A Rendering Method for 3D Origami Models using Face Overlapping Relations

Yohsuke Furuta, Jun Mitani, and Yukio Fukui

Graduate School of Systems and Information Engineering, University of Tsukuba,
1-1-1 Tennodai, Tsukuba, Ibaraki, 305-8573, Japan
furuta@npal.cs.tsukuba.ac.jp
{mitani,fukui}@cs.tsukuba.ac.jp

Abstract. When we construct a model of origami (a model of a folded sheet of paper) in a computer, it is usual to represent the model as a set of polygons having zero thickness. One of the typical characteristics of origami is that there are many instances where multiple faces are located on a same plane. In these cases, the usual z-buffer rendering method fails as multiple faces have the same depth. Furthermore, many penetrations between faces located close to each other frequently occur when the origami is folded almost flat. However, it is difficult to remove all such penetrations with moving or fixing the geometry of the model. In this paper, we propose a new rendering method that solves these problems one at a time by using the OR matrix. This is a matrix constructed from a crease pattern, and it represents the overlapping relation between every two faces.

Key words: computer graphics, rendering, origami

1 Introduction

Origami is an ancient Japanese tradition that is both a play activity and an art form involving the folding of a square sheet of paper. These days, origami is becoming popular all over the world. To describe the folding process or to show the folded shape of an origami, diagrams drawn by hand have been used for long time. If we could use three dimensional computer graphics (3DCG) instead of diagrams, it would help users to understand the shape of an origami more accurately, because they would be able to see the model from arbitrary angles. Hence we have developed a system that constructs a model of an origami in a computer and displays the model on a screen[1]. In this system, the origami model is represented as a set of polygons having zero thickness, as is commonly the case for 3DCG systems. One of the characteristics of origami is that there are many instances for which multiple faces are located on a same plane (or on very near planes). In these cases, multiple faces have the same depth (or almost the same depth). This causes problems, in that the usual z-buffer rendering method fails to render the model correctly. The reasons for this problem are:

(1) Most origami pieces are folded flat (we call them "flat Origami"). Even though the completed shape is not flat, there are parts where multiple faces are located on a almost same plane. In these origami models, multiple faces have the almost same depth in the z-buffer (Fig.1).
(2) Some origami models such as a Twist Fold (Fig.2) have a circle or multiple circles in the overlapping order of faces. For example, the face (A) is on (B), (B) is on (C), (C) is on (D), and (D) is on (A) in the folded shape in Figure 2. In this case, we cannot say which face is located lowest or uppermost.

Although problem (1) can be solved by using the painter's algorithm (also known as a priority fill) which draws faces in the order of face overlapping order or by adding an appropriate offset to each face according to the overlapping order, these methods cannot solve problem (2), namely that which face is located uppermost is not defined. The problems we mentioned above exist in origami pieces that are folded flat.

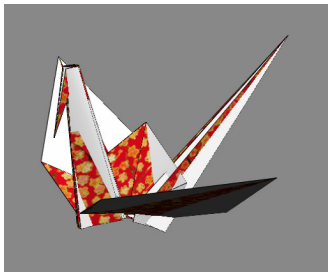


Fig. 1. An example of origami pieces, 'Crane'. As multiple faces are located on a same plane, the z-buffer method fails.

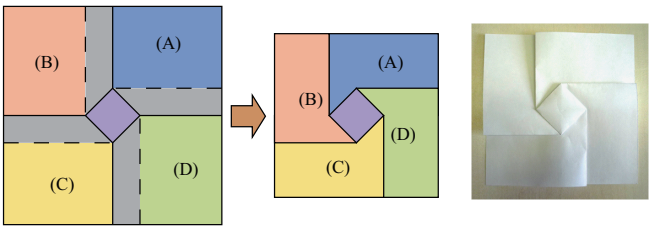


Fig. 2. The crease pattern and folded shape of a 'Twist fold'.

In the meantime, as the complete flat shape of an origami hardly looks solid and it is difficult for users to understand the configuration (Fig.3(a)), it is desirable to modify the model to have solidity by unfolding it a little (open its edge

angles a little) as shown in Fig.3(b). We call origami in this state of being almost (but not completely) flat as "nearly flat origami" hereinafter. Although the shape of nearly flat origami is helpful in understanding the corresponding configuration, many penetrations between faces commonly appear when the geometry of the flat origami model is modified (Fig.3(c)).

