Modeling the Dynamics of Chalk: Methods for Efficient Approximation

Joshua Greenman
Rensselaer Polytechnic Institute

May 15, 2010

Abstract

The modeling of various materials is often a refined process, wherein a combination of actual physical properties and visual shortcuts are used to achieve an accurate rendition. Solids can be modeled with varying rigidity and elasticity; Liquids can be modeled with varying viscosity; Even gases and particle fields can be represented with varying densities. Often, rigid solids must be able to support permanent deformation—for some materials, denting is sufficient. For many others, breakage must be considered. We find an interesting case to be those materials that are constantly crumbling on a small scale. Blackboard chalk is one such material, wherein its use relies on a perpetual breaking off of particles to mark another surface. In this paper, we will be discussing efficient and inexpensive approximation methods to accurately model the behavior of Chalk under normal circumstances.

1 Introduction

Any given object—if intended to be modeled in a semi-realistic fashion—must be modeled under a given set of conditions which can be used to evaluate its exact state at any given timestep, stored by a set of elements representing the objects' current state. In order to accurately represent the state of the stick of chalk, we chose to use a 3-dimensional field of densely packed particles. As the chalk was modified by its environment, the particle field would update to represent the new condition and position of the chalk. So, then, each interaction between the chalk and its environment would be atomic; That is, each individual particle would interact with the environment, and it's adjacent particles, eventually representing the entire object. It is therefore relevant to enumerate the forces acting on each particle as we define our problem space.

Global	
$F_{gravity}$	Gravitational Force
Chalk	
$F_{applied}$	External Movement
F_{bond}	Bond to rest of Chalk
Chalkboard	
F_{normal}	Normal to Chalk forces
$F_{friction}$	Perpendicular to Normal

Table 1: Forces on any particle

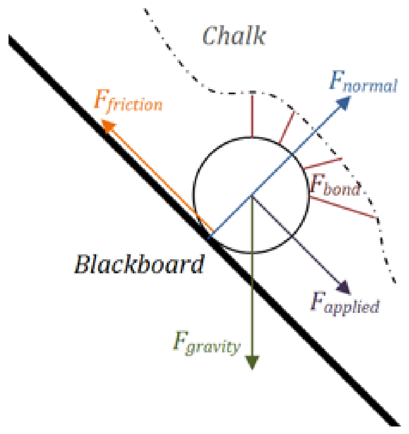


Figure 1: Chalk particle free-body diagram

As one might imagine, these forces (in conjunction with the current state of the particle at any given timestep) would be used to calculate the state (position & condition) of a particle in a given timestep.
Not surprisingly, a direct calculation of applied forces to each particles throughtout a dense mesh is quite computationally burdensome. However, given the dormant state of most non-surface particles in most environmental interactions, we believe an approximation that considers only surface particles will be sufficiently accurate, and much more practical.
In this paper, we will discuss the methods and approaches used to generate the initial representation of the chalk, to calculate the state of the chalk in a given timestep, and to represent the state of the chalk at any given time as a mesh. We will then review the overall results of these methods, identify specific challenges encountered along the way, and state our conclusions regarding our approach.

2 Generating the Initial Representation

In order to begin modeling our interactive chalk piece, we first had to create a field of particles that could accurately represent the particles making up the stick of chalk. Originally, we proposed generating a uniform field of particles, with surface particles flagged so that they might be the point of primary interaction when colliding with a chalkboard. In Figure 2, we can see the field of chalk particles generated in a cylindrical space, where red particles are "on the surface" of the chalk. Note that in this illustration, all surface particles are determined by their presence in the modified convex hull Surface solution discussed in QuickHull [1].

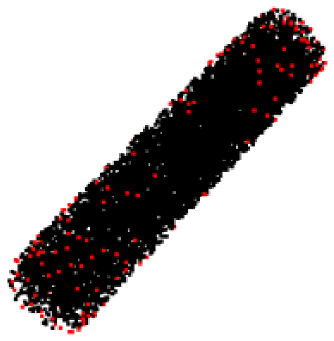


Figure 2: 10,000 particle cylinder

First, a 3-dimensional transformation matrix must be generated via inverse kinematics to determine the rotational transformation needed to place the chalk stick in the scene. Once such a matrix t exists, generation of particles is simple; given a center position c, orientation d, radius r, and chalk length L,generation of n particles is defined as:

```
For $n$ particles:
float $x$, $y = $ Rand[$-r$,$r$]
          where $x^2 + y^2 = 1$
float $z = $ Rand[$-\frac{L}{2}$,$\frac{L}{2}$]
store $(t * (x, y, z) + c)$
```

