ), the trapped water particle leaves  $v$  and moves along the edge towards  $A$  (respectively,  $B$ ).

We construct a directed *draining graph* whose nodes correspond to the concave vertices. Each node has two kinds of outgoing edges, for clockwise and counterclockwise rotation, that point to the nodes representing the concave vertices where the trapped water will ultimately settle when the workpiece is rotated clockwise and gravity coincides with  $g_{v(CW)}^*$  or counterclockwise and gravity coincides with  $g_{v(CCW)}^*$ . If the water particle trapped at a vertex exits the workpiece once it is rotated so that gravity coincides with  $g_{v(CW)}^*$  or  $g_{v(CCW)}^*$ , the corresponding edge is set to point to a node labeled “out” representing the workpiece exterior. An example of a draining graph for a 2D geometry is shown in Figure 4.

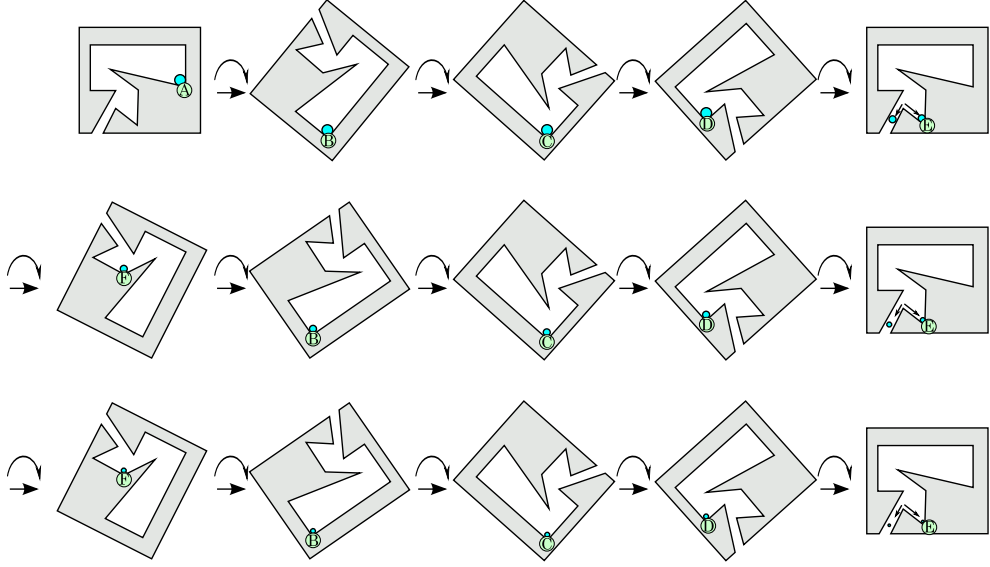
The draining graph is constructed as follows. For each concave vertex, we initialize a corresponding node in the draining graph. Then, we compute the two gravity directions when a water particle trapped at the concave vertex leaves it under clockwise and counterclockwise rotation. These gravity directions (the bounds of  $T_v$ ) are shown in a diagram next to each node in Figure 4 (a). Finally, we trace the path of a trapped water particle under both of these gravity direction to determine in what concave vertex it settles for each, adding a graph edge labeled as  $CW$  or  $CCW$  that points to the corresponding node. Figure 4 (b)-(g) show the paths a water particle takes under gravity from each concave vertex for the geometry shown in Figure 4 (a).

The destination is not necessarily unique since there may be multiple possible paths a water particle takes under gravity. Figure 4 (e) shows one such example. A water particle leaving concave vertex  $D$  with  $g_{D(CW)}^*$  may settle at concave vertex  $E$  or exit the workpiece. In this case, we assume that a particle splits into two particles.

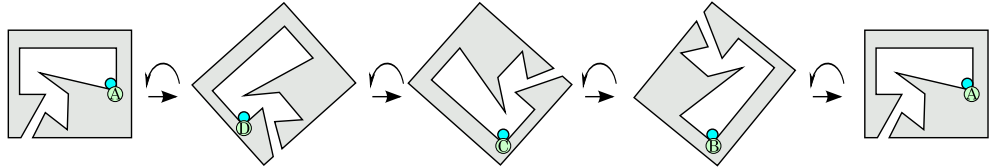
For each concave vertex, if there is some path consisting of edges with the same direction label from the corresponding node to the node labeled as “out,” then we can eventually drain the water particle trapped at that concave vertex through concave vertices corresponding to the nodes along the path by rotating in the given direction. For example, suppose a water particle is trapped at concave vertex  $A$ , there is a path “ $A \rightarrow B \rightarrow C \rightarrow D \rightarrow out$ ” in Figure 4 corresponding to the draining sequence shown in the top row of Figure 5. Since there is an edge from  $D$  not only to “out” but also to



**Figure 4:** (a) A sample geometry in 2D and the corresponding draining graph. The diagram next to each graph node shows the two gravity directions  $g_{v(CW)}^*$  and  $g_{v(CCW)}^*$  that let a water particle trapped at the corresponding concave vertex leave for the other concave vertices. For each node, when a current gravity direction is in  $T_v$ , if a water particle is at the corresponding concave vertex, it will be trapped. (b)-(g) Paths a water particle takes from each concave vertex under  $g_{v(CW)}^*$  (blue) and  $g_{v(CCW)}^*$  (orange).



**Figure 5:** (a) The transition and draining of a water particle trapped at concave vertex A by clockwise rotation. As we continue to rotate the geometry, the volume of the trapped particle gradually decreases and becomes negligibly small.



**Figure 6:** We cannot drain a water particle trapped at concave vertex A by counterclockwise rotation.

E, the water particle splits into two smaller particles and only one of them will be drained through the sequence. But as we can see in Figure 4, there is a path from  $E$  to “out” as well (“ $E \rightarrow F \rightarrow B \rightarrow C \rightarrow D \rightarrow \text{out}$ ”); therefore, the other smaller particle going to  $E$  will come back to  $D$ , split into two further smaller particles, and one of them will be drained. For each 360-degree rotation, the volume of the remaining trapped water particle decreases (Figure 5). When the volume of the remaining particle becomes negligibly small, we deem that the trapped water particle is drained. As we have seen through the example, as long as there is a path from each node



to the “out” node in the draining graph, the volume of the remaining water particles gradually decreases and will be drained eventually after rotating enough times.

On the other hand, if there are nodes that do not have a consistently labeled path to “out,” we can never drain their trapped water particles. For example, for a water particle trapped at  $A$  in the geometry shown in Figure 4, we cannot drain the water particle by counterclockwise rotation, because it just returns to  $A$  after each rotation, as shown in Figure 6. This also holds for counterclockwise rotation when a water particle is trapped at  $B$ ,  $C$ , or  $D$ . In the draining graph, there is no CCW path from the nodes corresponding to any of these vertices to “out.”

There is also a path “ $A \rightarrow D \rightarrow out$ .” But we cannot drain using this path because it corresponds to counterclockwise rotation from  $A$  to  $D$  and clockwise rotation from  $D$  to  $out$ . This violates our restriction that we can rotate around an axis in one direction only.

## 2.2. General case: each water trap is represented by a set of water particles

In the previous subsection, we took as our premise the case that each water trap consists of only one water particle, and provided the approach to solve the corresponding draining problem. We now show that if a solution exists for this case, it is also a solution for the general draining problem; that is, the case that each water trap consists of a set of water particles.

Suppose a water trap is currently formed at concave vertex  $v$ . For the general draining problem, after  $v$  is drained, not all the water particles constituting the original water trap will necessarily form a new water trap at the same concave vertex (see Figure 7); however, the key observation is that the last water particle to leave the concave vertex (we call this last particle the *core particle* of the water trap) moves in the same manner as a water particle approximating the water trap by only one particle. For example, suppose that a water trap is currently formed at a given concave vertex  $v$  as shown in Figure 7(a). Figure 7(b) shows draining when we approximate a water trap by only one particle and (c) shows draining when we approximate a water trap by a set of particles (general case). In the general case, when we start to rotate an input geometry, water particles constituting the water trap start to leave  $v$  and form another water trap at a different concave vertex (or may exit the geometry). As we continue to rotate, when one of the edges incident to  $v$  becomes perpendicular to the gravity direction (i.e. parallel to the ground), the core particle of the water trap leaves  $v$ . As shown in Figure 7 (c), at the

point when  $v$  is completely drained, water particles constituting the original water trap may constitute different water traps after a rotation; however, the core particle moves in the same manner as a water particle approximating a water trap by only one particle.

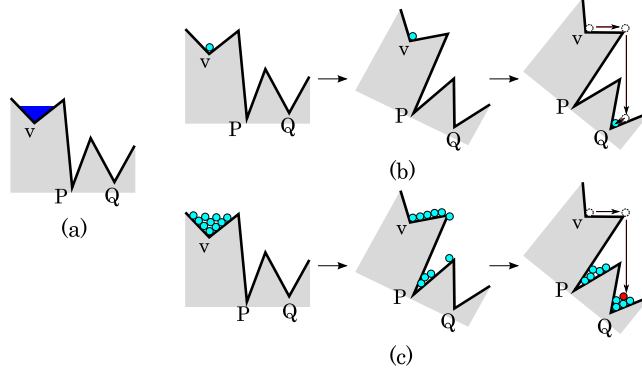
Now we show that the approach using the draining graph for the single-particle case works for general draining problems, too, assuming we rotate enough times. To see this, let us consider a simple example of a draining graph with only 4 nodes,  $A$ ,  $B$ ,  $C$ , and  $out$ , with three CW directed edges “ $C \rightarrow B \rightarrow A \rightarrow out$ ” such that a single water particle trapped at  $C$  is drained through  $B$  and  $A$  with one 360-degree rotation. Now we consider the general case for this example. For each 360-degree clockwise rotation, at least a core particle at  $A$  is drained (note that there is no guarantee that all the water particles at  $A$  exit the geometry). If there are remaining water particles (water traps), each of them exists at  $B$  or  $C$  because a water trap is always formed at a concave vertex. There is a path from  $C$  to  $B$ ; therefore, a core particle at  $C$  goes to  $B$  if a water trap is formed at  $C$ . There is a path from  $B$  to  $A$ ; therefore, a core particle at  $B$  goes to  $A$  if a water trap is formed at  $B$ . This implies that as long as there are remaining water particles, one of them must become a core particle at  $A$  and be drained in each 360-degree rotation. The number of trapped water particles never increases; therefore, as long as there are  $CW$  or  $CCW$  paths from all nodes to  $out$ , eventually all the particles will be drained. This holds no matter how complicated the draining graph becomes.

