Control Sequence Generation for 2D Rotational Draining

by

Peter Cottle

A thesis submitted in partial satisfaction of the
requirements for the degree of
Masters of Science

in

Mechanical Engineering

in the

Graduate Division

of the

University of California, Berkeley, California

Fall 2012

The thesis of Peter Cottle, titled Control Sequence Generation for 2D Rotational Draining, is approved:

Chair	______________________________	Date	____________
	______________________________	Date	____________
	______________________________	Date	____________

University of California, Berkeley, California

Control Sequence Generation for 2D Rotational Draining

Abstract

Control Sequence Generation for 2D Rotational Draining

by

Peter Cottle

Masters of Science in Mechanical Engineering

University of California, Berkeley, California

Professor Sara McMains, Chair

This paper provides several contributions related to the workpiece drainability problem in manufacturing. First we describe our parametric water particle model that uses parabolic path segments. This improvement allows for particles to maintain velocity, travel through rotating acceleration fields, and collide with workpiece edges while maintaining constant-time intersection tests per geometric primitive.

Next we provide a framework for approaching the drainability problem from an artificial intelligence perspective. We define our state space formulation, the size of which is linear with geometric features and exponential with the number of particles being drained. We then define our sampling approach to the successor function; this sampling approach allows us to discover the majority of connectivity information while only exploring a subset of the action space.

Finally, we describe how our algorithm searches for a solution to determine if a given workpiece is drainable. The primary result produced by our work is a full sequence of rotation angles throughout time to drain the workpiece; this represents a direct control sequence that can be fed into fixture rotator.

Keywords: workpiece drainability, parametric equations, fluid simulation

Contents

List of Figures

List of Tables

Acknowledgments

To everyone who helped me along this journey.

Chapter 1

Introduction & Motivation

During the manufacturing process for engine components, many byproducts are produced that contaminate the final workpiece [5]. These byproducts can include sand from sand casting, chips from additional machining, and burrs from hole-cutting [2]. These byproducts are commonly cleaned off with high-pressure water-jets; while these jets are effective at removing contaminants, they introduce large volumes of water that may settle in concave regions of the workpiece [2]. This fluid must be removed before the final stages of the manufacturing process can continue; the removal of this fluid is a non-trivial task due to complex workpiece geometry [3] [11]. Rotation is commonly used to remove this fluid in industry; the workpiece is held in a fixture that is rotated about an axis, allowing for the fluid to drain out of the workpiece. Determining how to drain this fluid from the workpiece is referred to as the "workpiece drainability" problem.

Manufacturers and designers have two key objectives to accomplish when presented with the drainability problem; the first is to analyze the "drainability" of a workpiece, and the second is to produce a specific sequence of rotations to actually drain the workpiece.

Drainability is defined as the ability for the workpiece to be fully drained by rotation, usually in reference to a given rotation axis. Drainability analysis takes in a workpiece geometry and a rotation axis; it produces a binary value representing if rotation about that axis would fully drain the workpiece.

Existing work has made great progress in this domain; Yasui et al.'s work is capable of producing a map of which rotation axes will fully drain the workpiece under an infinite number of rotations [11]. These results can be quite useful when designing rotation fixtures or altering the geometry of a workpiece, despite the algorithm using a simplified model of true fluid behavior.

While the drainability analysis for a workpiece is useful, it does not produce any bounds on the time required to drain the workpiece. Likewise, it also does not specify a rotation speed or total rotation angle. For this reason, manufacturers would also like to accomplish the second objective of the drainability problem – achieving a specific sequence of rotations that fully drains the workpiece under a given rotation axis.

This sequence of rotations would be immediately useful on the manufacturing floor to input as a control sequence to a workpiece rotator. Existing work has focused on analyzing drainability [11] or the number of additional drain holes needed to attain drainability [1]. Consequently, the focus of our work presented in this paper is to produce a specific sequence of rotations for drainability. If both halves of the drainability problem can be solved, the combined solution would serve great utility in reducing time and energy required to manufacture a given part.

Chapter 2

Modeling Approach

Solving either objective of the drainability problem requires that the behavior of fluid as it travels through the workpiece be modeled in some way. This behavior is complex; trapped gases, viscous effects, and fluid mechanics all interact during workpiece draining. There are many ways to model this fluid behavior, and each choice comes with both advantages and disadvantages.

Advanced fluid simulation methods like computational fluid dynamics or smoothed-particle hydrodynamics approximate fluid behavior with high accuracy, but their computation is quite expensive in both time and memory. This is primarily because both are integration methods – their accuracy relies on integrating many times over a small timestep.

Drainability analysis requires many different rounds of fluid simulation in many different configurations; consequently, it is worthwhile to sacrifice accuracy in the fluid model to obtain the ability to simulate many times during algorithm execution. Yasui et al.'s existing work chooses this exact tradeoff and produces meaningful and worthwhile results [11]. We choose a similar approach of a simplified fluid model that is used in many different simulations.

2.1 Straight-Line Multiple-Particle Model

Core Assumptions

Yasui et al. [11] chose a simplified particle modeling approach with the following core assumptions:

- each fluid body within the workpiece is approximated as a single particle,
- water particles only travel in straight lines and do not maintain a kinetic state,
- the workpiece rotates infinitesimally slow in one direction for an infinite amount of time.

Modeling Choice Advantages and Disadvantages

This modeling approach allowed Yasui et al. to reduce the fluid draining problem down to a particle draining problem. Furthermore, particle simulation within the workpiece is greatly simplified; particles always travel in straight lines and do not accumulate kinetic energy. This means that particles are always traveling either parallel to gravity or along a workpiece edge.

Consequently, particle simulation in this choice of model was essentially reduced to finding intersections between lines and surfaces. An intersection test between a line and a surface runs in constant time; consequently, intersecting a line with the entire workpiece is linear in the number of triangles. Yasui et al. used bounding-box methods to reduce this complexity to sub-linear in the number of triangles, meaning each intersection test (and particle simulation as a whole) was then computationally inexpensive [11].

Optimizing particle simulation gave Yasui et al. the ability to perform many particle simulations during analysis; the end result is that for every possible rotation axis, a particle is simulated out of every concave vertex in the workpiece [11].

This thoroughness of simulation produces a set of results that describes the workpiece drainability of any rotation axis. This analysis serves great utility when designing for manufacturability; engineers can quickly determine if a geometry change affects drainability of the workpiece before the workpiece is even manufactured.

Yasui et al.'s model assumes infinitesimally slow rotation of the workpiece for an infinite amount of time. Because of the assumption about the workpiece rotation, Yasui et al.'s results produce a binary value of drainability about a particular axis rather than a direct control sequence.

If this assumption about the rotation of the workpiece could be relaxed and a control sequence of how to drain the workpiece could be produced, manufacturers could further optimize the draining stage of product manufacturing.

This work aims to produce those complementary results while simultaneously improving the kinetic model of fluid behavior.

2.2 Parabolic Single-Particle Model

Our work utilizes a different (but related) model of the workpiece draining problem. The results produced are complementary to Yasui et al.'s results and are ideally used in conjunction.

This model is based on the following assumptions:

- each fluid body within the workpiece is approximated as a single particle,
- water particles travel in parabolic paths that are influenced by acceleration,
- the workpiece can be rotated in either direction along its rotation axis,

- the workpiece can alternate between rotating for any amount of time or holding a specific orientation for any amount of time,
- the workpiece is a two dimensional polygon.

These simplifying assumptions define the model that we use. Additionally, one more restriction is imposed due to simulation constraints:

- particles may not travel through open space during workpiece rotation.

The reasons for this additional assumption are detailed in Chapter 3.

In the following chapters, we first present our methods assuming a single particle will be drained for simplicity of presentation. We then extend this work to multiple particles in Chapter 5.

Omitted Kinetic Terms

It is important to note that our particle model (described further in Section 3.1) is based solely off of acceleration, velocity, and position. The following more complex kinetic terms (among others) are omitted from our model:

- aerodynamic drag,
- rotational velocity about the particle's center of mass,
- velocity imparted from workpiece rotation, and
- the Coriolis effect.

These more complex kinetic terms would affect particle trajectory if included; incorporating these terms, however, can jeopardize theoretical and performance-related qualities of our approach. Consequently, we choose to omit these terms for the same reasons as Yasui et. al [11].

Modeling Choice Advantages and Disadvantages

This modeling choice requires a different approach to drainability analysis than the approach used by Yasui et al.; our approach, however, draws inspiration from their previous work. Our different modeling choice allows us to produce a specific sequence of rotations to drain the workpiece (solving the second objective); the existence of this sequence of rotations then answers the drainability query (the first objective).

The key components of our approach break down into the water particle simulation (presented in Chapter 3) and the search for a drainability sequence (presented in Chapter 4).

2.3 Algorithm Inputs and Outputs

Before proceeding, it is important to formally define the inputs to the draining problem and the outputs that our work produces.

The draining problem is defined as removing bodies of fluid from a given workpiece. Consequently, our work accepts the following variables as input:

- a collection of 2D polygons that represent the workpiece,
- a set of concave vertices representing the bodies of water to drain.

Our work also accepts the desired rotation speed of the workpiece as an input. While this value could theoretically vary in time, we decide to use an algorithm-wide constant for simplicity.

Our work then produces the following two results:

- whether or not a solution exists (answering the "drainability" query),
- a C0 continuous function of rotation angle through time. If the workpiece is rotated according to this function, the input particles will be drained. This function is referred to as the "control sequence".

Figure 2.1 illustrates the control sequence that is generated from our work.

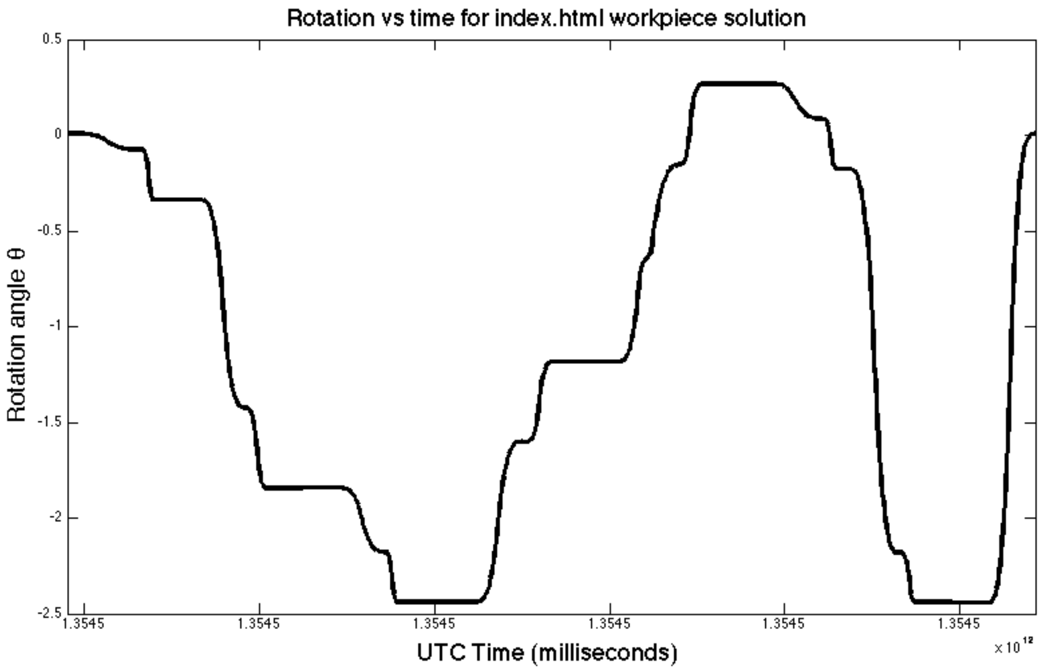

Figure 2.1: The C0 continuous function of rotation angle through time that will drain the input particles. This control sequence is the solution to one of the example geometries provided in the results chapter (Chapter 6)

The following chapters present how these results are obtained.

