From virtual to physical reality with paper folding

Shaun Bangay

Computer Science Department, Rhodes University, PO Box 94, Grahamstown, 6140, South Africa

Abstract

Objects in a virtual world can be converted into hardcopy by perpendicular projection of each face onto a sheet of paper, cutting and gluing. Previously, use of this technique was restricted to a limited class of polyhedral objects. This paper extends this process to realistic virtual objects, with the traditional origami restriction of using only a single sheet of paper.

A number of algorithms are explored to achieve this goal. The use of heuristics allows solutions to be found without exhaustive search of all possible layouts. Approaches to deal with pathological cases are described.

The techniques have already been successfully applied to a number of complex models, selected from a number of model archives on the Internet. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Polygon nets; Surface representation; Paper folding

1. Introduction

The challenges of origami are twofold: to construct a particular object using a single square of paper; and to achieve the construction without cutting or tearing the paper in any way. This paper investigates the relationship between virtual 3D models and a physical paper construction acceptable to the first requirement of origami.

Typical origami models are created from a skeleton, or base, containing the appropriate numbers of appendages in the correct proportion. The surface detail must be cunningly created from this skeleton. Each model is painstakingly designed by a talented individual who must then patiently diagram each step of the process if the construction is to be repeated. While designing bases has been automated to some extent [8], the remainder of the process still requires skill and insight.

The majority of models used in computer animation and virtual reality make use of a surface representation – a polyhedral approximation of the boundary of the object. Folding these models can be accomplished from a single sheet of paper, although currently cutting is required; a variation of the paper folding art known as kirigami.

E-mail address: cssb@cs.ru.ac.za (S. Bangay).
This paper discusses the conversion of 3D virtual objects into a paper representation of the model, as a form of hard copy from virtual environments. The challenge is to create the models from a single continuous piece of paper; and to work with real models, not only those with restrictive geometric properties.

2. Background

This paper explores the relationship between the standard representation of the 3D polyhedral model, as used to represent objects in virtual reality systems, and their flatten 2D forms. This flattened version is referred to as the folding net of the object. The stages of the process are illustrated in Figs. 2–4, which show the stages in the process of converting an electronic description of a Stegosaurus to a flat paper representation and reconstructing it by cutting and folding.

**Definition 1** (Folding net). The folding net corresponding to a 3D polyhedral object is a set of 2D polygons lying in a common plane. Every polygon in the polyhedral object has a corresponding face in the folding net. The shape of each face is the parallel projection of the corresponding polygon onto a plane perpendicular to that polygon. No face may overlap with any other.

By regarding each face in the net as a node, and each shared edge as an edge in a graph the folding net can be treated as a graph. The algorithms described in this paper prevent the occurrence of cycles in this graph resulting in a tree structure. By constraining the polyhedral models to contain only triangular faces, the maximum number of edges per face is three, and the resulting structure is the trinary tree (or a collection of trinary trees).

**Definition 2** (Trinary tree). The graph equivalent of a folding net, without cycles, produced from a triangulated polyhedron. The root node can be selected arbitrarily, but for convenience in this discussion is assumed to be the first node placed by any of the algorithms.

The techniques described in this paper are intended to be applied to real 3D models as found in virtual reality systems. Some restrictions on the nature of the surface are required. These are far more lenient than those applied elsewhere [2,4,10]. Some success has been experienced in automatically applying them to existing models, which often corrects flaws in the model at the same time.

**Definition 3** (Unfoldable polyhedron). A 3D polyhedron can be unfolded using the algorithms presented if:

1. no more than two faces share a common edge,
2. the surface must possess a consistent orientation (in practice this is equivalent to: two adjacent faces must specify common edges in opposite orders, where vertex order is used to infer orientation),
3. no duplicate faces, or faces with duplicate vertices.

The first requirement can be resolved by opening up any additional faces sharing a common edge. The process is illustrated in Fig. 1. The remainder of the constraints can also be automated.

This definition permits unfolding attempts on polyhedral objects that possess concavities, holes and openings, discontinuities, and self-intersections. The issue of whether these can be successfully unfolded,
Fig. 1. The process of opening up a face, when more than two faces share an edge.

and then refolded is still open [1,4,10]. The purpose of this paper is to find the maximal layout for the folding net, provided one exists.

**Definition 4** (Maximal layout). The maximal layout of a folding net requires every face in the net to be connected to every other face via at least one sequence of faces and common edges, provided such a sequence exists in the original polyhedral model.

