# Discrete Shells Origami

**Rob Burgoon**
**Graduate student**
**Computer Science**
**Calif. Polytechnic State Univ.**
**San Luis Obispo**

**Zoë J. Wood**
**Assistant Professor**
**Computer Science**
**Calif. Polytechnic State Univ.**
**San Luis Obispo**

**Eitan Grinspun**
**Assistant Professor**
**Computer Science**
**Colombia University**

**Abstract**

We introduce a way of simulating the creation of simple Origami (paper folding). The Origami is created in a thin shell simulation that realistically models the behavior and physical properties of paper. We demonstrate how to fold and crease the simulated paper wherever the user desires. This work employs cutting-edge advances in the field of discrete shell modeling to meet the challenge of simulating Origami. We found that the discrete shell model is capable of creating simple Origami that does not involve paper to paper collisions. For more advanced origami, however, some kind of collision detection and resolution scheme is required. Further research is necessary to implement collision handling while maintaining a practical simulation speed.

## 1. INTRODUCTION

Origami, the ancient Japanese art of folding paper to represent real objects, has been practiced as both an art form and as a form of entertainment. In order to create the desired representation, one must make precise folds on a square sheet of paper. Origami is usually fashioned using only folds; the use of cutting or gluing is frowned upon. One would not normally consider origami and computer science to be related, yet a fascinating topic emerges when the two subjects unite.

In computer science, the computational simulation of real world objects attempts to model the behavior of the object. It does this by utilizing math and physics to predict how the object would act under the same circumstances in real life. This alone will produce results in the form of raw data, but presenting the simulation as a graphical application will better engage the user. Simulating origami graphically by taking input and displaying its results is an intriguing fusion of these concepts.

The paper used for origami is an example of what in computer science is called a thin shell. Thin shells are defined as "thin flexible structures with a high ratio of width to thickness" [1]. Until recently, many thin shell simulations in graphics could only represent a mesh formed of plates that resists change from a flat configuration. Recent breakthroughs in thin shell modeling now allow shells to resist deviation from a curved, undeformed state. This allows the modeled paper to behave more like real paper, as folds can be modeled as local changes to the rest curvature of the paper. Additional benefits include being able to form origami from curled paper and causing the simulation of folded paper to realistically react to external forces. These developments make a fast, unscripted origami simulation possible.

### 1.1 Contribution

Using the discrete shell model as a foundation, we present an interface for creating origami. The interface manipulates the mesh used by the discrete shell model to create creases and folds in the simulated paper. This folding involves changing the connectivity of the mesh, as well as its properties. The paper simulation is a product of cutting-edge advances in the field of discrete shell modeling, whereas the creation of origami folds is a novel effort (further details on the paper simulation can be found in the next section). The mesh in the simulation is a two-dimensional triangular structure that can be moved, rotated, and changed using the mouse. The mesh changes to reflect the results as the simulation progresses. While the simulation does not progress in real time, folds applied to the paper are not scripted in advance; rather they are executed upon request as the application runs. This work represents a practical use for the discrete shell model. Figure 1 demonstrates the possibilities of origami simulation.

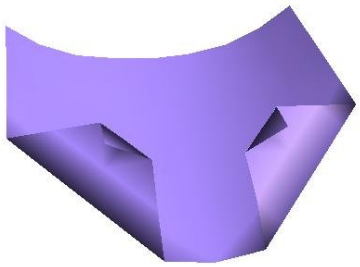


Figure 1: A curved piece of Origami paper that has been folded

### 1.2 Foundation and Related Work

Here we briefly overview the physical modeling concepts and some recent related work that represents the foundation of the Origami simulation. We will cover particle systems, cloth simulation, recent advances in cloth simulation [2], and the discrete shell model that is simulating the Origami.

