Real-Time Space Carving Using Graphics Hardware

Christian NITSCHKE†, Atsushi NAKAZAWA††(a), Nonmembers, and Haruo TAKEMURA††, Member

SUMMARY Reconstruction of real-world scenes from a set of multiple images is a topic in computer vision and 3D computer graphics with many interesting applications. Attempts have been made to real-time reconstruction on PC cluster systems. While these provide enough performance, they are expensive and less flexible. Approaches that use a GPU hardware-acceleration on single workstations achieve real-time framerates for novel-view synthesis, but do not provide an explicit volumetric representation. This work shows our efforts in developing a GPU hardware-accelerated framework for providing a photo-consistent reconstruction of a dynamic 3D scene. High performance is achieved by employing a shape from silhouette technique in advance. Since the entire processing is done on a single PC, the framework can be applied in mobile environments, enabling a wide range of further applications. We explain our approach using programmable vertex and fragment processors and compare it to highly optimized CPU implementations. We show that the new approach can outperform the latter by more than one magnitude and give an outlook for interesting future enhancements.

key words: 3D reconstruction, space carving, photo hull, visual hull, graphics hardware, GPU programming, real-time

1. Introduction

Reconstruction of the 3D space is widely used for creating realistic scene models or digitizing scenes with its current properties [1]. If the reconstruction can be obtained in real-time, a broad field of further applications is facilitated. Dynamic scenes enable for more realistic support in collaborative working [2] and entertainment [3], [4]. Moreover, high-level scene understanding techniques can be applied, e.g. non-intrusive human motion acquisition [5]. When reconstructing static scenes, real-time performance allows for interactive parameter adjustment and immediate feedback.

We focus on passive optical methods using camera images from multiple viewpoints [6]. Our aim is to present an algorithm for real-time volumetric reconstruction, gaining acceleration by the use of current computer graphics hardware (GPU). Two important approaches are shape from silhouette [7] and shape from photo-consistency [8]. The former rely on a robust image segmentation and can not recover smooth convex and concave surface patches not encoded in the silhouette images. As these are fast and simple to compute they are often applied where performance matters. The latter provide a high quality shape reconstruction and the possibility of approximating per-voxel color in a straightforward way. However, they suffer from a high computational cost. An interesting fact is that shape from photo-consistency can be seen as a generalization of shape from silhouette where further constraints are added [8]. The proposed approach exploits this relation by applying both techniques in an approximation and refinement process.

To achieve real-time volumetric reconstruction, attempts have been made to develop highly scalable parallel algorithms for PC cluster systems [9], [10]. However, by using inflexible expensive hardware their application is restricted to professional and static environments. On the other hand, an off-the-shelf GPU outperforms a CPU in SIMD (single instruction/multiple data) computations [11]. For approaches using the GPU, real-time is only achieved for generating a single novel viewpoint, not an entire volumetric model. In this context, Lok [12] proposes a plane-sweep approach for retrieving a 3D surface, suffering from high algorithmic complexity. Instead Li et al. [13] construct the camera silhouette cones and perform a volume intersection by making clever use of rendering techniques. While these methods perform reconstruction using shape from silhouette, Li et al. [14] continue proposing a plane-sweep technique for shape from photo-consistency. Yang et al. [15] present an approach to image-based stereo where a depthmap is obtained for a novel viewpoint. As they use a plane-sweeping to determine the best matches, the work is similar to the one of Li et al. [14]. From Matusik et al. [16] and Slabaugh et al. [17] there exist real-time approaches to shape from silhouette and shape from photo-consistency on the CPU, however using low output resolution and undersampling. The common ground of all these works is that the reconstruction (1) covers only a part of the scene, (2) is view-dependent as space is sampled from a particular viewpoint using a subset of nearby cameras and (3) is not explicitly represented if recovered on the GPU.

The similarity to our approach is that the scene is restored from particular novel viewpoints. However, we perform a volumetric reconstruction with a view-independent sampling of a regular grid of voxels. Further, we propose a setup of six perpendicular viewpoints as destination cameras from where the scene is recovered. This special setup helps to decrease algorithmic complexity and achieve real-time capability.

