Image-Based Remodeling

Alex Colburn, *Student Member*, *IEEE*, Aseem Agarwala, Aaron Hertzmann, *Senior Member*, *IEEE*, Brian Curless, *Member*, *IEEE*, and Michael F. Cohen, *Member*, *IEEE*

Abstract—Imagining what a proposed home remodel might look like without actually performing it is challenging. We present an image-based remodeling methodology that allows real-time photorealistic visualization during both the modeling and remodeling process of a home interior. Large-scale edits, like removing a wall or enlarging a window, are performed easily and in real time, with realistic results. Our interface supports the creation of concise, parameterized, and constrained geometry, as well as remodeling *directly from within the photographs*. Real-time texturing of modified geometry is made possible by precomputing view-dependent textures for all faces that are potentially visible to each original camera viewpoint, blending multiple viewpoints and hole-filling when necessary. The resulting textures are stored and accessed efficiently enabling intuitive real-time realistic visualization, modeling, and editing of the building interior.

Index Terms—Image-based rendering, modeling packages, visualization systems and software

1 Introduction

Remodeling a home is expensive, time consuming, and disruptive, and it is rarely practical to revise or undo the changes. Unfortunately, it is also hard to imagine what a proposed remodel might actually *look* like; ideally, one would like a realistic visualization of a proposed change to a home interior before any hammers are swung. Professionals can use CAD programs to create 3D models, but considerable skill and time are required to achieve photorealism. The alternative is to use image-based modeling and rendering, which allow photorealistic virtual environments from a few photographs. But how easy is it to *edit* an image-based model? Can large-scale edits, like removing a wall or enlarging a window, be performed easily and in real-time, with realistic results?

At present, the answer is no. The most visually realistic IBR systems combine geometric proxies with view-dependent texture mapping (VDTM) that blend multiple photographs from nearby viewpoints. If the geometry is too simple (planar proxies, as in PhotoTourism [1]) or too unstructured (a collection of polygons, as in Furukawa et al. [2]), there is no easy and effective way to edit it. Simple texture-mapped models are flat and unrealistic, while view-dependent texture mapping has no affordances for editing, since typically a few input photographs are simply blended together to create a rendering. If you suddenly remove a wall, how do you find the appropriate disoccluded bits of texture and blend them in seamlessly, in real time?

- *A. Colburn and B. Curless are with the Department of Computer Science and Engineering, University of Washington, Paul G. Allen Center, Seattle, WA 98195-2350. E-mail: alex@colburn.org, curless@cs.washington.edu.*
- *A. Agarwala is with Adobe Systems, Inc., 801 N 34th St, Seattle, WA 98103. E-mail: aseem@agarwala.org.*
- *A. Hertzmann is with the University of Toronto, 40 St George St. Rm 4283, Toronto, ON m5s2e4, Canada. E-mail: hertzman@dgp.toronto.edu.*
- *M.F. Cohen is with Microsoft Research, One Microsoft Way, Redmond, WA 98052. E-mail: mcohen@microsoft.com.*

Manuscript received 3 Nov. 2011; revised 22 Feb. 2012; accepted 12 Mar. 2012; published online 22 Mar. 2012.
Recommended for acceptance by G. Drettakis.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number TVCG-2011-11-0267.
Digital Object Identifier no. 10.1109/TVCG.2012.95.

Performing large-scale geometric edits within a realistic image-based virtual environment is an unsolved problem, and is the focus of our paper. Using home interior architectural modeling and remodeling as an example application, we present a method for building models using photographs in a manner that allows the geometry and texture to be interactively remodeled. We focus on a *representation* for both the geometry and the texture in a scene that allows large-scale edits to be easily performed and rendered in real time. Our interface supports the creation of concise, parameterized, and constrained geometry, as well as remodeling *directly from within the photographs*. This frees the user from the need to manually specify 3D coordinates or maintain a consistent architectural model. View-dependent texture mapping minimizes visual distortions while allowing the user to navigate within the home.

The user begins by collecting photographs of the interior, which are processed with structure from motion (SfM) and multiview stereo (MVS) to estimate camera poses and a semi-dense 3D point cloud. From these inputs, we provide an interactive technique for quickly creating a geometric model of the existing house. The original photographs provide photorealistic, view-dependent texture for rendering. The user may then remodel by adding, removing, and modifying walls, windows, and door openings directly within the photographs. All edits are visualized in real time, and can be viewed from any direction. Actual photographs depict natural lighting, and make the scene appear familiar since existing objects and decoration such as furniture, plants, and so on remain within the scene. The ability to remodel directly within the context of photographs provides a means to quickly experiment and understand the implications of possible changes (Fig. 1).

The work we present does include some simplifications and tradeoffs. Furniture, plants, etc., that are not explicitly modeled with geometry introduce some unavoidable artifacts during navigation. We also assume for now that the majority of walls are oriented in one of two orthogonal directions, and that floors are level. Remodeling operations do not currently support additions of new rooms, but rather focus on removal and additions within the existing building

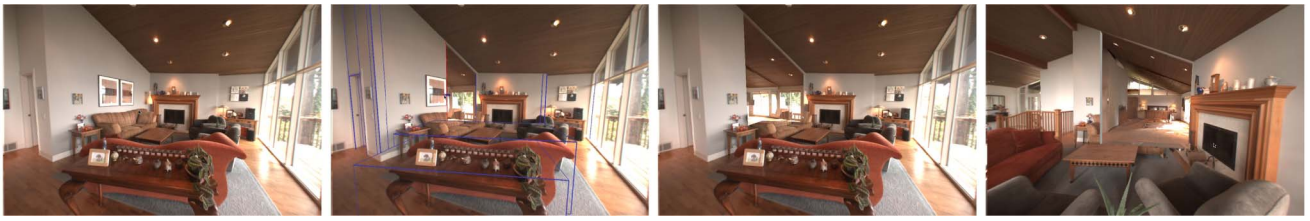

Fig. 1. After modeling an interior through the images, a user can start from an original viewpoint (left), click and drag to pull back a wall between a living room and a bedroom (center left), and view the edited result (center right). Then, the user navigates to visualize the edit from another viewpoint (right).

footprint. We also leave interior design operations such as lighting changes, furniture and cabinetry edits, and material property edits (such as paint color) to future work.

2 Related Work

Commercial tools are available to model existing home interiors and to visualize proposed renovations. Professional CAD software, such as Autodesk's Revit, allow the creation of parameterized and constrained models, so that changes propagate through associated primitives, e.g., changing the height of a wall affects adjoining walls. These systems do not handle the natural lighting nor the clutter in existing structures and thus the resulting visualizations are somewhat artificial. Our system of constraints and building primitives are, however, inspired by these systems.

While CAD modeling requires significant expertise, simpler tools such as Autodesk's Homestyler are targeted at nonexperts and can be used to author attractive—but highly-abstracted—floor plan renderings. These tools do not support the creation of photorealistic 3D models based on the original environment.

