國立交通大學

多媒體工程研究所

碩 士 論 文

自動產生立體紙雕之研究－以動物為題

Origamic Architecture: Automatic Animal Paper Card from 3D Models

研 究 生：蔡宜珊

指導教授：施仁忠　教授

魏德樂　教授

中 華 民 國　一 百　年 六 月

自動產生立體紙雕之研究 – 以動物為題

Origamic Architecture: Automatic Animal Paper Card from 3D Models

研 究 生：蔡宜珊　　Student：Yi-Shan Tsai

指導教授：施仁忠　　Advisor：Prof. Zen-Chung Shih

魏德樂　　Prof. Der-lor Way

國 立 交 通 大 學

多 媒 體 工 程 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Multimedia Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Multimedia Engineering

June 2011

Hsinchu, Taiwan, Republic of China

中華民國一百年六月

自動產生立體紙雕之研究－以動物為題

研究生: 蔡宜珊　　指導教授: 施仁忠教授

魏德樂教授

國立交通大學多媒體工程研究所

摘 要

紙雕卡片是一種很特別的立體卡片。我們只需要在紙上進行切割與摺疊就能在卡片攤開的時候展現出各種立體的結構，並且易於收納，因此不論是紙類藝術的新手或是專家都非常喜愛。但是製作這種紙雕卡片需要經驗的累積才能順利製作完成，對於一般沒有經驗的人來說仍然是個挑戰。而在中國傳統文化中，人們在新年時會互相餽贈新年賀卡來聯絡情感，其中賀年卡上的圖案又以十二生肖等動物居多。若是我們能夠將紙雕卡片與賀年卡上的圖案結合，將會使得賀年卡變得更加生動有趣。

因此本篇論文將利用使用者所輸入的動物模型投影至平面的影像進行切割，並產生紙雕卡片上的圖案。在製作版型時，系統會對切割完成的圖案進行排版以及連接，並且對紙雕卡片的穩定性做調校之後，輸出紙雕卡片的版型。

Origamic Architecture: Automatic Animal Paper Card from 3D Models

Student: Yi Shan Tsai

Advisor: Prof. Zen-Chung Shih

Prof. Der-Lor Way

Institute of Multimedia Engineering
National Chiao-Tung University

ABSTRACT

Origamic architecture is a paper art which creates a 3D reproduction by cut-out and folding a single sheet of paper. However, designing origamic architecture is a challenging work because of this special feature. In Chinese culture, people sends Chinese New Year's card to each other in the period of Chinese New Year, and Chinese zodiac animals are the most common topics. It will be interesting if we integrate origamic architecture and animal to paper card. In this paper, we propose a novel algorithm for helping users create an animal paper card from a 3D model, and output a layout of origamic architecture. The algorithm first do the 2D segmentation to the 3D model, and create layers of origamic architecture. After putting layers onto layout, our algorithm creates connection between layers. Finally, we check the stabilization of layers and merge un-stable layers and output the layout of origamic architecture.

Acknowledgements

First of all, I would like to express my sincere gratitude to my advisors, Prof. Zen-Chung Shih and Prof. Der-Lor Way for their guidance and patience. Without their encouragement, I would not complete this thesis. Thanks also to all the members in Computer Graphics and Virtual Reality Laboratory for their reinforcement and suggestion. I want to thank to all the people who ever supported me during these days. Finally, I will sincerely dedicate this thesis to my parents.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Motivation

Origamic architecture is a paper art which creates a pop-up card by cutting and folding a single sheet of paper. While opening the origami architecture, it's amazing that a two-dimensional pattern on the card gradually transforms to a three-dimensional object. In Chinese culture, people sends Chinese New Year's card to each other in the period of Chinese New Year. The patterns shown on the Chinese New Year's card most are Chinese zodiac animals. It will be interesting if we bind origamic architecture with various patterns of animals.

The two major topics of origamic architecture are architecture and creature. Unlike architectures, creatures are organized by smooth curvatures. It's crucial for origamic architecture that how to express the shape of creatures with limited number of curvatures and layers. In this thesis, we develop a system which transforms 3D animal models into 2D layers, and put 2D layers onto origamic architecture, and finally generate the layout of origamic architecture. By this system, people can easily design different kinds of origamic architecture by 3D animal models, and having fun from the joyful results.

Our contributions are shown as follow:

1. The proposed algorithm for generating origamic architecture follows the generating

process of origamic architecture by artist.

2. The proposed algorithm for generating layers of origamic architecture according to various features of input model.
3. The proposed method for generating connections between layers is suitable for various shapes of layers.
4. We propose a clear and simple rule for examining the stability of origamic architecture.

1.2 System Overview

Our system consists of three steps: **shape generation, layers generation**, **layout generation**. Figure 1.1 shows the flow of our proposed system.

First of all, the layers of origamic architecture are generated from a 3D model. The system will do orthogonal projection of the input model which placed by user, and then render the projected model by bi-level shading. As a result, the rendered model will be broken into pieces. Users can select pieces and merge them to form layers of origamic architecture. After computing the depth of layers using depth map, positions of layers on the layout are determined.

Then the system will construct connections between layers and form the layout of origamic architecture. The system will find the contour which located between two layers and analyze it. The result of analyzing will determine the type of connection between layers. After locating the position of layers and connections on the layout, we should do refinement to make the layout stable.

Finally, the system will refine the layout in order to make the origamic architecture stable. During the refinement process, layers can be merged, and the layout can be generated iteratively until the entire layout become stable.

After getting the layout, users can send their origamic architecture works to each other.

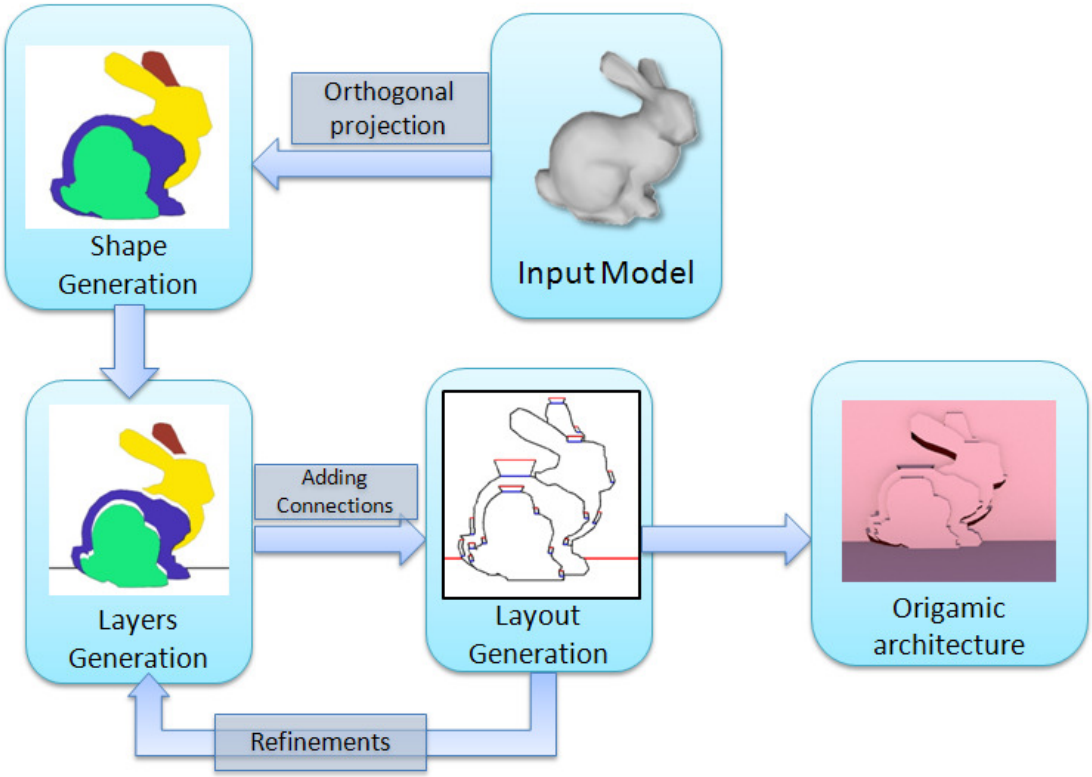


Figure 1.1: System Overview.

CHAPTER 2

Related Works

In this chapter, we review some previous works. First we focus on paper crafting and paper architecturing. Then we briefly survey the shape abstraction and non-photorealistic rendering.

2.1 Paper Crafting

Paper crafting includes several types of topics. The following are some major topics of paper crafting.

Paper cutting is a Chinese traditional art which cuts out stylistic patterns by making use the symmetry of patterns and paper overlapping. Xu [29] proposed a simple but efficient algorithm for generating paper-cutting pattern automatically. Li [19] extends the two-dimensional paper-cutting to three-dimensional paper-cuts, and generates animations with paper-cuts in interactive way.

Origami is the art which crafts a model using paper folding without damaging the paper. Hull [14] proposed a method for origami by using a piece of paper without using cutting or gluing. For the past few years, the folding algorithms and foldability of paper has been attracted extractive attention in the field of computer geometry [9]. Tachi [27] introduced an algorithm for automatically generating arbitrary polyhedral surfaces. For

curved folding, which is much more restricted than conventional origami, Kilian [16] proposed an algorithm for generating curved folding automatically based on the analysis of developable surface.

Paper modeling is to model 3D models using developable patches or strips. Mitani [24] proposed a method for modeling the surface of a model by strips. Other methods which use mesh simplification on paper modeling are Garland [11], Cohen [7], and Wei [28].

2.2 Paper Architecturing

Previous work on pop-up crafting is mostly computer–aided environment for designing pop-up crafts. Glassner [12] proposed a system which lets users design V-fold card interactively. Matini [23] introduced a computer-aided origamic architecturing system based on the concept of CAD. Although this system ensures that the output planar layout is foldable, it cannot ensure whether the output layout is stable or not.

Algorithmic solutions for automatically generating pup-up crafting have arisen in recent years. Li [20] proposed a system which automatically generates origamic architecture, and ensured the layout can erect in stable manner. Some results are shown in Figure 2.1. However, this method cannot work well for organic models which need to be retained important features while generating origamic architecture, as shown in Figure 2.2. In our work, we provide a method for generating non-architectural origamic architecture by considering the characteristics of an input object.

