Virtual Reality
DOI 10.1007/s10055-010-0186-2

SI: AUGMENTED REALITY

Augmenting aerial earth maps with dynamic information from videos

Kihwan Kim · Sangmin Oh · Jeonggyu Lee · Irfan Essa

Received: 17 November 2009 / Accepted: 11 December 2010

Abstract We introduce methods for *augmenting* aerial visualizations of Earth (from tools such as Google Earth or Microsoft Virtual Earth) with *dynamic information* obtained from videos. Our goal is to make *Augmented Earth Maps* that visualize plausible live views of dynamic scenes in a city. We propose different approaches to analyze videos of pedestrians and cars in real situations, under differing conditions to extract dynamic information. Then, we augment an *Aerial Earth Maps* (AEMs) with the extracted live and dynamic content. We also analyze natural phenomenon (skies, clouds) and project information from these to the AEMs to add to the visual reality. Our primary contributions are: (1) Analyzing videos with different viewpoints, coverage, and overlaps to extract relevant information about view geometry and movements, with limited user input. (2) Projecting this information appropriately to the viewpoint of the AEMs and modeling the dynamics in the scene from observations to allow inference (in case of missing data) and synthesis. We demonstrate this over a variety of camera configurations and conditions. (3) The modeled information from videos is registered to the AEMs to render appropriate movements and related dynamics. We demonstrate this with traffic flow, people movements, and cloud motions. All of these approaches are brought together as a prototype system for a real-time visualization of a city that is alive and engaging.

Keywords Augmented reality · Augmented virtual reality · Video analysis · Computer vision · Computer graphics · Tracking · View synthesis · Procedural rendering

1 Introduction

Various places on Earth can be visualized on the Internet using online Aerial Earth Maps (AEMs) services (e.g., Google Earth, Microsoft Virtual Earth, *etc*.). We can visually browse through cities across the globe from our desktops or mobile devices and see 3D models of buildings, street views and topologies. Additionally, information such as traffic, restaurants, and other services are also provided within a geo-spatial database. However, such visualizations, while rich in information, are static and do not showcase the dynamism of the real world. Such dynamism captured from the city and then mirrored on these AEMS would give us a more immersive experience and directly interpretable visual information. We are motivated to add such dynamic information to these online visualizations of the globe using distributed video resources. Imagine seeing the city with traffic flowing, people

This work was done while authors Sangmin Oh and Jeonggyu Lee were with Georgia Institute of Technology.

Project homepage URL: http://www.cc.gatech.edu/cpl/projects/augearth.

K. Kim (✉) · S. Oh · J. Lee · I. Essa
Georgia Institute of Technology, Atlanta, GA, USA
e-mail: kihwan23@cc.gatech.edu

S. Oh
Kitware Inc., Clifton Park, NY, USA
e-mail: sangmin.oh@kitware.com

J. Lee
Intel Corporation, Hillsboro, OR, USA
e-mail: jeonggyu.lee@intel.com

I. Essa
e-mail: irfan@cc.gatech.edu

Published online: 11 January 2011

moving around, sports being played, clouds moving, all visualized by Earth/Globe Visualization systems.

Figure 1 shows a still of such an augmented visualization being driven by the analysis of 36 video sources. Such an augmentation of static visuals of cities is possible because many sensory sources that capture the dynamic information within a city are now available. Among such sensors, there is huge growth of cameras in our environment, may these be traffic cameras, web cams and aerial videos or even handheld cameras by people in the context of crowdsourcing with images (see Snavely et al. 2006 for a compelling example of crowdsourcing with personal photographs). The direct use of such resources are limited due to (1) narrow field of view (FOV) of each camera, and (2) limited number of cameras compared to the size of the city.

In this paper, we present an approach to generate plausible visualizations of ALIVE cities that one can browse interactively in the form of dynamic and live Aerial Earth Maps. The rendition of these Alive Aerial Earth Maps is accomplished by spatio-temporal analysis of videos from different sources around the city. To achieve this goal, we address several technical challenges. We are specifically interested in using video data to augment these AEMs with dynamics. To achieve this, *first*, we need to develop a framework which extracts information about the geometry of the scene and also the movements in the environment from video. *Second*, we need to register the view from the given video to a view in the AEMs. In cases where we have multiple instances of views, but still not full coverage due to limited FOV, we need to infer what is happening in between the views, in a domain-specific manner. This requires designing models of dynamics from observed data. *Third*, we need to generate visualizations from the observed data onto the AEMs. This includes synthesizing behaviors based on videos, procedural information captured from them and updating views as they are manipulated in the AEMs.

Figure 2 shows an overview of our framework to take (1) visual measurements and adapting them for (2) registration onto the AEMs and subsequently (3) visualizing motions and viewpoints. It is important to note that in our framework, in addition to extracting viewpoints and registering them to new views, we also model the dynamic information for the purpose of synthesizing it on AEMs. We use a variety of view synthesis, behavioral simulation and procedural rendering approaches to achieve this. At present, the rendered alive cities are visualized in real time based on off-line analysis of the captured video data. As we synthesize behaviors driven by visual measurements, our visualization shows a plausible viewpoint of the state of the environment from aerial views with live dynamic information. Additionally, our system allows manual user manipulation and control to address issues such as mis-alignments and viewpoint mis-registration, if needed.

2 Scenarios under study for AEMs

Extracting dynamic information from video feeds for augmenting AEMs is a challenging problem, primarily due to wide variety of conditions/configurations that we will have to deal with. For example, we want to be able to take videos of clouds moving and project them onto AEMS for visualization. We want to see people moving in different situations, and also show traffic motions. Furthermore, in each of the above instances, we have to

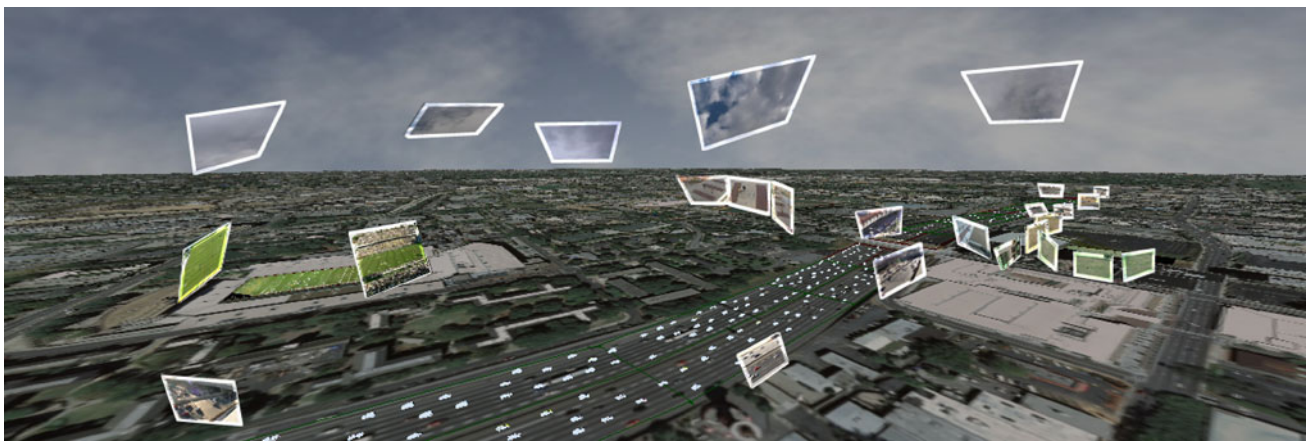

Fig. 1 An overview of the "Augmented Earth Map" generated by our system. We make use of 36 videos to add dynamic information to city visualization. Dynamism in each input video of traffic, people, and clouds is extracted, then it is mapped onto the Earth map in real time. Each video patch in the figure represents the input sources. See the video on our site for a fly-around

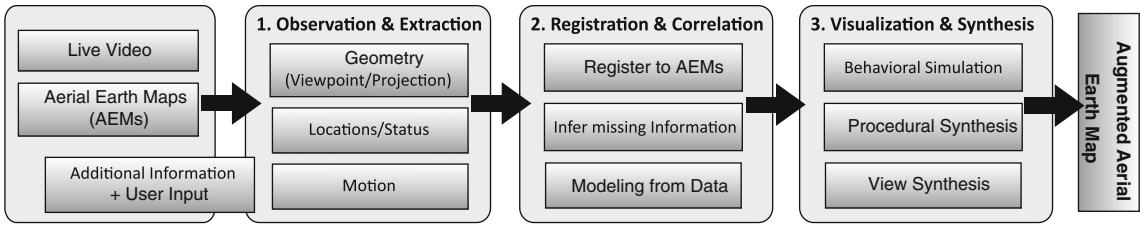

Fig. 2 Overview of our approach, highlighting the three main stages of observation, registration, and simulation

deal with different viewpoints of videos and in many cases with incomplete information. Finally, we have to track moving objects and determine coherence between different viewpoints.

