Interactive Texturing on Objects in Images via a Sketching Interface

Kwanrattana Songsathaporn[*]
The University of Tokyo

Henry Johan[†]
Nanyang Technological University

Tomoyuki Nishita[‡]
The University of Tokyo

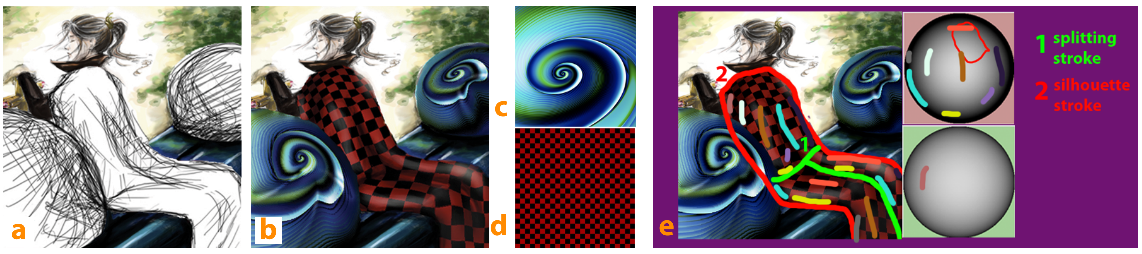

Figure 1: (a) An input image for texturing with our proposed system. (b) Result of texturing (a) with textures in (c,d). (c) An irregular texture. (d) A regular texture. (e) User specification for the texturing process of the cloak in (a).

Abstract

In this paper, we propose an interactive system for texturing objects in images without reconstructing the full 3D models. To make the texturing process easy for users, we emphasize on intuitiveness, and our system lets users perform texturing via a sketching interface. In addition, our system provides simple tools for specifying occlusion and perspective of the texturing objects. To texture with our system, first, with the sketching interface, users design the normal vector fields of objects in images that they wish to paste textures on. Next, to depict the shapes of the objects, the textures are deformed according to the normal vector fields, and pasted in the images over the objects. Once the textures are pasted on the images, users can manipulate (move, scale and rotate) the textures, and see the textures deformed interactively as if the textures were pasted on 3D models. The proposed system is tested with various kinds of objects in images, and we show that our system supports texturing with textures of regular, near-regular and irregular types.

CR Categories: I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction techniques; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

Keywords: image manipulation, 2D texture mapping, sketch-based user interface

1 Introduction

Because of the fact that textures can make a huge difference for both drawings or photographs, changing textures in photographs or adding textures in drawings are important image editing techniques.

[*]e-mail: kwsongsathaporn@nis-lab.is.s.u-tokyo.ac.jp
[†]e-mail: henryjohan@ntu.edu.sg
[‡]e-mail: nis@is.s.u-tokyo.ac.jp

The problem of texturing on objects in images is to paste textures on objects in images, which contain only 2D information, and the results give the impression that the textures are pasted on 3D objects as shown in Figure 1.

Several works such as [Liu et al. 2004; Fang and Hart 2004; Winnemöller et al. 2009] have been proposed for manipulation of textures in images, but they are either not interactive or require a lot of interaction from users. In addition, although textures can be classified into types according to the repetition pattern of the texels; regular, near-regular and irregular, several of the previous work are designed only for texturing with regular and near-regular textures. Moreover the previous work does not have a systematic algorithm for handling occlusion as shown in Figure 2(a), and perspective of objects as shown in Figures 2(c,d). Thus, texturing objects with these effects using a single texture requires both a lot of and effort from users.

Because images are the results of projecting 3D objects onto image planes, the objects in images contain only 2D information which is the positions on image planes in form of *xy* coordinates. Thus, the information in the third dimension of the objects, such as distances from the image planes and normal vector fields on the surfaces, are unknown. However, the information is required to depict the shapes of the objects during texturing. With only information of the positions on image planes, the results of texturing will always look like the textures are pasted on planes parallel to the image planes.

There are two alternative approaches to add the third dimensional information to the images. The first approach is using algorithms such as shape-from-shading algorithms [Frankot and Chellappa 1988; Kovesi 2005; Prados and Faugeras 2005] to automatically recover normal vector fields from the shading of the objects. However, shape-from-shading problem is well-known to be an ill-posed problem, and the algorithms used to solve the problem are constrained by many factors. Moreover, this approach requires the shading information of the objects which is usually not available in case of input images such as drawings as shown in Figure 1(a). Another approach is adopting a user interface. Since human is capable of perceiving the 3D information of the objects from a single image, many works [Wu et al. 2007; Zhang et al. 2001; Okabe et al. 2006; Kang 1998] have been proposed for adding depths and normal vector fields to the images via user interfaces.

Our purpose is to provide users an interactive system that supports texturing with images either drawings or photographs, and the types of the textures are not limited. Moreover, we want the system to be intuitive and requires minimum user interactions. Thus, we choose to approach the problem of obtaining the 3D information of the objects by letting users specify the normal vectors at some parts of the objects in images with a simple sketching interface. Then, we perform an optimization of an energy function to obtain normal vector fields of the objects. Finally, depth values and texture *uv* coordinates (see Figure 10(c)) are computed based on the normal vector field to deform and paste the textures over the images. We show that the *uv* coordinate calculation of our system let the system supports texturing without restriction on the types of textures, and our sketching interface is both intuitive and requires only small numbers of user interactions. In addition, our system provides simple tools to reduce users' burden in handling occlusion and perspective of the objects. An example result of our proposed system is shown in Figure 1. Figure 1(a) is the input of our system, the cloak and the two balls on the bench are textured with our proposed system, and the result is shown in Figure 1(b).

