Scene Chronology

Kevin Matzen and Noah Snavely

Cornell University, Ithaca NY
{kmatzen,snavely}@cs.cornell.edu

Abstract. We present a new method for taking an urban scene reconstructed from a large Internet photo collection and reasoning about its change in appearance through time. Our method estimates when individual 3D points in the scene existed, then uses spatial and temporal affinity between points to segment the scene into spatio-temporally consistent clusters. The result of this segmentation is a set of spatio-temporal objects that often correspond to meaningful units, such as billboards, signs, street art, and other dynamic scene elements, along with estimates of when each existed. Our method is robust and scalable to scenes with hundreds of thousands of images and billions of noisy, individual point observations. We demonstrate our system on several large-scale scenes, and demonstrate an application to time stamping photos. Our work can serve to **chronicle** a scene over time, documenting its history and discovering dynamic elements in a way that can be easily explored and visualized.

Keywords: Structure from motion, temporal reasoning, 4D modeling.

1 Introduction

In the last few years, our ability to model the world around us in 3D has greatly increased, due to the availability of vast numbers of photographs from sources such as Flickr, Facebook, and Picasa, as well as new reconstruction algorithms that can run on Internet-scale collections [1,2]. Indeed, the concept of "Internet-scale" now encompasses the scale of the world, as planet-sized reconstructions are now being built [3]. Once every stone has been turned to a point in a point cloud, does that mean the problem of 3D reconstruction is complete? While we often refer to scenes reconstructed via methods such as structure from motion (SfM) as being "static," in reality many urban scenes are quite dynamic over time scales of even a few years. Signs, lights, decorations, and street art are examples of such dynamism—they are often reconstructable with current methods, but transient in a longer-term view. Such visual elements provide opportunities for us to reconstruct and visualize our ever-changing world.

However, state-of-the-art 3D reconstruction methods remain largely agnostic to the time domain. While some systems work out of the box when applied to dynamic scenes, they usually provide an output model devoid of any temporal information, where points from different time spans simply co-exist. Other systems actively prune multiple points that occupy the same spatial location, which for time-varying scenes leads to broken reconstructions, veritable chimeras of different objects or appearances conflated in time.

Our aim is to bring temporal meaning back to these reconstructions. We propose a simple approach by which output from an existing 3D reconstruction method can be

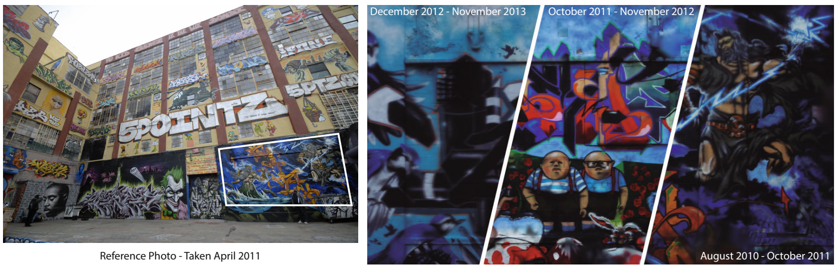

Reference Photo - Taken April 2011
December 2012 - November 2013
October 2011 - November 2012
August 2010 - October 2011

Fig. 1. A montage of a planar region of the 5 Pointz dataset, showing how this region has changed over time as different pieces of art have been overlaid atop one another

taken as input and segmented into distinct space-time objects, each with a specific extent in time. This method takes the web of observations and untangles it into temporally consistent components. Our approach not only augments a reconstruction with temporal information, but also leverages the time dimension as a unique signal for segmenting scenes into meaningful objects; just as object motion is an important cue for video segmentation, temporal structure is a useful cue for segmenting scenes.

Our method works by analyzing two key pieces of information: (1) the visibility relationship between images and points in a reconstruction, and (2) the timestamps on the images themselves. However, due to noise, it is difficult to use this raw information to directly reason about time. Incorrect timestamps, registration failures, uncontrolled illumination, photographer bias, sporadic temporal sampling, and photo manipulation all make temporal reasoning a challenging task. Our work in inspired by previous methods on temporal inference on 3D scenes [4,5], but our approach differs especially in *scale* and *granularity*—our method is designed for handling tens to hundreds of thousands of images sampled relatively densely in time. When reasoning about time in image collections, every observation has something to say, and so we focus on methods that are simple and highly scalable to the billions of individual observations in our datasets.

We demonstrate our method on several large-scale datasets, including dynamic city scenes, such as Times Square, as well as 5 Pointz, an outdoor street art exhibit space in NYC. Figure 1 shows a region of our 5 Pointz dataset, demonstrating the output we produce—extracted regions, sometimes overlapping in space, each with a temporal span. We use these results in two applications: a system for visualizing scenes over time, and a method for timestamping a new photo based on its contents. More broadly, we believe that our system can be used as the basis for *temporal visual discovery*—allowing one to automatically chronicle the rich visual appearance of a scene over time leveraging the wealth of online imagery, and to explore and visualize the results.

2 Related Work

Dynamic Scene Analysis. Scenes can be dynamic at many different time scales—seconds, minutes, years, decades. At video-rate time scales, there has been significant work on reconstruction of moving objects, spanning problem domains including motion capture [6], non-rigid SfM [7], scene flow methods [8,9], and video browsing [10]. Recent work has also sought to automatically *sequence* unstructured images captured over short time scales, based on motion constraints [11]. Somewhat longer time scales can be captured with time-lapse video, which only captures a single viewpoint but is especially useful for studying changes due to illumination [12,13] or long-term motions of objects [14]. Our domain of dynamic reconstruction methods from unstructured imagery at the scale of several years has received relatively little attention, but is becoming more and more feasible as the number of available images of scenes grows ever larger (imagine a future scenario where 100 years' worth of densely sampled imagery is available for any scene!)

