Convex Optimization Final Paper
Simulated Folding of Origami Crease Patterns

Gabriel Eiseman

May 10, 2019

1 Introduction

Origami, the art of making things by folding paper without cutting or gluing, is, in addition to being a fun hobby, related to a large number of interesting mathematical problems. These range from geometric questions like what is the maximum perimeter of a shape that can be folded from the unit square, to theoretical computer science connections like the NP-completeness of determining whether or not a crease pattern is flat-foldable, to the problem we consider here of physically simulating a crease pattern. There is quite a lot of jargon related to origami, a necessary subset of which we will now introduce.

Although the no cutting and no gluing restrictions are pretty much universal, other restrictions are not agreed upon, namely, can more than one piece of paper be used and can shapes besides squares be used? The solver we discuss in this paper could handle multiple pieces of paper of any shape including shapes which include cuts (and I'm personally willing to allow any number of simply connected pieces of paper (simply connected meaning lacking holes)).

Origami is a very repeatable art. Similar to music, there is a standard way of writing down the steps (called "folds") to create an origami "model". Origami instructions are less open to artistic interpretation and more similar to a recipe or even chemical formula: in the beginning of creating the model the instructions are effectively an exact mathematical description of how to make the model. Standard models that are supposed to resemble things like a dragon or a jack-in-the-box tend to start with the construction of a "base" which has the correct appendages and body segments in the correct proportions and positions and then apply shaping and the only interpretation would come in when doing the shaping or possibly when choosing variable proportions of the base. Other models are more geometric, these are often called "tesselations" and usually have some "unit" that repeats over a flat surface. Tesselations can be extended indefinitely in most cases. Every origami model consists of a planar graph of creases on the paper. The edges of this graph that are on the edge of the paper are called "raw edges" and the other edges are creases which are straight lines between some vertices of the graph (curved creases cannot really occur because the Gaussian curvature of paper is zero). This graph is called a "crease pattern" and for an expert it can be enough to construct a model without step by step instructions. Crease patterns usually indicate whether each crease is a "mountain" or "valley". Paper has two sides, which are sometimes referred to as the "colored", "top", or "preliminary" side and the "white" or "bottom" side. "Mountain" folds point upwards and "valley" folds point downwards, although this can be confusing sometimes if the paper is flipped or a fold is a mountain fold in the crease paper but it's being made on a part of the model that is inverted so it is actually a valley fold. Every fold in the instructions for an origami model is composed of mountains and valleys, although it is not always possible to make them using only a sequence of mountains and valleys.

A crease pattern defines the positions of vertices as coordinates in $\mathbb{R}^2$, but generally we want folds to have "landmarks" or ways of finding exactly where they should go. For mountains and valleys, there are seven ways of specifying landmarks including "make a fold through two existing points" and so on. These seven ways of constructing mountains and valleys are called the "Huzita-Hatori Axioms" and are the math behind the claim that origami is more powerful than compass and straightedge geometry. Working with an idealized compass and idealized straightedge and starting with a segment of unit length, one can construct any element of the closure of the rationals under the field operations and square roots. Working with an idealized unit square and idealized folding, one can also solve cubic equations in addition to quadratics. Here we are not concerned with landmarks and just take the crease pattern as a given.

In a crease pattern, every crease has a dihedral angle in the final folded model, from 0 for a completely closed valley to 2π for a completely closed mountain (in this case, the two are not exactly the same). If a final folded state with all dihedral angles 0 or 2π is reachable, the model is called "flat foldable". Some tesselations, including the waterbomb base tesselation we will consider later, are not flat foldable. The crease pattern for these would traditionally still only specify whether each crease is a mountain or valley (if that) and not its final folded dihedral angle.

For the purpose of this paper we will assume that we are given a crease pattern and the final dihedral angle for all folds. The goal is to model how the paper folds so we can simulate it or identify a reasonable approximation to the final folded state. One way to do this is by assigning every possible configuration of the vertices of the crease pattern an energy and minimizing this energy, which is what we will eventually do.

2 Related Work

I initially had the idea for a program to simulate origami last semester in quantum field theory while learning about Lagrangians. It turns out another, better, implementation of crease pattern simulation exists (available online at `http://www.amandaghassaei.com/projects/origami_simulator/` with a paper available at `http://www.amandaghassaei.com/files/projects/origami_simulator/FastInteractiveOrigamiSimGPU.pdf`). I avoided reading too much about this because I wanted to be able to figure it out on my own.

We want to model the folding of a crease pattern as a physical system, so we want to describe its potential energy or equivalently the forces involved in the system (we care about finding a final state with no motion and hence no kinetic energy so we know $\vec{F} = m\vec{a}$ and $\vec{F} = -\nabla E$ which lets us relate force and equivalently acceleration with potential energy) (ironically I did not try to solve this problem using Lagrangians, it was just the potential energy part that carried over).

We can imagine every crease in the crease pattern exerting a force on the paper to move it towards its final folded angle. Since we want this force to move the creases towards some angle, we can model each crease as a torsion spring with θ_0 as the final folded angle of that crease.

This actually takes care of the forces/potentials we need to describe the effect of the crease pattern in folding up the paper, but it totally ignores the material characteristics of the paper itself. We know paper has Gaussian curvature zero, and the only Gaussian curvature zero surfaces are extrusions of curves, but that does not easily translate to a potential energy or force. If we assume the faces of the crease pattern are all triangles (which we can do by introducing extra creases with final folded angle pi), then while any edge is in between flat (π) and fully folded (0 or 2π), the edge itself is bending so there cannot be any bending orthogonal to the edge or the curvature would not be zero. But since there cannot be any bending orthogonal to any of the three edges of the face, we can conclude the face must be flat.

