Double-sided 2.5D Graphics

Chih-Kuo Yeh, Peng Song, Peng-Yen Lin, Chi-Wing Fu, Chao-Hung Lin
and Tong-Yee Lee, *Senior Member, IEEE*

Abstract—This paper introduces *double-sided 2.5D graphics*, aiming at enriching the visual appearance when manipulating conventional 2D graphical objects in 2.5D worlds. By attaching a back texture image on a single-sided 2D graphical object, we can enrich the surface and texture detail on 2D graphical objects and improve our visual experience when manipulating and animating them. A family of novel operations on 2.5D graphics, including rolling, twisting, and folding, are proposed in this work, allowing users to efficiently create compelling 2.5D visual effects. Very little effort is needed from the user's side. In our experiment, various creative designs on double-sided graphics were worked out by the recruited participants including a professional artist, which show and demonstrate the feasibility and applicability of our proposed method.

Index Terms—2.5D modeling, vector art, layering.

1 Introduction

2D or 2.5D graphics have attracted great interests in a wide range of areas due to their simplicity and elegance for delivering conceptual and aesthetic-stylized art forms used in applications like manga, cartoon, and desktop publishing design. In this domain, the graphical elements in use are basically 2D meshes, raster images, and vector graphics, where 3D information is absent. However, the spatial scene can be 2.5D, meaning that layering can be used to arrange and order the 2D graphical elements in the scene to produce a visual illusion of proximity and occlusion among the on-stage objects.

Recent research in 2.5D graphics usually focuses on the creation of visual effects to bring out appealing and interesting visual perception to the audience. For instance, McCann and Pollard [1] developed the local layering method to create flexible and partial layering of 2D graphical objects, hence enabling more compelling and complicated local occlusion effects among the graphical elements in 2.5D worlds. Barnes et al. [2] developed a video-based puppetry system for users to quickly create interesting cutout-style and stop-motion animations. More recently, Rivers et al. [3] invented a new layer representation for 2.5D cartoon models, so that 3D rotation effects can be achieved even on 2D cartoon characters.

Following the spirit of these recent work in enriching 2.5D graphics, this paper proposes the novel idea of *making generic 2D graphics to be double-sided*, so that we can take advantage of the back image to provide additional information for 2.5D graphics. Moreover, we develop *a set of easy-to-use and user-manipulatable visual effects*: by using front and back images together, these new operations can greatly improve our ability to model and illustrate graphics in the 2.5D world:

- *Rolling*: Exposes a fraction of an object's back image along part of its silhouette to produce an effect of a (small-scale) pseudo rotation;
- *Twisting*: Produces a winding visual effect locally/globally on double-sided graphics by mixing elements from both front and back images;
- *Folding*: Partially or fully exposes the back image of a 2D graphical object and makes it self-layered in 2.5D. In addition, the folding boundary can be reshaped to improve the visual effect of folding.

Furthermore, these operations can be applied locally or globally on a given double-sided graphic tailored by the users for achieving their desired visual effects. Compared to existing work on 2.5D graphics modeling and manipulation, the main advantage of our proposed method is that it does not require any explicit/partial depth information or correspondence sketches through multiple views. Yet it can produce compelling visual effects on 2.5D graphics with very little modeling effort. Moreover, a family of easy-to-use operations is also designed and developed to maximize the usability and utilization of the back image. Lastly, various 2.5D graphical objects are presented in this work, and a number of participants, including a professional artist, are also recruited to try out our proposed interactive system.

The remainder of this paper is organized as follows. After the related work section (Section 2), Section 3 describes the modeling of double-sided graphics, and Section 4 presents our proposed operations to edit double-sided 2.5D graphics. Section 5 describes the user interface and the interaction procedures, whereas Section 6 presents the implementation detail and showcases the visual effects and interaction on as-

- *Chih-Kuo Yeh, Peng-Yen Lin, Chao-Hung Lin and Tong-Yee Lee are with the Department of Computer Science and Information Engineering, National Cheng-Kung University, Taiwan, R.O.C.*
- *Peng Song and Chi-Wing Fu are with Nanyang Technological University, Singapore.*

sorted 2.5D graphics. At the end, Section 7 draws the conclusion and discusses future extension work.

2 Related Work

In recent years, 2.5D graphical elements have been gaining increasing attention with several novel techniques, e.g., [1], [3], [4], proposed to work on them. In the work of McCann and Pollard [1], they proposed a local layering technique, allowing 2D graphical objects to overlap one another partially and locally. This technique generalizes the standard depth ordering mechanism employed in many conventional 2.5D modeling systems, enabling the design of more complicated and compelling visual effects with layering. More recently, McCann [4] proposed the soft stacking idea, where layers/objects can be mixed with one another in a volumetric or fog-like manner. By this, the volumetric media can be brought to 2.5D worlds with intriguing foggy effects. Another recent work is the 2.5D cartoon modeling method proposed by Rivers et al. [3]. They created a 2.5D cartoon model by associating user-drawn strokes with depth layers. Following the spirit of these 2.5D modeling work, we aim at enriching the visual appearance when manipulating 2D graphical objects in 2.5D world. Consequently, our work is more closely-related to [1] and [4] in terms of producing 2.5D visual effects through developing novel modeling strategies, rather than making 2.5D models to be fully 3D-rotatable as in [3].

Sketch-based modeling and animation [5], [6], [7], [8], [9], [10], [11], [12] is another stream of research highly related to this work. Robert et al. [5], Igarashi et al. [6], [11], and Karpenko et al. [9] proposed various compelling sketch-based interfaces for 2.5D or 3D modeling. These work focuses on providing an intuitive user interface and inferring depth information from sketches that are drawn on single or multiple views. Other than these work, Kho and Garland [8] presented an interactive sketching system that allows users to deform and edit 3D polygonal models by drawing sketches on the 2D screen. Cherlin et al. [7] developed a sketching interface that can produce convoluted cartoon-like features using rotational and cross-sectional blending surfaces. Nealen et al. [10] proposed a method, called FiberMesh, to build a 3D model by a collection of 3D curves. More recently, Li et al. [12] developed techniques that create cartoon facial animation from multi-view hand-drawn sketches. On the contrary, the goal of our proposed interface is to create interesting visual effects with 2.5D graphics by taking advantage of the back image rather than creating 3D information. Hence, we do not require inferring of depth information or sketching correspondence through single or multiple views.

Other related research work includes the followings. Winnemöller et al. [13] proposed a system that allows artists to design normal fields and texture maps to achieve the desired effects on image space. Di Fiore et al. [14] proposed the use of 3D skeletons to generate in-between views in hand-drawn cartoon by a multi-level 2.5D modeling approach. Igarashi et al. [15] introduced an as-rigid-as-possible 2D shape manipulation technique with multitouch capability, while Wiley [16] presented a vector-graphics drawing system, called Druid, which can handle self-overlapping surfaces by labeling the intersections of boundary curves. Eitz et al. [17] presented an image editing tool with a sketch-based interface, allowing users to deform and composite image regions intuitively. Barnes et al. [2] developed a video-based puppetry system for cutout-style and stop-motion animations. Sýkora et al. [18] employed block-based shape regularization to preserve local rigidity in hand-drawn cartoon animations, while Baxter et al. [19] developed a method that models a 2D animation as an N-way morphing problem. In our proposed approach, we develop novel ideas of *making 2D graphics double-sided* and *making use of the back images to produce a novel set of visual effects for the 2.5D worlds*. A family of novel and easy-to-use operations is also designed and proposed to support the creation of these visual effects and the manipulation of the 2D shapes.