Chapter 3

Physical Simulation of Water Particles

Water particle simulation is the underlying core of drainability analysis – the behavior of the fluid (or water particles in this case) must be simulated in order to analyze when and where the fluid can leave the workpiece.

Here we present the approach to simulating straight-line water particle motion based on parametric rays (the approach used in Yasui et al.'s work) and then our modification of this approach to fit parabolic water particles.

3.1 Reduced Ray Approach

Underlying Kinetic Model

Individual water particles under an acceleration field a with initial position x_0 and initial velocity v_0 can be modeled with a simple kinetic equation.

$$x(t) = x_0 + v_0 \cdot t + \frac{1}{2} a \cdot t^2 \tag{3.1}$$

Equation (3.1) shows the basic kinematic model of a water particle.

Reduction to Rays

In order to obtain particles that always travel in straight lines, one may omit the acceleration term from Equation (3.1). This is the approach that Yasui et al. chose to obtain straight-line particle motion [11]. Under this condition, these particles can effectively be modeled as "rays."

$$x(t) = x_0 + v_0 \cdot t \tag{3.2}$$

Equation (3.2) shows this simplified ray model. Once the motion of a particle under unobstructed movement can be easily produced, the primary challenge of particle simulation is finding the collision points of a particle's path.

Parametric Equations (rays)

There are many ways to find these intersection (or “collision”) points. One common approach is to model the ray equation as a parametric equation [4] [10].

$$\vec{x}(t) = \vec{x}_0 + \vec{v}_0 \cdot t \tag{3.3}$$

This modeling choice is shown in Equation (3.3) where $\vec{x}(t)$ describes the location of the particle at time t.

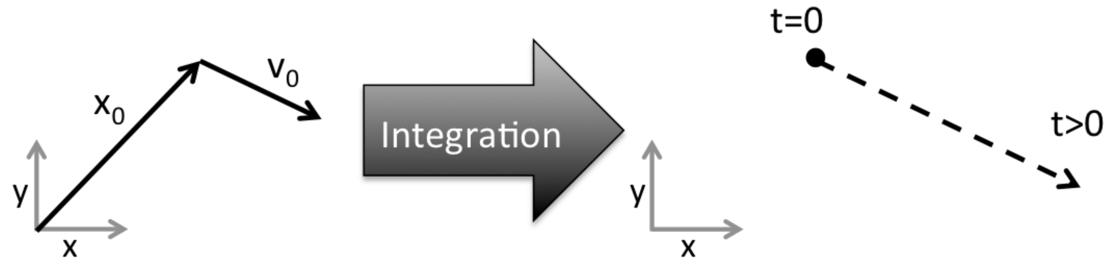

Figure 3.1: A visual depiction of a ray’s component vectors and its path through time

Ray Tracing

Once particles are modeled as parametric rays, all the existing techniques and libraries from “ray-tracing” (a standard approach to producing 3D computer graphics) can be used to find intersection points [4] [10].

Ray-tracing produces 3D computer graphics by sending out rays from a camera location. If one of these rays intersects a geometric primitive in the scene, the resulting color of that ray is calculated and stored in a pixel table. These ray-primitive intersections are the fundamental computational bottleneck for ray tracing; consequently, much work has been done to optimize intersection tests. Parametric modeling of geometric primitives combined with parametric rays is the industry standard [4] [10].

3.2 Kinetic Parametric Approach

While this straight-line particle approach is convenient, our model aims to examine particles with more realistic kinetic properties and paths. Consequently, we choose to not omit the acceleration term from the kinetic model of a particle. This means we combine a parametric approach with Equation (3.1) to produce Equation (3.4) – a parabola defined parametrically.

$$\vec{x}(t) = \vec{x}_0 + \vec{v}_0 \cdot t + \frac{1}{2}\vec{a} \cdot t^2 \tag{3.4}$$

Geometric Primitive Intersections

We would like to intersect these particle paths (represented by parabolas) with geometric primitives in the workpiece for simulation purposes. Since our algorithm takes in a set of polygons that define the workpiece, the only geometric primitive of interest is a polygon edge. We will first describe the parabola-line intersection test and then present the small amount of extra validation needed for the parabola-edge intersection test.

We begin by redefining the parabola's initial point $\vec{x}_0$ as $\vec{p}$ and initial velocity $\vec{v}_0$ as $\vec{v}$ for notation purposes:

$$\vec{x}(t) = \vec{p} + \vec{v} \cdot t + \frac{1}{2}\vec{a} \cdot t^2.$$

This gives us a parabola defined by three vectors:

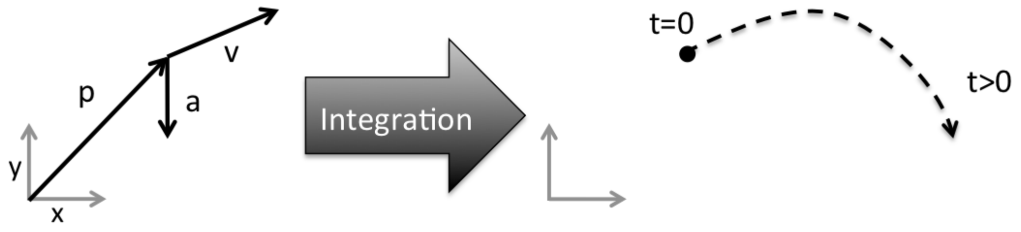

Figure 3.2: The parabola redefined for notation purposes.

This parabola needs to be intersected against an arbitrary edge with starting point $\vec{s}$, direction vector $\vec{d}$, and length $|\vec{d}|$:

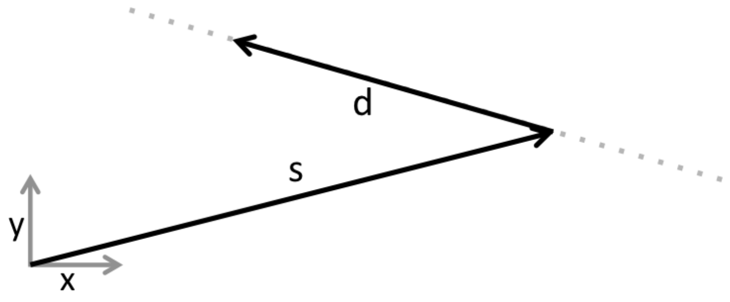

Figure 3.3: Edge for intersection test.

We know that any point α on the line defined by $\vec{d}$ and $\vec{s}$ has a cross product of 0:

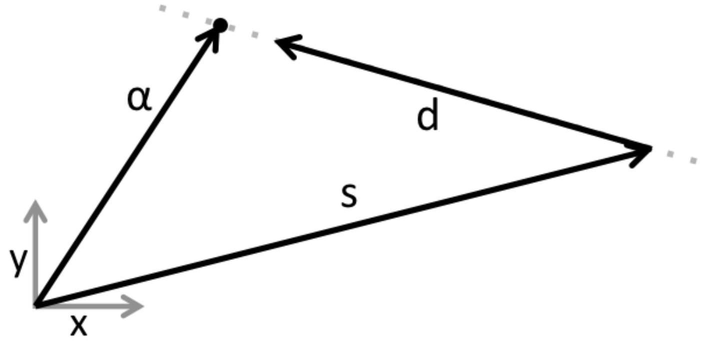

Figure 3.4: Definition of a point on the line defined by $\vec{d}$ and $\vec{s}$.

This can be represented algebraically as

$$(\vec{\alpha} - \vec{s}) \times \vec{d} = 0.$$

Knowing this, we can substitute our definition of the particle's path through time into this cross product:

$$(\vec{x}(t) - \vec{s}) \times \vec{d} = 0.$$

We can then be expanded algebraically into:

$$(\vec{p} + \vec{v} \cdot t + \frac{1}{2}\vec{a} \cdot t^2 - \vec{s}) \times \vec{d} = 0.$$

In order to obtain a non-vectorized equation, we then break each vector into orthogonal components as shown in Equation (3.5).

$$\vec{s} = (s_x, s_y) \qquad \vec{d} = (d_x, d_y)$$

$$\vec{p} = (p_x, p_y) \qquad \vec{v} = (v_x, v_y) \qquad \vec{a} = (a_x, a_y) \tag{3.5}$$

This gives us two versions of the above equation, where the cross product operation is replaced by simple multiplication:

$$\left[(p_x - s_x) + v_x t + \frac{1}{2}a_x t^2)\right] \cdot d_y = 0$$

$$\left[(p_y - s_y) + v_y t + \frac{1}{2}a_y t^2)\right] \cdot d_x = 0.$$

Subtracting the second equation from the first to perform the cross product and grouping terms by the power of t gives us:

$$(\frac{1}{2}a_x d_y - \frac{1}{2}a_y) \cdot t^2 + (v_x d_y - v_y d_x) \cdot t + [(p_x - s_x)d_y - (p_y - s_y)d_x] = 0 \tag{3.6}$$

Equation (3.6) can easily solved by the quadratic equation:

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a} = t.$$

Equation (3.7) defines variables a, b, and c.

$$a = (\frac{1}{2}a_x d_y - \frac{1}{2}a_y d_x) \qquad b = (v_x d_y - v_y d_x) \qquad c = [(p_x - s_x)d_y - (p_y - s_y)d_x] \tag{3.7}$$

Solving for t will yield two values, representing points along the parabola path that intersect the line defined by $\vec{d}$ and $\vec{s}$:

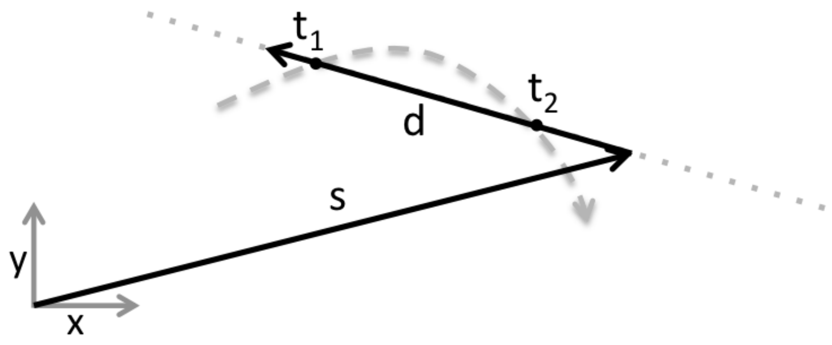

Figure 3.5: Solution points for the parabola-line intersection test.

This concludes the parabola-line intersection tests. For our simulation purposes, we would like to determine the first point that lies on the edge defined by $\vec{d}$ and $\vec{s}$ rather than the line. This requires a small extra validation step; first, negative values of t are eliminated, for these points represent the particle traveling backwards in time. Then, the smallest value of t is found that produces a point that lies in between points $\vec{s}$ and $\vec{s} + \vec{d}$. If no such value exists, this parabola does not intersect the given edge.

3.3 Equation Types

Now that we have defined a parabolic path segment in parametric form and demonstrated a geometric primitive intersection test, we would like to use this equation to simulate particle movement over time. During the course of simulation however, a particle may be traveling in a variety of conditions that may require modifications to Equation (3.4). The two primary conditions that determine the modifications necessary are:

- whether the workpiece is rotating,
- whether the particle is traveling along a workpiece edge or through open space.

These two conditions, when combined in all possible ways, produce four scenarios of particle simulation that must be modeled. The following sections cover each scenario and detail the equation used.

3.4 No Rotation

We first examine the two scenarios where no rotation of the workpiece occurs during simulation.