However, we also know that paper does not really stretch or compress. Materials are usually modeled as having a stress-strain curve that is linear below a certain threshhold (called the "elastic domain"), even materials we normally think of as much less stretch than paper, such as steel. Stress is force divided by unstressed cross section ($\sigma = \frac{F}{A_0}$) and strain is relative change in length ($\epsilon = \frac{l-l_0}{l_0}$). If we model paper as a material with an unbounded elastic domain, we will actually get that the edges of the crease pattern are linear springs obeying Hooke's law, and this means our model takes care of the forces caused by the crease pattern folding up the paper and also of the paper resisting shearing and stretching in plane. We will get an energy as a function of the positions of all the vertices of the crease pattern, so we can find the configuration of vertex positions that minimizes this energy, and equivalently we can find the force on each vertex in any given configuration in order to simulate the system. More details will be provided in the next section.

3 Optimization Problem

We want to find the configuration of vertex positions that minimizes the potential energy of our system. Obviously the set of optimal energy configurations decomposes into equivalence classes under rotations and translations in $\mathbb{R}^3$ so there will not be a unique minimizer. We are modeling the paper as a perfectly elastic material with zero thickness and zero mass aside from a unit mass at each vertex (a slightly better model would be uniform density, but for many tesselations including the waterbomb base tesselation we will be testing on, there is little or no difference in the sums of the areas of the faces connected to each vertex so our mass-lattice approximation is not too far off, plus it essentially amounts to tweaking the spring constants so they are not the same). Since this hypothetical paper is perfectly elastic and has zero thickness, it has a uniform elastic modulus

$$\rho_E = \frac{\sigma}{\epsilon} = \frac{F l_0}{A_0(l - l_0)}.$$

We want to get a spring constant out of this, so we plug in Hooke's law $F = k(l - l_0)$ to get

$$\rho_E A_0 = \frac{k(l - l_0)l_0}{l - l_0} \Longrightarrow k = \frac{\rho_E A_0}{l_0}$$

so we see every linear spring should get a spring constant inversely proportional to its unstressed length. Formally $A_0 = 0$ since our paper has zero thickness, so we will just take $\lambda_l = \rho_E A_0$ to be one parameter. Now Hooke's law tells us the potential energy for the linear spring between vertices i and j is

$$E_{l:i,j} = \frac{\lambda_l}{2l_0}\left(\|\vec{r_i} - \vec{r_j}\|_2 - l_{0:i,j}\right)^2$$

where $l_{0:i,j} = \|\vec{r_{0i}} - \vec{r_{0j}}\|_2$, $\vec{r_{0i}}$ is the initial position of vertex i, and $\vec{r_i}$ is the final position of vertex i (an optimization variable).

Now we need to derive the potential energy for the torsion springs. This is partly simpler because it does not depend on any material properties of the paper, it is more of a driving force we made up to replace a person folding the paper, but it is also more complicated because it involves angles. If we are applying a force to move all creases to their final folded angle, this force should be proportional to the length of the crease, so our spring constant for the torsion springs should be proportional to the length of the crease each torsion spring corresponds to. Hooke's law for torsion springs tells us $E = \frac{\kappa}{2}(\theta - \theta_0)^2$ where $\kappa = \lambda_\kappa l_0$ and θ_0 is the final folded angle. λ_κ is the linear torsion spring constant density, another parameter.

However, we still need to express θ in terms of the positions of the vertices of the crease pattern. If we have two triangular faces of the crease pattern with counterclockwise vertices a, c, and d; and b, d, and c which obviously meet at the crease edge from d to c with final angle (from the top side where these labelings are counterclockwise) θ_0, we can calculate the current angle given the positions of a , b, c, and d. First, if we define $\hat{n_2} = \widehat{\vec{r_c} - \vec{r_d}}$ as the unit vector in the direction from d to c, where $\hat{\cdot}$ denotes vector normalization, we have a vector in the plane of both triangles, which will help us get an outward normal for each using cross products. Now define $\vec{n_1} = \vec{r_a} - r_d$ and $\vec{n_3} = \vec{r_b} - \vec{r_d}$ and now we can find the outward normals $\hat{n_a} = \widehat{\hat{n_2} \times \vec{n_1}}$ and $\hat{n_b} = \widehat{\vec{n_3} \times \hat{n_2}}$. By construction, $\hat{n_a}, \hat{n_b} \perp \hat{n_2}$ so the volume of the oriented parallelopipid with edges $\hat{n_a}$, $\hat{n_b}$, and $\hat{n_2}$ which is $(\hat{n_a} \times \hat{n_b}) \cdot \hat{n_2}$ reduces to $\|\hat{n_a} \times \hat{n_b}\|\|\hat{n_2}\|$ since $\hat{n_a} \times \hat{n_b}$ must be a scalar multiple of $\hat{n_2}$. Since $\|\hat{n_a} \times \hat{n_b}\| = \|\hat{n_a}\|\|\hat{n_b}\|\sin(\theta)$ and $\hat{n_a}$, $\hat{n_b}$, and $\hat{n_2}$ are unit vectors, this further reduces to $\sin(\theta)$. You might think that we could just take the inverse sine of this number, but you would be wrong of course. Inverse trigonometric functions generally only have a range over half of the interval $[0, 2\pi)$ but we want a function with the full range. Note that also $\hat{n_a} \cdot \hat{n_b} = \cos(\theta)$ so we can write $\tan(\theta) = \frac{(\hat{n_a} \times \hat{n_b}) \cdot \hat{n_2}}{\hat{n_a} \cdot \hat{n_b}}$. This means if we use the range-corrected two argument inverse tangent function $\operatorname{atan2}(y, x)$ we can recover θ over $[0, 2\pi)$ as desired. $\operatorname{atan2}(y, x)$ is $\arctan(\frac{y}{x})$ for $x > 0$, $\arctan(\frac{y}{x}) + \pi$ for $x < 0 \wedge y \geq 0$, $\arctan(\frac{y}{x}) - \pi$ for $x, y < 0$, and $\frac{\pi}{2\operatorname{sgn}(y)}$ otherwise, where $\operatorname{sgn}(x)$ is the sign function, -1 for negative arguments, 0 for 0, and $+1$ for positive arguments. We still will not be able to distinguish 0 from 2π though so we will need to have an error tolerance for that. So the potential energy for the torsion spring between faces acd and bdc is

