Image Based Rendering from Handheld Cameras using Quad Primitives

J.-F. Evers-Senne and R. Koch

Christian-Albrechts-University of Kiel, Germany
Institute of Computer Science and Applied Mathematics
Email: {evers|rk}@mip.informatik.uni-kiel.de

Abstract

In this paper we will present a novel approach of using surface patches for Image Based Rendering. Based on image sequences acquired with a freely moving portable multi-camera-rig we can extrapolate novel views of complex real scenes in real-time. The cameras are calibrated from the image sequence itself and dense depth maps are computed for each camera view using an improved multiview depth estimation technique. The depth maps are then approximated with quad surface patches for efficient rendering.

1 Introduction

Displaying fotorealistic virtual worlds has always been one of the major goals in computer graphics. "Image Base Rendering" aims towards this by using images of real scenes to render new views. This should bring the quality of photographs into virtual 3D environments. Using today's CCD cameras it is possible to capture large and complex scenes with thousands of images in short time. These visual components can be used almost immediately, while the geometrical representation of the scene has to be reconstruced in form of depth maps.

In this paper we will present a novel approach to render complex scenes captured with uncalibrated handheld cameras in real-time. Significant improvements in multi-view stereo depth estimation allow us to use points as graphics primitives instead of triangles. The rendering performance depends on the density of the points which can be adopted to the complexity of the scene interactively. Further on, some editing of the scene in 3D is possible. Removing unwanted objects from image sequences normally means to remove parts of each image of the whole sequence. While in 3D, an object can be selected and removed by giving its position and size.

In the next section related work is discussed and the motivation for this work is given. The rendering stage is the last step of a complete system designed to capture arbitrary scenes and use them for virtual environments. In section 3 we will describe the necessary steps and precomputations of the complete chain briefly. In section 4 the construction and the usage of points for Image Based Rendering is explained in detail. In section 5 we show some results and examples, issues concerning quality and performance are discussed here. Finally this paper is concluded in section 6.

2 Motivation and Previous Work

The roots of Image Base Rendering date back to 1991 when Adelson [1] introduced the plenoptic function. In 1995 McMillan and Bishop published [10], which uses the plenoptic function and images for rendering. Levoy and Hanrahan [9] introduced lightfield rendering which was enhanced in many directions later on. While the scene geometry for the lightfield was assumed to be a plane, Gortler et al. [4] started to use an approximative 3D shape for the Lumigraph. Viewdependent Texture Mapping as described by Debevec et al. [2] is another method of rendering known geometry with textures from image sequences.

All these methods share the problem of relying on one globally consistent 3D model which is not always possible to obtain. Different methods have been published over the years to obtain such models. Some of them use specialized Hardware (Laser Range Scanner, calibrated multi-camera systems), others need controlled environments (markers). With the Structure from Motion approach Pollefeys et al. [12] presented methods to estimate the camera calibration along with a sparse geometrical representation of the scene for unknown scenes at the same time. After the calibration, stereo algorithms can be used to compute dense depth maps. Based on this Heigel et al. [5] discussed a plenoptic modeling approach which makes use of local depth maps to correct the interpolation of each ray. Due to occlusions, errors from the calibration and depth estimation, it is not always possible to construct a consistent global 3D model useable for VDTM for example. In [3] it was proposed to use viewdependent geometry. Halfway between Heigels ray-interpolation and Debevecs VDTM, a triangular surface mesh is constructed on-the-fly by fusing depth information from the selected cameras. Each triangle is then textured from these cameras. However, in regions with large variations in depth, distortions are visible, because triangles were created which connect foreground and background. Most often this connection does not reflect the real scene in which objects are grouped loosely in depth, but the mesh serving as warping surface is a convex hull of the scene. Whitted and Turner [14] were the first to use points instead of triangles as graphics primitives to avoid the connectivity problem associated with complex topology. Lately, different researchers took over to further develop this approach. Rusinkiewicz and Levoy used multiresolution pointsets as substitution of large meshes in their QSplat approach [13]. Representing textured surfaces with splats was published by Pfister et al. [11] and improved by Zwicker et al. [15].

