Fast, Interactive Origami Simulation using GPU Computation

Amanda Ghassaei, Erik D. Demaine, Neil Gershenfeld

Abstract:

We present an explicit method for simulating origami that can be rapidly computed on a Graphics Processing Unit (GPU). Previous work on origami simulation methods model the geometric or structural behaviors of origami with a focus on physical realism; in this paper we introduce a compliant, explicit numerical simulation method that emphasizes computational speed and interactivity. We do this by reformulating existing techniques for simulating origami so that they can be computed on highly parallel GPU architectures. We implement this method in an open-source, GPU-accelerated WebGL app that runs in any modern browser. We evaluate our method's performance, stability, and scalability to existing methods (Freeform Origami, MERLIN) and demonstrate its capacity for real-time interaction through a traditional GUI and immersive virtual reality.

1 Introduction

Physical simulations of origami allow designers to better understand how modifications to a crease pattern affect its folded state. Real-time simulation-based feedback can provide more intuitive ways to edit a folded pattern or to enforce desired geometric constraints, such as global developability. In particular, simulation provides an approach to inverse design, where computational tools generate valid designs according to high-level user-specified goals. An alternative to algorithmic inverse design (as in e.g. [Lang 96, Demaine and Tachi 17]) is to develop interactive design tools coupled with tightly integrated simulation capability [Tachi 10, Tang et al. 16]. This work aims to create a fast, interactive simulation environment that could serve as the backbone for new computational design tools for origami.

2 Simulation Methods

Numerical methods for simulating origami typically discretize an origami crease pattern into a rigid-body linkage or finite element mesh and implicitly solve a linear system of equations for small displacements. Through an iterative process, these methods compute large, non-linear deformations of origami from its initially flat or folded state. Rigid origami simulators [Tachi 06] model the rigid

motions of origami with an emphasis on kinematic accuracy. Finite element methods (FEM) explore the structural properties of the origami's folded or partially folded states using realistic material models and higher-order elements such as plates/shells [Schenk et al. 14, Peraza Hernandez et al. 16], or volumetric elements [Ablat and Qattawi 18].

Bar-and-hinge models are an efficient modeling abstraction for approximating the mechanical behavior of origami [Tachi 10, Liu and Paulino 16]. Despite their simplicity, bar-and-hinge models can be used to capture realistic behaviors, such as out-of-plane bending [Tachi 13, Schenk and Guest 11] and in-plane shearing [Filipov et al. 17] of regions between creases.

The purpose of our solver is not to model physical behavior that has not been previously explored, but rather to rapidly compute the folded state of origami for applications with direct user interaction. Our solver reframes prior implicit methods into a dynamic, explicit form that is solved in a highly parallel process on a GPU. We use compliant constraints to guide the folding of the origami model, allowing the user to trade off computational speed for accuracy (or vice versa).

Sections 2.1–2.4 outline the setup for the method (based on prior work by [Schenk and Guest 11] and [Tachi 10], repeated here for clarity); Section 2.5 introduces our explicit integration method; and Section 3 discusses the parallelization of this method on the GPU.

2.1 Meshing

First we discretize the origami crease pattern into a triangulated mesh. We triangulate the crease pattern by adding extra edges across any polygonal faces with more than three sides (Figure 1). Following [Schenk and Guest 11], we call these edges "facet creases" and elastically constrain them to remain flat while folding.

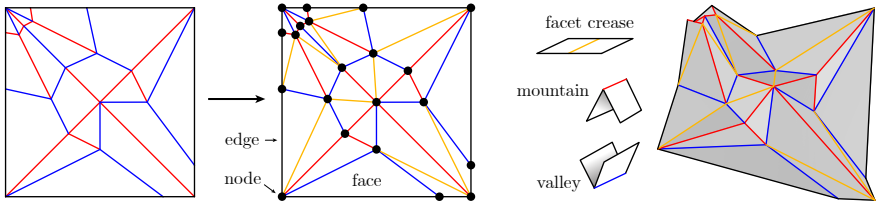

Figure 1: *Triangulation of an origami crease pattern (left) into a triangulated mesh (center). Polygonal faces may be triangulated in an automated process, or controlled directly by the user. (right) Unlike mountain and valley creases, facet creases formed by triangulation are driven flat by elastic constraints during folding.*

Each of the edges in the resulting mesh is modeled as a beam in a pin-jointed truss, subject to axial and angular constraints, which we now detail.

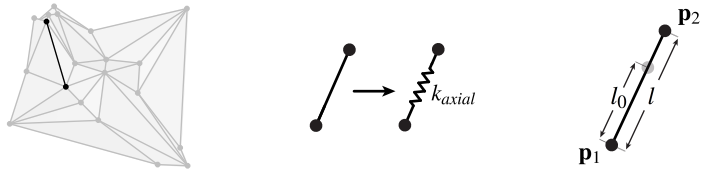

Figure 2: *(left) A crease pattern is triangulated and simulated as a pin-jointed truss network. (center) Beams in the truss are modeled as linear-elastic springs. (right) Formulation of axial constraints.*

2.2 Axial Constraints

Axial constraints prevent stretching and compression of the origami surface during folding. Each beam element behaves like a linear-elastic spring that applies forces only in its axial direction (Figure 2). At each step of the simulation, we calculate the length of each beam from the position of the nodes at its endpoints. We compute the force exerted on each node in the beam's local coordinate space by Hooke's Law:

$$F_l = -k_{axial}(l - l_0),$$

where F_l is the force along the beam's axis, k_{axial} is the axial stiffness of the beam element, l is the length of the beam, and l_0 is the nominal length of the beam.

