Computer aided design for Origamic Architecture models with polygonal representation

Jun Mitani and Hiromasa Suzuki
Department of Precision Engineering, The University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8656, Japan
mitani@cim.pe.u-tokyo.ac.jp

Abstract

An Origamic Architecture (OA) is a folded sheet of perforated paper from which a three-dimensional structure "pops up" when it is opened. It is similar to a "pop-up story book", but its unique feature is that it is made purely by cutting a single piece of paper. Because of this limitation, designing an OA requires considerable experience. We propose a computerised method which assists design of OAs. An OA is modelled using a set of planar polygons. This model must satisfy the conditions required of a valid, realisable OA. A unique point of our method is the application of boolean set operations to the polygons on the unfolded pattern to guarantee that the model can be made from a single sheet of paper. We also present a procedure for checking the model's validity. Additionally, we propose methods for creating openings, for generating unfolded patterns, and for displaying folding animation. We have implemented a system based on these methods and demonstrated its usefulness for creating OA. Our system allows designers to intuitively design OA models and to easily generate the unfolded patterns.

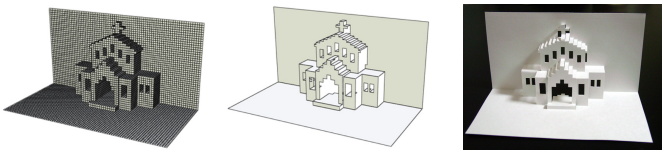
1 Introduction

An *Origamic Architecture* (OA) is a folded sheet of perforated paper from which a three-dimensional structure "pops up" when it is opened. This is one of various types of pop-up card named by Chatani[1]. Though OAs are very similar to pop-up cards in general in that 3D figures pop up when they are opened, OAs are made with single sheet of paper without any additional parts pasted on. By contrast, pop-up cards in general include such pasted-on parts. Another difference is that with many pop-up cards the three-dimensional structure pops up when the card is opened by 180°; OAs are opened by 90°. Books such as[2][4] provide a good introduction to such OAs; it is these which our paper deals.

Because of the limitation that the paper may be cut but not added to, designing OAs requires a great deal of experience. Traditionally they have been designed by a process of trial and error. We propose a new method in which a computer aids the design process.

Methods of handling OAs in computers have not been well studied, and we could find few related works. Chatani first demonstrated CG animation of OAs [3], but the data are pre-calculated and hard-coded in the program. Some works study the design of pop-up cards with computer[5][6][9]; these are all for the 180° types, and these techniques cannot be adapted to the design of the OAs we aim for.

We have previously proposed a method for designing OAs using a voxel data structure[7]. This enables interactive design of OAs and easy generation of the unfolded pattern by making use of the characteristics of voxel representation and OA(Fig.1). However, the characteristics of voxel representation only allow design of OAs in which all edges are parallel to one of coordinate axes.



(a) (b) (c)

Figure 1. OA designed using voxel representation

(a) Voxel representation. (b) CG image of the OA.
(c) Photo of the real OA.

In this paper, we propose a new method which uses a set of planar polygons to make it possible to design a wider variety of OAs. Section 2 reviews basic theory of OA. Section 3 describes our proposed method in datail. Section 4 shows

our results. Section 5 outlines ideas fro future work.

Fig.2 illustrates various terms used in this paper. The OA shown in Fig.2 cannot be designed using our previous voxel-representation approach, but can be designed using our new method.

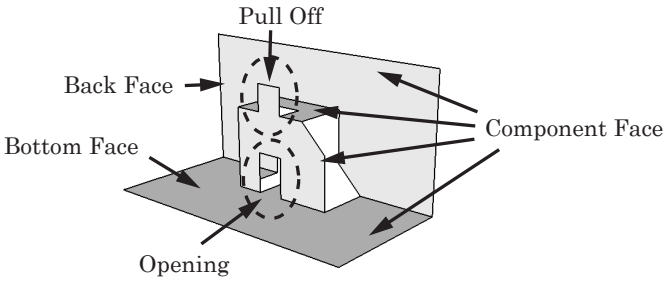


Figure 2. Definition of terms

2 Basic Theory of OA

2.1 Vertical and horizontal faces

The faces of the OA models we aim for will be either vertical or horizontal faces when it is opened by 90°. We term these faces *VFace* and *HFace* respectively, as in Fig.3.