Scenario #1: Free Fall Equation

When the particle is moving through open space with no workpiece rotation, we consider that particle to be in basic "free-fall." The particle travels through empty space subject to a constant acceleration field. The resulting motion is a parabola, and no modification to Equation (3.4) is necessary.

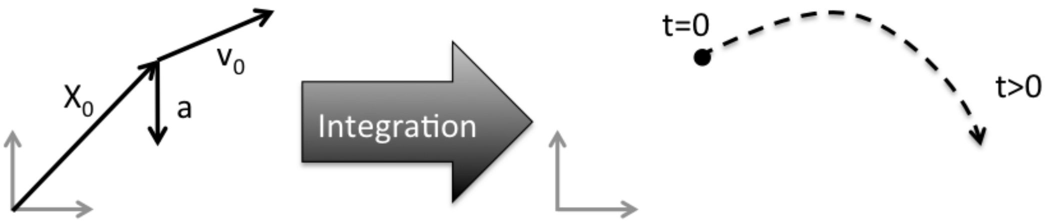

Figure 3.6: Example path traced out by the "free-fall" equation

Scenario #2: Sliding Equation

During simulation, a particle may also begin to travel along an edge. This usually occurs after a particle collides with an edge multiple times in a row and obtains a velocity that is parallel to the edge. Our simulation algorithm performs this test by checking if the velocity component perpendicular to the edge becomes smaller than some pre-defined ϵ value; if this is true, the particle is now under a "sliding" scenario.

In this scenario, the acceleration vector is projected along the surface of the workpiece. If this projected acceleration is zero, the particle will stay in place; if not, the particle will begin to "slide" along this edge.

We achieve the modeling of this scenario by projecting the "free-fall" acceleration in Equation (3.4) along the workpiece edge, which produces a straight-line path.

Note that while the particle is now traveling along an edge, the resulting equation takes on the same form as the free-fall equation; the only difference is that the acceleration is now projected. Recall that $\vec{x}(t)$ here describes the position of the particle at time t.

$$\vec{x}(t) = \vec{x}_0 + \vec{v}_0 \cdot t + \frac{1}{2}\vec{a}_{projected} \cdot t^2 \tag{3.8}$$

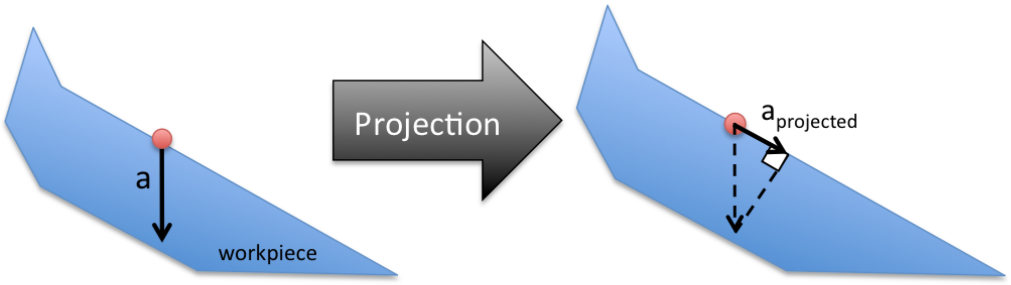

Figure 3.7: Demonstration of the sliding equation.

With these two formulations, the particle's path can be simulated on both edges and in open space when the acceleration field is constant (e.g. the workpiece is not rotating). Furthermore, because both of these equations are strictly quadratic in t, they can be substituted into the definition of geometric primitives to perform intersect tests.

3.5 Rotation

Here we begin to examine the scenarios where the workpiece is rotating during particle simulation.

In this paper, we choose our frame of reference to be the X and Y axes that define the workpiece geometry. This means that when the workpiece rotates, our frame of reference stays fixed to the workpiece. Consequently, rotation changes the orientation of the acceleration field.

We see now how this rotating acceleration field affects the following scenarios. While we originally presented the edge travel scenario (Scenario #2) second in the "No Rotation" section, we will present the edge travel scenario first in the "Rotation" section. This will illuminate the challenges we face when simulating the last scenario presented.

Scenario #3: Concurrent Rotation & Sliding Equation

In this scenario, the particle is traveling along a workpiece edge and the workpiece is rotating.

We know that the acceleration vector is projected along the edge during edge travel. This means that the direction of the acceleration in this scenario is fixed; only the magnitude of the projected acceleration varies as the workpiece rotates. Consequently, rotation of the acceleration field only results in the magnitude of the projected acceleration changing.

This leads to a special case of particle motion which we will refer to as "concurrent rotation and sliding." We first define $\hat{a}_{projected}$ as a constant unit-vector in the direction of the edge and $|a_{projected}|(t)$ as the magnitude of the projected acceleration over time. With these two new variables, we can show the resulting formulation of this scenario in Equation (3.9).

$$\vec{x}(t) = \vec{x}_0 + \vec{v}_0 \cdot t + \frac{1}{2}\hat{a}_{projected} \cdot t^2 \cdot |a_{projected}|(t) \tag{3.9}$$

This scenario is further illustrated in Figure 3.8.

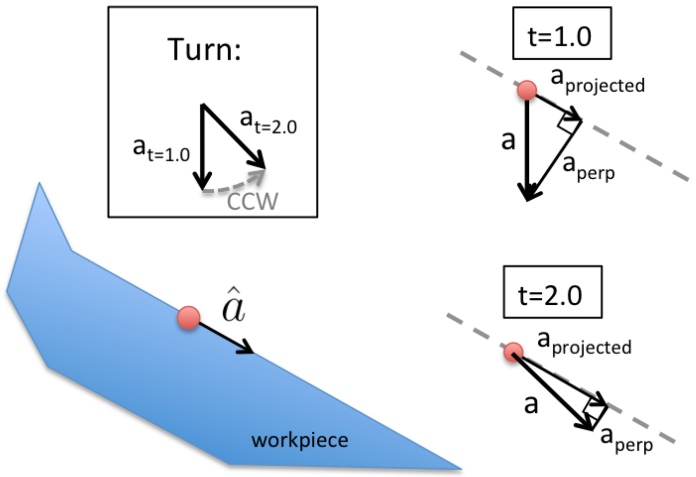

Figure 3.8: Illustration of Scenario #4 – concurrent rotation and free-fall.

If rotation occurs at a constant rate and the acceleration vector begins perpendicular to the edge, the magnitude term can be modeled as

$$mag_{accel} = cos(\omega \cdot t)$$

where ω is the angular velocity.

In Equation (3.9), we no longer have a formulation of particle motion that is strictly quadratic in t. The trigonometric terms introduced by the rotation of the workpiece make it impossible to solve for path-primitive intersections in constant time. We can, however, easily substitute in values of t to determine the position of the particle at a given time.

Despite the inability to intersect these trajectories with geometric primitives in constant time, we still simulate this scenario in practice. The reason for this will be detailed after Scenario #4 is presented, when a holistic comparison of these two scenarios can be conducted.

Scenario #4: Concurrent Rotation & Free-Fall Equation

The fourth scenario is when the particle is traveling through open space and the workpiece is rotating. In this case, the acceleration vector is no longer projected along an edge as shown in Equation (3.9). The acceleration vector now has a direction that changes in time (with a fixed magnitude), as shown in Equation (3.10).

$$\vec{x}(t) = \vec{x}_0 + \vec{v}_0 \cdot t + \frac{1}{2}\vec{a}(t) \cdot t^2 \tag{3.10}$$

Here, the $\vec{a}(t)$ term describes the acceleration vector over time as the workpiece rotates. Assuming a constant rotation, this term breaks down to:

$$a_x = cos(\alpha t) \cdot |a_g|;$$

$$a_y = sin(\alpha t) \cdot |a_g|.$$

This scenario produces paths that mathematically are not parabolic, as illustrated below in Figure 3.9.

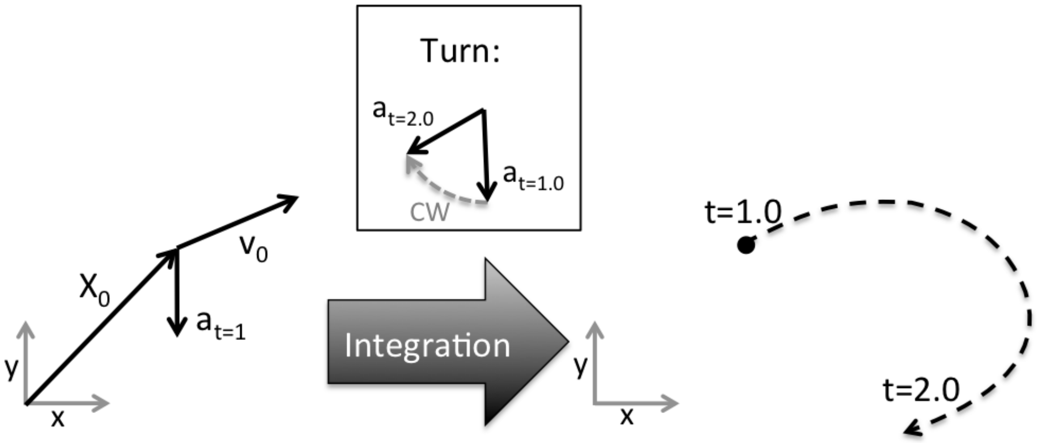

Figure 3.9: Demonstration of the particle path during concurrent rotating and free-fall

Assumption #1 - No concurrent Rotation and Free-fall

These non-linear trigonometric terms produce an equation that can no longer be intersected against a geometric primitive in constant-time. Intersection points can still be calculated for these path segments via approaches like Newton's method, but doing so requires additional computational time. Our approach to the draining problem requires many different particle simulations; consequently, we restrict our simulation scenarios to having constant-time intersection tests.

This restriction effectively prevents our ability to simulate particles in this "concurrent rotation and free-fall" scenario, leading us to our induced first assumption and restriction in this paper – particles may not travel through open space while the workpiece is rotating.

Our results chapter provides quantitative analysis of this scenario #4 rejection and a related justification.

Resolution of Scenario #3

The third scenario, concurrent rotation and sliding, similarly has a kinetic path equation that cannot be easily intersected with geometric primitives. The key difference between this scenario and scenario #4, however, is that the particle is traveling along a known edge. Since this edge length is known as well as the duration of the turn, this particle path does *not* need to be intersected with other primitives in the workpiece if it remains on the edge.

Instead, the position of the particle after the specified turn is calculated. If the particle is still on the same edge at the end of the turn, we can then proceed with simulation because we are guaranteed that the particle did not collide with any other surface during the turn. If the particle is beyond the length of the edge at the end of the turn, we do not have this no-collision guarantee and cannot proceed with simulation.

Thus although we have two scenarios where particle path and solid intersection tests are difficult, only one of them (concurrent rotation and free-fall) cannot be simulated at all. The other (concurrent rotation and sliding) can be simulated whenever the particle has not traveled beyond the edge at the end of the turn, which was the most common result during our tests.

3.6 Collisions

Collisions occur when the path for a given particle intersects a surface inside the workpiece.

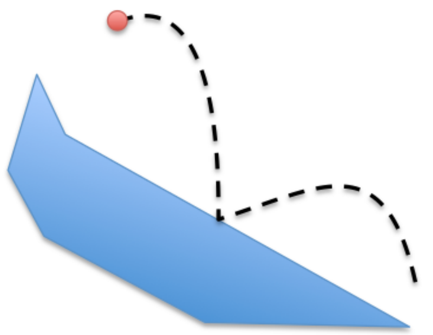

Figure 3.10: A particle colliding with a surface and the resulting elastic collision

After a collision occurs, the particle "transitions" into one of three states:

- the particle has come to rest and simulation terminates,
- the particle begins a new path through open space,
- ehe particle begins a new path along the surface of a polygon.