### 3. Graph Construction

In the previous section, we have shown that we can solve the general draining problem by considering the case that each water trap consists of a single water particle and the corresponding transitions using a draining graph. In this section, given a 3D geometric model and arbitrary 3D rotation axis as input, we explain how we construct the corresponding draining graph.

#### 3.1. Graph Nodes

The first step in constructing a draining graph is to determine its nodes, that is, to find the concave vertices. Although water traps in 3D (unlike in 2D) may also contain concave edges, in order for the concave edge to hold water, one of its endpoints must also be a concave vertex. Therefore, it is



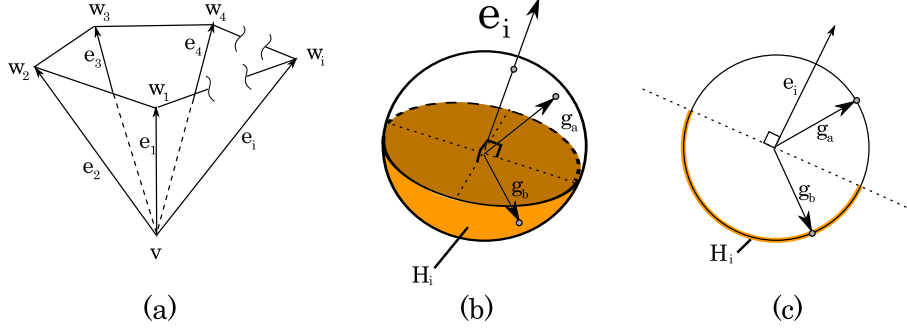
**Figure 7:** (a) A water trap is formed at concave vertex  $v$ . (b) The movement of a water particle when we approximate the water trap by only one particle. After  $v$  is drained, a new water trap is formed at concave vertex  $Q$ . (c) The movement of water particles when we approximate the water trap by a set of particles. After  $v$  is drained, new water traps are formed at  $P$  and  $Q$ . The core particle shown in red settles at a water trap at the same concave vertex where the particle in case (b) settles.

still sufficient to consider only the draining of concave vertices, since draining all concave vertices will drain all water traps.

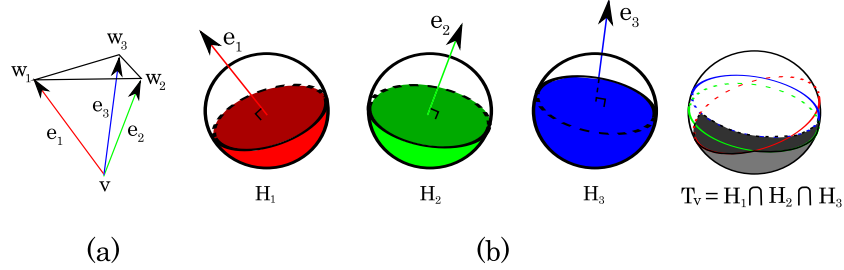
We define a concave vertex for a 3D geometry as follows. Given a vertex  $v$ , letting  $vale(v)$  be its valence, we check if there is a unit vector  $d$  such that, for all the adjacent vertices  $w_i$  of  $v$  ( $i = 1, 2, \dots, vale(v)$ ),  $(w_i - v) \cdot d < 0$  (i.e.  $v$  is a locally extreme vertex). If there is such a  $d$  and a point  $p = v + \epsilon d$  ( $\epsilon$  is a positive infinitesimal number) is inside of the given geometry,  $v$  is a concave vertex. Otherwise,  $v$  is not a concave vertex.

### 3.2. Graph Edges

Edges of a draining graph are set according to the transitions of water particles when the geometry is rotated around a given rotation axis. Let  $V_c$  be the set of concave vertices. First, for each concave vertex  $v \in V_c$ , we describe all gravity directions such that a water particle could be trapped at  $v$ . Then, we explain how to find the two gravity directions ( $g_{v(CW)}^*$  and  $g_{v(CCW)}^*$ ) at which a trapped water particle at  $v$  flows out when rotating clockwise or counterclockwise around the given axis. After finding these two gravity directions, for each of them, we find the concave vertices into which



**Figure 8:** (a) concave vertex  $v$ . (b)  $e_i$  and the corresponding  $H_i$ . Gravity direction  $g_a$  never causes a water trap at  $v$ , but  $g_b$  may cause a water trap at  $v$ . (c) The cross section of (b) including  $g_a$  and  $g_b$ .



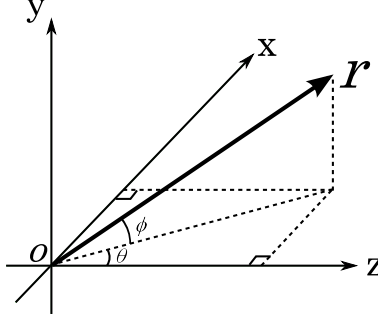
**Figure 9:** (a) concave vertex  $v$ . (b) The corresponding  $H_i$  and  $T_v$ .

water particles flowing out will settle by tracing the particle's path along geometric features (vertices, edges, and triangles).

### 3.2.1. Gravity directions causing a water trap at a concave vertex

In this subsection, we describe all gravity directions such that a water particle may be trapped at  $v$ , representing the gravity directions as points on the *Gaussian sphere* (a sphere whose radius is one and center is at the origin).

For each concave vertex  $v \in V_c$ , let  $w_i$  be a member of the set of adjacent vertices of  $v$  and let  $e_i$  be the vector from  $v$  to  $w_i$  (i.e.  $e_i = w_i - v$ ) (see Figure 8(a)). For each  $e_i$ , we define a half-space  $H_i$  of directions on the Gaussian sphere  $H_i = \{p \mid e_i \cdot p \leq 0, \|p\| = 1\}$ . Figure 8(b) shows a specific example. A gravity direction not in  $H_i$  does not cause a water trap at  $v$ . On the other hand, a gravity direction in  $H_i$  drags a water particle in the direction from  $w_i$  to  $v$  and may cause a water trap at  $v$ . For example, in Figure 8(b) and



**Figure 10:** We describe any rotation axis relative to workpiece geometry  $r = (r_x, r_y, r_z)$  by two variables  $\theta$  ( $0^\circ \leq \theta < 360^\circ$ ) and  $\phi$  ( $0^\circ \leq \phi \leq 90^\circ$ ) where  $\theta$  is the azimuthal angle in the  $xz$ -plane from the  $z$ -axis and  $\phi$  is the polar angle from the  $xz$ -plane.

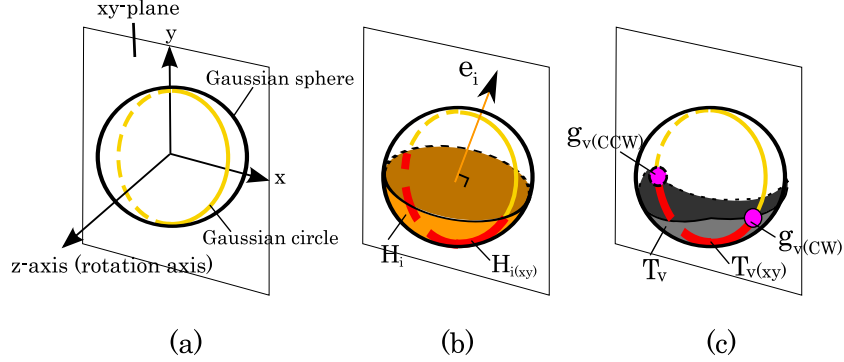
(c), the gravity direction  $g_a$  never causes a water trap at  $v$ , but  $g_b$  may cause a water trap at  $v$ .

We define the space  $T_v$  in 3D as  $T_v = \bigcap_i H_i$  (see Figure 9). Then, a gravity direction  $g$  in  $T_v$  potentially causes a water trap at  $v$ . On the other hand, since for the complement of  $T_v$ ,  $\overline{T_v}$ , we have  $\overline{T_v} = \overline{\bigcap_i H_i} = \bigcup_i \overline{H_i}$ , therefore  $g$  in at least one of  $\overline{H_i}$  does not cause a water trap at  $v$ .

To construct the draining graph, we need to determine, for each concave vertex  $v \in V_c$ , in which gravity directions the currently trapped water particle at  $v$  flows out when rotating clockwise and counterclockwise rotation around the given axis. These gravity directions correspond to points in  $\overline{T_v}$  adjacent to the boundary of  $T_v$ .

To find these gravity directions, given a rotation axis, we always choose the coordinate system such that the rotation axis coincides with the  $z$ -axis. Then, possible gravity directions are confined in the  $xy$ -plane because a gravity direction and the rotation axis are always orthogonal. In this configuration, any gravity direction  $g$  can be expressed as a point on a unit circle in the  $xy$ -plane with center  $(0,0)$  (i.e.  $x^2 + y^2 = 1$ ). This  $xy$ -plane Gaussian circle is the intersection between the Gaussian sphere and the  $xy$ -plane.

We can describe any rotation axis relative to workpiece geometry  $r = (r_x, r_y, r_z)$  by two variables  $\theta$  ( $0^\circ \leq \theta < 360^\circ$ ) and  $\phi$  ( $0^\circ \leq \phi \leq 90^\circ$ ) where  $\theta$  is the azimuthal angle in the  $xz$ -plane from the  $z$ -axis and  $\phi$  is the polar angle from the  $xz$ -plane as shown in Figure 10. Using these parameters, the components of  $r$  can be expressed as  $r_x = \cos \phi \sin \theta$ ,  $r_y = \sin \phi$ , and



**Figure 11:** (a) The relationship between the Gaussian sphere and the xy-plane Gaussian circle. (b) The relationship between  $H_i$  and  $H_{i(xy)}$ . (c) The relationship between  $T_v$  and  $T_{v(xy)}$ .

$r_z = \cos \phi \cos \theta$ . Then, by multiplying each vertex by the matrix  $R$  where

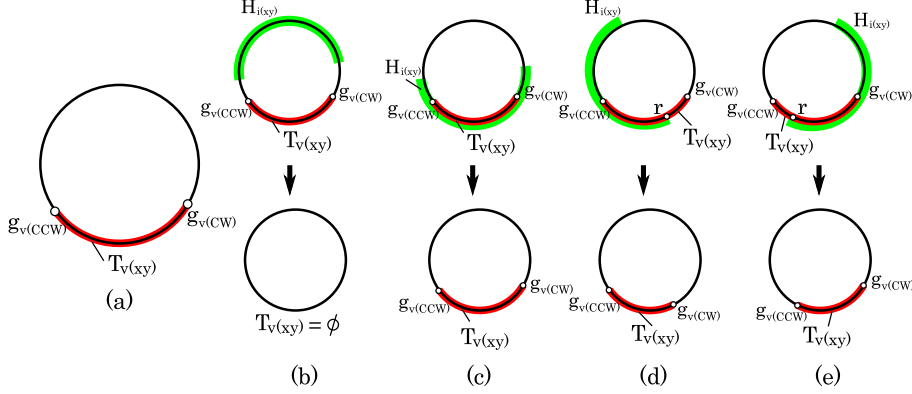
$$R = \begin{bmatrix} \cos \theta & 0 & -\sin \theta \\ -\sin \theta \sin \phi & \cos \phi & -\cos \theta \sin \phi \\ \sin \theta \cos \phi & \sin \phi & \cos \theta \cos \phi \end{bmatrix},$$

we can set the coordinate system such that a given rotation axis coincides with the z-axis.