Google SketchUp allows interactive modeling using photographs with the "Photomatch" feature. This feature is designed to support modeling and visualization by projecting textures onto an existing scene. However, it does not provide a good representation for remodeling. For example, objects are split by occlusions during texture projection, e.g., when a column interrupts a view of a wall. Split polygons make subsequent editing extremely difficult. SketchUp assumes a single, globally defined texture-mapping, which behaves poorly from novel viewpoints when using coarse proxy geometry (e.g., for plants, furniture, and other clutter), and when making edits to geometry. Combining multiple textures requires manually defining a common frame-of-reference, and there is no support to have the rendered texture respond to viewpoint changes. We illustrate these and other issues with remodeling in SketchUp in Fig. 2. In contrast, our system is designed to support efficient editing and refinements to geometry, and renders with view-dependent texture, thereby making the editing process both simpler and more visually faithful to the real space.

By starting with a collection of photographs, image-based modeling techniques can create photo-realistic virtual environments with little or no user effort. Automatic modeling and rendering of photographed environments range from systems like Photo Tourism [1], where simple planes are used as geometric proxies for rendering novel viewpoints, to automatic, dense, MVS reconstruction [3],

Fig. 2. Comparison of Google Sketchup's Photomatch using geometry exported from our system. Top row: Google Sketchup, Bottom row: Our System. In the center images, the column was removed and a passage opened into a study. Note how Sketchup breaks up polygons due to original occlusions in the image projections. This makes remodeling almost impossible. Also note poor textures in revealed and new surfaces. Right: the same edit viewed from a different viewpoint. Our view dependent texture automatically utilizes texture relevant to the new viewpoint.

[4], [5], [6], [2]. However, most MVS techniques produce either point clouds or unstructured meshes; such models are sufficient for rendering but not editing, since they lack architectural primitives that can be selected and manipulated, and also lack the connectivity, structural representations, and constraints of easy-to-edit models.

A few methods automatically reconstruct higher level primitives. Dick et al. [7] reconstruct whole buildings by optimizing a generative model of architectural primitives. Müller et al. [8] use procedural shape grammars to reconstruct building façades composed of regular, repeating structures. A few methods estimate room layouts of simple volumetric primitives, from single images [9], [10]. Our goal, instead, is user-controllable modeling; however, when these automatic techniques become robust enough for general use they should be compatible with our editing and rendering techniques.

The other approach to creating image-based models is interactive [11], [12], [13], [14], [15], [16]. The system of Sinha et al. [17] is closest to ours, in that they allow "in-image" modeling bootstrapped from an automatically reconstructed 3D point cloud. However, they focus on reconstructing exteriors as collections of disconnected slabs and polygons which are more challenging to edit than our solid structures. Similarly, Nan et al. [18] interactively fit solid architectural building blocks to 3D point cloud data from LiDAR scans for large-scale urban reconstruction. One advantage of interactive model construction is that the primitives are often larger and more coherent, e.g., a single polygon for a wall. Our interactive modeling system borrows many ideas from these methods and adapts them to building interiors. However, none of these previous systems address the problem of authoring and rendering edits that depart from the original structures being modeled, which is our focus.

A few previous authors have described methods for editing image-based models. These efforts have focused on pixel operations such as painting [19] and cloning [13], or larger operations such as compositing and local deformations [20], [21]. Carroll et al. [22] allow editing of perspective in a single photograph of a building interior or exterior. To the best of our knowledge, our technique is the first image-based modeling system to support large-scale geometric changes such as removing entire walls.

3 Overview

Interactive, real-time editing of geometry with view-dependent texture mapping is not a trivial extension of existing techniques. There are three main requirements for such a system.

First, the geometry proxy used during rendering should be concise, parameterized, constrained to architecturally meaningful edits, and have affordances for manipulation. The underlying geometric representation must support a user being able to change the shape and position of geometry without creating holes, while maintaining a consistent, coherent model. At the same time, the texturing operations must automatically follow the editing operation in an intuitive manner. In other words, the geometry cannot be a point cloud or unstructured mesh. We solve this problem by creating a constrained solid geometric representation for our model, and utilize homogeneous 3D texture coordinates for all texture operations.

Second, we need a mechanism to render the scene in real time. Our textures must not only maintain the appearance of the original geometry, but also provide a mechanism to display all geometry that might be revealed during an edit from any viewpoint. Since texture synthesis is currently not feasible in a real-time system, we precompute view dependent textures, and organize them in a manner that limits bottlenecks caused by hardware memory transfer. We also provide support for viewpoints not well represented by any view-dependent texture, by computing a view-independent texture to gracefully provide a degraded runtime view when needed.

Finally, a user should have an interaction metaphor with affordances that support modeling and remodeling without a deep understanding of 3D structure. To this end, we create an "in-image" modeling and remodeling interaction metaphor. The user can easily click and drag to create new walls, change existing walls, and perform basic texture modifications, without specifying 3D coordinates or worrying about the underlying 3D point cloud or source photographs. We discuss our solutions to each of these problems in turn in the following sections.

4 Interactive Modeling from Images

The modeling process begins by first capturing a collection of photographs of the interior. Structure from motion automatically recovers camera poses and a semidense point cloud. The user then interactively creates a solid 3D model working directly within the images. The 3D point cloud, together with geometric constraints typical of home interiors, are used to simplify the process for the user. The geometric constraints include a common thickness for all walls, and orthogonality of walls and floors; i.e., surface normals are typically aligned with orthogonal, canonical axes (sometimes called the "Manhattan World" assumption). These constraints can be relaxed as needed to handle non-axis-aligned features, such as tilted ceilings and angled walls. We next provide more detail on each of these steps.

4.1 Capture and Initial Reconstruction

Images of the home interior are shot with an SLR and wide-angle (typically, fish-eye) lens, with fixed focal length. We capture images using a tripod with the camera oriented in landscape mode, moving to many different viewpoints to cover the space well. Photographing interiors well is particularly difficult as the lighting in a home can vary dramatically, from dimly lit interior rooms with artificial light, to rooms with floor-to-ceiling windows lit by the sun, to views out the windows themselves. While not required, we get the best results by bracketing exposures (aperture priority, fixed ISO-value and f-number). From these we can recover HDR images in a linear radiometric space shared across all viewpoints, using the recorded exposure settings to scale pixel values suitably for each image. To minimize appearance differences between views, we keep the artificial lighting constant (typically all lights turned on) during capture. Natural lighting may still change during