These Point Base Rendering techniques avoid connections between primitives, which suits them well for the purpose of rendering geometry from depth maps where no topology information is given. Transfering meshes into points or surface splats can be done with arbitrary density to avoid holes in the point based representation. The same is true for ideal depth maps, where the depth for each pixel is exactly known. But depth maps from real image sequences of complex scenes are most often far from ideal. Holes from occlusions remain and depth estimation errors result in noise. In [3] we argued, that triangles are more useful to interpolate errors and holes in depth maps. But significant improvements in the calculation of the depth maps allows us to use points as rendering primitives not only on synthetic footage. This gives us the chance to avoid the distortions from interpolating triangles and use quads to gain the image quality.

3 Prerequisites

In this section we explain the necessary steps of preparation for Image Based Rendering from hand-held camera sequences. These modules operate offline and need to be performed only once for each acquired scene:

- Image capture,
- Camera calibration,
- multi-view depth estimation,
- generation of splats from depthmaps (as described in section 4).

Image Capture Arbitrary cameras can be used to capture the scene, but improvements can be achieved using Multi-Camera systems as described in [3]. The system consists of several uncalibrated cameras mounted on a pole looking into the same direction and operating synchronously. A 2D scanning of the viewpoint surface is obtained with a single walk-by and the rigid coupling can be exploited in the camera calibration.

Camera calibration The images are acquired with a hand-held multi-camera system with arbitrary camera motion. Since we want to avoid putting markers into the scene, we have no control over the scene content. Therefore, all camera views are uncalibrated and the calibration and the camera track must be estimated from the image sequence itself.

For tracking and calibration we have extended the SFM approach of Pollefeys and Koch [8, 12] to a multi-camera configuration. Some details can also be found in [6]. In this approach the intrinsic and extrinsic camera parameters for each image are estimated. The standard description of a projective camera consists of two 3x3 matrices K and R and the 3-dimensional vector C. K contains the intrinsic camera parameters focal length, aspect ratio and image center, R describes the rotation of the camera in space and C is the translation vector of the camera center. The projection matrix M_{proj} projects the homogeneous 3D point P into a 2D image at image point p with

$$zp = M_{\text{proj}} P \tag{1}$$

with z is the projective depth. M_{proj} is defined as the following projective 3x4-matrix:

$$M_{\text{proj}} = K[R^T | - R^T C] \tag{2}$$

The SFM approach automatically tracks salient 3D features (intensity corners) throughout the images. The calibration results in a projection matrix for each camera of the sequence and a sparse point cloud of the tracked 3D feature points.

Multi-viewpoint depth estimation With the calibrated image sequence at hand, one can obtain dense depth maps from multi-viewpoint disparity estimation. From the calibration the epipolar geometry between pairs of images is known and can be used to restrict the correspondence search to a 1-dimensional search. We extend the method of Koch et al. [7] for multi-viewpoint depth estimation to the multi-camera configuration. This method is ideally suited since we can exploit the 2D grid of linked depth maps for all cameras of the rig. Using more than one pair of cameras allows us to fill holes from occlusions and to enhance the precision of the depth maps. This results in very dense depth maps of the local scene geometry. Only in very homogeneous image regions it might not be possible to extract sufficient scene depth. Figure 1 shows a scene with complex geometry and one real view. Two depth maps in figure 2 give an idea about the quality of our improved multi-view stereo.

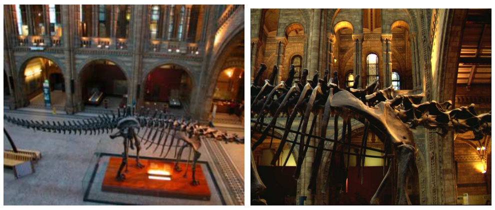
Figure 1: Left: An overview of the Dino scene inside the National History Museum, London. Right: One of 214 images from the multi-camera system.

4 Using Splats

After the steps of image capturing, camera calibration and depth estimation, we have data sets consisting of an image, a projection matrix and a depth map for each real camera.