We must convert F_l into a vector $\mathbf{F}_{axial}$ in global coordinate space before we apply it to the nodes. This axial force vector is related to node position $\mathbf{p}$ by

$$\mathbf{F}_{axial} = -\nabla V(\mathbf{p}) = -\frac{\partial V}{\partial \mathbf{p}},$$

where V is the potential energy of the system. By the chain rule, we have

$$\mathbf{F}_{axial} = -\frac{\partial V}{\partial l}\frac{\partial l}{\partial \mathbf{p}} = F_l \frac{\partial l}{\partial \mathbf{p}},$$

$$\mathbf{F}_{axial} = -k_{axial}(l - l_0)\frac{\partial l}{\partial \mathbf{p}}. \tag{1}$$

For the two nodes attached to each beam, we have

$$\frac{\partial l}{\partial \mathbf{p}_1} = -\hat{\mathbf{l}}_{12}, \qquad \frac{\partial l}{\partial \mathbf{p}_2} = \hat{\mathbf{l}}_{12},$$

where $\hat{\mathbf{l}}_{12}$ is a unit vector from node 1 to node 2 and $\mathbf{p}_1$ is the 3D position of node 1 in global coordinate space.

Axial stiffness can be related to the material properties of the substrate by

$$k_{axial} = \frac{EA}{l_0},$$

where E is Young's modulus and A is the cross-sectional area of the beam. For this analysis, we consider only the relative stiffness of the axial and angular constraints

and disregard their physical meaning. We choose constant EA for our model and calculate k_{axial} of each beam based on its l_0.

2.3 Crease Constraints

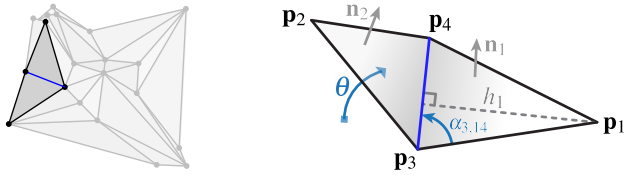

Figure 3: *Formulation of crease constraints.*

With distance constraints alone, we can model an arbitrary linkage connected by frictionless, spherical joints. We add constraints on the dihedral angle between triangulated faces of the mesh to drive and constrain folding.

The fold angle (θ) of a crease is the supplement of the dihedral angle between two neighboring triangular faces. We model angular constraints as linear-elastic torsional springs that drive neighboring triangular faces toward some target fold angle. Similarly to Equation 1, angular constraints apply forces to neighboring nodes by

$$\mathbf{F}_{crease} = -k_{crease}(\theta - \theta_{target})\frac{\partial \theta}{\partial \mathbf{p}}, \tag{2}$$

where $\mathbf{F}_{crease}$ is a 3D force vector in global coordinate space, θ_{target} is the desired fold angle of the crease, $\mathbf{p}$ is the position of the node, and k_{crease} is the stiffness of the constraint. Following prior work [Schenk and Guest 11], we choose k_{crease} according to its type and scale it by its nominal length (l_0):

$$k_{crease} = \begin{cases} l_0 k_{fold} & \text{for a mountain or valley crease,} \\ l_0 k_{facet} & \text{for a facet crease (from Section 2.1),} \\ 0 & \text{for a boundary edge or undriven crease.} \end{cases}$$

We choose our stiffnesses so that $k_{axial} \gg k_{fold}$ and k_{facet}. Angle θ_{target} also depends on the type of crease:

$$\theta_{target} = \begin{cases} < 0 & \text{for a mountain crease,} \\ > 0 & \text{for a valley crease,} \\ 0 & \text{for a facet crease.} \end{cases}$$

The precise value of θ_{target} for mountain/valley creases is set by the user; see Section 3.

Each angular constraint applies forces to four neighboring nodes, as indicated in Figure 3. The partial derivatives of θ with respect to $\mathbf{p}$ are given by

$$\frac{\partial \theta}{\partial \mathbf{p}_1} = \frac{\mathbf{n}_1}{h_1}, \tag{3}$$

$$\frac{\partial \theta}{\partial \mathbf{p}_2} = \frac{\mathbf{n}_2}{h_2}, \tag{4}$$

$$\frac{\partial \theta}{\partial \mathbf{p}_3} = \frac{-\cot \alpha_{4,31}}{\cot \alpha_{3,14} + \cot \alpha_{4,31}} \frac{\mathbf{n}_1}{h_1} + \frac{-\cot \alpha_{4,23}}{\cot \alpha_{3,42} + \cot \alpha_{4,23}} \frac{\mathbf{n}_2}{h_2}, \tag{5}$$

$$\frac{\partial \theta}{\partial \mathbf{p}_4} = \frac{-\cot \alpha_{3,14}}{\cot \alpha_{3,14} + \cot \alpha_{4,31}} \frac{\mathbf{n}_1}{h_1} + \frac{-\cot \alpha_{3,42}}{\cot \alpha_{3,42} + \cot \alpha_{4,23}} \frac{\mathbf{n}_2}{h_2}, \tag{6}$$

where $\mathbf{n}_1$ and $\mathbf{n}_2$ are face normals, h_1 and h_2 are lever arms from the crease to the outer nodes, and in-plane angles α are oriented according to Figure 3.

