



Tour Into the Picture:

Using a Spidery Mesh Interface to Make Animation from a Single Image

Youichi Horry^{*‡} Ken-ichi Anjyo[†] Kiyoshi Arai^{*}

Hitachi, Ltd.

ABSTRACT

A new method called TIP (Tour Into the Picture) is presented for easily making animations from one 2D picture or photograph of a scene. In TIP, animation is created from the **viewpoint of a camera** which can be three-dimensionally "walked or flown-through" the 2D picture or photograph. To make such animation, conventional computer vision techniques cannot be applied in the 3D modeling process for the scene, using only a single 2D image. Instead a **spidery mesh** is employed in our method to obtain a **simple scene model** from the 2D image of the scene using a graphical user interface. Animation is thus easily generated without the need of multiple 2D images.

Unlike existing methods, our method is not intended to construct a precise 3D scene model. The scene model is rather simple, and not fully 3D-structured. The modeling process starts by **specifying the vanishing point in the 2D image**. The **background** in the scene model then consists of at most five **rectangles**, whereas hierarchical polygons are used as a model for each foreground object. Furthermore a **virtual camera is moved around the 3D scene model**, with the viewing angle being freely controlled. This process is easily and effectively performed using the spidery mesh interface. We have obtained a wide variety of animated scenes which demonstrate the efficiency of TIP.

CR Categories and Subject Descriptors: I.3.3 [Computer Graphics]: Picture/Image Generation - viewing algorithms; I.3.7 [Computer Graphics] Three-dimensional Graphics and Realism, Animation

Additional Keywords: graphical user interface, image-based modeling/rendering, vanishing point, field-of-view angle

1 INTRODUCTION

Making animation from one picture, painting, or photograph is not a new idea. Such animations have been mainly used for

art and entertainment purposes, often with striking visual effects. For instance, 2D animations are commonly seen, where 2D figures of persons or animals in the original image move around, with the 2D background fixed. In relatively simple cases, these animations may be created using traditional cel animation techniques. If the animations are computer-generated, then 2D digital effects, such as warping and affine transformations, can also be employed.

However, it is still hard and tedious for a skilled animator to make computer animations from a single 2D image of a 3D scene without knowing its 3D structure, even if established digital techniques are fully available. When the input image is given in advance, first of all, the animator has to make the 3D scene model by trial and error until the projected image of the model fits well with the input image of the scene. At the very beginning of this process, the virtual camera position in 3D space must also be known as one of the conditions for the input image to be regenerated from the scene model. This poses the question, how is the camera position known by a single image? Unfortunately existing approaches to create models directly from photographs, such as image-based techniques, require multiple input images of photographs, and the cases discussed in this paper are outside their scope. If animating a painting is desired, making the animation may become more difficult, because a painting does not give as precise information for creating the 3D scene model as a photograph does.

The best possible approach currently available to making animation from a single image therefore depends largely on the skill, sense, and eye of the animators, though this naivety may place an excessive and tedious task load on the animators. They can then develop the scene structure freely, using vague and incomplete information included in the input to animate the scene to their liking. The scene structure, however, may still be incomplete. A more straightforward method is thus desired for creating the scene animation, in which the 3D modeling process of the scene is rather simplified or skipped.

In this paper we propose a simple method, which we call TIP (Tour Into the Picture), for making animations from one 2D picture or photograph of a scene. This method provides a **simple scene model**, which is extracted from the animator's mind. Thus the scene model is **not exactly 3D structured**, but is geometrically just a **collection of "billboards" and several 3D polygons**. Therefore, the animations obtained with our method are not strictly three-dimensional. However, as we show, the proposed method allows easy creation of various animations, such as **"walk-through"** or **"fly-through"**, while visually giving convincing 3D quality.

1.1 Related work

If a photograph is used as the 2D input image, then image-based methods, including [2, 4, 7] may be used effectively. In [2], the panoramic image is made from overlapping photographs taken by a regular camera to represent a virtual

^{*}Central Research Laboratory, 1-280 Higashi-Koigakubo Kokubunji Tokyo 185
{horry, arai}@crl.hitachi.co.jp

[†]Visualware Planning Department, 4-6 Kanda-Surugadai Chiyoda Tokyo 101
anjyo@cm.head.hitachi.co.jp

[‡]Currently visiting INRIA Rocquencourt, Domaine de Volceau - Rocquencourt 78153 Le Chesnay Cedex France horry@bora.inria.fr

environment, so that real-time walk-through animations can be made with the viewpoint fixed. The method in [7] provides animations, with many closely spaced images being required as input, and its theoretical background largely relies on computer vision techniques. This work can also be considered to belong to the category of techniques for light field representation [6], which gives a new framework for rendering new views using large arrays of both rendered and digitized images. Similarly, in [4] a "sparse" set of photographs is used for existing architectural scenes to be animated. Though the input condition is improved due to architectural use, multiple input images are still required. Despite successful results with these image-based approaches, we need a new methodology, especially for dealing with the situations where the input is a single photograph.

For paintings or illustrations, there are relatively fewer research reports on their animation. A new rendering technique was presented in [8] for making painterly animations. Assuming that the 3D geometric models of the objects in a scene are known in advance, animations in a painterly style are then made by the method using 3D particles and 2D brush strokes.