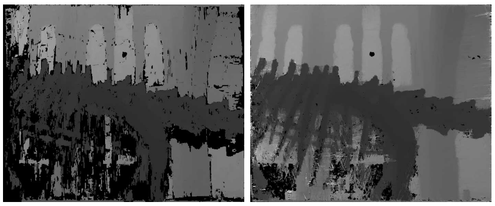
Figure 2: A depth map from multi-view stereo. Left: original algorithm as described in [7], right: our improved multi-view stereo exploiting the 2D grid of cameras.

4.1 Representing Splats as Quads

For point based rendering of textured models a point representation which can be textured easily has to be chosen. With respect to the 3D API OpenGL there is only a small number of possibilities: GL-Points, GL-Point-Sprites and Polygons. Standard points in GL can not be textured, they can only have one color. To display the texture, each point must have the color corresponding to the area in the image. This means that before each point can be drawn, the color has to be looked up and set appropriately, which prohibits the use of acceleration structures like vertex arrays and results in an enormous amount of function calls. Even worse, the size of points is specified in screen space, which means all points are of the same size on screen, independent of their original distance. To overcome these drawbacks the so called *Point-Sprites* have been introduced. Each point-sprite is defined by only one homogeneous 3D point, and their size is defined in object-space. While rendering, each point-sprite is expanded to a quad with the width and height of the given size. After the projection into screen space, the size depends on the distance in object space. Furthermore, texture can be applied onto each point-sprite, but texture coordinate generation and manipulation is limited. Using standard quads seems to be the best choice in this case. Each quad is specified by four vertices (homogeanous 3D points) and all of them can be stored in one vertex array, reducing the number of function calls down to 3. Quads can be textured like any other polygons, which allows very efficient projective texture mapping without the need to calculate and store each texture coordinate explicitly.

4.2 Creating Quads

Based on the discrete sampled images and a given spacing s in pixels the quads are arranged in the image plane in a regular grid. To avoid gaps in the projected visualization from non-overlapping splats, an enlargement e for each splat is chosen. Typically a fraction of one pixel enlargement ($e = 0.2$) is a good choice. The overlap of adjacent quads is $2e$. This ensures that for virtual views differing from a real view slightly, gaps between quads are filled. The value of e depends on the differences in depth between quads and the maximum angle between the real camera and the virtual camera. Gaps can also be filled from using more than one real camera to generate a new view. For sparse sampled scenes it could be useful to increase e. Quads placed at the image borders exceed the borders by the enlargement. During texturing, texture coordinates are clamped resulting in fragments drawn with repeated color values. With $e \leq 1$ these errors remain invisible.

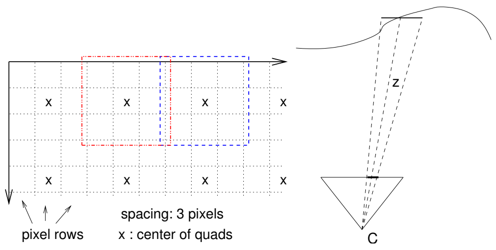

Figure 3: Left: Setting up the quads in the image plane with spacing $s = 3$ and enlargement $e = 0.2$. Right: Reprojecting quads from the image plane to approximate the geometry.

The following is done for each camera k seperately, taking $P = P_k$ the projection matrix combined from M, R, C as described above. For each quad i the homogeneous 2D coordinates of all four corners $j, j = 0, 1, 2, 3$ in the image plane p_i^j are calculated. The distance z_i of the new quad i to the camera center is determined from the depth map at the center of the quad. A median filter with adjustable size is applied to remove depth outliers. Taking the 2D point p_i^j in the image plane and the corresponding distance z_i from the depth map, the Euclidean 3D scene point P_i^j can be calculated as:

$$P_i^j = z_i * (KR^T)^{-1} * p_i^j + C$$

The basic setup and the reprojection is illustrated in figure 3. Different Level-of-Details can be created by varying the spacing s and repeating the above procedure. To obtain three different Levels-of-Detail, starting with $s_0 = 3$, the next steps are $s_i = 2s_{i-1} + 1$. All points P_i^j for a each Level-of-Detail are stored in separate vertex arrays and saved on disk to avoid recalculation for each start of the program.