Rotating the input geometry around the rotation axis is equivalent to fixing the geometry and moving the gravity direction on the xy-plane Gaussian circle. Given  $v \in V_c$ , suppose that a gravity direction  $g$  is currently in  $T_v$  and a water particle is trapped at  $v$ . As we move  $g$  on the Gaussian circle, when  $g$  passes through the boundary of  $T_v$ , the trapped water particle at  $v$  flows out. If  $T_v$  does not intersect with the xy-plane, water is never trapped at  $v$  with the given rotation axis.

As shown in Figure 9(b),  $T_v$  is bounded by a set of great circular arcs on the Gaussian sphere. If  $T_v$  is intersected by the xy-plane, it intersects its boundary at two points because  $T_v$  is convex. Since  $T_v = \bigcap_i H_i$ , each of the arcs is defined by the boundary of an  $H_i$ .

For the actual calculation to find the gravity directions where trapped water flows out, we do not have to construct  $T_v$  in its entirety since the gravity directions are confined in the xy-plane; constructing the portion of  $T_v$  intersecting with the xy-plane is sufficient. Call this portion of  $T_v$  defined on the



**Figure 12:** (a)  $g_v(CW)$ ,  $g_v(CCW)$ , and  $T_v(xy)$  on the Gaussian circle. Four cases of updating  $g_v(CW)$ ,  $g_v(CCW)$ , and  $T_v(xy)$  when a new  $H_{i(xy)}$  is introduced are shown in (b)-(e).

xy-plane Gaussian circle  $T_v(xy)$  (see Figure 11). Each of the boundary points of  $T_v(xy)$  is defined by the intersection between the xy-plane Gaussian circle and the boundary of one of the  $H_i$  because  $T_v(xy) = \bigcap_i H_{i(xy)}$ , where  $H_{i(xy)}$  is the intersection between  $H_i$  and the xy-plane<sup>1</sup>. Appendix A describes how to compute the boundary of  $H_{i(xy)}$ .

As shown in Figure 11(c) and Figure 12(a), we let  $g_v(CW)$  be the point on the Gaussian circle bounding  $T_v(xy)$  rotating clockwise (when seen from  $+\infty$  on the z-axis – the rotation axis) and  $g_v(CCW)$  the point on the Gaussian circle bounding  $T_v(xy)$  rotating counterclockwise. We compute the boundary points of  $T_v(xy)$ , that is,  $g_v(CW)$  and  $g_v(CCW)$ , incrementally as follows.

Initially,  $g_v(CW)$  and  $g_v(CCW)$  are set to the two corresponding boundary points of  $H_{1(xy)}$ . Then, for each  $i$  ( $i = 2, 3, \dots, \text{vale}(v)$ ), if necessary we update  $g_v(CW)$  and  $g_v(CCW)$ , that is, the boundaries of  $T_v(xy)$  as follows. For each  $i$ , if neither of the  $g_v(CW)$  or  $g_v(CCW)$  calculated thus far are in  $H_{i(xy)}$ ,  $T_v(xy)$  is empty (Figure 12 (b)). On the other hand, if both  $g_v(CW)$  and  $g_v(CCW)$  are in  $H_{i(xy)}$ , we do not have to update  $T_v(xy)$  (Figure 12 (c)). When one of  $g_v(CW)$  and  $g_v(CCW)$  is not in  $H_{i(xy)}$ , one of the boundary points of  $H_{i(xy)}$  is in  $T_v(xy)$  (let this be  $r$ ). If  $g_v(CW)$  is not in  $H_{i(xy)}$ , we set  $g_v(CW)$  to  $r$  (Figure 12 (d)). If  $g_v(CCW)$  is not in  $H_{i(xy)}$ , we set  $g_v(CCW)$  to  $r$  (Figure 12

<sup>1</sup> $T_v(xy) = T_v \cap xy = (\bigcap_i H_i) \cap xy = \bigcap_i (H_i \cap xy) = \bigcap_i H_{i(xy)}$   
where  $xy$  is the xy-plane.

(e)). After performing this update for each  $e_i$  ( $i = 2, 3, \dots, \text{vale}(v)$ ),  $g_{v(CW)}$  and  $g_{v(CCW)}$  will be the points bounding  $T_{v(xy)}$ .

We define  $g_{v(CW)}^*$  as the point on the xy-plane Gaussian circle that is not in  $T_{v(xy)}$  and is closest to  $g_{v(CW)}$ . In a similar manner, we define  $g_{v(CCW)}^*$  as the point on the xy-plane Gaussian circle that is not in  $T_{v(xy)}$  and is closest to  $g_{v(CCW)}$ . Thus  $g_{v(CW)}^*$  and  $g_{v(CCW)}^*$  are gravity directions at which a trapped water particle at  $v$  flows out when rotating clockwise or counterclockwise around a given rotation axis.

Letting  $g_{v(CW)} = ((g_{v(CW)})_x, (g_{v(CW)})_y, 0)$  and  $g_{v(CCW)} = ((g_{v(CCW)})_x, (g_{v(CCW)})_y, 0)$ , we can describe  $g_{v(CW)}^*$  and  $g_{v(CCW)}^*$  as

$$\begin{aligned} g_{v(CW)}^* &= \begin{pmatrix} (g_{v(CW)})_x \cos \epsilon - (g_{v(CW)})_y \sin \epsilon \\ (g_{v(CW)})_x \sin \epsilon + (g_{v(CW)})_y \cos \epsilon \\ 0 \end{pmatrix} \\ g_{v(CCW)}^* &= \begin{pmatrix} (g_{v(CCW)})_x \cos \epsilon + (g_{v(CCW)})_y \sin \epsilon \\ -(g_{v(CCW)})_x \sin \epsilon + (g_{v(CCW)})_y \cos \epsilon \\ 0 \end{pmatrix} \end{aligned}$$

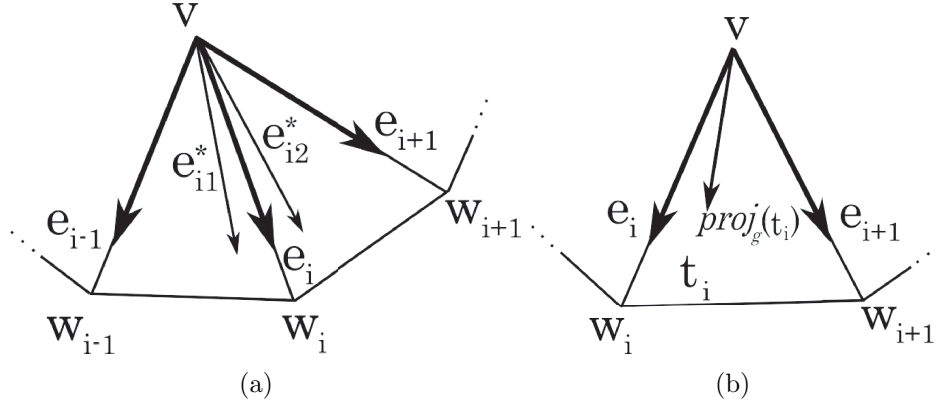
where  $\epsilon$  is a positive infinitesimal number representing the infinitesimal rotation.

### 3.2.2. Concave vertex where trapped water flowing out settles

In the previous subsection, for each concave vertex  $v \in V_c$ , we showed how we compute the two gravity directions when the trapped water particle at  $v$  flows out under rotation around the rotation axis. Now, we describe how to determine which concave vertex the water particle flowing out from  $v$  settles in (or if it exits the geometry).

We will use the following notation. Given a vertex  $v$ , let  $w_i$  be a member of the set of adjacent vertices of  $v$  and  $e_i$  be the normalized vector from  $v$  to  $w_i$ , that is,  $e_i = (w_i - v) / \|w_i - v\|$  ( $i = 1, 2, \dots, \text{vale}(v)$ ). We call edge  $e_i$  the *locally-steepest* edge from  $v$  with respect to unit gravity vector  $g$  if  $e_i \cdot g > 0$  and  $e_i$  is “steeper” with respect to gravity  $g$  than any vectors from  $v$  on  $e_i$ ’s adjacent triangles. An edge is steeper if  $e_i \cdot g > e_{i1}^* \cdot g$  and  $e_i \cdot g > e_{i2}^* \cdot g$ , where  $e_{i1}^* = (\epsilon e_{i-1} + (1 - \epsilon)e_i) / \|\epsilon e_{i-1} + (1 - \epsilon)e_i\|$  and  $e_{i2}^* = (\epsilon e_{i+1} + (1 - \epsilon)e_i) / \|\epsilon e_{i+1} + (1 - \epsilon)e_i\|$  ( $\epsilon$  is a positive infinitesimal number) (Figure 13(a)). Let  $t_i$  be the triangle incident to  $v$  defined by vertices  $v$ ,  $w_i$ , and  $w_{i+1}$ . We define  $\text{proj}_g(t_i)$  as the projection of gravity vector  $g$  onto the plane of triangle  $t_i$ ; thus, for triangle  $t_i$ ’s normal vector  $n_{t_i}$ , we can calculate

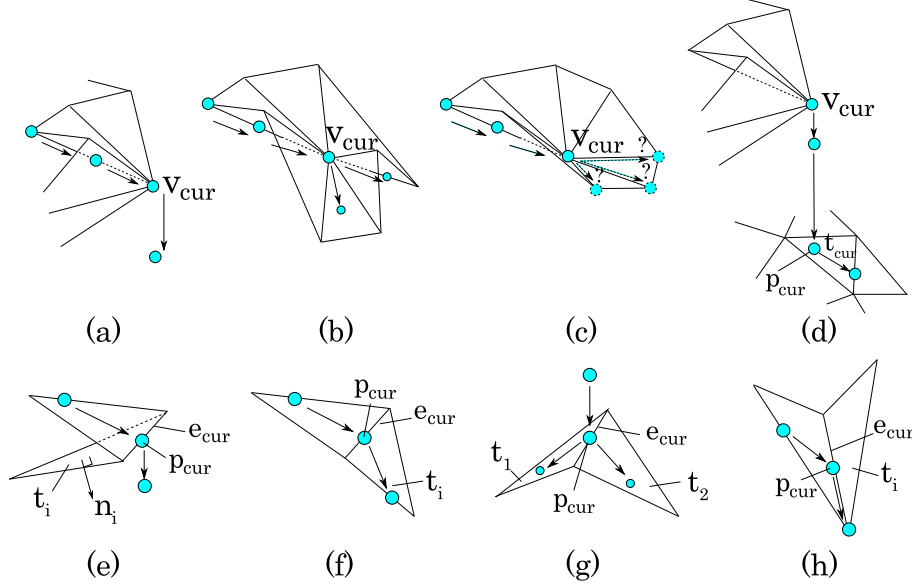




**Figure 13:** (a) Edge  $e_i$  is a locally-steepest edge if  $e_i \cdot g > 0$  and  $e_i$  is steeper with respect to gravity  $g$  than any vectors on  $e_i$ 's adjacent triangles, e.g.  $e_{i1}^*$  and  $e_{i2}^*$ . (b) Triangle  $t_i$  is a locally-steepest triangle with respect to gravity  $g$  if  $n_{t_i} \cdot g < 0$  and  $proj_g(t_i)$  lies on triangle  $t_i$ .