2.4 Face Constraints

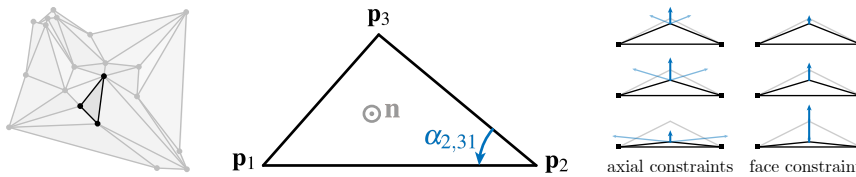

Figure 4: *(left and center) Formulation of face constraints. (right) As a triangular face (with two bottom nodes fixed) flattens, the restoring force applied to the center node by two neighboring axial constraints lessens the more the triangle is deformed. In contrast, three face constraints apply a greater force as the triangle deforms. Individual contributions shown in light blue, net force shown in dark blue, undeformed state shown in grey, fixed nodes indicated with squares.*

With beam and crease constraints alone, we have enough to simulate folding. Though not strictly necessary, the addition of constraints on each interior angle of the triangular faces of the mesh helps to prevent shearing of the folding surface, especially for high-aspect-ratio triangles (Figure 4). In practice, we have found that face constraints increase the stability of the simulation across a variety of input crease patterns. As with the previous constraints, we model this as a linear-elastic spring.

For each interior angle of a triangular face, we apply forces to its three neighboring nodes according to

$$\mathbf{F}_{face} = -k_{face}(\alpha - \alpha_0) \frac{\partial \alpha}{\partial \mathbf{p}},$$

where α is the current angle, α_0 is its nominal angle in the flat state, $\mathbf{p}$ is the position of a neighboring node, and k_{face} is the stiffness of the face constraint. The partial

derivatives of $\mathbf{p}$ with respect to α are given by

$$\frac{\partial \mathbf{p}_1}{\partial \alpha_{2,31}} = \frac{\mathbf{n} \times (\mathbf{p}_1 - \mathbf{p}_2)}{\|\mathbf{p}_1 - \mathbf{p}_2\|^2},$$
$$\frac{\partial \mathbf{p}_2}{\partial \alpha_{2,31}} = -\frac{\mathbf{n} \times (\mathbf{p}_1 - \mathbf{p}_2)}{\|\mathbf{p}_1 - \mathbf{p}_2\|^2} + \frac{\mathbf{n} \times (\mathbf{p}_3 - \mathbf{p}_2)}{\|\mathbf{p}_3 - \mathbf{p}_2\|^2},$$
$$\frac{\partial \mathbf{p}_3}{\partial \alpha_{2,31}} = -\frac{\mathbf{n} \times (\mathbf{p}_3 - \mathbf{p}_2)}{\|\mathbf{p}_3 - \mathbf{p}_2\|^2},$$

where $\mathbf{n}$ is the normal vector of the triangular face and α is defined according to Figure 4.

2.5 Numerical Integration

So far, the governing equations we have described in the previous sections are a mix of the methods described by [Tachi 10] and [Schenk and Guest 11]. Our methods differ from prior work by calculating small displacements of the nodes under axial and angular constraints using an explicit method; these computations occur in parallel on a per-node basis, without a global stiffness/Jacobian matrix.

We calculate the total force on a node as the sum of the forces applied by neighboring beams, creases, and faces:

$$\mathbf{F}_{total} = \sum_{beams} \mathbf{F}_{beam} + \sum_{creases} \mathbf{F}_{crease} + \sum_{faces} \mathbf{F}_{face}.$$

We compute nodal acceleration by

$$\mathbf{a} = \frac{\mathbf{F}_{total}}{m},$$

where $\mathbf{a}$ is a 3D acceleration vector in global coordinate space and m is the mass of the node. In this analysis, we assume the mass at each node is 1. A more accurate calculation of nodal mass would result in more realistic dynamics.

We calculate velocity and displacement at each node by forward Euler integration:

$$\mathbf{v}_{t+\Delta t} = \mathbf{v}_t + \mathbf{a}\Delta t,$$
$$\mathbf{p}_{t+\Delta t} = \mathbf{p}_t + \mathbf{v}_{t+\Delta t}\Delta t,$$

where Δt is a small time step. In general, Δt should be chosen so that it is small enough to keep the simulation numerically stable, but not too small that it slows down the simulation unnecessarily. We have chosen Δt to satisfy

$$\Delta t < \frac{1}{2\pi\omega_{max}}, \tag{7}$$

where ω_{max} is the maximum natural frequency of any constraint in the model. Because we will always choose $k_{axial} \gg k_{fold}$, k_{facet}, and $k_{axial} > k_{face}$, we consider

only axial constraints for this analysis:

$$\omega = \sqrt{\frac{k_{axial}}{m_{min}}}, \tag{8}$$

where m_{min} is the minimum mass of the two nodes on either end of the beam.

We assume the initial condition:

$$\mathbf{v}_0 = \mathbf{0}.$$

The above formulation may eventually converge to a static state due to numerical dissipation, but this may take more iterations than is practical. We introduce viscous damping between neighboring vertices in the mesh:

$$\mathbf{F}_{damping} = c(\mathbf{v}_{neighbor} - \mathbf{v}),$$

where c is the viscous damping coefficient and $\mathbf{v}_{neighbor} - \mathbf{v}$ is the relative velocity between a node and its neighbor. We use this simplified approach to damping rather than calculating the damping force for every constraint in the system to minimize floating-point operations per simulation cycle. Additionally, we found that even with critical damping on all constraints, the structure was globally underdamped [Hiller and Lipson 14]. A more thorough approach to damping could result in more realistic dynamics.

We compute c for each node according to the stiffest constraint in our system (k_{axial}):

$$c = 2\zeta\sqrt{k_{axial}m},$$

where ζ is the damping ratio and m is the mass of the node (again, assumed to be 1 in this analysis). We've found that most patterns are stable with $0.01 \leq \zeta \leq 0.5$.