4.3 Camera Selection

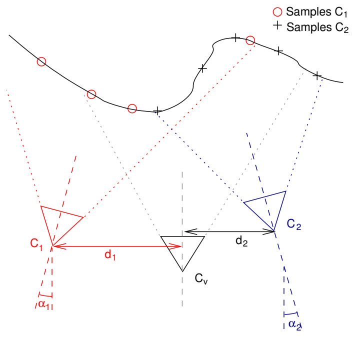

Figure 4: Two real cameras C_1, C_2 are evaluated in respect to the virtual camera C_v.

The first step in Image Based Rendering is to select the real cameras giving information for the new view. As described in [3], three different criterias are evaluated for each real camera. This is shown in figure 4. A visibility check verifies if a real camera shares a viewing volume with the virtual camera by projecting a few selected quads into the virtual camera. If these are not visible in the virtual new view, this real camera is marked invalid. The viewing angle α is the angle between the viewing directions of two cameras. If the viewing angle between the virtual camera and a real camera exceeds their field-of-view, this real camera is "off-axis" and is also marked invalid.

All cameras which are not marked invalid are ranked according to their orthogonal distance d to the viewing direction of the virtual camera. The closer to the viewing direction a real camera is, the better it is ranked. All cameras marked invalid can not be used for proper view interpolation.

From the remaining ranked cameras, the first N_{cam} are selected. This number of active cameras

N_{cam} and also the current Level-of-Detail l can be manipulated by the user interactively to control the quality and performance of the rendering.

4.4 Rendering and Texturing Quads

From now on, the first N_{cam} ranked cameras are traversed and for each of them the following procedure is followed:

- Use the image from the real camera as texture,
- Setup the texture matrix and
- draw all quads for the current LoD l.

To render textured quads, the original image of the real camera has to be used as texture. This is done by “binding” it as current texture in OpenGL. Using automatic texture coordinate generation means that texture coordinates for each vertex are generated from the vertex itself. At first, a linear combination of the vertex’ components is evaluated for each component of the texture coordinate. A direct mapping from the vertex serves well. This intermediate texture coordinate is multiplied with the current texture matrix and the result is taken to address the texture:

$$p_{\text{tex}} = M_{\text{tex}} P_{\text{vertex}}$$

This is equivalent to (1) with $z = 1$. The only difference is, that M_{proj} maps all points into image coordinates from $(0,0)^T$ up to $(x_{\max}, y_{\max})^T$ and texture coordinates have to be between $(0,0)^T$ and $(1,1)^T$. It suffices to decompose M_{proj} into C, R, K_{proj} with

$$K_{\text{proj}} = \begin{pmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

and adopt it to

$$K_{\text{tex}} = \begin{pmatrix} \frac{f_x}{x_{\max}} & \frac{s}{y_{\max}} & \frac{c_x}{x_{\max}} \\ 0 & \frac{f_y}{y_{\max}} & \frac{c_y}{y_{\max}} \\ 0 & 0 & 1 \end{pmatrix}$$

then M_{tex} is

$$M_{\text{tex}} = K_{\text{tex}}[R^T | - R^T C]. \qquad (3)$$

Setting M_{tex} the current texture matrix results in homogeneous texture coordinates generated on-the-fly. This maps the current texture projectively onto the quads.

Drawing all precalculated quads for LoD l is finally done by sending the vertex array corresponding to the current real camera to the GL-pipeline.

5 Quality and Performance

We implemented the proposed system in C++ using OpenGL and tested it on a standard PC (Linux) with 1.7 GHz Athlon, 1GB memory and a Geforce4Ti 4400 with 64MB memory. In the following section we will discuss the evaluation of quality and performance and give some examples.

5.1 Visual Quality

Measuring the visual quality of rendering algorithms is difficult. We decided to use a synthetic virtual scene to produce ground truth material. This avoids errors from camera calibration and depth estimation. The scene is a VRML reconstruction of the Arenberg castle in Leuven, Belgium, that has been modeled using Structure from motion and is now visualized with standard tools. Screenshots from four different heights simulating a walk-by with the multi-camera system were taken. The depth maps were taken from the z-buffer of the render hardware, while the projection matrices were computed from the motion parameters of the virtual camera. Based on this ideal footage, we chose one of the views as reference, removed it from the sequence and tried to interpolate this one from the remaining cameras. For better comparison, image subtraction was used to display the differences and the most critical region was magnified. Due to variations in depth, most artefacts are located here. A mean absolute difference (MAD) value over all image pixels is given for easy comparison.

