Finding Paths through the World's Photos

Noah Snavely
University of Washington

Rahul Garg
University of Washington

Steven M. Seitz
University of Washington

Richard Szeliski
Microsoft Research

Abstract

When a scene is photographed many times by different people, the viewpoints often cluster along certain paths. These paths are largely specific to the scene being photographed, and follow interesting regions and viewpoints. We seek to discover a range of such paths and turn them into controls for image-based rendering. Our approach takes as input a large set of community or personal photos, reconstructs camera viewpoints, and automatically computes orbits, panoramas, canonical views, and optimal paths between views. The scene can then be interactively browsed in 3D using these controls or with six degree-of-freedom free-viewpoint control. As the user browses the scene, nearby views are continuously selected and transformed, using control-adaptive reprojection techniques.

1 Introduction

Image-based rendering (IBR) has long been a fertile area of research in the computer graphics community. A main goal of IBR is to recreate a compelling experience of "being there"—virtually traveling to a remote place or interacting with a virtual object. Most research in IBR has focused on the rendering aspect of this problem, seeking to synthesize photo-realistic views of a scene from a database of captured images. However, *navigation* is just as much a key part of the experience. Even for the simplest scenes (e.g., a single object), certain modes of navigation can be much more effective than others. For more complex scenes, good controls are even more critical to guide the user to interesting parts of the scene. As IBR methods scale up to handle larger and larger scenes, the problem of devising good viewpoint controls becomes increasingly important.

One solution to this problem is to carefully plan a set of desired paths through a scene and capture those views. This approach is used for many simple IBR experiences such as panoramas, object movies [Chen 1995], and moviemaps [Lippman 1980]. While effective, this kind of approach cannot leverage the vast majority of existing photos, including the billions of images in community photo collections (CPCs) found on the Internet through resources like Flickr and Google. CPCs capture many popular world landmarks from thousands of different viewpoints and illumination conditions, providing an ideal data-set for IBR [Snavely et al. 2006]. For these types of collections, the problem is to discover or devise the best controls for navigating each scene, based on the distribution of captured views and the appearance and content of the scene.

For example, consider the overhead view of a reconstruction of Statue of Liberty from 388 photos downloaded from Flickr (Figure 1). Most photos are captured from the island or from boats out in the water, and are distributed roughly along two circular arcs.

Figure 1: *Paths through photo collections.* The reconstructed camera viewpoints from hundreds of Flickr photos of the Statue of Liberty (top) reveal two clear orbits (bottom), shown here superimposed on a satellite view. We seek to automatically discover such orbits and other paths through view space to create scene-specific controls for browsing photo collections.

This viewpoint distribution suggests two natural orbit controls for browsing this scene. While this scene's viewpoints have a particularly simple structure, we have observed that many CPCs can be modeled by a combination of simple paths through the space of captured views.

While deriving such controls is a challenging research problem, using CPCs to generate controls also has major advantages. CPCs represent samples of views from places people actually stood and thought were worth photographing. Therefore, through consensus, they tend to capture the "interesting" views and paths through a scene. We leverage this observation to generate controls that lead users to interesting views and along interesting paths.

In this paper, we explore the problem of creating compelling, fluid IBR experiences with effective controls from CPCs and personal photo collections. We address both the rendering and navigation aspects of this problem. On the rendering side, we introduce new techniques for selecting and warping images for display as the user moves around the scene, and for maintaining a consistent scene appearance. On the navigation side, we provide controls that make it easy to find the interesting aspects of a scene. A key feature of these controls is that they are generated automatically from the photos themselves, through analysis of the distribution of recovered camera viewpoints and 3D feature distributions, using novel path fitting and path planning algorithms.

Our approach is based on defining view scoring functions that predict the quality of reprojecting every input photo to every new

viewpoint. Our novel scoring function is used as the basis of our paper's two main contributions. The first is a real-time rendering engine that continuously renders the scene to the user's viewpoint by reprojecting the best scoring input view, compensating for changes in viewpoint and appearance. The second contribution is a set of path planning and optimization algorithms that solve for optimal trajectories through view space, using the view scoring function to evaluate path costs.

We demonstrate our approach on a variety of scenes and for a range of visualization tasks including free-form 3D scene browsing, object movie creation from Internet photos or video, and enhanced browsing of personal photo collections.

2 Related work

Creating interactive, photo-realistic, 3D visualizations of real objects and environments is the goal of image-based rendering. In this section, we review the most closely related work in this field.

Moviemaps and Object Movies In the pioneering Moviemap project from the late 1970's and early 1980's [Lippman 1980], thousands of images of Aspen Colorado were captured from a moving car and registered to a street map. Once the images were stored, a trackball-based user interface allowed a user to interactively move through the streets by recalling and displaying images based on the user's locations. The authors noted that playing "canned" image sequences of a scene in new orders transforms the passive experience of watching a video into a very compelling *interactive* experience.

While creating the original Aspen Moviemap was a monumental task requiring more than a year to complete, more recent efforts have used omnidirectional video cameras and tracking systems to simplify the image acquisition process [Aliaga and Carlbom 2001; Taylor 2002; Uyttendaele et al. 2004]. Google StreetView has now captured entire cities in this manner.

Chen [1995] further developed the moviemap idea to enable two DOF rotational motion, allowing a user to interactively rotate an object within a hemisphere of previously captured views. Although the setup required to capture such *object movies* is simpler than with more complex scene tours, creating object movies still typically requires carefully planning and/or special equipment. EyeVision [Kanade 2001], made famous for its use in the Superbowl, presents a hardware solution for creating time-varying object movies of real-time events, but requires specially calibrated and instrumented cameras, and cannot be applied to unstructured photo collections.

In contrast, our work makes it easy to create not only object movies, but also to navigate a range of more general scenes.

View Interpolation An alternative to simply displaying the nearest image, as is done in moviemaps and object movies, is to smoothly *interpolate* between neighboring views using computer vision and image warping techniques. Early examples of these approaches include z-buffer-based view interpolation for synthetic scenes [Chen and Williams 1993], Plenoptic Modeling [McMillan and Bishop 1995], and View Morphing [Seitz and Dyer 1996]. Specialized view interpolation techniques can also be developed using *image-based modeling* techniques to reconstruct full 3D models of the scene, such as the Facade system [Debevec et al. 1996] and more recent large-scale architectural modeling systems [Pollefeys et al. 2004]. The Facade system uses the concept of *view-dependent* texture maps and *model-based stereo* to enhance the realism in their texture-mapped polyhedral models. Unfortunately, view interpolation and image-based modeling methods are only as good as the 3D computer vision algorithms used to build the models or depth maps, which still have problems with the kinds of highly variable photographs found on the Internet, though promising progress is being made on this problem [Goesele et al. 2007].

Light Fields To best capture the full visual realism inherent in the scene, ray-based rendering can be used to synthesize novel views. Examples of this approach include the Light Field [Levoy and Hanrahan 1996] and Lumigraph [Gortler et al. 1996], the Unstructured Lumigraph [Buehler et al. 2001], and Concentric Mosaics [Shum and He 1999]. Our work is closest to the Unstructured Lumigraph in that an arbitrary set of images can be used as input and for rendering. Unfortunately, these techniques require a *lot* of input images to get high-fidelity results, even when 3D geometric proxies are used, so they have yet to be applied to large-scale scenes. Applying ray-based methods to photos taken under variable conditions (illumination, weather, exposure, etc.) is particularly problematic, in that the appearance conditions may vary incoherently between different pixels of the same rendered image.

Photo Tourism Our original work on Photo Tourism [Snavely et al. 2006] presented a robust technique for registering and browsing photos from both Internet and personal photo collections. The work presented here is a significant advance over Photo Tourism and Microsoft's related Photosynth[1] in two primary ways.

First, we present a fundamentally different rendering engine that addresses key limitations of these previous methods. The IBR approach that underlies Photo Tourism and Photosynth is based on the assumption that the scene is well approximated by a planar facades, enabling good results on scenes like the Notre Dame Cathedral, the Trevi Fountain, and Half Dome. These same techniques break down, however, for general objects that contain many sides (e.g., statues, monuments, people, plants, etc.) and for large rotational motions. In addition, navigation in Photosynth and Phototourism is based on the user selecting a photo and moving to it, and does not support the fluid, free-form 6-DOF navigation capabilities common in games and other interactive 3D applications. Our approach addresses both of these issues.

