Software Laboratory for Camera Networks Research

Wiktor Starzyk† and Faisal Z. Qureshi‡, *Member, IEEE*

***Abstract*—We present a distributed virtual vision simulator capable of simulating large-scale camera networks. Our virtual vision simulator is capable of simulating pedestrian traffic in different 3D environments. Simulated cameras deployed in these virtual environments generate synthetic video feeds that are fed into a vision processing pipeline supporting pedestrian detection and tracking. The visual analysis results are then used for subsequent processing, such as camera control, coordination, and handoff. Our virtual vision simulator is realized as a collection of modules that communicate with each other over the network. Consequently, we can deploy our simulator over a network of computers, allowing us to simulate much larger camera networks and much more complex scenes then is otherwise possible. Specifically, we show that our proposed virtual vision simulator can model a camera network, comprising more than one hundred active pan/tilt/zoom and passive wide field-of-view cameras, deployed in an upper floor of an office tower in downtown Toronto.**

***Index Terms*—virtual vision, camera networks, virtual reality**

Fig. 1: A view of our virtual world showing pedestrians walking on an upper floor of an office building. Toronto (Canada) skyline is visible through floor to ceiling panoramic windows. Our scripted pedestrians use motion-capture data to simulate realistic motion and cast dynamic shadows on the floor and the walls.

I. Introduction

We envision that the next generation large scale smart cameras networks will be able to gather imagery over large areas, analyze this imagery, store it, and later retrieve it to support a multitude of applications ranging from surveillance and security to infrastructure management and entertainment with no or minimal human intervention. The sheer scale of these networks will render human intervention infeasible. Studying and designing such camera networks is a daunting task as problems in such disparate fields as image processing, embedded systems and networking must be addressed concomitantly just to setup the basic experimental infrastructure. This observation led us to develop the virtual vision paradigm of camera networks research [1]. Virtual vision advocates using software tools capable of simulating camera networks under realistic scenarios to study and develop new camera networks algorithms.

Virtual vision paradigm for computer vision research advocates using visually and behaviourally realistic 3D environments, populated with pedestrians, automobiles, etc. to study camera networks. Virtual camera networks of suitable complexity can be simulated within these synthetic environments, which we call *virtual vision simulators*. A virtual vision simulator offers several advantages over traditional physical camera setups during ideation, prototyping, and evaluation phases, including:

- legal issues surrounding access to physical camera networks installed in public spaces disappear when dealing with a simulated camera network;
- developing a virtual vision simulator is a major undertaking; however, once such a simulator becomes available, the cost of carrying out camera networks research within this simulator is minimal compared to performing research on a physical camera network—a virtual vision simulator runs on standard PCs and does not require any special hardware;
- virtual vision offers quick prototyping—it is much easier and faster to reconfigure a virtual camera network than it is to reconfigure a physical camera network;
- complex vision and control algorithms that need to be studied in "real time" can be easily studied in a virtual vision simulator by slowing down the virtual clock of the simulated environment;
- virtual vision offers far faster design/evaluation iterations when compared to a physical camera network;
- ground truth is readily available; and
- camera control and coordination algorithms can easily be compared against each other since scenes are perfectly repeatable.

Qureshi and Terzopoulos demonstrated a first virtual vision simulator of its kind in [1]. They deployed a network of 16 active pan/tilt/zoom (PTZ) cameras in a 3D reconstruction of the Penn train station. Here the Penn station was inhabited by up to 1000 self-animating pedestrians—tourists, commuters, etc.—that look and behave like real humans [2]. They used this simulator to study camera scheduling and coordination

W. Starzyk and F.Z. Qureshi are with the Faculty of Science, University of Ontario Institute of Technology, Oshawa, ON, L1H 7K4 Canada.
† wiktor.starzyk@uoit.ca
‡ http://faculty.uoit.ca/qureshi

Manuscript received X; revised X.