In figure 7 the best result with viewdependent geometry from [3] is shown. Even with multipass texturing the remaing errors give an MAD of 11.8.

Using quads from only one nearest camera results in holes from occlusions, but activating two cameras, for the given scene all holes are filled as shown in figure 7. A spacing of 5 with an enlargement of 0.2 gives an MAD of 5.16. This shows that expecially for critical regions using points instead of meshes serves far better for geometry approximation.

Decreasing the spacing result in an slightly increased quality, a spacing of 3 gives an MAD of 4.37 (figure 8). Figure 9 shows the rendered image for spacing 1 with MAD=1.67, but heavily increased computational costs. Compared to spacing 3 the polygon count increases from 44,000 to

393,000 for the given scene, because each quad represente one pixel.

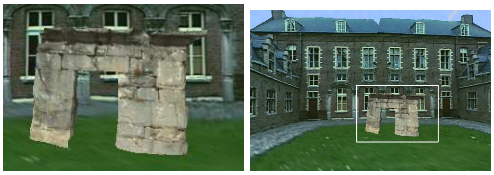

Figure 5: An input view of a synthetic scene (right) and a magnified closeup of critical regions (left).

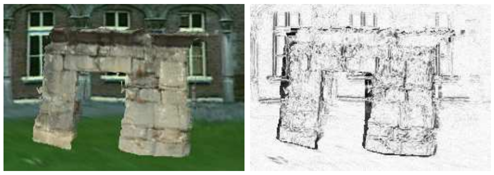

Figure 6: Viewdependent Geometry with multi-pass texturing from [3] results in a MAD = 11.8

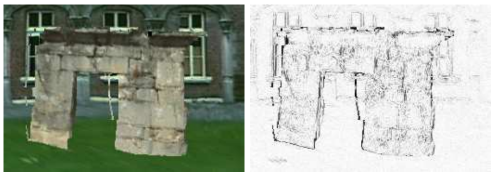

Figure 7: Using quads with a spacing 5 from 2 cameras gives a MAD = 5.16.

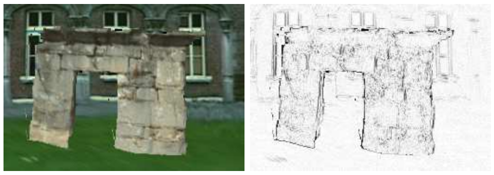

Figure 8: With spacing spacing 3 a slightly better MAD of 4.37 is achieved.

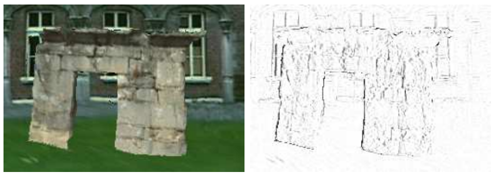

Figure 9: Significant enhancements result from a spacing 1, the MAD is 1.67. But computational costs grow by factor 9 compared to a spacing of three.

5.2 Evaluation of real scenes

In the previous section we demonstrated that our rendering approach gives good results on synthetic scenes, especially ideal depth maps. But the main target for Image Base Rendering is to use images from real cameras for view generation. To test our approach we selected a scene with a high complexity and lots of occlusions. The "Dinosaur" sequence

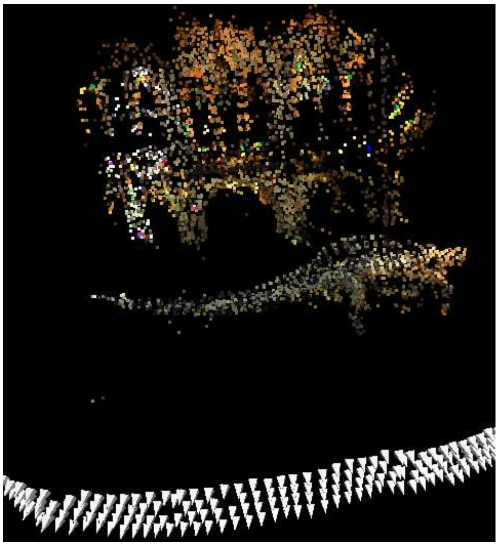

Figure 10: 4-camera acquisition of the Dinosaur showing the results of SFM-calibration with camera positions (pyramids) and 3D feature points.

was taken in the entrance hall of the National History Museum in London, figure 1 gives an overview of the hall. With four cameras a transversal scan alongside the skeleton of a dinosaur was recorded. Due to difficult lighting conditions, a tripod with dolly was used to avoid motion blur. The positions and orientations of all cameras and the point cloud from the calibration is shown in 10.