3 Implementation

We implemented this solver in an open-source[1], interactive WebGL app that runs in any modern web browser[2]. We use the FOLD format of [Demaine et al. 16] for input/output and as an internal data structure to maximize interoperability with other origami software; we also support SVG input/output with opacity mapped to final fold angle and color mapped to crease type. We make few assumptions about the topology of the folded structure, allowing our solver to simulate origami, kirigami, discretized curved creases, underconstrained patterns with undriven hinges, patterns with holes, and 3D structures that do not have a flat state. We provide an interface to control simulation parameters in real time, such as stiffness and damping coefficients and the number of simulation steps per render cycle (Figure 6).

[1] https://github.com/amandaghassaei/OrigamiSimulator/
[2] http://apps.amandaghassaei.com/OrigamiSimulator/

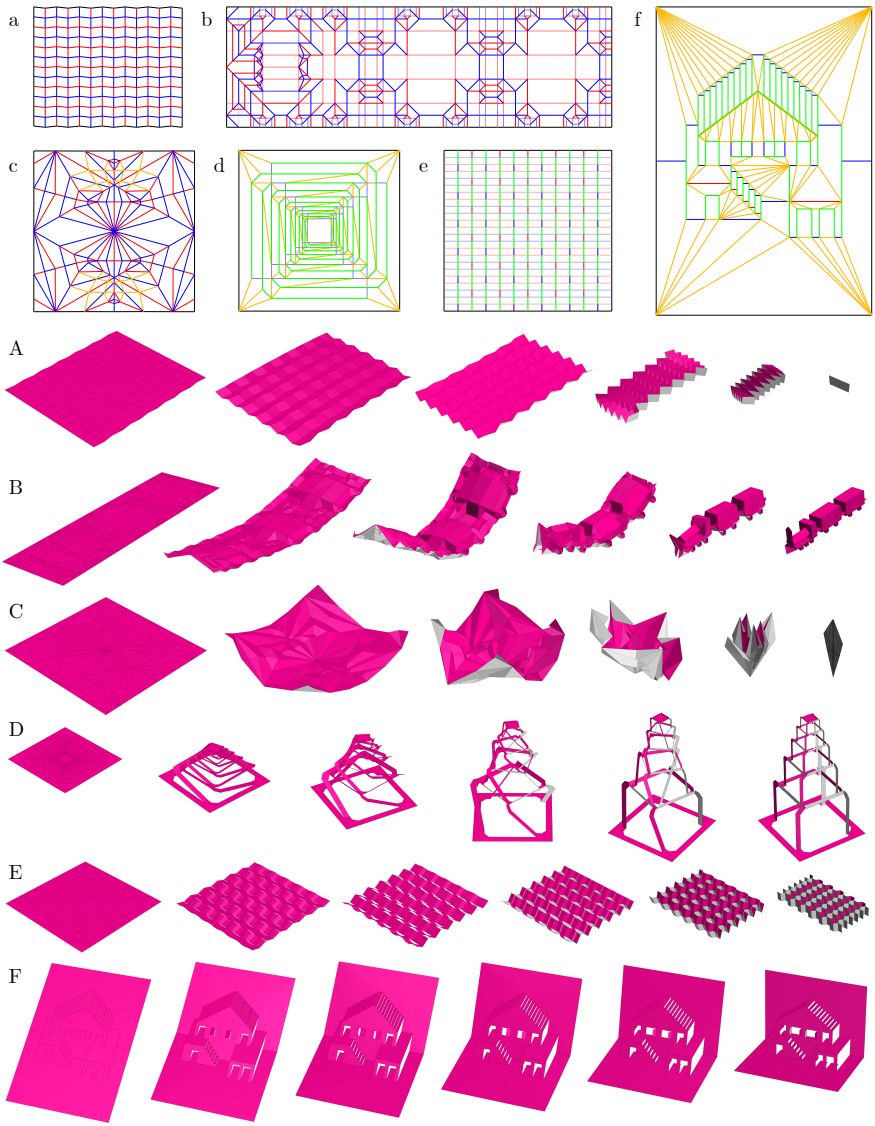

Figure 5: *Static solution of simulated origami at 0%, 20%, 40%, 60%, 80%, and 100% of target fold angle. Patterns depicted are (A) Miura-ori tessellation, (B) Mooser's Train (rigidly foldable design by William Gardner), (C) Robert Lang's Orchid Blossom, (D) Yoshinobu Miyamoto's RES Square Tower, (E) kirigami honeycomb [Saito et al. 14], and (F) pop-up house by Popupology. Corresponding crease patterns above with line opacity indicating final fold angle and color indicating crease type: mountain (red), valley (blue), border (black), cut (green), user-defined facet crease (orange). EA = 20, k_{fold} = 0.7, k_{facet} = 0.7, and k_{face} = 0.2.*

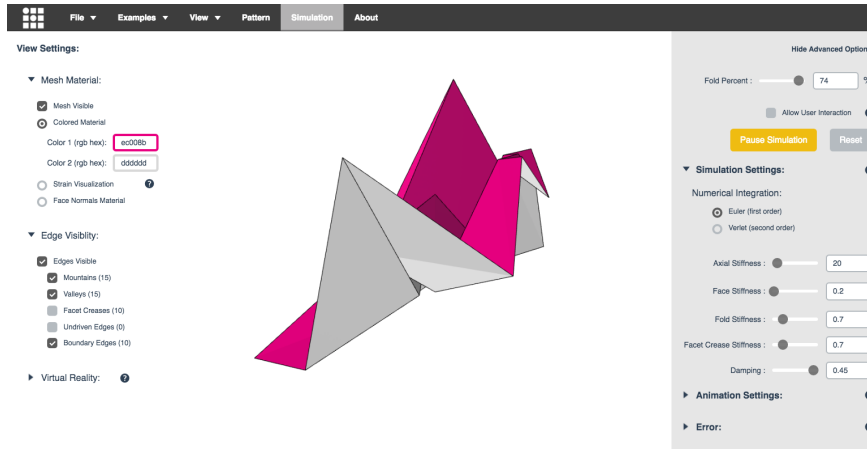

Figure 6: *Screenshot of Origami Simulator application.*

Before the solver begins, we precompute stiffness and damping coefficients as well as a lookup table of geometric relationships between elements. We run a series of GPU programs in WebGL fragment shaders to compute one step of the simulation:

1. Calculate face normals of all triangular faces in mesh (one face per thread).
2. Calculate current fold angle for all edges in mesh (one edge per thread).
3. Calculate coefficients of Equations 3–6 for all edges in mesh (one edge per thread).
4. Calculate forces and velocities for all nodes in mesh (one node per thread).
5. Calculate positions for all nodes in mesh (one node per thread).

In other GPU programming frameworks (e.g., NVIDIA CUDA), this solver could be implemented in fewer steps, but we chose to use WebGL shaders to maximize portability across operating systems and for easy browser support.

After a specified number of simulation steps, we render the geometry to the screen. Figure 5 shows a collection of simulated structures.

3.1 Strain Visualization

We implement a strain visualization tool to help users identify locally deformed regions in the mesh (Figure 7). We approximate the engineering strain across the surface of a folded origami structure by measuring the displacement of the axial constraints. For a given beam in the pin-jointed truss, we define engineering strain ε by

$$\varepsilon = \frac{l - l_0}{l_0},$$

where l is the length of the beam, and l_0 is the nominal length of the beam.

The magnitude of engineering strain at a node with N beams is calculated by averaging the absolute value of the strain contribution from each beam:

$$\varepsilon_{node} = \frac{1}{N} \sum_{i=1}^{N} \frac{|\Delta l_i|}{l_i}.$$

We translate this strain into an RGB color on the spectrum of blue (no strain) to red (high strain) and apply it to the 3D model for visualization. The vertex colors are linearly interpolated across the faces of the mesh.

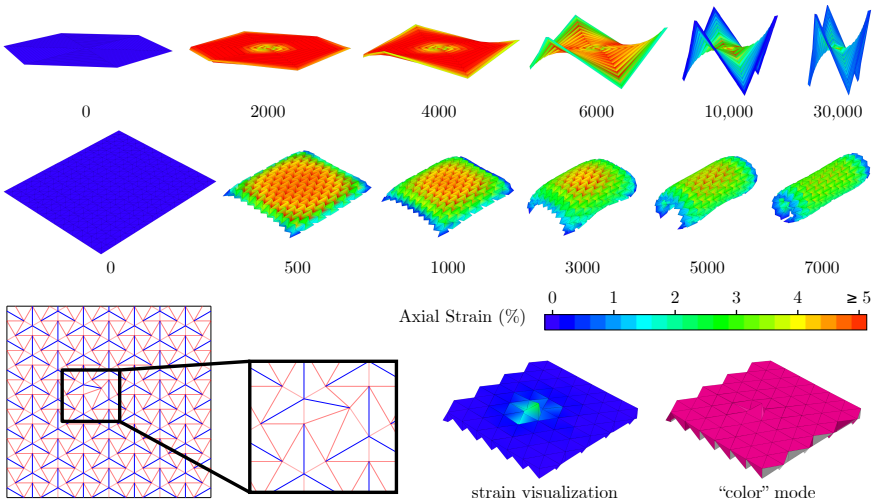

Figure 7: *Axial strain visualization of a hexagonal hypar variant (top, folded to $\pm 3\pi/4$ at all creases) and waterbomb tessellation (middle, folded to $\pm 3\pi/5$ at all creases). The sequences show our method solving for a fixed final fold angle in order of increasing time, with the number of simulation steps indicated below. Initially, both models move to a nearly planar state with high internal strain, which is slowly relaxed as they curl into the third dimension. (bottom) An irregular vertex in a Resch tessellation deforms the mesh slightly in material "color" rendering mode, but is clearly visible in the axial strain visualization. Simulation parameters for hypar variant: EA = 100, k_{fold} = 0.7, k_{facet} = 0.7, and k_{face} = 1, ζ=0.45. Simulation parameters for waterbomb and Resch tessellation: EA = 20, k_{fold} = 0.7, k_{facet} = 0.7, and k_{face} = 0.2, ζ=0.45.*

3.2 User Interaction

We introduce user interaction by modifying the boundary conditions of the simulation in real time. Compliance in the simulated structure allows users to toggle between stable states of bistable patterns (Figure 8A). Using the WebVR API, we are able to run our app on the HTC Vive and Oculus virtual reality (VR) headset and controllers (Figure 8B). In this mode, users can manipulate the origami with both hands and view the rendering in an immersive 3D environment. Future work

will close the loop between design and simulation, allowing for interactive modifications to the folded structure.