Thus in a maximal layout, all connected (via a sequence of common edges) polygons in the polyhedral object can be flatten onto a single piece of paper. The definition of an unfoldable polygon is necessary, but not sufficient to guarantee the existence of a maximal layout. A simple counter-example is an open polyhedron with all faces having one vertex in common and sum of the angles between face edges sharing that vertex exceeding \( \frac{360}{\pi} \).

### 3. Related work

Construction of Platonic and Archimedean solids and other regular polygons using paper is common. Modular origami is often used to construct such symmetrical geometric objects [5], by combining a number of identical modules, each folded from a single sheet of paper. Mention has been made of constructing more complex 3D objects during the early days of computer graphics, by cutting out the individual faces and sticking them together.

The earliest mention of nets to represent polyhedra is attributed to Albrecht Durer in his book *Painter's Manual*, published in 1525. Numerous books (see, for example, [7,11]) which provide the nets to construct geometric polyhedra are currently available. Puzzles and space filling polyhedra kits may also be found [6]. Nets for complex, or irregular surfaces are less readily available.

Touch-3D [9] is a modeling package which allows unfolding of the models to create the corresponding net. The nets can be manipulated manually to reorganize the layout. While no further information regarding the generation of the net is available, the demonstration uses simple models, and appears to separate the net where conflicts occur.
HyperGami [2] (and its successor JavaGami) is a package designed specifically to create folding nets. The models created in this package are limited to collections of convex polyhedra. Faces which overlap with another face during net construction are queued until an alternative position is available.

Both packages support texturing of the faces in the object, providing additional surface detail for the reconstructed model.

4. Creating folding nets from 3D models

The process of creating a folding net is not particularly challenging, particularly when applied to simple, convex objects. In reality, 3D objects that do conform to these constraints require additional effort to ensure that a maximal non-overlapping layout can be achieved.
while there are faces still to be added to the net
    Clear the current net.
    Choose an initial face and add it to the net.
    while there unplaced faces, sharing an edge with a face in the net
        Choose one of these faces.
        Calculate the projection of the face.
        if the face overlaps with an existing face in the net then
            Return face to the list of unplaced faces.
        else
            Add the face to the net.
        end if
    end while
end while

Output the current net on its own page.
end while

Fig. 5. Construction of a folding net.

The algorithm for creating the folding net (shown as Fig. 5) is similar to that described for the HyperGami system [3], but with a few significant variations.

This form of the algorithm has an enhancement over others that have been reported. Multiple pages are produced when the model consists of several disjoint objects. It does however assume that a maximal layout will be produced.

Checking for intersections between polygons in general can be a complex operation, but in this case some simplifying assumptions can reduce the complexity of the problem. A new face must be checked against faces already present in the net. The new face will share an edge with one of the faces in the net. If the requirements for an unfoldable polyhedron are satisfied then it will cover at least some unoccupied territory. Face intersection can then reduce to edge segment intersection between edge of the new face and those already in the net. With some allowance for arithmetic rounding, this works well for all irregular
shapes. Regular polyhedra can have faces whose segments overlap exactly in the folding net. These can be detected easily as a special case.

The order in which faces are chosen for addition need not be purely random, but can be determined in a manner which influences the layout of the folding net. If one considers the trinary tree representation of the folding net, then choosing faces can be done in:

- depth first order: produces a diagram with sometimes fewer overlaps, but with long tendrils that sprawl over the page, leaving a lot of wasted whitespace.
- breadth first order: may cause more overlaps, but produces a compact diagram that covers the page more efficiently.

In addition, choosing faces that are coplanar to faces already in the net reduces the numbers of folds required in the finished model. However, from an assembly point of view there are two schools of thought on the matter: one which finds rejoining coplanar polygons easiest; and one that does not!

To achieve a maximal layout, the algorithm needs to be enhanced to cope with situations where naive ordering of faces is insufficient. The following sections describe two algorithms for deciding the correct ordering. The first manipulates the order of the faces as they are extracted from the object, the second manipulates the partially completed folding net directly. Considering the analogy to hidden surface removal algorithms [12], they are referred to as object space and image space routines, respectively.

Two approaches for extending these algorithms in the event of their failure are also suggested. To date unresolvable situations have not arisen and implementation of these approaches has thus not been attempted.

4.1. The object space algorithm

The object space algorithm (Fig. 6) takes into account the situation where a net cannot be created by incremental addition of faces.