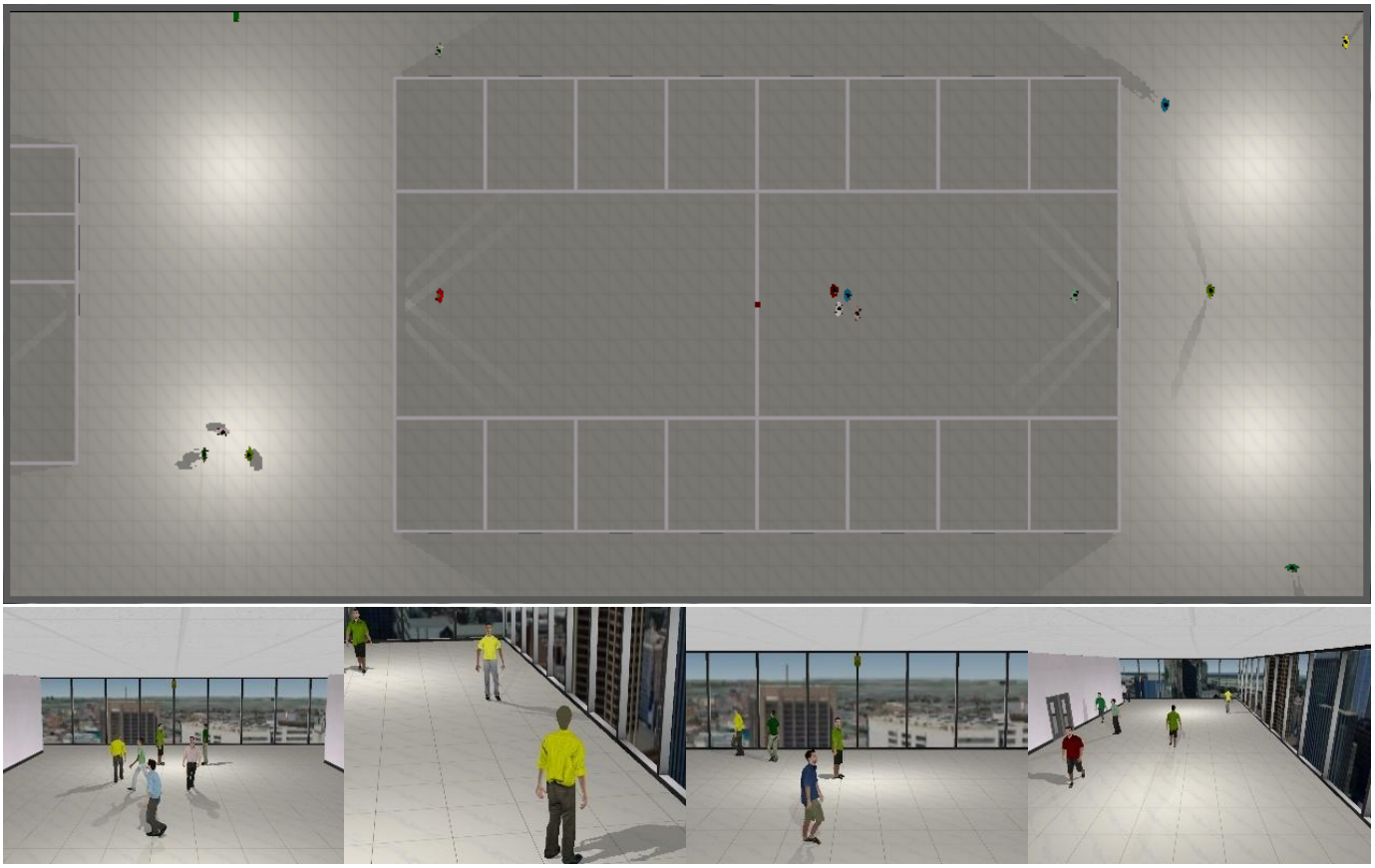


Fig. 2: VW consists of the top floor of an office building situated in downtown Toronto. Toronto skyline is visible through floor-to-ceiling panoramic windows. Sunlight filters through the windows and casts dynamic shadows. (Top-Row) Top-down view of the office floor, showing individual offices, conference rooms, common areas, and elevator lobbies. (Bottom-Row) Synthetic imagery captured by 4 different cameras. Notice the subtle lighting and shadow effects, which were missing in the virtual vision simulator developed by Shao and Terzopoulos [1].

problems that arise in networks of active PTZ cameras. The primary drawback of their virtual vision simulator is that it is tied to a single machine. Therefore, it does not scale. The largest camera network that they have studied in their work comprises 16 cameras.

Motivated by the promise of the virtual vision paradigm for camera networks research, here we present a distributed virtual vision simulator.[1] Our simulator is designed to address some of the shortcomings of the virtual vision simulator employed by Qureshi and Terzopoulos in their work on smart camera networks. Specifically our simulator can be deployed over a network of computers; where as, there's is tied to a single machine. The ability to spread the computational load over multiple computers assumes urgency as we begin to simulate richer, more complex synthetic worlds and larger, more sophisticated virtual camera networks. Later in the paper we demonstrate our virtual vision simulator simulating a network of more than 100 cameras. Additionally the virtual vision simulator presented herein also supports advanced rendering effects missing from the Penn train station simulator, including lighting, shadows, and transparency, which adds to visual realism of the scene (Fig. 1).

[1]A preliminary version of this work has appeared in [3].

A. Contributions and Overview

The contributions of the research presented here are three-fold: First, we present a distributed virtual vision simulator. The proposed simulator is highly scalable and can simulate networks comprising more than 100 cameras. Scalability is achieved by spreading the computational load over multiple computers. Previous virtual world simulators are constrained to a single machine; therefore, are unable to simulate large networks of cameras. Second, we demonstrate the suitability of this simulator for camera networks research by providing examples of some recent work on smart camera networks that was carried out within this simulator. Lastly, we empirically evaluate our virtual vision simulator. We study the scalability properties of our simulator and focus on how performance of the simulator is related to the size of the camera network being simulated and the computational resources available to simulate this camera network. This study may suggest best practices when using our simulator for camera networks research.

The rest of the paper is organized as follows. We briefly discuss the related work in the next section. The following section provides an overview of our virtual vision simulator. Sec. IV presents the virtual world engine. We introduce the

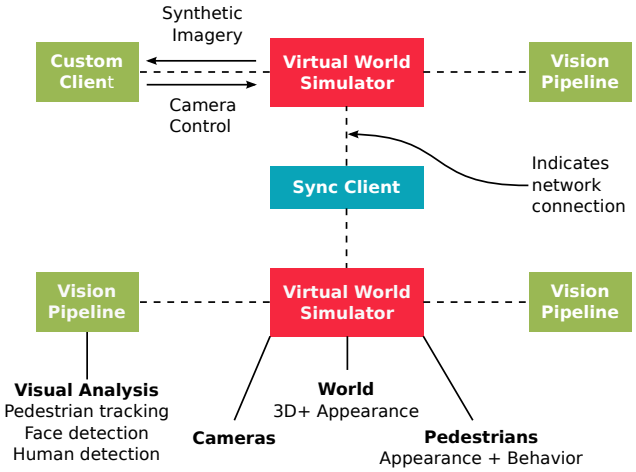


Fig. 3: An overview of our distributed virtual vision simulator.

(a) Intersection (b) City scene

Fig. 4: For the purposes of this paper we have deployed a simulated camera network in an office environment. It is however straightforward to use other 3D environments. It is therefore possible to simulate camera networks in a wide-variety of scenarios.

visual analysis pipeline in the following section. Sec. VI discusses the synchronization unit and the network operation of the virtual vision simulator. We present three camera networks research projects that used our simulator in Sec. VII. We provide experimentation results in Sec. VIII and Sec. IX concludes the paper with a brief discussion.

II. Related Work

In 1995, Terzopoulos and Rabie introduced a software based approach to designing active vision systems called animat vision [4]. The animat approach replaces physical robots and cameras with artificial animals referred to as animats [5]. The animats are placed in physics-based virtual worlds to study active vision systems. Eschewing the need for real cameras, robots, and other hardware, at least during the early stages of research, this approach promises huge savings in time and money.

In 2003, Terzopoulos proposed using reality emulators for computer vision research [6]. Santuari *et al.* developed a virtual museum simulator, populated with scripted visitors, to study computer vision algorithms [7]. This simulator was aimed at developing low level pedestrian segmentation and tracking algorithms. The virtual museum simulator uses sophisticated 3D rendering techniques with support for global illumination, shadows and different visual artifacts such as motion blur and interlacing.

In 2005, Shao and Terzopoulos developed a train station simulator, populated with self-animating pedestrians (commuters and visitors) [2]. Qureshi and Terzopoulos used this train station simulator to develop the first of its kind virtual vision simulator [1]. They demonstrated their virtual vision simulator by studying high-level camera control and coordination problems in camera networks comprising both passive and active cameras. The virtual vision simulator presented in this paper addresses many of the shortcomings of the virtual vision simulator developed by Qureshi and Terzopoulos in [1]. Our simulator has superior synthetic imagery quality; it supports subtle lighting effects and dynamic shadows. But more importantly the simulator developed here is distributed. Consequently, it is capable of simulating much larger camera networks and far richer synthetic environments. Lastly, our simulator only uses open source libraries.