This scene is divided into two distinct layers of depth: the background with archways and windows and the skeleton occluding parts of the background. View-adaptive meshing as in [3] creates one single mesh connecting foreground and background, which results in distortions. Using our point based approach, the resolution and quality of the reconstruction relies on the quality of the depth maps mainly. Fine grained structures like rib bones are modeled and distortions are avoided because the background visible between the bones is not connected to the foreground. Problems arise from regions in the depth maps where the depth estimation failed. During the rendering, these regions have to be filled from other cameras or they remain black. Activating more and more cameras leads to another problem. Miss-matches in the depth estimation result in miss-located quads. The more cameras are active, the more misplaced quads disturb the rendering. Filtering the depth maps could reduce these artefacts significantly, but small objects are also removed by filtering.

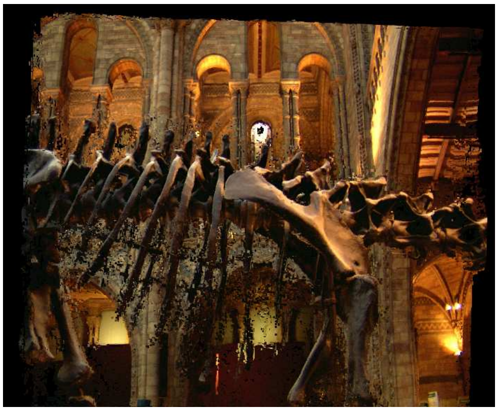
Figure 11: Rendered Image using four cameras. Misplaced quads due to errors in the depth estimations results are visible in critical regions.

5.3 Performance

Image based rendering typically involves large amounts of data. To handle several hundred images as textures, these are only loaded on demand. Typically only a few cameras are active to create a new view. Before using a texture, it is checked if this one resides in the main memory already, otherwise it is loaded and activated. The amount of memory used for textures can be limited, an LRU-mechanism removes the least recently used textures if necessary. The required time to load and bind a texture can be significantly reduced with texture compression. Compressed textures use only 1/6 of the original size which saves time loading them from disk into the graphic hardware. In a preprocessing phase all images are stored as compressed textures and the quads are also precomputed and stored as vertex arrays. For camera ranking only very few data for each camera is used. It would be possible to load the vertex arrays on demand, too. Using these techniques, we are able to handle image sequences of very large size.

The synthetic castle sequence consists of 214 images of the size 768x512. The images and the depth maps use 566 MByte space. After the preprocessing, the textures use 109 MByte. Creating quads with spacing 3 for 4 different Level-of-Details produces 496 MByte of vertex arrays, using a spacing of 5, this drops to 183 MByte.

The framerate which can be achieved during rendering depends mainly on the number of quads to draw. The total number of quads N_{total} is affected by the number of quads of the current Level-of-Detail N_l and the number of active cameras used N_c: $N_{\text{total}} = N_l * N_C$. N_0, which is the number of quads for the finest LoD depends on the image size x, y and the spacing s : $N_0 = \frac{xy}{s^2}$. For all $l > 0$, is $N_l = \frac{1}{4}N_{l-1}$. For the given example, N_0 is approximatly 44,000 quads. Using 5 active cameras the resulting framerate is 36 fps for $N_{\text{total}} = 218,000$ quads. Activating more cameras to gain an overview over a large scene can easily be compensated by reducing the Level-of-Detail. All 214 cameras can be used at once with 28 fps setting the Level-of-Detail to 3 such that $N_3 = 409$ and $N_{\text{total}} = 87,000$.