The purpose of our work lies in recovering an en-
tire volumetric model. There exist other approaches for this task. Hasenfratz et al. [3] perform a plane-sweeping on a regular grid of voxels. They achieve real-time performance, however by using a basic shape from silhouette technique with low volumetric resolution. Sainz et al. [18] present a multi plane-sweep implementation of space carving [8]. The approach suffers from many expensive CPU-GPU data transfers in each sweeping pass, leading to a performance that is far from real-time. More recently, Zach et al. [19] also propose a multiple plane-sweep approach avoiding expensive transfers by keeping all necessary data on the GPU. Although the system makes better use of hardware-acceleration, it does achieve interactive or real-time framerates.

Instead of the commonly used plane-sweeping, we propose a ray-based volume sampling. For every viewpoint around the scene we render a set of parallel rays aligned with the volumetric grid. The voxels along each ray are sampled employing a two-step strategy of silhouette and photo-consistency testing. If a consistent voxel is found, its position and color are written to texture and read-back to the CPU. Beside enabling load-balancing between vertex and fragment processing, the ray-based sampling preserves us from unnecessary testing and transferring interior voxels. Moreover, instead of explicitly keeping track of voxel visibility, we propose a novel heuristic to implicitly account for that. As the whole set of input cameras is used in testing a particular voxel, the partial reconstructions form up a global consistent model. Hence, an expensive merging and alignment operation as known from image-based stereo methods is not necessary.

Currently, our system is applied in an offline context where multi-viewpoint video is captured and transferred to the reconstruction server in advance. Up to our knowledge, we are the first group doing the necessary image processing entirely on the GPU, enabling the future application of our space carving algorithm for mobile environments. Refer to Fig. 1 for a feature comparison.

The remainder of this paper is organized as follows. Section 2 introduces a basic hardware-independent reconstruction algorithm that is extended and mapped to the GPU in Sect. 3. Section 4 explains a concrete system setup with experimental results, followed by Sect. 5 which discusses current limitations and future enhancements. A conclusion is given by Sect. 6.

2. Basic Algorithm

2.1 Image Processing

The setup consists of N static calibrated source cameras \(S_k = 1, \ldots, N\). For later image segmentation, it is necessary to acquire a set of background images \(B_k\) without any objects to be reconstructed. The images are compensated for lens distortion.

At every timeframe \(t\), a set of images \(I_k\) is captured with the objects present. These are used to compute a set of binary foreground images representing the silhouettes of the object \(F_k = I_k - B_k\). Applying a distance measure \(d_F\) with a threshold \(\tau_F\) for a pair of color values at the pixels \(I_k(u,v)\) and \(B_k(u,v)\) we obtain a simple classifier to decide, if a pixel is occupied by either object or background:

\[
F_k(u,v) = \begin{cases} 
1 & d_F(I_k(u,v), B_k(u,v)) > \tau_F \\
0 & \text{otherwise}
\end{cases}
\]  

For the distance measure \(d_F\) we apply the \(L^2\)-norm over CIELab color space and assign a smaller weight for the lightness component. This results in a shifting of \(d_F\) towards a pure chromaticity distance what compensates for shadows introduced by insufficient lighting conditions.

2.2 Reconstruction

We assume a tessellation of the reconstruction space into discrete voxels \(V(x,y,z)\). The voxels are binary labeled as either transparent or opaque, where latter have a color value assigned. The initial set of opaque voxels is defined by a given bounding box and voxelsize.

2.2.1 Maximal Volume Intersection

For every voxel \(V\) in the bounding box we test if it projects to valid pixels in the set of camera image planes \(S_k(u_k, v_k)\). If this fails for at least one camera, \(V\) is not included in the Maximal Volume Intersection \(MVI\) and carved by labeling transparent. This computation can be performed in advance.

2.2.2 Visual Hull Reconstruction

For the framewise reconstruction we consider only the set of voxels \(V \in MVI\). All of its projections \(F_k(u_k, v_k)\) have to be contained in the silhouette of the object. Otherwise, \(V\) is carved by labeling transparent. All remaining voxels belong to the approximation of the visual hull \(VHA\) which is known to enclose the object [7].

2.2.3 Photo Hull Reconstruction

Every remaining voxel \(V \in VHA\) is tested for photo-consistency of its projection pixels \(I_k(u_k, v_k)\). The color val-
ues of the pixels are not directly used to carry out the photo-consistency test. Since we do not maintain voxel visibility for the source cameras, we can not easily decide which cameras should contribute to the test. Therefore a decision about active cameras is made by computing a score value for each camera \(S_k\), based on similarities between pairs of color values. A higher score indicates that a voxel is more likely to be seen by the related camera.