### 1.3 Particle Systems

Particle systems are a way of modeling something composed of many elements without having to focus on the behavior of each individual element. Common examples of this include models of fireworks, sparks, or a school of fish [3]. This is done by giving the elements initial conditions, then applying forces to the system, such as gravity and drag. The forces can be those found in the laws of physics, or they can be behavioral forces, such as a fish being attracted to the center of the school but repelled by nearby fish. The simulation is broken up into time steps in which new positions and velocities are found in each step, using the old positions, velocities, and forces. Classic particle systems use Euler's method of integration to solve for the new properties of each particle [4]. Euler's method as it is applied to particle systems is shown in Figure 2 below.

$$velocity(t) = velocity(t-1) + dt(acceleration(t-1))$$
$$position(t) = position(t-1) + dt(velocity(t-1))$$

Figure 2: Euler's method used to find new positions and velocities for particles

### 1.4 Classic Cloth Simulation

A cloth simulation is a particle system that attempts to physically model cloth [5]. The cloth is represented as a triangle mesh, in which the vertices serve as the particles of the system. Each vertex has a mass associated with it, as well as springs that connect the vertex to nearby vertices. The underlying spring system gives the mesh cloth-like properties. Like the particle systems that it is based on, classic cloth uses Euler's method to make its time steps. One of Euler's method's shortcomings is that when strong forces are applied, there is a danger of instability when the time steps being taken are too large. For example, if a vertex of a rigid cloth model is displaced, strong forces will occur to return the vertex to a lower energy state. If the next time step is too large, then the vertex might repeatedly overshoot the lower energy position, resulting in divergent and unstable behavior. Stiff cloth tends to be either computationally expensive or unstable.

### 1.5 Large Steps in Cloth Simulation

Recent work by Baraff and Witkin introduces a cloth simulation scheme that can stably model stiff cloth while taking large time steps [2]. The authors abandon Euler's integration method in favor of an implicit integration scheme shown in Figure 3. The implicit method they describe solves for the new position and velocity. This is interesting because it uses the new position and velocity as input, in addition to the old position and velocity. This method, backwards Euler's time step, is more complex than Euler's, taking longer to solve during each time step, but it yields improved stability, which allows much larger time steps. This greatly enhances the overall performance when simulating stiff cloth, such as clothing. The following section on Discrete Shells uses this work as inspiration, but uses a simpler implicit integration method.

$$\begin{pmatrix} \Delta x \\ \Delta v \end{pmatrix} = h \begin{pmatrix} v_0 + \Delta v \\ M^{-1} f(x_0 + \Delta x, v_0 + \Delta v) \end{pmatrix}$$

Figure 3: Baraff and Witkin's implicit integration scheme

### 1.6 Discrete Shells

In order to allow users to interactively create virtual origami, we need both a way to simulate the paper and a way to create folds. The paper simulation is built upon previous work [1], whereas the creation of origami folds is a novel effort.

The simulation of paper is done using the discrete shell model created by Grinspun et. al. [1]. In essence, the discrete shell model is a cloth simulator, modified to graphically model thin shells that have a curved rest state. The discrete shell model is based upon Baraff and Witkin's cloth simulation, using the Newmark integration scheme. The Newmark scheme is similar to Baraff and Witkin's in that it is more stable than Euler's, but is much more straightforward to solve. The Newmark scheme is shown in Figure 4. The beta and gamma constants are used to control the amount of bias between the old and new forces. Gamma values above 0.5 add damping.

$$x_{i+1} = x_i + \Delta t_i v_i + \Delta t^2 \left( (1/2 - \beta) a_i + \beta a_{i+1} \right)$$
$$v_{i+1} = v_i + \Delta t_i \left( (1 - \gamma) a_i + \gamma a_{i+1} \right)$$

Figure 4: Newmark integration scheme

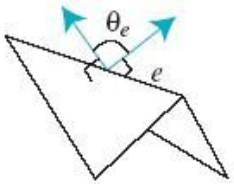
Thin shells can be anything from soda cans, to hats, to sheet metal, to origami paper. The advantage of modifying a cloth simulator to handle discrete shells is the existence of robust and reasonably fast methods of simulating cloth. When a fast cloth simulation is combined with a discrete method for modeling the curved thin shells, the result is a fast and powerful model for not only cloth, but other thin shells as well.

Simulating a curved rest state implies that the material can resist changes in its angle of curvature. For example, a sheet of paper that has been rolled tightly will tend to maintain that curvature and resist being straightened out or rolled tighter. In this example, a simulated sheet of paper that has been creased and rolled has been pinned to a wall to demonstrate its resistance to changes from its curved and folded state.

Membrane forces and flexure forces are necessary to construct shells that will attempt to keep their shape and curvature. Membrane forces can be

thought of as the shell's resistance to stretching and shearing, and flexure forces as the shell's resistance to bending from its rest state. The existing cloth systems already account for membrane forces on the thin shell, and Grinspun introduces a discrete way to model flexure forces.

