Image-Based Modeling and Photo Editing

Byong Mok Oh Max Chen Julie Dorsey Frédo Durand

Laboratory for Computer Science
Massachusetts Institute of Technology [*]

Abstract

We present an image-based modeling and editing system that takes a single photo as input. We represent a scene as a layered collection of depth images, where each pixel encodes both color and depth. Starting from an input image, we employ a suite of user-assisted techniques, based on a painting metaphor, to assign depths and extract layers. We introduce two specific editing operations. The first, a "clone brushing tool," permits the distortion-free copying of parts of a picture, by using a parameterization optimization technique. The second, a "texture-illuminance decoupling filter," discounts the effect of illumination on uniformly textured areas, by decoupling large- and small-scale features via bilateral filtering. Our system enables editing from different viewpoints, extracting and grouping of image-based objects, and modifying the shape, color, and illumination of these objects.

1 Introduction

Despite recent advances in photogrammetry and 3D scanning technology, creating photorealistic 3D models remains a tedious and time consuming task. Many real-world objects, such as trees or people, have complex shapes that cannot easily be described by the polygonal representations commonly used in computer graphics. Image-based representations, which use photographs as a starting point, are becoming increasingly popular because they allow users to explore objects and scenes captured from the real world. While considerable attention has been devoted to using photographs to build 3D models, or to rendering new views from photographs, little work has been done to address the problem of manipulating or modifying these representations. This paper describes an interactive modeling and editing system that uses an image-based representation for the entire 3D authoring process. It takes a single photograph as input, provides tools to extract layers and assign depths, and facilitates various editing operations, such as painting, copy-pasting, and relighting.

Our work was inspired, in part, by the simplicity and versatility of popular photo-editing packages, such as *Adobe Photoshop*. Such tools afford a powerful means of altering the appearance of an image via simple and intuitive editing operations. A photo-montage, where the color of objects has been changed, people have been removed, added or duplicated, still remains convincing and fully "photorealistic." The process involves almost no automation and is entirely driven by the user. However, because of this absence of automation, the user has direct access to the image data, both conceptually and practically. A series of specialized interactive tools complement one another. Unfortunately, the lack of 3D information sometimes imposes restrictions or makes editing more tedious. In this work, we overcome some of these limitations and introduce two new tools that take advantage of the 3D information: a new "distortion-free clone brush" and a "texture-illuminance decoupling filter."

Clone brushing (a.k.a. rubberstamping) is one of the most powerful tools for the seamless alteration of pictures. It interactively copies a region of the image using a brush interface. It is often used to remove undesirable portions of an image, such as blemishes or distracting objects in the background. The user chooses a source region of the image and then paints over the destination region using a brush that copies from the source to the destination region. However, clone brushing has its limitations when object shape or perspective causes texture foreshortening: Only parts of the image with similar orientation and distance can be clone brushed. Artifacts also appear when the intensity of the target and source regions do not match.

The existing illumination also limits image editing. Lighting design can be done by painting the effects of new light sources using semi-transparent layers. However, discounting the existing illumination is often difficult. Painting "negative" light usually results in artifacts, especially at shadow boundaries. This affects copy-pasting between images with different illumination conditions, relighting applications and, as mentioned above, clone brushing.

In this paper, we extend photo editing to 3D. We describe a system for interactively editing an image-based scene represented as a layered collection of depth images, where a pixel encodes both color and depth. Our system provides the means to change scene structure, appearance, and illumination via a simple collection of editing operations, which overcome a number of limitations of 2D photo editing.

Many processes involving the editing of real images, for aesthetic, design or illustration purposes, can benefit from a system such as ours: designing a new building in an existing context, changing the layout and lighting of a room, designing a virtual TV set from a real location, or producing special effects. Some of these applications already obtain impressive results with 2D image-editing tools by segmenting the image into *layers* to permit separate editing of different entities. A particular case is cell animation, which can take immediate and great advantage of our system.

We will see that once this segmentation is performed, an image-based representation can be efficiently built, relying on the ability of the user to infer the spatial organization of the scene depicted in the image. By incorporating depth, powerful additional editing is possible, as well as changing the camera viewpoint (Fig. 1).

One of the major advantages of image-based representations is their ability to represent arbitrary geometry. Our system can be used without any editing, simply to perform 3D navigation inside a 2D image, in the spirit of the *Tour into the Picture* system [HAA97], but with no restriction on the scene geometry.

[*] http://graphics.lcs.mit.edu/

ACM SIGGRAPH 2001, 12-17 August 2001, Los Angeles, CA, USA
© 2001 ACM 1-58113-374-X/01/08...$5.00

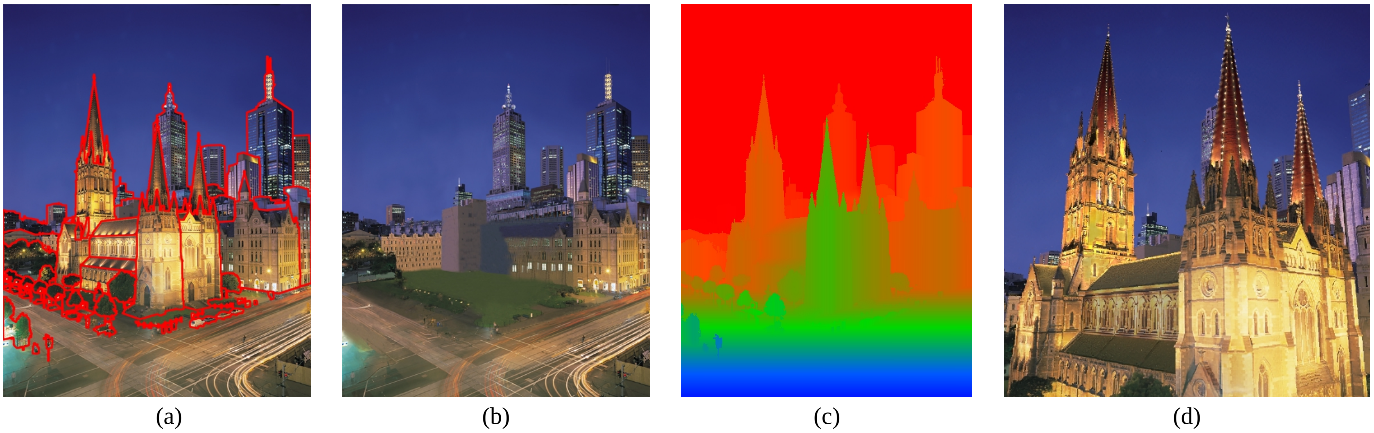


Figure 1: St Paul's Cathedral in Melbourne. (a) Image segmented into layers (boundaries in red). (b) Hidden parts manually clone brushed by the user. (c) False-color rendering of the depth of each pixel. (d) New viewpoint and relighting of the roof and towers.

1.1 Previous work

We make a distinction between two classes of image-based techniques. The first is based on sampling. View warping is a typical example that uses depth or disparity per pixel [CW93, LF94, MB95, SGHS98]. Higher dimensional approaches exist [LH96, GGSC96], but they are still too costly to be practical here. The representation is purely independent of the geometry, but for real images, depth or disparity must be recovered, typically using stereo matching. Closer to our approach, Kang proposes to leave the depth assignment task to the user via a painting metaphor [Kan98], and Williams uses level sets from silhouettes and image grey levels [Wil98].

The second class concerns image-based modeling systems that take images as input to build a more traditional geometric representation [Pho, Can, DTM96, FLR+95, LCZ99, POF98, Rea]. Using photogrammetry techniques and recovering textures from the photographs, these systems can construct photorealistic models that can be readily used with widely-available 3D packages. However, the use of traditional geometric primitives limits the geometry of the scene, and the optimization techniques often cause instabilities.

