https://doi.org/10.2352/ISSN.2470-1173.2017.10.IMAWM-177

Texture re-rendering tool for re-mixing indoor scene images

Tongyang Liu [a]***; Chun-Jung Tai*** [a]***; Fengqing Zhu*** [a]***; Judy Bagchi*** [b]***; Jan P. Allebach*** [a]
[a]***Purdue University, West Lafayette, IN, U.S.A.;*** [b]***Dzine Steps, Spring, TX, U.S.A.***

Abstract

We propose a novel tool for re-rendering objects in indoor scene images with new textures. It aims to address the problem of too much manual work of positioning and alignment when applying new texture onto an object surface in an indoor scene image. The algorithm of the tool is based on establishing 2D projective transformation between texture images and planar object surfaces in scene images. In order to find the transformation, we use a sampled rectangular texture pattern from a large synthesized planar texture and a planar quadrangle corresponding to object surface orientation estimation, which is generated by a geometric orientation hypothesis framework. The tool also puts effort in adjusting the scaling and reducing artifacts for re-rendered textures. We present the re-rendering results for ceilings, walls, floors, etc. that naturally correspond to room geometry layout.

1. Introduction

Digital imaging and rendering technology has brought us tremendous amount of benefits, one of which is the privilege of creating variable media contents through user customizations. One case that is receiving increasing interest these days, is in the application of customizing images on websites. Specifically, users can replace some part of images that is already on websites with new textures that they prefer, and then create new images by mixing up the replaced texture with original images. This kind of customization is known as image personalization [1]. The specific instance of image personalization to which our tool is targeted is the virtual customization of images taken from indoor scenes of residential structures, such as kitchen, living room or bedroom. This kind of customization lets users replace the original textures from object surfaces in the scene with preferred new textures, which is called texture re-rendering and image re-mixing. Figure 1 shows examples of texture re-rendering and image re-mixing. In the figure, the original floor textures and/or side wall textures from the indoor bathroom image are replaced with new types of textures, and then a re-mixed scene image is created. Figure 1b and Figure 1d separately show us that original appearance of floor and/or side wall is replaced by wooden and brick like textures. The highlighted green lines in Figure 1a illustrate the orientation of parallel lines within original floor texture. And the highlighted yellow lines in Figure 1b illustrate the orientation of parallel lines within replaced wooden-like texture. The brick-like texture in Figure 1d is clearly showing the orientation of parallel lines. By comparing the parallel line orientations in the re-rendered texture in Figure 1b and Figure 1d, we are able to see that the re-rendering result in Figure 1d looks more natural than that in Figure 1b, with respect to the room spatial layout. The position and orientation of the re-rendered texture in Figure 1b apparently needs to be manually adjusted before it appears to natrually corresponds to room spatial layout. Figure 1c shows a different type of texture replacement,

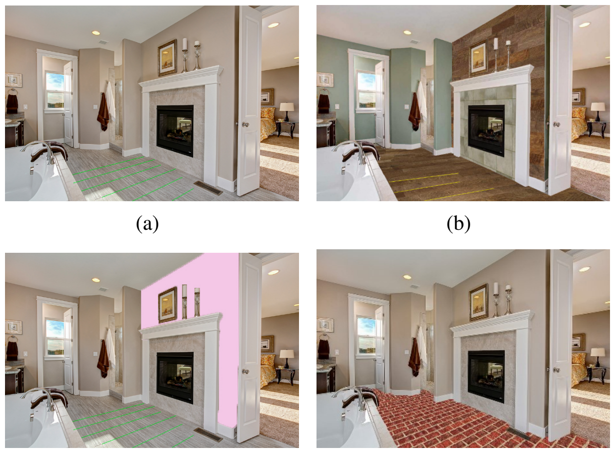


Figure 1. *Examples of texture re-rendering for the indoor bathroom image. Highlighted lines show the orientation of parallel lines in floor texture. (a) Original scene image taken from bathroom (b) Re-rendered floor and side wall with wooden texture. (c) Re-rendered side-wall with pure color texture. (d) Re-rendered floor with brick texture that naturally corresponds to geometry layout of the room.*

where the surface of side walls is replaced with pure color textures. This type of texture re-render, however, can only be suitable to the situation where original textures of the object surface have no complicated texture patterns. Currently, several commercial websites has been devoted to developing web-based interfaces for indoor scene image re-texturing, among which [6]-[9] are quite noticeable. [7] and [8] provide a functionality that is similar to the one shown in Figure 1c. [7] and [9] let users apply preferred textures in a virtual design environment. Although the visualization tool provides quite reasonable rendering result, it is hard to see the effect of newly applied texture in original scene images. As for [6], which provides a functionality as shown in Figure 1b, even though their tools are able to let users select various types of textures and render them in the original scene images, the result does not look as if it is naturally corresponding to the geometry orientation of the scene, thus an amount of manual adjustment afterwards is needed. In addition, the zoomed-in view of their rendered texture has poor quality. As a result, a tool is desired that has the following properties: firstly, it supports various types of textures; secondly, it is able to allow the re-rendered texture be directly mixed with the original scene image; thirdly, the re-rendering result natrually corresponds to room layout; finally, it is able to allow high-resolution rendering. Our proposed tool aims at these targets. In [1], the authors proposed a method for inserting text in images based on pinhole camera model with camera parameter estimation. However, text-insertion usually happens within a limited spatial range, thus the error for parameter estimation is not going to greatly affect the alignment and orientation of inserted text. In addition, their approach is based on straight line detection where the lines are either perpendicular or parallel with each other. This is not applicable to indoor scene images since straight lines in these images do not have to be aligned either horizontally or vertically to each other.