Second, a major focus of our paper is discovering scene-specific controls by analyzing the distribution of camera viewpoints, appearance, and scene geometry. Achieving this goal enables much more natural and efficient exploration of many scenes that are currently cumbersome to navigate using Photo Tourism and Photosynth. The benefits of our approach become particularly apparent as the scenes become more complex.

3 System overview

Our system takes as input a set of photos taken from a variety of viewpoints, directions, and conditions, taken with different cameras, and potentially with many different foreground people and objects. From this input, we create an interactive 3D browsing experience in which the scene is depicted through photographs that are registered and displayed as a function of the current viewpoint. Moreover, the system guides the user through the scene by means of a set of automatically computed controls that expose orbits, panoramas, and interesting views, and optimal trajectories specific to that scene and distribution of input views.

Our system consists of the following components:

A set of input images and camera viewpoints. The input is an unstructured collection of photographs taken by one or more photographers. We register the images using structure-from-motion and pose estimation techniques to compute camera viewpoints.

Image reprojection and viewpoint scoring functions that evaluate the expected quality of rendering each input image at any possible camera viewpoint. The reprojection process takes into account such factors as viewpoint, field of view, resolution, and image appearance to synthesize high quality rendered views. The viewpoint scoring function can assess the quality of any possible rendered

[1] http://labs.live.com/photosynth/

view, providing a basis for planning optimal paths and controls through viewpoint space.

Navigation controls for a scene. Given the distribution of viewpoints in the input camera database and view selection function, the system automatically discovers scene-specific navigation controls such as orbits, panoramas, and representative images, and plans optimal paths between images.

A rendering engine for displaying input photos. As the user moves through the scene, the engine computes the best scoring input image and reprojects it to the new viewpoint, transformed geometrically and photometrically to correct for variations. To this end, we introduce an *orbit stabilization* technique for geometrically registering images to synthesize motion on a sphere, and an appearance stabilization technique for reducing appearance variation.

A user interface for exploring the scene. A 3D viewer exposes the derived controls to users, allowing them to explore the scene using these controls, move between different parts of the scene, or simply fly around using traditional free-viewpoint navigation. These controls can be sequenced and combined in an intuitive way.

These components are used in the three main stages of our system. First, an offline structure from motion process recovers the 3D locations of each photograph. Next, we introduce functions to evaluate the quality of reprojecting each input image to any possible new camera viewpoint. This information is used to automatically derive controls for the scene and optimal paths between images. Finally, the scene can be browsed in our interactive viewer. The following sections present these components in detail.

4 Scene reconstruction

We use our previously developed structure from motion system to recover the camera parameters for each photograph along with a sparse point cloud [Snavely et al. 2006]. The system first detects SIFT features in each of the input photos [Lowe 2004], matches features between all pairs of photos, and finally uses the matches to recover the camera positions, orientations, and focal lengths, along with a sparse set of 3D points. For efficiency, we run this system on a subset of the photos for each collection, then use pose estimation techniques to register the remainder of the photos. A more principled approach to reconstructing large image sets is described in [Snavely et al. 2008].

A sample of the inputs and outputs of this procedure for the Statue of Liberty data set is shown in Figure 1.

5 Viewpoint Scoring

Our approach is based on 1) the ability to reproject input images to synthesize new viewpoints, and 2) to evaluate the expected quality of such reprojections. The former capability enables rendering, and the latter is needed for computing controls that move the viewer along high quality paths in viewpoint space. In this section we describe our approach for evaluating reprojection quality.

First some terminology. We assume we are given a database of input images $\mathcal{I}$ whose camera parameters (intrinsics and extrinsics) have been computed. The term *camera* denotes the viewing parameters of an input image. The term *image* denotes an *input* photo I, with associated camera, from the database. The term *view* denotes an *output* photo v that we seek to render. A view is produced by *reprojecting* an input photo, through a rendering process, to the desired new viewpoint.

We wish to define a *reprojection score* $S(I, v)$ that rates how well a database image I can be used to render a new view v. The best reprojection is obtained by maximizing $S(I, v)$ over the database, yielding a *viewpoint score* $S(v)$:

$$S(v) = \max_I S(I, v) \tag{1}$$

Ideally, $S(I, v)$ would measure the difference between the synthesized view and a real photo of the same scene captured at v. Because we do not have access to the real photo, we instead use the following three criteria:

1. Angular deviation: the relative change in viewpoint between I and v should be small.
2. Field of view: the projected image should cover as much of the field of view as possible in v.
3. Resolution: I should be of sufficient resolution to avoid blur when projected into v.

For a given image and viewpoint, each of these criteria is scored on a scale from 0 to 1. To compute these scores, we require a geometric proxy for each image in order to reproject that image into a view v; the proxy geometry is discussed in Section 7.2.

The angular deviation score $S_{\text{ang}}(I, v)$ is proportional to the angle between rays from the current viewpoint through a set of points in the scene and rays from image I through the same points. This is akin to the *minimum angular deviation* measure used in Unstructured Lumigraph Rendering [Buehler et al. 2001]. Rather than scoring individual rays, however, our system scores the entire image by averaging the angular deviation over a set of 3D points observed by I. These points, $\text{Pts}(I)$, are selected for each image in a preprocessing step. To ensure that the points are evenly distributed over the image, we take the set of 3D points observed by I, project them into I, and sort them into a 10×10 grid of bins defined on the image plane, then select one point from each non-empty bin. The average angular deviation is computed as:

$$S'_{\text{ang}}(I, v) = \frac{1}{n} \sum_{p \in \text{Pts}(I)} \text{angle}(p - \mathbf{p}(I), p - \mathbf{p}(v)). \tag{2}$$

where $\mathbf{p}(I)$ is the 3D position of camera I, $\mathbf{p}(v)$ is the 3D position of v, and $\text{angle}(a, b)$ gives the angle between a and b. The average deviation is clamped to a maximum value of $\alpha_{\max}$ (for all our examples, we set $\alpha_{\max} = 12°$), and mapped to the interval $[0, 1]$:

$$S_{\text{ang}}(I, v) = 1 - \frac{\min(S'_{\text{ang}}(I, v), \alpha_{\max})}{\alpha_{\max}}. \tag{3}$$

The field-of-view score $S_{\text{fov}}(I, v)$ is computed by projecting I onto its proxy geometry, then into v, and computing the area of the view that is covered by the reprojected image. We use a weighted area, with higher weight in the center of the view, as we found that it is generally more important to cover the center of the view than the boundaries. The weighted area is computed by dividing the view into a grid of cells, $\mathcal{G}$, and accumulating weighted contributions from each cell:

$$S_{\text{fov}}(I, v) = \sum_{G_i \in \mathcal{G}} w_i \frac{\text{Area}(\text{Project}(I, v) \cup G_i)}{\text{Area}(G_i)}, \tag{4}$$

where $\text{Project}(I, v)$ is the polygon resulting from reprojecting I into v (if any point of the projected image is behind the view, Project returns the empty set).

Finally, the resolution score $S_{\text{res}}(I, v)$ is computed by projecting I into v and finding the average number of pixels of I per screen pixel. This is computed as the ratio of the number of pixels in I to the area, in screen pixels, of the reprojected image $\text{Project}(I, v)$:

$$S'_{\text{res}}(I, v) = \frac{\text{Area}(I)}{\text{Area}(\text{Project}(I, v))}. \tag{5}$$

If this ratio is greater than 1, then, on average, the resolution of I is sufficient to avoid blur when I is projected onto the screen (we use

mip-mapping to avoid aliasing). We then transform S'_{res} to map the interval $[\text{ratio}_{\min}, \text{ratio}_{\max}]$ to $[0, 1]$:

$$S_{\text{res}}(I, v) = \text{clamp}\left(\frac{S'_{\text{res}}(I, v) - \text{ratio}_{\min}}{\text{ratio}_{\max} - \text{ratio}_{\min}}, \epsilon, 1\right), \quad (6)$$

where $\text{clamp}(x, a, b)$ clamps x to the range $[a, b]$. We use values of 0.2 and 1.0 for $\text{ratio}_{\min}$ and $\text{ratio}_{\max}$, and enforce a non-zero minimum resolution score ϵ because we favor viewing a low-resolution image rather than no image at all.