Like the cloth simulation described earlier, the discrete shell model represents the material as a mesh of triangles. To account for the flexure forces, each adjacent pair of triangles, or faces, has an energy that is a function of the angle between the faces ($\theta_e$ in Figure 5) and the angle that represents the rest state between the two faces. The angle that represents the rest state is known as the rest-angle. The function uses the length of the edge (e in Figure 5) and one third of the average adjacent face height ($h_e$ in Figure 5). The height average is used to account for the curvature of the material being modeled. As the flexural energy changes, forces are applied to the masses in the faces to reflect the edge's desire to return to the rest state. This allows a curved surface to resist changes in curvature.




$$W_B(x) = \sum_e \left(\theta_e - \bar{\theta}_e\right)^2 \| \bar{e} \| / \bar{h}_e$$

Figure 5: An adjacent pair of faces and bending energy along the shared edge

### 1.7 Inelastic Deformation of Thin Shells

A work that expands on the Discrete Shells paper is "A Discrete Model for Inelastic Deformation of Thin Shells" [6]. This paper adds additional functionality to the discrete shell model by simulating how thin shells deform and fracture under stress. Examples include a light bulb being broken, or the crushing of an aluminum can. A difficulty overcome by the authors was to have fractures occur that are not limited to pre-existing edges. The simulation of origami presents a similar challenge of allowing folds to be made away from the pre-existing edges of the mesh that models the paper.

### 1.8 Additional Related Work

There have been other recent advances in the use of physically based modeling techniques for the purpose of simulating real world effects. Physical modeling is not only used for thin shell applications. It can also be used to model parts of living things such as organs, muscles, and skin. Mimicking the mesmerizing properties of fire is another fascinating application of physically based modeling. Stahl et al. [7] have applied a physical model to the simulation of bags of particles which can be made to model the behavior of vital organs such as a human heart as it fills, stretches, and compresses. Like the discrete shell model, the bag of particles is a sort of shell that attempts to retain its shape as forces are applied to it.

Not all physically based models use a system that is spring-based, or even compressible for that matter. Significant progress has been made in the area of simulating fire. Nguyen et al. [8] produce a great looking flame simulation using incompressible flow equations applied to voxels (volume pixels).

A recent work that is very similar to discrete shells attempts to simulate the characteristics of the human hand. Using pseudo muscles that deform in ways that mimic human muscles, Albrecht et al. [9] are able to move the bones of the hand using physical laws. Covering the muscles and bones is a skin model is quite similar to the discrete shell model. The skin uses a mass and spring system to react to the changing shape of muscles and bone it contains.

## 2 IMPLEMENTATION

In this section we cover the interface presented to the user, and how the origami simulation works beneath the user interface. The user interacts with a 3D model of the origami paper using the mouse. The paper is shown as a mesh and as the user makes folds, the mesh simulates a paper's reaction to the folds. The results are displayed as the simulation progresses, and the paper's behavior can be recorded for playback at a later time. The Origami simulation runs in C++, using OpenGL for displaying the paper, along with some math modules used in the discrete shells work [1].

### 2.1 Discrete Shells

The Origami simulation is implemented using the discrete shell model as an underlying engine. The shell model is adjusted to model the physical attributes and behavior of Origami paper. This allows the user to modify properties of the paper in the same way that folding changes physical paper, while the discrete shell simulation attempts to react to these changes in a realistic fashion. The remaining sections of this chapter cover the major additions and modifications made to the discrete shell model to make virtual Origami possible.

### 2.2 Edge Making

The Origami simulation should support folds from any pair of vertices that the user specifies. The fold will be made along a straight line between two points according to where they would be if the paper were still flat. Since the paper can only be changed along the edges of the mesh, if the desired fold does not lie along existing edges, new edges must be created. These new edges will

be created as if the paper is lying flat, and will be transformed to where they belong in the paper's current state.

One must first find the line along which the fold is to be made. Then, new edges along the line must be created. After each new edge is made, another edge must be added to ensure that all faces are triangles.

The first task in creating a fold is to find an equation for the line that intersects the vertices selected for the beginning and the end of the fold. The representation of the vertices used are the two dimensional coordinates from when the mesh was still a flat piece of paper. An example of a possible current configuration of the paper and its original flat orientation are shown in Figure 6.

