Karlsruhe Reports in Informatics 2011,36
Edited by Karlsruhe Institute of Technology,
Faculty of Informatics
ISSN 2190-4782

Creating Optimized Cut-Out Sheets for Paper Models from Meshes

Raphael Straub and Hartmut Prautzsch

2011

KIT – University of the State of Baden-Wuerttemberg and National Research Center of the Helmholtz Association

Please note:
This Report has been published on the Internet under the following Creative Commons License:
http://creativecommons.org/licenses/by-nc-nd/3.0/de.

Creating Optimized Cut-Out Sheets for Paper Models from Meshes*

Raphael Straub and Hartmut Prautzsch

{raphael.straub,hartmut.prautzsch}@kit.edu
Karlsruhe Institute of Technology (KIT), Germany

Abstract. We propose an algorithm that creates optimized cut-out sheets for paper models from textured polygon meshes. Crafting time and usage of paper and glue are reduced by first computing an initial folding tree to unfold the mesh into a plane. Overlaps in the plane are then eliminated by adding new cuts. Finally, glue tabs are generated along all cuts before fitting all unfolded parts onto paper sheets using a 2D bin packing algorithm.

Keywords: polyhedron unfolding, paper model, card model

1 Introduction

Rapid advancement in technology has made virtual 3D models popular and increasingly affordable. However, 3D displays alone are usually insufficient for a complete understanding of virtual objects. Often, physical models are required, but producing them can be laborious and expensive. To build prototypes or small quantities of objects, rapid prototyping systems, such as 3D printers, stereolithography machines, and selective laser sintering systems are commonly used. Injection molding and punching techniques are not suitable as the production of the mold is relatively expensive. To save costs, an alternative solution is to build paper models. Paper models are suitable if high accuracy and stability are not mandatory and if the geometric structures of the model are not too small. Other than lower costs and higher ease of crafting, paper modeling is by itself a challenge enjoyed by many. In Japan, the art of origami, perhaps the first form of paper modeling, was a popular pastime and came about shortly after paper became available. Traditionally, origami only involves folding a square sheet of paper to create simple models. Over the years, it evolved to include cutting and pasting parts of the model together to create complex models.

Origami and paper modeling in general have seen a renaissance in the last few years. This has led to a significant rise in the number of related publications recently, such as [9], with new ideas and theories about paper modeling. An overview over the polyhedron unfolding problem and many references to existing software are given in [13]. Mitani and Suzuki [11] present a method for producing

* This report is based on a presentation given in November 2005 on the SIAM Conference on Geometric Design and Computing in Phoenix, USA.

an unfolded papercraft from a 3D computer model represented by a triangular mesh by approximating the mesh with triangle strips. These triangle strips can be unfolded easily. The main advantage of this strip-based method is that it can be applied to rounded models with many triangles. The final model is assembled using adhesive tape, which is generally faster than using tabs. A disadvantage of using tapes instead of tabs is that it is more difficult to craft closed models as it is difficult to attach a tape on the inside when closing the model in the last step. On the other hand, glue tabs have the advantage that glue joints can be easily realigned before the glue is dry. Although there exist tapes that can be easily removed from paper, they are not strong enough usually to guarantee the stability of the paper model. Therefore, we are working with tabs in this paper.

Many papers deal with the theory of unfolding. The question if one can cut every convex polyhedron along its edges to get a non-overlapping unfolding is a long-standing and still open question in geometry [7, pp. 73–75]. On the other hand, there are many examples of non-convex polyhedra that do not have a non-overlapping unfolding. In many papers, the authors concentrate on special cases like unfolding polyhedra with orthogonal faces [5], or prove that some special polyhedra do not have a non-overlapping unfolding [3]. Other authors discuss cuts across faces [1] or the folding problem [12], which is inverse to the unfolding problem. In [16] several heuristic methods to get a non-overlapping edge unfolding are studied. However, no heuristic algorithm has succeeded in finding a non-overlapping unfolding for every test case.

In this paper, we present an algorithm for creating optimized cut-out sheets for paper models in detail. This algorithm has been implemented in [15] and [4]. Section 3 shows some paper models that were crafted with cut-out sheets automatically generated by our algorithm. Finally, Section 4 concludes this paper with some ideas for future work.

2 Creating Cut-Out Sheets

Our program begins by reading the input, which is represented by an *indexed face set*, stored in a *VRML97* file. The input is a 2-manifold and possibly textured mesh with planar faces that are simple polygons without holes. If there are faces with holes, they can be subdivided. For efficient traversal of the mesh, a *half-edge data structure* [6] is used to represent the mesh internally such that any adjacent entity of the mesh can always be computed in constant time.

2.1 Initial Unfolding

To create the cut-out sheets, we need to unfold the model into the plane by cutting the model along existing edges of the mesh. We refer to these edges as *cut edges* and call the remaining ones *folding edges*.