Our goal is to bring these two classes of approaches together. We wish to build a flexible image-based representation from a single photograph, which places no constraints on the geometry and is suitable for editing.

The work closest to ours is the plenoptic editing approach of Seitz et al. [SK98]. Their goal is also to build and edit an image-based representation. However, their approach differs from ours in that their system operates on multiple views of the same part of the scene and propagates modifications among different images, allowing a better handling of view-dependent effects. Unfortunately, the quality of their results is limited by the volumetric representation that they use. Moreover, they need multiple images of the same object viewed from the outside in.

We will see that some of our depth acquisition tools and results can be seen as a generalization of the *Tour Into the Picture* approach, where central perspective and user-defined billboards are used to 3D-navigate inside a 2D image [HAA97]. Our work, however, imposes no restrictions on the scene geometry, provides a broader range of depth acquisition tools, and supports a variety of editing operations.

Our work is also related to 3D painting, which is an adaptation of popular 2D image-editing systems to the painting of textures and other attributes directly on 3D models [HH90, Met]. This approach is, however, tightly constrained by the input geometry and the texture parameterization.

1.2 Overview

This paper makes the following contributions:

- An image-based system that is based on a depth image representation organized into layers. This representation is simple, easy to render, and permits direct control. It is related to layered depth images (LDIs) [SGHS98], but offers a more meaningful organization of the scene. We demonstrate our system on high-resolution images (megapixels, as output by current high-end digital cameras).

- A new set of tools for the assignment of depth to a single photograph based on a 2D painting metaphor. These tools provide an intuitive interface to assign relevant depth and permit direct control of the modeling.

- A non-distorted clone brushing operator that permits the duplication of portions of the image using a brush tool, but without the distortions due to foreshortening in the classical 2D version. This tool is crucial for filling in gaps, due to occlusions in the input image, that appear when the viewpoint changes.

- A filter to decouple texture and illuminance components in images of uniformly textured objects. It factors the image into two channels, one containing the large-scale features (assumed to be from illumination) and one containing only the small-scale features. This filter works even in the presence of sharp illumination variations, but cannot discount shadows of small objects. Since it results in uniform textures, it is crucial for clone brushing or for relighting applications.

- Our system permits editing from different viewpoints, e.g. painting, copy-pasting, moving objects in 3D, and adding new light sources.

2 System overview

2.1 Layers of images with depth

All elements of our system operate on the same simple data structure: images with depth [CW93]. This permits the use of standard image-based rendering techniques [CW93, MB95, SGHS98, MMB97]. Depth is defined up to a global scale factor.

The representation is organized into layers (Fig. 1(a) and 2), in the spirit of traditional image-editing software and as proposed in

```
layer {
    reference camera  : transformation matrix
    color channels    : array of floats
    alpha channel     : array of floats
    depth channel     : array of floats
    optional channels : arrays of floats
}
```

Figure 2: Basic layer data structures.

computer vision by Wang and Adelson [WA94]. An alpha channel is used to handle transparency and object masks. This permits the treatment of semi-transparent objects and fuzzy contours, such as trees or hair. Due to the similarity of data structures, our system offers an import/export interface with the Adobe Photoshop format that can be read by most 2D image-editing programs.

The image is manually segmented into different layers, using selection, alpha masks, and traditional image-editing tools (Fig. 1(a)). This is typically the most time-consuming task. The parts of the scene hidden in the input image need to be manually painted using clone brushing. This can more easily be done after depth has been assigned, using our depth-corrected version of clone brushing.

A layer has a reference camera that describes its world-to-image projection matrix. Initially, all layers have the same reference camera, which is arbitrarily set to the default OpenGL matrix (i.e. identity). We assume that the camera is a perfect pinhole camera, and unless other information is available, that the optical center is the center of the image. Then, only the field of view needs to be specified. It can be entered by the user, or a default value can be used if accuracy is not critical. Standard vision techniques can also be used if parallelism and orthogonality are present in the image (see Section 3). Note that changing the reference camera is equivalent to moving the objects depicted in the layer in 3D space. Throughout the paper we will deal with two kinds of images: *reference images* that correspond to the main data structure, and *interactive images* that are displayed from different viewpoints to ease user interaction. The degree to which the viewpoint can be altered, without artifacts, is dependent on the particular scene, assigned depth, and occluded regions.

Our organization into layers of depth images is related to the LDIs [SGHS98], with a major difference: In an LDI, the layering is done at the pixel level, while in our case it is done at a higher level (objects or object parts). LDIs may be better suited for rendering, but our representation is more amenable to editing, where it nicely organizes the scene into different higher-level entities.

Additional channels, such as texture, illuminance, and normal (normals are computed for each pixel using the depth of the 4 neighboring pixels), may be used for specific applications (relighting in particular).

2.2 System architecture

The architecture of our system is simple, since it consists of a set of tools organized around a common data structure (Fig. 3). It is thus easy to add new functionality. Although we present the features of our system sequentially, all processes are naturally interleaved. Editing can start even before depth is acquired, and the representation can be refined while the editing proceeds.

Selection, like channels, is represented as an array corresponding to the reference image. Each pixel of each layer has a selection value, which can be any value between 0 and 1 to permit feathering. Selection is used not only for copy-pasting, but also for restricting the action of the tools to relevant areas.

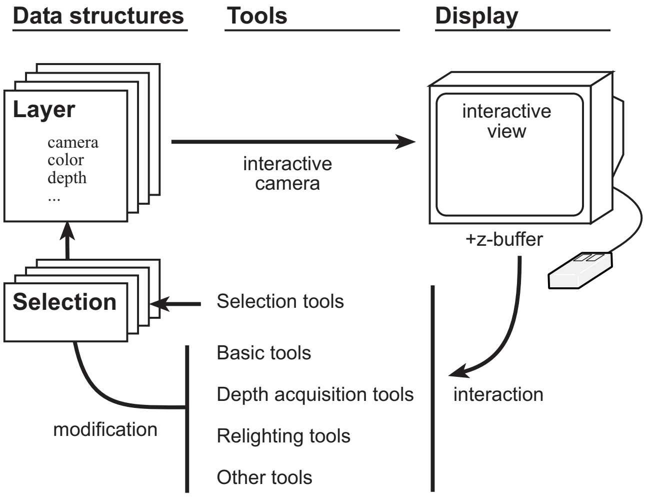


Figure 3: Architecture of our system.

The interactive display is performed using triangles [McM97, MMB97] and hardware projective texture mapping [SKvW+92]. The segmentation of the scene into layers greatly eliminates rubber-sheet triangle problems. Obviously, any other image-based-rendering technique such as splatting could be used [CW93, SGHS98, MB95].

The tools, such as depth assignment, selection or painting, can be used from any interactive viewpoint. The z-buffer of the interactive view is read, and standard view-warping [CW93, McM97] transforms screen coordinates into 3D points or into pixel indices in the reference image. The texture parameter buffers of Hanrahan and Haeberli could also be used [HH90].

3 Depth Assignment

We have developed depth assignment tools to take advantage of the versatility of our representation. The underlying metaphor is to paint and draw depth like colors are painted, in the spirit of Kang [Kan98]. This provides complete user control, but it also relies on the user's ability to comprehend the layout of the scene. The level of detail and accuracy of depth, which can be refined at any time, depend on the target application and intended viewpoint variation.