In this paper, we explain the related work in Section 2 and describe our proposed user interface in Section 3. The underlying algorithms are explained in Section 4. We show results of texturing using the proposed system and discuss its limitations in Section 5. Finally, we conclude this paper in Section 6.

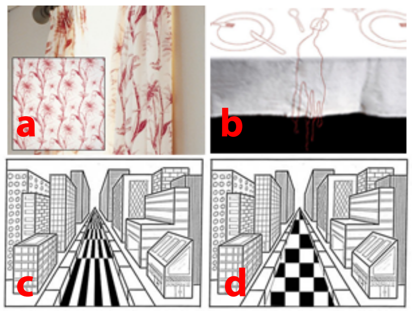

Figure 2: (a) Object with occlusion, in this example, curtains. (b) Object with a sharp edge. (c) Texture pasted taking perspective into account (with relative depth specification). (d) Texture pasted without considering perspective.

2 Previous Work

We review the previous work related to two problems; texturing in images and normal construction with user interface.

2.1 Texturing in Images

Many texture synthesis papers proposed methods to paste textures in specified regions in 2D images such as [Liang et al. 2001; Efros and Freeman 2001]. However, these proposed methods paste the results on a plane parallel to the image plane only. The shapes of the objects under the textures are not considered in these works.

[Liu et al. 2004] presented a system for replacing textures in 2D images that uses 3D meshes to create 3D effects. Initially, the vertices of the meshes are spread uniformly in a grid form, then the system lets users manually arrange the positions of each vertex over each texel of the replaced textures in the images. Next, the deformed meshes are used for calculating texture coordinates and shadings for new textures. Thus, the system not only does not support irregular texture but also requires a lot of effort from users.

[Fang and Hart 2004] proposed a system that attempts to reduce user interactions by letting users specify the regions for replacing textures, next a shape-from-shading computation is performed to recover the normal vector fields of the regions. The textures are then deformed according to the recovered normal vector field in order to depict the shapes of the underlying objects. The texture parameterization proposed in this system requires splitting of the recovered normal vector field into patches, then handles the discontinuities on the edges of each patch by a blending algorithm. Due to the blending algorithm, this system only supports regular and near-regular textures, and the texturing is not interactive.

The system proposed by [Winnemöller et al. 2009] lets users design the normal vector fields and the *uv* coordinate maps of texturing regions in images with diffusion curve (DC) [Orzan et al. 2008]. Using this method, the system can handle texturing on the specified objects in images including objects with occlusion. However, since several designs with DC must be provided by users, a complete texturing proposed in this paper requires many interactions from users. Moreover, this system does not consider depth, thus users must carefully and manually design the *uv* coordinate map in order to perform texturing on an object such as the one shown in Figures 2(c,d).

[Mihalik and Ďurikovič 2010] proposed a system to transfer material appearance between two objects in images based on the normal vector fields from a shade-from-shading algorithm. Although the purpose and approaches of their proposed system is different from ours, the idea of the problem is rather similar.

2.2 Normal Vector Construction with User Interface

Since we choose to construct the normal vector field of the objects using user interface, we also review some recent techniques.

The normal reconstruction via user interfaces were proposed by [Zhang et al. 2001; Okabe et al. 2006; Wu et al. 2007]. [Okabe et al. 2006] proposed a system that lets users specify normal vector at parts of the images through pen tablets whose measurement of angle between the pen and the pad is provided. The angles measured by the device is interpreted as normal vector of the parts of the objects, and by changing the angles, users can specify the normal vectors. However, controlling the device requires skills, and specifying normal vectors lying parallel to the image plane is difficult with this device.

In [Wu et al. 2007], users interact with the system via a simple sketching interface. Sparse normal vector field is constructed by copying normal vectors along the strokes drawn on an interface called shape palette to their corresponding strokes drawn on an identified region in an image. Then its dense normal vector field is obtained by optimizing an energy function. Copying normal vectors helps users who are not very skilled in controlling pen tablet and ease the difficulty of specifying normal vectors that lie parallel to the image plane.

3 Proposed User Interface for Texturing

Using our interface, to paste textures on the objects in images, users design and construct normal vector field of the texturing region first, then users can interactively adjust the textures over the region as if it were pasted on a 3D object. We provide users the following functions to adjust the textures; pinpoint specification, texture translation, texture rotation, texture scaling and relative depth specification. Thus, the proposed interface is divided into two parts which are interface for texturing region design and interface for texturing control.

3.1 User Interface for Texturing Region Design

Due to the intuitiveness, we choose to employ a sketching interface similar to shape palettes [Wu et al. 2007] for normal vector design. The user interface is shown in Figure 3. On the left side with an image of a woman in white dress is the *canvas* where input image is shown and users draw stroke upon. The two spheres in Figure 3 are shape palettes. For ease of specification, the upper is a convex shape palette or the outer surface of a hemisphere, and the lower is a concave shape palette or the inner surface of a hemisphere. The meaning of shape palettes are shown in Figure 4. Drawn strokes on the shape palette in Figure 4(a) hold meaning in 3D as shown in Figure 4(b). Specified normal vectors from drawing the blue stroke in Figure 4(a) are shown as normal vectors (orange arrows) in Figure 4(b).