There are many types of collisions depending on the direction of the acceleration vector, the kinetic energy that the particle contains, and the angle between surfaces on a polygon. We present the two primary collisions (or transitions) of concern:

Planar To Sliding Transition

When a particle collides with an edge and obtains a resulting perpendicular velocity (v_{perp}) less than some epsilon (ϵ) value, the particle transitions from free-fall to sliding along an edge.

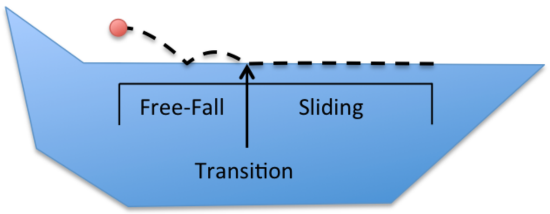

Figure 3.11: Particle path transition from free-fall to sliding.

Sliding To Free-Fall Transition

When a particle reaches the end of an edge (effectively intersecting the adjacent edge), it may enter free-fall, where it is further simulated.

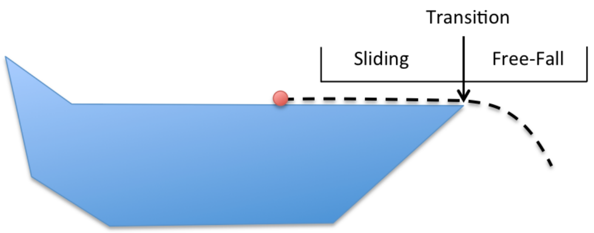

Figure 3.12: Particle path transition from sliding to free-fall.

Conservation of Momentum

The only source of energy in this demo is the potential energy from gravity. Note that as the workpiece rotates, the gravity vector (and corresponding field) rotates. This means that a particle in a low-energy configuration can transition to a high-energy configuration through rotation; consequently, rotation may add energy into the system.

Settling Guarantee

The only addition of energy is from rotation, so in the absence of rotation, no energy is added to the system. Because the elasticity κ of the particle-edge collisions is chosen to be less than 1 in simulation, energy dissipates as a particle collides with other surfaces.

This leads us to the settling guarantee: the simulation of a particle during a time period with no rotation is guaranteed to terminate in one of two ways:

- the particle settles into a concave vertex with a kinetic energy less than ϵ, or
- the particle exits the workpiece, which is the goal of the search.

Chapter 4

Solution Search

4.1 General A.I. Search

Artificial Intelligence search is a field of study and class of algorithms that discover (or “search” for) plans that turn a start state into a goal state in a deterministic world. A.I. search makes heavy use of simulation to compute solutions offline before enacting those solutions in the real world.

Based on our ability to rapidly simulate water particles and the deterministic nature of the workpiece draining problem, we choose A.I. search to discover a series of rotations that fully drains a workpiece.

Terminology

In order to describe how A.I. search applies to the draining problem in manufacturing, some terminology must be defined first. In order to aid our definitions, we will use a simple positional search problem as an example, shown below.

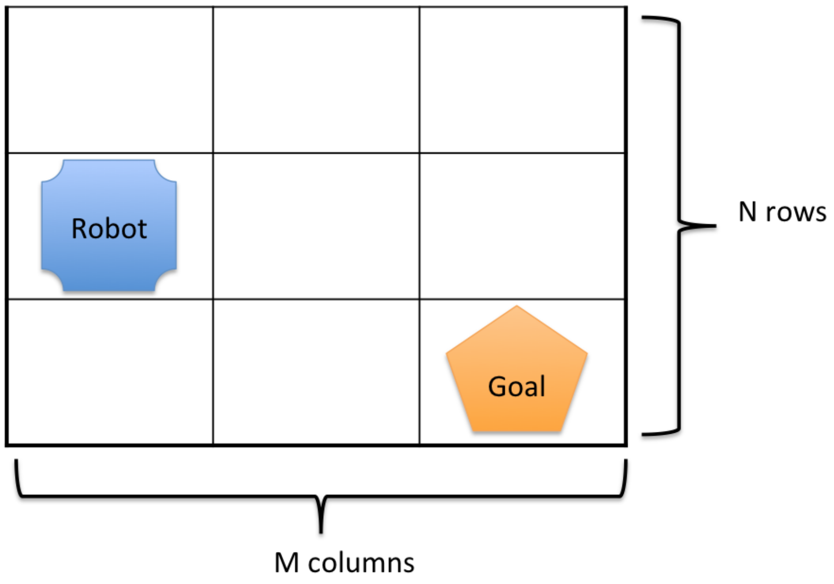

Figure 4.1: A small positional search problem; a robot on a M by N grid is attempting to reach a goal position G_{pos}

The “state” is defined as a set of information that minimally represents the current progress to the goal. For this positional search problem, the minimal state representation would be the position tuple of the robot (R_x, R_y), where R_x is the horizontal position of the robot and R_y is the vertical position of the robot. Here these variables can take on positive integer values up to M and N respectively.

For every state, the agent under control has a number of actions available to take. In our example, the robot would have the actions $North$, $South$, $East$, and $West$ available to execute.

When the agent under control takes an action, a new state is produced. The “transition function” is responsible for transforming a state action pair into a successor state. For example, if the robot above was in state $(M = 1, N = 2)$ and took the action $North$, the transition function would return the state $(M = 1, N = 3)$.

This transition function is also responsible for performing validity checks against the environment. In our positional search example, the transition function would be responsible for making sure the robot does not move off the grid or into a square occupied by another agent or wall. If an invalid state action pair is given to the transition function, the transition function commonly returns *null* to designate this invalid combination.

The “successor function” is a meta-function that uses the transition function and actions available to the agent. It is responsible for generating a list of successor states reachable (in one action) from a given state. For the above problem, the successor function would return the list of adjacent squares the robot could move into and the actions that produce those states.

A "plan" is simply a sequence of actions. While reaching the goal state determines that the search algorithm has succeeded, the sequence of actions to achieve a goal state is usually the objective of the solution. Consequently, plans are usually maintained and updated by the successor function when exploring states.

General Tree Search

These definitions are enough to implement the standard tree search algorithm [9].

Algorithm 1 Standard A.I. Tree Search

```
 1: function TreeSearch(problem)
 2:     queue ← (initial state of problem, empty plan)
 3:     while queue not empty do
 4:         state, plan ← REMOVE-FRONT(queue)
 5:         if Goal-Test(problem,state) then
 6:             return plan                                          ▷ Success
 7:         end if
 8:         InsertAll(queue, SuccessorFunction(state, plan))    ▷ Inserts a list of new states
    and plans into the queue
 9:     end while
10:     return failure                ▷ Failure from exhausting all states on the queue
11: end function
```

Importance of Formulation

The above definitions of state, actions, and successor functions are quite general. This means that there are several valid formulations (or definitions) of states for a given problem. The particular formulation chosen can have large consequences on the effectiveness of A.I. search and the algorithmic runtime.

The size of the state space is the important factor when considering different problem formulations. A state space size is defined by the size of the set of the different values the state could hold.

For our example above, we define the state to be the position tuple (R_x, R_y). Since R_x and R_y can take on positive integer values up until M and N respectively, the size of the set of possible state values is MN.

While a positional search problem with one agent does not present a large (or "expansive") state space, these state spaces can quickly get out of control for slightly more complex problems. For instance in the general case of l agents on a board of M by N size, the state space is defined by the list of position tuples $(R_x^1, R_y^1), (R_x^2, R_y^2)...(R_x^l, R_y^l)$. In this case the size of the state space is all possible permutations of these variables, or $(MN)^l$.

State spaces often grow exponentially large with the addition of new variables. If the state space becomes too expansive, the effectiveness of general search algorithms can be jeopardized. This is because general A.I. search has to enumerate through (or "process") many different states before reaching a goal state. If there are too many states to process before reaching a goal state, the algorithm may not terminate in a reasonable amount of time.

4.2 Adaption of A.I. Search

Our work focuses on a traditional application of A.I. search to a novel state space formulation for workpiece draining. Our unique state formulation reduces the size of the state space to make it manageable for general A.I algorithms.

Traditional Formulation

Rotating a water particle out of a workpiece can be considered a kinematic planning problem. Traditional kinematic planning problems often have state spaces that encode the kinematic properties of the object (or agent) being planned for [6] [8]. The most common state formulation is to encode the position, velocity, and acceleration of the agent as the state formulation.

The actions available to the agent at any time are usually actions that change the acceleration of the agent. The transition function integrates this kinematic state and acceleration change in time with a pre-defined accuracy. Consequently, many transition function calls must be made to accurately integrate the path of the agent through time.

The problem with this type of state formulation is that the state space commonly becomes far too expansive for general A.I. search algorithms. Consider a reduced problem where there are only 1000 possible position, velocity, and acceleration vectors. The size of the state space in this reduced problem is 1000^3. This presents a large number of possible states for a search algorithm to enumerate through; the scale of this state space may even prevent the algorithm from terminating.

In kinematics problems, this type of "traditional" state formulation may jeopardize the entire algorithm by producing a state space that is too expansive. We present a different formulation that overcomes this obstacle.

Our State Formulation

Our state formulation overcomes these difficulties by drastically reducing the size of the state space. We instead offload most of the computational overhead to the successor function.

The key insight for this state space reduction is to observe only the concave vertices of the workpiece. This insight was first used by Yasui et al. [11] in order to reduce the scope of their drainability analysis problem. Because particles cannot stick to edges and are always under the influence of gravity, particles only stop moving when they "settle" in a concave vertex. Consequently, any water particles placed anywhere inside the workpiece will move under the influence of gravity until they arrive at a concave vertex (or exit the workpiece).

Here this insight is instead applied to the state space formulation for the first time in order to maintain the effectiveness of general A.I. search. We define our state as the "settled" position of the particle, meaning the state is either:

- The concave vertex the particle is resting in, or
- outside of the workpiece.

By defining our state this way, we produce a state space with size of

$$V_{concave} + 1$$

where $V_{concave}$ is defined as number of concave vertices in the workpiece. The size of this state space is small compared to kinetic state definitions which can be exponential in the accuracy of each value.

Because the maximum number of states visited is now only linear in concave vertices, our search algorithms have far fewer states to expand and a lower upper bound on running time.

4.3 Exploration

The bulk of the computational work will now be in the successor function, which is responsible for producing the set of states that are one "action" away from a given input state. If a given state S_i is one action away from an input state S_j, we define S_i to be "reachable" from S_j.

Successor Function

The successor function produces the list of reachable states from a given input state. For our problem, this means that the successor function will produce the set of concave vertices (or workpiece exit status) reachable from a given concave vertex.

$$successor(V_c^i) = \left\{V_c^l, V_c^m, ...\right\}$$

Action Definition

The "actions" available to our agent will be represented by the possible set of rotations of the workpiece. In this problem we choose our frame of reference to be the workpiece geometry, so the rotation of the workpiece is observed as a rotation of the gravity vector through time. This means that our action results in a change in the acceleration field.

More formally, we define one action (or "turn") as a sweep in the gravity vector from g_{start} to g_{end}. The duration of this sweep is computed from the algorithm-wide rotation speed constant; consequently, the duration of the turn is a derived property rather than a free variable.

We restrict all turns to be less than 180°; consequently, the direction of the turn from g_{start} to g_{end} is then implied. The interpolation function between g_{start} and g_{end} could take on any form, but we restrict ourselves to linear interpolation for simplicity.

Transition Function

Now we have fully defined both elements for the transition function; the states are concave vertices and the actions available to the agent are specific turns.

