CS270 project report:
Find an axis of rotation for drainage of workpieces

Yusuke Yasui

Very interesting problem with A a nice analysis and algorithm.

1 Introduction

When we machine a workpiece by cutting processes, the removed surface of the workpiece forms a chip. To remove such chips from the surface of the workpiece, a water jet is used. The water jet can successfully remove the chips; however, if the geometry of the workpiece has concave regions, the water ejected from the water jet nozzle may be trapped within the workpiece. We would like to drain such trapped water by rotating the workpiece.

Due to the limitation of our device, we are allowed to choose only one rotation axis and to rotate in one orientation around the axis. Our goal is to find a rotation axis for a given part geometry such that when the part is rotated around this axis, all water drains from all voids of the part. As a first step toward this goal, we propose an algorithm to check whether a rotation around a given axis can drain all trapped water or not in this report.

2 Overview

2.1 Observation

Our input geometry is given as a set of triangles i.e. polygonal mesh. A polygonal mesh consists of three kinds of geometric objects, that is, vertices, edges, and triangles. The vertices are further classified into concave vertices, convex vertices, and saddle vertices. The key observation for this problem is that water traps are always caused by concave vertices. Water traps are not caused by triangles or edges because water particles are free to move on them. Among the three types of vertices, only a concave vertex causes a water trap because it can hold water particles while a convex vertex and a saddle vertex cannot. Based on this, our goal is to drain all the concave vertices of an input geometry since it is equivalent to drain from all voids of the part.

2.2 Assumption

We assume that a rotation is slow enough so that water particles move only by gravity force (assuming that any other forces such as friction force are negligible). When we rotate an input geometry from one orientation to another orientation, the water particles move from one concave vertex to another concave vertex (or may go outside of the geometry). We assume that once a trapped water particle starts to take off a current concave vertex, it moves to another concave vertex instantaneously.

2.3 Approach

Based on these observations and assumptions, we construct a graph whose nodes correspond to the concave vertices. The edges of the graph are set between nodes if the trapped water particle at one corresponding concave vertex moves to another corresponding concave vertex in a specific gravity direction. The example in 2D is shown in Figure 1.

For each concave vertex, we set the corresponding node in the graph. Then, we compute the gravity directions when the trapped water starts to flow out and the concave vertices the trapped water flowing out settles. These gravity directions are shown next to each node in Figure 1. As shown in Figure 1, the edges in a graph are labeled as CW or CCW. When we rotate the geometry around the given rotation axis in clockwise (respectively, counterclockwise) rotation, trapped water particles move from one concave vertex to another concave vertex through only the edges labeled as CW (respectively, CCW).

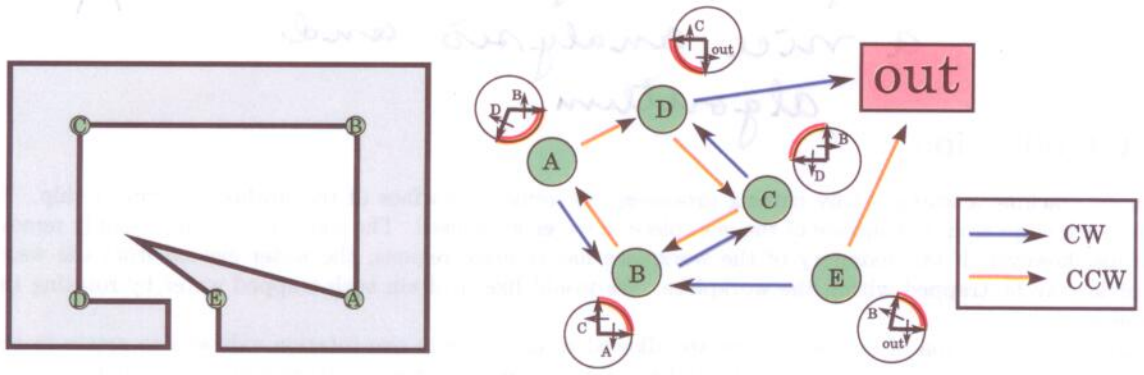

Figure 1: Geometry and the corresponding graph in 2D. The diagram next to each node shows the gravity directions which move water particles trapped at the corresponding concave vertex to the other concave vertices. When a current gravity direction is in the red part bounded by these gravity directions, water particles may be trapped at the corresponding concave vertex.