In this paper, we propose a texture re-rendering tool that is based on 2D projective geometry [3], and we are specially focused on establishing the 2D projective transformation from texture images to scene images. To begin with, we adopt a room layout estimation framework proposed in [2], which is based on line sweeping algorithm and convex edge detection to generate layout hypothesis. And then we apply Direct Linear Transform (DLT) algorithm to calculate the transformation matrix for texture mapping. The details are discussed in the rest of the paper.

2. Projective Geometry

In this section, we briefly introduce projective geometry, which is the study of 2D planar geometry, as the starting point for our texture re-rendering method. And in the following discussions, we denote 2D plane as $\mathbb{P}^2$.

2.1 Homogeneous representation for 2D points

The coordinate pair (x,y) can represent a point in 2D plane. Therefore we identify a plane as $\mathbb{R}^2$, and the coordinate pair (x,y) is then identified as a 2D vector, thus a plane can be considered as a vector space. A point $(x,y) \in \mathbb{R}^2$ lies on line (a,b,c) if and only if $ax+by+1=0$, which, in matrix form, can be written as $(x,y,1)\cdot(a,b,c)^\top = 1$. Here we can see that $(x,y) \in \mathbb{R}^2$ is represented as a 3D vector $(x,y,1)$. And since it is also true that $k(x,y,z)\cdot(a,b,c)^\top = 0$, we are able to draw the conclusion that (kx,ky,k) and $(x,y,1)$ represents the same point (x,y) in $\mathbb{P}^2$, thus the 3-vector representative of the form (x_1,x_2,x_3), where $x_3 \neq 0$, is the homogeneous representation for points $(x_1/x_3, x_2/x_3)$ in $\mathbb{R}^2$.

Now we discuss the case when $x_3 = 0$. The homogeneous representation for points in $\mathbb{R}^2$ with the form of $(x_1,x_2,0)$ is known as ideal points, representing the points at infinity. If there are two parallel lines (a,b,c) and (a,b,c') in $\mathbb{R}^3$, the ideal point $(-b,a,0)$ then lies on both of the two lines. Therefore, we can actually consider ideal point $(x_1,x_2,0)$ as the intersection of two parallel lines in 3D world, which apparently extends to infinity. Based on the above discussions, we are now able to introduce the concept of vanishing points, as illustrated in Figure 2. Note that vanishing points are indeed the projection of ideal points on $\mathbb{P}^2$.

2.2 Projective transformation between texture image and scene image

The core idea for our texture re-rendering tool is to estimate the transformation between pixels in the texture image and pixels in the scene image, which is called texture mapping. And it is actually a projective planar transformation that maps one set of points in $\mathbb{P}^2$ to another set of points in $\mathbb{P}^2$. Figure 3 shows one instance of planar transformation from plane π to plane π', and as is shown, point x is mapped to point x'. Now we are going to check the linearity of this transformation. As in Figure 3, $ABB'A'$ is a plane in 3D world that passes through center of projection

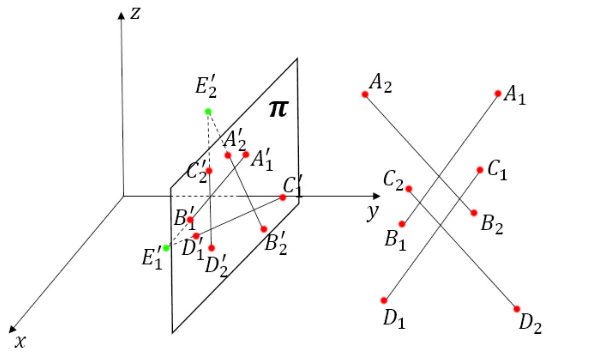


Figure 2. An illustration for projective geometry. Lines A_1B_1, A_2B_2, C_1D_1 and C_2D_2 are lines in 3D world space. A_1B_1 is parallel to C_1D_1 while A_2B_2 is parallel to C_2D_2, also A_1B_1 is perpendicular to A_2B_2. $A_1'B_1'$, $C_1'D_1'$, $A_2'B_2'$ and $C_2'D_2'$ are respective projections on plane π. Note that $A_1'B_1'$ and $C_1'D_1'$ intersect at point E_1', while $A_2'B_2'$ and $C_2'D_2'$ intersect at point E_2'. E_1' and E_2' are vanishing points in plane π. And these vanishing points correspond to orientations of lines in 3D world space.

O, plane π and plane π', which intersects plane π at line AB and plane π' at line $A'B'$. Furthermore, line $A'B'$ is the mapping of line AB from plane π to plane π', thus lines in $\mathbb{P}^2$ are mapped to lines in $\mathbb{P}^2$. Therefore, the projective transformation between two lines in $\mathbb{P}^2$ is a linear transformation on homogeneous 3D vectors, and it indeed can be represented by a non-singular 3 by 3 homogeneous matrix H [3]. In texture mapping, plane π in Figure 3 is the texture image, and $\boldsymbol{x}$ represents one pixel in the texture image. Meanwhile, plane π' is the scene image that is to be re-rendered, and $\boldsymbol{x'}$ represents one pixel in the scene image. Note that although $\boldsymbol{x}$ and $\boldsymbol{x'}$ are both points in $\mathbb{P}^2$, we are using homogeneous representations. Therefore $\boldsymbol{x}$ and $\boldsymbol{x'}$ are both 3D vectors. As a result, we denote $\boldsymbol{x}$ as $(x_1,x_2,x_3)^\top$ and vector $\boldsymbol{x'}$ as $(x_1',x_2',x_3')^\top$. Then the transformation between vector $\boldsymbol{x}$ and $\boldsymbol{x'}$ can be expressed as:

$$\begin{pmatrix} x_1' \\ x_2' \\ x_3' \end{pmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \tag{1}$$

Here we use h_{ij} to represent ith row and jth column element of homogeneous matrix H.

