Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®

Computers & Graphics 28 (2004) 3–14

COMPUTERS & GRAPHICS

www.elsevier.com/locate/cag

3D video fragments: dynamic point samples for real-time free-viewpoint video

Stephan Würmlin*, Edouard Lamboray, Markus Gross

Computer Graphics Laboratory, Computer Science Department, ETH Zurich, Zurich 8092, Switzerland

Abstract

We present 3D video fragments, a dynamic point sample framework for real-time free-viewpoint video. By generalizing 2D video pixels towards 3D irregular point samples we combine the simplicity of conventional 2D video processing with the power of more complex polygonal representations for free-viewpoint video. We propose a differential update scheme exploiting the spatio-temporal coherence of the video streams of multiple cameras. Updates are issued by operators such as inserts and deletes accounting for changes in the input video images. The operators from multiple cameras are processed, merged into a 3D video stream and transmitted to a remote site. We also introduce a novel concept for camera control which dynamically selects the set of relevant cameras for reconstruction. Moreover, it adapts to the processing load and rendering platform. Our framework is generic in the sense that it works with any real-time 3D reconstruction method which extracts depth from images. The video renderer displays free-viewpoint videos using an efficient point-based splatting scheme and makes use of state-of-the-art vertex and pixel processing hardware for real-time visual processing.

MSC: I.3.2; I.3.7

Keywords: Graphics systems; Three-dimensional graphics and realism

1. Introduction

Over the years, telepresence has become increasingly important in many applications including computer supported collaborative work (CSCW) and entertainment. While there are commercial solutions available for 2D teleconferencing in combination with CSCW, it is only in recent years that 3D video processing has been considered as a means to enhance the degree of immersion and visual realism of telepresence technology. The most comprehensive program dealing with 3D telepresence is the National Tele-Immersion Initiative (http://www.advanced.org/teleimmersion.html). Such 3D video processing poses a major technical challenge and is thus gaining interest in the computer graphics and computer vision communities. Here, a lot of research has been dedicated in particular to the extraction and reconstruction of real objects. The *representation* of 3D video streams, however, a fundamental prerequisite for efficient processing, has less intensively been investigated. In fact, most representations for 3D video streams are tailored for off-line postprocessing and, hence, share various limitations that makes them less practicable for advanced real-time 3D video processing.

Our work is devoted to the efficient representation, control and encoding of 3D video streams, facilitating sophisticated 3D rendering and visual effects. By introducing the concept of 3D video fragments, we generalize 2D video pixels towards irregular spatio-temporal point samples. Conceptually, each video fragment is a point sample with a set of attributes, like position, normal and color, which can be dynamically updated. A point-based object representation on the

*Corresponding author.
E-mail addresses: wuermlin@inf.ethz.ch (S. Würmlin), lamboray@inf.ethz.ch (E. Lamboray), grossm@inf.ethz.ch (M. Gross).

0097-8493/$ - see front matter © 2003 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cag.2003.10.015

remote site stores all active fragments and can be accessed for efficient rendering. Our differential update stream inserts, deletes or updates fragments on-the-fly in real-time. It exploits the spatio-temporal coherence of individual 2D video streams by inter-frame prediction of input changes in image space. Our prediction does not require expensive calculations like texture motion fields or 3D scene flows. While being conceptually lean and simple the presented approach effectively cuts down the number of expensive 3D shape computations. As opposed to mesh based representations, 3D video fragments provide a one-to-one mapping between points and associated color and normal attributes avoiding interpolation and alignment artifacts. In particular the lack of local connectivity makes 3D video fragments much more efficient for updating, coarse-to-fine sampling, progressive streaming, and compression.

Another benefit of retaining an underlying point based representation is graphics rendering. Since update operators of the 3D video stream dynamically change the representation, we have to carry out all necessary computations for rendering on-the-fly. Our 3D video renderer supports a wide range of visual effects, like explosions and warps, altering position and color attributes of individual fragments. To preserve data consistency we defer such operations to the final rendering stage and employ programmable hardware. By using a feedback loop which confines the number of active cameras we dynamically control the acquisition process and scale smoothly from view-dependence to view-independence. Moreover, virtual viewpoint and resolution-driven sampling allows smooth transitions between a subset of the reference cameras and adapts to bandwidth or processing bottlenecks. The method features efficient rendering from arbitrary spatio-temporal positions and supports multiple viewers.

Our 3D video pipeline is designed and optimized for real-time applications. During run-time, it performs fully automatically and does not need human intervention.

1.1. Related work

Given the design philosophy of our 3D video fragments pipeline the work that is most related to ours includes real-time 3D acquisition, the MPEG-4 standard, and point sample rendering.

Concepts for 3D video acquisition. There is a variety of methods for reconstruction of 3D video sequences. We distinguish between methods requiring off-line *postprocessing* and *real-time* methods. Examples of postprocessing algorithms include the work of Moezzi et al. [1], who propose a batch-oriented computation of 3D video sequences. Voxel representations are frequently derived by volume carving methods [2]. While these methods can provide for point sampled representations they are not performing in real-time. An appealing approach for utilizing spatio-temporal coherence for 3D video is the work of Vedula et al. [3] which computes a 3D scene flow for spatio-temporal view interpolation. It produces impressing results but the lack of real-time performance makes it impractical for our purposes. Carceroni and Kutulakos [4] present a dynamic surfel sampling representation and algorithm for estimation of 3D motion and dynamic appearance. However, they use a volumetric reconstruction for a small working volume and do not demonstrate real-time performance either. Würmlin et al. [5] present a 3D video recorder which stores a spatio-temporal representation in which users can freely navigate.

As opposed to post-processing approaches real-time methods are much more demanding with regard to computational efficiency. Matusik et al. [6] present an image-based 3D acquisition system which calculates the visual hull [7] of an object. It is build on epipolar geometry and outputs a view-dependent LDI representation. Their system neither exploits spatio-temporal coherence, nor is it scalable in the number of cameras. Similarly, polyhedral visual hulls [8] are based on epipolar geometry and provide view-independent rendering through a mesh and texture representation. It shares the same limitations and furthermore introduces interpolation artifacts due to improper alignment of geometry and texture, a common drawback of mesh-based methods. Kanade et al. [9] and Narayanan et al. [10] employ a triangular texture-mapped mesh representation. A similar approach was presented by Mulligan and Daniilidis [11] utilizing trinocular stereo depth maps from overlapping triples of cameras. It also features the aforementioned limitations of mesh based techniques. Pollard and Hayes [12] utilize depth map representations for novel view synthesis by morphing live video streams. This representation can suffer from inconsistencies between different views. In Gross et al. [13], a high-level overview of a similar real-time 3D video system is given.