In order to simplify the description of the unfolding process, we first define two graphs.

Definition 1 (Graph and dual graph of a mesh). *Let M be a mesh. Then*

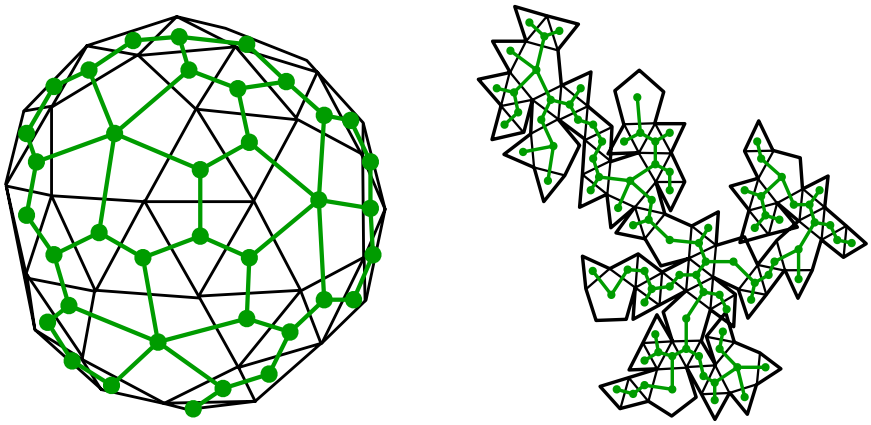
$$\begin{aligned} G(M) &= (V, E)\,, \text{ where} \\ V &= \{\text{vertices of } M\} \text{ and} \\ E &= \{\text{edges of } M\} \end{aligned}$$

is called the graph *of M and*

$$\begin{aligned} D(M) &= (V_\mathrm{d}, E_\mathrm{d})\,, \text{ where} \\ V_\mathrm{d} &= \{\text{faces of } M\} \text{ and} \\ E_\mathrm{d} &= \{(f, g) \mid f \text{ and } g \text{ are faces with a common edge in } M\} \end{aligned}$$

is called the dual graph *of M. An edge (f, g) of $D(M)$ is said to be* dual *to the common edge of f and g in M.*

Figure 1a shows the graph and the dual graph of an example mesh.



(a) A graph (black) and its dual graph (green).

(b) An unfolding (black) with corresponding folding tree (green).

Fig. 1. Initial unfolding of a polyhedral mesh.

Any spanning tree of $D(M)$ defines an unfolding of M. By cutting all edges that have no dual in the spanning tree of $D(M)$, we can unfold M into the plane. For example, we can unfold the faces of M in order of a depth first traversal of the spanning tree.

We call a spanning tree of $D(M)$ a *folding tree* (see Fig. 1b for an example), because it contains only the folding edges. To get an unfolding, we have to determine a spanning tree of $D(M)$. In order to minimize the usage of glue and crafting time, we try to avoid self-overlapping unfoldings forcing us to partition

the unfolding by further cuts into non-overlapping parts as shown in Section 2.2. As we pointed out already, there are polyhedra without a non-overlapping unfolding. On the other hand, for polyhedra that have a non-overlapping unfolding, the only known algorithm to obtain such an unfolding is to try all possibilities. According to [14], the number n of spanning trees of a graph with $|V_\mathrm{d}|$ vertices and $|E_\mathrm{d}|$ edges satisfies

$$n \geq 2^{\left\lceil \frac{1}{2}\left(-1+\sqrt{1+8(|E_\mathrm{d}|-|V_\mathrm{d}|+1)}\right)\right\rceil} \approx 2^{\left\lceil \frac{1}{2}\left(-1+\sqrt{9+8v}\right)\right\rceil}$$

if we assume that the mesh is almost regular and triangular, i. e., $|E_\mathrm{d}| \approx 3v$ and $|V_\mathrm{d}| \approx 2v$, where v is the number of vertices in the original mesh. So the number of spanning trees is exponential in $\sqrt{v}$ and listing all possible spanning trees is impossible in practical cases (e. g. for $v = 500$, we get $n \gtrsim 2^{32} \approx 4.3 \cdot 10^9$). Therefore, we use a heuristic to generate a spanning tree with few overlaps. Below, we define a weight function $w : E_\mathrm{d} \longrightarrow \mathbb{R}^+$ that assigns a weight to every edge in the dual graph, where small weights are assigned to edges whose dual edges in the mesh should not be cut. Accordingly, edges with large weights are more likely to be cut. The folding tree for the unfolding is the minimum spanning tree of the weighted dual graph and it can be computed efficiently using, e. g., Kruskal's algorithm. The weight function is defined as a weighted sum of heuristically determined weights $h(e)$ and user defined weights $u(e)$:

$$w(e) := (1-\alpha) \cdot h(e) + \alpha \cdot u(e)\ ,\ \alpha \in [0,1]\ . \tag{1}$$

Here, α is a user specified parameter that reflects the importance of the heuristically over the user defined weights. If the user wants to cut an edge with dual edge e, he could choose $u(e) = \infty$ and $\alpha > 0$. If he wants to prevent the edge from being cut, he could choose $u(e) = 0$.

