The Visual Computer
https://doi.org/10.1007/s00371-020-01823-7

ORIGINAL ARTICLE

A low-cost, practical acquisition and rendering pipeline for real-time free-viewpoint video communication

Sverker Rasmuson[1] · Erik Sintorn[1] · Ulf Assarsson[1]

Abstract
We present a semiautomatic real-time pipeline for capturing and rendering free-viewpoint video using passive stereo matching. The pipeline is simple and achieves agreeable quality in real time on a system of commodity web cameras and a single desktop computer. We suggest an automatic algorithm to compute a constrained search space for an efficient and robust hierarchical stereo reconstruction algorithm. Due to our fast reconstruction times, we can eliminate the need for an expensive global surface reconstruction with a combination of high coverage and aggressive filtering. Finally, we employ a novel color weighting scheme that generates credible new viewpoints without noticeable seams, while keeping the computational complexity low. The simplicity and low cost of the system make it an accessible and more practical alternative for many applications compared to previous methods.

Keywords Free-viewpoint video · 3D acquisition · Stereo reconstruction

1 Introduction

In recent years, there has been a revival for true 3D display environments such as head-mounted displays for virtual reality (VR) and augmented reality (AR), multi-view displays and holographic displays. Many new systems aimed for both consumer and professional use have reached the market, which has sparked an interest in content creation for these platforms, both with regard to entertainment, but also for, e.g., instruction videos and telepresence applications [8,19].

A lot of research has been done to solve this multifaceted problem of capture, 3D reconstruction, streaming and rendering. Several high-quality end-to-end systems for producing such content have been presented, with convincing results for many applications such as free-viewpoint video (FVV) and telepresence.

These systems, however, have the problem that they require huge, bulky and expensive equipment to be feasible. This is true both in terms of capturing equipment, e.g., special cameras, but also in terms of the raw processing power needed. So even with the current momentum of VR and AR, no techniques have been presented that are accessible at consumer level for content production in real time.

One of the major difficulties in achieving real-time frame rates is the high cost of global surface reconstruction. As such, attempts have been made with different combinations of templates and artificial intelligence, but this of course restricts what type of scenes that can be captured, and it introduces problems handling dynamic content [26,27]. We circumvent this by proposing a very fast and high-quality reconstruction for a pair of cameras. When extended to multiple camera pairs, we achieve sufficiently good coverage that, in combination with aggressive filtering, no explicit surface reconstruction is needed.

Our ultimate goal is an affordable system which can scan and reconstruct from a novel viewpoint (e.g., another user's current HMD orientation) in real time. Thus, unlike previous work where quality has been the primary concern, the main responsibility of our system is to produce an acceptable image within 33 or 16 ms. In this work, we mainly use faces

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s00371-020-01823-7) contains supplementary material, which is available to authorized users.

✉ Sverker Rasmuson
sverker.rasmuson@chalmers.se

Erik Sintorn
erik.sintorn@chalmers.se

Ulf Assarsson
uffe@chalmers.se

[1] Chalmers University of Technology, Gothenburg, Sweden

Published online: 07 March 2020

Fig. 1 Novel reconstructed views of three different scenes are generated in under 15 ms, on affordable hardware, using our method. The two rightmost images show a comparison between our method (top) and using a KinectV2 (bottom)

for evaluation, but we do not make any assumptions about specific topologies. To overcome this challenging problem, the following difficulties must be addressed:

- At 30 ms per frame, we cannot afford the use of high-quality global mesh reconstruction algorithms (e.g., Poisson reconstruction [14]).
- With consumer grade cameras, overall image quality is much lower than in previous work.
- With consumer grade cameras, we do not have access to explicit camera synchronization.
- Since we want to allow for any topology, we cannot rely on modifying an existing mesh, nor using templates or other strong priors.
- We want good scene coverage and therefore must be able to handle multiple cameras recording from multiple viewpoints.

As such, the main contribution in this paper is a semi-automatic, real-time end-to-end pipeline for capturing and rendering, using only commodity hardware and a single desktop computer. The technical contributions in this pipeline include:

- Efficient automatic computation of geometric boundaries, based on the visual hull, for reduced search ranges.
- A novel, highly optimized, hierarchical view-space stereo matching algorithm that utilizes estimated normals for improved quality.
- Consistent colors in the novel viewpoint, by blending input using a screen-space distance-to-silhouette map.

The system captures six video streams from off-the-shelf webcams and reconstructs a novel viewpoint on a single desktop computer at real-time frame rates (Fig. 1).

2 Previous work

Free-viewpoint video and performance capture Collet et al. [8] implement a full end-to-end free-viewpoint video pipeline which achieves high-quality reconstruction for a number of scenes. They combine RGB, active stereo in IR and silhouette information with Poisson surface reconstruction [14] to compute high-quality meshes for each frame and track mesh deformations to handle topology changes. The results of their method are very compelling, but they use a high number of cameras and a large studio of specialized equipment. Their method is also computationally expensive; processing one frame on a machine with a dual 12-core processor and an AMD Radeon R9 200 GPU takes them 28.2 min.

Recently, there have been a number of real-time implementations capable of handling dynamic content satisfactory. A template-based approach by Zollhöfer et al. achieves high-quality reconstruction of deformable models in real time using a custom-built RGBD camera [27]. While compelling, this method uses only a single view and requires a template first to be acquired for each scene. This template model is fixed and cannot handle changing scene topologies.

Another strategy is to extend Kinect Fusion [12], a popular method for scanning static geometry, which fuses depth maps from a single Kinect sensor at a high frame rate into a volumetric signed distance field. This method handles dynamic scenes by using nonrigid volumetric fusion, where new depth frames are warped to a reference model in a nonrigid manner [18]. This approach does not need an explicit template but still has problems with changing scene topologies, since the method relies on a reference surface model captured at a single point in time.

Fusion4D [9] combines a volumetric approach with estimation of a nonrigid motion field between frames, as well as using active stereo for acquiring depth information.

This approach is state of the art in terms of quality and handles multiple views in real time. It also handles foreground/background segmentation without a green screen unlike Collet et al. [8]. However, it is still very computationally expensive and uses dedicated machines with 3.4GHz Intel Core i7 CPUs and two NVIDIA Titan X GPUs to generate each RGBD stream. In contrast, we produce 5 RGBD streams on a single machine in under 10ms.

Following and extending this approach, Holoportation[19] creates a room-sized reconstruction that allows for an immersive telepresence system. This system uses a similar approach as Fusion4D [9] but extends it to an interactive system, allowing users to communicate with each other in real time using AR or VR headsets. Again, this system is computationally expensive, requiring dedicated machines for each depth and color streams.