Special-purpose hardware solutions for real-time depth estimation from video images, such as 3DV Systems' *ZCam*™ (http://www.3dvsystems.com), and Tyzx's *DeapSea* chips (http://www.tyzx.com) have recently become available. They can be seen as complementary to our work since they solve the problem of real-time 3D reconstruction and can be incorporated into our framework.

3D video standards. Even though the MPEG-4 committee is actively deliberating future 3D video standards, no standard for dynamic, free view-point 3D video objects has yet been defined [14]. The MPEG-4 multiple auxiliary components can encode depth maps and disparity information. But these are not complete 3D representations and possible shortcomings and artifacts due to DCT encoding and unrelated texture

motion fields and depth or disparity motion fields still need to be investigated. Our concept for real-time 3D video fits well into the system architecture proposed by the MPEG committee including acquisition, representation and display stages with back-channel transmission of viewpoint selection. Additionally, we support all types of interactivity, i.e., interaction at the encoder side and interaction with or without all data available at the decoder side.

Point sample rendering. In recent years points have experienced a renaissance as a graphics primitive. While there are various methods for fast and high quality rendering of point sampled geometry at our disposal, to date, none of them can efficiently cope with dynamically changing objects or scenes. For instance, the surfel system [15] samples an object with three orthogonal LDI's, building a so-called LDC-tree. It is well suited for progressive rendering but has to be rebuilt once the object changes. Surface splatting [16] extends the surfel system with a high-quality interactive software renderer which is based on a screen space formulation of the elliptical weighted average (EWA) filter adapted for irregular point samples. While Ren et al. [17] present a hardware-accelerated extension based on multi-pass rendering they all share the aforementioned limitations imposed by pre-processing and setup. Wand and Strasser [18] propose a multi-resolution point sample rendering algorithm for keyframe animations which can deal with highly complex scences but also relies on extensive preprocessing. Qsplat [19] is a progressive point sample system for representation and display of very large geometry. They represent static objects by a multi-resolution hierarchy of point samples based on bounding spheres. As with the surfel system they rely on extensive pre-processing for splat size and shape estimation making it impracticable for our needs. As we will discuss, our dynamic 3D video engine performs all computations for high quality point sample rendering on-the-fly in real-time.

1.2. Conceptual overview

Fig. 1 depicts a conceptual overview of the 3D video fragments pipeline. We acquire images from multiple calibrated video cameras. The images are processed to segment foreground from background. By means of dynamic camera control (Section 3) we determine a set of active cameras from which we generate 3D point samples as well as a set of supporting cameras delivering additional data to improve the 3D reconstruction. Using inter-frame prediction in image space we generate a stream of differential operators (Section 2) which dynamically update point sample attributes including position or color. We thus avoid to recompute the full 3D representation in each frame. The dynamic point samples are rendered by an efficient point splatting

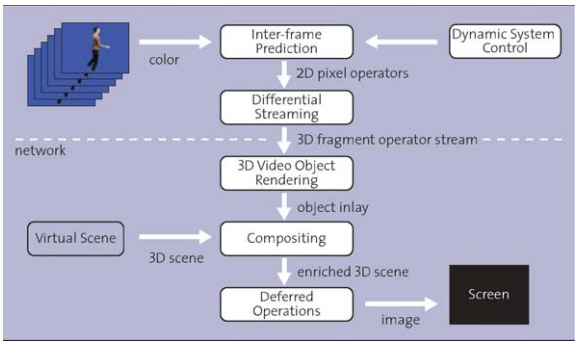

Fig. 1. Conceptual components of the 3D video fragments pipeline.

scheme (Section 4) and are composited with a virtual scene. In a final stage we apply deferred operations like visual effects by running them directly on the graphics hardware.

2. Differential 3D fragment operators

Our concept of 3D video fragments exploits the spatio-temporal interframe coherence of multiple input streams by using a differential update scheme for dynamic point samples. The basic primitives of this scheme are the 3D video fragments, point samples with different attributes like, e.g., a position, a surface normal vector, and a color. The update scheme is expressed in terms of 3D *fragment operators*, each of which is derived from a 2D pixel operator as illustrated in Fig. 2.

We distinguish between three different types of operators:

- INSERT adds new 3D video fragments into the representation after they have become visible in one of the input cameras. Insert operators are streamed coarse-to-fine as discussed in Section 3.
- DELETE removes fragments from the representation once they vanish from the view of the input camera.
- UPDATE corrects appearance and geometry attributes of fragments that are already part of the representation, but whose attributes have changed with respect to prior frames.

The time sequence of these operators creates a differential fragment operator stream that updates a 3D video data structure on a remote site. An INSERT operator results from the reprojection of a pixel with color attributes from image space back into three-dimensional object space. Any real-time 3D reconstruction method which extracts depth and normals from images can be employed for this purpose. Note that the point primitives feature a one-to-one mapping between depth and color/texture samples. The depth value is

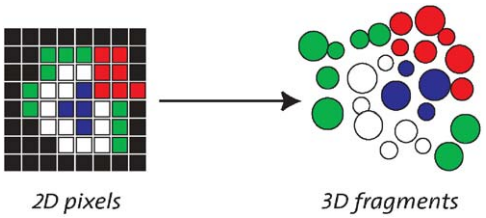

Fig. 2. Relationship between 2D pixel operators and 3D fragment operators.

stored in a *depth cache*. This structure accelerates the DELETE operator which performs a lookup in the depth cache and can thus be carried out very efficiently.

UPDATE operators are generated by all pixels which have been inserted into previous frames and which are still foreground pixels. They can be divided into three categories: The detection of *color changes* is performed during interframe prediction and leads to an UPDATECOL operator. UPDATEPOS operators take care of *geometry changes* and are analyzed on spatially coherent clusters of pixels in image space. We also use the depth cache for this purpose. We define independent blocks of points according to a predefined grid. For the 640×480 resolution, a block comprises 16×16 pixels. In each frame new depth values are calculated for the four-grid corners only, the explicit check of all pixels being computationally too expensive. If the differences to the previous depths exceed a threshold, we recompute 3D information for the entire block of points. Thus, our scheme proposes an efficient solution to the problem of uncorrelated texture and depth motion fields. Note that position and color updates can be combined to an UPDATEPOSCOL operator. All other candidate pixels for updates remain *unchanged* and no further processing is necessary. The 3D operators and associated data can be summarized as follows:

$$\begin{aligned}
&\text{INSERT} && : (\bar{\mathbf{p}}, \mathbf{c}) \rightarrow (\dot{\mathbf{P}}, \mathbf{c}, \mathbf{n}),\\
&\text{DELETE} && : (\bar{\mathbf{p}}) \rightarrow (\dot{\mathbf{P}}),\\
&\text{UPDATECOL} && : (\bar{\mathbf{p}}, \mathbf{c}) \rightarrow (\dot{\mathbf{P}}, \mathbf{c}),\\
&\text{UPDATEPOS} && : (\bar{\mathbf{p}}) \rightarrow (\dot{\mathbf{P}}_{old}, \dot{\mathbf{P}}_{new}, \mathbf{n}_{new}),\\
&\text{UPDATEPOSCOL} && : (\bar{\mathbf{p}}, \mathbf{c}) \rightarrow (\dot{\mathbf{P}}_{old}, \dot{\mathbf{P}}_{new}, \mathbf{n}_{new}, \mathbf{c}),
\end{aligned} \tag{1}$$

where $\bar{\mathbf{p}}$ are the coordinates of a pixel, $\mathbf{c}$ its color, $\dot{\mathbf{P}}$ the respective 3D coordinates of the point sample, and $\mathbf{n}$ its surface normal. The encoding of the 3D operators will be explained in Section 5.3.

