Vision, Modeling, and Visualization (2012)
M. Goesele, T. Grosch, B. Preim, H. Theisel, and K. Toennies (Eds.)

Interactive Geometry-Aware Segmentation for the Decomposition of Kaleidoscopic Images

O. Klehm[1,2] I. Reshetouski[1,3] E. Eisemann[2,4] H.-P. Seidel[1] I. Ihrke[1,3]

[1]MPI Informatik [2]Intel Visual Computing Institute [3]Saarland University [4]Delft University of Technology / Télécom ParisTech

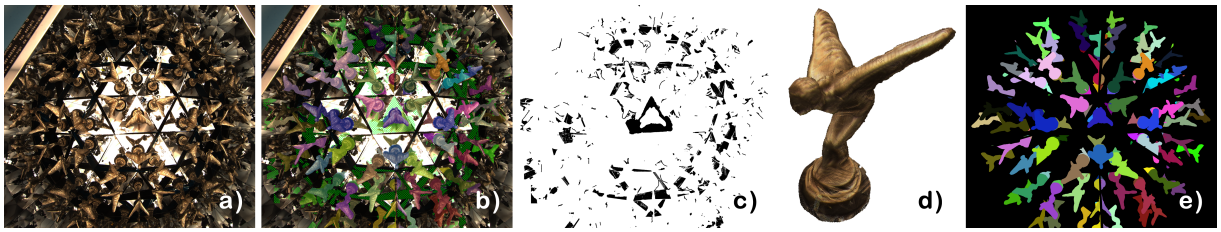

Figure 1: *a) Input: Kaleidoscope image. b) User-drawn mask (green checkerboard pattern). c) User-drawn mask isolated. d) Approximate visual hull generated from (c) using image-based shading. e) Resulting labeling via visual hull.*

Abstract

Mirror systems have recently emerged as an alternative low-cost multi-view imaging solution. The use of these systems critically depends on the ability to compute the background of a multiply mirrored object. The images taken in such systems show a fractured, patterned view, making edge-guided segmentation difficult. Further, global illumination and light attenuation due to the mirrors make standard segmentation techniques fail.
We therefore propose a system that allows a user to do the segmentation manually. We provide convenient tools that enable an interactive segmentation of kaleidoscopic images containing three-dimensional objects. Hereby, we explore suitable interaction and visualization schemes to guide the user. To achieve interactivity, we employ the GPU in all stages of the application, such as 2D/3D rendering as well as segmentation.

Categories and Subject Descriptors (according to ACM CCS): I.4.6 [Image Processing and Computer Vision]: Segmentation—Region growing, partitioning

1. Introduction

Until recently, surround multi-view imaging has required the use of hardware-parallel multi-camera systems [NRK98, WJV*05] or the time-sequential acquisition of different view points by moving the camera around the object [GGSC96]. An initial study that employed inter-reflections in mirror systems to acquire a surround light field view of a flat surface was presented by Han and Perlin [HP03]. The short-coming of their technique is that extended three-dimensional objects cannot be easily handled. The main challenge for using so-called kaleidoscopic images of extended three-dimensional objects is the occlusion of the virtual copies of an object by the object itself or by some other virtual copy, see Fig. 1a). Recently, Reshetouski et al. [RMSI11] presented a geometrical solution to this problem. The authors developed a segmentation scheme that is based on the approximation of the object geometry by means of visual hull reconstruction, Fig. 1d). The approximate geometry is then used to determine visibility of the object to different virtual perspective projections generated by the mirror system, Fig. 1e).

Their method relies on a segmentation of the kaleidoscopic image into foreground and background. The background pixels are used to determine the object geometry by space carving. The accuracy of this approach is determined by the quality of the image segmentation. The complexity of images in a mirror system makes automatic segmentation a non-trivial task. Using edges [RKB04, UPT*08] is difficult due to the numerous non-object edges arising from mirror boundaries. Standard background subtraction, taking a

background image and subtracting it from the image containing the object, is challenging due to changing global illumination conditions. Similarly, color-based segmentation is difficult because of the attenuation introduced by the mirror surfaces.