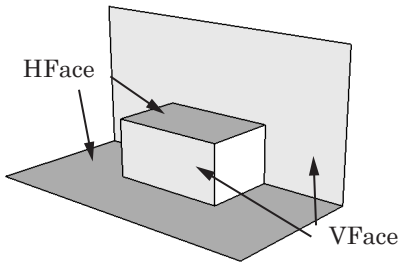


Figure 3. OA: VFace and HFace.

2.2 Model Coordinate(3D) and Pattern Coordinate(2D)

We define the coordinate systems for 3D structure and 2D pattern as "Model Coordinate" and "Pattern Coordinate" respectively, as shown in Fig.4. Since target OA is made from a single sheet of paper without any other pieces of paper pasted on, there is a one-to-one mapping between 3D structure and 2D pattern. Given a distance t_v between any VFace and the *Back Face* and a distance t_h between HFace and the *Bottom Face*, we can convert between the two coordinate systems as follows.

[Model(3D) → Pattern(2D)] conversion

$$\begin{cases} x_{(2D)} = x_{(3D)} \\ y_{(2D)} = z_{(3D)} - y_{(3D)} \end{cases} \quad (1)$$

[Pattern(2D) → Model(3D)] conversion

VFace:

$$\begin{cases} x_{(3D)} = x_{(2D)} \\ y_{(3D)} = t_v \\ z_{(3D)} = y_{(2D)} + t_v \end{cases} \quad (2)$$

HFace:

$$\begin{cases} x_{(3D)} = x_{(2D)} \\ y_{(3D)} = -y_{(2D)} + t_h \\ z_{(3D)} = t_h \end{cases} \quad (3)$$

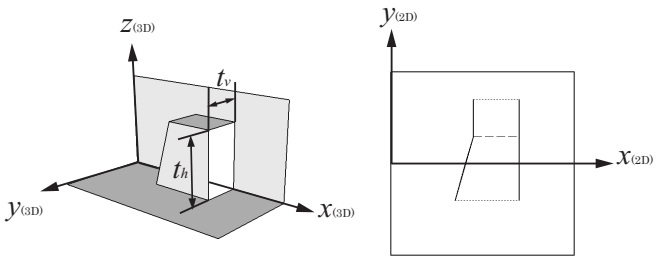


Figure 4. Model Coordinate(3D) and Pattern Coordinate(2D)

2.3 Data structure

When we consider how to represent an OA as a data structure, we note that each component face must be reconstructed from the data. Because of the one-to-one mapping between Model Coordinate values and Pattern Coordinate values using the equations 1 and 2, it is sufficient to store either one of these. Since the boolean operations we shall use later to create faces are applied to pattern coordinates, it is more convenient to store the Pattern Coordinate values.

Thus, the data for one component face must contain the Pattern Coordinate values of the 2D vertices which define the outline of the face, a flag which defines the type of the face (HFace or VFace), and value of t ($t = t_v$ when the face is VFace, $t = t_h$ when the face is HFace).

The data structures OAFace, which defines a component face, and OrigamicArchitecture, which defines the whole OA as a set of OAFaces, are defined as pseudo-C++ in Fig.5.

```
enum faceType {VFACE, HFACE};

class Polygon2D {
    list<Point2D> points;
};

class OAFace {
    double value_of_t;
    faceType type_of_this_face;
    Polygon2D polygon_on_unfolded_pattern;
};

class OrigamicArchitecture {
    list<OAFace> faces;
};
```

Figure 5. Definition of the data structure

2.4 Condition for pattern generation

As the OA is made from single sheet of paper, the polygons of component faces (OAFace.polygon2D in Fig.5) must not overlap in the pattern. Together, these non-overlapping polygons cover the entire sheet.

These requirements are expressed in equation 4, where F_i is the polygon of the ith component face, S is the whole area of the sheet of paper, and the OA consists of n faces. The OAs we aim for must satisfy this *Condition for pattern generation*.

$$F_i \cap^* F_j = \phi$$
$$F_1 \cup^* F_2 \cup^* \ldots \cup^* F_n = S \quad (4)$$

$\cap^*$ and $\cup^*$ mean normalised boolean product and summation respectively[8].