To address these issues of variation across different domains of interest to us, we divide our study into four scenarios that address the distribution of cameras and motion of objects. We acknowledge that accurate tracking and registration from video still remains an open research question. In our work, we have found satisfactory solutions that help us to get to our goal of generating new augmented visualizations from real data (video). We leverage analysis within a specific context, with domain knowledge as needed, to help our attempt to build these AEMs. We do allow for simple user interaction to help to remove errors in registration.

Now, we describe in brief the various cases we concentrate on and will discuss them in detail in the rest of the paper. The results will be used to showcase the variations in configurations and how we generate Augmented Visualizations of Live Cities.

1 Direct Mapping (DM). Video is analyzed directly and tracked. Data is projected onto the limited regions covered by camera's field of view. We showcase several examples of people walking around.

2 Overlapping Cameras with Complex Motion (OCCM). Several cameras with overlapping views observe a relatively small region concurrently. The motion within the area is complex. We demonstrate this with people playing sports, and this can be considered valid for other CCTV domains.

3 Sparse Cameras with Simple Motions (SCSM). Sparsely distributed cameras cover a wide area but dynamic information is simple. For example, traffic cameras are separately observing a highway and the motion of vehicles is relatively simple and is predictable between nearby regions.

4 Sparse Cameras and Complex Motion (SCCM). Cameras are sparsely distributed and each of them observes a different part in a larger area. Moreover, the motion of the target object is complex. This is the case where we observe and model natural phenomena such as clouds in the sky.

3 Related and motivating previous work

Our work builds on existing efforts in computer vision on object-tracking and multi-view registration. We rely on behavioral animation approaches from the graphics community for our simulations. We briefly describe some of the efforts that guided and motivated us before discussing the technical details of our work.

Multi-view geometry research (Hartley and Zisserman 2000) has established techniques to compute the geometric properties between two planes. Using these approaches, we need match the 2D video scenes from our videos to the planes in the virtual 3D environment. We focus primarily on extracting coarse geometry information from monocular images rather than reconstructing exact 3D models. To aid the correspondence selection process, Spidery mesh (Horry et al. 1997) interface and a modified version of a vanishing point estimation algorithm (Kosecka and Zhang 2002) are employed as necessary (the direction of the moving objects from tracking step is used to determine the most dominant vanishing points).

Some of our works on view synthesis builds on the work of Seitz and Dyer (1996) and Seitz et al. (2006), which uses morphing with stereo camera models to generate in-between views. We are also inspired by the EyeVision System (Kanade 2001). While those approaches reconstruct compelling intermediate views, they need precise stereo pairs and local correspondences. Those are not suitable for a practical application where a large amount of videos, from the crowd-resources, need to be registered in real time. Thus, we used global blending approaches to register a number of views and visualize them immediately in the AEMs.

Object tracking is widely studied in computer vision (Yilmaz et al. 2006). We adopt a low-level feature tracking method, namely KLT feature tracker (Bouguet 2003; Shi and Tomasi 1994). It provides necessary information to compute the position of pedestrians or cars and average velocities reliably and in real time for a large range of videos recorded under varying weather and viewpoint conditions.

To simulate a large number of moving objects (cars in our traffic example), we employ the flock simulation

approach of Reynolds (1987). In our work, we developed a parameterized version of Reynolds' steering behavioral model (Reynolds 1999). Our behavioral models are dynamically adjusted during the simulation to reflect the incoming traffic information obtained from videos. While there are many approaches for simulating behaviors of autonomous agents, we adopt recent work (Shao and Terzopoulos 2007; Treuille et al. 2006) on crowd simulation, which produces good results reliably across various behavioral models.

There are a number of approaches for rendering sky and clouds. Harris (2005) and Wang (2004) introduce methods for rendering realistic clouds using imposters and sprites generated procedurally for real-time games. While extremely realistic, these approaches do not suit our purposes as we are interested in driving clouds from video data. Turk and O'Brien (1999) use radial basis function (RBF) interpolation to find an implicit surface from scattered data using constraint points. Building on this, we use RBF to interpolate global distribution of clouds density using video data, in a manner similar to Chen et al. (2008), where it is employed for generating city blocks. We also rely on Perlin noise (1985) for generating clouds volume or sprites for generating cloud maps.

A closely related system work is the Video Flashlight system (Sawhney et al. 2002), a surveillance application, which tracks people in a fixed region using multiple cameras and maps the results to the observed region. Several other efforts have also appeared in the same thread and are relevant to our work (Girgensohn et al. 2007; Sebe et al. 2003; Wang et al. 2007).

A worthwhile effort to mention is proposed by Snavely et al. (2006) to reconstruct 3D scenes from large set of images taken at different views. To some extent, we share similar goals, but we work with videos instead of images and synthesize dynamic information extracted from those videos.

4 Tracking and geometry extraction

We rely on the vast body of works in computer vision to extract geometric information and track the movement of objects from videos. Subsequently, they are registered and projected onto AEMs. We brief the details of each here, and then we will describe variations of these (Sect. 5) as we employ them for the different scenarios discussed in Sect. 2.

While most methods used in augmented reality applications need a structure of the real world to find an adequate plane to put a virtual object onto it, our methods do not need to use Structure from Motion (Klein and Murray 2007). This is because the plane of interest in AEM can be easily defined as a patch of the virtual earth environment. Thus, in our case, to register the tracked objects from video onto the aerial view, we only need to find a homography between video scene and the corresponding observed area in the aerial view. Consider the relationship of two viewpoints $\mathbf{f}_1$ and $\mathbf{f}_2$ with respect to a 3×3 homography matrix $\mathbf{H}$ such that $\mathbf{x}' = \mathbf{H}\mathbf{x}$, where $\mathbf{x}$ and $\mathbf{x}'$ are homogeneous point positions at scenes observed from viewpoints $\mathbf{f}_1$ and $\mathbf{f}_2$ each. Using the matrix $\mathbf{H}$, we can project objects in one view to another view. In our case, the matrix $\mathbf{H}$ is a homography induced from a patch in the plane of the Earth Map environment and the region in the screen space of the video frame corresponding to the patch. Then, we can project the location of a point in the video onto the patch in the Earth Map environment. Since the matrix $\mathbf{H}$ has 8 DOF, the 8 unknowns can be solved for using the Direct Linear Transform (DLT) with a pair of four points, if the correspondences are exactly matched on both images. To obtain stable results, we normalize the coordinates of correspondences in the same way as to compute the fundamental matrix in Hartley (1997).

In the video scene, a user can manually assign four points that correspond to a vertex in the texture map by comparing both scenes. Estimating the homography by manually selecting these points may be unreliable because even small difference between points cause unstable results. For this reason, we first estimate the vanishing points (VP) of the video scene. We used an algorithm for estimating vanishing points as described in Kosecka and Zhang (2002) shown in Fig. 3. This estimation finds two horizontal vanishing points for the video scene and then the user can select four points by assigning a rectangle area same as the one in aerial view image. But if the estimation algorithm does not find two vanishing points or the estimation result is not exact, we use spidery mesh interface, integrated into our system, to locate the four points (Horry et al. 1997).

Now, we briefly summarize the tracking algorithm used in our work. The same algorithm is used for every scenario with different settings. First, we make a few simplifying assumptions to help us in this directions. We assume that most scenes used in our work are taken from a high vantage point. This allows limited occlusions between moving objects and the tracked objects do not deform too much across views. We do not want to rely on videos from the top view because such videos are hard to obtain. We require the orientation and position of the video camera to be fixed after initialization of the geometry. On the other hand, we do require that the tracking be automatically initialized. For our tracking, at present we are using the well-known KLT trackers (Bouguet 2003, Shi and Tomasi 1994) as an entry step for finding tracks. However, any tracking algorithm can be used if the algorithm meets the conditions above.

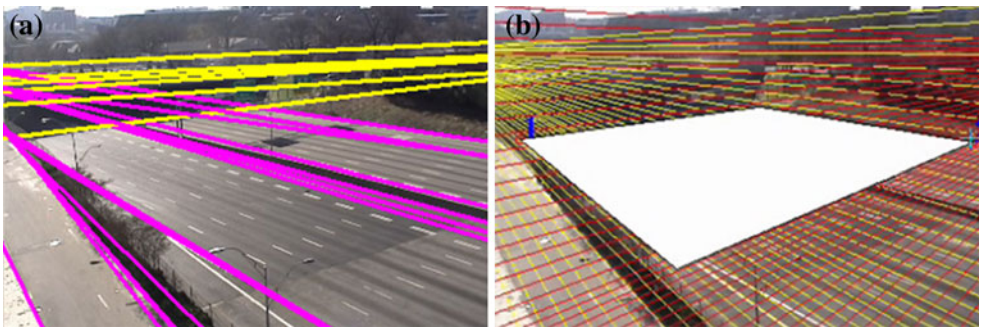

Fig. 3 Vanishing point estimation: **a** Two horizontal vanishing points are estimated using expectation maximization which is used in Kosecka and Zhang (2002). **b** Spidery mesh is applied to define a plane of interest

KLT trackers are aimed at determining a motion vector $\mathbf{d}$ between a local window W from one frame to a consecutive frame with a minimal residual function ε,

$$\varepsilon = \iint_W [J(\mathbf{A}x + \mathbf{d}) - I(\mathbf{x})]^2 \omega(\mathbf{x}) \mathrm{d}\mathbf{x}, \quad (1)$$

where I is the image from a video frame and J is the next frame, $\mathbf{A}$ an affine parameters of a deformation matrix, ω a weighting function that defines weights in the local window (it can be 1 or a normal distribution). After the tracker finds features to track, we merge groups of features as an object by using deterministic settings (Rabaud and Belongie 2006; Veenman et al. 2001). However, in the case of tracking pedestrians in DM (Sect. 5.1), where the number of initialization is relatively small, we partly adopt a particle filter (Koller-meier and Ade 2001) to track the ground trajectories.