Following [16], we used two heuristics to define the weights $h(e)$. The first one is the *minimum perimeter* heuristic. Because the perimeter of the unfolding of a closed mesh is twice the sum of the lengths of all cut edges, this method minimizes the perimeter of the unfolding by favoring cuts along short edges using the normalized weight function

$$m(e) := 1 - \frac{l - l_\mathrm{min}}{l_\mathrm{max} - l_\mathrm{min}} \in [0,1]\ ,$$

where l is the length of the edge with dual edge e and l_min and l_max are the lengths of the shortest and longest edge, respectively. The other heuristic is called *flat spanning tree*. Here cuts along edges having roughly an arbitrary but constant direction $\mathbf{c}$ ($\|\mathbf{c}\| = 1$) are preferred:

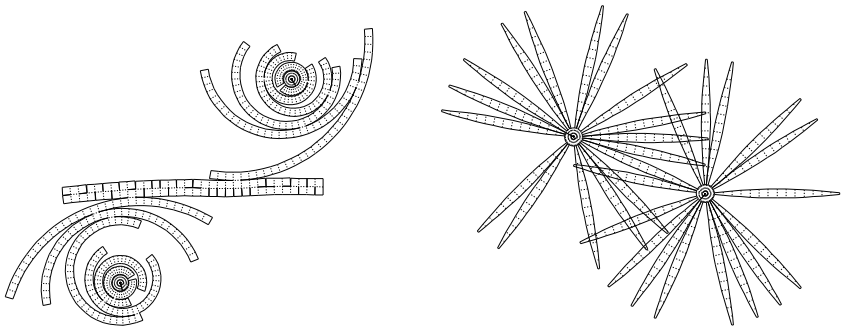
$$f(e) := \frac{|\mathbf{c}^\mathrm{t}(\mathbf{p}-\mathbf{q})|}{\|\mathbf{p}-\mathbf{q}\|}\ ,$$

where e is the dual edge of the edge $(\mathbf{p}, \mathbf{q})$. Combining both weights, we obtain

$$h(e) := (1-\beta) \cdot m(e) + \beta \cdot f(e)\ ,\ \beta \in [0,1]\ .$$

Here again, β is a user specified parameter, similar to α in Equation (1).

After computing the minimum spanning tree with the weights in Equation (1), the computation of the corresponding unfolding is straightforward as explained above. Starting from an arbitrary face of the mesh, all neighboring faces in the spanning tree are unrolled onto the plane. Figure 2 shows the results of using both heuristics on a globe model with meridians and parallels. In this



(a) Minimum perimeter. (b) Flat spanning tree.

Fig. 2. Applying different unfolding heuristics to a globe.

case, the flat spanning tree heuristic yields overlaps, but these overlaps can be removed by only one additional cut.

2.2 Removing Overlaps

As mentioned before, an unfolding could have overlaps. To remove these overlaps, we subdivide the unfolding into several parts by introducing new cuts. Again, as we try to minimize crafting time, we should minimize the number of these cuts.

First, we detect all overlaps in the plane using an efficient line intersection algorithm. The result is a set of pairs of faces that overlap. As every face corresponds to a vertex in the folding tree, there is exactly one simple path in the tree between two overlapping faces. Figure 3 shows the paths between all overlapping faces in our example. If all these paths are cut at least once, then all overlaps are eliminated. This problem can be expressed as a *minimum set cover problem*. Let $T = (V_\mathrm{d}, E'_\mathrm{d})$ (where $E'_\mathrm{d} \subseteq E_\mathrm{d}$) be the folding tree and $\mathbf{P} = \{P_1, \ldots, P_n\}$ be the set of all paths $P_i \subseteq E'_\mathrm{d}$ between overlapping faces. In addition, let $c : E'_\mathrm{d} \longrightarrow \mathbb{R}^+$ be a cost function that assigns a cost to every edge of the folding tree.

Definition 2 (Minimum set cover). *The* minimum set cover S *of a set of paths* $\mathbf{P} = \{P_1, \ldots, P_n\}$ *is a set* $S \subseteq P_1 \cup \ldots \cup P_n$ *with minimum cost that covers*

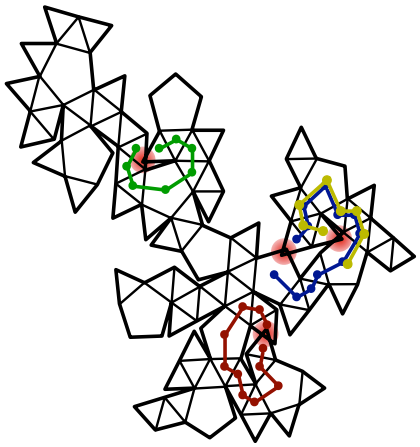


Fig. 3. An unfolding with four overlapping pairs of faces and corresponding paths in the folding tree.