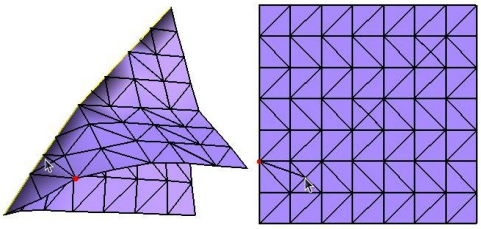
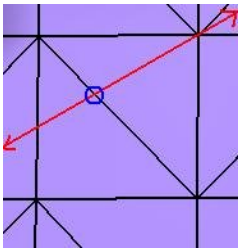


Figure 6: Relationship between the path of a new fold and the original mesh

Using the original two dimensional coordinates greatly simplifies mapping the line to the mesh and determining where new edges must be made. If the current location of the vertices and the mesh were used, then the line would have to be projected onto the mesh faces to determine where new edges should lie. This would make the simulation less user-friendly, as one would have a very difficult time making the straight folds that origami requires.

The main disadvantage of mapping the folds onto the original flat paper is that it must be possible to flatten the paper into two dimensions. This constrains the origami simulation to initially flat paper, as opposed to other possibilities that the virtual domain makes possible. Someday we might desire virtual origami constructed from a hollow sphere of paper instead of a sheet. Such an undertaking would certainly be a radical move. However, for the purposes of this work, we feel that requiring our paper to begin flat is a good compromise for now.

After finding the fold-line, we then make new edges in the mesh which follow the fold line. Starting from the first vertex, we split faces as we proceed toward the final vertex. Each face has an edge opposite the current vertex that could be split to create a new edge from the current vertex to some point on the opposite edge. The equation of a line along the opposite edge is found and used to find an intersection (see Figure 7). As mentioned earlier, we use the flat coordinates of the edge. If the intersection occurs inside the boundaries of the edge, the edge will be split, creating a new edge along the desired path of the fold. The new vertex created will become the current vertex, and the process will repeat until the final vertex is reached. The results of this process are shown in Figure 8. Sometimes, the fold specified by the user will happen along existing edges. When this is the case, instead of creating a new edge, the existing edge will be traversed and the loop will continue with the vertex on the other side of the existing edge. Another complicating situation is when the opposite edge is to be split near an existing vertex. In this case the existing vertex is simply shifted to where the new vertex should be.



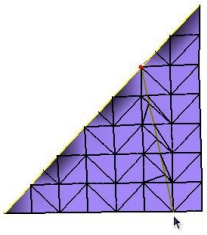
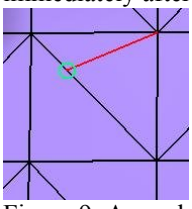


Figure 7: Applying the fold line to a face to determine where to add a new edge

Figure 8: A new fold is created and additional edges are formed as a side effect

In Figure 8 there are many other new edges besides those along the new fold. These are added to repair faces adjacent to a face that has been split in two to create an edge. In the mesh system used by the discrete shells model, all faces must be only three-sided. Having a vertex in the middle of an edge makes the edge become two edges, one on either side of the vertex. After a face is split by a new edge, the adjacent face will find itself with four edges. This condition is shown in Figure 9. In order to address this, when a face is split in two, the face that shares the split edge will split itself as well in order to preserve the mesh. This remedy is demonstrated in Figure 10. This is done during the fold-making process immediately after any edge is split.



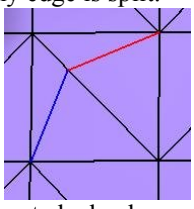


Figure 9: A newly created edge has accidentally formed a four-sided face

Figure 10: Another edge is created to split the four-sided face into two triangles

### 2.3 Reference Angle Modification

As the Discrete Shells simulation runs, it uses a variable called phisRef, stored in a face, that holds a function of the rest-angle to each neighboring face. A primitive way to create folds would be to immediately change phisRef to the new desired angle. This does not

work well, as it would be akin to a real sheet of paper, initially flat, suddenly informed that it should be at rest folded in half. Early experiments demonstrated that when the thin shell model attempts to simulate this, it most often results in violent, unnatural oscillations. In this case, either the oscillations last until the damping eventually removes them, or the model simply blows up.

