

## Image, Colour and Illumination in Animation

# Image and Video Retexturing

By Yanwen Guo\*, Jin Wang, Xiang Zeng, Zhongyi Xie,  
Hanqiu Sun and Qunsheng Peng

*We propose a novel image/video retexturing approach that preserves the original shading effects without knowing the underlying surface and lighting conditions. For static images, we first introduce the Poisson equation-based algorithm to simulate the texture distortion on the projected interest region of the underlying surface, while preserving the shading effect of the original image. We further work on videos by retexturing the key frame as static image and then propagating the results onto the other frames. In video retexturing, we have introduced the mesh based optimization for object tracking to avoid texture drifting, and the graph cut algorithm to effectively deal with visibility shift between frames. The graph cut algorithm is applied on a trimap along the boundary of the object to extract the textured part inside the trimap. The proposed approach is developed in image/video retexturing at nearly interactive rate, and our experimental results have showed the satisfactory performance of our approach. Copyright © 2005 John Wiley & Sons, Ltd.*

KEY WORDS: image/video retexturing; Poisson equation; graph cut

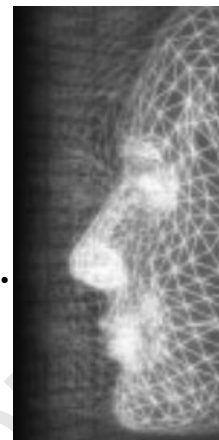
## Introduction

Retexturing is the process of replacing existing textures in the concerned region of images/videos with new ones while preserving the original shading effects. It has wide applications in special effects in TV and film producing, art and industrial design, distance learning, digital entertainment, and E-commerce. To achieve realistic retexturing effects, two basic problems must be solved. One is to make the new texture adequately wrapped and shaded so that it is consistent with the unknown shape of the underlying surface as well as the unknown lighting condition encoded by the original image. The other is how to prevent the new texture drifting on the interested region between adjacent frames in video. The related research is mainly proposed for the problems individually, but not the problems simultaneously for both image/video retexturing

with the same motivations. In this paper, we propose a novel image/video retexturing approach while preserving the original shading effects.

Manipulating textures in real images has been fascinating people for a long time, and the understanding of texture evolves meanwhile. Early works, model texture as a statistical attribute<sup>1</sup> of a surface and decompose real-world texture into a texture part and a lighting part.<sup>2</sup> Recent studies categorize real-world textures into regular and irregular types<sup>3,4</sup> and decompose the texture into geometry, lighting, and color components. Nevertheless, recovering the geometry, lighting components of a texture in a single image is very difficult. Because real-world images are usually taken under very complex environment, physically based techniques like shape-from-shading (SFS) are sometimes complex, unstable, and inaccurate.

Rather than recovering the geometry and lighting information, our retexturing approach of images aims at producing an illusion such that the replaced texture inherits the geometry and lighting information implied in the input image. By solving the Poisson equations, a non-linear mapping between the new texture and the concerned region on the image is derived reflecting the original shape of the underlying surface. Lighting effect on the original image is retained by adopting the YCbCr color space to represent the previous texture value at



\*Correspondence to: Yanwen Guo, State Key Lab of CAD&CG, Department of Mathematics, Zhejiang University, Hangzhou, China. E-mail: ywguo@cad.zju.edu.cn

Contract/grant sponsor: 973 Program of China; contract/grant number: 2002CB312102.

Contract/grant sponsor: NSFC; contract/grant numbers: 60033010; 60403038.