For these reasons, we propose an intuitive environment to perform the segmentation manually, Fig. 1c). However, even for humans the task is usually non-trivial and tedious. Our interactive application enables a fast manual foreground/background segmentation of kaleidoscopic images for high-quality results by guiding the user and giving constant feedback Fig. 1b). In summary:

- We present an application, that allows for manual foreground/background segmentation and provides feedback to the user in the 2D kaleidoscope domain as well as the 3D domain of the visual hull.
- We describe an on-line technique for estimating the visual hull of an object inside a kaleidoscope system of arbitrarily positioned planar mirrors.
- We explore suitable interaction schemes and visualization techniques that guide the user in rapidly creating an accurate segmentation.
- We handle the heavy computations of drawing, rendering, visual hull derivation, and labeling by exploiting the GPU.

2. Related Work

Mirror systems are an attractive low-cost alternative to multiple-camera systems for the acquisition of multi-view imagery. Early work included virtual stereoscopic systems and the analysis of single camera/mirror configurations for the design of catadioptric camera systems. Han and Perlin [HP03] pioneered the use of kaleidoscopic imaging systems for recording a large number of virtual views with a low number of mirrors, exploiting inter-reflections inside the system. They used the system in a bi-directional manner by projecting light into it, which enables reflectance scanning. However, the imaged surface had to be flat in order to avoid self-occlusion of the object and its virtual counterparts. Levoy et al. [LCV*04] allow for three-dimensional objects by positioning the mirrors such that no inter-reflections, and therefore no occlusion can occur. In this setting, a separate mirror for every virtual camera is required. Alternatively, inter-reflections can be allowed, but the object needs to be positioned such that overlap in the camera image [FNJV06] is avoided, restricting flexibility. Two-mirror systems have also been used in an active illumination context [LCT09] and it has been shown that mirror systems with flexible object/camera positioning can be used for multi-view imaging [RMSI11] by decomposing the image into its constituent virtual views. The basis of the computation exploits the visual hull of the object under investigation from a single silhouette image. The method can be extended to decompose active illumination patterns such that they strike the object from a single direction only [IRM*12]. The applicability of such flexible new imaging techniques depends heavily on an accurate segmentation of the kaleidoscope image into foreground and background.

Segmentation is one of the fundamental problems in computer vision. In spite of its importance it is not considered a solved topic. The reason for this is that, usually, semantic information dictates the goal of a segmentation algorithm. Often, these goals are user-dependent. Since the topic is vast, we can only give a coarse overview. The earliest approaches to the problem were based on color classification schemes and thresholding, e.g. in suitable color spaces and by exploiting histogram information [SS04]. Region-based approaches are based on identifying areas of similar color, an example is watershed segmentation [VS91]. A major step in the development of segmentation algorithms came with the introduction of active contours [KWT88], deformable curves that adapt to edges while minimizing some energy functional such as the length of the boundary curve of the segmented region. Initially, these energies were minimized using parametric curve models. Geodesic active contours [CKS97] formulated the task as a level set problem which allows for arbitrary topology of the segmented parts. The energy functionals can, e.g., be minimized with graph cuts [BJ00] but also with convex relaxation schemes [UPT*08]. Since the goals of the user are usually not known beforehand, a number of interactive segmentation methods utilizing the energy minimization schemes described above have been developed [RKB04, CFRA07, UPT*08]. Finally, alpha matting [CCSS01] can be regarded as a continuous-valued segmentation of the image whereas the techniques discussed before result in a binary segmentation.