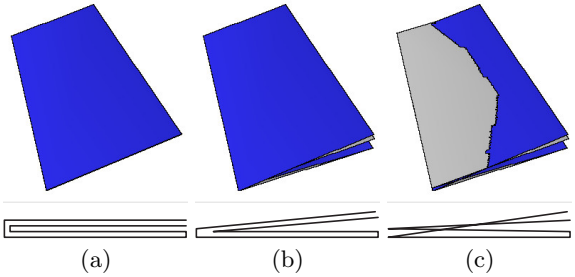


Fig. 3. (a) Flat origami. (b) Nearly flat origami. (c) Nearly flat origami with penetrations between faces.

To avoid this problem, we have to fix the geometry of the model by determining all penetrations and moving vertices correctly. This, however, becomes difficult when many faces are located very closely in a small area as is often seen in origami models. In this paper, we propose a new rendering method that solves the problems mentioned above. With our method, a flat origami model, which has a circle or multiple circles in the face overlapping order can be rendered correctly and an origami model which has many penetrations between faces can be rendered as if the penetrations are removed without modifying the geometry. The proposed method uses a matrix to represent the relative overlapping relation between every pair of faces (we call the matrix the "OR matrix" hereinafter) and the face ID buffer, which is a 2 dimensional array for storing face IDs which are located uppermost at each pixel. The OR matrix can be obtained from ORIPA[2], which is an application designed for building an origami model from its crease pattern (we introduce ORIPA in the next section). The face ID buffer has the same size as the frame buffer we use for rendering.

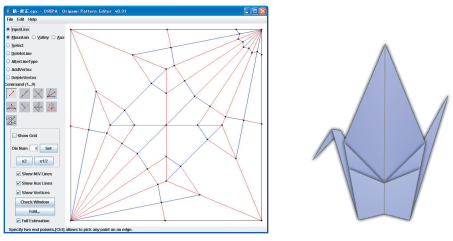
2 Related work

Recently we can find many studies about origami. In 2006, 4OSME (The Fourth International Conference on Origami in Science, Mathematics, and Education) was held in Pasadena. As the name of the conference indicates, an origami is a topic of science, mathematics, and education today. Nowadays, it is not unusual to use a computer for studying origami.

Miyazaki et al. developed a system with which a user can fold a virtual origami on a PC with a simple mouse interface [3]. The system keeps information about the face overlap order during operation and renders the origami model

using Painter's algorithm, referring to the face overlap order. Although this system works well for some simple origami, the shape that this system can fold is limited and a model that has circles in the face overlap order cannot be dealt with. Tachi developed a system "Rigid Origami Simulator" [4] that simulates the movement of folding an origami assuming that it is consists of a set of rigid polygons connected by hinges. As this system uses polygons and renders the model using the standard z-buffer method, flat origami cannot be rendered correctly. FOLIO[1] is a system that makes use of the concept in Miyazaki's system. As FOLIO uses a spring-mass model for representing the origami shape, the polygons slightly change shape during operations. By allowing slight deformations, an intuitive interface is realized and a user can easily build a complicated model. Our system uses the same interface as FOLIO for modifying an origami shape, as is mentioned in Section 3. As FOLIO also uses the standard z-buffer method for rendering, the image of the origami in which multiple faces are located on a same plane is incorrect as is shown in Fig.1. We also describe about the interface of the system in Section 3.1.

ORIPA is a dedicated editor for inputting origami crease patterns that was developed by Mitani[2] (Fig.4). ORIPA has a feature that judges whether the input crease pattern can be folded into a flat pattern. When the crease pattern can be folded into a flat pattern, ORIPA calculates the folded shape and exports the data. As our method uses the data exported by ORIPA, we will describe the details of the data here.The data output by ORIPA contains 2 dimensional coordinates, x and y, of vertices of a folded origami model. And it also contains a 2 dimensional matrix that represents the overlap relation between every pair of faces. When the number of faces of an origami model is N, the size of the matrix is $N \times N$. The element m_{ij} takes one of the 3 states as shown in Fig.5. ORIPA also has a renderer to get an origami shape [5] (Fig.4(b)). But the method is applied only to flat origami; our method can treat non-flat origami. By the way, the algorithm that applies gradations in the faces is used in our system (in Section 3.3).



(a) (b)

Fig. 4. (a) Main window of ORIPA with the crease pattern of Crane. (b) The image of the folded shape generated by ORIPA.

For example, the OR matrix of the Fig.5(a) (folded in three layers) becomes Fig.5(c). Our method refers to this OR matrix to determine the overlap relation between two faces. Although ORIPA itself has a feature to render the folded shapes of the origami, it targets only flat origami. Our method targets not only rendering the flat origami but also rendering nearly flat origami and non-flat origami in a single system.

