Augmenting Aerial Earth Maps with Dynamic Information

Kihwan Kim, Sangmin Oh, Jeonggyu Lee and Irfan Essa *
School of Interactive Computing, GVU Center, Georgia Institute of Technology
http://www.cc.gatech.edu/cpl/projects/augearth/ †

ABSTRACT

We introduce methods for *augmenting* aerial visualizations of Earth (from services like Google Earth or Microsoft Virtual Earth) with *dynamic information* obtained from videos. Our goal is to make *Augmented Aerial Earth Maps* that visualize an alive and dynamic scene within a city. We propose different approaches for analyzing videos of cities with pedestrians and cars, under differing conditions and then created augmented *Aerial Earth Maps* (AEMs) with live and dynamic information. We further extend our visualizations to include analysis of natural phenomenon (specifically clouds) and add this information to the AEMs adding to the visual reality.

1 INTRODUCTION

Earth can be visualized on the Internet with the growth of online Aerial Earth Maps(AEMs) services (*e*.g., Google Earth, Microsoft Virtual Earth, *e*tc.). 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. Information such as traffic, restaurant locations, tourist sites, and other services is provided, within a geo-spatial database. However, such visualizations, while rich in information, are static and do not showcase the dynamism of what is happenning in real world. We are motivated to add dynamic information to such online visualizations of the globe.

In this paper, we introduce an approach to generate ALIVE cities. So that one can browse and see a city with dynamic and alive Aerial Earth Maps. Our approach relies on analysis of videos from different sources around the city. Fig. 1 shows a static still of such an augmented visualization driven by analysis and subsequent registration of 36 video sources.

To achieve this goal, we have to address several technical challenges. *First*, we develop a framework to extract information about the geometry of the scene, the status of the scene and also the movements in the environment from video. *Second*, we 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, 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 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.

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 see people moving in different situations, and also show traffic motions. We also want to show moving clouds. In each of the above instances, we have to deal with different viewpoints of videos and in many cases with incomplete information. We also 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 consider four scenarios that address the distribution of cameras and motion of objects. We describe briefly these cases here. The results showcase the variations in configurations in how we generate a variety of augmented visualizations of live cities.

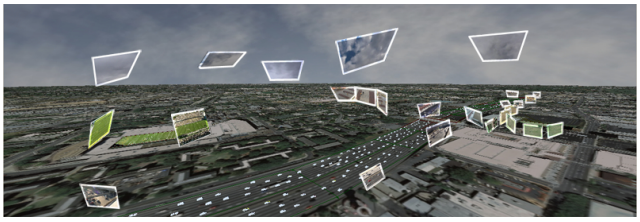


Figure 1: An overview of the Augmented Earth Map generated by our system. We make use of 36 videos to add dynamic information to city visualization. Information from each input video of traffic, people, and clouds is extracted, then it is mapped onto the Earth map in real-time. See video on our website.

#1. Direct Mapping: 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: Several cameras with overlapping views observe a relatively small region concurrently. The motion within the area is complex. We demonstrate this case with people playing sports.

#3. Sparse Cameras with Simple Motions: 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: Cameras are sparsely distributed and each of them observes a different part in a larger area and the motion of the target scene is complex. This is the case where we observe and model natural phenomena such as clouds in the sky.

3 RELATED AND MOTIVATING PAST WORK

Our work builds on existing efforts in computer vision on tracking objects [16, 13] and multi-view registration [5]. We also rely on behavioral animation approaches [8, 9] from computer graphics.

We leverage on the work of Seitz and Dyer [12], which uses morphing with camera models to generate in-between views (view synthesis). They reconstruct intermediate views using precise stereo pairs and local correspondences. In our approach, where a large amount of videos are available and need to be registered in real-time, we use global blending approaches to register a number of views and visualize them immediately in the AEMs.

