

Testing an Axis of Rotation for 3D Workpiece Draining

Yusuke Yasui
UC Berkeley

Sara McMains
UC 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 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 check 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.

CR Categories: J.6 [Computer Applications]: COMPUTER-AIDED ENGINEERING—Computer-aided design (CAD); J.6 [Computer Applications]: COMPUTER-AIDED ENGINEERING—Computer-aided manufacturing (CAM)

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 [Arbelaez et al. 2008; Avila et al. 2006; Berger 2006]. 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. 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 check 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 CFD. Although the computational power of CPUs and GPUs 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 [Müller et al. 2008]. The first real-time GPU implementation of fluid simulation using a regular grid of cubical cells was reported in [Nguyen 2007]. 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 [Müller-Fischer et al. 2003; Müller-Fischer et al. 2007]. 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, 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. [Aloupis et al. 2008]. Given a closed polygon and trapped water particles 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 [Bradley et al. 1993] 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. [Bose et al. 1998] 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.

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. [Tang et al.

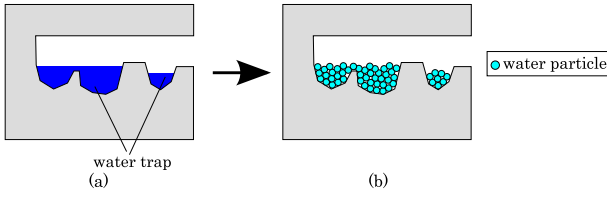


Figure 1: We assume that we can approximate a volume of water (shown in (a) for a 2D example) by a set of water particles (shown in (b)). A water trap is a set of water particles directly or indirectly touching each other and some of which are touching the input geometry. In this example, two water traps are formed.

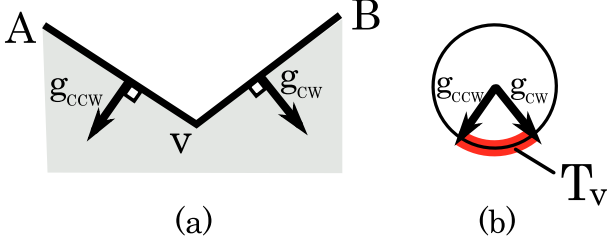


Figure 2: (a): Concave vertex v (b): The diagram showing T_v of v .

1998]. 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 [Woo 1994].

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 forces such as friction 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 1).

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.

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.

2 Approach and Theory

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

2.1 Limit case: each water trap is represented by a single water particle

First, we consider the limit 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 2 shows a specific example. In the 2D case, each of the two gravity directions g_{CCW} and g_{CW} bounding T_v are orthogonal to the two edges incident to v . For a workpiece orientation with corresponding gravity direction currently in T_v , when the workpiece rotates far enough that the gravity direction coincides with g_{CCW} (respectively, g_{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 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_{CW} or counterclockwise and gravity coincides with g_{CCW} . If the water particle trapped at a vertex exits the workpiece once it is rotated so that gravity coincides with g_{CW} or g_{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 3.

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 3. Finally, we trace the path of a trapped water particle under both of these gravity directions 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.

For each concave vertex, if there is a path consisting of edges with the same direction label from the corresponding node to the node labeled as “out,” then we can drain the trapped water particle at that concave vertex through concave vertices corresponding to the nodes along the path by rotating in the given direction. For example, there is a path “ $A \rightarrow B \rightarrow C \rightarrow D \rightarrow out$ ” in Figure 3 corresponding to the draining sequence shown in Figure 4. 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. We can also see that when water is trapped at any one of A , B , C , or D , we cannot drain by counterclockwise rotation because there is no path from the nodes corresponding to any of these vertices to “out”; this is physically consistent.

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 limit case that each water trap consists of only one water particle, and pro-

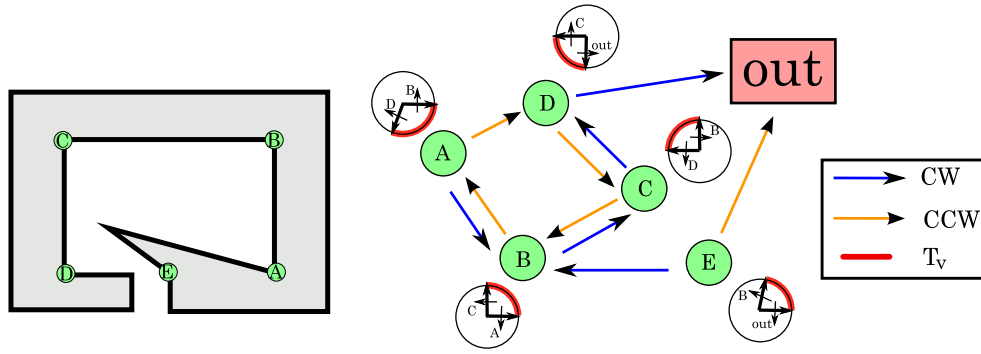


Figure 3: A sample geometry in 2D and the corresponding draining graph. The diagram next to each graph node shows the two gravity directions that let a water particle trapped at the corresponding concave vertex leave for the other concave vertices. For each node, when a current gravity direction is in T_v , if a water particle is at the corresponding concave vertex, it will be trapped.