2.5 Condition for Pop-Up

Even though an OA which satisfies the *Condition for pattern generation* can always be made from a single sheet of paper, it may not "pop up" when it is opened. For example, the OAs in Fig.7 are invalid as they do not "pop up".

The OA in Fig.7a has a part floating in mid-air. The upper and lower faces of the OA in Fig.7b are separated, and as a result they are not pulled up when the Back Face and Bottom Face are opened. In order to "pop up", the OA must satisfy a *Condition for Pop-Up*. The algorithm for determining whether an OA satisfies this condition or not is discussed in section 3.6.

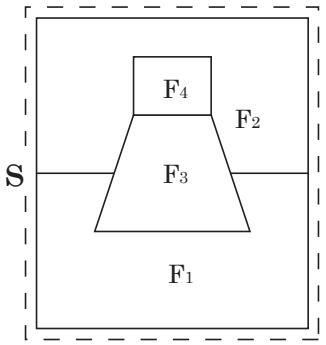


Figure 6. Polygons that construct OA pattern

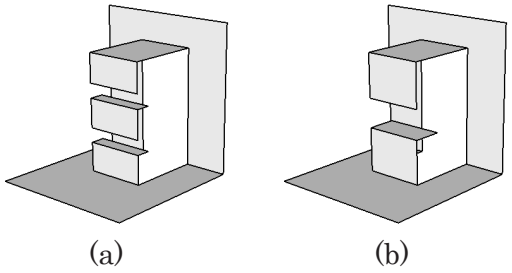


Figure 7. Invalid cases

3 Computer Aided Design for OA

In the previous section, we discussed the conditions the OA must satisfy and how data is stored. In this section, we propose an interactive computerised method to aid design of OAs so that users can easily design OAs which satisfy the conditions.

3.1 User interface

Traditionally, OAs have been designed by a process of trial and error which continues until they are move correctly and represent the desired structures. The designer had to design 2D patterns for OAs by hand, guessing how the 3D structure would appear when they are opened. To address this problem, we propose an interactive interface which enables users to design 3D figures intuitively by viewing 3DCG images on a computer display.

The OA, comprising vertical and horizontal component faces, can be designed if we can place these faces as we wish in 3D space. The steps of the interface we propose are as follows:

1. Initialise an OA to the simplest possible figure (this has only a Back Face and a Bottom Face) (Fig.8a).

2. Move the vertical (horizontal) edit plane up and down (back and forth) to determine where the user desires to place a new component face (Fig.8b shows the vertical case).

3. Input the outline of the new component face by selecting grid points on the edit plane defined in step 2 (Fig.8c).

4. Generate a new OAFace and then display the CG image of the OA including the newly-added OAFace.

5. Repeat steps 2 to 4 until the desired OA is achieved.

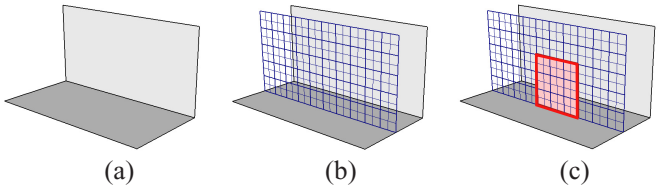


Figure 8. Interface for OA design.

3.2 Creation of faces

Using the interface we proposed in the previous section, we can construct OAs interactively. However, as it is not easy for the user to bear in mind the *Condition for pattern generation* defined in section 2.4, the system should automatically modify the whole OA pattern so that this condition remains satisfied when the user adds a new component face. Hiding this process in the system allows the user to concentrate on the design itself.

To achieve this, the system performs the following operations between steps 3 and 4 of the interface proposed previously.

1. Generate a new OAFace and add the OAFace to the list in OrigamicArchitecture. OAFace.Polygon2D is a Pattern Coordinate polygon which is calculated using equation 1 from the 3D component face input by the user.