$$E_{t:d,c} = \frac{\lambda_t l_{0:d,c}}{2}(\theta_{d,c} - \theta_{0:d,c})^2$$

where $\vec{n_1} = \vec{r_a} - r_d$, $\hat{n_2} = \widehat{\vec{r_c} - \vec{r_d}}$, $\vec{n_3} = \vec{r_b} - \vec{r_d}$, $\hat{n_a} = \widehat{\hat{n_2} \times \vec{n_1}}$, $\hat{n_b} = \widehat{\vec{n_3} \times \hat{n_2}}$, and

$$\theta_{d,c} = \operatorname{atan2}((\hat{n_a} \times \hat{n_b}) \cdot \hat{n_2},\ \hat{n_a} \cdot \hat{n_b}).$$

Note that since we triangulated the crease pattern there is only one possible pair a, b for any (non-raw) edge dc. We also define $E_{T:d,c}$ similarly but with λ_T instead of λ_t; both of these are parameters.

Adding up these energies for all edges in the crease pattern gives us the total potential energy which is what we want to minimize.

We have the given variables $\vec{r_{0i}}$ and $\theta_{0:d,c}$, parameters λ_l, λ_T, and λ_t (for linear springs, synthetic crease edge torsion springs, and ordinary crease edge torsion springs, respectively), and optimization variables $\vec{r_i}$ and we want to minimize:

$$E = \sum_{\{i,j\}\in edges} E_{l:i,j} + \sum_{\{d,c\}\in edges_{\text{oc}}} E_{t:d,c} + \sum_{\{d,c\}\in edges_{\text{sc}}} E_{T:d,c}$$

where $edges$, $edges_{\text{oc}}$, and $edges_{\text{sc}}$ are the sets of all edges in the crease pattern, all ordinary crease edges, and all synthetic crease edges (added when we force the crease pattern to be a triangulation), respectively, and $E_{l:i,j}$, $E_{t:d,c}$, and $E_{T:d,c}$ are as defined above.

There are (kind of) no constraints. Really, we want the paper to not self intersect, but since we require the final folded angles to be specified, these final folded angles should already specify a configuration that is not self intersecting so we do not need to explicitly require this. We also want the folded state the solver gives us to actually be reachable from the flat paper initial state, but topology tells us that any two nonintersecting simply connected surfaces are homotopic so we know any nonintersecting solution will be reachable and this would not generate additional constraints. The explicit constraints for nonintersection would be for every pair of nonadjacent faces auv and bxy, we have normal vectors $\hat{n_a}$ and $\hat{n_b}$ so we can get the direction vector of the line of intersection $\hat{n_c}$ and then get tangent vectors inside each plane pointing from $\vec{r_a}$ to the intersection and $\vec{r_b}$ to the intersection $\hat{t_a}$ and $\hat{t_b}$. We can find the exact intersection $\vec{p}$ by solving the system of three equations $\vec{p} = \vec{r_a} + \lambda\hat{t_a} = \vec{r_b} + \mu\hat{t_b}$. Then we take $\max\limits_{f\in\{auv,bxy\}} (\prod_{i\in f}(\vec{r_i} - \vec{p}) \cdot \hat{t_b})$ which ensures at most one of the triangles is split in half by the plane of the other triangle: if both are, they intersect internally and not on an edge which implies a self intersection of the paper. Because this condition is complicated, there are quadratically many pairs of nonadjacent faces, and valid final folded angles already imply a folded model that is not self intersecting, we did not actually use these constraints.

4 Convex Form

Now we have to prove the potential energy function we constructed is actually convex in terms of $\vec{r_i}$. It will suffice to show that $E_{l:i,j}$ and $E_{t:d,c}$ are always convex since the sum of convex functions is convex. Notice that both of these functions (linear spring energy and torsion spring energy) are invariant under translation and rotation.