3 Model Double-sided 2.5D Graphics

Input to our system are double-sided 2D graphics with both front and back images (see the left and right figures in Fig. 1). Comparing the two common formats for 2D graphics, bitmap images require higher resolution and anti-aliasing, while vector graphics, on the other hand, are defined mathematically, and thus can be smooth at any scale and resolution. To avoid jaggy and blurry effects in editing, scalable vector graphics (SVG) are adopted as our input.

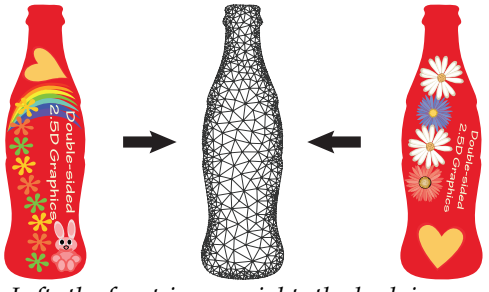


Fig. 1: *Left: the front image; right: the back image; middle: the boundary-aware triangulation.*

In addition, to support the proposed operations on double-sided graphics, the input shape is first triangulated to generate a 2D shape mesh. However, rather than a regular triangulation, we propose to adapt the triangulation to the boundary (see Fig. 1 (middle)), so that the triangulation can lead to more cost-efficient computation to support our proposed 2.5D operations while preserving the smoothness in the deformed silhouette. In detail, we employ the *Triangle* library

developed by Shewchuk [20] to produce this *boundary-aware triangulation*. Lastly, attributes including the visible side (front or back) and uv-coordinates (i.e., the coordinates in the parametric space) are attached to each mesh vertex to facilitate further computation in later stages (see next subsection).

4 Operations on Double-sided 2.5D Graphics

A family of operations on double-sided 2.5D graphics is proposed in this work, and this section describes each of them in turn as follows.

4.1 The Rolling Operation

Texture rolling is often used as a visual trick to create animation effects such as moving clouds and words spinning around an object. Formulating this rolling idea on double-sided graphics can enable us to generate pseudo 3D rotation (see Fig. 2). In short, such effect can be achieved by exposing (a fraction of) an object's back image along its silhouette, which requires only very little amount of resource in our case, i.e., the front and back images.

Fig. 2: *The front image (the leftmost one) and the back image (the rightmost one) are rolled along the silhouette to generate a pseudo 3D rotation effect (middle).*

In general, the rolling operation can be easily performed on input graphics that are square or rectangular, but in 2.5D worlds, since the shape of 2D graphical objects may not be that regular, the object boundaries can be convex or concave, thus making rolling more complicated to perform. To address this, our idea is to first embed the given shape (for both the front and back images) onto a rectangular domain by a mesh parameterization process.

The procedure is as follows: First, the user can optionally select (by marking on the SVG) part of the input shape as the region-of-interest (ROI) for performing rolling. In this way, we can localize the effect of rolling to that part of the input shape. If this substep is skipped (as in the case of Fig. 3), the rolling operation affects the entire shape. After that, the user is required to just successively mark up four points, say a, c, b, and d, on the silhouette of the ROI, which define the corners of a parametric domain and the rolling direction. The rolling direction

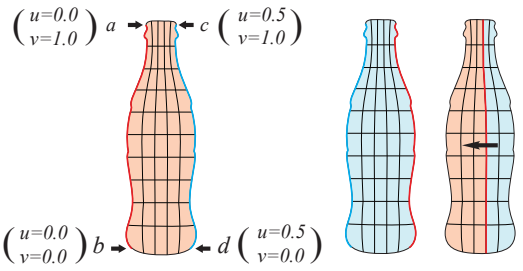


Fig. 3: *Left: Illustrations of the user-specified boundary points, a, b, c and d, and the defined parametric space (u,v). Right: illustration of rolling direction. The front and back images are represented by red and blue colors, respectively.*

is set to be perpendicular to the boundary lines, $\overline{ab}$ and $\overline{cd}$ (see the red and blue boundary lines shown on the left of Fig. 3), in the parametric domain. In computing this parameterization, the goal is to find a mapping that minimizes the distortion between the original mesh and the parameterized mesh with fixed and given boundaries, that are, $\overline{ab}$, $\overline{cd}$, $\overline{ac}$, and $\overline{bd}$. Once these boundaries are fixed in the parametric domain, we minimize the distortion by enforcing each parameterized vertex u_i to satisfy

$$\sum_{u_j \in N(u_i)} (cot\alpha_{ij} + cot\beta_{ij})(u_i - u_j) \;=\; 0\;, \qquad (1)$$

where $N(u_i)$ is the 1-ring neighborhood of vertex u_i; vertex u_i is the corresponding vertex of v_i in the parametric domain; α_{ij} and β_{ij} are the angles at the opposite sides of the edge (v_i, v_j) in the original mesh; and the conformal weight $cot\alpha_{ij} + cot\beta_{ij}$ is adopted to preserve the triangle shape in the parameterization (see also [21]).

To efficiently perform rolling interactively, the shape mesh with both the front and back images are embedded and packed side-by-side in a common uv parametric space with normalized range $[0,1] \times [0,1]$. As shown in Figs. 3 and 4, the front and back images are embedded in parametric range u: $[0.0, 0.5]$ $\times$ v: $[0.0, 1.0]$ and u: $[0.5, 1.0]$ $\times$ v: $[0.0, 1.0]$, respectively.

After this parameterization, we can obtain uniformly-packed front and back images (see the top right of Fig. 4). Essentially, it is ready for rolling, but applying rolling to such a parameterization may make rolling appear like uniform translation of a plane. Hence, we can optionally add depth cue into the rolling by embedding a simple cylindrical mapping into the parameterization as shown on the left hand side of Fig. 4. By distorting the parameterization by such a cylindrical mapping (see bottom right of Fig. 4), we could introduce certain pseudo perspective visual effect into the rolling operation.

Furthermore, since we use SVG as the input shape data structure in our implementation, we actually can examine its patch-based hierarchy (see Figs. 4 and 5), and then perform the rolling operation by sliding a clipping window in the parametric space. Figs. 4

and 5 show an example, where the brighter region is the clipping window (the amount of sliding is based on the magnitude of rolling), and those objects who overlap with the window boundary are clipped in the parametric space before taken to be rendered. Here we apply the polygon clipping algorithm by Vatti [22] to efficiently clip these geometric objects that are represented by discrete and closed polygons. In addition, since the geometric objects in SVG are stored in a hierarchical structure, we use a depth-first search strategy to look for overlapping shapes and thus can skip those nodes whose ancestors are completely inside or outside the clipping window.

To further enrich the visual quality of rolling, we may additionally put in shading as a depth cue. Here we define a shading map to make the colors in the middle of the rolling region to be brighter and those colors near the boundaries to be darker (see also the results shown in Fig. 2). This shading map can be efficiently obtained by utilizing the shape parameterization. Since the shape mesh is embedded in parametric range u: $[0.0, 0.5] \times v$: $[0.0, 1.0]$, we can define the pixel intensity in the shading map as $ShadingMap(p) = 1 - 2(2u(p) - 0.5)^2$, where $u(p)$ is the u-component of pixel p in case of a horizontal rolling (or we use the v-component in case of a vertical rolling). By this means, we can simulate 3D shading with the rolling effect. Lastly, it is worth noting that the silhouette and shape mesh of the rolling object is unchanged during the rolling because to perform the rolling, we only need to modify the way we map the SVG onto the ROI. Hence, we do not require re-triangulation and re-parameterization during the rolling action, and can adjust the amount of rolling interactively in our system. Note that boundary-aware triangulation is just a one-time offline pre-process for each input shape while the re-parameterization is done only after marking up the four corner points.