2. Update the Polygon2D of previous OAFace'(i) by applying the following equation, where i is the index of OAFaces and $-^*$ in the equation means normalised boolean subtraction[8].

$$\text{OAFace}'(i).\text{Polygon2D} = \text{OAFace}'(i).\text{Polygon2D} -^* \text{OAFace.Polygon2D} \quad (5)$$

3. Calculate the Model Coordinate values from the Pattern Coordinate of OAFace and all OAFace'(i) using equation 2 and 3, and then display them using 3DCG.

If the *Condition for pattern generation* is satisfied before a new component face is to be added, the above operation ensures that the condition is also satisfied after the face has been added. The initial pattern (only Back and Bottom faces) satisfies this condition, so the patterns generated by a sequence of face-adding operations also satisfy the condition. With this method, the shape of the newly-added face is not changed but the shapes of previous faces may, allowing intuitive use as this is what the user will expect.

3.3 Creation of openings

With the interface we proposed in the section 3.1, OA patterns can be designed by adding faces incrementally. Another operation which must be considered is creating an opening (see Fig.2).

By using the face-create operation which we have already described, we can produce the same result as would be produced by an opening-create operation which subtracts a defined polygon from the existed polygons. For example, in order to create an opening such as the square area in Fig.9a and create the structure shown in Fig.9c, we can create a horizontal rectangular component face as shown in Fig.9b. Thus, face creation is dual-purpose as it can also be used for opening creation.

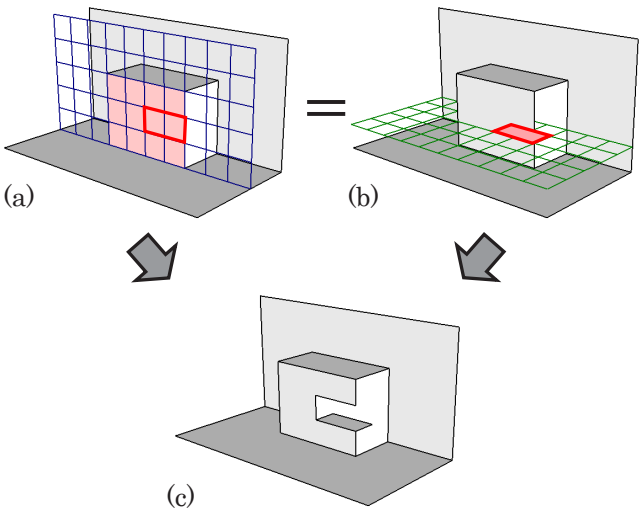


Figure 9. Creation of Opening

Since we can generate an opening by the corresponding face-create operation, implementation is easy: we can use the same algorithm to perform both operations. However,

it is not intuitive for users to create faces when they want to create openings, so this conversion of operations should be done automatically by the system and hidden from the user. The system converts a polygon representing the shape of the opening, as input by the user, to the polygon which represents the shape of the corresponding new face.

We designate the functions of creating a vertical and a horizontal opening CreateVOpening and CreateHOpening respectively, and the functions of creating a vertical and a horizontal component face CreateVFace and CreateHFace respectively. Each of these functions requires as its input an array of 3D points P[] which represent the outline of a polygon.

Now we translate CreateVOpening and CreateHOpening into CreateHFace and CreateVFace respectively by translating the positions of each point in P[] to the appropriate points P'[] as equation 6.

$$\begin{aligned} \mathsf{CreateVHole}(P_v[]) &= \mathsf{CreateHFace}(P'_v[]) \\ \mathsf{CreateHHole}(P_h[]) &= \mathsf{CreateVFace}(P'_h[]) \end{aligned} \tag{6}$$

Each point in the converted point arrays P'_v[] and P'_h[] is calculated using equation 7, where the coordinate value of ith point of P_v[] and P_h[] are $(x_v[i], y_v[i], z_v[i])$*and*$(x_h[i], y_h[i], z_h[i])$.

$$\begin{aligned} P'_v[i] &= (x_v[i], y_v[i] - z_v[i] + minZ, minZ) \\ P'_h[i] &= (x_h[i], minY, z_h[i] - y_h[i] + minY) \end{aligned} \tag{7}$$

Each point in P_v[] lies on a vertical polygon, so $y_v[i]$ are constant, and each point in P_h[] lies on a horizontal polygon, so $z_h[i]$ are constant. $minZ$ is the minimum z-coordinate value in P_v[], and $minY$ is the minimum y-coordinate value in P_h[].