First, consider $E_{l:i,j}$. Because it is invariant under rotation and translation, we can assume without loss of generality that $\vec{r_i} = \vec{0}$ and $\vec{r_j} = l\hat{e_1}$ where $l \in \mathbb{R}_{\geq 0}$ and $\hat{e_1}$ is the first standard basis vector. Then $E_{l:i,j} = \frac{\lambda_l}{2l_0}(l - l_0)^2$ which is clearly convex, so since we can show for every $\vec{r_i}$ and $\vec{r_j}$ there is an invertable affine transformation on which $E_{l:i,j}$ is convex, a convex function applied to an affine transform is convex, and any fixed invertable affine transform can be turned into any affine transform by some affine transform, then $E_{l:i,j}$ is convex.

Next, we need to show $E_{t:d,c}$ is convex. This time, the function is also invariant under scaling, so first we can assume that $\widehat{\vec{r_c} - \vec{r_d}}$ is pointing in the $-z$ direction with $\hat{n_a}$ pointing in the $+y$ direction (by rotation), then that $\vec{r_d}$ and $\vec{r_c}$ are on the z axis (by translation). Now we will want to show that $E_{t:d,c}$ is convex on this restricted, representative domain. We have made the edge we are calculating the dihedral angle of the $+z$ axis, and we have made the first plane the xz plane, so the dihedral angle is the angle from the $+x$ axis to the ray from the origin to the (x, y) coordinate of $\vec{r_b}$ in the xy plane. So we reduced $E_{t:d,c}(\vec{r_a}, \vec{r_b}, \vec{r_c}.\vec{r_d})$ to $E_{t:d,c}(x, y) = \frac{\lambda_t l_0}{2}(\text{atan2}(y, x) - \theta_0)^2$. If $y > 0$, we can assume $y = 1$ (by scaling). Then the function reduces to (ignoring the leading constant) $f(x) = (\text{atan2}(1, x) - \theta_0)^2 = (\arctan(\frac{1}{x}) + /fracpi2(1 - \frac{x}{|x|}))^2$. $y > 0$ corresponds to the $\theta < \pi$ case, but even graphing this for values of $\theta_0 < \pi$ we see it is not convex! However, we see that around the minimum, which always exists and is zero, this function is convex, and we also see that it is always decreasing towards the minimizer. This shows that gradient descent and some other convex optimization techniques can still be applied to this problem. Precisely, we can calculate that the minimizer of $f(x)$ is $x_0 = \cot(\theta_0)$ and so the second order Taylor expansion of $f(x)$ about x_0 is $\frac{2}{(1+x_0^2)^2}(x - x_0)^2$ (see `https://www.desmos.com/calculator/wtu2hjurrn`). Something similar happens for $y < 0$. Since there is an invertable affine transform so that $E_{t:d,c}$ applied to it is quasiconvex, $E_{t:d,c}$ itself is quasiconvex (and convex within a neighborhood of the optimal solution). As stated this is good enough for some techniques including gradient descent and the simulation technique we will be using. A physical simulation cannot gain energy, so the simulation will explore points in the epigraph under the energy line, which decreases over the course of the simulation for various reasons that will be discussed in the next section. This will eventually force the positions of the crease pattern vertices into a configuration within the convex part of the domain of all the torsion spring constraints.

5 Convex Solver

I decided to use explicitly coded simulation to solve this problem, which is a form of gradient descent because in a physical simulation we have $\vec{F} = -\nabla E$. A physical simulation will also mimic the homotopy between the unfolded and folded sheets meaning it is easy to avoid self intersections using this approach. Most convex optimization techniques are iterative and simulation is no different. Every vertex will now have not only a position $\vec{r_i}$ but also a velocity $\vec{v_i}$. The velocities add another kind of energy to the system, kinetic energy $E = \frac{1}{2}mv^2$. Initially, all velocities and hence kinetic energy is zero, and we can see that Hooke's law conserves energy: for example, in a single spring system, we have one end of a massless spring with spring constant k and unstressed length $x_0$fixed at 0 on the real line and a mass m on the other end with position x and velocity $\dot{x}$:

$$E = \frac{1}{2}k(x - x_0)^2 + \frac{1}{2}m\dot{x}^2$$

we get $F = k(x - x_0)$ by Hooke's law, so we introduce an acceleration $\ddot{x}$ according to $-F = m\ddot{x}$ and we compute

$$\dot{E} = k(x - x_0)\dot{x} + m\dot{x}\ddot{x} = k(x - x_0)\dot{x} + m\frac{-k(x - x_0)}{m}\dot{x} = 0$$

as desired. This conservation of energy extends to torsion springs, multispring systems, our system, and any system with forces defined as the negative gradient of potential energy.

Explicitly, for the linear spring between i and j we have the force

$$\vec{F} = -\nabla E_{l:i,j} = -\frac{\lambda_l}{2l_0}(\|\vec{r_i} - \vec{r_j}\| - l_0)(\widehat{\vec{r_i} - \vec{r_j}}),$$

half of which should be applied to i and half of which should be negated and applied to j.

For the torsion spring between d and c, we know the torque instead:

$$\vec{\tau} = -\frac{\lambda_t l_0}{2}(\theta - \theta_0)\hat{n_2},$$

but $\vec{\tau} = \vec{r} \times \vec{F}$, so we get the signed magnitude of the force is $F = -\frac{\lambda_t l_0}{4r_0}(\theta - \theta_0)$ along $\hat{n_a}$ for a and along $\hat{n_b}$ for b. "Signed magnitude" means the force is a scalar multiple of the stated unit vector but possibly a negative scalar multiple. Note that I already divided the force for each of the vertices by two in the above equations, and r_0 is the initial distance from $\vec{r_a}$ to the axis for a and similar for b; this distance can be found as the magnitude of the projection of $\vec{r_a} - \vec{r_d}$ onto the plane with normal $\hat{n_2}$.