Harris [4] introduces a method for rendering realistic clouds using imposters and sprites generated procedurally for real-time games. While extremely realistic, these do not suit our purposes as we are interested in driving clouds from video data. Turk and

*e-mail: {kihwan23,sangmin,glaze,irfan}@cc.gatech.edu
†Videos and other details available from our website.

(a) (b)

Figure 2: Example of DM: (a)Single video observing pedestrian and car (b)Screen space position mapped onto virtual plane space

O'Brien [14] uses RBF interpolation to find an implicit surfaces from scattered data using constraint points. We also rely on Perlin noise [7] for generating clouds volume or sprites for generating clouds maps.

A work most closest to our work is the Video Flashlight system [10]. This is a surveillance application, which tracks people in a fixed region using multiple cameras and maps onto the observed region. Several other similarly motivated surveillance applications have also been introduced [11].

4 Video to Augmented Aerial Maps

Now we provide technical details of our approaches and also present how the configurations of cameras and dynamic information described in section 2 are analyzed and then visualized. We first start with the simplest scenario, then we introduce more complex situations with different approaches. More details about each approach are available from our project website.

4.1 Direct mapping from Single Video: Pedestrains

In our first scenario, we have video from a single viewpoint and we are interested in projecting the video and the related motions onto an aerial view from an AEM. This scenario requires direct mapping of tracked objects in a scene frame onto the virtual plane. This is the simplest of our scenarios and in essence a basic building block for all of the other scenario cases described in Section 2.

We rely on direct mapping from video to geometry to visualize pedestrians in videos. As shown in Fig. 2, we first track [1, 13] the pedestrian and extract screen-space coordinates and velocities. These measurements are directly registered onto plane space of virtual environment. If the homogeneous coordinates in 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 $\hat{\mathbf{p}} = \mathbf{H}\mathbf{p}_{\mathbf{x},\mathbf{y}}$, where $\mathbf{H}$ is a planar homography. 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 human motion capture data gathered off-line (from [15]) to visualize moving pedestrians. We first sample the trajectories of objects, then insert exact one cycle of walking data onto them and interpolate the positions.

In our current implementation, we do not classify objects or recognize their states in the scene. We assume all moving objects on a sidewalk are pedestrians walking or doing some simple linear actions. On the road, we assume the object of interest is a car moving.

Direct mapping, as described here is used when (1) a region of interest is covered by a single view point, and (2) the motions of objects simple. In Sections 4.2-4.4, we introduce methods to handle more complex scenarios.

4.2 Overlapping Cameras, Complex Motions: Sports

We now move to the domain where we have videos with overlapping views and motions that have some structure, with several motions occurring at the same time. While we have employed this case for a variety of scenarios, we demonstrate it here in the domain of sports.

Sports videos usually 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 single view (Section 4.1).

(a) (b)

Figure 3: The range of approximately invariant to distortion:(a) and (b) both are back-projected scenes from same video.

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 AEM) 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 AEM.

For each video, the rectified top view is 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 view and its the back-projected view as θ.

Once multiple views covering the same region are registered onto the AEM plane, their rectified views are also overlapped. 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 and denote rectified planar texture as $\hat{\mathbf{f}}_i$ and $\hat{\mathbf{f}}_{i+1}$.

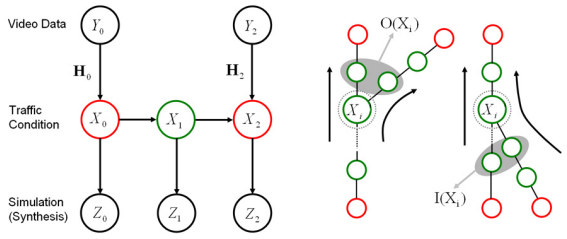
Then, we compute the two weights for both views based on the angle differences where the smaller angle leads to a larger weight : $\omega_i = \frac{\theta_{i+1}}{\theta_i+\theta_{i+1}}$ and $\omega_{i+1} = \frac{\theta_i}{\theta_i+\theta_{i+1}}$. If the ω_i is close to one, which indicates that the angle between the virtual view and the given view $\mathbf{f}_i$ are similar, then it is safe for a fragment shader to render the back-projected view based on $\hat{\mathbf{f}}_i$ only. This is the case where the virtual view is similar to $\hat{\mathbf{f}}_i$, so that the virtually back-projected scene seems approximately invariant to distortions (Fig. 3(a)(b)).