Other more lightweight telepresence systems try to solve the problem by using sets of depth cameras [17,24]. While attractive due to their simplicity, these methods do not scale well due to interference problems when using multiple units. There are also simpler methods that rely solely on visual hull for capturing geometry [20], in which speed and simplicity are traded for geometric fidelity. This method can only successfully capture the structure of convex objects.

Face reconstruction There exists a vast body of research dedicated to facial reconstruction. One appealing method uses depth cameras for real-time reconstruction, computing a dense correspondence field between the input image and a generic face model[13]. More recent methods use deep-learning-based approaches from single images [22,26]. Some of these approaches also achieve real-time speeds [10,25]. All of them, however, require strong priors and explicit or learned models of faces. One method without such priors uses passive stereo and achieves convincing results using an iterative stereo refinement approach [3]. This method is, however, far from real time, taking about 20 min per frame in their implementation.

Depth estimation Real-time 3D reconstruction has gained momentum since the advent of cheap RGBD cameras such as the Kinect. These devices are often compelling because they can produce depth maps at a high frame rate while being low-cost commodity devices. One common approach that these devices use is structured light, where a known pattern is projected onto a scene, and depth values are computed by analyzing how this pattern is deformed when striking surfaces. While giving accurate results, this method cannot be used in a multi-view setup, since the projected patterns interfere across devices. Another common strategy is using time-of-flight-based methods for depth estimation [11]. These cameras measure the time-of-flight of an emitted light pulse for each pixel of the camera, which can then be used to infer the depth since the speed of light is known. For our approach, this is not a feasible method because of the problem of using multiple devices (e.g., the KinectV2), and the inherent problem of multi-path interference, in which the device measures the depth incorrectly in corners.

Passive stereo methods use two RGB images to compute depth maps via triangulation. Popular methods that achieve high quality include PatchMatch Stereo [5], which has also been extended to real time [9,19,27]. One hierarchical stereo matching approach achieves high-quality face reconstruction using an iterative refinement scheme [3]. Another approach, using an iterative refinement method implemented using CUDA, achieves real-time performance, albeit with striping artifacts that would make it unusable for our application [15].

While this is a well-studied problem and there exist many compelling approaches, our system requires simultaneous reconstructions from 5 to 10 camera pairs to be completed within 16 or 30 ms on a single computer, which none of the above methods achieves while retaining an acceptable quality. As a trade-off, we also prefer quality over high resolution for the depth estimation.

3 System overview

In our system, we use a single desktop computer, equipped with an Intel i7-8700 CPU, 16 GB of system RAM and an NVIDIA GTX 980 GPU. We have six Logitech C922 webcams that are placed in an arc in front of the scene. These are placed so that each adjacent pair can be used for stereo matching, i.e., pairing camera one and two, two and three, three and four, etc., in total five match pairs. This exact number of cameras used is a compromise between runtime performance, ease of calibration, coverage and depth accuracy for our applications. There are, however, nothing specific about this number, and we have tested the system successfully with as little as three and up to eight cameras. We use a green screen and two diffuse light sources to help with background segmentation and uniform lighting, respectively. The cameras are calibrated once with a semiautomatic method and corrected for scale and can then be used for any motif.

The real-time pipeline consists of multiple stages (see Fig. 2) that start with image capture and end with a final view rendered from the current position of a user-controlled virtual camera. All major steps of the pipeline are computed on the GPU, using CUDA for the stereo reconstruction and OpenGL for the rest. We start the capture of the scene and preprocess the images by undistorting from the camera's distortion parameters, converting to grayscale, and applying background segmentation. We also compute the visual hull

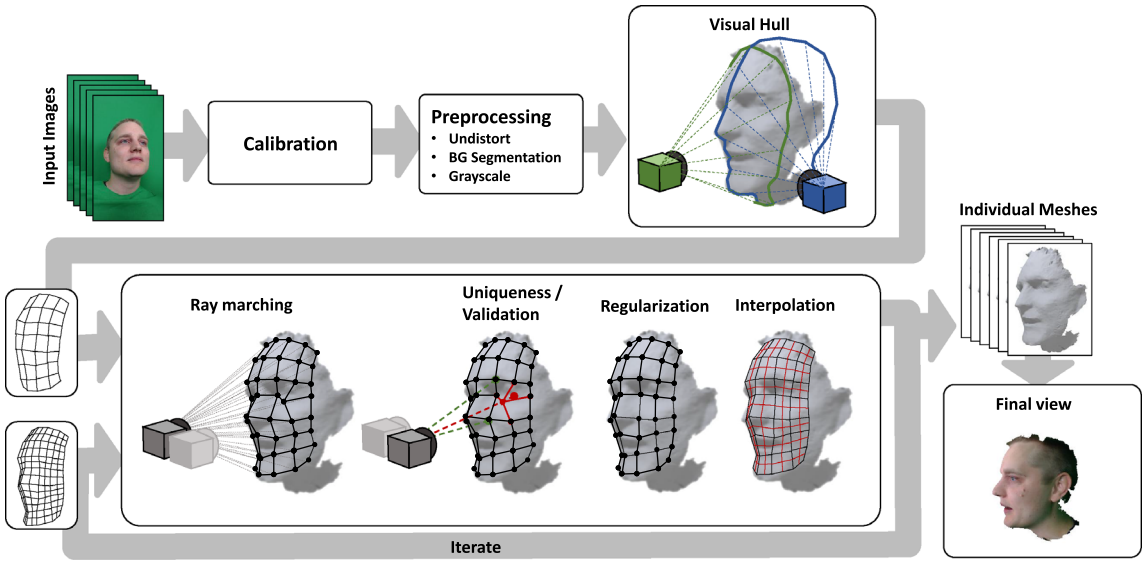


Fig. 2 An overview of our real-time pipeline. The corresponding silhouette boundary for the blue and green camera is marked in their respective colors. The silhouette boundaries from all cameras are used to compute the visual hull of the geometry. The gray camera pair depicted under raymarching and uniqueness/validation illustrates that we raymarch from one camera in each camera pair and then perform uniqueness/validation with the other camera in the pair. Output meshes are not merged explicitly but rendered from the final view where fragments are weighted together

of the geometry to constrain the search space for our stereo matching algorithm.

The geometry is computed pairwise for adjacent cameras, using a novel optimized view-space stereo matching algorithm, see Sect. 6. The algorithm is composed of multiple steps and is applied hierarchically in a coarse-to-fine measure (see Fig. 2). Early in the algorithm, we estimate geometry normals, which we use to orient our matching filters as well as for regularization and interpolation. We apply constraints to discard invalid parts of the geometry.

Position and normal buffers from the stereo reconstruction are used to create one mesh per camera pair. Finally, these meshes are rendered from the current virtual camera position. In this camera's screen space, we resolve which meshes are visible for each pixel and compute a final color using blending. This blending function uses two weights: one that represents the angle between the virtual camera and the recording cameras, and one that is sampled from a distance-to-mesh-silhouette map computed in screen space for the virtual camera, as discussed in Sect. 8.