Consequently, the transition function is then defined as a function that takes in a concave vertex and a turn; it produces the settled state of the particle after simulation.

$$transition(V_c^i, turn(g_{start}, g_{end}) = V_c^n \qquad or \qquad exit$$

Possible set of Turns

A turn is defined by two different variables: g_{start} and g_{end}. These variables could take on many values which means many different turns can be defined. Consequently, a large number of actions are available to the agent at any given state which leads to computational explosion (yet again).

In this work, we reduce the possible set of "turns" from a given concave vertex to within reasonable limits. This is accomplished by reducing the set of gravity vectors available from which turns are defined.

Ideally, we reduce the space of all possible actions to something "manageable" which still represents the full range of kinetic possibilities. An action or state space is "manageable" if it does not jeopardize termination of the algorithm.

To understand how the set of gravity vectors is reduced, some terminology must first be defined. A given concave vertex has two incident edges E_1 and E_2 defined by outward vectors $\vec{e}_1$ and $\vec{e}_2$.

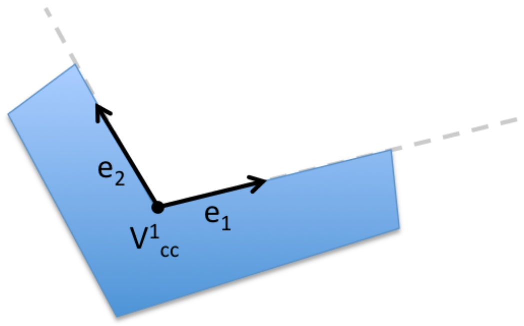

Figure 4.2: A concave vertex V_c^1 and its two corresponding outward edge vectors $\vec{e}_1$ and $\vec{e}_2$

We start with a full Gaussian circle of possibilities to sample g_{start} and g_{end} from, as illustrated in Figure 4.3.

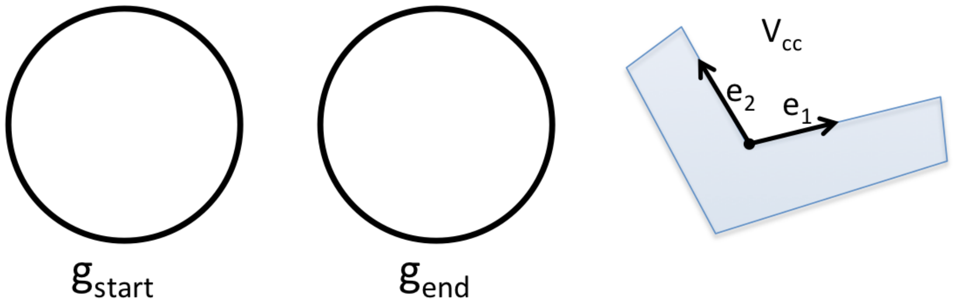

Figure 4.3: The initial set of possible vectors to sample g_{start} and g_{end} from.

We first restrict the possible values of g_{start}. We define g_{start} as a vector that does not produce motion of the particle but is as close as possible to producing motion.

If g_{start} satisfies these two conditions, two benefits arise. The first is that if g_{start} does not induce motion of the particle, a turn that ends in g_{start} does not require simulation once the turn ends. This will be greatly useful for our solution post-processor. The second benefit is that any turn defined with g_{start} spends the maximum amount of time inducing motion of the particle.

There are only two vectors that satisfy these two conditions – the perpendicular vectors to $\vec{e}_1$ and $\vec{e}_2$ that point into the workpiece, denoted as $\vec{e}_1^p$ and $\vec{e}_2^p$ in Figure 4.4.

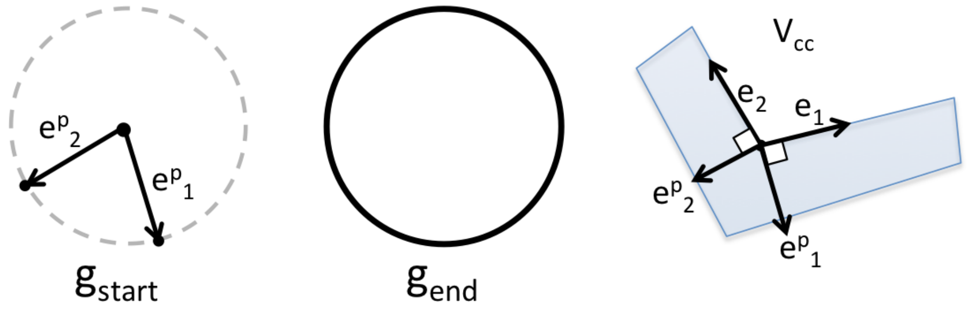

Figure 4.4: g_{start} is reduced to one of the two perpendicular vectors

Our second restriction is that g_{end} must be a vector that produces motion of the particle while simultaneously keeping the particle on either incident of the edges of the concave vertex. This produces two quarter-circle segments of possibilities for g_{end}, each of which are defined by a incident edge and the perpendicular vector to that edge as illustrated in Figure 4.5.

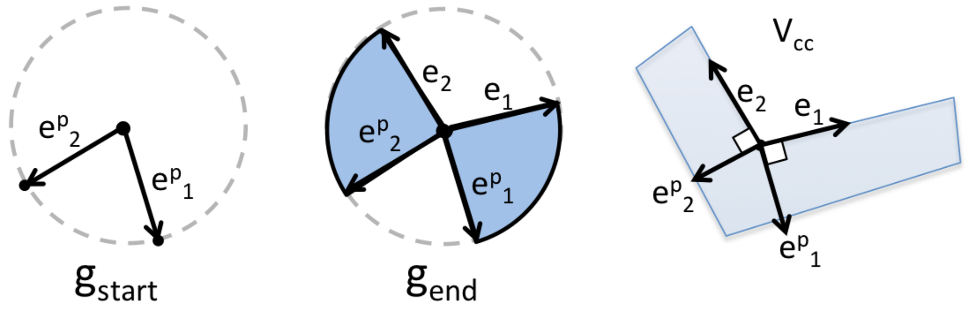

Figure 4.5: g_{end} is reduced to vectors that produce motion while keeping the particle on the incident edge

This leaves two arcs of gravity vectors from which g_{end} can be selected. Note that once g_{start} is selected, g_{end} is selected from the arc of vectors that borders g_{start} in order to produce turns that maximize motion of the particle.

Ideal Successor Function

The ideal successor function produces the set of all "reachable" concave vertices from a given input concave vertex (and the actions to achieve those states).

Unfortunately there are an infinite number of turns possible from each concave vertex, since g_{end} can be selected from a continuous range of vectors. Clearly enumerating through every possible turn at every concave vertex is not possible.

Sampling

To overcome this difficulty, we will instead sample from the continuous range of g_{end} vectors and produce a finite number of turns. Ideally the set of concave vertices produced by these sampled turns would be identical to the set achieved by the full enumeration of the sampling space. This is not likely to happen in practice, but with careful design we can discover the majority of this set.

Representative Coverage Between Limits

The way we produce the majority of this set is to obtain a representative coverage over the possible kinetic paths out of a concave vertex. This is done by choosing a sampling method that produces the most diverse set of kinetic paths.

If the possible range of g_{end} vectors is defined between the limits of 0 and 1 and our interpolation function $f(x)$ is a function of this progress variable x, then an initial best-guess sampling function might be to use uniform sampling:

$$f(x) = x.$$

With this sampling function, turns are selected uniformly between the limits of g_{end}. This sampling function unfortunately does not produce a uniform distribution of kinetic paths. This is primarily because acceleration effects compound with time; uniform sampling produces many kinetic paths that are almost identical. This is illustrated in Figure 4.6.

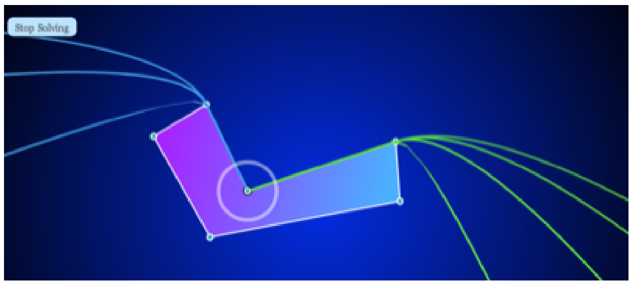

Figure 4.6: The range of kinetic paths produced from 10 per side with uniform sampling. Many of the sampled kinetic paths are nearly identical and thus visually appear as one.

A far better sampling function is to use cubic sampling:

$$f(x) = x^3.$$

This function produces many samples near the lower bound of g_{end} vectors and only a few samples near the upper bound. These low-end samples produce accelerations that are minimally different, but since acceleration effects compound with time, they produce drastically different kinetic paths. This is illustrated in Figure 4.7.

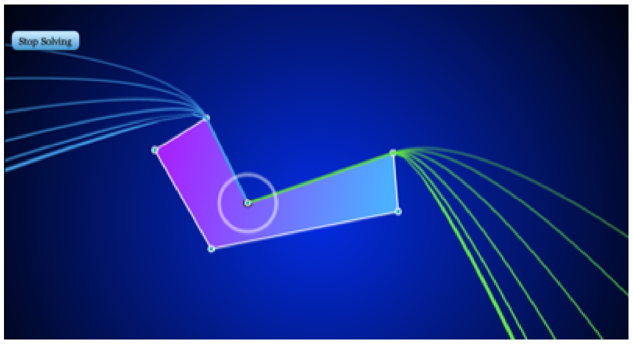

Figure 4.7: The range of kinetic paths produced from 6 samples per side with cubed sampling. Note the resulting kinetic paths evenly cover the space after the incident edge.

With this sampling function, we can obtain better coverage with lower computational cost.

4.4 Complete Formulation

Now we have all the pieces to implement A.I. search. Our state is defined as the concave vertex the particle rests in (or workpiece exit). Our successor function samples from an available range of actions to produce a representative set of reachable concave vertices. Simply combining these two definitions together can produce an algorithm that discovers a sequence of actions to rotate a water particle out of a workpiece.

We now present two modifications to general A.I. search to enhance performance.

Uniform Cost Graph Search

The first search modification is to switch to the standard graph implementation of our search algorithm; this effectively converts tree search (detailed in Algorithm 1) into graph search [9]. This graph version of the algorithm prevents a state from being expanded twice [9]. Because the connectivity graph within workpieces often has many cycles [11], preventing more than one expansion from each state helps considerably in cutting down computational cost.

Furthermore, we use the “uniform cost” version of graph search [9] to ensure that the solution discovered is optimal in cost. This is accomplished by enqueueing partial plans onto the priority queue by their associated cost, as detailed in Algorithm 2.

Algorithm 2 Uniform Cost Graph Search

```
1: function UniformCostGraphSearch($problem$, $fringe$)
2:     $closed$ ← an empty set
3:     $fringe$ ← Insert(Make-Node(Initial-State[$problem$]),$fringe$)
4:     while $fringe$ not empty do
5:         $node$ ← Remove-Front($fringe$)
6:         if Goal-Test($problem$,State[$node$]) then
7:             return $node$                                          ▷ Success
8:         end if
9:         if State[$node$] not in $closed$ then
10:            add State[$node$] to $closed$
11:            $fringe$ ← InsertAllByPriority(Expand($node$, $problem$), $fringe$)
12:        end if
13:    end while
14:    return $failure$                       ▷ Failure from exhausting all states
15: end function
```

Optimal Actions

The second search modification is implemented directly in the successor function. By definition, the successor function returns a set of "reachable" states; each state is paired with a list of associated actions to transition to that state. Normally during search, every one of these associated actions generates a new plan that is placed in the priority queue.

Our optimization here is simply to return a single optimal action that produces a state, rather than a list of arbitrary actions. This results in fewer plans enqueued onto the fringe while maintaining optimality.