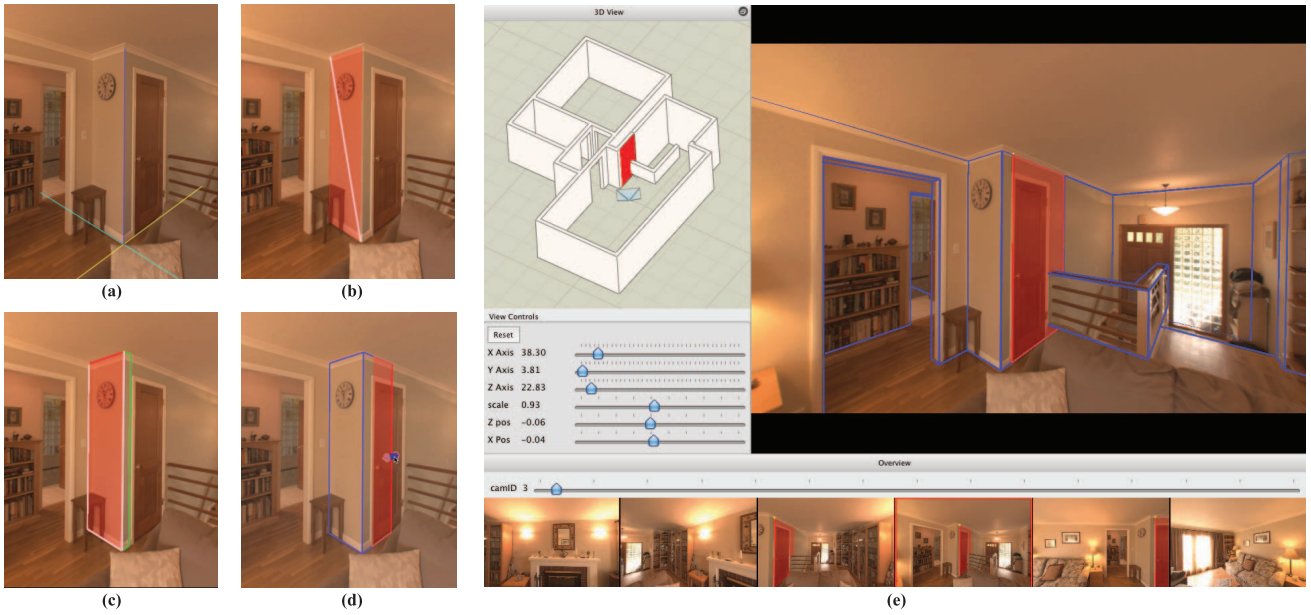

Fig. 3. (a)-(d) A corner composed of two walls is built interactively. (a) The canonical axes of the interior are refined to align with a corner in a photograph; these axes are used for the entire model building process. (b) The user drags out the corners of a wall face. A 3D rectangle is constructed and projected into the image in real time; the 3D rectangle is estimated based on the floor position and point cloud and automatically snapped to align with one of the canonical axes. (c) A solid wall is added to the model using the default wall thickness. (d) The user can then extend the model, adding walls as needed, ultimately constructing a complex wall unit. (e) A snapshot of the system's interface. The current model with the current view frustum and currently selected primitive highlighted in red is shown in the upper left, rendered orthographically. A wireframe is rendered over the current photograph on the right. A strip of alternate photos is shown on the bottom.

capture due to variations in cloud cover or in the angle of the sun; this variation is not problematic for SfM or MVS, but later requires more sophisticated compositing techniques when combining images, as discussed in Section 5. These techniques also help to handle non-HDR images taken under less controlled circumstances. The fisheye images are reprojected to wide angle perspective images using the nominal distortion parameters for that lens. Camera poses and sparse scene points are then recovered by the Bundler SfM tool [23]. Finally, we use PMVS [24] to recover semi-dense scene points with normals.

4.1.1 *Canonical Axes, Floor Height, and Wall Thickness*

An initial set of canonical axes is recovered with the method of Furukawa et al. [25], which finds three cluster centers of normal directions that are nearly orthogonal. We take the "up" axis to be the axis most closely aligned with the average of the camera up vectors. Cross products with the other cluster directions generate the other two canonical axes spanning a horizontal world plane, call them "east-west" and "north-south." An initial floor height is determined from all points with orientation within 25 degrees of the up direction. We compute distances of these points to the horizontal world plane, perform k-means clustering on these distances ($k = 3$), and take the distance of the largest cluster to be the floor height.

The resulting canonical axes and floor height may not be precisely aligned to the actual home interior. We provide a simple interface for adjusting them. The user clicks on a room corner at floor level in an image, and coordinate axes are centered at the intersection of the viewing ray and the floor and projected into the image (Fig. 3a). The user can then adjust the orientation of the object coordinates until the axes align with the lines where the two walls meet each other and the floor. Switching to another view of the corner (from a significantly different viewpoint), the user can adjust the floor height until it coincides with the corner in this second view. Finally, the user can specify the global wall thickness through direct measurement in an image, e.g., across the width of a wall that terminates at a passageway.

4.2 Modeling within the Images

Given the camera poses, canonical axes, floor height, and wall thickness, the user is now ready to build interior geometry directly within the images. To create a new wall, the user chooses an image, clicks on a wall and drags out a rectangle (Fig. 3b). A rectangular portion of a 3D wall appears automatically aligned with one of the canonical wall axes based on the predominance of point cloud normals within the rectangle. The wall position is also automatically determined by the predominant point locations. The wall has thickness and sits squarely on the floor plane (Fig. 3e). Moving to any new image, the newly created wall appears in place. Fine adjustments can made to the wall position from the new view.

The user continues modeling by extending walls through automatically created affordances for dragging the ends of walls to the corners of rooms. New wall segments can then be added to turn a corner. The new walls are automatically extended to include the solid intersection of the two walls resulting in a constrained, functional intersection. Continuing around a room often requires moving from image to image to complete the room. Acceleration tools allow the user to simply extend a wall to meet another already existing wall and to close the loop in a room with a join operation.

Fig. 4. A VDTA for a single viewpoint viewed from the side; note that possibly disoccluded textures are precomputed.

Walls can be split to create passageways, and holes can be punched in walls to create windows and doors. The doorway and window holes have affordances automatically created for dragging the edges to adjust to match them to images. Exterior windows autogenerate a separate thin slab for the glass aligned with the outside for texturing. (The modeling process can be seen in the accompanying video, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TVCG.2012.95.)

Alongside the image being viewed, a flat-shaded, orthographic view of the 3D model is shown, with a frustum icon denoting the viewpoint of the currently selected image. Fig. 3e illustrates the interface. As the model is updated through the images, the 3D view updates as well; similarly, the user may make simple adjustments to the model in the 3D view and see the updates propagated to the image view. Note that the model does not need to be extremely precise to give plausible visualizations of viewpoint motion and remodeling, as we will see in Section 7.