$proj_g(t_i) = (I - n_{t_i} n_{t_i}^T)g$ . We call triangle  $t_i$  the *locally-steepest* triangle with respect to gravity  $g$  if  $n_{t_i} \cdot g < 0$  and  $proj_g(t_i)$  lies on triangle  $t_i$ . That is, there exist scalar values  $\alpha$  and  $\beta$  that satisfy  $v + proj_g(t_i)\alpha = w_i(1 - \beta) + w_{i+1}\beta$ ,  $\alpha > 0$ , and  $0 < \beta < 1$ . (Note that there may be more than one locally-steepest edge or triangle for a given vertex.)

We only describe the case when we rotate the geometry clockwise because the same procedure works for the counterclockwise case. Let the set of concave vertices where the water particle leaving  $v$  settles when we trace the particle with  $g_{v(CW)}^*$  be  $S_{v(CW)}$  and when we trace the particle with  $g_{v(CCW)}^*$  be  $S_{v(CCW)}$ . For the sake of simplicity of notation, we let  $S_v \stackrel{def}{=} S_{v(CW)}$  and  $g \stackrel{def}{=} g_{v(CW)}^*$  for this explanation. The three cases of a particle leaving a vertex  $v_{cur}$  and falling through space, traveling along an edge, or traveling along the face of a triangle are handled by Procedure 1: **TraceFromVertex** (Figure 14 (a)(b)(c)). The three cases for a particle leaving a location  $p_{cur}$  in the middle of an edge  $e_{cur}$  and falling through space, traveling along the face of a triangle, or traveling along an edge are handled by Procedure 2: **TraceFromEdge** (Figure 14 (e)(f)(g)(h)). Procedure 3: **ParticleDrop**, Procedure 4: **FindNextEdge**, and Procedure 5: **TraceOnFlatRegion** handle the transitions between these states. We outline the logic below; the corresponding detailed



**Figure 14:** Transition cases of a water particle under gravity on various geometric shapes. (a)(b)(c) Possible movements from a vertex. (d) Movement when a particle drops vertically. (e)(f)(g)(h) Possible movements from an edge.

pseudocode for each subroutine is shown in Algorithms 1, 2, 3, 4, and 5.

We find  $S_v$  for  $v$  under gravity force  $g$  by tracking the particle location starting with Procedure 1: TraceFromVertex, initially setting  $v_{cur}$  to  $v$ .

#### 1. TraceFromVertex:

- If a point  $v_{cur} + \epsilon g$  ( $\epsilon$  is a positive infinitesimal number) is outside of the geometry, the water particle falls down parallel to  $g$  from  $v_{cur}$  (Figure 14 (a)). To simulate this, we shoot a half-ray  $v_{cur} + \gamma g$  (where  $\gamma$  is a positive scalar). **Go to 3.**
- Otherwise, we define  $m = \arg \max_i (e_i \cdot g)$  (i.e.  $\forall i, e_m \cdot g \geq e_i \cdot g$ ).
  - if  $e_m \cdot g < 0$ , the water particle flowing out from  $v$  settles at  $v_{cur}$ . We add  $v_{cur}$  to  $S_v$ .
  - if  $e_m \cdot g > 0$ , we let the set of locally-steepest edges of  $v_{cur}$  be  $E_s$  and the set of locally-steepest triangles of  $v_{cur}$  be  $T_s$ . For each edge  $e_j$  in  $E_s$  ( $j = 1, \dots, |E_s|$ ), we set  $v_{cur}$  to  $w_j$  (the endpoint of  $e_j$  that is not  $v$ ) and **Go to 1** for each  $e_j$ . For each

triangle  $t_j$  in  $T_s$  ( $j = 1, \dots, |T_s|$ ), we solve  $v_{cur} + \text{proj}_g(t_j)\alpha = w_j(1 - \beta) + w_{j+1}\beta$ . We set  $e_{cur}$  to the edge  $\overline{w_j w_{j+1}}$  and  $p_{cur}$  to point  $w_j(1 - \beta) + w_{j+1}\beta$ . **Go to 2** for each  $t_j$ . (Figure 14 (b).)

- otherwise ( $e_m \cdot g = 0$ ); we cannot decide whether the water particle settles at  $v_{cur}$  or moves to another point by looking only at local information at  $v_{cur}$  (Figure 14 (c)). We set  $p_{cur}$  to  $v_{cur}$ . **Go to 5**.

2. **TraceFromEdge:** Let the two triangles adjacent to  $e_{cur}$  be  $t_1$  and  $t_2$  with normals  $n_1$  and  $n_2$  respectively.

- If  $e_{cur}$  is a ridge edge and  $n_1 \cdot g \geq 0$  or  $n_2 \cdot g \geq 0$ , the water particle falls down; we shoot a half-ray  $p_{cur} + \gamma g$  (where  $\gamma$  is a positive scalar). **Go to 3**. (Figure 14 (e).)
- Otherwise, if  $t_1$  is not perpendicular to  $g$ , we set  $t_{cur}$  to  $t_1$  and **Go to 4**; then, if  $t_2$  is not perpendicular to  $g$ , we set  $t_{cur}$  to  $t_2$  and **Go to 4**. (Figure 14 (f)(g).)

If the particle does not move along either  $t_1$  or  $t_2$ , the particle goes along  $e_{cur}$  (Figure 14 (h)). Letting the two endpoints of  $e_{cur}$  be  $v_a$  and  $v_b$ ,

- if  $v_a \cdot g > v_b \cdot g$ , we set  $v_{cur}$  to  $v_a$ . **Go to 1**.
- if  $v_a \cdot g < v_b \cdot g$ , we set  $v_{cur}$  to  $v_b$ . **Go to 1**.
- otherwise ( $v_a \cdot g = v_b \cdot g$ ), **Go to 5**.

3. **ParticleDrop:**

- If the ray does not hit any part of the input geometry, the water particle exits the geometry; we add “out” to  $S_v$ .
- Otherwise,
  - if the ray hits a vertex, we set  $v_{cur}$  to the vertex. **Go to 1**.
  - else if the ray hits an edge, we set  $e_{cur}$  to the edge and  $p_{cur}$  to the point the ray hits. **Go to 2**.
  - else if the ray hits a triangle, we set  $t_{cur}$  to the triangle and  $p_{cur}$  to the point the ray hits. If  $t_{cur}$  is not perpendicular to  $g$ , **Go to 4** (Figure 14 (d)). Otherwise, **Go to 5**.

4. **FindNextEdge:** We find the edge of  $t_{cur}$  such that, letting the two endpoints of the edge be  $v_a$  and  $v_b$ , there exist scalar values  $\alpha$  and  $\beta$  that satisfy  $p_{cur} + \text{proj}_g(t_{cur})\alpha = v_a(1 - \beta) + v_b\beta$ ,  $\alpha > 0$ , and  $0 \leq \beta \leq 1$ .

- If we find such an edge, then
  - when  $\beta = 0$ , set  $v_{cur}$  to  $v_a$ . **Go to 1.**
  - when  $\beta = 1$ , set  $v_{cur}$  to  $v_b$ . **Go to 1.**
  - when  $0 < \beta < 1$ , set  $e_{cur}$  to this intersecting edge and set  $p_{cur}$  to  $v_a(1 - \beta) + v_b\beta$ . **Go to 2.**
- Otherwise, the particle does not move along  $t_{cur}$  with  $g$ .

5. **TraceOnFlatRegion:** On a horizontal region (perpendicular to  $g$ ), a particle does not move via gravity; therefore, we assume that particles diffuse concentrically and flow out through the closest point from  $p_{cur}$  that can be reached along edges and triangles perpendicular to  $g$ . We call such a closest point  $p_f$ .

We define  $E_{perp}$  as the set of edges that are perpendicular to  $g$  and can be reached from  $p_{cur}$  only traversing edges and triangles perpendicular to  $g$ . We also define  $T_{perp}$  as the set of triangles perpendicular to  $g$  and incident to edges in  $E_{perp}$ . We define  $V_{cand}$  and  $E_{cand}$  as the vertices and edges where a water particle leaving  $p_{cur}$  may flow out through:  $V_{cand}$  consists of vertices each of which is an endpoint of  $E_{perp}$  and has an incident edge  $e_i$  such that  $e_i \cdot g > 0$ ;  $E_{cand}$  consists of ridge edges in  $E_{perp}$ .

- If  $V_{cand}$  and  $E_{cand}$  are empty, the particle is trapped at this flat region. We add all the concave vertices incident to the edges in  $E_{perp}$  to  $S_v$ .
- Otherwise, we find  $p_f$  (Appendix B describes how to find  $p_f$ , given  $p_{cur}$  and  $E_{perp}$ ,  $T_{perp}$ ,  $V_{cand}$ , and  $E_{cand}$ ).
  - If  $p_f$  is on a vertex in  $V_{cand}$ , set  $v_{cur}$  to  $p_f$ . **Go to 1.**
  - If  $p_f$  is on an edge in  $E_{cand}$ , set  $p_{cur}$  to  $p_f$  and  $t_{cur}$  to the triangle incident to the edge and not in  $T_{perp}$ . **Go to 3.**

We repeat this procedure for each  $v$  until we find all the possible concave vertices (possibly plus “out”) that should be added to each  $S_v$ . For each concave vertex  $v \in V_c$ , we connect the corresponding node to the nodes corresponding to the elements in  $S_{v(CW)}$  by an edge labeled  $CW$  and to the nodes corresponding to the elements in  $S_{v(CCW)}$  by an edge labeled  $CCW$ .