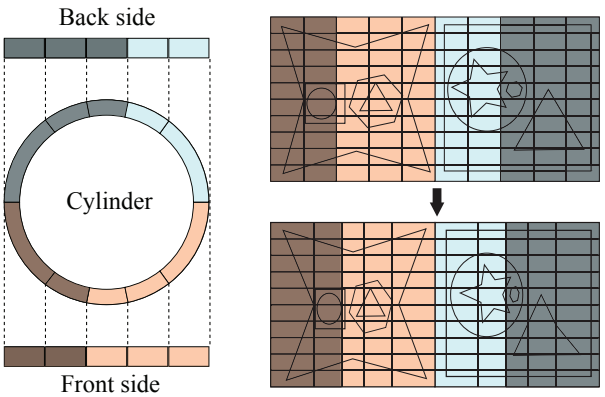


Fig. 4: *The parametric space. The front (red) and back (blue) images are embedded side-by-side in a common parametric space. Hence, rolling can be performed by sliding a clipping window (the brighter region) in this space.*

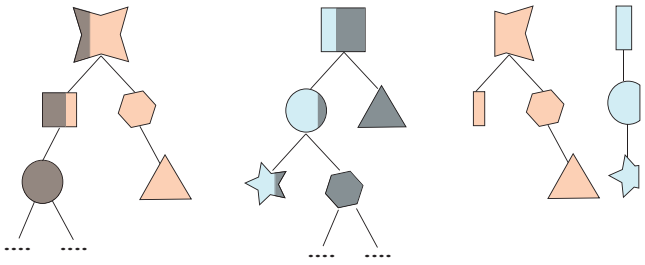


Fig. 5: *Illustration of hierarchical SVG clipping. Left: the clipping of the front graphics; center: the clipping of the back graphics; right: the clipping result, which can still be represented as a hierarchical SVG.*

4.2 Twisting Operation

Twist is a characteristic feature, which has been explored in many different research areas in computer graphics such as free-form object deformation [23], virtual cloth simulation [24], and sketch-based model deformation [8]. In conventional 2D cartoons, twist has also been used to produce an intriguing, provocative effect that is not necessarily physically-based (see Fig. 6 for examples). Using double-sided graphics, we can create and simulate such a visual effect by mashing the front and back images.

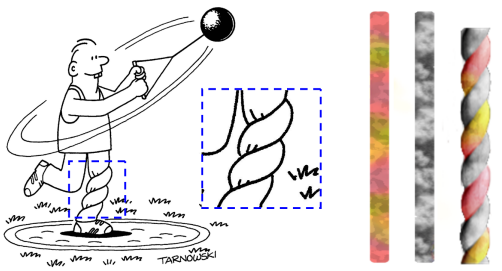


Fig. 6: *Twist examples.*

Our proposed method is described as follows. First, a family of cosine curves, denoted by f_k with frequency α and phase (shift) β, is created to model the front and back region boundaries (see Fig. 7):

$$f_k = \cos(\ \alpha x - k\beta\ ), \quad 0 < \beta < \pi\ , \tag{2}$$

and the first curve (last curve likewise) is specially modeled as

$$f_s = \begin{cases} f_{-1} & \text{if } x > x_0 \\ -1 & \text{otherwise} \end{cases}\ , \tag{3}$$

where $x_0 = (\pi - \beta)/\alpha$ is the x-coordinate of the point on f_{-1} with $y = -1$.

Observing that every visible element on a graphical object being twisted comes either from the front or back side, the front and back images can be interchanged in the twisting region (see the second row in Fig. 7). The divisions are based on the intersections between neighboring f_k's:

$$\begin{cases} s_k &= (k + 1/2)\ \beta/\alpha \\ e_k &= \pi/\alpha\ +\ (k - 1/2)\ \beta/\alpha\ . \end{cases} \tag{4}$$

To derive s_k, we base on the fact that s_0 is mid-way between the peaks of f_0 and f_1 (see Fig. 7) while successive s_k can be shifted from s_0 by $k\beta/\alpha$. As for e_k, it can be computed by the fact that the first intersection point $(\pi/2\alpha + k\beta/\alpha)$ between f_k and the x-axis is always mid-way between e_k and s_k. Hence, we can mix front and back images within $x \in [s_k, e_k]$, while the other ranges involve only a single side.

In order to deform the original mesh to these twisted boundaries, each mesh vertex within the twist region has to be deformed by perturbing its y-coordinates (see again Fig. 7) based on the $[s_k, e_k]$-range that it falls into. In detail, we first determine the pair of corresponding boundary curves (among the f's) that bound the vertex point, and compute the two bounding positions on the curve pairs, which have the same x-coordinate as the vertex. After that, the y-coordinate of the vertex can be perturbed by interpolating between the two bounding positions on the curves to produce the twisting shape, see Fig. 7 for the result. For non-rectangular shape, we additionally have to apply the parameterization model described in Section 4.1 to scale the interpolation, so that we can maintain the original shape after the twist, see the last row in Fig. 18 for an example.

After fixing the twisting shape, the next step is to distort the texture image. Similar to mesh deformation, texture distortion only takes place along y. Hence, we can first find the twisting curves in the image domain and the texture coordinates along y for each vertex can be computed by interpolating between the texture coordinates of its corresponding two bounding positions, see also Fig. 7.

Another example is shown in Fig. 8 where we adjust

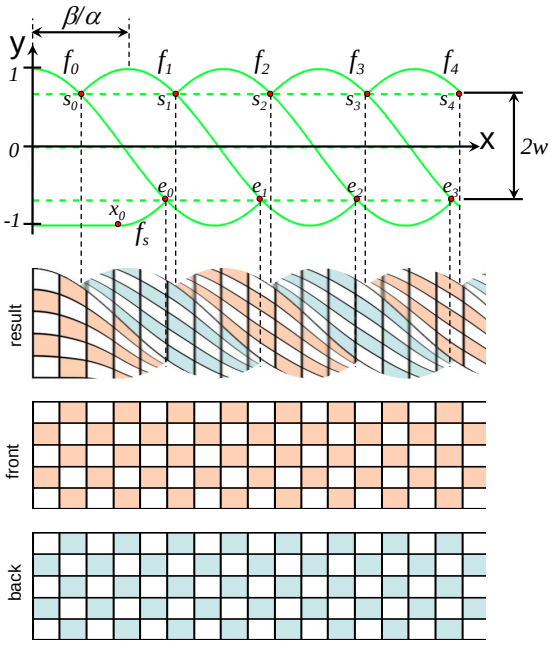


Fig. 7: *Modeling the twisting effect in 2D.*

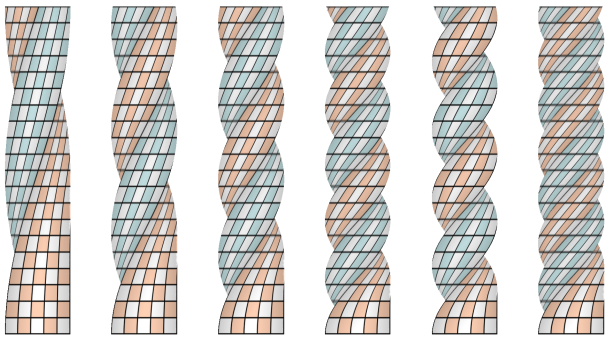


Fig. 8: *Twisting results. From left to right, the α values are 1.0π, 1.6π, 2.1π, 2.8π, 2.8π, and 2.8π; and the β values are 0.5π, 0.5π, 0.5π, 0.5π, 0.4π, and 0.6π.*

both α and β. In practice, we control the number of front and back regions by α and fix β (like a tightening factor) to be 0.5π. Lastly, since the twisting effect may still look flat, we can bump the normal of each vertex on the twisted mesh according to its nearest distance to the corresponding twisting boundary, and add shading effect to improve the perception of twisting.