We propose an image space inter-frame prediction mechanism which derives the 3D fragment operators from the original video images. We define two functions for pixel classification: A foreground-background segmentation defines a Boolean function $\mathrm{fg}(\bar{\mathbf{p}}, t)$ returning TRUE if the pixel $\bar{\mathbf{p}}$ is in the foreground at frame t. A second function $\mathrm{cd}(\bar{\mathbf{p}}, t, t')$ returns TRUE if the color difference of a pixel $\bar{\mathbf{p}}$ exceeds a certain threshold in between the time instants t and t'. These two functions allow us to assign to each pixel one of the following five classes using simple Boolean operations:

$$\begin{aligned}
&\mathrm{fg}(\bar{\mathbf{p}}, t) \wedge \mathrm{fg}(\bar{\mathbf{p}}, t-1) \wedge \neg \mathrm{cd}(\bar{\mathbf{p}}, t, t-1) && : \textit{colour unchanged},\\
&\mathrm{fg}(\bar{\mathbf{p}}, t) \wedge \mathrm{fg}(\bar{\mathbf{p}}, t-1) \wedge \mathrm{cd}(\bar{\mathbf{p}}, t, t-1) && : \textit{colour changed},\\
&\mathrm{fg}(\bar{\mathbf{p}}, t) \wedge \neg \mathrm{fg}(\bar{\mathbf{p}}, t-1) && : \textit{new},\\
&\neg \mathrm{fg}(\bar{\mathbf{p}}, t) \wedge \mathrm{fg}(\bar{\mathbf{p}}, t-1) && : \textit{expired},\\
&\neg \mathrm{fg}(\bar{\mathbf{p}}, t) \wedge \neg \mathrm{fg}(\bar{\mathbf{p}}, t-1) && : \textit{background},
\end{aligned} \tag{2}$$

where $\wedge$ denotes the Boolean AND and $\neg$ the Boolean NOT operator. Fig. 3 illustrates the image acquisition processing and depicts the five possible pixel states.

Finally, a *new* pixel invokes an INSERT operator, an *expired* pixel a DELETE operator and a *color change* an UPDATECOL operator. As previously described, *unchanged* and *color changed* pixels can nonetheless lead to an UPDATEPOS operator.

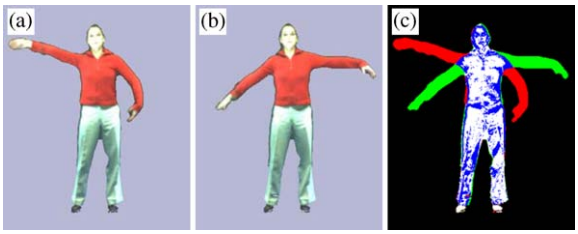

Fig. 3. 2D pixel operators: (a) silhouette at time $t-1$; (b) silhouette at time t; (c) pixel classification. Green indicates new pixels, red expired pixels, blue color changed pixels, white color unchanged pixels, and black background.

3. Dynamic system adaptation

Many real-time 3D video systems are employed for point-to-point communication. In such cases, the 3D video representation can be optimized for a single view point. Multipoint connections, however, require truly view-independent 3D video. In addition, 3D video systems can suffer from performance bottlenecks at all pipeline stages. Some performance issues can be locally solved, for instance by lowering the input resolution, or by utilizing hierarchical rendering. However, only the combined consideration of application, network and 3D video processing state leads to an effective handling of critical bandwidth and 3D processing bottlenecks. In the point-to-point setting the current virtual viewpoint allows one to optimize the 3D video computations by confining the set of relevant cameras. As a matter of fact, reducing the number of involved cameras or the

resolution of the reconstructed 3D video object implicitly decreases the required networking bandwidth. Furthermore, the acquisition frame rate can be adapted dynamically.

The aforementioned issues suggest a concept for dynamic system adaptation for the 3D video system, which will be described in this section.

3.1. Active camera control

We devise a concept for dynamic control of *active* cameras which allows for smooth transitions between subsets of reference cameras and efficiently reduces the number of involved cameras for 3D reconstruction. Furthermore, increasing the number of so-called *texture active* cameras enables a smooth transition from a view-dependent to a view-independent representation for 3D video.

A texture active camera is a reference camera applying the intra-frame prediction scheme as explained in Section 2. Each pixel classified as foreground in such a camera frame contributes color or texture samples to the set of 3D points in the 3D representation. Additionally, each camera might provide auxiliary information for the employed 3D reconstruction algorithm. We call the state of these cameras *reconstruction active*. Note that a camera can be both texture and reconstruction active. The state of a camera which does not provide data at all is called *silent*. Fig. 4 illustrates the dynamic control of active cameras.

In order to select the k-closest cameras for the desired viewpoint as texture active cameras, we compare the angles between all camera look-at vectors and the desired viewing vector. Choosing the k-closest views minimizes artifacts arising from occlusions in the reference views. Experimentally, we found that for our target objects, i.e. humans, $k = 3$ performs well. The selection of reconstruction active cameras has to be computed for all texture active cameras and is dependent on the employed 3D reconstruction method. Since our prototype system uses a shape-from-silhouette algorithm, each reconstruction active camera provides silhouette contours. The set of candidate cameras is chosen by two simple rules. First, the angles between a texture active camera and its associated reconstruction active cameras have to be smaller than some threshold, 100° in our setting. The candidate set is thus confined to cameras lying in approximately the same hemisphere. Second, the angle must not be smaller than 20°. Although this is hardly the case in our setup, cameras which are too close to each other only provide insignificant differences in their silhouette information. Optionally, we reduce the candidate set to a maximum size. We compute the angle between all candidate camera pairs and subsequently discard one camera of the closest pair. In our case, this scheme leads to an optimally smooth coverage of silhouettes for every texture active camera. The set of texture active cameras needs to be computed on-the-fly accounting for viewpoint changes. The map of corresponding texture and reconstruction active cameras can be pre-computed at system start-up time.