Note that the ray tracing performance in Procedure 3 will be very expensive for large inputs unless we use a bounding volume hierarchy (BVH) [13] to limit the number of triangles tested. We used a kd-tree [14] for the BVH in our implementation.

---

**Algorithm 1** TraceFromVertex

---

*Input:* vertex  $v_{cur}$   
**if**  $v_{cur} + \epsilon g$  is outside of the geometry **then**  
    ParticleDrop( $v_{cur} + \gamma g$ )  
**else**  
     $m \leftarrow \arg \max_i (e_i \cdot g)$   
    **if**  $e_m \cdot g < 0$  **then**  
        // water particle settles at  $v_{cur}$   
         $S_v \leftarrow S_v \cup v_{cur}$   
    **else if**  $e_m \cdot g > 0$  **then**  
        **for**  $j = 1$  to  $|E_s|$  **do**  
            TraceFromVertex( $w_j$ )  
        **end for**  
        **for**  $j = 1$  to  $|T_s|$  **do**  
            Solve for  $\alpha$  and  $\beta$  s.t.  $v_{cur} + \text{proj}_g(t_j)\alpha = w_j(1 - \beta) + w_{j+1}\beta$   
            TraceFromEdge( $\overline{w_j w_{j+1}}$ ,  $w_j(1 - \beta) + w_{j+1}\beta$ )  
        **end for**  
    **else**  
        //  $e_m \cdot g = 0$   
        TraceOnFlatRegion( $v_{cur}$ )  
    **end if**  
**end if**

---

#### 4. Checking Drainability

Now, using the draining graph constructed, we test whether or not a rotation around a given rotation axis can completely drain trapped water. For each concave vertex  $v \in V_c$ , if there is a path from the corresponding node to the *out* node in the draining graph, we can drain water trapped at  $v$  by rotating the input geometry around this rotation axis. Note that when we rotate the geometry clockwise, we can use only edges labeled *CW*, and when we rotate counterclockwise, we can use only edges labeled *CCW*.

##### 4.1. Checking Procedure

Letting the number of concave vertices be  $n = |V_c|$ , if we take a naive approach, we may have to trace  $n$  nodes from each concave vertex  $v \in V_c$  in the worst case. Therefore, the total running time becomes  $O(n^2)$ . However, we observe that if there is a path from one node to the *out* node, it means

---

**Algorithm 2** TraceFromEdge

---

*Input: current edge  $e_{cur}$ , current point  $p_{cur}$*   
**if**  $e_{cur}$  is a ridge edge, and  $n_1 \cdot g \geq 0$  or  $n_2 \cdot g \geq 0$  **then**  
    ParticleDrop( $p_{cur} + \gamma g$ )  
**else**  
     $found1 \leftarrow \text{FindNextEdge}(t_1, p_{cur})$   
     $found2 \leftarrow \text{FindNextEdge}(t_2, p_{cur})$   
    **if**  $found1 = false$  and  $found2 = false$  **then**  
        **if**  $v_a \cdot g > v_b \cdot g$  **then**  
            TraceFromVertex( $v_a$ )  
        **else if**  $v_a \cdot g < v_b \cdot g$  **then**  
            TraceFromVertex( $v_b$ )  
        **else**  
            //  $v_a \cdot g = v_b \cdot g$   
            TraceOnFlatRegion( $p_{cur}$ )  
        **end if**  
    **end if**  
**end if**

---

---

**Algorithm 3** ParticleDrop

---

*Input: half-ray  $h\_Ray$*   
**if**  $h\_Ray$  does not hit any part of the input geometry **then**  
     $S_v \leftarrow S_v \cup out$   
**else if** half-ray  $h\_Ray$  hits a vertex  $v_{hit}$  **then**  
    TraceFromVertex( $v_{hit}$ )  
**else if** half-ray  $h\_Ray$  hits an edge  $e_{hit}$  **then**  
     $p_{hit} \leftarrow$  point where  $h\_Ray$  hits  $e_{hit}$   
    TraceFromEdge( $e_{hit}, p_{hit}$ )  
**else if** half-ray  $h\_Ray$  hits a triangle  $t_{hit}$  **then**  
     $p_{hit} \leftarrow$  point where  $h\_Ray$  intersects  $t_{hit}$   
    **if**  $t_{hit}$  is not perpendicular to  $g$  **then**  
        FindNextEdge( $t_{hit}, p_{hit}$ )  
    **else**  
        TraceOnFlatRegion( $t_{hit}$ )  
    **end if**  
**end if**

---

---

**Algorithm 4** FindNextEdge

---

*Input: triangle  $t_{cur}$ , current point  $p_{cur}$*   
**for all** three edges  $e_i$  of  $t_{cur}$  ( $i = 1, 2, 3$ ) **do**  
     $v_a \leftarrow$  one endpoint of  $e_i$   
     $v_b \leftarrow$  other endpoint of  $e_i$   
    Solve for  $\alpha$  and  $\beta$  s.t.  $p_{cur} + proj_g(t_{cur})\alpha = v_a(1 - \beta) + v_b\beta$   
    **if**  $\alpha > 0$  and  $0 \leq \beta \leq 1$  **then**  
        **if**  $\beta = 0$  **then**  
            TraceFromVertex( $v_a$ )  
        **else if**  $\beta = 1$  **then**  
            TraceFromVertex( $v_b$ )  
        **else if**  $0 < \beta < 1$  **then**  
            TraceFromEdge( $e_i, v_a(1 - \beta) + v_b\beta$ )  
        **end if**  
        **return** *true*  
    **end if**  
**end for**  
// particle does not move along  $t_{cur}$   
**return** *false*

---

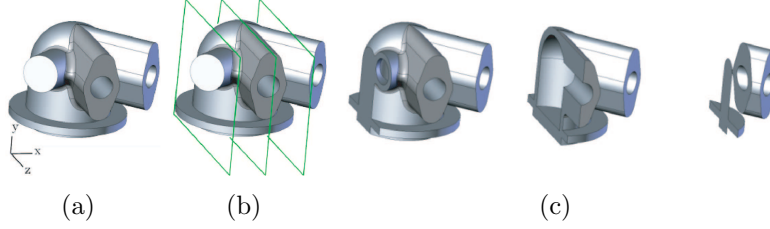
---

**Algorithm 5** TraceOnFlatRegion

---

*Input: current point  $p_{cur}$*   
**if**  $V_{cand}$  and  $E_{cand}$  are empty **then**  
    // flat region is a water trap  
     $S_v \leftarrow S_v \cup$  (all the concave vertices incident to edges in  $E_{perp}$ )  
**else**  
    Find  $p_f$  from  $p_{cur}$ ,  $E_{perp}$ ,  $T_{perp}$ ,  $V_{cand}$ , and  $E_{cand}$  // (see Appendix B)  
    **if**  $p_f$  is on a vertex in  $V_{cand}$  **then**  
        TraceFromVertex( $p_f$ )  
    **else**  
        //  $p_f$  is on an edge in  $E_{cand}$   
         $t_{cur} \leftarrow$  triangle incident to the edge and not in  $T_{perp}$   
        FindNextEdge( $t_{cur}, p_f$ )  
    **end if**  
**end if**

---



**Figure 15:** *We applied our theory to a simple mechanical workpiece shown in (a). The three sections indicated in (b) are shown in (c).*

that there is also a path from the intermediate nodes on this path to the *out* node (because draining is transitive). For example, in Figure 4, if we find a path from A through B, C, and D to *out*, we know that there is also a path from B, through C and D to *out*, and so on.

Based on this observation, we can improve the running time through the following procedure. Suppose we rotate the geometry in a clockwise direction. Then, trapped water particles at the concave vertices whose corresponding nodes are directly connected to the *out* node by the edges labeled as *CW* can be drained. Let the set of these nodes be  $N_d$ . Next, consider trapped water particles at concave vertices whose corresponding nodes are directly connected to the nodes in  $N_d$  by edges labeled as *CW*; these can be drained as well. We add these nodes to  $N_d$ , and continue recursively. This recursion stops when all the nodes connecting to at least one of the nodes in  $N_d$  by the edges labeled as *CW* are in  $N_d$ . Then, after the recursion stops, if  $|N_d| = n$ , we can guarantee that trapped water particles at all of the concave vertices are completely drained by rotation around the given rotation axis. Through this approach, we do not have to check the same node more than once. Therefore, the time complexity becomes  $O(n)$ .

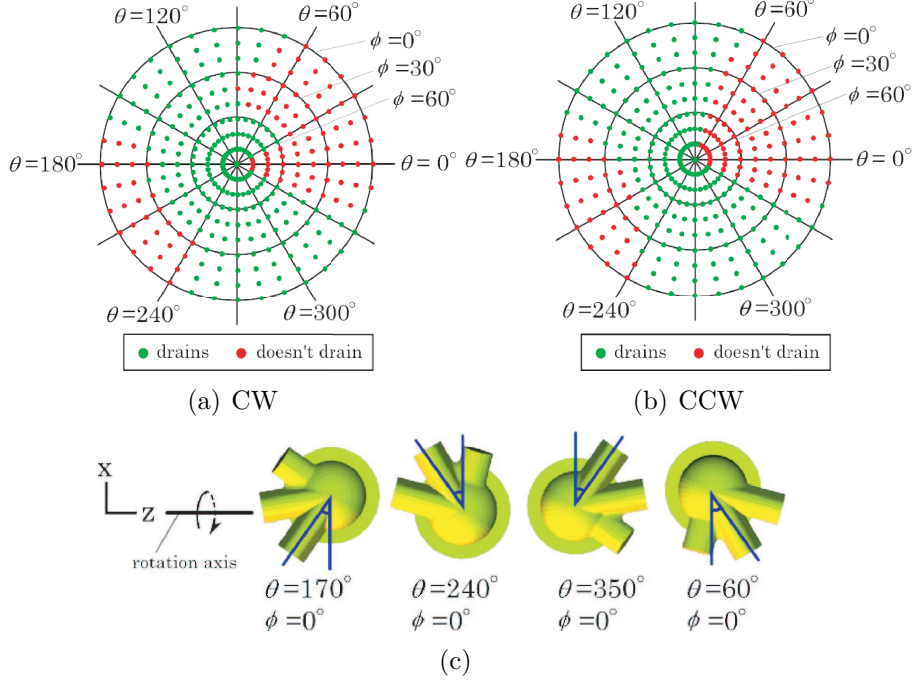
## 5. Results

We first visualize the analysis output for two sample parts, one simple and one complex, and then discuss the performance.