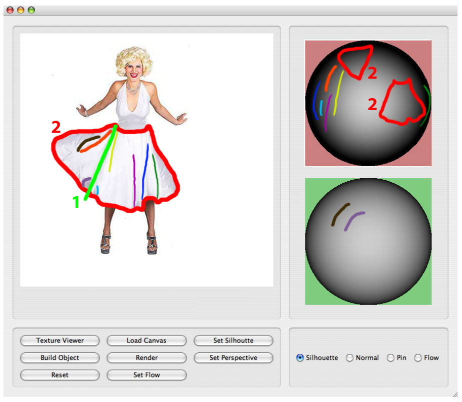

Figure 3: Specification of texturing region and its normal vector field. The red *"2"* loop on the canvas is the silhouette of the texturing region. Two spheres on the right side are shape palettes. The *"1"* line is a splitting stroke that splits the specified region into two subregions, and the initial normal vector field of each subregion is specified separately as shown on the upper shape palette (*"2"* loops). Other strokes are drawn to specify normal vectors. The same color strokes represent the matching strokes for normal transfer.

First, users draw the object silhouette on the canvas to identify the texturing region, then users can place, scale and rotate the silhouette strokes on a palette to specify the initial guess of the normal vector field of the region as shown in Figure 3 by the *"2"* stroke. This initial normal vector fields provide users with the rough pictures of the shapes being drawn. Second, users refine the normal vector field by drawing strokes on the canvas within the silhouette and their corresponding strokes on the palettes. The normal vector on the region along the strokes are modified to the normal vectors from their corresponding strokes on the palettes. The length of the strokes on the canvas and their corresponding strokes on the palettes are normalized, then normal vectors on the strokes on the palettes are directly mapped to the strokes on the canvas, which give a sparsely specified normal vector field. The final normal vector field is then obtained by optimizing an energy function [Wu et al. 2007] associated with the specified normal vector field.

The normal vector field produced by the interface proposed in [Wu et al. 2007] is assumed to be a normal vector field of a surface that contains no discontinuity. Thus, construction of objects with occlusion (Figure 2(a)) is not possible, and to specify sharp edges

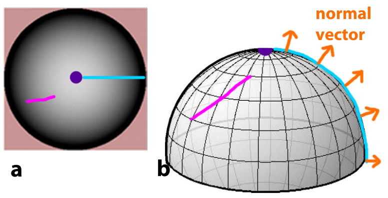

Figure 4: Normal vectors specification with shape palettes. (a) Convex shape palette in our user interface. (b) Imaginary 3D model of the convex shape palette.

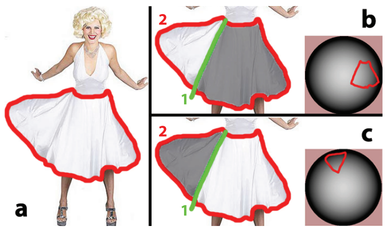

Figure 5: Splitting a region into two subregions with a splitting stroke (*"1"*). (a) Specification of one region. (b,c) Splitting the region in (a) into two subregions with a splitting stroke. Then, the initial normal vector field of each subregion is specified separately.

(Figure 2(b)) more than one strokes are necessary per one sharp edge as shown in Figure 14. To handle occlusion and reduce user interactions in sharp edge specification, we let users draw *splitting strokes* across the specified region as shown in Figure 5. The splitting strokes let users specified the position of discontinuity on the surface, thus the construction of normal vector field with our interface is now not limited to the assumption of surface with no discontinuity. Due to the produced discontinuity, we called the areas in the specified region that contain no discontinuity in them as *subregions*. So we can see that a splitting stroke splits the region it was drawn on into two subregions as shown in Figure 5. Note that the subregions are textured together at the same time as one object.

Specifying discontinuity is enough to reduce the number of strokes required for producing sharp edges as shown in Figure 14, however to texture object with occlusion using a single texture, only discontinuity is not enough to identify the occluded area on the texture. Thus, we let users to specify the distance into the screen between subregions after drawing a splitting stroke with an user interface as shown in Figure 6. This distance is used in texture coordinate calculation to approximate the portion of the texture that is occluded. Notice that when the distance is zero the texture on the two subregions will be continuous without any occluded part. However when the specified distance between the two subregions is not zero some parts of the texture is occluded (Figure 6(c)). Our system does not aim to accurately calculate the occlusion, but its goal is to provide users a tool that help them create the occlusion effect.

3.2 User Interface for Texturing Control

Besides constructing the normal vector field, users can also manipulate the textures on the texturing regions.

3.2.1 Pinpoint Specification

During texturing, users might want to specify a point of the texture to appear on a particular point of the specified region. Thus, our

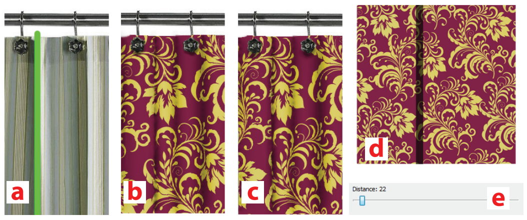

Figure 6: The effect of occlusion as the distance between the subregions is specified. (a) Specify two subregions by a splitting stroke. (b) Result of texturing when the distance between the subregions is zero. (c) Result of texturing when the distance between the subregions is greater than zero. (d) Black strip shows the occluded part on the original texture. (e) User interface for specifying the distance between the subregions.

user interface provides tool for specifying a matching point on the texture and the region called *pinpoint*. Specification of a pinpoint is done by clicking a point on the texture, and then another point on the region. After the texture is pasted on the object, the specified pinpoint serves as the center for moving, scaling and rotating the texture.

3.2.2 Relative Depth Specification

Figure 7 shows that only normal vectors are not sufficient for the calculation of the geodesic distance between points of the objects which is important for identifying texture scale as shown in Figure 2(d). Since the texture scale is nearly uniform on every point, the image does not give the impression that the texture is pasted on the road. To give the impression that the texture is pasted on the road, which is gradually further from the viewer, the texture scale must be gradually decreased.