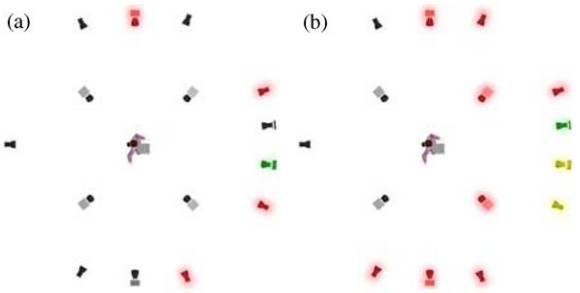

Fig. 4. Illustration of the dynamic camera control. Green cameras are texture active, red cameras are reconstruction active, and yellow cameras are both texture and reconstruction active. Uncolored cameras are silent: (a) for one active camera; (b) for three active cameras.

Although we did not investigate 3D reconstruction techniques other than shape-from-silhouette, our active camera approach is versatile and can be employed with other algorithms. A multi-view stereo algorithm for example requires from each reconstruction active camera the texture of the whole frame in combination with some features. In this case, the k-nearest cameras would be selected such to enable correspondence calculations.

Overall, the dynamic camera control allows to actively trade-off 3D reconstruction performance versus 3D video quality. In our shape-from-silhouette setting, the quality of the 3D reconstruction is improved by a growing number of reconstruction active cameras, but so increases the processing time. Currently, we use five reconstruction active cameras and thus five silhouettes for 3D reconstruction per texture active camera. Overall, this leads to an average of 10 reconstruction active cameras per frame.

3.2. Texture activity levels

A second strategy for dynamic system adaptation involves the number of reconstructed fragments. We define a *texture activity level* A_i for each camera i to determine the number of pixels fed into the 3D video pipeline. Initial levels for k texture active cameras are derived from the weight formulas for Unstructured Lumigraph Rendering [20,21].

$$r_i = \frac{\cos\theta_i - \cos\theta_{k+1}}{1 - \cos\theta_i}, \quad w_i = \frac{r_i}{\sum_{i=1}^{k} r_j}. \tag{3}$$

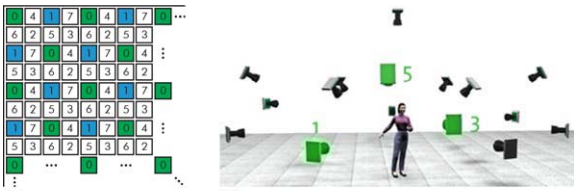

Fig. 5. Texture activity levels: (a) linear image pixel sampling pattern; (b) activity levels for three texture-active cameras as seen from the virtual viewpoint.

Here, r_i represent the relative weights of the closest k views. r_i is calculated from the cosine of the angles between the desired view and each texture active camera. w_i are the normalized weights which sum up to one. The texture activity level allows for smooth transitions between cameras and enforces epipole consistency. The resolution of the virtual view is taken into account with a factor ρ. In addition, texture activity levels are scaled with a system load penalty $penalty_{load}$ reflecting the reconstruction process. The penalty takes into account the load of the current frame and the activity levels of prior frames. If the load becomes too high, the texture activity level is reduced such that less pixels need to be processed. The following equation summarizes the texture activity level computation:

$$A_i = s_{max}\, w_i \rho - penalty_{load} \quad \text{with } \rho = \frac{res_{\text{target}}}{res_{\text{camera}}}. \tag{4}$$

Note that this equation is recomputed in each frame for each texture active camera. The maximum number of sampling levels s_{max} discretizes A_i to a linear sampling pattern in the camera image, allowing for coarse-to-fine sampling. All negative values of A_i are clamped to zero. Fig. 5a illustrates the linear pixel sampling pattern which is basically a multi-grid sampling, i.e. the index of each pixel in Fig. 5a defines the sampling level a pixel belongs to. Fig. 5b shows activity levels for a set of cameras for a given virtual viewpoint. Currently, we use NTSC-cameras which leads to 38,400 pixels per sampling level.

4. Dynamic point processing and rendering

The final stage of our 3D video pipeline constitutes the processing and rendering of the 3D video fragments. All necessary computations for rendering must be performed on-the-fly and in real-time. In particular, the size and shape of the splat kernels for high-quality rendering must be estimated dynamically for each point sample. For that purpose we propose a new data structure for 3D video rendering. We organize the point samples for processing on a per-camera basis similar to a depth image. However, instead of storing a depth value per pixel we store references to the respective point

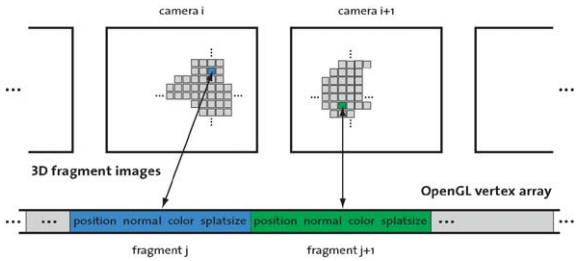

Fig. 6. 3D fragment images. Fragments can be referenced from an image space representation for point processing, but rendered efficiently from a flat OpenGL vertex array.

attributes. We name this representation *3D fragment image*. The point attributes themselves are organized in an OpenGL vertex array which can be directly transferred to graphics memory. With this representation we combine efficient insert, update and delete operations with efficient processing for rendering. Fig. 6 illustrates the data structure.

In addition, the 3D video renderer supports compositing with a virtual scene by Z-buffering. Finally, we also support deferred operations, such as 3D visual effects, which are applicable to the real-time 3D video stream without destroying the consistency of the data structure.

4.1. Local density estimation

For static objects, local point sample densities can either be estimated in a pre-processing step [19] or during the acquisition procedure [16,17]. In our approach however, the acquisition process leads to irregular 3D point sampling patterns. Hence, we cannot estimate the local point sample density during acquisition. Moreover, in a dynamic real-time system, it is not economical to maintain an advanced spatial search structure supporting point density estimations like nearest-neighbor searches [22]. Instead, we propose to estimate the local point sample density for each point based on incremental nearest-neighbor search in the 3D fragment image. The resulting neighbors are only approximations of the real neighbors, but they prove to be sufficiently close for local sampling density estimation. Our algorithm, which considers only two neighbors, uses the following heuristics. First, it calculates the nearest-neighbor N_1 of a given point in the 3D fragment image. Then we search for a second neighbor N_{60}, forming an angle of at least 60° with N_1. Our neighbor search needs approximately four more search iterations for finding N_{60}.