There are a few systems that operate on unstructured imagery over large time scales. Notably, Schindler and Dellaert [5] build probabilistic temporal reconstructions of scenes from historical photo collections. By analyzing occluder-occludee relationships of buildings and considering timestamps of varying precision, they formulate a graphical model and perform joint inference to determine the most likely temporal ordering for a set of photographs, a set of improved timestamps, and temporal extents for the observed buildings. Our work differs in several key respects. First, their analysis was conducted on photo collections containing hundreds of photos, and performs an inference involving potentially expensive steps, such as an occlusion-test subroutine. In contrast, a key goal of ours is to scale to very large collections (our largest input set includes over 1 million photos). Second, we focus on more fine-grained temporal reasoning, involving photos sampled densely in time over the last ten years. We also focus primarily on changes in *appearance* (e.g., one billboard changing to another), rather than changes in geometry. Finally, their work performs an initial object segmentation based on proximity and a simple camera co-visibility metric. In our work, we integrate time-based reasoning directly into the scene segmentation, avoiding hard segmentation decisions early on.

Change Detection. A related field is that of change detection, where the goal is to identify differences between two or more 2D images or 3D scenes. Pollard and Mundy proposed a change detection method for 3D scenes observed from an aerial vehicle [15]. By building a model of occupancy and appearance over a voxel grid, they can determine if a new view was generated either by the background model or by some change. Other related work proposes geometric change detection by comparing 3D reconstructions built from video [16,17]. However, these formulations only produce differences between two points in time. Our work provides a full temporal model across a large timespan. Fard et al. proposed a method for using unstructured photo collections to monitor the progress of a building under construction by comparing 3D reconstructions to 3D building specifications [18]. In our case, however, we have no a priori scene model.

3 Overview

Our overall goal is to build a dense 3D point cloud with appearance information, e.g., using methods such as PMVS, and then to segment this reconstruction into a set of temporally coherent geometric components (corresponding to signs, billboards, storefronts, etc.), as well as estimate a time range $[t_{\min}, t_{\max}]$ for each component defining its temporal extent. We use the timestamps on images for this task. However, due to noise in timestamps and in our estimates of point visibility, this task is difficult to perform robustly. In addition, due to the scale of our data, we require very efficient algorithms.

Assumptions and Scene Representation. Our method makes several assumptions about the input scene. First, we apply a standard SfM and multi-view stereo (MVS) pipeline (with a few modifications) to derive an initial reconstruction from a set of photos. Hence, even for a dynamic scene, we assume we can reconstruct accurate cameras and points using SfM. This assumption relies on there being sufficient scene content that is constant over time, or at least there being sufficient continuity in a scene from one time to the next. Second, we assume that the geometry of the scene is largely static, but that the appearance can change dramatically from one time to the next. Such appearance changes are a key source of dynamism in scenes like Times Square, where storefronts, billboards, and other ads are changing, often from one week to the next. Finally, because we focus on urban scenes, we assume that the scene geometry can be well-approximated with a set of planes, as in other work on urban reconstruction [19,20].

This last assumption leads to our choice of output scene representation. We assume that the components we segment the scene into can be approximated as (not necessarily axis-aligned) planar facets—i.e., at any given time, the 3D scene is approximated by 2D planes. Further, as in Schindler et al. [5], we assume that unique objects come into existence at some time, exist for a while, then are removed or replaced permanently. Hence, each object exists for a single, contiguous span of time. If we consider a dynamic scene as a 4D space-time volume, then our planar objects form 3D space-time "cuboids" in this volume—2D in space, 1D intervals in time. Hence, our output representation consists of a set of such objects, which we refer to as "plane-time cuboids."

Space-Time Reconstruction System. Our overall pipeline is illustrated in Figure 2. We take a large collection of Internet photos of a scene, and generate a dense 3D point cloud reconstruction (Fig. 2(1)), which initially is the union of all points across time. Next, we reason about which cameras observe which points (2), and use the observation profile of each 3D point to reason about the time interval in which it existed (3). We then segment the scene into sets of points that are spatio-temporally coherent ((4) and (5)), from which we can render visualizations of each part of the scene over time (6). We now describe our space-time reconstruction and segmentation algorithm in detail.

4 Method

4.1 Dense 3D Reconstruction

Our input is a large set of photographs (tens of thousands to millions) downloaded from the Web using targeted keyword searches such as `times square`. We restrict our download to photos with timestamps (e.g., for Flickr, photos that have a `datetaken` field). Using a random subset of 20,000 photos, we build an initial SfM reconstruction using Bundler to obtain camera parameters and a sparse 3D point cloud [21]. We then use 2D-to-3D matching techniques to register the remaining photos to the reconstruction [22]. This process results in a set of reconstructed images $\mathcal{I}$, camera parameters