Figure 2: The actual drainage by a rotation corresponding to the path "$A \to B \to C \to D \to out$" in Figure 1.

For each concave vertex, if there is a path consisting of edges with the specified orientation label from the corresponding node to the node labeled as "outside-of-geometry", it is equivalent to the fact that we can drain the trapped water at the concave vertex through concave vertices corresponding to the nodes on the path by rotating around a given axis. For example, there is a path "$A \to B \to C \to D \to out$" in Figure 1 and this path corresponds to a drainage shown in Figure 2. There is also a path "$A \to D \to out$". But we cannot drain using this path because we need 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 orientation.

Even in 3D, the change of a gravity direction is confined in 2D plane and therefore the similar approach works.

3 Graph Construction

In this section, we describe how to construct a graph from an input geometry.

3.1 Gravity direction causing water trap at a concave vertex

Any gravity direction g can be expressed as a point on a sphere whose radius is one and center is at origin. This sphere is called the *gaussian sphere*.

Let V_c be the set of concave vertices. For each concave vertex $v \in V_c$, let w_i be a member of the adjacent vertices of v and e_i be the vector from v to w_i (i.e. $e_i = w_i - v$) (see Figure 3 (a)). For each e_i, we define a space H_i on the gaussian sphere consisting of points p such that $e_i \cdot (p-v) \leq 0$. (i.e. $H_i = \{p \mid e_i \cdot (p-v) \leq 0, \|p-v\| = 1\}$). Figure 3 (b) shows the specific example. A gravity direction not in H_i drags a water particle to the direction from v to w_i and does not cause the water trap at v. On the other hand, a gravity direction in H_i drags a water particle to the direction from w_i to v and may cause a water trap at v. For example, in Figure 3 (c), gravity direction g_a never causes a water trap at v, but g_b may cause a water trap at v.

We define a space $T_v = \bigcap_i H_i$. Figure 4 shows the specific example of T_v. 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 a graph shown in Figure 1, we would like to know, for each concave vertex $v \in V_c$, in which gravity direction the currently trapped water at v starts to flow out. These gravity directions correspond to the points on the boundary of T_v.

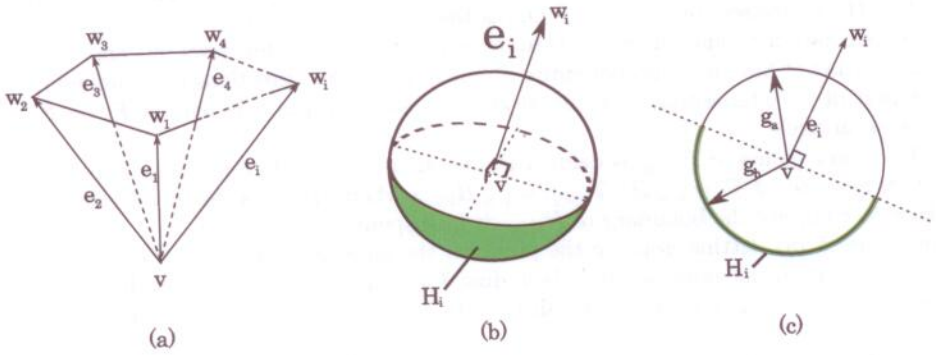

Figure 3: (a) concave vertex v. (b) e_i and the corresponding H_i. (c) Gravity direction g_a never causes a water trap at v, but g_b may cause a water trap at v.

3.2 Which gravity direction makes the trapped water flows out

Given a rotation axis, we first rotate the input geometry 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 a rotation axis are always orthogonal. In this configuration, any gravity direction g can be expressed as a point on a circle in the xy-plane with center $(0,0)$ (i.e. $x^2 + y^2 = 1$). This circle is called the *gaussian circle*, which is the intersection between the gaussian sphere and the xy-plane.