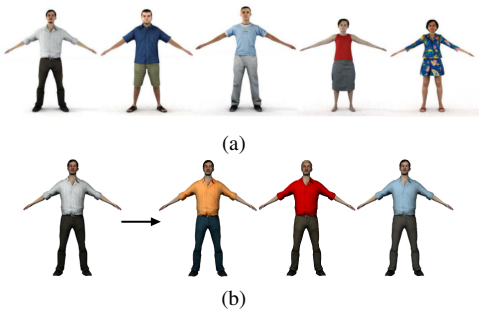
In view of the primary thrust of this paper, we refer the kind reader elsewhere for background literature related to pan/tilt/zoom (PTZ) camera networks and high-level control and coordination problems in smart camera networks.

III. System Overview

Our virtual vision simulator consists of three types of modules: 1) virtual world engine (VW), 2) visual analysis pipeline (VP) and 3) synchronization unit (SYNC). In a typical realization of the virtual vision simulator, one or more instances of VW and VP modules are spread over a network of computers (see Fig. 3). These modules communicate with each other over the network. VP modules analyze images captured by the cameras simulated in VWs and use the results of this analysis to control and coordinate these cameras. Typically each camera has its own dedicated VP module. A single instance of a SYNC module ensures that all VW modules are in sync with each other. The next three sections describe each of the three components of our virtual vision simulator. SYNC module is only necessary when we are simulating a camera network where each camera is synchronized to a central clock. It is best not to use the SYNC module in situations where camera synchronization is not desired, as SYNC comes with a performance penalty.

IV. Virtual World Engine

VW is responsible for simulating virtual 3D scenes, inhabited with self-animating pedestrians, automobiles, etc. It consists of a 3D model of the environment being simulated and the appearance and behaviour models for pedestrians inhabiting this environment. Specifically our VW models the top floor of an office tower in downtown Toronto, populated with virtual humans who work on this floor (Fig. 2). It is relatively straightforward to change the 3D environment. All that is needed is a 3D model of the scene that we wish to simulate. Such 3D models can be constructed by hand using a 3D modeling package or can be purchased from a digital assets retailer (Fig. 4). Virtual humans use motion capture data and scripted way-points for locomotion and exhibit highly-realistic movements. Virtual cameras deployed in this environment can



(a)

(b)

Fig. 5: Starting with just five 3D human models (a), we are able to generate many more pedestrians with unique appearances by modifying the appearance textures (b).

generate synthetic imagery mimicking video captured by a typical surveillance system. Currently VW supports passive, wide Field-of-View (FOV) and active Pan/Tilt/Zoom (PTZ) cameras. These cameras can be easily configured and placed anywhere in the virtual environment to prototype a camera network having the desired configuration. Imaging artifacts—such as camera color response, compression artifacts, etc.—can also be easily introduced into the captured image stream. The office floor is complete with floor-to-ceiling panoramic windows looking out at the Toronto skyline. VW supports dynamic shadows and subtle lighting effects, such as sunlight filtering through glass windows. Such lighting effects are missing in Qureshi and Terzopoulos' virtual vision simulator.

A. Virtual Humans

The 3D office floor is inhabited by self-animating virtual pedestrians. The pedestrians exhibit life-like movements (walk, turn, run, stop, etc.) by relying upon motion capture data. These pedestrians move around the environment following scripted paths, defined as a series of motor actions. Our approach provides the user with full control over where each pedestrian is at any given time, while at the same time keeps script complexity manageable. Each pedestrian maintains a queue of these actions and executes them in order.

Currently we have 3D models for three males and two females; these are shown in Fig. 5(a). We are able to generate many more humans with unique appearances by modifying the textures available for these 5 models (Fig. 5(b)). This allows us to realize scenes with far more individuals then is otherwise possible. It is straightforward to incorporate more human models into the system. Such human models are available for purchase from many firms dealing with digital assets for computer games.

1) Locomotion: Each virtual human is described by a 3D model, texture images representing clothes and animation data. The animation data consists of a number of motion-capture sequences, such as walk, turn left, run, etc. In order to generate realistic looking motions we define a motion graph over the motion capture sequences available for each human (Fig. 6) [8]. Motion capture sequences currently available for each human are: 1) walk, 2) run, 3) stand idle, 4) about turn, 5) turn left 90 degrees, 6) turn right 90 degrees, 7) turn left 45 degrees, 8) turn right 45 degrees, 9) start walking from idle, 10) come to a stop, 11) start running from a walking gait, and 11) run to walk. A typical motion script for a pedestrian might look like: `walk left90 stand walk walk 180`.

B. Virtual Cameras

Our system currently supports passive wide-FOV cameras and active PTZ cameras. Clients can connect to these cameras over the network. In that respect our simulated passive and active cameras behave similarly to typical IP surveillance cameras. Each camera exposes a control interface through which a client can control the camera and request images from this camera. Table. I lists the commands that are available to access, configure, and control these cameras over the network. Furthermore each camera is uniquely addressable, consequently it is possible to open a communication/control channel to any camera situated in the virtual world.

Camera Type	Command	Description
PTZ & wide-FOV	setResolution	Set the resolution of the image
	getImage	Get an image from the camera
PTZ	panLeft	Pan left by θ degrees
	panRight	Pan right by θ degrees
	tiltUp	Tilt up by θ degrees
	tiltDown	Tilt down by θ degrees
	zoomIn	Zoom in by θ degrees
	zoomOut	Zoom out by θ degrees
	default	Reverts the camera to its default settings

TABLE I: Simulated passive wide-FOV and active PTZ cameras support a set of commands similar to those available in a typical IP camera.

We have developed a high-throughput, light-weight protocol for accessing these simulated cameras. Our protocol can be easily mapped to communication protocols used by physical IP cameras, such as the Pelco-D standard [9]. It suggests that the camera networks developed within the virtual environment can be easily ported to a physical camera network.