In order to eliminate this problem, the folds are made gradually. When a fold is made, the face records the current rest-angle, the desired angle, and the current simulation time. During each step of simulation, the phisRef values for each edge on each face are changed linearly to the new rest angle with respect to time. Making folds gradually and not making sudden changes to the rest state of the mesh improves results.

The angle of the fold is not used directly in the discrete flexural energy formula. Instead, the angle is plugged into an equation using a tangent function (Figure 11) in order to intensify the forces as the angle approaches 180 degrees. This is done to compensate for the lack of collision detection in the simulation. For this reason, when a tight fold is needed, it is better to fold at an angle no greater than 170 degrees, to prevent the forces from becoming too dramatic.

$$refAngleUsed = 2 * \tan\left(1/2 * refAngle\right)$$

Figure 11: Reference angle function to compensate for lack of collision detection

### 2.4 Vertex Constraint

In order to allow the user to hold the paper in place while folding it, vertices can be constrained. This means that the vertex will no longer be able to move and react, and its immobility will affect the way the rest of the mesh behaves. This is done by setting the velocity of the vertex to zero, thus locking its current position. The discrete shells simulation handles this by assigning the constrained vertices very large values for mass. Doing so causes the rest of the mesh to behave in a manner taking the stationary parts into account. The simulation can then progress toward equilibrium with those vertices remaining in their current position.

### 2.5 Adding a New Vertex

In order to allow folds to be started and finished from any point along an edge, the user should be able to create new vertices along an edge. As mentioned in the User Interface section, the user simply selects an edge with the mouse, and a new vertex is created at the location of the cursor as shown in Figure 13. In order to do this, we need to know which edge the user selects, and at what point along the edge the mouse cursor lies when the selection is made. The discrete shells simulator uses openGL picking code to determine which edge the user wishes to manipulate. To determine the location of the mouse at the time the edge is clicked, we need to calculate the coordinates of the mouse click in the same coordinate system as the endpoints of the edge. This is done using the openGL function call gluUnProject which translates 2D pixel coordinates to 3D world coordinates. Feeding the function the current transforms and the pixel coordinates of the click yields the world coordinate position of the click. The world coordinates of the mouse are compared to the world coordinates of the edge's endpoints, allowing the edge to be properly split in two. When the face is split, the new vertex is created. Finally, to ensure the integrity of the triangular mesh, the neighboring faces must have an edge added to prevent four-sided faces from forming as in Figure 9.

## 3 USER INTERFACE

When creating virtual origami on a computer, there are some basic operations the user will wish to have available. The user will want to be able to view the paper from any angle, create folds in the paper, and prevent parts of the paper from moving. These three abilities are integral to the folding of origami in both reality and virtual reality. The interface allows the user to perform these actions, as well as start and stop the simulation to watch the folds form.

In order to watch from any direction, the user can rotate the paper with the left mouse button on a virtual trackball, and zoom in and out using the middle mouse button. To choose pieces of the mesh for folding along and making other changes to the mesh, the user can employ the right mouse button. It selects vertices, edges, and faces using OpenGL's picking functions.

The user can create folds by selecting two separate vertices of the mesh. When the second vertex is selected, a pop-up window is displayed that requests the angle of the desired fold in degrees (see Figure 12). For valley folds, a positive angle is given, and for mountain folds, a negative angle is given. In the case of valley folds, either side of the fold moves toward the camera, whereas the sides of a mountain fold move away from the camera. These definitions refer to the initial camera and object positions. To make the fold, the user enters the desired angle and selects the OK button with the mouse. This causes the paper to fold along the line between the vertices. The angle at which the paper is inclined to rest along a crease is called the rest-angle. The rest angle along the fold does not change to the specified angle immediately; instead the rest angle between the faces on either side of the fold will gradually change, to minimize the potential energy in the paper. More information about rest angles can be found in the Reference Angle Modification section.