all paths in $\mathbf{P}$*, i. e., S contains an edge of every P_i:*

$$S := \underset{\substack{S' \subseteq P_1 \cup \ldots \cup P_n \\ \forall i=1,\ldots,n:\, S' \cap P_i \neq \emptyset}}{\operatorname{argmin}} \sum_{e \in S'} c(e)\,.$$

The minimum set cover problem is NP-hard, but fortunately there are some approximation algorithms that are sufficiently efficient in practice. We implemented the greedy set cover algorithm from [19, p. 16] given in Algorithm 1. If all costs $c(e)$ are equal to some constant γ, a minimum set cover gives the minimum number of cuts.

Algorithm 1 Greedy set cover algorithm.

$S := \emptyset$ {set of new cut edges}
$\mathbf{C} := \emptyset$ {set of already covered paths}
while $\mathbf{C} \neq \mathbf{P}$ **do**
  {Determine the edge with minimum average cost at which it covers new elements.}
  $e := \underset{e' \in P_1 \cup \ldots \cup P_n}{\operatorname{argmin}} \dfrac{c(e')}{|\{P \in \mathbf{P} \,|\, e' \in P\} \backslash \mathbf{C}|}$
  $S := S \cup \{e\}$
  $\mathbf{C} := \mathbf{C} \cup \{P \in \mathbf{P} \,|\, e \in P\}$
end while
return S

In Section 2.1, we defined weights indicating a cutting priority. We use these weights also to define the cost function $c(e)$ whose meaning is opposite. Hence,

we set

$$c(e) := (1 - \gamma) \cdot (1 - w(e)) + \gamma \,,$$

where $\gamma \in [0, 1]$ controls the importance of the weights $w(e)$ compared to the number of additional cuts.

2.3 Packing on Paper Sheets

After unfolding and removing remaining overlaps, there might still be some parts that are too large to fit on a paper sheet. To determine the size of each part, the minimum area bounding box—which is not necessarily axis aligned—of each part is computed with the *rotating calipers algorithm* [18]. Each bounding box that is too large, is iteratively subdivided along its x- or y-axis until it fits on a paper sheet. In each step, the axis that does not fit is selected. Then the path in the dual graph between two faces with extremal vertex positions, with respect to this axis, is computed. The edge in this path intersecting the centerline of the bounding box perpendicular to the axis determines where a new cut is introduced (cf. Fig. 4).

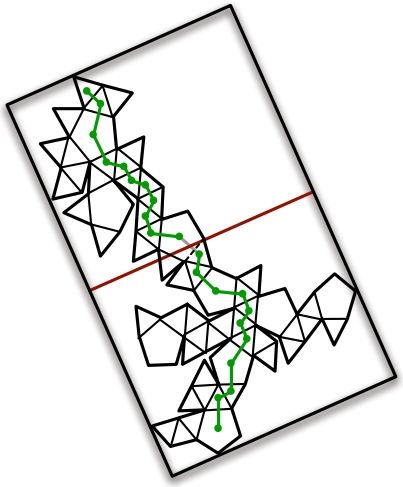


Fig. 4. Subdividing a part that is too large to fit on a sheet of paper.

To pack all parts on paper sheets, we only consider the bounding boxes of the parts and not the parts themselves. The problem of packing the parts onto as few paper sheets as possible is a so called *2D bin packing problem*. Even if the parts have a simple geometry—in our case we have rectangular bounding boxes—this problem is known to be NP-hard. Although exact algorithms for solving small bin packing problems in a reasonable amount of time exist [10], we apply the approximative 2D bin packing algorithm used in [8] for packing

textures on a texture atlas, since the latter algorithm is easier to implement. In addition, the exact packing algorithm only finds an optimal solution for given orientations of the bounding boxes and not for all possible orientations.

2.4 Computing Glue Tabs

So far, we did not consider glue tabs which need to be added to every edge being cut. The computation of the glue tabs is done in parallel to the previous steps of our algorithm. While the tabs should have a minimum size to guarantee the stability of the glue joints, they should not be too large since large tabs would use more paper and cause overlaps. Since our tabs have the usual trapezoidal shape (cf. Fig. 5), the geometry of a tab is described by the two base angles α and β and by its width d. If the width is too large for the base angles, the

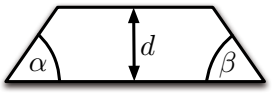


Fig. 5. A trapezoidal glue tab with base angles α and β and width d.

tab will become triangular. The range of tab sizes can be specified by the user through a global minimum and maximum base angle and width.

The tab of each cut edge has two possible positions. To formalize this, we assign to every potential tab a boolean variable, which is true exactly if we realize this tab. Consequently, the two variables x_i and x_j associated with a cut edge satisfy $x_i \not\leftrightarrow x_j$.