However, even if a user can easily infer the spatial organization and shapes depicted in the image, it is not always easy to directly paint the corresponding depth. Hence we have also developed hybrid tools that use pre-defined shapes to aid in painting accurate depth. In the development of our interface, we have emphasized 2D, rather than 3D interaction, the direct use of cues present in the image, and the use of previously-assigned depth as a reference.

Depth can be edited from any interactive viewpoint, which is important in evaluating the effects of current manipulations. Multiple views can also be used [Kan98]. We will see that some tools are easier to use in the reference view, where image cues are more clearly visible, while for others, interactive views permit a better judgment of the shape being modeled.

The use of selection also permits us to restrict the effect of a tool to a specific part of the image, providing flexibility and finer control. And since our selections are real-valued, the effect of depth tools can be attenuated at the selection boundary to obtain smoother shapes. In our implementation, we use the selection value to interpolate linearly between the unedited and edited values. Smoother functions, such as a cosine, could also be used.

In contrast to optimization-based photogrammetry systems [Can, DTM96, FLR+95], the field of view of the reference camera must be specified as a first step (as aforementioned, we assume a perfect pinhole camera). If enough information is available in the image, the field of view can be calculated (Section 3.2). The user can also set the focal length manually. Otherwise, the focal length is set to a default value (50mm in practice).

3.1 Depth painting

The user can directly paint depth using a brush, either setting the absolute depth value or adding or subtracting to the current value (chiseling). Absolute depth can be specified using a tool similar to the color picker, by clicking on a point of the image to read its depth. The relative brush tool is particularly useful for refining already-assigned depth (Fig. 5(b)). The size and softness of the brush can be interactively varied.

The whole selected region can also be interactively translated. Translation is performed along lines of sight with respect to the reference camera: The depth of each selected pixel is incremented or decremented (Fig. 4). However, it is desirable that planar objects remain planar under this transformation. We do not add or subtract a constant value, but instead multiply depth by a constant value. Depth-translating planar objects therefore results in parallel planar objects.

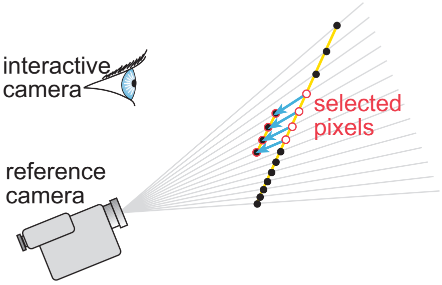


Figure 4: Depth translation is performed along lines of sight with respect to the reference camera.

In the spirit of classical interactive image-editing tools, we have developed local blurring and sharpening tools that filter the depth channel under the pointer (Fig. 5(c)). Blurring smooths the shape, while sharpening accentuates relief. Local blurring can be used to “zip” along depth discontinuities, as described by Kang [Kan98]. A global filtering is also possible, in particular blurring to smooth the 3D shape, noise to add complexity, and median filtering to remove outliers.

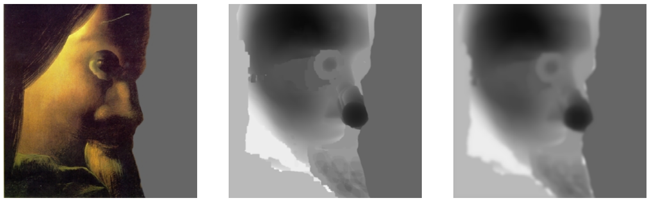


Figure 5: (a) Face. (b) Chiseled depth. (c) Blurred depth.

Similar to Kang’s [Kan98] and William’s [Wil98] methods, the user can use the rgb channels to assign depth. This is motivated by cases where darker values correspond to more distant pixels (such as trees) or by atmospheric perspective making distant objects bluer. The user specifies the z_{min} and z_{max} depth values, and the vector specifying color direction $\vec{C}$ (e.g. dark to light or amount of blue), and the effect can be applied absolutely or relatively. In the absolute case, for example, depth is then specified from the color at each pixel $\vec{c}(x,y)$ from:

$$z(x,y) = z_{min} + (z_{max} - z_{min}) * \vec{C} \cdot \vec{c}(x,y).$$

3.2 Ground plane and reference depth

The tools presented so far work best if some initial depth has been assigned, or if a reference is provided for depth assignment. Similar to the perspective technique used since the Renaissance, and to the spidery mesh by Horry et al. [HAA97], we have found that the use of a reference ground plane greatly simplifies depth acquisition and improves accuracy dramatically, since it provides an intuitive reference. The position with respect to the ground plane has actually been shown to be a very effective depth cue [Pal99]. Specifying a ground plane is typically the first step of depth assignment.

The ground plane tool can be seen as the application of a gradient on the depth channel (Fig. 6). However, an arbitrary gradient may not correspond to a planar surface. In our system, the user specifies the horizon line in the reference image, which constrains two degrees of freedom, corresponding to a set of parallel planes. The remaining degree of freedom corresponds to the arbitrary scale factor on depth. We can thus arbitrarily set the height of the observer to 1, or the user can enter a value.

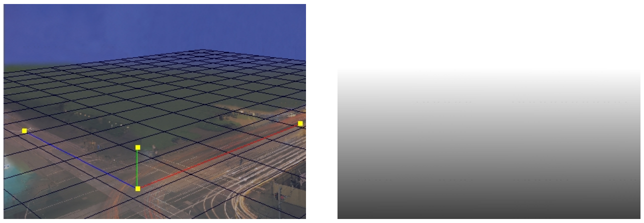


Figure 6: (a) Ground plane. (b) Depth map.

We have also implemented the method by Liebowitz et al. [LCZ99] that allows the acquisition of architectural models and camera parameters from one image using parallelism and orthogonality constraints. This provides both the camera parameters and an accurate reference ground plane. This also allows us to compute the position of the optical axis if it is not in the center of the image.

Depth picking and depth painting can then easily be used to depth-paint billboards parallel to the image plane. Since most objects in a scene touch the ground, or their projection on the ground can be inferred by the user, this proves to be very efficient. This is similar to the placement of billboards on the “spidery mesh” of Horry et al. [HAA97]. However, their system is limited to central perspective and polygonal orthogonal objects, while we can refine our representation and obtain arbitrary shapes.

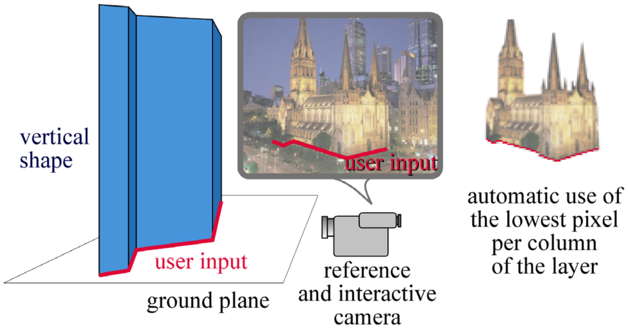

vertical shape
user input
user input
ground plane
reference and interactive camera
automatic use of the lowest pixel per column of the layer

Figure 7: Vertical tool. The user draws the contact of the vertical geometry with the ground plane.

We have extended the notion of billboards to allow the user to paint the depth of arbitrary vertical objects. This works best when the interactive camera is set to the reference camera: The contact or projection of the object on the ground plane is drawn, and a vertical depth is extruded (Fig. 7). In practice, the contact drawn by the user is represented as a polyline, the corresponding vertical polygons are rendered using OpenGL, and the z-buffer is read to update the selected pixels.

We are not limited to the use of a planar ground reference. The 3D locations of the points of the contact polyline drawn by the user are read from the interactive z-buffer. This means that if the ground has been chiseled or translated for better terrain modeling, vertical objects will be extruded accordingly.