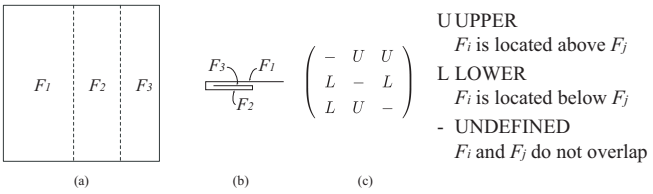


Fig. 5. (a) A simple crease pattern. (b) Side view of the folded one. (c) The OR matrix.

3 Method

The flow of our system is as follows:

(1) A user inputs crease pattern of target origami on ORIPA.
(2) System obtains the data of the folded origami which contains the coordinates of vertices and the OR matrix from ORIPA.
(3) Based on the OR matrix and the position of the user's viewpoint, the system stores the ID of the face which can be seen from the viewpoint into each pixel of the face ID buffer using the scanline algorithm.
(4) Render the origami model to the frame buffer based on the values stored in the face ID buffer.
(5) As the need arises, the user changes the view angle or changes the shape of the origami by rotating faces around the user specified edge, Then the system updates the rendering window by taking steps from 3.

In the following subsections, we describe the details of our method.

3.1 Constructing an origami model

To construct an origami model as a set of polygons, we use ORIPA. Firstly, a user input results in the crease pattern on ORIPA as shown in Fig.4(a). As we introduced in Section 2, ORIPA has a feature to calculate the folded shape of flat origami from the crease pattern. We can obtain the following data from ORIPA.

– X and Y coordinates of vertices composing polygons of folded origami shape. As we are assuming that the folded model is flat, coordinates are in 2D.

– The OR matrix that represents the overlap relations between every pair of faces.

If necessary, the user can change the shape of origami by using a mouse. We implemented the user interface in the same way as FOLIO. Firstly, the user specifies an edge as a rotational axis. Then a face rotates around the edge by a user's dragging operation (Fig.6). The associated faces also move at the same time because neighboring vertices are connected by virtual springs. However, some penetrations may occur after the operation because we do not check collisions between faces.

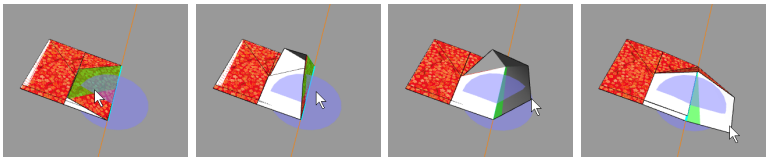


Fig. 6. User interface of modeling by our system.

3.2 Render to the face ID buffer

During the user operation, we render the model to the face ID buffer referring to the OR matrix. Firstly, we project each polygon of the origami model to the face ID buffer and scan the pixels inside the polygon using the standard scanline algorithm. If a value of the scanlined pixel is empty, the ID of the scanlined face (assume the ID is i) is stored at the pixel of the buffer. If a value (assume j) is already stored at the pixel, the m_{ij}th element of the OR matrix is referred. Then the ID ($= i$) is stored only if the value is "U"(Upper), which means that the face F_i is located upper than F_j. (When the view position is located at the backside of the origami, in other words, the origami piece is viewed from the backside, the operation mentioned above is inverted.)

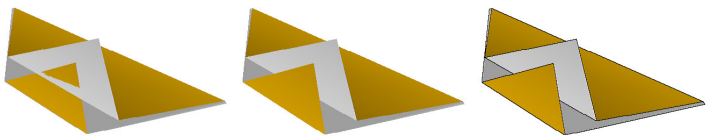
3.3 Render to the frame buffer

After adopting the operation described in the previous subsection, the face ID that should be displayed at each pixel of the frame buffer is stored in the face ID buffer. According to the face ID, we calculate the color considering the color of the face and the direction of the normal. Then we assign colors to each pixel of the frame buffer. To add shade, we multiply the pseudo brightness B' of each vertex of the polygonal faces. The value of B' is set using the following equation proposed by [5],

$$B' = (1 - w_b) + \frac{M - V + 2}{4} w_b \tag{1}$$

M and V are the number of mountains and valleys, respectively, of folded lines connected to the vertex on the contour of a face. This equation is based on a generally experienced rule; the vicinity of the valley folded lines becomes dark because little light reaches the area; on the other hand, the vicinity of the mountain folded lines becomes bright. Since the values $M-V$ of the inner vertex can take the values -2 or 2 [6], the value of B' becomes $1-w_b$ or 1. The w_b is the weight of the parameter, and is set equal to 0.4 in our system. The color of each pixel in a face is calculated by linearly interpolating the value of vertices along the contour.