Morphing techniques including [1] provide 3D effects visually, requiring at least two images as input, although actually only 2D image transformations are used. For example the view interpolation technique [3] is an efficient application of morphing, which generates intermediate images, from images prestored at nearby viewpoints. View morphing [9] also gives a strong sense of 3D metamorphosis in the transition between images of the objects. Then we note that most of these techniques require no knowledge of 3D shape in morphing.

Existing methods cited above work effectively, when multiple input images are available, or when the 3D geometric structure of a scene to be animated is known in advance. Our approach treats the cases when one input image of a scene is given **without any knowledge of 3D shapes in the scene**. Theoretically it is impossible to create an animation from a single view of the scene. Instead, our approach actually gives a **new type of visual effect for making various animations**, rather than constructing a rigid 3D model and animation of the scene.

1.2 Main Idea

If we consider traditional paintings or landscape photographs, their perspective views give a strong impression that the scenes depicted are 3D. It is hard for us to find an *exact* position for the vanishing point of the scene in the picture. In particular, for paintings or drawings, the vanishing point is not precisely prescribed, being largely dependent on the artist's imagination. Therefore, rigid approaches, such as computer vision techniques, are not valid for the purpose of exactly finding the vanishing point. However, it is relatively easy for us to **roughly specify the vanishing point**, by manually **drawing guide lines for perspective viewing**. Then we can expect that the "visually 3D" geometry of the **scene's background** is defined as a simple model (with polygons, for instance) **centering around the user-specified vanishing point**. Similarly, in many cases, we can easily tell the foreground objects from the background through our own eyes. A simple and intuitive model of the **foreground object can then be like a "billboard"** that stands on a polygon of the background model.

The main idea of the proposed method is simply to provide a user interface which allows the user to easily and interactively perform the following operations.

- (1) **Adding "virtual" vanishing points** for the scene - The specification of the vanishing point should be done by the user, not automatically, as mentioned above.
- (2) Distinguishing foreground objects from background - The decision as to **whether an object in the scene is near the viewer should be made by the user**, since no 3D geometry of the scene is known. In other words, this means that the user can **freely position the foreground object**, with the camera parameters being arranged.
- (3) Constructing the background scene and the foreground objects by simple polygons - In order to approximate the geometry of the background scene, **several polygons should be generated to represent the background**. This model is then a **polyhedron-like form with the vanishing point being on its base**. The "billboard"-like representation and its variation are used for foreground objects.

These three operations are closely related to each other so that

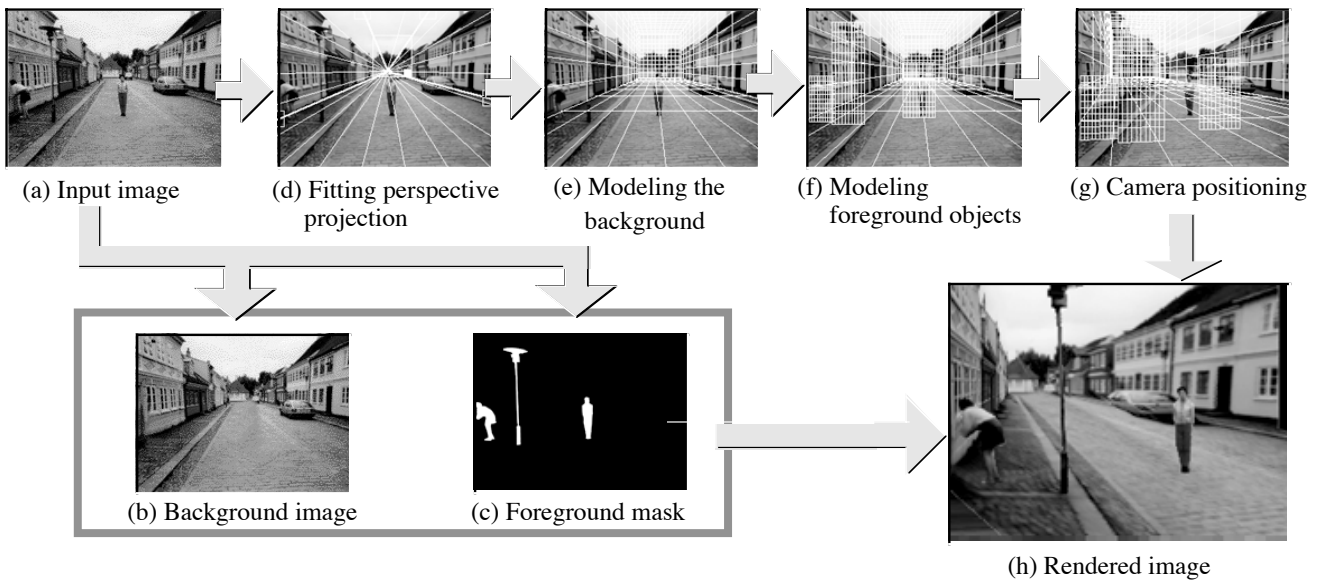


Figure 1. Process flow diagram

the interactive user interface should be able to provide their easy and simultaneous performance. A *spidery mesh* is the key to fulfilling this requirement.

The proposed method is outlined as follows. Fig. 1 shows the process flow.