4.2. Point sample rendering

As in Ren et al. [17], we render the point samples as polygonal splats with a semi-transparent alpha texture

using a two-pass algorithm. In the first pass, opaque polygons are rendered for each point sample and thus visibility splatting is performed [15]. The second pass renders the splat polygons with an alpha texture. The splats are multiplied with the color of the point sample and accumulated in each pixel. A depth test with the Z-buffer from the first pass resolves visibility problems during rasterization. This ensures correct blending between the splats.

The neighbors N_1 and N_{60} can be used for computing polygon vertices of our splat in object space. The splat lies in the plane which is spanned by the coordinates of the point p and its normal n. We now distinguish between circular and elliptical splat shapes. In the first case, all side lengths of the polygon are twice the distance to the second neighbor N_{60}, which corresponds also to the diameter of the enclosing circle. For elliptical shapes, we determine the minor axis by projecting the first neighbor N_1 onto the tangential plane. The length of the minor axis is determined by the distance to N_1. The major axis is computed as the cross product of the minor axis and the normal. Its length is the distance to N_{60}. Fig. 7 illustrates the polygon setup for elliptical splats.

The alpha texture of the polygon is a discrete unit Gaussian function, stretched and scaled according to the polygon vertices deploying texture mapping hardware. The vertex positions of the polygon can be entirely calculated in the programmable vertex processor of contemporary graphics engines.

4.3. Deferred operations

We implemented a framework for deferred operations on all attributes of the 3D video fragments. Since vertex programs only modify the color and position attribute of the point samples during rendering, we do not destroy the consistency of the representation and of the differential update mechanism. Note that fragments can only be processed independently, and, because of the consistency constraints, fragments cannot be created or deleted in the vertex programs. Furthermore, temporal effects across multiple frames cannot be implemented by storing intermediate results, because the 3D operator stream modifies the representation asynchronously. Nevertheless, we designed a large number of visual effects from procedural warping to explosions and beaming. Periodic functions can be employed to devise effects such as *ripple*, *pulsate*, or sine waves. In the latter, we displace the fragment along its normal based on the sine to its distance to the origin in object space. For explosions, a fragment's position is modified along its normal according to its velocity. As illustrated in the accompanying video the explosion operation can take gravity into account. Note that all deferred operations are performed on-the-fly during rendering without any pre-processing.

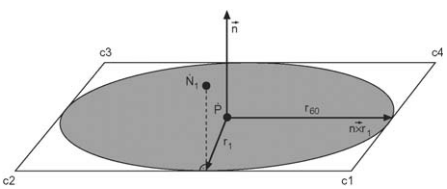

Fig. 7. Polygon setup for elliptical splat rendering. r_1 and r_{60} denote the distances from the point sample P to N_1 and N_{60}, respectively. c_1 to c_4 denote the calculated polygon vertices.

5. Prototype system

We built a prototype real-time 3D video fragments system together with a coding scheme which will be explained in the following subsections.

5.1. Physical setup

We use the blue-c [13] as test environment which is an immersive telepresence installation allowing simultaneous acquisition and projection. Sixteen Point Grey Research Dragonfly digital cameras are installed, all equipped with 640×480 CCD imaging sensors. Fifteen cameras surround the acquisition stage on a circular shape and one camera captures the scene from the top. We use CS-mount lenses with focal lengths between 2.8 and 6 mm. Note that we have a large working volume of approximately $3 \times 3 \times 2.2$ m^3 which needs to be covered by all cameras. The 16 cameras are calibrated by a fully automatic procedure based on self-calibration [23]. For estimating the radial lens distortion, we employ the Caltech camera calibration toolbox and we use the Open Computer Vision Library for correction. For simultaneous projection in the blue-c, switchable glasses are between the cameras and the object producing slightly less saturated textures from most cameras. We cope with this problem by adjusting the colors of individual cameras in a correction procedure. Although our cameras are capable of capturing synchronized images at 15 frames per second, the test environment limits us to five or nine frames per second. For most of the tests we triggered the cameras with five frames per second. The physical setup of the acquisition environment is depicted in Fig. 8.

Our processing system can be decomposed into three major parts. A set of camera nodes runs the acquisition clients, which do all image space processing steps in parallel, and finally transmit their data to a reconstruction node. The reconstruction node, in our case a dual processor machine with two AMD AthlonMP 2400+ CPUs, runs the multithreaded 3D point processing which transforms the 2D pixel operations into 3D

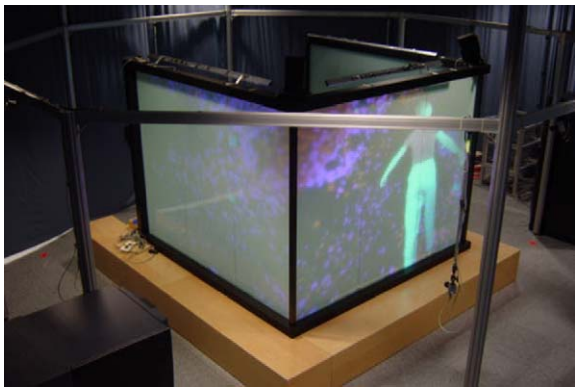

Fig. 8. The blue-c: An immersive telepresence installation allowing simultaneous acquisition and projection.

fragment operations. The rendering node receives the 3D fragment operations from the reconstruction node and maintains the data structure representing the 3D video object. This data structure is finally rendered to the screen. We currently use an 1.8 Ghz Pentium4 machine equipped with an NVIDIA GeForce4 Ti200 graphics accelerator. The rendering node transmits feedback information, i.e., the current viewpoint, to the reconstruction node. The reconstruction node controls the acquisition process at the camera nodes accordingly. All nodes are currently interconnected in a Fast-Ethernet local area network at 100 Mbit/s. In a typical application, the rendering node could also be connected via a wide area network.

5.2. 3D processing

For 3D video fragment processing we use a 3D reconstruction built upon the image-based visual hulls method [6]. However, instead of calculating depth and normal in a desired view, our 3D video fragments approach calculates 3D information for the camera views. For each pixel, we calculate a depth value which is then projected to a point in 3D space with associated normal. Our current implementation would be able to deal with up to 85k INSERT or UPDATEPOS operations or more than 800k UPDATECOL and DELETE operations per second. A caching scheme [6] ensures that the computation time for the costly INSERT and UPDATEPOS operations decreases logarithmically with the number of processed operations. The raw performance is sufficient for processing objects with less than 30k points. In a typical 3D video sequence, processed at 5 frames per second and containing in between 15k and 25k points, the position of 6% of the 3D video fragments is updated in each frame, whereas more than 10% might get a new color.