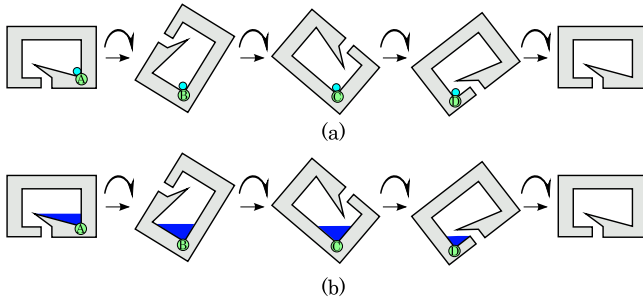


Figure 4: (a) The transition of a water particle by a rotation corresponding to the CW path “A → B → C → D → out” in Figure 3. (b) The corresponding drainage of water.

vided the approach to solve the corresponding draining problem. We now show that if a solution exists for this limit 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 5); 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 5 (a). Figure 5 (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 5 (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, a 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, as-

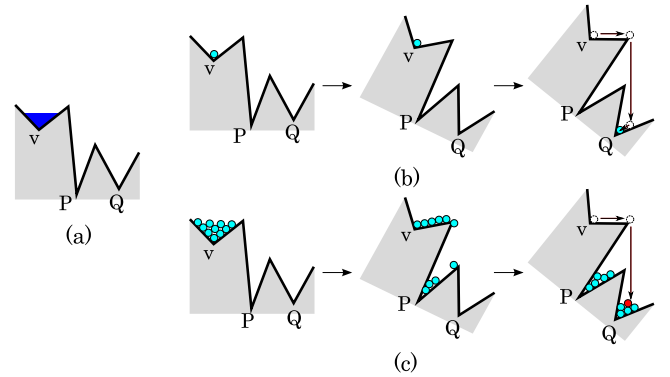


Figure 5: (a) A water trap is formed at concave vertex v . (b) The movement of a water particle when we approximate the water trap by only one particle. After v is drained, a new water trap is formed at concave vertex Q . (c) The movement of water particles when we approximate the water trap by a set of particles. After v is drained, new water traps are formed at P and Q . A core particle shown in red settles at a water trap at the same concave vertex where the particle in case (b) settles.

suming 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 → B → A → out.” 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 concave vertices 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 limit case that each water trap

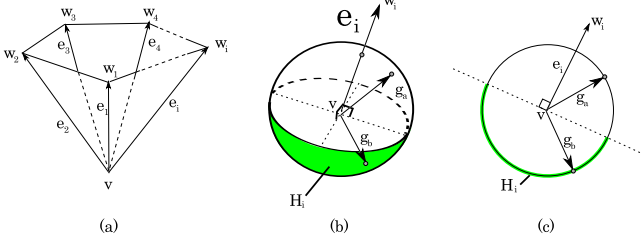


Figure 6: (a) concave vertex v . (b) e_i and the corresponding H_i . Gravity direction g_a never causes a water trap at v , but g_b may cause a water trap at v . (c) The cross section of (b) including g_a and g_b .

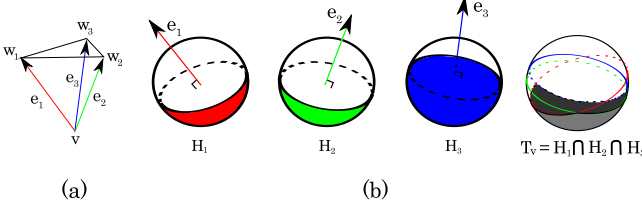


Figure 7: (a) concave vertex v . (b) The corresponding H_i and T_v .

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. We define the three types of vertices (convex, concave, and saddle) as follows:

Vertex classification Given a vertex v , letting $\text{vale}(v)$ be its valence, v is either convex or concave if there is a unit vector d such that, for all the adjacent vertices w_i of v ($i = 1, 2, \dots, \text{vale}(v)$), $(w_i - v) \cdot d < 0$ (i.e. v is a locally extreme vertex). If there is no such d , v is a saddle vertex.

Letting ϵ be a positive infinitesimal number, if a point $p = v + \epsilon d$ is inside of the given geometry, v is a concave vertex. Otherwise, v is a convex 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_{CW} and g_{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 vertex into which a water particle flowing out will settle by tracing the particle's path along geometric features (vertices, edges, and triangles).

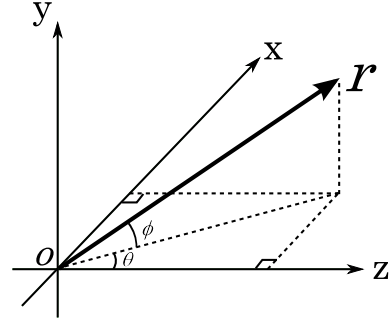


Figure 8: We describe any rotation axis $r = (r_x, r_y, r_z)$ by two variables θ ($0^\circ \leq \theta < 360^\circ$) and ϕ ($0^\circ \leq \phi \leq 90^\circ$) where θ is the azimuthal angle in the xz -plane from the z -axis and ϕ is the polar angle from the xz -plane.

3.2.1 Gravity direction causing 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 6 (a)). For each e_i , we define a half-space H_i of directions on the Gaussian sphere $H_i = \{p \mid e_i \cdot (p - v) \leq 0, \|p - v\| = 1\}$. Figure 6 (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 6 (b) and (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 7). Then, a gravity direction g in T_v potentially causes a water trap at v . On the other hand, as $\overline{T_v} = \overline{\bigcap_i H_i} = \bigcup_i \overline{H_i}$ suggests, 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 on the boundary of T_v . To find these gravity directions, given a rotation axis, consider first rotating the input geometry such that the rotation axis coincides with the z -axis (setting the coordinate system so that the z -axis is horizontal, parallel to the ground). Then, possible gravity directions are confined in the xy -plane because a gravity direction and a 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 $r = (r_x, r_y, r_z)$ by two variables θ ($0^\circ \leq \theta < 360^\circ$) and ϕ ($0^\circ \leq \phi \leq 90^\circ$) where θ is the azimuthal angle in the xz -plane from the z -axis and ϕ is the polar angle from the xz -plane as shown in Figure 8. Using these parameters, the components of r can be expressed as $r_x = \cos \phi \sin \theta$, $r_y = \sin \phi$, and $r_z = \cos \phi \cos \theta$. Then, by multiplying each vertex by the matrix R where