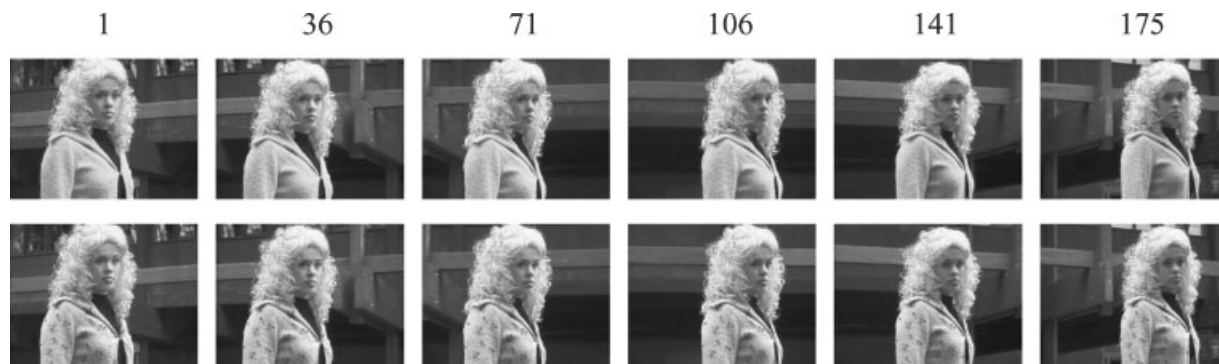


Figure 1. An example of our video retexturing. The top row lists 6 frames selected from a video clip with 175 frames, and the bottom row shows their corresponding retexturing results.

each concerned pixel and making use of its Y component which encodes the brightness information of each pixel on the interest region.

As for video retexturing, the main idea is to retexture a user specified key frame using the image retexturing algorithm, and then propagate iteratively the replaced texture adhering to the key frame onto other frames. Here, two key issues for video retexturing need to be addressed carefully: one is the texture drifting problem among frame sequences, and the other is the visibility shift between adjacent frames. We introduce a tracking algorithm of feature-points coupled with a mesh based optimization scheme to resolve the texture drifting problem efficiently. Meanwhile, graph cut algorithm is applied to a trimap along the interest region boundary to handle the case of visibility shift. Figure 1 demonstrates an example of using our video retexturing algorithm on a video sequence with 175 frames.

The remainder of this paper is organized as follows: Section ‘Related Work’ presents a brief overview of related previous work. Section ‘Image Retexturing’ describes the image retexturing of our approach including mesh generation, texture-coordinates calculation, lighting effects and experimental results. Section ‘Video Retexturing’ further addresses video retexturing using motion tracking and the graph cut algorithms, as well as the video retexturing results. Finally, the summary and future research is given in Section ‘Conclusions and Future Work’.

## Related Work

Texture mapping needs to set a correspondence between each point on the 2D texture image and that on the specified 3D surface. When a surface is displayed on

the screen, it must undergo a projective transformation. The resultant image is therefore a non-trivial mapping of the 2D manifold of the original surface depending on the shape of the surface. The problem becomes harder for retexturing because the 3D shape of the underlying surface is unknown. To simulate the non-linear mapping, Liu *et al.*<sup>3</sup> introduced a user-assisted adjustment on the regular grid of the real texture, and obtained a bijective mapping between the regular grid of the texture and the deformed grid of the surface image. Obviously, this method requires elaborate user interaction and is only suitable to regular textures. Assuming that the lighting satisfies Lambertian reflectance model, Fang *et al.*<sup>5</sup> recovered the geometry of the specified area using SFS approximation and derived a propagation rule to recalculate the mapping between the surface image and the new texture.

Extracting lighting information from real images is another challenge for retexturing, Tsin *et al.*<sup>2</sup> suggested a Bayesian framework based on certain lighting distribution model, which relies on the color observation at each pixel. Oh *et al.*<sup>6</sup> presented an algorithm for decoupling texture illuminance from the image by applying an image processing filter. They assumed that large scale luminance variations are due to the lighting, while small scale details are due to the texture. Welsh *et al.*<sup>7</sup> proposed a texture synthesis like algorithm for transferring color into grayscale image, the algorithm works on the  $\alpha\beta$  space and transfers  $\alpha, \beta$  components from the sample color image to the grayscale image. Its results automatically preserve the lighting effect of the original grayscale image.

Keeping good track of moving objects in video is a common goal in the vision and video editing field. Those pixel-wise, non-parametric algorithms (i.e., optical flow<sup>8</sup>) are robust to small-scale motion only. For

large-scale motions, tracking methods based on feature points and parametric models are more preferable. Feature based tracking can capture the motion of rotation, scaling etc.<sup>9</sup> Tracking features with underlying model can further reduce the risk of error, for example, Jin *et al.*<sup>10</sup> used a combined model of geometry and photometry to track features and detect outliers in video.

Visibility change may cause problems in tracking, as new part may appear and old part may disappear in a video sequence. Both Agarwala *et al.*<sup>11</sup> and Wang *et al.*<sup>12</sup> introduced an interpolation based, user assisted contour tracking framework for tracking interested part in video sequences. Chuang *et al.*<sup>13</sup> described a video-matting algorithm based on accurate tracking of the specified trimap. A trimap is a labeling image for which 0 stands for background, 1 stands for foreground and the rest is the unknown region to be labeled.