4.3 Folding Operation

Folding is a useful operator employed in a number of 2D or 2.5D applications, such as the work of Beaudouin-Lafon [25] on managing windows in

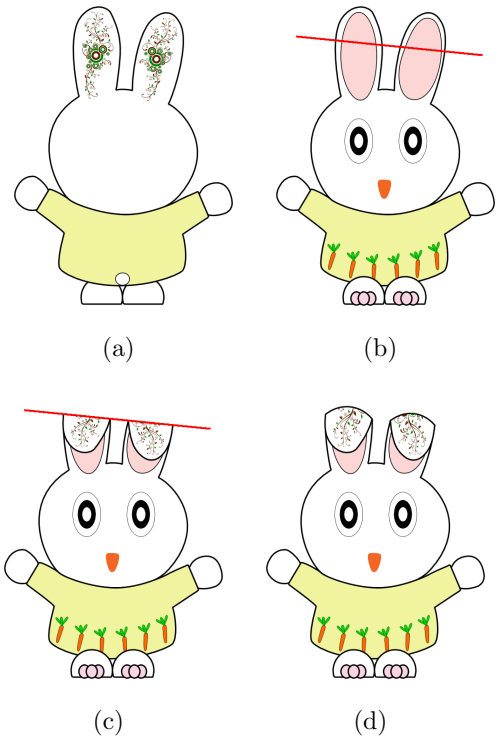


Fig. 9: *Folding a double-sided graphics. (a) The back graphics; (b) the front graphics and the folding line (in red); (c) forward folding result before refining the boundary along the folding line; (d) after refining the boundaries.*

desktop interfaces and the work of Igarashi and Mitani [26] on manipulating deformable objects. In this work, folding is proposed to be used to partially or fully expose the back side of a double-sided graphical object. The procedure is outlined in Fig. 9. After the user sketches a folding line and initiates a folding action (Fig. 9b) (forward/backward folding), our system bends the corresponding region(s) and creates a local layering (Fig. 9c). Further than that, we can also deform the boundaries on the folding line to make the results appear more natural (Fig. 9d).

In detail, this operation is implemented as follows. First, the user (optionally) selects an ROI and then specifies a folding line by sketching. Our system then computes the intersections between the folding line and the shape mesh within the ROI, and re-triangulates the shape mesh to make it pass through the folding line. After that, the user can select an object part to fold and specify the folding direction, i.e., forward or backward folding. Then, the selected part can be interactively bended to create a mirror-reflected local layering about the folding line. In addition, a tunable refinement curve can also be applied in terms of a handle constraint to locally deform the mesh by using the shape manipulation engine (see subsection 6.2).

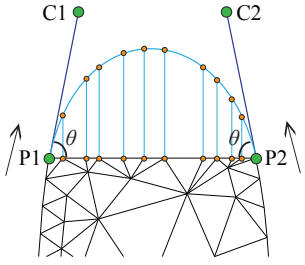


Fig. 10: *The generation of a cubic Bézier curve for deforming the boundaries on the folding line.*

In generating the refinement curve, a cubic Bézier curve is used. As illustrated in Fig. 10, the intersection between the folding line and shape mesh is denoted by the line segment $P_1 - P_2$ with additional control points, C_1 and C_2, for the Bézier curve. In our system, the two parameters, angle Θ and length $|P_1C_1|$, are tunable, and the default values for them are 45 degrees and $|P_1P_2|/2$, respectively. Besides, we fill the enlarged area resulted from the refinement by deforming the related region from the back (for forward folding) or front image (for backward folding).

Furthermore, to generate a folding animation, our system allows us to set keyframes for folding. In detail, we can smoothly compute the in-between folding lines along the boundary of the ROI, see Fig. 11, and thus animate the folding over time. Note further that by using the boundary-aware triangulation, we can efficiently preserve the shape silhouette while providing sufficient geometric information (with not too many triangles) for performing the three proposed interactive operations.

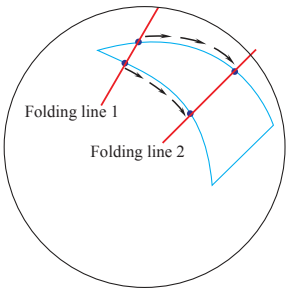


Fig. 11: *Illustration of keyframe setting. The in-between folding lines along the boundary of the ROI can be generated to produce a folding animation. Note that the black bounding circle outlines the shape of the orange, which is the actual 2D graphics that demonstrate this keyframe animation, see 1^{st} row in Fig. 16.*

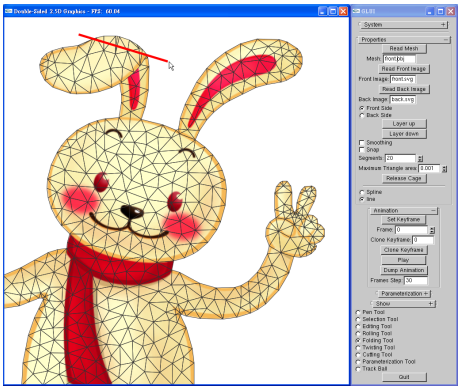


Fig. 12: *Our user interface. Easy-to-use sketching and markup are provided as interactive inputs. For instance, after sketching a folding line (in red) on the graphics, we can update the mesh and render the result in real-time.*

5 User Interface and Interaction Scenarios

Our user interface is shown in Fig. 12. All the operations can be interactively performed with real-time visual responses. Thus, the users can interactively and intuitively edit the double-sided graphics, which is very helpful for them to design and create their own graphical work. To provide an easy-to-use interface, we use sketching and markup as interactive inputs instead of low-level manipulation through primitive elements (e.g., vertex and face) on the shape mesh. Besides, the users can also animate the double-sided graphics by time-stamping poses of the shape. The time-stamped poses can act as keyframes, so that our system can generate animations by interpolating user mark-up and sketch information in-between keyframes. In the following, the interaction scenarios for the proposed operations are described.