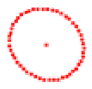


Figure 3: 3-D Particle Generation

As mentioned earlier, significant advantages are to be gained from minimal representation of the chalk. So, recognizing that chalk generally writes with its tip, and its sides are more-or-less non-interactive, we decided to make two major changes in representation:

1. Rather than even distribution throughout the volume of the cylinder, gather a dense plane of particles at the tip, perpetually representing the state of the chalk-tip's geometry

2. Because of the significance of the cylindrical shape and form of the body of the chalk, intentionally and evenly generate the circular edge particles to allow for a more consistent and realistic model of the shaft.

In Figure 4, we see a representaion of this form, where all particles are surface particles (black, because non-extremity, and intentionally generated edge particles are emphasized in red).

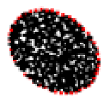


Figure 4: 1,000 particle approximation

Particle generation from Figure 3 is simplified here by setting the z-component for each generated particle to 0. A simple trigonometric function is implemented for selecting ring points at an evenly spaced θ.

In the next section, we will discuss a generic method for parsing the state of the chalk at a given timestep, and later implement the changes needed to support this improved representation.

3 Calculating Successive States

The change-of-state of the field of chalk particles is given by a rather complex algorithm, which can be simplified when broken down into the following four phases:

1. Calculate F_{normal} & $F_{friction}$ for this iteration based on the previous number of collisions
2. "Remove" all particles marked as having a broken bond from the previous iteration
3. Evaluate the state of all particles, validating surface penetrations and marking broken bonds
4. Update forces, & apply to chalk position

Before proceeding, let us immediately dispel any confusion related to the order of the steps above. If a particle is to contact a surface with enough force to potentially break its bond, it does not actually break off the chalk until the chalk is further moved. for this reason, we process those particles that are deemed "marked" in the following state, when we can also evaluate their actual connection to the chalk. Otherwise, chalk chould just disentegrate by being pushed continually downward, which is quite obviously inaccurate.
Let us further elaborate upon the details of each step.

1. Based on the number of collisions in the previous step, we calculate both the Normal and the friction to be applied in the upcoming iteration. If there were no collisions from the previous step, both of these values are 0, because there is no contact between the chalk and the surface. If there are any touching particles, we determine the friction acting based on the motion of the chalk, and divide it up amongst the total number of contacting particles. Note that, if motionless, the friction will instead be dependent on the perpendicular component of the net force—if this is less than the maximum force of static friction for these two surfaces, unless a bond breaks, the chalk will not move.

2. For all those particles marked as having their bond broken from the previous iteration, a mark is made on the surface, and the particle is finally "removed" from the chalk. Note that, in accordance with having a single field of particles representing the tip, we instead "receed" the particle into the body of the chalk, anti-parallel to the chalk's orientation, by twice the radius of a given particle. This achieves appropriate deformation of the surface mesh, but dodges the need for immediate surface recalculation. Given the small scale of the following diagrams, the success of this technique may be more visibly observed in Figure 8 in the results section.

3. The current particles must be evaluated to determine whether or not they lie on (or have crossed through) the contact surface. For each penetrating particle, the force of friction is compared to the force of the covalent bond holding the particle to the chalk. If the force is indeed large enough, a flag is set on the particle indicating its bond will be broken. In the diagram below, this will be represented by the color green.
If the force is not large enough to break

the bond, it may be possible that a correction of chalk positioning is in order—if the particle has penetrated the surface, but did not have enough friction force applied to actually break off the chalk stick, then it could not have possibly penetrated the surface in the first place. If the particle is on the surface, nothing needs to be corrected; however, anything further, and the entire mesh must be moved perpendicular to the surface, such that the point in question now lies ON the surface instead of through it. When this happens, the number of collided edges may change, and therefore must be recalculated. The friction is then divided up amongst the new number of collided particles, as this new number represents what the "actual" state of collision must have been. These such particles are marked below in blue.

Consider below the red identified penetrations. One is identified as impossible, and is marked blue. The blue particle is moved above the surface, and the cycle repeats. Eventually, red particles instead become green as they are deemed as proper collisions, and set up to be broken bonds in the next iteration.