V. Visual Analysis Pipeline

We have developed pedestrian detection and tracking routines that can be put together to construct a visual analysis pipeline, capable of tracking individual pedestrians in videos captured via passive wide-FOV and active PTZ cameras. Currently we have implemented two pipelines: 1) a pedestrian tracker capable of tracking multiple pedestrians in video feeds from passive wide-FOV cameras and 2) a pedestrian tracker capable of tracking one or more "selected" pedestrians in video feeds from active PTZ cameras (Fig. 7). These pipelines serve our purpose well, since we are currently only interested in camera networks comprising passive wide-FOV and active PTZ cameras deployed in urban settings for observing pedestrians in the scene. It is important to note that our visual analysis pipelines work equally well for synthetic

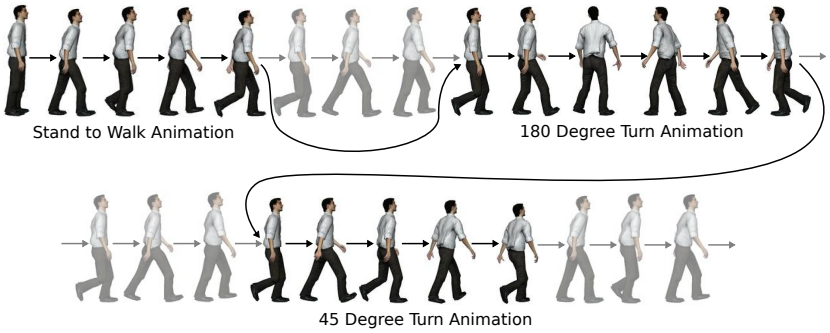
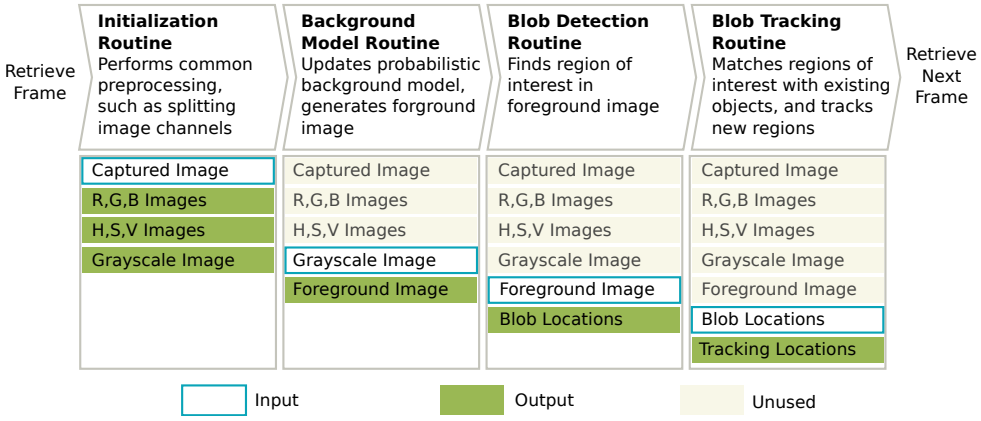
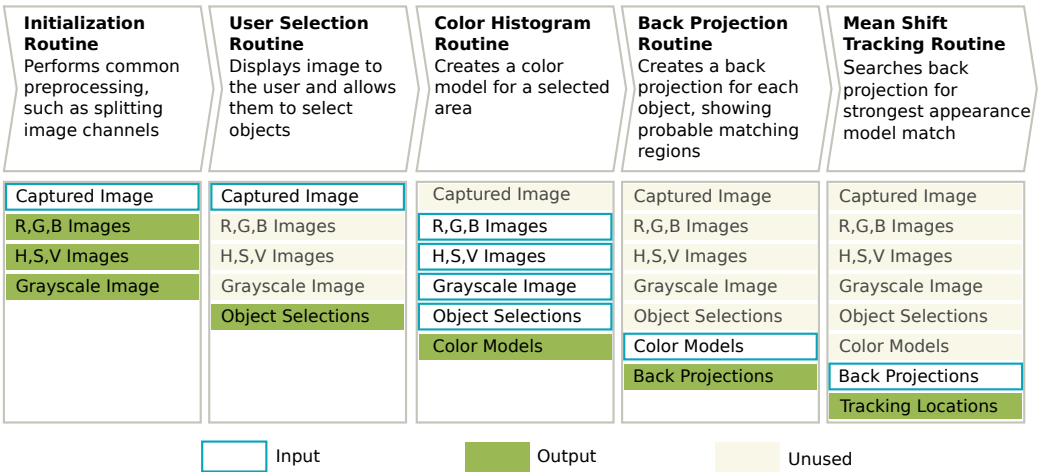


Fig. 6: Transition between different locomotion states (walk to run, etc.) is accomplished by defining a motion graph over motion capture data corresponding to different locomotion states. The motion graph minimizes the jumps when transitioning from one state to the other.



(a) Visual analysis pipeline for tracking pedestrians in wide-FOV cameras.



(b) Visual analysis pipeline for tracking pedestrians in PTZ cameras.

Fig. 7: Visual analysis pipelines are realized as a collection of reusable vision routines.

imagery captured by virtual cameras and real footage captured using physical cameras (Fig. 8). This will make it easier to port camera networks implemented within our virtual vision simulator to real, physical camera networks.

Each visual analysis pipeline is realized within a visual analysis module that can acquire imagery over the network and send back visual analysis results. We use this module to provide visual analysis for our synthetic cameras situated in our virtual vision simulator. A visual analysis module is instantiated for each simulated camera. The visual analysis module processes the video stream from the camera and sends back control commands to the camera (capture an image, select resolution, pan, tilt, zoom, etc.). Since each simulated camera is uniquely addressable, the visual analysis module connected to a particular camera is able to open a message-passing connection to any other camera in the system. The proposed virtual vision simulator; therefore, supports "smart" camera networks lacking a central server.

Fig. 8: Our visual analysis pipeline is designed from the ground up to work with both synthetic (right) and real video (left) without any modifications. Consequently, our vision pipeline faithfully mimics the performance of a vision pipeline implemented on physical cameras.

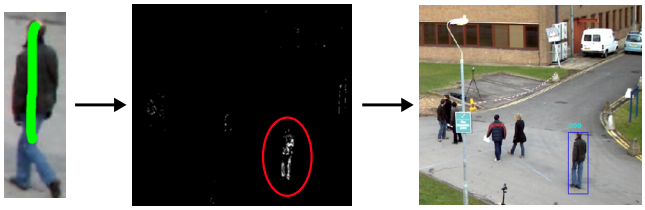


Fig. 9: A stroke gesture is provided to select a pedestrian to be tracked in active PTZ cameras. Appearance signatures computed by passive wide-FOV cameras can also be used to track individuals in active PTZ cameras.