### 5.1. Output

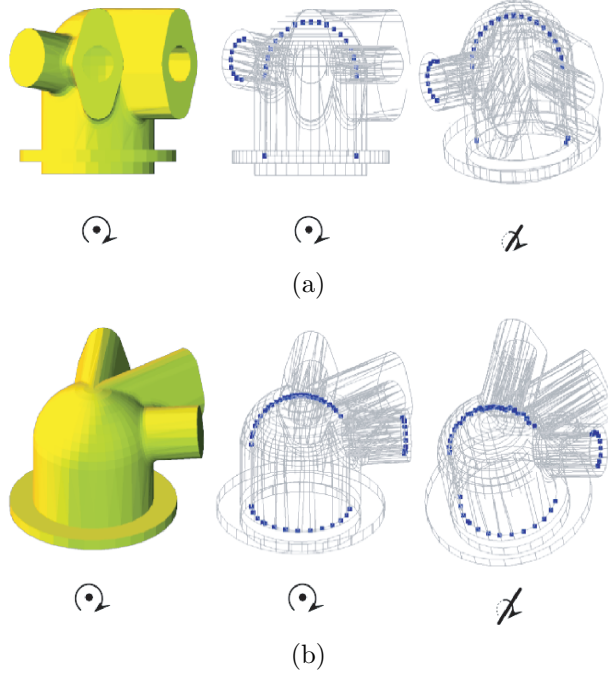
We first show our output graphically for the simple mechanical part shown in Figure 15(a). Figure 15(b) indicates the locations of the three cross-





**Figure 16:** Whether or not a rotation around a given axis  $(\theta, \phi)$  completely drains the workpiece (a) under CW and (b) CCW rotation. (c) The configurations of the workpiece when  $\phi = 0$  and  $\theta = 170^\circ, 240^\circ, 350^\circ$ , and  $60^\circ$  that are the limits in the  $\phi = 0$  plane of whether the rotation axis drains the workpiece or not. We can see that the angle between the x-axis and the outlet closer to the x-axis is the same for all four cases.

sections shown in Figure 15(c), revealing an inside void of the workpiece where water can be trapped. Figure 16 (a) and (b) plot whether or not a rotation around a given axis  $(\theta, \phi)$  completely drains the workpiece under CW and CCW rotation respectively. To verify these results, we show some representative configurations in Figure 16(c) for the CW case. All these configurations are when  $\phi = 0$  and viewed from  $+\infty$  on the y-axis. If we fix  $\phi = 0$ , when  $170^\circ \leq \theta \leq 240^\circ$  and  $350^\circ \leq \theta \leq 420^\circ (= 60^\circ)$ , we cannot drain the workpiece as shown in Figure 16(a). The four configurations shown in Figure 16(c) are set at these four limits. Notice that the angle between the x-axis and the outlet closer to the x-axis is the same for all four cases; this angle is a threshold for whether or not a given rotation axis works for

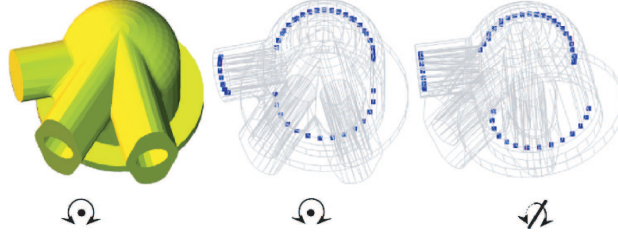


**Figure 17:** *Some representative results. (a)  $\theta = 0^\circ$ ,  $\phi = 0^\circ$ . (b)  $\theta = 230^\circ$ ,  $\phi = 30^\circ$ . For both (a) and (b), the rotation axis is set perpendicular to the paper for the left and center figures. The right figure is a view from a different angle. For the center and right figures, the vertices shown in blue are concave vertices such that once a water particle is trapped there, it will never exit the workpiece when we rotate it around the corresponding rotation axis.*

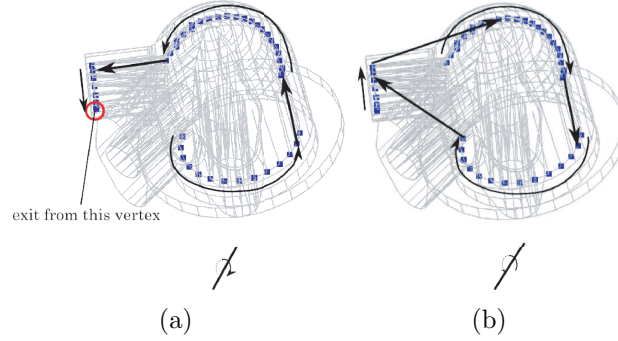
draining. This shows that our algorithm can capture this threshold.

Figure 17 shows representative results of two additional CW cases ((a)  $\theta = 0^\circ$ ,  $\phi = 0^\circ$ , (b)  $\theta = 230^\circ$ ,  $\phi = 30^\circ$ ). For the center and right figures, the vertices shown in blue are concave vertices where a water trap is potentially formed when we rotate the workpiece around the corresponding rotation axis; we cannot drain the workpiece. Figure 18 shows a result when  $\theta = 30^\circ$  and  $\phi = 60^\circ$ . This is an example where CW rotation works but CCW rotation does not work (see Figure 16 (a) and (b)). Figure 19 shows the corresponding transition of a water particle when we rotate (a) clockwise and (b) counterclockwise around this rotation axis.

To get a sense of how sensitive our algorithm is to the coarseness of the



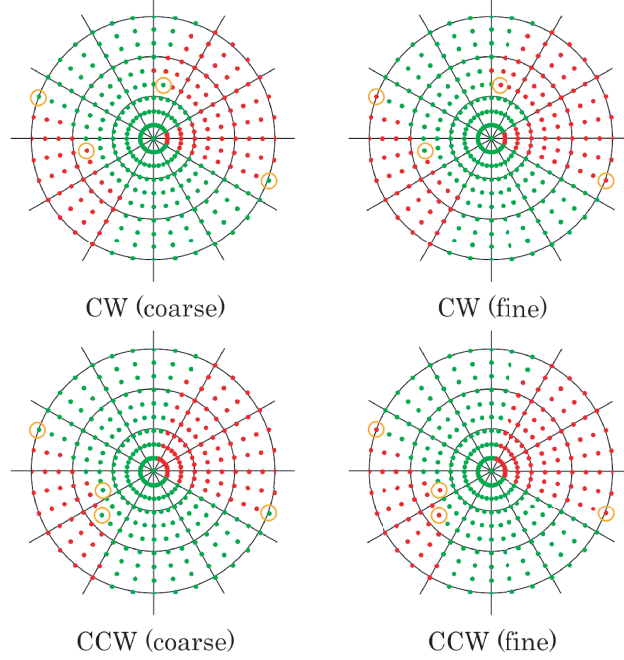
**Figure 18:**  $\theta = 30^\circ$ ,  $\phi = 60^\circ$ . With this rotation axis, *CW* rotation drains the workpiece but *CCW* rotation does not. The rotation axis is set perpendicular to the paper for the left and center figures. The right figure is a view from a different angle.



**Figure 19:** The transition of a water particle when we rotate (a) clockwise and (b) counterclockwise around a rotation axis  $\theta = 30^\circ$ ,  $\phi = 60^\circ$ . *CW* rotation drains the workpiece but *CCW* rotation does not.

triangulation, we compared these results to those on a fine tessellation of the same model. We found only slight shifts in the boundary between the drainable and non-drainable regions (see Figure 20).

We also applied our algorithm to a complex automotive model shown in Figure 21(a). Our algorithm can quickly compute whether or not a given rotation axis will drain the workpiece even when internal passages (Figure 21(b)(c)) are very complex, as in this example. Figure 21(d) and (e) plot whether or not the indicated rotation axis drains this model. Figure 21 (f) shows concave vertices (colored blue) where a water trap is potentially formed when the rotation axis is set to  $\theta = 270^\circ$ ,  $\phi = 0^\circ$ . Figure 21 (g) shows the







**Figure 20:** Comparison of the results shown in Figure 16 (a) and (b) with results for a finer tessellation of the same model with almost five times the number of vertices. The results that differ are circled.

corresponding axis-aligned magnified view.

## 5.2. Performance

Table 1 shows the performance of our implementation on a 2.66GHz CPU with 4GB of RAM. The initialization mainly consists of identifying the concave vertices of the input mesh and constructing a BVH for speeding up the ray tracing phase of the geometry. This information can be reused no matter what rotation axis is being considered, and typically more than one possible rotation axis will need to be tested. Then, after the initialization, we tested  $36 \times 9 = 324$  (sampled at every 10 degrees in both  $\theta$  and  $\phi$  directions) rotation axes for each model and report the average and maximum time in Table 1. We can see that we can test a given rotation axis very quickly and give near-interactive feedback to designers for testing each additional axis. Table 2 shows the detailed timing data, showing the individual timing for each of three phases to test a rotation axis, i.e. finding  $g_{v(CW)}^*$  and  $g_{v(CCW)}^*$  (section

**Table 1:** Time for initialization and to test each additional rotation axis (average and maximum).

		triangles	vertices	concave vertices	initialization time (sec)	average time (sec)	maximum time (sec)
(a)		3,572	1,796	428	0.575	0.006	0.035
(b)		12,0004	59,920	18,203	5.841	0.195	0.278
(c)		160,312	79,982	31,829	9.319	0.360	0.760
(d)		289,956	144,546	57,412	20.742	0.933	5.730

**Table 2:** Detailed timing data, showing the individual timing (average and maximum) for each of three phases to test each additional rotation axis.