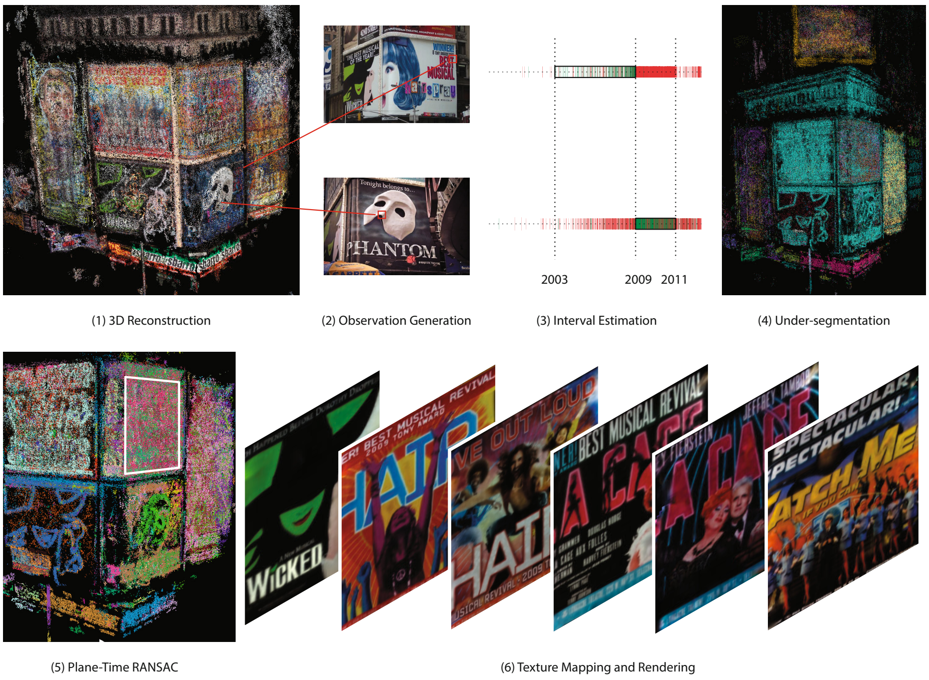


Fig. 2. System pipeline. Given a dense 3D reconstruction (1), we expand a visibility graph (2), and then use that graph to estimate per-patch time intervals (3). Using spatial and temporal cues, we construct a spatio-temporal graph and perform an undersegmentation (4). Then we fit plane-time cuboids to this graph (5) and generate texture maps for visualization (6). This example shows an exploded view of several billboards at the same location over time.

for each image, and a sparse set of 3D points. Finally, we use CMVS to factor the reconstruction into smaller components [23], then use a modified version of PMVS to generate a dense 3D point cloud [24]. PMVS contains a number of geometric filters that disallow two 3D points from occupying the same (or very similar) spatial location. We modify PMVS to remove this restriction on output geometry, as detailed in the supplemental material. The result of CMVS+PMVS is a dense set of 3D patches $\mathcal{P}$. Each patch $p \in \mathcal{P}$ is represented with a 3D position, a surface normal, and a set of images $V_{\mathrm{PMVS}}(p) \subseteq \mathcal{I}$ in which p is deemed to be visible (as determined by high photoconsistency with a reference patch for p). One of these views is selected as a reference view.

4.2 Computing the Visibility Graph

Our system relies on a good initial estimate of the visibility of each patch p in the set of cameras, as visibility (along with timestamps on images) provides strong information about when in time a patch existed. The visibility sets V_{PMVS} produced by PMVS are very approximate, and so we expand these sets into a richer representation of visibility.

We distinguish between two notions of visibility in our work, which we call *projection-visibility* and *photoconsistent-visibility*:

- A patch p is *projection-visible* in view $v \in \mathcal{I}$ if p is in front of v, p projects inside the field of view of v, and p passes an occlusion test with other patches.
- A patch p is *photoconsistent-visible* in view $v \in \mathcal{I}$ if p is projection-visible in view v and further p satisfies a photoconsistency criterion in view v.

We use these definitions to further define notions of "positive" and "negative" observations of a patch p in an image v, encoded as a *visibility graph*. The visibility graph $\mathcal{V}$ is a bipartite graph $(\mathcal{I}, \mathcal{P}, \mathcal{E})$ with edges $e \in \mathcal{E}$ connecting views to patches. $\mathcal{V}$ has two types of edges. A *positive observation* edge e_{ij+} encodes a temporal event where p_j is photoconsistent-visible view v_i. A *negative observation* edge e_{ij-} encodes a temporal event where view v_i *should* have observed p_j under projection-visibility (i.e., p_j is projection-visible to v_i), but p_j is not photoconsistent-visible to v_i. Negative observations can occur for several reasons, including dynamic occlusion and misregistration, but more importantly, they occur when a patch does not exist at a particular point in time. These observations help us narrow down exactly when a patch *does* exist. Note that observations that are not projection-visible are treated as providing no information.

CMVS+PMVS typically produces only a small subset of possible positive observations $V_{\mathrm{PMVS}}(p)$ for each patch p, for reasons of efficiency; only a few views are necessary for computing a well-conditioned estimate of a point's location, not the hundreds or thousands of potential views available for that patch. In our work, however, we want as much information as possible (as well as both positive and negative observations) to precisely estimate $t_{\min}$ and $t_{\max}$ for each patch (i.e., answering the question "when did this patch exist?"). Therefore, we perform a visibility graph expansion step.

To do so, for each image $v \in \mathcal{I}$, we generate a depth map by projecting all reconstructed patches into v's image frame using a z-buffering algorithm. Each patch adds a 3-pixel radius contribution to the depth map. After generating this depth map, we project each patch p once more into the image; if p projects inside v's field of view and satisfies a soft depth test against this depth map, we deem p projection-consistent. We then compute a photoconsistency score by comparing a reference patch for p extracted from p's reference view (computed according to PMVS's patch extraction method) to the projection of the same oriented planar patch in v. If the normalized cross-correlation of both windows is above a threshold τ_{pos}, then a positive observation is recorded. Similarly, if the normalized cross-correlation is below a threshold τ_{neg}, then a negative observation is recorded. For all experiments, we set $\tau_{\mathrm{pos}} = 0.8$ and $\tau_{\mathrm{neg}} = -0.2$.

4.3 Initial Time Interval Estimation

The set of observations for a given patch p gives us useful information about when in time p existed, via the timestamps on images that observe p. For the segmentation step of our pipeline, we require an approximate initial time span $[t_{\min}(p), t_{\max}(p)]$, which we compute from the associated image timestamps. A natural algorithm is to simply take the minimum (earliest) and maximum (latest) positive observation timestamps as the time interval. However, this algorithm doesn't work well due to false positive observations and incorrect timestamps; surprisingly, even in a time when many cameras are networked, there are many reasons why timestamps can be wrong. We also tried or-