Rolling in action. As shown in Fig. 13, the rolling operation can be done by the following steps: First, we define the ROI by sketching a rough curve or a closed polyline. Next, we mark up four corner points on the silhouette of the ROI, and our system can then compute the parameterization and the rolling direc-

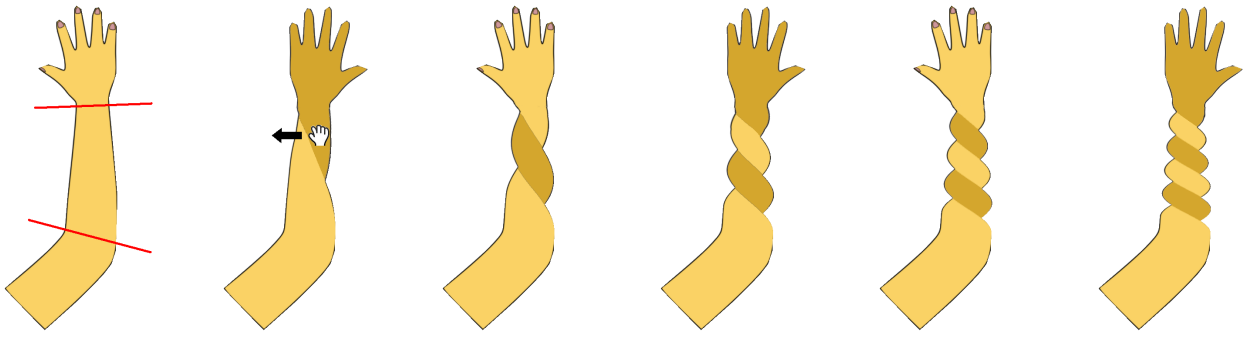


Fig. 14: *Twisting example. Left-to-right: sketch two boundary lines on the graphics, and then drag the mouse (to the left) to interactively adjust the amount of twisting.*

tion. After that, we can drag the mouse left-to-right or right-to-left to produce and interactively adjust the (amount of) rolling effect. This handy operation can quickly create pseudo 3D rotation on vector graphics.

Twisting in action. Twisting is performed as follows. First, we can mark up a pair of lines on the input shape to define the boundaries for twisting (see the pair of red lines in Fig. 14 (leftmost)). Then, we can drag the mouse in a direction roughly parallel to the lines to produce the twisting effect (see again Fig. 14). The magnitude of drag controls the amount of twisting and dragging to different directions can produce clockwise or anti-clockwise twisting.

Folding in action. As shown in Fig. 12, the folding operation can be done by simply sketching a folding line (the red line in the figure) after marking up the ROI. Then, our system can create and render the folding result with local layering in real-time. Moreover, we can also enable the user to tune the refinement curve to smooth the folding boundary. Lastly, one can also fold the same object multiple times by iteratively applying the folding operator with different folding lines, see Fig. 15 for an example, where we fold the snake object multiple times at different locations.

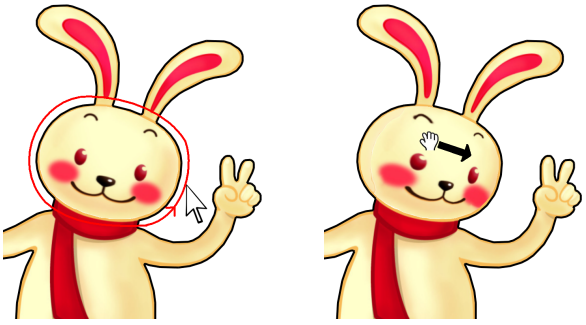


Fig. 13: *Rolling example. Left: sketch to select the ROI (note that this rabbit figure is a vector graphics composed of two ears, a face, etc.; hence a lasso can appropriately select its face); right: drag the mouse to roll the ROI.*

6 Implementation and Results

In this section, we first describe our system implementation with some performance data. Then, we present the shape manipulation engine to support the 2.5D operations. After that, we present the experimental results on using our system.

6.1 System implementation

Our proposed system is implemented and evaluated on a personal computer with a 2.66 GHz CPU and 4 GB memory. On average, for a double-sided graphical object modeled with a shape mesh of $3,800$ triangles and around $40,000$ line segments on both the front and back SVG graphics, the computation time for the preprocessing (i.e., mesh parameterization) and the mesh manipulation engine (see next subsection) are around 0.08 seconds and 0.018 seconds, respectively. In addition, the time taken for performing the rolling, twisting, and folding operations are 0.195 seconds, 0.035 seconds, and 0.327 seconds, respectively, thus showing that our system is able to provide real-time visual responses upon users' editing actions.

6.2 Shape Manipulation

To support the proposed 2.5D operations, we also need a basic engine for performing 2D shape manipulation: rotate, squash, stretch, and deform an input

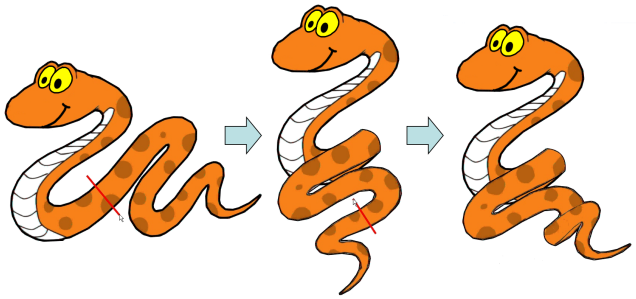


Fig. 15: *Folding a double-sided graphics multiple times. Left: the input object and the first folding line (in red); middle: the result after the first folding and the second folding line; right: the final result after the second folding.*

shape according to the user's specified point handles. Regarding this, we apply and integrate the as-rigid-as-possible shape manipulation method by Igarashi et al. [15] and the concept of conformal energy by Zhang et al. [27], and then tailor the engine for SVGs.

Since the input SVG is mathematically described, our shape manipulation engine begins by first converting the mathematical descriptions, i.e., B-spline curves in SVG, to line segments, and then constructing the boundary-aware triangulation based on the current screen resolution. Without loss of generality, the shape mesh is denoted by $\mathbf{M} = \{\mathbf{V}, \mathbf{E}, \mathbf{F}\}$, where $\mathbf{V} = [\mathbf{v}_0^T, \mathbf{v}_1^T, ...\mathbf{v}_{n-1}^T]$ denotes a set of vertex $\mathbf{v} = (x, y)$ in $\mathbf{R}^2$, $\mathbf{E}$ denotes a set of edges, and $\mathbf{F}$ denotes a set of triangles (faces). The conformal energy introduced in [27] is utilized to preserve the shape in deformation, i.e., minimizing the shape distortion between the original and deformed triangles. Specifically, a vertex in a shape mesh is transformed by

$$\begin{bmatrix} s & -r \\ r & s \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix}, \quad (5)$$

where s and r represent the scaling and rotation factors, respectively, and $[u, v]^t$ is the translation vector. Let $\mathbf{v}_{i_1}, \mathbf{v}_{i_2}, \mathbf{v}_{i_3}$ be the vertices of face f, and define

$$\mathbf{A}_f = \begin{bmatrix} x_{i_1} & -y_{i_1} & 1 & 0 \\ y_{i_1} & x_{i_1} & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ x_{i_3} & -y_{i_3} & 1 & 0 \\ y_{i_3} & x_{i_3} & 0 & 1 \end{bmatrix}, \quad \mathbf{b}_{f'} = \begin{bmatrix} x'_{i_1} \\ y'_{i_1} \\ \vdots \\ x'_{i_3} \\ y'_{i_3} \end{bmatrix}, \quad (6)$$