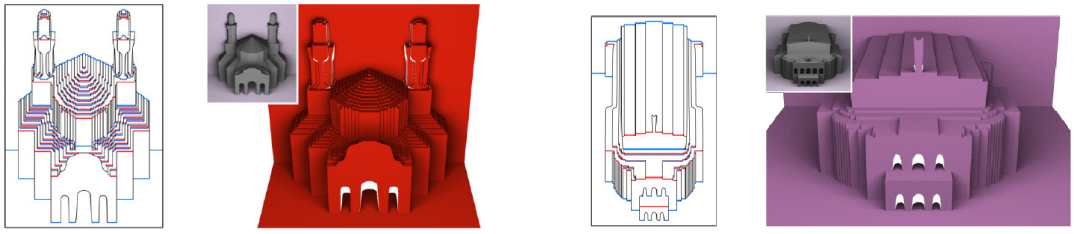


Figure 2.1: Results of architectures in automatic origamic architectures [20]

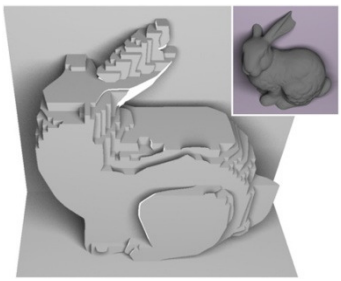


Figure 2.2: Result of non-architecture in automatic origamic architectures [20]

2.3 Shape Abstraction

There are many methods for approximating three-dimensional models. Lai [17] proposed a method based on segmentation of model surface and approximation of model by simplified patches. Kalogerakis [15] proposed the method of model segmentation based on training fashion. Mehra [21] introduced an algorithm for abstracting three-dimensional models by characteristic curves, and reconstructing the abstracted model by these curves. Eisemann [10] proposed a view-dependent method for converting 3D models into 2D layers. Our model layering approach considers not only viewer's direction but also lighting effect on the model in order to make the origamic architecture looks stereo.

2.4 Non-Photorealistic Rendering

Non-photorealistic rendering is concerned about how to extract lines which can illustrate the shape of objects. Hertzmann [13] proposed suggestive contour to illustrate the shape of model with smooth surface. Except contours, ridge lines and valley lines [25] are also the useful information to define object characteristics.

CHAPTER 3

Background

3.1 Origamic Architecture

In this section, we will introduce the development of origamic architecture and animal works of origamic architecture.

The technique of paper-making is originated by Cai Lun(蔡倫) in the late Eastern Han Dynasty of China in 114 AD. After paper spread from China to Europe in the thirteenth century, European artists began to develop sculptures of paper in the mid-eighteenth century. Using the skills of cutting, crimping, folding, and gluing, the planar paper can turn into a three-dimensional work.

Origamic architecture is developed by Masahiro Chatani in Japan. He began to develop a brand new kind of pop-up card using techniques of origami, paper folding, and kirigami, paper cutting, in 1980. There are several types of origamic architecture. The type introduced in this thesis is the type of 90 degree-opened which doesn't use the skill of gluing.

Figure 3.1: Origamic architectures creations of architectures worked by Masahiro Chatani.

The topics of origamic architecture are extensive, ranging from geometric parallel patterns or world famous buildings (shown as Figure 3.1) to animals and plants. Buildings are mostly regular structured and with clear layering. With these features, designing origamic architecture of buildings is much simpler than animals. However, the origamic architectures creations of animals can be more touching than buildings. Figure 3.2 shows the origamic architectures of animals created by Masahiro Chatani.

Figure 3.2: Origamic architectures creations of animals worked by Masahiro Chatani.

3.2 Designing Process of Origamic Architecture

In this section, we observe the process of designing origamic architecture and illustrate our concept of algorithm in constructing an origamic architecture.

Analyzing origamic architectures of Chatani [5], we summarize the following steps for designing origamic architecture for animals:

1. Decide the depths of patterns.
2. Put patterns onto the layout according to their depths and form the layers.
3. Create connections between layers.

As Li [20] defined, two outer regions that meet at the central fold called ***backdrop*** and ***ground***. Li [20] first discussed the behavior of patches that parallel to backdrop and ground in an origamic architecture, and then constructed origamic architecture by these patches to approximate shape of input model under the rule of stability. In this way, for models which are regular in shape and consist of straight lines or planes which are mutually parallel to each other such as architectures, the algorithm performs well. However, for models which are consisted of smooth curves and irregular surfaces, the algorithm of Li [20] will fail. For Li[20] only approximating the shape of model by two directions of patches under the constraint of origamic architecture. Therefore, for models such as animals whose normal direction are various on surface, the result of approximation will not be pleasant.

Therefore, we propose a concept of ***layers*** and ***connections***. We define the patterns parallel to backdrop as ***layers*** for illustrating features of models in shape, and the patterns parallel to the ground as ***connections*** between layers. When users open the origamic architecture by moving background and backdrop, the patterns will “pop-up” along the fold lines. Figure 3.3 shows layout of origamic architecture. The yellow regions are called backdrop, the blue regions are called gound. The red regions are layers, and green regions are connections between two layers.

We illustrate the features of animals by layers which are parallel to backdrop, and connect these layers for ensuring the stability of origamic architecture.

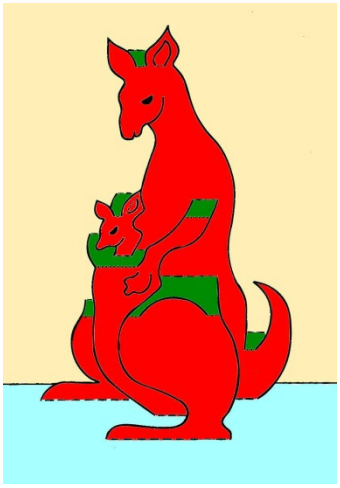


Figure 3.3: Origamic architecture layout of kangaroo [5].

In Li [20], the stability of origamic architecture is defined on patches which may have two directions. Therefore, the definition for stability will be complex and difficult to understand. In this thesis, we concentrate on relationships between layers and ground or backdrop, and we obtain a clearer and simpler rule for determine stability of layers as follow:

1. If a layer has connection with both ground and backdrop, it is stable.
2. If a layer has connection with ground or backdrop, and having connection with a stable layer, it is stable.
3. If a layer has no less than two connections with different stable layers, it is stable.

If all the layers are stable, the origamic architecture is called stable. A simple example is shown in Figure 3.4. There are two layers in this example. The former layer connects with both ground and backdrop. So it is a stable layer. The latter layer has connection with a stable layer (the former layer) and backdrop. Thus it is also a stable layer. As a result, this origamic architecture is stable because the layers in it are all stable.

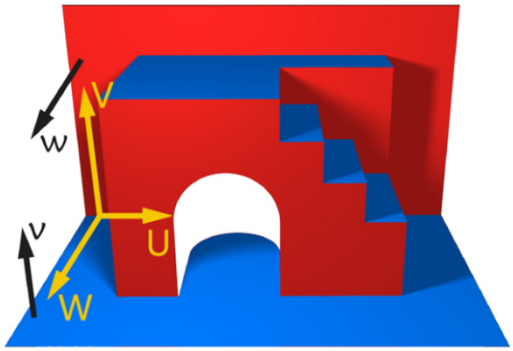


Figure 3.4: A simple example of stable origamic architecture from [20].

CHAPTER 4

Layers Generation

In this chapter, we describe how to get segmentations from the input model, and how to generate layers of origamic architecture with easy user operations. Figure 4.1 shows the flowchart of layers generation.

First, the system will do orthogonal projection of the 3D model input by user and then render the projected model by bi-level shading. User can move the light source and model at ease. After user determines the directions and positions of light source and model for generating origamic architecture, the system will detect edges of depth map and rendering result. The extracted edges will segment the image of input model into pieces, and then the system will colorize these pieces with different colors. User can merge the broken pieces by simple operations. Finally, the system will compute the depth of each merged piece, and these pieces will be outputted as layers of origamic architecture.

This chapter is organized as follows. In section 4.1, we segment the input model through the rendering results of bi-level shading and depth map. Then we describe the generation of layers in section 4.2.

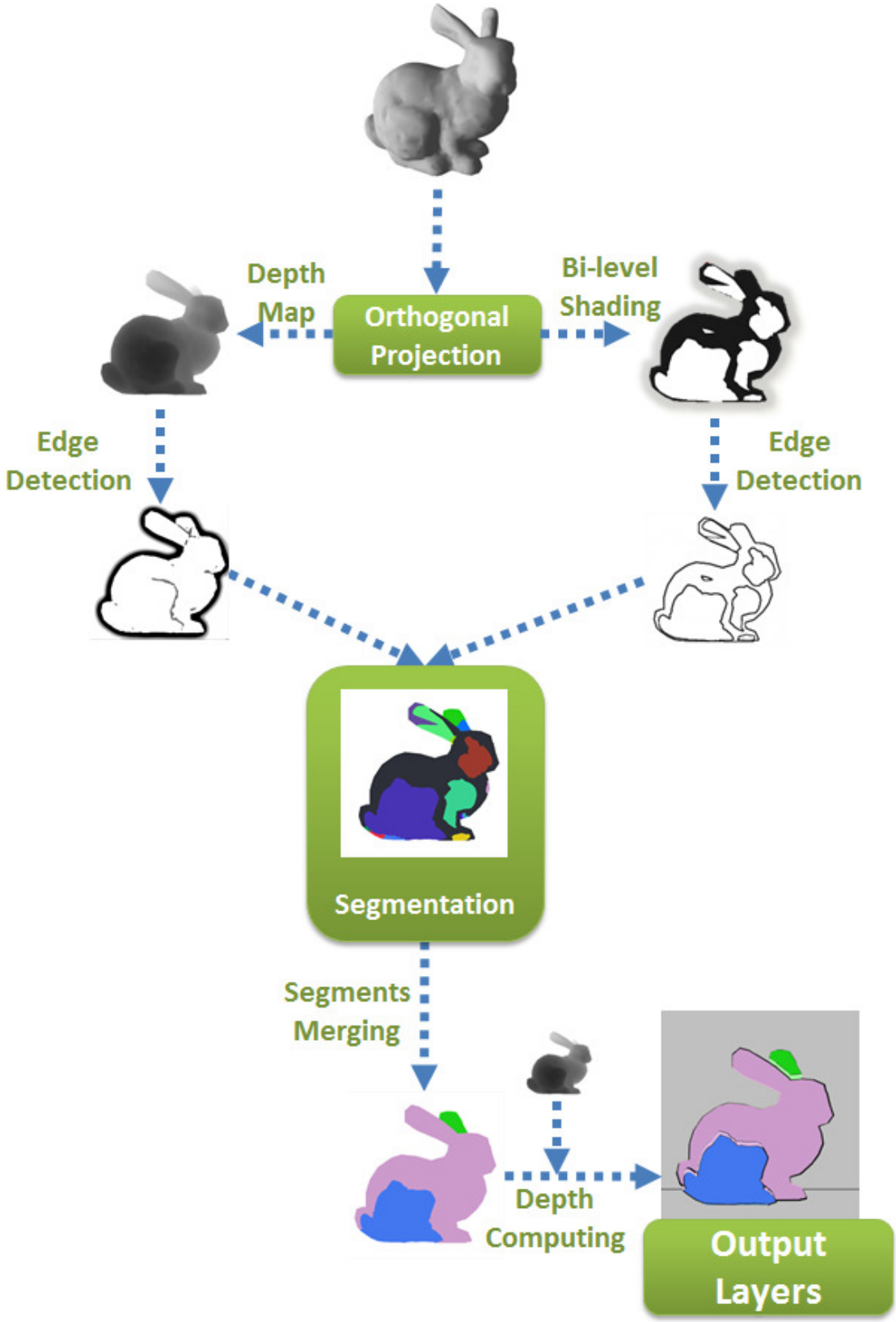


Figure 4.1: The flowchart of layers generation

4.1 Shape Generation

For generating layers of an origamic architecture, we expect that each layer shows features of the model. Our goal of model segmentation is to separate model into layers while preserving features of a model. Here we propose an image-based method for separating a 3D model.