In general, 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. Then, we blend each pixel in the virtual view as : $\mathbf{p}_v = f(\omega_i)\mathbf{p}_i + g(\omega_i)\mathbf{p}_{i+1} + \omega_{bkg}\mathbf{p}_{bkg}$, where $\omega_{bkg} = 1 - (f(\omega_i) + g(\omega_i))$, $\mathbf{p}_{bkg} = \omega_i\mathbf{p}_i^{bkg} + \omega_{i+1}\mathbf{p}_{i+1}^{bkg}$, and $\mathbf{p}_i^{bkg}$ and $\mathbf{p}_{i+1}^{bkg}$ are the pixels of the subtracted background computed sperately from $\hat{\mathbf{f}}_i$ and $\hat{\mathbf{f}}_{i+1}$ respectively.

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 transitions to the back-projected view of the closest viewpoint. In Section 4.3 we introduce an approach to visualize moving objects when multiple videos are not overlapped and each camera is distributed sparsely.

4.3 Sparse Cameras with Simple Motion: Traffic

We demonstrate this scenario as the case of analyzing videos of traffic and synthesizing traffic movements dynamically on AEMs. 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. To deal with this problem, we



(a) (b) (c)

Figure 4: 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. (b) split: outgoing regions $\mathbf{O}(\mathbf{X_i})$ from node X_i are marked. (c) merging: incoming regions $\mathbf{I}(\mathbf{X_i})$ to node X_i are marked.

need to model the movements in each view and connect the observations between cameras, *i*.e., to model flows from one view to another. In our framework, we add two functionalities. (1) Modeling the flow using a graph-based representation to infer a plausible traffic-flow across unobservable regions. (2) Develop a synthesis framework that can be driven from such data.

Fig. 4(a) 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 cameras. From visual tracking [1, 13] , we get estimates of position and velocities of objects from video. The measurement Y_i consists of the positions and the velocities of the low-level features $\mathbf{f}_i$ and the detected cars $\mathbf{r}_i$.

The state X_i is designed to capture the essential information necessary to visualize traffic flow: (1) the average 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.

Then, the entire traffic system is defined 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 *i*-th node is obtained from the available geospatial 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 other hand, the *unobservable* regions are denoted by $\tilde{\mathbf{M}}(\mathbf{X})$ where $\mathbf{X} = \mathbf{M}(\mathbf{X}) \cup \tilde{\mathbf{M}}(\mathbf{X})$.

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 $\mathbf{f}_i$ w.r.t. the homography $\mathbf{H}_i$. Secondly, the flow of the cars passing through the *i*th 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., $N_{\mathbf{r}_i} = \hat{n}_i \cdot (d_i/\hat{v}_i)$.

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 [6] 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. 4). 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 [6, 3].

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 : $X_i|_n = \sum_{\mathbf{I}(X_i)} X_j|_n + w_i$. 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 : $X_i|_v = (\sum_{\mathbf{I}(X_i)} X_j|_v)/N_{\mathbf{I}(X_j)} + q_i$. 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*.

Finally, to visualize traffic flow based on the estimated traffic states $\hat{\mathbf{X}}$, we developed a parameterized version of Reynolds' behavior simulation approach [8]. 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 conditions. The controller of a car in the *i*-th region is parameterized by X_i. Hence, the behavior of a car Z_i varies if the estimated traffic condition within a region changes or if the car moves onto other adjacent traffic regions based on given X_i.