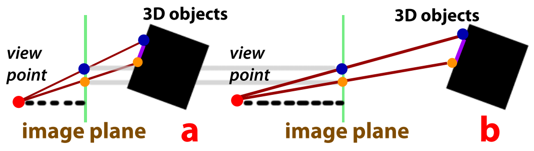

Figure 7: Perspective projection. Distance between the viewpoint and image plane, also the distance between the object and the image plane in (a) is smaller comparing those in (b). Normal vectors of the two points on the image planes are identical, but the distances between the points on the 3D objects are not the same.

The system proposed by [Horry et al. 1997] uses a spidery mesh interface to obtain other parameters concerning perspective projection in order to construct the 3D model of the scene from a single image. However, in our system the 3D model of the whole scene is not necessary since we want to modify only some parts of the images. We let users specify the relative depth to the image plane of two points on the objects. The first point is the furthest point on the object to the image plane, and another point is the nearest point on the object to the image plane as shown in Figure 8(a) with pink points. The user interface for specifying the relative depths are shown in Figure 8(b), and the effect of the specifications is shown in Figures 8(c,d).

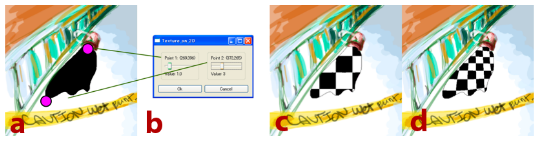

Figure 8: Specification of the relative depth. (a) The pink points estimated by the system show the furthest point and the nearest point to the image plane on the object. (b) Interface for specifying the depths. (c) The relative depths of the two points are very close to each other. (d) The difference of the relative depths of the two points are increased.

4 Algorithms

In this section, we describe the underlying algorithms of the proposed system. To create 3D effects to the texture, first we construct the normal vector fields of the objects from the users' specifications via the sketching interface (Section 4.1). Then, the furthest and nearest points on object are estimated, to let users specify the relative depth (Section 4.2). Finally, the texture coordinates are calculated according to the results of Section 4.1 and Section 4.2 (Section 4.3).

4.1 Normal Vector Field Construction

We first explain the idea of constructing the normal vector field by energy optimization as proposed in [Wu et al. 2007], then we propose a method to modify the optimization, so that it serves our system's goal better.

Normal vector at pixel i are represented as a vector (x_i, y_i, z_i), then it can be rewritten as $\frac{1}{\sqrt{p_i^2+q_i^2+1}}(p_i, q_i, 1)$ where $p_i = \frac{x_i}{z_i}$ and $q_i = \frac{y_i}{z_i}$. To construct a normal vector field, an energy function for estimating p_i and q_i of every pixels is shown as Equation 1. The goal of optimizing energy function (Equation 1) is to obtain a normal vector field that is closest to the sparsely specified normal vectors (the first part of Equation 1) while enforcing smoothness on the changing in orientation of the normal vectors (the second part of Equation 1), and keeping the curvature of the surface, which its normal vector field is the result of this optimization, minimal (the third part of Equation 1).

$$\begin{aligned} E(G) = \quad & \sum_{k\in S} \frac{(p_k - \bar{p_k})^2}{constant_1}) + \sum_{i\in G}\sum_{j\in N_i} \frac{(p_i - p_j)^2}{constant_2} \\ & + \sum_{i\in G} \frac{(\sum_{j\in N_i} p_i - ||N_i|| p_j)^2}{constant_3}. \end{aligned} \tag{1}$$

In Equation 1, $G = (p_i | i \in 1, \ldots, M)$, G is a set of p of every pixel in the specified region, and M is the total number of pixels in the region. N_i is a set of first order neighbours of pixel i. S is a set of pixels whose normal vectors are specified, and $\bar{p_k}$ are from the specification on shape palette. Equation 1 only shows the estimation of p of every pixel, q of every pixel is estimated in the same way by replacing p in the equation with q.

We can see that the optimization of Equation 1 gives a smooth normal vector field. However, sharp edges and occlusions on surfaces produces discontinuity, thus in order to create sharp edges and occlusions, we modify the optimization process such that the optimization results contain discontinuities based on the splitting strokes specified by users as in Equation 2.

$$E(G) = \sum_{k\in S}\frac{(p_k-\bar{p}_k)^2}{constant_1}+\sum_{i\in G}\sum_{j\in N_i\wedge j\in R_i}\frac{(p_i-p_j)^2}{constant_2} + \sum_{i\in G}\frac{(\sum_{j\in N_i\wedge j\in R_i} p_i-||N_i||p_j)^2}{constant_3}. \quad (2)$$

When users specified splitting strokes, the region G, which is initially one region, are divided into several subregions represented with R. R_i is a subregion that pixel i belongs to, where $\bigcup_{i\in G} R_i = G$, $R_i \neq \emptyset$ and any two subregions are mutually exclusive. This means that the smoothness and curvature minimization are enforced on each subregion separately, thus discontinuities are allowed along the borderlines between subregions, which are splitting strokes that users specified (Figure 5).

In our implementation, we solve the optimization problem with Gauss-Seidel solver on each subregion separately. This allow the optimization to converge faster, and increase parallelism of the program. The initial normal vector fields (Figure 3), which are specified by place, scale and rotate the silhouette stroke on the shape palettes, not only give users the general pictures of the shapes they are creating, but also serve as the initial guess of the optimization solution.