## Image Retexturing

In this section, we present a novel approach for image retexturing. Assume that a new texture with adequate size is given, as discussed above, the key issue here lies in how to construct a mapping from the new texture domain to the concerned region on the original image. To achieve this, we first generate an initial 2D mesh on the concerned region and let its shape conform with the underlying geometry of this region on the original image.

### Mesh Generation

Generating a proper initial mesh for video tracking has been addressed in the field of video compensation for compression. In Reference [14], nodes of the mesh are first extracted based on the image features, such as the spatial gradient, the displaced frame difference (DFD). A mesh is then built with these nodes, using the constrained Delaunay triangulation.

Here we propose a semi-automatic algorithm accounting for both the image feature of edges and gradients. It performs in three steps. The user first interactively outlines a boundary along the interested region using snakes. Then, the standard edge detection operator, for example, Canny operator, is applied inside the confined region, so some initial nodes are automatically generated on these detected edges. Other points can be introduced by the user when necessary. Delaunay triangulation algorithm is finally applied to yield an initial mesh  $\mathcal{M}$  over the region of interest.

## Texture Coordinates Calculation

The mapping from the new texture to the concerned region should be non-linear, to account for the distortion effect of the replaced texture induced by the underlying geometry. As reconstructing the geometry of the underlying surface with SFS and then performing texture mapping or synthesis can be unstable and costly, we calculate the texture coordinates for each pixel within the interest region directly by solving an energy minimization problem.

For the further description, we use the following notations. We use  $I(x, y)$  to denote the color intensity of a pixel  $(x, y)$  on the image,  $\nabla I(x, y) = (I_x, I_y)$  to represent the color gradient at  $(x, y)$ ,  $I_x = I(x, y) - I(x - 1, y)$  for the horizontal component of the gradient, and  $I_y = I(x, y) - I(x, y - 1)$  for the vertical component. Suppose that a new texture with adequate size is first laid on the concerned region without distortion. In this case, the initial texture coordinate for the pixel  $(x, y)$  within the concerned region is  $(u_0(x, y), v_0(x, y))$ . Similar to conventional methods, we use  $(u(x, y), v(x, y))$  to specify the final texture coordinates incurred by the texture distortion.

Our algorithm for computing the final texture coordinates is based on the assumption that, in the intensity field of the image, the depth variation of the local surface is considered proportional to the local gradient transition. In fact the mapping between a point on the new texture domain and a pixel within the concerned region is determined by concatenate transform. That is, the texture coordinates of adjacent pixels are inter-related, and there exists an offset between them. Actually, this offset can be conducted via the underlying local geometry.

Figure 2 illustrates the calculation of offset in 1D case. Let  $x$  and  $x - 1$  be two adjacent points, then according to above assumption, the variation of their underlying depths can be written as  $h(x) - h(x - 1) = k \cdot \nabla I(x)$ , where  $\nabla I(x)$  is the image gradient at the position  $x$  in 1D case, and  $k$  is the proportion derived from the assumption. This follows that the offset for the texture coordinates between  $x$  and  $x - 1$  should be the length of the green line in Figure 2:

$$\sqrt{1 + (k \cdot \nabla I(x))^2} \quad (1)$$

Similarly, as for the 2D case of texture coordinates, there exists the offsets for the texture coordinates

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61

62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122

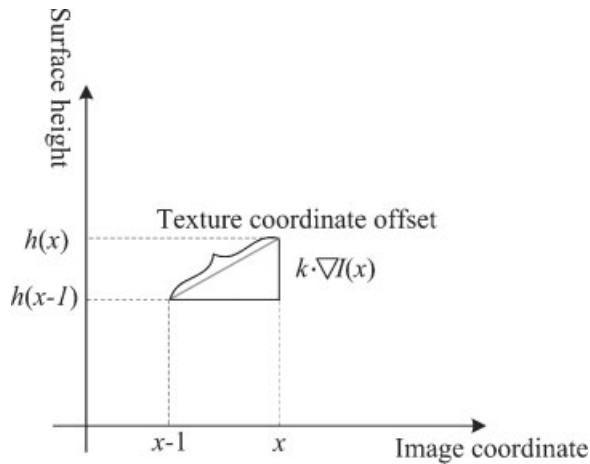


Figure 2. Texture coordinate offset for adjacent pixels in 1D case.