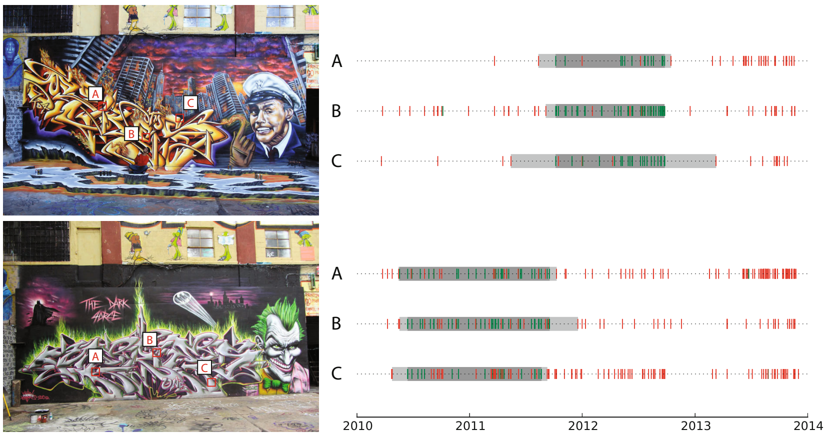


Fig. 3. Time profiles (right), for two different appearances of the same wall (left). Green (red) marks on the timeline indicate positive (negative) observations. Dark gray intervals indicate the shortest interval with the maximal F_1 score. Any interval with endpoints contained within the light gray regions will have the same F_1 score. Note the false positive red marks inside the gray intervals and the false negative green marks outside the gray intervals, as well as the very non-uniform distribution of observations, which highlight the need for robust methods.

dering positive observations by time and trimming off the bottom and top k-percentile. However, because our observations are very non-uniform samples in time (with some times much more densely sampled than others), we found that this approach also performed poorly. In general, we found that considering only positive observations was insufficient to compute robust time bounds for a patch, and we must also use negative observations.

We found it effective to formulate this problem is in terms of classification, where we wish to select a single time interval that classifies observations as being positive (inside the interval) or negative (outside the interval), with the goal of achieving both good precision and recall. Here, *precision* is the ratio of positive observations inside the interval to the total number of observations in the interval. Similarly, *recall* is the ratio of the number of positive observations in the interval to the total number of positive observations across all of time. One common way to balance these measures is to use the F_1 score, which is the harmonic mean of these two terms:

$$\text{Precision} = \frac{\#\text{TP}}{\#\text{TP} + \#\text{FP}} \qquad \text{Recall} = \frac{\#\text{TP}}{\#\text{TP} + \#\text{FN}} \qquad F_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

For each patch p, we search over all possible start and end times $t_{\min}(p)$ and $t_{\max}(p)$ (where we can limit the search to only consider the discrete set of timestamps corresponding to our observations of that patch), and select the pair defining the time interval with the maximum F_1 score. This can be computed very quickly using dynamic programming. Note that in general there is a family of intervals with the same F_1 score for a set of discrete observations, as we can vary the endpoints continuously between two consecutive observations without changing the score. Therefore, we record four scalar values per patch, the range of best start and end times for the beginning of the interval, $t_{\min}(p)$, and the start and end times for the end of the interval, $t_{\max}(p)$. Figure 3 shows several patches with their estimated time intervals.

4.4 Spatio-Temporal Graph

Now that we have patches with initial time intervals, we have the information we need to segment the scene into a set of plane-time cuboids. Section 4.5 describes our repeated RANSAC procedure for segmenting the scene based on geometric and temporal cues. First, however, our scenes are quite large, with millions of patches, and for such scenes segmentation methods built on random sampling (e.g., RANSAC) can be difficult to scale. To address this problem, we take an approach inspired by GroupSAC [25], and preprocess our data to under-segment it into more manageable chunks where RANSAC is more likely to quickly find good hypotheses. To do so, we build a graph on the set of patches where pairs of patches are connected if they have high *spatio-temporal affinity*—if they are proximate in space and time—and then compute connected components of this graph. This method is related to the object segmentation in [5], but considers time as well, and serves as a preprocess to our final segmentation algorithm.

We connect two patches p and q if they satisfy the following criteria:

- **Spatial proximity.** p and q are within a 3D distance δ_d, and have normals that differ by no more than an angle δ_θ.
- **Temporal overlap.** p and q have time observations with a high degree of overlap.
- **Common view.** p and q are seen by at least one common camera from the original PMVS reconstruction ($|V_{\text{PMVS}}(p) \cap V_{\text{PMVS}}(q)| > 0$).

For the temporal overlap criterion (item 2), we could use the degree of overlap in the time intervals for p and q estimated in Section 4.3 above. Instead, we found a somewhat stricter notion of temporal overlap to be useful, in particular, we use the positive observations for each patch to form an *observation profile*. For each positive observation for a patch p, we pad it by ± 1 week to form a time interval. Overlapping time intervals for a single patch are aggregated into larger time intervals. The final set of (not necessarily contiguous) time intervals forms the observation profile for p. To measure the temporal overlap of a pair of patches (p, q), we compute the size of the intersection divided by the size of the union of the two observation profiles (where here "size" means the total time mass in the intersection or union of two observation profiles). This gives us a single scalar temporal affinity score. If this score is greater than a threshold δ_t, then the pair (p, q) satisfies the temporal overlap criterion.

To efficiently evaluate the three criteria above, for each patch p, we use FLANN [26] to find spatial neighbors within a 3D distance δ_d, then prune these potential candidate neighbors by the surface normal, temporal overlap, and common view criteria. The surviving edges define a graph G_{st} whose connected components form a rough initial undersegmentation into objects, as illustrated in Figure 2(4). For all experiments, we use $\delta_\theta = 60°$, $\delta_t = 0.75$, and we manually select a per-reconstruction δ_d equal to roughly 1-2 meters. Note, an alternative to manually selecting δ_d is to geo-register the reconstruction so that it is to metric scale.

4.5 Plane-Time Clustering

Finally, we further segment each component of G_{st} from the initial spatio-temporal segmentation into a set of plane-time cuboids that represent coherent objects in the scene. We perform the segmentation by greedily and repeatedly running a special "Plane-Time RANSAC," on each component, after each round removing the induced subgraph given by the RANSAC inliers from G_{st} to form G'_{st}, and recursively applying the same procedure to the connected components of G'_{st}. We terminate the recursion if a connected component does not contain some minimum number of vertices or if we are unable to find any valid RANSAC hypotheses after some fixed number of iterations. This process is run in parallel on each connected component.