This algorithm assumes that the polyhedral object can be decomposed into a satisfactory folding net by permuting the order in which faces are added during the net construction. In this case, satisfaction is achieved with a maximal layout. For a well behaved object, this will result in the net being produced on a single sheet of paper.

Exploring every permutation of faces is of factorial complexity, and is not feasible for most models. The effects of reordering faces cannot be easily predicted without actually re-attempting the layout. The algorithm uses a heuristic to choose each new ordering by moving colliding faces to the leaf nodes of the trinary tree where there is more space and less chance of collision, and by moving unplaceable faces toward the root to secure them a position.

Selection of faces is performed by choosing the face with the lowest priority. Thus migration toward the leaf nodes involves raising priority values. At the end of each attempt to generate a net, the priority values are raised according to the value of the face factors. The factors are measured during the building phase, and are generated as follows:

- Factor A: number of times the face is a common polygon, found at the root of the two branches which contain in the two colliding faces.
- Factor B: number of times a face is involved in a collision, when the face is already part of the net.
- Factor C: number of times a face collides, when it is in the process of being added.
while there are faces still to be added to the net
repeat
Clear the current net.
Choose an initial face and add it to the net.
while there unplaced faces and a free edge in the net to which they might yet be fitted
Choose one of these faces according to the face priorities.
Calculate the projection of the face.
if the face overlaps with an existing face in the net then
Update face factors.
Return face to the list of unplaced faces.
else
Add the face to the net.
end if
end while
Adjust the face priorities using the face factors.
until the created net is of satisfactory quality
Output the current net on its own page.
end while

Fig. 6. The object space algorithm.

The weightings of these factors, and the method for applying them when used to adjust priorities is not obvious. A number of different priority schemes have been tested, and the most promising are described below. The variables $a$, $b$ and $c$ represent weights for factors $A$, $B$ and $C$, respectively.

**Scheme 1**

$$\text{Pri}_{\text{face}} \leftarrow \text{Pri}_{\text{face}} + aA_{\text{face}} + bB_{\text{face}} - cC_{\text{face}}.$$  

(1)

The face factors are used directly manipulate the priority of every face. This scheme allows large changes in face priority for each cycle. Changes occur rapidly and the system moves quickly toward its goal. This can cause it to overshoot the mark, and have difficulty homing in on the final value.

**Scheme 2**

$$\text{Pri}_{\text{face}} \leftarrow \text{Pri}_{\text{face}} + a \begin{cases} 0 & \text{if } A_{\text{face}} = 0 \\ 1 & \text{if } A_{\text{face}} > 0 \end{cases} + b \begin{cases} 0 & \text{if } B_{\text{face}} = 0 \\ 1 & \text{if } B_{\text{face}} > 0 \end{cases} - c \begin{cases} 0 & \text{if } C_{\text{face}} = 0 \\ 1 & \text{if } C_{\text{face}} > 0 \end{cases}.$$  

(2)

This version allows the priority to increase by at most 1 unit for each of the factors, causing the permutations of faces to change more slowly, and explore intermediate permutations more thoroughly.

**Scheme 3 and 4**

As for schemes 1 and 2, but only applying factor $C$ to those faces which could not be placed at all, even after deferral.
Schemes 5 and 6
As for schemes 3 and 4, but calculating the face factors for collisions only when a face no longer has a chance of placement after deferral.

4.2. The image space algorithm

The image space algorithm manipulates a partially correct folded net directly in the attempt to improve the level of correctness. The action of this algorithm can be compared to unraveling only the tangled portion of the net, and reweaving in the hope of producing less overlapping faces. The outline of the algorithm is given as Fig. 7.

Clear the current net.
call GenerateNet.
while collisions occur
    Choose a collision based on current criteria.
    Choose a node in the net related to this collision.
    Make a backup of the current net.
    Delete portion of the net from the selected node.
call GenerateNet.
    if current net is worse than original then
        Restore original net.
    end if
end while
Output the current net on its own page.

procedure GenerateNet
    while a face can be added to the current net
        Choose a face for addition.
        Calculate the projection of the face.
        if face collides with another in the net then
            if the face could be attached to another face then
                Defer the face.
            else
                Record the collision.
                Include the face in the net.
            end if
        else
            Include the face in the net.
        end if
    end while
end procedure

Fig. 7. The image space algorithm.
This algorithm also makes use of heuristics, attempting to successively resamble portions of the net in order to meet certain criteria. Using only one criterion usually results in the algorithm being locked into a local minimum for that value. Thus, for best success with this algorithm, it is recommended that multiple criteria be used, each taking over when its predecessor is unable to optimize further. The criteria that are found to be most useful are described below.