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

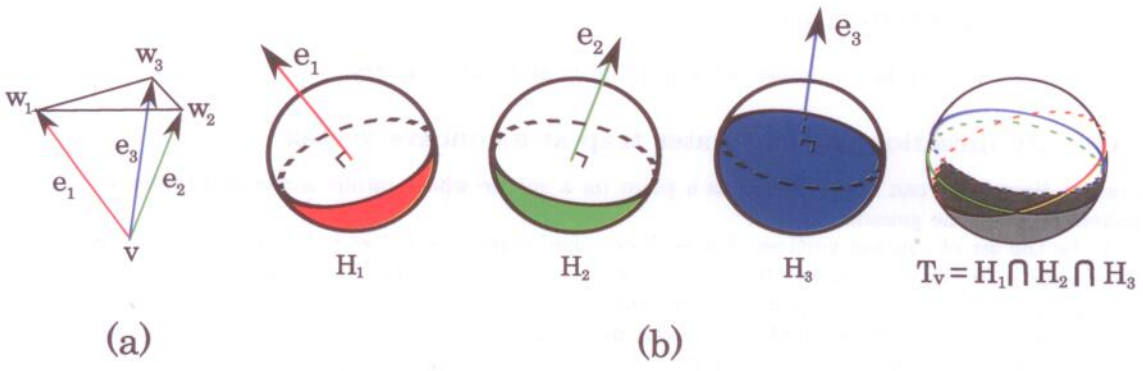

Figure 4: (a) concave vertex v. (b) The corresponding H_i and T_v.

3.2.1 Edge through which water flows out and the corresponding gravity direction

As Figure 4 (b) shows, T_v is bounded by a set of arcs on the gaussian sphere. If the xy-plane intersects with T_v, they intersect at two points because T_v is convex. Each of these two points corresponds to the gravity direction which makes the trapped water at v flows out when we rotate in clockwise and in counterclockwise, respectively.

Since $T_v = \bigcap_i H_i$, each of the arcs is defined by the boundary of H_i. The edge which defines the boundary of H_i where each of these intersection points is found is the one the water 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 entire T_v since the gravity directions are confined in the xy-plane; constructing the part of T_v intersecting with the xy-plane is enough. Let this part of T_v be $T_{v(xy)}$. $T_{v(xy)}$ is defined on the gaussian circle.

Each of the boundary points of $T_{v(xy)}$ is defined by the intersection between the gaussian circle and one of the boundary of H_i because, as $T_v = \bigcap_i H_i$, $T_{v(xy)} = \bigcap_i H_{i(xy)}$ where $H_{i(xy)}$ is the intersection between H_i and the xy-plane. How to compute the boundary of $H_{i(xy)}$ is in Appendix.

As shown in Figure 5 (a), letting g_{CW} be the point on the gaussian circle bounding $T_{v(xy)}$ in a clockwise direction, g_{CCW} the point on the gaussian circle bounding $T_{v(xy)}$ in a counterclockwise direction, and e_{CW} and e_{CCW} be the edge defining g_{CW} and g_{CCW} (recall that H_i is defined by edge e_i), we compute the boundary points of $T_{v(xy)}$ as follow.

Initially, g_{CW} and g_{CCW} are set to the boundary of $H_{1(xy)}$ and e_{CW} and e_{CCW} are set to e_1. Then, for each i ($i = 2, 3, \cdots$, "valence of v"), we update g_{CW} and g_{CCW}, that is, the boundaries of $T_{v(xy)}$, and e_{CW} and e_{CCW} if necessary as follow.

For each i, if both g_{CW} and g_{CCW} are not in $H_{i(xy)}$, $T_{v(xy)}$ is empty (Figure 5 (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 5 (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 5 (d)). If g_{CCW} is not in $H_{i(xy)}$, we set r to g_{CCW} and e_i to e_{CCW} (Figure 5 (e)). After performing this update for each e_i ($i = 2, 3, \cdots$, "valence of v"), g_{CW} and g_{CCW} become the points bounding $T_{v(xy)}$.

