DOI: 10.2352/ISSN.2470-1173.2016.21.3DIPM-399

Depth Assisted Composition of Synthetic and Real 3D Scenes

Santiago Cortes, Olli Suominen, Atanas Gotchev; Department of Signal Processing; Tampere University of Technology; Tampere, Finland

Abstract

In media production, previsualization is an important step. It allows the director and the production crew to see an estimate of the final product during the filmmaking process. This work focuses in a previsualization system for composite shots, which involves real and virtual content. The system visualizes a correct perspective view of how the real objects in front of the camera operator look placed in a virtual space. The aim is to simplify the workflow, reduce production time and allow more direct control of the end result. The real scene is shot with a time-of-flight depth camera, whose pose is tracked using a motion capture system. Depth-based segmentation is applied to remove the background and content outside the desired volume, the geometry is aligned with a stream from an RGB color camera and a dynamic point cloud of the remaining real scene contents is created. The virtual objects are then also transformed into the coordinate space of the tracked camera, and the resulting composite view is rendered accordingly. The prototype camera system is implemented as a self-contained unit with local processing, and it runs at 15 fps and produces a 1024x768 image.

Introduction

Previsualization (previs) is the name given by the movie industry to any system that allows the director and the staff to view an approximation of the end result before actually shooting the scene. It helps save time by minimizing the errors and the iterations necessary to materialize the view of the director.

The system proposed is a blend of two relatively recent concepts and the technology that has been developed from them. These concepts are mixed reality (MR) and the virtual camera. Mixed reality systems are a relatively recent advancement in technology. The term MR encompasses systems that merge virtual and real worlds to produce environments and visualizations where physical and virtual objects coexist and interact. The first general term for such a system, augmented reality (AR), was coined in 1990 to define systems where computer-generated sensory information (sound, video, etc.) is used to enhance, augment or supplement a real world environment.

A virtual camera captures the position and orientation of the camera inside a space and returns a render of the image from a point of view relative to it. To do this, the position of the camera has to be tracked. Examples of systems that have been adapted for camera pose estimation include accelerometers and gyroscopes in smartphones or motion capture in movie studios.

In order to improve on current previs virtual cameras, the system developed takes into account the position of the objects in the space. The real objects in the space are then mixed with the virtual objects, and a complete 3D joint space is produced. Real objects can occlude virtual objects and vice versa. The end result is a mixed reality rendered in place for the virtual camera system. This allows a movie director to explore how the real characters will blend in with a virtual background. This can help reduce errors and corrections done in postproduction thus reducing the cost and making the production cycle faster.

Prior work

An extensive description and formalization of MR systems can be found in [1]. After the release of the ARToolkit [2] by the Nara Institute of Technology, the research community got much better access to real time blend of real and virtual data. Most of the AR applications running on web browsers produced during the early 2000s were developed using the ARToolkit. MR has evolved with the technology used to produce it, and the uses for it have multiplied. Nowadays, in the embedded system era, a single device can have most of, if not all, the sensors and computing power to run a good MR application. This has led to a plethora of uses, from training and military simulation to simple videogames.

Any system that mixes data from a virtual and real environment to produce a joint world can be classified in the spectrum proposed in [1]. Most systems can be divided into a capture component and display components. The capture component is usually some kind of image capture device, a single camera [3], a stereo pair [4], or one or more depth sensors [5]. The display component shows the user the resulting environment, employing either screens, projectors, head mounted displays (HMD), or other visualization means.

The potential success of the Oculus Rift has revitalized the development of mixed reality systems using HMD. Relevant examples are the MR systems developed by University of California aimed at tele meeting [6]. The aim of these systems is to create a full 3D model of a person and place it in a shared environment with other people. Microsoft has announced the HoloLens [7]. This system is a view-through mixed reality display that places virtual objects on top of the real world using a semitransparent screen. Many of the recent MR developments use head mounted displays; these are a natural interface for systems that try to achieve presence, (the feeling of ownership towards a body), but this is not always the best option. The system presented in this paper uses a more familiar interface for a camera operator, where the display and capture systems are attached to a commercially available video camera shoulder mount.

The virtual camera concept has been introduced in the context of computer graphics [8]. It was described as a “cyclops” device, which renders a virtual scene according to a set of sensors attached to a camera tripod. The concept has evolved through the years and has been applied to videogames [9] and movies [10]. This evolution has been linked to the advancement in technology: better rendering capabilities, better sensing techniques, more powerful computers, etc. The most common use of the device is to visualize camera motions through a virtual environment in media productions. For example, the movie Avatar used a virtual camera to visualize and plan virtual camera movements [11].

In the University of Kiel, a mixed reality camera [5] was developed in 2008. This camera is also based on Time of Flight technology. The system scans the scene and creates a model of the background prior to the operation. Once in operation, the camera is fixed and the mixed space is created.

©2016 Society for Imaging Science and Technology
DOI: 10.2352/ISSN.2470-1173.2016.21.3DIPM-399

Description of algorithms

This section describes the algorithms and theoretical concepts used in the implementation of the proposed system. In order to keep track of the processing, three different coordinate spaces are defined. The camera coordinate space (CCS) is fixed to the moving camera. Its origin is in the optical center, the z coordinate is perpendicular to the image plane and the y coordinate is in the vertical direction of the image plane. The real coordinate space (RCS) is defined by the motion capture calibration and is static. The positions and orientations given by the motion capture system are all in the RCS. The virtual coordinate space (VCS) is defined by the modelling tool used to create the virtual objects. All virtual objects are in the VCS. Figure 1 shows the algorithmic blocks, which facilitate the flow of information from the sensor to the display.