The ultimate goal of our paper is an interactive multi-valued segmentation of an image taken within a system of planar mirrors, Fig. 1e). The image is to be decomposed into a number of virtual views that are present in the image. Each pixel is to be assigned a view-projection matrix for the corresponding view. The basis for this is a silhouette image, a binary segmentation of the camera image in conjunction with the calibration parameters of the camera/mirror system [RMSI11], Fig. 1a, c). We employ a high-level symmetry prior (the kaleidoscope geometry) in this work in contrast to the above techniques. In the case of kaleidoscopic image segmentation it is not sufficient to apply the low-level prior of preferring strong edges while maintaining a short length of the boundary curve. There are two main reasons. First, many fractured multiply mirrored intersection lines between mirrors result in a high total count of strong edges that do not demark object boundaries. Second, on the order of hundreds of virtual objects are present in a single kaleidoscopic camera image, resulting in rather long and inhomogeneous boundaries between foreground and background.

Cosegmentation has recently been introduced as a novel tool for segmenting a large number of images showing similar objects simultaneously. The task usually explored in this

area is the segmentation of internet image collections. Exemplary works use scribbles in a number of images in order to segment a much larger collection [BKP*10, KBCC10].

While an interesting avenue for future work towards an automated solution, currently none of the proposed techniques are applicable to our problem. We therefore aim our efforts at developing an easy to use manual tool for the kaleidoscopic segmentation problem that provides instant feedback and thus enables high-quality segmentation results.

3. Overview

We start by describing the setting of our system. We assume to have a calibrated setup consisting of a camera C, a kaleidoscope S containing planar mirrors, and an object O with unknown geometry inside of it. Due to reflections, the camera image K, taken by C, contains a combination of several views that are fused together into a single multi-view image. As shown in [RMSI11], K can be decomposed into parts that correspond to a single virtual view of the object. This decomposition is called a *labeling* L of K (see Fig. 1e)).

The principle of a kaleidoscope is illustrated in Figure 2. It shows a number of reflected ray paths of different reflection orders shown in different colors. The unfolding procedure introduced in [RMSI11] straightens the ray paths by mirroring the physical system instead of the rays, creating a representation of the virtual mirror world. In this representation, the physical camera is being multiplied into a number of virtual counterparts illustrated by the dashed lines, each describing a different virtual view. The virtual camera rays correspond to reflected view rays of the actual camera that is used to capture K.

We use the method of [RMSI11] to obtain the total set of virtual cameras $\{C1, C2, ..., Cn\}$ in conjunction with their view-projection matrices. A *virtual-camera image* $\mathcal{C}$ can be used to store for every pixel in the kaleidoscope image K an associated set of potential virtual cameras, i.e., for every pixel $p(\mathcal{C})$, we have a set of associated virtual cameras: $\mathcal{C}_p = \{Cp_1, Cp_2, ..., Cp_{n_p}\}$, $n_p < N$, where N is the maximum level of reflections that we take into account in the decomposition (typically $N = 7$), which is reasonable because each mirror attenuates the image. We refer to the set of pixels that are potentially assigned to a virtual camera as its *footprint*. This set can be precomputed, as it only depends on the configuration of S and C. Any cone through a pixel from the camera center can be followed through S. Care has to be taken when a cone is near the border between two mirrors. Here, it is split into two different light paths and our assumption of a virtual camera would fail. Hence, we exclude these pixels entirely from our considerations. The process is fast and maps nicely on the GPU.

As soon as we have an object O, rays might not follow their original path inside the kaleidoscope S, but might be intercepted by O. The ray path of the camera pixels are usually

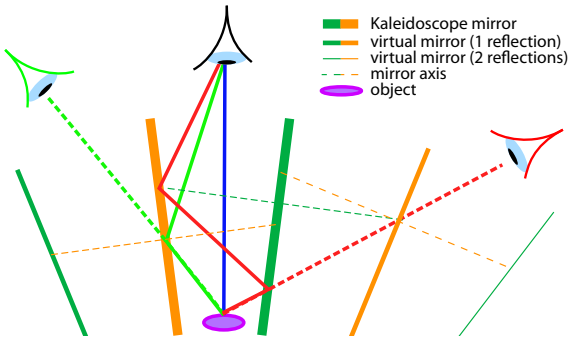

Figure 2: *Unfolding of the ray trajectories in the kaleidoscope results in virtual camera and object positions.*