When water is trapped at v, if the gravity direction coincides with g_{CW} in a clockwise rotation, the trapped water starts to flow out through edge e_{CW}. In a similar manner, if the gravity direction coincides with g_{CCW} in a counterclockwise rotation, the trapped water starts to flow out through edge e_{CCW}.

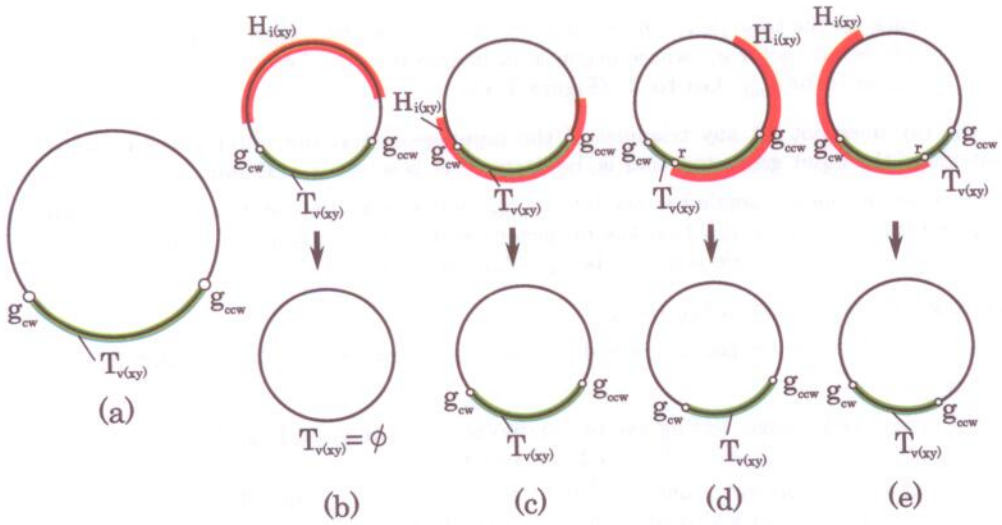

Figure 5: (a) g_{CW}, g_{CCW}, and $T_{v(xy)}$ on the gaussian circle. (b)-(e) How to update g_{CW}, g_{CCW}, and $T_{v(xy)}$ when new $H_{i(xy)}$ is introduced.

3.3 Concave vertex where trapped water flowing out settles

In the previous subsection, for each concave vertex $v \in V_c$, we computed the gravity directions when the trapped water at v starts to flow out and the edges through which the trapped water flows out at this time. In this subsection, we describe how to determine which concave vertex the water flowing out from v settles (or goes outside of the geometry) with these information.

For each concave vertex v, we have to determine which concave vertex the water particle flowing out settles for two cases when we rotate the geometry in clockwise and in counterclockwise. Let the concave vertex where the water particle settles in clockwise rotation be $v_{t(CW)}$ and in counterclockwise rotation be $v_{t(CCW)}$. We only describe the case when we rotate the geometry in clockwise here because the similar procedure works for the counterclockwise case.

3.3.1 Algorithm

For sake of simplicity, we let $v_t \stackrel{def}{=} v_{t(CW)}$ and $g \stackrel{def}{=} g_{CW}$ for a moment.

First, we initialize v_t with the endpoint of e_{CW} which is the opposite side of v. We have assumed that a force applied to a water particle is only gravity force. Then, we trace it as follow.

1.
   - If g is in $T_{v_t(xy)}$, the water particle flowing out from v settles at vertex v_t. **Done**.
   - Otherwise, we compute $n_j \cdot g$ where n_j is a normal vector of incident face j of v_t ($j = 1, 2, \cdots$, "valence of v_t").
     - If $n_j \cdot g > 0$ for at least one j, the water particle falls down parallel to g (Figure 6 (a)). To simulate this, we shoot a ray parallel to g from v_t. **Go to 2**.
     - Otherwise,
       * if there is a valley edge e_k incident to v_t such that $e_k \cdot g > 0$, compute $e_k \cdot g$ for all k and set the endpoint of e_k such that $e_k \cdot g$ becomes the largest to v_t. **Go to 1**. (Figure 6 (b))
       * if there is no such valley edges incident to v_t, the water particle moves on a triangle surface. We project g onto each incident triangle t_j of v_t. Letting the projected vector onto t_j be g_{t_j},

pick up the triangle t_l where $\|g_{t_l}\|$ becomes the largest. Then, we find an edge of t_l that the projected vector g_{t_l} whose origin is v_t intersects with. Let this edge be e_t and the triangle across e_t be t_{adj}. **Go to 3**. (Figure 6 (c))