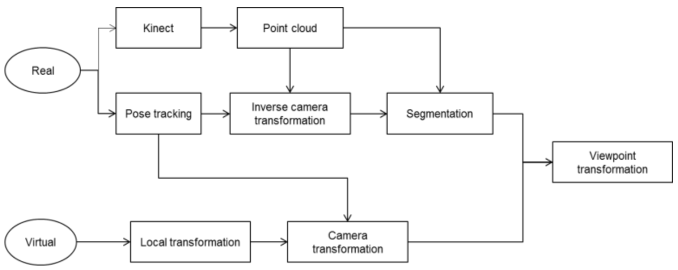


Figure 1. *Algorithm block diagram.*

Real data

The Real space is the volume inside the capture studio. It contains all objects inside the volume, the marker constellations and the RCS definition. The motion capture software is calibrated to have all camera poses relevant to an arbitrary reference frame selected by the user; this becomes the origin of the RCS.

The calibration of the system has three different phases presented below.

Pose tracking

OptiTrack camera system is used to estimate the pose of the camera [12]The camera is assumed to be a rigid body; its 6 degrees of freedom (DoF) pose is tracked using optical motion capture. The calibration of the pose tracking is done according to the procedures of the OptiTrack system. The position of the cameras is given in reference to a user defined point in the space. A final calibration is done to estimate the position of the camera sensor relative to the marker constellation. This process is referred to as external calibration. The calibration is designed specifically for this system and it is based on multilateration estimation. A brief description is presented below.

External Calibration is different from extrinsic calibration and its goal is to find the position of the camera sensor. The position is in reference to the marker constellation that is fixed to the body of the system. Since the final image is reprojected from the color camera point of view, the output is an estimate of the color camera sensor relative to the marker constellation. To setup the calibration, the camera is fixed to a stable position. A second constellation of markers is placed in the visible space in front of the camera. Table 1 summarizes the data necessary to perform the calibration and how to obtain it.

Table 1. Data to perform external calibration

Data	Format	description	Source
Kinect capture	$\boldsymbol{K} = (\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{z})$	$(\boldsymbol{u}, \boldsymbol{v})$ are the coordinates of the visible markers in the depth image. $\boldsymbol{z}$ is the depth measurement.	Kinect depth image
Camera constellation N markers	$\boldsymbol{C_i} = (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ $\boldsymbol{i} < \boldsymbol{N}$	The list of positions of the camera markers	OptiTrack camera system
Calibration constellation M markers	$\boldsymbol{S_i} = (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ $\boldsymbol{i} < \boldsymbol{M}$	The list of positions of the second markers	OptiTrack camera system
Intrinsic and Stereo calibration data	$[\boldsymbol{c_x}, \boldsymbol{c_y}]$ is the principal point. $\boldsymbol{f_x}, \boldsymbol{f_y}$ are the focal lengths.	The result of the previous calibration steps	Camera and stereo calibration (Kinect SDK)

Since the depth data given by the used depth sensor is already in a global coordinate system, first the points in the depth map have to be transformed back into distances to the sensor. Using triangle similarity in Figure 2, the distance can be calculated as

$$D(\mathrm{u}, \mathrm{v}, \mathrm{Z}) = \left\| \left(Z \frac{u - c_x}{f_x}, Z \frac{v - c_y}{f_y}, Z \right) \right\|, \tag{1}$$

where the calibration data is attained from the depth camera SDK.

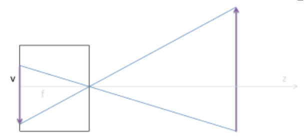


Figure 2. *2D Pinhole camera model*

The set of points of the calibration constellation S and their corresponding distances to the sensor can be treated as a set of spheres. The center of the sensor should intercept all of them. To find the maximum likelihood estimate (MLE) of the intersection of the spheres, we use the multilateration LS algorithm

$$\begin{matrix} D \\ \mathrm{S} \end{matrix} \rightarrow [Mlat] \rightarrow \vec{d}, \tag{2}$$

where $[Mlat]$ is the LS algorithm for the multilateration problem, and $\vec{d}$ is the MLE estimate of the position of the sensor. Using the translation parameter of the stereo calibration $\vec{t}$, the position of the color sensor $\vec{x}$ can be estimated as

$$\vec{x} = \vec{d} + \vec{t}. \tag{3}$$

The translation parameter has to be given in the correct direction (from depth to color). If it is not, inverting the extrinsic calibration matrix will yield the correct translation vector.

©2016 Society for Imaging Science and Technology
DOI: 10.2352/ISSN.2470-1173.2016.21.3DIPM-399

Finally, the camera marker constellation is redefined with the found camera sensor as the origin. Since the orientation remains the same, it is done by subtracting the sensor position from the marker position,

$$\dot{C}_i = C_i - \vec{x} \quad , \quad i < N, \tag{4}$$

where $\dot{C}$ is the corrected camera constellation.

The set of corrected camera marker positions can be put into the tracking software. This redefines the rigid body with the estimated camera sensor as the origin.