We have also implemented an automatic version that processes the whole selection or layer at once. It assumes that the layer or selection is in contact with the ground reference at its lowest pixels. Each column of pixels in the reference image is assigned the depth corresponding to its lowest visible or selected pixel.

3.3 Geometric primitives

Some geometric shapes, such as boxes, spheres, or cylinders, are hard to depth-paint accurately. We therefore utilize geometric primitives that can be drawn transparently as 2D objects. For example, the user draws a circle or clicks on three points to assign spherical depth. We use similar interfaces for cylinders (the user draws the edges), boxes (the user draws three edges of a corner), and pyramids (the user draws the base and apex). These tools work best when used from the reference camera.

The primitive is rendered from the reference camera using OpenGL, and the z-buffer is read to assign depth. This requires the use of a depth buffer at the resolution of the reference image. Since our system treats images that can be larger than the screen, we use tiling.

Once the image projection of the primitive has been provided, its distance must be specified. The user can use an arbitrary distance and then refine it with the translation tool. He can also use the already-assigned depth as a reference by clicking on one point. By default, the first point clicked by the user is used as a reference depth (e.g. corner of a box).

To improve the quality of depth when a ground plane has been assigned, the user can use *primitive snapping* to enforce the verticality of boxes, cylinders or pyramids, or to constrain them along the normal of a given pixel. A least-square error minimization is then run to optimize the 3D shape position.

3.4 Organic shapes

In the spirit of Williams [Wil98] and related to the Teddy fast-modeling system [IMT99], we propose a tool that assigns depth using level sets. It is well suited to giving organic shapes a bulgy appearance, by specifying more distant depth at the boundary and closer depth in the center (Fig. 8(a) and (b)). This tool is relative, and the range r of depth addition is specified by the user.

We compute the level sets using an erosion technique (e.g. [SD95]). The initial active interior of the object is defined as the set of pixels of the layer or selection with non-null alpha. The distance to the boundary d_{bound} is initialized to 0, and we iteratively "erode." For each iteration, we discard pixels that have a non-active neighbor, and increment the distance of active pixels by 1.

We use the normalized distance to the centroid $d' = 1 - \frac{d_{bound}}{d_{bound}max}$ and update depth according to $z = z + r\sqrt{1 - d'^2}$. This formula was chosen because it assigns a spherical shape to a disk under orthographic projection.

3.5 Faces

The specific case of human faces is important. The output of the impressive morphable model of Blanz et al. [BV99] could be used to retrieve accurate depth. However, this technique is not easy to implement since it requires a large database of 3D face scans, and unfortunately, it takes tens of minutes to acquire the geometry of a face from a photograph.

We have developed a simpler method that trades accuracy and user intervention for speed and simplicity. This method could be further generalized to a broader class of template shapes. We use a generic arbitrary 3D face model, optimize its 3D position to match the photograph, and then use 2D morphing to refine the match (Fig. 8).

The user specifies correspondence points between the image and the 3D model. These points are used to find the rotation, scale, and position of the 3D model using Levenberg-Marquardt optimization [PSVF92]. Rotation and scale are optimized independently to avoid shear in the resulting transformation. The 3D face is rendered, and the z-buffer is read back. We then use the same correspondence points to morph the z-buffer and obtain a better match with the image using triangle-based morphing and linear interpolation [GDCV98].

4 Non-distorted clone brushing

Standard clone brushing has its limitations when perspective causes texture foreshortening (Fig. 9(a)). In practice, only parts with similar orientation and distance to the camera can be clone brushed. Moreover, only regions of similar intensity can be copied. The former problems will be treated in this section, while the latter will be addressed in the next section.

Since our system has information about depth, we can correct distortion due to both perspective and surface shape. In the general case of arbitrary geometry, the problem is similar to low-distortion texture mapping: We want to map the source region of the image-based representation to the destination, with as little distortion as possible. Our idea is to compute a (u, v) texture parameterization for both the source and destination regions, and use this mapping for the clone brush.

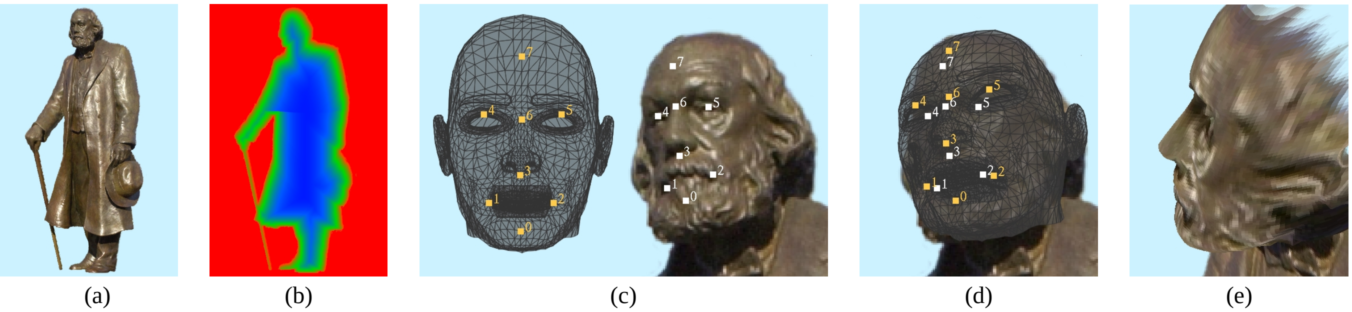
4.1 Non-distorted parameterization

Our parameterization optimization is based on the work by Levy et al. [LM98, Mal89], with three important differences: Our approach is local around the clone-brushed regions, has no boundary condition, and needs to run in real-time. We first quickly review the key points of their method in order to describe the specifics of our flood-fill adaptation. This overview is presented in our specific context, where each pixel is seen as a vertex connected to its 4-neighbors. We refer the reader to their article for details [LM98, Mal89].

Levy et al. propose to minimize two classes of distortions: angular (preserve orthogonal angles) and iso-parametric distance (make isolines equidistant). The former requires that the gradient of u and v be orthogonal, and the latter requires constant magnitude for their gradients. Different weights can be assigned to emphasize one constraint over another.

They use *discrete smooth interpolation* [Mal89], where each discrete value (u or v at a pixel in our case) is smoothed using a linear combination of its neighbors. The combinations minimize a *roughness* function to obtain a valid mapping, and quadratic constraints to minimize distortion. Smoothing steps on u and v are interleaved and iterated.

In our case, the roughness R for a pixel p depends on its 4-neighbors $N(p)$:



(a) (b) (c) (d) (e)

Figure 8: Organic and face tools. (a) Initial statue image. (b) False-color depth map after level set tool. (c) User-specified correspondence points. (d) Transformation after optimization. (e) Face from new viewpoint after depth is applied.

$$R(u,v) = \sum_p \left(-4u_p + \sum_{p'\in N(p)} u_{p'}\right)^2 + \left(-4v_p + \sum_{p'\in N(p)} v_{p'}\right)^2 . \quad (1)$$

The minimum of R is reached when its partial derivatives $\frac{\partial R}{\partial u}$ and $\frac{\partial R}{\partial v}$ are null for each pixel p. This yields an expression of u and v at a pixel as a linear combination of the values of the neighbors of its neighbors:

$$\begin{aligned} u_p &= \sum_{p'\in N(N(p))} a^u_{p,p'} u_{p'} \\ v_p &= \sum_{p'\in N(N(p))} a^v_{p,p'} v_{p'} , \end{aligned} \quad (2)$$

where the a^i_j are given coefficients.