These are all the forces in our model, and since we modeled the paper as being massless except for a unit mass at each vertex, the acceleration of each vertex is just the sum of the forces on it (ie not that sum divided by the mass since the mass is one). Every frame of the simulation we can add the time step of that frame times the velocity of each vertex to its position and then the acceleration of each point (computed according to all the forces acting on it in our model) to its velocity. Initially I thought this was all that was needed to achieve a simple simulation, but it is actually missing something fairly obvious.

We saw that all the forces we are using are conservative, meaning they do not change the total energy of the system, but we start in a state with nonzero energy and want to reach a state with zero energy. This is impossible of course. In the single spring example previously, the energy oscillates between kinetic and potential energy forever, exactly the same as a ball rolling up and down in a frictionless half pipe. So our simulation could move erratically around the subset of the configuration space with the initial energy or cycle through some subset of configurations periodically, but the only way it could ever reach the desired final folded state with zero potential energy would be if all its energy became net kinetic energy. In none of these scenarios does it reach zero energy since in fact it never decreases the energy, but there are two important ways we can cause the energy of the system to decrease over time, both of which I alluded to in the previous sentence.

Firstly, we can subtract off net velocity and rotation every few frames, which decreases energy over time, prevents the case where all energy becomes net kinetic energy, and therefore conveniently prevents the model from flying out of the viewport we are rendering during the simulation or spinning uncontrollably.

Secondly, we can add some kind of "friction". True friction opposes the movement of an object which is pressed against a surface and appears as $F_k = -\mu F_N$. For a mass m in a uniform gravitational field g on a flat surface, this would reduce to $F_k = -\mu mg$, but regardless this is a constant force which would produce a constant "drag" acceleration, but this is not ideal because in our simulation the velocities will get smaller over time so once they are smaller than the acceleration (decceleration) due to friction times the time step of a single frame, the simulation will stop moving the model and just sit there.

So we would like the "friction" to depend on velocity so it is smaller for smaller velocities. Two other kinds of "friction" besides kinetic sliding friction we could try (ie nonconservative forces) we could draw inspiration from are drag, which is quadratic in velocity but also very complicated, and cooling, which in an absolute zero ambient is linear in temperature. I ended up multiplying the velocity of every point by 0.99997 or so every frame which can be thought of as a friction force which is decreasing over time or, more accurately to the rate of energy loss, losing heat to the environment. This is an interesting comparison because we're already removing the net kinetic energy, and temperature can be thought of as average kinetic energy, although this is really only accurate for a system with an extremely large number of particles and the model we are testing with has 32 vertices which is not quite large (not any significant fraction of a mole).

The exact equations for removing net kinetic energy involve the center of mass, net velocity, moment of inertia, net angular momentum, and change of coordinates into a rotating reference frame. First, we compute the center of mass

$$\vec{R} = \frac{1}{M}\sum_i m_i \vec{r_i}$$

where $M = \sum m_i$ is the net mass, but in our case $m_i = 1$ and M is simply the number of vertices. We then subtract this from every vertex's position. Next, we compute the net velocity (really we compute net momentum to find the velocity of the center of mass) as

$$\vec{V} = \frac{1}{M}\sum_i m_i \vec{v_i}.$$

We then subtract this from every vertex's velocity, which decreases the kinetic and hence overall energy. Finally, we have a system with center of mass stationary at the origin so we can more easily find and cancel its angular motion about its center of mass. We compute net angular momentum

$$\vec{L} = I\vec{\omega} = \sum_i \vec{r_i} \times m_i \vec{v_i}$$

so we can find net angular velocity as

$$\vec{\omega} = \frac{\sum_i \vec{r_i} \times m_i \vec{v_i}}{\sum_i m_i v_i^2}.$$

We then subtract $\vec{\omega} \times \vec{r_i}$ from each velocity $\vec{v_i}$, and we subtract $2\vec{\omega} \times \vec{v_i} - \omega^2 \vec{r_i}$ from each acceleration $\vec{a_i}$, although if we do this in between frames instead of in the middle of a frame we do not need to adjust the accelerations. This also decreases the kinetic and hence overall energy (although if we do it in the middle of a frame it will introduce a fictitious centripetal force and therefore possible decrease the energy less).

Adding "friction" to the simulation also has the benefit of decreasing the likelihood of 0 and 2π creases overshooting. In a single spring system, if we start at $x_0 - \delta$ we will overshoot to $x_0 + \delta$. For a multispring system we have a similar overshoot, except if it is a 0 or 2π crease and we overshoot its dihedral angle snaps to 2π or 0 respectively and causes the torque to flip. This is not only nonphysical because it corresponds the two faces on the sides of the crease passing through each other, but also because it adds a lot of energy to the system. The torsion springs have a small tolerance around 0 and 2π where they will not apply torque which helps this a little, but the tolerance is $\pm 1 \times 10^{-9}$ radians and in a frictionless single spring system the overshoot is the same as the initial deviation, which can be up to π in this case. A lot of the potential energy from torsion springs is lost to net velocity along the direction pointing away from the crease from the side from which it is a valley fold, and friction additionally helps to prevent overshooting.