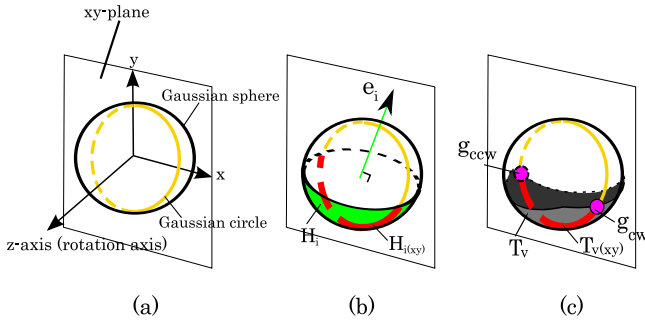


Figure 9: (a) The relationship between the Gaussian sphere and the xy-plane Gaussian circle. (b) The relationship between H_i and $H_{i(xy)}$. (c) The relationship between T_v and $T_{v(xy)}$.

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

we can align the rotation axis to 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 hits 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 7 (b), T_v is bounded by a set of great circular arcs on the Gaussian sphere. If the xy-plane intersects with T_v , it intersects its boundary at two points because T_v is convex. These two points correspond to the gravity direction at which the trapped water particle at v flows out when we rotate clockwise and counterclockwise. Since $T_v = \bigcap_i H_i$, each of the arcs is defined by the boundary of an H_i . The two edges (i.e. the e_i corresponding to the H_i) that define the boundary of T_v intersecting with the xy-plane are the ones the water particle flows out through.

For the actual calculation to find the edges through which trapped water flows out and the corresponding gravity directions, 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 xy-plane Gaussian circle $T_{v(xy)}$ (see Figure 9). 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-plane¹. The appendix describes how to compute the boundary of $H_{i(xy)}$.

As shown in Figure 9 (c) and Figure 10 (a), we let g_{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_{ccw} the point on the Gaussian circle bounding $T_{v(xy)}$ rotating counterclockwise. Let e_{cw} and e_{ccw} be the edges defining g_{cw} and g_{ccw} respectively (recall that H_i is defined by edge e_i). Suppose that a water particle is trapped at $v \in V_c$, that is, g is in $T_{v(xy)}$. Once the gravity direction coincides with g_{cw} as the geometry is

¹ $T_{v(xy)} = T_v \cap xy = (\bigcap_i H_i) \cap xy = \bigcap_i (H_i \cap xy) = \bigcap_i H_{i(xy)}$ where xy is the xy-plane.

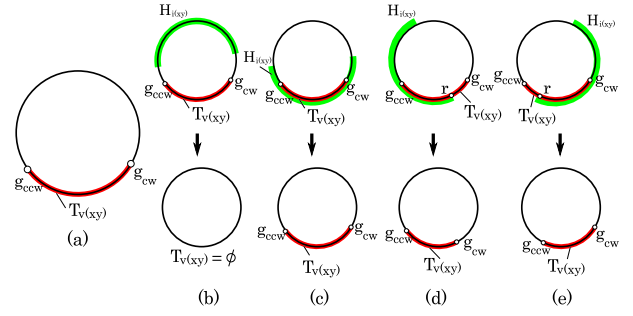


Figure 10: (a) g_{cw} , g_{ccw} , and $T_{v(xy)}$ on the Gaussian circle. (b)-(e) Updating g_{cw} , g_{ccw} , and $T_{v(xy)}$ when a new $H_{i(xy)}$ is introduced.

rotated clockwise around the axis, the trapped water particle flows out along edge e_{cw} . In a similar manner, once the gravity direction coincides with g_{ccw} under counterclockwise rotation, the trapped water particle flows out along edge e_{ccw} . We compute the boundary points of $T_{v(xy)}$, that is, g_{cw} and g_{ccw} , and the corresponding e_{cw} and e_{ccw} incrementally as follows.

Initially, g_{cw} and g_{ccw} are set to the two corresponding boundary points of $H_{1(xy)}$, and e_{cw} and e_{ccw} are both set to e_1 . Then, for each i ($i = 2, 3, \dots, \text{vale}(v)$), if necessary we update g_{cw} and g_{ccw} , that is, the boundaries of $T_{v(xy)}$, and e_{cw} and e_{ccw} as follows.

For each i , if neither of the g_{cw} or g_{ccw} calculated thus far are in $H_{i(xy)}$, $T_{v(xy)}$ is empty (Figure 10 (b)). On the other hand, if both g_{cw} and g_{ccw} are in $H_{i(xy)}$, we do not have to update $T_{v(xy)}$ (Figure 10 (c)). When one of g_{cw} and g_{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_{cw} is not in $H_{i(xy)}$, we set r to g_{cw} and e_i to e_{cw} (Figure 10 (d)). If g_{ccw} is not in $H_{i(xy)}$, we set r to g_{ccw} and e_i to e_{ccw} (Figure 10 (e)). After performing this update for each e_i ($i = 2, 3, \dots, \text{vale}(v)$), g_{cw} and g_{ccw} will be the points bounding $T_{v(xy)}$.

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 given axis, and the two corresponding edges along which the trapped water flows. Now, we describe how to determine which concave vertex the water particle flowing out from v settles in (or if it exits the geometry). Given the gravity direction and outflow edge, we determine it by tracing the path of the water particle, assuming that the particle always follows the path such that the gravity force works at a maximum (i.e. take the possible path such that the angle between the vector representing the path and the gravity direction is minimized).