First we do orthogonal projection of the input model, and render the model using bi-level shading. We define the color C of the model:

$$C_s = \begin{cases} 0, & if \quad n \cdot l < k \\ 1, & if \quad n \cdot l \geq k \end{cases} \tag{4.1}$$

where *n* is the normal of a point on the model surface; *l* is the direction of light; *k* is the threshold value between 0 and 1.

We use ***Canny edge detector*** [3] to detect edges of the result of bi-level shading, and separate the image of model into pieces. On the other hand, we also detect the edges of the depth map by Canny edge detector. Before applying Canny edge detector, the shading of the depth map will be changed as follows:

$$C_d' = clamp(\,\beta(\alpha - C_d), 0, 1) \tag{4.2}$$

Where C_d and C_d' are current and new magnitude of a pixel in depth map respectively, $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are parameters which change the shading of the depth map and are controlled by the user.

When detecting edges of a depth map, user can control two threshold values $\boldsymbol{h}$ and $\boldsymbol{l}$ of edge detector, as shown in Table 4.1.

Parameter	Function
h	An upper threshold. If the magnitude of a pixel is larger than this value, then it will be considered as an edge pixel.
l	A lower threshold. If the magnitude of a pixel is smaller than this value, then it will be considered as a non-edge pixel.

Table 4.1: Parameters in edge detection.

Figures 4.2 and 4.3 show examples of different parameter settings of depth map and edge detection. By different settings, the result of detected edges shows different features of the input model.

(a)Original depth map (b) $\alpha = 4.16, \beta = 0.27$ (c) $\alpha = 4.94, \beta = 0.24$

Figure 4.2: Depth maps with different parameters α and β.

(a) Original image $\alpha = 4.12, \beta = 0.23$ (b) $h = 0.14, l = 0.0$ (c) $h = 0.67, l = 0.28$

Figure 4.3: Results of Canny edge detection with different parameters *h and l.*

Then the system will separate the image of input model into segments by edges of bi-level shading and depth map, and apply erode operation to each segment in order to remove insignificant pieces. Finally, we colorize each survival segment with different color and expand each segment iteratively until segments touch each other or the border of model image.

Moreover users can choose neighboring segments arbitrarily and merge them into one segment. The final segments are imported into the process of layers generation. Figure 4.4 shows the results of model segmentation with and without merging segmentations by user.

(a) (b)

Figure 4.4: Results of model segmentation with and without merging neighboring segments. (a) Initial result of model segmentation. (b) Refined result of model segmentation by merging segments.

4.2 Layers Generation

In this section, we introduce the process of layers generation and the deletion of unreasonable layers.

4.2.1 Layers Initialization

For each segment obtained in Section 4.1, the system will calculate the depth value of each segment according to the new depth map. We define the initial depth of each segment as:

$$D_{S_i} = \frac{\sum_{x \in S_i} C'_d(x)}{N_{S_i}} \tag{4.3}$$

where S_i is the set of pixels in segment i, N_{S_i} is the number of pixels of S_i, and C'_d is the depth value of new depth map.

4.2.2 Layers Refinement

For building stable origamic architecture, the first thing we should take into account is the hierarchical structure of layers. A stable layer in origamic architecture should be supported by ground or another neighboring layer which is shallower than it at contacting points. Consider Figure 4.5. L_2 has two neighboring layers L_3 and L_4 which are shallower than L_2 at contacting points. As a result, L_3 and L_4 become supporting layers for L_1. Therefore, we construct a bottom-up hierarchical structure to ensure the stability of layers.

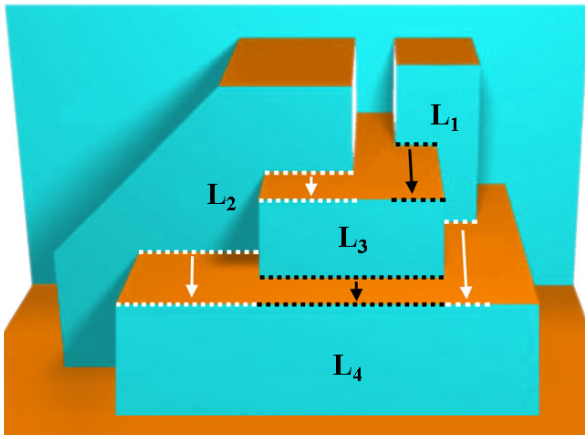


Figure 4.5: Example of origamic architecture with all layers stable.

First, we construct a directed graph which records the connections between layers where:

- A node represents a layer.
- The root of the graph is a pseudo node which points to layers that touch ground directly.
- Each edge points from node P to node Q represents that P and Q are neighboring layers and P is shallower than Q.

For the layer which does not have parent which is shallower than it, we merge it into its neighboring layer which is closer than other neighbors in depth field.

Then we determine the depth of each layer. To make origamic architecture look stereo and layered, we sort the layers by their depth value D_{L_i} and define new depth value D'_{L_i} as:

$$D'_{L_i} = D_{L_0} + \kappa \cdot i \quad , \ 0 \leq \mathrm{i} < N_L \qquad (4.4)$$

Where N_L is amount of layers, L is the sorted array of layers and $D_{L_i} < D_{L_{i+1}}$, and κ is a

constant value.

Finally, we can define relationships between layers and folding line in origamic architecture, and put layers onto the layout. The position of folding line represents the depth of backdrop. As backdrop is deeper than all layers, we define the depth of backdrop as:

$$D_{Backdrop} = D_{L_0} + \kappa \cdot N_L \tag{4.5}$$

For allocate the position of layers onto the layout, we define the following equations:

$$\begin{cases} x_{(2D)} = x_{(3D)} \\ y_{(2D)} = z_{(3D)} + y_{(3D)} \end{cases} \tag{4.6}$$

Where $z_{(3D)}$ represents the depth value of layer. Figure 4.5 shows the coordinate systems of 3D origamic architecture and 2D layout.

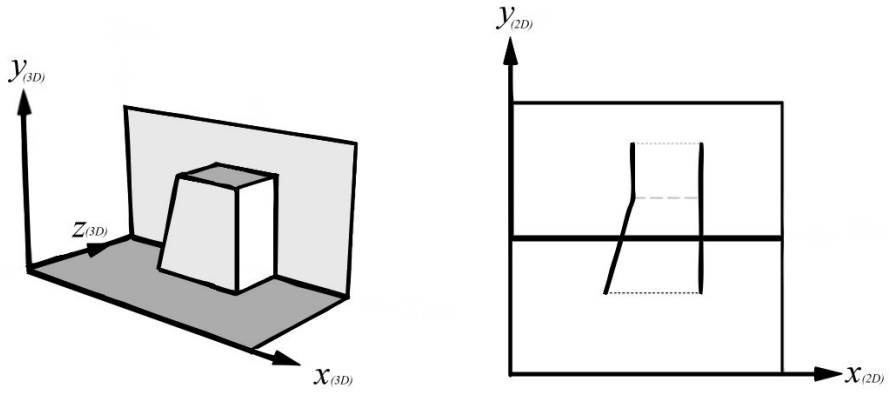
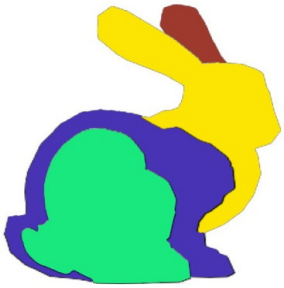


Figure 4.6: Coordinates of 3D origamic architecture and 2D layout.

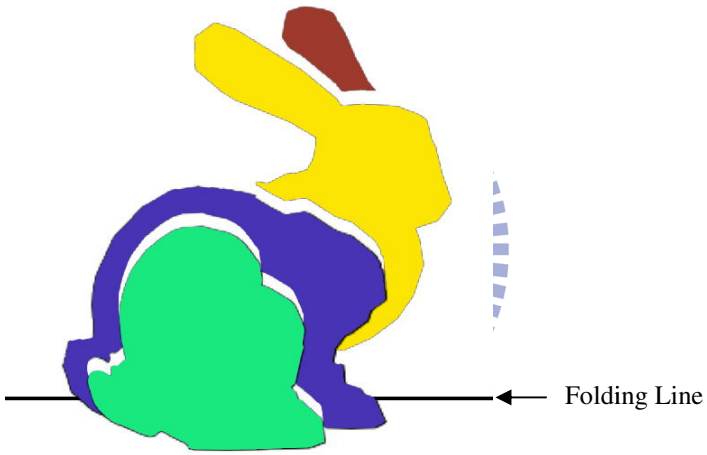
Figure 4.6 shows the result of allocating layers onto layout according to the result of model segmentation and its corresponding depth map. If one layer in the layout is overlapped by other layers and splitted into pieces, the system will cut this layer and split it

into new layers which share the same depth.



(a) Result of model segmentation.

(b) Depth map corresponding to model segmentation.



(c) Results of allocating layers onto layout of origamic architecture.

Figure 4.7: Result of allocating layers according to model segmentation and its corresponding depth map.

CHAPTER 5

Layout Generation

In this chapter, we describe process of generating connections between layers and method to ensure the stability of the origamic architecture. Figure 5.1 shows the flowchart of layout generation.

First, the system will analyze contours between layers and compute scores of two kinds of connection: horizontal connection and vertical connection. According to the result of scores, the system will pick up segments which lie on border between layers and decide how to generate connections. Then these connections will be put onto the layout of origamic architecture by taking overlap of connection into consideration. After putting all available connections onto layout, the system will test stability of origamic architecture. If the origamic architecture is stable, the layout will be output as result of layout generation. Otherwise, the system will choose an unstable layer and merge it with another layer and re-generate the layout.

The rest of this chapter is organized as follows. In Section 5.1 we introduce two types of connection used between layers, horizontal connection and vertical connection, and describe generating process of connections in Sections 5.2 and 5.3 separately. In Section 5.4 we describe how to ensure stability of origamic architecture and output layout.