The geometry of the 3D mesh and of the unfolded planar mesh implies further constraints on the tabs. For any two potential tabs that overlap in 2D or 3D, we require that their variables x_i and x_j satisfy the clause $\overline{x_i \wedge x_j}$ (cf. Fig. 6). Further, for any potential tab that overlaps a face in 2D or a wrong one in 3D, we require that its variable x_i satisfies the clause $\overline{x_i}$.

The conjunction of all these clauses gives us the *arrangement formula*. We are looking for an unfolding with a satisfiable arrangement formula. First, we determine all edges e with a tab/face conflict in 3D on both sides and set their weights $w(e)$ to zero. This minimizes the chances that these edges are cut. If however, such an edge is cut, there exists a cycle in the dual graph with zero weight. In this case, it is impossible to find an unfolding with a satisfiable glue tab arrangement. We inform the user in such a case and recommend to reduce the (minimum) tab size. Second, we compute an unfolding and only check the clauses $x_i \not\leftrightarrow x_j$ and the clauses corresponding to 3D conflicts. If these are not satisfiable, the user is also asked to reduce the tab size. Third, we try to satisfy further clauses associated with 2D conflicts and remove overlaps by Algorithm 1. This algorithm introduces further cut edges and therefore possibly further conflicts. Whenever we encounter such a conflict, the second or next best edge is chosen in

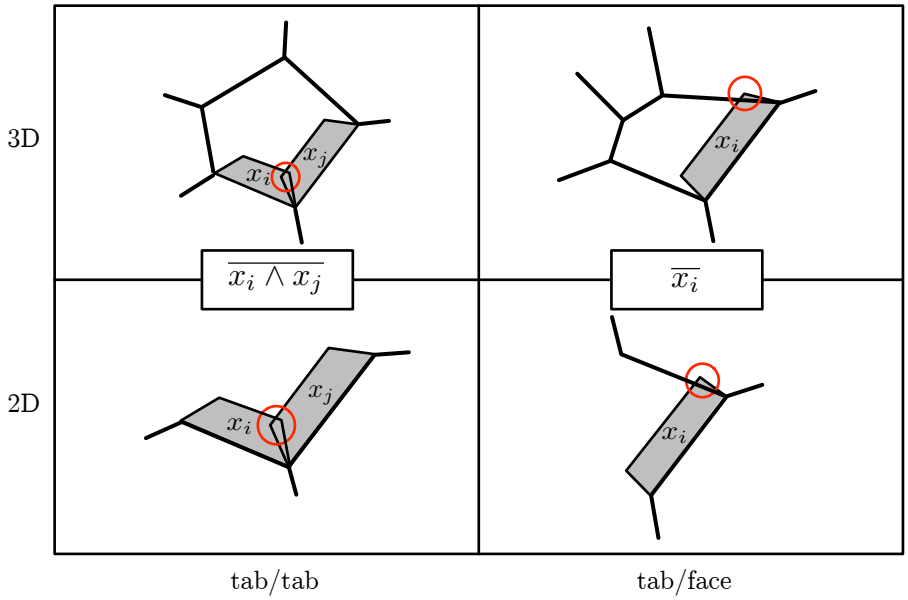


Fig. 6. Glue tab conflicts.

Algorithm 1. If this does not remedy the conflict, the user is informed to change the weights used for unfolding or to reduce the tab size. If conflicts occur due to the subdivision of the parts, then other edges near the ideal edge in the middle are cut. In the case that no edge can be cut, then again the user has to choose smaller minimum tab sizes.

The tab arrangement formula has at most two literals in each clause and can be easily converted to *conjunctive normal form (CNF)* by substituting

$$\begin{aligned} x_i \not\leftrightarrow x_j \quad &\text{by} \quad (x_i \vee x_j) \wedge (\overline{x_i} \vee \overline{x_j}) \,, \\ \overline{x_i \wedge x_j} \quad &\text{by} \quad \overline{x_i} \vee \overline{x_j} \,, \text{ and} \\ \overline{x_i} \quad &\text{by} \quad \overline{x_i} \vee \overline{x_i} \,. \end{aligned}$$

With the help of an *implication graph*, both the satisfiability of a 2CNF formula can be decided and—if the formula is satisfiable—a fulfilling assignment can be computed in linear time [2].