A. Construction

Visual analysis pipelines are realized as a set of vision routines (Fig. 7). Whenever a new image arrives, a frame object is constructed. Every vision routine in the pipeline operates upon this frame object in sequence, modifying the frame object in the process. Vision routines that come after can access results computed by previous routines through this frame object. Consequently, the frame object provides a communication and coordination mechanism between the various vision routines that constitute a visual analysis pipeline. E.g., a vision routine can dynamically check if it needs to run a particular algorithm. This can lead to efficiencies and more intelligent management of resources.

So far we have implemented the following set of computer vision routines:

- Routines that produce appearance models of objects, and routines that use these appearance models to construct the back-projection image;
- Routines for foreground detection, using state-of-the-art background subtraction algorithms;
- Routines for blob detection and tracking; and
- Routines for face and head detection.

We briefly describe the vision pipeline below. Implementation details are not topical to this paper since one can as easily implement a different set of vision algorithms within the visual analysis module.

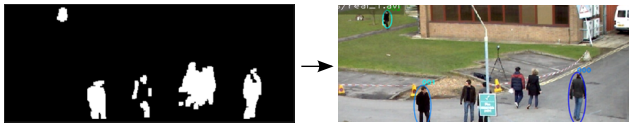


Fig. 10: Background Subtraction

B. Pedestrian Tracking

The pedestrian tracker for passive wide-FOV cameras relies upon background subtraction to detect and subsequently track pedestrians. This tracker works completely autonomously and does not require any human assistance. After a few minutes of training the tracker is able to pick out and track pedestrians present in the scene. It also constructs appearance based signatures of the pedestrians being tracked. Background subtraction, however, is not available for active PTZ cameras. The pedestrian tracker developed for active PTZ cameras, therefore, employs an appearance based signature to track one or more pedestrians. Appearance signatures of pedestrians that need to be tracked are available from the wide-FOV cameras. We have also developed a stroke-based interaction mechanism that allows a user to quickly identify (select) the person to be tracked in the video feed from an active PTZ camera (Fig. 9).

Our visual analysis pipelines are able to track individual pedestrians reasonably well in low density environments, such as office buildings. These will not work in situations involving a large number of pedestrians crammed together in a small physical space, such as a subway station during rush hours. Nevertheless, these visual analysis pipelines enable us to study camera control and coordination strategies in our virtual vision simulator.

VI. Synchronization Unit

Fig. 3 depicts a possible configuration of our virtual vision simulator. Each VW is responsible for simulating the visually and behaviourally realistic 3D environments within which virtual cameras are deployed. VP modules are responsible for analyzing video data captured by simulated cameras. VPs are also able to control the simulated cameras. For example, a VP can choose to pan a PTZ camera to follow an individual of interest. Additionally, VPs are also able to communicate with each other (via the VW or SYNC); e.g., during camera handoffs, etc. VWs and VPs are spread over multiple computers and we use a SYNC unit to ensure that all VWs evolve in lockstep. This ensures that images captured by various cameras and commands issued to these cameras are perfectly synchronized across the whole (simulated) camera network no matter the speed of the computer hosting a particular VW or VP. Of course it is straightforward to simulate asynchronous image capture and processing.

Fig. 11 illustrates how the SYNC unit communicates with multiple VW modules. SYNC unit begins by registering every VW in its database. Next, SYNC unit, which keeps track of the global clock, sends step signals to each VW unit to take a fixed number, say n, of simulation steps. The next step is only issued once every VW has successfully completed n simulation steps. This ensures that the internal clocks of any two VWs are never

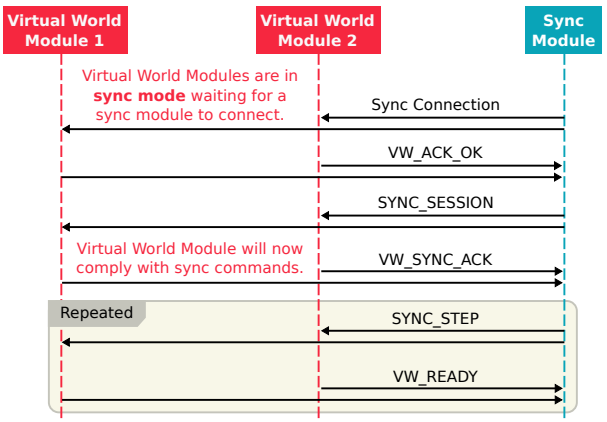


Fig. 11: Communication between VWs and SYNC unit.

more than n simulation steps apart. VWs update their clocks upon receiving the sync message.

To create a smart camera network, it is critical that the cameras be able to communicate with each other. There are two ways this can be achieved using our system. When both the sender and the recipient cameras reside on the same VW, the message can be transferred without going through the network interface. However, if the recipient camera is being simulated on a different VW, the message is sent to the SYNC unit which can then pass the message on to the appropriate VW. Alternately our system allows anyone to implement their own method of communication that bypasses VW and SYNC entirely. Aforementioned fact that each camera is uniquely identified helps us setup message passing communication between any two cameras, whether or not these cameras are being simulated on the same physical computer.[2]

VII. Examples of Camera Networks

Here we briefly describe three prototype camera networks that were deployed and studied within our virtual vision system.

A. Multi-tasking PTZ Cameras

The first project focuses on developing a system that automatically tunes the sensing parameters of PTZ cameras in response to the scene activity, choosing to capture close-up video when the number of pedestrians present in the scene is low and electing to capture lower-resolution video as the number of pedestrians increase, thus always keeping every pedestrian in view [10]. These cameras enable the video surveillance system to intelligently respond to scene complexity, automatically capturing close-up imagery of the pedestrians present in the scene when possible, and behaving as wide-FOV cameras when the number of pedestrians increases. Fig. 12 shows a multi-tasking PTZ camera: the PTZ camera is able to capture higher resolution video of the pedestrians present in the scene when there are only a few pedestrians present; however, it begins to behave like a wide-FOV camera as the number of pedestrians present in the scene spread out and move away from the camera.

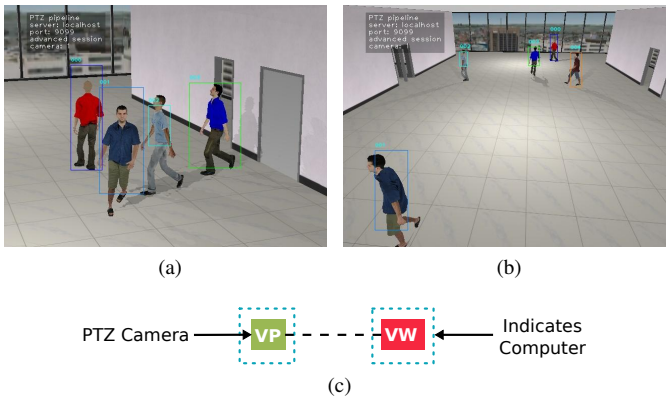


Fig. 12: PTZ cameras automatically decide how best to observe a scene. (a) When possible, the PTZ camera selects a higher zoom to capture higher resolution images of the individuals present in the scene. (b) As the individuals spread out and move away from the camera, the PTZ camera selects a lower zoom setting to keep everybody in view, albeit at a much lower resolution. (c) Virtual vision simulator consisted of one VW and one VP module, spread over two computers.