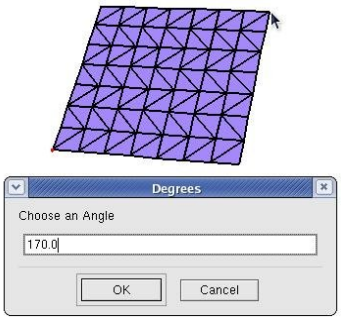


Figure 12: The interface for creating paper folds

Although folds can only be made from vertex to vertex, the user might wish to create an endpoint for a fold where currently a vertex does not exist. In this case, the user can create new vertices at an arbitrary point on an edge. This is done by selecting the edge at the location where a new vertex is desired. The edge will split, creating a new vertex and new edges as shown in Figure 13. Details on new vertex creation can be found in the section entitled "Adding a New Vertex" in this chapter. The additional edges formed after the split are an intentional side effect, to prevent four-sided faces from forming on either side of the selected edge. The problems of having a four-sided face are covered in the Edge Making section and are illustrated in Figure 9 and Figure 10. With the new vertex or vertices in place, the fold can be made as described earlier.

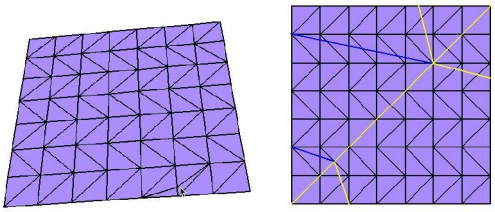


Figure 13: Creation of a new vertex in the paper mesh
Figure 14: Folding diagram of the Origami dog

When the user would like to prevent part of the paper from moving, vertices can be fixed in their current position. This can be done by selecting the same vertex twice, with the right mouse button. This causes the vertex to change color, indicating that it will no longer move, and the rest of the paper will act as if that particular point is immobile. Additional details on holding vertices in place can be found in the Vertex Constraint section of this chapter.

As discussed above, the origami simulation effort mainly concerns tuning the discrete shells simulation to make it accept folds gracefully, and finding ways to convert the user's will into corresponding actions in the simulation. Creating new edges along new fold lines requires careful consideration of the coordinate systems of the paper, the folded paper, and the user's window. These changes also mandate special attention to the connectivity of the mesh and how it must be modified to suit the user's desire. Now that the inner workings have been covered, we are free to move on to the fruit of these efforts.

## 4 RESULTS

The first piece of origami that comes to mind is the famous origami crane. The crane, however, while simple for the origami master is quite difficult to model due to the necessity of creating multiple folds on top of each other. These multiple folds require collision detection and resolution. Handling paper collisions would entail additional computational complexity that would slow down the simulation. Instead, we tested our origami simulation on more feasible origami. This origami involves less than five folds and attempts to capture the spirit and shape of the objects with as little detail as possible. This is similar to drawing techniques in which the artist draws as few lines as possible in order to allow the viewer's imagination to fill in the rest. The simple origami figures created are not original, but were inspired by the examples found at Jean-Jerome Casalonga's website [10].

Our first example of simple origami is the dog. The dog is created in five folds and features a double fold in the tail, where two layers of paper are folded in the same way. It also features an inverse reverse fold in the head. An inverse reverse fold is a fold in which a person folds a paper in half, spreads the corner, and folds the tip of the corner toward the space between the two sides of the crease. The inverse reverse fold is used in making the head of the famous paper crane. Figure 14 above shows the folds needed to make the simple origami dog. In the diagram, folds of 170 degrees are shown by yellow lines, while folds of -170 degrees are shown by blue lines.

The dog looks quite nice during the simulation compared to one folded in real life (see Figure 15). What cannot be shown in the illustrations is how the animation of the folding of the dog, when displayed, seems to give the dog much more life. If the dog is folded for someone with real paper, they may not recognize the dog immediately. If a video of the simulated folding is shown instead, the viewer has an easier time identifying the dog. In lieu of a video, a slide show of the dog being folded is shown in Figure 16.