(1) Active Camera Decision

For a Voxel \(V\), Color distances \(d\) are defined as the \(L^2\)-norm over CIELab color space between a pair of color values at a voxel’s projection in two distinct images \(I_k(u_k, v_k)\) and \(I_l(u_l, v_l)\). Unlike for image segmentation, no weighting is applied. CIELab is used as it linearizes the perception of color differences. This property guarantees that two distances having the same value taken from different pairs of colors are perceived with the same amount [20].

\[
d(I_k(u_k, v_k), I_l(u_l, v_l)) | k, l = 1, \ldots, N \land k \neq l
\]  

For each camera \(S_k\), the minimal distance \(d_{k,\text{min}}\), the maximal distance \(d_{k,\text{max}}\) and the range \(r_k\) between these values are computed.

\[
d_{k,\text{min}} = \text{MIN} \ d(I_k, I_l)
\]
\[
d_{k,\text{max}} = \text{MAX} \ d(I_k, I_l)
\]
\[
r_k = d_{k,\text{max}} - d_{k,\text{min}}
\]

Further, a scale \(s_k \in [0, 1]\) is determined from the minimum and maximum over all \(d_{k,\text{min}}\) and applied to the ranges \(r_k\). The scale gets big for small color distances which are thus favored.

\[
d_{\text{min, min}} = \text{MIN} \ d_{k,\text{min}}
\]
\[
d_{\text{min, max}} = \text{MAX} \ d_{k,\text{max}}
\]
\[
s_k = \frac{d_{\text{min, max}} - d_{\text{min, min}}}{d_{\text{min, max}} - d_{\text{min, min}}}
\]
\[
r_k \leftarrow s_k \cdot r_k
\]

Normalizing the ranges \(r_k\) leads to the score \(sc^c_k \in [0, 1]\) for deciding active cameras.

\[
r_{\text{min}} = \text{MIN} \ r_k
\]
\[
r_{\text{max}} = \text{MAX} \ r_k
\]
\[
sc^c_k = \frac{r_k - r_{\text{min}}}{r_{\text{max}} - r_{\text{min}}}
\]

With this algorithm, the score \(sc^c_k\) is depending on (a) the minimal distance \(d_{k,\text{min}}\) and (b) the range between minimal and maximal distance \(r_k\). Cameras where (a) is small and (b) is big are favored by obtaining a big score. This simulates (a) a very simple color space clustering for determining dense regions. Further employing of the ranges \(r_k\) (b) makes the score more robust for reconstructing object parts of similar color. Hence, colors of dense regions that have a high distance to the color scatter are favored.

Thresholding the scores \(sc^c_k\) leads to a binary weight \(w_k\) encoding the active camera decision.

\[
w_k = \begin{cases} 1 & sc^c_k \geq \tau \text{sc} \\ 0 & \text{otherwise} \end{cases}
\]

(2) Photo Consistency Test

We assume lambertian reflectance properties for the objects surface. A voxel’s photo-consistency \(C(V)\) is computed by thresholding the variance \(\sigma^2\) of the color values from all active cameras.

\[
\bar{\mu} = \frac{\sum_{k=1}^{N} w_k \cdot I_k(u_k, v_k)}{\sum_{k=1}^{N} w_k}
\]
\[
\bar{\sigma} = \frac{\sum_{k=1}^{N} w_k \cdot (I_k(u_k, v_k) - \bar{\mu})}{\sum_{k=1}^{N} w_k}
\]
\[
\sigma^2 = \frac{\sum_{k=1}^{N} w_k \cdot (I_k(u_k, v_k) - \bar{\mu})^2}{\sum_{k=1}^{N} w_k}
\]

\[
C(V) = \begin{cases} 1 & \sigma^2 < \tau \\ 0 & \text{otherwise} \end{cases}
\]

If a voxel is photo-consistent with the set of active cameras we assign \(\bar{\mu}\) as its color value, which is the mean over the color values of all active cameras. The voxel is said to be included in the photo-consistent surface \(V \in \text{PCS}\).

3. Hardware-Accelerated Algorithm

In the following, we apply the basic algorithm in a way that it fits to the programmable graphics transformation pipeline [11]. The developed concepts are independent of a specific programming environment. However, a prototype of the framework has been implemented using OpenGL and Nvidia Cg. Whenever necessary, detailed information will be given according to the prototype.

