

# Simulating pleated tension folds

G. Konjevod

keywords: origami simulation, pleat tessellations, tension folding

## Abstract

We present an approach to simulating curved shapes created by tension in a family of origami models derived from Paul Jackson’s *Bulge* [Enc. Origami and Papercraft Techniques, 1991], explored primarily by the author since 2005. Most pieces alternate horizontal and vertical pleats. The pleats interlock, but in some areas of the folded sheet, the arrangement may allow some movement. Since real paper creases don’t fold flat but prefer to stay at a small but non-zero angle, alternating pleats create tension, which curves the models into the third dimension.

Double Wave (Fig. 1, right) is an example of a sequence designed to achieve a form. However, this is generally not easy to do. Thus, it would be of interest to save time by deferring the experimentation to the machine through a model that allows simulation of the equilibrium shape of a folded paper sheet.



**Figure 1:** Basic Form (Bulge) and Double Wave—comparison of photograph and rendering.

Most simulations of origami handle only geometry and not dynamics. Notable exceptions are Schenk and Guest [Origami<sup>5</sup> (91–303)] and Tachi [J. Geom. Graphics, v.14, 2010 (203–215)]. A recent example that extends these is Amanda Ghassaei’s Origami Simulator [<https://tinyurl.com/y9og3aho> 2017]. The sheet is modeled using a system of point masses (nodes) linked by three types of springs (edge lengths, crease angles and face-vertex angles). Then, given a crease pattern together with the preferred fold angle for each crease, forces on the nodes are computed, and the nodes moved over a small time interval, iteratively relaxing towards an equilibrium. Origami Simulator stands out for its efficiency, allowing real-time interaction even for large crease patterns, and the availability of source code, without which the work presented here would have been much more difficult.

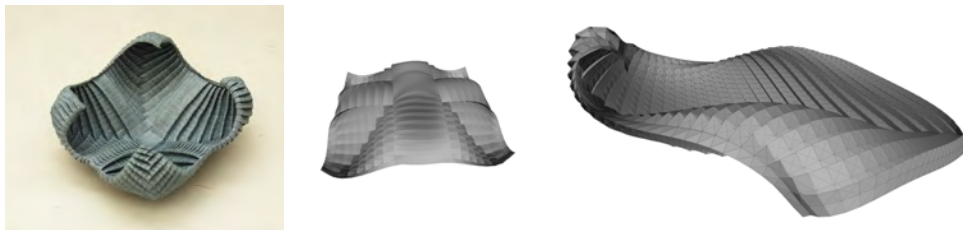
To model the equilibrium state of a tension-folded model, we need additional components. The physics model is basically the same as in Origami Simulator, but in order to simulate tension-

folded forms such as those in Figure 1, we make a step towards handling contact between folded layers. The closest similar problem, collision detection as studied in computer graphics, deals with a smaller number of interacting layers in arbitrary—but typically less complex—arrangements. Fortunately, general collision detection is unnecessary as long as we assume that paper trapped within pleat intersections cannot slip out easily. This allows us to precompute the sets of nodes in contact during the relaxation. Then, it suffices to “glue” such sets of nodes together and apply the resultant of all the individual forces to their union.

**Flat-folded grid.** The mesh for the relaxation is generated step by step, by “folding” each pleat. The internal representation separates the locations of nodes in the plane of the flat-folded grid from the layer information, which is stored using small offsets in 3D. Offsets have two roles: they allow the recovery of local layer ordering (both for nodes and for faces) and, when combined with the flat coordinates, they provide a reasonable starting point for the relaxation.

**Free complex.** In order to avoid modeling contact forces, instead of the mesh derived directly from the flat-folded grid, we find and identify groups of nodes that move together. The assumed high friction between faces in contact allows us to consider the *free complex* of a pleat tessellation. The free complex is generated from the flat-folded grid by merging sets of nodes “trapped together” by the folds. The relaxation process is then applied to the free complex. Forces are computed as before, but the nodes in each equivalence class are treated as one: the forces are added up, as well as their (unit) masses, and a common displacement applied.

**Implementation.** The current implementation is in pure Python. The physics model is based on the Origami Simulator, with minor modifications. Being sequential, it is much slower than the GPU-based Origami Simulator. The flat-folded patterns can be saved in the FOLD format and, using a format extension, so can the simulated meshes and free complex.



**Figure 2:** (Left, middle.) Simple Bowl comparison shows a computed local optimum different from the folded one. It may be possible to “drape” the flat-folded grid over a curved surface before starting the relaxation and change the local optimum found.

(Right.) Basic form (from a different angle of view) using 32 pleat pairs exhibits both extreme curvature at the initial corner (curved tip at the left) and the somewhat messy optimized mesh.

**Future work.** Both initial folding and the free complex computation assume axis-aligned folds. Generalization will complicate the free complex derivation by requiring new nodes if a node is glued to a non-node point of a face. Another useful extension would be to allow changes to target angles of particular creases, to be alternated with relaxation steps. Finally, a dynamic general collision detector, even if applied only every few hundred iterations (the renderings shown here took 10 to 20 thousand) would be useful to prevent very small self-intersections and would help produce meshes easier to manipulate afterwards.