between the pixel  $(x, y)$  and its neighbors  $(x - 1, y)$  and  $(x, y - 1)$ , which can be expressed as follows:

$$u(x, y) - u(x - 1, y) = \sqrt{1 + (k_1 \cdot I_x)^2} \quad (2)$$

$$u(x, y) - u(x, y - 1) = \sqrt{1 + (k_2 \cdot I_y)^2} \quad (3)$$

$$v(x, y) - v(x, y - 1) = \sqrt{1 + (k_1 \cdot I_y)^2} \quad (4)$$

$$v(x, y) - v(x - 1, y) = \sqrt{1 + (k_2 \cdot I_x)^2} \quad (5)$$

where, both  $k_1$  and  $k_2$  are the proportions derived from the assumption. The horizontal component  $I_x$  of the image gradient impacts more greatly on the  $u$  offset than on the  $v$  offset, whereas the vertical component  $I_y$  impacts more greatly on the  $v$  offset than on the  $u$  offset, so we set  $k_1$  and  $k_2$  with different values 0.6 and 0.3, respectively in our experiments.

For the pixel lying on the edges of the generated mesh  $\mathcal{M}$ , consider the  $u$  component of its texture coordinate, which exists:

$$u(x, y)|_{(x, y) \in \partial \mathcal{M}} = u(x - 1, y) + \sqrt{1 + (k_1 \cdot I_x)^2} \quad (6)$$

Making an approximation that  $u(x - 1, y) = u_0(x - 1, y)$ , and considering  $u_0(x - 1, y) = u_0(x, y) - 1$ , the above equation is transformed into:

$$u(x, y)|_{(x, y) \in \partial \mathcal{M}} = u_0(x, y) + \sqrt{1 + (k_1 \cdot I_x)^2} - 1 \quad (7)$$

However, for the pixels lying in the triangles of  $\mathcal{M}$ , directly application of the offset equations (2)–(4) to them may result in a wried mapping. To reduce the error, we obtain their texture coordinates by solving the following energy minimization problem with respect to the  $u$  components (We compute their  $v$  components similarly):

$$\min_{u(x, y)} \int_{\mathcal{M}} |\nabla u(x, y) - D_u(x, y)|^2 \quad (8)$$

here  $\nabla u(x, y) = (u(x, y) - u(x - 1, y), u(x, y) - u(x, y - 1))$ , and  $D_u(x, y) = (\sqrt{1 + (k_1 \cdot I_x)^2}, \sqrt{1 + (k_2 \cdot I_y)^2})$ .

Minimizing equation (7), it can be easily converted into a set of Poisson equations with the form:

$$\Delta u(x, y) = \text{div} D_u(x, y) \quad (9)$$

in which  $\Delta$ ,  $\text{div}$  represent the Laplacian and divergence operator separately. The boundary conditions for above Poisson equations are determined by the  $u$  components of the texture coordinates of those pixels lying on the edges of  $\mathcal{M}$ , which are calculated using Equation (7). We adopt the conjugate gradients algorithm to solve them and it runs very fast.

As discussed above, the non-linear mapping between the point on the new texture domain and the pixel within the concerned region has been converted into solving a set of linear equations. Although some approximation is assumed, the presented algorithm is trivial to implement and presents satisfactory effects for most of our experiments.

## Lighting Effects

As we have obtained the non-linear correspondence between each pixel within the concerned region and that on the new texture, the next step is to map the new texture while preserving the lighting information encoded in the original image. Normally, the intensity of a texture can be regarded as the accumulated effect of the color and brightness. If the brightness information of the original texture can be extracted independently, fusing it with the new texture will resolve the problem of preserving the lighting. Fortunately, the  $YCbCr$  color space illuminates us.

$YCbCr$  is a well-known color space compliant to the digital video standard, where  $CbCr$  mainly represents the hue of each textured pixel and  $Y$  component encodes its brightness. We simply copy the  $CbCr$  components of the new texture to the target image at each concerned pixel during texture mapping, and use a weighted blending of the  $Y$  component of both the displayed

intensity of the concerned pixel on the original image and that of the corresponding sample point on the new texture plane. Define  $Y_t, Cb_t, Cr_t, Y_i, Cb_i, Cr_i$  and  $Y_r, Cb_r, Cr_r$  as the corresponding components of the new texture, intensity of the concerned pixel on the original image, and that of final result, respectively, the new intensity of the concerned pixel can be expressed as:

$$Cb_r = Cb_t \quad (10)$$

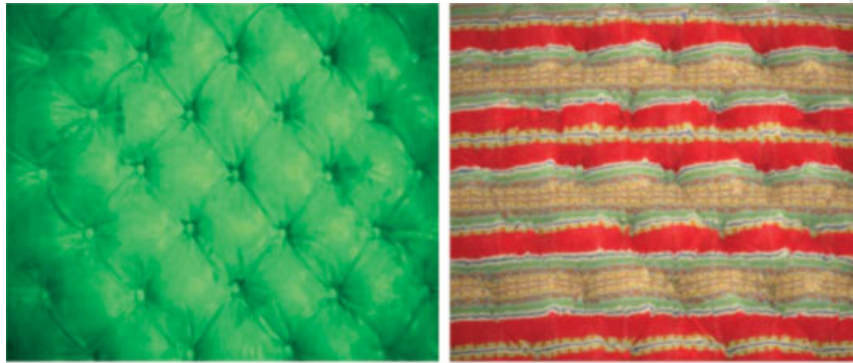
$$Cr_r = Cr_t \quad (11)$$

$$Y_r = m_t \times Y_t + (1 - m_t) \times Y_i \quad (12)$$

Here  $m_t$  stands for the weight balancing between the new texture and the brightness of the concerned pixel on the image, the bigger  $m_t$  is, the further the lighting of the retextured image resembles that of the original image. We empirically value it with 0.6 in our experiment.

### Results of Image Retexturing

Figure 3 demonstrates our experimental results of image retexturing. We can see that the effects preserve the shading information, and meanwhile yield the illusion



(a)



(b)

Figure 3. In (a) and (b), Left is the original images, right is the retexturing results.

62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122

that the new texture adheres to the underlying surface of the original image.

It takes 0.610 seconds, 0.896 seconds to retexture (a) and (b) respectively, in addition to the user interaction, under an environment of a Pentium IV 2.4 GHz PC with 1G main memory. Note that the woman body (b) is retextured by several textures with diverse patterns, and each pattern is mapped onto the concerned region independently. As our algorithm mainly focuses on the texture distortion induced by the underlying geometry, the global orientation of the replaced texture, for example, the orientation of the texture on the arm in (b), is achieved by user intervention.

## Video Retexturing

As video is composed of a sequence of frames, video retexturing is more complicated than image retexturing.

On the one hand, it is unpractical to retexture each frame of the video clip separately. A feasible way is to retexture a key frame selected from the video, for example, the first frame or the user-specified frame with maximum visibility, using the algorithm described in Section 'Image Retexturing', and then to iteratively propagate the replaced texture adhering to the key frame onto other frames. In this approach, the texture coordinate of the same point on adjacent frames must be accurately tracked. Although there are relevant algorithms in the field of computer vision, most of them are not robust enough to deal with all cases.

On the other hand, the problem of visibility shift must be taken into account, which occurs when part of the texture near the boundary shows up or disappears. Bradshaw<sup>15</sup> assumed that the motion discontinuity modeling can be decoupled from the affine motion estimation. With the same assumption, we introduce a trimap along the boundary based on mesh constraints, which is used to track the motion on the boundary more accurately and to efficiently handle the problem of the visibility shift.

In the following section, we describe our approaches for dealing with each of the above problems in detail.

## Motion Tracking

The generated mesh for the key frame facilitates the motion tracking. Starting from the key frame, positions of the mesh points are tracked frame by frame. The texture within each triangle of the current frame is

transferred to its corresponding triangle of the tracked frame. We calculate the corresponding mesh points in the next frame using the optical flow algorithm.<sup>8</sup> However, the tracking may not be accurate enough, some further constraints are thus made on the mesh to keep the coherence of the mesh topology.

There are three major motion tracking inconsistencies to be considered as bad tracking:

- *Matching inconsistency.* If the color difference between a mesh point in the current frame and that in the tracked frame is greater than a threshold, then the tracking point is considered as a mismatch.
- *Orientation inconsistency.* It is caused when any of the orientations of the triangles incident upon a mesh point is flipped.
- *Relative motion inconsistency.* It happens when the motion vector of a mesh point differs too much from its neighbors.