Definition 3 (Implication graph). *Let $F = c_1 \wedge \ldots \wedge c_n$ be a formula in 2CNF. Then the* implication graph *$G_F = (V, E)$ of F is a directed graph that has two vertices x and $\overline{x}$ for each variable x in F, and two edges $(\overline{x_i}, y_i)$ and*

$(\overline{y_i}, x_i)$ for each clause $c_i = x_i \vee y_i$:

$$\begin{aligned} V &:= \{x_i \,|\, i = 1, \ldots, n\} \cup \{y_i \,|\, i = 1, \ldots, n\} \\ &\quad \cup \{\overline{x_i} \,|\, i = 1, \ldots, n\} \cup \{\overline{y_i} \,|\, i = 1, \ldots, n\}\,, \\ E &:= \{(\overline{x_i}, y_i) \,|\, x_i \vee y_i = c_i,\ i = 1, \ldots, n\} \\ &\quad \cup \{(\overline{y_i}, x_i) \,|\, x_i \vee y_i = c_i,\ i = 1, \ldots, n\}\,. \end{aligned}$$

The two edges $(\overline{x_i}, y_i)$ and $(\overline{y_i}, x_i)$ represent the equivalent implications $\overline{x_i} \to y_i$ and $\overline{y_i} \to x_i$, respectively.

It can be easily proven that F is satisfiable if and only if for each variable x in F the vertices x and $\overline{x}$ are in different strongly connected components of G_F. In our algorithm, the tab assignment formula is represented by the transitive hull of the implication graph since this does not change the formula because implications are transitive. Each time a new clause is added, the corresponding two edges are inserted and the transitivity of the graph is reestablished. With this method the satisfiability of the formula can easily be checked. The formula is satisfiable if and only if there is no edge from a literal to the negated literal in the transitive hull of the implication graph.

By processing the strongly connected components of the implication graph in reverse topological order and assigning truth values, a fulfilling assignment can be computed in linear time. As there are several assignments fulfilling the formula, in general, we can add additional constraints to reduce crafting time. For example, it is better if all glue tabs along adjacent cut edges are on the same side. This can be expressed by adding equivalence clauses $x_i \leftrightarrow x_j$ to the tab arrangement formula. However, by adding equivalence clauses for all cuts, the formula often becomes unsatisfiable. As a consequence, we use a MAX-2-SAT algorithm [17] to guarantee that all original clauses and as many added equivalence clauses as possible are fulfilled.

This gives a valid assignment with tabs of a user defined minimum size. In a further and last step of our glue tab arrangement algorithm, we iteratively grow all tabs until they reach the maximum specified size, or touch another tab or face in the original or in the unfolded mesh (cf. Fig. 7). In each iteration, the area of

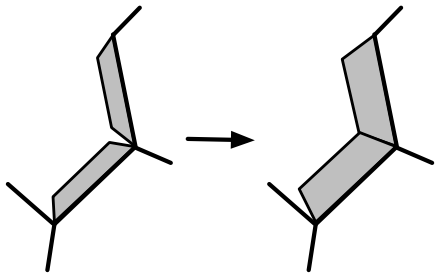


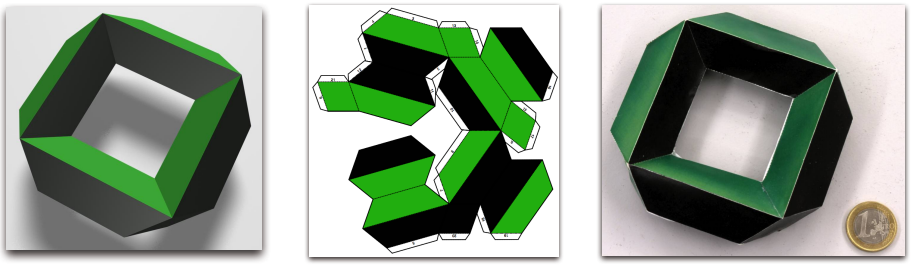
Fig. 7. Post-optimization of the size of the glue tabs.

the tab with maximum possible base angles without overlaps is compared to the area with maximum possible width. The greater area determines whether the tab grows its base angles or its width in the current iteration. The tab is allowed to grow only up to a certain percentage—the *growth factor*—of its maximum possible size since it would otherwise prevent neighboring tabs from growing. In the final iterations the growth factor should be incrementally increased to 100 % in order to close remaining gaps between tabs and faces.

3 Results

We tested our algorithm on several models. The cut-out sheets for the following models were computed within a few seconds. This is very short compared to the time it takes to craft the model. If a cutting machine was used, crafting time could be halved.

Figure 8 shows a 3D version of our former university logo, which only consists of 20 faces. The resulting cut-out sheet (in Fig. 8b) is written to a PDF file. The numbers on the glue tabs correspond to the same numbers printed on the back near the edge inside the corresponding face. Note that, although the model is not convex, a non-overlapping unfolding is produced. After cutting, bending, and gluing the model together, we obtain the paper model shown in Fig. 8c.



(a) (b) (c)

Fig. 8. The logo of the Universität Karlsruhe in 3D.