The operation scheduling at the reconstruction node is organized as follows: The contour data can simply be handed over to the visual hull reconstruction module. DELETE and UPDATE operations are immediately applied to the corresponding points. INSERT operations however, require a prescribed set of contours, which is derived from the active camera control. Furthermore, an efficient 3D video fragment processing requires that all DELETE operations from one camera node are executed before the INSERT operations of the same. The camera nodes support this operation flow by first transmitting contour data, then DELETE and UPDATE operations and, finally, INSERT operations. Note that the INSERT operations are generated in the order prescribed by the sampling strategy of the input image. On the reconstruction node, the operation scheduler will only forward INSERT operations to the visual hull unit if no other type of data is at hand.

Furthermore, every camera node, even the contour inactive cameras, transmit at least an empty set of contours for every frame. This strategy allows the reconstruction node to check if all cameras are still synchronized. The acknowledgement message of contour data contains the new state information for the corresponding camera node. The reconstruction node detects a frame switch while receiving contour data of a new frame. At that point in time, the reconstruction node triggers state computations, i.e. recomputes the sets of reconstruction and texture active cameras for the following frames.

The 3D video fragment operations are transmitted in the same order in which they are generated. But the relative ordering of operations from the same camera node is guaranteed. This property is sufficient for a consistent 3D data representation. Fig. 9 depicts an example of a differential 3D video stream.

The design of our 3D video system leads to a system inherent latency of three frames, which results from the three pipeline stages: acquisition, processing and rendering. In the worst case, the round trip latency, which has to be taken into account for viewpoint changes, sums up to 5 frames. Since our 3D video pipeline is triggered by

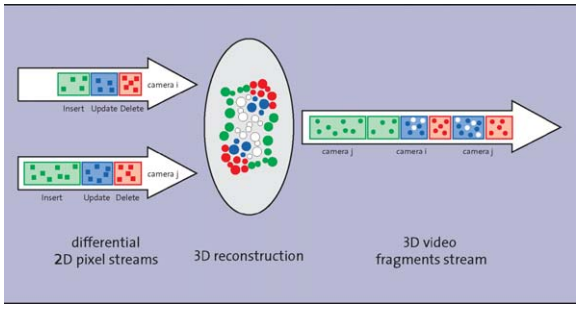

Fig. 9. Differential 3D video fragments streaming pipeline.

the synchronized cameras, the absolute latency is directly depending on the camera acquisition rate. The processing bottleneck in our pipeline are currently the camera nodes which can only handle up to nine frames per second at full NTSC-resolution.

5.3. Streaming and compression

The 3D video fragments pipeline requires a distributed consistent data representation. Each camera node shares with the reconstruction node a coherent representation of its differentially updated input image. The incremental updates of the rendering data structure also require a consistent data representation between the reconstruction and rendering nodes. Hence, all network links must support lossless and in-order data transmission. A TCP byte stream fulfills these requirements, however, TCP is not very well suited for real–time systems. The latency and jitter introduced by the built–in retransmission scheme and flow control and congestion management cannot be directly influenced by the application. We implemented an appropriate retransmission scheme for reliable data transmission based on the connectionless and unreliable UDP protocol and on explicit positive and negative acknowledgements. Since the application layer is now responsible of retransmissions, it is straightforward to detect transmission problems and adapt the application accordingly. An application with multiple renderers can be implemented by multicasting the 3D video fragments stream, using a similar technique as the Reliable Multicast Protocol RMP in the source-ordered reliability level [24]. The implementation of our communication layer is based on the TAO/ACE framework (http://www.cs.wustl.edu/~schmidt/TAO.html).

A 3D video fragment is defined by a position, a surface normal vector and a color. For splat footprint estimation issues (Section 4.1), the renderer further needs knowledge about the camera identifier and the image coordinates of the original 2D pixel. The geometry reconstruction is computed in float precision, but the resulting 3D position can accurately be quantized using 27 bits. This position encoding scheme leads in our acquisition stage to a spatial resolution of approximately $6 \times 4 \times 6\ \mathrm{mm}^3$. The remaining 5 bits of a 4 byte word can be used to encode the camera identifier. We encode the surface normal vector by quantizing the two angles describing the spherical coordinates of a unit length vector. We implemented a real-time surface normal encoder, which does not require any trigonometric computations on-the-fly, and which uses variable precision of up to 16 bits as suggested by [25,26]. Colors are encoded in RGB 5:6:5 format. At the reconstruction node, color information and 2D pixel coordinates are simply copied into the corresponding 3D video fragment.

Since all 3D fragment operators are transmitted over the same communication channel from the reconstruction node to the renderer, we need to encode the operation type explicitly. For simplicity, we use one prefix byte to each operation, and encode the operation type within. For UPDATE and DELETE operations, it is necessary to reference the corresponding 3D video fragment. We exploit the feature that the combination of quantized position and camera identifier references every single primitive. The renderer maintains the 3D video fragments in a hash table. Thus, each primitive can efficiently be accessed by its reference key. Fig. 10 shows the byte layout for all possible attributes of a 3D fragment operator.

Fig. 11 shows the bandwidth required by a typical sequence of differential 3D video, generated from five contour active and three texture active cameras at five frames per second. The average bandwidth in this 30 s sample sequence is 1.2 Megabit/s. The bandwidth is strongly correlated to the movements of the reconstructed person and to the changes of active cameras, which are related to the changes of the virtual viewpoint. The peaks in the sequence are mainly due to switches between active cameras. On average, INSERT and DELETE operators contribute to 25% and 12% of the bandwidth, respectively, the remaining bandwidth is consumed by UPDATE operators. The compression performance of our system is difficult to analyze, since, up to our knowledge, no previous work was done on streaming and compressing dynamically changing geometric data. Relating our work to 2D video compression, the bandwidth required by a 3D video fragment stream is in the same order of magnitude than the bandwidth required by two MPEG-2 streams. At five frames per second, MPEG-2 recommendations ask for three intra-coded frames per second, which leads to a bitrate of approximately 1 Megabit/s.

Furthermore, entropy coding can be applied to the 3D video fragments stream. Our experiments show that an additional compression ratio of 2:1 can be expected.

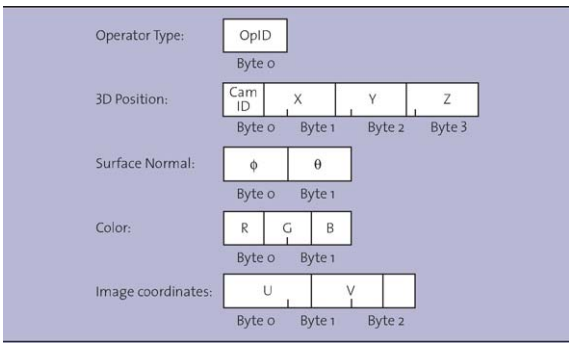

Fig. 10. Byte layout of all possible attributes of a 3D fragment operator.