5.4 Limitations

The usage of quads to represent points is limited down to a spacing of 3. In this case, each 9 pixels are combined into one quad and 9 values from the depth map are represented by 4 vectors of size three: 12 values. It is possible to decrease the spacing, but at $s = 1$, each single depth value is substituted by one quad, which means that the data for the geometry grows by factor 12. For the given scene this results in 1.5 million vertices and 18 MByte data for each camera.

Other techniques like point-sprites, displacement maps or elevation maps should be used if per-pixel precision is required. But using depth maps from uncalibrated cameras it is sufficient to use a spacing of 3 or even 5. The missing accuracy and some noise normally require filtering, which allows to subsample the depth maps.

6 Conclusions

In this paper we presented a method to generate new virtual views from handheld camera sequences. 3D geometry is approximated using points as primitives. In contrast to viewdependent geometry with meshes, no topology is assumed which reduces distortions for complex scenes. Based on synthetic data, the results show significantly improved rendering quality. Using multi-camera systems and improved multi-view depth estimation methods, the increased demands for the quality of local depth maps can be satisfied. A method combining meshes and points is currently under developement which should again increase quality and performance. In homogeneous regions of the depth maps local static mesh can be created and rendered. Regions with large variation in depth will be filled with points, avoiding topological connections between distinct objects.

Acknowledgements

The work is being funded by the European project IST-2000-28436 ORIGAMI. The images for the dinosaur scene were supplied by Oliver Grau, BBC Research.

References

[1] E.H. Adelson and E.H. Bergen. *Computation models of visual processing*, chapter "The plenoptic function and the elements of early vision". MIT Press, 1991.

[2] Paul Debevec, Yizhou Yu, and George Boshokov. Efficient view-dependent image-based rendering with projective texture-mapping. Technical Report CSD-98-1003, 20, 1998.

[3] J.-F. Evers-Senne and R. Koch. Image based interactive rendering with view dependent geometry. In *accepted to Eurographics 2003*, Computer Graphics Forum. Eurographics Association, 2003.

[4] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. *Computer Graphics*, 30(Annual Conference Series):43–54, 1996.

[5] Benno Heigel, Reinhard Koch, and Mark Pollefeys. Plenoptic modeling and rendering from image sequences taken by a hand-held camera. In *Proceedings of DAGM 1999*, 1999.

[6] R. Koch, J.F. Frahm, J.M.and Evers-Senne, and J. Woetzel. Plenoptic modeling of 3d scenes with a sensor-augmented multi-camera rig. In *Tyrrhenian International Workshop on Digital Communication (IWDC): proceedings*, Sept. 2002.

[7] R. Koch, Pollefeys M., and L. Van Gool. Multi viewpoint stereo from uncalibrated video sequences. In *Proc. ECCV'98*, number 1406 in LNCS. Springer, 1998.

[8] R. Koch, M. Pollefeys, B. Heigl, L. Van Gool, and H. Niemann. Calibration of handheld camera sequences for plenoptic modeling. In *Proceedings ICCV 99*, Korfu, Greece, 1999.

[9] Marc Levoy and Pat Hanrahan. Light field rendering. *Computer Graphics*, 30(Annual Conference Series):31–42, 1996.

[10] Leonard McMillan and Gary Bishop. Plenoptic modeling: An image-based rendering system. *Computer Graphics*, 29(Annual Conference Series):39–46, 1995.

[11] Hanspeter Pfister, Matthias Zwicker, Jeroen van Baar, and Markus Gross. Surfels: Surface elements as rendering primitives. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 335–342. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.

[12] Marc Pollefeys, Reinhard Koch, and Luc J. Van Gool. Self-calibration and metric reconstruction in spite of varying and unknown internal camera parameters. *International Journal of Computer Vision*, 32(1):7–25, 1999.

[13] S. Rusinkiewicz and M. Levoy. QSplat: A multiresolution point rendering system for large meshes. In *SIGGRAPH 2000, Computer Graphics Proceedings*, pages 343–352. ACM Press, 2000.

[14] Turner Whitted and Marc Levoy. The use of points as a display primitive. Technical report, Computer Science Department University of North Carolina, 1985.

[15] Matthias Zwicker, Hanspeter Pfister, Jeroen van Baar, and Markus Gross. Surface splatting. In *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 371–378. ACM Press, 2001.