We only describe the case when we rotate the geometry clockwise because the same procedure works for the counterclockwise case. Let the concave vertex where the water particle settles under clockwise rotation be $v_{s(cw)}$ and under counterclockwise rotation be $v_{s(ccw)}$. For the sake of simplicity of notation, we let $v_s \stackrel{\text{def}}{=} v_{s(cw)}$ and $g \stackrel{\text{def}}{=} g_{cw}$ for a moment. Given a vertex v , let w_i be a member of the set of adjacent vertices of v and e_i be the vector from v to w_i whose length is normalized, that is, $e_i = (w_i - v) / \|w_i - v\|$ ($i = 1, 2, \dots, \text{vale}(v)$). We also define $\text{proj}(t)$ as the projection of gravity vector g onto the plane of

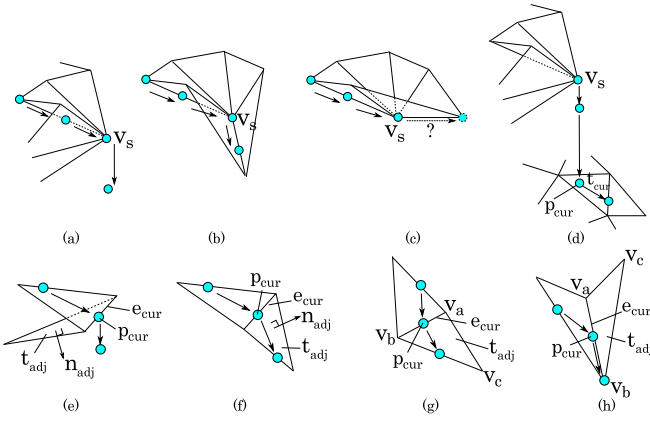


Figure 11: Transition cases of a water particle under gravity on various geometric shapes. (a)(b)(c) Possible movements from a vertex. (d) Movement when a particle drops vertically. (e)(f) Possible movements from a ridge edge. (g)(h) Possible movements from a valley edge.

triangle t ; thus, for triangle t 's normal vector n_t , we can calculate $proj(t) = (I - n_t n_t^T)g$. The three cases of a particle leaving a vertex and falling through space, traveling along an edge, or traveling along edges perpendicular to g are handled by Procedure 1: **TraceFromVertex** (Figure 11 (a)(b)(c)). The three cases for a particle leaving a location in the middle of an edge and falling through space, traveling along the face of a triangle, or traveling along an edge are handled by Procedure 4: **TraceFromEdge** (Figure 11 (e)(f)(g)(h)). Procedure 2: **ParticleDrop** and Procedure 3: **FindNextEdge** handle the transitions between these states. We outline the logic below; the corresponding detailed pseudocode for each subroutine is shown in Algorithm 1, 2, 3, and 4.

First, we initialize v_s with the endpoint of e_{CW} that is not v . Then, starting from Procedure 1, we update v_s if necessary by tracking the particle location, assuming that the only relevant force applied to water particles is the gravity force g .

1. TraceFromVertex

- If a point $v_s + \epsilon g$ (ϵ is a positive infinitesimal number) is outside of the geometry, the water particle falls down parallel to g from v_s (Figure 11 (a)). To simulate this, we shoot a half-ray $v_s + \gamma g$ (where γ is a positive scalar). **Go to 2.**
- Otherwise, for each e_i of v_s , we compute $e_i \cdot g$. We define $m = \arg \max_i (e_i \cdot g)$ (i.e. $\forall i, e_m \cdot g \geq e_i \cdot g$),
 - if $e_m \cdot g > 0$, we set the other endpoint of e_m to v_s . **Go to 1.** (Figure 11 (b).)
 - if $e_m \cdot g = 0$ (g is on the boundary of $T_{v_s(xy)}$), we cannot decide whether the water particle moves to another vertex or settles at v_s by looking only at local information at v_s (Figure 11 (c)). Hence, we try to find a vertex v_{fo} which has at least one e_j such that $e_j \cdot g > 0$ by traversing only edges perpendicular to g .
 - * if any such v_{fo} is found, we set v_s to the v_{fo} with maximum $e_j \cdot g$ value. **Go to 1.**
 - * otherwise, the water particle settles at v_s . **Done.**

Algorithm 1 TraceFromVertex

```

if  $v_s + \epsilon g$  is outside of the geometry then
  ParticleDrop( $v_s + \gamma g$ )
else
   $m \leftarrow \arg \max_i (e_i \cdot g)$ 
  if  $e_m \cdot g > 0$  then
     $v_s \leftarrow w_m$ 
    TraceFromVertex()
  else if  $e_m \cdot g = 0$  then
    Queue  $q$ 
     $q.enqueue(v_s)$ 
     $max\_val \leftarrow 0$ 

    while  $q.isEmpty() == \text{false}$  do
       $v_{temp} \leftarrow q.dequeue()$ 
      for  $j = 1$  to  $vale(v_{temp})$  do
        if  $w_j$  has not been in  $q$  yet then
          if  $e_j \cdot g > max\_val$  then
             $v_s \leftarrow w_j$ 
             $max\_val \leftarrow e_j \cdot g$ 
          else if  $e_j \cdot g = 0$  then
             $q.enqueue(w_j)$ 
          end if
        end if
      end for
    end while
    if  $max\_val > 0$  then
      TraceFromVertex()
    end if
  end if

```

- if $e_m \cdot g < 0$ (g is in $T_{v_s(xy)}$), the water particle flowing out from v settles at v_s . **Done.**

2. ParticleDrop

- If the ray does not hit any triangles of the input geometry, the water particle exits the geometry; v_s is set to *out*. **Done.**
- Otherwise, letting the triangle the ray hits be t_{cur} and the point the ray hits be p_{cur} , **Go to 3.** (Figure 11 (d).)