4.4 Sparse Cameras with Complex Motion: Clouds

Our final scenario aims to use videos of natural phenomenon, primarily clouds and adding them to AEMs for an additional sense of reality. 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. For measuring dynamic movement of clouds, we also extract velocities from videos. We assume that the videos always have to look at 90 degree of elevation, and zero degree azimuth. We refer to this video as an *anchor video*.

We use a Radial Basis Function(RBF)([2]) 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 [14] where they interpolate implicit surface from given set of scattered data points. We use this method to interpolate density of unobservable region in the sky. In our work, constraint points are the location of feature points(x_i, y_i) extracted from each input video, and 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 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.

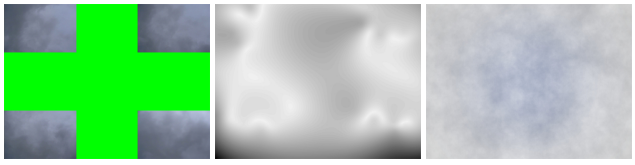
Now, the generated density map is used as a density function for procedural modeling of cloud textures([7]). Figure 5(a)(b) shows an example of density map generated from four videos. However, if the generated density map is not appropriate due to mis-detection of feature vectors, the system provides an user interface to edit the density map by adding additional basis vectors.

Once cloud textures are generated, the generated sky textures are mapped onto the sky domes [17]. To visualize sky, representing a dynamic of current sky, the mapped sky textures moves based on the velocity captured from anchor video.

5 Discussion of Experiments and Results

To validate our approach, we undertook a series of experiments for different scenarios on a variety of challenging domains, under varying conditions. Fig. 6 shows results of the static stills of each scenarios. Please see our website for videos and other results.

The prototype system is developed using C++/OpenGL on a computer with Quad-core 2.5GHz, 2GB RAM, and NVidia Quadro



(a) (b) (c)

Figure 5: Generating clouds layers procedurally using videos: (a) videos (b) Interpolated map from RBF (c) resulting cloud layer.

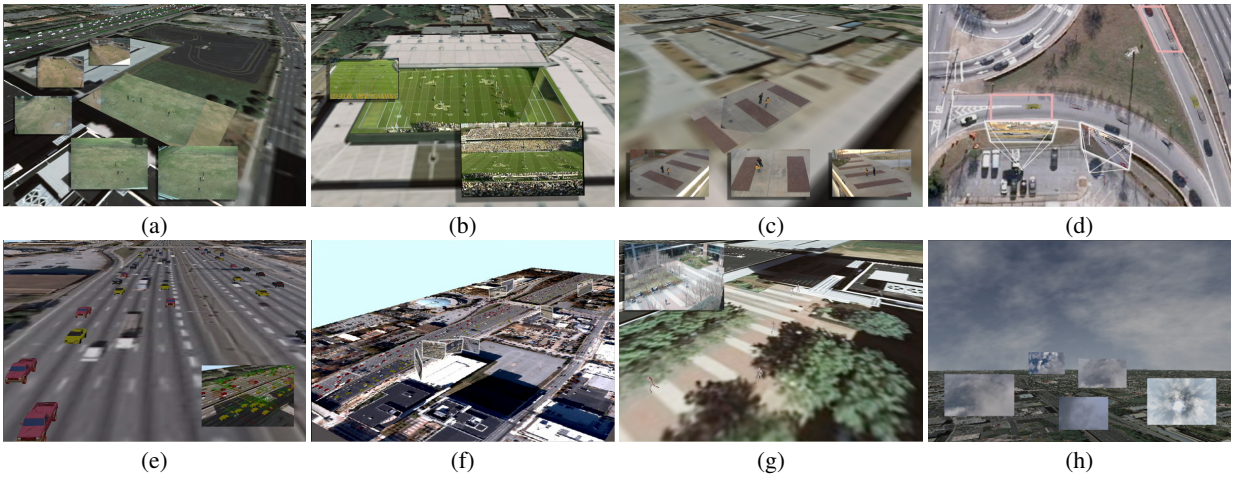


Figure 6: Results from our prototype system using 36 videos : (1) View Blending : (a) 5 Cameras for soccer game (b) Two broadcasting footages of NCAA Football game (c) Three surveillance camera. (2) Traffic : (d) Merging Lanes (e) Rendered traffic Scene and corresponding simulated scene (f) 8 cameras for larger scale traffic simulation including merge and split (3) Pedestrians : (g) Direct mapping of pedestrian having simple motion (4) Clouds : (h)Four videos for clouds and sky generation.