2.3 Calculating entries of homogeneous transformation matrix H

In Section 2.2, we discussed the homogeneous transformation matrix for mapping points between $\mathbb{P}^2$. The next step is how we calculate the elements in homogeneous matrix H. As is indicated in previous sections, we are given a set of points $\boldsymbol{x}$ represented by homogeneous 3D vectors in $\mathbb{P}^2$, and another set of points $\boldsymbol{x'}$ in $\mathbb{P}^2$, which are also represented by homogeneous 3D vectors. The homogeneous matrix H between $\mathbb{P}^2$ is then established such that $\boldsymbol{x'} = H\boldsymbol{x}$.

The first question is how many coordinate sets $(\boldsymbol{x},\boldsymbol{x'})$ are needed for H calculation. We notice that matrix H has 9 entries, but the number of degrees of freedom for projective transformation between $\mathbb{P}^2$, is indeed 8 [3]. The reason is that suppose we

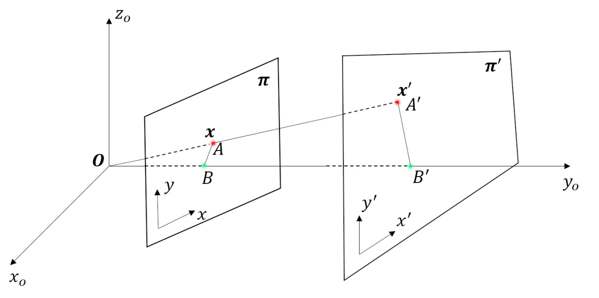


Figure 3. *Projective transformation between two planes. $x_0y_0z_0$ is world coordinate frame, and its origin O is the center of projection. $\boldsymbol{x}$ is a point in plane π, and $\boldsymbol{x'}$ is a point in plane π'. The mapping from plane π to π' is a linear mapping H between homogeneous coordinates such that $\boldsymbol{x'} = H\boldsymbol{x}$.*

have two homogeneous matrices H_1 and H_2, where $H_2 = aH_1$ and a is a scalar, and then we apply projective transformation on same homogeneous 3D vector $\boldsymbol{x}$. According to Equation 1, we will get $\boldsymbol{x'_1} = H_1\boldsymbol{x}$ and $\boldsymbol{x'_2} = H_2\boldsymbol{x}$. Since $H_2 = aH_1$, we have $\boldsymbol{x'_2} = a\boldsymbol{x'_1}$. According to Section 2.1, we already know the fact that homogeneous 3D vectors $\boldsymbol{x'_2}$ and $\boldsymbol{x'_1}$ correspond to the same point in $\mathbb{P}^2$. Therefore, we can draw the conclusion that homogeneous matrices H_1 and H_2 indeed are the same projective transformation between points in $\mathbb{P}^2$. Finally, we are able to draw the conclusion that homogeneous matrix H is defined up to a scalar, thus the 2D projective homogeneous transformation has 8 degrees of freedom.

Now we consider the way of calculating the entries' values for homogeneous transformation matrix H using Direct Linear Transformation (DLT) algorithm [3]. Actually, we can infer the fact from Section 2.1 that 3D homogeneous representation (x_1, x_2, x_3) for a point in $\mathbb{P}^2$ has 2 degrees of freedom. This is because x_3 just stands for an arbitrary none-zero ratio. Furthermore, we also infer from Section 2.2 that the degrees of freedom for point $\boldsymbol{x}$ must correspond to the degrees of freedom for its mapped point $H\boldsymbol{x}$. As a result, one coordinate mapping pair $(\boldsymbol{x}, H\boldsymbol{x})$ actually reduces the total degrees of freedom of the transform by 2. Therefore, in order to get efficient estimation for the entries of homogeneous transformation matrix H, it is necessary that we obtain 4 pairs of corresponding coordinate points to fully specify the matrix H. Now we set out to solve homogeneous matrix for our texture mapping method whereby it denotes the transformation from pixels of $\mathbb{P}^2$ in texture image to corresponding pixels of $\mathbb{P}^2$ in scene image. Note that here we are using homogeneous 3-vectors for pixels in $\mathbb{P}^2$, as is explained in Section 2.1. Therefore, pixel (x_1, x_2) in texture image is represented as:

$$\boldsymbol{x} = (x_1, x_2, 1)^\top$$

Similarly, pixel (x'_1, x'_2) in scene image is represented as:

$$\boldsymbol{x'} = (x'_1, x'_2, 1)^\top$$

Consequently, our texture mapping between $\boldsymbol{x}$ and $\boldsymbol{x'}$ is the homogeneous transformation matrix H such that

$$\begin{pmatrix} x'_1 \\ x'_2 \\ 1 \end{pmatrix} = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix} \tag{2}$$

where $h_i, i = 1, ..., 9$ stands for the entries of matrix H. Now we use the denotation $\boldsymbol{h}^i$ to represent row vectors for matrix H, that is:

$$\begin{aligned} \boldsymbol{h}^1 &= (h_1, h_2, h_3) \\ \boldsymbol{h}^2 &= (h_4, h_5, h_6) \\ \boldsymbol{h}^3 &= (h_7, h_8, h_9) \end{aligned}$$