3. 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 α and β that satisfy $p_{cur} + proj(t_{cur})\alpha = v_a(1 - \beta) + v_b\beta$, $\alpha > 0$, and $0 \leq \beta \leq 1$.
 - if $\beta = 0$, assign v_a to v_s . **Go to 1.**
 - if $\beta = 1$, assign v_b to v_s . **Go to 1.**
 - if $0 < \beta < 1$, let this intersecting edge be e_{cur} , the triangle across e_{cur} be t_{adj} , and $p_{cur} = v_a(1 - \beta) + v_b\beta$. **Go to 4.**

4. TraceFromEdge

- If e_{cur} is a ridge edge, letting the normal vector of t_{adj} be n_{adj} ,
 - if $n_{adj} \cdot g \geq 0$, the water particle falls down; we shoot a half-ray $p_{cur} + \gamma g$ (where γ is a positive scalar). **Go to 2.** (Figure 11 (e).)

Algorithm 2 ParticleDrop

Input: half-ray h_Ray
if half-ray h_Ray hits a triangle t_{cur} **then**
 $p_{cur} \leftarrow$ point where h_Ray intersects t_{cur}
 FindNextEdge(t_{cur}, p_{cur})
else
 $v_s \leftarrow out$
end if

Algorithm 3 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 α and β s.t. $p_{cur} + proj(t_{cur})\alpha = v_a(1 - \beta) + v_b\beta$
 if $\alpha > 0$ **then**
 if $\beta = 0$ **then**
 $v_s \leftarrow v_a$
 TraceFromVertex()
 break
 else if $\beta = 1$ **then**
 $v_s \leftarrow v_b$
 TraceFromVertex()
 break
 else if $0 < \beta < 1$ **then**
 $t_{adj} \leftarrow$ triangle across e_{cur}
 $p_{cur} \leftarrow v_a(1 - \beta) + v_b\beta$
 TraceFromEdge(e_i, t_{adj}, p_{cur})
 break
 end if
 end if
end for

– otherwise, the water particle moves on the surface of the adjacent triangle. We set t_{adj} to t_{cur} . **Go to 3.** (Figure 11 (f).)

- Otherwise (e_{cur} is a valley edge), letting the two endpoints of e_{cur} be v_a and v_b , and the vertex of t_{adj} that is neither v_a nor v_b be v_c ,

– if $v_a \cdot g < v_c \cdot g$ and $v_b \cdot g < v_c \cdot g$, the water particle moves on the surface of the adjacent triangle. We set t_{adj} to t_{cur} . **Go to 3.** (Figure 11 (g).)

– otherwise (Figure 11 (h).),

* if $v_a \cdot g > v_b \cdot g$, set v_a to v_s . **Go to 1.**

* otherwise, set v_b to v_s . **Go to 1.**

We repeat this procedure for each v until we find v_s such that g is in $T_{v_s(xy)}$, or g is on the boundary of $T_{v_s(xy)}$ and v_{fo} is not found, or v_s is set to out .

For each concave vertex $v \in V_c$, we connect the corresponding node to the node corresponding to $v_{s(CW)}$ by an edge labeled CW and to the node corresponding to $v_{s(CCW)}$ by an edge labeled CCW . Note that the ray tracing performance in Procedure 2 will be very expensive for large inputs unless we use a bounding volume hierarchy (BVH) to limit the number of triangles tested. We used a kd-tree for the BVH in our implementation.

Algorithm 4 TraceFromEdge

Input: current edge e_{cur} , adjacent triangle t_{adj} , current point p_{cur}
if e_{cur} is a ridge edge **then**
 if $n_{adj} \cdot g \geq 0$ **then**
 ParticleDrop($p_{cur} + \gamma g$)
 else
 FindNextEdge(t_{adj}, p_{cur})
 end if
else
 (* e_{cur} is a valley edge *)
 $v_a \leftarrow$ one endpoint of e_{cur}
 $v_b \leftarrow$ other endpoint of e_{cur}
 $v_c \leftarrow$ vertex of t_{adj} that is neither v_a nor v_b
 if $v_a \cdot g < v_c \cdot g$ and $v_b \cdot g < v_c \cdot g$ **then**
 FindNextEdge(t_{adj}, p_{cur})
 else
 if $v_a \cdot g > v_b \cdot g$ **then**
 $v_s \leftarrow v_a$
 TraceFromVertex()
 else
 $v_s \leftarrow v_b$
 TraceFromVertex()
 end if
 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 that there is also a path from the intermediate nodes on this path to the out node. For example, in Figure 3, 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 S_n . Next, consider trapped water particles at concave vertices whose corresponding nodes are directly connected to the nodes in S_n by edges labeled as CW ; these can be drained as well. We add these nodes to S_n , and continue recursively. This recursion stops when all the nodes connecting to at least one of the nodes in S_n by the edges labeled as CW are in S_n . Then, after the recursion stops, if $|S_n| = n$, we can guarantee that trapped water particles at all of the concave vertices are completely drained by a rotation around the given rotation axis. Through this approach, we do not have to check the same node more than once. Therefore, the

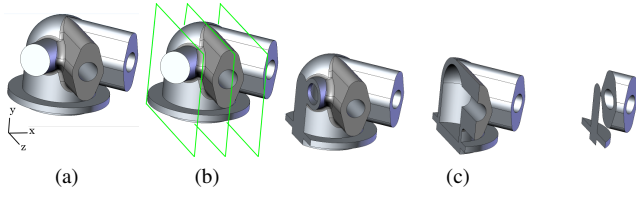


Figure 12: We applied our theory to a simple mechanical workpiece shown in (a). The three sections indicated in (b) are shown in (c).