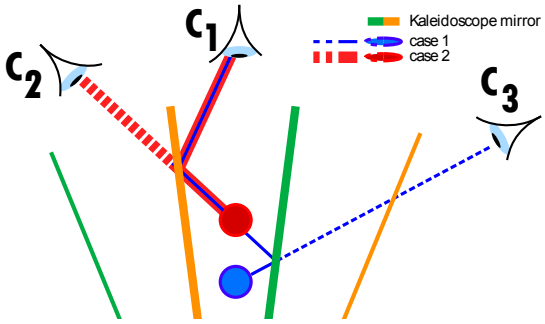

Figure 3: *Placing an opaque object inside the mirror system determines the virtual camera that sees the object from a set of potential cameras $C1, C2, C3$. For the blue object position, the ray undergoes two reflections before hitting the object resulting in camera $C3$ to be the virtual view. For the red object position there is only a single physical reflection event before hitting the object, resulting in the virtual camera $C2$ to be the virtual view.*

shorter than the maximum ray path utilizing all N reflections. The presence of O, hence, reduces the actual sequence of virtual cameras for pixel $p(\mathcal{C})$: $\{Cp_1, Cp_2, ..., Cp_{n'_p}\}$, where $n'_p \leq n_p$. In Fig. 3, we show how the insertion of an object into the kaleidoscope can determine a particular virtual camera from $\mathcal{C}_p$ for a pixel p.

The goal of this paper is to develop an interactive technique for determining the correct assignment of one virtual camera ($Cp_{n'_p}$) to every image pixel in a kaleidoscope image K. Hence, the kaleidoscopic image can be decomposed into its constituent views, enabling the application of standard multi-view geometry reconstruction techniques as a post-process. Hereby, the origin of the multi-view data is abstracted and makes it appear as if it had been recorded with a physical system of a large number of physical cameras.

We rely on a semi-automatic process for labeling with suitable editing and visualization tools that help the user to interactively achieve the goal with little effort. In the following, we will detail the steps of our algorithm.

The input to our algorithm is the kaleidoscope image K with the associated *virtual-camera image* $\mathcal{C}$. Our goal is to label the pixels of K; a pixel in the resulting label image L is either classified as background or we associate the corresponding virtual camera ($Cp_{n'_p}$). Determining L manually is difficult; one would not know which virtual camera to associate. Instead, we offer a simpler solution; one only marks background pixels (those not showing any part of the object). We propose an easy-to-use and efficient user interface (Sec.3.1) because despite the input being binary, automatic methods are likely to fail.

Internally, our algorithm uses the background-pixel input to update and determine the labeling (Sec. 3.2). The principle is to employ a space-carving process [RMSI11]. In fact, every ray corresponding to a background pixel can be used to determine empty regions of the bounding box B of O, i.e., where there is no object. By marking sufficiently many background pixels and removing the corresponding regions from B, the visual hull of the object is reconstructed on the fly. Using the visual hull, a labeling can be determined. At the same time, we inherit the limitations of [RMSI11] by using the visual hull instead of the true object geometry.

Because our solution interactively updates a 3D reconstruction, we can use it directly to guide the user when marking background pixels (Sec.3.2). Further, we can shade the model with a novel image-based technique, that applies surface color from K, capturing even view-dependent effects, to ease understanding of the shape (Sec.3.3).

3.1. User Interaction

In theory, the user could click on each pixel in K and associate it to the background or the object. In practice, such an approach would be cumbersome. Instead, we propose an on-the-fly computation of the visual hull while attributing pixels to the background. Hereby, the user can visualize the influence of his indications on the reconstruction and then add pixel classifications only where needed.