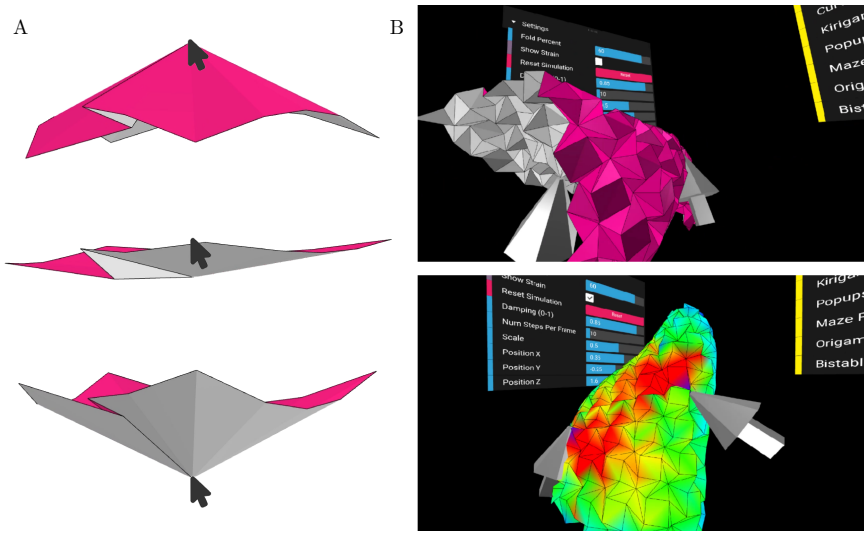

Figure 8: *(A) User interaction pushing bistable pleat model from one stable state to the other. (B) Virtual reality interface showing direct user interaction with folding Huffman waterbomb model. (B, bottom) With strain visualization turned on, users can pull and push on the mesh while visualizing strain in real time.*

4 Discussion

We performed a series of benchmarking tests to understand how the simulation speed of our solver scales with the number of nodes in the origami model (Figure 9). We ran these tests using the Chrome browser on two different GPUs: an Iris Graphics 6100 with 348 cores and a 300MHz clock, and two SLI-linked GeForce GTX 980's with 4096 combined cores and a 1.1GHz clock. We imported a series of $n \times n$ Miura-Ori tessellations of different dimensions into the solver and measured the simulation speed without rendering.

We found that the time to compute one cycle of the simulation was approximately constant while the number of nodes in the mesh (N) was less than the number of available GPU cores (indicated by the blue X's in Figure 9a). As N grows beyond the available cores, we see approximately linear scaling of simulation time with N as the GPU queues batches of threads and executes the batches in series.

4.1 Comparison with Existing Methods

We compare the speed of our solver to existing methods MERLIN [Liu and Paulino 17] and Freeform Origami [Tachi 10] (Figure 10). Both methods implicitly solve for small displacements of a triangulated origami mesh on the CPU.

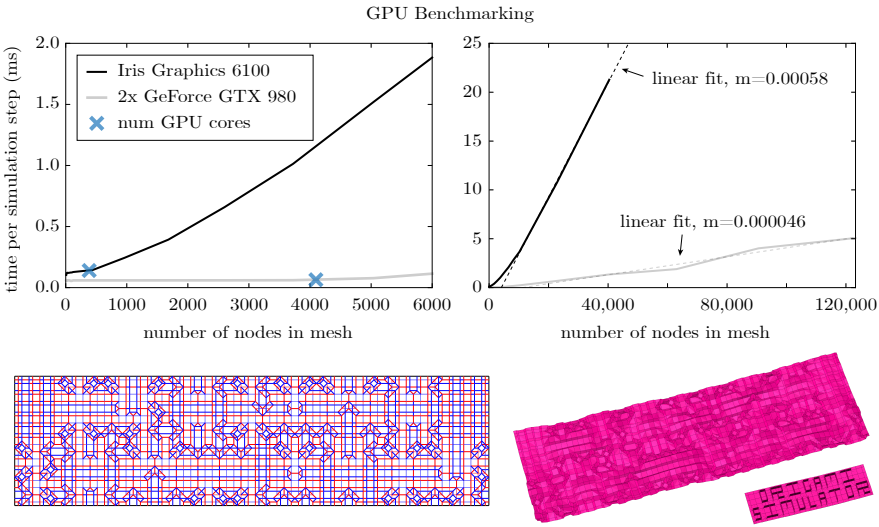

Figure 9: *Benchmarking time per simulation step as the number of nodes (N) in the mesh increases. (top left) When the number of nodes is smaller than the number of available GPU cores (indicated by the blue X's) the time to complete one simulation cycle is approximately constant. (top right) For very large meshes, simulation time scales approximately linearly with N. (bottom) Real-time simulation of a mesh with 2374 nodes.*

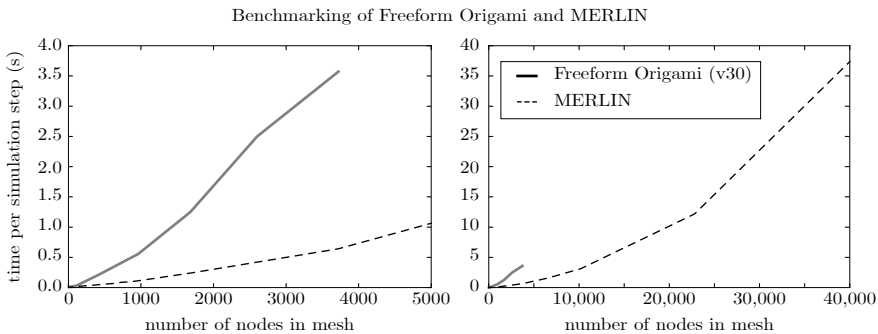

Figure 10: *Benchmarking solver iteration time versus mesh size in Freeform Origami and MERLIN.*

Each iteration of the MERLIN solver computes a global stiffness matrix and solves a system of equations that relates residual force to internal and external applied forces using the modified generalized displacement control method. Though the time to compute each of these steps is about 1000X longer than a single step in our method (Figure 10), MERLIN takes larger steps while maintaining stability for stiffer material settings. A 10×10 cell Miura-ori tessellation (121 nodes)

reaches its final folded state in about 500 iterations of the MERLIN solver with material stiffnesses k_{fold} = 0.1, k_{facet} = 10,000, EA = 100,000, while about 150,000 iterations are needed using our method at these stiffer settings.