A preprocessing is done in advance. Each frame, image processing and reconstruction are performed by multi-pass rendering. The algorithm mainly relies on the following features of modern graphics hardware:

- Texture handling capabilities for accessing image data,
- OpenGL Vertex Buffer Objects Extension (VBO) [21] for a speedup when rendering equal content each frame,
- Programmable vertex and fragment processors for executing the space carving,
- OpenGL Framebuffer Objects Extension (FBO) [22] for storing the result in textures that can be read-back to the CPU.

For an overview of the process flow see Fig. 2. The following process description refers to a particular timeframe \(t\).
3.1 Texture Upload and Image Processing

All image data has to be uploaded to the GPU. While the static background images $B_k$ are transferred in advance, the images $I_k$ need to be updated each frame. Further, we are relying on Vertex Texture Fetches (VTF). Current hardware supports the access of up to four vertex textures with a maximal resolution of $4096 \times 4096$. Hence, we store the set of $N$ images in a single 2D texture by using off-sets. Our implementation uses eight XGA cameras, so the whole set fits into one texture. The two textures for $B_k$ and $I_k$ are created in advance with glTexImage2D(). Each frame, the upload is done using glTexSubImage2D().

After performing the image processing we obtain a set of color and foreground images $I_k$ and $F_k$ which are stored as textures. Since the images $F_k$ should be later accessed from a vertex program the corresponding texture is restricted to the 1x32 bit format GL_LUMINANCE_FLOAT32_ATI. Doing so, texture upload becomes the bottleneck. Experiments showed that it is faster to just upload the raw images $I_n$ in 3x8 bit format GL_RGB8 and perform the image processing on the GPU in an additional renderpass. Therefore, the camera is configured with the identity as modelview matrix and glOrtho2D(). Each frame, the upload is done using glTexSubImage2D().

For a constant number of virtual views, the algorithm complexity increases linearly with the number of source cameras. Applying the source camera views instead would result in a quadratic increase, similar to image-based stereo where a source camera pixel is triangulated with its corresponding pixels in every other camera. Moreover, by using the ray-sampling approach we do not need to test voxels behind the reconstructed surface. We can also guarantee to test a particular voxel not more than once. The advantage of rendering points is, that we are able to move some computation to the vertex processor which has not been used extensively by other approaches.

3.2 Destination Cameras

For the space carving, we cannot directly render and test the voxels $V \in MVI$. Also, using a plain-sweep approach as proposed in [8] has a high impact on performance. Instead, we define a set of six virtual views from where the scene is reconstructed. These views are referred to as destination cameras $D,i = 1, \ldots, M$ that are aligned like shown in Fig. 4 (a). The reconstruction is done by rendering the pixels of each destination camera image plane as 3D points. Starting at each point, a ray is constructed axis-aligned to the volumetric grid. The reconstructed surface is retrieved by incrementally sampling the voxels along such a ray till an intersection is found.

For a constant number of virtual views, the algorithm complexity increases linearly with the number of source cameras. Applying the source camera views instead would result in a quadratic increase, similar to image-based stereo where a source camera pixel is triangulated with its corresponding pixels in every other camera. Moreover, by using the ray-sampling approach we do not need to test voxels behind the reconstructed surface. We can also guarantee to test a particular voxel not more than once. The advantage of rendering points is, that we are able to move some computation to the vertex processor which has not been used extensively by other approaches.

A camera $D_i$ is configured for orthogonal projection using glOrtho(). This enables for equidistant sampling of the CPU. Undistortion, RGB-CIELab conversion and image segmentation occupy 20, 42 and 38 percent of the processing time. GPU upload/processing shows nearly similar performance compared to an upload of CPU-processed images. However, this ratio will change with future GPU generations since processing speed generally increases faster than memory bandwidth. The main advantage of using GPU image processing is the independence of any CPU processing time, making the CPU cluster redundant.
the volume with increasing distance from the image plane. The field of view is set according to the size of \( MVI \). Each camera is assigned to an FBO. The sizes for FBO and viewport correspond to the volumetric resolution (Fig. 4 (c)).