Note that though a wireframe is visible in the picture below, it is only done so that the picture may be more discernable. At this point in time, the field of particles is just that, and has no real edges connecting them.

4. The only force that has been changed over the traversal of this iteration of the algorithm is the friction force, which may have been reduced by the breakage of bonds. To clarify, when a bond breaks, instead of the chalk experiencing that particle's fraction of friction resistance, it instead experiences resistance equivalent to the amount of force used to break the bond (always less resistance than the friction would have been, else the bond would not have broken).

From this actual-applied-friction value, we re-calculate the net force. Because the final net force is particle-independent, divide the Net Force by the chalk's mass to determine its acceleration. Determine the velocity by multiplying its acceleration by the timestep interval, and adding it to the current velocity. Finally, deterine the position by applying the velocity to the current position with the appropriate time interval. Create a transformation matrix from the difference between the target center position and the starting center position, and apply it to all particles.

Although admittedly arduous and confusing at first, this very thorough algorithm guarantees a relatively accurate approximation of state, assuming a reasonable timestep has been chosen (too small a timestep, and not enough force to create a broken bond will exist; too large a timestep, and too many particles for accurate determination will cross through the surface at once).

Finally, let us present both the officalized form of the algorithm, and also identify the computational time of this schema.

Note that determining current collisions at a timestep takes O(n) time. Also, consider the following abbreviations to be used throughout the algorithm specification:

$Chalkboard = cb$

$friction_{fraction} = f_f$

$f_{perpendicular} = f_{perp}$

$resistance_{actual} = resist$

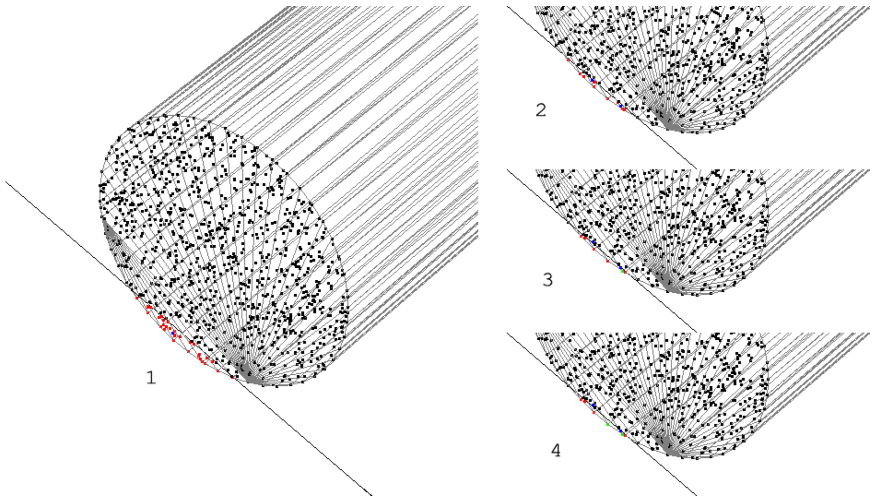


Figure 5: The progressive collision-parsing algorithm. Red particles are not-yet-validated penetrations; Blue particles are deemed invalid penetrations, and are moved up to the top of the colliding surface; Green particles are broken bonds.

3.1 Algorithm

```
$f_{normal}$, $f_{friction}$, $f_f$, $resist = 0$
if $current - collisions > 0$:
    $f_{normal} = (f_{gravity} + f_{applied}).Dot(cb_{normal}) * cb_{normal}$
    $f_{perp} = f_{normal}.Perp()$
    if $vel == 0$:
        if $f_{perp} <= f_{normal} * mu_{static}$:
            $f_{friction} = f_{perp}.Negate()$
        else
            $f_{friction} = f_{perp}.Norm() * |f_{normal}| * mu_{static}$
        endif
    else
        $f_{friction} = f_{perpendicular}.Norm() * |f_{normal}| * mu_{kinetic}$
    endif
    $f_f = f_{friction} / num_collisions$
endif
for all previous bond-broken particles $p$:
    $cb.mark(p)$
    $p.setCollidedFlag(false)$
    $p.receed()$
endfor
 for each particle $p$:
  if $p$ has collided:
    if $f_f < f_{bond}$:
      if $p$ is through $cb$:
        Determine $translationT$ between $p$ and closest point $p^1$ on
surface
        Apply $T$ to all particles
        Reset $resist$ to 0
        Reset $f_f$ to $f_{friction}/num.collisions$
      endif
      $resist+ = f_f$
    else
      $p.setCollidedFlag(true)$
      $resist+ = f_{bond}$
    endif
  endif
endfor
$f_{friction} = resist$
$f_{net} = f_{gravity} + f_{applied} + f_{normal} + f_{friction}$
$acceleration = f_{net}/mass_{chalk}$
$velocity = velocity + acceleration * (timestep)$
Translation $T = velocity * (timestamp)$
Apply T to all particles
```

