Journal of Manufacturing Systems 37 (2015) 494–504

Contents lists available at ScienceDirect

Journal of Manufacturing Systems

journal homepage: www.elsevier.com/locate/jmansys

Technical Paper

Pool segmentation for predicting water traps

Yusuke Yasui [a], Sara McMains [a,*], Thomas Glau [b]

[a] University of California, Berkeley, USA
[b] Daimler AG Research & Development, Stuttgart, Germany

ARTICLE INFO

Article history:
Received 8 July 2014
Accepted 14 July 2014
Available online 9 September 2014

Keywords:
Waterjet cleaning
Reeb graph
Water traps
Cleanability
Slicing
Segmentation
Computational geometry

ABSTRACT

We propose a new method to detect water trap regions in voids of oriented polygonal models that approximate the geometry of mechanical parts. Since water traps decrease cleaning and draining efficiency, accurately predicting such regions allows re-orienting parts to reduce manufacturing time and cost. We construct a directed graph that captures the flow of water in voids of a 3D input model, based on a fast orientation-dependent volume segmentation approach. We can quickly find the water trap regions by analyzing the directed graph. Since we take a purely geometric approach to solve this problem without employing any physical simulation, even if the geometry of the voids is complicated, we can calculate water traps quickly.

1. Introduction

As the complexity and precision of mechanical parts and assemblies have increased, the possibility of in-service failures caused by manufacturing-related hard particle contamination (such as detached burrs and chips from machining) has increased considerably. Reliably removing solid particle contaminants from the surfaces of mechanical parts has become increasingly important in the automotive industry. However, miniaturization and increased geometric complexity has made it more difficult to access all the surfaces of parts to remove contaminants.

In this paper, we consider a manufacturing planning problem that arises when cleaning with high-pressure water jets. Water jets are effective for removing contaminants from the surface of mechanical parts, but the water may become trapped inside the part if the geometry of voids is complex. Since contaminants may accumulate in such regions and trapped water must be drained after the cleaning, finding an orientation that minimizes the potential water trap regions is important to increase the cleaning efficiency and reduce the draining time and effort after cleaning.

We propose a new method to pre-identify the regions of cleaning-incompatible water traps in voids of mechanical parts using a geometric volume segmentation method, given the part orientation. We assume that the part geometry is given as a 2-manifold triangulated polygonal mesh and that the draining force applied to the water is only the gravitational force.

2. Previous work

To increase the efficiency of cleaning processes, analytical tools that predict cleaning effectiveness at the design and process planning stages are needed. Initial research has focused on understanding the effect of key cleaning process parameters [3,2].

Since simulating multiphase fluid flow using computational fluid dynamics (CFD) can take hours to converge, some purely geometric approaches have been proposed for manufacturing planning with fluids. Bose and Toussaint proposed an algorithm to find an orientation for a gravity casting mold that minimizes the number of venting holes that need to be added to allow air to escape to insure a complete fill [4]. Aloupis et al. solved a 2D rotational draining problem for a closed polygon and a trapped single particle inside of the polygon, proposing an algorithm to find how many holes must be punctured to "drain" the particle [1]. For industrial applications of cleaning, on the other hand, we typically do not have the option of modifying the part by adding venting or draining holes to eliminate air or water traps. Yasui and McMains proposed an algorithm to test whether a given rotation axis can fully drain a workpiece when the workpiece is rotated around the axis [10]. The method we propose in this paper is also purely geometric, avoiding the computationally costly simulations of CFD.

* Corresponding author. Tel.: +1 510 852 9359.
E-mail address: mcmains@me.berkeley.edu (S. McMains).

http://dx.doi.org/10.1016/j.jmsy.2014.07.006
0278-6125/

3. Algorithm overview

Our algorithm predicts regions in voids of the geometry where, for a given orientation, the effectiveness of cleaning with water jets will be compromised due to water traps. We assume throughout this paper that the part geometry has been rotated to the desired test orientation, so that gravity always acts vertically (i.e. down the z-axis).