Joint bilateral filtering

The depth image usually suffers from noise and errors close to the edges of the color image. These impede the color and depth images. In order to tackle this problem, structural information from the color image is used into a bilateral filter to properly correct errors and reduce noise [13]

$$C(p) = \frac{1}{T} \sum_{p_i \in R} D(p_i)\, f(\|D(p_i) - D(p)\|)\ g(\|I(p_i) - I(p)\|), \tag{5}$$

where I is the intensity of the RGB color matrix (one color channel of the luminance channel in luminance-chrominance color space representation), D is the input depth matrix, and C is the output depth matrix. All matrices have the same size. R is the window of the filter; f and g are the kernels of the intensity difference and distance metric respectively. These kernels are arbitrarily selected, a common choice is to use Gaussian functions. T is the normalization term,

$$T = \sum_{p_i \in R} f(\|D(p_i) - D(p)\|)\ g(\|I(p_i) - I(p)\|), \tag{6}$$

which ensures that the filter preserves image energy.

The intensity distance is done relative to the difference in color rather than in depth. This aligns the edges of the depth image to the color image. [13]

Real point cloud

The point cloud is built by calculating the 3D coordinates $[x, y, z]^T$ of the points in the depth image

$$\vec{p} = [x, y, z]^T = \left[Z\frac{u - c_x}{f_x}, Z\frac{v - c_y}{f_y}, Z \right]^T \tag{7}$$

where (u, v) are the coordinates of the pixel, Z is the depth measurement and the rest are the parameters extracted from the calibration matrix. The depth image is already aligned to the color image in the earlier processing. Hence, the texture mapping is the same as the grid of the image. The output is the point cloud in the CCS.

Some points in the real point cloud will be outside of the capture volume. These points are of no interest, and should not be rendered. To find out which points are to be ignored all the points are transformed from the CCS to the RCS using

$$\vec{p}' = P\vec{p} \qquad P = \begin{bmatrix} R & \vec{t} \\ \mathbf{0} & 1 \end{bmatrix}. \tag{8}$$

Points whose corresponding positions in the RCS are outside of the capture volume are discarded from the render. The rest are rendered normally.

Virtual elements

The virtual space is a set of 3D objects, described in mesh format. Each object has a texture and a rigid body position associated to it. The set of objects forms the virtual scene and is defined in the VCS.

Virtual objects are transformed from the VCS to the RCS using their own associated rigid body matrix.

$$\vec{v}' = T\vec{v}, \tag{9}$$

where $\vec{v}$ is the vector of coordinates of a vertex in the mesh description and $\vec{v}'$ in the vector of coordinates in the RCS. This builds the virtual room.

The camera transformation puts the virtual objects in the CCS

$$\vec{v} = P^{-1}\vec{v}', \tag{10}$$

where P is defined in equation 95.

The viewpoint transformation puts the objects in the image plane of the virtual camera. Since both the real and the virtual objects are in the CCS, only the intrinsic matrix is necessary.

The pixels are painted with the corresponding texture from the object. Z buffering lets the renderer know which object is closer to the camera and which ones are occluded. This blends the virtual meshes and the point cloud in the same 3D space with correct occlusion order.

Software and hardware implementation

The hardware composition is shown in Figure 3. The algorithm description is grouped in information processing units, which are not necessary implemented in the same system or covered in the same block in the computational description.

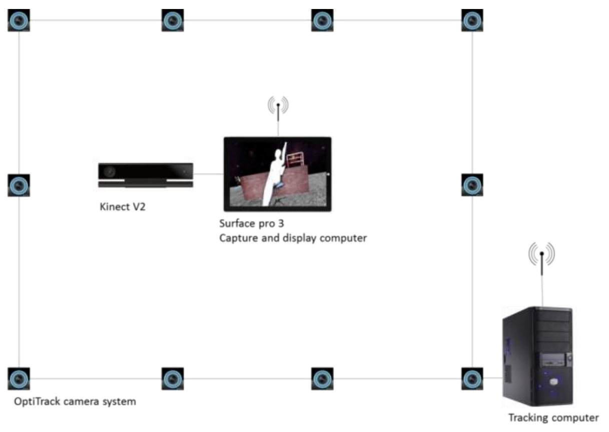


Figure 3. Hardware diagram, only information links are shown, power connections are ignored.

The implementation of the algorithm presented in the previous chapter has several distinct parts. Figure 4 shows the block diagram that describes the system. Commercial SDKs and software are considered black boxes and are only described in terms of inputs and outputs.

©2016 Society for Imaging Science and Technology
DOI: 10.2352/ISSN.2470-1173.2016.21.3DIPM-399

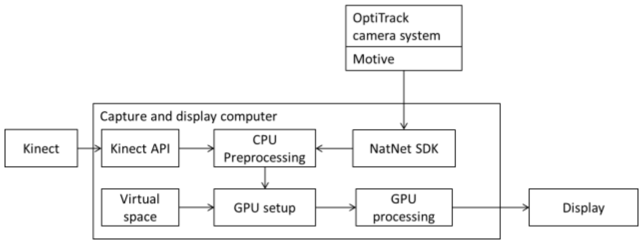


Figure 4. *Computational Block Diagram*

Kinect V2 depth sensor

A Kinect V2 depth sensor is used to sample the real scene. It captures a color image with 1920x1080 px resolution and a ToF depth image with 512x424 px resolution. The two images are then resampled and aligned together. Access to the different modalities is provided through the Kinect SDK [48], which drives the capture. The API also performs the mapping from the depth camera to the color camera. In Figure 5, the flowchart of the processing done in the SDK is shown.