6. Results

We have tested our 3D fragments pipeline in an immersive telepresence system with different persons. Fig. 12 shows some snapshots of 3D video sequences and Fig. 13 shows some images with corresponding depth maps. Especially in Fig. 13b the depths from the involved texture active cameras are visible. In the blue-c environment, the cameras have to be shuttered at approximately 5 ms for simultaneous acquisition and projection. Furthermore, active illumination is required during the acquisition phase. As shown in Fig. 8, LED arrays are mounted on the upper and lower edge of the projection screens. Within these constraints, it was not possible to achieve a homogeneous illumination of the user. Unfortunately, achieving a reasonable overall texture quality and a robust segmentation in this environment requires a high camera gain which leads to oversaturated regions in the camera images.

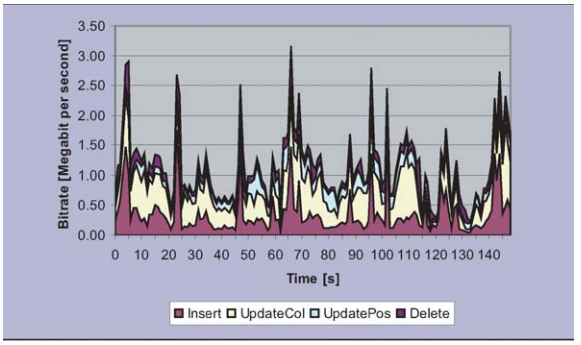

Fig. 11. Bitrate of a differential 3D video stream. The contributions from the different 3D operators are highlighted.

Due to performance reasons we use a relatively small number of silhouettes for 3D reconstruction which only leads to a rough shape approximation of the person. Temporary visible geometry artifacts can be explained by the inherent nature of the visual hull reconstruction method which is not capable of properly reconstructing concave regions. We use a simple linear color calibration method which, however, proved to be insufficient for removing all texture artifacts due to improper color alignment between cameras.

Fig. 12d illustrates the sine wave operation on the 3D video object from Fig. 12c. Fig. 14 shows an explosion of a 3D video object. The accompanying video shows example 3D video sequences with arbitrary virtual viewpoints and more effects. For our 3D video objects of less than 30k point samples we never experienced performance problems on the point rendering engine, even with enabled deferred operations.

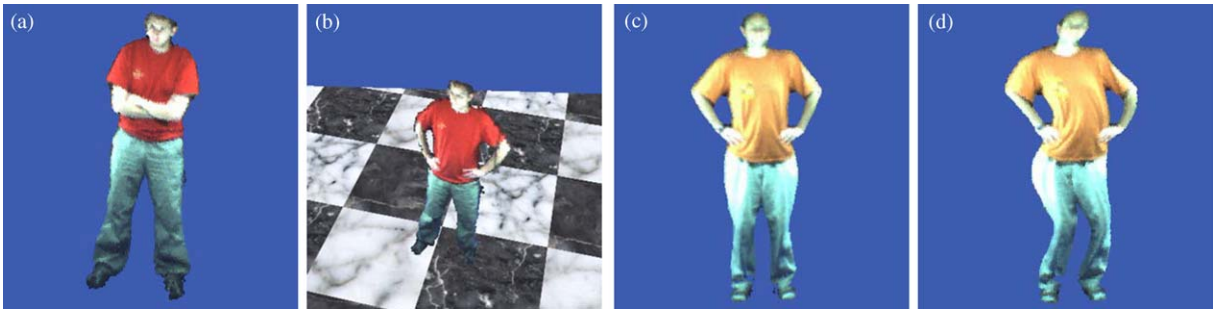

Fig. 12. Examples for 3D video fragment objects: (a) point rendered view; (b) another view from the same 3D video sequence; (c) point rendered view from another sequence; (d) deferred operation: sine wave.

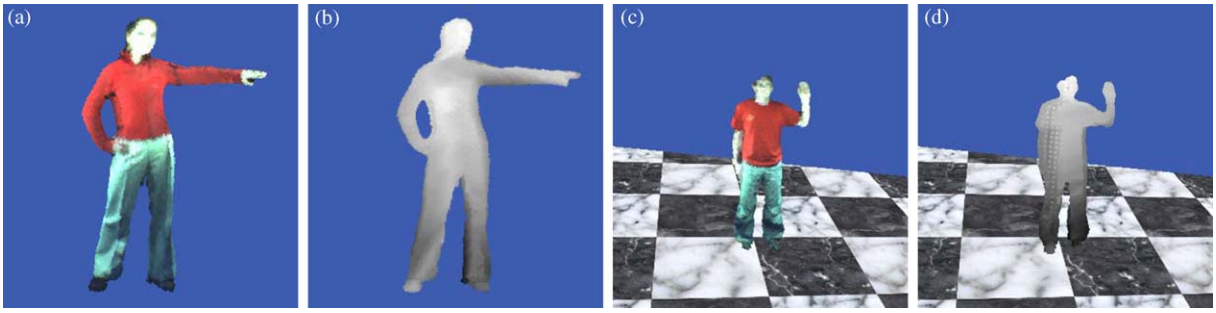

Fig. 13. 3D video fragment objects: (a + c) point rendered views; (b + d) depth maps from the respective views.

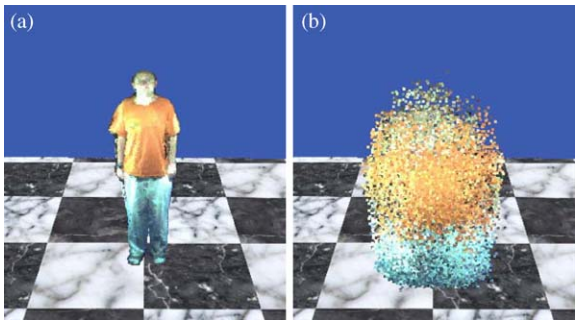

Fig. 14. Deferred operations: (a) point rendered view; (b) view from (a) exploded with gravity taking into account.

7. Conclusions and future work

The 3D video fragments framework proposes a real-time 3D video pipeline which generalizes the 2D video pixels towards 3D irregular point samples. A differential update scheme exploits the inter-frame coherence of consecutive frames and a dynamic camera control scheme continuously reconfigures the 3D video process for optimal performance, according to the feedback from application and pipeline stages. The 3D video sequences are rendered using efficient point based splatting schemes and state-of-the-art vertex and pixel processing hardware.