3.2.1 Interleaved Depth Sampling

As mentioned before, we do not test a voxel by more than one destination camera. This is realized using an Interleaved Depth Sampling (IDS) as shown in Fig. 4(d1)-(d2). IDS refers to a periodic shift in the sampling pattern among neighboring rays. The advantage is that we can increase the sampling distance by three times without increasing the sampling error. Nevertheless, this is only true when sampling unoccluded space which is seen by at least three perpendicular destination cameras. Otherwise, the sampling error may increase up to three times and smaller details are not sampled. However, the step of shape from photo-consistency refinement is not affected as IDS is not applied here. Due to the high performance gain of \( \sim 300\% \) we generally recommend the use of IDS.

3.2.2 Active Destination Cameras

Depending on the source camera setup, we are able to reduce the number of destination cameras. The idea of determining active destination cameras is that parts of the reconstruction space which are not visible by any source camera do not need to be sampled by the corresponding destination camera. For each destination camera, it is tested if there exists at least one viewing ray in any source camera leading to an angle \( \alpha \leq \pi/2 \) when intersected with the sampling direction of the respective destination camera (Fig. 4 (b)). Otherwise, the camera is labeled inactive and not considered for reconstruction.

3.3 Reconstruction

The reconstruction is carried out by rendering the pixels of the destination camera image planes as 3D points and constructing axis-aligned rays passing through the points into \( MVI \). Voxels are generated by incrementally sampling along
3.3.1 Vertex Data

In the following, each rendered pixel of a destination camera image plane is referred to as a vertex. Necessary per-vertex parameters are computed in advance and stored in texture memory using VBO. This prevents from expensive data uploads at runtime.

Each vertex is assigned with the two depth values \( MV1_{D(u,v)} \) and \( MV1_{D(u,v)} \), where the respective sampling ray \( R_{D(u,v)} \) enters and leaves \( MV1 \). Both depth values are generated according to IDS. Testing only voxels in the depth range \([MV10, MV1]\) prevents from expensive verification that all of its projections lead to valid source camera pixels.

An interesting property of our setup is that each ray intersects the two image planes of parallel destination cameras. Early Ray Carving (ERC) can be applied when the opposite destination camera has already been rendered. For each ray, a lookup into the corresponding texel in the rendered texture is done to test if the ray will hit the object. If not, all voxels along the ray can be carved without testing. Thus, respective coordinates for performing the texture lookup are assigned to the vertices in advance (Fig. 4 (e)).

3.3.2 Vertex Shader Visual Hull Computation

If ERC can not be applied, voxels are generated by sampling along the ray \( R_{D(u,v)} \). For each voxel is tested if it is contained in \( VHA \). When an intersection \( V^{VHA}_{R_{D(u,v)}} \) is found, the position is assigned to the vertex position. Otherwise, if no intersection can be found while reaching \( MV1 \), a position outside the camera viewing frustum is used. This implicitly carves the ray by the projection operation and insures that it is not passed further in the graphics transformation pipeline.

3.3.3 Fragment Shader Photo Hull Computation

If an intersection \( V^{VHA}_{R_{D(u,v)}} \) can be found by the vertex shader, the fragment shader continues sampling from this position within a given depth-threshold. Every voxel is tested for photo-consistency. If an intersection \( V^{PCS}_{R_{D(u,v)}} \) can be found, we assign the blended color value converted back from CIELab to RGB. Otherwise, we use a special color value outside the range of used colors, indicating that the ray is carved (Fig. 5). In this case, it might be necessary to fill holes caused by the failed photo-consistency test. This can be done by using the depth value along the ray

- representing \( V^{VHA}_{R_{D(u,v)}} \),
- achieving the minimal variance \( \sigma^2 \) for the photo-consistency test or
- resulting from the interpolation of the depth values from voxels found by neighboring rays.

(1) Modified Active Camera Decision

For increasing the robustness of the active source camera decision we add a view-dependent geometric score \( s_{ck} \in [0, 1] \). This helps to compensate for problems introduced by larger areas of uniform color, seen from source cameras at the other side of the scene. \( s_{ck} \) scores the angle \( \beta_k \) between the sampling direction \( \hat{d}_{ri} \) and the vector \((x, y, z)^T - S_k\) which points from the center of source camera \( S_k \) to the current voxel \( V \). It favors small angles.

\[
\beta_k = \arccos \left( \frac{(x, y, z)^T - S_k \cdot \hat{d}_{ri}}{\|d_{ri}\|} \right) \tag{17}
\]

\[
s_{ck} = \frac{\pi - \beta_k}{\pi} \tag{18}
\]

A compound score \( s_{ck} \in [0, 1] \) is constructed as

\[
s_{ck} = s_{ck} \cdot s_{ck}^2
\]

and thresholded instead of \( s_{ck}^2 \) to compute the binary weight \( w_k \) which encodes the active camera decision.