we can obtain the equation $\mathbf{A}_f[s, r, u, v]^T = \mathbf{b}_{f'}$. Let $\Omega_s = \|(\mathbf{A}_f(\mathbf{A}_f^T\mathbf{A}_f^T)^{-1}\mathbf{A}_f^T - \mathbf{I})\mathbf{b}_{f'}\|^2$, where the optimization $[s, r, u, v]^T = (\mathbf{A}_f^T\mathbf{A}_f)^{-1}\mathbf{A}_f^T\mathbf{b}_{f'}$, see also [27]. Note that Ω_s is used for measuring the conformal error between the original triangles and the corresponding deformed triangles under optimal scaling, rotation, and translation. In addition, the deformed mesh, say $\mathbf{M}'$, must satisfy the given handle constraints. Let $\mathbf{H}$ be a set of handle positions that are used to manipulate or deform the input shape. $\Omega_H = \sum_{i\in\mathbf{H}} \|\mathbf{v}'_i - \mathbf{H}_i\|^2$, where $\mathbf{H}_i$ is the i-th handle position. The deformed mesh $\mathbf{M}'$ is then solved by minimizing $\sum_f \Omega_s + w\Omega_H$, where $w = 1000$ in all our experiments. This optimization can be solved by a linear least-squares equation. As mentioned in [15], the above solver does not yield an as-rigid-as-possible deformed mesh, and thus requires a second optimization step to adjust the scale of the deformed mesh $\mathbf{M}'$. In Equation 5, the two-by-two matrix, say $\mathbf{T}_f$, denotes its similarity transformation with s and r, and its rotation component $\mathbf{T}'_f$ can be found by re-scaling $\mathbf{T}_f$ by $1/\sqrt{s^2 + r^2}$. Then, we formulate Ω_ξ as:

$$\Omega_\xi = \sum_{(i,j)\in\mathbf{E}(f)} \|(\mathbf{v}'_i - \mathbf{v}'_j) - \mathbf{T}'_f(\mathbf{v}_i - \mathbf{v}_j)\|^2 , \quad (7)$$

where Ω_ξ is used for measuring the scaling error between the original edges and the corresponding deformed edges under optimal similarity transformation, and $\mathbf{E}(f)$ refers to the set of edges around face f. In the second step, we minimize $\sum_f \Omega_\xi + w\Omega_H$ and compute the final deformed mesh $\mathbf{M}''$.

6.3 Results

To demonstrate the feasibility of our proposed system, we further recruited seven participants, including one professional artist, whose ages range from 20 to 38. After about 10 minutes' tutoring time on the system usage, each participant was given 15 minutes to practice and try the system followed by another 15 minutes to prepare their designs and corresponding materials. After that, the participants can use our system to make up their designs. Figs. 16 and 18 show some example designs, which took the participants about 3 minutes to create while the more complex design shown in Fig. 17 took the participant 15 minutes' time to finish.

As demonstrated in Fig. 16, the artist used the folding operation to create an animation showing the peeling of an orange, pea husking, and the peeling of a banana. He took about 3 minutes, 1.5 minutes, and 2 minutes to create these animations (from top to bottom), respectively. The results shown in Fig. 18 were created by other participants; these animation results were created in about 3 minutes, 2 minutes, 1 minute, 1.5 minutes, 20 seconds, and 20 seconds (from top to bottom), respectively. Operations including rolling, folding, twisting, and mesh manipulation were all involved in these examples. Fig. 17 shows a more complicated example created by the artist; it involves the use of multiple double-sided graphics with layering. To create this work, the folding animation on each double-side graphics is first created separately by using our system; then, these animations are combined with a specified rendering order by layering. These participant-made results showed that a variety of interesting effects and animations can be easily and efficiently created by our system.

6.4 Discussion on limitations

Our current system has certain limitations.

- First, we do not fully address the collision problem in folding since it is not the focus of this work. Hence, if the users do not explicitly control the layering by hand, self-collision or collision between objects may occur. However, we would like to highlight that by means of the interactive controls, one can usually interactively adjust the folding (folding direction, folding order, and layering control) to avoid the collision. In addition, for the case of collision between two different

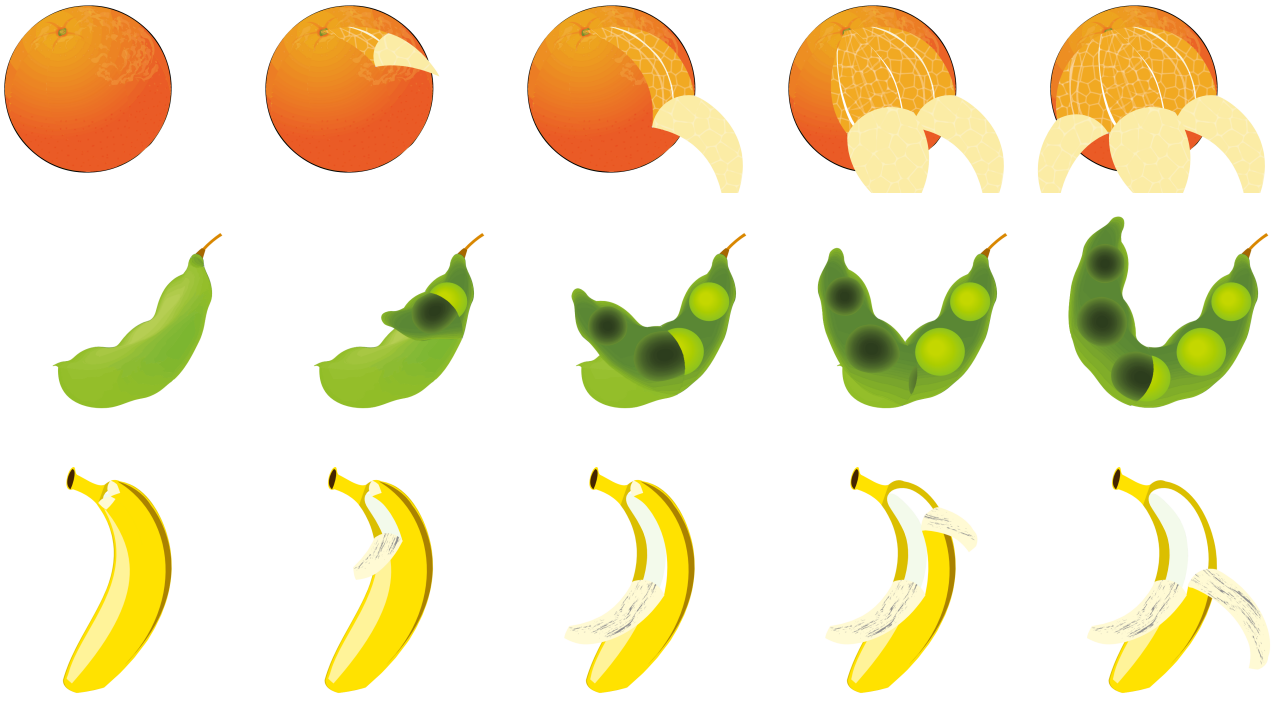


Fig. 16: *Folding animations created by an artist. Top: peeling an orange; three rolling regions are defined; Middle: husking a pea; Bottom: peeling a banana; two rolling regions are defined.*

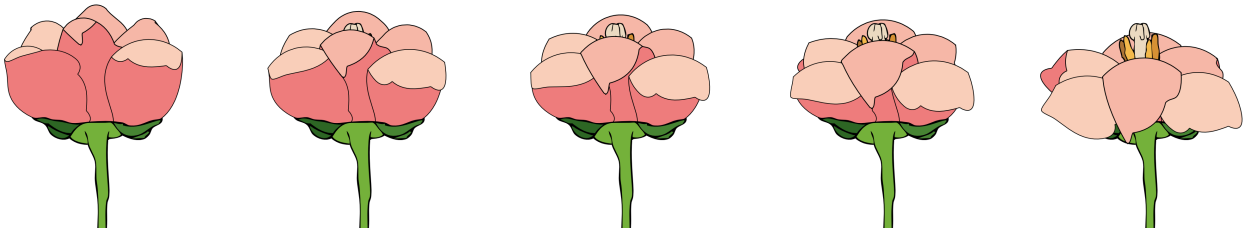


Fig. 17: *Animation of flower blossoming created by the artist. This work is done by layering multiple double-sided graphics. The folding operation is applied to each doubled-sided graphics first, and the folding results are then combined to generate this animation.*

objects, more flexible multi-layer or local layering operations can facilitate the handling of this problem. We leave this as future work.