5 Video to augmented aerial maps

Now we provide some technical details of our approaches and also present how the configurations of cameras and dynamic information described in Sect. 2 are analyzed and then visualized. Specifically, we first start with the simplest scenario, then we introduce more complex situations with different approaches.

5.1 Direct mapping from single video: pedestrians

Our first scenario is where we capture a video from a single viewpoint and we are interested in projecting it and the related motions onto an aerial view from an AEM. Specifically, we put a virtual characters onto the AEM to visualize pedestrians. This scenario requires direct mapping (DM) of tracked objects in a scene frame onto the virtual plane. This is the basic scenario and an essential building block for all of the other scenario cases described in Sect. 2. Our approaches here build on previous efforts aimed at surveillance-type applications (Girgensohn et al. 2007; Sawhney et al. 2002; Sebe et al. 2003; Wang et al. 2007).

We rely on direct mapping to visualize pedestrians in videos. As shown in Fig. 4, we first track pedestrians and extract screen-space coordinates and velocities. These measurements are directly registered onto the plane space of the virtual environment. If the homogeneous coordinates in a video frame are $\mathbf{p}_{\mathbf{x},\mathbf{y}} = [\mathbf{x}, \mathbf{y}, \mathbf{1}]^{\mathbf{T}}$, the new 2D location at planar space on virtual environment $\hat{\mathbf{p}}$ is simply calculated by $\hat{\mathbf{p}} = \mathbf{H}\mathbf{p}_{\mathbf{x},\mathbf{y}}$. Subsequently, if the objects (pedestrians) are moving in the video with the velocity $\mathbf{v}$, we can also project the velocity onto the earth map plane by $\hat{\mathbf{v}} = \mathbf{H}\mathbf{v}_{\mathbf{x},\mathbf{y}}$. This velocity is used to match simple motion data gathered off-line to visualize moving pedestrians. We used motion capture data from (CMU Graphics Lab Motion Capture Database http://mocap.cs.cmu.edu/) to generate similar character animations for our stand-in avatars. We first sample the trajectories of objects, then insert exactly one cycle of walking data onto them and interpolate the positions.

There is no accounting for visualizing movements that are outside the camera view and we are dependent on tracking approaches. This can be problematic when we have crowds in the scene and tracking fails due to occlusion. Some of these issues are addressed in the other scenarios, discussed next.

Note that at present, we are not interested in classifying objects or recognizing their states in the scene, which is beyond the scope of this work. In this scenario, we assume moving objects on a sidewalk are only pedestrians and they are engaged in simple walking motion and other similarly simple linear actions. On the road, we can assume the object of interest is a car.

In summary, DM can be used when (1) a region of interest covered by a single viewpoint (2) the motions of objects have to be simple. In Sects. 5.2 to 5.4, we will introduce methods to handle more complex situation in different scenarios.

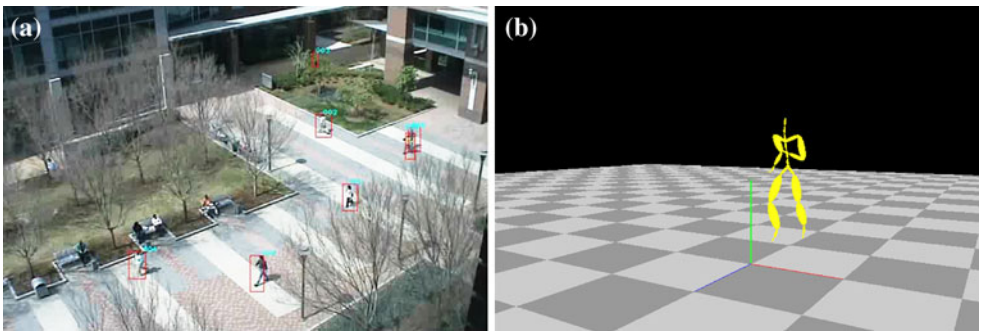

Fig. 4 Example of DM: **a** Pedestrians are tracked from single video. **b** Tracked pedestrians are matched to motion capture data, then mapped onto virtual plane space, then rendered only within a coverage of the given field of view

5.2 Overlapping cameras, complex motions: sports

We now move to the domain where we have overlapping views and motions that have some structure, but there are several motions at the same time. While we have employed this case for a variety of scenarios, we are going to demonstrate this in the domain of sports (other examples are also shown in video and in Fig. 12c). Sport is an interesting domain as we usually do have multiple views and in most instances we can rely on the field markings to help with registration. The overlapping views in this domain also require additional types of modeling and synthesis beyond the direct mapping from a single view (Sect. 5.1). Particularly, we also need to employ techniques to support view blending as the viewpoints are changed from one to the other, as we visualize the AEMs.

5.2.1 *Observations and estimation*

We start by obtaining field of views (FOVs) $\mathbf{f}_i$ and camera homographies $\mathbf{H}_i$ (from each view to a corresponding patch in the virtual plane) from the videos as described earlier. Then, the videos are rectified to top-views based on the extracted homographies and registered onto the corresponding regions of the virtual earth environment. Additionally, we also calculate camera locations via calibration. We rely on the fact that (1) we have multiple videos, and (2) ground yard commonly provides good features, *e.g.*, lines and known distances on the field.

Once the videos are registered, the rectified top views are used as a texture on the AEM plane. Then, this textured video is re-projected to a virtual view based on the model view matrix in the AEM environment. We refer to this view as *Back-projected view* and the angle between the original and the back-projected views as θ. Figure 5a shows the relationship between back-projected virtual view $\mathbf{f}_v$, rectified view $\hat{\mathbf{f}}_i$, and original video $\mathbf{f}_i$.

5.2.2 *View synthesis and blending*

Now, we address the issue of view synthesis as we move from one view to another within the AEM environment. Past approaches use billboard-style methods that change the orientation of the original view so that it always look at the virtual view (Wang et al. 2007). While commonly used in practice, this approach only shows rough segment of video streams at a given 3D environment. To visualize sports games, LiberoVision system (LiberoVision, http://www.liberovision.com/) uses segmentation of players and foreground warping. But it requires extensive batch-process time to make realistic view transition and only used in static scene. Seitz (1996) proposed view morphing methods where they reconstruct virtual views from the a pair of images under the assumption that the cameras have well-calibrated epipolar geometry. This can be done when we have precise stereo pairs and have exact pair sets of local correspondence. Thus, it is not the case for us where the camera center of the virtual views is often out of the epipolar line and finding the correspondences at every frame is computationally demanding (if not impossible).

We propose a more flexible global view blending method that can incorporate views and their subtracted backgrounds with overlapping regions and does not require an extensive pre/post-processing. This approach is adequate for the scenario of our system in which the rapid registration of crowd-sourced videos is needed.

Once multiple views covering the same region are registered onto the AEM plane, their rectified views also overlap. Our goal is to generate virtual views based on the two consecutive views that exhibit the most similar viewing angle θ. First, we search for the pair of the closest two views exhibiting small θ's. Let these consecutive views and the corresponding angles be denoted by $\mathbf{f}_i$, $\mathbf{f}_{i+1}$ and θ_i, θ_{i+1}, respectively (Fig. 5b). Then, we compute the two blending weights for both views based on the angle differences