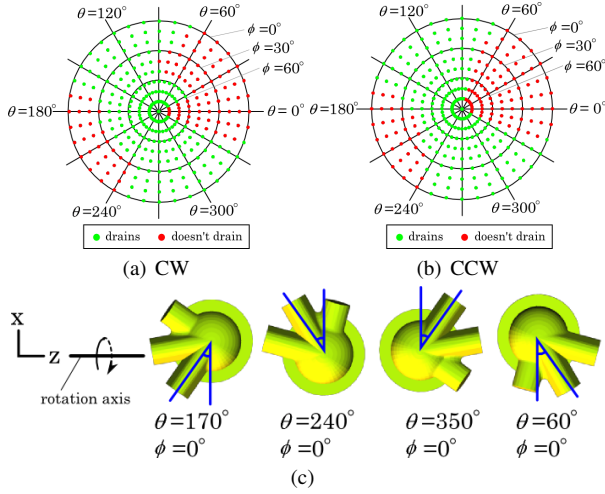


Figure 13: Whether or not a rotation around a given axis (θ , ϕ) completely drains the workpiece (a) under CW and (b) CCW rotation. (c) The configurations of the workpiece when $\phi = 0$ and $\theta = 170^\circ$, 240° , 350° , and 60° that are the limits in the $\phi = 0$ plane of whether the rotation 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.

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 12(a). Figure 12(b) indicates the locations of the three cross-sections shown in Figure 12(c), revealing an inside void of the workpiece where water can be trapped. Figure 13 (a) and (b) plot whether or not a rotation around a given axis (θ , ϕ) completely drains the workpiece under CW and CCW rotation respectively. To verify these results, we show some representative configurations in Figure 13(c) for the CW case. All these configurations are when $\phi = 0$ (corresponding to rotation axes in the xz plane) and viewed from $+\infty$ on the y-axis; therefore, the corresponding rotation axis is horizontal, in the plane of the paper. 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 13(a). The four configurations shown in Figure 13(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

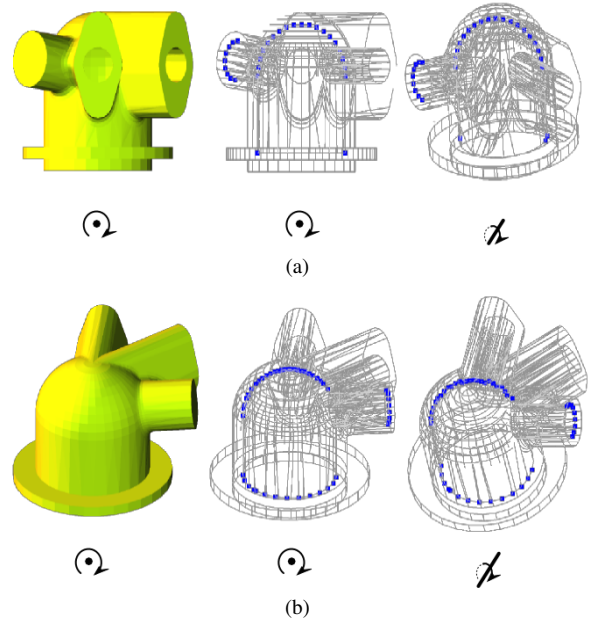


Figure 14: Some representative results. (a) $\theta = 0^\circ$, $\phi = 0^\circ$. (b) $\theta = 230^\circ$, $\phi = 30^\circ$. For both (a) and (b), the rotation axis is set perpendicular to the paper for the left and center figures. The right figure is a view from a different angle. For the center and 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.

same for all four cases; this angle is a threshold for whether or not a given rotation axis works for draining. This shows that our algorithm can capture this threshold.

Figure 14 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 15 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 13 (a) and (b)). Figure 16 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 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 17).

We also applied our algorithm to a complex automotive model shown in Figure 18(a). Our algorithm can quickly compute whether or not a given rotation axis will drain the workpiece even when internal passages (Figure 18(b)(c)) are very complex, as in this example. Figure 18(d) and (e) plot whether or not the indicated rotation axis drains this model. Figure 18 (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 18 (g) shows the corresponding axis-aligned magnified view.

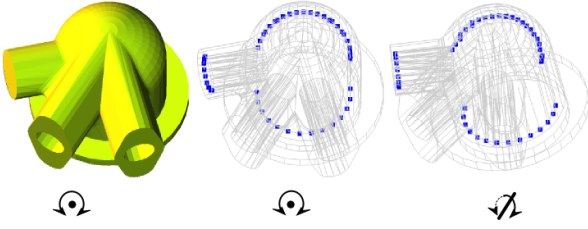


Figure 15: $\theta = 30^\circ$, $\phi = 60^\circ$. With this rotation axis, CW rotation does not drain the workpiece but CCW rotation does. The rotation axis is set perpendicular to the paper for the left and center figures. The right figure is a view from a different angle.

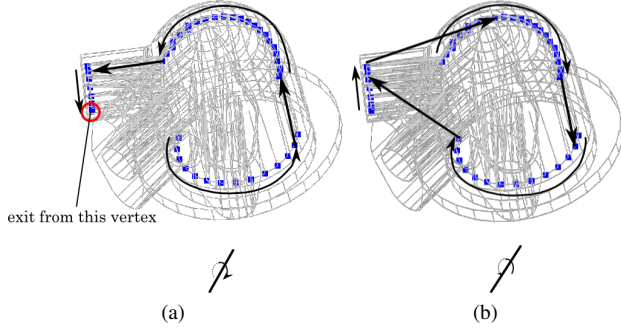


Figure 16: The transition of a water particle when we rotate (a) clockwise and (b) counterclockwise around a rotation axis $\theta = 30^\circ$, $\phi = 60^\circ$. CW rotation drains the workpiece but CCW rotation does not.