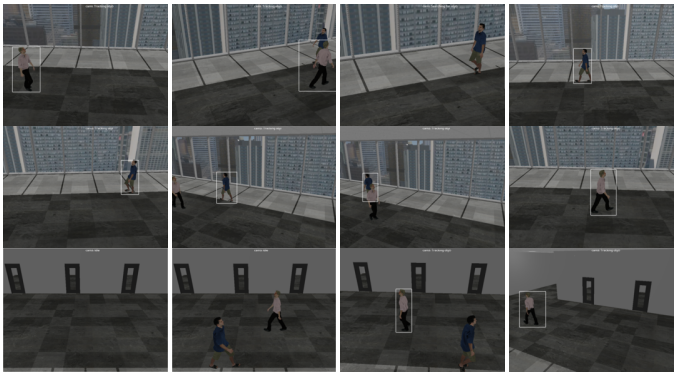
B. Learning Proactive Control Strategies

The second project focuses on developing a PTZ camera network that learns proactive control strategies by generalizing and storing the results of a reasoning process (Fig. 13). The reasoning process—which considers both immediate and far-reaching consequences of different camera assignments when constructing a "plan" most likely to succeed (in a probabilistic sense) at the current observation task(s)—is capable of performing camera assignments and handoffs in order to provide persistent coverage of a region. The results of this reasoning activity are then stored as rules in a production system [11]. Later when a similar situation is encountered, the production system bypasses the reasoning process and performs camera assignments. Initially the camera network relies mostly on the reasoning process, over time however, the camera assignments become instinctive.

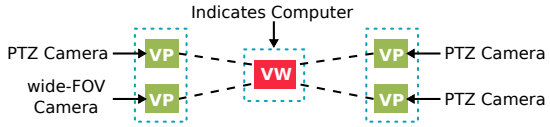
C. Camera Handoffs

Fig. 14 illustrates camera handoff between a passive wide-FOV camera and a nearby active PTZ camera. In this case, a passive camera detected the individual through background subtraction and began tracking it. Once it detected that the individual is about to leave its field-of-view, the passive camera sent the appearance signature for this individual to a nearby active camera. The active PTZ camera then is able to track the individual using the appearance signature sent to it by the passive camera.

[2]As we shall see in the following sections inter-camera communication makes it possible to transfer pedestrian appearance signature between cameras to initiate handoffs.

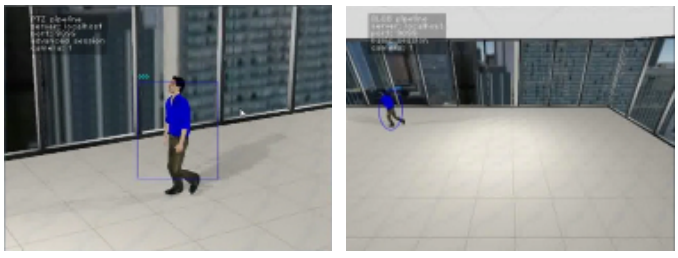


(a)



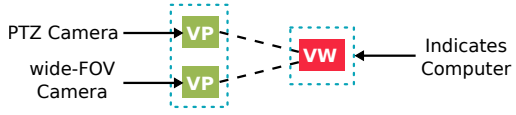
(b)

Fig. 13: (a) The three rows show three cameras observing two pedestrians as they cross each other on their way to the opposite sides of the lobby. The three cameras are able to perform a handoff while keeping both pedestrians in view. This is achieved through a reasoning mechanism that considers both short-term and long-term consequences of camera assignments. (b) Virtual vision simulator consisted of one VW and four VP modules, spread over three computers.



Active PTZ Camera Passive Wide-FOV Camera

(a)



(b)

Fig. 14: A passive wide-FOV camera hands off a pedestrian to a PTZ camera by sending it the pedestrians appearance signature. (b) Virtual vision simulator consisted of one VW and two VP modules, spread over two computers.

VIII. Evaluation & Results

We evaluated our virtual vision simulator by simulating camera networks with a wide range of cameras spanning different numbers of virtual world engines. Our experiments were performed on mid-range personal computers with the specifications shown in Table II. VW1-PC, VW2-PC and VW3-PC each ran a single virtual world engine along with a vision analysis pipeline for each camera being simulated by the VW on that machine. The VPs did not do any processing other than displaying the received images to the screen. We also ran Panda3D's [12] *PStats* server to collect statistics about each frame which were written to a text file. For all of our experiments, a SYNC module was running on SYNC-PC which was connected to the VWs via a network connection to control and synchronize their clocks. The simulation was configured to run at 30 frames per second by setting the time increment to 0.033.

	CPU	GPU	RAM
VW1-PC	AMD Phenom II X4 955 3.2 GHz	ATI Radeon 5750	8GB
VW2-PC	Intel Core i5 2380P 3.10GHz	Nvidia GeForce GTX 560Ti	8GB
VW3-PC	Intel Core i5 2300 2.8GHz	Nvidia GeForce GT 420	6GB
SYNC-PC	Intel Core2 Duo P8600 2.4GHz	ATI Mobility Radeon HD 3650	4GB

TABLE II: The computers that were used for our evaluation.

All of our experiments used the same scene and pedestrian configuration files. We used the office floor scene shown in Fig. 2, which was populated with 21 virtual pedestrians. These pedestrians enter and leave the scene at different times. Each spending a different amount of time wandering in the office.

A. Experiments

To evaluate our simulator, we experimented with different numbers of cameras spread over 1 to 3 VWs. For each experiment we recorded the physical (world clock) time required to simulate 5 minutes. The time was recorded by SYNC using the time function from Python's *Time* module.