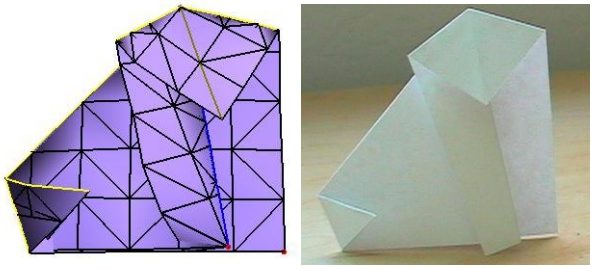


Figure 15: A simulated Origami dog and a paper Origami dog

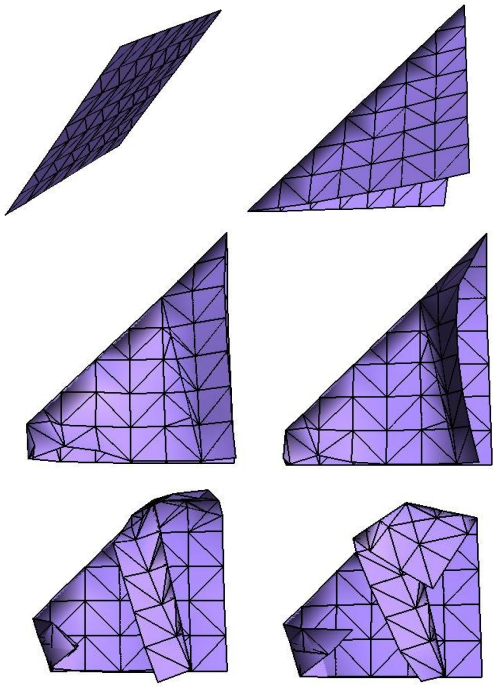


Figure 16: Slide show demonstrating the dog being folded

A second example is the elephant. The elephant has a fairly basic design, consisting of two vertical folds to form the head and trunk, and two diagonal folds in the pieces that are not being used for the trunk. These diagonal folds create the broad elephant ears. These folds are shown in Figure 17.

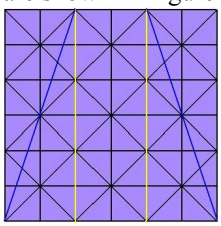


Figure 17: Folding diagram of the Origami elephant

The simulated elephant, along with a real paper elephant is shown in Figure 18. It matches the real life results of the same folding algorithm very well. Like the dog, the ears flap a bit as they come to rest at the desired angles during the simulation. This gives the elephant an energy that folding paper by hand does not quite have.

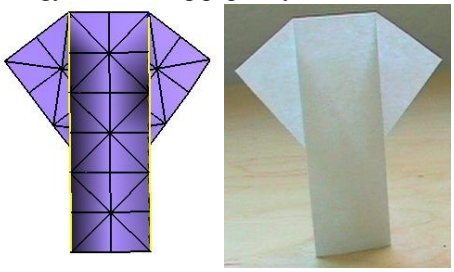


Figure 18: A simulated Origami elephant and a paper Origami elephant

The last example presented is the apple. The apple at first glance appears simple, but is actually quite complicated regarding the behavior of the paper. The apple mainly consists of a mountain fold and a valley fold with a small angle between them. This causes the paper to warp into a concave bowl shape. This three-dimensional effect is what makes the origami apple unusual. Finally, two of the corners on the apple are folded away to make the apple's perimeter appear curved. Although apples are convex in real life, the origami apple is deliberately made concave. The folding diagram of the apple is shown below in Figure 19.

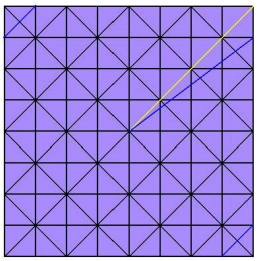


Figure 19: Folding diagram of the Origami Apple

This example is where the thin shell model proves itself. The simulated paper greatly resembles the behavior of real paper as the fold is made to create the stem. Just as in real life, the paper reacts to the fold by becoming concave. The two apples, simulated and real, are shown in Figure 20.

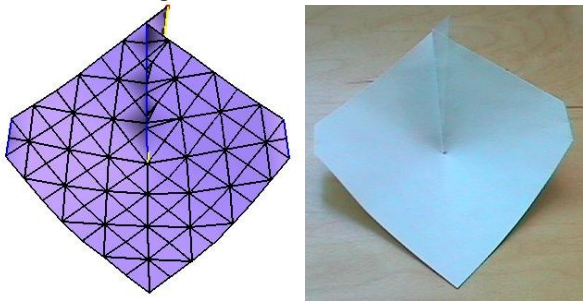


Figure 20: A simulated Origami apple and a paper Origami apple

The teaser image shown in Figure 1 is not an example of real Origami, but is made using the same Origami creation interface as the rest of the results. In this case slight folds are made across the sheet to create a curled effect, and then the corners are dog-eared. This is the result of testing to ensure that curled effects can be made as well.