Some other ways of preventing overshooting are changing the target angle slightly so it is properly inside the interval $(0, 2\pi$ by some tolerance, although this is not ideal because it changes the optimal configuration to be something other than what we actually want, and adding some self avoiding potential which is $\mathcal{O}(\frac{1}{\theta^2})$ or $\mathcal{O}(\frac{1}{\theta^4})$ (here θ should be the distance to the nearest multiple of 2π, which must be 0 or 2π) and possibly also decreasing over time. These are fairly common techniques for preventing overshooting in physical simulations.

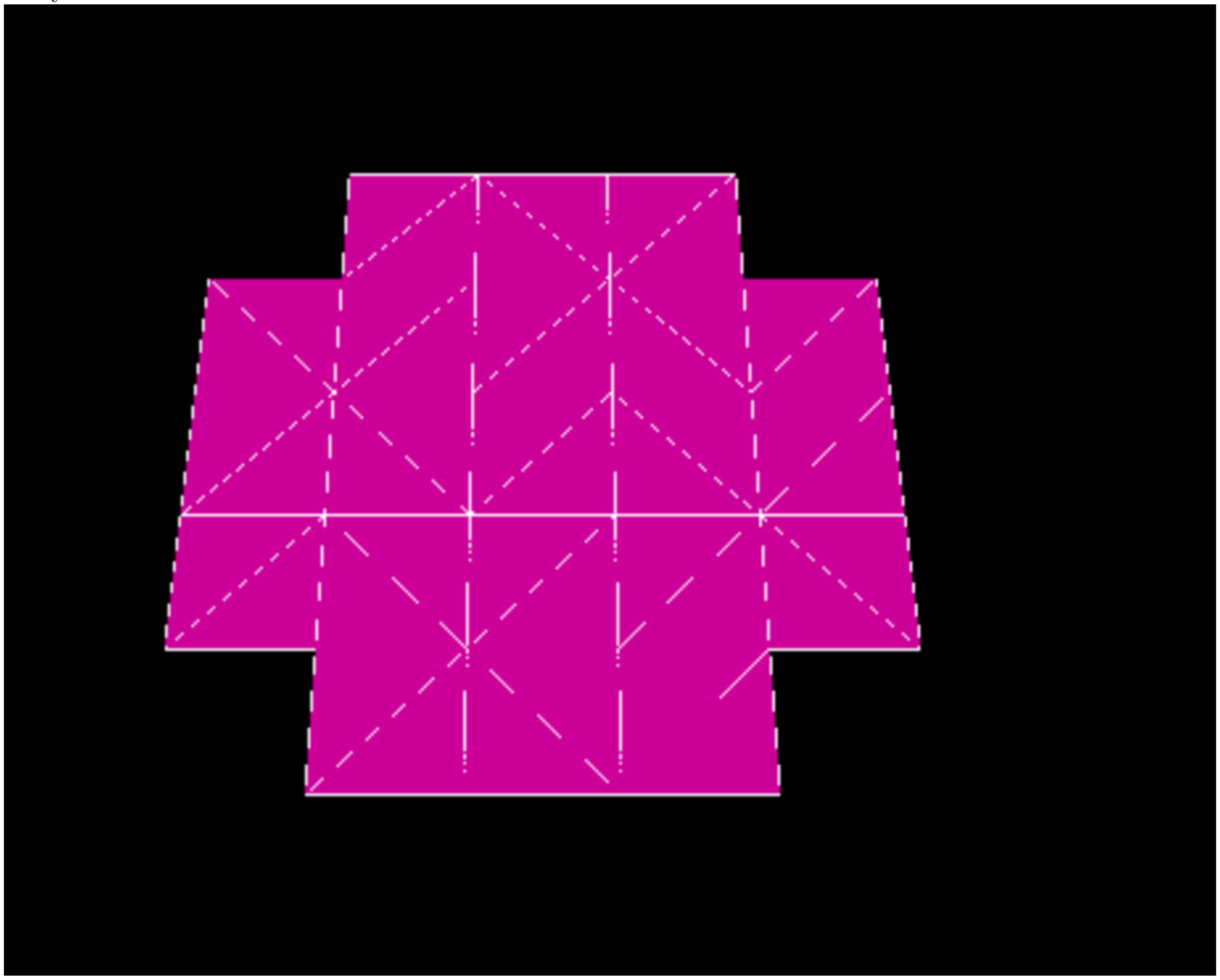
Besides the inherent overshooting in spring potentials and other harmonic oscillator potentials, time stepped simulations have another problem that can lead to overshooting. Since the simulation is processed in discrete time steps, it jumps from one state to the next, so for instance a simulation that has per-frame (instead of inter-frame which is not commonly used) collision detection and say we have two small bullets travelling towards each other in opposite directions on the x axis. It is unlikely that the short time the bullets would overlap if they could pass through each other instead of colliding will overlap with one of the frames of the simulation, so per-frame collision detection would not notice the collision. Even slower moving objects have a possibility of this happening, and even if our method of preventing collisions only requires near collisions (like a $\mathcal{O}(\frac{1}{\theta^2})$ potential), it is still possible that the collision occurs between frames or the entire region of states where the force from the self avoiding potential would be enough to prevent self intersection occurs between frames. To deal with this, it is pretty much universal for simulations to have multiple "physics frames" per "rendering frame". In this case, we render at 60 frames per second but run the physics simulation at 60000 frames per second. Since the physics framerate is decoupled from the rendering framerate, we can even vary the physics framerate and increase it near erratic behavior (ie when we have large velocities or accelerations or small distances). This makes sense because a physics simulation is really a numerical vector field integration or differential equation solving technique (with $\ddot{\vec{x}} = \frac{1}{m}\vec{F}$ and $\vec{F} = -\nabla E$), and for instance in a one dimensional numerical integration we might sample more points near poles or high slope regions to get a better approximation.