Since $\boldsymbol{x'} = H\boldsymbol{x}$, we infer that homogeneous vector $\boldsymbol{x'}$ and $H\boldsymbol{x}$ are indeed in same direction. As a consequence, we know that:

$$\boldsymbol{x'} \times H\boldsymbol{x} = \mathbf{0} \tag{3}$$

where $\mathbf{0} = (0, 0, 0)^\top$, is the null 3D vector. And actually $H\boldsymbol{x}$ can be expanded as:

$$H\boldsymbol{x} = \begin{pmatrix} \boldsymbol{h}^1 \cdot \boldsymbol{x} \\ \boldsymbol{h}^2 \cdot \boldsymbol{x} \\ \boldsymbol{h}^3 \cdot \boldsymbol{x} \end{pmatrix}$$

Therefore

$$\begin{aligned} \boldsymbol{x'} \times H\boldsymbol{x} &= \begin{pmatrix} x'_1 \\ x'_2 \\ 1 \end{pmatrix} \times \begin{pmatrix} \boldsymbol{h}^1 \cdot \boldsymbol{x} \\ \boldsymbol{h}^2 \cdot \boldsymbol{x} \\ \boldsymbol{h}^3 \cdot \boldsymbol{x} \end{pmatrix} \\ &= \begin{pmatrix} x'_2\boldsymbol{h}^3 \cdot \boldsymbol{x} - \boldsymbol{h}^2 \cdot \boldsymbol{x} \\ \boldsymbol{h}^1 \cdot \boldsymbol{x} - x'_1\boldsymbol{h}^3 \cdot \boldsymbol{x} \\ x'_1\boldsymbol{h}^2 \cdot \boldsymbol{x} - x'_2\boldsymbol{h}^1 \cdot \boldsymbol{x} \end{pmatrix} = \mathbf{0} \end{aligned} \tag{4}$$

Since:

$$\boldsymbol{h}^i \cdot \boldsymbol{x} = \boldsymbol{x}^\top \cdot \boldsymbol{h}^{i\top}$$

expressions in Equation 4 may actually be re-written as:

$$\begin{aligned} \boldsymbol{x'} \times H\boldsymbol{x} &= \begin{bmatrix} \mathbf{0}^\top & -\boldsymbol{x}^\top & x'_2\boldsymbol{x}^\top \\ \boldsymbol{x}^\top & \mathbf{0}^\top & -x'_1\boldsymbol{x}^\top \\ -x'_2\boldsymbol{x}^\top & x'_1\boldsymbol{x}^\top & \mathbf{0}^\top \end{bmatrix} \begin{pmatrix} \boldsymbol{h}^{1\top} \\ \boldsymbol{h}^{2\top} \\ \boldsymbol{h}^{3\top} \end{pmatrix} \\ &= L\boldsymbol{h} = \mathbf{0} \end{aligned} \tag{5}$$

Note that here we have:

$$\boldsymbol{h} = (\boldsymbol{h}^1, \boldsymbol{h}^2, \boldsymbol{h}^3)$$

is a 9-vector that is constituted by 9 entries of homogeneous matrix H. Furthermore, we notice the fact that if we left multiply matrix L by two elementary matrices in Equation 5:

$$L' = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x'_1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & x'_2 & 1 \end{bmatrix} L$$

we will end up getting

$$\begin{aligned} L'\boldsymbol{h} &= \begin{bmatrix} \mathbf{0}^\top & -\boldsymbol{x}^\top & x'_2\boldsymbol{x}^\top \\ \boldsymbol{x}^\top & \mathbf{0}^\top & -x'_1\boldsymbol{x}^\top \\ \mathbf{0}^\top & \mathbf{0}^\top & \mathbf{0}^\top \end{bmatrix} \begin{pmatrix} \boldsymbol{h}^{1\top} \\ \boldsymbol{h}^{2\top} \\ \boldsymbol{h}^{3\top} \end{pmatrix} \\ &= \mathbf{0} \end{aligned} \tag{6}$$

As a consequence, we are able to reach the conclusion that Equation 5 actually give rise to 2 linearly-independent constraints for solving homogeneous transformation matrix H. At the beginning of Section 2.3 we explained that in order to efficiently calculate H, it is necessary that four corresponding coordinate pairs $(\boldsymbol{x}_i, \boldsymbol{x}'_i)$ are given, where $i = 1,2,3,4$. And following the denotations in Section 2.3, we have $\boldsymbol{x}'_i = (x'_{1i}, x'_{2i}, 1)$, where (x'_{1i}, x'_{2i}), $i = 1,2,3,4$ are the pixel values in scene image, and $\boldsymbol{x}_i = (x_{1i}, x_{2i}, 1)$, where (x_{1i}, x_{2i}), $i = 1,2,3,4$ are the pixel values in texture image. Consequently we will have 8 linear equations. And based on the result in Equation 6, the four equation sets for solving texture mapping matrix H are expressed as:

$$\begin{bmatrix} \mathbf{0}^\top & -\boldsymbol{x}_i^\top & x'_{2i}\boldsymbol{x}_i^\top \\ \boldsymbol{x}_i^\top & \mathbf{0}^\top & -x'_{1i}\boldsymbol{x}_i^\top \end{bmatrix} \begin{pmatrix} \boldsymbol{h}^{1\top} \\ \boldsymbol{h}^{2\top} \\ \boldsymbol{h}^{3\top} \end{pmatrix} = \mathbf{0} \tag{7}$$

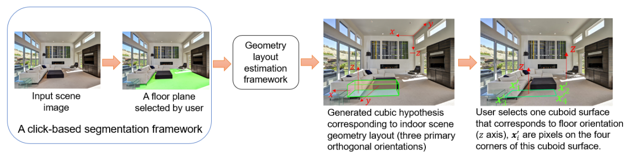
where $i = 1,2,3,4$. Note that matrix H is defined up to a scalar, and in homogeneous 3D vector (x_1, x_2, x_3), component x_3 is actually the scaling factor, thus without loss of generality, we set $h_9 = 1$, in other words, $\boldsymbol{h}^3 = (h_7, h_8, 1)$. Consequently, as denoted in Equation 2, we are able to calculate the remaining 8 entries h_j, where $j = 1,2,...,8$ in texture mapping matrix H.