Fig. 1 illustrates an overview of our algorithm. Letting $\mathcal{M}$ be the geometry of the input model and $\mathcal{B}$ be a slightly enlarged bounding box that encloses $\mathcal{M}$, the space $\mathcal{W}$ where water flows can be represented as $\mathcal{W} = \mathcal{B} \setminus \mathcal{M}$. We split the space $\mathcal{W}$ horizontally into multiple regions called *pools* based on topological changes of $\mathcal{W}$ with respect to the z-axis. Then, we build a directed graph whose nodes correspond to the pools and whose edges connect two nodes if water flowing out of the source node's corresponding pool could enter the destination node's corresponding pool. We determine water trap regions by analyzing the directed graph.

The directed graph we construct is mathematically equivalent to a Reeb graph of a 3-manifold with boundary with respect to the height function (z-value). Hence, we could construct the directed graph from $\mathcal{W}$ using a Reeb graph construction algorithm [8,9] and segment $\mathcal{W}$ into pools based on the Reeb graph constructed. However, since those Reeb graph construction approaches require the extra burden of tetrahedralizing $\mathcal{W}$, we propose an alternative efficient approach of segmenting $\mathcal{W}$ into pools and constructing the corresponding directed graph simultaneously in our work.

3.1. Preliminaries

We introduce some notation that we will use to explain how we split $\mathcal{W}$ into pools and add the directed edges between nodes corresponding to pools. We consider a sweep plane $p_{sweep}(z)$ perpendicular to the z-axis (i.e. the gravity direction) intersecting it at z. Given a sweep plane $p_{sweep}(z)$, we define the *slice* at z, $S(z)$, as the intersection of $\mathcal{W}$ and $p_{sweep}(z)$: $S(z) = \mathcal{W} \cap p_{sweep}(z)$. As shown in Fig. 1(d), slice $S(z)$ may consist of multiple disconnected slice components, which in 3D will be 2D polygons (possibly with holes). We call these *slice polygons*. We denote the different slice polygon constituting $S(z)$ as $s_i(z)$ $(1 \le i \le |S(z)|)$.

Then, we let $proj(s_i(z))$ be the projection of $s_i(z)$ to the plane perpendicular to the z-axis, and the z-value just below z be $z^- = z - \epsilon$. We let the z-value just above z be $z^+ = z + \epsilon$, ϵ a positive infinitesimal number. Given a slice polygon $s_i(z) \in S(z)$, we define overlapping slice polygon(s) just below $s_i(z)$, $S_{below}(s_i(z))$, as the set of slice polygons $s_j(z^-) \in S(z^-)$ such that $proj(s_i(z)) \cap proj(s_j(z^-)) \neq \emptyset$. Similarly, we define overlapping slice polygon(s) just above $s_i(z)$, $S_{above}(s_i(z))$, as the set of slice polygons $s_j(z^+) \in S(z^+)$ such that $proj(s_i(z)) \cap proj(s_j(z^+)) \neq \emptyset$.

Based on the cardinality of $S_{below}(s_i(z))$ and $S_{above}(s_i(z))$, the slice polygons just below and above $s_i(z)$, we classify each slice polygon $s_i(z)$ as one of four types as follows. Given a slice polygon $s_i(z)$, if $|S_{below}(s_i(z))| = 0$, we call $s_i(z)$ a *beginning slice polygon* since a new slice polygon appears as the sweep plane moves from $p_{sweep}(z^-)$ to $p_{sweep}(z^+)$. On the other hand, if $|S_{above}(s_i(z))| = 0$, we call $s_i(z)$ an *ending slice polygon*, since an existing slice polygon disappears as the sweep plane moves from $p_{sweep}(z^-)$ to $p_{sweep}(z^+)$. If $|S_{below}(s_i(z))| \ge 2$ and $|S_{above}(s_i(z))| \ge 1$, or if $|S_{below}(s_i(z))| \ge 1$ and $|S_{above}(s_i(z))| \ge 2$, we call $s_i(z)$ a *merge/split slice polygon* since multiple slice polygons merge into one slice polygon and/or one slice polygon splits into multiple slice polygons as the sweep plane moves from $p_{sweep}(z^-)$ to $p_{sweep}(z^+)$. Finally, if $|S_{below}(s_i(z))| = |S_{above}(s_i(z))| = 1$, we call $s_i(z)$ a *no-change slice polygon*, since no topological change of slice polygon $s_i(z)$ occurs as the sweep plane moves from $p_{sweep}(z^-)$ to $p_{sweep}(z^+)$.