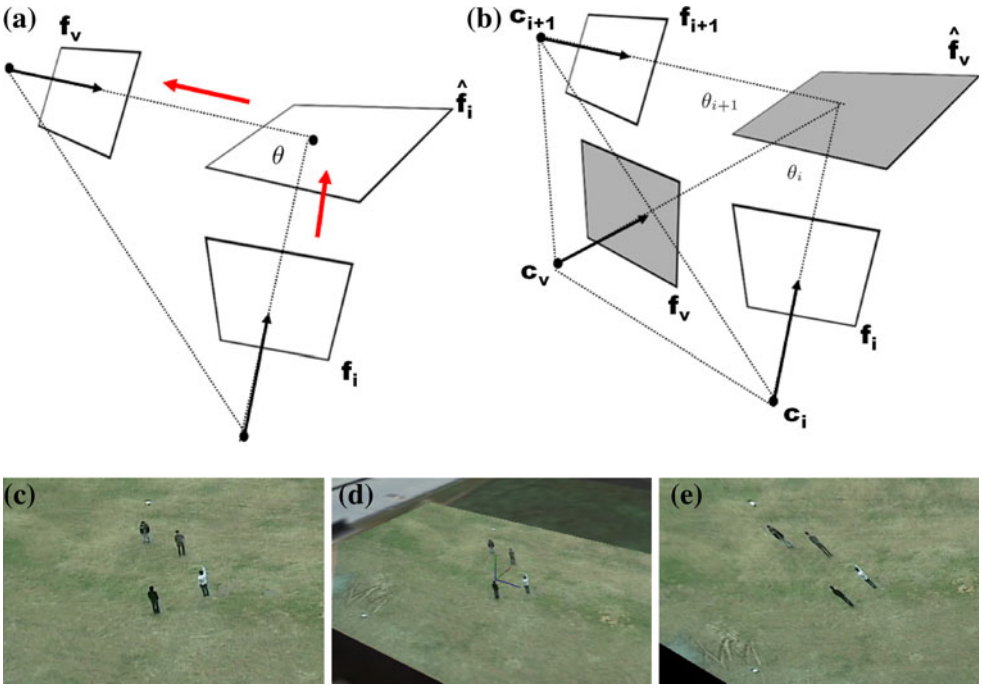

Fig. 5 **a** Virtual view $\mathbf{f}_v$, Sample view $\mathbf{f}_i$ and the angle between them θ. **b** Two consecutive views $\mathbf{f}_i$, $\mathbf{f}_{i+1}$, Back-projected virtual view $\mathbf{f}_v$ and blended rectified view $\hat{\mathbf{f}}_v$. Note that θ_i is an angle between $\mathbf{f}_i$ and $\mathbf{f}_v$ and θ_{i+1} is an angle between $\mathbf{f}_{i+1}$ and $\mathbf{f}_v$. **c** Source video having view $\mathbf{f}_i$. **d** Back-projected virtual view $\mathbf{f}_v$. **e** Rectified view $\hat{\mathbf{f}}_v$: Note that **c** and **d** look almost similar view since the θ is almost zero

giving larger weight to the smaller angle: $\omega_i = \frac{\theta_{i+1}}{\theta_i+\theta_{i+1}}$ and $\omega_{i+1} = \frac{\theta_i}{\theta_i+\theta_{i+1}}$. If the ω_i is close to one, which indicates that the angle θ_i becomes zero, so that the viewing vector of virtual view and the given view $\mathbf{f}_i$ are almost identical. This is the case where the virtual view is almost similar to $\mathbf{f}_i$, so that the virtually back-projected scene seems approximately invariant to distortions (See Fig. 5c, d). Even when ω_i is less than 1.0, the distortion of the back-projected view is still negligible within some ranges (noted as gray arrows in Fig. 7c). This example is shown in Fig. 6.

In case we cannot find a very similar view, we use interpolation and blending approach based on the weights computed from the angles between two adjacent views. In blending approach, if we blend $\mathbf{f}_i$, $\mathbf{f}_{i+1}$ using a linear interpolation, the blended region near the moving objects suffer from the ghosting artifacts, especially when both ω_i and ω_{i+1} become near 0.5, see Fig. 7d for an example. This is because the virtual view $\mathbf{f}_v$ is placed out of range from both $\mathbf{f}_{i+1}$ and $\mathbf{f}_i$, and the camera center is out of the line $\mathbf{C_iC_{i+1}}$ in Fig. 5b. Even if we use bicubic interpolation, the same problem occurs (Fig. 7e) although view transition becomes smoother.

In our solution, we blend not only a pair of rectified scenes $(\hat{\mathbf{f}}_i,\hat{\mathbf{f}}_{i+1})$ but also the subtracted background scenes generated from a pair of view videos. Now, suppose that $f(t)$ and $g(t)$ is a bicubic function for both views as shown in Fig. 7c. Then, we blend each pixel in the virtual view as follows:

$$\mathbf{p}_v = f(\omega_i)\mathbf{p}_i + g(\omega_i)\mathbf{p}_{i+1} + \omega_{\text{bkg}}\mathbf{p}_{\text{bkg}} \tag{2}$$

where $\omega_{\text{bkg}} = 1 - (f(\omega_i) + g(\omega_{i+1}))$, $\mathbf{p}_{\text{bkg}} = \omega_i\mathbf{p}_i^{\text{bkg}} + \omega_{i+1}\mathbf{p}_{i+1}^{\text{bkg}}$, and $\mathbf{p}_i^{\text{bkg}}$ and $\mathbf{p}_{i+1}^{\text{bkg}}$ are the pixels of the subtracted background computed separately from $\hat{\mathbf{f}}_i$ and $\hat{\mathbf{f}}_{i+1}$, respectively (See Fig. 7c).

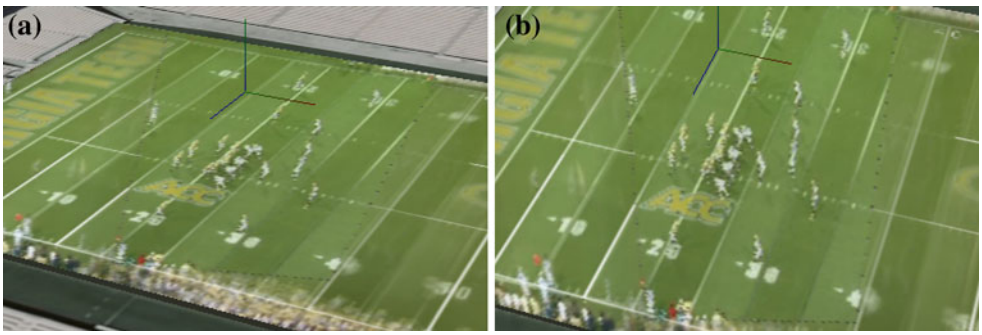

Fig. 6 The range of approximately invariant to distortion: **a** and **b** both are back-projected scenes from same video (notice that within small change of viewing angle it still looks reasonable)

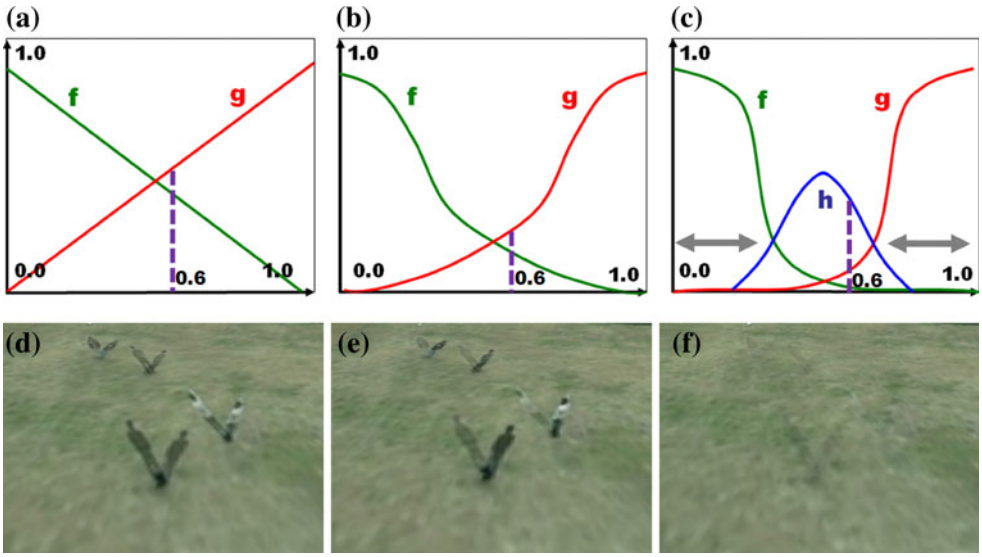

Fig. 7 View blending with different interpolations where ω_i is 0.6: **a** Linear **b** Bicubic **c** Bicubic and Background blending: f is $f(\omega_i)$, g is $g(\omega_{i+1})$ and h is ω_{bkg} which is a weight for subtracted background image. **d**–**f** The back-projected views based on each interpolation. The *gray arrows* in **c** show the ranges of ω where the distortion is minimized, as shown in Fig. 6a, b

Now, the virtual view is almost identical to the background if the target view is out of range from the both views. If the target view approaches a view to some extent, the synthesized view smoothly transit to the back-projected view of the closest viewpoint. Examples of smooth transition according to the change of virtual viewpoint is shown in the video. In Sect. 5.3, we introduce an approach to visualize moving objects when multiple videos are not overlapped and each camera is distributed sparsely.

5.3 Sparse cameras with simple motion: traffic

We demonstrate this scenario in the setting of analyzing videos of traffic and synthesizing traffic movements dynamically on AEMs. We are working in this domain following the ever-growing number of traffic cameras being installed all over the world for traffic monitoring. While the videos we are using are not recorded by actual traffic cameras installed by a department of transportation or the city government, our views are practically similar. The biggest technical challenge with this scenario is that as we have sparsely distributed, non-overlapping cameras, we do not have the advantage of knowing how the geometry or the movement from each camera is related to the other. While the topology of the camera networks still needs to be specified by user input, we do want to undertake the analysis of both registration of views and movement as automatically as possible.