- Second, when applying rolling or twisting to objects with concave or large protrusion structures, visual artifacts may occur, see Fig. 19 for an example, where sharp features such as the thunders on the cloud image are rather distorted.
- Lastly, when rolling is applied to a shape (or its local region) that is not logically (or at least causally) associated with a certain surface of revolution, e.g., the nose on the donkey's face shown in Fig. 20, such rolling is likely not sensible and the result could look very weird.

7 Conclusions and Future Work

This paper introduces a novel model of 2.5D graphics, namely double-sided 2.5D graphics, which allows us to bring in novel 2.5D effects through rolling, twisting, and folding. Similar to the spirit of local layering, this proposed idea can help enrich the way we model, render, and animate graphics in 2.5D worlds; in particular, we can perform the proposed operations interactively with our system. Key contributions in this paper include the idea of enriching 2.5D graphics with back images, the boundary-aware triangulation to efficiently preserve the shape silhouette while supporting the 2.5D operations, a tailored shape manipulation engine that integrates previous techniques for SVGs, a set of novel visual effects, i.e., rolling, twisting, and folding, produced from double-sided 2.5D graphics, and a family of easy-to-use user interface operations to produce these effects efficiently. Very little modeling and editing effort is needed from the user.

The effectiveness of our approach is demonstrated with several examples presented in this paper, includ-

ing those on various cartoon characters, fruit graphics, and a cola bottle. Furthermore, we also provide average-time statistics on using our system, which shows that our approach can be implemented as an interactive system. Lastly, we recruited a number of participants including a professional artist to try out our system; a variety of creative works on double-sided graphics were designed and created by them with our system, which demonstrate the feasibility, applicability, and efficiency of this work.

In the future, we plan to develop a nonhomogeneous parameterization approach that can efficiently preserve the shapes of highly salient objects by propagating the distortions to the homogeneous regions, as inspired by the work of Tzur and Tal [28]. In addition, we would also like to develop layer control operations similar to those in [29] to extend our framework to handle multi-layer double-sided graphics for designing more complex 2.5D graphics. Lastly, we also plan to study rendering methods to add shadows or halos around the silhouette of the folded parts in order to enhance the visual perception of folding (we thank one of the reviewer for this suggestion).

ACKNOWLEDGMENTS

We would like to thank the designer (membership No. 905020928 in *www.nipic.com*) for sharing the cute cartoon rabbit shown in Fig. 12, artist Paul Tarnowski for the hammer throwing cartoon shown in Fig 6 (license obtained from *www.CartoonStock.com*), and also the reviewers for the many constructive comments that help improve the paper. This work was supported in part by the National Science Council (contracts NSC-99-2221-E-006-066-MY3, NSC-100-2628-E-006-031-MY3, and NSC-100-2221-E-006-188-MY3), Taiwan, and MOE Tier-1 (RG 13/08) and MOE Tier-2 (MOE2011-T2-2-041), Singapore.

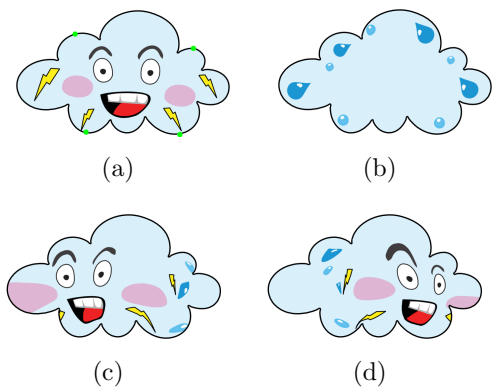


Fig. 19: *Failure example #1: Rolling a cloud shape (with protrusion structures). (a) front graphics; (b) back graphics; (c) result of rolling to the left; and (d) result of rolling to the right.*

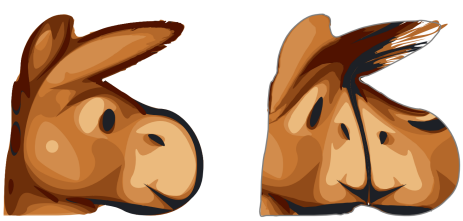


Fig. 20: *Failure example #2: Applying rolling to the nose of the donkey (not derived from a surface of revolution). Left: input image (same for both front and back); right: result.*

REFERENCES

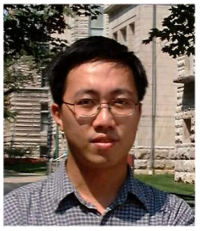
[1] J. McCann and N. Pollard, "Local layering," *ACM Trans. on Graphics (SIGGRAPH 2009)*, vol. 28, no. 3, article no. 84, 2009.
[2] C. Barnes, D. E. Jacobs, J. Sanders, D. B. Goldman, S. Rusinkiewicz, A. Finkelstein, and M. Agrawala, "Video puppetry: A performative interface for cutout animation," *ACM Trans. on Graphics (SIGGRAPH Asia 2008)*, vol. 27, no. 5, article no. 124, 2008.
[3] A. Rivers, T. Igarashi, and F. Durand, "2.5D cartoon models," *ACM Trans. on Graphics (SIGGRAPH 2010)*, vol. 29, no. 4, article no. 59, 2010.
[4] J. McCann, "Image editing and creation with perception-motivated local features," 2010, PhD Dissertation: CMU-CS-10-130.
[5] R. C. Zeleznik, K. P. Herndon, and J. F. Hughes, "Sketch: An interface for sketching 3D scenes," *ACM Trans. on Graphics (SIGGRAPH 1996)*, pp. 163–170, 1996.
[6] T. Igarashi, S. Matsuoka, and H. Tanaka, "Teddy: a sketching interface for 3D freeform design," *ACM Trans. on Graphics (SIGGRAPH 1999)*, pp. 409–416, 1999.
[7] J. J. Cherlin, F. Samavati, M. C. Sousa, and J. A. Jorge, "Sketch-based modeling with few strokes," in *Proceedings of the 21st spring conference on Computer graphics*, 2005, pp. 137–145.
[8] Y. Kho and M. Garland, "Sketching mesh deformations," in *ACM Symposium on Interactive 3D graphics and games*, 2005, pp. 147–154.
[9] O. A. Karpenko and J. F. Hughes, "SmoothSketch: 3D free-form shapes from complex sketches," *ACM Trans. on Graphics (SIGGRAPH 2006)*, vol. 25, no. 3, pp. 589–598, 2006.
[10] A. Nealen, T. Igarashi, O. Sorkine, and M. Alexa, "FiberMesh: designing freeform surfaces with 3D curves," *ACM Trans. on Graphics (SIGGRAPH 2007)*, vol. 26, no. 3, article no. 41, 2007.
[11] Y. Gingold, T. Igarashi, and D. Zorin, "Structured annotations for 2D-to-3D modeling," *ACM Trans. on Graphics (SIGGRAPH Asia 2009)*, vol. 28, no. 5, article no. 148, 2009.
[12] X. Li, J. Xu, Y. Ren, and W. Geng, "Animating cartoon faces by multi-view drawings," *Computer Animation and Virtual Worlds*, vol. 21, no. 3-4, pp. 193–201, 2010.
[13] H. Winnemöller, A. Orzan, L. Boissieux, and J. Thollot, "Texture design and draping in 2D images," *Computer Graphics Forum*, vol. 28, no. 4, pp. 1091–1099, 2009.
[14] F. D. Fiore, P. Schaeken, K. Elens, and F. V. Reeth, "Automatic inbetweening in computer assisted animation by exploiting 2.5D modelling techniques," *Proc. of Computer Animation 2001*, pp. 192–200, 2001.
[15] T. Igarashi, T. Moscovich, and J. F. Hughes, "As-rigid-as-possible shape manipulation," *ACM Trans. on Graphics (SIGGRAPH 2005)*, vol. 24, no. 3, pp. 1134–1141, 2005.
[16] K. Wiley, "Druid: Representation of interwoven surfaces in 2 1/2 D drawing," Ph.D. dissertation, University of New Mexico, 2006.
[17] M. Eitz, O. Sorkine, and M. Alexa, "Sketch based image deformation," *Proceedings of Vision, Modeling and Visualization (VMV)*, pp. 135–142, 2007.
[18] D. Sýkora, J. Dingliana, and S. Collins, "As-rigid-as-possible image registration for hand-drawn cartoon animations," *Proceedings of International Symposium on Non-photorealistic Animation and Rendering*, pp. 25–33, 2009.
[19] W. Baxter, P. Barla, and K. Anjyo, "N-way morphing for 2D animation," *Computer Animation and Virtual Worlds*, vol. 20, no. 2-3, pp. 79–87, 2009.

