Vision, Modeling, and Visualization (2011)
Peter Eisert, Konrad Polthier, and Joachim Hornegger (Eds.)

FreeCam: A Hybrid Camera System for Interactive Free-Viewpoint Video

C.Kuster[1] , T. Popa[1] , C. Zach[1] , C. Gotsman[1,2] , M. Gross[1]

[1]ETH Zurich, Switzerland
[2]Technion - Israel Institute of Technology, Haifa, Israel

Abstract
We describe FreeCam - a system capable of generating live free-viewpoint video by simulating the output of a virtual camera moving through a dynamic scene. The FreeCam sensing hardware consists of a small number of static color video cameras and state-of-the-art Kinect depth sensors, and the FreeCam software uses a number of advanced GPU processing and rendering techniques to seamlessly merge the input streams, providing a pleasant user experience. A system such as FreeCam is critical for applications such as telepresence, 3D video-conferencing and interactive 3D TV. FreeCam may also be used to produce multi-view video, which is critical to drive new-generation autostereoscopic lenticular 3D displays.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation

1. Introduction

Free-viewpoint video is an area of active research in computer graphics. The goal is to allow the viewer of a video dataset, whether recorded or live, to move through the scene by freely and interactively change his viewpoint, despite the footage being recorded by just a small number of static cameras. Potential applications can be easily imagined in the sports, gaming, entertainment, and defense industries. Most prominent are telepresence, interactive TV and immersive video-conferencing, which has emerged as a key Internet application with the now widespread use of commercial voice-and-video-over-IP systems, such as Skype.

Free-viewpoint video relies on *novel view synthesis* from the acquired set of video images, which means rendering the scene from a viewpoint different from those acquired by the physical cameras. Another important use for novel-view synthesis is so-called *multi-view video*. The goal is to generate *simultaneous* views of the scene from multiple viewpoints. This is necessary to drive auto-stereoscopic lenticular 3D displays, such as those of Philips and Sharp, which require simultaneous images of the scene from multiple (typically eight or sixteen) viewpoints, providing a 3D stereo viewing experience. Such displays will take off only if it is easy to generate the appropriate multi-view content. At the heart of our FreeCam free-viewpoint video system, described in this paper, is an efficient algorithm for novel-view synthesis.

Novel view synthesis has been studied for the last 15 years, beginning with the pioneering work of Kanade et al. [KNR95], but progress has been limited due to the lack of accurate dynamic scene geometry information. Although some encouraging results have been obtained, it seems that unless a very large number of video cameras are used, it is difficult to obtain high-quality results for omni-directional viewpoints. Unfortunately, the more cameras used, the more processing is required to take advantage of all the cameras and the more the system becomes sensitive to camera calibration inaccuracies, so application speed will deteriorate as the image quality improves. Due to this, most of the work to date on novel view synthesis is offline, i.e. can be applied only to pre-recorded video and pre-determined camera trajectories. This prevents its use for the more exciting applications of novel view synthesis technology, which require real-time processing of live video. FreeCam provides a big step towards this goal by relying on only a small number of sensors.

In general, a small number of video cameras do not capture sufficient information about the scene in order to be able

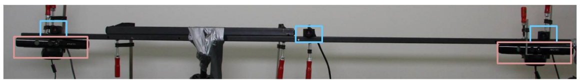

Figure 1: *Hardware setup. The two Kinect depth cameras are circled in pink and the three color cameras in blue.*

to render novel views reliably. They must be complemented by accurate and dense depth information about the scene. Although scene depth may be reconstructed from close pairs of video cameras using software stereo techniques, it is quite difficult to do so reliably or fast, so high-quality real-time stereo in software is still elusive. FreeCam overcomes this by using new camera technologies capable of real-time depth acquisition.

Real-time depth video cameras have evolved over the past decade. These cameras, called *depth cameras*, provide a depth value for each pixel, along with the usual RGB color values. Various sensing technologies are used in depth cameras, such as infrared time-of-flight, stereo, or structured light. While progress in these technologies has been slow, hampered by costly hardware components and slow software processing speeds, a recent breakthrough has been obtained with the introduction of the Microsoft Kinect gaming console[†]. This cheap (USD 150) device contains a depth video camera based on active stereo (structured light) in the infrared domain, developed by Primesense[‡]. Kinect provides 640×480 pixels of 11-bit precision depth at 30Hz. Although primarily intended for markerless motion capture in gaming applications, the depth sensor may be used independently for other applications. FreeCam tightly integrates the Kinect with traditional RGB video to provide the information necessary for novel-view synthesis.

In this paper we describe in detail our FreeCam system—a hybrid system for free-viewpoint video based on both color and depth video cameras. Combining the various data streams seamlessly to synthesize a novel view of a general scene is a significant challenge, and multiple issues, such as sensor noise, missing data, coverage, seamless data merging and device calibration require non-trivial processing to generate quality artifact-free imagery. It is all the more challenging to perform all this at real-time rates. Using off-the-shelf hardware, including the Kinect, and highly parallelized multi-pass data processing, to be described in detail, we are able to generate free-viewpoint video at interactive rates. We expect that future low-level optimizations will bring the performance to near real-time.