Cycle avoidance code is necessary when operating these heuristics.

**Minimal collisions criterion**
- **Optimization requirement**: minimize the number of collisions.
- **Collision selection method**: none, all nodes are applicable.
- **Node selection method**: node closest to a leaf node to which this criterion has not recently been applied.

**Migrate collisions criterion**
- **Optimization requirement**: minimize the distance of collisions from nearest leaf node. Minimizing number of collisions as a secondary requirement also improves the speed of convergence.
- **Collision selection method**: collision furthest from a leaf node.
- **Node selection method**: node which serves as the common root node of the pair involved in the collision.

### 4.3. The divide-and-conquer approach

There are some changes to the model which can be justified in the context of having specified the problem as being one of producing paper models of 3D polyhedral surfaces. Introduction of extra faces which can be folded as tabs and hidden inside the reconstructed model will not affect the appearance of the final product. These tabs can be used to open up the resulting net, providing additional area for faces placement, and resolving some of the potential collisions between faces in the folding net. The addition of the tab faces into a polyhedron, and the effect on the folding net is illustrated in Fig. 8.

This construction could potentially allow two folding nets to be generated independently, and rejoined using the tab construction. This would produce a divide-and-conquer algorithm that could simplify the original polyhedron into components for which folding nets can be easily generated. These nets could be produced using the algorithms described previously, and combined using this construction.

### 4.4. The hybrid approach

The image space algorithm currently uses a hybrid approach by alternating between heuristics when each becomes locked in a local minimum for its criterion. This idea can be extended to incorporating a hybrid approach to interleaving the algorithms presented to date.

The object space algorithm is well suited to rapidly achieving a layout containing most of the faces, while the image space algorithm is able to fit in the final missing pieces. Divide-and-conquer could be used to resolve problems areas which would otherwise require excessive computing time.
5. Results

The relative performance of each implemented algorithm is discussed below.

5.1. The object space algorithm

The different schemes for the object space algorithm have been compared by calibrating them with three test objects. These are (as illustrated in Fig. 9):
Table 1
Optimal performance points for each priority scheme. Values of the weights [a b c] are given for the three best results, together with the number of cycles before convergence.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Wine glass</th>
<th>Rocket</th>
<th>Space ship</th>
<th>Combined</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[2 3 0] 8</td>
<td>[0 3 2]99</td>
<td>[2 3 2]15</td>
<td>[0 4 1]</td>
</tr>
<tr>
<td></td>
<td>[3 1 1]20</td>
<td>[1 3 1]147</td>
<td>[4 0 3]62</td>
<td>[1 4 2]</td>
</tr>
<tr>
<td></td>
<td>[1 2 1]21</td>
<td>[4 1 3]272</td>
<td>[1 2 2]97</td>
<td>[3 1 1]</td>
</tr>
<tr>
<td>2</td>
<td>[1 3 0] 6</td>
<td>[0 3 2]157</td>
<td>[4 1 1] 4</td>
<td>[0 4 1]</td>
</tr>
<tr>
<td></td>
<td>[3 4 1]7</td>
<td>[0 2 1]203</td>
<td>[4 2 1]14</td>
<td>[1 4 1]</td>
</tr>
<tr>
<td>3</td>
<td>[2 1 3]4</td>
<td>[1 3 1]29</td>
<td>[4 3 3]12</td>
<td>[0 3 2]</td>
</tr>
<tr>
<td></td>
<td>[0 4 3]7</td>
<td>[4 1 3]227</td>
<td>[0 0 1]34</td>
<td>[0 2 3]</td>
</tr>
<tr>
<td></td>
<td>[1 3 0]6</td>
<td>[2 3 2]37</td>
<td>[4 1 1]5</td>
<td>[0 3 1]</td>
</tr>
<tr>
<td></td>
<td>[0 2 2]7</td>
<td>[2 1 1]39</td>
<td>[0 3 1]30</td>
<td>[1 4 1]</td>
</tr>
<tr>
<td>5</td>
<td>[2 2 1]7</td>
<td>[0 4 1]24</td>
<td>[0 2 3]31</td>
<td>[0 2 2]</td>
</tr>
<tr>
<td></td>
<td>[0 4 3]13</td>
<td>[1 2 2]61</td>
<td>[0 0 1]49</td>
<td>[0 3 2]</td>
</tr>
<tr>
<td>6</td>
<td>[3 4 2]8</td>
<td>[0 4 3]56</td>
<td>[2 0 1]7</td>
<td>[0 4 3]</td>
</tr>
<tr>
<td></td>
<td>[2 3 0]10</td>
<td>[2 2 3]84</td>
<td>[0 4 3]33</td>
<td>[2 2 3]</td>
</tr>
<tr>
<td></td>
<td>[3 4 1]10</td>
<td>[2 4 3]90</td>
<td>[0 2 0]39</td>
<td>[0 4 2]</td>
</tr>
</tbody>
</table>