3.2. Pool segmentation overview

We define a pool as the union of no-change slice polygons bounded by either a beginning or a merge/split slice polygon from below and either an ending or a merge/split slice polygon from above. Given a slice polygon $s_i(z)$, we let $pool(s_i(z))$ be the pool $s_i(z)$ defines.

We segment $\mathcal{W}$ into pools using a sweep plane algorithm, where we imagine moving $p_{sweep}(z)$ from $z = -\infty$ to $z = +\infty$. If $\mathcal{W} \cap p_{sweep}(z)$ yields a beginning slice polygon, we generate a new pool bounded from below by the beginning slice polygon. If $\mathcal{W} \cap p_{sweep}(z)$ yields a no-change slice polygon, the no-change slice polygon $s_i(z)$ defines the pool $pool(s_j(z^-))$ where $s_j(z^-) \in S_{below}(s_i(z))$. If $\mathcal{W} \cap p_{sweep}(z)$ yields an ending slice polygon, we complete the corresponding existing pool, bounding it from above with the ending slice polygon. Finally, if $\mathcal{W} \cap p_{sweep}(z)$ yields a merge/split slice polygon, we complete the corresponding existing pool(s) by bounding from above with the merge/split slice polygon, and generate new pool(s) by bounding from below with the same merge/split slice polygon. Then, for $1 \le i \le |S(z^-)|$ and for $1 \le j \le |S(z^+)|$, we compute $proj(s_i(z^-)) \cap proj(s_j(z^+))$. If there are p and q such that $proj(s_p(z^-)) \cap proj(s_q(z^+)) \neq \emptyset$, and $pool(s_p(z^-)) \neq pool(s_q(z^+))$, we add a directed edge from the node corresponding to $pool(s_q(z^+))$ to the node corresponding to $pool(s_p(z^-))$ in the directed graph (Fig. 1(f)).

3.3. Predicting water trap regions

After completing the sweep from $z = -\infty$ to $z = +\infty$, the space $\mathcal{W}$ is segmented into pools that are connected to each other in the graph by edges oriented in the direction of gravity if they are bounded by the same merge/split slice polygon. Each pool represents a region that could potentially be a water trap region (except the bottom-most pool, which represents the exterior of $\mathcal{M}$). Water flowing in $\mathcal{W}$ under gravity will flow between pools according to the directed edges. Once such flowing water reaches the bottom-most pool, since by construction it is outside the input geometry, we consider the water to be drained. Thus, as shown in Fig. 1(g), given a pool, if there is no path such that we can reach the bottom-most pool from the corresponding node, the pool is a potential water trap region (whether or not this water trap is actually formed depends upon the inflow location). Since we can compute the volume of water each pool can hold, we can also quantitatively evaluate a given part orientation by summing the volumes of pools that are determined to be water trap regions.

4. Pool segmentation

In this section, we describe the details of our pool segmentation algorithm summarized above, given a 2-manifold triangulated input mesh $\mathcal{M}$.

From $\mathcal{M}$, we can easily obtain the corresponding $\mathcal{W}$ by flipping the orientation of the triangles in $\mathcal{M}$ and introducing six rectangles that represent the enlarged axis-aligned bounding box $\mathcal{B}$. Each rectangle should be split into two triangles such that all the faces of $\mathcal{W}$ are represented by triangles as well.

To implement the pool segmentation algorithm, we have to know for which values of z beginning, ending, and merge/split slice polygons occur. We determine all of these values of z by tracking the evolution of the boundary of slice polygons. Even when a slice polygon boundary appears, disappears, merges, or splits, the corresponding slice polygon does not necessarily appear, disappear, merge, or split (e.g. because the boundary could correspond to a hole in a polygon). However, when a slice polygon appears, disappears, merges, or splits, the corresponding slice polygon boundary