1) One VW: In our first set of experiments, cameras were placed in a single virtual world module running on VW1-PC. The SYNC module was also executed for consistency.[3] We ran the simulation with 1, 5, 10 and 15 cameras. The times required to simulate 5 minutes are shown in Fig. 15

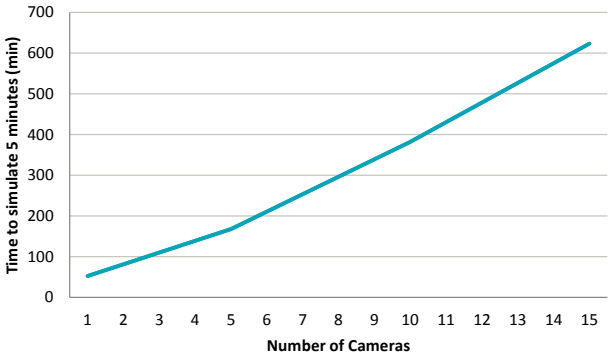


Fig. 15: The time required to simulate 5 minutes with different numbers of cameras.

For every camera that is added to the simulation, an extra frame has to be rendered at every time step. Rendering quickly

[3]The SYNC module is not needed when running a single VW engine.

becomes the bottleneck as shown in Fig. 16, which limits the number of cameras that can be simulated on a single machine. This is exactly the behaviour that we expected. It is for this reason a virtual vision simulator running on a single machine can not simulate large camera networks.

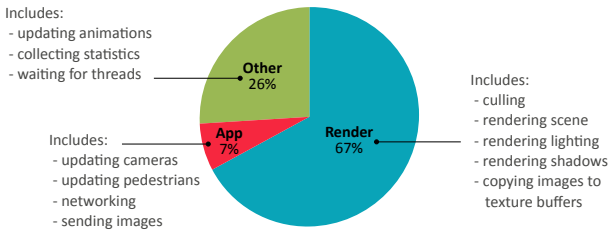


Fig. 16: Time breakdown. The majority of the time is spent on rendering.

2) Multiple VW's: To test the scalability of our simulator, we ran tests using 2 and 3 VWs. For the two VW scenario, we ran tests with 2, 6 and 10 cameras equally distributed between the two computers. For the 3 VW scenario, we ran simulations of 3, 9, 15 and 48 cameras. The results are shown in Fig. 17 and Fig. 18.

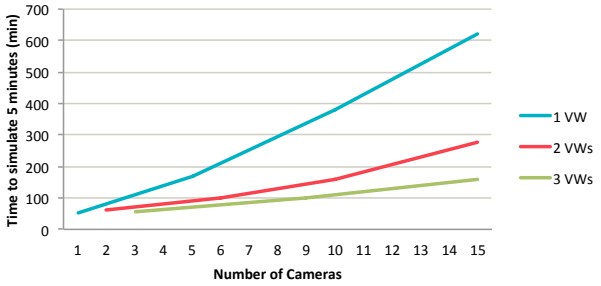


Fig. 17: Distributing cameras across multiple computers greatly improves performance and allows simulation of a large number of cameras.

There is a considerable performance increase by distributing cameras over multiple VWs. Fig. 18 shows that 3 VWs can simulate almost 3 times as many cameras as a single virtual world in the same amount of time. If not for a slower graphics card in one of the machines, the times would have been much closer. Because we use a light weight packet structure, using SYNC provides very little overhead as shown in Fig. 19.

3) A 100 Camera Network: The above scenarios show the benefit of distributing our simulator over a network of computers; however, the tests were run for a limited amount of time with a relatively small number of cameras. That is why we also ran a simulation for 1 hour of simulated time and broke the 100 camera barrier.

The 1 hour long simulation was executed using a single VW running on VW1-PC which was accompanied by SYNC. In this experiment, we simulated 5 cameras which took 32.3 hours to execute. For this simulation we also ran the PStats server and a VP for every camera.

To break the 100 camera barrier, we simulated a 102 camera network spread across 3 virtual world modules. Each VW

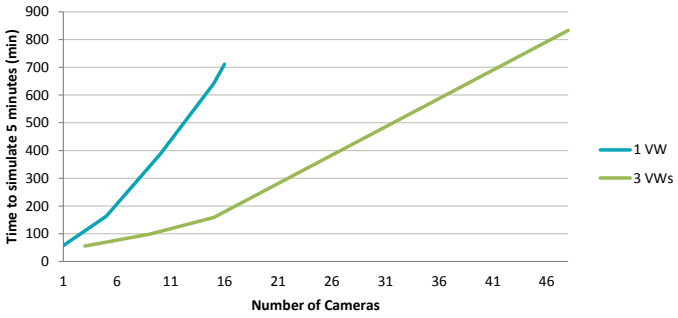


Fig. 18: Three virtual world modules can simulate almost 3 times as many cameras as a single virtual world can in the same amount of time.

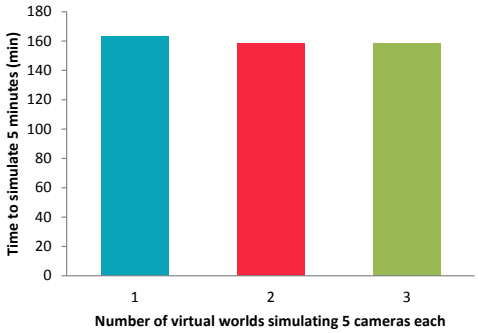


Fig. 19: Little, if any, performance degradation or time overhead is observed due to SYNC module. Distributing cameras over multiple computers does not decrease performance.

simulated 34 cameras in our office floor environment. Again, each VW PC also executed a PStats server and a VP for every camera. We ran the simulation for 5 minutes of simulated time and it took 43.9 hours to execute.

B. Discussion

All of our tests clearly show that distributing cameras over multiple computers provides a significant improvement in performance. By just using 2 virtual worlds modules instead of 1, you can double the number of cameras that you can simulate in the same amount of time. By distributing cameras over multiple computers, there is no limit to the size of the camera network that can be simulated.

Although the time required to simulate even 5 minutes is quite high, there are many ways to improve performance. For all of our tests, we ran VPs for each camera on the same machine as the VW. Distributing the VPs over multiple computers would provide a major performance improvement. We would also like to note that the performance of our virtual vision simulator can render up to 5 frames per second when simulating a single camera. It is possible to further bump up the framerate by turning off the camera control/communication circuitry. Rendering appears to the most costly operation in the proposed system. This suggests that we need to spread rendering load over multiple machines. The best case scenario is to simulate a single camera per graphics card. Short of that there will always be a performance hit as the graphics context switches between different cameras.

For all of our tests, we also ran Panda3D's PStats server which in some cases produced text files as big as a gigabyte in size. To see the impact of this, we ran a 1 camera simulation on VW1-PC for 5 minutes of simulated time with and without the PStats server running. We also disabled all networking, so no SYNC and VPs were connected. Without PStats running, the simulation took 890.33 seconds, while with PStats it took 1128.41 seconds. It suggests that collecting statistics using PStats adds a 20 percent overhead.