The three scores are multiplied to give the view score S:

$$S(I, v) = S_{\text{ang}}(I, v) \cdot S_{\text{fov}}(I, v) \cdot S_{\text{res}}(I, v). \quad (7)$$

6 Scene-specific navigation controls

The development of controls for navigating virtual 3D environments dates back to at least the work of Sutherland [1968]. Since then, such controls have appeared in numerous settings: games, simulations, mapping applications such as Google Earth, etc. Providing good navigation controls is critical for 3D interfaces in general, whether they are based on a 3D model, IBR, or other scene representations; without good exploration controls it can be easy to get lost in a scene. But even beyond simply keeping the user oriented, navigation controls should make it easy to carry out some set of navigation tasks [Tan et al. 2001]. We focus mainly on tasks a user unfamiliar with a scene might want to perform: familiarizing oneself with its basic layout and finding its interesting parts.

In general, controls that facilitate these exploration tasks are scene-specific. One reason is that certain types of controls naturally work well for certain types of content. For instance, Ware and Osborne [1990] showed that for scenes comprised of a dominant object, users prefer controls for orbiting the scene (the *scene-in-hand* metaphor) over controls that let the user pan the camera and move it forward and backward (the *flying vehicle* metaphor). A second reason why good controls are scene-specific is that different scenes have different parts that are "interesting." For instance, in a virtual art museum, a good set of controls might naturally lead a user from one painting to the next. Indeed, some approaches, such as Galyean's River Analogy [1995], simply move users automatically along a pre-specified path, but give the user some freedom to control certain parameters, such as viewing direction and speed.

CPCs can be helpful in creating controls for exploration tasks, as they represent samples of how people actually experienced the scene, where they stood, and what views they found interesting [Simon et al. 2007]. Accordingly, the distribution of samples can help inform what controls would help a user find and explore interesting views, e.g., orbit controls for the Statue of Liberty. Of course, the regions near the input samples will also be the areas where we can likely render good views of the scene. We take advantage of this information through a set of automatic techniques for deriving controls from a reconstructed scene. The result of this analysis is a set of *scene-specific controls*. For instance, the Statue of Liberty scene shown in Figure 1 might have two scene-specific controls, one for the inner orbit, and one for the outer orbit.

In the rest of this section, we describe the navigation modes of our system, focusing particularly on how scene-specific controls are discovered.

6.1 Navigation modes

Our system supports three basic navigation modes:

1. Free-viewpoint navigation.
2. Constrained navigation using scene-specific controls.
3. Optimized transitions from one part of the scene to another.

Free-viewpoint navigation. The free-viewpoint navigation mode allows a user to move around the scene using standard 6-DOF (3D translation, pan, tilt, and zoom) "flying vehicle" navigation controls, as well as an *orbit* control.

While free-viewpoint controls give users the freedom to move wherever they choose, they are not always the easiest way to move around complex scenes, as the user has to continually manipulate many degrees of freedom while (at least in IBR) ideally staying near the available photos.

Scene-specific controls. Our system supports two types of scene-specific controls: orbits and panoramas. Each such control is defined by its type (e.g., orbit), a set of viewpoints, and a set of images associated with that control. For an orbit control, the set of viewpoints is a circular arc of a given radius centered at and focused on a 3D point; for a panorama the set of viewpoints is a range of viewing directions from a single 3D nodal point. When a control is active, the user can navigate the corresponding set of viewpoints using the mouse or keyboard. In addition to scene-specific controls, we also compute a set of representative *canonical images* for a scene.

Transitions between controls. The final type of control is a transition between scene-specific controls or canonical images. Our interface allows a user to select a control or image. The user's viewpoint is then moved on an automated path to the selected destination. The transition is computed using a new path planning algorithm that adapts the path to the database images, as described in Section 8. This method of directly selecting and moving to different parts of the scene is designed to make it easy to find all the interesting views.

6.2 Discovering controls

Once a scene is reconstructed, our system automatically analyzes the recovered geometry to discover interesting orbits, panoramas, and canonical images.

Orbit detection. We define an orbit to be a distribution of views positioned on a circle all converging on (looking at) a single point. We further constrain the point of convergence to lie on the axis passing through the center of the circle, which in our implementation must be perpendicular to the ground plane. The height of this convergence point determines the tilt at which the object of interest is viewed. Because full 360° view distributions are uncommon, we allow an orbit to occupy a circular arc. We wish to find orbits that optimize the following objectives, as illustrated in Figure 2:

- **quality**: maximize the quality of rendered views everywhere along the arc.
- **length**: prefer arcs that span large angles.
- **convergence**: prefer views oriented towards the center of the orbit.
- **object-centered**: prefer orbits around solid objects (as opposed to empty space).

Given these objectives, the problem of detecting orbits involves 1) defining a suitable objective function, 2) enumerating and scoring candidate orbits, and 3) choosing zero or more best-scoring candidates. One could imagine many possible techniques for each of these steps; in what follows, we describe one approach that we have found to work quite well in practice.

We first define our objective function for evaluating orbits. We note that an orbit is fully specified by a center orbit axis o and an image I; the image defines the radius of the circle (distance of camera center from o), and the convergence point p_{focus} on the orbit axis (p_{focus} is the closest point on the axis o to the optical axis of I). Assume further that I is the point on the arc midway between the arc endpoints.

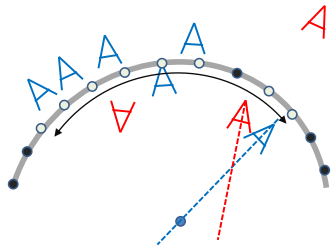

Figure 2: *Scoring an orbit.* An orbit is evaluated by regularly sampling viewpoints along the arc. For each such position, we want to find a nearby image with a high reprojection score that is oriented towards the orbit center (thus eliminating the red cameras). The light samples score well on these objectives while the black samples do not. We search for large orbits where the sum of the sample scores is high, and that do not contain large low-scoring gaps.

We define our objective scoring function, $S_{\text{orbit}}(o, I)$, as the sum of individual view scores, $S_{\text{orbit}}(o, I, \theta)$, sampled at positions θ along the arc. To compute $S_{\text{orbit}}(o, I, \theta)$ at a sample location $v(\theta)$ (the view on the arc at angle θ from I), we look for support for that view in the set of database images $\mathcal{I}$. In particular, we score each image $J \in \mathcal{I}$ based on (a) how well J can be used to synthesize view v (estimated using our reprojection score $S(J, v)$), and (b) whether J is looking at the orbit axis. $S_{\text{orbit}}(o, I, \theta)$ is then the score of the best image J at $v(\theta)$:

$$S_{\text{orbit}}(o, I, \theta) = \max_{J \in \mathcal{I}} \{S(J, v(\theta)) \cdot f_o(J)\}. \tag{8}$$

The convergence score f_o is defined as:

$$f_o(J) = \max(0, 1 - \frac{\psi}{\psi_{max}}) \tag{9}$$

where $\psi = \text{angle}(\mathbf{v}(J), p_{\text{focus}} - \mathbf{p}(J))$, i.e., the angle between the viewing direction $\mathbf{v}(J)$ and the ray from the optical center $\mathbf{p}(J)$ of J to p_{focus} (we use a value of $\psi_{max} = 20^\circ$). This term down-weights images for which p_{focus} is not near the center of the field of view.

We place a few additional constraints on the images J considered when computing $S_{\text{orbit}}(o, I, \theta)$:

- p_{focus} must be in the field of view of J.
- The tilt of J above the ground plane is less than 45° (orbits with large tilt angles do not produce attractive results).
- There are a sufficient number (we use $k = 100$) of 3D points visible to J whose distance from J is less than the orbit radius. We enforce this condition to ensure that we find orbits around an object (as opposed to empty space).