4.2.1 Representation

Observing that most interiors are comprised of abutting walls of common thickness that are oriented parallel or perpendicular to one another, we initially model the geometry as a union of axis-aligned, fixed-thickness rectangular solids. We represent this model as a set of polygon meshes, each traversed and managed with a half-edge data structure [26]. Each mesh is closed, yielding a solid model. This provides a data structure for fast access to mesh connectivity information for mesh editing operations, initial texture creation, and runtime texture reassignment. Edits are constrained to those deemed architecturally meaningful. Thus, for example, grabbing the edge of a doorway and moving horizontally is interpreted as opening the doorway. Or, grabbing the end of a wall and pulling along the wall normal moves the whole wall which might also automatically shorten or lengthen connecting walls.

5 Precomputed View-Dependent Texture

The heart of what makes our system effective for real-time remodeling is the crafting of precomputed view-dependent texture atlases (VDTA). The goal is to provide the ability to view the geometry with a photorealistic quality, while being able to navigate and, more importantly, alter the geometry to visually assess remodeling operations.

Fig. 5. A viewpoint is rendered using the VDTA (top). The bottom image is the same viewpoint using the VITA. Note the perspective distortions, and flattened clutter.

Our work draws on view-dependent texture mapping [12], which renders new viewpoints as a weighted blend of photographs taken from nearby viewpoints. Like Photo Tourism [1], our user interface encourages the user to be at or near the original viewpoints except when transitioning between views. A key advantage of VDTM is that rendering the original geometry from the point-of-view of where an image was captured results in exactly the original captured photograph.

However, conventional VDTM does not support live changes to the scene geometry. For example, suppose that a wall is removed, revealing the geometry behind. Since this geometry was not visible, VDTM provides no guidance on how to texture the revealed surface. One could generate texture for the newly visible geometry by searching for other views of it among the input photographs. There are a number of problems with this approach. For one, there often is no single good view of the newly visible geometry, requiring blending or hole-filling to produce realistic results. Second, searching the input images is too costly for real-time rendering. Consider the viewpoint rendered in Fig. 6. This scene has low-depth complexity, yet the texture generated for the newly visible geometry in the bottom image uses texture from 150 photos. The more complex scene in Fig. 4 uses data from 261 cameras. The memory transfer rates of our test machine from the host to the CPU is roughly 2,500 MB per second (as measured by oclBandwidthTest). If each images utilizes roughly 10 MB of texture RAM, the memory transfer alone would take a significant fraction of a second when assembling texture on the fly. The full set of images would require more RAM than the host's memory.

Fig. 6. Remodel with central walls removed. Images used to create the VDTA were captured on separate days under different lighting conditions, without HDR exposure bracket capture. The VDTA construction is robust to such lighting changes.

5.1 View-Dependent Texture Atlases

To address these problems, for each view we create a *View-Dependent Texture Atlas* as seen in Fig. 4. In addition to the VDTA for each view, we also create a single, low resolution, global *View-Independent Texture Atlas* (VITA), which is similar in size, form, and function to the VDTA. The VITA, as shown in Fig. 5, fills in any texture gaps during transitions between adjacent VDTAs.

Each VDTA stores a texture for all polygons that could possibly be seen or revealed in that view—that is, all faces within the view's frustum. The textures are preprojected to the corresponding viewpoint (i.e., with perspective foreshortening). This avoids wasted texture resolution on polygons that are highly oblique to the view. The textures are packed into a set of buffers to preserve memory coherence and to greatly reduce the number of unique textures required.

We pack the textures using a classic greedy first-fit algorithm [27]. The number of buffers is dependent on the page size of the buffer. We choose a page size of $1{,}024 \times 1{,}024$ which typically results in three or four buffers for a given view. We order the textures from largest to smallest, placing them sequentially in the first empty block in which they fit, continuing until all textures are placed. This simple scheme sufficiently obviates the need for frequent texture buffers switches when rendering.

The VDTA has an associated list of faces textured in each buffers, and a projection matrix that transforms for each face's 3D coordinates to image coordinates in the buffer. This is simply the projection matrix of the original viewpoint recovered by SfM translated and scaled to correspond to the position in the packed texture.

Since the texture for any given face may be clipped by the viewing frustum, we take care during rendering to only use texture coordinates within the original field of view. Validating texture coordinates uses a matrix multiply and bounds checking between the texture coordinate and the original viewpoint.

The texture atlases reduce the total number of runtime texture buffers per viewpoint from hundreds to a few combined texture buffers, while also reducing the total memory requirements by half. Expensive runtime texture swaps are reduced from hundreds per frame, to a few for each active VDTA. Once computed, any face that might become visible to any view, is rendered using only the VDTAs from nearby views, with some possible fill-in by the VITA. While texture atlases have been used before for image-based rendering [28], these methods do not support real-time rendering of edits.

5.2 Synthesizing the View-Dependent Textures

Having computed an assignment of faces to texture buffers, we still need to determine the appearance of each face. Each face can combine three kinds of texture:

1. **Original**: texture contained in the photograph from this viewpoint.
2. **Warped**: texture occluded from this viewpoint, but seen in other photographs.
3. **Synthesized**: texture not seen from any viewpoint.

The following procedure is used to fill the view-dependent texture atlas corresponding to each face.

5.2.1 Original Texture

Visibility of portions of a given face from the original viewpoint is computed using an ID buffer. Any pixels that are visible are copied directly from the original photograph. However, pixels within five pixels of any significant depth discontinuity are discarded. This prevents slight misalignments of depth discontinuities from creating erroneous mappings, such as a post being projected onto a wall far behind it. The remaining pixels of visible portions of faces are then filled from nearby viewpoints, or by hole-filling if not seen by other cameras.

5.2.2 Warped Texture from Other Views

Filling pixels occluded from the original viewpoint but visible in others requires solving two problems. First, we need to choose which other viewpoints to copy texture from. Second, we need to modify the copied content to blend seamlessly with the original view. The latter step is required because small lighting changes, non-Lambertian effects, and vignetting can lead to differences in appearance between different photographs of the same geometry.

As with the original view, candidates for views to fill-in occluded pixels are determined by projecting the occluded pixels into the ID buffers from each candidate view. If the ID matches (and is sufficiently far from a depth discontinuity) then this view becomes a candidate for fill-in. Of the possible candidates, we choose the best one by leveraging

the criteria used by Unstructured Lumigraph Rendering (ULR) [29]. Pixels are filled from a viewpoint that is similar in 3D location and view direction of the original view, and with a pixel coordinate toward the center of the image to avoid vignetting and other distortions. We use these criteria to select the single best candidate for each occluded pixel.

To solve the second problem, we composite using both the texture from the selected viewpoint and the corresponding texture gradients through the use of gradient-domain blending [30]. A weighted least-squares problem is solved where the occluded pixels' RGB values are variables. Original viewpoint pixels act as hard boundary constraints.