4 Calibration

The system is calibrated using a semiautomatic method similar to Beeler et al. [3], using a calibration sphere and the aid of OpenCV [6]. After computing the approximate Euclidean frame, we, however, employ the Ceres Solver [2] instead for final bundle adjustment. We use a simple pinhole camera model with one coefficient of radial distortion. The output world coordinate system has its origo aligned with the center of the first camera used in the calibration.

5 Capturing and preprocessing

We use Logitech C922 webcams that are connected via USB 3.0 to a single machine equipped with USB 3.0 PCI Express breakout cards, to which each video feed is streamed in Full HD resolution at 30Hz. We use FFmpeg to capture the video streams directly into our pipeline, or to replay pre-recorded video streams. These FFmpeg instances run on separate CPU threads decoding video in the background. We use two different synchronization schemes depending on the source. For live video, we ensure that the incoming streams use small buffers and that the pipeline is emptied as fast as possible (possibly by dropping frames), so that we always get the latest incoming frames into our pipeline. For recorded streams, we synchronize the videos by using the embedded presentation time stamps for each frame. This synchronization will of course only be approximate, since the cameras do not have a global synchronization mechanism.

The cameras are configured to have automatic exposure disabled, to ensure comparable colors across cameras, and auto-focus disabled so as not to disrupt the calibration. For simplicity and higher quality, we employ a green screen to help with background/foreground segmentation, and use two diffuse light sources to get reasonably uniform lighting. In our examples with human motifs, we also use a green shirt to get a precise segmentation of the neck region.

Image preprocessing uploaded to GPU memory and all subsequent computation is performed on the GPU. This

leaves the CPU free to decode the video streams for the next frame and perform application logic. At the cost of a single frame of latency, we upload the this frame asynchronously while processing the current. The first step in preprocessing is undistortion of the captured images from the distortion parameters captured in the calibration step. Next, we use a standard chroma-key algorithm to get a high-quality foreground/background segmentation of the scene [7]. Finally, the images are converted to grayscale and a mipmap hierarchy per image is computed.

Visual hull generation We can significantly restrict the volume of where the potential geometry resides by utilizing our foreground segmentation and our wide-angle setup (see Fig. 3). For each camera, a surface is created, which consists of triangles with one vertex in the camera center and two vertices along the silhouette on the camera's far plane (see the left image Fig. 3). These triangles are the output of a geometry shader which is executed for each foreground pixel. Going anticlockwise around the pixel's 8-connected neighborhood, a triangle is output if both the current and the previous neighbor are considered background. The intersection of the created silhouette cones [16], the red area in the right image of Fig. 3, creates a boundary for the geometry which is commonly called the visual hull [16]. To compute this intersection efficiently for a specific camera, every other camera's silhouette cone (approximated by a triangle mesh) is rendered from it in turn to a depth map. As each of the other cameras is rendered in this manner, the new values are blended into the frame buffer, for each pixel retaining the smallest value (e.g., furthest away from the camera), effectively tightening the boundary around the geometry. This is then repeated for all cameras. This computed depth map can then be used as a starting position for the following stereo reconstruction, which both decreases the number of matches that need to be performed, as well as the risk of outliers in the stereo matching (the effect of which can be seen in Sect. 9.2).

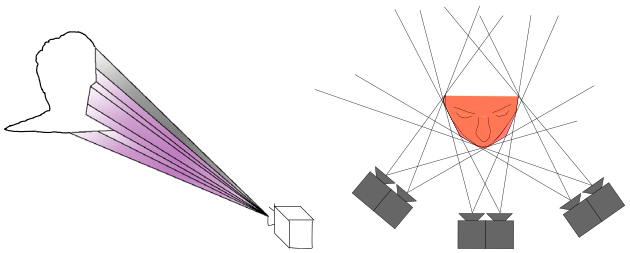


Fig. 3 To the left, we see how one camera's silhouette cone is constructed with triangles that extend from the camera center to the silhouette of the segmented foreground. To the right, the intersection of these surfaces, the red area known as the visual hull, creates a bounding volume for where the geometry must reside

6 Stereo reconstruction

The conventional method of doing local stereo matching is to use epipolar geometry and compute the matching over an NxN pixel window in a rectified image pair, i.e., two images where the image planes are parallel. This is done both as a simplification to the matching problem and as an optimization, since a view ray from the first image projects to a horizontal line in the second image. This greatly increases performance in CPU implementations since it leads to a much more cache-friendly data-access pattern. However, there are two major drawbacks with using image rectification. The first is the inevitable distortion of the rectified image under reprojection; the other is the implicit use of fronto-parallel matching windows (see Fig. 4). Popular techniques such as PatchMatch Stereo [5] have shown that there can be significant quality gains from using oriented matching windows. In this paper, we estimate normals that we can use to orient matching windows, as well as for geometry-aware interpolation and smoothing (see Sect. 6.4), and therefore skip rectification completely. This does not affect performance significantly in our GPU centric implementation (see Sect. 9.1), since GPUs rely much more on latency hiding than caches and since the GPU's texture cache is optimized for spatial locality.

For each camera pair, we use a hierarchical local stereo matching algorithm, loosely based on the stereo reconstruction by Beeler et al. [3]. However, to be able to easily estimate and utilize normal information, we instead do stereo matching by raymarching view rays from one camera in each pair and storing the results as view-space geometry buffers. The matching is divided both into multiple steps and in a hierarchical fashion, where the resolution is doubled in each dimension for each level of the hierarchy (see Fig. 2). Below, we start by describing each step for the first hierarchical level,

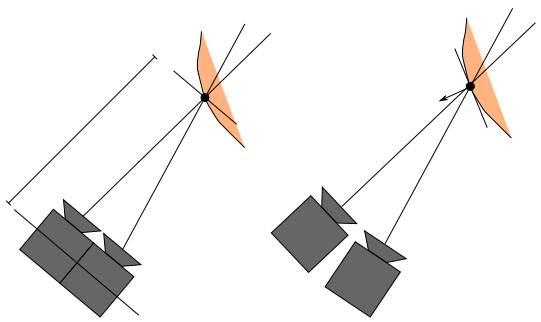


Fig. 4 In a rectified image pair (left), the images are reprojected so that the image planes are parallel. When sampling an NxN region in the images, this implies using a fronto-parallel matching window in world space, which may or may not be aligned to the underlying geometry. Using surface-oriented matching windows (right), there is no such restriction, and the matching window will more closely approximate the geometry

followed by a subsection of how higher hierarchical levels are handled.