Note that the kinetic energy is a convex function of the velocities of the vertices of the crease pattern, so when we add velocity the problem remains convex (or actually quasiconvex and convex in a neighborhood of the solution because we showed the torsion spring potential only satisfies that weaker condition). This means we could add velocity to the initial problem and still have it be solvable by a general gradient descent solver or other convex optimization solver, but it would be less efficient because we would have twice as many variables and the optimal solution must have zero kinetic energy. If we look at the total energy function, it clearly splits into a velocity dependent part and a position dependent part, so a solver optimizing them separately would easily find zero velocity is optimal. However, the velocity is not really added as another optimization variable, it is added to allow us to use physical simulation and is really just a way of making the solver's incremental motion towards the optimal solution very explicit. I like physical simulations for this reason because they usually show very visibly what is happening to reach the solution, as opposed to an ordinary solver or even worse a machine learning model which is really a black box and does not offer any insight for how it works. Physical descriptions of a problem involving things like energy, potentials, Lagrangians, conserved quantities, and forces are well understood in a way that allows us to solve problems in a way that lets us visually see what is happening.

6 Dataset

For this problem, we have to test on some crease pattern. I chose the waterbomb base tesselation for several reasons. Firstly, it looks cool and it is what Amanda Ghassaei's origami solver defaults to when opened so it sort of serves as a "Hello World" or head-to-head comparison for origami solvers. Also, its vertices have little variance in the total surface area of their adjacent faces, so the unit mass approximation it pretty valid here. Finally, it is the kind of model an origami solver that applies forces locally on creases would be expected to perform well on, as opposed to something like the traditional crane which I would not expect to be solvable. The basic intuition for why the waterbomb tesselation is solvable but the crane is not is that the waterbomb tesselation can emerge from locally applying crease forces and locally minimizing the potential energy we derived, but some other models like the crane have to have certain forces in the crease pattern completely formed before others can be formed. In general, this solver would be expected to work well on many "open" tesselations like the waterbomb base and also on traditional models as far as the making the base and modifying the base with "open" folds like reverse folds, crimp folds, and sink folds. Open folds are pretty much folds that do not lock parts of the paper together or otherwise force the folds to be performed in a particular order. It might be possible to try to identify the folds in a crease pattern that are not open and remove them so they can be folded after the open folds have been collapsed, but this would be a much, much harder, probably intractable problem.

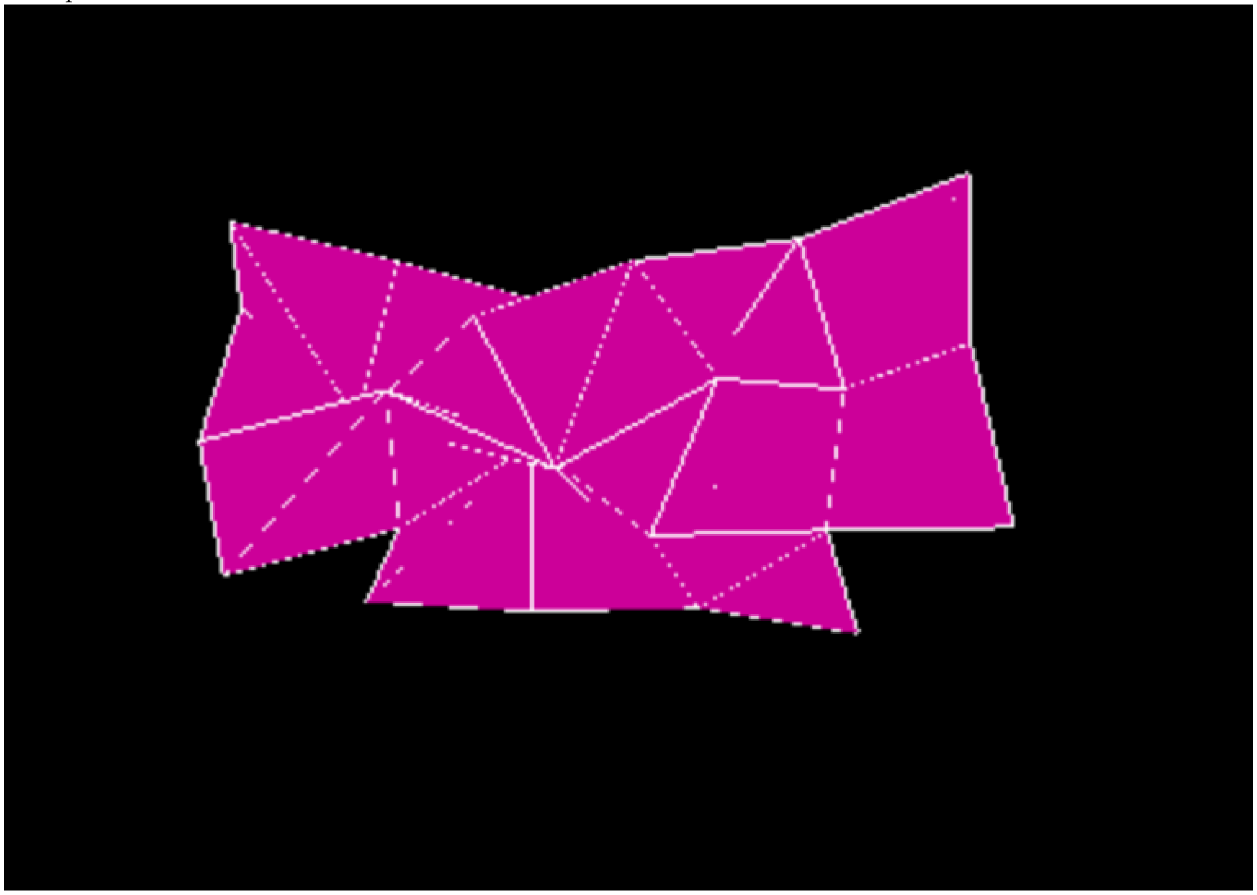
This picture of the initial state of the simulator on the waterbomb tesselation shows the crease pattern of a 2 by 2 version.