By means of position conversion (equation 7) and function conversion (equation 6), we enable users to input polygons wherever they want in order to create an opening; the system can generate the opening using the same algorithm as face creation.

3.4 Unfolded pattern generation

As described above, the OA data is stored as a set of Pattern Coordinate positions of each component face. As a result, the pattern can be easily generated simply by outputting the outlines of each face, using Pattern Coordinate values, on a sheet of paper. We can aid the user further by varying the style of lines according to their type (e.g. solid line for cut, dotted line for ridge break, dashed line for valley and so on), and to do this lines must be classified by type. Since opening the OA rotates part of the pattern about a line parallel with the Pattern Coordinate x-axis, any line which is not parallel to the x-axis cannot unfold and must therefore be a cut line. By contrast, a line parallel to the x-axis can be ridge, valley or cut, and requires classification. This classification can be achieved by comparing the relative positions in the Model Coordinate of two faces which share a part or whole edge in the pattern. If the faces do not share an edge in the Model Coordinates, the line is a cut line. If the faces share a convex line, the line is a ridge; otherwise, the line is a valley. In cases where the neighbour faces share a part of edge, such as shown in Fig.10a, we divide the lines at the end points of the shared part (the black points in Fig.10b), and then classify the line segments as before.

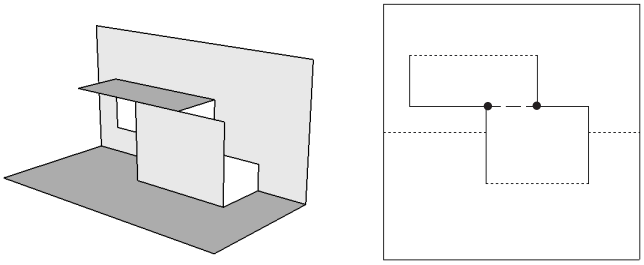


Figure 10. Line segments sharing a part each other

3.5 Folding and unfolding simulation with CG animation

The appearance of an OA at any point in the process of folding and opening can be displayed by transforming the coordinate value (x, y, z) of each point on the component faces using the equation eq:animation, changing the parameter θ as required. Fig.11 is an example of animation by changing the value of θ.

$$\begin{cases} X &= x \\ Y &= y - z\cos\theta \qquad (0 \le \theta \le 180^\circ) \\ Z &= z\sin\theta \end{cases} \tag{8}$$

3.6 Judgment for Condition for Pop-Up

In this section, we propose a method for testing whether or not an OA satisfies the *Condition for Pop-Up* defined in section 2.5. This test can be performed by checking the connectivity of Pattern Coordinate polygons of component faces, using the following algorithm.

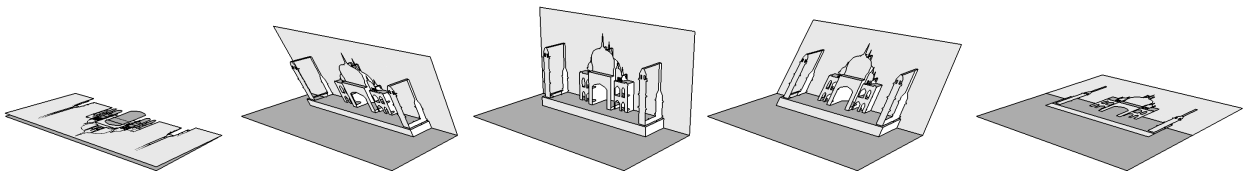


Figure 11. Folding and unfolding animation

1. (a) Initialise a set of faces to contain only the Back Face. Continue adding to this set any HFace which is joined by a valley line to a VFace already in the set, and any VFace which is joined by a ridge line to an HFace which is already in the set, until no more faces can be added (we use a recursive algorithm for this).