6.1 Raymarching

In the first raymarching step, at hierarchical level zero, we shoot rays corresponding to a given *geometric resolution*. We march along each ray for a fixed length starting from the conservative estimate of the geometry constructed as described in Sect. 5. For each step along the ray, a rectangular filter is constructed in world space and projected on both cameras. Colors are sampled with mipmapping in the corresponding color textures, and the distance between both patches is computed using a mean-subtracted sum of absolute difference (SAD) cost function, where the distance D_{SAD} is computed as

$$D_{\text{SAD}} = \frac{1}{n}\sum_{i=0}^{n} |(p_i - \mu_p) - (q_i - \mu_q)|, \tag{1}$$

where n is the number of samples, p is each sample from the first image with corresponding mean value μ_p and q is each sample from the second image with corresponding mean value μ_q. This cost function is chosen since it is robust to local lighting conditions while still being cheap to compute, since we can sample an approximate mean value by using the mipmap hierarchy. Other more expensive cost functions were tested without any significant improvement in quality. The position with the best score along the ray is written to the position buffer.

6.2 Uniqueness

In the marching step, we do not have a threshold on the matching score and will always take the best matching point along our ray. This gives rise to invalid geometry in areas where we only have visibility for one of the cameras. In the second step, we address this problem by identifying these problematic areas by projecting the obtained position buffer from the first camera onto the second camera and then do stereo matching along *its* view rays for the projected points. If the best match along this ray does not closely match the best match from the first camera, the point is discarded. This method is similar (but not identical) to the uniqueness constraint employed by Beeler et al. [3].

6.3 Normals

Up until this step (for the first hierarchical level), we have not had any normal information for the geometry, and the filters used have been oriented with the z-plane of its corresponding camera. Now that we have obtained a first initial guess of the geometry, we can also estimate the normals. This is done using a simple average of computed normals using the cross product of neighbors of each position in the buffer.

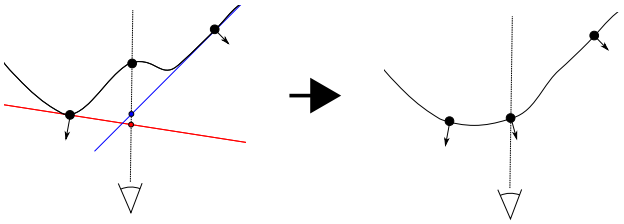


Fig. 5 The regularization algorithm looks at the intersection of the current view ray with planes, red and blue, constructed from neighboring positions and normals. The regularized position is then computed as an edge-preserving weighting of the ray/plane intersections

6.4 Regularization

The raw stereo matching is rather noisy, and we want to enforce that we are reconstructing a continuous surface. This is done in a regularization step, where the current view ray is intersected with planes constructed from each of the neighboring points and normals (see Fig. 5). An edge-preserving weighted average of these intersections is computed as a new position along the ray. With t_n being the parameter along the ray corresponding to an intersection between the current view ray r_v and a plane constructed with a neighboring point P_n and corresponding normal N, we have

$$t_n = \frac{-(P_n \cdot N)}{r_v \cdot N} \tag{2}$$

for each neighbor n. We can compute a weight w_n which is the inverse absolute distance between this point along the ray and the corresponding point along the ray for the current point P_0 as

$$w_n = \max\left(\epsilon, \frac{1}{|t_n - |P_0||}\right), \tag{3}$$

for some small value of ϵ. The new point P along the ray can then be computed as

$$P = \frac{P_0}{|P_0|}\frac{\sum_n w_n t_n}{\sum_n w_n} \tag{4}$$

for all neighboring positions.

In this step, we also do basic hole filling if a point has many valid neighbors, and pruning, if a point has very few valid neighbors. After this step, the normals are recomputed in the same manner as Sect. 6.3.

6.5 Interpolation

The last step of the stereo reconstruction algorithm is to interpolate positions and normals for the next level in the hierarchy. For performance, this is done in a separable algorithm, first horizontally and then vertically in the view-aligned geometry buffers. If there are two valid neighbors, the new point is the average of these two points projected on the current view ray. If there is only one valid neighbor, we use the intersection of the view ray with a plane constructed from the neighbor, similarly as in Sect. 6.4.

6.6 Higher levels of the hierarchy

The next levels of the hierarchy are executed in the same manner, doubling the geometric resolution for each iteration until the highest hierarchical level has finished. A few differences from the first hierarchical level should, however, be noted. We now have estimated normals in each step, and the filters used for stereo matching are thus oriented in the corresponding direction. We no longer use a fixed raymarch length for the raymarching, but the range is instead computed before starting stereo matching for the current hierarchical level. The range is computed as double the maximum distance from the current position to the intersection of the current view ray with planes constructed by all neighboring positions and their corresponding normals. This gives us a rough estimate of the curvature of the local geometry and adapts the search range accordingly. Finally, since the uniqueness step is very costly (equally expensive as the actual marching step), it is only performed in the first hierarchical level. This does, however, not lead to invalid points being introduced again, since these will not be rematched further up the hierarchy.

6.7 Temporal Hole Filling

Immediately following the stereo reconstruction, we employ a simple hole-filling scheme in our computed geometry buffers. We delay the pipeline with N frames and then look at N frames forward in time and N frames backwards in time, centered around the current frame. A value of $N = 5$ is used in this paper. If the current frame is missing a value but temporally surrounding frames are valid, we copy that nearest valid sample in time to the current frame. This has both a hole-filling effect and helps to mitigate flickering along silhouettes.

7 Mesh validation

For each camera pair, we now have view-space position and normal buffers. In offline algorithms [3,8], these buffers would be merged in a global surface reconstruction step, using, e.g., Poisson surface reconstruction [14], along with further refinement steps. This is, however, a very expensive procedure, with state-of-the-art methods at most running at interactive frame rates for the number of point samples that we employ [4,21].

For performance reasons, we thus opt for a much simpler approach. Each set of position buffers and normals is triangulated into meshes using a simple grid-based algorithm. Since we have a high amount of overlap between the pairwise computed meshes, we aggressively filter out potentially invalid triangles using two complementary methods.

The first method samples a number of view-space points over the surface of the triangle. These points are then used as input to our stereo matching algorithm using the SAD score. This is cheap since we only require a fixed set of points per triangle, but is also highly discriminating since we use the actual surface of our mesh for sampling.

In the second method, we remove triangles that are at an oblique angle with respect to the camera pair that they were constructed from. Here we assume that we have enough coverage so that another stereo pair will have a better view of the surface in question.

8 Final view generation

The quality of the meshes will reflect the quality of the stereo reconstruction and is thus dependent on well-textured areas that are visible in both cameras. The quality will generally be worst at silhouettes, where the geometry has a high angle toward the camera and the visibility for the whole projected matching filter is not guaranteed for both cameras. Following this argument, the heuristic we use is based on that the quality of the mesh is lower closer to its silhouettes.