(1) Wine glass, with 224 faces. This is a simple object of rotation, with a great deal of symmetry.
(2) Model rocket, with 144 faces. The body is an object of rotation, with convexities and concavities. Four projecting fins complicate the unfolding process.
(3) Space ship, with 122 faces. This is a space ship, resembling a futuristic aircraft.
The tests are run with each of the weights for each face factor ranging from 0 to 4. The optimal weights for each model, and for all the models together are shown in Table 1. The number of cycles before convergence is also given. In real time terms, the wine glass took about 0.45 s/cycle, the rocket 0.17 s/cycle and the space ship 0.13 s/cycle on a 400 MHz Pentium II processor.
The results illustrate the difference between the incrementing approaches used in schemes 1 and 2. While scheme 1 can sometimes reach the optimal state rapidly, it is less consistent about reaching this
The speed of convergence can be clearly seen to be dependent on the nature of the surface, rather than on the number of faces involved. If one ignores the occasional random effect, the speed of convergence is related to the angle between adjacent faces. Sharp corners, as are found in the rocket, are more difficult to flatten than the more smoothly rounded surfaces found in the other two models. This is consistent with experiments on other models as well.

At present scheme 4 appears to give best results. The variation in optimal weighting factors, and presence of boundary weights suggests that a greater range of models and weights need be explored before an optimum can safely be determined.

5.2. The image space algorithm

Results for the image space algorithm are harder to quantify. The amount of work for every cycle varies since only a portion of the net is reconstructed. Often the change is discarded when no improvement results. The image space algorithm does have substantial performance advantages.

It is found to be better suited to larger, and more complex models. While often slower to converge than the object space algorithm, it is far more reliable in achieving a result. For the models used in Section 5.1, the image space algorithm takes longer to produce a maximal layout of the folding net. The actual run times are shown in Table 2. On the other hand, it produces nets for more complex models, including ones which the object space algorithm has yet to complete. The largest object yet generated with the image space algorithm contains 822 faces, and took 2 days to flatten on an SGI Octane. This is about twice the size of the largest model completed by the object space algorithm.

6. Conclusions

This paper describes a number of algorithms that can be used to layout real 3D models onto a single sheet of paper. The position of the faces are then fixed relative to the others, reducing correlation and reassembly effort. Reconstruction of the objects in physical reality can be a rewarding and entertaining experience. The reader is invited to attempt the reconstruction of one of the objects mentioned in Section 5.1, whose net can be found in Fig. 10. This folding net uses standard origami diagramming conventions [5]. Further examples can be found at http://cs.ru.ac.za/~cssb/kirigami/.

Generation of the folding nets successfully uses a number of heuristics. Combinations of different heuristic strategies are used to iteratively improve on the quality of the generated net. The heuristics must be tuned to remove dependencies on the surface geometry, achieve rapid convergence and prevent cycling.
Fig. 10. A maximal layout of the folding net for one of the objects from Section 5.1.
Physical instantiation of models from a number of sources on the Internet has been completed. While some preprocessing is required to ensure that topology of the model satisfies the constraints of the algorithm (see Section 2), no other significant manual intervention or changes to the nature of the model are required. While no proof is offered as to the existence of a maximal layout for any class of object, I offer a paraphrasing of the finding given by previous researchers:

“These strategies have been successful in generating a maximal layout for every closed object, and many open polyhedra, conforming to the definition of an unfoldable polyhedron, that has been tested to date.”

6.1. Future work

This paper barely begins to touch on the possibilities offered by this problem. Improvements to the basic algorithms are planned. Where multiple maximal layouts are possible, then some assessment is required to choose the most appropriate form. A method of reducing the amount of paper cutting would also bring this work closer to the ideals of true origami.

The reverse process, of refolding the objects from the folding nets offers great potential for the field of 3D object digitization, and work is currently underway on this problem.

References