   (b) Initialise a set of faces to contain only the Bottom Face. Continue adding to this set any HFace which is joined by a ridge line to a VFace already in the set, and any VFace which is joined by a valley line to an HFace which is already in the set, until no more faces can be added (we use a recursive algorithm for this).

2. The OA satisfies the *Condition for Pop-Up* if and only if all component faces are in both sets or joined by a ridge or valley line to a face that is in the set.

Fig.12 shows examples of applying the above. The left pattern is the connectivity determined using rule (1a) and the right pattern is the connectivity determined using rule (1b). OAs (a) and (b) are found not to satisfy the *Condition for Pop-Up* because they have component faces (marked with [×]) which do not meet rule 2.

By including this test, our system allows users to verify that the designed OA can "pop up" correctly before printing out the pattern and cutting the lines.

4 Results

We have implemented our method on a PC and used this implementation to design some OAs. Fig.13 shows two examples. Example (1) is a test case which has a *pull off* and an *opening*. Example (2) is an attempt to design the Taj Mahal. In both Figs, (a) is the CG image, (b) is the pattern and (c) is a photograph of the final structure. We found the interactive design approach intuitive to use; it took a few minutes to design pattern (1) and about half an hour to design pattern (2).

5 Future Work

We have presented a new method for interactive design of OAs in which the OA is represented as a set of polygons. The issue of how the rounded faces may be designed remains unresolved. Curves may be generated by approximating them to polylines, and stored as control points, but