Our interface offers various means to mark regions, including tools, such as different brushes and brush sizes, eraser, magic wand, flood fill, and undo. These tools are used to mark pixels in K as background. Starting with the magic wand and flood fill, one can then rely on larger, afterwards finer brush strokes. Every change, triggers visual hull updates (Sec.3.2), which are illustrated efficiently (Sec.3.3) and we overlay the resulting label image L on the canvas K.

While it would be possible to store a single mask, it is more user friendly, to store the regions defined by one of the previous drawing operations in a so-called *patch*. These patches are small textures that are kept together with an offset to localize them in K. All patches are shown with a small preview in a sidebar of the interface and the user can simply decide to remove a patch by deleting it from the sidebar. The second advantage of keeping these interactions in patches is that we can employ various resolutions for the different patches, in case sub-pixel drawing is necessary. Technically, we store the patches as binary textures and their content is *OR*ed when being splat into the view. Usually, each brush stroke can be added with a single splat, yet, when deleting a brush stroke, all patches are splat. In practice, this operation remains fast enough to not disturb the user.

Each time a new background pixel is added, we read its underlying camera set $\mathcal{C}_p$. We can conclude that the corresponding viewing rays of all cameras missed the object. This observation implies that a single background pixel actually delivers information about various viewing rays at once. In order to make use of this insight, we compute and display the reconstructed visual hull. With this visualization, the user can easily decide whether a sufficient amount of pixels have been classified (see Fig. 4). Vice versa, a click on a voxel in the current 3D reconstruction reveals all its corresponding pixels in K. We further allow the user to re-fill voxels. Here, the user clicks on a voxel and spans a sphere. We then back-project all voxels inside the sphere to K and erase the background mask at those pixels. We will see how to perform the visual hull reconstruction from the background mask and how to even shade the resulting 3D model convincingly in the following sections.

3.2. Efficient Visual Hull Reconstruction and Labeling

The visual hull is initially represented by a voxel volume defined by B. All voxels are initialized to one. For each viewing ray resulting from a background pixel, we can carve (set to zero) several voxels from the volume. Consequently, only a few region indications are often enough to yield a good reconstruction of the model. To understand this point, one can consider a simple example. When no object is placed in the kaleidoscope, all rays are background. Marking only those that are in the footprint of one camera is often sufficient because, then, B will be found to be empty.

To keep the updates of the visual hull efficient, we make use of several observations. First, changes to the visual hull only occur where the user placed a background brush stroke. Hence, we do not reconstruct everything in each frame, but restrict the visual hull update to this particular region. To perform the voxel carving for a given background pixel p, we perform a ray marching procedure following the corresponding straightened viewing ray for each of the virtual cameras in $\mathcal{C}_p$. The use of the virtual camera concept avoids intersection computations with the mirror planes and we can simply intersect the bounding box of the object with each camera ray in order to determine the ray marching region. The per pixel, virtual camera ray marching can be performed in parallel, even without the need for memory synchronization, making it highly suitable for the GPU.

Care has to be taken to not miss voxels. So, instead of performing a fixed-step ray marching, we rely on an accurate

method [AW87], which can be efficiently implemented on the GPU. The idea is to advance from one plane along the principle axes to the next, where all planes are defined by the voxels' faces. For every voxel traversed by the ray, we update the visual hull (represented as a 3D texture) and set the corresponding voxel value to zero.

To compute a labeling using the visual hull, we go over $\mathcal{C}$ and launch for each virtual camera and pixel a corresponding ray against the visual hull in a shader. Starting with the lowest camera index (the least reflections), we can stop the loop per pixel, as soon as a tested ray leads to an intersection. The corresponding camera index is then stored in L.

We experimented with various implementations. Ultimately, the best performance was reached, when storing all virtual camera matrices as a group of four *vec4* in a 1D texture buffer. The voxel volume was best represented with an 8bit 3D texture containing a single color channel and updated using the new OpenGL image-load-store extension. To reduce memory consumption, we also tried solutions that pack groups of voxels into bit sequences [ED08, SS10], e.g. packing a group of 4x4x4 voxels in a RG32UI texture. Unfortunately, this requires read-write synchronization (atomic texture access) for the visual hull computation, which makes the computation slow. Furthermore, despite the lower bandwidth, even the display of the visual hull, as explained in the next section, proved too slow because the bit sequences needed to be reinterpreted.