Distortions are minimized by introducing a set of linear constraints C depending on the gradients of u and v. Levy et al. use a least-square approach, which defines a generalized roughness $R^\star$:

$$R^\star(u,v) = R(u,v) + \sum_{c\in C} \left\{ \left(\sum_p A_p^{c,u} u_p\right) - b_c^u \right\}^2 + \left\{ \left(\sum_p A_p^{c,v} v_p\right) - b_c^u \right\}^2 , \quad (3)$$

where the A^i_j and b^i_j are given coefficients *when one parameter, u or v, is assumed constant*. The smoothing on u and v are interleaved, which means that the values of $A^i_{c,u}$ depend on v, and vice-versa.

This generalized roughness $R^\star$ reaches its minimum when its partial derivatives, $\frac{\partial R^\star}{\partial u}$ and $\frac{\partial R^\star}{\partial v}$, are null for each pixel p. This can be expressed as a set of linear equations:

$$\begin{aligned} u_p &= \sum_{p'\in N(N(p))} a^u_{p,p'} u_{p'} + \alpha^u_{p,p'} u_{p'} \\ v_p &= \sum_{p'\in N(N(p))} a^v_{p,p'} v_{p'} + \alpha^v_{p,p'} v_{p'} , \end{aligned} \quad (4)$$

where α^u_j are coefficients that depend on v, and α^v_j depend on u.

This method is, however, too slow to optimize over an entire layer in our interactive context. Moreover, it requires boundary conditions. We therefore adapt it using two strategies: a flood-fill approach and a simple but effective initialization.

4.2 Flood-fill parameterization

We adapt the discrete smooth interpolation method in a “flood-fill” manner to optimize the parameterization around the current position of the clone brush. We compute the parameterization for only a subset of pixels, called *active pixels*. This subset is expanded as time progresses and the user drags the brush. We describe the method for a single layer, however, it runs concurrently for both the source and destination layers. We interleave *optimization* steps, where coordinates are refined, and *expansion* steps, where new pixels are declared active and initialized. We moreover *freeze* the coordinates of already-brushed pixels.

To initialize the process, we use the first point clicked by the user as a seed, assign it the coordinates $(0,0)$, and set the gradient of u, $\vec{\nabla}u$ orthogonal to the vertical direction and to the pixel normal. $\vec{\nabla}v$ is then orthogonal to $\vec{\nabla}u$.

The set of pixels at the boundary of the active region is called the *active front*. More formally, a pixel is declared in the active front if it is active and if one of its 4-neighbors has an inactive 4-neighbor (Fig. 9(c)). Intuitively, the active front corresponds to pixels that lack neighbors necessary to compute smoothing in Eqs. (2) and (4). Active pixels not in the active front are said to be *fully active*.

Optimization

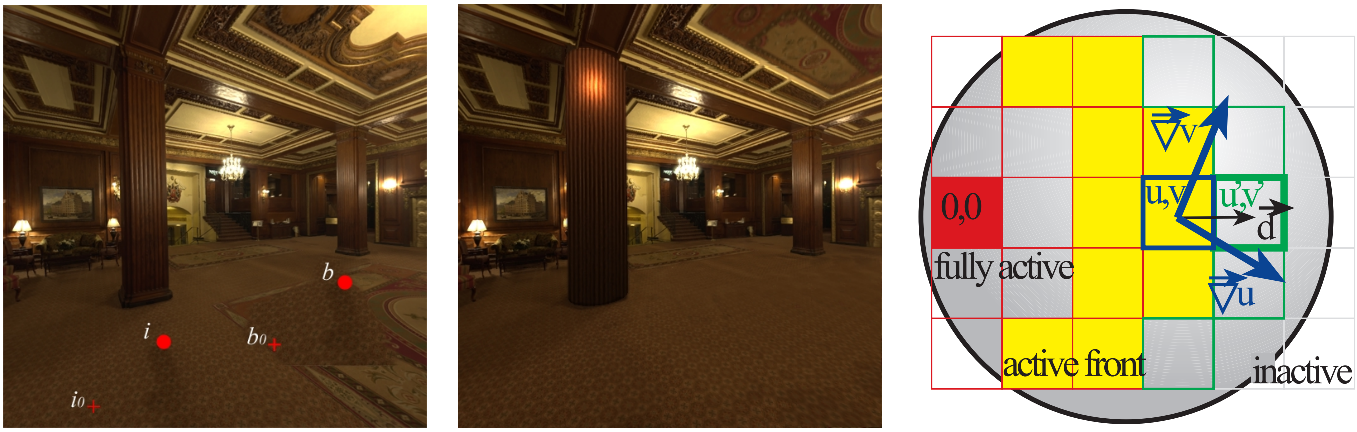
The optimization steps follow the lines of the original discrete smooth interpolation. Optimizations on u and v are interleaved, and active pixels are treated as vertices of a mesh and smoothed accordingly using linear operations.

The active front requires special care. Due to the absence of some neighbors, Eq. (4) cannot be directly used. If these neighbors are simply discarded from the formula, the parameterization will “shrink,” because the roughness term R from Eq. (1) is no longer balanced. We thus only optimize the gradient constraints for these pixels and omit the mapping of the roughness term (the a^i_j in Eq. (4)). A good initial value for the active-front pixels is then the key to the stability of the process.

Expansion and initialization

An expansion step extends the active region by one pixel in the direction of the current mouse location. The active front is accordingly updated, and each new active pixel receives initial coordinate values. This is done according to its active neighbors, by using a local planar approximation of the geometry. For each neighbor, the coordinates (u',v') of the new pixel are computed using the current gradients of an active neighbor, $\vec{\nabla}u$ and $\vec{\nabla}v$, and the object-space vector $\vec{d}$ between the two pixels (Fig. 9(c)):

$$(u',v') = (u + \vec{d}\cdot\vec{\nabla}u, v + \vec{d}\cdot\vec{\nabla}v).$$



(a) (b) (c)

Figure 9: (a) Classical clone brushing. The user first clicks on i_0 to specify the source region and then paint brushes starting at b_0, which assigns the translation. b is then a simple copy of i. (b) Perspective-corrected clone brushing. The column geometry has been changed to a cylinder, and the carpet has been removed and clone brushed onto the ceiling. (c) Flood-fill parameterization. Fully active pixels are in red. The active front is in yellow. The green pixel is set active, and its initial (u', v') parameters are computed using the gradient of its active neighbors.

The average of the values computed from the active neighbors is used. This formula results in an optimal initial value, provided the geometry is planar.

Freezing and clone brushing

The parameterization proceeds as the user interactively clone brushes. It must be faster than the speed of the brush to ensure smooth interaction. In practice, we have found that subsampling the layer was necessary in order to obtain real-time feedback. We compute (u, v) values every 4×4 pixels and interpolate bilinearly. This process does not take into account the local bumps in the geometry, but fits the global shape.

As soon as a pixel has been clone brushed, its (u, v) coordinates must be *frozen* to avoid artifacts that would occur if the same pixel were re-used with different coordinate values due to subsequent optimization iterations.

Clone brushing a destination pixel with coordinate (u, v) involves inverting the mapping of the source image. Note that in the general case, no pixel will have the exact $(u, v,)$ coordinates. We thus use the four pixels with the nearest coordinates and perform bilinear interpolation. To find these pixels, we use a marching method. Since a brush corresponds to a set of contiguous pixels, we only need to compute a seed value and march from it along the gradient to find the inverse mapping of subsequent pixels.

Our optimization is clearly not as accurate as the method of Levy and Mallet [LM98, Mal89]. However, it provides an exact solution in the case of planar geometry and has worked well in practice for curved geometry. This is because our case is simpler than the general mesh-parameterization problem. Our data is a height field transformed by a perspective matrix, which greatly decreases potential distortions. Moreover, our layers are segmented by the user into different spatial objects that prevent strong discontinuities.