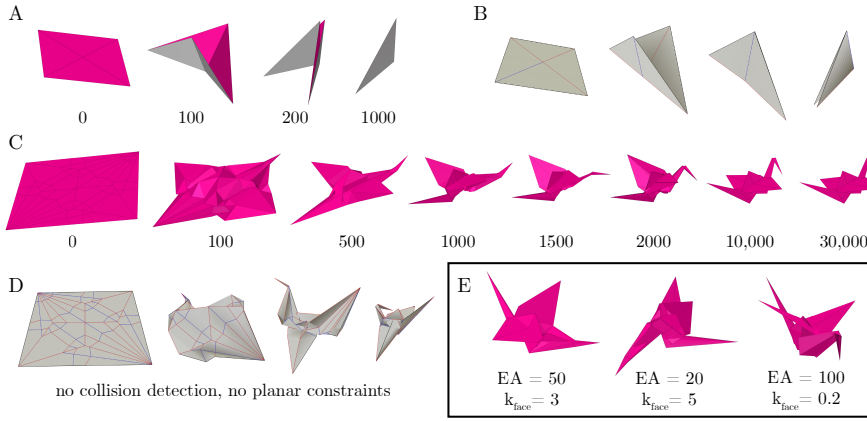

Figure 11: *Comparison of our method (magenta) and Freeform Origami (gray) [Tachi 10]. Unless otherwise indicated, simulation settings using our method are EA = 20, k_{fold} = 0.7, k_{facet} = 0.7, k_{face} = 0.2, and ζ = 0.45, and the number of simulation steps is indicated below each image in the sequence. (A) Due to compliance in our axial and face constraints, some deformation of the mesh occurs as it tries to simultaneously fold a simply folded vertex; (B) by contrast, Freeform Origami more accurately captures the sequential nature of the folds. (C) The compliance of our method allows non-rigidly foldable designs like the crane to find their final folded state (allowing for some self-intersection). (D) Freeform Origami minimizes geometric error at each step of the simulation, effectively modeling infinitely stiff meshes. Even with collision detection and planar constraints (infinitely stiff facet creases) turned off, Freeform Origami is not able to correctly fold a crane. (E) Similarly, increasing the material stiffness in our solver prevents the crane from reaching the correct folded state.*

Freeform Origami is a rigid origami simulator (though it can also model elastic out-of-plane bending deformations of facets [Tachi 13]). The data points in Figure 10 were gathered from the FPS display in the app with face, edge, and vertex rending disabled and collision detection disabled. Each rendering cycle of Freeform Origami is comprised of many iterations of a conjugate gradient solver, which should be taken into consideration when comparing the benchmarking data. Figure 11 depicts the modeling differences between our method and Freeform Origami. Though Freeform captures the sequential folding of a simple vertex better than our method, Freeform's rigid constraints prevent it from folding non-rigidly foldable patterns like the crane; our method fails similarly when the axial and face stiffnesses are increased (Figure 11E).

In contrast with the quasi-static methods of Freeform Origami and MERLIN, the intermediate steps of our explicit solver depict the dynamics of the structural

system. Though we have not taken much care to construct a realistic model of mass distribution and damping, our solver can be used to produce animations with plausible time-dependant behavior (Figure 12).

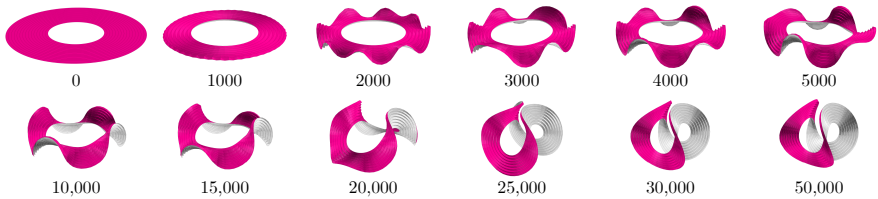

Figure 12: *Dynamic animation of a circular pleat pattern with iteration number indicated below. Simulation settings: target fold angle of all mountain/valley creases* $= \pm 7\pi/10$, *EA = 20,* $k_{fold} = 0.7$, $k_{facet} = 0.7$, $k_{face} = 0.2$, *and* $\zeta = 0.2$.

4.2 Limitations

According to Equations 7 and 8, there is a tradeoff between geometric accuracy (constraint stiffness) and simulation speed (Δt). Implicit formulations are advantageous in this regard because they can model very stiff systems while maintaining large step sizes, even though each step requires more computational effort to compute. To mitigate this, we provide easy access to stiffness parameters so that users may increase constraint stiffness when the model is near its static solution to avoid unnecessary computational time in the early stages of the simulation.

As is typical with other bar-and-hinge simulation methods, the behavior of the simulation is sensitive to the discretization of its mesh. This is especially prominent in the meshing of facets, where there may be more than one way to triangulate an arbitrary polygonal face (Section 2.1). If specific out-of-plane bending modes are desired, users may manually define facet creases before importing the designs into our app (Figure 13). Filipov et al. describe the effect of facet meshing on in-plane and out-of-plane deformations in bar-and-hinge models of origami in more detail [Filipov et al. 17].