2. Related Work

The earliest algorithms for novel view synthesis were based on simply interpolating between the image streams obtained from a large set of video cameras. This straightforward approach is still valid if many cameras can be tolerated, in terms of cost, engineering effort to calibrate and synchronize them, and computing resources required to process such large amounts of data in real-time. A recent system which still advocates this approach is the 16-camera setup of Matusik and Pfister [MP04].

More ambitious systems attempt to extract some *geometric* information from a smaller number of image streams and render novel views based on that. When the scene consists only of a single object, it is possible to use the concept of the *visual hull*, introduced by Laurentini [Lau94], which is the best (i.e. tightest) approximation of the shape geometry consistent with a set of given silhouettes of the shape from multiple views. Thus it makes sense to use it in lieu of the shape geometry when rendering a novel view [MBR*00, GWN*03, LMS04b, PLM*10]. For known subject shapes, such as human characters, an alternative to the visual hull as a proxy geometry is a 3D template. Carranza [CTMS03] first recover the pose of the character from the silhouettes and use it to deform the template proxy geometry. Other techniques are based on the related *photo hull* concept [YWB02, SSH03, LMS04a], which uses photo-consistency (between the geometry and image data) to achieve a tighter fit to the true single-object geometry.

While hull- or template-based methods could be effective for novel view synthesis of single objects, they are not very suitable for general multi-object scenes. In these scenes, unless a large number of cameras are employed, extraneous so-called *phantom geometry is generated [FB09]. This "virtual" geometry may be eliminated somewhat if extra information is used (e.g. correspondence information between views [BG08]), but it seems that completely different, more generic techniques must be employed for these scenes.* The first attempt to use explicit and dense depth information for novel view synthesis was made by [KNR95]. In their "virtualized reality" system, they computed depth images from stereo pairs, and rendered (offline) novel views by simply rendering the textured depth maps from the new viewpoints. Methods followed [PCD*97, WWC*05, KVLB08] using several color images registered with range images, the latter acquired using stereo techniques, to create novel views of a static object. The focus of these attempts, and those of their contemporaries, is on the reliable acquisition of depth images using stereo which are registered well to the color image. Zitnick et al. [ZKU*04] describe a system based on eight closely-positioned video cameras. In an offline step, they use stereo techniques to generate as accurate depth maps as possible, which are then used together with the video streams to render novel views in real-time (on the GPU). A key technique in their system is the use of "layered depth images" and "boundary matting" along object silhouettes, which are the part of the image most sensitive to errors in the depth map. Recent work of Waschbüsch et al. [WWG07] describes a system for recording and playback

[†] www.xbox.com/en-US/kinect

[‡] www.primesense.com

of dynamic 3D scenes which uses structured light-assisted space-time stereo for depth map computation and planes as rendering primitives. None of these works attempt to approximate the missing geometry of an entire scene, rather rely on a sufficient coverage of the scene by the sensors. More recent methods for free-viewpoint rendering [SH07, dST*08] produce convincing rendering results, but are hindered by lengthy processing times that make them impractical for real-time application.

The advent of depth cameras has motivated the more recent systems to incorporate depth video directly. Several methods use a separately-acquired depth map to complement or speed up the geometric reconstruction produced by classical stereo methods [GFP08, ZWYD08, HA09, KTD*09]. The first attempt to use depth cameras for novel view synthesis was by Bogomjakov et al. [BM06], who used two depth cameras and two color cameras, registered in space and synchronized in time, to generate free-viewpoint video of a single object (typically the upper torso of a person). They did this using the so-called *depth hull*, an extension of the classical visual hull. The hybrid system of Tola et al. [TZCZ09] incorporates a single depth camera and two color cameras. The acquired depth data facilitates a constrained stereo approach where the depth of each rendered pixel is computed in real-time based on correspondence to the two color images. Since the appearance of the Kinect on the market, a number of popular systems combining two such devices to generate novel views have been demonstrated [§]. However, these use the most obvious and simplest technique for the synthesis - rendering two textured depth surfaces, without any conditioning of the data. While these are an important step in the same direction as FreeCam, simple systems such as these do not address a variety of other issues critical to achieve quality free-viewpoint video, such as data filtering, data completion, occlusion resolution and visual artifact reduction, which may be achieved only by paying some price in performance.

3. Hardware

The FreeCam hardware system consists of three Grasshopper color video cameras from Point Grey[¶] (1024×768 pixel 24-bit color output at 30Hz) and two *Kinect* systems from Microsoft (640×480 pixel 11-bit output at 30Hz). FreeCam uses just the infrared active-stereo depth video system within the Kinect, accessing this raw data with Zephod's Win32 Kinect driver software[‖]. In principle, we could have used the color cameras already bundled in the Kinect, but we preferred the superior quality of the Grasshoppers. The cameras are positioned on a straight rig, where the three Grasshoppers are positioned at equal distances from each other (approximately 1m), and the two Kinects directly above the two extreme Grasshoppers (see Fig. 1). The Grasshoppers were connected by IEEE 1394B Firewire and the Kinects by USB 2.0 on separate buses to a PC containing an Intel Xeon X560 3.33GHz processor and a NVidia Quadro 6000 graphics card with 448 CUDA cores. Nothing special was done to synchronize the Kinects and Grasshoppers in time, although in a future high-speed version of the system, this will probably be more critical.