[20] J. R. Shewchuk, "Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator," in *Applied Computational Geometry: Towards Geometric Engineering*, ser. Lecture Notes in Computer Science, M. C. Lin and D. Manocha, Eds. Springer-Verlag, May 1996, vol. 1148, pp. 203–222, from the First ACM Workshop on Applied Computational Geometry.
[21] M. S. Floater and K. Hormann, "Surface parameterization: a tutorial and survey," *Advances in Multiresolution for Geometric Modelling*, pp. 157–186, 2005.
[22] B. R. Vatti, "A generic solution to polygon clipping," *Communications of the ACM*, vol. 35, no. 7, pp. 56–63, 1992.
[23] G. M. Draper and P. K. Egbert, "A gestural interface to free-form deformation," in *Proceedings of Graphics Interface*, 2003, pp. 113–120.
[24] P. Decaudin, D. Julius, J. Wither, L. Boissieux, A. Sheffer, and M.-P. Cani, "Virtual garments: A fully geometric approach for clothing design," *Computer Graphics Forum (Eurographics 2006)*, vol. 25, no. 3, pp. 625–634, 2006.
[25] M. Beaudouin-Lafon, "Novel interaction techniques for overlapping windows," in *ACM symposium on User interface software and technology*, 2001, pp. 153–154.
[26] T. Igarashi and J. Mitani, "Apparent layer operations for the manipulation of deformable objects," *ACM Trans. on Graphics (SIGGRAPH 2010)*, vol. 29, no. 4, article no. 110, 2010.
[27] G.-X. Zhang, M.-M. Cheng, S.-M. Hu, and R. R. Martin, "A shape-preserving approach to image resizing," *Computer Graphics Forum*, vol. 28, no. 7, pp. 1897–1906, 2009.
[28] Y. Tzur and A. Tal, "FlexiStickers: photogrammetric texture mapping using casual images," *ACM Trans. on Graphics (SIGGRAPH 2009)*, vol. 28, article no. 45, 2009.
[29] C. Fu, J. Xia, and Y. He, "LayerPaint: a multi-layer interactive 3D painting interface," in *ACM SIGCHI Conference on Human Factors in Computing Systems (CHI)*, 2010, pp. 811–820.

Chih-Kuo Yeh is a PhD student in Department of Computer Science and Information Engineering, National Cheng-Kung University, Taiwan. He received the BS degree in Department of Information Engineering and Computer Science from Feng Chia University in 2005 and the MS degree in Institute of Bioinformatics from National Chiao Tung University in 2007. His research interests include scientific visualization, computer animation and computer graphics.

Peng Song is a PhD student in the school of computer engineering at the Nanyang Technological University in Singapore. He received his B.S. in Automation from Harbin Institute of Technology (2007), and M.S. in Control Science and Engineering from Harbin Institute of Technology Shenzhen Graduate School (2010). His research interests include human computer interaction and computer graphics. He is a student member of ACM.

Peng-Yen Lin received the BS degree in Department of Computer Science and Information Engineering from the National Central University, Taiwan in 2009, and the MS degree in Computer Science and Information Engineering from the National Cheng-Kung University, Taiwan in 2011. His research interests include computer graphics and cartoon animation.



Chi-Wing Fu is an assistant professor in the school of computer engineering at the Nanyang Technological University in Singapore. He received his B.Sc. and M.Phil. in Computer Science and Engineering from the Chinese University of Hong Kong in 1997 and 1999, respectively, and his Ph.D. in Computer Science from Indiana University in Bloomington in December, 2003. His research interests include computer graphics, visualization, and human-computer interaction. He is a member of ACM and the IEEE Computer Society.

Chao-Hung Lin received his MS and PhD degree in computer engineering from National Cheng-Kung University, Taiwan in 1998 and 2004, respectively. He is currently an associate professor in the department of geometrics at National Cheng-Kung University in Tainan, Taiwan. He leads the Digital Geometry Laboratory, National Cheng-Kung University. His research interests include digital geometry processing, digital map generation, information visualization, and remote sensing. He is a member of IEEE and ACM.

Tong-Yee Lee received the PhD degree in computer engineering from Washington State University, Pullman, in May 1995. He is currently a distinguished professor in the Department of Computer Science and Information Engineering, National Cheng-Kung University, Tainan, Taiwan, ROC. He leads the Computer Graphics Group, Visual System Laboratory, National Cheng-Kung University (http://graphics.csie.ncku.edu.tw/). His current research interests include computer graphics, nonphotorealistic rendering, medical visualization, virtual reality, and media resizing. He is a senior member of the IEEE and the member of the ACM.

Fig. 18: *Animations created by general participants (non-artist). 1^{st} row: a rabbit animation created by combining rolling, folding, and mesh manipulation. 2^{nd} row: a cartoon character (mushroom man) animation created by combining rolling and mesh manipulation. 3^{rd} row: a bottle animation created by combining rolling and mesh manipulation. 4^{th} row: a cartoon character animation created by combining folding and mesh manipulation. 5^{th} row: a ghost animation created by rolling. 6^{th} row: twisting the hair of a cute girl character. Note that for each result that involves rolling (1^{st}, 2^{nd}, 3^{rd}, and 5^{th} rows), we draw green dots to indicate the corresponding four corner points for rolling on the base graphics (1^{st} column); all these results are done with a set of four corner points except the mushroom man on 2^{nd} row, where we use two sets of corner points simultaneously to create more lively rolling on both of its face and hat.*