This is accomplished by placing actions into a cost-sensitive closed set [9]. A cost-sensitive set behaves like a normal set; the only difference is that each entry is keyed by a "cost" variable. When inserting a new item, that item will be inserted into the set if it does not already exist *or* if it has a cost that is lower than the existing member.

By using this slightly different data structure, our successor function produces the same set of reachable states, but now each state is paired with am optimal action.

4.5 Cost

The uniform cost graph search algorithm we use ensures the plan produced is optimal in cost. This is because the goal check for a partial plan is performed after the plan is dequeued from the priority queue. When a partial plan is dequeued from a priority queue that is ranked by cost, we are guaranteed that no other plan has a lower cost. This guarantee of optimality is the foundation of uniform cost and A* search [9].

Our definition of cost however is very flexible and can be modified according to different manufacturing priorities.

Time

The primary concern in manufacturing is often the processing time for a given procedure. In this case, the plans enqueued into the fringe are ordered by the time needed to execute that plan. This total time includes the sum of all turn durations as well as the time needed to wait for particles to settle. Enqueuing plans by this value ensures that when we dequeue a plan that ends in the goal, it has the lowest total time out of any plan on the fringe (ensuring optimality in time).

Energy

Another concern in manufacturing is the energy required for a given procedure. If energy is the optimization goal, plans can be instead enqueued by the estimated total amount of energy required to execute that series of turns.

There are many ways to estimate the amount of energy required to execute a series of turns. The exact details are dependent on the fixture setup and associated motor; we present three simple models of energy estimation.

Energy - Total Rotation Angle

The first estimation of energy is based on the total rotation angle. If friction in the motor bearings is the primary cause of energy loss for the draining process, plans can be enqueued by their total absolute angle change.

This model would be appropriate if the workpiece was well balanced in a fixture that produced high energy loss due to friction.

Energy - Workpiece Center of Gravity

Another model for energy is based on the work the motor has to perform against gravity. If the workpiece's center of gravity is positioned far away from the center of rotation, the workpiece is unbalanced in the fixture. Turns that raise the center of gravity will produce counter-torques, whereas turns that lower the center of gravity will produce added acceleration.

If this model is more appropriate for the fixture setup, the plans can be enqueued onto the fringe based on the integrated change in height for the center of gravity.

Energy - Acceleration And Deceleration

Finally, if the workpiece is well balanced and friction in the rotator is low, the majority of energy loss will be due to acceleration and deceleration of the workpiece. In this case, the most appropriate model for energy cost may be one based on the total number of turns in the plan.

This contrasts with the energy model of total angle; while the total number of turns in a plan commonly scales with the total angle of a plan, these two properties are not necessarily the same.

Our Implementation

Our implementation of this paper (described further in Chapter 6) uses time as the cost function during solution search. This is primarily an interaction design decision; users are more likely to sense optimality in time over other possible cost functions.

Solution Post Processing

A full solution is defined as the sequence of gravity transitions to drain the workpiece. The graph search algorithm returns a list of gravity turns from each concave vertex to drain the workpiece; the only issue is that the g_{end} from one turn rarely matches the g_{start} of the following turn.

In order to rectify this situation and produce a full control sequence, our implementation contains a solution post-processor that stitches together these disparate turns and creates interpolations between one turn's g_{end} and the next turn's g_{start}. We refer to these turns as "setup" turns, for they transition the workpiece's orientation from the previous action ($A = i$) to the beginning of the next action ($A = i + 1$). These "setup" turns can be seen during the solution animation process in our implementation which is linked in the results chapter.

These "setup" turns never induce motion of the particle; this is because only one particle is simulated at a time and the range of vectors from $g_{end}^{A=i}$ to $g_{start}^{A=i+1}$ does not produce motion.

We arrive at this guarantee with the following reasoning. A particle only settles when the terminal acceleration field g_{end} points "into" a concave vertex. Since the g_{start} of the next turn is defined as one of the outermost vectors that *also* does not produce particle motion, a sweep from $g_{end}^{A=i}$ to $g_{start}^{A=i+1}$ never produces an acceleration that induces movement of the particle. This guarantee is illustrated in Figure 4.8.

When simulating multiple particles, this guarantee no longer holds. We present a further reduction of the action space in Chapter 5 to restore this guarantee when working with multiple particles.

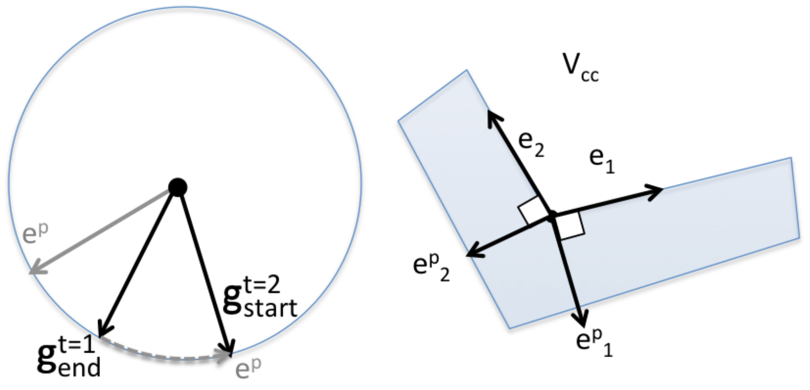

Figure 4.8: The geometric relationship between $g_{end}^{A=i}$ and $g_{start}^{A=i+1}$ as it relates to the guarantee that a setup turn does not induce motion

4.6 Time Complexity

Now that the entire search algorithm has been defined, we can analyze the time complexity of each subroutine. The most time complex components of our approach are:

- the intersection between a particle path and the workpiece,
- the simulation of a particle until settling,
- the search expansion for a particular state, and
- the overall search for a solution.

We start with the intersection between a particle and the workpiece – our fundamental unit of computation. The intersection between a particle path and the workpiece is linear in the number of edges in the workpiece e, since each edge in the workpiece is intersected against a test path.

Next, the simulation of a single particle continues until the particle settles in a given location. Each time a particle collides with a surface, it obtains a new kinetic state and resulting parabola. This new parabola then needs to be intersected against the workpiece; consequently, the time complexity of particle simulation is determined by the number of collisions the particle experiences before settling.

The number of collisions for a given particle simulation is not easily bounded; if the elasticity of the system κ is 1, the particle could never settle in a concave vertex depending on the geometry of the workpiece. The best case lower bound is when a particle's simulation terminates after one collision or immediately exits the workpiece.

In practice we found that most particles had a number of collisions that was proportional (linearly) to the number of edges in the polygon e. We cannot however theoretically bound the time complexity of the particle simulation step.

If a particle were to collide with each of the e edges in the workpiece once during simulation, there would be e^2 total intersection tests for a single particle simulation.

Next, the search expansion for a state results in many particle simulations. The exact number is specified by c_1, the user-specified number of samples per state. If each particle simulation involves e^2 intersection tests, an expansion from a given state results in $c_1 \cdot e^2$ intersection tests.

Finally, the time complexity of the entire solution search must be analyzed. The state space of our formulation has a size of $V_{concave} + 1$. In the worst case scenario of solution search, the algorithm must explore every possible state. If each state expansion requires $c_1 \cdot e^2$ intersection tests, the entire algorithm requires:

$$c_1 \cdot e^2 \cdot V_{concave} \tag{4.1}$$

intersection tests.

Note that Equation (4.1) provides a meaningful upper-bound on the number of intersection tests that must be performed during solution search with only one core assumption – every particle simulation produces a path that collides with each edge of the workpiece once. The actual behavior of particle simulation may involve far fewer collisions.

Chapter 5

Extension to Multiple Particles

We now present an extension of this work for draining multiple particles out of a workpiece. The initial adaption to our approach is straightforward, but the two search optimizations we present shortly afterwards dramatically improve search performance.

5.1 Initial Adaption

Only two elements of our approach need to be modified in order to drain multiple particles out of a workpiece. The first is a modification to the state definition; the state is now a set of settled locations, one for each particle:

$$(V_c^1, V_c^2, V_c^3, ..., V_c^n). \tag{5.1}$$

The second modification is to our successor function definition; the successor function now takes in a set of particle locations and produces a set of "reachable" states:

$$successor((V_c^1, V_c^2, ..., V_c^n)^i) = \left\{(V_c^1, V_c^2, ..., V_c^n)^1, (V_c^1, V_c^2, ..., V_c^n)^2, ...\right\}. \tag{5.2}$$

With only these modifications alone we are now able to search for solutions that drain multiple particles out of the workpiece.

Implementation

These two modifications to our approach result in a fair number of changes in the implementation. One primary implementation difference is that now a given turn must be simulated across all particle locations. This simulation is usually performed by the transition function:

$$transition(V_c^i, turn(g_{start}, g_{end}) = V_c^n \qquad or \qquad exit.$$

Earlier we had restricted our action space to only turns that produced motion of the particle and kept the particle on the incident edge of the concave vertex. This means there was only one possible outcome for a turn simulation – motion of the particle was induced, and that motion began immediately when the turn was performed.

Now that any arbitrary turn can be simulated against an arbitrary particle, this guarantee no longer holds true. Consequently, we now have four possible simulation scenarios in the transition function as described below.

- The particle has exited the workpiece; a turn does not change this particle's state.
- The turn results in no motion; any linear combination between g_{start} and g_{end} does not produce motion of the particle.
- Motion of the particle starts immediately when the turn begins; g_{start} is a vector that produces motion.
- Motion of the particle starts midway through the turn; g_{start} does not produce motion, but as the gravity vector sweeps to g_{end}, motion is induced at some intermediate point.

Because of these multiple simulation scenarios, the multiple-particle version of the transition function may not make assumptions about the input turn. While not of great importance for the overall picture of the approach, this presents a significant obstacle during implementation that should be considered.

5.2 Multiple Particle Time Complexity

Now that our state is defined as a set of concave vertex locations, the state space is unfortunately exponential in the number of particles n:

$$(V_{concave} + 1)^n.$$

As in Section 4.6, we would like to produce an expression that estimates the upper bound of intersection tests that must be performed during search. With the same previous assumption that each particle simulation involves e^2 intersection tests, our expression for the upper bound intersection tests for multiple-particle search becomes:

$$c_1 \cdot e^2 \cdot (V_{concave})^n. \tag{5.3}$$

Equation (5.3) is exponential in the number of concave vertices rather than linear as in Equation (4.1).

Although earlier we had avoided exponential state spaces and runtimes due to our search formulation, the introduction of multiple particles has yet again vastly expanded our state space. Our algorithm as presented thus far will still find solutions to these problems; unfortunately, it will take several factors of time longer to obtain solutions.

In order to improve running time of the algorithm, we present two search optimizations that significantly reduce the number of states expanded and obtain solutions much faster.

These optimizations sacrifice the guarantee of optimality in exchange for an improvement in running time.

5.3 Search Heuristics

The high level goal of a search heuristic is to "guide" the exploration of state space towards potential solutions [9]. This is accomplished by either preventing the exploration of certain states or by intelligently ranking potential plans on the fringe [9]. If fewer states are explored before a solution is found, the algorithm's performance will improve.

Because the definition of a heuristic is quite general, there are many possible heuristics to choose from – the space of heuristic choices is wide. We present our specific choice of a heuristic for the multiple-particle draining problem below, but many other valid (and potentially better) heuristics exist.

5.4 Multiple Particle Heuristics

The one advantage of the multiple-particle formulation of the drainability problem is that it allows for more creativity and flexibility when composing search heuristics. Previously in single-particle draining, our state was simply defined as a concave vertex location. This meant that any potential heuristic had only the workpiece geometry and a position as input variables.

Now that our state is a set of settle locations, we can instead create heuristics that operate on direct properties of the state in addition to workpiece geometry.