3.3.4 Read-Back of the Result to the CPU

The recovered values for position (3x32 bit) and color (3x8 bit) are written into texture using FBO. Current GPUs support up to four 4x32 bit Multiple Render Targets (MRT) in a single renderpass. As all MRT must have the same format we would waste download bandwidth when we store the values in two distinct textures. Instead, we use a single 4x32 bit texture where position is stored as 3x24 bit and color encoded in the remaining channel. A non-found intersection \( V^{PCS}_{R_{D(u,v)}} \) is assigned a special color value in the unused range of colors between 24 and 32 bit. At the CPU, we easily decode the voxel data from the texels.
4. Experimental Results

We have implemented our proposed method in a system with eight synchronized Sony XCD-X710 cameras where each is connected to a PC through IEEE-1394 Firewire. Each PC consists of an Intel Pentium4 3 Ghz CPU with 3.7 GB RAM and is used for image capturing. The images are sent to a single server PC for image processing and volumetric reconstruction. The server consists of an AMD AthlonX2-3800+ CPU with 3 GB RAM and an Nvidia Geforce 7800GTX PCI-Express graphics card with 512 MB texture memory running Microsoft Windows XP Pro x64 (Fig. 6). All cameras are calibrated through a simple marker-based method using AR Toolkit [23]. Geometrical relations between the cameras are obtained by using a known-sized calibration box representing the common world coordinate system.

The framework is applied in an offline context, so the images are captured in advance. For comparison we implemented two CPU versions. One, realizing the reconstruction through a simple independent testing of each voxel. The other, implementing the GPU approach of reconstruction by ray-sampling from virtual views. All implementations are compiled for the x64 platform using the Microsoft Visual Studio 2005 Compiler enabling performance optimizations. Therefore, we ported the GLUT and OpenCV libraries to x64.

4.1 Quality

We test our system reconstructing a human dancer in a volumetric bounding box of 1.1 m × 1.1 m × 1.8 m.

Shadows are a problem when performing image segmentation as they might be easily classified as foreground and thus belonging to the object. Increasing the segmentation threshold reduces the impact of shadows but the same time increases false classifications of object parts as background (Fig. 7(b)). This 'silhouette thinning' directly affects reconstruction quality. Employing the proposed color distance function with a small weight applied to the CIELab lightness channel results in a more robust segmentation (Fig. 7(c)).

Fig. 7 (d)-(o) present reconstruction results. Each row shows for a particular viewing camera (1) the result without
and with applying shape from photo-consistency (PCS) and (2) the corresponding depthbuffer images. Latter is used to indicate differences in shape that are crucial but not easily perceived from the color images. Generally, it can be observed that computing PCS achieves better color blending where bleeding effects can be widely removed. Considering the shape, concavities and further details are recovered. Especially for a difficult scene as in row (l), PCS reveals several details of the body, e.g. the straight right arm, the free space below this arm and the hole between legs and belly. Unfortunately, holes are introduced by reconstructing scene parts having less color variation using a high consistency threshold.

4.2 Performance

Five distinct configurations are used to compare the GPU and CPU implementations. The provided framerates are average measurements. For the GPU, image resolution matters due to the obligatory texture upload. Thus, we compare system performance using QVGA and XGA resolution. In the following, the experimental results in Fig. 8 are discussed.

At small voxelsizes the performance for the GPU gets bound by rendering and converges. The balance shifts towards texture upload when the size is increased. At very big voxelsizes the GPU is even outperformed by the CPU. For the CPU, we observe an intersection of the performance curves. At a small voxelsize the ray approach shows superior behavior since less interior voxels have to be tested for photo-consistency. On the other hand, due to its constant overhead the ray approach shows a poor performance for an increased voxelsize. The speed-up for the GPU increases with decreasing voxelsize. Refer to Fig. 8 (a).