5 Texture-illuminance decoupling filter

We now present a filter that factors the image into a texture component and an illumination component. This is useful both for relighting and clone brushing, since the decoupled texture channel has a uniform level of illumination.

Most previous relighting work relies on a light transport simulation to remove the effect of existing lighting [FGR93, DRB97, Deb98, YDMH99, LFD$^+$99, LDR00]. Loscos et al. use texture synthesis to remove artifacts along shadow boundaries, but still require an initial physical simulation [LDR00]. In contrast, our approach is not physically based. It is an image-processing filter that removes lighting effects from uniformly textured objects.

A related approach was introduced by Nayar and Bolle [NB93]. Our approach differs from theirs in that they deal with non-textured regions and focus on the segmentation and computation of reflectance ratios, while we deal with texture extraction.

5.1 Large- and small-scale feature separation

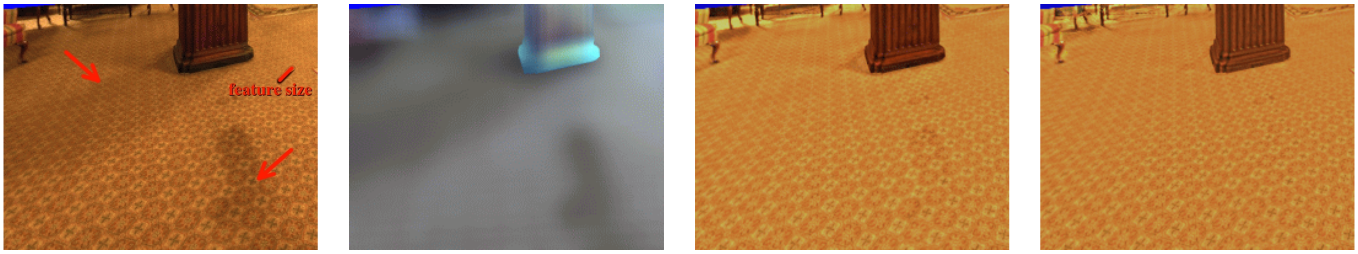
We make the following simple assumption: Large-scale luminance variations are due to the lighting, while small-scale details are due to the texture. In practice, this means that large stains will be treated as illuminance variations (which is actually desirable in most practical cases), while shadows of small objects will not be handled correctly. Small detailed shadows are the main limitation of our technique.

We have developed a non-linear filter that factors an image into a texture channel and an illuminance channel respecting the above assumption. We do not claim that these are the true texture and illuminance, but we will use these terms for simplicity. This problem is related to image denoising, but the "noise" in this case is the texture information that we want to retrieve.

To begin, the user specifies a *feature size* of the texture by dragging a line segment over a pattern. The basic idea is to blur the image with a low-pass Gaussian filter G, specified by the feature size (in practice we use $\sigma_{spatial} = feature\ size$). If I_0 is the input image, and p and p' are pixel locations, we have:

$$I_1(p) = \frac{\Sigma_{p'} G(p, p', \sigma_{spatial}) I_0(p')}{\Sigma_{p'} G(p, p', \sigma_{spatial})}. \qquad (5)$$

Only large-scale illuminance variations remain. We moreover assume that the average color comes from the texture component, so we divide the illuminance obtained by the normalized average color value. We then divide the initial image by this blurred version to compute a uniform texture component (Fig. 10(b) and (c)).



(a) (b) (c) (d)

Figure 10: Texture-illuminance decoupling. (a) Input image. (b) Initial illuminance estimation using simple Gaussian filtering. (c) Initial texture estimation, note the artifacts corresponding to shadow boundaries. (d) Texture computed using bilateral filtering.

This approach works well for slowly varying illumination, but fails at shadow boundaries, where haloing effects occur (see Fig. 10(c) and 11). This simply means that shadow boundaries introduce frequencies that are in the range of the feature size, and that they are treated as texture frequencies by the Gaussian blur. In addition, texture foreshortening needs to be treated to make consistent use of the feature size.

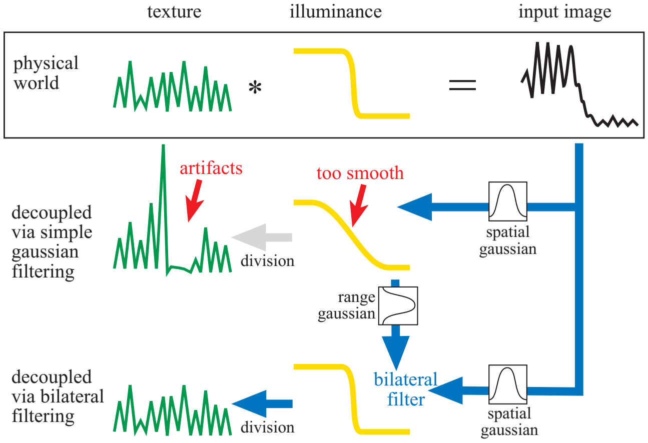


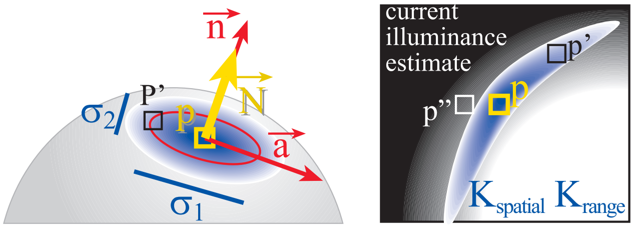
Figure 11: Texture-illuminance decoupling in 1D. The example exhibits sharp illumination variations, which cannot be captured using simple Gaussian filtering. This first pass is, however, used for the bilateral filtering pass, to average only pixels with similar illumination, due to the use of an additional range Gaussian.

5.2 Depth correction

Due to foreshortening and surface orientation, the feature size is not constant in image space. We therefore use the depth channel to compensate for foreshortening, and normals for orientation. The user specifies feature size at a reference pixel p_{ref}. For other pixels, the spatial kernel is scaled by $\frac{z_{ref}}{z}$. To compensate for orientation, we use a local planar approximation of the surface: Gaussian ellipses oriented along the scene surfaces (Fig. 12(a)).

Let $\vec{N}$ be the unit normal and $\vec{E}$ the unit viewing direction. The small axis of the ellipse is along $\vec{n}$, the unit image projection of $\vec{N}$. We note $\vec{a}$ the long axis of the ellipse, which is a unit vector orthogonal to $\vec{n}$. The small/large ratio $\frac{\sigma_2}{\sigma_1}$ is given by the dot product $\vec{N} \cdot \vec{E}$, where $\sigma_1 = \frac{z_r ef}{z(p)} \sigma_{spatial}$ as described above. We then have:

$$K_{spatial}(p', p, \sigma_{spatial}) = G(\vec{pp'} \cdot \vec{n}, \sigma_2) G(\vec{pp'} \cdot \vec{a}, \sigma_1). \qquad (6)$$



(a) (b)

Figure 12: (a) Orientation correction of the spatial Gaussian. (b) The kernel filter for pixel P is the product of the spatial and range Gaussians: The weight of point p'' is lower than the weight of p' although it is closer, because the estimated illuminance of p' is more similar to that of p.

5.3 Bilateral filtering

To handle discontinuities, we use a non-linear edge-preserving filter. Anisotropic diffusion could be an option [PM90], but since we have information about the feature size, we prefer the use of bilateral filtering [TM98], which is more controllable. The principle is to use an additional Gaussian in the range domain to average only pixels of similar intensity. The total kernel is the product of the spatial Gaussian (Eq. (5)) and this range Gaussian (Fig. 12(b)).