3. Methodology

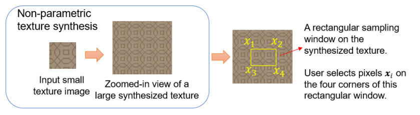
3.1 User selection of 4 coordinates sets

In Section 2, we proved that in order to get the texture mapping matrix H, we need four corresponding pixel sets, separately from the texture image and the indoor scene image. Also we noticed in Figure 2 that different points A, B, C, D on $\mathbb{P}^2$ may lead to different vanishing points, thus corresponding to different orientations in 3D world. As a result, an arbitrary user selection of the four coordinate sets from images may not lead to a proper texture mapping matrix that corresponds to indoor room geometry layout. Consequently, judging the room layout while selecting points from images is challenging for normal users, which is not what we want for a user-friendly interface. Hence in our method, we introduce a semi-automatic way to make it easy for users to select the reasonable coordinate sets from scene images and texture images.

The flow chart in Figure 4a illustrates an instance of how the user selects four pixels from the scene image. Firstly, the user


Input scene image
A floor plane selected by user
A click-based segmentation framework
Geometry layout estimation framework
x
y
z
Generated cubic hypothesis corresponding to indoor scene geometry layout (three primary orthogonal orientations)
z
x'_1
x'_2
x'_3
x'_4
User selects one cuboid surface that corresponds to floor orientation (z axis), $\boldsymbol{x}'_i$ are pixels on the four corners of this cuboid surface.

(a)


Non-parametric texture synthesis
Input small texture image
Zoomed-in view of a large synthesized texture
x_1
x_2
x_3
x_4
A rectangular sampling window on the synthesized texture.
User selects pixels $\boldsymbol{x}_i$ on the four corners of this rectangular window.

(b)

Figure 4. *Flow charts that illustrate how the user selects four corresponding pixels from the scene image and the texture image. (a) From left to right: A click-based segmentation framework [4], a geometry layout estimation framework [2], a generated cubic hypothesis from scene image and a user-selected cuboid surface. (b) From left to right: A non-parametric texture synthesis framework [5] and a rectangular window in texture image.*

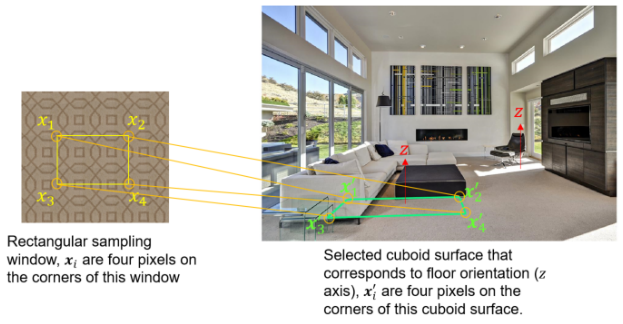

x_1
x_2
x_3
x_4
Rectangular sampling window, $\boldsymbol{x}_i$ are four pixels on the corners of this window
z
z
x'_1
x'_2
x'_3
x'_4
Selected cuboid surface that corresponds to floor orientation (z axis), $\boldsymbol{x}'_i$ are four pixels on the corners of this cuboid surface.

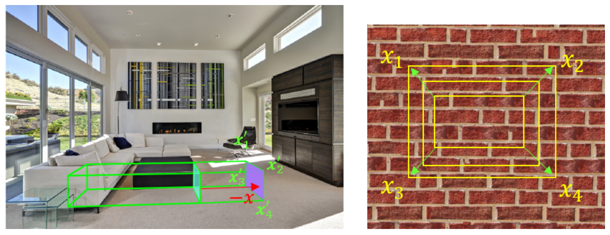
Figure 5. *Generating texture mapping matrix H, given four corresponding points in a texture image and a scene image. Through H, pixels at $\boldsymbol{x}_i$ are mapped to pixels at $\boldsymbol{x}'_i$, for $i = 1,2,3,4$.*

interacts with our tool through a click-based segmentation framework for indoor scene images [4], which is capable of generating a mask for the components in indoor scene images onto which users would like to re-render new textures. In this case, a floor plane is picked out. Secondly, an embedded geometry layout estimation framework [2] is able to autonomously generate spatial estimation of room layout by presenting users parametrized cubic hypotheses that correspond to 3 primary orthogonal orientations of the room scene, as shown in Figure 4. In addition, each of the cuboid surfaces has the same orientation as a rectangular plane in world scene. In next step, the users is able to interact with our tool by selecting one of the surfaces from the generated cubic. Note that in Figure 4, the bottom surface of the cubic is selected, since it is corresponding to z axis in room scene configuration, which matches the orientation of the selected floor plane. Once the cubic surface is selected, acquiring the four coordinates from scene image is straight-forward. They are indeed the pixel values on the four corners of the selected cuboid surface.