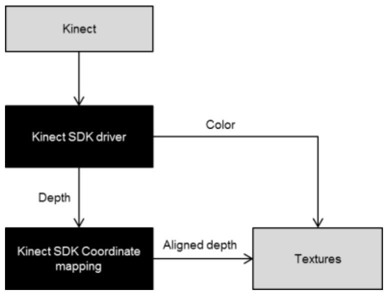


Figure 5. *Kinect process diagram*

OptiTrack and the NatNet SDK

The OptiTrack camera system is a motion capture system. It uses a set of infra-red (IR) light sources and IR cameras (8 in the current space) to find the position of IR reflective markers within the scene. The user defines sets of markers attached to rigid bodies and the system solves their 6 DoF position and orientation.

Natural point provides an SDK that interfaces their software (Motive) and the user's application. The SDK sends the data packets through a previously setup wireless connection and the synchronization of the data transfer. The data is sent at a constant framerate set in motive (12fps, 30fps, and 60fps). The block diagram is shown in Figure 6.

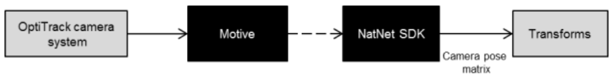


Figure 6. *Motion capture process diagram*

CPU Preprocessing

The point cloud structure is built before starting the capture process. It is never modified in the CPU once it has been built. The virtual scene is preprocessed by applying any changes to the transforms (placing the objects in their initial position). The real scene is created as an empty template; a point cloud is built where all points have the same depth and are distributed as a rectangle.

The flowchart in Figure 7 shows the steps to prepare the scene before rendering, which are all performed in CPU every frame. This process calls upon the Kinect SDK to deliver the color and depth and the motion capture to deliver the pose of the camera. If either of them has not produced a new frame of information, the previous one is used. The depth and color textures are loaded into the point cloud and the transformation is loaded into the virtual world.

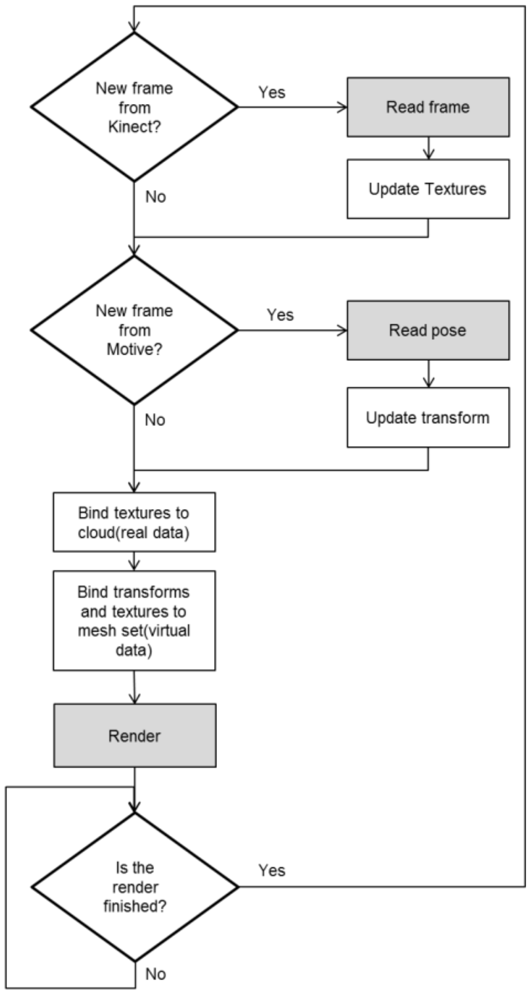


Figure 7. *Flowchart of preprocessing*

GPU processing

The processing in the GPU is programmed in shaders .There are two sets of shaders, one for the real point cloud and one for the virtual objects. The point primitive is used to render the real point cloud. The triangle primitive is used to render the virtual objects, Figures 8 and 9 show the processing done by the GPU divided into the shaders.

It is important to note that the inner workings of the rasterizer are not included in the block diagrams, and that the blue dashed arrows mean transfer of information. This information is used to interpolate and re-sample the textures and meshes by the rasterizer.

©2016 Society for Imaging Science and Technology
DOI: 10.2352/ISSN.2470-1173.2016.21.3DIPM-399

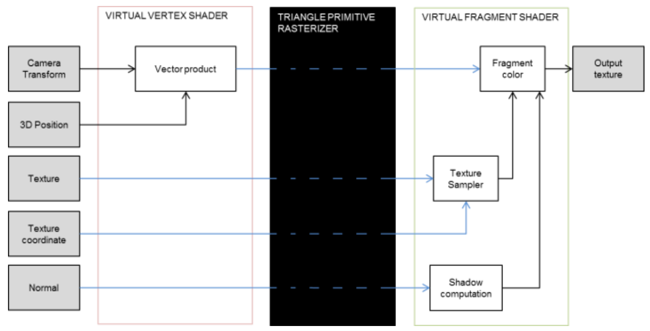


Figure 8. *Shader block diagram for virtual objects. Blue lines indicate information transfer. The rasterizer interpolates and samples the corresponding values for each fragment.*

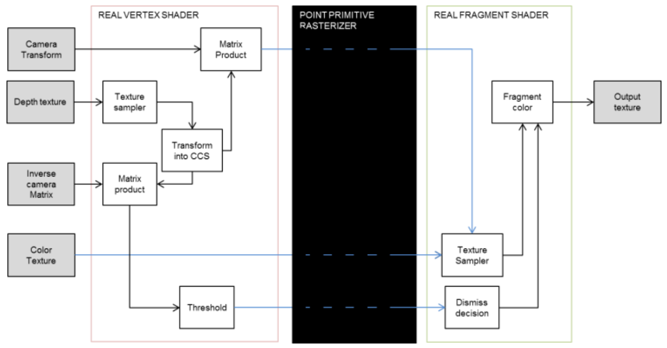


Figure 9. *Shader block diagram for real point cloud, blue lines indicate information transfer. The rasterizer interpolates and samples the corresponding values for each fragment.*