3.1. Camera and Rig Calibration

While camera calibration using designated calibration patterns is a standard procedure in computer vision, some care has to be exercised if high accuracy is critical, which is the case for FreeCam. Camera calibration proceeds in a number of steps. First, to determine the intrinsic parameters of the cameras - focal length, principal point and lens distortion parameters - we use the coded marker approach of Irschara et al. [IZB07], which requires acquiring some 10-20 images of the marker pattern from each camera, but does not require knowledge of the 3D geometry of the pattern. Second, the same coded markers, if imaged by all cameras at fixed positions, allow to establish accurate correspondences between the different sensor images, facilitating extrinsic calibration of the sensor positions and orientations in a common "vision" coordinate system by standard structure-from-motion methods. The global scale ambiguity of the vision-based extrinsic calibration is resolved by establishing a correspondence between reconstructed markers and their mutual distance in the real world.

Finally, the parameters for the conversion between disparities returned by the Kinect system and true metric depth must be determined. Empirical studies[**] show that the relation between disparity d and metric depth z in the Kinect system is given by

$$z(d) = \frac{8bf}{d_{off} - d} \qquad (1)$$

where b is the unknown baseline of the depth sensor, f is the known focal length, and d_{off} is the unknown offset in the disparity range. The unknowns b and d_{off} can be solved for (using robust estimation) from the (z,d) pairs gleaned from color and depth images of the corresponding coded markers. The values obtained vary slightly between different Kinect systems, but we found that they are usually close to b = 7.5cm and d_{off} = 1090.

4. Novel View Synthesis

At the heart of the FreeCam software lies our novel view synthesis algorithm. This carefully combines the various live color and depth data streams in real-time and renders an image from a desired novel viewpoint. The obvious tradeoff is between data quantity, processing time and rendered image

§ www.cs.unc.edu/∼maimone/KinectPaper/kinect.html

¶ www.ptgrey.com

‖ www.openkinect.org

** www.ros.org/news/2010/12/technical-information-on-kinect-calibration.html

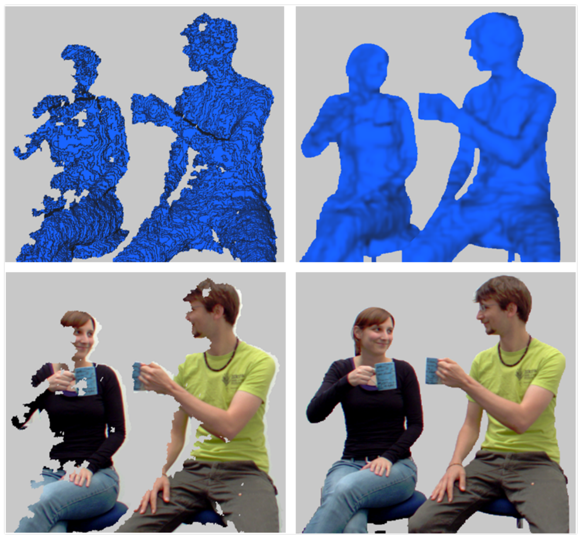
Figure 2: *Processing of single Kinect depth map. (Top row) Depth map only. (Bottom row) Textured depth map. (Left column) Raw Kinect data. (Right) Filtered and completed data.*

quality, and with current hardware and software technologies, it is still a significant challenge to achieve artifact-free imagery at interactive rates. Among other things, correcting erroneous data and filling in (or "faking") data where it is missing to provide a compelling result requires significant processing, which can be quite costly.

4.1. Rendering Algorithm Overview

Given the parameters of the novel view V, our synthesis (i.e. rendering) algorithm proceeds as follows:

1. Acquire m depth maps $D_1,..,D_m$ from viewpoints $V_1,..,V_m$ and n color images $C_1,..,C_n$ from viewpoints $U_1,..,U_n$.
2. Render each D_i as a triangle mesh from the view of the color camera closest to it to obtain new depth maps $E1,..,Em$, each corresponding to some color camera.
3. **Segmentation.** Segment each color image C_j into foreground and background objects using the geometry from the closest E_i as a guide.
4. **Geometry Completion and Smoothing.** Process each of E_i to fill holes and remove sensor noise while preserving depth discontinuities.
5. **Depth Map Fusion:** Fuse the filtered depth maps E_i to obtain a single complete depth map E for view V.
6. **Photo-consistent Blending:** Render the image for V by back-projecting E into each of U_i, and blending the colors obtained after testing for occlusion and checking photo-consistency between them.

In the following sections we elaborate in detail on each of these steps.