Plan Ranking Heuristic

The first search optimization (and true search heuristic) we implemented was simply a ranking of the partial plans in the priority queue during uniform cost search. Rather than ranking only by cost, we also implemented the following two ranking modifications:

- If the number of exited particles is not the same, prefer the plan with a higher number of exited particles regardless of cost;
- otherwise, number of particles in the same location shall be combined with cost as the ranking key.

During our testing of the algorithm, we found that ranking potential plans by

$$cost - 10 \cdot numSame$$

where *numSame* is the number of particles in the duplicate locations produced an effective mix between optimality of plans and speed of search.

With this plan ranking heuristic, our search algorithm prefers states that have a high number of particles out of the workpiece or a high number of particles in duplicate locations. Plans that fit these characteristics have an additional performance benefit; exited particles do not have to be simulated in the transition function, and each concave vertex location only has to be simulated once.

This essentially reduces the time complexity of the algorithm. If o is the number of particles that have exited out of the workpiece and m is the number of particles in duplicate locations, our expression from Equation (5.3) for the upper bound of intersection tests becomes

$$c_1 \cdot e^2 \cdot (V_{concave})^{n-o-m} \tag{5.4}$$

which is a factor of $(V_{concave})^{o+m}$ smaller than Equation (5.3). Chapter 6 provides quantitative details on the performance improvements gained from this ranking heuristic; the average speedup was approximately 1.5×.

Action Space Reduction Technique

The second search optimization we present is centered around sampling from the action space. Recall that for a single particle, we defined our action space to sample from as a fixed pair of g_{start} vectors and a range of g_{end} vectors that produce motion of the particle as illustrated in Figure 4.5. Now with multiple particles, we have multiple sets of these defined vectors as illustrated in Figure 5.1.

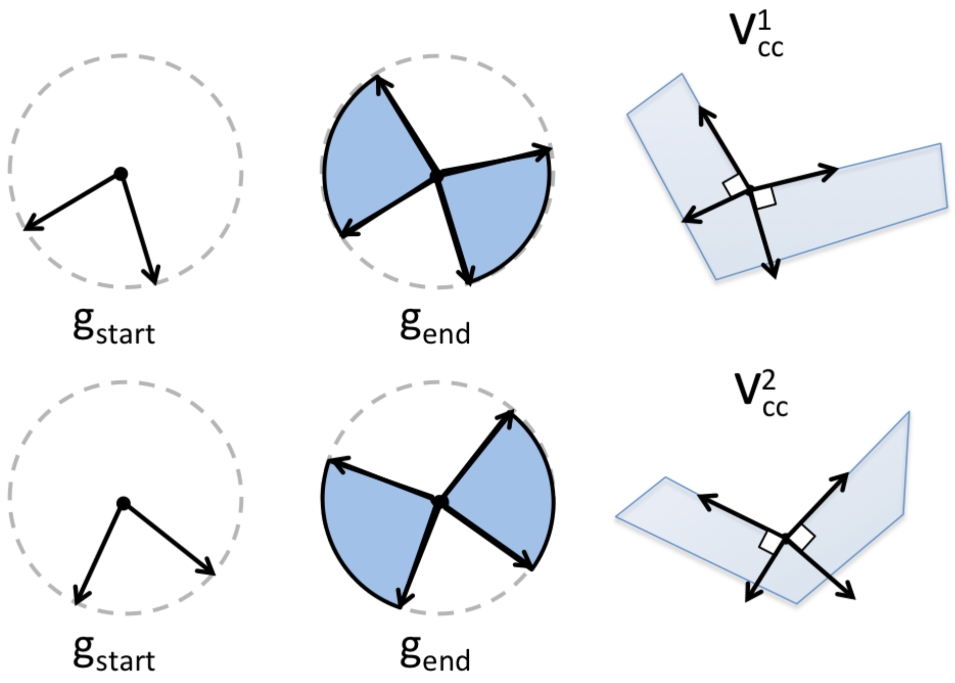

Figure 5.1: Action Space sampling for each concave vertex.

When sampling turns for our successor function, we may sample from any of these defined spaces. Doing so, however, presents three primary concerns as described below.

- This action space is fairly large and grows linearly with the number of particles in the workpiece, further increasing our algorithm's time complexity.
- Although the set of g_{end} vectors for a particular concave vertex have been reduced to keep that particular particle on the incident edge of its concave vertex, a g_{end} vector from this space may cause other particles to leave the surface of the workpiece during rotation. This results in scenario #4, which causes the sample to be rejected immediately.
- A particular turn's g_{start} may be well into the "induce motion" space of another concave vertex. If the workpiece were able to rotate infinitely fast to a particular orientation this would be of no concern; unfortunately, we assume finite rotation speed, which means a "setup" turn to this g_{start} vector would induce motion of another particle. This means that our solution post-processor can no longer assume that all turns in between partial plan turns do not induce motion. Consequently, in order to accurately simulate a turn with this condition we must rotate g_{start} back to the first vector that produces motion for any particle.

The second and third concerns are the primary reasons why an action space reduction technique is employed. These two concerns result in a high number of sample rejections or modifications, which leads to wasted computation.

The goal of our action space reduction technique is to find a pair of g_{start} vectors that do not induce motion of *any* particle. The sample space of g_{end} vectors will then be defined by this pair of g_{start} vectors as before.

Reducing our action space in this way resolves each of our previous concerns. First, the action space will now be a constant size regardless of the number of particles. Secondly, because the g_{end} sample space is generated from a pair of g_{start} vectors that do not induce motion for any particle, no g_{end} vector will directly cause scenario #4. Finally, no turn sampled from this space will induce motion of a particle immediately, meaning our solution post-processor requires no further modifications.

This action space reduction technique is accomplished in several steps. The first is to examine all the g_{start} vectors for each concave vertex, as illustrated in Figure 5.2.

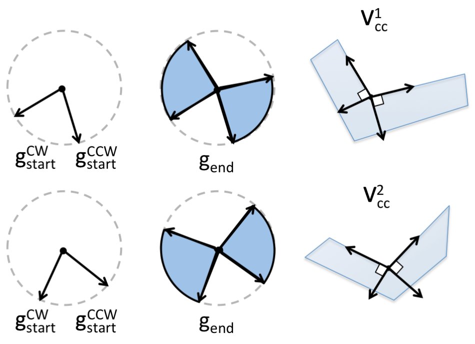

Figure 5.2: g_{start} vectors from each concave vertex.

We then define a region of the Gaussian circle between each pair of g_{start} vectors that does not result in particle motion. This region is referred to as the "dead space" for a particular concave vertex, as illustrated in Figure 5.3.

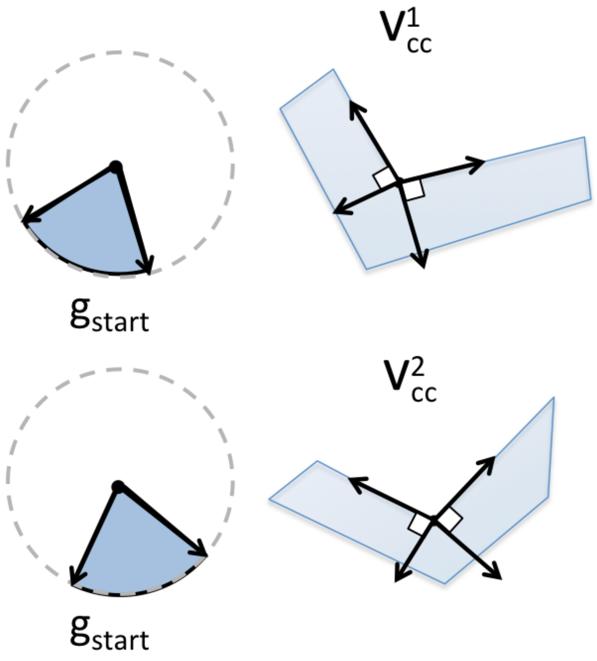

Figure 5.3: The "dead space" for each concave vertex visualized as a shaded region of the Gaussian circle.

By intersecting all of these "dead space" regions together, we can then find the total intersection region as illustrated in Figure 5.4.

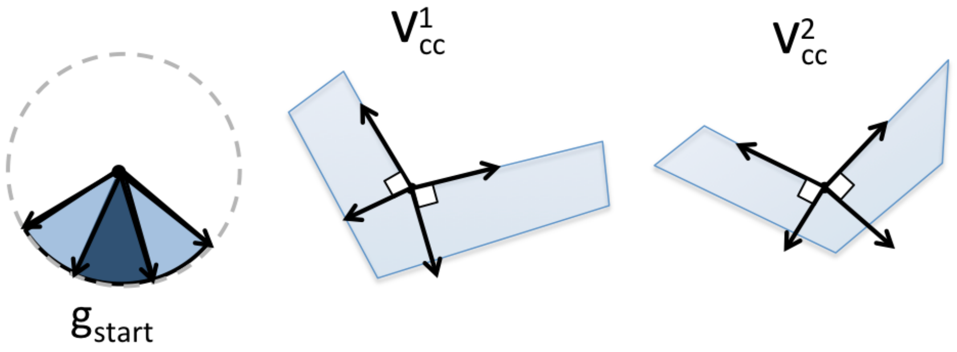

Figure 5.4: The intersection between all "dead space" regions.

Lastly, we can then select the pair of g_{start} vectors that define this region to compute our final pair of g_{start} vectors as illustrated in Figure 5.5.

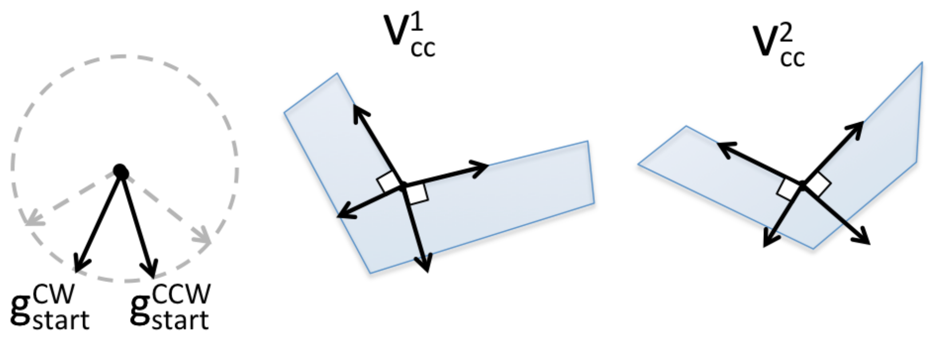

Figure 5.5: Pair of g_{start} vectors selected that do not produce motion of any particle.

Now with a defined set of g_{start} vectors, we can simply define the g_{end} sample space as those vectors which keep all particles on a incident edge of the workpiece. This is simply the set of vectors that range from parallel to orthogonal from g_{start}:

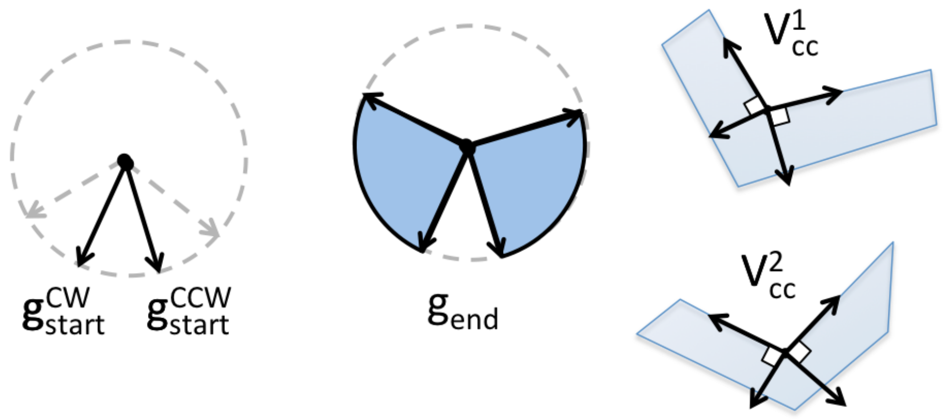

Figure 5.6: g_{start} and g_{end} vectors defined from a set of concave vertices.