Consequently, we need to compute the distance to the silhouette for each point in each mesh. Optimally, we would like to compute the actual geodesic distance along the mesh, but for performance reasons we instead compute the distance in pixels for the projected mesh. This computation is done in screen space with a GPU version of the flood-fill algorithm, called jump flooding [23]. We first compute the location of all silhouette points in a single pass by just looking at the validity of neighboring pixels for each pixel. The pixel locations of these silhouettes are then propagated to every other pixel in the projected mesh and stored in a distance map. If a propagated silhouette location is closer to the current pixel than what was previously written in the distance map, or if it did not have any previous value, it is updated with this new location. When finished, the distance map contains the screen-space coordinates of the silhouette closest to that particular screen-space location, for each projected pixel in the mesh.

This distance map is used to successfully weight colors and geometry together for the final view generation. The ultimate purpose of our pipeline is to generate a novel view of the scene from the perspective of a virtual camera controlled by the user. The final step is thus to compute this view using our reconstructed geometry, the set of aligned meshes and the captured camera images. The meshes are rendered to the current virtual camera position into separate geometry buffers. The colors and the geometry are then weighted together using the computed distance-to-silhouette map for each mesh.

Mesh visibility Since the meshes are overlapping, we need to decide which positions that belong to the same surface if we have fragments from more than one mesh rendered to a pixel. These fragments could belong to samples of the same surface, but could also belong to other surfaces now occluded by this front-most surface. In the geometry buffers for each mesh, we rendered both the closest front-facing surface but also the closest back-facing surface. If we sort our view samples according to depth, we can find if any such back-facing samples are in between two front-facing ones. In that case, we know that the subsequent samples surely belong to another part of the geometry and can be discarded. We also employ a simple threshold to discern between surfaces to help with situations where imperfect geometry prevents us from using this scheme. If the distance between two front-facing surfaces is greater than this threshold, they are considered different surfaces and only the closest is kept.

Color blending The fragments that have been deemed visible from the current view are now projected on to its two corresponding camera images, where the final colors are sampled. These colors, sampled from one or many camera pairs, are weighted together using a linear combination of two weights $\omega_{d,n}$ and $\omega_{v,n}$ for each sample n. The first weight $\omega_{d,n}$ is sampled directly from the previously computed distance map and ensures that we get smoothly varying colors on the overlapping edges between meshes. The second weight $\omega_{v,n}$ captures view dependence and is computed as

$$\omega_{v,n} = (v_{0,n} - v_{c,n}) \cdot \frac{v_{0,n}}{|v_{0,n}|}, \tag{5}$$

where $v_{0,n}$ is the view-space position of the fragment n and $v_{c,n}$ is its corresponding recording camera position. Using these weights for each color sample, the final color C can be computed as

$$C = \frac{\sum_n \phi\, \omega_{d,n} c_n + (1-\phi)\omega_{v,n} c_n}{\sum_n \phi\, \omega_{d,n} + (1-\phi)\omega_{v,n}}, \tag{6}$$

where ϕ is a parameter between 0 and 1 and c_n is each color sample.

Table 1 Final parameters used in the pipeline

General		
Numer of cameras	6	#
Number of camera pairs	5	#
Camera resolution	1920 × 1080	W × H
Stereo reconstruction		
Geometry resolution (level 0)	60 × 35	W × H
Geometry resolution (level 1)	120 × 70	W × H
Geometry resolution (level 2)	240 × 140	W × H
Raymarch length (level 0)	5	cm
Raymarch steps (level 0)	128	#
Raymarch steps (level 1)	32	#
Raymarch steps (level 2)	8	#
Match filter size (level 0)	10 × 10	mm
Match filter size (level 1)	5 × 5	mm
Match filter size (level 2)	2.5 × 2.5	mm
Match filter samples	9 (3 × 3)	#
Temporal hole filling		
Filter width	5	#

9 Results

We have recorded a number of scenes of people talking, turning their heads and touching their face with their hand, as well as one scene with a teddy bear (see Fig. 1). The final parameters used for our pipeline can be seen in Table 1. For more details, see the video in the supplementary material.

9.1 Performance

In Fig. 6, we see a performance breakdown of the main algorithmic steps of the pipeline, for 100 frames using five camera pairs, in the order that the pipeline steps are employed for each frame. The timings reported are the relevant OpenGL and CUDA times, since they totally dominate the pipeline.

Not included in this graph, but happening in the background on the CPU, we have one thread per camera fetching and decoding each incoming frame into an OpenGL Pixel Buffer Object. This frame is then uploaded to the GPU asynchronously using the dual copy engine feature of modern GPU's. By delaying the pipeline one frame, the updating of incoming frames to textures is done entirely in the background at no extra cost.

The visual hull generation step contains the creation of silhouette cones and the projection of this geometry onto each camera to constrain the search space for stereo reconstruction, as described in Sect. 5. The expensive part of this step is the rendering of each camera's silhouette cones from each other camera, since it has quadratic complexity with respect to the number of cameras.

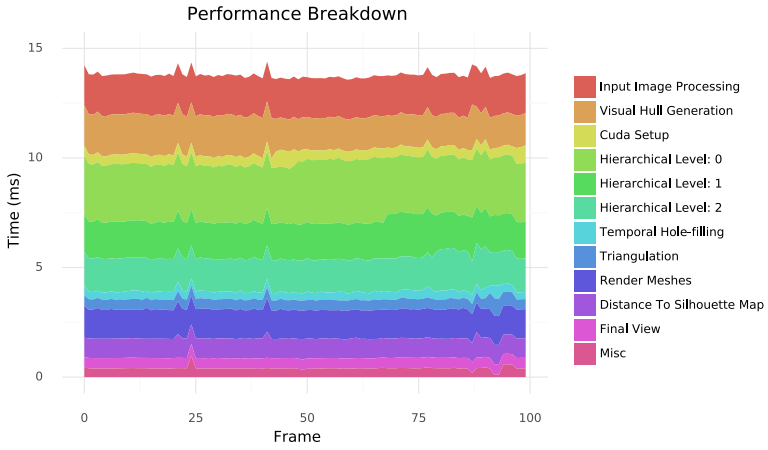


Fig. 6 Performance breakdown of the main steps of the proposed pipeline

Next we have the actual stereo reconstruction implemented in CUDA as described in Sect. 6, reported for each hierarchical level. We note that the difference in computation time between hierarchical levels is small and does not increase in proportion to the higher resolution, since we adjust the number of steps taken along the ray when going up the hierarchy. The stereo marching step is dominated by the high number of texture lookups required when matching filters and is thus limited by the available bandwidth to memory, how well the pipeline manages to hide the latency, and how well the caches are utilized. We use an implementation where we parallelize across raymarching steps to ensure that we have enough thread-level parallelism even for low resolutions. We also get a high utility of the spatial coherence of the texture cache with a hit rate of over 95%.