4.2. Segmentation

In order to restrict further processing to the relevant foreground objects, the color images are segmented into into (dynamic) foreground and (static) background regions. For each color image C_i we use a standard binary segmentation model, minimizing

$$E(\Gamma^i) = \int_{int(\Gamma^i)} \rho^i_{fg}\,dx + \int_{ext(\Gamma^i)} \rho^i_{bg}\,dx + \mathrm{Len}(\Gamma^i) \qquad (2)$$

where Γ^i is the (oriented) segmentation boundary, $int(\Gamma^i)$ is the interior region interpreted as foreground, and $ext(\Gamma^i)$ corresponds to the background region in the image. ρ^i_{fg} and ρ^i_{bg} are per-pixel negative log-likelihood functions indicating the preference of foreground or background, respectively. The (weighted) length of the segmentation boundary is used to regularize the result. This formulation, based on explicitly representing the segmentation boundary, may be replaced by an implicit one, minimizing:

$$\begin{aligned} E(u^i) &= \int_\Omega \rho^i_{fg} u + \rho^i_{bg}(1-u)\,dx + TV_{g^i}(u^i) \\ &= \int_\Omega (\rho^i_{fg} - \rho^i_{bg}) u^i\,dx + TV_{g^i}(u^i) + const \qquad (3) \end{aligned}$$

where $u^i : \Omega \to \{0,1\}$ is a binary function indicating whether a pixel in C_i belongs to the foreground ($u^i(x) = 1$) or to the background ($u^i(x) = 0$). $TV_{g^i}(u^i)$ is the weighted total variation equivalent to the weighted length of Γ^i in the explicit formulation. This minimization problem can be solved e.g. by the graph cut method, but we prefer a GPU-friendly approach [ZSN09] for more efficient run-time performance. The log-likelihood functions ρ^i_{fg} and ρ^i_{bg} incorporate color and depth similarities between the current image and the reference background image. We use

$$\rho^i_{bg} = \lambda_{color}\|W_{YUV}(YUV_{cur} - YUV_{bg})\|_1 + \lambda_{depth}|d^i_{cur} - d^i_{bg}|$$
$$\rho^i_{fg} = \lambda_{color} bias_{color} + \lambda_{depth} bias_{depth},$$

where $\lambda_{color}, \lambda_{depth}$ are weighting constants ($\lambda_{color} = \lambda_{depth} = 5$ in our setting). Our model for background color similarity assumes a heavy-tail Laplacian distribution of color differences in YUV color space, $\|W_{YUV}(YUV_{cur} - YUV_{bg})\|_1$. In order to suppress false positive foreground detections due to shadows cast by the foreground objects, the matrix $W_{YUV} = \mathrm{diag}(1/4, 1, 1)$ reduces the influence of the Y (luminance) component. Similarly, a heavy-tail distribution is used to model deviations from the reference background depth. Since we do not explicitly maintain a probabilistic model for the foreground, we use a uniform likelihood corresponding to the bias values ($bias_{color} = 1/20$ and $bias_{depth} = 1/20$). We use a weighted smoothness term, $TV_{g^i}(u^i)$, to enable alignment of the silhouette with dominant edges in the color image, where $g^i = 1/(1 + 80\|\nabla C^i\|^2)$, after C_i has been normalized to the range $[0,1]^3$.

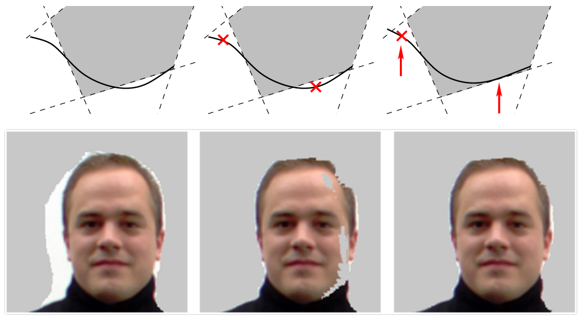
Figure 3: *Top: Geometry correction and clipping with respect to the (gray) visual hull. Bottom: (Left) Virtual view from the middle camera without any processing. Silhouette does not match. (Center) Clipping geometry outside the visual hull (red x in diagram). Silhouette now matches but it has artifacts. (Right) Result with some geometry "pushed" into the visual hull (red arrow in diagram), and clipped otherwise.*

4.3. Geometry Completion

The depth maps obtained from the Kinect sensor are usually missing data near depth discontinuities and on specular surfaces. They also exhibit artifacts corresponding to the pixel quantization of the disparities. Further, mostly due to the interference between the two Kinect systems, the returned depth map may contain outliers. The reliability of the data is particularly low near a depth discontinuity, as can be seen in Fig 2. Consequently, the raw depth data requires processing that should complete the missing data and filter it, taking the noise characteristics of the sensor into account, while preserving depth discontinuities. Geometry completion is restricted to the segmented foreground region (see Section 4.2). Let Ω_d^i be the set of pixels in color image C_i with depth values $E_i(x)$. We solve for depth values $\hat{E}_i(x)$ minimizing

$$F(\hat{E}_i) = \int_{\Omega_{fg}} g_i(x) \|\nabla \hat{E}_i(x)\| \, dx \tag{4}$$