4.2 Estimation of Nearest and Furthest Points

Since the position on the image plane of the projected point on 3D object depends on $\frac{D_{view,object}}{D_{view,plane}}$ as shown in Equation 3, where $D_{view,object}$ is the distance between the view point of the perspective projection (we assume that the view point is at some distance from the center of the image plane in +z direction) and the projected point, and $D_{view,plane}$ is the distance between the view point and the image plane, to control the texture scale, users have to only specify the relative depth of the furthest point and the nearest point from image plane on the object. Then, the ratio $\frac{D_{view,object}}{D_{view,plane}}$ is automatically calculated, and the ratio is used in texture coordinate calculation to give the right texture scale.

$$\begin{aligned} x-coordinate_{object} &= x-coordinate_{plane}\times\frac{D_{view,object}}{D_{view,plane}}, \\ y-coordinate_{object} &= y-coordinate_{plane}\times\frac{D_{view,object}}{D_{view,plane}}, \end{aligned} \quad (3)$$

where *x-coordinate*$_{object}$ and *y-coordinate*$_{object}$ are the x and y components of the point on the object before the projection, and *x-coordinate*$_{plane}$ and *y-coordinate*$_{plane}$ are the x and y component of the projected result on the image plane.

The points on the object that are furthest and nearest to the image plane can be estimated from the specified normal vector field, thus to save users from the complication of specifying the points, our system estimates these two points on the objects for users automatically. Next, we describe how to estimate the pixels which are the results of projecting the nearest point and the furthest point from the image plane on the object.

Estimation of the relative depth between two adjacent pixels are shown in Figure 9(a). At first we do not know how much one pixel distance is before the projection, which is distance s in Figure 9(a), thus we assume s to be a constant value called $K_{default}$ which means the ratio of $\frac{D_{view,object}}{D_{view,plane}}$ is assumed to be $K_{default}$. Since we know the normal vectors of each pixel, we can fit in a curve, whose normal vectors at the two points are the known pixel's normal vectors.

Thus, the relative depth d can be calculated. Assume, the depth from the image plane at the left (purple) point is known, then we add the relative depth d (Figure 9(a)) to it to obtain the depth from the image plane at the right (orange) point.

We estimate the depth of every pixel in the region to obtain the pixels which represent the nearest point and the furthest point from the image plane on the object. The estimation is done subregion by subregion first, and finally the users' specified distance between subregions (Figure 6) are added to each pixel in the subregions to obtain the complete estimation.

The estimation in each subregion is done as following. First, the depth of the pixel in the middle of the subregion is set to zero. Then, from the middle pixel, the depth of the adjacent pixels are estimated with the above-mentioned method. The estimation is performed repeatedly from the pixels whose depths from image plane are already estimated to their adjacent pixels whose depths from the image plane are still unknown. The estimation stops when depth of every pixel in the subregion is estimated.

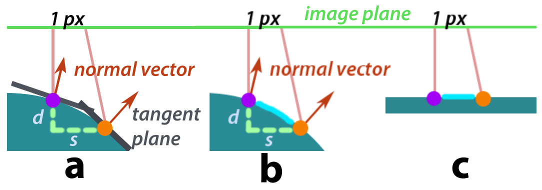

Figure 9: (a) Estimation of the relative depth between two pixels. (b,c) Difference in geodesic length. (b) The object is not a plane. (c) The object is a plane. After projection the left points (purple) and the right points (orange) in both (b) and (c) are separated by one pixel distance, however the geodesic distance between the left point and the right point in (b) is greater than in (c).

Once the pixels that represent the nearest point and the furthest point on the object are identified, users can specify depth from the image plane of the two points as stated in Section 3.2.2. The users' specified depths are then, used to correct the ratio $\frac{D_{view,object}}{D_{view,plane}}$ which we assumed to be $K_{default}$ at first. The correction of the ration is done as in Equation 4.

$$\frac{D_{view,object}}{D_{view,plane}} = K_{default}\frac{\bar{D}_{furthest}-\bar{D}_{nearest}}{D_{furthest}-D_{nearest}}, \quad (4)$$

where $\bar{D}_{furthest}$ and $\bar{D}_{nearest}$ are the depths of the furthest and nearest points specified by users with the interface in Figure 8, and $D_{furthest}$ and $D_{nearest}$ are the estimated depths of the furthest and nearest points by our method.

4.3 Texture Mapping with Designed Normal

As shown in Figures 9(b,c), although the distance between the purple points and the orange points in both Figure 9(b) and Figure 9(c) are the same, the geodesic length between the two points (purple and orange) on the surfaces are not the same, thus to paste the texture over the specified region such that the result gives the correct shape of the object, texture coordinates, or *uv* coordinates, must be calculated for texture mapping based on the obtained 3D information of the objects. In this section, we show how to calculate texture coordinates with the normal vector field and the ratio $\frac{D_{view,object}}{D_{view,plane}}$ obtained from the algorithms in the previous sections.

We can see that in order to depict the shape of the object, the correct distance on the texture between the pixels on the image is equal to the geodesic distance on the object. We calculate the geodesic distance on the object between two pixels as shown in Figure 10. Let the *uv* coordinate of p is known, and the *uv* coordinate of r is unknown. The positions on the 3D object (p' and r') of the two points (p and r) on the image plane are calculated by inverse perspective projection with ratio $\frac{D_{view,object}}{D_{view,plane}}$ as in Equation 3 (finding $x\text{-}coordinate_{object}$ and $y\text{-}coordinate_{object}$). We assume that the geodesic curve between the p' and r' is closed to a straight line. Since the *uv* coordinate of pixel p is known, if we know the relationship between the position of p' and r', then the *uv* coordinate of pixel r can be calculated by adding a vector that represent the relationship between the position of p' and r' (black dash vector in Figure 10(c)) to the *uv* coordinate at p as shown in Figure 10(c). Thus, we aim to find such vector.