To deal with the fact that we do not have any measurements in between the camera views, we also need to model the movements in each view in general and connect the observations between cameras, i.e. to model the flow from one view to another. To achieve this, in our framework, we add two functionalities. (1) Modeling the flow using a graph-based representation. This allows using the information obtained from observable regions and propagate it to infer a plausible traffic flow across unobservable regions. (2) Develop a synthesis framework that can be driven from such data. We employ a behavior-based animation approach to synthesize flow across view and onto the AEM. Fig. 8a shows the topology of traffic nodes. Each node is a patch of the road and has a traffic state X_i and a synthesized state Z_i. A measurement Y_i is defined only in observable nodes monitored by videos.

5.3.1 *Observation and measurements*

Using the tracking approach mentioned in Sect. 4, we can get estimates of position and velocities of objects from the video. The measurement of ith node Y_i consists of the positions and the velocities of the low-level motion features and the detected cars in the corresponding video frames. Once the low-level features within an observable node i are obtained, they are merged to form a set of objects $\mathbf{r}_i = \{r_{i,j}\}\forall j$—cars, with j denoting an index of each car, based on the similarity and the connectivity formed from the features using deterministic approach (Rabaud and Belongie 2006; Veenman et al. 2001). Experimental results demonstrate that it is sufficient for visualization and meet the overall purpose of the system. In summary, the measurement is defined to be the pair of the information on the low-level features $\mathbf{f}_i$ and the detected cars $\mathbf{r}_i$: $Y_i = \{\mathbf{f}_i, \mathbf{r}_i\}$.

The entire traffic system is also denoted as $\mathbf{X} = \{X_i | 1 \leq i \leq k_{\mathbf{X}}\}$, where $k_{\mathbf{X}}$ is the number of nodes. Additionally, the physical length of every ith node is obtained from the available geo-spatial database and is denoted by d_i. Once the traffic system is decomposed into a graph manually, a set of *observable* regions $\mathbf{M}(\mathbf{X}) \subset \mathbf{X}$ with available video measurements are identified. On the

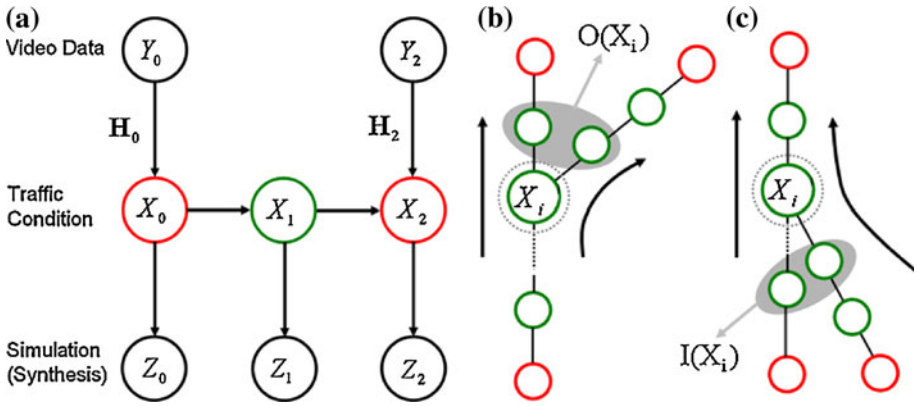

Fig. 8 Graph-based representation of traffic nodes: *Red nodes* indicate observable region $\mathbf{M}(\mathbf{X})$ and *Green nodes* are unobserved regions $\tilde{\mathbf{M}}(\mathbf{X})$. **a** The middle chain corresponds to the traffic conditions on the graph which represents the traffic system. Node index i is 0 to 2 in this example**. b** Split: outgoing regions **O(X_i)** from node X_i are marked. **c** Merging: incoming regions **I(X_i)** to node X_i are marked (color figure online)

other hand, the *unobservable* regions are denoted by $\tilde{\mathbf{M}}(\mathbf{X})$, where $\mathbf{X} = \mathbf{M}(\mathbf{X}) \cup \tilde{\mathbf{M}}(\mathbf{X})$. Note that every measurement Y_i is a set of low-level information computed from a video, rather than being the raw video itself.

The state X_i is designed to capture the essential information necessary to visualize traffic flow: (1) the rate of traffic flow n_i, and (2) the average velocity v_i of cars, i.e., $X_i = (n_i, v_i)$. By average flow, we mean the average number of cars passing through a region in unit time, whereas v_i denotes the average velocity of the cars.

The obtained measurement information Y_i is used to estimate an observable state $X_i \in \mathbf{M}(\mathbf{X})$, after the projection onto the virtual map plane using the available homography $\mathbf{H}_i$ for that region. First, the average speed $\hat{v}_i$ of the cars is estimated as an average of projected speeds of the low-level features w.r.t. the homography $\mathbf{H}_i$. Secondly, the flow of the cars passing through the ith region, $\hat{n}_i$, can be computed using the fact that the number of cars in the region $N_{\mathbf{r}_i}$ is the product of the flow multiplied by the average time d_i/v_i for cars to pass a region, i.e., $N_{\mathbf{r}_i} = \hat{n}_i \cdot (d_i/\hat{v}_i)$.

5.3.2 *Modeling from data: spatial correlation to infer missing data*

Once the set of states $\mathbf{M}(\hat{\mathbf{X}})$ are estimated for the observable regions, they are used to estimate the unobserved states $\tilde{\mathbf{M}}(\mathbf{X})$. We adopt the Bayesian networks (Pearl 1988) formalism to exploit the fact that the unknown traffic conditions $\tilde{\mathbf{M}}(\mathbf{X})$ can be estimated by propagating the observed information from the spatial correlation models. The whole traffic graph is a directed graph where an edge from a region X_j to another region X_i exists whenever traffic can move from X_j to X_i. For every node X_i, a local spatial model $P(X_i|\mathbf{I}(X_i))$ between the node X_i and the incoming nodes $\mathbf{I}(X_i)$ is specified (See Fig. 8). Once all the local spatial models are defined, the posterior traffic conditions $P(X_i|\mathbf{M}(\hat{X}_i)), \forall X_i \in \tilde{\mathbf{M}}(\mathbf{X})$ at the unobserved nodes are inferred using belief propagation (Frey and MacKay 1998, Pearl 1988).

In detail, we make an assumption that the average flow $X_i|_n$ of the cars in a region X_i matches the sum of the average flow of the cars from the incoming regions $\mathbf{I}(X_i)$ with a slight variation w_i which follows white Gaussian noise: $X_i|_n = \sum_{j \in \mathbf{I}(X_i)} X_j|_n + w_i$. For velocity, we assume that the average speed in a region matches the average speed of the cars with a slight variation q_i which is again a white Gaussian noise: $X_i|_v = (\sum_{j \in \mathbf{I}(X_i)} X_j|_v)/N_{\mathbf{I}(X_j)} + q_i$. The variance of the Gaussian noises, both w_i and q_i, are set to be proportional to the length d_i of the target region i.

The above assumptions are applied in case the road configuration is straight or merging (see Fig. 8a,c). For the case of splitting (see Fig. 8b), we make a different assumption on the dependencies for the flow $X_i|_n$. Let us denote the outgoing regions (children) of a node X_i by $\mathbf{O}(X_i)$. To maintain a tractable solution for the overall inference phase using belief propagation, the dependencies $P(X_j|X_i)$ between the parent node X_i and every child node $X_j \in \mathbf{O}(X_i)$ are encoded individually in a Gaussian form. The new assumption is that there is a known ratio on the traffic portion $0 \leq \alpha_j \leq 1$ $(\sum_{\forall j} \alpha_j = 1)$ flowing from the parent node X_i to an outgoing node X_j. Then, the individual functional dependency is modeled as follows: $X_j|_n = \alpha_j X_i|_n + u_j$ where a slight white Gaussian noise is denoted by u_j, which is again proportional to the region length d_j. Note that the flow estimate at an outgoing region $X_j|_n$ is estimated based on not only the propagated information $\alpha_j X_i$ from its parent, but also on the information propagated back from its descendants in the directed traffic region graph.

The posteriors $P(X_i|\mathbf{M}(\hat{X}_i))$ for the unobserved regions are concise Gaussians as all the spatial correlations are Gaussians. In the simplest case, e.g., straight configurations, the maxima (mean) of the resulting posterior estimates $\hat{X}_i$ is analogous to the linear interpolation of the nearest observable regions.

5.3.3 Synthesis: parameterized traffic simulation

To visualize traffic flow based on the estimated traffic states $\hat{\mathbf{X}}$, we developed a parameterized version of Reynolds' behavior simulation approach (Reynolds 1987) where we adopted the steering behavior (Reynolds 1999) for our implementation. By parameterized behavior simulation, we mean that the cars are controlled by the associated behavior-based controller, but the behaviors are parameterized by the current traffic condition. The controller of a car in the i-th region is parameterized by X_i. Hence, the behavior of a car varies if the estimated traffic condition within a region changes or if the car moves onto adjacent traffic regions.