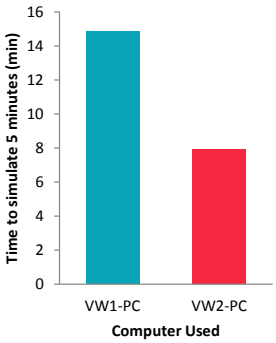


Fig. 20: Having a good graphics card can significantly decrease simulation time.

Our computers also did not use top of the line graphics cards. Because our simulation is built on top of a game engine, using a top of the line graphics card provides a big improvement in the framerate. This is evident in Fig. 20 where we ran a simulation of 1 camera on different machines with all networking disabled. VW1-PC takes almost twice as long to run the exact same simulation as VW2-PC. This shows that all computers running virtual world modules must have similar performing graphics cards, otherwise the whole simulation will be limited by the speed of the slowest computer.

Finally, our simulator was implemented using the Panda3D game engine. The Panda3D game engine is an open source project that does not provide the same performance such as other commercial game engines. Rewriting our simulator using an engine such as Unity3D [13] would provide a significant performance improvement.

IX. Conclusion

Reality Emulators—visually and behaviourally realistic environments, inhabited with life-like flora and fuana—have the potential to revolutionize camera networks research. These synthetic environments can serve as software laboratories within which simulated camera networks can be deployed, tested and evaluated. Inspired by this vision, here we present a 3D environment, along with the necessary camera models, communication infrastructure, and computer vision routines, that can be beneficial for camera networks research.[4] With the ability to quickly change the scene, the number of pedestrians and the locations and properties of the cameras, it is easy for researchers to study their camera networks in a variety of settings (See Fig. 4).

[4]The simulator used in this paper is released under Gnu Public License v3 and can be obtained by contacting the authors.

We are currently working to improve the quality of our virtual pedestrians so that they can better interact with each other and their surroundings. Our eventual goal is to simulate city scale 3D environments, populated with people and cars, animals, plants, etc. We also plan to implement the Pelco-D protocol for the simulated cameras, to further facilitate the transfer of camera networks algorithms from our simulated environment to physical camera networks.

Acknowledgment

The authors would like to thank Natural Science and Engineering Research Council of Canada for their generous financial support of this research. We would also like to thank Xerox Foundation for their support. Our gratitude to Adam Domurad who developed the visual analysis client that we use to evaluate our virtual vision simulator.

References

[1] F. Z. Qureshi and D. Terzopoulos, "Smart camera networks in virtual reality," *Proceedings of the IEEE (Special Issue on Smart Cameras)*, vol. 96, no. 10, pp. 1640–1656, October 2008.
[2] W. Shao and D. Terzopoulos, "Autonomous Pedestrians," *Graphical Models*, vol. 69, no. 5-6, pp. 246–274, September 2007.
[3] W. Starzyk, A. Domurad, and F. Z. Qureshi, "A virtual vision simulator for camera networks research," in *Proc. Ninth Conference on Computer and Robot Vision (CRV 12)*, Toronto, Canada, May 2012, pp. 1–8.
[4] D. Terzopoulos and T. F. Rabie, "Animat vision: Active vision in artificial animals," in *Proceedings of the Fifth International Conference on Computer Vision*, ser. ICCV '95. Washington, DC, USA: IEEE Computer Society, 1995, pp. 801–. [Online]. Available: http://portal.acm.org/citation.cfm?id=839277.840048
[5] S. W. Wilson, "The animat path to ai," in *Proceedings of the first international conference on simulation of adaptive behavior on From animals to animats*. Cambridge, MA, USA: MIT Press, 1990, pp. 15–21. [Online]. Available: http://portal.acm.org/citation.cfm?id=116517.116519
[6] D. Terzopoulos, "Perceptive agents and systems in virtual reality," in *Proceedings of the ACM symposium on Virtual reality software and technology*, 2003.
[7] R. B. A. Santuari, O. Lanz, "Synthetic movies for computer vision applications," in *Proceedings of the third IASTED International Conference on Visualization, Imaging, and Image Processing*, 2003.
[8] L. Kovar, M. Gleicher, and F. Pighin, "Motion graphs," *ACM Transaction on Graphics*, vol. 21, pp. 473–482, July 2002.
[9] "Pelco-d protocol manual," pp. 1–8, March 1999.
[10] W. Starzyk and F. Z. Qureshi, "Multi-tasking smart cameras for intelligent video surveillance systems," in *Proc. 8th IEEE International Conference on Advanced Video and Signal based Surveillance (AVSS 2011)*, Klagenfurt, Austria, August 2011, pp. 1–6.
[11] F. Z. Qureshi and W. Starzyk, "Learning proactive control strategies for ptz cameras," in *Proc. Fifth ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC 2011)*, Ghent, Belgium, August 2011, pp. 1–6.
[12] "Panda3d," http://www.panda3d.org, 2012, accessed June 2012.
[13] "Unity3d," http://unity3d.com/, 2012, accessed June 2012.

Wiktor Starzyk Wiktor Starzyk is currently pursuing a masters in computer science at the University of Ontario Institute of Technology (UOIT). He obtained a Bachelor of Science in Computer Science degree from UOIT in 2011. His research interests include: computer graphics, computer animation, crowd simulation and camera networks. He has previously published at camera networks and video surveillance conferences.

Faisal Z. Qureshi Faisal Qureshi is an Assistant Professor of Computer Science and the founding director of the visual computing laboratory at the University of Ontario Institute of Technology (UOIT), Oshawa, Canada. He obtained a PhD in Computer Science from the University of Toronto in 2007. He also holds an M.Sc. in Computer Science from the University of Toronto, and an M.Sc. in Electronics (with distinction) from Quaid-e-Azam University, Pakistan. Prior to joining UOIT, he worked as a Software Developer at Autodesk. His research interests include sensor networks, computer vision, and computer graphics. He has also published papers in space robotics. He has interned at ATR Labs (Kyoto, Japan), AT&T Research Labs (Red Bank, NJ, USA), and MDA Space Missions (Brampton, ON, Canada).

Dr. Qureshi is also active in conference organizations, serving as the general co-chair for the Workshop on Camera Networks and Wide-Area Scene Analysis (co-located with CVPR) in 2011-13, technical program committee chair ICDSC 2013, and publicity chair AVSS 2013. He is a member of the IEEE and the ACM.