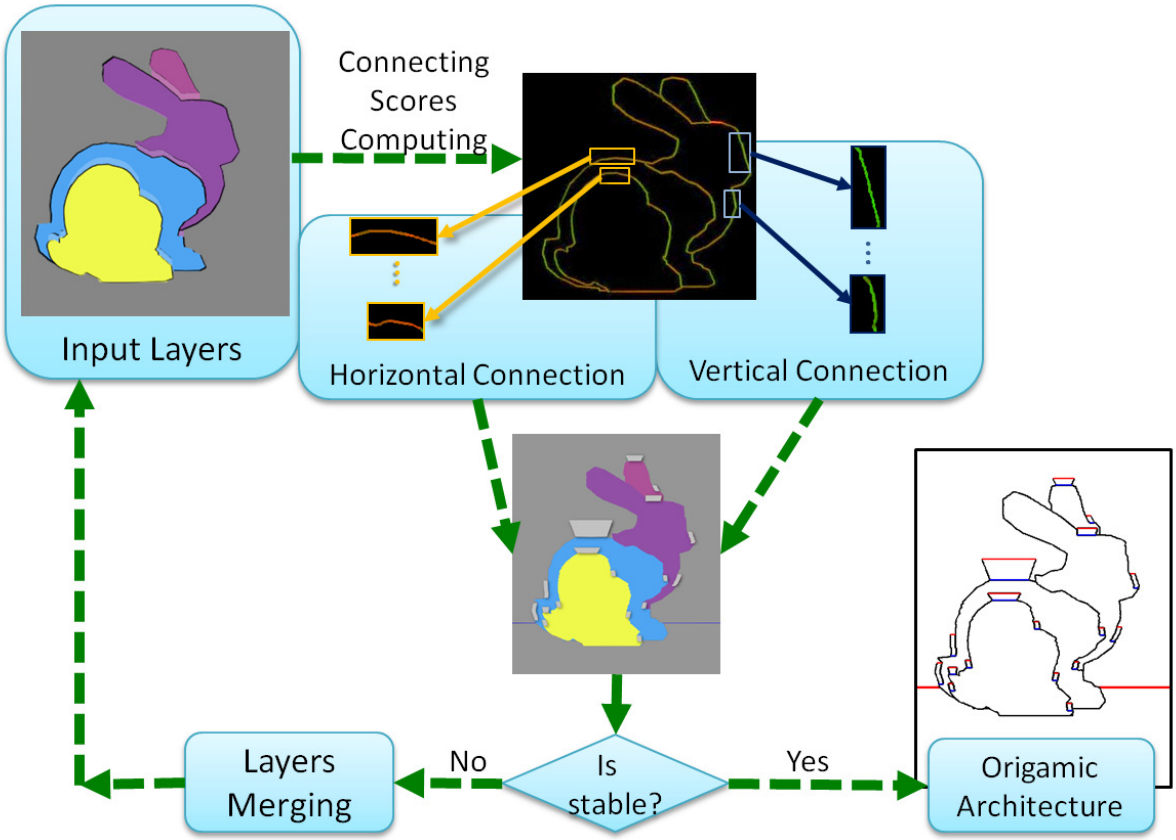


Figure 5.1: The flowchart of layout generation.

5.1 Analysis of Connections

In this section, we analyze features of connections between layers from examples of Chatani [5], and conclude with a rule for generating connections between layers.

For generating connection between two layers, first we should know the position where the connection is located at. Figure 5.2 illustrates a simple condition for connecting two layers L_1 and L_2. In Figure 5.2, L_2 and L_1 are two layers and L_2 is in front of L_1 and lower than L_1. C is a connection between L_1 and L_2 which is located at the bottom of L_1 and the top of L_2.

If we would like to locate another connection C' between L_2 and another layer which is

in back of L_2 and lower than L_2, we will find that there is no more space of origamic architecture to form connection C'. Therefore, we conclude that a layer cannot form connections, to forward or backward, at top and bottom of the layer simultaneously. As a result, for all pairs of layers in origamic architecture, we generate connections from top of the front one to bottom of the back one.

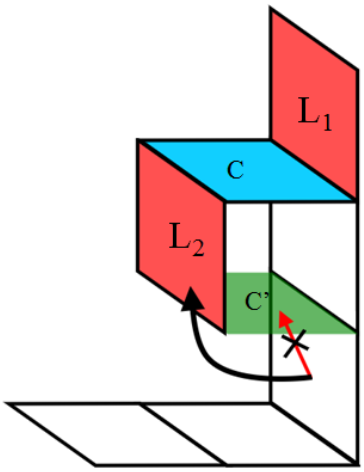


Figure 5.2: A sketch illustrates the location of connection between L_1 and L_1's supporting layer L_2.

Then we find that there are two major kinds of connection: horizontal connection and vertical connection. An horizontal connection often lies on segment of border which is near-horizontal, and a vertical connection often lies on a near-vertical segment of border. Figure 5.3 shows different examples for horizontal connection and vertical connection. Therefore, we propose a method for extracting the segments of border between layers, and classify these segments for different kinds of connections.

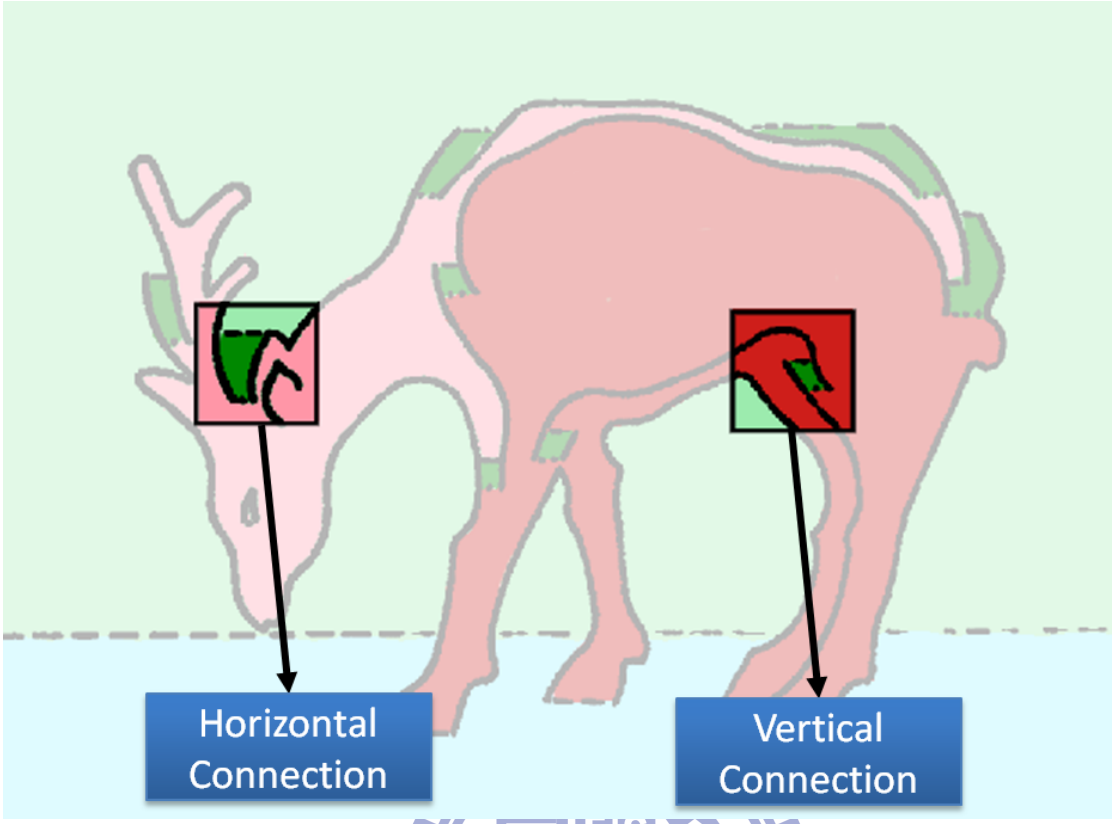


Figure 5.3: Examples for two major kinds of connections in origamic architecture [5].

First, we compute scores for horizontal connection S_H and vertical connection S_V of each point $\boldsymbol{x}$ of layer's contour as follows:

$$S_H(L_i, x) = (\frac{1}{|slope(x)|} + \delta_{11}) \times \delta_{12} \quad (5.1)$$

$$S_V(L_i, x) = (|slope(x)| + \delta_{21}) \times \delta_{22} \quad (5.2)$$

where δ_{ij} is a constant for striking a balance between S_H and S_V. Figure 5.4 shows the result of computing S_V and S_H. For segments which are greenish, the system will generate vertical connections. Otherwise, horizontal connections will be generated for reddish segments.

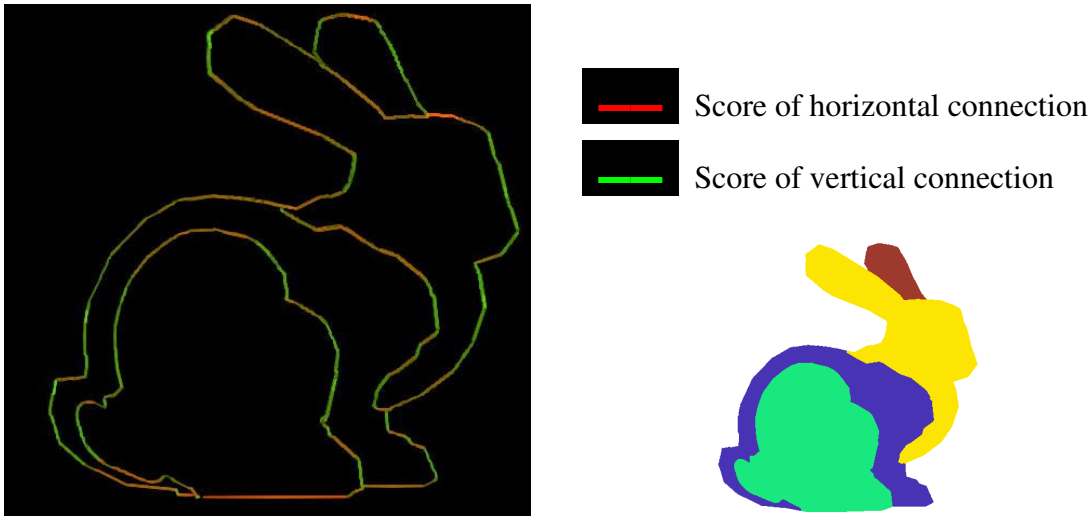


Figure 5.4: Scores of contour for horizontal connection and vertical connection of Stanford bunny model.