Next we are going to discuss how the user selects four pixels from the texture image. We already know that four pixels selected from the scene image corresponds to the orientation of a rectangular plane in the world scene. Thus it is quite straight-forward that a rectangular window on a texture image can be used to determine four pixels on the texture image. The flow chart in Figure 4b illustrates an instance of selecting four pixels from a texture image. In the backend, an embedded texture synthesis framework [5] takes a small texture pattern as input and presents the user with a synthesized large texture pattern. Then the user is able to choose a rectangular window on the synthesized texture. Finally, the pixel values on the four corners of the rectangular window act as the four coordinates from texture image. After all four corresponding coordinate sets

$$(\boldsymbol{x}_i,\ \boldsymbol{x}'_i),\ i = 1,2,3,4$$

where $\boldsymbol{x}_i$ is the homogeneous 3-vector representing pixels in the texture image and $\boldsymbol{x}'_i$ is the homogeneous 3-vector representing pixels in the scene image, are picked, by using Equation 7, we are able to generate texture mapping matrix H (as shown in Figure 5). And then the tool is able to re-render the selected component (as in Figure 4a) with the new texture (as in Figure 4b).



(a) (b)

Figure 6. *Proposed methods for refining texture re-rendering results: (a) Cuboid surface shifting, the green surface is before shifting, and the purple surface is after shifting. The shifting direction, in this case, is along $-x$ axis. (b) Rectangular sampling window re-sizing, three yellow rectangular shapes corresponds to three sampling windows in texture image with different sizes. In this case, the window is expanding.*

3.2 Re-rendering adjustments

In this section, we talk about adjusting re-rendering result. Using the texture mapping matrix generated from Section 3.1 can be useful enough if we only consider the re-rendering result correspondence with the indoor scene (room) geometry layout. Nevertheless, some re-rendering results may be visually over-sized and thus unfitted for some certain scene configurations (as shown in Figure 8b). Thus, there is necessity for our tool to propose approaches to refine texture re-rendering results through user-interactions. And Figure 6 shows our proposed two methods, which include cubic surface shifting in the scene image and sampling window re-sizing in the texture image. As is seen in Figure 6a, the cubic surface before and after shifting has the same orientation (both perpendicular to x axis). Meanwhile, the sampling windows with different sizes obviously have same orientation, since the texture image itself is a plane. Consequently, through the adjustment, the scaling of the re-rendering result is changing, while the orientation of the re-rendered texture, with respect to room geometry layout, is not changing. Hence users can interact with our texture re-rendering tool in either of the two ways to get a satisfying scaling effect for the re-rendering result.

3.3 High-resolution rendering

A critical case that we are especially interested in is to increase the re-rendering quality, while not increasing time consumption too much. Before our method, the contradictory point is that on one hand, for low resolution images, it is fast to generate cubic hypothesis (see Table 1), but the rendering quality is poor. The re-rendered texture may come with a bundle of artifacts which are certainly unwelcome by users. On the other hand, for high-resolution images, although it is time-consuming to generate cubic hypothesis (see Table 1), the rendering quality is satisfying. In practical cases, users can neither accept that it takes too long to get a result, nor that the rendering quality is of low-quality. Thus we propose a matrix scaling method that enables high-resolution images to utilize the texture mapping matrices generated from low-resolution images, for the purpose of re-rendering textures on high-resolution images while reduce time consumption for matrices generation.

Let us now think of a pair of two images. One image has a resolution $m \times n$ pixels and the other image has a higher resolution $am \times an$ pixels ($a>1$). Therefore, one pixel (x_0, y_0) in the low-resolution image corresponds to a pixel (ax_0, ay_0) in the high-resolution image. According to Section 2.1, the 3-vector homogeneous representations for (x_0, y_0) and (ax_0, ay_0) are $\boldsymbol{x}'_{Low\_Res} = (x_0, y_0, 1)$ and $\boldsymbol{x}'_{High\_Res} = (ax_0, ay_0, 1)$, respectively. It is easy to see that the transformation between these 2 points is

$$\begin{pmatrix} ax_0 \\ ay_0 \\ 1 \end{pmatrix} = \begin{bmatrix} a & 0 & 0 \\ 0 & a & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x_0 \\ y_0 \\ 1 \end{pmatrix} \tag{8}$$

Now, our method takes the low resolution image as input, and then follows the routine described in Section 3.1 to generate the texture mapping matrix from a texture image to the low-resolution image, which is denoted as H_{Low_Res}. Then, in order to get the texture mapping matrix from the same texture image to the high-resolution image, denoted as H_{High_Res}, we follow the discussion in Section 2.2:

$$\boldsymbol{x}'_{Low\_Res} = H_{Low\_Res}\boldsymbol{x}$$

and

$$\boldsymbol{x}'_{High\_Res} = H_{High\_Res}\boldsymbol{x}$$

where $\boldsymbol{x}$ is 3-vector homogeneous representation for pixels in texture image. By comparing Equation 8 and the above two relationships, we have

$$H_{High\_Res}\boldsymbol{x} = \begin{bmatrix} a & 0 & 0 \\ 0 & a & 0 \\ 0 & 0 & 1 \end{bmatrix} H_{Low\_Res}\boldsymbol{x} \tag{9}$$

We know that $\boldsymbol{x}$ actually corresponds to a pixel in texture image, thus it cannot be a null vector. Thus, a straight-forward solution for Equation 9 is

$$H_{High\_Res} = \begin{bmatrix} a & 0 & 0 \\ 0 & a & 0 \\ 0 & 0 & 1 \end{bmatrix} H_{Low\_Res} \tag{10}$$

From Equation 10 we know that H_{High_Res} can actually be acquired from H_{Low_Res}, just by left-multiplying H_{Low_Res} with a scaling matrix. In this way, we can avoid estimating H_{High_Res} in a time-consuming way.