We iterate over each viewpoint that has been selected to be *best* for at least one pixel and render the face into a buffer. Each *best* pixel contributes a weak data constraint: the final composited color should be close to the rendered color from that viewpoint. Each pair of *best* neighboring pixels, and each adjacent *best* and non-*best* pair of neighboring pixels, contribute a gradient constraint: the difference between neighboring composited pixels should equal the difference between the pair of rendered pixels. The weak data constraints are weighted by a factor of 10^{-5} relative to the boundary and gradients constraints.

As noted by Bhat et al. [31], outlier gradients can cause bleeding artifacts when compositing in the gradient domain. We suppress the influence of large gradients; if the gradient is larger than 0.1, we down-weight the constraint by a factor of 10^{-5}. Finally, we solve the weighted least squares problem to create a composite.

5.2.3 Synthesized Hole Filling

Some pixels are not seen from any original photograph. These pixels are filled with a gradient-domain blend [30], [31] using the average color of the face as a weak constraint. More sophisticated hole filling methods could also be used, but is left for future work.

The above process is repeated for each viewpoint, beginning with the original view, then other views to fill in, and finally hole filling. This completes the offline construction of the VDTAs.

5.3 View-Independent Texture Atlas

Creating the single view-*independent* texture atlas proceeds in a similar fashion to the VDTAs with two exceptions. Since there is no preferred view direction, each surface is projected orthogonally before packing and texturing. There are also no original (preferred) textures, so all texture pixels are derived from either a warped texture or hole filling. Best views for each pixel are selected with the same unstructured lumigraph weighting, resulting in a view that is most orthogonal to the surface, and closer to the surface. Pixels from competing views are blended using the gradient domain blending as before but with no hard boundary constraints. The weak data terms from each view provide the anchor for the optimization. As before, any pixels not seen from any view are filled with synthesized hole fill.

5.4 Texturing New Surfaces

Finally, we also need to be able to texture newly created surfaces resulting from remodeling operations. These occur both when a completely new wall is added, but also when a new doorway is cut through an existing wall, since new, small, wall-thickness faces are created to complete the passageway. If the newly created face has coplanar faces, we extend the texture associated with the coplanar face by simply repeating the neighboring texture edge. For example, if one of the walls of an outside corner (Fig. 7) is extended, the texture on the edge of the colinear wall is simply stretched to fill the new wall. New surfaces without a coplanar surface are given a default wall color. The user may override the initial texture by copying texture from a selected rectangular patch on an existing wall. Newly created textures are treated as view independent and thus share the same texture across all views. This can cause minor artifacts when moving between views, but provides a good balance between visual fidelity and minimizing the need for user intervention.

6 Rendering

Given the geometric model and the complete set of VDTAs, we can render from any vantage point in real time, regardless of which faces are revealed. Rendering is performed at interactive frame rates using a combination of OpenGL and GLSL shaders. The geometry is textured from our VDTA iterating over each texture buffer. We load each texture buffer, and render only those faces associated with the buffer. Using a pixel shader, the homogeneous 3D texture coordinates of our model are transformed to 2D texture coordinates by multiplying the coordinate at each pixel by the texture matrix of the source texture taking care to avoid out of bounds projections. There are two boundary tests that must be performed. First, we discard texture coordinates projected outside of the original viewpoint. Second, we discard texture coordinates projected outside of the texture matrix.

When rendering from an original viewpoint, only the VDTA from that view is required. When navigating between viewpoints, the scene is rendered three times using VDTAs for the source and destination camera views, and the VITA to three off-screen buffers. The VDTAs are linearly blended to form a smooth transition, and the VITA is only used to fill any remaining holes.

The pixel shader adjusts the texture for exposure compensation. By default, the values are scaled to set the saturation point to be the same as the camera's autoexposure would. A GUI is provided to override this for manual exposure if desired.

7 Remodeling

Once the initial modeling is complete and VDTAs computed, the user can navigate through the model, change the virtual camera exposure, and remodel the structure by adding/removing walls, or adding and expanding/shrinking openings. All changes to the viewpoint and model are rendered in real time. Most importantly, the geometric model and texture atlases support rendering during remodeling operations.

As with the creation of the original model, remodeling operations are carried out in the original views with a similar set of tools. The user is able to click on a wall, and then grab edges of the wall or any opening and simply slide

Fig. 7. Three remodeling results. First column: original viewpoint. Second column: remodeled geometry. Third column: same remodel from different viewpoint. First row: opening a wall to connect two rooms. Second row: expanding a window vertically. Third row: adding a wall segment. Fourth row: creating a cutout.

them to make the wall or opening larger or smaller. As an opening is enlarged (as in Fig. 1), the room behind the wall is revealed by rendering those faces previously occluded in the original view. Widening a window (Fig. 7) simply stretches the outdoor texture on that window since we have no additional textured geometry for the outdoors. Textures on walls are similarly stretched by locking the vertex texture coordinates as is. In some situations it is preferable to avoid a stretched texture appearance by cropping the texture. In this case the texture coordinates are updated during editing and constrained to the texture limits, (Fig. 7, fourth row). As expected, all geometry changes are global and thus appear in all camera views.

7.1 Navigation

In addition to simply switching between original views as displayed in the filmstrip in Fig. 3, the user can change the view more continuously. We support two modes. In the first mode, using the GUI or keyboard, the user can move left/right and forward/back, or turn the view direction left/right and up/down, and zoom in/out. Translational motion introduces parallax which accentuates the 3D nature of the model enhancing the visualization. This is made possible by the fully textured layered model. Newly revealed surfaces, for example, through doorways appear naturally. Because the details such as furniture are not fully modeled, parallax does cause some artifacts.

The second mode involves moves from one original source view to a final destination view. The textures are interpolated between these two views as described in Section 6. Fly through paths are automatically created by interpolating between intermediary viewpoints with a similar motivation to that done in [2] and [32]. The idea is that the picture takers intuitively select good positions from which to view the scene. To determine the path from source to destination we construct a graph with a vertex for each

original viewpoint. Each vertex has an edge to another vertex if a line segment between the camera centers does not intersect the model. We set the edge weight w_{AB} between two views, A and B, to

$$w_{AB} = 1 + d_{AB} + \alpha \frac{\theta_{AB}}{d_{AB}},$$

where the 1 penalizes additional path segments, d_{AB} is the euclidean distance between the camera centers, and θ_{AB} is the angle between the optical axes of the cameras. We set $\alpha = 2/\pi$. We then solve for the shortest path between source and destination using Dijkstra's method.

When traversing the path, we treat each node, except the endpoints, as a virtual camera with camera center the same as the original viewpoint's, but the optical axis pointing in the direction from that node's camera center to the next node's camera center. We then linearly interpolate the camera centers and optical axes when moving from node to node. An extra node is inserted along the edge leaving the source and another along the edge arriving at the destination, positioned so that it takes one second to ease out of the starting orientation and 0.5 second to ease into the final orientation. Thus, the camera orientation begins with the source orientation and over one second interpolates to point along the path. One half second before the destination is reached, the orientation begins to interpolation between the path direction and destination camera orientation, ending finally on the destination camera position and orientation.