The temporal hole-filling step (see Sect. 6.7) is insignificant in terms of computation. However, if the filter width is W, it requires that the whole pipeline is delayed with a number of frames equal to $(W-1)/2$.

The last two steps before the final view generation are to triangulate meshes from the geometry buffers and to employ our triangle validation step, as described in Sect. 7. Both of these steps are fairly cheap since they only scale with the chosen geometric resolution.

The next three areas of the graph represent the time it takes to do final view generation as described in Sect. 8. This includes rendering the meshes to the final view, computing a distance to silhouette for each such projected mesh, and computation of mesh visibility and blending of sampled colors.

In general, we can see that the performance is stable and only varies slightly from frame to frame. We can also note that all graphs are well below the 30 Hz frame rate of the used webcams and could even be used with a 60 Hz source.

9.2 Reconstruction quality

As described in Sect. 6, our suggested mesh reconstruction algorithm is a combination of a number of techniques. Figure 7 shows how the results improve with each added technique. A first bare-bones implementation of our stereo reconstruction can be seen in Fig. 7a, where we simply march along the view ray for each pixel, in full resolution, stripping away everything else from the pipeline. The intersection of each camera's view frustum is used to constrain our search, as suggested by Beeler [3]. As can be seen in Fig. 7b, the results can be greatly improved by using the silhouette information available in the input videos to constrain the search region. This also reduces the number of raymarching steps required significantly. Next, in Fig. 7c, we see the results of applying regularization to the noisy results as described in Sect. 6.4. This smooths the results when neighboring matches are close to the true surface, but outliers will still cause disturbing artifacts. Therefore, we apply the uniqueness test described in Sect. 6.2, *prior* to regularization, to remove outliers (see Fig. 7d). In Fig. 7e, we show that averaging the results of the current frame with a few previous and upcoming frames will further improve the results.

At this point, the obtained matches are mostly close to the true surface, but, since only very local regularization has been performed, there are still many high-frequency artifacts where remaining erroneous matches are found. Figure 7f shows the results of instead reconstructing the mesh hierarchically. Here, a low-resolution mesh is constructed first, using all the steps described above. Then, higher-resolution meshes are obtained by iteratively repeating the process with a shorter search range around the results obtained from the previous hierarchical step. The hierarchical algorithm also means that we can get an estimation of the surface's normal which improves the results of raymarching and regularization in each step. The final resulting mesh from one camera pair is shown in Fig. 7g.

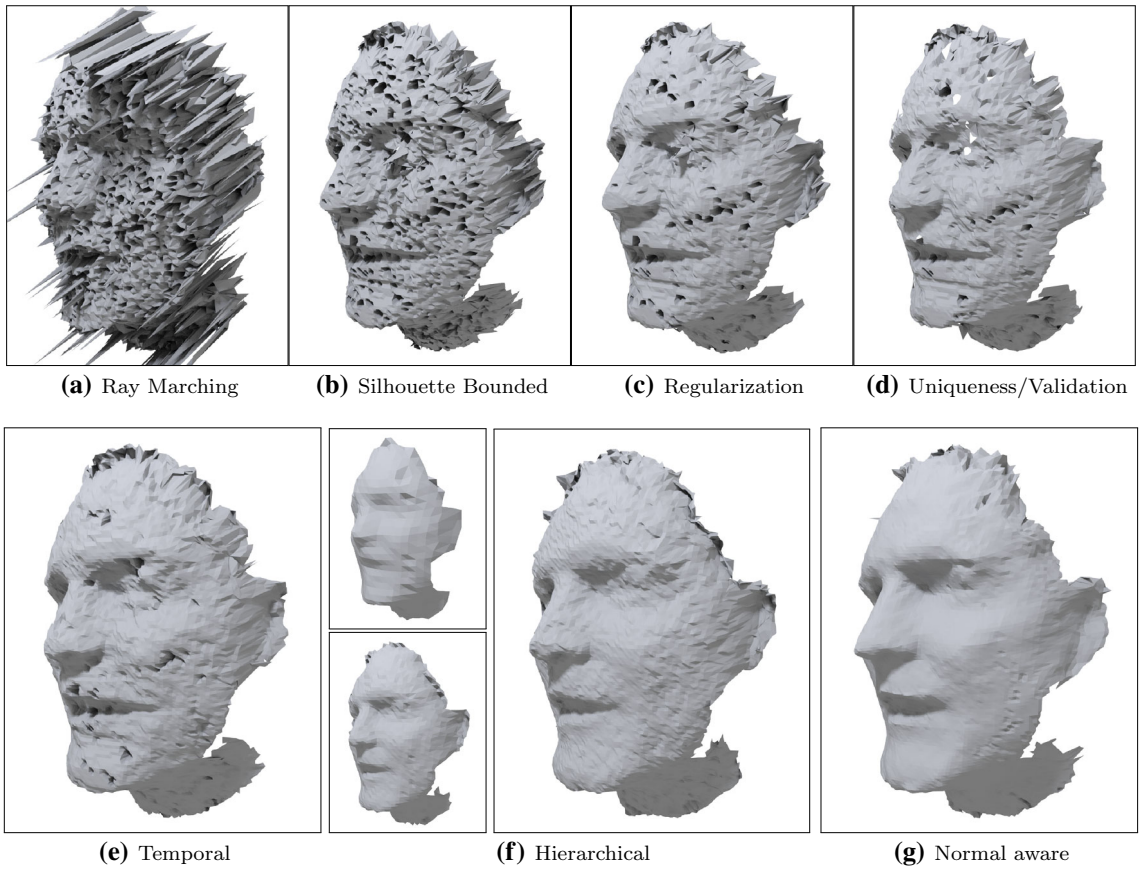


Fig. 7 The different steps of our mesh reconstruction. **a–e** Show the results of applying different steps immediately at full resolution. **f** Shows the benefit of applying the same steps hierarchically, and **g** shows the importance of using our estimated normals

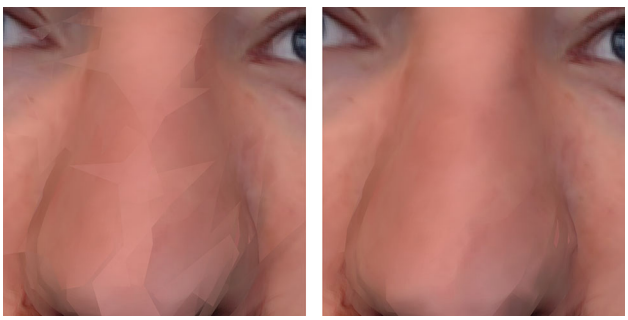


Fig. 8 Results of our color weighting scheme compared to a straight average. Left: Average, Right: Our scheme

Finally, as described in Sect. 8, we compute the final view using a color weighting scheme. The results of using this scheme as compared to just averaging colors can be seen in Fig. 8.