For each layer L, we choose a point x which has the highest score in the sequence of contour points, either S_V of S_H, as the seed of connecting segment. The segment extends for continuous points x_i if :

$$((S_H(L, x_{seed}) - S_V(L, x_{seed})) \times ((S_H(L, x_i) - S_V(L, x_i)) > 0 \qquad (5.3)$$

and $$N_L(x_i) \equiv N_L(x_{seed}) \ , \gamma_1 < i < \gamma_2 \qquad (5.4)$$

where connecting segment ranges from x_{γ_1} to x_{γ_2} and $N(x)$ indicates the neighboring layer of L at point $\boldsymbol{x}$.

After extracting a connecting segment, the scores along this segment will be set to zero. Then the system will extract next connecting segment until there is no more segment to extract. As a result, the contour of layer L will be cut into several segments. These segments are classified into two groups, horizontal connection and vertical connection, according to the higher score of segments. As a result, connections will be generated in different ways according to the groups these segments belong to.

As discussed in section 5.1, connections will be only generated at top of front layer and bottom of back layer. In this thesis we only discuss connections which generated backward. Therefore, the connecting segment which located at bottom of front layer and top of back layer will be neglected. While generating connections for a pair of layers, we define the front one as the source layer, the back one as the target layer, and difference of depths between source layers as $|\boldsymbol{D}|$. Moreover, holes on the layout between source layer and target layer will be filled with the color of target layer to avoid fragments of the layout.

5.2 Horizontal Connection Generation

In this section, we introduce how to generate horizontal connection between layers. Figure 5.5 shows the process of generating horizontal connection.

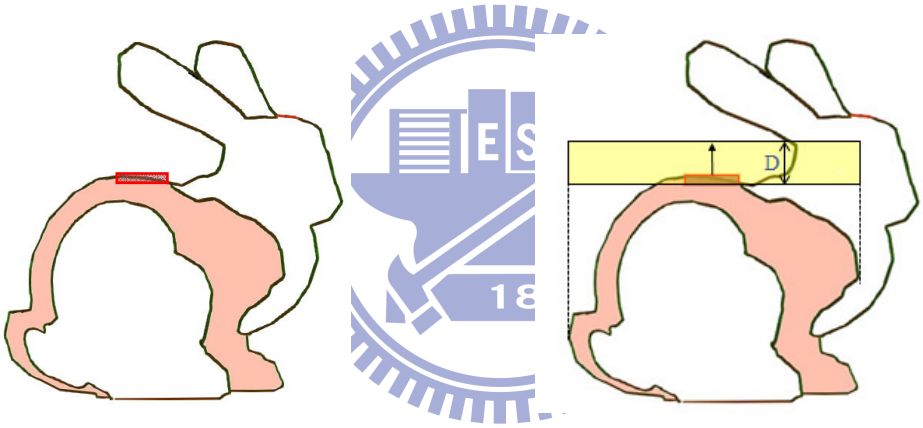
Given a horizontal connecting segment extracted from Section 5.1, the system will first find a sub-segment $\boldsymbol{S_i}$ which has the widest axis-aligned bounding box $\boldsymbol{B_{S_i}}$ with its height $\leq \frac{2}{3}\boldsymbol{\kappa}$. Then the system will create an examining area $\boldsymbol{E}$ with its bottom aligned with $\boldsymbol{B_{S_i}}$'s bottom, and set its height as $|\boldsymbol{D}|$ and width as the width of the layer, as shown in Figure 5.5 (b) .

As a result, the system can find a segment at the bottom of $\boldsymbol{E}$ which includes the bottom of $\boldsymbol{B_{S_i}}$ and has intersection with the source layer, as shown in Figure 5.5(c). For this segment, called ***connecting base***, the system generates a connection C_{S_i} with the top wider than connecting base, as shown in Figure 5.5(c), and defines the score of erosion Δ for it:

$$\Delta = \frac{N(C_{S_i} \cap L)}{\ell} \tag{5.5}$$

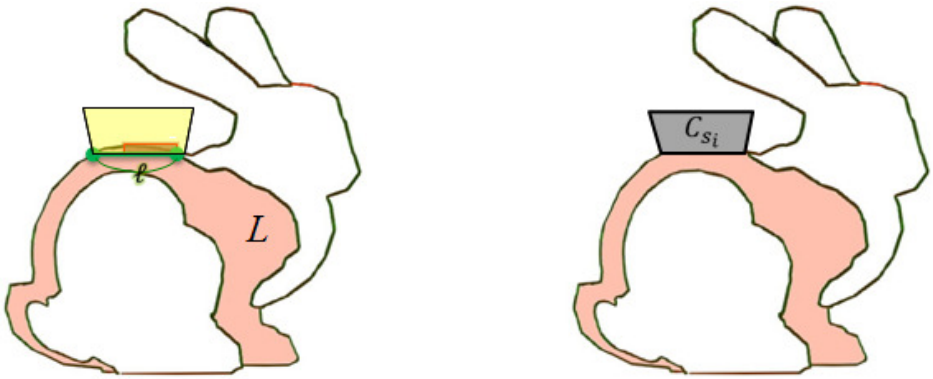
where ℓ means length of connecting base, and *N(x)* means the number of pixels of x.

If the score is smaller than a threshold value $\boldsymbol{\varepsilon}$, then this connection is generated. Otherwise, for reducing the score of erosion, $\boldsymbol{B_{S_i}}$ will be moved upward slightly as $\boldsymbol{B_{S_i}}'$ and the system will regenerate the connection until $\boldsymbol{B_{S_i}}'$ has no intersection with $\boldsymbol{B_{S_i}}$, the system will abandon this connecting segment.



(a) Widest sub-segment S_i and its bounding box.

(b) Creation of examining area.



(c) Segment (green) for generating horizontal connection.

(d) Construction of horizontal connection.

Figure 5.5: Generating process of a horizontal connection.

After generating a connection, the system will examine the layout and eliminate the connection according to the following rules:

1. The connection should not overlap with any layer which does not belong to source layer or target layer.
2. If target layer will be broken into pieces after generating connection, the connection will be eroded horizontally, as shown in Figure 5.6, to make the target layer continuous.

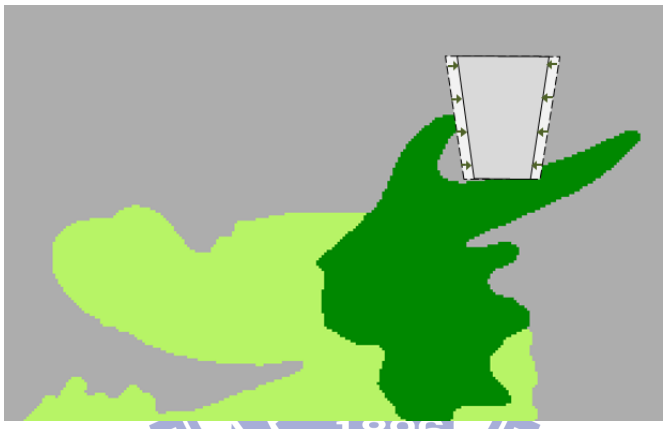


Figure 5.6: When target layer (gray) is broken by the connection, the connection will be eroded horizontally to make the target layer continuous.

If the eroded connection touches the source layer and target layer without broken, then the connection will be retained. Otherwise, the connection will be abandoned.

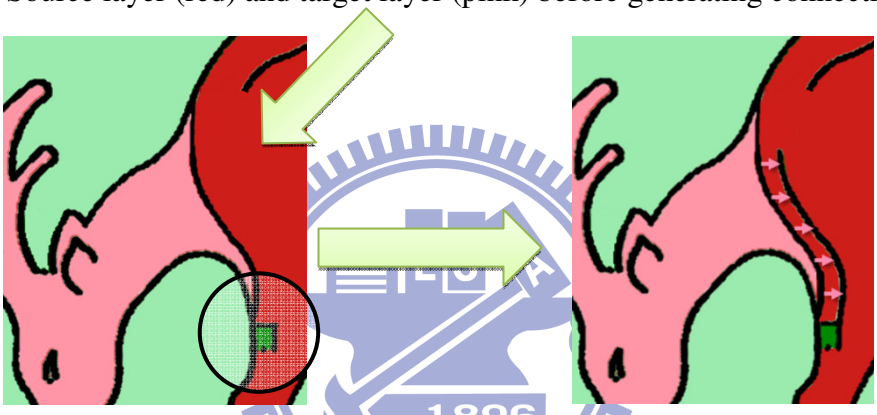
5.3 Vertical Connection Generation

From the example in Figure 5.3 (b), we find that the process of generating a vertical connection consists of the following steps, as shown in Figure 5.7:

1. Choosing a segment of contour between two layers with its height equals to $|\boldsymbol{D}|$.
2. Putting a vertical connection on the source layer side.
3. Pushing target layer toward source layer along the border higher than the

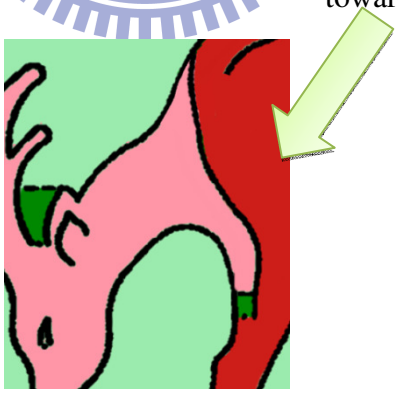
connection.

(a) Source layer (red) and target layer (pink) before generating connection.



(b) Putting connection on the source side.

(c) Moving border from target layer toward source layer.



(d) Result of generating vertical connection.

Figure 5.7: Generating process of vertical connection.

Therefore, given a vertical connecting segment, we split it into three parts: source layer ensuring part, connecting part, and eroding part. Figure 5.8 shows an example of segment

separation.

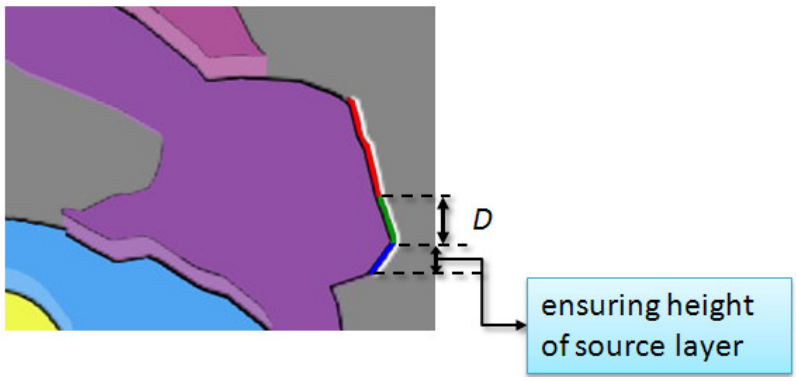
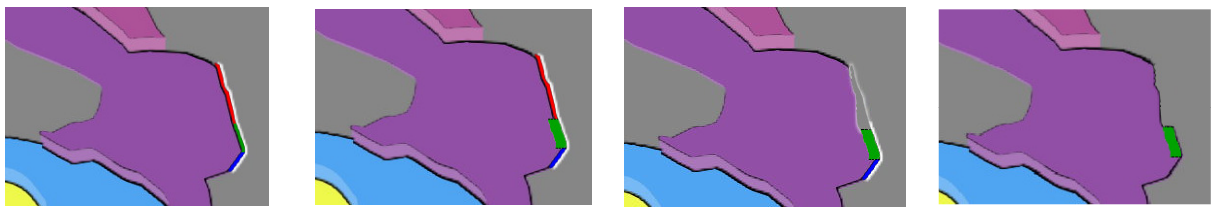


Figure 5.8: The vertical connecting segment is splitted into three parts. Bottom to top: source layer ensuring part (blue), connecting part (green), eroding part (red).