Results

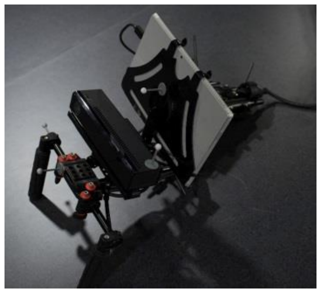


Figure 10. *Virtual camera*

The main result of the work is the actual camera seen in Figure 10. The prototype is implemented in the CIVIT Motion Capture Studio at TUT. The components are a Kinect v2 for capture, a Microsoft Surface Pro3 for display and processing, four IR markers and a shoulder camera mount that holds the system together and allows the operator a natural operation of such a device.

The system is evaluated based on the following parameters: refresh rate, delay, and visual quality. First, the external calibration was designed specifically for this system; hence, the results of the calibration for the current prototype are shown in this section. Second, the offline experiments designed to measure the performance of the system are presented. These include the current system and the bilateral filter meant to refine the images after data fusion. Finally the real time performance is presented. This includes timing parameters and example photos to evaluate the visual quality.

Joint calibration

The joint calibration was developed specifically for this particular system and the detailed results are discussed below. Figure 11 contains the setup used for calibration. Figure 12 shows the markers as seen by the mocap. The two sets are easily differentiated since the camera has four markers and the calibration constellation has six.

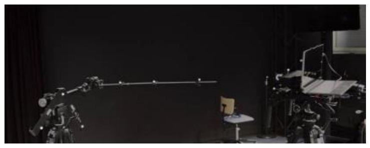


Figure 11. Setup for joint calibration

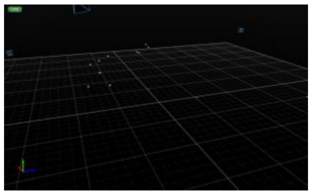


Figure 12. *Markers for joint calibration as seen by the mocap system*

With this data, the external calibration is performed. The resulting model is shown in Figure 13. After the calibration, the origin of the constellation is a good estimate of the optical center of the camera and sufficient for virtual view rendering.

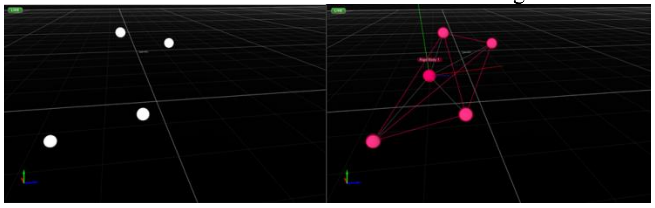


Figure 13. *Left, set of markers that define the position of the virtual camera. Right, set of markers with virtual origin. Ideally this origin corresponds to the optical center of the color camera.*

Offline experiments

Due to performance constrains, some of the experiments were performed offline. Each experiment below is performed offline, the data acquisition and the processing performed are describes in this section.

Joint bilateral filter

A test program is written in the Kinect SDK that simultaneously captures the depth and color and exports them into images as well as the transformation matrix provided by the mocap system. The depth is encoded into the G and B channels of a bmp file.

©2016 Society for Imaging Science and Technology
DOI: 10.2352/ISSN.2470-1173.2016.21.3DIPM-399

The data is imported into Matlab, where the bilateral filter proposed in [14] is implemented. The filter is performed for several kernel sizes. For each size, the time and resulting depth are saved. These values are used to create several renderings of the same joint reality scene. Evaluating the performance is not straight forward, since most metrics rely on having a ground truth to compare against. Since the main issue with the camera is the pixels outside of an object that appear in the same plane, the percentage of wrong pixels reduced from the original depth image is taken as the performance metric. A pixel is wrong if it is farther than 10 cm from its actual location. Figure 14 shows several joint spaces in which the input was filtered with different kernel sizes while Figure 15 shows the goodness metric computed.

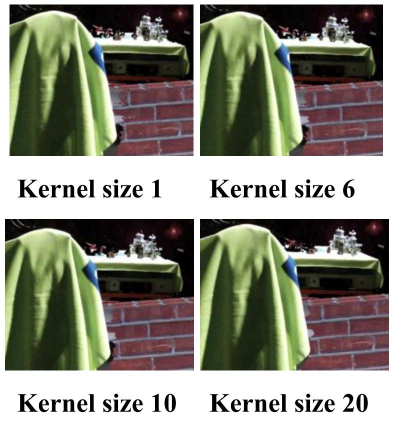


Figure 14. Detail of result of bilateral filter. See how the edge starts to match the color edge

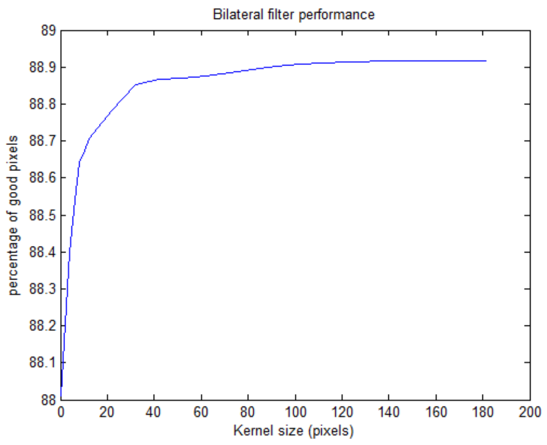


***Figure 15.** Bilateral filter performance in the real scene*

Note that the noise is rapidly changing and quite localized to the edges. This makes it distracting and damages the visual appearance. The filtered image is much smoother and subjectively more naturally looking.