As the geodesic curve between point p' and r' is assumed to be closed to a straight line, we calculate a vector from p' to r' to represent both the direction and the distance of r' relative to p' (yellow solid vector in Figures 10(a,b)). The obtain vector although represents the relationship between the position of p' and r' as we want, it cannot be added to the *uv* coordinate of pixel p directly since the vector is in 3D coordinate system. We have to transform the vector to a 2D coordinate system first, while the distance and the direction in 2D of r' relative to p' have to be preserved. We construct a 2D coordinate system around p' as a tangent plane at p'. The tangent plane is constructed with the normal vector of pixel p as the result of energy optimization in Section 4.1. If we projected the point r' to the tangent plane at p', we can see that the direction in 2D of r' relative to p' is preserved however the distance between p' and r' is not preserved. Instead of projection, we rotate the yellow vector to the tangent plane as shown in Figure 10 to preserve both the distance and the direction in 2D of r' relative to p'. By the rotation of the vector, the black dashed vector in Figures 10(a,b) is obtained. Finally, the black dashed vector, which is a 2D vector, is added to pixel p's *uv* coordinate on the texture to obtain the *uv* coordinate of r (left red point in Figure 10(c)).

The calculation of the *uv* coordinate of every pixel in the specified region is the extension of the above-mentioned method. First, since users specify a pinpoint to control the position in the specified region, we set the *uv* coordinate of the pinpoint to be the coordinate on the texture as specified. Then, the *uv* coordinates of the pinpoint's adjacent pixels are calculated with the above-mentioned method. The *uv* coordinate calculation is repeatedly performed from the calculated pixels to their adjacent pixels until *uv* coordinate of every pixel in the region is known. During the calculation, for a pixel whose uv coordinate is unknown, it is possible that more than one of its adjacent pixels are already calculated for their *uv* coordinates. In this case, we calculate the *uv* coordinate for the new pixel from each of these adjacent pixels separately, and then, the final *uv* coordinate is the weighted arithmetic mean of the coordinates. Since error piles up little by little as the calculation moves away from the pinpoint, we propose to use $\frac{1}{|p_j - p_{pinpoint}|}$ as the weight applied to the coordinate calculated from the adjacent pixel j. p_j is the position of pixel j and $p_{pinpoint}$ is the position of the pinpoint pixel in the specified region.

To handle occlusion, users specify splitting strokes and the distances between subregions. Thus, we need to consider the user-specified distance between subregions when we compute the geodesic distance between any two pixels across the splitting strokes where the distance between the subregions are not zero. Along a splitting stroke, we first find two pixels from each subregion whose difference of the relative depth (to the image plane) of their inverse projected points is the smallest. We call these two pixels *bridge* pixels of the subregions as shown in Figure 11, and call the difference of their relative depths $\Delta d_{smallest}$. We limit the above-mentioned *uv* calculation within the same subregion when the distance between the subregions are greater than zero. To continue the calculation from one subregion to the another, we, first, calculate the black dash vector from the bridge pixel of the former subregion to the bridge pixel of the later subregion, then set the vector's magnitude to $\Delta d_{smallest}$ before adding the vector to the *uv* coordinate of the bridge pixel of the former subregion. Then, the above-mentioned *uv* coordinate calculation is continued.

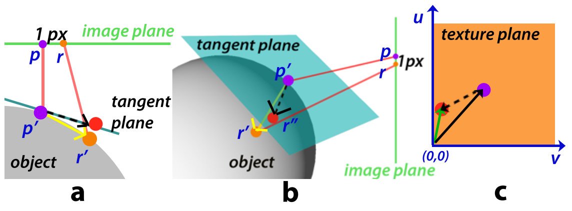

Figure 10: Calculation of the *uv* coordinate from one pixel on the image plane to another pixel. (a) Pixel p and pixel r on the image plane are inverse projected to find their positions on the object which is p' and r'. (b) Shows how to rotate the vector from p' to r' to the tangent plane at p' which is constructed with the normal vector at pixel p (result of optimization in Section 4.1). (c) the result of rotating the vector to the image plane is the dash vector. Since the *uv* coordinate of pixel p is known, we calculate the *uv* coordinate for pixel r by vector addition.

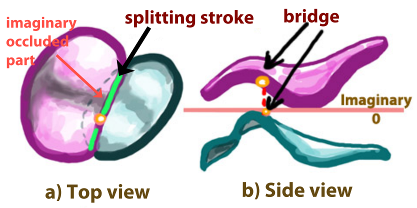

Figure 11: Bridge pixels along the splitting stroke. (a) Top view of the subregions. (b) Side view of the subregions. The upper subregion is the subregion in which the *uv* coordinate calculation is performed first, and the lower subregion is the next subregion in which the *uv* coordinate calculation will continue on.

5 Results

Using our proposed system, users can perform texturing interactively, and in this section we show the results of texturing various kind of objects in images. A result of pasting textures over a plain mask with our proposed system is shown in Figure 12. We demonstrate that texturing with our proposed system supports any types of textures as the texture in Figure 12(c) is an irregular texture, and the texture in Figure 12(f) is a regular texture.

Occlusion handling is demonstrated with an irregular texture as shown in Figure 13. In Figure 13(b) the texturing region is not split. However, in Figure 13(c) the texturing region is split by a splitting stroke into two subregions, and the distance between the two subregions are specified to be a value greater than zero. Thus, we can see that some portion of the texture are occluded and the result of texturing in Figure 13(c) is not continuous. The specification of strokes used to produce this result is shown in Figure 3.