In the actually impossibe worst case of each and every particle being both a bond-breaking collision, and non-bond-breaking (mesh shifting) collision, the time for this algorithm is $O(3n) + O(n^2) = O(n^2)$. However, in a realistic average case, it would most likely be $O(4n) = O(n)$.

4 Interpolating the Surface Mesh

As mentioned earlier about the Figures thus far, the wireframe models shown are not actually an implicit part of the scene data, but are rather intentially displayed to augment the viewer's understanding of each given illustration. It is not surprising that, just as we have needed to display a mesh to make the illustrations of this paper more legible, that we also must display a mesh to make the deliverables of this paper comprehensible. Therefore, we must implement a method of creating a full mesh from just a collection of otherwise unassociated particles.

This problem was first tackled on a 2-dimensional plane, by R.A. Jarvis in 1973, who proposed a "Gift-Wrapping Algorithm" that could effectively place a large rubber-band around any given set of 2d pegs. Since then, many schemes have been developed to achieve similar effects, in both 2 and 3 dimensions. Now, we know this as the Convex Hull algorithm, and there are many modern variations and optimizations that can be employed, varying by the data to be wrapped.

A basic form of Convex Hull was used to wrap the chalkstick represented in Figure 1 (though the mesh itself is not shown, the vertices selected to compose the mesh are shaded red). The algorithm works as follows:

```
   1.  Select two random, unique points $p_1$ and $p_2$
2.  Select a third point $p_3$, not colinear with $p_1$ and $p_2$, forming a
triangle with those two points
3.  Select a fourth point $p_4$, not coplanar with $p_1$, $p_2$, and $p_3$, forming
a tetrahedron with those three points
4.  For every point $p_n$:
   if ($p_n$) is outside the currently enclosed area
      for each triangle with its normal facing $p_n$:
         store any edges that share faces whose normals do NOT face
$p_n$ (silhouette edges)
         delete all other edges, along with this triangle
      endfor
      for each silhouette edge:
         create a triangle between its two vertices, and $p_n$
      endfor
   endif
endfor
```

Consider the diagram below, illustrating the original tetrahedron, following iteration, and followed by iterations 16 and then 35.

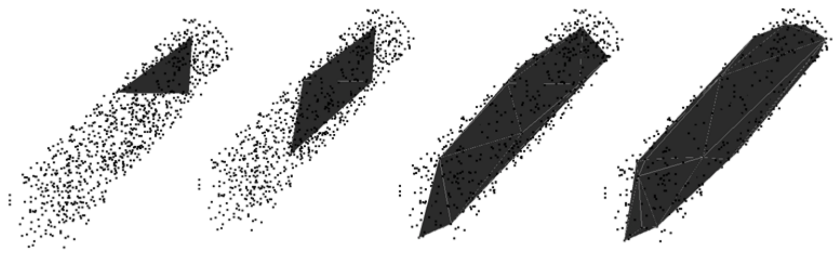
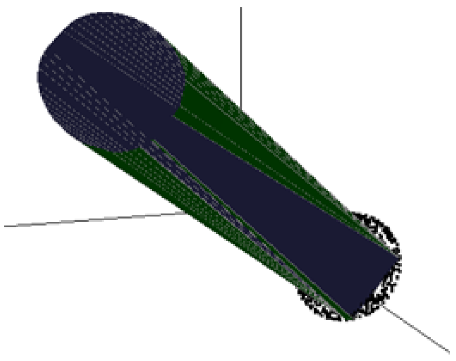
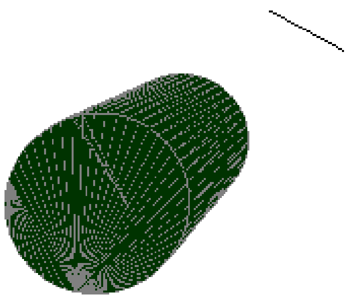


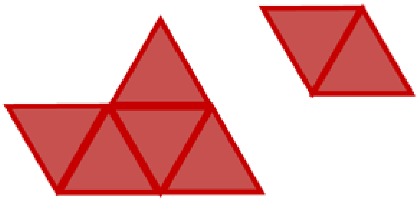
Figure 6: The progressive succession of the convex hull algorithm.