8 Results

We have used our image-based remodeling technique on three house interiors. (The most effective way to see the usage and results is through the accompanying video, available in the online supplemental material.) Each photograph set was captured with a Canon EOS 10D and Sigma fisheye lens. The first two sets were captured with three bracketed images separated by two stops.

The first data set seen in Fig. 3 has 143 viewpoints, and requires 15 minutes of interaction to yield a model of 223 faces. This is followed by about 30 minutes of processing time on an 18-node cluster to produce the VDTAs. The input photographs comprise 2.5 GB of camera RAW files, processed into 1.3 GB of HDR files in EXR format, resulting in 1.1 GB of VDTA files, also in EXR format.

The second data set visualized in Figs. 1, 4, and 7 has 462 camera viewpoints, and required about 45 minutes of interaction to produce a model of 661 faces, modeling eight rooms, followed by 2 hours of processing to produce the VDTAs. The input photographs were processed to 2 GB of HDR files, resulting in 5.2 GB of VDTA files.

The third data set shown in Fig. 6 has 326 viewpoints, captured on separate days under different lighting conditions without bracketed exposures. It required 30 minutes of interaction to model five rooms with 438 faces, followed by 1.5 hours of preprocessing to produce the VDTAs. The input photographs were processed to 1.7 GB of HDR files, resulting in 2.84 GB of VDTA files.

Rendering during modeling, remodeling, and navigation proceeds at 50 frames/second. Remodeling operations can be seen in Figs. 1, 6, and 7. These operations include widening doors, removing support beams, removing and adding walls, and resizing a window.

9 User Feedback

We conducted an informal study to solicit feedback on the use of the system for remodeling. We included five subjects, with professional experience ranging from two professional architects and designers with extensive CAD experience to three novice modelers including the homeowner of one model. Each subject was asked to navigate through viewpoints in a house, and then to perform several remodeling operations on a preconstructed model. These included expanding a doorway as well as punching a hole in a wall between rooms. Note we did not ask the subjects to construct the initial model from photographs although we did show them how the models were constructed. We were most interested in the perceived benefits of the image-based remodeling visualization.

The subjects universally said that they experienced a sense of being in the house. They all felt that they knew what the house was like, even if they had never been there before. The realistic image-based rendering enhanced their belief that they understood the result of a remodel. The owner of one house expressed an emotional response to the remodel operations. She said that "seeing real photos of my stuff added realism, and made the remodel seem more predictable." The architects liked the fact that a coarse model could be built quickly and was usable for rapid prototyping, exploring ideas, and visualizing changes. They thought our system could save time and effort by reducing the level of detail required in traditional models, details that are never used except to provide more realism. They wished the results could be integrated or exported to traditional CAD systems for further refinement and architectural notation and standards support. None of the users noticed the lack of relighting when asked.

10 Discussion

We have demonstrated a new interface for home modeling and remodeling using a through-the-photograph methodology. A synthesis of computer vision technology, texture atlas construction, and user-interface results in a system that provides an intuitive way to realistically visualize remodeling ideas in real time as one edits the geometry. We have demonstrated how models can be constructed efficiently, and textured in a way that enables visually plausible reediting. Precomputation of view dependent texture atlases allows visualization of edits in real time.

Although capturing large sets of images can be somewhat tedious, and the precomputation is expensive, we are confident the advantages of freeing the user from having to model in an abstract space, coupled with the real-time realistic visualization provides significant advantages over current alternatives.

Our technique has a number of current limitations. Since we do not model all the geometry precisely, there are rendering artifacts, most commonly in the rendering of household objects such as furniture and plants. Architectural details such as baseboards and window or door trim are also not modeled. We hope to leverage deeper architectural knowledge to help complete such details.

We do not currently support relighting, however our system has many advantages that can make this operation possible. For example, our HDR images provide environment (i.e., illumination) maps capable of acting as light sources. These maps plus the geometry can help separate

lighting from material properties. While lighting inconsistencies due to remodeling edits were not apparent to our small sample of users, this is an interesting problem for future work. Diffuse component relighting can be approximated by simply comparing the virtual views from a surface point before and after an edit. Specular components can be similarly approximated but require sensitivity to incident and outgoing lighting directions. In general, the full global relighting problem is more complex, however the work of Yu et al. on inverse global illumination [33], Ramamoorthi and Hanrahan [34] on inverse rendering, and Romeiro and Zickler on blind reflectometry [35] provides some guidance and inspiration for our future work.

Finally, it would be helpful to support structural scale edits such as adding a complete new room extension. This will require building a hybrid system that supports both image-based rendering where possible and more conventional modeling and rendering elsewhere. Again, we expect to leverage the captured imagery to aid in the description of the new geometry. An important question is how to visually represent large-scale additions. Should the addition be a blend of a stylized architectural rendering with the existing imagery, or should the addition visually mimic the existing structures in a seamless manner?

Achieving a visually seamless mix of novel geometry and image-based rendering is an interesting problem. First, the material properties and texture of the new geometry must match the appearance of the existing geometry. Second, the lighting model must be consistent across the existing and new models. Recent work using incident light fields by Unger et al. [36] and work using user annotations by Karsch et al. [37] demonstrate how well one may integrate artificial objects into image-based rendering environments within complex light fields. However, the artificial geometry does not yet take on visual characteristics of the environment. We would like to explore extracting incident light fields from our existing data and apply them as light sources for novel geometry.

We believe that the techniques we have demonstrated to date can be extended into a complete system for visualizing remodels. Also, allowing users to edit directly within images should open such applications to a much wider set of users.

Acknowledgments

This work is funded in part by Intel Science and Technology Center for Visual Computing, Microsoft Research, Adobe, National Science Foundation grants IIS-0811878 and IIS-0963657, NSERC, CIFAR, and the University of Washington Animation Research Labs.

References

[1] N. Snavely, S.M. Seitz, and R. Szeliski, "Photo Tourism: Exploring Photo Collections in 3D," *Proc. ACM SIGGRAPH '06 Papers,* http://doi.acm.org/10.1145/1179352.1141964, pp. 835-846, 2006.

[2] Y. Furukawa, B. Curless, S.M. Seitz, and R. Szeliski, "Reconstructing Building Interiors from Images," *Proc. IEEE 12th Int'l Conf. Computer vision (ICCV),* pp. 80-87, Oct. 2009.

[3] M. Pollefeys, L. Van Gool, M. Vergauwen, F. Verbiest, K. Cornelis, J. Tops, and R. Koch, "Visual Modeling with a Hand-Held Camera," *Int'l J. Computer Vision,* vol. 59, pp. 207-232, http://portal.acm.org/citation.cfm?id=986694.986705, Sept. 2004.