First we eliminate the bottom side of the connecting segment to ensure that the space of source layer is under the connection. Then the system puts the connection from the bottom of remaining segment with its height equals to $|\boldsymbol{D}|$ and user-defined width. As a result, the system generates eroding part for the rest of segment S_E. The width of eroding part W_E is calculated as:

$$W_{\boldsymbol{E}}(x) = W_C \times \frac{h(x)}{H} \qquad (5.6)$$

where x is a point of S_E, h is the height of x from bottom of S_E , and H is the height of S_E. Figure 5.9 illustrates the generating process of a vertical connection.



(a) (b) (c) (d)

Figure 5.9: Generating process of vertical connection.

(a) Separating process of connecting segment. (b) Generation of connection. (c) Erosion to source layer. (d) Result.

5.4 Layout Refinement

After generating connections, we compute scores of them, and put them onto layout from high score to low score. For an horizontal connection, the wider the connection is, the stronger the origamic architecture will be. For a vertical connection, the higher the erosion is, the smoother the segment between eroding region and source layer will be. Therefore, we define the scores of connection as:

$$S = \begin{cases} \ell \times \nu & \text{, if up} - \text{down connection} \\ H & \text{, if left} - \text{right connection} \end{cases} \tag{5.7}$$

where ν is a constant parameter to strike a balance between two types of connections.

For each layer, the system sorts connections by score, and puts connections onto layout sequentially from the highest score. In this thesis, we define a “good” connection as:

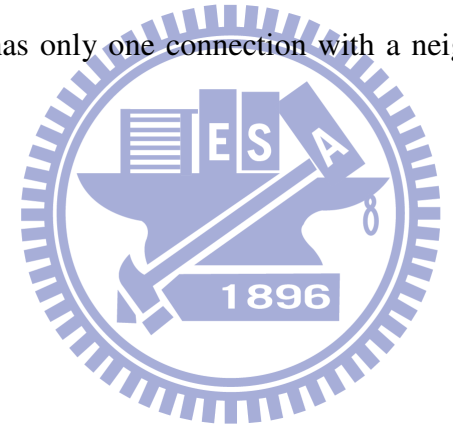
1. A connection whose width from bottom to top is wider than a user-defined threshold.
2. A connection which does not break source layer or target layer into pieces.

Whenever putting a connection onto layout, the system checks the remaining space of layout for connections. If the remaining space is enough for making this connection as a “good” connection, then this connection will be put onto layout. Otherwise, the system skips this connection and puts the next one. Note that backdrop could be broken into pieces before putting connections onto layout. Some of these pieces are cut out while generating origamic architecture. Considering stability of origamic architecture, these pieces will

eliminate the remaining space of layout for connection.

After putting all available connections onto the layout, the system checks the stability of each layer. If all of layers are stable, the layout will be output as the result for origamic architecture. Otherwise, the system will choose an unstable layer and merge it with the nearest neighboring layer in depth field, and re-generate layers and layout iteratively until all of layers are stable.

When choosing an unstable layer for merging process, the system gives the first priority to the layer which does not have any connection with other layers, and the second priority to the layer which has only one connection with a neighboring layer, and the last priority to all others.



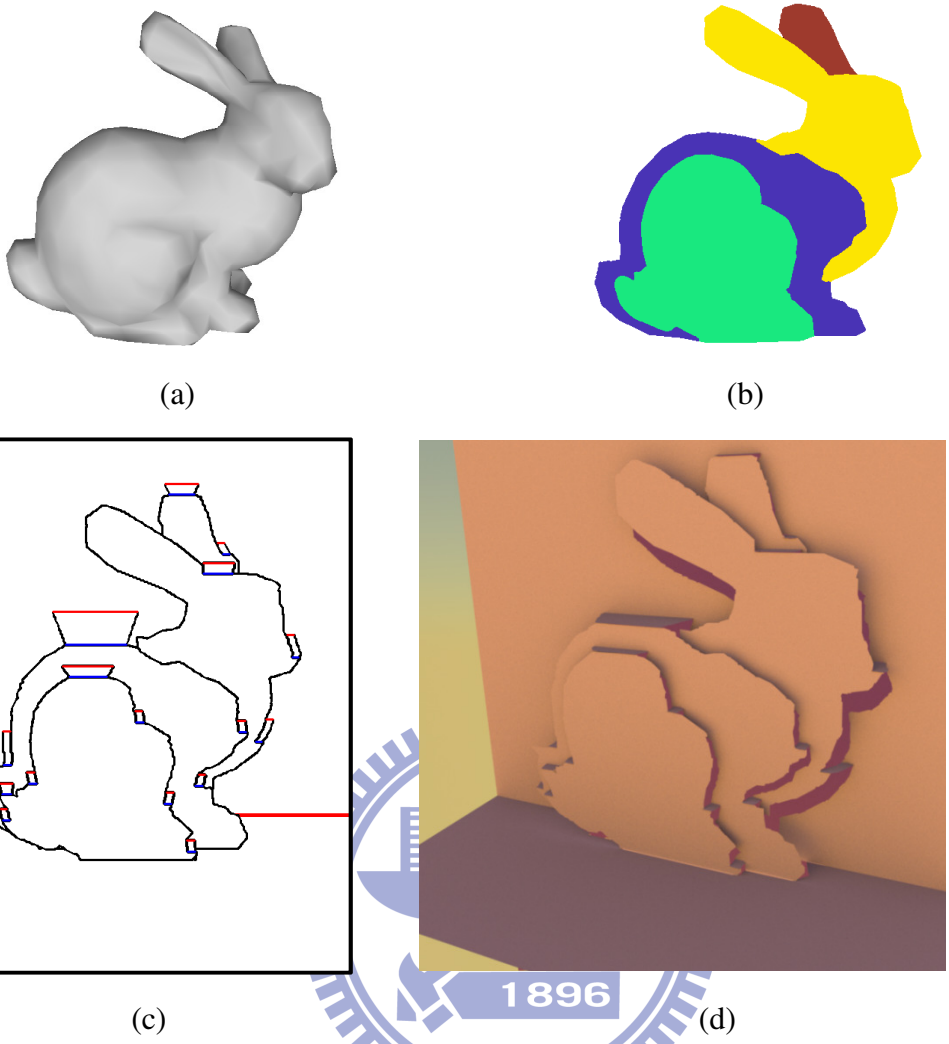
CHAPTER 6

Results

In this chapter, the implementation and results are presented. The input sources are 3D triangle meshes, and the output results are 2D layout images. The algorithm is implemented in C++ language using OpenGL and OpenCV. The experiment was carried out on a Intel® Core™ i7 PC with 3GHz CPU and 12GB memory.

In our system, users need to define some parameters for generating origamic architecture. In layers generating process, users have to control position of light source and input model, and change four parameters of depth map for extracting features of depth. Then users can decide how to merge neighboring segments for creating layers. In layout generating process, users need to choose minimum width for connections.

Example 1 is a simplified Stanford bunny which consists of 1,068 triangles as shown in Figure 6.1 (a). Figure 6.1 (b) shows the result of model segmentation. Figure 6.1(c) shows the layout of origamic architecture. Figure 6.1(d) shows the 3D result of origamic architecture. Figure 6.2 shows the result of Li [20], the shape of Stanford bunny looks fuzzy because of complex structure of origamic architecture. Comparing with the result of Li [20], our origamic architecture represents features of Stanford bunny with simpler structure clearly.



(a) (b)

(c) (d)

Figure 6.1: (a) Original 3D model of example 2. (b) Result of segmentation. (c) Result of Layout. (d) Result of origamic architecture.

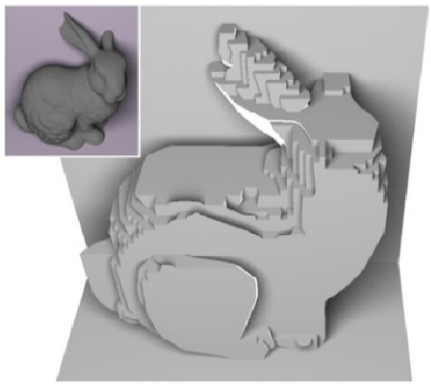
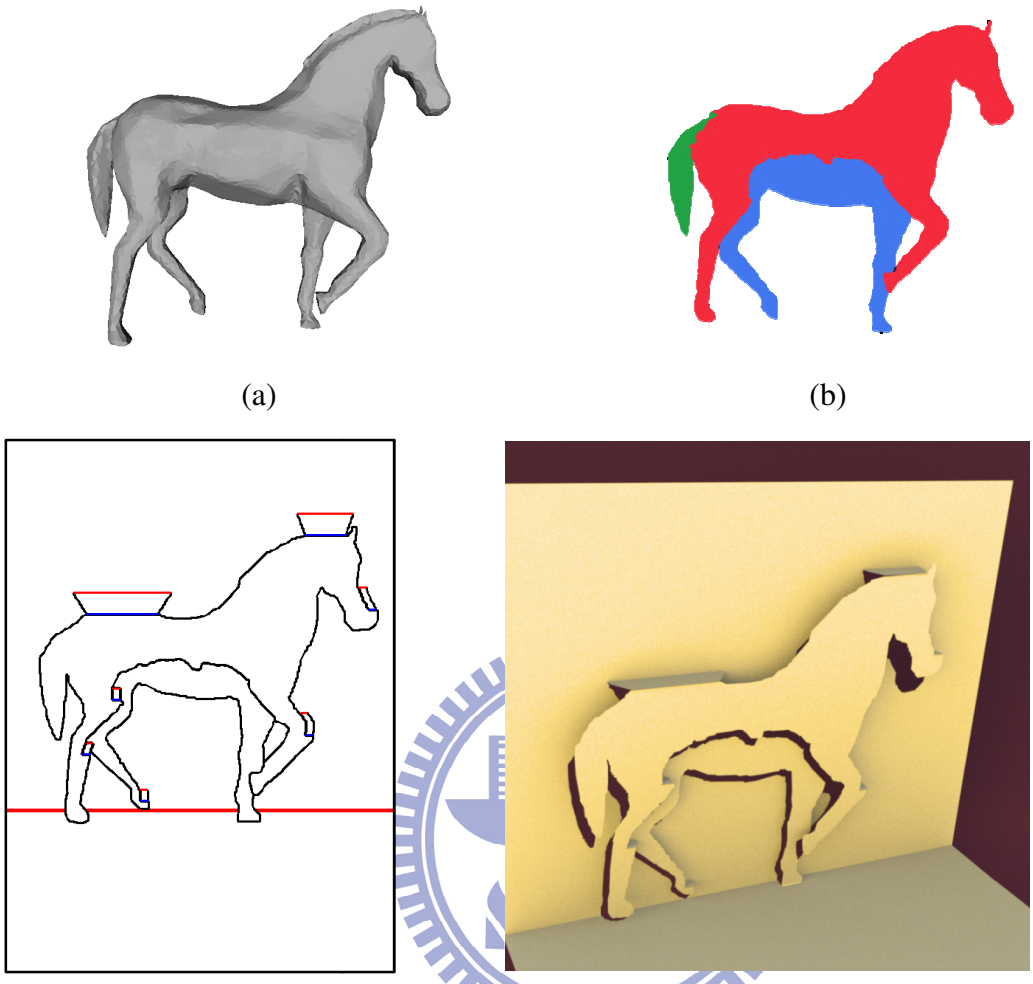


Figure 6.2: Result of Li [20].