Plane-Time RANSAC combines a standard plane-fitting inlier criterion with a new time interval hypothesis criterion. In particular, the algorithm samples three patches at random from the component and fits a plane to the three samples. It also computes a time interval hypothesis by taking the union of the patches' time intervals as estimated in Section 4.3. Thus a hypothesis is a combination of a plane and a time interval. A patch is considered an inlier to this hypothesis if (1) it is within some spatial distance of the hypothesis plane, (2) its normal is within some cosine distance of the hypothesis plane, and (3) its time interval has a high degree of overlap with the hypothesis time interval. Once again, we define overlap of two time intervals to be the size of their intersection divided by the size of their union. Because the estimated patch time interval is really a family of intervals with endpoints that are allowed to vary slightly, we select the unique patch time interval that maximizes the overlap between the patch and the hypothesis. Note that when generating a hypothesis, three non-collinear points uniquely define a plane, but the normals and time intervals of the sample points themselves may not satisfy the inlier criterion for the hypothesis. We check for such cases and discard the hypothesis if any sample is not an inlier to its own hypothesis. In our experiements, RANSAC was allotted 1000 attempts to generate a hypothesis, but was limited to 100 valid hypotheses.

At termination, we have a set of plane-time cuboids (associated with each cluster), and each patch has either been assigned to a cluster or has been discarded (as not belonging to a large enough cluster). The time interval associated with each plane-time cuboid can then be transferred back to the individual patches in the cluster to obtain refined per-patch time intervals, since we assume the object represented by the cluster appears and disappears as one unit. The distance, normal, and time overlap thresholds used in all RANSAC runs are identical to their analogous counterparts in Section 4.4.

5 Experiments and Results

Datasets. We ran our pipeline on three highly dynamic datasets of urban environments from photos taken over a span of several years. `TimesSquare` in Manhattan contains a quarter million registered photos exhibiting a wide variety of change in appearance in terms of billboards, signs, and building facades. `Akihabara` in Tokyo offers a related setting where most of the dynamic content is localized to billboards and signs. However, `Akihabara` also contains dramatic instances where entire buildings change appearance. Finally, `5Pointz` in Queens, NYC, served for several years as an outdoor

Table 1. Dataset statistics

Dataset	Photos Retrieved	Photos Registered	Patches	Positive Obs.	Negative Obs.
Times Square	1,222,792	246,218	13,590,341	5,274,437,797	12,347,622,570
5 Pointz	48,375	12,777	12,263,640	635,602,574	821,189,516
Akihabara	171,469	13,946	1,732,784	132,520,809	254,624,878

art exhibit space for graffiti artists, and offers a visually rich experience involving many pieces of art over time. In this last case, our method can be seen as a step towards automatic, distributed, art documentation. Table 1 shows the scale of these datasets.

5.1 Viewing and Rendering

Our plane-time cuboid representation provides a simple yet effective way to explore the evolution of a scene through space and time. We have created an interface where a user can explore the scene using standard 3D navigation controls, but can also move back and forth in time using a slider control.[1] At any selected time, we can render a snapshot of the scene as a point cloud using the patches that existed during that time, or can additionally render each cuboid as a texture mapped "quad." To do so, for each plane-time cuboid proxy, we generate a bounding 3D rectangle with one axis perpendicular to the scene's estimated up vector. For each camera that observes at least six patches associated with the cuboid, we apply the homography mapping image space to the quad's texture space. To enable clean blending between multiple cameras despite transient occlusions and geometric misregistration, we generate an alpha matte using Gaussian splats of uniform size centered at the projections of the positive patch observations. Examples of these quads are shown in Figures 2, 4, and 7.

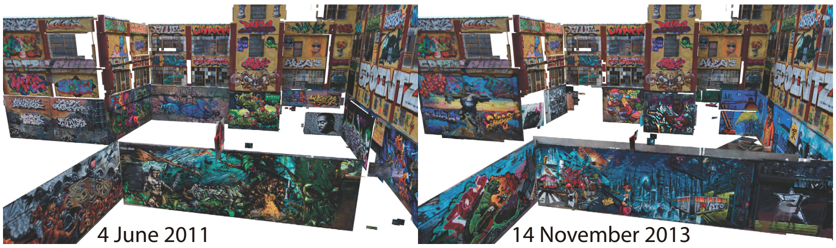
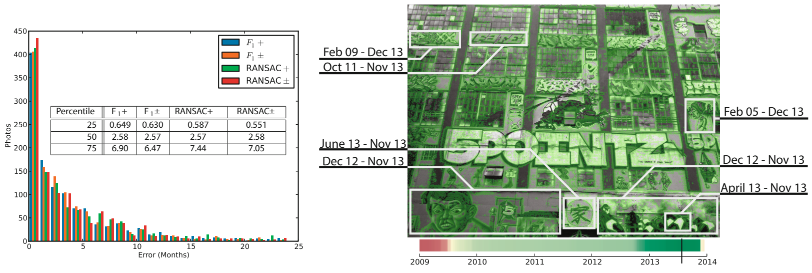


Fig. 4. A texture mapped rendering of `5Pointz` at two different points in time. Please refer to the project webpage for a video exploring all three datasets.

Our plane-time cuboids qualitatively represent meaningful semantic units. Figure 4 shows a birds-eye view of `5Pointz` at two distant, visually distinct points in time. Throughout time, segments come in and out of existence, sometimes in synchrony,

[1] Please visit `http://www.cs.cornell.edu/projects/chronology/` for videos and other visualizations.