We compute $S_{\text{orbit}}(o, I, \theta)$ at every degree along the circle $-180 < \theta \leq 180$. For simplicity, we refer to these samples as s_θ. A good orbit will have a long arc of relatively high values of s_θ. Simply summing the values s_θ, however, could favor orbits with a few sparsely scattered good scores. Instead, we explicitly find a long chain of uninterrupted good scores centered around I, then sum the scores on this chain. We define this chain as the longest consecutive interval $[-\theta_i, \theta_i]$ such that the maximum s_θ in each subinterval of width 15° is at least $\epsilon = 0.01$. This definition allows for small "gaps," or intervals with low scores, in the arc. If this longest chain subtends an angle less than 60°, the score of the orbit is zero. Otherwise, the score is the sum of the individual scores in the chain:

$$S_{\text{orbit}}(o, I) = \sum_{-\theta_i}^{\theta_i} s_\theta. \tag{10}$$

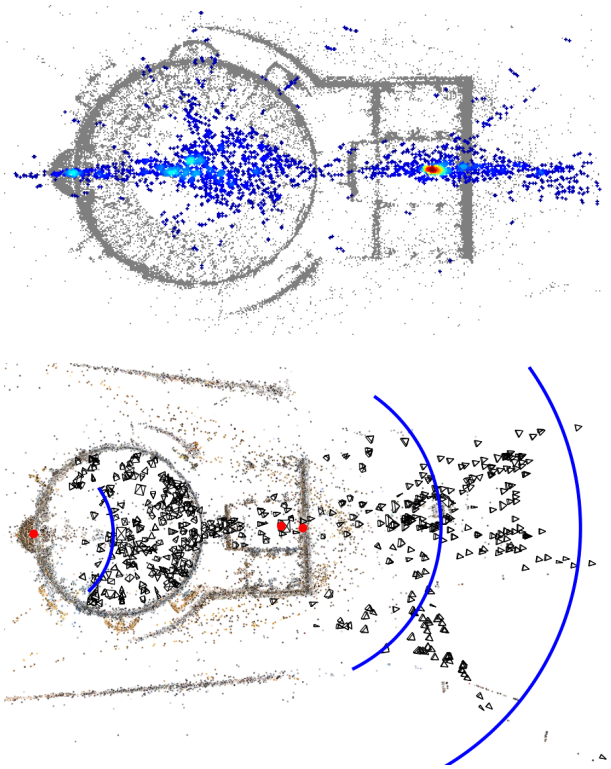

Figure 3: *Density function and detected orbits for the Pantheon dataset.* The top image shows each intersection point superimposed on the point cloud of the Pantheon. The color of each point corresponds to the density of points in its neighborhood (a red point has the highest density, a dark blue point the lowest). There are several clear maxima of this function, including a point just behind the front facade and a point at the altar (at the extreme left of the figure). The bottom image shows the three detected orbits as blue arcs centered at the red orbit points.

Our strategy for enumerating candidate orbits operates in two stages. First, we compute good orbit axes, by finding axes in 3D space on which many database images are converged. Second, we evaluate $S_{\text{orbit}}(o, I)$ only at database images I.

To identify a set of candidate orbit axes, we take an approach similar to that of Epshtien *et al.*[2007] and use the idea that an object of interest will often occupy the center of the field of view of an image. We first project all cameras onto the ground plane and consider the 2D intersections of the optical axes of all pairs of cameras (discarding intersection points which lie in back of either camera, or which are not approximately equidistant from both cameras). We compute the density D of these intersection points at each point x:

$$D(x) = \text{number of intersection points within distance } w \text{ of } x.$$

(we use $w = 0.02$, although this value should ideally depend on the scale of the scene). The local maxima in this density function identify vertical axes in the scene which are at the center of many different photos, and are thus potentially interesting. A plot of the density function for the Pantheon dataset is shown in Figure 3.

Next, we find the point with the highest density $D_{\max}$, then select all local maxima (points that have the highest density in a circle of radius w and that have a density at least $0.3D_{\max}$). These points form the set of candidate orbit axes.

The next step is to find arcs centered at these axes. We form a set of candidate orbits by considering all pairings of orbit axes o and input images I and evaluate $S_{\text{orbit}}(o, I)$ for each such combination.

We only accept orbits that satisfy the three constraints enumerated above, i.e., that the point of convergence is in the field of view, the tilt is less than 45°, and a sufficient number of points are visible in front of the orbit radius.

We now have a set of orbits and a score for each orbit. To form a final set of orbits, we select the orbit with the highest score, remove it and all similar orbits from the set of candidates, then repeat, until no more orbits with a score of at least 0.5 times the maximum score remain. Two orbits are deemed similar if the area of intersection of the two circles defined by the orbits is at least 50% of their average area. Detected orbits for the Pantheon collection are shown in Figure 3.

When computing viewpoint scores for orbit samples, we use a default vertical field of view of 50 degrees and a default screen resolution of 1024x768 (for the purposes of computing the field of view and resolution scores). Extending the approach to automatically compute a good field of view for viewing a given orbit is an interesting topic for future work.

Panorama detection. A panorama consists of a set of images taken close to a nodal point. Similar to orbits, a good panorama has good views available from a wide range of directions. To find panoramas in a scene, we first consider each image I to be the center of a candidate panorama and compute a panoramas score $S_{\text{pano}}(I)$ for each candidate. $S_{\text{pano}}(I)$ is computed as the sum of view scores S for a range of viewing directions around I:

$$S_{\text{pano}}(I) = \sum_{\phi=-5^\circ}^{25^\circ} \sum_{\theta=0^\circ}^{360^\circ} \max_{J \in \mathcal{I}} S(J, v_I(\theta, \phi)) \qquad (11)$$

where $v_I(\theta, \phi)$ is the viewpoint located at the optical center of image I with viewing direction given by angles θ (pan) and ϕ (tilt). We select the top scoring candidate panorama I^*, remove all images that have a non-zero reprojection score for some view $v_{I^*}(\theta, \phi)$, then repeat this process until no remaining candidate's score is above a threshold.

Choosing canonical images. To find canonical images, we use the scene summarization algorithm of Simon *et al.*[2007]. This algorithm seeks to capture the essence of a scene through a small set of representative images that cover the most popular viewpoints. It selects the images by clustering them based on their SIFT feature matches, then choosing a representative image for each cluster. We also choose a representative image for each detected orbit and panorama, which is shown as a thumbnail in the scene viewer. As in [Simon et al. 2007], we represent each image I as a feature incidence vector f_I. f_I has an entry for each 3D point p_j in the scene (or, in the case of orbits, every 3D point inside the orbit circle); $f_I(j) = 1$ if p_j is visible in I and 0 otherwise. These vectors are then normalized. For a given set of images $\mathcal{S}$, the representative image is chosen as: $\arg\max_{I_j \in \mathcal{S}} \sum_{I_k \in \mathcal{S}, I_j \neq I_k} f_{I_j} \cdot f_{I_k}$, i.e., the image whose normalized feature vector is most similar to those of all other images in $\mathcal{S}$.

7 Scene viewer

The 3D reconstruction and derived controls are used in our interactive scene viewer. The viewer situates the user in the scene with the object, exposes the derived controls to the user, and depicts the scene by continually selecting and warping appropriate images as the user moves. This section describes the navigation interface, view selection, and rendering components of the viewer.

7.1 Navigation interface

Our viewer supports standard 3D translation, panning, and zooming controls, as well as an orbit control, which rotates the camera about a fixed 3D point or axis. Typically, the orbital motion is constrained to a ring around an orbit axis, as many objects are viewed from a single elevation, but our viewer also supports orbital motion on a sphere. When the user is moving on a discovered orbit, the orbit axis is automatically set to be the axis discovered for that control. Alternatively the user can manually specify an orbit point by clicking on a 3D point in the scene. Once an orbit point is defined, the user can drag the mouse left and right to orbit around a vertical axis, or up and down to move the viewpoint vertically (when two dimensional orbital motion is enabled). The process is rapid and seamless: the user simply shift-clicks on a point in the image (the closest 3D feature defines the orbit point), and the orbit begins as soon as the mouse is moved.