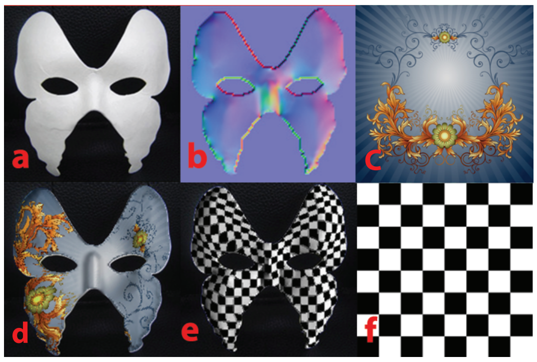

Figure 12: Results of texturing a plain mask with our proposed system. (a) Input image. (b) Calculated normal vector field from user specifications. (c) An irregular texture. (d) Texturing result with the texture in (c). (e) Texturing result with the texture in (f). (f) A regular texture.

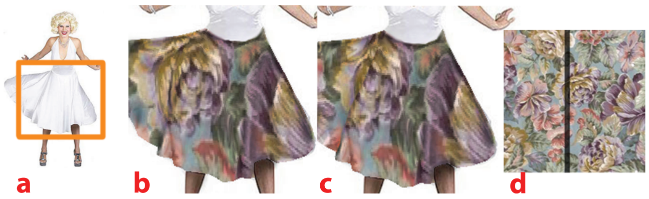

Figure 13: The effect of occlusion. (a) The input image. The results of texturing the skirt in the rectangle is shown in (b) and (c). (b) The texture is not occluded. (c) The texture is occluded. (d) The texture used for texturing of (b) and (c). The black strip shows the occluded part.

A result of pasting texture on an object with a sharp edge is shown in Figure 14. In Figures 14(e,f) we also show the comparison of specifying the normal vector to create the sharp edge with and without our splitting stroke. As we can see without the splitting stroke, users must specify two close strokes on both sides of the sharp edge, so that the result of the optimization contain sudden changes in the orientation of the normal vector. However, specification of the two strokes are not easy since the stroke must be as closed to each other as possible yet it must not cross each other. With a splitting stroke, users just draw the stroke across the region which take much less time and effort.

In Figure 1, the user textured three objects in the image; the cloak and the two balls on the bench. Two textures are used in this result, one is a regular texture (chessboard pattern) and another one is an irregular texture (blue spiral pattern). The two balls on each side of the bench is textured with one region, no splitting stroke is specified. Notice that the balls initially are smooth and round, however for artistic purpose the user carved ridges on them. The cloak is textured as one object with three subregions, two splitting strokes are specified as shown in Figure 1(e). Only the initial normal vector field of the uppermost subregion is specified, for the other two subregions, the normal vector field is specified purely with normal strokes. This shows that the optimization works correctly even without the prior knowledge of the shape of the object. However, the optimization converges faster with the specification of the initial normal vector field and rough picture of the shape helps users draw strokes easier. Notice that the texture scale at the lower part of the cloak is larger comparing to the texture scale around the shoulders. This is due to the relative depth specification to produce a result that is consistent to the perspective of the scene.

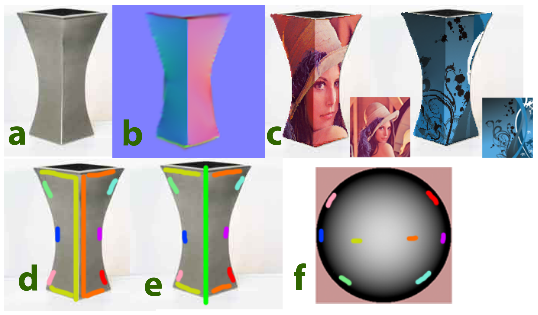

Figure 14: An example of texturing an object with a sharp edge. (a) Input image with a sharp edge. (b) Constructed normal vector field. (c) Results of texturing. (d) Specification of the normal vector field in (b) without a splitting stroke. (e) Specification with a splitting stroke. (f) Show the matching strokes on the shape palette.

We compare the texturing using our proposed system to the results produced by [Winnemöller et al. 2009]'s system (Figure 15) since their purpose is the most similar to ours. Although the system proposed by [Winnemöller et al. 2009] also includes the design of the texture, we compare only the texturing time of the two systems without taking their texture design time into account. Texturing time of Figures 15(b,c) with our system is 16 minutes per image while texturing of Figure 15(a) takes 30 minutes, and our texturing time of Figures 15(d,e) is 30 minutes while texturing time of Figure 15(c) is an hour in [Winnemöller et al. 2009]'s system, according to the paper. The time used to perform texturing for each result in this paper is shown in Table 1.

Table 1: Texturing time.

Figure	Texturing Time
Knight (Figure1)(c)	42 min
Mask (Figure 12)(d,e)	8 min (each)
White Dress (Figure 13)(b)	10 min
Vase (Figure 14)(c)	5 min (with splitting stroke) 6 min (without splitting stroke)
Fashion (Figure 15)(b,c)	16 min (each)
Sari (Figure 15)(e)	30 min

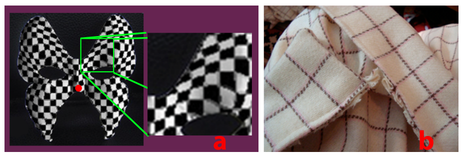

Figure 16: Limitations of the proposed system. (a) The texture are distorted at parts that are far away from the pinpoint (red point on the mask). (b) Image with complex occlusion.