The flock simulation status within the i-th region is denoted by $Z_i = \{s_{i,k}\}_{k=1}^{N_{Z_i}}$ which comprises of a set of simulated cars $s_{i,k}$ with corresponding position $s_{i,k}|_p$ and velocity $s_{i,k}|_v$ attributes, where N_{Z_i} denotes the total number of cars. The synthesis of the flock status Z_i^t at time t should be consistent with both the already existing cars from the past scene Z_i^{t-1} and the newly added cars coming in from the virtual scenes corresponding to the incoming regions $\{Z_{i \in \mathbf{I}(j)}{}^{t-1}\}$. In terms of the adjustments of the speed $s_{i,k}|_v$ between the frames, the cars adjust their speeds to meet the estimated average speed of the region $\hat{X}_i|_v$ where their accelerations are set in such a way that the speeds are approximately linearly interpolated within the region. Figure 9 shows an example of successful simulation in (a) normal and (b) heavily loaded traffic.

5.4 Sparse cameras with complex motion: clouds

Our final scenario is aimed at using videos of natural phenomenon, primarily clouds and adding them to AEMs for an additional sense of reality. This is a hardest case, since clouds consist of particles, thus it does not have a specific shape. Moreover, the coverage of field of view from each observing camera is relatively small compared to the region displayed as sky. In this scenario, we use video of moving clouds to extract procedural models of motion, which are then interpolated and rendered onto the AEM, with a compelling effect. Some of the AEMs are now moving towards rendering clouds from satellite imagery and our work is similar to their goals, but we use sparse video and use procedural modeling to generate the clouds, rather than playback what was captured directly.

5.4.1 Observation and extraction

In this scenario, cameras are spatially distributed and only a small subset of the targeted sky area that is to be synthesized is visible by the FOVs of the cameras. The entire sky scene is synthesized by: (1) the extracted local clouds density within the observed regions based on the video, (2) the interpolated cloud density information within the unobserved regions from the radial basis function (RBF) technique, and (3) entire cloud imagery synthesis based on the cloud template obtained by procedural synthesis techniques and the computed cloud density map. For measuring dynamic movement of clouds, we also extract velocities from videos. We assume that the video used to extract velocity is always taken by camera having 90 degree of elevation, and zero degree azimuth. We call this video an *anchor video*.

5.4.2 Cloud density interpolation from multiple sky videos

We use a radial basis function (RBF) (Buhmann and Ablowitz 2003) to globally interpolate density of clouds in unobserved sky region based on multiple videos. The main concept of our interpolation follows a method described in Turk and O'Brien (1999). They interpolate an implicit surface from a given set of scattered data points. We use this method to interpolate density of unobservable region in the sky using densities extracted from given set of clouds videos. In our work, constraint points are the location of feature points(x_i, y_i) extracted from each input video, and the basis vector is defined as $\mathbf{G} = [G_1(x_1, y_1), \ldots, G_n(x_n, y_n)]^\top$ encoded by strong gradient and velocity vectors representing

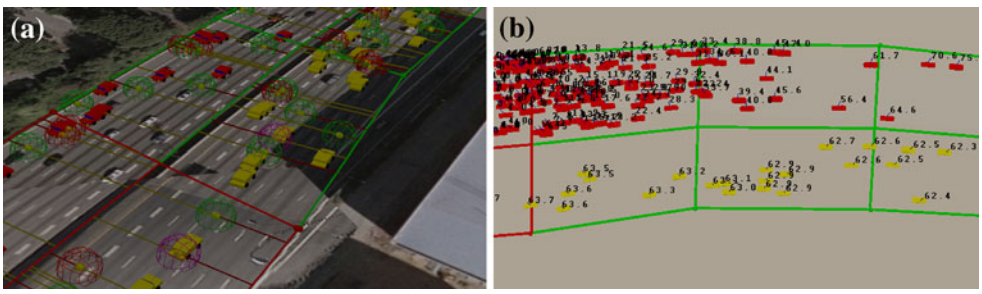

Fig. 9 Simulated cars: **a** Each object (car) acts as an autonomous agent controlled by (1) local reaction based on their own perception range, (2) global dynamics extracted from video. **b** Cars reaction when average speed of upper-left corner is forced to be very slow, while other corners are controlled by real data

density of clouds. Now, a basis function d_{ij} between any constraints points is chosen as $||(x_i - x_j)^2 + (y_i - y_j)^2||$.

Using these measurements, we can globally interpolate the cloud density of any points in unobserved sky region by weighted sum of basis function. The weights for the interpolated density is obtained by solving following linear system:

$$\begin{bmatrix} d_{11} & \cdots & d_{1n} & 1 & x_1 & y_1 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ d_{n1} & & d_{nn} & 1 & x_n & y_n \\ 1 & \cdots & 1 & 0 & 0 & 0 \\ x_1 & \cdots & x_n & 0 & 0 & 0 \\ y_1 & \cdots & y_n & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \vdots \\ \lambda_n \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} G_1 \\ \vdots \\ G_n \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (3)$$

where the λ_i are weights, and c_i is a degree one polynomial that accounts for the linear and constant portions of density. More details of underlying basis for RBF interpolation is described at Turk and O'Brien (1999).

Now, the generated density map is used as a density function for procedural modeling of cloud textures. Figure 10a, b shows an example of density map generated from four videos. However, if the generated density map is not appropriate due to mis-detection of feature vectors, the system provides an user interface to edit the density map by adding additional basis vectors.

5.4.3 *Synthesis: procedurally generated clouds*

A procedural texture is a synthetic image generated based on random functions (Lewis 1989; Perlin 1985). We used the 2D version of the approach suggested by (Man 2006) to synthesize a 2D cloud template. In detail, the function f below is used to generate a cloud template containing a set of weighted sub-templates at multiple scales. A random number generator g is applied to each scale with different seeds in such a way that the details of each particular template increases as follows:

$$f(x, y) = \left| \sum_{i=0}^{K-1} P^i \cdot g\left(2^i \cdot x, 2^i \cdot y\right) \right| \text{ where } P \subset [0, 1]$$

where K is the number of scales and P denotes the persistence constant. Note that the sub-template for the lower scale having smoother characteristics has larger weight compared to the sub-template for the higher scale. While there are many options for the random number generator g, we used the function suggested by Man (2006).

We also generate multiple cloud templates with different persistent levels and seeds for realistic sky movement. Once cloud textures are generated, we make additional layers having different effects to generate realistic sky animation. Similar approaches are in use for real-time sky-rendering systems (Harris 2005; Wang 2004). The sun mask and glow mask are generated based on the time (i.e., At noon, sun would be appeared on the center of texture). A sky map is used to make color of sky to be brighter if the location is near the ground. Then, the finally generated sky textures are mapped onto the sky domes (Zotti and Groller 2005). To visualize sky, representing a dynamic of current sky, the mapped sky textures moves based on the velocity captured from anchor video.

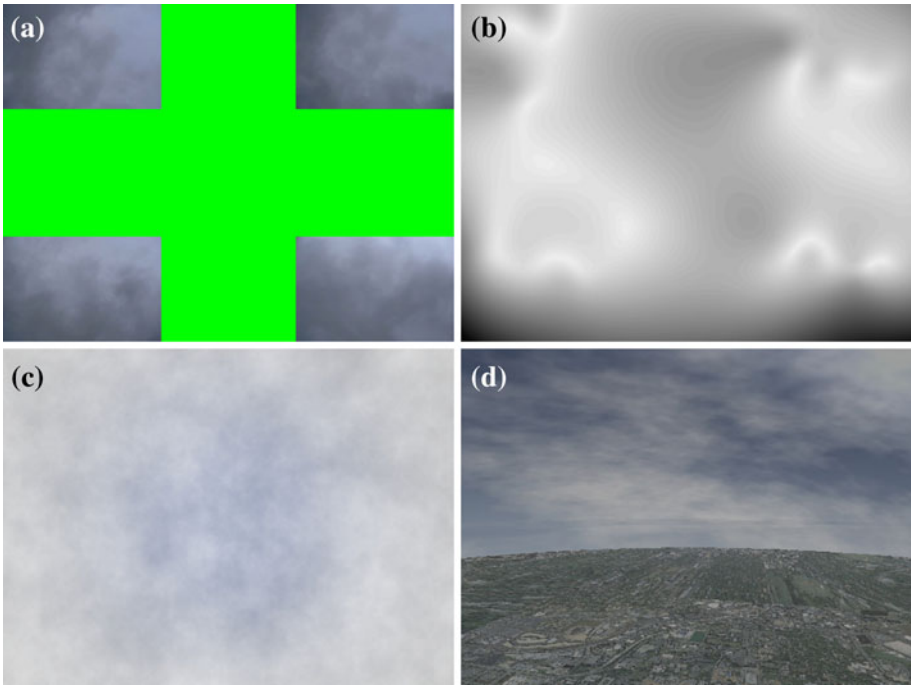

Fig. 10 Generating clouds layers procedurally using videos: **a** 4 input videos (*green region* is an unobserved region). **b** Globally interpolated density map from RBF. **c** Resultant cloud layer. **d** Registered clouds in the sky dome in Earth Map environment (color figure online)

6 Discussion of experiments and results

To validate our approaches, we undertook a series of experiments for different scenarios on a variety of challenging domains, under varying conditions. For the direct mapping scenario, cars and pedestrians are animated based on single videos where we used available motion capture data to animate individual pedestrians. For the view blending examples of sports and other instances, videos are collected from three different testing locations where a total of 10 cameras were used. For the football game visualization in the stadium, we used the video footage provided by the Georgia Tech Athletic Association (made available for research purposes). For applications of traffic visualization, 6 different data sets are collected with number of video feeds varying from 2 to 8 each, where 3 data sets were captured at an identical site but at different time and under different weather conditions. Finally, for the cloud visualization, we collected four different data sets where 3 to 6 cameras were used each. For each cloud data set, a separate anchor video was captured to measure the velocity of clouds. Since a variety of the above experiments are showcased in the enclosed video, we would briefly highlight the results in this section.