subject to $\hat{E}_i(x) = E_i(x)$ in $\Omega_{fg}^i \cap \Omega_d^i$. We use the same weighting function g_i as used for the foreground segmentation to guide regularization at depth discontinuities. To solve this, we use a quadratic relaxation and alternating minimization approach similar to [AGCO06]. We are aware that Eq. 4 can be minimized exactly, but found that the smoothing effect induced by the quadratic relaxation yields visually better results. Enforcing the geometry to be consistent with the silhouette constraint with respect to one color image is not sufficient. Figure 3 illustrates how independent depth map completion can lead to inaccurate reconstructions. Even if the geometry perfectly matches the silhouettes as seen in the two extreme cameras, this might not be the case for the central camera. The resulting image exhibits artifacts due to incorrectly reconstructed geometry. We address this problem by adding an additional constraint that all points must lie inside the visual hull. This can be done at minimal computational cost by projecting the pixels into the color cameras and checking if they are contained in the segmented interior. If not we move the depth pixels backwards up to 10cm along the lines of sight until they are inside the silhouettes of all the color cameras C_i. Geometry still outside the visual hull is not rendered (i.e. clipped). Figure 3 (right) shows a result of this correction technique. After this processing we replace the old depth maps E_i with the new ones $\hat{E}_i$.

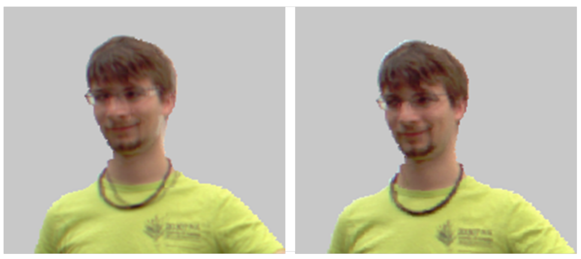
Figure 4: *Photo-consistent blending. (Left) Basic rendering using angle-based weights, as in [BBM*01]. Note the ghosting of the necklace and general blurring. (Right) FreeCam photo-consistent blending.*

4.4. Depth Map Fusion

At this point we have complete depth maps $E_1,..,E_m$. The goal is now to merge these into one depth map, as seen from the novel viewpoint V, while filtering out noise in the data and correcting geometric alignment errors due to inevitable calibration errors. We warp again each of the E_i to viewpoint V by rendering each E_i as a triangle mesh and averaging the resulting (possibly multiple) depth values per pixel to obtain a depth image E'. Alas, if the input data is of questionable quality, the result will be no better, and probably even worse due to poor alignment of the maps, resulting in significant rendering artifacts. Therefore, we merely smooth the new depth map E' and compensate for the remaining geometry inaccuracies in the blending stage.

4.5. Photo-consistent Blending

Once an estimate for the depth of a pixel of V is obtained, this may be back-projected into each of the color images $C_1,..,C_k$ where it is visible, and the color of the pixel looked up there. In an ideal setup where the estimate is accurate, the cameras are perfectly calibrated and there are no specular surfaces, the k colors obtained should be identical. In practice, though, none of these assumptions are true and the colors of the back-projected pixels can have a significant variance. The standard "basic" solution to this is to perform a visibility test using projective texturing to determine which cameras see the respective point and blend accordingly [BBM*01] the available back-projected pixels. However, as observed in [TZCZ09,EDDM*08], blending will often blur the image. Furthermore, due to errors in geometry, particularly around depth discontinuities, the visibility test is often unreliable and will result in blending pixels that have very different colors (Fig. 5 Left). Following Eisemann et al. [EDDM*08], we propose an approach for blending that uses photo-consistency to search for *b*etter pixels to blend, avoiding blending pixels that are too different, thus keeping the image sharp. We also propose a more robust occlusion test that does not rely only on the potentially erroneous geometry, but also on color consistency.

More precisely, the following algorithm is run for every point of the depth map E':

1. **Standard occlusion test.** Using precomputed visibility maps to determine the k cameras where this 3D point is visible.
2. **Select blending cameras.** if $k = 0$, render the background color. If $k = 1$, render the pixel with the color from this camera. If $k > 1$, select the two closest color cameras C_1 and C_2. Denote the color of the relevant pixels in these two images by c_1 and c_2.
3. **Color correction.** To compensate for geometric and calibration error and to avoid blurring artifacts, search in a (4×4) neighborhood of the pixels in the two color images for a better block match. Denote these new color values by c'_1 and c'_2. This is illustrated in Fig. 4.
4. **Robust occlusion detection.** If c'_1 and c'_2 are similar (using a perceptually linear color space, blend these values using weights proportional to the angle between V and C_1 and C_2 respectively. If not, this photometric discrepancy can be caused by either: A) an unreliable occlusion test or B) misaligned or incorrect geometry. In either case blending will produce artifacts, therefore select only one of the two. The key is to differentiate between A) and B) in order to select the correct color. To accomplish this we perform a more conservative occlusion test by applying a min-convolution filter to the visibility maps. If only one camera survives this tests, this is a strong indication that the pixel is near a depth discontinuity and that the initial occlusion test was faulty. Therefore, select the color from the remaining visible camera. If both cameras survive or fail the test there is no additional information that can be used so we select the camera closer (by angle) to the virtual view. This is illustrated in Fig. 5

5. Results and the Video

Parallel implementation of our rendering algorithm is a key to interactive-rate performance. Luckily, much of the algorithm is amenable to GPU parallelization, programmed using GLSL. Our current implementation runs at 7Hz output at 1024x768 pixel resolution. The current version of FreeCam does not employ multi-core CPU parallelization at all, so we expect further speedups to be possible in the future.