[4] N. Cornelis, B. Leibe, K. Cornelis, and L. Gool, "3D Urban Scene Modeling Integrating Recognition and Reconstruction," *Int'l J. Computer Vision,* vol. 78, pp. 121-141, http://portal.acm.org/citation.cfm?id=1355822.1355831, July 2008.

[5] G. Bahmutov, V. Popescu, and M. Mudure, "Efficient Large Scale Acquisition of Building Interiors," *Proc. ACM SIGGRAPH '06 Sketches,* http://doi.acm.org/10.1145/1179849.1180008, 2006.

[6] M. Goesele, N. Snavely, B. Curless, H. Hoppe, and S. Seitz, "Multi-View Stereo for Community Photo Collections," *Proc. IEEE 11th Int'l Conf. Computer Vision (ICCV '07),* pp. 1-8, oct. 2007.

[7] A.R. Dick, P.H.S. Torr, and R. Cipolla, "Modelling and Interpretation of Architecture from Several Images," *Int'l J. Computer Vision,* vol. 60, pp. 111-134, http://portal.acm.org/citation.cfm?id=993451.996343, Nov. 2004.

[8] P. Müller, G. Zeng, P. Wonka, and L. Van Gool, "Image-Based Procedural Modeling of Facades," *Proc. ACM SIGGRAPH '07 papers,* http://doi.acm.org/10.1145/1275808.1276484, 2007.

[9] V. Hedau, D. Hoiem, and D. Forsyth, "Recovering the Spatial Layout of Cluttered Rooms," *Proc. IEEE 12th Int'l Conf. Computer Vision (ICCV '09),* pp. 1849 -1856, Oct. 2009.

[10] D.C. Lee, A. Gupta, M. Hebert, and T. Kanade, "Estimating Spatial Layout of Rooms Using Volumetric Reasoning about Objects and Surfaces," *Proc. Adavances in Neural Information Processing Systems (NIPS '10),* vol. 24, Nov. 2010.

[11] R. Cipolla and D. Robertson, "3D Models of Architectural Scenes from Uncalibrated Images and Vanishing Points," *Proc. 10th Int'l Conf. Image Analysis and Processing (ICIAP '99),* pp. 824-829, http://portal.acm.org/citation.cfm?id=839281.840826, 1999.

[12] P.E. Debevec, C.J. Taylor, and J. Malik, "Modeling and Rendering Architecture from Photographs: A Hybrid Geometry- and Image-Based Approach," *Proc. 23rd Ann. Conf. Computer Graphics and Interactive Techniques,* pp. 11-20, http://doi.acm.org/10.1145/237170.237191, 1996.

[13] B.M. Oh, M. Chen, J. Dorsey, and F. Durand, "Image-Based Modeling and Photo Editing," *Proc. 28th Ann. Conf. Computer Graphics and Interactive Techniques,* pp. 433-442, http://doi.acm.org/10.1145/383259.383310, 2001.

[14] A. Criminisi, I. Reid, and A. Zisserman, "Single View Metrology," *Int'l J. Computer Vision,* vol. 40, pp. 123-148, http://portal.acm.org/citation.cfm?id=365875.365888, Nov. 2000.

[15] S. El-Hakim, E. Whiting, and L. Gonzo, "3D Modeling with Reusable and Integrated Building Blocks," *Proc. Seventh Conf. Optical 3-D Measurement Techniques,* pp. 3-5, http://www.iit-iti.nrc-cnrc.gc.ca/iit-publications-iti/docs/NRC-48228.p df, Jan. 2005.

[16] A. vanden Hengel, A. Dick, T. Thormählen, B. Ward, and P.H.S. Torr, "Videotrace: Rapid Interactive Scene Modelling from Video," *Proc. ACM SIGGRAPH '07 Papers,* http://doi.acm.org/10.1145/1275808.1276485, 2007.

[17] S.N. Sinha, D. Steedly, R. Szeliski, M. Agrawala, and M. Pollefeys, "Interactive 3D Architectural Modeling from Unordered Photo Collections," *Proc. ACM SIGGRAPH Asia '08 Papers,* pp. 159:1-159:10, http://doi.acm.org/10.1145/1457515.1409112, 2008.

[18] L. Nan, A. Sharf, H. Zhang, D. Cohen-Or, and B. Chen, "Smartboxes for Interactive Urban Reconstruction," *Proc. ACM SIGGRAPH '10 Papers,* pp. 93:1-93:10, http://doi.acm.org/10.1145/1833349.1778830, 2010.

[19] S.M. Seitz and K.N. Kutulakos, "Plenoptic Image Editing," *Int'l J. Computer Vision,* vol. 48, pp. 115-129, http://portal.acm.org/citation.cfm?id=598431.598504, July 2002.

[20] D.R. Horn and B. Chen, "Lightshop: Interactive Light Field Manipulation and Rendering," *Proc. Symp. Interactive 3D Graphics and Games (I3D '07),* pp. 121-128, http://doi.acm.org/10.1145/1230100.1230121, 2007.

[21] B. Chen, E. Ofek, H.-Y. Shum, and M. Levoy, "Interactive Deformation of Light Fields," *Proc. Symp. Interactive 3D Graphics and Games (I3D '05),* pp. 139-146, http://doi.acm.org/10.1145/1053427.1053450, 2005.

[22] R. Carroll, A. Agarwala, and M. Agrawala, "Image Warps for Artistic Perspective Manipulation," *Proc. ACM SIGGRAPH '10 Papers,* pp. 127:1-127:9, http://doi.acm.org/10.1145/1833349.1778864, 2010.

[23] N. Snavely, "Bundler: Structure from Motion for Unordered Image Collections," http://phototour.cs.washington.edu, http://phototour.cs.washington.edu/bundler/, 2008.

[24] Y. Furukawa and J. Ponce, "Patch-Based Multi-View Stereo Software," http://grail.cs.washington.edu/software/pmvs/, http://grail.cs.washington.edu/software/pmvs/, 2009.

[25] Y. Furukawa, B. Curless, S. Seitz, and R. Szeliski, "Manhattan-World Stereo," *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR '09),* pp. 1422-1429, 2009.