If any type of the above inconsistencies is detected for a mesh point, we call it an unstable point; otherwise a stable point. The position of the unstable point needs to be recalculated to compensate for the inconsistency. We adopt the algorithm similar to Reference [14] to interpolate the motion of the point from its neighbors, with an inverse distance weight scheme. Rather than including all the neighboring nodes, we only select those stable points. Assume  $p_0$  is an unstable point with neighboring nodes  $p_1, p_2, \dots, p_n$ ,  $(t_{xi}, t_{yi})$  denotes the motion vector for  $p_i$ , and  $d_i$  specifies the distance from  $p_i$  to  $p_0$ , then the new interpolated motion vector  $(t_{x0}, t_{y0})$  is expressed as:

$$t_{x0} = \frac{\sum_{i=1}^n \sigma(i)t_{xi}/d_i}{\sum_{i=1}^n \sigma(i)/d_i} \quad (13)$$

$$t_{y0} = \frac{\sum_{i=1}^n \sigma(i)t_{yi}/d_i}{\sum_{i=1}^n \sigma(i)/d_i} \quad (14)$$

with

$$\sigma(i) = \begin{cases} 1 & \text{if } p_i \text{ is a stable point} \\ 0 & \text{otherwise} \end{cases}$$

The second and third rows of Figure 5 compare the results (the third row) generated by our algorithm with those (the second row) obtained without mesh optimization. We can see that in the case when the material of the object is too smooth inside the concerned region, the optical flow algorithm fails frequently, while by using our mesh constraint algorithm, the errors can be mostly corrected.

### Visibility Shift

The unstable mesh points might be detected on the boundary of the concerned region, this is often caused by the pose changing of the body or camera movement. For example, if the interest object rotates in, the current boundary triangles should shrink in, part of texture shifts from visible to invisible. On the opposite, if the object rotates out, the boundary triangles extend out and some previously occluded part shifts from invisible to visible, this part must be retextured by fetching a texture patch from the boundary of the texture region occupied by the key frame on the initial texture plane. So the key issue is to find the new boundary for the concerned region on the current frame. We introduce a trimap Referece [13] along the boundary to handle this problem as shown in Figure 4(a). The unknown region  $T_U$  of the trimap is a group of triangles which is built by the mirrored reflection of those boundary triangles with

respect to the boundary edges. The interior region  $T_F$  of  $T_U$  belongs to the concerned region doubtless, whereas the exterior region  $T_B$  falls certainly into the background.

After constructing automatically the trimap along the boundary, the graph cut algorithm<sup>16</sup> is then applied to extracting the texture part in it. The energy function of graph cut is endowed with the form defined by<sup>17</sup> which captures the texture feature efficiently. Additionally, the foreground and background model is learnt only once in the key frame, therefore, the speed does not slow down.

Finally, as the texture part is extracted inside the trimap, mesh points are either appended or abandoned with respect to whether there is a triangle inside covering 90% of the texture and a remeshing operation is performed accordingly. Figure 4 illustrates our idea.

### Algorithm for Video Retexturing

The whole algorithm for retexturing on video is summarized as Table 1:

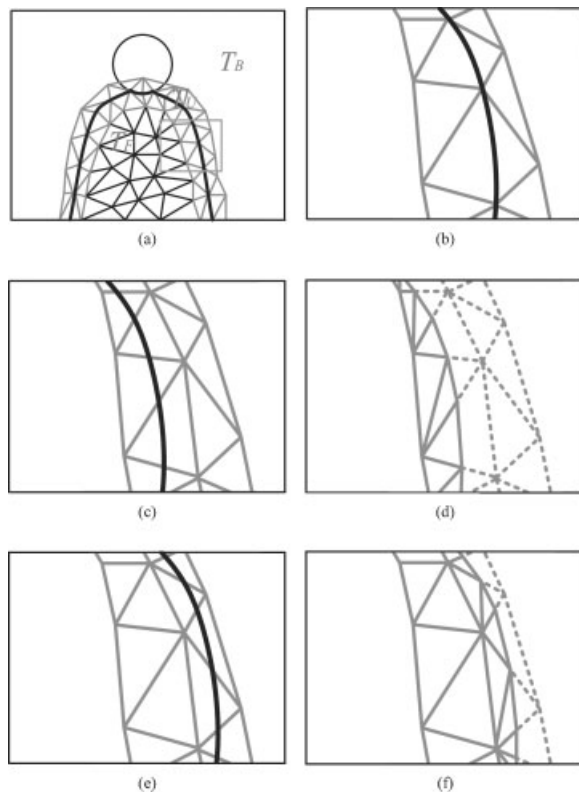


Figure 4. Illustration of trimap. (a) The current frame and constructed trimap. (b) A zoomed-in subregion of  $T_U$ . (c) & (e) The objects rotates in and out. (d) & (f) Apply graph cut and perform remeshing.

```

Select a key frame and triangulate the concerned region;
Perform image retexturing on the key frame;
Starting from the key frame, tracking and retexturing other frames backward and forward:
begin
  Calculate the corresponding mesh points in the subsequent frame using optical flow algorithm;
  for each obtained mesh point
    Detect whether it is an unstable point;
    if true
      Recalculate its position according to its neighboring points;
    end
  end
  Construct trimap along the boundary and apply graph cut algorithm to obtain the texture part in the trimap; append or delete points if necessary, and remesh the relevant region;
  Transfer the texture of the previous frame to the current frame;
  Incorporate the shading information encode by the current frame;
end
    
```

Table I. Algorithm for video retexturing

Note that, for the tracked frame, the final retextured effect should incorporate with the shading information encoded by this frame in the light of the algorithm described in *subsection 3.3*.

### Results on Video Retexturing

We have performed our experiments on an Intel Pentium IV 2.4GHz PC with 1G main memory under the Windows XP operating system. The timing for retexturing a video varies from several seconds to less than a minute, in addition to the user interaction, according to different video length and area of the concerned region.

Figure 5 shows the results of a video clip (123 frames) with and without mesh constraints, the 9th frame is selected as the key frame. We can see that in the case of retexturing a smooth object whose feature is less obvious, the optical flow algorithm fails frequently. Some nodes tracked without constraints may wave severely,

after a few frames. Using our mesh optimization algorithm, however, those unstable nodes are detected and their positions are adjusted.

Figure 6 demonstrates the capability of our approach in dealing with the visibility shift (this video contains 19 frames), the 14th frame is specified as the key frame. Although the arm of the woman moves significantly in the sequences, our trimap based algorithm accurately tracked this, the shading effect is also realistic. See the attached video for the above video retexturing examples.

### Conclusions and Future Work

We have proposed and developed a novel image/video retexturing approach that preserves the original shading effects with the unknown surface geometry and lighting conditions. The key issue of image retexturing



Figure 5. Part of the frames of our video retexturing results proving the validity of our mesh-based constraints. The second row illustrates the tracked mesh without constraints. The third row demonstrates the mesh with mesh-based constraints.

62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122



Figure 6. Several frames of a video retexturing result with visibility shift (the waving arm). Note that the shadow effect is realistic.

is how to construct the non-linear mapping between the new texture plane and the concerned region on the image. We successfully convert this problem into solving the linear Poisson equations based on some assumptions. Video retexturing is further accomplished through the stable tracking of the mesh points and the graph cut algorithm to treat with the visibility shift.

Although initial experiments have shown some encouraging results, our approach is not robust enough to handle all cases. The image retexturing mainly focuses on the texture distortion, and user interaction is needed for the creation of global orientation of the replaced texture. Our method of preserving lightness works for the input images with almost textureless surface, the method in Reference [3] may provide a more general solution. Our mesh optimization is mainly designed to handle large-scale inconsistencies accounting for the serious mesh distortion, slight waving of new texture between adjacent frames still exists in our experiment. The solution is to define more strict threshold value for unstable nodes detection, or to allow the users to adjust those unsatisfactory tracked nodes interactively. Besides, our algorithm has not yet solved the problem of occlusion and uncovering completely.

Future work includes exploring better mesh optimization using 'harder' constraints, and better treatment of occlusion and uncovering problems. Graphics hardware acceleration can also be embedded in our image/video retexturing approach.

## ACKNOWLEDGEMENTS

We wish to thank the anonymous reviewers for their valuable comments. Most of our video clips for experiments are fetched from the demo video of the paper.<sup>13</sup> The graph cut source code is obtained from the website of the author of Reference [18]. This project is supported in partial by 973 Program of China (No.2002CB312102) and NSFC (No. 60033010) & (No.60403038).