We adapt this method to our particular case. It is iterative, and the current estimate of the illuminance is used to drive the range Gaussian:

$$\begin{aligned} I_{i+1}(P) &= \frac{\Sigma_{p'} K(p',p) I_0(p')}{\Sigma_{p'} K(p',p)} \\ K(p',p) &= K_{spatial}(p', p, \sigma_{spatial}) G_{range}(I_i(p), I_i(p'), \sigma_{range}). \end{aligned}$$

The main difference between this approach and standard bilateral filtering is that we always filter the initial image I_0. Unlike denoising approaches, we are interested in factoring the initial image, not in removing noise. Because the kernel averages only pixels of similar estimated illuminance, the filter captures shadow boundaries (Fig. 10(d) and 11). The process converges quickly, and we use I_3 as our final illuminance estimate. The only hard-coded parameter is the variance of the range Gaussian. In practice, we have found that $\sigma_{range} = 0.01 max(I_1)$ gives good results.

This filter is very effective on high-dynamic range images [DM97]. For 24-bit images, it works best if the illumination does not vary too dramatically. Otherwise, very dark or very bright regions can lead to artifacts, because the texture information has been destroyed by color quantization. In this case, texture synthesis is the only solution to resynthesize the lost information.

6 Implementation and Results

Our system has been implemented on SGI workstations using the QT GUI library and the graphics API OpenGL. The accompanying video demonstrates a variety of results obtained with the system. In this section, we first describe in detail how the church example was acquired and then describe other examples and editing operations.

6.1 Church example

For the church example (Fig. 1), we used a single 24-bit 1000x1280 scanned input photograph. The most time-consuming part of the acquisition was the manual segmentation of the input image into layers. Because our system uses only one input photograph, the area behind the church was manually clone brushed (Fig. 1(b)). The segmentation and clone brushing took about 10 hours. 52 different layers were extracted: Each tree is represented as a separate layer, and the church itself is decomposed into 13 layers for easier depth assignment and to allow for occlusion and parallax effects (Fig. 1(a)).

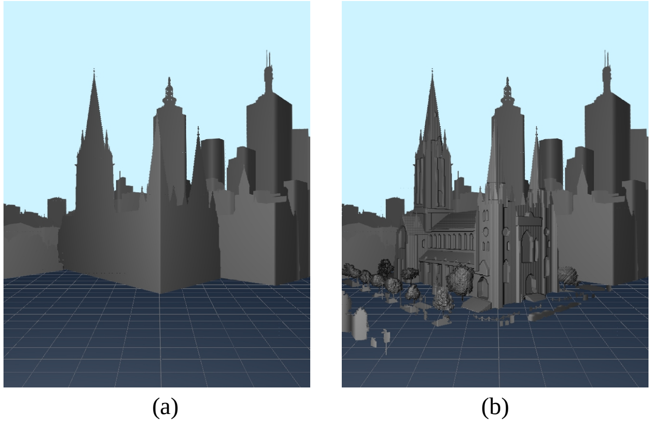


Figure 13: St Paul's Cathedral acquired geometry. (a) Coarse depth after the use of the vertical tool. (b) Refined depth. Note the chiseling on the façades and the shapes of the trees, obtained using a combination of level sets and depth-from-rgb.

Depth acquisition took an additional three hours. We first defined the ground plane and camera parameters using the method by Liebowitz et al. [LCZ99]. Three orthogonal pairs of parallel lines were specified. Each layer was given a coarse billboard depth by utilizing the automatic vertical plane tool. For those layers without a clear point of contact with the ground, we used a user-specified vertical plane. Fig. 13(a) shows this initial coarse depth. We then used the level-set method to provide the trees and bushes with a bulgy appearance, which we refined using the depth-from-rgb tool. The depths for the cars and street furniture were refined with the box tool.

The church depth was acquired using several tools. The coarse depth from the vertical tool provided a starting point. We used the push/pull tool to add relief to the façade windows and buttresses. We then used the pyramid tool for the tower and turrets, and the plane tool for the roofs (Fig. 13(b)).

6.2 Editing and other examples

Various editing operations are possible including painting, filtering, and clone brushing. The ability to paint from different viewpoints makes it easy to edit foreshortened surfaces and obtain correct perspective. Copy-pasting, 3D scaling, translation, and rotation of objects are also possible. The range of rotation, like changes in viewpoint, are ultimately limited by the geometry of the scene and disocclusion artifacts.

We also demonstrate relighting applications. Texture and illuminance are decoupled using our new filter. Then, the illuminance channel is modified and multiplied by the texture channel to obtain the new image. The illuminance can be edited either by specifying 3D light sources, or by directly painting light on the channel. The latter solution often provides simpler interaction. This is due to the difficulty of specifying 3D positions and anticipating the resulting lighting effects. Sketching approaches, e.g. [PRJ97] could be helpful in this context.

We also use cubical panoramas as input, representing them as a collection of six images (Fig. 14). This provides a dramatic immersive experience and could greatly improve techniques such as Quicktime VR [Che95] at a reasonable cost.

7 Conclusion and Future Work

We have presented an image-based modeling and editing system that takes a single photo as input, and allows a user to build a representation consisting of layers of images with depth. Interactive views and a variety of editing operations permit the definition and manipulation of image-based scenes. The system has been demonstrated on several examples, including relighting.

Future work includes the merging of this approach with 3D painting programs, or with alternative image-editing approaches [EG01]. Many other techniques could be incorporated into our system. For example, depth assignment could benefit from shape from shading or from multiple images via stereo correspondence and epipolar geometry. In addition, handling multiple images of the same portion of a scene would permit the inclusion of view-dependent effects.

We believe that our clone brushing and texture-illuminance decoupling tools have application beyond the scope of this system. For example, a simpler version of clone brushing could be adapted to the 2D case for simple configurations. Our decoupling filter could be useful in a variety of contexts, including enabling classical image-based modeling to retrieve uniform texture or to preprocess the input of texture generation algorithms.

Acknowledgments

We would like to thank Aparna Das, Rujira Hongladaromp, and Sini Kamppari for experimenting with the system, providing feedback, and working on the examples. Thanks to Steven Gortler, George Drettakis, Nicolas Tsingos, Leonard McMillan, and Neel Master for encouragement and helpful discussions. Barry Webb and Associates, Lighting and Technology Consultants in Sydney, Australia, kindly provided the photograph of St. Paul's Cathedral in Melbourne. This work was supported by an NSF CISE Research Infrastructure Award (EIA-9802220) and a gift from Pixar Animation Studios.

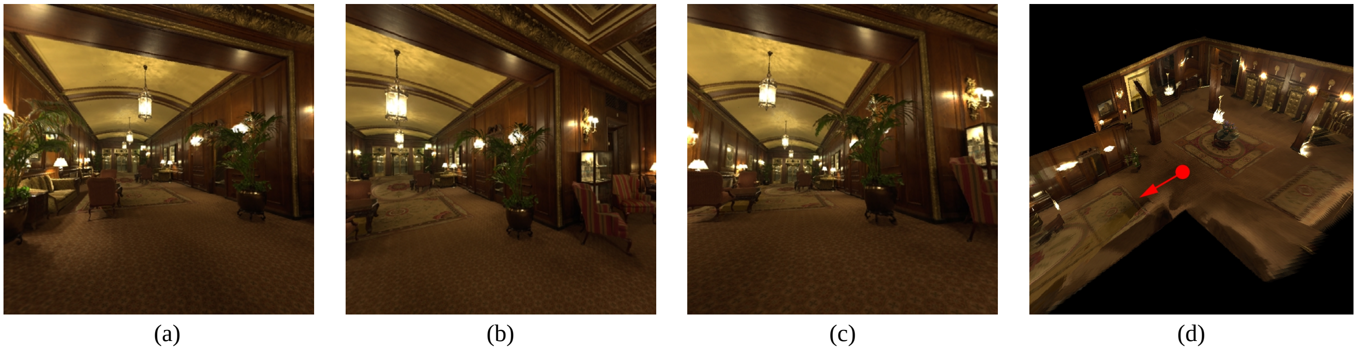


Figure 14: Panoramic view of a hotel lobby. (b) is the original viewpoint, and (a),(c) are synthetic viewpoints. (d) visualizes the representation from a birds-eye view. The red arrow shows the original acquisition point and direction of the panorama. Although the geometry is coarse, the immersive experience within the room is very convincing.