4. Experimental Results

In order to test the efficiency of our texture re-rendering tool, we firstly did texture re-rendering on different room configurations to check whether the newly applied texture corresponds to room spatial layout. And then we tested the efficiency of our tool with respect to re-rendering adjustment. Finally, we compared the results between low-resolution and high-resolution images.

Figure 7 shows the indoor scene image re-mixing results for different room configurations. The appearance of re-rendered textures in Figures 7a, 7b, 7c, 7e, 7g and 7i look plausible, because they not only visually correspond to room spatial layout, but also present a harmonious visual effect when their scaling in sizes are compared with the objects nearby. The appearance of re-rendered textures in Figures 7d, 7f, 7h look reasonable in the sense of correspondence with room geometry orientation. However, the results in these figures are basically generated from raw texture mapping

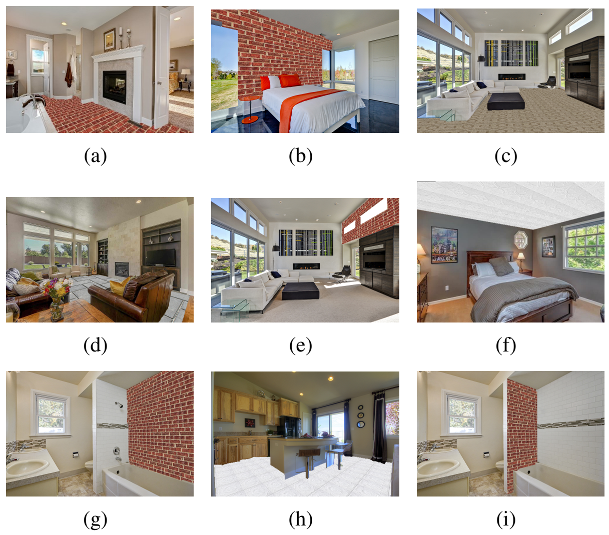


Figure 7. *Texture re-rendering and indoor scene image re-mixing results in various room layout configurations. (a) re-rendered floor with bricks texture. (b) re-rendered side wall with bricks texture. (c) re-rendered floor with carpet-like texture. (d) re-rendered floor with stone texture. (e) re-rendered accent wall with bricks texture. (f) re-rendered ceiling with tiles texture. (g) re-rendered accent wall with bricks texture. (h) re-rendered floor with tile texture. (i) re-rendered side wall with bricks texture.*

matrix with less refinement and user adjustments. Therefore they present us an inharmonious visual effect when their sizes are compared with that of nearby objects. Nevertheless, scaling can be refined and adjusted through our user-interface that is introduced in Section 3.2.

Figure 8 actually shows how user interaction affects the scaling of re-rendering results. As the highlighted brick tiles suggest, re-rendering result in Figure 8b is visually inharmonious because the brick tiles are oversized when compared with cabinet top edges. However, as is observed in Figures 8c and 8d, with the changes in cuboid surface shifting distance and the sampling window sizes, the scaling of the brick tiles also changes accordingly. We now discuss this in detail. As suggested by highlighted brick tiles, in Figure 8c, around 4 brick tiles are aligned with cabinet top edge. And in Figure 8d, around 6 brick tiles are aligned with cabinet top edge. So re-rendering results in Figures 8c and 8d are actually showing brick tile with different scaling in size. In addition, we notice the fact from Figure 8 that all the 3 different re-rendering results have the same geometry orientation, which visually corresponds to room layout.

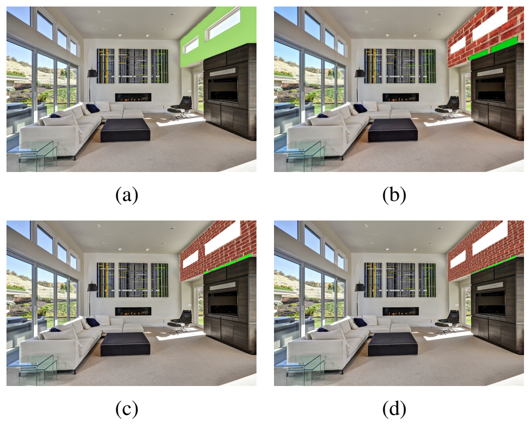


Figure 8. *Illustration of re-rendering adjustment, the adjustments are introduced in Figure 6. We denote cuboid surface shifting distance as d in pixel, and sampling window size as win_size in pixels $\times$ pixels. Green blocks are highlighted brick tiles. (a) User-selected component to be re-rendered. (b) Re-rendering result when $d = 100$ and $win\_size = 800 \times 800$. (c) Re-rendering result when $d = 100$ and $win\_size = 1400 \times 1400$. (d) Re-rendering result when $d = 120$ and $win\_size = 1800 \times 1800$.*

Table I: Time consumption comparison for different approaches to generate texture mapping matrix, experiment performed in MATLAB®

Image in Figure 9	Image size (in pixels × pixels)	Approach for generating texture mapping matrix	Time consumed for generating texture mapping matrix
a	1024 × 683	Geometry layout estimation [2]	~5 secs
b	3861 × 2574	Geometry layout estimation [2]	>20 mins
c	3861 × 2574	Our proposed matrix scaling method	~5 secs

Finally, we discuss the result for our proposed high-resolution re-rendering method. In the process of generating the results in Figure 9, we are actually using the same synthesized texture image with resolution 5000×5000 pixels for both low-resolution scene images and high-resolution scene images. But as is shown in Figure 9, the quality of re-rendered texture on the low-resolution image is poor – a lot of artifacts are visually notice-