Percentile	F_1+	F_1±	RANSAC+	RANSAC±
25	0.649	0.630	0.587	0.551
50	2.58	2.57	2.57	2.58
75	6.90	6.47	7.44	7.05

Fig. 5. `5Pointz` timestamp prediction results. Positive and negative observations provide clues as to when a photo was taken. Left: histogram and statistics of errors in timestamping. Right: positive observations for a sample photo are shown in green. Several components of this scene are well-localized in time and can be used to build a timestamp scoring function. The timestamp with the highest score is marked on the timeline. We predict this photo was taken on July 25, 2013; the photo metadata indicates it was taken on August 10, 2013.

sometimes independently. It is not until the whole collection is analyzed that the behavior of the scene becomes clear. For these sorts of scenes, it would be difficult for an expert to manually analyze each photo to determine the scene's temporal structure over the span of several years. However, we can achieve our results fully automatically. Please refer to the video on our project webpage for a rich visualization of the evolution of all three scenes. Figure 7 shows a number of interesting examples of automatically recovered space-time cuboids. Of particular interest are examples with temporal cues contained within the image. For example, a poster advertising the 2008 US presidential election is estimated to exist during the election period. Anime ads found in Akihabara advertise products with recent release dates. Graffiti in 5Pointz can be matched with coarse-grained databases found online. Each of these examples gives us clues about the world's appearance at a given point in time. While it is difficult to evaluate results quantitatively, subjectively we achieve segmentations with close correspondence to temporal objects in each scene.

5.2 Timestamp Prediction

Another application of our system is for the task of predicting the date of a photo, in case a date is missing or incorrect. Our time-stamping algorithm analyzes what portions of our time-varying reconstruction are observed in a query photo, what portions are not observed, and determines a date that best fits that information.

Given a new query photo registered to the model (e.g., using 2D-to-3D matching), we use the same observation generation procedure as described in Section 4. However, we select a more conservative $\tau_{\text{neg}} = -0.5$ to obtain more confident observations. Given the positive and negative patch observations, we use a voting scheme to determine a time interval where the positive observation time intervals overlap, but where the negative time intervals do not. For each observation, we generate a start event and an end event; each event carries a vote. For a positive observation, the start event carries a vote of $+w$ and the end event carries a weight of $-w$ where w is a tunable weight, $w = 10$ in

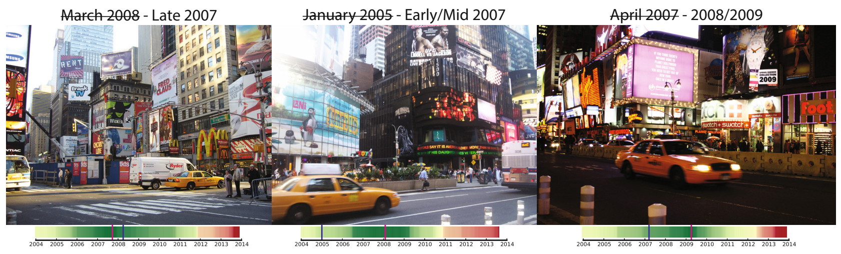

~~March 2008~~ - Late 2007
~~January 2005~~ - Early/Mid 2007
~~April 2007~~ - 2008/2009
2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014

Fig. 6. Retimestamped photos. The timeline indicates our timestamp scoring function with the given timestamp indicated in blue and our predicted timestamp indicated in pink. The originally recorded date and estimated date range as determined by manual inspection are provided above.

our experiments. For a negative observation, the start event carries a vote of -1 and the end event carries a weight of $+1$. The score for each time subinterval is computed by sorting all events by timestamp and then stepping along the timeline, accumulating the vote carried by each event. The subinterval with the maximal score is then found. To generate a single predicted timestamp, we select the midpoint of this interval.

In practice, we found that we had to consider the effect of unmodeled occlusions and its impact on negative observations. By unmodeled occlusion, we mean the effect of holes in the depth buffer used to determine projection-visibility of a point due to parts of the scene geometry not being reconstructed. While this is not as much of a concern when aggregating observations at each patch since this effect is more or less random for a given patch, it is very consistent for aggregating observations for each camera. Therefore, we only include a negative observation if at least one positive observation is in close 3D spatial proximity. This implies that we could see the surface geometry at one point in time, but we do not currently see a certain appearance in the query photo.

To evaluate this method, we conducted a timestamping experiment on `5Pointz`. We selected a random 10% of registerable photos as a test set and ran our pipeline as a training stage on the remaining 90%. The test set was selected such that no photographer overlaps between training and test sets, to prevent any bias a single photographer might introduce. Note that while we measure distance to the original timestamps as ground truth, these contain noise, and so our results should be interpreted with this in mind.

We present four variations on the algorithm described above. Methods denoted F_1 use the per-patch time estimates obtained by our F_1 interval procedure before Plane-Time RANSAC segmentation is performed. Methods denoted RANSAC use per-cluster time estimates obtained via Plane-Time RANSAC. Methods denoted $+$ use only positive observations while those denoted $\pm$ use both positive and negative observations. Figure 5 shows our results as a histogram of distances between predicted and original timestamps. We observe improvement with both the F_1 and RANSAC methods after we introduce negative observations, achieving a median distance of around 2.5 months, with 25% of images having about 0.5 month distance (quite low given that all the only cue we are using is changing scene appearance). However, there is an increase in error for the 75th percentile when we move to the RANSAC methods from the F_1 methods. One explanation is that by clustering the points, we might be losing some precision