References

[BV99] V. Blanz and T. Vetter. A morphable model for the synthesis of 3d faces. *Proc. of SIGGRAPH*, 1999.

[Can] Canoma. http://www.canoma.com.

[Che95] E. Chen. Quicktime VR - an image-based approach to virtual environment navigation. *Proc. of SIGGRAPH*, 1995.

[CW93] E. Chen and L. Williams. View interpolation for image synthesis. In *Proc. of SIGGRAPH*, 1993.

[Deb98] P. Debevec. Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *Proc. of SIGGRAPH*, 1998.

[DM97] P. Debevec and J. Malik. Recovering high dynamic range radiance maps from photographs. *Proc. of SIGGRAPH*, 1997.

[DRB97] G. Drettakis, L. Robert, and S. Bougnoux. Interactive common illumination for computer augmented reality. *Eurographics Rendering Workshop*, 1997.

[DTM96] P. Debevec, C. Taylor, and J. Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In *Proc. of SIGGRAPH 96*, 1996.

[EG01] J. Elder and R. Goldberg. Image editing in the contour domain. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 23(3), 2001.

[FGR93] A. Fournier, A. Gunawan, and C. Romanzin. Common illumination between real and computer generated scenes. *Graphics Interface*, 1993.

[FLR$^+$95] O. Faugeras, S. Laveau, L. Robert, G. Csurka, and C. Zeller. 3-d reconstruction of urban scenes from sequences of images. In A. Gruen, O. Kuebler, and P. Agouris, editors, *Automatic Extraction of Man-Made Objects from Aerial and Space Images*. Birkhauser, 1995.

[GDCV98] J. Gomes, L. Darsa, B. Costa, and L. Velho. *Warping And Morphing Of Graphical Objects*. Morgan Kaufman, 1998.

[GGSC96] S.. Gortler, R. Grzeszczuk, R. Szeliski, and M. Cohen. The lumigraph. In *Proc. of SIGGRAPH*, 1996.

[HAA97] Y. Horry, K. Anjyo, and K. Arai. Tour into the picture: Using a spidery mesh interface to make animation from a single image. In *Proc. of SIGGRAPH 97*, 1997.

[HH90] P. Hanrahan and P. Haeberli. Direct wysiwyg painting and texturing on 3d shapes. *Proc. of SIGGRAPH*, 1990.

[IMT99] T. Igarashi, S. Matsuoka, and H. Tanaka. Teddy: A sketching interface for 3D freeform design. In *Siggraph*, Los Angeles, 1999.

[Kan98] S. Kang. Depth painting for image-based rendering applications. Tech. report, CRL, Compaq Cambridge Research Lab, 1998. http://www.research.microsoft.com/Users/sbkang/publications/index.html.

[LCZ99] D. Liebowitz, A. Criminisi, and A. Zisserman. Creating architectural models from images. In *Proc. of Eurographics*, 1999.

[LDR00] C. Loscos, G. Drettakis, and L. Robert. Interactive virtual relighting of real scenes. *IEEE Trans. on Visualization and Computer Graphics*, 6(3), 2000.

[LF94] S. Laveau and O. Faugeras. 3-D scene representation as a collection of images and fundamental matrices. In *Proc. of 12th Int. Conf. on Pattern Recognition*, volume 1, pages 689–691, 1994.

[LFD$^+$99] C. Loscos, M.C. Frasson, G. Drettakis, B. Walter, X. Granier, and P. Poulin. Interactive virtual relighting and remodeling of real scenes. *Eurographics Rendering Workshop*, 1999.

[LH96] M. Levoy and P. Hanrahan. Light field rendering. In *Proc. of SIGGRAPH*, 1996.

[LM98] B. Lévy and JL Mallet. Non-distorted texture mapping for sheared triangulated meshes. In *Proc. of SIGGRAPH*, 1998.

[Mal89] JL Mallet. Discrete smooth interpolation. *ACM Trans. on Graphics*, 8(2):121–144, 1989.

[MB95] L. McMillan and G. Bishop. Plenoptic modeling: An image-based rendering system. In *Proc. of SIGGRAPH*, 1995.

[McM97] L. McMillan. *An Image-based Approach to Three-Dimensional Computer Graphics*. PhD thesis, U. of North Carolina, Chapel Hill, 1997.

[Met] MetaCreations. http://www.metacreations.com.

[MMB97] W. Mark, L. McMillan, and G. Bishop. Post-rendering 3D warping. In *ACM Symp. on Interactive 3D Graphics*, 1997.

[NB93] S. K. Nayar and R. M. Bolle. Computing reflectance ratios from an image. *Pattern recognition*, 7, 1993.

[Pal99] S. Palmer. *Vision Science : Photons to Phenomenology*. MIT Press, 1999.

[Pho] Photomodeler. http://www.photomodeler.com.

[PM90] P. Perona and J. Malik. Scale-space and edge detection using anisotropic diffusion. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 12(7):629–639, July 1990.

[POF98] P. Poulin, M. Ouimet, and M.C. Frasson. Interactively modeling with photogrammetry. In *Eurographics Workshop on Rendering*, 1998.

[PRJ97] P. Poulin, K. Ratib, and M. Jacques. Sketching shadows and highlights to position lights. In *Proc. of Computer Graphics International 97*, 1997.

[PSVF92] W. Press, S.Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes*. Cambridge Univ. Pr., 2nd edition, 1992.

[Rea] Realviz. Image modeler. http://www.realviz.com.

[SD95] F. Sillion and G. Drettakis. Feature-based control of visibility error: A multi-resolution clustering algorithm for global illumination. In *Proc. SIGGRAPH*, 1995.

[SGHS98] J. Shade, S. Gortler, L. He, and R. Szeliski. Layered depth images. In *Proc. of SIGGRAPH*, 1998.

[SK98] S. Seitz and K. Kutulakos. Plenoptic image editing. In *Proc. 5th Int. Conf.on Computer Vision*, 1998.

[SKvW$^+$92] M. Segal, C. Korobkin, R. van Widenfelt, J. Foran, and P. Haeberli. Fast shadows and lighting effects using texture mapping. *Proc. of SIGGRAPH*, 1992.

[TM98] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *IEEE Int. Conf. on Computer Vision*, 1998.

[WA94] J. Y. A. Wang and E. H. Adelson. Representing moving images with layers. *IEEE Trans. on Image Processing*, 3(5):625–638, 1994.

[Wil98] L. Williams. Image jets, level sets and silhouettes. Workshop on Image-Based Modeling and Renderingt, http://www-graphics.stanford.edu/workshops/ibr98/, March 1998.

[YDMH99] Y. Yu, P. Debevec, J. Malik, and T. Hawkins. Inverse global illumination: Recovering reflectance models of real scenes from photographs. *Proc. of SIGGRAPH*, 1999.