After an *input image* is digitized (Fig. 1 (a)), the *2D image of the background* and *2D mask image of the foreground objects* are made (Figs. 1 (b), (c)). TIP uses a spidery mesh to prescribe a few perspective conditions, including the specification of a vanishing point (Fig. 1 (d)). In the current implementation of TIP, we can specify one vanishing point for a scene. This is not restrictive because many paintings, illustrations, or photographs can actually be considered one-point perspective, and because, as demonstrated later, the one-point perspective representation using spidery mesh works very well even for the cases where it is hard for us to tell if the input is one-point perspective or not.

Next, the *background is modeled with less than five 3D rectangles* (Fig. 1 (e)), and simple *polygonal models for the foreground objects* are also constructed (Fig. 1 (f)). Finally, by *changing the virtual camera parameters* (Fig. 1 (g)), images at different views are rendered (Fig. 1 (h)), so that the desired animation is obtained.

In section 2 the modeling process of the 3D scene (Figs. 1 (a) - (f)) in TIP is described. In section 3, after the rendering technique (Figs. 1 (g), (h)) is briefly mentioned, several animation examples are shown, which demonstrate well the efficiency and usefulness of the proposed method. Conclusions and future research directions of the method are summarized in section 4.

2 SCENE MODELING FROM A SINGLE IMAGE

In our method we use one picture or photograph of a 3D scene as input, from which we wish to make a computer animation. Then we specify one “virtual” (i.e. “user-specified”) vanishing point for the scene. As described later, this does not always mean that the input image must be one-point perspective. For convenience, the *line that goes through the vanishing point and view point is vertical to the view plane*. As for camera positioning, default values of *camera position*, *view-plane normal*, and *view angle* (field-of-view angle) are assigned in advance (see [5] for technical terms). These parameters are *changed later using our GUI* (Graphical User Interface) in 3.1 for making animations. For simplicity, the input images used are taken by the virtual camera without tilting, (though actually this condition can easily be eliminated). This means that the *view up vector*, which is *parallel to the view plane* in this paper, is *vertical to the ground of the 3D scene* to be modeled.

2.1 Foreground Mask and Background Image

In the modeling process we first derive two types of image information from the input 2D image: *foreground mask* and *background image*. Let F_1, F_2, \dots, F_p be subimages of the input image I , each of which is supposed to *correspond to a foreground object in the 3D scene* and is relatively close to the virtual camera. In practice the subimages $\{F_i\}_{i=1}^p$ are specified by a user and are modeled as *polygonal objects* in the corresponding 3D scene (see 2.3). The *foreground mask* is then defined as the 2D image consisting of $\{a_i\}_{i=1}^p$, where a_i is a grey-scaled masking value (a-value) of F_i . The *background image* is the 2D image which is made from I by retouching the traces of $\{F_i\}$ after the subimages $\{F_i\}$ are removed from I . The retouching process consists of occluding the traces of these subimages using color information for the neighborhood of each point (pixel) in F_i .

There is commercially available software, such as 2D paint tools, that enable us to easily make 2D images for the foreground mask and the background, from an input image. Fig. 1 presented an example. Fig. 1(a) showed the input image (of a photograph). The background image in Fig. 1 (b), as well as the foreground mask in Fig. 1 (c), were obtained using a standard 2D paint tool. To get the foreground mask in Fig. 1 (c), a street lamp and two persons were selected by the user, as the subimages $\{F_i\}$ mentioned above.

2.2 Specifying the Vanishing Point and Inner Rectangle

In order to model the 3D scene from the input image, we use our software called TIP, starting with the specification of the vanishing point of this image. TIP employs a unique GUI with a *spidery mesh*, which plays an essential role not only in the specification process but also in the processes thereafter.

Fig. 2 (a) shows the initial state of the spidery mesh in applying it to the input image in Fig. 1 (a). In general, as illustrated in Fig. 2 (a), the *spidery mesh* is defined as the 2D figure consisting of: a *vanishing point*; and an *inner rectangle*, which intuitively means the *window out of which we look at infinity*; *radial lines* that radiate from the vanishing point; an *outer rectangle* which corresponds to the *outer frame of the input image*. Each side of the inner rectangle is made to be parallel to a side of the outer rectangle. In TIP, the specification of the inner rectangle is done as well as that of the vanishing point. It should then be noted that, as described later, the inner rectangle is also used to specify the *rear window* in the 3D space (see 2.3 and 2.4). The rear window is a *border that the virtual camera*, which will be used in making an animation, *cannot go through*. The inner rectangle is consequently defined as the 2D projection of this window onto the 2D image space (i.e., the projection plane). In practice the 3D window is considered to be so distant from the current (initial) position of the virtual camera, that the camera does not zoom in beyond this window from the current position.