[26] C.M. Eastman and S.F. Weiss, "Tree Structures for High Dimensionality Nearest Neighbor Searching," *Information Systems,* vol. 7, no. 2, pp. 115-122, http://www.sciencedirect.com/science/article/B6V0G-48TD1M7-6H/2/34fd7dcac0ef3c2fa82a5eb688697662, 1982.
[27] D.S. Johnson, "Approximation Algorithms for Combinatorial Problems," *Proc. Fifth Ann. ACM Symp. Theory of Computing (STOC '73),* pp. 38-49, http://doi.acm.org/10.1145/800125.804034, 1973.
[28] H. Buchholz and J. Dollner, "View-Dependent Rendering of Multiresolution Texture-Atlases," *Proc. IEEE Visualization Conf.,* pp. 215-222, 2005.
[29] C. Buehler, M. Bosse, L. McMillan, S. Gortler, and M. Cohen, "Unstructured Lumigraph Rendering," *Proc. 28th Ann. Conf. Computer Graphics and Interactive Techniques,* pp. 425-432, http://doi.acm.org/10.1145/383259.383309, 2001.
[30] P. Pérez, M. Gangnet, and A. Blake, "Poisson Image Editing," *Proc. ACM SIGGRAPH '03 Papers,* pp. 313-318, http://doi.acm.org/10.1145/1201775.882269, 2003.
[31] P. Bhat, C.L. Zitnick, M. Cohen, and B. Curless, "Gradientshop: A Gradient-Domain Optimization Framework for Image and Video Filtering," *ACM Trans. Graphics,* vol. 29, pp. 1-14, http://doi.acm.org/10.1145/1731047.1731048, Apr. 2010.
[32] N. Snavely, R. Garg, S. Seitz, and R. Szeliski, "Finding Paths through the World's Photos," *Proc. SIGGRAPH '08,* http://portal.acm.org/citation.cfm?id=1399504.1360614, Aug. 2008.
[33] Y. Yu, P. Debevec, J. Malik, and T. Hawkins, "Inverse Global Illumination: Recovering Reflectance Models of Real Scenes from Photographs," *Proc. 26th Ann. Conf. Computer Graphics and Interactive Techniques,* pp. 215-224, http://dx.doi.org/10.1145/311535.311559, 1999.
[34] R. Ramamoorthi and P. Hanrahan, "A Signal-Processing Framework for Inverse Rendering," *Proc. 28th Ann. Conf. Computer Graphics and Interactive Techniques,* pp. 117-128, http://doi.acm.org.offcampus.lib.washington.edu/10.1145/383259.383271, 2001.
[35] F. Romeiro and T. Zickler, "Blind Reflectometry," *Proc. 11th European Conf. Computer Vision (ECCV '10),* K. Daniilidis, P. Maragos, and N. Paragios, eds., pp. 45-58, 2010.
[36] J. Unger, S. Gustavson, P. Larsson, and A. Ynnerman, "Free Form Incident Light Fields," *Computer Graphics Forum,* vol. 27, no. 4, pp. 1293-1301, 2008.
[37] K. Karsch, V. Hedau, D. Forsyth, and D. Hoiem, "Rendering Synthetic Objects into Legacy Photographs," *Proc. SIGGRAPH Asia Conf.,* pp. 157:1-157:12, http://doi.acm.org/10.1145/2024156.2024191, 2011.

Alex Colburn received the undergraduate degrees in biology and computer science from Beloit College and Colorado State University, respectively, and the MS degree in computer science from the University of Washington. He is a graduate student in the University of Washington's Computer Science & Engineering Department where he is working on computer graphics and computer vision. Prior to graduate school, he spent 10 years at Microsoft Research working on a variety of projects ranging from building and animating virtual worlds to processing microphone array input for sound source localization to combining image stacks into photomontages and segmenting video. He is a student member of the IEEE.

Aseem Agarwala received the PhD degree in June 2006 from the University of Washington's Computer Science & Engineering Department. He is a principal scientist at Adobe Systems, Inc., and an affiliate assistant professor at the University of Washington's Computer Science & Engineering Department. His areas of research are computer graphics, computer vision, and computational imaging. Specifically, he researches computational techniques that can help us author more expressive imagery using digital cameras. He won an Honorable Mention (runner-up) for the 2006 ACM Doctoral Dissertation Award, and is an associate editor for *ACM Transactions on Graphics*. His inventions can be found in multiple products, including Adobe Photoshop and After Effects.

Aaron Hertzmann received the BA degree in computer science and art & art history from Rice University in 1996, and the MS and PhD degrees in computer science from New York University in 1998 and 2001, respectively. He is a professor of computer science at the University of Toronto. In the past, he has worked at Pixar Animation Studios, University of Washington, Microsoft Research, Mitsubishi Electric Research Lab, Interval Research Corporation and NEC Research Institute. His awards include the MIT TR100 (2004), an Ontario Early Researcher Award (2005), a Sloan Foundation Fellowship (2006), a Microsoft New Faculty Fellowship (2006), a UofT CS teaching award (2008), the CACS/AIC Outstanding Young CS Researcher Award (2010), and the Steacie Prize for Natural Sciences (2010). He is a senior member of the IEEE.

Brian Curless received the BS degree in electrical engineering at the University of Texas at Austin in 1988 and the PhD degree from Stanford University in 1997. He is an associate professor in the Department of Computer Science & Engineering at the University of Washington. Following the PhD degree, he helped cofound the Digital Michelangelo Project as a research associate at Stanford. He joined the faculty at the University of Washington in January 1998 and spent his 2004-2005 sabbatical at Columbia University. He received the Gores Teaching Award and the Arthur Samuels Computer Science Thesis Award while at Stanford, and a US National Science Foundation (NSF) Career Award, Sloan Fellowship, and ACM Student Chapter Teaching Award while at the University of Washington. He is also co-editor-in-chief of the journal *Foundations and Trends in Computer Graphics and Vision*. He is a member of the Graphics and Imaging Laboratory (GRAIL), which is dedicated to innovation in computer graphics and computer vision. His research is focused on computational and 3D photography. He is a member of the IEEE.

Michael F. Cohen received the undergraduate degrees in art and civil engineering from Beloit College and Rutgers University, respectively, and the MS degree in computer graphics from Cornell, and the PhD degree in 1992 from the University of Utah. He is a principal researcher at Microsoft Research which he joined in 1994 from Princeton University where he served on the faculty of Computer Science. He also served on the Architecture faculty at Cornell University and was an adjunct faculty member at the University of Utah. His early work at Cornell and Princeton on the radiosity method for realistic image synthesis is discussed in his book "*Radiosity and Image Synthesis*" (coauthored by John R. Wallace). For this work he received The 1998 SIGGRAPH Computer Graphics Achievement Award. His work at the University of Utah focused on spacetime control for linked figure animation. At Microsoft, he has worked on a number of projects ranging from image-based rendering, to animation, to camera control, to artistic nonphotorealistic rendering, and most recently to computational photography applications. One project focuses on the problem of image-based rendering; capturing the complete flow of light from an object for later rendering from arbitrary vantage points, dubbed The Lumigraph. He also continued his earlier work on linked figure animation, focusing on means to allow simulated creatures to portray their emotional state. For the past decade, he has focused on computational photography applications. These range from creating new methods for low bandwidth teleconferencing, segmentation and matting of images and video, technologies for combining a set of "image stacks" as a Photomontage, to the creation of very high-resolution panoramas. Much of this work is summarized in his concept for The Moment Camera. He is a member of the IEEE.