It is difficult to evaluate the performance of any free-viewpoint video system, including FreeCam, without seeing it in live action. Rather than describe at length the various pros and cons of the FreeCam output, we encourage the reader to simply view the video accompanying this paper, which demonstrates live recordings from FreeCam. A selection of still images from these sequences and others are reproduced in Fig. 6. The images are taken from a variety of virtual views that sample the entire spectrum of camera positions. The top row is the rendering of the raw geometry using for the color a blending of the two closest cameras weighted by the distance to the virtual camera by angle. The bottom row is the rendering using our FreeCam system. Note that the the sequences in the video are rendered at 24 f/s.

6. Limitations

FreeCam provides free- and multiple- viewpoint video at interactive rates from live feeds. While the quality of the resulting imagery is good, it is still not perfect, due to inevitable coverage issues. This is despite a number of heuristic “tricks” that are able to compensate for this.

The output frame rate on our system is still only 7Hz, but we have not invested much time on optimization, and are optimistic that we will be able to improve this by up to x4 by more careful optimization and by utilizing multiple CPU cores. An important hardware limitation of the Kinect systems is that they cannot be synchronized, so processing at high frame rates might encounter some difficulty in matching frames between sensors.

Calibrating the cameras in the FreeCam system is relatively straightforward. However, any change in the relative positions of the cameras requires the extrinsic calibration - a tedious process - to be performed again. Fortunately, in a realistic setup, the cameras are fixed to a rigid rig, which may be moved without disturbing the relative camera positions.

7. Discussion

FreeCam provides live free-viewpoint video at interactive rates using a small number of off-the-shelf sensor components and quite standard computing power. It is one of the first free-viewpoint system to incorporate the new Kinect depth sensor, and we expect this trend to strengthen as the sensor evolves.

FreeCam operates at a favorable point in the speed/quality tradeoff space. It is not yet fully artifact-free and does not yet run at true real-time rates. However, it provides an excellent starting point for a number of other hardware and software optimizations which, according to our calculations, should ultimately achieve this goal. Should there be "spare time" for extra processing of each frame, this would probably be best invested in a 2D post-process which will meticulously remove the more obvious artifacts.

The current version of FreeCam operates on a purely per-frame basis, and has no memory whatsoever of previous frames. An obvious direction for future improvement would be to take advantage of temporal coherence, either "learning" the scene as it is viewed, or simply remembering a few

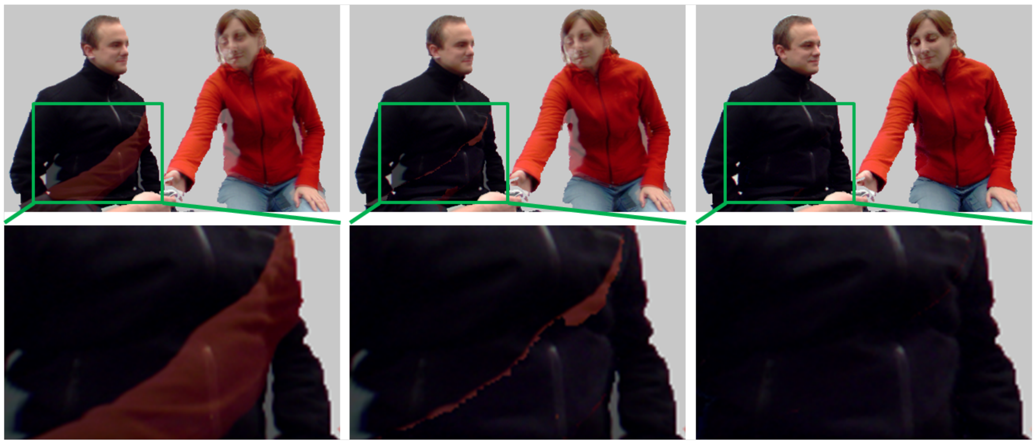

Figure 5: *Robust occlusion handling. (Left) Rendering without occlusion test. Note how the red jacket arm is blended into the black jacket. (Middle) Rendering with simple occlusion test without photo-consistency test. Red color still bleeds onto black. (Right) Rendering using our robust occlusion testing and photo-consistency testing. No red ghost remains.*

frames back, and using them as a reference to detect and remove artifacts.

It is still not clear what the optimal number of sensors is for FreeCam. As we have demonstrated, two Kinects and three color cameras provide good results, but it is possible that another camera of each type would be better, with carefully designed "surround" positioning to optimize coverage. This is an active area of future investigation.