Example 2 is a model of horse as shown in Figure 6.3 (a). Figure 6.3 (b) shows the result of model segmentation. Figure 6.3(c) shows the layout of origamic architecture.

Figure 6.3(d) shows the 3D result of origamic architecture.



(a) (b)

(c) (d)

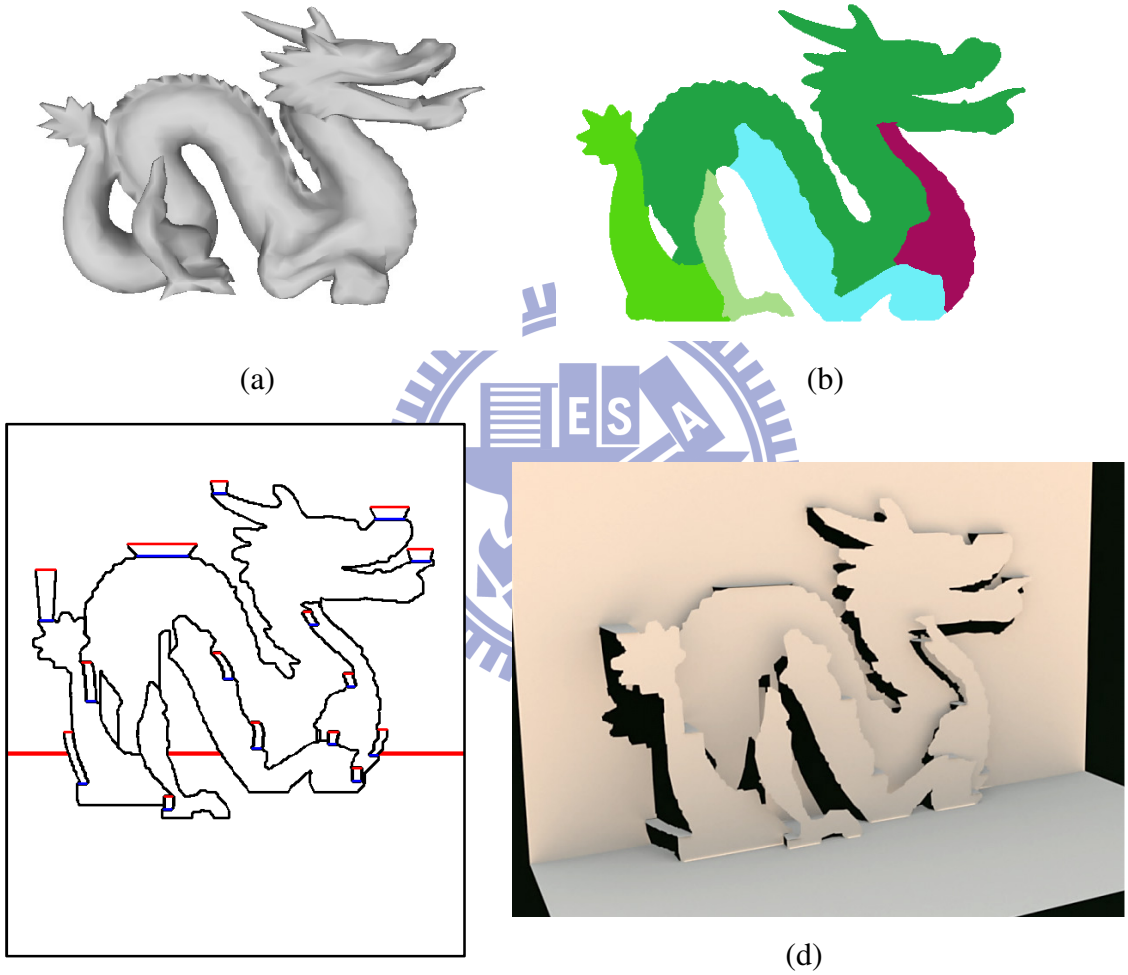
Figure 6.3: (a) Original 3D model of example 1. (b) Result of segmentation. (c) Result of Layout. (d) Result of origamic architecture.

Example 3 is a simplified Stanford Dragon which consists of 4,588 triangles as shown in Figure 6.4 (a). Figure 6.4 (b) shows the result of model segmentation. Figure 6.4(c) shows the layout of origamic architecture. Figure 6.4(d) shows the 3D result of origamic architecture.

Example 4 is a model of dairy cattle as shown in Figure 6.5 (a). Figure 6.5 (b) shows the result of model segmentation. Figure 6.5(c) shows the layout of origamic architecture.

Figure 6.5(d) shows the 3D result of origamic architecture.

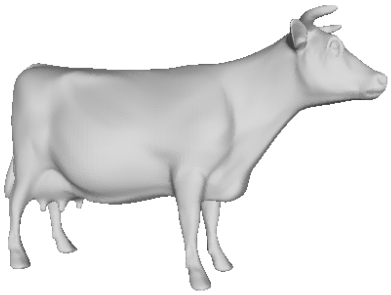
Example 5 is a model of walking cat as shown in Figure 6.6 (a). Figure 6.6 (b) shows the result of model segmentation. Figure 6.6(c) shows the layout of origamic architecture. Figure 6.6(d) shows the 3D result of origamic architecture.



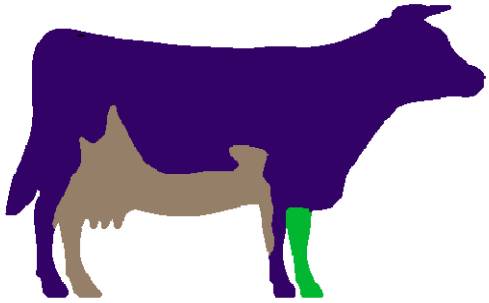
(a) (b)

(c) (d)

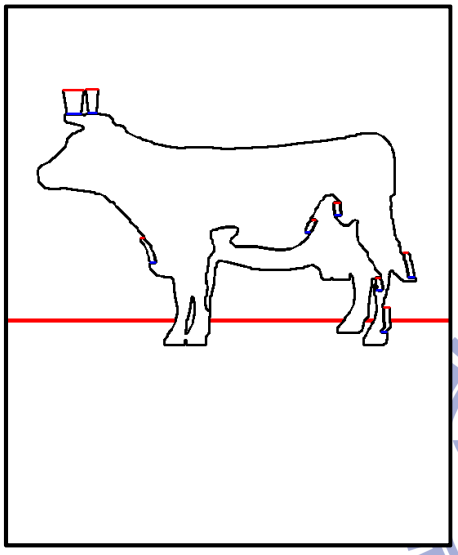
Figure 6.4: (a) Original 3D model of example 3. (b) Result of segmentation. (c) Result of Layout. (d) Result of origamic architecture.



(a)



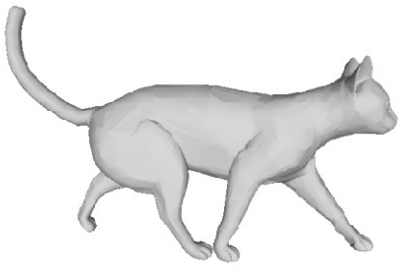
(b)



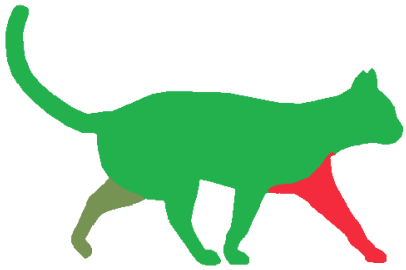
(c)

(d)

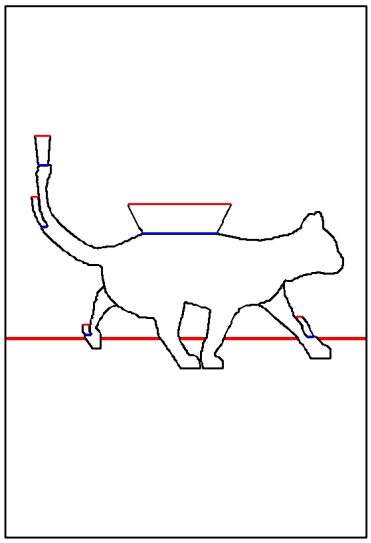
Figure 6.5: (a) Original 3D model of example 4. (b) Result of segmentation. (c) Result of Layout. (d) Result of origamic architecture.