2.
   - If the ray does not hit any triangles of the input geometry, the water particle falling down goes outside of the input geometry, that is, be drained; v_t is set to *out*. **Done**.
   - Otherwise, letting a triangle the ray hits be t_{hit} and a point the ray hits be p_{hit}, we project g onto t_{hit} and find an edge of t_{hit} that the projected vector whose origin is p_{hit} intersects with. Let this edge be e_t and the triangle across e_t be t_{adj}. **Go to 3**. (Figure 6 (d))

3.
   - Letting a normal vector of t_{adj} be n_{adj},
   - if $n_{adj} \cdot g > 0$, the water particle falls down; we shoot a ray. **Go to 2**. (Figure 6 (e))
   - Otherwise,
     - if e_t is a valley edge, letting the two endpoints of e_t be v_a and v_b, if $(v_a - v_b) \cdot g > 0$, set v_a to v_t; otherwise, set v_b to v_t. **Go to 1**. (Figure 6 (f))
     - Otherwise, we project g onto t_{adj} and find an edge of t_{adj} that the projected vector intersects with. Replace e_t with this edge and t_{adj} with the triangle across this edge. **Go to 3**. (Figure 6 (g))

We repeat this series of the procedures for each v until we can find v_t such that g is in $T_{v_t(xy)}$ or v_t is set to *out*.

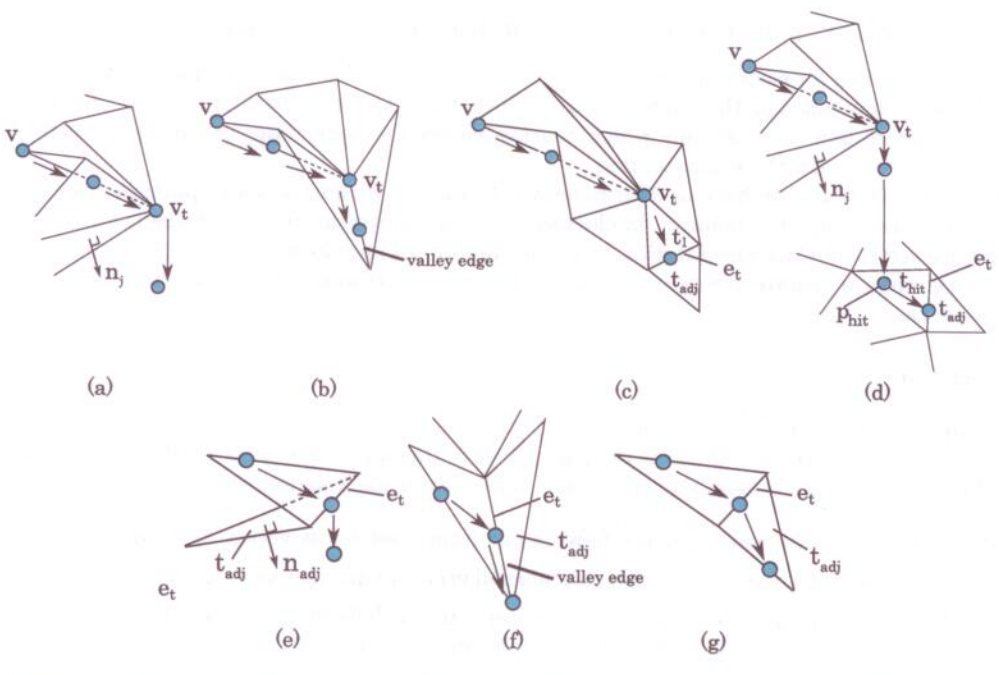

Figure 6: Movement of a water particle by gravity force on various geometric shapes.