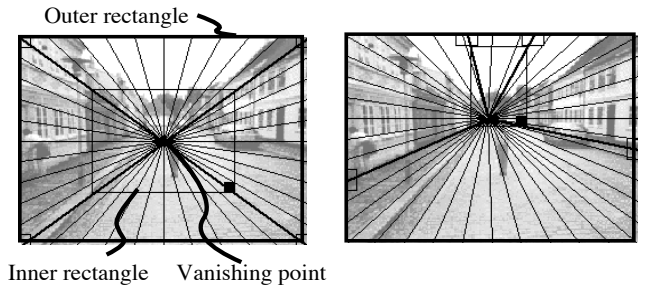


Figure 2. Spidery mesh on the 2D image

We now describe how to use the spidery mesh in order to specify the vanishing point. As described above, we then position the inner rectangle, along with the vanishing point. First we consider typical cases where the vanishing point is located in the input image (i.e., within the outer rectangle of the spidery mesh). Fig. 3 (a) is such a case. Then, using a *pointing device* (a mouse in the current implementation), we can *control the geometry of the spidery mesh* using the following functions.

- [a] *Deformation of the inner rectangle* - If the *right-bottom edge of the inner rectangle is dragged* with the pointing device, then the left-top edge of the rectangle is fixed, and the right-bottom edge is moved according

to the dragging (see Fig. 3 (a)).

- [b] **Translation of the inner rectangle** - If we drag a point on one of the sides of the rectangle (except the point at the right-bottom corner), then the rectangle is moved by the dragging distance (Fig. 3 (b)).
- [c] **Translation of the vanishing point** - If the vanishing point is dragged, then it is translated. The four **radial lines**, which are drawn boldly in Fig. 3, are also moved under the condition that these radial lines always go through the four edges of the inner rectangle, respectively (Fig. 3 (c)). If the cursor is dragged out of the inner rectangle, then the vanishing point is moved in the direction, and by the distance of, the dragging. Conversely, if one of these bold radial lines is translated by moving its edge on the outer rectangle, the vanishing point is moved based on a certain rule that we call **servility of the vanishing point** to the four (bold) radial lines. This means, for example, that, if we drag the edge of radial line L_1 in Fig. 3 (d) along the outer rectangle, then radial line L_2 is fixed and the vanishing point is moved along L_2 . The dotted lines in Fig. 3 (d) show the new positions of the bold radial lines with the source point of the dotted lines obtained as a result for the vanishing point.

Using these functions in our GUI, we can specify the vanishing point and the inner rectangle. In practice the radial lines are very helpful in the specification process. For example a user can specify the vanishing point, while **controlling the radial lines so that they go along the borderlines between buildings and roads** (see Fig. 2 (b)). Then servility of the vanishing point in [c] is useful in controlling the radial lines. It should also be noted that the concept of the spidery mesh is totally 2D, which assures easy-to-use and **real-time** feedback in the specification process.

As for the cases when the vanishing point is out of the input image (outer rectangle), functions similar to those described above can be applied, so that the inner rectangle is specified in the outer rectangle.

2.3 Modeling the 3D Background

The next thing we do is to **model the 3D background** of the scene using **very few polygons**.

Let us suppose that the vanishing point and the inner rectangle are specified as shown in Fig. 4 (a). We can then

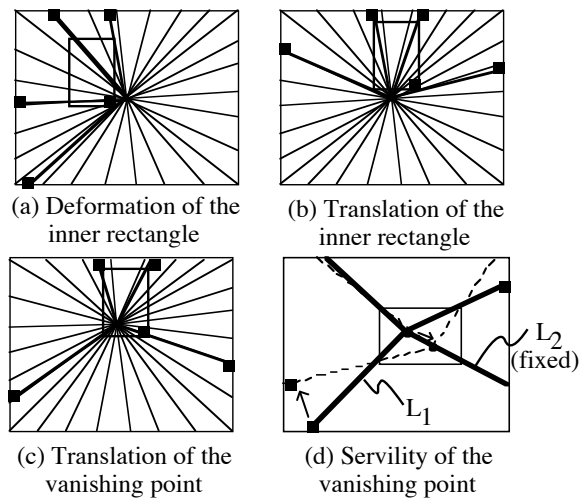


Figure 3. Controlling the spidery mesh

make a **2D decomposition of the outer rectangle** into five smaller regions each of which is a **2D polygon** in the outer rectangle. As illustrated in Fig. 4 (b), the five 2D rectangles may be deduced from these regions, and the rectangles are tentatively called the floor, right wall, left wall, rear wall, and ceiling, respectively (the rear wall is actually the **inner rectangle**). We define the **textures of these 2D rectangles** to be taken **from the background image**. Suppose that these rectangles are the **projection of the 3D rectangles**. We name each of these 3D rectangles the same as the 2D corresponding projection. We then define the **3D background model** in 3D space as being these five 3D rectangles, assuming that the following conditions hold:

- [A-1] Every adjacent 3D rectangle mentioned above is **orthogonal to the others**.
- [A-2] The 3D **rear wall is parallel to the view plane**.
- [A-3] The 3D **floor is orthogonal to the view up vector**.
- [A-4] The textures of the 3D rectangles are inherited from those of the corresponding 2D rectangles.