Delay calculation

Since the system is a combination of different systems, calculating the entire delay is not straightforward. The experiment designed to measure the delay requires a separate camera. The camera used is a Basler acA1920 running at SD resolution at 60 fps. The camera is capable of higher resolution and faster capture, though, given the refresh rate of the display, no higher specs are necessary.

The camera is framed so it captures both the space and the render in the screen of the camera. A periodic motion is created in front of the camera with a highly contrasted round object. A white circle was hung from the ceiling and its pendulum oscillation against a black background was recorded in the high-speed camera. Figure 16 contains a frame of the video.

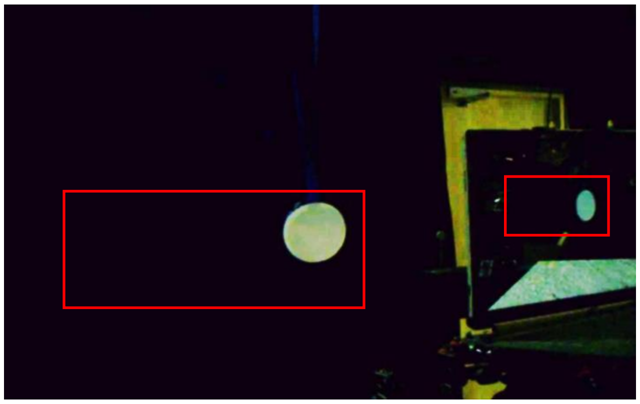


***Figure 16.** Frame of delay calculating vide, sub-images are shown as red rectangles.*

The area of movement from both circles is estimated by inspection and for each frame; they are extracted and analyzed separately. The first step is to binarize both sub-images with a threshold calculated according to Otsu's method [4]. Then, since there are no other white objects in the background of the sub-images, simple gravity center of the intensity values is used to estimate the center of each circle.

Since both cameras are aligned to be parallel to the ground, the horizontal coordinates of each circle are strongly correlated. The vertical coordinates have a much smaller range and double the frequency (see Figure 17) so they are not used for the delay estimation.

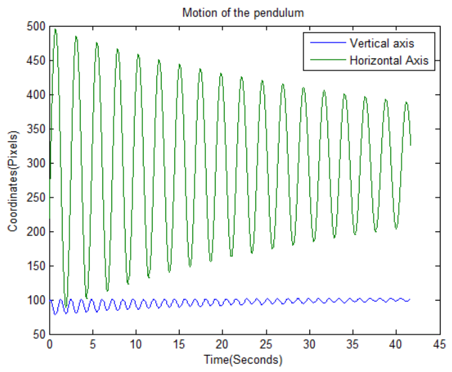


***Figure 17.** Motion of the center of the pendulum*

The pixel position is irrelevant, what is important is the relative position in the pendulum path. For this, the horizontal position is scaled and translated so its highest value is one and its center is zero. This is done for circles, see Figures 18 and 19.

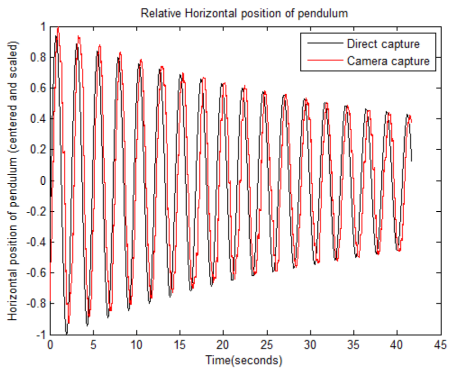


***Figure 18**. Relative horizontal motion*

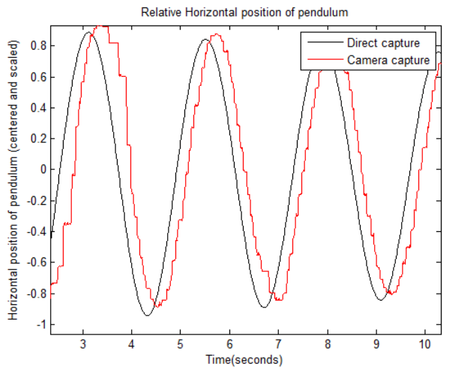


***Figure 19.** Closer look into Relative horizontal motion*

The position of the center point in the camera capture has a staircase shape when plotted against time, this is due to the refresh rate of the camera system (15 fps) being smaller than the capture rate (60 fps).

Filtering the signal would average the delay along the time axis and reduce the delay variance. Since the autocorrelation estimates the delay along the whole signal, it estimates the mean delay. In this case this mean is invariant to a time domain filter. Filtering the input will have no change and may introduce unwanted artifacts.