Acknowledgments

This work was supported in part by the ETH Zurich Research Grant Program and by the BeingThere Centre, a collaboration of UNC Chapel Hill, ETH Zurich, NTU Singapore, and the Media Development Authority of Singapore. T. Popa was partially funded by NSERC.

References

[AGCO06] Aujol J.-F., Gilboa G., Chan T., Osher S.: Structure-texture image decomposition: Modeling, algorithms, and parameter selection. *IJCV 67* (2006), 111–136. 5

[BBM*01] Buehler C., Bosse M., McMillan L., Gortler S., Cohen M.: Unstructured lumigraph rendering. In *SIGGRAPH '01* (2001), ACM, pp. 425–432. 5

[BG08] Bogomjakov A., Gotsman C.: Reduced depth and visual hulls of complex 3D scenes. *Computer Graphics Forum 27*, 2 (2008). 2

[BM06] Bogomjakov A. C. G., Magnor M.: Free-viewpoint video from depth cameras. *VMV* (2006). 3

[CTMS03] Carranza J., Theobalt C., Magnor M., Seidel H.-P.: Free-viewpoint video of human actors. *ACM Trans. Graph. 22* (2003), 569–577. 2

[dST*08] de Aguiar E., Stoll C., Theobalt C., Ahmed N., Seidel H.-P., Thrun S.: Performance capture from sparse multi-view video. *ACM Trans. Graph. 27* (2008), 98:1–98:10. 3

[EDDM*08] Eisemann M., De Decker B., Magnor M., Bekaert P., De Aguiar E., Ahmed N., Theobalt C., Sellent A.: Floating textures. *Computer Graphics Forum 27*, 2 (2008), 409–418. 5, 6

[FB09] Franco J.-S., Boyer E.: Efficient polyhedral modelling from silhouettes. *IEEE Trans. on Patt. Anal. and Mach. Int. 31*, 3 (2009), 414–427. 2

[GFP08] Guan L., Franco J.-S., Pollefeys M.: 3D obj. reconstruction with heterogeneous sensor data. *3DPVT* (2008). 3

[GWN*03] Gross M., Würmlin S., Naef M., Lamboray E., Spagno C., Kunz A., Koller-Meier E. E., Svoboda T., Van Gool L., Lang S., Strehlke K., Moere A. V., Staadt O.: Blue-C: a spatially immersive display and 3D video portal for telepresence. In *SIGGRAPH* (2003), pp. 819–827. 2

[HA09] Hahne U., Alexa M.: Depth imaging by combining time-of-flight and on-demand stereo. In *Dynamic 3D Imaging* (2009), pp. 70–83. 3

[IZB07] Irschara A., Zach C., Bischof H.: Towards wiki-based dense city modeling. In *Workshop on Virtual Representations and Modeling of Large-scale Environments* (2007). 3

[KNR95] Kanade T., Narayanan P., Rander P.: Virtualized reality: concepts and early results. In *IEEE Workshop on Representation of Visual Scenes* (1995), pp. 69 –76. 1, 2

[KTD*09] Kim Y. M., Theobalt C., Diebel J., Kosecka J., Miscusik B., Thrun S.: Multi-view image and ToF sensor fusion for dense 3D reconstruction. In *3DIM* (2009). 3

[KVLB08] Kurillo G., Vasudevan R., Lobaton E., Bajcsy R.: A framework for collaborative real-time 3D teleimmersion in a geographically distributed environment. In *International Symposium on Multimedia* (2008), pp. 111–118. 2

[Lau94] Laurentini A.: The visual hull concept for silhouette-based image understanding. *PAMI 16* (1994). 2

[LMS04a] Li M., Magnor M., Seidel H.-P.: Hardware-accelerated rendering of photo hulls. *Computer Graphics Forum 23*, 3 (2004), 635–642. 2

[LMS04b] Li M., Magnor M., Seidel H.-P.: A hybrid hardware-accelerated algorithm for high quality rendering of visual hulls. In *Graphics Interface* (2004), pp. 41–48. 2

[MBR*00] Matusik W., Buehler C., Raskar R., Gortler S. J., McMillan L.: Image-based visual hulls. In *SIGGRAPH* (2000), pp. 369–374. 2

[MP04] Matusik W., Pfister H.: 3D TV: a scalable system for real-time acquisition, transmission, and autostereoscopic display of dynamic scenes. In *SIGGRAPH* (2004), pp. 814–824. 2