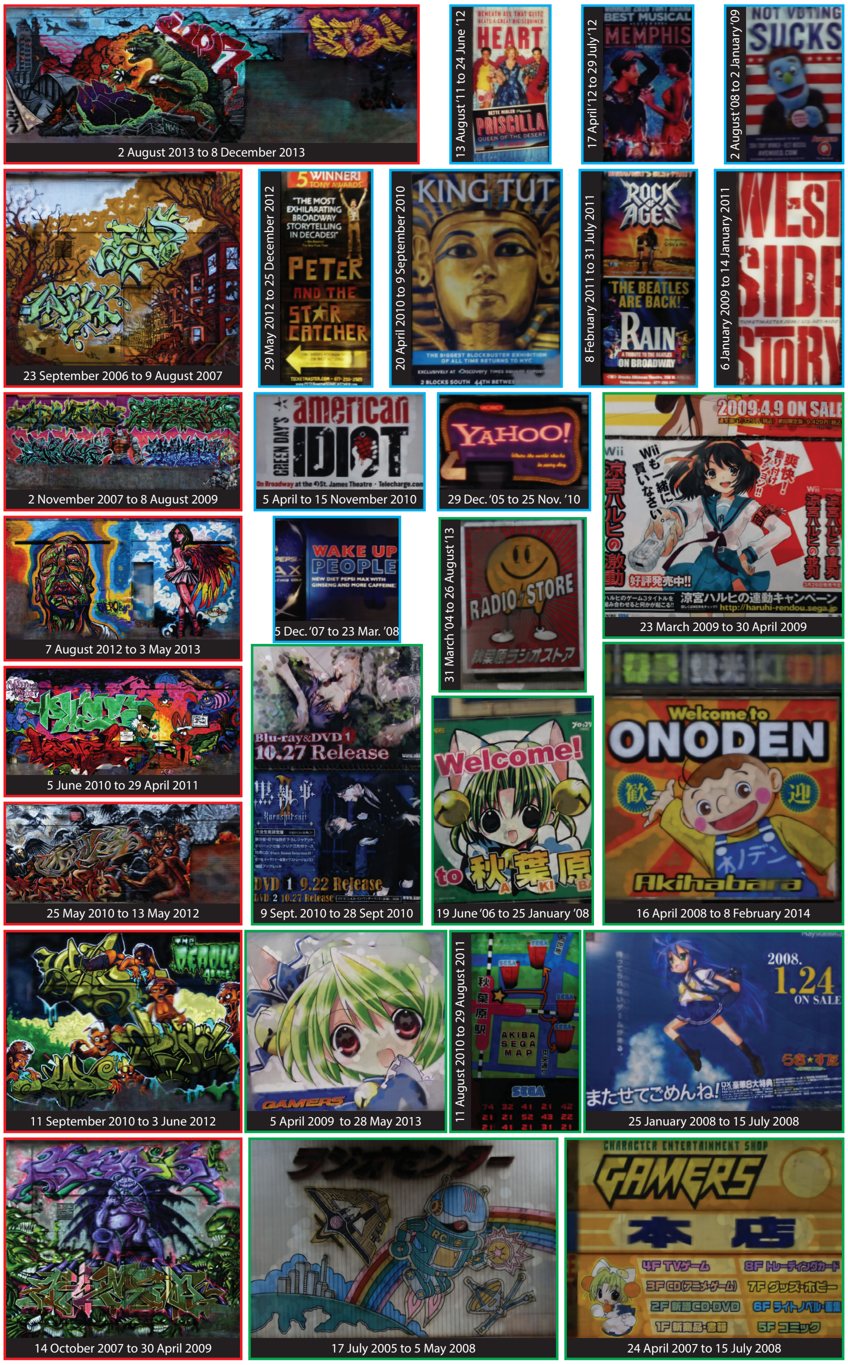

2 August 2013 to 8 December 2013
13 August '11 to 24 June '12
17 April '12 to 29 July '12
2 August '08 to 2 January '09
23 September 2006 to 9 August 2007
29 May 2012 to 25 December 2012
20 April 2010 to 9 September 2010
8 February 2011 to 31 July 2011
6 January 2009 to 14 January 2011
2 November 2007 to 8 August 2009
5 April to 15 November 2010
29 Dec. '05 to 25 Nov. '10
23 March 2009 to 30 April 2009
7 August 2012 to 3 May 2013
5 Dec. '07 to 23 Mar. '08
31 March '04 to 26 August '13
5 June 2010 to 29 April 2011
9 Sept. 2010 to 28 Sept 2010
19 June '06 to 25 January '08
16 April 2008 to 8 February 2014
25 May 2010 to 13 May 2012
11 September 2010 to 3 June 2012
5 April 2009 to 28 May 2013
11 August 2010 to 29 August 2011
25 January 2008 to 15 July 2008
14 October 2007 to 30 April 2009
17 July 2005 to 5 May 2008
24 April 2007 to 15 July 2008

Fig. 7. An assortment of plane-time cuboids from `TimesSquare` (blue), `Akihabara` (green), and `5Pointz` (red) with texture maps and estimated time intervals produced by our system

in our time estimates; another is that we are seeing effects from noise in the original timestamps.

We also applied our timestamping method to photos from `TimesSquare`. Figure 6 shows examples of images that we believed to have incorrect timestamps based on the content in the scene (i.e. ads, weather, clothing), and for which our method predicted timestamps that more closely matches our belief of when these photos were taken. In general for this dataset, applying RANSAC $\pm$ resulted in a 25-50-75 percentile errors of 1.22, 5.47, and 20.1 months, respectively, indicating that this is a harder dataset for timestamping than `5Pointz`.

6 Discussion and Conclusion

Our system has several limitations, some common to traditional 3D reconstruction systems and some unique to the 4D reconstruction problem. One key issue is that photo misregistration can prevent accurate timestamp estimation. If several photos are localized incorrectly in 3D due to some repeated structure in the scene, then all of the timestamps associated with those photos will contaminate their associated geometry. Since these photos themselves are in photoconsistent agreement, our system maybe decide to merge them into a single, potentially long-lasting segment. Another issue is that many structures in the real world cannot be approximated accurately using planar segments. In that case, our system often oversegments these regions into several smaller facets. An improvement would be to extend the set of models available in the RANSAC segmentation routine, or to have a more flexible geometric model that uses time as a strong segmentation cue.

Even in the scenes we studied, there were examples of changes in the physical structure of the buildings which violate our static geometry assumption. In 2009 a staircase at 5Pointz collapsed and our reconstruction captures this in its segmentation. However, since the z-buffer used in our system is time-agnostic, the projection-visible test culls potential positive observations that became visible after the staircase's removal.

The assumption that objects come into and leave existence only once can be violated. The 5Pointz dataset has at least one example where graffiti was covered up and then later restored. Both the original graffiti as well as its protective surface were reconstructed as two separate segments and the reappearance of the graffiti was suppressed.