The vertices of these 3D rectangles are therefore easily estimated. For simplicity, we set the coordinate system of the 3D space so that the view up vector = (0, 1, 0) and the 3D floor is on the plane $y = 0$. Then the vertices of the 3D rectangles, which are numbered as shown in Fig. 4 (c), are

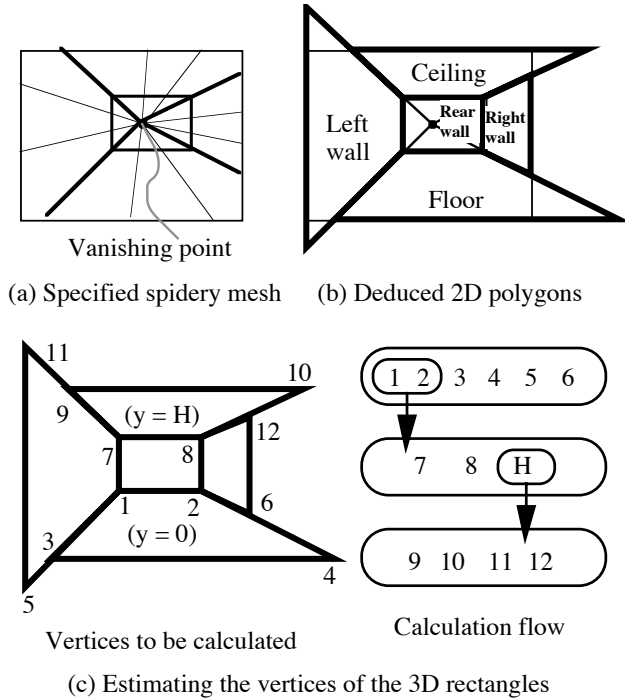


Figure 4. Modeling the 3D background

calculated as follows (also see calculation flow in Fig. 4 (c)). First we note that the 3D coordinate values of a point are easily obtained, if we know that it is on a certain (known) plane, and that its view plane coordinate values are known. Since we see the 2D positions of vertices 1 - 4 in Fig. 4 (c), we get the 3D positions of these four points, considering that these are on the plane $y = 0$. Similarly we get the values of vertices 5 and 6. Next we consider the plane which the 3D rear wall is on. The equation of this plane is then known, because it is vertical to the plane $y = 0$ containing the known vertices 1 and 2. Since vertices 7 and 8 are on this known plane, we can get the values of these two vertices. Then we estimate the "height" of the 3D ceiling. Since the 3D ceiling is on the plane parallel to the plane $y = 0$, we may assume that the 3D ceiling is on the plane $y = H$, for some H . If calculation of the y -values of vertices 7 and 8 contained no error, the y -values would be equal to H . However, in our implementation, we set H as the mean of the two y -values, in order to avoid errors. Thereafter, the y -values of vertices 7 and 8 are reset as being H . Consequently the remaining vertices 9 - 12 are estimated.

The 3D background model described above employs five rectangles, as shown in Figs.4 (c), (d). There are, however, some other cases when the background model uses fewer 3D rectangles. Treatments including these special cases are briefly described later in 3.3, along with the application examples.

2.4 Hierarchical Polygonal Models for the Foreground Objects

For the foreground objects in the scene, the foreground mask is prepared in advance. Based on the mask information, we construct the 3D polygonal model for a foreground object in the scene as described below. For simplicity, this model is hereafter referred to as a foreground object model.

First we consider the case in which the foreground object model is a quadrangle. The 2D quadrangle in the input image is then specified, so as to surround the 2D image of a

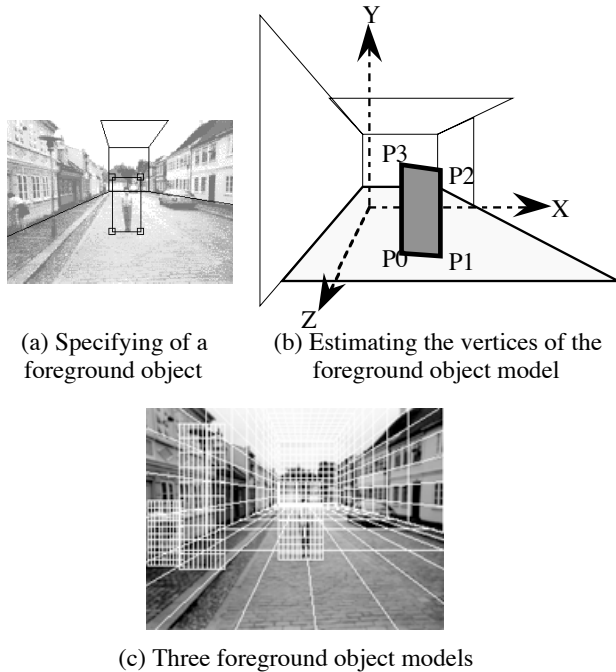


Figure 5. Modeling foreground objects

foreground object (i.e., F_i in 2.1). Next we specify the 3D position of the quadrangle in the 3D background model, under the condition that the quadrangle should be perpendicularly put on one of the five 3D regions: floor, right wall, left wall, rear wall, and ceiling. In the example of Fig. 5 (a), the person is a foreground object to be modeled, and is surrounded by the quadrangle (which is a rectangle in this case). The quadrangle in the 3D scene is perpendicularly attached to the 3D floor. By an argument similar to that in 2.3, we know the 3D positions of P_0 and P_1 in Fig. 5 (b). Then we get the equation of the plane which the quadrangle is on, and consequently the 3D positions of P_2 and P_3 are known. Thus the 3D quadrangle, which is the polygonal model for the person in Fig. 5 (a), is explicitly specified. In Fig. 5 (c), each of the three foreground objects (see Fig. 1 (a)) is modeled with a single 3D polygon, which has a fine mesh for clarity.