5.2 Performance

Table 1 shows the performance of our implementation on an Intel Pentium 4 3GHz CPU with 2GB of RAM. We tested $36 \times 9 = 324$ (sampled at every 10 degrees in both θ and ϕ 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. Note that since we can reuse the same BVH for speeding up the ray tracing phase, no matter how the rotation axis is changed, we did not include the preprocessing time in the times reported (recent GPU algorithms for BVH construction run in less than a second for models of similar complexity [Lauterbach et al. 2009]). The performance bottleneck of our current implementation is 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 1-4 in determining $v_{s(CW)}$ and $v_{s(CCW)}$ for each concave vertex $v \in V_c$. Although the maximum number of calls is higher for the more complex models, the average was similarly low, in single digits, for all models tested.

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. Vertex classification is linear in the size of the input geometry, $O(n)$. In the graph construction phase, for each concave vertex $v \in V_c$, first we compute the gravity

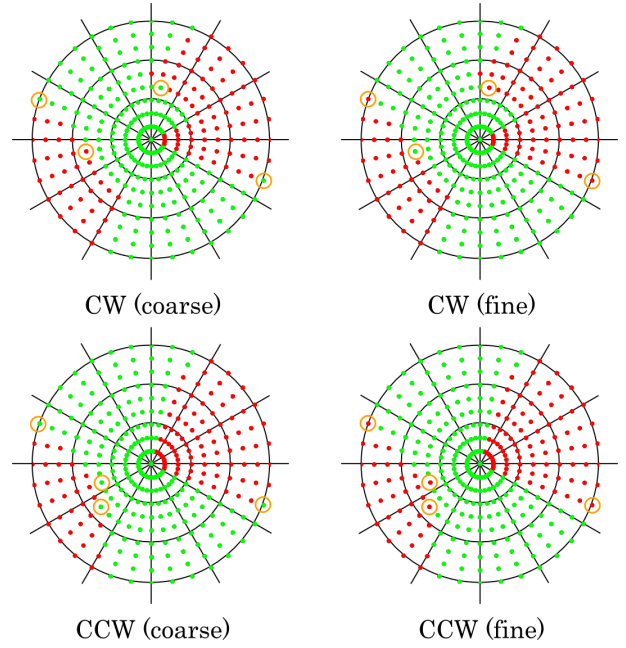


Figure 17: Comparison of the results shown in Figure 13 (a) and (b) with results for a finer tessellation of the same model with almost five times the number of vertices. The results that differ are circled.

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 also 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. 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 in practice; experimental results shown in Table 1 support this. Once the graph is constructed, the checking phase runs in $O(n)$ time as described in section 4.

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 assumptions 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. Similarly, at saddle or saddle-like vertices, for the case of multiple water

particles, not all will follow the steepest descent path, as we have assumed the core particle does. Capturing the salient behavior of the other water trap particles, without modeling them all explicitly, remains future work.

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 check 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.

References

- ALOUPIS, G., CARDINAL, J., COLLETTE, S., HURTADO, F., LANGERMAN, S., AND O'ROURKE, J. 2008. Draining a polygon - or - rolling a ball out of a polygon. In *CCCG*.
- ARBELAEZ, D., AVILA, M., KRISHNAMURTHY, A., LI, W., YASUI, Y., DORNFELD, D., AND MCMAINS, S. 2008. Cleanability of mechanical components. In *Proceedings of 2008 NSF Engineering Research and Innovation Conference*.
- AVILA, M., REICH-WEISER, C., DORNFELD, D., AND MCMAINS, S. 2006. Design and manufacturing for cleanability in high performance cutting. In *Proceeding of 2nd International High Performance Cutting Conference*.
- BALASUBRAMANIAM, M., LAXMIPRASAD, P., SARMA, S., AND SHAIKH, Z. 2000. Generating 5-axis NC roughing paths directly from a tessellated representation. *Computer-Aided Design* 32, 4, 261 – 277.
- BERGER, K. 2006. Burrs, chips and cleanness of parts - activities and aims in the German automotive industry. In *Presentation at CIRP Working Group on Burr Formation*.
- BOSE, P., VAN KREVELD, M., AND TOUSSAINT, G. 1998. Filling polyhedral molds. *Computer-Aided Design* 30, 4, 245 – 254.
- BRADLEY, F. J., HEINEMANN, S., AND HOOPES, J. A. 1993. A hydraulics-based/optimization methodology for gating design. *Applied Mathematical Modelling* 17, 8, 406 – 414.
- BRIDSON, R., AND MÜLLER-FISCHER, M. 2007. Fluid simulation: Siggraph 2007 course notes. In *SIGGRAPH '07: ACM SIGGRAPH 2007 Courses*, ACM, 1–81.
- DHALIWAL, S., GUPTA, S. K., HUANG, J., AND PRIYADARSHI, A. 2003. Algorithms for computing global accessibility cones.

Journal of Computing and Information Science in Engineering 3, 3, 200–209.

- LAUTERBACH, C., GARLAND, M., SENGUPTA, S., LUEBKE, D., AND MANOCHA, D. 2009. Fast BVH construction on GPUs. Eurographics Association.
- LI, Y., AND FRANK, M. C. 2007. Computing non-visibility of convex polygonal facets on the surface of a polyhedral CAD model. *Computer-Aided Design* 39, 9, 732 – 744.
- MÜLLER, M., STAM, J., JAMES, D., AND THÜREY, N. 2008. Real time physics: class notes. In *SIGGRAPH '08: ACM SIGGRAPH 2008 classes*, ACM, New York, NY, USA, 1–90.
- MÜLLER-FISCHER, M., CHARYPAR, D., AND GROSS, M. 2003. Particle-based fluid simulation for interactive applications. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Eurographics Association, 154–159.
- MÜLLER-FISCHER, M., MARK, P., MARKUS, G., RICHARD, K., AND MARTIN, W. 2007. Physics-based animation. In *Point-Based Graphics*, M. Gross and H. Pfister, Eds. Morgan Kaufmann, Burlington, 340 – 387.
- NGUYEN, H. 2007. *Gpu gems 3*. Addison-Wesley Professional.
- TANG, K., CHEN, L.-L., AND CHOU, S.-Y. 1998. Optimal work-piece setups for 4-axis numerical control machining based on machinability. *Computers in Industry* 37, 1, 27 – 41.
- WOO, T. C. 1994. Visibility maps and spherical algorithms. *Computer-Aided Design* 26, 1.