3.4 Graph completes

For each concave vertex $v \in V_c$, we set the corresponding node n_v in the graph. Then, we connect the node n_v and the node corresponding to $v_{t(CW)}$ by an edge and label the edge as CW. Also, we connect the node n_v and the node corresponding to $v_{t(CCW)}$ by an edge and label the edge as CCW. Figure 1 is the specific example.

4 Checking

Now, using the graph constructed, we test whether 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 *out* node in the graph, we can drain water trapped at v by rotating the input geometry around a given rotation axis. Note that when we rotate the geometry in clockwise, we can use only edges labeled as CW, and when we rotate in counterclockwise, we can use only edges labeled as CCW.

4.1 Checking procedure

Letting the number of concave vertices be $n = |V_c|$, if we take a naive approach, we have to trace n nodes for each concave vertex $v \in V_c$ in the worst case. Therefore, the total running time becomes $O(n^n)$. However, we can notice that if there is a path from one node to *out* node (let this path be p), it means that there is also a path from the intermediate nodes on path p to *out* node. For example, in Figure 1, if we find a path from A to *out* through B, C, and D, we can know that there is also a path B, C, and D to *out* node.

Based on this observation, we can improve this by taking the following procedure. Suppose we rotate the geometry in clockwise orientation. Then, trapped water 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. Then, trapped water at concave vertices whose corresponding nodes are directly connected to the nodes in S_n by edges labeled as CW can be drained as well. We put these nodes to S_n and do the same thing 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 at all of the concave vertices is completely drained by a rotation around the given rotation axis. In this approach, we do not have to check the same node twice. Therefore, the time complexity becomes $O(n)$

5 Discussion

We showed the algorithm to check whether we can drain trapped water by rotating around a given rotation axis. As described in the introduction, our ultimate goal is to find a rotation axis for a given geometry such that when the part is rotated around this axis, all water drains. We can approximately achieve this by testing infinitely many rotation axes using the presented algorithm. Among the set of rotation axes tested, if there is a rotation axis that can drain all trapped water, the rotation axis is what we would like to obtain.

For this, we would like to test as many rotation axes as possible to find an answer. Therefore, our testing algorithm should run fast. We now analyze the performance 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 the number of operations equal to the number of edges incident to v. Assuming that the number of incident edges for each concave vertex is constant (generally, this is 6 ± 3), this is done in $O(n)$ (n is a number of concave vertices in the geometry). Then next, for each concave vertex $v \in V_c$, we find a concave vertex where the trapped water flowing out from v settles. In theory, for each v, we have to check all triangles and vertices of the geometry to find it in the worst case. In practice, we are still under investigation; however, by 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 the

space, we can expect that the number of the vertices and triangles checked is less than 10 for almost all cases. Once the graph is constructed, the checking phase runs in $O(n)$ as described in the previous section.

6 Conclusion

In this report, we presented the algorithm to check whether a rotation around a given rotation axis can drain an input geometry. Using this, we can also find a rotation axis to drain a given geometry with a heuristic approach. To make this approach be a reliable one, we would like to test as many rotation axes as possible. Therefore, our testing algorithm must run fast. For now, the bottleneck is the particle tracing operation in the graph construction phase. If we can guarantee that this operation runs fast, our approach becomes more practical.

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 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,

$$(e_i)_x(I_x) + (e_i)_y(I_y) + (e_i)_z(0) = 0 \;\Rightarrow\; I_x = -\frac{(e_i)_y}{(e_i)_x} I_y$$

Substituting this into ${I_x}^2 + {I_y}^2 = 1$,

$$(\frac{(e_i)_y}{(e_i)_x})^2 {I_y}^2 + {I_y}^2 = 1 \;\Rightarrow\; (\frac{(e_i)_y}{(e_i)_x})^2 + 1){I_y}^2 = 1 \;\Rightarrow\; I_y = \pm\sqrt{\frac{1}{(\frac{(e_i)_y}{(e_i)_x})^2 + 1}}$$

Note that the boundary of H_i and the gaussian circle intersect at two points.