If the foreground object models are all quadrangles, the models may be restrictive in dealing with more complicated objects. The foreground object models in our method are therefore endowed with a hierarchical structure in the sense that

- 1) Each model consists of one or more polygons. In particular a single polygon itself is a foreground object model.
- 2) For any polygon F_1 belonging to the model, another polygon F_2 can be added to the model, if F_2 is orthogonally attached to F_1 so that one side of F_2 is on F_1 . Then F_2 is called a child of F_1 (or F_1 is a parent of F_2). This constitutes a hierarchy among the polygons belonging to a foreground object model.
- 3) If a polygon of the model is at the highest level in the hierarchy, it is orthogonally attached to one of the five 3D regions of the 3D background. Then only one side of the highest level is only on the region.

Fig. 6 illustrates how to construct the foreground object models. First, two quadrangles F_0 and F_1 are defined on the 3D floor (top sketch). Then F_2 is added to F_1 (middle sketch); and F_3 is added to F_2 (bottom sketch). In this way the

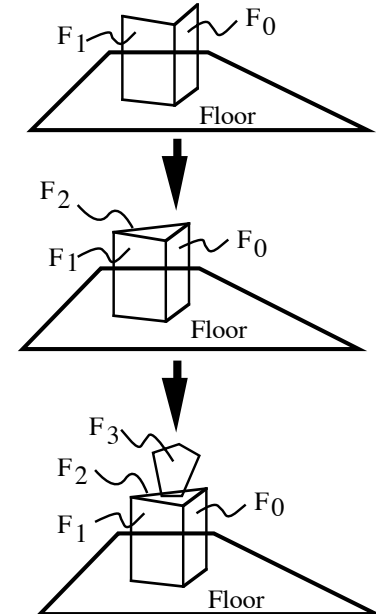


Figure 6. Hierarchical positioning of the foreground objects

foreground object models become more flexible (see 3.3 for a more concrete example).

3 ANIMATING AND RENDERING THE 3D SCENE

3.1 Camera Positioning

This section describes how to decide the virtual camera position for making an animation of the 3D scene. In using the virtual camera in our method, three parameters can be controlled: **camera position**, **view-plane normal** (vector), and **view angle** (i.e., field-of-view angle).

To visually control these parameters, the following commands are supported by our GUI in TIP, so that they are specified using pointing-device operations.

- [a] **Rotation**: The **view-plane normal** is changed by **rotations**. This essentially means **panning and tilting** at a fixed camera position.
- [b] **Translation**: With the view-plane normal fixed, the **viewpoint** (camera position) goes up and down; and right and left.
- [c] **Zoom**: The viewpoint is moved in the direction of the fixed view-plane normal.
- [d] **View angle**: The **magnitude of the view angle** is controlled with the viewpoint and the view-plane normal fixed.
- [e] **Look around**: After selecting the **center of attention** at

an object in the scene, the **viewpoint is changed on a sphere whose center is the center of attention** and whose radius is equal to the distance between the viewpoint and the center of attention.

3.2 Rendering via 2D-3D Transformations

The rendering method in TIP which is briefly described below is useful in making animation, particularly on a standard PC. The rendering procedure essentially involves an **ordinary texture mapping technique**. Of course much faster rendering techniques are available when using a powerful workstation.

Let (h_0, v_0) be an arbitrary point (pixel) in the 2D image to be rendered. Since we have the simple models for the 3D scene, as described in 2.3 and 2.4, we can get the **3D point P** which is **projected to (h_0, v_0)** . We also know the **2D point (h_1, v_1)** that is the 2D projection of **P** with the initial camera position. If **P** is a point on the 3D background model, then the color at (h_0, v_0) in the output image is the **color C_A** at (h_1, v_1) in the background image mentioned in 2.1. (C_A is the mean value of the colors at the four pixels nearest to (h_1, v_1) .) If **P** is a point on a foreground object model, we also know the color C_B at (h_1, v_1) in the input image and that there exists the subimage F_i including (h_1, v_1) . The color at (h_0, v_0) in the output image is then defined as $(1 - \alpha_i) C_A + \alpha_i C_B$, where α_i is taken from the foreground mask.

3.3 Animation Examples

The **camera positioning and the rendering processes** described



(a)



(b)



(c)



(d)

(a) Input image (b)-(d) Rendered images

Figure 7. Animation example preserving the view angle

above are performed by **turns**, in order to get a desired animation. The animation is then made by a **key-framing** method, based on the camera positioning information. The rendering process may still be rather time-consuming, compared to the camera positioning, and may be a bottleneck especially when performing a real-time walk-through by TIP on a standard PC. **Previewing at lower resolutions** would then be effective.

The following animations are TIP-implemented on a standard PC. The pixel resolutions of all the images in the animations have resolutions of 640×480 pixels. The wall clock time for **rendering per frame is on average 0.1 sec**, while real-time previewing is performed at 320×240 pixels.