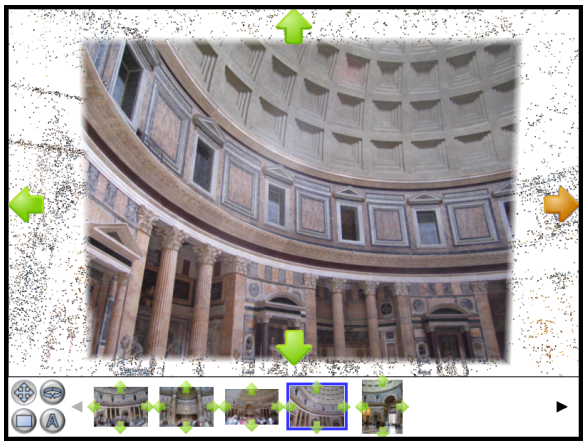

Figure 4: *Scene viewer interface.* The scene viewer displays the currently photo in the main view, and shows suggested controls in the thumbnail pane at the bottom of the screen. The pane is currently showing detected panoramas. Arrows on the sides of the screen indicate which directions the user can pan the view.

The user interface shows the pre-defined controls in a thumbnail pane at the bottom of the screen (see Figure 4). The user can choose between displaying pre-defined panoramas, orbits, and canonical images. Each control is represented with a thumbnail in this pane. The thumbnails are annotated with small arrow icons to show the type of control. This pane is designed to make it easy to find and explore the interesting parts of the scene.

When the user clicks on a thumbnail, the system computes a path from the current image to the selected control as described in Section 8 and animates the camera along that path. When the user arrives at a panorama, left, right, up, and down arrows are drawn on the sides of the screen indicating directions in which more images can be found, as as shown in Figure 4. For an orbit, left and right "orbit" arrows appear. To determine if a particular arrow cue should be shown, the system computes the viewpoint score for the view the user would see by moving in that direction. If the score is above a threshold, the arrow is displayed.

7.2 Rendering

As the user moves through the scene, the viewer continually chooses an image to display based on the reprojection score, $S(I, v)$, which rates how well each database image I can be used to render the current viewpoint v (Section 5). The image with the top score is selected as the next image to be displayed. If no image has a non-zero viewpoint score, only the point cloud is displayed.

If the input images densely sample the space of all viewpoints (as in Quicktime VR object movies and moviemaps), the rendering process is straightforward—simply display the photo corresponding to the desired viewpoint. In practice, however, casually ac-

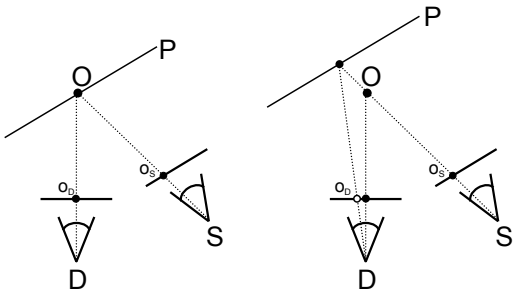

Figure 5: *Proxy planes should intersect the orbit point.* Left: to warp an image from view S to D, image S is projected onto the *proxy* plane P, which is then rendered into D. P passes through the orbit point o, ensuring that o is rendered to the correct position in D. Right: P does not pass though the orbit plane, causing o to be rendered to the wrong position.

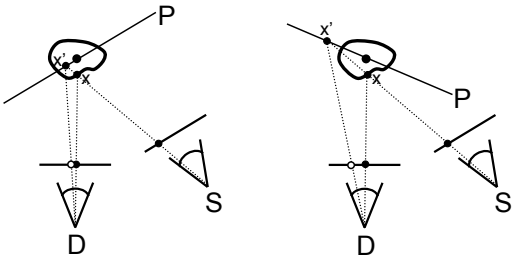

Figure 6: *Proxy plane orientation.* Left: If P is parallel to S's image plane, a point x near the orbit point will get mapped to a nearby point x' on P, causing a small error in D. Right: An oblique choice of P will generally result in larger errors.

quired image collections tend to be incomplete and irregularly sampled, with objects centered and rotated differently in each image. Hence, it is necessary to warp each image to better match the desired viewpoint. This is the function of our rendering engine.

We warp an image to match a desired viewpoint by projecting the image onto a geometric proxy. Our system normally renders the scene using planar proxies for the scene geometry, but can also render using a dense 3D model, if available.

Warping with proxy planes. When using planar proxy geometry, we associate a plane with each image and render the image by projecting it onto the plane and back into the virtual view. Planar proxies are also used in the Photo Tourism system, which fits planes to image points for use in transitions. While these best-fit planes work well for some scenes and navigation modes, they can produce jerky motion in situations where the user is moving rapidly through a wide range of views. This is especially true when orbiting; while the viewer is fixated on a particular object, the per-image best-fit planes can stabilize different parts of the scene (including the background) in different images, causing the object to jump around from frame to frame (please see the video for a demonstration).

Our solution to this problem is extremely simple but effective. Suppose the user is orbiting around an orbit point o, indicating an object of interest, and suppose we wish to render the scene captured by a "source" photo S into the "destination" viewpoint D. Consider a proxy-plane P in the scene. We compute the warp by perspectively projecting S onto P, then back into D. As shown in Figure 5, making P intersect the orbit point o ensures that o projects to the correct location in D. Hence, we can *stabilize* o in the rendered images (make o project to the same pixel (x, y) in all views) by 1) choosing P to intersect o for each input view, and 2) orienting the rendered views so that the viewing ray through (x, y) for each view passes through o. While any choice of P that passes through o will suffice, choosing P to be parallel to the image plane of S results in well-behaved warps (Figure 6). When we wish to stabilize an orbit axis, rather than a single point, we choose the normal to P to be the projection of the viewing direction of S onto the plane orthogonal to the axis. We call this form of image stabilization *orbit stabilization*.

Orbit stabilization performs a similar function to software anti-shake methods that reduce jitter in video. However, it has the advantage of performing a *globally-consistent* stabilization, producing the effect of rotation about a single center, and avoiding the drift problems that can occur with frame-to-frame video stabilization methods. Note also that orbit stabilization does not require any knowledge of scene geometry, although it does require known camera viewpoints and a reasonable orbit point.

Our system defaults to using best-fit planar proxies, until an orbit point is selected, at which point it switches to orbit stabilization. The user can also opt to use best-fit planes even when an orbit point is selected, which can produce better results if the scene is truly planar (as in the video of orbiting the facade of the Notre Dame cathedral).

Warping with a 3D model. When a 3D scene model is available, it can be used in place of the planar proxy to further improve rendering quality. In particular, Goesele *et al.*[2007] demonstrated a multi-view stereo method capable of reconstructing dense 3D models from images on Flickr. To render an image with a 3D proxy, we project the image onto the proxy and back into the image, and place an additional plane in back of the model to account for unmodeled geometry.

Using a 3D model for rendering usually results in a more realistic experience, but can also suffer from artifacts resulting from holes in the model or from projecting foreground objects onto the geometry. In our experience, planar proxies tend to produce less objectionable artifacts in these situations.

Rendering the scene. The scene is rendered by first drawing a background layer consisting of the reconstructed point cloud drawn on top of a solid color, then rendering the currently selected image, projected onto its proxy plane. Rather than instantaneously switching between images as new ones are selected for display, images are faded in and out. The system maintains an alpha value for each image; whenever a new image is selected for display, the alpha of the previous image decays to zero, and the alpha of the new image rises to one, over a user-specified interval of time. When the user is moving on a planned path, the system can look ahead on the path and fade images in early, so that each image reaches full opacity when it becomes optimal. When the user moves on a free-form path, this prediction is more difficult, so the system starts fading in an image at the moment it becomes optimal.

If the user is moving fairly quickly, multiple images can simultaneously have non-zero alphas. We blend the images by first normalizing all alphas to sum to one, then compositing the rendered images in an off-screen buffer. This image layer is then composited onto the background layer.

Once the photos are cached in memory, our object movie viewer runs at over 30 frames per second with up to 1700 photos (the most we tried) on a machine with a 3.3GHz processor and an nVidia Quadro FX graphics card.

8 Path planning

The discrete controls supported by our system move the user automatically on a path from one image to another. Unlike in the Photo Tourism system [2006], which performs two-image morphs, our system can display multiple images in between the endpoints. This makes our system much more effective for moving on long, complex paths. In order to make effective use of intermediate images, we *plan* paths between pairs of images. Our path planning algorithm attempts to find a path along which there are many good views, so that at any point on the path the user is presented with a high quality view of the scene. An additional benefit of constraining the path to pass near photos in the database is that it will be more likely to be physically plausible, e.g., to not pass through walls or other obstacles.