The method described until now does not use depth values (z values) of faces but refers to the overlapping relation between two faces of origami models. Hence even if a face placed below penetrates a face placed above, the penetration does not affect the rendering results as shown in Fig.7. On the other hand, we have to take care when drawing contours of polygons onto the frame buffer. We cannot use the depth value and it becomes invalid to draw with polygonal lines by connecting contour vertices when a part of a line should be hidden by other faces. So we extract boundary lines from the face ID buffer using the Roberts filter. The extracted edges on the face ID buffer are translated to the frame buffer. Fig.7 shows the example of the result.



(a) (b) (c)

Fig. 7. (a) Rendered with standard z-buffer method. (b) Rendered by referring OR matrix. (c) Rendered with edges.

3.4 Utilizing both the OR matrix and the depth values of faces

In the previous section, we described a method that renders origami models with reference to the OR matrix, but not using the depth values of faces. This is based on the assumption that the model is flat or nearly flat origami. This approach does not work well for non-flat origami or an origami which contains both flat and non-flat states.

Hence we utilize both the OR matrix and the depth values of faces so that the system can render both states appropriately. To enable this, we allocate a z-buffer that stores the depth values of each pixel, the same as in the standard z-buffer method. When the system renders a polygon, the depth value is stored using the scanline algorithm. When the depth of the face is smaller than the already stored value, the stored value is updated with the smaller value and the face ID buffer is also updated with the face ID at the same time. At that time,

if the z value is close to the stored value, it means that the face already exists is located close to the new face. In this case, we do not use the z buffer but refer to the OR matrix to update the face ID buffer. Finally, the frame buffer is rendered using the face ID buffer. The threshold of difference of depth between the two faces for deciding which shall be referred to between the depth value or the OR matrix was set to about 3% of the size of a sheet of origami, which was decided, based on our heuristic.

4 Result

We implemented the three rendering methods, the z-buffer method that was implemented in software (referred to hereinafter as the "Z-method"), a method that refers to the OR matrix without using the z-buffer as described in subsection 3.1, 3.2, and 3.3 (referred to hereinafter as the "OR-method"), and the combination of both methods as described in subsection 3.4 (referred to hereinafter as the "ORZ-method") using Java. We evaluated these methods on a PC (CPU: Intel Dual-Core Xeon 2.66 GHz $\times$ 2, RAM: DDR-2 FB-DIMM 9GB, GPU: NVIDIA GeForce 7300 GT). The size of buffer we used was 2048 $\times$ 1536 pixels, and the z-buffer had 32 bit depth. Fig.8 shows the results of the rendered origami pieces using 3 different methods; the Z-method, the OR-method, and the ORZ-method respectively. We used 3 models (Simple fold, Medal, and Organ) and 2 states (nearly flat, and non-flat) for evaluation.

We can see the Z-method failed to render the nearly flat origami with a lot of noise (it is the motivation of our study) in Fig.8. On the other hand, the results using the OR-method are fine. Penetrations of nearly flat status of the Simple fold shown in the result of the Z-method are eliminated, and the result looks natural. However, the results of the non-flat origami are incorrect. The backside faces that should not be seen are rendered in front. This problem of the OR-method is solved by the ORZ-method. Although a geometrical conflict can be seen in the middle image of Organ, all other models are rendered correctly. The times required for constructiong the OR Matrices and rendering were shown in Table 1. The matrix calculation is done only once before the start of the renderings. When the origami has n faces, the size of its OR Matrix becomes $n \times n$. Although the Z-method was fastest as expected, the difference was not so great, and a model can be rendered in real-time rate with all 3 methods.

We also tested a unique origami model, "an Origami Tessellation", that is a flat origami including many circles in the face overlapping order. The crease pattern of an origami tessellation is generated from a tiled plane as shown in Fig.9. The details are described by Thomas Hull[7], and an application that generates the crease pattern for the origami tessellation is available on the Web[8]. The result of rendering this model is shown in Fig.10. Although the Z-method fails, we can see that both the OR-method and the ORZ-method render the model correctly.

Table 1. Rendering time in milliseconds (average over 30 frames).