The vertical and horizontal lines are mountain folds that should reach a dihedral angle from the top of $\frac{3}{2}\pi$ (besides the raw edges which are obviously raw edges and not creases). The diagonal lines that meet at a vertex with 4 total diagonal lines are valley folds that should reach a dihedral angle from the top of 0. The remaining diagonal lines are triangulation edges that were added to make square faces in the crease pattern triangular. Therefore they should have a dihedral angle of π. Some lines are not clearly visible because of z-fighting in the rendering, but there should be four long vertical lines and four long horizontal lines (besides the raw edges) dividing the plus shaped paper into 21 squares, each of which has a line on exactly one of its diagonals.

7 Results

Currently the simulation just keeps going until stopped manually, but once it has clearly stopped moving this is a picture of the final state



This is not fully collapsed, but it is rather close and definitely close enough to see what's happening. Unfortunately the rendering is not shaded because I could not get shaded materials to work with a sheet. The crease lines are also not rendered very well because of z-fighting.

8 Analysis

The dual form of this problem is simply "maximize 0" since we have no constraints and the minimum energy for a properly defined crease pattern is zero, but this is not very interesting. Moreover, the perturbation function is just the potential energy function, and we showed each linear spring potential energy is a nonnegative quadratic which is zero in the affine subset of $\mathbb{R}^{3n}$ where $\|\vec{r_i} - \vec{r_j}\| = l_{0:i,j}$. We also showed each torsion spring potential energy is a nonnegative quadratic (to a second order Taylor series) in a neighborhood of the optimal solution, which is zero at the optimal solution. This tells us the overall function, which is a sum of these potential energies, is a nonnegative quadratic which is zero on an affine subset of configurations which are equivalent under rotation and translation. This means if we wanted to we could find a quadratic cone problem that approximates our problem in a neighborhood of our solution.

Similarly, the way we stated the problem there are no penalty functions, however, if we consider that we really want to find an exact solution to the crease pattern that has all side lengths and dihedral angles correct, then the entire potential energy function is a penalty function on a feasibility problem, where the parameters λ_l, λ_t, and λ_T tell us how averse we are to violating side length constraints, ordinary crease angle constraints, and synthetic triangulation crease flat angle constraints, respectively. I did play around with these three parameters a bit, but we really know that $\lambda_l > \lambda_T > \lambda_t > \mu$ should hold because λ_l is a material characteristic based on the stretchiness of the paper, λ_T is pretty much a material characteristic based on the bendiness of the paper, λ_t tells us how strong our driving forces that simulate collapsing the crease pattern should be, and we want the surface area preserving force to dominate the surface flattening force which we want to dominate the driving force which we want to be able to overcome friction. Also, if these constants are too large, they cause the simulation to become unstable because the motion between physics frames is too great, and if they are too small, the simulation becomes unbearably slow. From testing, the variables in the code should have $kl \leq 1$ and $kt \geq 0.00002$, while preserving about a 20 times difference from one constraint to the next. I did not make a histogram because it would have three explanatory variables plus the simulation does automatically stop yet since although it reaches a reasonable configuration that illustrates the form of the final model clearly, it does not reach an energy level I considered good enough to stop. The initial energy is 0.009890 and the final energy is around 0.009519 which seems like a negligable change but also like it should be very close to the correct solution. Neither of these are really true though: it is so small because the spring constants are very small, and the reason even such a proportionally small change is good (besides that the simulation accurately depicts the folding of a nontrivial model) is that the differences between the neutral length and final length of all the springs is much less than 1 so the quadratic energies are quite sublinear in this area.

9 Code

The code can be found in separate files. I used JavaScript and THREE.js so that I could generate 3D rendering code and an actually useful physics simulation in a very constrained amount of time despite no experience with either. The code in THREE.js and CCapture.all.min.js is not mine of course.

To use the code, make sure that index.html and js/ are in the same folder and then open index.html with a reasonably non-vintage web browser. Press F12 or otherwise open the JavaScript console, press F5 to reload the page, and then click the picture to start or stop the simulation. Whenever clicked, the current energy will also be printed to the console, so opening the console is not necessary unless seeing the energy is desired.

Most of the important parts of the code have been explained, but note that the global variable n is the number of tesselation units per side, not the number of crease pattern vertices, since I was initially trying to automatically generate a waterbomb tesselation crease pattern given a size but it proved to be more work than just hard coding it.