Path planning, often been used in robotics, has also been used in computer graphics for computing camera paths through a 3D environment. For instance, Drucker and Zeltzer [1994] use planning to help create paths through a 3D scene which satisfy task-based objectives (such as focusing on a specific object) and geometric constraints. In the realm of IBR, Kang *et al.*[2000] analyze a sequence of images to predict which views or portions of views can be synthesized. In our work, we extend these ideas to use our prediction score to plan good camera paths.

In order to find the best path between two images given a database of existing image samples $\mathcal{I}$, suppose we have a cost function $Cost_{\mathcal{I}}(v)$ (to be defined shortly) over the space of possible viewpoints, where $Cost$ is low for viewpoints close to existing samples, and large for distant views. The optimal path between two viewpoints is then defined as the lowest-cost path (geodesic) connecting them.

The dimension of the viewpoint space, however, is relatively high (five in our interface, or six if zoom is included), and therefore this continuous approach is computationally expensive. We instead find a discrete solution by first computing an optimal piecewise linear path through the existing camera samples, and then smooth this path. This discrete problem can be posed as finding a shortest path in a transition graph G_T whose vertices are the camera samples $\mathcal{I}$.

G_T contains a weighted edge between every pair of images (I_j, I_k) that see common 3D points. One component of the edge weight $w(I_j, I_k)$ is the *transition cost* $\tau(I_j, I_k)$, i.e., the integral of the cost function over a straight-line path $\gamma(t)$ between I_j and I_k. Because (I_j, I_k) represents a two-image transition, we compute each edge weight using a cost function $Cost_{I_j,I_k}(v)$ that restricts the rendering process to consider only I_j and I_k when generating in-between views on $\gamma(t)$. Thus,

$$\tau(I_j, I_k) = \int_0^1 Cost_{I_j,I_k}(\gamma(t))\, dt. \tag{12}$$

We define $Cost_{I_j,I_k}$ by first considering the cost of rendering a new viewpoint with one of the images samples, I_j. In Section 5 we defined a scoring function S for computing how well an image I_j can be used to synthesize a new view v. We now turn this scoring function into a cost function:

$$Cost_{I_j}(v) = e^{\alpha(1-S(I_j,v))} - 1. \tag{13}$$

This function evaluates to 0 when $S(I_j, v) = 1$, and to $e^{\alpha} - 1$ when $S = 0$ (for our experiments, we use a value $\alpha = 8$). We now define the two-view cost function $Cost_{I_j,I_k}$ over the path $\gamma(t)$ as the weighted sum of the single viewpoint cost function:

$$Cost_{I_j,I_k}(\gamma(t)) = (1-t)\, Cost_{I_j}(\gamma(t)) + t\, Cost_{I_k}(\gamma(t)). \tag{14}$$

We approximate the integral in Eq. (12) by computing the average value of $Cost_{I_j,I_k}$ at 30 samples along $\gamma(t)$.

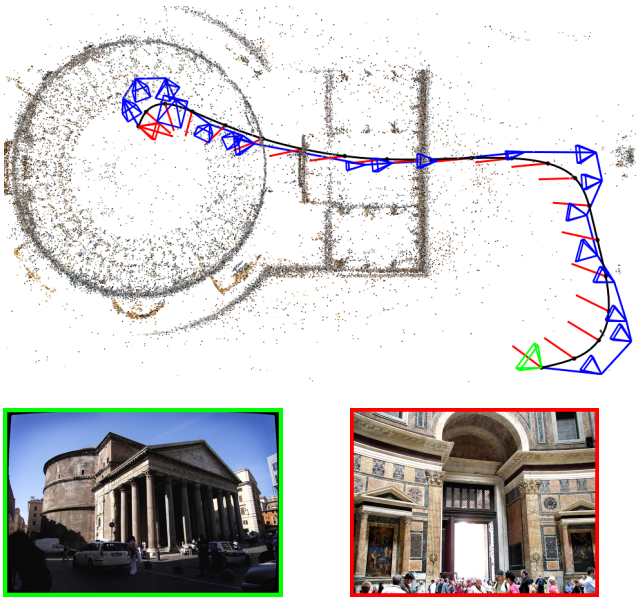

Figure 7: *Transition computed using path planning.* We use our algorithm to compute a transition from an image outside the Pantheon (green) to an image inside (red). The blue cameras are the intermediate nodes visited on the transition graph, and the blue line is the linearly interpolated path. The black curve shows the path resulting from smoothing this initial discrete path, and the red lines indicate the viewing directions at samples along this path.

If we weight edges using the transition cost alone, the shortest paths in the graph are not guaranteed to be smooth, and could traverse a convoluted path through viewpoint space. To avoid such paths, we add a smoothness cost $\sigma(I_j, I_k)$ to the edge weight w. This cost is simply the length of the edge in viewpoint space, which we compute as a weighted combination of the difference in position and orientation between I_j and I_k:

$$\sigma(I_j, I_k) = \| \mathbf{p}(I_j) - \mathbf{p}(I_k) \| + \beta \operatorname{angle}(\mathbf{v}(I_j), \mathbf{v}(I_k)), \tag{15}$$

where $\mathbf{p}(I)$ is the 3D position of image I and $\mathbf{v}(I)$ is its viewing direction. We use a value $\beta = 3.0$ in our experiments.

The final weight of an edge (I_j, I_k) is the weighted sum of the transition cost τ and the smoothness cost σ:

$$w(I_j, I_k) = \tau(I_j, I_k) + \lambda \sigma(I_j, I_k). \tag{16}$$

For our experiments, we used a value $\lambda = 400$.

Generating smooth paths. We use Djikstra's algorithm to compute a shortest path π^* between two images in G_T. π^* can also be interpreted as a piecewise linear physical path through viewpoint space. In order to produce a more continuous path for animating the camera, we smooth this initial path. First, we uniformly sample π^* to produce a sequence of viewpoint samples $v_i^0, i = 1$ to n (we use 100 samples in our implementation). We then repeatedly average each sample with its two neighbors, and with its original position v_i^0 (in order to keep the sample close to π^*):

$$v_i^{t+1} = \frac{1}{1+\mu} \left(0.5 \left(v_{i-1}^t + v_{i+1}^t \right) + \mu v_i^0 \right). \tag{17}$$

We iterate this smoothing for 50 rounds. The parameter μ controls how closely the samples match π^*, versus how smooth the path is. In our implementation we set $\mu = 0.02$, which produces nice,

smooth paths which still stay close enough to the images along π^* to produce good views. An example of a path computed between two images in the Pantheon collection is shown in Figure 7.

9 Appearance stabilization

Unstructured photo sets can exhibit a wide range of lighting and appearance variation, which can include night and day, sunny and cloudy days, and photos taken with different exposures. The simple version of our browser displays photos based only on the user's current viewpoint, which can result in large changes in scene appearance as the user moves. These large, random variations can be useful in getting a sense of the variation in appearance space of a scene, but they can also be visually distracting. To reduce this appearance variation, the user can enable a *visual similarity* mode, which limits transitions to visually similar photos, and a color compensation feature, which hides appearance changes by modifying the color balance of new images. In addition, our system allows the photo collection to be separated into classes, such as day and night, to allow the user to explicitly control the appearance state of the object.

This section describes these features in more detail. We first discuss the metric used to compute photo similarity, then describe how this metric is incorporated into the viewer and how color compensation is done. Finally, we describe how an object can be browsed in different states.

9.1 Computing image distances

To reduce the amount of appearance variation that occurs while viewing a scene, we first need a way to measure the visual distance between two images. To compute this distance, we first register the images geometrically and photometrically, then find the L_2 distance between pixel values.

Geometric alignment. To compute the distance between images I and J, we first downsample I to a resolution of 64x64 pixels, and downsample J to approximately the same sampling rate with respect to the scene, resulting in low-resolution images I' and J'.

Next, we warp J' into geometric alignment with I'. If we know the complete scene geometry, we can use this information to produce the warped version of J', but since we do not assume geometry is available, we instead use a non-rigid 2D deformation, namely thin-plate splines (TPS) [Bookstein 1989] to model the warp.

In particular, we project all 3D points visible to I into both I' and J' to form a set of 2D basis points, and compute the corresponding TPS deformation D mapping I' onto J' (so as to transform J' through an inverse warp).