	Find $g_{v(CW)}^*$ and $g_{v(CCW)}^*$		Find $S_{v(CW)}$ and $S_{v(CCW)}$		Checking Drainability		Total	
	average(sec)	max(sec)	average(sec)	max(sec)	average(sec)	max(sec)	average(sec)	max(sec)
(a)	0.001	0.011	0.002	0.015	0.004	0.009	0.006	0.035
(b)	0.045	0.064	0.138	0.227	0.013	0.017	0.195	0.278
(c)	0.067	0.106	0.279	0.654	0.014	0.025	0.360	0.760
(d)	0.131	0.169	0.774	5.575	0.028	0.040	0.933	5.730

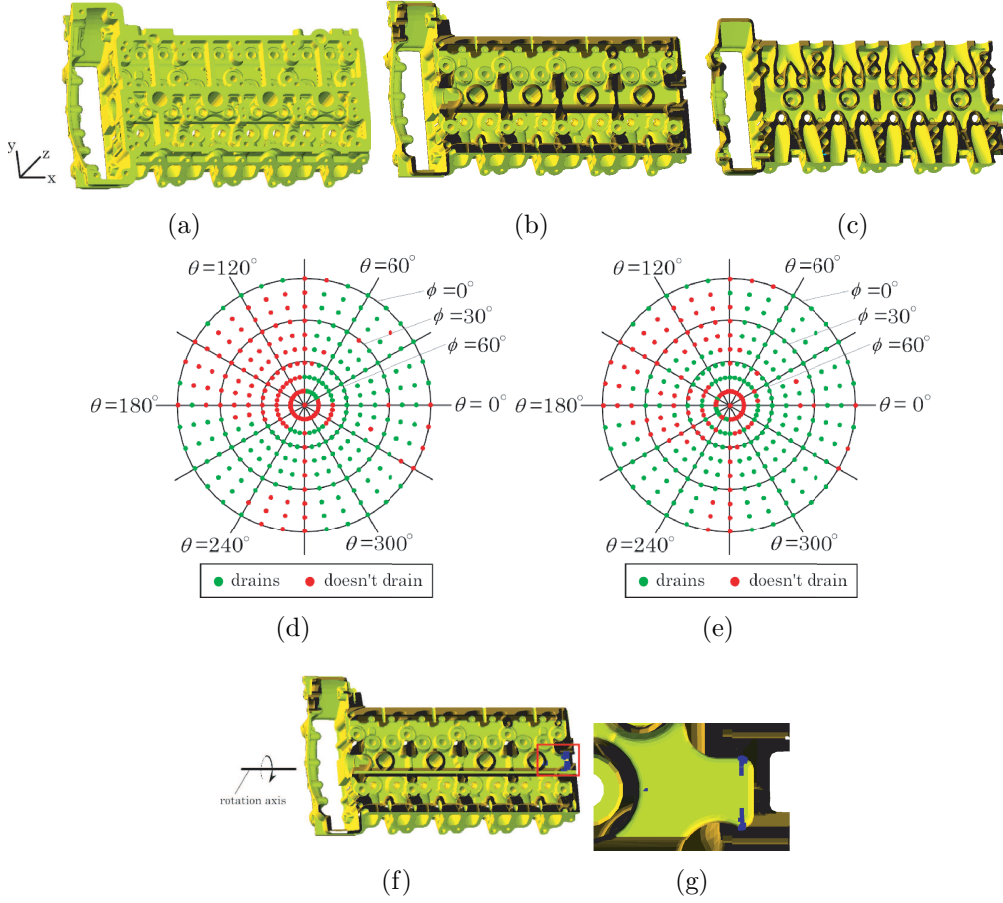
3.2.1), finding  $S_{v(CW)}$  and  $S_{v(CCW)}$  (section 3.2.2), and testing drainability by analyzing the draining graph constructed through the preceding two phases (section 4). The performance bottleneck of our current implementation is finding  $S_{v(CW)}$  and  $S_{v(CCW)}$  (i.e. the particle tracing operation in the graph construction phase), so we will investigate offloading some of this work to the GPU in future work. We also measured the number of function calls made to Algorithm 1-5 in determining  $S_{v(CW)}$  and  $S_{v(CCW)}$  for each concave vertex  $v \in V_c$  (shown in Table 3). Although the maximum number of calls is higher for the more complex models, the average was low for all models tested.

**Table 3:** *The number of function calls (average and maximum) to determine  $S_{v(CW)}$  and  $S_{v(CCW)}$  for each concave vertex  $v \in V_c$ .*

	TraceFromVertex		TraceFromEdge		ParticleDrop		FindNextEdge		TraceOnFlatRegion		Total	
	average	max	average	max	average	max	average	max	average	max	average	max
(a)	3.02	31	5.42	329	0.26	11	5.72	333	0.01	2	14.43	694
(b)	4.42	1108	6.50	1661	0.44	152	7.37	1772	0.001	2	18.74	4046
(c)	3.35	538	4.97	1338	0.40	235	5.53	1558	0.005	5	14.25	3440
(d)	3.87	14260	8.25	121779	0.59	7392	9.16	134672	0.020	16	21.89	278103

## 6. Complexity Analysis

Since our ultimate goal is to find a rotation axis for a geometric model such that when the workpiece is rotated around this axis, all water drains, we may need to test many candidate axes. Therefore, it is important that our testing algorithm run quickly. We now analyze the scalability of our algorithm. In the graph construction phase, for each concave vertex  $v \in V_c$ , first we compute the gravity directions when the trapped water at  $v$  starts to flow out. For each concave vertex  $v$ , this takes a constant number of operations equal to the number of edges incident to  $v$ , so it is in  $O(n)$ . For each concave vertex  $v \in V_c$ , we find the concave vertex into which the trapped water particle flowing out from  $v$  settles. In theory, for each  $v$ , we have to check all triangles and vertices of the geometry to find the final location in the worst case. Therefore, as Table 3 shows, the maximum number of function calls possibly becomes very high. We are still investigating the performance of particle tracing to construct this graph. However, from the fact that a water particle is driven by only a fixed gravity force and the assumption that the input triangles and vertices are uniformly distributed in space, in practice the number of vertices and triangles checked are only a very small fraction of  $n$ , reducing worst case  $O(n^2)$  growth to close to linear on average in practice; experimental results shown in Table 3 support this. Once the graph is constructed, the checking phase runs in  $O(n)$  time as described in section 4.



**Figure 21:** (a) Cylinder head model (b)(c) Cross sections revealing the internal passages of the model shown in (a). (d) Plot of whether or not rotation around a given rotation axis completely drains the workpiece under CW rotation and (e) CCW rotation. (f) The concave vertices (colored blue) such that once a water particle is trapped there, it will never exit the workpiece when we rotate it around rotation axis  $\theta = 270^\circ$ ,  $\phi = 0^\circ$ , which is set horizontally in the plane of the paper. (g) Magnified view of the region indicated in (f).

## 7. Discussion and Future Work

Since this is the first research to our knowledge that addresses testing a rotation axis for drainability, we have made a number of simplifying as-

sumptions to make the problem more tractable. In our future work, we plan to test and/or relax these assumptions as we build on this work to develop more sophisticated variations of our algorithm. The impact of some of our assumptions must be tested experimentally, such as ignoring the effect of viscosity. As a first step, we can also compare our results with the output of a physics-based approach.

Although we have shown theoretically that a rotation axis that drains all the core particles must eventually drain the entire part, our existing algorithm would need some modifications to calculate how many rotations will be needed. As we showed, for a water trap containing multiple water particles, not all water particles will move to the same water trap that the core particle moves to.

Ultimately, of course, we hope to move beyond testing given axes (a sample-based approach) to finding all drainable axes (using a configuration space approach).

## 8. Conclusion

In this paper, we presented a new geometric algorithm to test whether a rotation around a given rotation axis can drain an input geometry. Our proof-of-concept implementation can test input meshes of complex industrial parts containing over 100,000 vertices in about a second, a huge improvement compared to using commercial general-purpose simulation packages that can take hours to converge.

## Acknowledgments

We would like thank Sushrut Pavanaskar for background research on particle systems and physical simulations in computer graphics and feedback on the presentation. We also would like to thank Adarsh Krishnamurthy, Wei Li, and the anonymous reviewers for additional valuable feedback. This material is based on work supported in part by Daimler AG, UC Discovery under Grant No. DIG07-10224, and the National Science Foundation under Grant No. 0621198.

This paper is an extended version of the work presented in SPM '09: 2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling [15].



## Appendix A. Boundary of $H_{i(xy)}$

The boundary of  $H_{i(xy)}$  is defined by the intersection points between the boundary of  $H_i$  and the xy-plane Gaussian circle. Let the intersection point be  $I = (I_x, I_y, 0)$ . Since it is confined on the Gaussian circle,  $I_x^2 + I_y^2 = 1$ . From the definition, the boundary of  $H_i$  is defined by the plane perpendicular to  $e_i$ . Letting  $e_i = ((e_i)_x, (e_i)_y, (e_i)_z)$ , this plane is expressed as  $(e_i)_x x + (e_i)_y y + (e_i)_z z = 0$ . Then, assuming  $(e_i)_x \neq 0$ , we can solve for  $I_x$ ,

$$\begin{aligned} (e_i)_x(I_x) + (e_i)_y(I_y) + (e_i)_z(0) &= 0 \\ I_x &= -\frac{(e_i)_y}{(e_i)_x} I_y \quad ((e_i)_x \neq 0) \end{aligned}$$

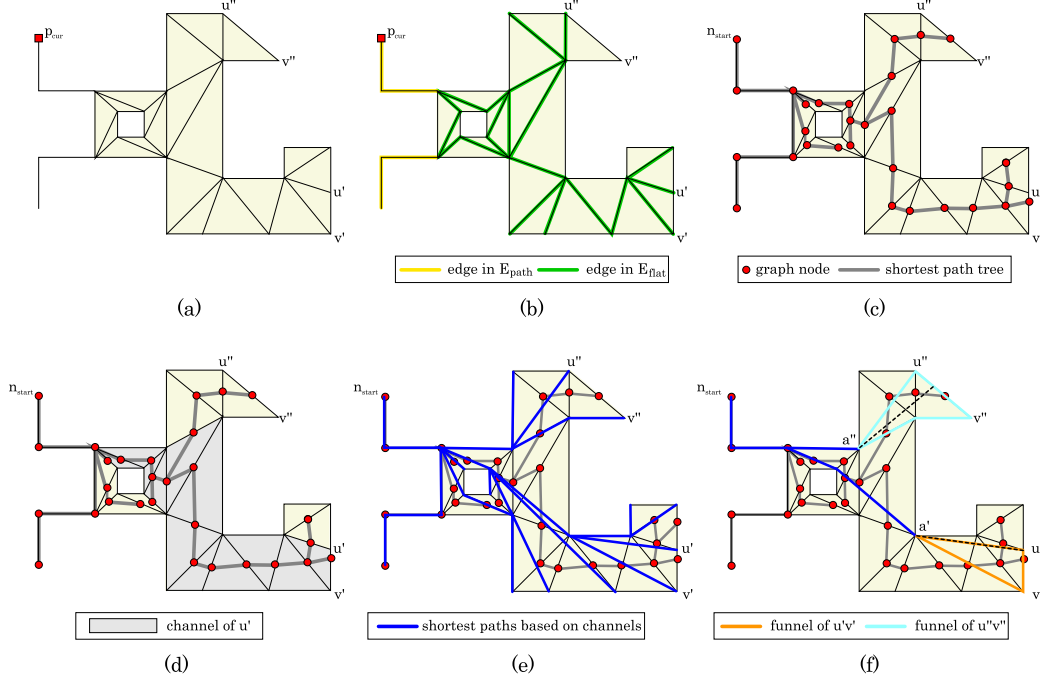
Substituting into  $I_x^2 + I_y^2 = 1$ ,

$$\begin{aligned} \left(\frac{(e_i)_y}{(e_i)_x}\right)^2 I_y^2 + I_y^2 &= 1 \\ \left(\frac{(e_i)_y}{(e_i)_x}\right)^2 + 1) I_y^2 &= 1 \\ I_y &= \pm \sqrt{\frac{1}{\left(\frac{(e_i)_y}{(e_i)_x}\right)^2 + 1}} \quad ((e_i)_x \neq 0) \end{aligned}$$

Note that the boundary of  $H_i$  and the Gaussian circle intersect at two points. When  $(e_i)_x = 0$ , if  $(e_i)_y \neq 0$ ,  $I_x = \pm 1$  and  $I_y = 0$ , and if  $(e_i)_y = 0$  as well, the entire xy-plane Gaussian circle defines the boundary of  $H_i$ .