The first animation example is made from the landscape painting in Fig. 7 (a)[§] which we can clearly take for a **one-perspective projection**. The frames in Figs. 7 (b) - (d), following the input image in Fig. 7 (a), are excerpts from the animation. Since the trees at the left-front of the input image are modeled as a foreground object (with a single quadrangle), the scene in Fig. 7 (b) is generated by setting the virtual camera in front of the trees. In Fig. 7 (c) the camera goes up toward the top of the trees, using the translation of camera movement (see [b] in 3. 1). Then the camera zooms in toward the **figures in the center, which are a foreground object model defined as a single quadrangle**. The natural perspective views, with the view angle fixed, which cannot be achieved by traditional 2D image extension operations, are obtained in this animation. It should be noted that the **vanishing point cannot be uniquely specified**. This may be a drawback for knowing the exact position of the vanishing point. However, if we wish to have a variety of animations from one 2D picture, the non-uniqueness of the vanishing point is a big advantage. Actually, just by changing the geometry of the spidery mesh, different types of animations from the same input image are provided.

The next animation example is made from the input image in Fig. 8 (a), which is not clearly identified with a one-point perspective projection. However, we can model the scene from this image, by specifying the vanishing point. The dynamic changes in Figs. 8 (b)-(d) are then obtained by our method. Though we may apply the spidery mesh in Fig. 4 (a) to this case, specifying the five rectangles as the 3D background in Fig. 4 (c), a simplified spidery mesh in Fig. 8 (e) is more convenient for practical use. In addition, the 3D background model is rather simple, as shown in Fig. 8 (f). It actually consists of only two rectangles for the floor and the rear wall.

The third example in Fig. 9 illustrates the efficiency of the foreground object model. In Fig. 9 (a), the input image is shown, and the linework in Fig. 9 (b) presents the modeled scene. The box-like object and the plant in Fig. 9 (a) are then considered as one foreground object which has an hierarchical structure (see 2. 4). As shown with the linework, the box-like object is modeled with several polygons, while a single polygon is used for the plant. The polygonal models in Fig. 9 (c) are used for previewing, and then Fig. 9 (d) gives a different view in the obtained animation.

The final example in Fig. 10 shows **view angle effects**. It is very interesting that, in a one-point perspective view, the **view angle can be specified independently** of the other

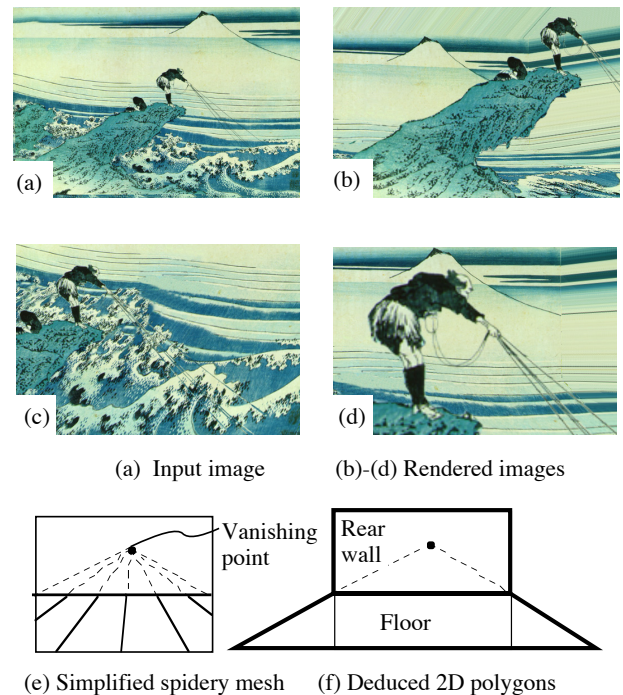
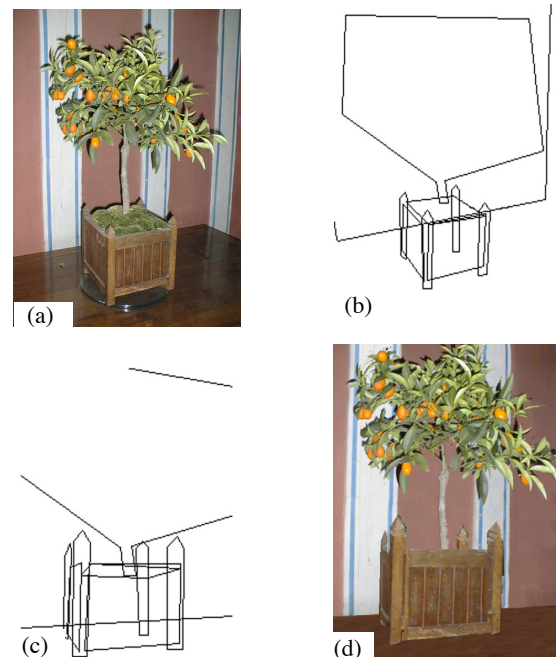


Figure 8. Animation example using the 3D background modeled with two rectangles



(a) Input image (b) 3D scene model obtained
(c) Different view of the 3D scene model (d) Rendered image

[§] Hitachi Viewsum image (<http://www.viewsum.com>); The Hudson River Portfolio Engraved by J.R. Smith (1775-1849) and John Hill (1770-1850) from watercolors by William Guy Wall (1792-c.1862) View Near Fishkill, c. 1821-25 Engraving/acquatint with hand-painted watercolor, 13 15/16 x 21 1/8" (image only, no text) Published by Henry I. Megary & W.B. Gilley, New York, and John Mill, Charleston, S.C., 1823 Gift of Miss Susan D. Bliss Collection The Hudson River Museum of Westchester Photo: Quesada/Burke

Figure 9. Animation example with the foreground object model hierarchically positioned

parameters. Based on the photograph in Fig.10 (a), completely different animations can be generated just by changing the view angles. Figs.10 (b) and (c) show different views with different view angles, both of which are excerpts from the animations starting with the same frame in Fig. 10 (a).