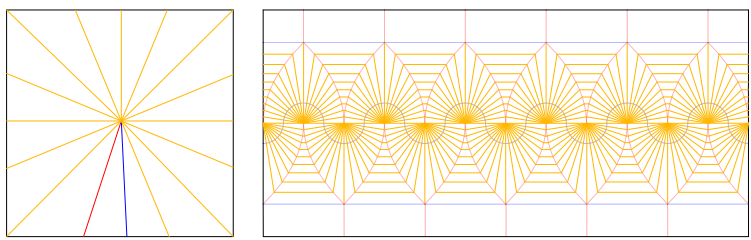

Figure 13: *User defined facet creases along rule lines of curved crease patterns guide expected out-of-plane bending of regions between mountain/valley creases.*

High-aspect-ratio triangles may pose a problem in simulation if they become so deformed that their face normal (defined by the cross product of two edges) flips

orientation. This problem is not unique to our method, but may be exacerbated by the compliance of the constraints. This instability is mitigated by carefully chosen, user-defined facet creases or by increasing axial and face stiffnesses relative to fold stiffness.

Finally, our method requires a target fold angle for each crease as an input to the solver. Incorrect target fold angles may deform the final folded state, though this can usually be diagnosed with the strain visualization tool.

5 Future Work

Parallelization on the GPU is not unique to the explicit integration method introduced in Section 2.5 or to bar-and-hinge models in general. We chose to start with the methods outlined in this paper in part because of the simplicity of their implementation, but (as noted in Section 4.2) there are limitations to our current approach. In immediate future work, we plan to apply GPU parallelization to implicit formulations of bar-and-hinge models, so that we can model stiffer structural systems in a highly efficient manner. Through GPU acceleration, it may even be possible to perform more complex FEM in real time, opening up new possibilities for interactive origami design, optimization, and analysis tools.

Future work may also grow to include collision detection, adaptive time stepping, more robust mesh triangulation methods, and the incorporation of a design interface in our app.

Acknowledgments

This work was supported by sponsors of the Center for Bits and Atoms.

References

[Ablat and Qattawi 18] Muhammad Ali Ablat and Ala Qattawi. "Finite Element Analysis of Origami-Based Sheet Metal Folding Process." *Journal of Engineering Materials and Technology* 140:April (2018), 1–7.

[Demaine and Tachi 17] Erik D Demaine and Tomohiro Tachi. "Origamizer: A Practical Algorithm for Folding Any Polyhedron." *33rd International Symposium on Computational Geometry* :34 (2017), 1–15.

[Demaine et al. 16] Erik D. Demaine, Jason S. Ku, and Robert J. Lang. "A New File Standard to Represent Folded Structures." In *Abstracts from the 26th Fall Workshop on Computational Geometry*, 2016. See https://github.com/edemaine/fold.

[Filipov et al. 17] E T Filipov, K Liu, T Tachi, M Schenk, and G H Paulino. "Bar and hinge models for scalable analysis of origami." *International Journal of Solids and Structures* 124 (2017), 26–45.

[Hiller and Lipson 14] Jonathan Hiller and Hod Lipson. "Dynamic Simulation of Soft Multimaterial 3D-Printed Objects." *Soft Robotics* 1:1 (2014), 88–101.

[Lang 96] Robert J Lang. "A Computational Algorithm for Origami Design." *Proceedings of the Twelfth Annual Symposium on Computational Geometry*.

[Liu and Paulino 16] Ke Liu and Glaucio H Paulino. "MERLIN: A MATLAB implementation to capture highly nonlinear behavior of non-rigid origami." *Proceedings of the IASS Annual Symposium*.

[Liu and Paulino 17] K Liu and G H Paulino. "Nonlinear mechanics of non-rigid origami: an efficient computational approach." *Proceedings of the Royal Society A*.

[Peraza Hernandez et al. 16] Edwin A Peraza Hernandez, Darren J Hartl, Ergun Akleman, and Dimitris C Lagoudas. "Computer-Aided Design Modeling and analysis of origami structures with smooth folds." *Computer-Aided Design* 78 (2016), 93–106.

[Saito et al. 14] Kazuya Saito, Sergio Pellegrino, and Taketoshi Nojima. "Manufacture of Arbitrary Cross-Section Composite Honeycomb Cores Based on Origami Techniques." *Journal of Mechanical Design* 136:May (2014), 1–9.

[Schenk and Guest 11] Mark Schenk and Simon D Guest. "Origami Folding: A Structural Engineering Approach." In *Origami*^{5}*: Proceedings of the 5th International Meeting of Origami Science, Mathematics, and Education*, pp. 291–304. CRC Press, 2011.

[Schenk et al. 14] M Schenk, S D Guest, and G J Mcshane. "Novel stacked folded cores for blast-resistant sandwich beams." *International Journal of Solids and Structures* 51:25-26 (2014), 4196–4214.

[Tachi 06] Tomohiro Tachi. "Simulation of Rigid Origami." In *Origami*^{4}*: Proceedings of the 4th International Meeting of Origami Science, Mathematics, and Education*, 2006.

[Tachi 10] Tomohiro Tachi. "Freeform Variations of Origami." *Journal for Geometry and Graphics* 14:2 (2010), 203–215.

[Tachi 13] Tomohiro Tachi. "Interactive Form-Finding of Elastic Origami." *Proceedings of the International Association for Shell and Spatial Structures (IASS) Symposium* :5 (2013), 7–10.

[Tang et al. 16] C C Tang, P B Bo, J Wallner, and H Pottmann. "Interactive Design of Developable Surfaces." *Acm Transactions on Graphics* 35:2 (2016), 12.

Amanda Ghassaei
Center for Bits and Atoms, Massachusetts Institute of Technology, Cambridge, MA, e-mail: amandaghassaei@gmail.com

Erik D. Demaine
Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 32 Vassar Street, Cambridge, Massachusetts 02139, USA e-mail: edemaine@mit.edu

Neil Gershenfeld
Center for Bits and Atoms, Massachusetts Institute of Technology, Cambridge, MA, e-mail: neil.gershenfeld@cba.mit.edu