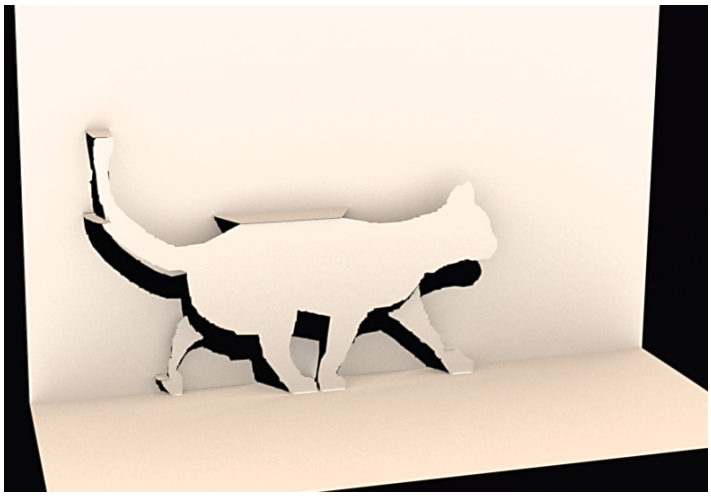
(a)



(b)



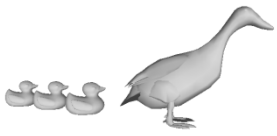
(c)



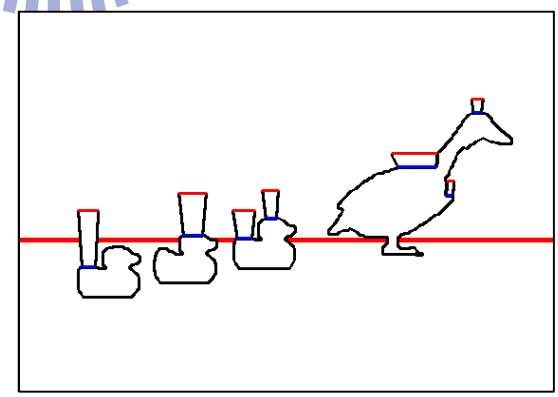
(d)

Figure 6.6: (a) Original 3D model of example 5. (b) Result of segmentation. (c) Result of Layout. (d) Result of origamic architecture.

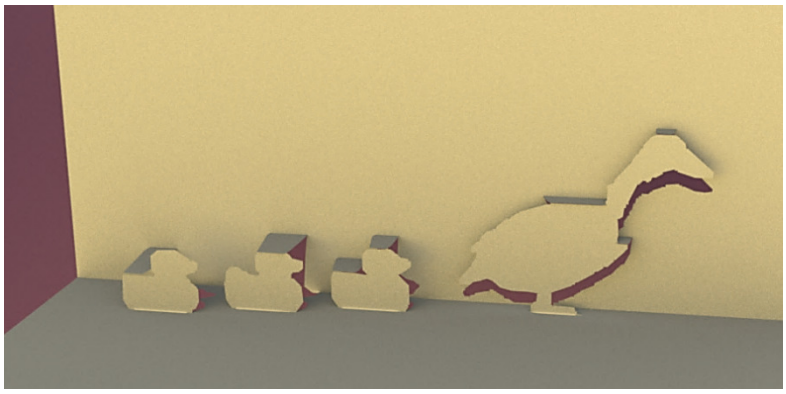
Our algorithm is flexible; users can design origamic architecture using multiple models. Example 6 puts two kinds of model in an origamic architecture as shown in Figure 6.7 (a). Figure 6.7(b) shows the layout of origamic architecture. Figure 6.7(c) shows the 3D result of origamic architecture.



(a)



(b)



(c)

Figure 6.7: (a) 3D models of example 6. (c) Result of Layout. (d) Result of origamic architecture.

For non-animal models, our algorithm is also very well for models which are special in shape. Figure 6.8(a) shows origamic architecture of liberty of statue created by our algorithm. Figure 6.8(b) shows the similar work of Masahiro Chatani.

(a) (c)

Figure 6.8: Two origamic architecture of statue of liberty. (a) The result of our algorithm. (b) The work designed by Masahiro Chatani.

For 90° animal origamic architecture, our algorithm works well. However, there are still some limitations. The rules of our algorithm cannot be applied to other types of origamic architecture, such as 180° and 360° origamic architecture.

CHAPTER 7

Conclusion and Future Work

In this thesis, we propose a system for generating origamic architecture of animals which takes features of animals into account. We propose a concept of ***layers*** and ***connections*** which is different from Li [20] in generating origamic architecture. For layers, we extract the features of model and put them on layers of origamic architecture. For connections, we take various types of border between layers into account, and ensure the stability of origamic architecture.

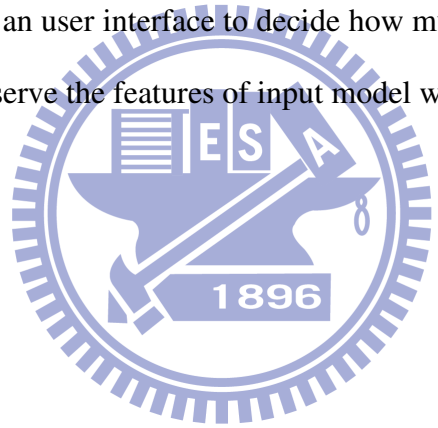
We implement this system by two major processes: layers generation and layout generation. In the former process, we provide an intuitive user interface for extracting the shapes of layers. First we extract the features of model by bi-level shading which reflects the feature of normal direction of model surface, and use depth map to extract the feature of model in depth of input model. In generating layout, we define two types of connection, horizontal connection and vertical connection, for handling various situations in generating connections at border between layers. Moreover, we define clear rules for checking stability of layers, and refine the layout to maintain the stability of origamic architecture. As a result, users can design an origamic architecture of animals without any skill or experiment in designing origamic architecture.

However, there are still some issues left to be studied in the future.

1. While extracting the segmentations of models, users have to change position of light source and model carefully. If segments do not show features of the model, the result of generated origamic architecture is hard to be recognized.

2. In some cases of artist designed origamic architecture, features of shape are also considered while generating connections. If the shapes of connections also reflect the features of model, it will be more attractive for generated origamic architecture.

3. If an origamic architecture has too much layers, the features of input model will be destroyed. Therefore, for a stable origamic architecture whose structure is too complex, we would like implement an user interface to decide how much and which layers should be merged in order to preserve the features of input model with few layers.



Reference

[1] 洪新富。 The Collection of the Paper Crafts . 台北市, 三采文化,民 85.

朝倉直巳。 紙的立體構成與設計. 台北市, 大陸, 民 79.

[2] 潘倩君。 “紙雕 - 台灣大百科全書”. In http://taiwanpedia.culture.tw/web/content?ID=7437 ,2010.

[3] Canny J. A computational approach to edge detector. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. Pami-8, NO.6 November, 1986.

[4] Chatani, M. *Origamic Architecture of Masahiro Chatani*. 東京, 彰國社, 1983.

[5] Chatani, M. *Origamic Architecture*, 彰国社, 1993.

[6] Chatani, M. and K. *Nakazawa Paradise of Origamic Architecture*, 彰国社, 1990.

[7] Cohen, J., Olano, M.,and Manocha, D. 1998. Appearance preserving simplification. In *SIGGRAPH '98: Proc. 25th annual conference on Computer graphics and interactive techniques, ACM*, New York, NY. USA, 115–122.

[8] DeCarlo, D., A. Finkelstein, et al. “Suggestive contours for conveying shape” . In *ACM Trans. Graph.* Vol.22, No.3, pp. 848-855, 2003.

[9] Demaine, E. D. and J. O'Rourke. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra, Cambridge University Press*, 2007.

[10] Eisemann, E. , Sylvain P. , Frédo D., A visibility algorithm for converting 3D meshes into editable 2D vector graphics, *ACM Transactions on Graphics (TOG)*, v.28 n.3, August 2009.

[11] Garland, M., and Heckbert, P. S. Surface simplification using quadric error metrics. In *SIGGRAPH '97: Proc. 24th annual conference on Computer graphics and interactive,* 1997.

[12] Glassner, A. “Interactive pop-up card design. Part 2” . In *Computer Graphics and*

Applications, IEEE Vol.22(, No.2), pp.74-85, 2002.

[13] Hertzmann, A. and D. Zorin. “Illustrating smooth surfaces” . In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques,* ACM Press/Addison-Wesley Publishing Co. , pp. 517-526, 2000.

[14] Hull, T. 1994. On the mathematics of flat origamis. *Congr. Numer*. 100, 215–224.

[15] Kalogerakis, E., A. Hertzmann, et al. “Learning 3D mesh segmentation and labeling”. In *ACM Transactions on Graphics (TOG)* Vol.29, No.4: 1-12, 2010.

[16] Kilian, M., Flöry, S., Chen, Z., Mitra, N. J., Sheffer, A., And Pottmann, H. 2008. Curved folding. *ACM Trans.* Graphics 27, 3, 75:1–9.

[17] Lai, Y.-K., Q.-Y. Zhou, et al. “Feature sensitive mesh segmentation”. In *Proceedings of the 2006 ACM symposium on Solid and physical modeling*. Cardiff, Wales, United Kingdom, ACM, pp. 17-25, 2006.

[18] Lang., R. J. *Origami 4*, A K Peters, Ltd, 2009

[19] Li, Y., Yu, J., Ma K.-L., and Shi, J. 2007 3d paper-cut modeling and animation. *Comput. Animat. Virtual Worlds* 18, 4-5, 395–403.2007.

[20] Li, X.-Y., C.-H. Shen, et al. “Popup: automatic paper architectures from 3D models” . In *ACM Trans*. Graph. Vol.29, No.4, pp.1-9, 2010.

[21] Mehra, R., Q. Zhou, et al. “Abstraction of man-made shapes”. In *ACM Trans. Graphics* Vol.28, No.5,pp. 137, 2009.

[22] Mi, X., D. DeCarlo, et al. “Abstraction of 2D shapes in terms of parts” . In *Proceedings of the 7th International Symposium on Non-Photorealistic Animation and Rendering*. New Orleans, Louisiana, ACM, pp. 15-24, 2009.

[23] Mitani, J., H. SUZUKI, et al.. “Computer aided design for origamic architecture models with voxel data structure” . In *Transactions of Information Processing Society of Japan* Vol.44, No.5, pp. 1372-1379, 2003.

[24] Mitani, J., and Suzuki, H. 2004. Computer aided design for origamic architecture models

with polygonal representation. In *CGI '04: Proceedings of the Computer Graphics International, IEEE Computer Society*, Washington, DC. USA, 93–99.

[25] Ohtake, Y., A. Belyaev, et al.. "Ridge-valley lines on meshes via implicit surface fitting". In *ACM Trans. Graph.* Vol.23, No.3,pp. 609-612, 2004.

[26] Okamura, S. and T. Igarashi. "An Interface for Assisting the Design and Production of Pop-Up Card". In *Proceedings of the 10th International Symposium on Smart Graphics*. Salamanca, Spain, Springer-Verlag, pp. 68-78, 2009.

[27] Tachi, T. 2009. Origamizing polyhedral surfaces. *IEEE Transactions on Visualization and Computer Graphics* 16, 2, 298–311.

[28] Wei, J., and Lou, Y. Feature preserving mesh simplification using feature sensitive metric. *Journal of Computer Science & Technology* 25, 3, to appear. 2010.

[29] Xu, J., Kaplan, C. S., and Mi, X. 2007. Computer-generated papercutting. In *PG '07: Proc. 15th Pacific Conference on Computer Graphics and Applications*, IEEE Computer Society, Washington, DC. USA, 343–350.