The prototype system is developed using C++/OpenGL on a computer with Quad-core 2.5 GHz, 2 GB RAM, and NVidia Quadro FX770M graphics card. The resulting visualizations are rendered in real time at approximately 20 frames per second where 500 targets can be tracked at maximum for the traffic flow scenario.

Through our experiments, the scenario (Fig. 12a–c) that requires view and background blending demonstrates view transitions that are smooth and provides dynamic visualizations.

In the traffic scenario, the visualized traffic closely matches the average speeds of the real traffic system. However, it was noted that the quality of the estimated traffic flow deteriorates in proportion to the distance between the cameras. The cameras used in our results are placed no more than 0.5 miles away and provide qualitatively plausible visualizations. Nonetheless, it can be observed that our results show fairly accurate *global tendency*, which reflects real-world average flows (velocities). However, it is important to note that it does not guarantee the exact individual position of every car in unobserved regions because the positions of cars in the unobserved region are designed to be driven by behavioral model where target velocities are computed through interpolation of observed ones.

To evaluate the parameterized flock behavior, we artificially forced a very slow speed only to the left-top observable node, as already shown in Fig. 9b, while the other nodes still received live information, to mimic a traffic congestion. The objects approaching the congested region changed their velocity sharply but realistically under the poised behavioral rules.

Our system produces reliable results under diverse lighting and weather conditions for traffic visualization (Fig. 13). In terms of the test results on a challenging road topology, Fig. 12d, e depicts a scene with curved roads that merge into a speedy highway. The cars simulated based on the live videos successfully adjust their speed on the curved roads (approximated by a series of rectangular regions) after which they accelerate into the highway to catch up with the average speed.

In the challenging large-scale experiments shown in Figs. 8, 12f cameras are installed at different locations where the visualized road system consists of 13 observed and 26 unobserved regions. Some cameras observe one-way road, while others observe two-way, and the traffic topology include merge, exits and bridge crossings.

The results for the DM scenario (Fig. 12h, i) demonstrate that pedestrians and cars are animated properly based on the successful tracking results.

Various styles of sky and clouds are generated as shown in Figs. 10 and 12j. Although the visualization results do not capture the exact shape of the individual clouds or the exact sky atmosphere, movement and density of the distributed clouds reflect the characteristics of the input videos plausibly.

Finally, it is worth noting that the interface developed in our prototype system allows us to make the customized

Fig. 11 Various results of sky rendering based on multiple videos at different times

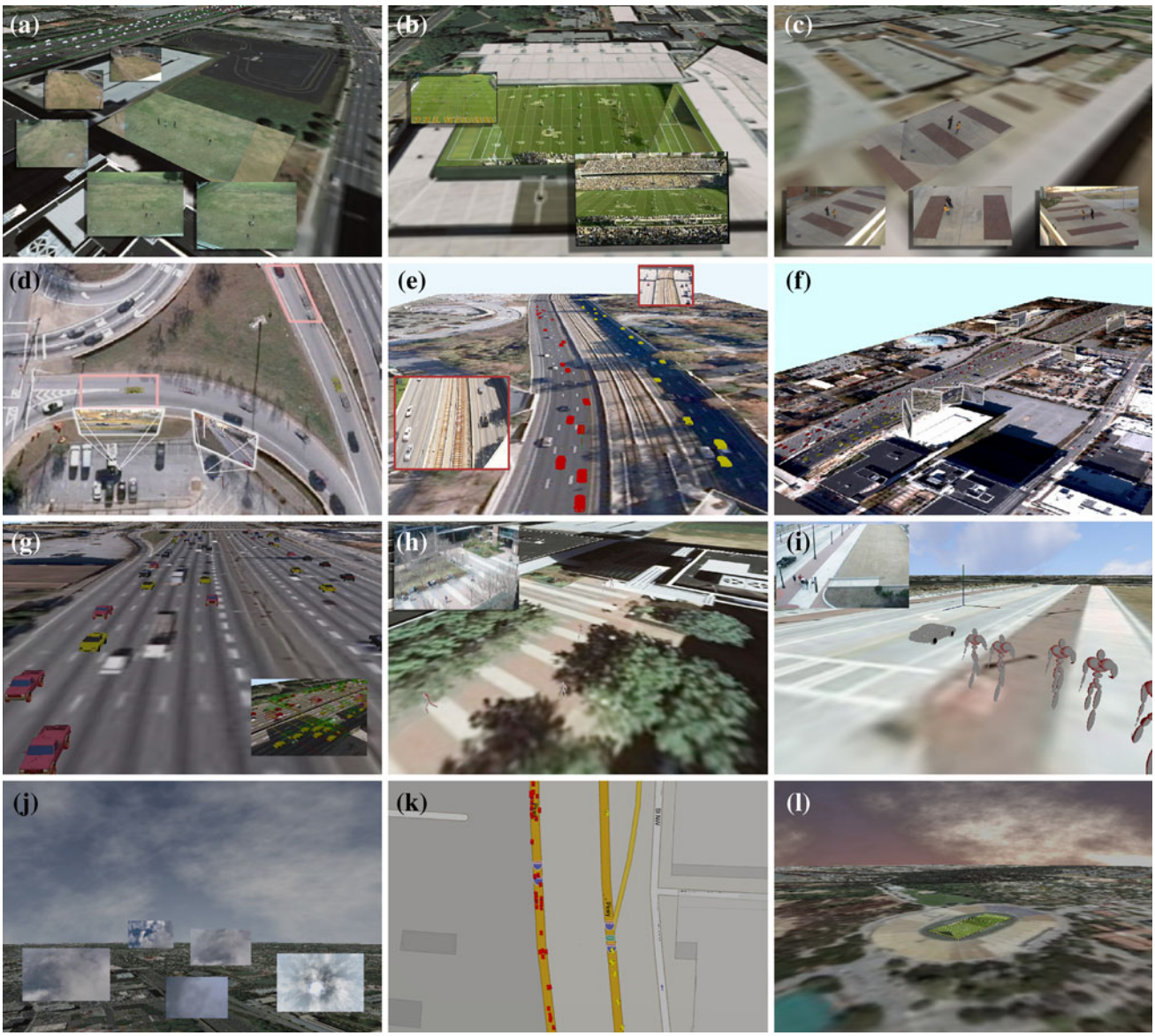

Fig. 12 Results from our prototype system using 36 videos: *1. OCCM (View Blending):* (**a**). 5 Cameras for soccer game (**b**). Two broadcasting footages of NCAA Football game (**c**). Three surveillance cameras. *2. SCSM (Traffic):* (**d**). Merging Lanes (**e**). The steering behaviors of cars are demonstrated in the slightly curved way (**f**). 8 Cameras for larger-scale traffic simulation including merge and split (**g**). Rendered traffic scene and corresponding simulated scene *3. DM (Pedestrians):* (**h**). Direct mapping of pedestrian having simple motion (**i**). Visualization of pedestrians and cars in the street *4. SCCM (Clouds):* (**j**). Four videos for clouds and sky generation *5. Customized visualization using user interface:* (**k**). Traffic flow are visualized in a line map by adjusting the scale of moving objects (**l**). The game scenes can be augmented into different stadium. Please see the video on the project website

visualizations with ease. Figure 12k demonstrates the traffic visualization on the line map (i.e. google map), and Fig. 12l shows an example where we map a football game onto a different stadium far away from the original location.

7 Summary, limitations, and discussion

In this paper, we introduced methods to augment Aerial Earth Maps (AEMs) with diverse types of real-time information, namely pedestrians, sports scenes, traffic flows, and skies. The proposed set of solutions are targeted to address different types of scenes in terms of camera network configuration/density and the dynamism presented by the scenes. The prototype system which integrates all the components run in real time and demonstrates that our work provides a novel, more vivid, and more engaging virtual environment through which the users would browse the cities of now. Our work showcases a unique approach that provides improved experience for users to visually

Fig. 13 *Qualitative evaluation of traffic in different lighting and weather conditions* Rendered Dynamic scenes based on (*Top*) video data captured at night: Resulting scene demonstrates sparse traffic distribution with high velocity across almost entire nodes, (*Bottom*) video data captured during snowing weather condition: Despite heavy noises in the video due to snow, the result shows flows that realistically demonstrate real-world traffic; one lane (marked as *red*) shows densely populated heavy traffic, while the other lane (*yellow*) is relatively sparser (color figure online)

consume "what is happening where", to augment the conventional text-web-browsing experience currently provided by static AEMs.