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.

For the scenario that require view blending, the resulting view transitions are smooth and provide dynamic visualizations (Fig. 6(a,b,c,)). In the traffic scenario, the visualized traffic closely matches the average speeds of the real traffic system (Fig. 6(d,e)). 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.

In the challenging large-scale experiments shown in Fig. 6(f), 8 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.

Various styles of sky and clouds are generated as shown in Fig. 6(h). 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.

6 Summary, Limitations, and Future Work

In this paper, we introduced methods for Augmented Aerial Earth Maps (AAEMs) with diverse types of real-time live information, namely pedestrians, sports scenes, traffic flows and sky. 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, vivid, and more engaging virtual environment through which the users would browse the cities of now.

It is also important to note a few of our limitations. First, 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. Secondly, our solutions do not support the automatic retrieval of high-level semantic information, *e*.g., car accidents or street burglaries.

In our future work, we aim to overcome the above limitations, and incorporate even more types of additional dynamic information such as flowing rivers, trees with wind, sun, weather patterns, environmental condition and even aerial objects like birds and airplanes.

Acknowledgements: We would like to thank the Georgia Tech Athletic Association for providing us with the videos of the football game. Thanks also to the reviewers for their insightful comments.

References

[1] J.-Y. Bouguet. Pyramidal implementation of the lucas kanade feature tracker. In *Intel Corporation*, 2003.

[2] M. D. Buhmann and M. J. Ablowitz. *Radial Basis Functions : Theory and Implementations.* Cambridge University., 2003.

[3] B. Frey and D. MacKay. A revolution: Belief propagation in graphs with cycles. In *NIPS'98*, 1998.

[4] M. J. Harris. Real-time cloud simulation and rendering. In *SIGGRAPH '05 Courses*, page 222. ACM, 2005.

[5] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge Univ. Press, 2000.

[6] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[7] K. Perlin. An image synthesizer. *SIGGRAPH Comput. Graph.*, 19(3):287–296, 1985.

[8] C. W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *SIGGRAPH'87*, pages 25–34. ACM Press, 1987.

[9] C. W. Reynolds. Steering behaviors for autonomous characters. In *GDC '99*, pages 768–782, 1999.

[10] H. S. Sawhney, A. Arpa, R. Kumar, S. Samarasekera, M. Aggarwal, S. Hsu, D. Nister, and K. Hanna. Video flashlights. In *13th Eurographics workshop on Rendering*, pages 157–168, 2002.

[11] I. O. Sebe, J. Hu, S. You, and U. Neumann. 3d video surveillance with augmented virtual environments. In *IWVS '03*, pages 107–112, 2003.

[12] S. M. Seitz and C. R. Dyer. View morphing. In *SIGGRAPH '96*, pages 21–30, 1996.

[13] J. Shi and C. Tomasi. Good features to track. In *Proceedings of IEEE CVPR*, pages 593–600. IEEE Computer Society, 1994.

[14] G. Turk and J. F. O'Brien. Shape transformation using variational implicit functions. In *SIGGRAPH '99*, pages 335–342, 1999.

[15] WWW. "cmu motion capture database http://mocap.cs.cmu.edu/".

[16] A. Yilmaz, O. Javed, and M. Shah. Object tracking: A survey. *ACM Comput. Surv.*, 38(4):13, 2006.

[17] G. Zotti and M. E. Groller. A sky dome visualisation for identification of astronomical orientations. In *INFOVIS '05*, pages 2–10, 2005.