3.3. Visual Hull Display

Having a visual hull representation efficient display methods are needed for visual feedback. We provide two different ones; a direct rendering of the visual hull via ray marching, as well as an advanced mode that shades the model via an image-based solution using derived normals.

The standard mode that shows the volume as a semi-transparent object is useful when wanting to illustrate all its intersections with virtual rays corresponding to a drawn patch (see Fig. 4). Here, we apply a simple emission and absorption model, giving the voxels specified colors and densities depending on whether they are part of the visual hull, the current patch, or the intersection of both.

In order to produce a realistic view, we make use of K to shade the object. Here, we apply a deferred rendering approach. In a first rendering pass, we create a position and normal image of the first visible layer of the visual hull for the current view camera. The images are used in a second pass, where we perform the actual shading, accessing K. Finding the first visible layer is again achieved via ray marching. While voxels have 6 different normals (one for each face) that we can directly use for visualization purposes, better results are obtained by computing normals from an isosurface of a downsampled volume. In other words, we compute an average density of the voxels in a small neighborhood. Next, we compute a normal via finite differences. Depending on the neighborhood size, these *surface normals* are usually of sufficient quality. In the second pass, we project the first visible voxel V to all virtual camera views. For each view, we check via L whether V is classified as being visible by this view. For each corresponding color value C_i^c from view i, we compute a weight w_i and display the color $\sum w_i C_i^c$ on the screen. If V is not visible (label is not valid) for camera i, the surface normal n is opposing the virtual camera direction C_i^ω, or the virtual camera direction is opposing the view camera direction ω_p, then w_i is zero. Else, we define $w_i := (C_i^\omega \cdot \omega_p)\,(C_i^\omega \cdot n)$.The first factor favors virtual-camera views close to the current view (to capture specular effects), the second term ensures that no false views are considered. Fig. 4 shows results of this method.

Figure 4: *Semi-transparent mode (left): Depending on its state, each voxel is assigned color and density: part of the visual hull (white, low density), covered by the brush (blue, medium density), intersection of both (red, high density; top part of image). Image-based shading (middle and right): Color values are transferred from the kaleidoscope image.*

4. Results

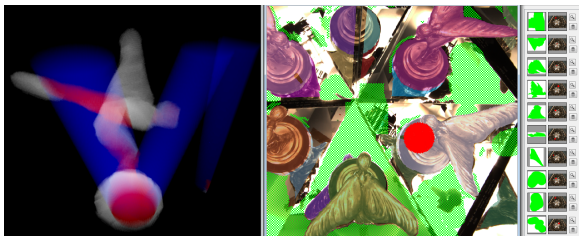

Figure 5: *Showing the application in action: The user draws in the window on the right side. Already drawn patches are rendered with a green checkerboard pattern and are listed in a widget at the right border. The current visual hull and a preview of the current brush are rendered in the left window.*

Our software provides the possibility of direct user feedback on a standard computer system with an NVIDIA GeForce 560Ti. Even when just hovering the mouse, the application shows the effect of the potential drawing operation in 3D, see Fig. 5. During drawing, the visual hull visualization is constantly updated, giving the user instant feedback in the 3D domain. This gives a clear advantage over drawing with