	Simple fold	Medal	Organ
OR Matrix calculation	5	158	24
Z-method	241	472	277
OR-method	246	610	316
ORZ-method	254	646	346
number of Polygons	48	129	60

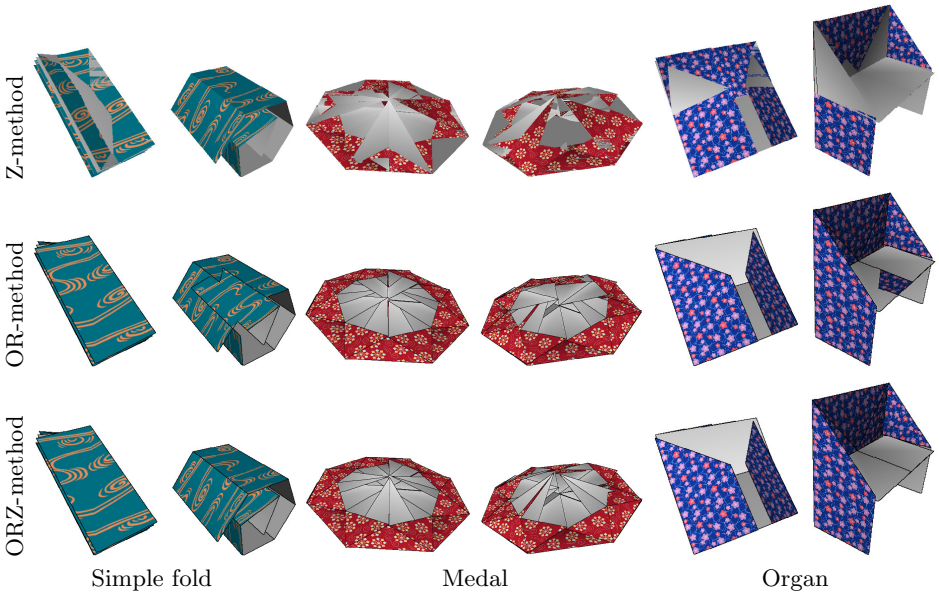


Fig. 8. Results of the methods. Tops are results with Z-method, the middles are with OR-method. The bottoms are with ORZ-method.

5 Conclusion and future work

In this paper we have proposed a new rendering method that solves the problems of rendering origami models in which multiple faces are located on a same plane or on very near planes by using the OR matrix. By referring to the OR matrix, not using the z-buffer, we can also eliminate penetrations between faces. We implemented our system on a computer and showed that our method can generate appropriate images. The limitation of our approach is that we have to firstly input the crease pattern on ORIPA. Also the crease pattern has to satisfy the condition that it is foldable into a flat configuration. Although we did not consider changes of topology of origami models during modification of the model, these would be supported if we could update the OR matrix at the time

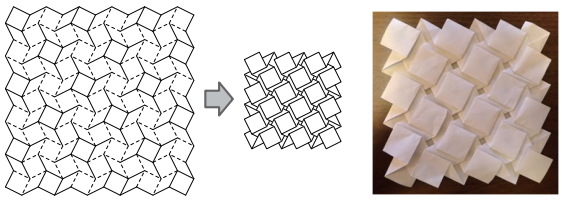


Fig. 9. The crease pattern and the photo of the origami tesselation.

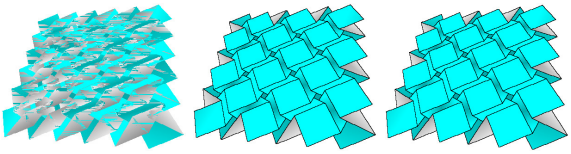


Fig. 10. Results of the origami tesselation.

of operation. As we use simple rendering algorithms in this paper, it would be desirable to render origami more photo-realistically.

References

1. Yohsuke Furuta, Haruo Kimoto, Jun Mitani, and Yukio Fukui. Computer model and mouse interface for interactive virtual origami operation. *IPSJ Journal*, Vol. 48, No. 12, pp. 3658–3669, December 2007.
2. Jun Mitani. Development of origami pattern editor (oripa) and a method for estimating a folded configuration of origami from the crease pattern. *IPSJ Journal*, Vol. 48, No. 9, pp. 3309–3317, September 2007.
3. S Miyazaki, T Yasuda, Shigeki Yokoi, and J Toriwaki. An origami playing simulator in the virtual space. *The Journal of Visualization and Computer Animation*, Vol. 7, No. 1, pp. 25–42, 1996.
4. Tomohiro Tachi. Simulation of rigid origami. September 2006. appear in proceedings of the 4OSME, Pasadena USA.
5. Jun Mitani. Rendering method for flat origami. In *Eurographics'08: Annex to the Conference Proceedings*, pp. 291–294. the 29th annual conference of the European Association for Computer Graphics, April 2008.
6. J. Justin. Towards a mathematical theory of origami. In *Proceedings of the Second International Meeting of Origami Science and Scientific Origami*, pp. 15–29, 1994.
7. Thomas Hull. *Origami 3: Third International Meeting of Orgami Science, Mathematics, and Education Sponsored by Origami USA*. A K Peters Ltd, 2002.
8. Alex Bateman. Paper mosaics. `http://www.papermosaics.co.uk/`.