In our current implementation, blending between several cameras might still lead to discontinuities. Furthermore, the requirement of a coherent distributed data representation is a severe constraint for the networking layer and an error-resilient representation for our 3D video pipeline needs to be investigated. In the future, we also plan to utilize the concepts of 3D video fragments for 3D video recording and try to integrate high-quality re-shading which requires smoothing of the poorly reconstructed normals. Moreover, we want to investigate new concepts for real-time 3D motion estimation.

Acknowledgements

We thank Wojciech Matusik for sharing the IBVH source code with us. Furthermore, we would like to thank our students Stefan Hösli, Nicky Kern, and Christoph Niederberger for implementing parts of the system and Martin Näf for producing the video. A special thanks to all members of the blue-c team for many fruitful discussions. This work is part of the blue-c project which has been funded by ETH grant No. 0-23803-00 as an internal poly-project.

References

[1] Moezzi S, Katkere A, Kuramura DY, Jain R. Immersive video. In: Proceedings of the 1996 Virtual Reality Annual International Symposium. Silver Spring, MD: IEEE Computer Society Press; 1996. p. 17–24.

[2] Szeliski R. Rapid octree construction from image sequences. CVGIP: Image Understanding 1993;58(1):23–32.

[3] Vedula S, Baker S, Kanade T. Spatio-temporal view interpolation. In: Proceedings of the 13th Eurographics Workshop on Rendering, Eurographics Association, 2002. p. 65–76.

[4] Carceroni R, Kutulakos K. Multi-view scene capture by surfel sampling: from video streams to non-rigid 3D motion, shape & reflectance. In: Proceedings of the Seventh International Conference on Computer Vision, IEEE Press, 2001. p. 60–7.

[5] Würmlin S, Lamboray E, Staadt OG, Gross MH. 3D video recorder. In: Proceedings of Pacific Graphics'02. IEEE Computer Society Press; 2002. p. 325–34.

[6] Matusik W, Buehler C, Raskar R, Gortler SJ, McMillan L. Image-based visual hulls. In: Akeley K, editor. Proceedings of SIGGRAPH 2000. New York: ACM Press/ACM SIGGRAPH; 2000. p. 369–74.

[7] Laurentini A. The visual hull concept for silhouette-based image understanding. IEEE Transactions on Pattern Analysis and Machine Intelligence 1994;16(2):150–62.

[8] Matusik W, Buehler C, McMillan L. Polyhedral visual hulls for real-time rendering. In: Proceedings of 12th Eurographics Workshop on Rendering, Eurographics Association, 2001. p. 115–25.

[9] Kanade T, Rander P, Narayanan P. Virtualized reality: constructing virtual worlds from real scenes. In: IEEE MultiMedia 1997;4(1):43–54.

[10] Narayanan PJ, Rander P, Kanade T. Constructing virtual worlds using dense stereo. In: Proceedings of the International Conference on Computer Vision ICCV 98, 1998. p. 3–10.

[11] Mulligan J, Daniilidis K. View-independent scene acquisition for telepresence. In: Proceedings of the International Symposium on Augmented Reality, 2000. p. 105–8.

[12] Pollard S, Hayes S. View synthesis by edge transfer with application to the generation of immersive video objects. In: Proceedings of the ACM Symposium on Virtual Reality Software and Technology. New York: ACM Press/ACM SIGGRAPH; 1998. p. 91–8.

[13] Gross M, Würmlin S, Näf M, Lamboray E, Spagno C, Kunz A, Koller E, Svoboda T, Van Gool L, Lang S, Strehlke K, Vande Moere A, Staadt O. A spatially immersive display and 3D video portal for telepresence. In: Proceedings of SIGGRAPH 2003. New York: ACM Press/ACM SIGGRAPH, 2003. p. 819–27.

[14] Draft requirements for 3DAV. In: Smolic A, Yamashita R, editors. JTC1/SC29/WG11 N4795. ISO/IEC, May 2002.

[15] Pfister H, Zwicker M, van Baar J, Gross M. Surfels: surface elements as rendering primitives. In: K. Akeley, editor. Proceedings of SIGGRAPH 2000. ACM Press/ACM SIGGRAPH; 2000. p. 335–42.

[16] Zwicker M, Pfister H, van Baar J, Gross M. Surface splatting. In: Proceedings of SIGGRAPH 2001. New York: ACM Press/ACM SIGGRAPH; 2001. p. 371–8.

[17] Ren L, Pfister H, Zwicker M. Object space EWA surface splatting: a hardware accelerated approach to high quality point rendering. In: Proceedings of Eurographics 2002, COMPUTER GRAPHICS Forum, Conference Issue, Blackwell Publishing Ltd, Oxford, UK, 2002. p. 461–70.

[18] Wand M, Strasser W. Multi-resolution rendering of animated scenes. In: Proceedings of Eurographics 2002, COMPUTER GRAPHICS Forum, Conference Issue, Blackwell Publishing Ltd, Oxford, UK, 2002. p. 483–91.

[19] Rusinkiewicz S, Levoy M. QSplat: a multiresolution point rendering system for large meshes. In: Proceedings of SIGGRAPH 2000. New York: ACM Press/ACM SIGGRAPH; 2000. p. 343–52.

[20] Buehler C, Bosse M, McMillan L, Gortler SJ, Cohen MF. Unstructured lumigraph rendering. In: SIGGRAPH 2001 Conference Proceedings, ACM Siggraph Annual Conference Series, ACM Press, New York, 2001. p. 425–32.

[21] Vlasic D, Pfister H, Molinov S, Grzeszczuk R, Matusik W. Opacity light fields: interactive rendering of surface light fields with view-dependent opacity. In: Proceedings of the 2003 Symposium on Interactive 3D Graphics, ACM Press, New York, 2003. p. 65–74.

[22] Pauly M, Gross M. Spectral processing of point-sampled geometry. In: Proceedings of SIGGRAPH 2001. ACM Press/ACM SIGGRAPH; 2001. p. 379–86.

[23] Pollefeys M, Koch R, Van Gool L. Self-calibration and metric reconstruction in spite of varying and unknown internal camera parameters. International Journal of Computer Vision 1999;32(1):7–25.

[24] Whetten B, Montgomery T, Kaplan SM. A high performance totally ordered multicast protocol. In Dagstuhl Seminar on Distributed Systems, Lecture Notes in Computer Science, Springer, 1994. p. 33–57.

[25] Deering MF. Geometry compression. In: Cook R, editor. SIGGRAPH 95 Conference Proceedings, Annual Conference Series, Los Angeles, CA, 06–11 August 1995. ACM SIGGRAPH. Reading, MA: Addison-Wesley; August 1995. p. 13–20.

[26] Botsch M, Wiratanaya A, Kobbelt L. Efficient high quality rendering of point sampled geometry. In: Proceedings of the 13th Eurographics Workshop on Rendering, Eurographics Association, 2002. p. 53–64.