Given the deformation D, for each pixel location i of I', we compute the corresponding pixel location $D(i)$ of J'; if $D(i)$ lies inside J', we sample J' at $D(i)$ using bilinear interpolation. This results in a sequence of pairs of RGB samples:

$$[I'(i_1), J'(D(i_1))], [I'(i_2), J'(D(i_2))], \ldots, [I'(i_n), J'(D(i_n))]$$

The TPS deformation D will not necessarily extrapolate well far away from the basis points, i.e., we have more confidence in the deformation near known 3D point projections. To make the image comparison more robust to misregistration, we precompute a spatial confidence map W_I for each image I. W_I is created by centering a 2D Gaussian at the projection of each 3D point observed by I, with standard deviation proportional to the scale of the SIFT feature corresponding to that point, and with height $\frac{1}{2}$. The Gaussians are then summed, sampled at each pixel, and clamped to the range $[0, 1]$.

When comparing images I and J, we sample the weight maps W_I and W_J in the same way as the images, and store the minimum of the two sampled weights, giving a sequence of weights, $w_1, w_2, \ldots, w_n$.

Figure 8: *Similarity and color compensation.* The first row shows a sequence of images, going from left to right, from an object movie of the Trevi Fountain resulting from orbiting the site. The second row shows the result of orbiting through the same path with similarity mode turned on. Note that a different set of images with more similar lighting is selected. The third row shows the same images as in the second row, but with color compensation turned on. Now the color balance of each image better matches that of the first.

Photometric alignment. After applying a spatial warp to J', we next align the color spaces of the two images. In order to achieve invariance to exposure, we use a simple gain and offset model to warp each color channel of I to match that of J. To achieve robustness to bad samples from misaligned or saturated pixels, we use RANSAC [Fischler and Bolles 1987] to compute the gain and offset, resulting in a color compensation transform $C_{I,J}$.

Finally, we compute the distance measure $\mathrm{d}(I, J)$ as the weighted average of color-shifted RGB samples:

$$\mathrm{d}(I, J) = \frac{1}{\sum w_k} \sum_{k=1}^{n} w_k \left\| C_{I,J}(I'(i_k)) - J'(D(i_k)) \right\| \quad (18)$$

using RGB values in the range $[0, 255]$.

Because it is difficult to reliably warp photos with wide baselines into alignment, we only compute image distances between pairs of photos that are relatively close to each other. In our viewer, it is still possible to move a large distance when similarity mode is enabled, via a sequence of transitions between nearby, similar images.

9.2 Browsing with similarity

At startup, the object movie viewer reads the pre-computed similarity scores. When the similarity mode is enabled, these scores are used to prune the set of possible image transitions and to favor transitions between images that are more similar. This is implemented by multiplying the reprojection score $S(I, v)$ with a similarity factor $S_{\mathrm{sim}}(I, I_{\mathrm{curr}})$, where $I\mathrm{curr}$ is the currently displayed image. To compute the function $S_{\mathrm{sim}}(I, J)$, we remap the interval $[\mathrm{d}_{\min}, \mathrm{d}_{\max}]$ to $[0, 1]$ and clamp:

$$S_{\mathrm{sim}}(I, J) = 1 - \mathrm{clamp}\left(\frac{\mathrm{d}(I, J) - \mathrm{d}_{\min}}{\mathrm{d}_{\max} - \mathrm{d}_{\min}}, 0, 1 \right). \quad (19)$$

We use values of $\mathrm{d}_{\min} = 12$ and $\mathrm{d}_{\max} = 30$. Enabling similarity mode results in the selection of a sparser set of photos, so though their visual appearance is much more stable, the motion can be more jerky.

9.3 Color compensation

The object movie viewer can also use the pairwise RGB gain and offset parameters estimated during similarity computation to disguise changes in appearance by adjusting the color balance of new

images to match that of the previously displayed image. At runtime, the viewer maintains a 3x4 color compensation matrix $C(I)$ for each image, which it applies to the rendered image using a fragment shader. When color compensation is enabled, during a transition from an image I to an image J, $C(J)$ is set to $C_{J,I}$, premultipled by the color compensation already in effect for I,

$$C(J) = C(I) \cdot C_{J,I}.$$

Examples of color corrected images are shown in Figure 8. To reduce problems with accumulated drift over time and eventually return images to their true color balance, the matrices $C(I)$ fade back to the identity transform $[I|0]$ over time.

9.4 Viewing different appearance states

As with QuickTime VR object movies, our system allows an object to be viewed in different appearance states, such as day and night. This feature requires the photos to be classified into sets corresponding to each state; once the photos are classified and a user selects a certain state, the system will only display photos from that state. At a given viewpoint, the user can toggle to any state for which a photo I with non-zero viewpoint score $S(I)$ exists.

We experimented with this feature using the Trevi Fountain data set, which contains a large number of both daytime and nighttime photos. We were able to semi-automatically classify the photos into day and night sets fairly easily, using the observation that many 3D points (corresponding to SIFT features in the original images) are highly correlated with either daytime or nighttime images. We first hand-labeled seven daytime and seven nighttime photos, then automatically propagated that information to other photos through the 3D points. In particular, we iteratively updated a set of image weights $U(I)$ and point weights $V(I) \in [-1, 1]$, where -1 corresponded to a nighttime image and 1 to a daytime image. Initializing $U(I) = 1$ for images labeled daytime and $U(I) = -1$ for images labeled nighttime, we iteratively adjusted the point and image weights with the following update equations:

$$V(p) = \frac{\sum_{I \in \mathrm{Imgs}(p)} U(I)}{\sum_{I \in \mathrm{Imgs}(p)} |U(I)|}, U(I) = \frac{1}{|\mathrm{Pts}(I)|} \sum_{p \in \mathrm{Pts}(I)} V(I),$$

where $\mathrm{Pts}(I)$ is the set of points visible in image I, and $\mathrm{Imgs}(p)$ is the set of images in which point p is visible. In other words, points that were seen in mostly "night" (resp. "day") images are labeled as "night" (resp. "day") points, and vice versa. For the Trevi Fountain data set, the update steps converged after about five iterations, and cleanly separated the images into daytime ($U(I) > 0$) and nighttime ($U(I) < 0$) sets.

Please see the accompanying video for an example of toggling between day and night states at the Trevi Fountain.

10 Results

We have applied our system to several large collections of images downloaded from Flickr. Please refer to the companion video to see interactions with these scenes in our viewer. Two of these scenes consist of dominant objects and provide an object movie experience: the Statue of Liberty, created from 388 photos, and the Venus de Milo, created from 461 images. Our system detected two orbits for the Statue of Liberty, and one orbit for the Venus de Milo. Our reconstruction of the Notre Dame Cathedral (created from 597 photos) has a wide distribution of camera viewpoints on the square in front of the Cathedral, and is therefore well suited for free-form 6-DOF navigation. This is a case where automatic orbit detection is less useful, as you can produce a good orbit from almost anywhere on the square, as shown in the video. Our reconstruction of the Trevi Fountain (1771 photos) contains a large numbers of both day- and night-time images, making this a good candidate for evaluating both appearance stabilization and also state-based modes.

We also demonstrate how our system makes it easy to create an object movie experience by manually rotating a hand-held object in front of a camera. In this case, the user manually specified a sphere of orbits, as our current implementation does not support spherical orbit detection.

Finally, we demonstrate our system with a collection of photos of the Pantheon (602 images), a relative complex scene consisting of both interior and exterior views. For this scene, our system detected three orbits, several panoramas, and a number of canonical images, including photos of the front facade, the altar, the oculus, and several sculptures inside the building. The accompanying video shows sample interactions with each of these types of controls, and demonstrates the results of our path planning algorithm. We also created a 3D slideshow of a collection of personal photos taken at the Pantheon, by planning paths between the personal photos using the community photos to fill in the gaps.

11 Discussion

We have successfully used our approach to create IBR experiences for several different community photo collections. However our approach also has several limitations. Our geometric model for orbits is a circle, whereas many paths around objects are ellipses, lines, or more freeform shapes. In the future, it would be interesting to explore the detection of more general types of paths in a scene, perhaps by unifying our path planning algorithm with our orbit and panorama detection algorithms. An additional challenge is to devise better rendering algorithms for these more general paths, as orbit stabilization is not applicable.