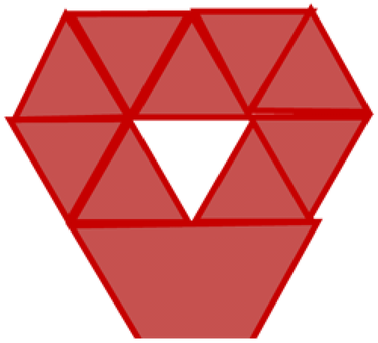
As one can see, convex hull works quite nicely for enlarging these uniform particle fields. However, our optimization to the chalk model have introduce an interesting set of complications:
Because our model is quite angular, particularly at creation, finding normals that face any given point can become quite troublesome. In our situation, floating point error begins to become significant: if many edges are just-about-90-degrees in angle, it is difficult for floating point to consistently recognize this. Consider the Figures below, which are examples of failed convex hull calculations as a result of this precision error.





Therefore, when choosing which edges to "silhouette" the mesh at any given state during convex hull, we must find a reliable method of preventing mistakes. Implementing a static error margin is actually quite inconsistent, and presents a similar inconsistency in completion of the hull. However, by observing the number of silhouette edges found relative to the number of "facing" triangles identified, we can use two relatively speedy modifications to guarantee a safe iteration.
The primary observation leading to this methodology was the realization that, given n triangles, if all triangles are contiguous—and there are no holes—there cannot be more than $n+2$ silhouette edges. So, if at any iteration, more than $n+2$ silhouette edges exist, it means at least one of the following two scenarios has occurred:
A) Multiple discontiguous triangles or triangle clusters exist
B) A "hole" of one or more triangles exists in the cluster.





First, let us propose a method to resolve A, considering list L of silhouette edges: For each silhouette edge, enumerate all of its neighbor silhouettes until its entire cluster has been closed. Do this for all clusters. Retain the cluster with the highest count, and discard all other clusters.

Second, let us propose a method to resolve B, considering the lists generated in A, and a new list T of triangles contained by L: For every remaining edge in M, locate the single triangle adjacent to the opposite of the edge. If the triangle is adjacent to any triangle in L, this is an internal hole. Add all triangles adjacent to this triangle not already in L until no more such triangles exist.

5 Results

Before we can begin to summarize our results, we must first enumerate the variables that played heavily into the actual calculations. The chalk input file allows the bond force, particle mass, particle radius, cylinder resolution (number of sides), and surface resolution to be set. The chalkboard input file allows the coefficients of kinetic and static friction to be set.

Out of these, the only variables that can influence the validity and general success of the results (assuming reasonable size and positions for chalk and chalkboard) are bond force, particle mass & raidus (i.e. density), and coefficients of friction. In order to minimize error from these three degrees of freedom, I attempted to find official determination for these values, but was unable to find a reliable source. I was, however, able to locate a class project performed by a few high-school students [2], and average their results to obtain reasonable value for these variables.

$Density(kg/m3) = (1100 + 1100 + 1800 + 1700)/4 = 1425kg/m3$

$Coefficientofkineticfriction = (0.68 + 0.68 + 0.66 + 0.63 + 0.52)/5 = 0.634$

Not surprisingly, the "force of covalent bond" isn't exactly an accurate paradigm for the force of holding a particle to a stick of chalk, but rather was a convenient visualization for the sake of this project. So, this ends up as our only entirely undetermined variable.

We also needed to play around with our timestep to find a safe quantity. The safest timestep ended up being 0.02 - 0.03 seconds, wherein no more than 20 and no less than 5 particles on average crossed the surface in any given timestep. Given the procedure our algorithm follows, this seems to be ideal.

Finally, when we used a value of 0.15 for f_{bond} at a timestep of 0.02 seconds, we got the following images:

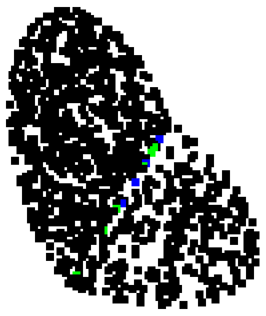


Figure 7: Displacement of the chalk surface. Blue and green particles are recently either moved or receeded.