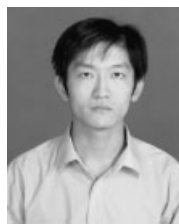
## References

1. Efros AA, Freeman WT. Image quilting for texture synthesis and transfer. In *Proceedings of SIGGRAPH 2001*, Los Angeles, CA, U.S.A., August 2001, pp. 341–346.
2. Tsin Y, Liu Y, Ramesh V. Texture replacement in real images. In *IEEE Conference on Computer Vision and Pattern Recognition 2001*, Kauai Marriott, Hawaii, U.S.A., July 2001, pp. 539–544.
3. Liu YX, Lin WC, Hays J. Near regular texture analysis and manipulation. *ACM Transactions on Graphics* 2004; **23**(3): 368–376.
4. Liu YX, Lin WC. *Deformable Texture: The Irregular-Regular-Irregular Cycle*. Technical Report CMU-RI-TR-03-26, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, 2003.
5. Fang H, Hart J. Textureshop: texture synthesis as a photograph editing tool. *ACM Transactions on Graphics* 2004; **23**(3): 354–359.
6. OH BM, Chen M, Dorsey J, Durand F. Image-based modeling and photo editing. In *Proceedings of SIGGRAPH 2001*, Los Angeles, CA, U.S.A., August 2001, pp. 433–442.

62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122

7. Welsh T, Ashikhmin M, Mueller K. Transferring color to greyscale images. *ACM Transactions on Graphics* 2002; **21**(3): 277–280.
8. Beauchemin SS, Barron JL. *The Computation of Optical Flow*. ACM Computing Surveys 1995; **27**(3): 433–467.
9. Shi JB, Tomasi C. Good features to track. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94)*, Seattle, U.S.A., June 1994, pp. 593–600.
10. Jin HL, Favaro P, Soatto S. Real-time feature tracking and outlier rejection with changes in illumination. In *The Eighth IEEE International Conference on Computer Vision*, Vancouver, Canada, July 2001, pp. 684–689.
11. Agarwala A, Hertzmann A, Salesin DH, Seitz SM. Key-frame-based tracking for rotoscoping and animation. *ACM Transactions on Graphics* 2004; **23**(3): 584–591.
12. Wang J, Xu YQ, Shum HY, Cohen MF. Video toning. *ACM Transactions on Graphics* 2004; **23**(3): 574–583.
13. Chuang YY, Agarwala A, Curless B, Salesin DH, Szeliski R. Video matting of complex scenes. *ACM Transactions on Graphics* 2002; **21**(3): 243–248.
14. Altunbasak Y, Tekalp AM. Closed-form connectivity-preserving solutions for motion compensation using 2-D meshes. *IEEE Transactions on Image Processing* 1997; **6**(9): 1255–1269.
15. Bradshaw DB. *Motion estimation and compensation of video sequences using affine transforms*. PhD thesis, University of Cambridge, 1999.
16. Boykov Y, Jolly MP. Interactive graph cuts for optimal boundary and region segmentation of objects in n-d images. In *The Eighth IEEE International Conference on Computer Vision*, Vancouver, Canada, July 2001, pp. 105–112.
17. Li Y, Sun J, Tang CK, Shum HY. Lazy snapping. *ACM Transactions on Graphics* 2004; **23**(3): 303–308.
18. Boykov Y, Kolmogorov V. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 2004; **26**(9): 1124–1137.

Authors' biographies:



**Yanwen Guo** is a Ph.D candidate in State Key Lab of CAD&CG, Zhejiang University, China. Now his research interests include geometry processing, video-based rendering, computer vision, etc.



**Jin Wang** is a researcher in State Key Lab of CAD&CG. Now his research interests include computer graphics & computer vision.



**Xiang Zeng** is a graduate student in Zhejiang University. He received his B.E. degree in Computer Science in 2004 from Zhejiang University. His research interests include vision-based graphics and computational complexity.



**Zhongyi Xie** received his B.E. degree in Computer Science in June 2005 from Zhejiang University. His research interest is geometry processing.

62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122



**Hanqiu Sun** is now an associate professor of Department of Computer Science & Engineering of CUHK. She received her Ph.D degree in Computer Science from University of Alberta, Canada. Her current research interests include interactive animations, virtual & augmented reality, hypermedia, computer-assisted surgery, etc.



**Qunsheng Peng** is a professor of State Key Lab of CAD&CG in Zhejiang University. He received his Ph.D in School of Computing studies from University of East Anglia, U.K, 1983. He is the Vice Chairman of VR & Visualization Professional Committee, China Computer Federation. His research interests include virtual reality, computer animation, Infrared Image Synthesis, etc.

UNCORRECTED PROOF

1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61

62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76  
77  
78  
79  
80  
81  
82  
83  
84  
85  
86  
87  
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113  
114  
115  
116  
117  
118  
119  
120  
121  
122