Finally, timespan estimation is much more precise for segments that are clearly limited to some range of time in the middle of the dataset's timeline. Segments that occur very early (pre-2005) or very late (post-2013) often have no negative observations to help limit either their starting or ending points.

Conclusion. Point observations over time can play an important role in the semantic segmentation of 3D point clouds. We introduce the first fully automatic and Internet-scalable system for segmenting 3D point cloud reconstructions for scenes containing largely static geometry, but changing appearance. By leveraging a simple classification accuracy statistic, we are able to robustly estimate time intervals for each point in a reconstruction by considering both positive and negative observations, despite severe noise. Our method then leverages spatial, temporal, and co-visibility cues to segment points into spatio-temporally consistent components that we call plane-time cuboids.

The segments produced by our system often are natural units (signs, facades, art, etc.), and are useful for both visualization and time stamping applications.

Future work will involve extending state-of-the-art image-based rendering techniques into the time domain for immersive photo exploration through space and time using the plane-time cuboid proxies generated by our system. In addition, we hope to integrate our appearance-based method with modeling of larger-scale geometric changes, as well as more general types of visual changes over time [27].

Acknowledgements. This work was funded in part by NSF grants IIS-1111534 and IIS-1149393, by support from the Intel Science and Technology Center for Visual Computing, and by support from Amazon Web Services in Education.

References

1. Agarwal, S., Snavely, N., Simon, I., Seitz, S.M., Szeliski, R.: Building Rome in a day. In: ICCV (2009)
2. Frahm, J.-M., et al.: Building rome on a cloudless day. In: Daniilidis, K., Maragos, P., Paragios, N. (eds.) ECCV 2010, Part IV. LNCS, vol. 6314, pp. 368–381. Springer, Heidelberg (2010)
3. Klingner, B., Martin, D., Roseborough, J.: Street view motion-from-structure-from-motion. In: ICCV (2013)
4. Schindler, G., Dellaert, F., Kang, S.B.: Inferring temporal order of images from 3D structure. In: CVPR (2007)
5. Schindler, G., Dellaert, F.: Probabilistic temporal inference on reconstructed 3D scenes. In: CVPR (2010)
6. Shotton, J., Fitzgibbon, A., Cook, M., Sharp, T., Finocchio, M., Moore, R., Kipman, A., Blake, A.: Real-time human pose recognition in parts from single depth images. In: CVPR (2011)
7. Bregler, C., Hertzmann, A., Biermann, H.: Recovering non-rigid 3D shape from image streams. In: CVPR, pp. 690–696 (2000)
8. Vedula, S., Baker, S., Rander, P., Collins, R.T., Kanade, T.: Three-dimensional scene flow. PAMI 27(3) (2005)
9. Ulusoy, A.O., Biris, O., Mundy, J.L.: Dynamic probabilistic volumetric models. In: ICCV (2013)
10. Ballan, L., Brostow, G.J., Puwein, J., Pollefeys, M.: Unstructured video-based rendering: Interactive exploration of casually captured videos. In: SIGGRAPH (2010)
11. Basha, T., Moses, Y., Avidan, S.: Photo sequencing. In: Fitzgibbon, A., Lazebnik, S., Perona, P., Sato, Y., Schmid, C. (eds.) ECCV 2012, Part VI. LNCS, vol. 7577, pp. 654–667. Springer, Heidelberg (2012)
12. Sunkavalli, K., Matusik, W., Pfister, H., Rusinkiewicz, S.: Factored Time-Lapse Video. In: SIGGRAPH (2007)
13. Jacobs, N., Roman, N., Pless, R.: Consistent temporal variations in many outdoor scenes. In: CVPR (2007)
14. Rubinstein, M., Liu, C., Sand, P., Durand, F., Freeman, W.T.: Motion denoising with application to time-lapse photography. In: CVPR (2011)
15. Pollard, T., Mundy, J.: Change detection in a 3-D world. In: CVPR, pp. 1–6 (June 2007)
16. Taneja, A., Ballan, L., Pollefeys, M.: Image based detection of geometric changes in urban environments. In: ICCV (November 2011)

17. Taneja, A., Ballan, L., Pollefeys, M.: City-scale change detection in cadastral 3D models using images. In: CVPR (June 2013)
18. Fard, M.G., Peña-Mora, F., Savarese, S.: Monitoring changes of 3D building elements from unordered photo collections. In: ICCV Workshops (2011)
19. Sinha, S., Steedley, D., Szeliski, R.: Piecewise planar stereo for image-based rendering. In: ICCV (2009)
20. Furukawa, Y., Curless, B., Seitz, S.M., Szeliski, R.: Reconstructing building interiors from images. In: ICCV (2009)
21. Snavely, N., Seitz, S.M., Szeliski, R.: Modeling the world from Internet photo collections. IJCV 80(2), 189–210 (2008)
22. Li, Y., Snavely, N., Huttenlocher, D., Fua, P.: Worldwide pose estimation using 3D point clouds. In: Fitzgibbon, A., Lazebnik, S., Perona, P., Sato, Y., Schmid, C. (eds.) ECCV 2012, Part I. LNCS, vol. 7572, pp. 15–29. Springer, Heidelberg (2012)
23. Furukawa, Y., Curless, B., Seitz, S.M., Szeliski, R.: Towards Internet-scale multi-view stereo. In: CVPR (2010)
24. Furukawa, Y., Ponce, J.: Accurate, dense, and robust multi-view stereopsis. PAMI 32(8), 1362–1376 (2009)
25. Ni, K., Jin, H., Dellaert, F.: GroupSAC: Efficient consensus in the presence of groupings. In: ICCV (2009)
26. Muja, M., Lowe, D.G.: Fast approximate nearest neighbors with automatic algorithm configuration. In: Int. Conf. on Computer Vision Theory and Application (2009)
27. Lee, Y.J., Efros, A.A., Hebert, M.: Style-aware mid-level representation for discovering visual connections in space and time. In: ICCV (2013)