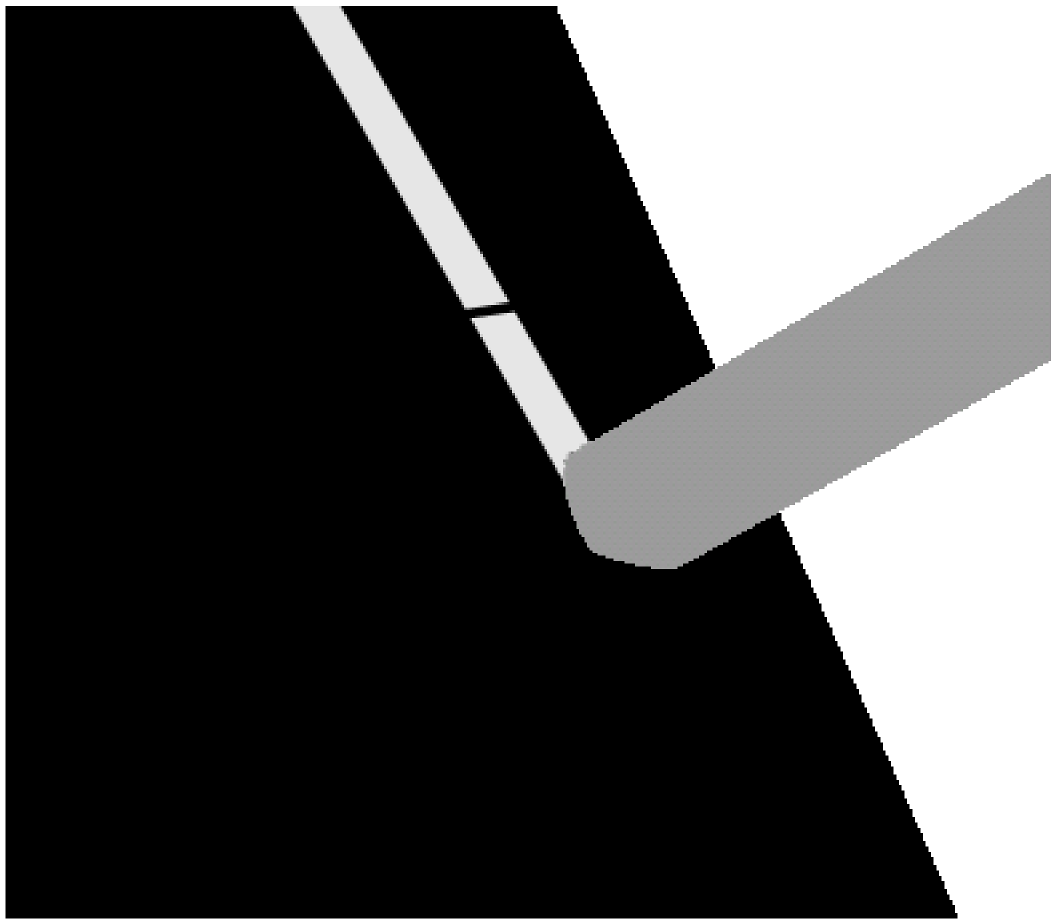


Figure 8: The final rendering.

6 Challenges & Limitations

Listed below here are some of the more frustrating technical challenges I faced.

- Random appearances of NaN
- "Painted" line on chalkboard showed up randomly
- Continuous issues with repairing opposites in Half-edge
- My T43p was not built to deal with 10,000 particles

Limitations with this approach included an inability to accurately model sideways strokes, as well as more-or-less of a guessing game with various constants.

7 Conclusion

The approximation scheme used to emulate chalk particle destruction seems viable, although my implementation is rather unreliable and unstable. I imagine future work in cleaning up the codebase would significantly stabilize results, and would also allow for an exploration of topics such as breaking and larger fracturing.

I learned quite a bit from this project—among those things, that this was far too large of a project to undertake alone. Regardless, it was a phenomenal learning experience, and I hope to continue to explore the benefits of semi-realistic approximations in computer graphics.

References

[1] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Trans. Math. Softw.*, 22(4):469–483, 1996.

[2] C. Ting, C. Lok, L. See, M. Ka, and M. Long. Talking chalk. *St. Francis Xavier's College*, 1(1), 2002.