Appendix

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. Letting 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 ,

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


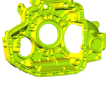
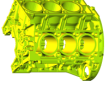
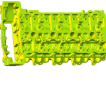
Substituting into $I_x^2 + I_y^2 = 1$,

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

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 .

Table 1: The required time to test one rotation axis and the number of function calls to determine $v_{s(CW)}$ and $v_{s(CCW)}$ for each concave vertex $v \in V_c$.

	triangles	vertices	concave vertices	average time (sec)	max time (sec)	average no. of function calls	max no. of function calls
	3,572	1,796	428	0.0048	0.047	4.34	66
	12,0004	59,920	18,203	0.335	0.407	6.67	127
	160,312	79,982	31,829	0.590	0.953	5.01	187
	289,956	144,546	57,412	1.10	2.14	6.55	416

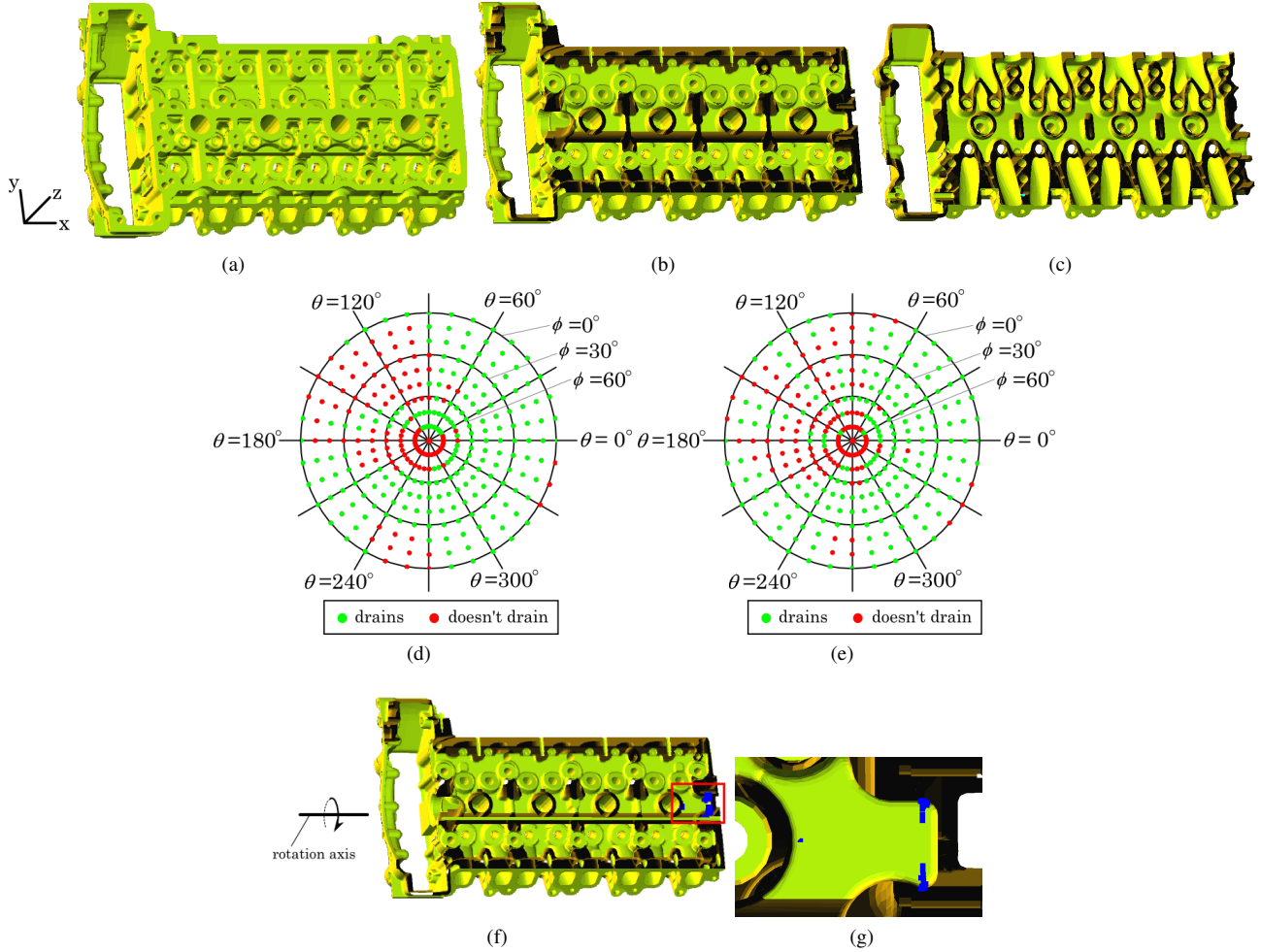


Figure 18: (a) Cylinder head model (b)(c) Cross sections revealing the internal passages of the model shown in (a). (d) Plot of whether or not rotation around a given rotation axis completely drains the workpiece under CW rotation and (e) CCW rotation. (f) The concave vertices (colored blue) such that 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).