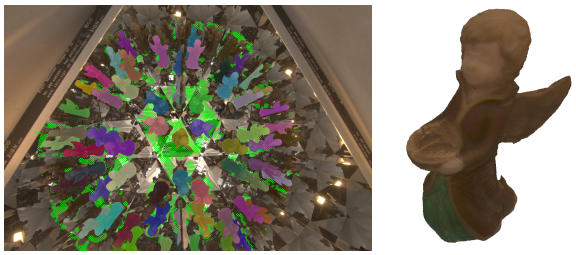

Figure 6: *Left: Labeling result for the angel. Right: Corresponding visual hull rendered with image-based shading.*

standard image-editing tools like Photoshop and performing a *blind* visual hull reconstruction as in previous work. Further, the labeling can be quickly recomputed after each operation and visualized in the 2D kaleidoscope image (although this particular step is not optimized in our current implementation, it takes less than 400ms for a 20MP image). Via the feedback, the user is guided towards areas where a refinement of the background mask is useful, making the entire labeling very simple. Similarly, the concurrent 3D visualization and label image help the user in adjusting false background markings. In such a case not just the 3D visualization will look incorrect, also the labeling does not correspond to the kaleidoscope color image anymore. Direct feedback mechanisms enable quick and correct background markings.

While the labeling is a two-dimensional feedback, the 3D visual hull can be useful as well. If the user marks pixels in K, the corresponding ray bundles can affect many parts of the 3D space at once. The semi-transparent rendering of the visual hull and the ray bundles enables the user to quickly estimate the voxels part of the ray bundles in relationship to the voxels of the visual hull.

The application itself scales well with a high number of strokes. Memory consumption is kept low as stroke data is stored in form of compact patches. Further, visual hull updates are fast, especially when limiting to the bounding box of a stroke, leading to instant feedback of the 3D visual hull (less than 200ms). Operations like undo, erasing, or deleting strokes are slightly more costly as they require to process all strokes and, hence, a full reconstruction of the visual hull. Here, the computation time may reach around a second, and a delay can be experienced. Nonetheless, we observed that those operations are not very often performed and the entire labeling process stays user-friendly.

Overall, the system allows a user to rapidly segment an object and the examples in this paper all involved less than 15 minutes (usually 10-15) of user interaction. The effectiveness is best illustrated in the accompanying video. An additional example is shown in Fig. 6.

5. Conclusions

Kaleidoscope systems have many advantages over multi-camera systems; they are cheap and easy to use. Nonetheless, separating the fused views can be difficult. Our solution is an easy-to-use system that allows a user to rapidly segment object pixels from the background. The input is automatically transferred to other parts of the image by relying on tests against an on-the-fly construction of the visual hull of the object. We presented an efficient algorithm to perform these computations and various rendering strategies to support the user in its task. With our system, the use of kaleidoscope imagery can become an interesting alternative to more complex multi-camera setups.

An interesting area for future work is the extension of our solution by adding more automation to the segmentation process. This can be done, for example, by incorporating user-supervised cosegmentation or GraphCut-based approaches into the existing system.

Acknowledgements

This work was supported by the German Research Foundation (DFG) through the Emmy-Noether fellowship IH 114/1-1 and by the Intel Visual Computing Institute at Saarland University.

References

[AW87] AMANATIDES J., WOO A.: A Fast Voxel Traversal Algorithm for Ray Tracing. In *Eurographics '87* (1987), pp. 3–10. 5

[BJ00] BOYKOV Y., JOLLY M.-P.: Interactive Organ Segmentation using Graph Cuts. In *Proc. of MICCAI, LNCS 1935* (2000), Springer, pp. 276–286. 2

[BKP*10] BATRA D., KOWDLE A., PARIKH D., LUO J., CHEN T.: iCoseg: Interactive Co-segmentation with Intelligent Scribble Guidance. In *Proc. CVPR* (2010), pp. 1–8. 3

[CCSS01] CHUANG Y.-Y., CURLESS B., SALESIN D., SZELISKI R.: A Bayesian Approach to Digital Matting. In *Proc. CVPR* (2001), pp. 264–271. 2

[CFRA07] CREMERS D., FLUCK O., ROUSSON M., AHARON S.: A Probabilistic Level Set Formulation for Interactive Organ Segmentation. In *Proc. of SPIE Medical Imaging* (2007). 2

[CKS97] CASELLES V., KIMME R., SAPIRO G.: Geodesic Active Contours. *IJCV 22*, 1 (1997), 61–79. 2