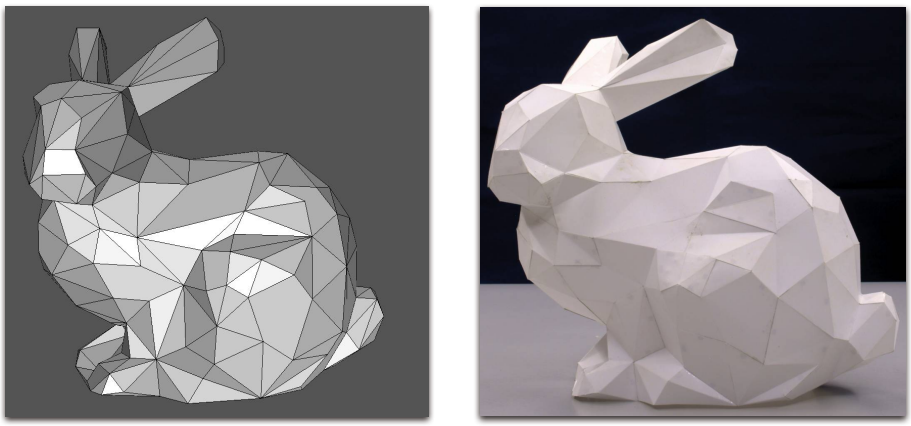
Another textured model with 62 polygons is the space ship in Fig. 9a. The time taken to craft the model in Fig. 9b, which includes the time to cut-out, bend, and glue the model together, is dependent on the experience and skills of the user. It took us about two hours to craft this model.

A more complex model is the model of the Stanford bunny in Fig. 10. The original Stanford bunny was simplified to 348 polygons by successively applying an edge collapse and a region growing algorithm. The paper model in Fig. 10b was crafted in about 12 hours.

We also collaborated with architects and civil engineers to apply our method to models of buildings. Figure 11 shows a model of our university library building.

(a) (b)

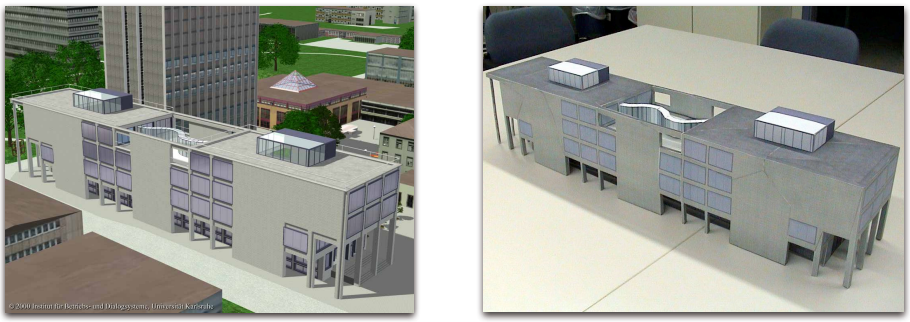
Fig. 9. A textured, low polygon space ship (62 polygons, 2 h crafting time).



(a) (b)

Fig. 10. The Stanford bunny (348 polygons, 12 h crafting time).

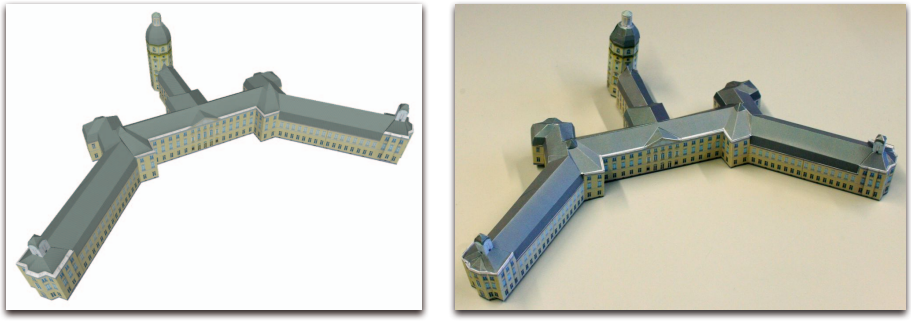
Several large polygons of the library model had to be subdivided to fit into A4 paper sheets. Although the model has nearly the same number of polygons as the Stanford bunny model, the crafting time (25 hours) is roughly doubled. This is due to the fact that the library model is much larger than the bunny model. Another building, namely Karlsruhe Palace, is shown in Fig. 12. We simplified the original model of the palace, which we got from the City of Karlsruhe, to 254 polygons and created the paper model in Fig. 12b.



(a) Rendered model, courtesy of Matthias Baas.

(b) Paper model.

Fig. 11. The new building of the KIT Library South (347 polygons, 25 h crafting time).



(a) Rendered model, courtesy of Thomas Hauenstein (real estate office of the City of Karlsruhe).

(b) Paper model.

Fig. 12. Karlsruhe Palace (254 polygons, 12 h crafting time).