Given the two signals, the delay is estimated by using the cross-correlation signal, shown in Figure 20

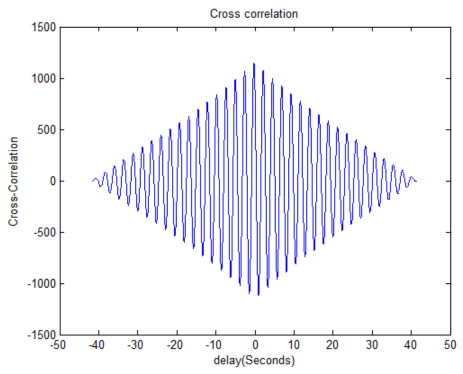


***Figure 20.** Cross-correlation of the signals*

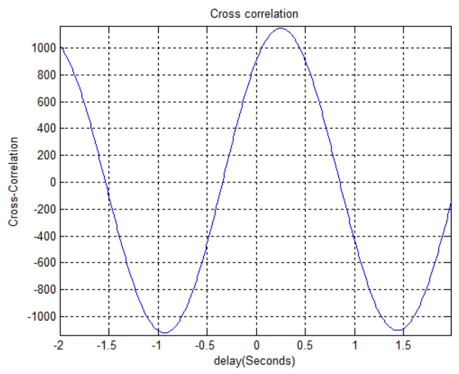


***Figure 21.** Closer look into Cross-correlation of the signals*

The highest peak of the cross correlation is at a delay of 250 ms, as seen in Figure 21, which is the estimated delay of the system.

Performance

The basic performance metrics are shown in Table 4.

Table 2. System performance metrics

Refresh rate	15 fps
Delay	250 ms
Point cloud size	960x540
Texture resolution (internal)	1920x1080*

* Internal resolution of the point cloud, output resolution depends on visualization.

The tracking marker accuracy is given by the motion capture system. The LS algorithm returns the accuracy of the center estimate. For the orientation error, all possible orientations contained in the markers with the given variance are calculated and the worst case scenario is used as the error. These values are shown in Table 5.

Table 3. Accuracy of motion capture

Marker error	±0.9 mm
Position error	±0.5mm
Orientation error	±0.4°

©2016 Society for Imaging Science and Technology
DOI: 10.2352/ISSN.2470-1173.2016.21.3DIPM-399

Figure 23 shows the capture space, the IR cameras are attached to rafters in the ceiling and along the walls. The markers are placed in the camera in such a way that only one solution to the model fitting problem exists (see Figure 22).

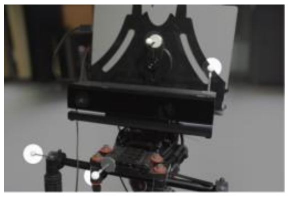


Figure 22. *IR markers highlighted in virtual camera.*

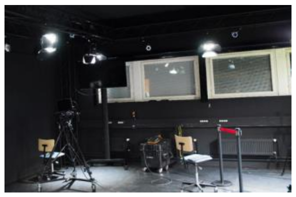


Figure 23*. Motion capture studio, the blue rings are the IR cameras.*

Figure 24 shows the marker constellation as seen by the mocap software. For four markers, a proper constellation is not coplanar and isosceles triangles should be avoided.

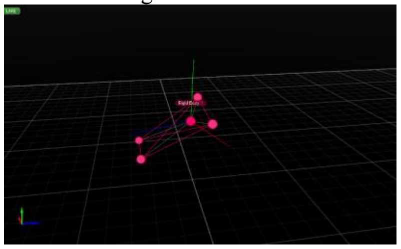
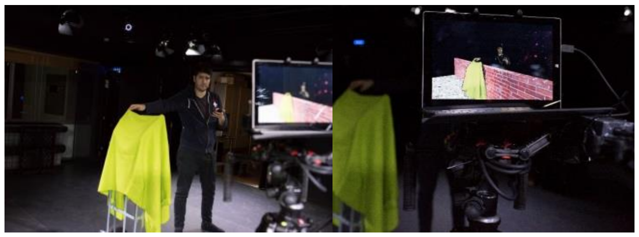


Figure 24. *Marker disposition in camera (shown by MOCAP)*

Figure 24 shows two angles of the same scene in order to demonstrate how the system is coherent in its mix of real and virtual space. The relative positions of objects are not lost when moving the camera around. As discussed, this movement is continuous and the camera responds in an interactive manner.



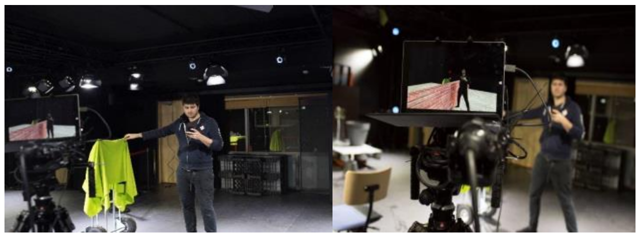


Figure 25. *Sample of position and orientation consistency [4] [5, 7]*

Conclusions and future work

In this work, a complete system for previsualization of mixed real and synthetic content has been presented and documented. The system is a functioning prototype, a laboratory proof of concept that showcases the potential of data fusion and multimodal sensing in a movie production context.

The OptiTrack camera system and the Motive software are used to perform optical motion tracking. A total of 8 cameras are installed in the studio, and a four marker constellation is attached to the camera. The resulting matrix is transmitted via Wi-Fi to the camera. Every 33 milliseconds this matrix is updated.

In order to add the real data, a Kinect V2 camera is used. The depth and color feeds are aligned by the Kinect API and are provided to the system every 66 milliseconds. The system sends the frames as textures into the GPU and lets it do the rest of work.