Our proposed system has some limitations. The texture coordinate calculation described in Section 4.3 produces distortion at parts of the regions that are far from the pinpoint (Figure 16(a)). This is due to the error from the estimation of the black dash vector in Figure 10. As the calculation continues to propagate, the error from the calculation piles up little by little, and could be seen more clearly at parts that are far away from the pinpoint. In addition, our system has a limitation in terms of handling complex occlusion. For example, objects in Figure 16(b) which have many occlusions and some parts of the texture are highly occluded. To texture such objects, many splitting strokes must be drawn and distances between

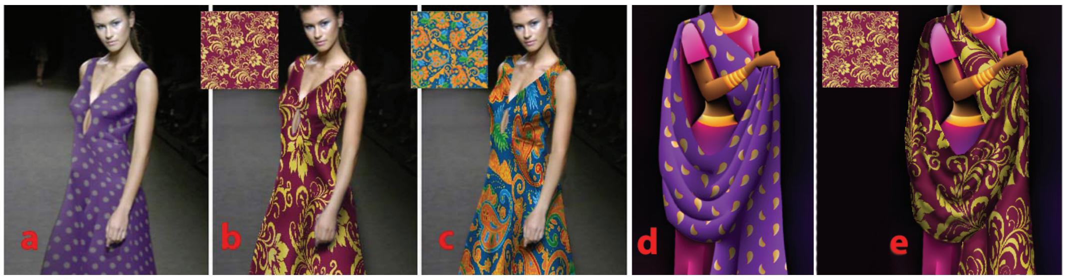

Figure 15: Comparison to texturing results in [Winnemöller et al. 2009]. (a,d) Result images from the paper; Fashion (a) and Sari (d). (b,c) Our results of texturing (a). (e) Our Result of texturing (d). On the top left corner of (b,c,e) are the textures used in the results.

subregions must be specified several time, or the texturing must be performed on several separate regions per one object.

6 Conclusions and Future Work

We have proposed an interactive system for pasting textures on objects in images. A sketching interface with normal specifying tool is employed to design normal vector fields. Then, texture mapping is performed based on the obtained normal vector field and the texture scale value are controlled according to the relative depth specification. Our system provides a tool for specifying pinpoints, which allows users to control the texture position on the object. Since our texture mapping can be performed in real-time, users can adjust (move, scale and rotate) the textures after they were pasted on the objects interactively. Our proposed system does not only provides a systematic relative depth control, but also provides users with an intuitive way to handle occlusion of objects in images. By specifying splitting strokes and distance between the regions, users can create an effect of occlusion easily. Moreover, with splitting strokes, users can specify sharp edges with less effort and less user interactions.

For future work we would like to develop a user interface that can texture an object with complex occlusion as shown in Figure 16(b) more systematically. We also intend to conduct a user study on human perception to determine the fitness of our proposed interface. Moreover, we would like to develop a texture calculation technique that does not cause distortion at parts of the regions that are far from the pinpoint.

References

EFROS, A. A., AND FREEMAN, W. T. 2001. Image quilting for texture synthesis and transfer. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM, SIGGRAPH '01, 341–346.

FANG, H., AND HART, J. C. 2004. Textureshop: texture synthesis as a photograph editing tool. *ACM Trans. Graph. 23*, 3, 354–359.

FRANKOT, R. T., AND CHELLAPPA, R. 1988. A method for enforcing integrability in shape from shading algorithms. *IEEE Trans. Pattern Anal. Mach. Intell. 10* (July), 439–451.

HORRY, Y., ANJYO, K.-I., AND ARAI, K. 1997. Tour into the picture: using a spidery mesh interface to make animation from a single image. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., SIGGRAPH '97, 225–232.

KANG, S. B., 1998. Depth painting for image-based rendering applications.

KOVESI, P. 2005. Shapelets correlated with surface normals produce surfaces. In *Proceedings of the Tenth IEEE International Conference on Computer Vision - Volume 2*, IEEE Computer Society, ICCV '05, 994–1001.

LIANG, L., LIU, C., XU, Y.-Q., GUO, B., AND SHUM, H.-Y. 2001. Real-time texture synthesis by patch-based sampling. *ACM Trans. Graph. 20*, 3, 127–150.

LIU, Y., LIN, W.-C., AND HAYS, J. 2004. Near-regular texture analysis and manipulation. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, ACM, 368–376.

MIHALIK, A., AND ĎURIKOVIČ, R. 2010. Appearance transformation of 3d objects depicted in images. *Journal of the Applied Mathematics, Statistics and Informatics (JAMSI) 6*, 1, 27–37.

OKABE, M., ZENG, G., MATSUSHITA, Y., IGARASHI, T., QUAN, L., AND YEUNG SHUM, H. 2006. Single-view relighting with normal map painting. In *In Proceedings of Pacific Graphics 2006*, 27–34.

ORZAN, A., BOUSSEAU, A., WINNEMÖLLER, H., BARLA, P., THOLLOT, J., AND SALESIN, D. 2008. Diffusion curves: A vector representation for smooth-shaded images. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2008)*, vol. 27, 92:1–92:8.

PRADOS, E., AND FAUGERAS, O. 2005. Shape from shading: A well-posed problem? In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, IEEE Computer Society, vol. 2 of *CVPR '05*, 870–877.

WINNEMÖLLER, H., ORZAN, A., BOISSIEUX, L., AND THOLLOT, J. 2009. Texture design and draping in 2d images. *Computer Graphics Forum (Proceedings of the Eurographics Symposium on Rendering 2009) 28*, 4, 1091–1099.

WU, T.-P., TANG, C.-K., BROWN, M. S., AND SHUM, H.-Y. 2007. Shapepalettes: interactive normal transfer via sketching. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, ACM, 44:1–44:5.

WU, T.-P., SUN, J., TANG, C.-K., AND SHUM, H.-Y. 2008. Interactive normal reconstruction from a single image. In *SIGGRAPH Asia '08: ACM SIGGRAPH Asia 2008 papers*, ACM, 1–9.

ZHANG, L., DUGAS-PHOCION, G., SAMSON, J.-S., AND SEITZ, S. M. 2001. Single view modeling of free-form scenes. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on 1*, 990.