## Appendix B. Finding a closest point on a flat region

Among vertices in  $V_{cand}$  and points on edges in  $E_{cand}$  ( $V_{cand}$  and  $E_{cand}$  are the candidate vertices and edges where a water particle leaving  $p_{cur}$  may flow out through), we find the point  $p_f$  that is closest to  $p_{cur}$  along edges in  $E_{perp}$  and triangles in  $T_{perp}$ . Since this problem can be NP-hard [16], we solve the problem using a modification of the approximation method proposed by Kallmann [17]. First, we define a set  $E_{path}$ , the set of edges in  $E_{perp}$  both of whose incident triangles are not in  $T_{perp}$ . We also define a set  $E_{flat}$ , the set of edges in  $E_{perp}$  both of whose incident triangles are in  $T_{perp}$  (Figure B.22 (b)). Then, we consider a graph whose nodes consist of  $p_{cur}$ , vertices incident to  $E_{path}$ , and midpoints of edges in  $E_{flat}$  and  $E_{cand}$ . The graph's edges are those in  $E_{path}$  plus edges between any pair of nodes on the same triangle in  $T_{perp}$ . Letting  $n_{start}$  be a node corresponding to  $p_{cur}$  in the



**Figure B.22:** (a)  $p_{\text{cur}}$ , edges in  $E_{\text{perp}}$ , and triangles in  $T_{\text{perp}}$ . Suppose  $E_{\text{cand}} = \{\overline{u'v'}, \overline{u''v''}\}$ . (b) Edges in  $E_{\text{path}}$  (yellow) and edges in  $E_{\text{flat}}$  (green) are highlighted. (c) Shortest path tree from  $n_{\text{start}}$  (corresponding to  $p_{\text{cur}}$ ) on the graph whose nodes consist of  $p_{\text{cur}}$ , vertices incident to  $E_{\text{path}}$ , and midpoints of edges in  $E_{\text{flat}}$  and  $E_{\text{cand}}$ . Edges of the graph are those in  $E_{\text{path}}$  plus edges between any pair of nodes on the same triangle in  $T_{\text{perp}}$ . (d) Channel of  $u'$ . (e) Shortest path from  $n_{\text{start}}$  to each vertex along the channels. (f) The shortest path from  $n_{\text{start}}$  on edge  $\overline{u'v'}$  (respectively,  $\overline{u''v''}$ ) is the shortest path from the apex of the corresponding funnel  $a'$  (respectively,  $a''$ ). The shortest path from  $n_{\text{start}}$  to each edge lies on the dashed lines.

graph, we construct a shortest path tree from  $n_{\text{start}}$  on the graph using, for example, Dijkstra's algorithm (Figure B.22 (c)). At this point, the minimum distance to each vertex incident to edges in  $E_{\text{path}}$  is determined; therefore, if  $T_{\text{perp}}$  is empty (note that  $E_{\text{cand}}$  is also empty in this case), a vertex in  $V_{\text{cand}}$  corresponding to a graph node with a minimum distance on the shortest path tree is  $p_f$ . Otherwise, we find the minimum distances to other vertices using the funnel algorithm [18, 19]. The funnel algorithm finds the shortest path to each vertex inside a *channel*, a chain of triangles along the shortest path

tree (Figure B.22 (d)(e))). For an explanation of how the funnel algorithm works, refer to [16].

We have to find the shortest path from  $n_{start}$  to each edge in  $E_{cand}$ , since  $p_f$  may be located on some edge in  $E_{cand}$ . As shown in Figure B.22 (f), the shortest path from  $n_{start}$  to the edge's two endpoints  $u$  and  $v$  on the corresponding channel travel together and diverge at a vertex  $a$  (called the *apex*). The region bounded by edge  $\overline{uv}$  and concave chains from  $u$  and  $v$  to  $a$  is called the *funnel* [16]. The shortest path from  $n_{start}$  to edge  $\overline{uv}$  passes through  $a$ ; therefore, we can find the shortest path from  $n_{start}$  by finding the shortest path from  $a$ . If  $\overline{uv}$  and a half-line extending from  $a$  and perpendicular to  $\overline{uv}$  intersect, the line segment between  $a$  and the intersection point is the shortest path from  $a$  to  $\overline{uv}$  (as for  $a''$  and  $\overline{u''v''}$  in Figure B.22 (f)). Otherwise, the line segment between  $a$  and an intersection point between  $\overline{uv}$  and a tangent line of the funnel extending from  $a$  is the shortest path. There are two candidates, so we pick the shorter one (an example is the line from  $a'$  to  $u'$  in Figure B.22 (f)). Finally, the minimum distance from  $n_{start}$  to  $\overline{uv}$  along edges in  $E_{perp}$  and triangles in  $T_{perp}$  is the minimum distance from  $n_{start}$  to  $a$  plus the length of the line segment from the apex.

Now, we find the minimum distance from  $n_{start}$  to each vertex in  $V_{cand}$  and point in  $E_{cand}$  along edges in  $E_{perp}$  and triangles in  $T_{perp}$ . The candidate vertex or point on a candidate edge that has the minimum distance from  $n_{start}$  is  $p_f$ .

## References

- [1] D. Arbelaez, M. Avila, A. Krishnamurthy, W. Li, Y. Yasui, D. Dornfeld, S. McMains, Cleanability of mechanical components, in: Proceedings of 2008 NSF Engineering Research and Innovation Conference, 2008.
- [2] M. Avila, C. Reich-Weiser, D. Dornfeld, S. McMains, Design and manufacturing for cleanability in high performance cutting, in: Proceeding of 2nd International High Performance Cutting Conference, 2006.
- [3] K. Berger, Burrs, chips and cleanness of parts - activities and aims in the German automotive industry, in: Presentation at CIRP Working Group on Burr Formation, 2006.
- [4] M. Müller, J. Stam, D. James, N. Thürey, Real time physics: class notes, in: SIGGRAPH '08: ACM SIGGRAPH

- 2008 classes, ACM, New York, NY, USA, 2008, pp. 1–90.  
doi:<http://doi.acm.org/10.1145/1401132.1401245>.
- [5] H. Nguyen, GPU gems 3, Addison-Wesley Professional, 2007.
  - [6] M. Müller-Fischer, D. Charypar, M. Gross, Particle-based fluid simulation for interactive applications, in: SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, Eurographics Association, 2003, pp. 154–159.
  - [7] M. Müller-Fischer, P. Mark, G. Markus, K. Richard, W. Martin, Physics-based animation, in: M. Gross, H. Pfister (Eds.), Point-Based Graphics, Morgan Kaufmann, Burlington, 2007, pp. 340 – 387.
  - [8] G. Aloupis, J. Cardinal, S. Collette, F. Hurtado, S. Langerman, J. O'Rourke, Draining a polygon - or - rolling a ball out of a polygon., in: CCCG, 2008.  
URL <http://dblp.uni-trier.de/db/conf/cccg/cccg2008.html#AloupisCCHL008>
  - [9] F. J. Bradley, S. Heinemann, J. A. Hoopes, A hydraulics-based/optimization methodology for gating design, Applied Mathematical Modelling 17 (8) (1993) 406 – 414.
  - [10] P. Bose, M. van Kreveld, G. Toussaint, Filling polyhedral molds, Computer-Aided Design 30 (4) (1998) 245 – 254.
  - [11] K. Tang, L.-L. Chen, S.-Y. Chou, Optimal workpiece setups for 4-axis numerical control machining based on machinability, Computers in Industry 37 (1) (1998) 27 – 41.
  - [12] T. C. Woo, Visibility maps and spherical algorithms, Computer-Aided Design 26 (1).
  - [13] H. Samet, Foundations of Multidimensional and Metric Data Structures (The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling), Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.
  - [14] J. L. Bentley, Multidimensional binary search trees used for associative searching, Commun. ACM 18 (9) (1975) 509–517.  
doi:<http://doi.acm.org/10.1145/361002.361007>.

- [15] Y. Yasui, S. McMains, Testing an axis of rotation for 3D workpiece draining, in: SPM '09: 2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling, ACM, New York, NY, USA, 2009, pp. 223–233. doi:<http://doi.acm.org/10.1145/1629255.1629283>.
- [16] J. Hershberger, J. Snoeyink, Computing minimum length paths of a given homotopy class, *Comput. Geom. Theory Appl.* 4 (2) (1994) 63–97. doi:[http://dx.doi.org/10.1016/0925-7721\(94\)90010-8](http://dx.doi.org/10.1016/0925-7721(94)90010-8).
- [17] M. Kallmann, Path planning in triangulations, in: *Proceedings of the IJCAI Workshop on Reasoning, Representation, and Learning in Computer Games*, Edinburgh, Scotland, 2005.
- [18] D. T. Lee, F. P. Preparata, Euclidean shortest paths in the presence of rectilinear barriers, *Networks* 14 (3) (1984) 393–410.
- [19] B. Chazelle, A theorem on polygon cutting with applications, in: *SFCS '82: Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, IEEE Computer Society, Washington, DC, USA, 1982, pp. 339–349. doi:<http://dx.doi.org/10.1109/SFCS.1982.58>.