In the GPU, the worlds are mixed and projected to the corrected position of the camera. The processing is done in such a way that minimizes the time spent between capture and display. The result is a mixed reality render of virtual and real worlds which updates fast enough to provide an “interactive” experience.

This system is still in an early development state and needs further work.

First, the depth has to be refined so mitigate erroneous depth edges. Such errors cause an aesthetic issue where noise is present at the edge of an object in the final image. An edge aware filter that aligns the depth edges with the color edges has to be utilized. In this work, a joint bilateral filter was implemented. However, this implementation proved to be rather slow and unsuitable for the system with its current processing power. Having the processing done in a local machine reduces the delay and thus helps the immersion. Having the processing done in a separate unit with more processing power enables more complex algorithms for the price of some time delays. Therefore, further optimization tests should be held.

Second, focusing should be integrated into the system since filmmakers use focus as a tool to help move audience’s attention. Since the scene is already available in 3D, the refocusing should be straightforward. However, it is important to find a fast implementation and integrate it with the rest of the processing.

Third, some helpful camera interface controls have to be added in order to provide the camera operator with better controls and extend his/her experience. The basic controls are a focus barrel, a zoom controller and some basic lighting controls. Similar controls do exist in the Natural Points virtual camera, albeit the lack of 3D scene capture, and have been used in several movies.

©2016 Society for Imaging Science and Technology
DOI: 10.2352/ISSN.2470-1173.2016.21.3DIPM-399

References

[1] P. Milgram and A. F. Kishino, "Taxonomy of Mixed Reality Visual Displays," in *IEICE Transactions on Information and Systems. pp. 1321–1329*, 1994.

[2] H. Kato and M. Billinghurst, "Marker Tracking and HMD Calibration for a video-based Augmented Reality Conferencing System," in *International Workshop on Augmented Reality*, San Francisco, USA, 1999.

[3] A. J. Davison, I. D. Reid, N. D. Molton and O. Stasse, "MonoSLAM: Real-Time Single Camera SLAM," in *IEEE Transactions on Pattern Analysis and Machine Intelligence, vol.29, no.6, pp.1052-1067*, 2007.

[4] M. Kanbara, T. Okuma, H. Takemura and N. Yokoya, "A stereoscopic video see-through augmented reality system based on real-time vision-based registration," in *proceedings for IEEE Virtual Reality conference*, 2000.

[5] I. Schiller, B. Bartczak, F. Kellner, R. Kollmann and R. Koch, "Increasing Realism and Supporting Content Planning for Dynamic Scenes in a Mixed Reality System incorporating a Time-of-Flight Camera," in *5th European Conference on Visual Media Production, pp 1-10*, 2008.

[6] G. Kurillo, R. Bajcsy, K. Nahrsted and O. Kreylos, "Immersive 3D Environment for Remote Collaboration and Training of Physical Activities," in *IEEE Virtual Reality Conference pp.269-270*, 2008.

[7] Midrosoft, "Microsoft Hololens," [Online]. Available: http://www.microsoft.com/microsoft-hololens/en-us. [Accessed 25 June 2015].

[8] B. W. Paley, "Interaction in 3D Graphics," in *Proceedings of SIGGRAPH 98, pp 43--54.*, 1998.

[9] A. Lecuyer, J.-M. Burkhardt, J.-M. Henaff and S. Donikian, "Camera Motions Improve the Sensation of Walking in Virtual Environments," in *Virtual Reality Conference, pp.11-18*, 2006.

[10] K. Hansung, R. Sakamoto, I. Kitahara and T. Toriyama, "Virtual Camera Control System for Cinematographic 3D Video Rendering," in *3DTV Conference 2007, pp.1-4*, 2007.

[11] B. J. Lee, J. S. Park and M. Sung, "Vision-based real-time camera match moving with a known marker," in *In Entertainment Computing - ICEC 2006*, 2006.

[12] Natural Point, "Optitrack camera System," [Online]. Available: www.optitrack.com. [Accessed 1 October 2015].

[13] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Sixth International Conference on Computer Vision, pp.839-846*, 1998.

[14] C. Rhemann, A. Hosni, M. Bleyer, C. Rother and M. Gelautz, "Fast cost-volume filtering for visual correspondence and beyond," in *2011 IEEE Conference on Computer Vision and Pattern Recognition (CVPR),pp.3017-3024*, 2011.

Author Biography

Santiago Cortes received his BS in electronic engineering from the Universidad de los Andes (2012) and his Ms in Information Technology from Tampere University of Technology (2015). Since 2014 he has worked in the centre for immersive visual technology (CIVIT) in Tampere, Finland. His work has focused on image processing and computer graphics applied to media production.

Olli Suominen received his M.Sc degree in information technology at Tampere University of Technology in 2012. Currently, he is a researcher at the Department of Signal Processing at Tampere University of Technology, pursuing a Ph.D. in signal processing. His research interests are in developing efficient methods for 3D scene reconstruction, depth processing and multi-modal light-field capture.

Atanas Gotchev received M.Sc. degrees in radio and TV engineering (1990), and applied mathematics (1992), and a Ph.D. in telecommunications (1996) from the Technical University of Sofia, Sofia, Bulgaria and the D.Sc.Tech. degree in information technologies from Tampere University of Technology, Tampere, Finland, in 2003. Currently, he is an Associate Professor (tenure track) with Department of Signal Processing at TUT. His recent work concentrates on algorithms for multi-camera and multi-sensor 3D capture, transform-domain light-field reconstruction, and Fourier analysis of 3D displays.