Our results compared to a KinectV2 using libfreenect2 [1] can be seen in Fig. 1. Apart from lower resolution in geometry and especially colors, the overall shape of the head and the nose looks distorted, possibly due to multi-path interference for the KinectV2. We have used one KinectV2 for this comparison, since using multiple KinectV2's is neither trivial nor officially supported.

Fig. 9 Our method (left, 20 ms) in comparison with a reference offline method (right, 20 min)

We have also made a comparison to a reference offline method by Beeler et al. [3] (see Fig. 9). In this comparison, we had access to a reconstructed mesh for each frame from the reference method, which we projected colors on using the associated video streams. An approximate visibility was

computed using depths maps of the mesh seen from each camera. Though they use seven cameras and quite a different setup than what we used, this did not prove to be of any problem to our pipeline.

The biggest difference in quality between our method and the reference method is found along the silhouettes, which are much cleaner in their method. We attribute this to that we lack the global surface reconstruction necessary to average out these types of errors.

Note that there is a huge difference in runtime performance between the two methods; our whole pipeline takes about 20ms per frame, while just their reconstruction is reported to take about 20 minutes per mesh with the exact same input.

10 Discussion and limitations

To limit the scope of this paper, we decided to focus our evaluation on human faces. With that in mind, there is nothing in this paper that is actually specific to this type of reconstruction. In the supplemented video, we show a working example of a scene with a teddy bear (see Fig. 1). We also claim that faces are a good candidate benchmark for this type of reconstruction, since humans are very sensitive to the appearance of faces; errors in proportion or geometry are very easily picked up and considered disturbing, especially in moving scenes. For consumer-level applications, it is also reasonable to believe that much content will contain humans and human faces.

While developing our pipeline, we soon realized that there are diminishing returns in how the resolution of the reconstructed geometry affects the perceived quality of the final view. As long as the quality, in terms of, e.g., accuracy, holes and smoothness of the mesh is high enough, the requirements on the de facto resolution seem to be moderate. We believe that the reason for this is that there are such strong shading cues in the color images that they totally dominate over the actual geometric fidelity when it comes to the perceived quality of the view. This reflects several design choices in our pipeline, notably the use of a view-space aligned matching algorithm in combination with normal-oriented filters. The reasoning for this is that the implicit fronto-parallel matching windows used in conventional stereo matching are a decreasingly valid approximation of the local geometry as the size of the matching windows in world space goes up. We have also tuned our CUDA implementation of the stereo marching to this by parallelizing across raymarching steps instead of using on thread per view ray, which would have been the natural way to do it.

This ensures that we can supply the GPU with enough active threads for successful latency hiding even for low resolutions. This is especially important in this case, since stereo matching is constrained by memory bandwidth in the form of texture lookups.

Currently, we do not have any method to know beforehand which mesh that has the highest quality at a certain overlap, and rely on filtering to remove as much as possible of poor reconstruction. Sometimes this method fails and we have problems with triangles with oblique angles that protrude in different directions. This, along with geometry along silhouettes which pop in and out of existence, can create disturbing flickering effects. This is a downside of not using global surface reconstruction.

11 Conclusions and future work

Although we do not achieve the same quality as, e.g., Fusion4D [9], we achieve usable free-viewpoint video in under 15ms with a fraction of the resources. Our simple setup enables almost anyone with a few webcams to be able to create content suitable for new medias such as AR and VR head-mounted displays and holographic screens. Since it is in real time, the content could be streamed in a live setting from a home computer or from a small studio.

Our pipeline manages to handle dynamic scenes reasonably well, even though the cameras used lack global synchronization. However, to improve how the pipeline handles fast-moving objects, it would be interesting to extend the stereo reconstruction to be temporally aware, by using, e.g., optical flow.

To further decrease the barrier of recording this type of content, it would be preferable if the pipeline could skip the step of using a green screen and also work robustly under natural lighting. These are for many applications solved problems but would of course increase the complexity and computational cost of the pipeline.

Another road is to optimize the pipeline for systems with limited resources. With only two camera pairs and low-resolution geometry, one could still obtain decent reconstruction, and further optimizations could possibly scale down the computational complexity, e.g., to allow for smartphones with multiple cameras to run the pipeline.

Acknowledgements Open access funding provided by Chalmers University of Technology. This work was supported by the Swedish Research Council under Grant 2014-4559.

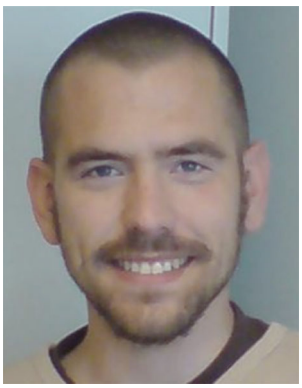
Compliance with ethical standards

Conflict of interest All authors have declare that they have no conflict of interest