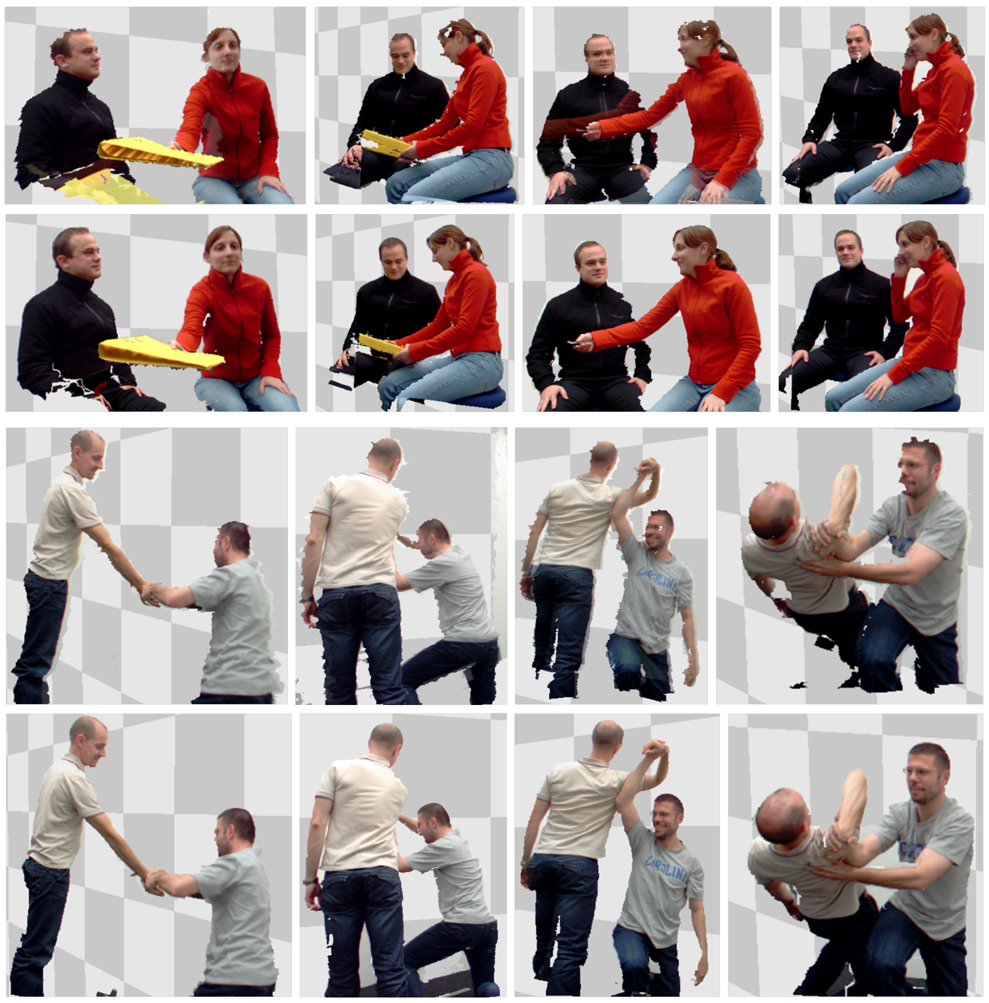

Figure 6: *Odd rows: novel viewpoints rendered without any processing (but with weighted blending). Even rows: novel viewpoints rendered by FreeCam.*

[PCD*97] Pulli K., Cohen M., Duchamp T., Hoppe H., Shapiro L., Stuetzle W.: View based rendering: Visualizing real objects from scanned range and color data. *Proc. EG Workshop on Rendering* (1997). 2

[PLM*10] Petit B., Lesage J.-D., Menier C., Allard J., Franco J.-S., Raffin B., Boyer E., Faure F.: Multicamera real-time 3D modeling for telepresence and remote collaboration. *Intern. Journ. of Digital Multi. Broadcasting* (2010). 2

[SH07] Starck J., Hilton A.: Surface capture for performance-based animation. *IEEE Computer Graphics and Applications 27* (2007), 21–31. 3

[SSH03] Slabaugh G. G., Schafer R. W., Hans M. C.: Image-based photo hulls for fast and photo-realistic new view synthesis. *Real-Time Imaging 9*, 5 (2003), 347–360. 2

[TZCZ09] Tola E., Zhang C., Cai Q., Zhang Z.: Virtual view generation with a hybrid camera array. *Technical Report EPFL/CVLA* (2009). 3, 5

[WWC*05] Waschbüsch M., Würmlin S., Cotting D., Sadlo F., Gross M.: Scalable 3D video of dynamic scenes. *The Visual Computer 21* (2005), 629–638. 2

[WWG07] Waschbüsch M., Würmlin S., Gross M.: 3D video billboard clouds. *Comp. Graphics Forum 26*, 3 (2007). 2

[YWB02] Yang R., Welch G., Bishop G.: Real-time consensus-based scene reconstruction using commodity graphics hardware. *Pacific Conference on Computer Graphics and Applications* (2002), 225. 2

[ZKU*04] Zitnick C. L., Kang S. B., Uyttendaele M., Winder S., Szeliski R.: High-quality video view interpolation using a layered representation. *SIGGRAPH 23* (2004). 2

[ZSN09] Zach C., Shan L., Niethammer M.: Globally optimal Finsler active contours. In *Pattern Recognition* (2009), vol. 5748 of *LNCS*, pp. 552–561. 4

[ZWYD08] Zhu J., Wang L., Yang R., Davis J.: Fusion of ToF depth and stereo for high accuracy depth maps. In *CVPR* (2008). 3