In our current system, zoom is handled differently than other viewing parameters when computing paths, because we found that it is difficult to produce good transitions while both adjusting the field of view and moving the virtual view. Developing a principled way of integrating zoom into our path planning and orbit detection algorithms is an interesting direction for future work.

Our color compensation method works well for images that are fairly similar, so it goes hand in hand with the similarity mode. However, because our color compensation only models simple transformations, compensating two very different images (e.g., sunny and cloudy) can result in unstable estimates and limit the number of possible transitions between images. Developing a more flexible appearance compensation model would help avoid these problems. It would be interesting to explore more sophisticated image models that detect and treat foreground objects, such as people, separately from the scene (e.g., removing them during transitions, or popping them up on their own planes).

In summary, we have developed a new approach for creating fluid 3D experiences with scene-specific controls from unstructured community photo collections. We believe that our techniques represent an important step towards leveraging the massive amounts of imagery available both online and in personal photo collections in order to create compelling 3D experiences of our world.

Acknowledgements. We thank Kevin Chiu and Andy Hou for their invaluable help with this project. This work was supported in part by National Science Foundation grants IIS-0413198, IIS-0743635, and CNS-0321235, the Office of Naval Research, Microsoft, and an endowment by Rob Short and Emer Dooley.

Many thanks to the following people for allowing us to reproduce their photos in our paper and video (the full name and Flickr user ID are listed; photos for a user can be found at `http://www.flickr.com/photos/flickr-id/`): Storm Crypt (storm-crypt), James McPherson (jamesontheweb), Wolfgang Wedenig (wuschl2202), AJP79 (90523335@N00), Tony Thompson (14489588@N00), Warren Buckley (studio85), Keith Barlow (keithbarlow), beautifulcataya (beautifulcataya), Smiley Apple (smileyapple), crewealexandra (28062159@N00), Ian Turk

(ianturk), Randy Fish (randyfish), Justin Kauk (justinkauk), Airplane Lane (photons), Katie Holmes (katieholmes), Cher Kian Tan (70573485@N00), Erin Longdo (eel), James McKenzie (jmckenzie), Eli Garrett (portenaeli), Francesco Gasparetti (gaspa), Emily Galopin (theshrtone), Sandro Mancuso (worldwalker), Ian Monroe (eean), Noam Freedman (noamf), morbin (morbin), Margrethe Store (margrethe), Eugenia and Julian (eugeniayjulian), Allyson Boggess (allysonkalea), Ed Costello (epc), Paul Kim (fmg2001), Susan Elnadi (30596986@N00), Mathieu Pinet (altermativ), © Ariane Gaudefroy (kicouette), Briana Baldwin (breezy421), Andrew Nguyen (nguy0833), Curtis Townson (fifty50), Rob Thatcher (pondskater) (rob@hypereal.co.uk), Greg Scher (gregscher).

References

ALIAGA, D. G., AND CARLBOM, I. 2001. Plenoptic stitching: A scalable method for reconstructing 3D interactive walkthroughs. In *SIGGRAPH Conf. Proc.*, 443–450.

BOOKSTEIN, F. L. 1989. Principal warps: Thin-plate splines and the decomposition of deformations. *IEEE Trans. on Pattern Analysis and Machine Intelligence 11*, 6, 567–585.

BUEHLER, C., BOSSE, M., MCMILLAN, L., GORTLER, S., AND COHEN, M. 2001. Unstructured lumigraph rendering. In *SIGGRAPH Conf. Proc.*, 425–432.

CHEN, S., AND WILLIAMS, L. 1993. View interpolation for image synthesis. In *SIGGRAPH Conf. Proc.*, 279–288.

CHEN, S. E. 1995. QuickTime VR – an image-based approach to virtual environment navigation. In *SIGGRAPH Conf. Proc.*, 29–38.

DEBEVEC, P. E., TAYLOR, C. J., AND MALIK, J. 1996. Modeling and rendering architecture from photographs: a hybrid geometry- and image-based approach. In *SIGGRAPH Conf. Proc.*, 11–20.

DRUCKER, S. M., AND ZELTZER, D. 1994. Intelligent camera control in a virtual environment. In *Proc. of Graphics Interface*, 190–199.

EPSHTIEN, B., OFEK, E., WEXLER, Y., AND ZHANG, P. 2007. Hierarchical photo organization using geometric relevance. In *ACM Int. Symp. on Advances in Geographic Information Systems*.

FISCHLER, M., AND BOLLES, R. 1987. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Readings in computer vision: issues, problems, principles, and paradigms*, 726–740.

GALYEAN, T. A. 1995. Guided navigation of virtual environments. In *SI3D '95: Proc. Symposium on Interactive 3D Graphics*, 103–104.

GOESELE, M., SNAVELY, N., SEITZ, S. M., CURLESS, B., AND HOPPE, H. 2007. Multi-view stereo for community photo collections. In *Proc. Int. Conf. on Computer Vision*.

GORTLER, S. J., GRZESZCZUK, R., SZELISKI, R., AND COHEN, M. F. 1996. The lumigraph. In *SIGGRAPH Conf. Proc.*, 43–54.

KANADE, T., 2001. Carnegie Mellon goes to the Superbowl. `http://www.ri.cmu.edu/events/sb35/tksuperbowl.html`.

KANG, S. B., SLOAN, P.-P., AND SEITZ, S. M. 2000. Visual tunnel analysis for visibility prediction and camera planning. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 2195–2202.

LEVOY, M., AND HANRAHAN, P. 1996. Light field rendering. In *SIGGRAPH Conf. Proc.*, 31–42.

LIPPMAN, A. 1980. Movie maps: An application of the optical videodisc to computer graphics. In *SIGGRAPH Conf. Proc.*, 32–43.

LOWE, D. 2004. Distinctive image features from scale-invariant keypoints. *Int. J. of Computer Vision 60*, 2, 91–110.

MCMILLAN, L., AND BISHOP, G. 1995. Plenoptic modeling: An image-based rendering system. In *SIGGRAPH Conf. Proc.*, 39–46.

POLLEFEYS, M., VAN GOOL, L., VERGAUWEN, M., EST, F. V., CORNELIS, K., TOPS, J., AND KOCH, R. 2004. Visual modeling with a hand-held camera. *Int. J. of Computer Vision 59*, 3, 207–232.

SEITZ, S. M., AND DYER, C. M. 1996. View morphing. In *SIGGRAPH Conf. Proc.*, 21–30.

SHUM, H.-Y., AND HE, L.-W. 1999. Rendering with concentric mosaics. In *SIGGRAPH Conf. Proc.*, 299–306.

SIMON, I., SNAVELY, N., AND SEITZ, S. M. 2007. Scene summarization for online image collections. In *Proc. Int. Conf. on Computer Vision*.

SNAVELY, N., SEITZ, S. M., AND SZELISKI, R. 2006. Photo tourism: exploring photo collections in 3D. In *SIGGRAPH Conf. Proc.*, 835–846.

SNAVELY, N., SEITZ, S. M., AND SZELISKI, R. 2008. Skeletal sets for efficient structure from motion. In *Proc. Computer Vision and Pattern Recognition (to appear)*.

SUTHERLAND, I. E. 1968. A head-mounted three dimensional display. In *Proc. Fall Joint Computer Conf.*, 757–764.

TAN, D. S., ROBERTSON, G. G., AND CZERWINSKI, M. 2001. Exploring 3d navigation: combining speed-coupled flying with orbiting. In *Proc. Conf. on Human Factors in Computing Systems*, ACM Press, 418–425.

TAYLOR, C. J. 2002. VideoPlus: a method for capturing the structure and appearance of immersive environments. *IEEE Transactions on Visualization and Computer Graphics 8*, 2 (April-June), 171–182.

UYTTENDAELE, M., CRIMINISI, A., KANG, S. B., WINDER, S., SZELISKI, R., AND HARTLEY, R. 2004. Image-based interactive exploration of real-world environments. *IEEE Computer Graphics and Applications 24*, 3, 52–63.

WARE, C., AND OSBORNE, S. 1990. Exploration and virtual camera control in virtual three dimensional environments. In *Proc. Symposium on Interactive 3D Graphics*, ACM Press, 175–183.