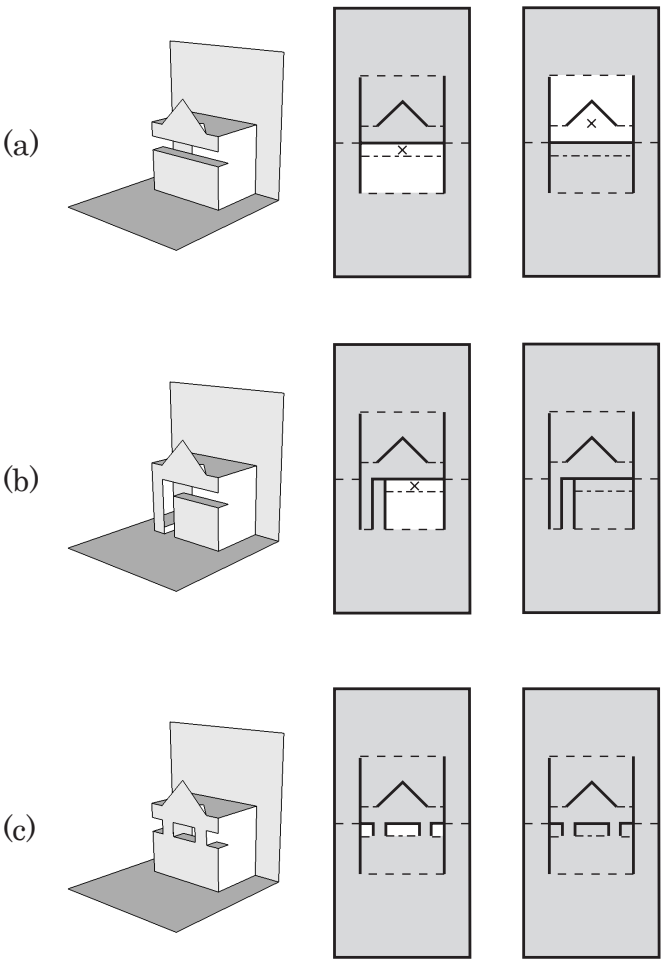


Figure 12. Judgment for the Condition for Pop-Up

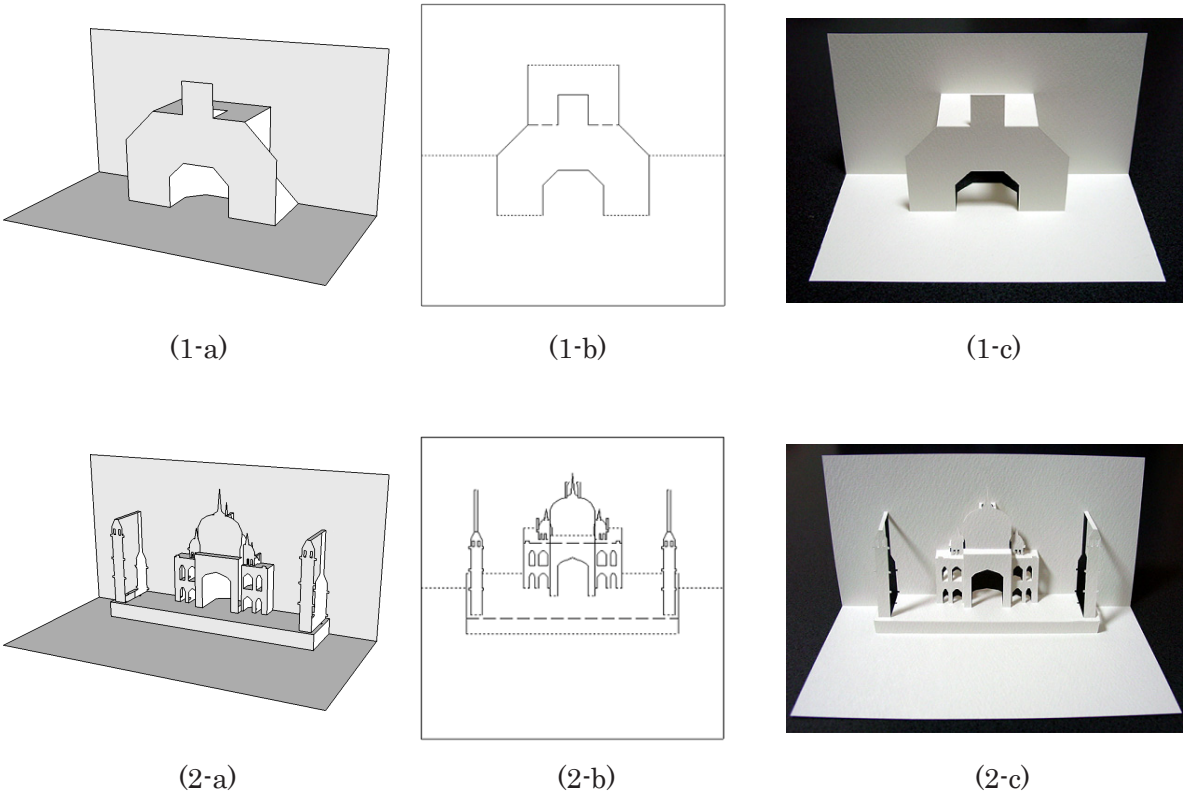


Figure 13. Examples (a) CG image (b) Unfolded pattern (c) Photo

for editing purposes a more flexible data representation is needed.

Acknowledgment

We would like to thank Dr. Peter Varley for his advice in the writing of this paper.

References

[1] M. Chatani. *Origamic Architecture Toranomaki*. Shokokusya, Tokyo, 1985.

[2] M. Chatani. *Origamic Architecture Patterns-2*. Shokokusya, Tokyo, 1986.

[3] M. Chatani, S. Nakamura, and N. Ando. *Practice of Origamic Architecture and Origami with Personal Computer*. Kodansya, Tokyo, 1987.

[4] M. Chatani and K. Nakazawa. *Oriagmic Architecture journey to Kyoto*. Shokokusya, Tokyo, 1994.

[5] A. Glassner. Andrew glassner's notebook: Interactive pop-up card design, part 1. *IEEE Computer Graphics and Applications*, 22(1):79–86, 2002.

[6] A. Glassner. Andrew glassner's notebook: Interactive pop-up card design, part 2. *IEEE Computer Graphics and Applications*, 22(2):74–85, 2002.

[7] J. Mitani and H. Suzuki. Computer aided design for origamic architecture models with voxel data structure. *J. IPSJ*, 44(5):1372–1379, 2003.

[8] A. Requicha. Representation of rigid solids: Theory, methods, and systems. *ACM Computing Surveys*, 12(4):437–464, 1980.

[9] Y.T.Lee, S.B.Tor, and E.L.Soo. Mathematical modelling and simulation of pop-up books. *Computers & Graphics*, 20(1):21–31, 1996.