It is important to note a few of our limitations. First, the direct mapping method only visualizes the object moving within its view, and movement modeling outside is only possible when overlapping views are available. Second, the tracking within the direct mapping and the traffic flow estimation assume that the occlusion among targets is modest, and the target association can be computed relatively accurately. Accordingly, none of our current approaches for tracking will work for dense crowds or for low-frequency videos (which are often the case for many surveillance cameras), due to significant occlusions. Third, the tracking algorithms used in our work are not able to recognize the detailed postures of complex targets, e.g., human postures. While efforts have been made to handle this task in computer vision (e.g., Efros et al. 2003; Ramanan and Forsyth 2003), practical performance still depends on specific constraints. Fourth, we cannot directly apply our traffic flow approaches to the scenes with higher-level controls and behaviors, e.g., intersections with traffic lights, which is an interesting avenue for future research. Finally, our solutions do not support modeling or analysis of high-level semantic information, e.g., car accidents or street burglaries.

We do note that the widespread distribution of cameras, which are used by our approach, rightfully brings up serious privacy issues. This is due to the concerns of direct exposure of individuals (and their privacy) to massive distribution of public surveillance cameras (Wood et al. 2006). *Street Views* by Google has similarly raised privacy concerns because it sometimes displays people.

In our current work, we are intentionally "abstracting" individuals and our camera distance to individuals is large enough that personal identity is hard to ascertain. We believe our approach of symbolizing moving objects by tracking without identification can alleviate the privacy concerns. However, we do see the concerns raised and propose that the data sources and the entire system can be managed properly. In the present prototype system, they are explicitly controlled.

It is also worthwhile to discuss a bit more on the potential of our system. *First*, OCCM application can be

used as the interactive sport broadcasting where multiple broadcasting cameras or crowd-sourced videos incorporate to visualize the sport event on virtual environment. It will allow users to actively watch sports games by selecting favorite views and regions of interest. *Secondly*, new types of advertisement could be brought into our system. For instance, crowd events within the field of views of public cameras can be used as the novel types of marketing methods by immediate visualization in the virtual environment or map. *Third*, our approach provides visually fused information that integrates individual sensor observations and is potentially useful for surveillance, security, and military applications. *Finally*, our system can be used as a mirror that one can interact between real worlds and virtual worlds. This is possible because *Augmented Reality* (AR) techniques allow us put the virtual objects onto the screen, which displays the real-world scene, while our approach puts the present dynamic information onto virtual world. Imagine your avatar in the virtual world application such as Second Life (Linden Research, http://www.liberovision.com/) looking at the moving objects that are actually reflecting the movements in the real world and vice versa. We believe such interactions can be an important milestone for the future of virtual reality as well as new types of cyber-cultures (Smart et al. 2007).

In summary, we have presented a novel prototype system to augment dynamic virtual cities based on real-world observations and spatio-temporal analysis of them. We feel that our approach opens doors for interesting forms of new augmented virtual realism.

In our future work, we aim to overcome the above limitations and incorporate even more types of additional dynamic information such as river, swinging forest, sun, weather patterns, environmental condition and even aerial objects like birds and airplanes. Additionally, we would like to apply our system onto various types of possible applications mentioned above.

Acknowledgments This project was in part funded by a Google Research Award. We would like to thank Nick Diakopoulos, Matthias Grundmann, Myungcheol Doo and Dongryeol Lee for their help and comments on the work. Thanks also to the Georgia Tech Athletic Association (GTAA) for sharing with us videos of the college football games for research purposes. Finally, thanks to the reviewers for their valuable comments.

References

Bouguet J-Y (2003) Pyramidal implementation of the lucas kanade feature tracker. In: Intel Corporation

Buhmann MD, Ablowitz MJ (2003) Radial basis functions: theory and implementations. Cambridge University, Cambridge

Chen G, Esch G, Wonka P, Müller P, Zhang E (2008) Interactive procedural street modeling. ACM Trans Graph 27(3):1–10

Efros AA, Berg EC, Mori G, Malik J (2003) Recognizing action at a distance. In: ICCV03, pp 726–733

Frey B, MacKay D (1998) A revolution: belief propagation in graphs with cycles. In: Neural information processing systems, pp 479–485

Girgensohn A, Kimber D, Vaughan J, Yang T, Shipman F, Turner T, Rieffel E, Wilcox L, Chen F, Dunnigan T (2007) Dots: support for effective video surveillance. In: ACM MULTIMEDIA '07. ACM, New York, pp 423–432

Harris MJ (2005) Real-time cloud simulation and rendering. In: ACM SIGGRAPH 2005 Courses. New York, p. 222

Hartley RI (1997) In defense of the eight-point algorithm. PAMI Int J Pattern Anal Mach Intell 19(6):580–593

Hartley R, Zisserman A (2000) Multiple view geometry in computer vision. Cambridge University Press, Cambridge

Horry Y, Anjyo K-I, Arai K (1997) Tour into the picture: using a spidery mesh interface to make animation from a single image. In: Proceedings of ACM SIGGRAPH, New York, pp 225–232

Kanade T (2001) Eyevision system at super bowl 2001. http://www.ri.cmu.edu/events/sb35/tksuperbowl.html

Klein G, Murray D (2007) Parallel tracking and mapping for small AR workspaces. In: Proceedings of sixth IEEE and ACM international symposium on mixed and augmented reality (ISMAR'07)

Koller-meier EB, Ade F (2001) Tracking multiple objects using the condensation algorithm. JRAS

Kosecka J, Zhang W (2002) Video compass. In: Proceedings of ECCV. Springer, London, pp 476–490

Lewis JP (ed) (1989) Algorithms for solid noise synthesis

Man P (2006) Generating and real-time rendering of clouds. In: Central European seminar on computer graphics, pp 1–9

Pearl J (1988) Probabilistic reasoning in intelligent systems: networks of plausible inference. Morgan Kaufmann, Massachusetts

Perlin K (1985) An image synthesizer. SIGGRAPH Comput Graph 19(3):287–296

Rabaud V, Belongie S (2006) Counting crowded moving objects. In: CVPR '06: Proceedings of IEEE computer vision and pattern recognition. IEEE Computer Society, pp 17–22

Ramanan D, Forsyth DA (2003) Automatic annotation of everyday movements. In: NIPS. MIT Press, Cambridge

Reynolds CW (1987) Flocks, herds and schools: a distributed behavioral model. In: ACM SIGGRAPH 1987. ACM Press, New York, pp 25–34

Reynolds CW (1999) Steering behaviors for autonomous characters. In: GDC '99: Proceedings of game developers conference. Miller Freeman Game Group, pp 768–782

Sawhney HS, Arpa A, Kumar R, Samarasekera S, Aggarwal M, Hsu S, Nister D, Hanna K (2002) Video flashlights: real time rendering of multiple videos for immersive model visualization. In: 13th Eurographics workshop on Rendering. Eurographics Association, pp 157–168

Sebe IO, Hu J, You S, Neumann U (2003) 3d video surveillance with augmented virtual environments. In: IWVS '03: First ACM SIGMM international workshop on Video surveillance. ACM, New York, pp 107–112

Seitz SM, Dyer CR (1996) View morphing. In: SIGGRAPH '96. ACM, New York, pp 21–30

Seitz SM, Curless B, Diebel J, Scharstein D, Szeliski R (2006) A comparison and evaluation of multi-view stereo reconstruction algorithms. In: IEEE CVPR '06, pp 519–528

Shao W, Terzopoulos D (2007) Autonomous pedestrians. Graphi Models 69(5):246–274

Shi J, Tomasi C (1994) Good features to track. In: Proceedings of IEEE CVPR. IEEE computer society, pp 593–600

Smart J, Cascio J, Paffendorf J (2007) Metaverse roadmap: pathways to the 3d web. Metaverse: a cross-industry public foresight project

Snavely N, Seitz SM, Szeliski R (2006) Photo tourism: exploring photo collections in 3d. In: Proceedings of ACM SIGGRAPH'06. ACM Press, New York, pp 835–846

Treuille A, Cooper S, Popović Z (2006) Continuum crowds. In: ACM SIGGRAPH 2006 papers, pp 1160–1168

Turk G, O'Brien JF (1999) Shape transformation using variational implicit functions. In: SIGGRAPH '99. New York, NY, pp 335–342

Veenman CJ, Reinders MJT, Backer E (2001) Resolving motion correspondence for densely moving points. IEEE Trans Pattern Anal Mach Intell 23:54–72

Wang N (2004) Realistic and fast cloud rendering. J Graph Tools 9(3):21–40

Wang Y, Krum DM, Coelho EM, Bowman DA (2007) Contextualized videos: Combining videos with environment models to support situational understanding. Abdom Imag 13:1568–1575

Wood DM, Ball K, Lyon D, Norris C, Raab C (2006) A report on the surveillance society. Surveillance Studies Network, UK

Yilmaz A, Javed O, Shah M (2006) Object tracking: a survey. ACM Comput Surv 38(4):13

Zotti G, Groller ME (2005) A sky dome visualisation for identification of astronomical orientations. In: INFOVIS '05. IEEE Computer Society, Washington, DC, p 2