## 5 CONCLUSION

Whenever one tries to simulate computationally what humans or nature can do easily, many problems arise. This is true whether in the field of artificial intelligence, communications protocol, simulated car

crashes, voice recognition, computer vision, or simply translating requirements into software. The innate limitations of computers and machines are repeatedly underestimated by humans. An example of this is the results of the 2004 DARPA autonomous vehicle rally. Of all the entrants, the most successful only made it a small fraction of the distance required to complete the race, failing a task that most humans can complete while half asleep.

Similarly, attempting to simulate origami proved to be deceptively challenging. The attempts to create dense folds highlighted the lack of collision detection, and folds close to 180 degrees produced overly large forces. Leveraging from existing work also had some inherent difficulties. A large existing code base required tough decisions between reusing and rewriting code. While testing the simulation, correct behavior was difficult to conclusively determine, as incorrect behavior is either hard to detect, or easy to detect but hard to understand.

We were not able to eclipse the great origami masters with a giant three-dimensional paper dragon, nor even make a modest paper crane. We were, however, able to make a great-looking little origami dog out of five folds, and other such simple origami. We allow the user to specify folds and desired fold angles across the paper, as well as hold pieces of the paper in place. With this module plugged into an efficient thin shells simulation, we were able to create our simple origami in a matter of minutes.

### 5.1 Future Work

Future work in creating origami will need to address the challenge of simulating the great number of paper-on-paper collisions that occur within densely folded paper, such as in the paper crane. Baraff and Witkin survey some of the best ways to implement collision resolution. Detecting collisions is fairly simple; testing for masses passing through faces, and edges passing through other edges is straightforward. The difficulty lies in resolving the collisions once they have been detected. One approach is to apply a strong force to separate the colliding cloth. This is effective at resolving the collision, but the use of a strong force requires a smaller step size, slowing the simulation, thus making it impractical. An alternative technique is to simply displace the cloth to prevent the collision. This proves to be an even worse solution, as this introduces violent reactionary forces by the underlying spring system. The reactionary forces mandate a small step size, which slows the simulation [2].

Baraff and Witkin have worked around these issues, and use a combination of these techniques in their simulation. When preventing collisions they apply a strong damped force, which they claim is usually not severe enough to ruin the step size. They solve the displacement problem by modifying their backward Euler equations to prepare the neighbors for the collision that is coming, preventing a violent reaction [2]. The inability to simulate dense folds is one of the weaker points of this work, and implementing Baraff and Witkin's modified differential equation and collision resolution scheme would greatly increase the creative potential of the origami simulation.

Further complicating the desire for collision resolution is the speed of the simulation. Currently, the discrete flexural energy of an edge is based in part upon the heights of the adjacent triangles. Creating folds often leaves triangle slivers, which can result in strong forces and small step size. This could be remedied by either flipping the triangles to improve their aspect ratio, or eliminating the triangle height component entirely. The height component is intended to estimate the curvature of the fold, which, in the case of paper, is either zero or infinite, so it could safely be ignored.

## 6 REFERENCES


[1] Eitan Grinspun et al., "Discrete Shells," Eurographics/SIGGRAPH Symposium on Computer Animation, (2003).

[2] Baraff and Witkin, "Large Steps in Cloth Simulation", Eurographics/SIGGRAPH Symposium on Computer Animation, (2003).

[3] William T. Reeves, "Particle Systems - A Technique for Modeling a Class of Fuzzy Objects", Eurographics/SIGGRAPH Symposium on Computer Animation, (1983).

[4] William H. Press et al., Numerical Recipes in C, (1988).

[5] Breen and House, Cloth modeling and animation, (2000).

[6] Yotam Gingold et al. "A Discrete Model for Inelastic Deformation of Thin Shells," Eurographics/SIGGRAPH Symposium on Computer Animation, (2004).

[7] D. Stahl et al., "Bag-of-particles as a deformable model", Proceedings of the symposium on Data Visualisation (2002).

[8] Duc Quang Nguyen et al., "Physically Based Modeling and Animation of Fire," Eurographics/SIGGRAPH Symposium on Computer Animation, (2002).

[9] Irene Albrecht et al., "Construction and animation of anatomically based human hand models," Eurographics/SIGGRAPH Symposium on Computer Animation, (2003).

[10] Jean-Jerome Casalonga, "Simple Origami", 18 Jan. 1999, online, available from Internet Explorer at http://membres.lycos.fr/jjcasalo/