4 CONCLUSION AND FUTURE WORK

Creating 3D animations from one 2D picture or photograph of a 3D scene is possible for a skilled animator but often very laborious, since precise information for making the 3D scene model cannot be extracted from the single 2D image. However, the incompleteness of 3D information derived from one image allows animators to create the 3D scene model in a more flexible way. In this paper we have proposed a new technique called TIP which provides a simple scene model that transforms the animator's imaginings into reality. The key to our method lies in the GUI using a spidery mesh, which allows animators to easily model the 3D scene. This lets animators utilize the incomplete 3D scene information to freely create scenery to their liking and obtain enjoyable and "visually 3D" animations.

In this paper we restricted ourselves to the cases where **only one vanishing point** is specified by the animator. The animation examples demonstrated that our method works very well, without insisting that an input image is strictly one-point perspective projection. Actually, we showed that relaxing use of the one-point perspective representation can allow new visual effects for animations. For example, we can get various background and foreground models just by changing the geometry of the spidery mesh, which therefore provides different types of animations from the same input image. Changing field-of-view angle also provides a new visual deformation effect.

Of course there are many things to do next. Hierarchical foreground mask information would be more powerful in describing the scenes with more complex foreground objects. Multiresolution images would support finer zooming. We are currently extending our method, in order to treat two-point perspective projections. **Two-point perspective** is commonly used in the various fields of engineering, industrial design and advertising drawings. Unlike one-point perspective, the field-of-view angle is uniquely fixed so that the animations obtained will be more rigid, but still have many applications. Such an extended version of TIP would thus be used mainly for engineering or industrial design, whereas the current version

provides new and easy-to-use visual effects for making animations in art and entertainment.

ACKNOWLEDGMENTS

We are very grateful to the anonymous reviewers for their invaluable suggestions which made this a significantly better paper. Many thanks go to Tsuneya Kurihara and Hiroyuki Nomura for discussions at the early stages of this work. Thanks to Carol Kikuchi and Peter Lee for proofreading and comments. Thanks also to KimuAyu for her help and encouragement.

REFERENCES

- [1] Beier, T., and Neely, S. "Feature-Based Image Metamorphosis" Proc. SIGGRAPH '92 (Chicago, Illinois, July 26 - 31, 1992). In Computer Graphics, 26, 2 (July 1992), pp. 35-42.
- [2] Chen, S. E. "Quicktime VR - An Image-based Approach to Virtual Environment Navigation" Proc. SIGGRAPH '95 (Los Angeles, California, August 6 -11, 1995). In Computer Graphics Proceedings, Annual Conference Series, 1995. ACM SIGGRAPH, pp. 29-38.
- [3] Chen, S. E. and Williams, L. "View Interpolation for Image Synthesis" Proc. SIGGRAPH '93 (Anaheim, California, August 1 - 6, 1993). In Computer Graphics Proceedings, Annual Conference Series, 1993. ACM SIGGRAPH, pp. 279-288.
- [4] Debevec, P.E., Taylor C.A., and Malik J. "Modeling and Rendering Architecture from Photographs: A Hybrid Geometry- and Image- based Approach" Proc. SIGGRAPH '96 (New Orleans, Louisiana, August 4 - 9, 1996). In Computer Graphics Proceedings, Annual Conference Series, 1996. ACM SIGGRAPH, pp. 11-20.
- [5] Foley, J.D., van Dam, A., Feiner, S.K., and Hughes, J.F. *Computer Graphics: Principles and Practice*, Addison-Wesley, Reading, Mass., 1990.
- [6] Levoy, M. and Hanrahan, P. "Light Field Rendering" Proc. SIGGRAPH '96 (New Orleans, Louisiana, August 4 - 9, 1996). In Computer Graphics Proceedings, Annual Conference Series, 1996. ACM SIGGRAPH, pp. 31- 42.
- [7] McMillan, L. and Bishop, G. "Plenoptic Modeling: An Image-based Rendering System" Proc. SIGGRAPH '95 (Los Angeles, California, August 6 -11, 1995). In Computer Graphics Proceedings, Annual Conference Series, 1995. ACM SIGGRAPH, pp. 39-46.
- [8] Meier, B.J. "Painterly Rendering for Animation" Proc. SIGGRAPH '96 (New Orleans, Louisiana, August 4 - 9, 1996). In Computer Graphics Proceedings, Annual Conference Series, 1996. ACM SIGGRAPH, pp. 477-484.
- [9] Seitz, S. M., and Dyer, C.R. "View Morphing" Proc. SIGGRAPH '96 (New Orleans, Louisiana, August 4 - 9, 1996). In Computer Graphics Proceedings, Annual Conference Series, 1996. ACM SIGGRAPH, pp. 21-30.



(a) Input image



(b) View angle = 54 (deg.) (c) View angle = 150 (deg.)

Figure 10. View-angle effects