[ED08] EISEMANN E., DÉCORET X.: Single-Pass Solid Voxelization for Real-Time Applications. In *Proc. of Graphics Interface* (2008). 5

[FNJV06] FORBES K., NICOLLS F., JAGER G. D., VOIGT A.: Shape-from-Silhouette with two Mirrors and an Uncalibrated Camera. In *Proc. ECCV* (2006), pp. 165–178. 2

[GGSC96] GORTLER S. J., GRZESZCZUK R., SZELISKI R., COHEN M. F.: The Lumigraph. In *Proc. SIGGRAPH* (1996), pp. 43–54. 1

[HP03] HAN J. Y., PERLIN K.: Measuring Bidirectional Texture Reflectance with a Kaleidoscope. In *Proc. SIGGRAPH* (2003), pp. 741–748. 1, 2

[IRM*12] IHRKE I., RESHETOUSKI I., MANAKOV A., TEVS A., WAND M., SEIDEL H.-P.: A Kaleidoscopic Approach to Geometry and Reflectance Acquisition. In *Proc. Workshop on Computational Cameras and Displays* (2012), pp. 1–8. 2

[KBCC10] KOWDLE A., BATRA D., CHEN W.-C., CHEN T.: iModel: Interactive Co-segmentation for Object of Interest 3D Modeling. In *Workshop on Reconstruction and Modeling of Large-Scale 3D Virtual Environments* (2010), pp. 1–14. 3

[KWT88] KASS M., WITKIN A. P., TERZOPOULOS D.: Snakes: Active Contour Models. *IJCV 1*, 4 (1988), 21–31. 2

[LCT09] LANMAN D., CRISPELL D., TAUBIN G.: Surround Structured Lighting: 3-D Scanning with Orthographic Illumination. *CVIU 113*, 11 (2009), 1107–1117. 2

[LCV*04] LEVOY M., CHEN B., VAISH V., HOROWITZ M., MCDOWALL I., BOLAS M.: Synthetic Aperture Confocal Imaging. *ACM TOG 23* (August 2004), 825–834. 2

[NRK98] NARAYANAN P. J., RANDER P., KANADE T.: Constructing Virtual Worlds using Dense Stereo. In *Proc. ICCV* (1998), pp. 3–10. 1

[RKB04] ROTHER C., KOLMOGOROV V., BLAKE A.: GrabCut: Interactive Foreground Extraction using Iterated Graph Cuts. *ACM TOG 23*, 3 (Aug. 2004), 309–314. 1, 2

[RMSI11] RESHETOUSKI I., MANAKOV A., SEIDEL H.-P., IHRKE I.: Three-Dimensional Kaleidoscopic Imaging. In *Proc. CVPR* (2011), pp. 353–360. 1, 2, 3, 4

[SS04] SEZGIN M., SANKUR B.: Survey over Image Thresholding Techniques and Quantitative Performance Evaluation. *SPIE JEI 13*, 1 (2004), 146–168. 2

[SS10] SCHWARZ M., SEIDEL H.-P.: Fast Parallel Surface and Solid Voxelization on GPUs. *ACM Trans. Graph. 29*, 6 (Dec. 2010), 179:1–179:10. 5

[UPT*08] UNGER M., POCK T., TROBIN W., CREMERS D., BISCHOF H.: TVSeg - Interactive Total Variation Based Image Segmentation. In *Proc. BMVC* (2008), pp. 335–354. 1, 2

[VS91] VINCENT L., SOILLE P.: Watersheds in Digital Spaces: An Efficient Algorithm based on Immersion Simulations. *IEEE Trans. PAMI 13*, 6 (1991), 583–598. 2

[WJV*05] WILBURN B., JOSHI N., VAISH V., TALVALA E.-V., ANTUNEZ E., BARTH A., ADAMS A., HOROWITZ M., LEVOY M.: High Performance Imaging using Large Camera Arrays. *ACM TOG 24*, 3 (July 2005), 765–776. 1