4 Conclusion and Future Work

We presented an algorithm that automatically creates optimized cut-out sheets for paper models from polygonal meshes. Our method can be divided into four basic steps, where the last step is done in parallel with the first three steps: computing an initial unfolding, removing remaining overlaps, fitting and packing the parts onto paper sheets, and computing glue tabs. Due to the complex nature of the combinatorial optimization problem, only a few parts of the entire optimization problem were solved optimally. For the rest, we used heuristic approximation algorithms which may not yield a globally optimal result because the result of each step will influence all possible results in the next step.

For the future, one could improve our heuristics by evaluating the properties of the results statistically. One could also extend our method to include calculation of the ideal assembling order and sequential numbering of the glue tabs as the assembling order of the paper model has a great impact on the crafting time. An unsuitable order would make it more difficult to assemble the model. In addition, as our method is only applicable for coarse polygon meshes with relatively few vertices, we are currently exploring mesh simplification methods that remove geometric details that cannot be crafted. The details can be rendered into textures and projected onto the coarse mesh. These methods should also conserve possible symmetry properties of the object and remove inner parts and self-penetrations.

5 Acknowledgements

We would like to thank Oliver Römer and Benjamin Bertram for implementing the algorithm and for their helpful ideas. We would also like to thank Hao Li and Yuanshan Lee for proof-reading this paper.

References

1. Agarwal, P.K., Aronov, B., O'Rourke, J., Schevon, C.A.: Star unfolding of a polytope with applications. SIAM Journal on Computing 26(6), 1689–1713 (1997)
2. Aspvall, B., Plass, M.F., Tarjan, R.E.: A linear-time algorithm for testing the truth of certain quantified boolean formulas. Information Processing Letters 8(3), 121–123 (1979)
3. Bern, M., Demaine, E.D., Eppstein, D., Kuo, E., Mantler, A., Snoeyink, J.: Ununfoldable polyhedra with convex faces. Computational Geometry: Theory and Applications 24(2), 51–62 (2003)
4. Bertram, B.: Optimierte Bastelbögen aus Netzen. Studienarbeit, Universität Karlsruhe (TH) (Sep 2006), `http://geom.ibds.kit.edu/pages/Research/DiplomaTheses/OptimierteBastelboegenAusNetzen2.html`
5. Biedl, T., Demaine, E., Demaine, M., Lubiw, A., Overmars, M., O'Rourke, J., Robbins, S., Whitesides, S.: Unfolding some classes of orthogonal polyhedra. In: Soss, M. (ed.) Proceedings of the 10th Canadian Conference on Computational Geometry. pp. 70–71. School of Computer Science, McGill University, Montréal, Québec, Canada (1998)

6. Botsch, M., Steinberg, S., Bischoff, S., Kobbelt, L.: Openmesh – a generic and efficient polygon mesh data structure. In: OpenSG Symposium (2002)
7. Croft, H.T., Falconer, K.J., Guy, R.K.: Unsolved problems in geometry. Springer (1991)
8. Igarashi, T., Cosgrove, D.: Adaptive unwrapping for interactive texture painting. In: SI3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics. pp. 209–216. ACM Press, New York, NY, USA (2001)
9. Lang, R.J.: Origami Design Secrets – Mathematical Methods for an Ancient Art. A K Peters, Ltd. (2003)
10. Martello, S., Pisinger, D., Vigo, D.: The three-dimensional bin packing problem. Operations Research 48(2), 256–267 (2000)
11. Mitani, J., Suzuki, H.: Making papercraft toys from meshes using strip-based approximate unfolding. ACM Transactions on Graphics, Special Issue: Proceedings of the 2004 SIGGRAPH Conference 23(3), 259–263 (Aug 2004)
12. O'Rourke, J.: Folding and unfolding in computational geometry. In: JCDCG '98: Revised Papers from the Japanese Conference on Discrete and Computational Geometry. pp. 258–266. Springer-Verlag, London, UK (2000)
13. Polthier, K.: Imaging maths—unfolding polyhedra. Plus Magazine (Nov 2003), http://plus.maths.org/issue27/features/mathart/
14. Read, R.C., Tarjan, R.E.: Bounds on backtrack algorithms for listing cycles, paths, and spanning trees. Networks 5, 237–252 (1975)
15. Römer, O.: Optimierte Bastelbögen aus Netzen. Studienarbeit, Universität Karlsruhe (TH) (Jan 2003), http://geom.ibds.kit.edu/pages/Research/DiplomaTheses/OptimierteBastelboegenAusNetzen.html
16. Schlickenrieder, W.: Nets of Polyhedra. Diplomarbeit, Technische Universität Berlin (Jun 1997)
17. Shen, H., Zhang, H.: Improving exact algorithms for MAX-2-SAT. Annals of Mathematics and Artificial Intelligence 44(4), 419–436 (2005)
18. Toussaint, G.T.: Solving geometric problems with the rotating calipers. In: Proceedings of IEEE MELECON '83. pp. A10.02/1–4 (May 1983)
19. Vazirani, V.V.: Approximation Algorithms. Springer-Verlag (2001)