Now our action space is a function of the set of concave vertex locations; this means that our action space is always the same size, regardless of the number of particles being simulated. This greatly improves performance of the algorithm.

Chapter 6

Results & Future Work

6.1 Online Demo

Our implementation of this paper and a brief solution demonstration is available online at the following url (Mozilla Firefox, Google Chrome, or Apple Safari recommended for full compatibility):

`http://petercottle.com/liquidGraph/index.html?demo`

Another solution demonstration for a more complex part is available at this url:

`http://petercottle.com/liquidGraph/yc.html?demo`

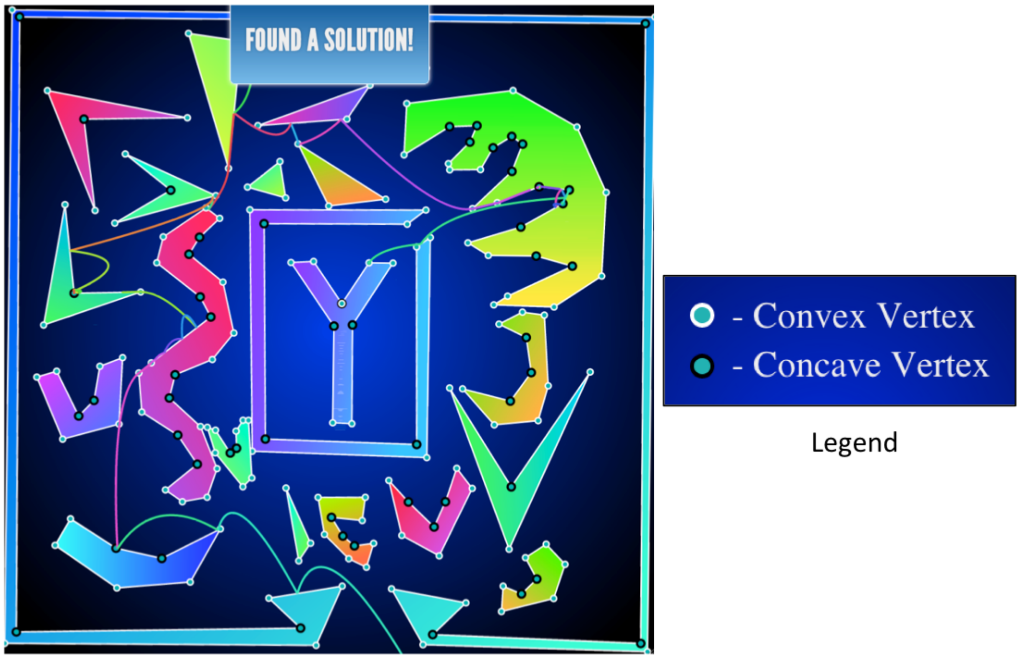

Figure 6.1: A screen capture from our implementation, available online.

Our goal with the implementation of this work was to create an interactive, dynamic experience that visualized both solution search and final results. Creating this experience required a significant investment in graphical design and subroutines solely dedicated to visualization; it also required that our implementation be designed to run asynchronously between frame draws.

The result of this design objective was a highly visual experience for the user at the expense of raw performance. Consequently, we remind readers that the following run time analysis was conducted on an implementation optimized for visuals rather than performance.

Run Time

Because our algorithm performs a significant amount of graphics processing, absolute time benchmarks for our algorithm do not provide as much utility for analysis. Consequently, we supplement our analysis with percentage breakdowns of CPU time between high-level functions to give readers an idea of which subroutines require the most computation.

The following CPU profile was collected during a solution search on a workpiece with 180 total vertices, 180 edges, 55 concave vertices, and 21 polygons. The solution was obtained in 14.64 seconds on a Macbook Air with a 1.86 GHz Intel Core 2 Duo and 2 GB of 1067 MHz DDR3 RAM. This benchmark time, however, was obtained while a CPU profiler was running (in addition to other applications) and thus overstates the runtime for the algorithm.

In Table 6.1, CPU "Total Time" is defined as the sum of computational time spent inside a given function call and all other functions called from that location. This means that high-level functions that simply call other parts of the program will have a large total time, despite very little computational time being spent inside them.

High Level Category	CPU Total Time	Function
Idle Time	53.92%	(time spent idle waiting for next animation frame)
Program	46.08%	(total time spent processing in Javascript)
Graphics	21.35%	(total time spent in graphics processing)
Physical Simulation		
	2.8%	Edge-Parabola intersection
	0.64%	Quadratic equation solver
	0.30%	Edge-Point containment test
	0.38%	Particle sliding equation
	0.11%	Particle collision
A.I. Search		
	3.83%	Gravity turn sampling

Table 6.1: CPU Time breakdown for a solution search

Ranking Heuristic Performance Improvement

Our plan ranking heuristic presented in Chapter 5 produced great performance improvements during multiple-particle search. Here we present a table summarizing the running time difference between our implementation with and without this plan heuristic.

Test Piece URL	Number of Particles	Time without Ranking Heuristic	Time with Ranking Heuristic	Factor Improvement
index.html?demo	4	20.672 secs	7.009 secs	2.949×
yc.html?demo	2	100.947 secs	62.104 secs	1.625×

Constants

Our implementation uses the following values for simulation constants:

Constant Description	Symbol	Value
Particle Elasticity	κ	0.5
Perpendicular Edge Velocity	ϵ	0.9

Scenario #4 Rejection

Our preliminary tests indicate that our restriction on scenario #4 during simulation does not have a large effect on obtaining a solution. While we are not able to determine if workpieces exist in which every solution contains a scenario #4 particle path, we present a summary of the number of samples rejected due to this restriction below in Table 6.1.

Test Piece URL	Total Samples	Scenario #4 Samples Rejected	Percentage
index.html?demo	3400	854	25.12%
yc.html?demo	1200	264	22%

It is also important to note that manufacturers may prioritize obtaining *any* solution to the draining problem in a reasonable amount of time over obtaining *the* optimal solution (at the expense of time). If this is the case, it is reasonable to reject scenario #4 for performance reasons.

6.2 Future Work & Discussion

Further optimizations and improvements could be made to this work, namely to enhance the usability of the results and increase efficiency.

Heuristics for A* Search

One standard improvement to uniform cost search is the addition of a consistent, admissible heuristic to convert the search into A* search. A heuristic is simply an estimation of the remaining cost from a state to the goal state. Admissibility of a heuristic requires that the heuristic never over-estimates the remaining cost, and consistency requires that the heuristic does not rapidly change values between states.

Heuristics are, by nature, very dependent on the problem they are being applied to. Many heuristics exist for positional search problems and other common search formulations; since our search formulation is rather unique, there is not as much prior work to draw from [7].

One initial idea for a heuristic would be to use the free-fall time from the current concave vertex to the outside of the workpiece bounding box. This essentially would be the optimistic estimation that the particle would free-fall immediately from the concave vertex at the next turn. While this heuristic would be admissible and consistent, it does not take into account workpiece geometry. It is probable that better heuristics exist for this search formulation.

Kinetic Models

Another possible way to extend this work is to further expand on the kinetic model used during particle simulation. As detailed in Section 2.2, many complex terms from true particle dynamics were omitted from our kinetic model. By reformulating the kinetic model with additional terms like rotational inertia or the Coriolis effect, particle paths obtained from simulation would better represent true particle behavior.

Inclusion of these terms may jeopardize the time complexity of the algorithm. One possible way to incorporate more accurate particles while retaining performance is to first discover a solution with our simplified kinetic model and then verify that solution with a more complex kinetic model. If the previously obtained solution is no longer valid with a more complex kinetic model, that solution could then perhaps be intelligently modified towards validation. This approach may have better performance than an approach consisting purely of search based on a more accurate kinetic model.

Bounding Box Method Adaption for Particle Path Intersections

When intersecting rays with geometric primitives, most implementations using bounding-box methods to reduce the number of primitives tested against the ray. This bounding-box idea could be adapted to parabolic paths as well; determining which bounding boxes are needed still presents a fairly reasonable computational load, but great gains in efficiency could be made for complex input with many edges.

Adaption to 3D - Fixed Axis

Adapting this work to 3D polygonal meshes when the rotation axis of the workpiece is fixed is mainly an exercise in implementation. The primary difference is that parabolic paths would then need to be intersected with planes in 3D space rather than edges in 2D space.

The rest of the approach, from solution search to particle simulation, would remain the same.

Adaption to 3D - Free Axis

Adapting this work to 3D polygonal meshes where the workpiece can rotate about *any* rotation axis at any time presents a substantially more difficult problem. The chief obstacle here is that there is an explosion of possible turns available from any given concave vertex. The final gravity vector of the turn, g_{end}, could be sampled from an entire sphere rather than two small arcs in a planar circle.

There may be ways to intelligently reduce the sample space and still maintain a representative coverage of the kinetic paths attainable from a concave vertex, but much work is needed before those methods can be validated.

Chapter 7

Conclusion

We have provided a new approach to the workpiece draining problem from manufacturing that produces results that are complementary to existing work.

We first described our model of the problem – one that introduces a more accurate kinetic model of the water particles and allows for bi-directional draining with finite rotation speeds. We then introduced a parametric simulation approach that maintains constant-time intersection tests between particle paths and geometric primitives.

We then combined this simulation method with a unique search formulation of the draining problem to obtain a solution. By reducing the size of the state space and offloading complexity to the transition function, we can then "sample" from the action space until we are satisfied with the coverage of kinetic paths. This search formulation is then solved by standard uniform cost graph search; a solution post-processor then produces a control sequence that can be fed into machinery capable of rotating a workpiece.

Finally, we extended this work to draining multiple particles from within the workpiece and provided two optimizations that significantly improve search performance. We believe this work makes significant progress towards a fully-realized solution for manufacturers and designers alike.

Bibliography

[1] G. Aloupis et al. “Draining a Polygon–or–Rolling a Ball out of a Polygon”. In: *Proc. 20th Canad. Conf. Comput. Geom.* 2008, pp. 79–82.

[2] D. Arbelaez et al. “Cleanability of mechanical components”. In: *Proceedings of 2008 NSF engineering research and innovation conference.* 2008.

[3] M. Avila et al. “Design and manufacturing for cleanability in high performance cutting”. In: *CIRP 2nd international high performance cutting conference.* 2006.

[4] A.S. Glassner. *An introduction to ray tracing.* Morgan Kaufmann, 1989.

[5] Jeff Hancock. “Ultrasonic Cleaning”. In: *ASM Handbook Volume 5, Surface Engineering* 5 (1994), pp. 44–47.

[6] Steven. LaValle. *Planning Algorithms.* Cambridge University Press, 2006.

[7] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley Pub, 1984.

[8] J. Reza. *Theory of Applied Robotics: Kinematics, Dynamics, and Control.* Springer, 2010.

[9] S. Russel and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 2009.

[10] P. Shirley, M. Ashikhmin, and S. Marschner. *Fundamentals of Computer Graphics.* 3rd. A K Peters, 2009.

[11] Yusuke Yasui and Sara McMains. “Testing a rotation axis to drain a 3D workpiece”. In: *Computer-Aided Design* 43.7 (2011). The 2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling, pp. 795–808. ISSN: 0010-4485. DOI: `10.1016/j.cad.2010.05.004`. URL: `http://www.sciencedirect.com/science/article/pii/S0010448510000941`.