References

1. libfreenect2. https://doi.org/10.5281/zenodo.50641
2. Agarwal, S., Mierle, K.: Others: Ceres solver. http://ceres-solver.org
3. Beeler, T., Bickel, B., Beardsley, P., Sumner, B., Gross, M.: High-quality single-shot capture of facial geometry. ACM Trans. Graph. **29**(4), 40:1–40:9 (2010). https://doi.org/10.1145/1778765.1778777
4. Berger, M., Tagliasacchi, A., Seversky, L.M., Alliez, P., Guennebaud, G., Levine, J.A., Sharf, A., Silva, C.T.: A survey of surface reconstruction from point clouds. Comput. Graph. Forum **36**(1), 301–329 (2017). https://doi.org/10.1111/cgf.12802
5. Bleyer, M., Rhemann, C., Rother, C.: Patchmatch stereo–stereo matching with slanted support windows. In: BMVC (2011). https://www.microsoft.com/en-us/research/publication/patchmatch-stereo-stereo-matching-with-slanted-support-windows/
6. Bradski, G.: The OpenCV Library. Dr. Dobb's Journal of Software Tools (2000)
7. Cannon, E.: Greenscreen code and hints. http://gc-films.com/chromakey.html
8. Collet, A., Chuang, M., Sweeney, P., Gillett, D., Evseev, D., Calabrese, D., Hoppe, H., Kirk, A., Sullivan, S.: High-quality streamable free-viewpoint video. ACM Trans. Graph. **34**(4), 69:1–69:13 (2015). https://doi.org/10.1145/2766945
9. Dou, M., Khamis, S., Degtyarev, Y., Davidson, P., Fanello, S.R., Kowdle, A., Escolano, S.O., Rhemann, C., Kim, D., Taylor, J., Kohli, P., Tankovich, V., Izadi, S.: Fusion4D: real-time performance capture of challenging scenes. ACM Trans. Graph. **35**(4), 114:1–114:13 (2016). https://doi.org/10.1145/2897824.2925969
10. Feng, Y., Wu, F., Shao, X., Wang, Y., Zhou, X.: Joint 3D face reconstruction and dense alignment with position map regression network. CoRR arXiv:1803.07835 (2018)
11. Hansard, M., Lee, S., Choi, O., Horaud, R.: Time-of-Flight Cameras: Principles. Methods and Applications. Springer, Incorporated (2012)
12. Izadi, S., Kim, D., Hilliges, O., Molyneaux, D., Newcombe, R., Kohli, P., Shotton, J., Hodges, S., Freeman, D., Davison, A., Fitzgibbon, A.: Kinectfusion: Real-time 3D reconstruction and interaction using a moving depth camera. In: Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology, UIST '11, pp. 559–568. ACM, New York, NY, USA (2011). https://doi.org/10.1145/2047196.2047270
13. Kazemi, V., Keskin, C., Taylor, J., Kohli, P., Izadi, S.: Real-time face reconstruction from a single depth image. In: 2014 2nd International Conference on 3D Vision, vol. 1, pp. 369–376 (2014)
14. Kazhdan, M., Bolitho, M., Hoppe, H.: Poisson surface reconstruction. In: Proceedings of the Fourth Eurographics Symposium on Geometry Processing, SGP '06, pp. 61–70. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland (2006). http://dl.acm.org/citation.cfm?id=1281957.1281965
15. Kowalczuk, J., Psota, E.T., Perez, L.C.: Real-time stereo matching on cuda using an iterative refinement method for adaptive support-weight correspondences. IEEE Trans. Circuits Syst. Video Technol. **23**(1), 94–104 (2013). https://doi.org/10.1109/TCSVT.2012.2203200
16. Laurentini, A.: The visual hull concept for silhouette-based image understanding. IEEE Trans. Pattern Anal. Mach. Intell. **16**(2), 150–162 (1994). https://doi.org/10.1109/34.273735
17. Maimone, A., Fuchs, H.: Encumbrance-free telepresence system with real-time 3D capture and display using commodity depth cameras. In: Proceedings of the 2011 10th IEEE International Symposium on Mixed and Augmented Reality, ISMAR '11, pp. 137–146. IEEE Computer Society, Washington, DC, USA (2011). https://doi.org/10.1109/ISMAR.2011.6092379
18. Newcombe, R.A., Fox, D., Seitz, S.M.: Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time. In: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2015)
19. Orts-Escolano, S., Rhemann, C., Fanello, S., Chang, W., Kowdle, A., Degtyarev, Y., Kim, D., Davidson, P.L., Khamis, S., Dou, M., Tankovich, V., Loop, C., Cai, Q., Chou, P.A., Mennicken, S., Valentin, J., Pradeep, V., Wang, S., Kang, S.B., Kohli, P., Lutchyn, Y., Keskin, C., Izadi, S.: Holoportation: Virtual 3D teleportation in real-time. In: Proceedings of the 29th Annual Symposium on User Interface Software and Technology, UIST '16, pp. 741–754. ACM, New York, NY, USA (2016). https://doi.org/10.1145/2984511.2984517
20. Petit, B., Lesage, J.D., Ménier, C., Allard, J., Franco, J.S., Raffin, B., Boyer, E., Faure, F.: Multicamera real-time 3D modeling for telepresence and remote collaboration. Int. J. Digit. Multimed. Broadcasting **2010**, 247108:1–247108:12 (2010)
21. Preiner, R., Mattausch, O., Arikan, M., Pajarola, R., Wimmer, M.: Continuous projection for fast l1 reconstruction. ACM Trans. Graph. **33**(4), 47:1–47:13 (2014). https://doi.org/10.1145/2601097.2601172
22. Richardson, E., Sela, M., Or-El, R., Kimmel, R.: Learning detailed face reconstruction from a single image. CoRR (2016). arXiv:1611.05053
23. Rong, G., Tan, T.S.: Jump flooding in GPU with applications to voronoi diagram and distance transform. In: Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games, I3D '06, pp. 109–116. ACM, New York, NY, USA (2006). https://doi.org/10.1145/1111411.1111431
24. Tan, F., Fu, C.W., Deng, T., Cai, J., Cham, T.J.: Facecollage: A rapidly deployable system for real-time head reconstruction for on-the-go 3D telepresence. In: Proceedings of the 25th ACM International Conference on Multimedia, MM '17, pp. 64–72. ACM, New York, NY, USA (2017). https://doi.org/10.1145/3123266.3123281
25. Tewari, A., Zollhöfer, M., Garrido, P., Bernard, F., Kim, H., Pérez, P., Theobalt, C.: Self-supervised multi-level face model learning for monocular reconstruction at over 250 Hz. CoRR (2017). arXiv:1712.02859
26. Tewari, A., Zollhöfer, M., Kim, H., Garrido, P., Bernard, F., Pérez, P., Theobalt, C.: Mofa: Model-based deep convolutional face autoencoder for unsupervised monocular reconstruction. CoRR (2017). arXiv:1703.10580
27. Zollhöfer, M., Nießner, M., Izadi, S., Rehmann, C., Zach, C., Fisher, M., Wu, C., Fitzgibbon, A., Loop, C., Theobalt, C., Stamminger, M.: Real-time non-rigid reconstruction using an rgb-d camera. ACM Trans. Graph. (2014). https://doi.org/10.1145/2601097.2601165

Sverker Rasmuson "Sverker Rasmuson is a Ph.D. student in the graphics research group at Chalmers University of Technology and holds a M.Sc. in Engineering Physics from Lund University. His research interests include free-viewpoint video, stereo reconstruction and 3D acquisition."



Erik Sintorn "Erik Sintorn is an assistant professor in Computer Graphics at Chalmers University of Technology. His main research topics are real-time global illumination, compression for rendering and efficient shadow rendering."

Ulf Assarsson "Ulf Assarsson is a professor in Computer Graphics and head of the graphics research group at the Department of Computer Science and Engineering, Chalmers University of Technology, Sweden. His main research interests include free-viewpoint video, voxel compression, real-time rendering, global illumination and GPU techniques. He is co-author of the book Real-Time Shadows."

A low-cost, practical acquisition and rendering pipeline for real-time free-viewpoint video communication

Sverker Rasmuson, Erik Sintorn & Ulf Assarsson

The Visual Computer
International Journal of Computer Graphics

ISSN 0178-2789

Vis Comput
DOI 10.1007/s00371-020-01823-7

Springer