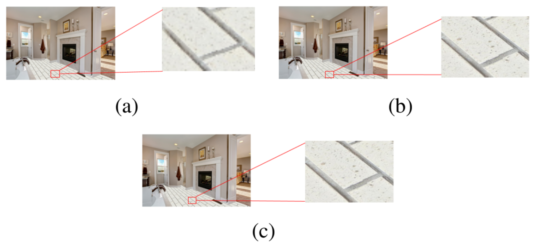


Figure 9. *High-resolution texture re-rendering and Low-resolution re-rendering with zoomed-in view. (a) Re-rendering result on 1024×683 pixels scene image. (b) and (c) Re-rendering result on 3861×2574 pixels scene image. (b) and (c) are different in the way of generating texture mapping matrix (see Table 1).*

able; while the re-rendering quality on the high-resolution image is plausible. The critical point here for re-rendering on the high-resolution image is that, although results in Figures 9b and 9c look very similar, the time consumed for generating these two results are quite different. As shown in Table 1, in order to generate the texture mapping matrix, time consumed for Figure 9b is more than 20 minutes while for Figure 9c, it is around 5 seconds. The results indicates that using our proposed matrix scaling method greatly enhances the efficiency of re-rendering on high-resolution images.

5. Conclusion

In the paper we propose a novel tool for texture re-rendering and indoor scene image re-mixing. The tool is able to autonomously align newly rendered texture with room spatial layout. Also it provides users an interface for adjusting the scaling of the re-rendered texture in order to match with practical object sizes. In addition, our tool is capable of proving high-quality rendering result efficiently. In the future, we plan to research options to autonomously enable more realistic outcomes: one is to autonomously adjust the scaling of re-rendered texture, the other is to add light and shadow effects.

References

[1] H. Ding, R. Bala, Z. Fan, R. Eschbach, J. Allebach, and C. Bouman, "Semi-automatic object geometry estimation for image personalization," in *Computational Imaging*, SPIE vol. 7533, C. Bouman, I. Pollak, and P. Wolfe, Eds., San Jose, CA, 17-21 January 2010 (2010).

[2] D. C. Lee, A. Gupta, M. Hebert and T. Kanade, "Estimating Spatial Layout of Rooms using Volumetric Reasoning about Objects and Surfaces," *Advances in Neural Information Processing Systems 23*, 1288–1296 (2010).

[3] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, Cambridge University Press, ISBN: 0521540518 (1998).

[4] C. Tai, T. Liu, J. Bagchi, F. Zhu, and J. Allebach, "Click-Based Interactive Segmentation with Graph Cut," *Imaging and Multimedia Analytics in a Web and Mobile World 2017*, (Part of IS&T Electronic Imaging 2017), J. Allebach, Z. Fan, and Q. Lin, Eds., San Francisco, CA, 29 January -2 February 2017 (2017).

[5] K. Ziga, J. Bagchi, J. Allebach, and F. Zhu, "Non-Parametric Texture Synthesis Using Texture Classification," *Computational Imaging XIV*, (Part of IS&T Electronic Imaging 2017), C. Bouman and R. Stevenson, Eds., San Francisco, CA, 29 January -2 February 2017 (2017).

[6] Dzine Steps: Create, Customize, Collaborate! `http://dzinesteps.com/`

[7] Zillow Digs: Find inspiration for your home project. `http://www.zillow.com/digs/`

[8] Sherwin-Williams ColorSnap Visualizer `https://www.sherwin-williams.com/visualizer#/active/default`

[9] M S International, Inc. Virtual Kitchen Designer `https://www.msistone.com/virtual-kitchen-designer/#`

Author Biographies

Tongyang Liu is currently a PhD student working under Professor Jan P. Allebach in the School of Electrical and Computer Engineering at Purdue University. His research is focused on color image processing, imaging and printing. He obtained his bachelor's degree from University of Science and Technology of China (2014).

Chun-Jung Tai received her BS in Electrical and Computer Engineering from the National Chiao Tung University, Hsinchiu, Taiwan (2011). She is currently pursuing her PhD degree in Electrical and Computer Engineering at Purdue University, West Lafayette, IN, USA. She works with DzineSteps on her current research, where she completed the work reported in this paper.

Fengqing Zhu is an Assistant Professor of Electrical and Computer Engineering at Purdue University, West Lafayette, IN. Dr. Zhu received her Ph.D. in Electrical and Computer Engineering from Purdue University in 2011. Prior to joining Purdue in 2015, she was a Staff Researcher at Huawei Technologies (USA), where she received a Huawei Certification of Recognition for Core Technology Contribution in 2012. Her research interests include Image processing and analysis, video compression, computer vision and computational photography.

Judy Bagchi is founder and CEO of Dzine Steps, a cloud software provider to home builders and independent design centers nationwide. Prior to this Judy held Research & Development Management roles at Hewlett-Packard and Nortel. She is an industry veteran with more than 20 years of experience in the entire business value chain. She has a keen interest in and has led and participated in various activities supporting Women in Technology.

Jan P. Allebach is Hewlett-Packard Distinguished Professor of Electrical and Computer Engineering at Purdue University. Allebach is a Fellow of the IEEE, the National Academy of Inventors, the Society for Imaging Science and Technology (IS&T), and SPIE. He was named Electronic Imaging Scientist of the Year by IS&T and SPIE, and was named Honorary Member of IS&T, the highest award that IS&T bestows. He has received the IEEE Daniel E. Noble Award, and is a member of the National Academy of Engineering. Most recently, he received the Edwin H. Land Medal from IS&T and the Optical Society of America. He currently serves as an IEEE Signal Processing Society Distinguished Lecturer (2016-2017).