The configurations are sorted from highest to lowest performance. Best performance is obtained if no photo-consistency testing (PCS) is done. However, this results in a less accurate reconstruction. The impact of the algorithmic techniques Early Ray Carving (ERC), Interleaved Depth Sampling (IDS) and vertex/fragment processor load balancing can be seen when these features are not used. Refer to Fig. 8 (b).

When performing reconstruction to virtual views, sampling along a particular ray is terminated if a photo-consistent voxel is found. So for all such rays, a higher depth does not have any impact on performance. When increasing depth, a convergence can be also observed for the success rate since most possible consistencies are already found at a lower depth. The chosen depth-threshold in each concrete case is depending on voxelsize and scene complexity. An interesting observation can be made when comparing the GPU approaches at different image resolutions. As might be assumed, a higher resolution results in a higher success rate which corresponds to reconstruction quality. Refer to Fig. 8 (c)-(d).

Comparing CPU and GPU versions achieves promising results. The GPU one suffers from texture upload performance, which does not hold for the the CPU. Since image processing is done on the GPU, we need to upload only the
raw 3x8 bit image data. For eight source cameras at QVGA resolution, the GPU shows a speed-up of $\sim 25x$, at XGA it is still $\sim 10x$. Up to our knowledge, no other group dealing with real-time reconstruction is using such a high resolution.

5. Discussion and Future Work

The use of virtual views has several advantages on performance but also limits the reconstruction quality. With an increasing number of source cameras, more details will be visible. Unfortunately, these can not be recovered if occluded by reconstructed surface in the virtual views. This issue leads to ambiguities when deciding the occupancy of occluded voxels.

Reconstruction quality suffers when using the heuristic to determine active source cameras as it may come to wrong guesses. Together with the quality of reconstructed voxels, also the number of holes increases. To apply explicit visibility information, a plane-sweep approach has to be used instead of ray-sampling. To make use of the vertex processor, a plane of voxels has to be rendered as a set of points. After reconstructing the plane, the occupancy information is projected into the source camera image planes to update visibility information before reconstructing the next plane. This is similar to the method described in [14] applying only fragment processing.

Offline-captured image data is used for reconstruction. For online reconstruction the images have to be transmitted over network using an efficient protocol. With eight cameras at a framerate of 30 Hz, we obtain 4.2/1.6/0.4 GBit/s for XGA/VGA/QVGA resolution. Image compression has to be applied, depending on network bandwidth and resolution. In this case, the effects of lossy image compression on the reconstruction result have to be examined. All processing is done on a single server with the PC cluster used only for image capturing. If we are able to perform this on the server as well, we obtain a more lightweight system architecture. Capturing can be done using FireWire800 or network cameras. For synchronization purpose, image capturing and data transmission need to be done separately. First, all cameras are triggered by sending a capture signal. Afterwards, the image data is retrieved sequentially. Combined with a fast or automatic camera calibration technique, the framework can be applied in a mobile environment.

6. Conclusion

With this work we have shown our efforts in developing a system for hardware-accelerated space carving. We explained how to achieve high quality through combining shape from silhouette and shape from photo-consistency techniques for determining the photo hull of a 3D scene. Applying programmable vertex and fragment processing on current off-the-shelf graphics hardware with algorithmic techniques for using structural coherence, all necessary processing is done on a single PC in real-time, enabling future application in a mobile environment. We introduced a set of conservative nested volumes BB, MVI, VHA and PCS that are incrementally computed to speed-up the reconstruction process. Further, we proposed a robust and fast image-segmentation to account for shadows and a heuristic for implicit visibility computation. Compared to implementations for the CPU, the new GPU-based approach leads to a speed-up of 10-25x depending on input image resolution. Nevertheless, we think that the results can be further improved and presented promising enhancements to be realized in the future.

An earlier version of this work has been presented in [24]. A diploma thesis has also been published on this topic. Hence, the reader may refer to [25] for a more detailed coverage of the topic.

Acknowledgment

This work is supported by Ministry of Education, Culture, Sports, Science and Technology under the “Development of fundamental software technologies for digital archives” project.

References


Christian Nitschke is currently working as a freelance software engineer and plans to start his Ph.D. program at Cybermedia Center, Osaka University in late 2007. He studied Media System Science at the Bauhaus-University of Weimar, Germany and received his diploma degree in 2006. Between 2005 and 2006 he has been working for Cybermedia Center, Osaka University as a special research student. His research interests include projection and display technologies, realtime rendering, graphics hardware, computer vision and 3D reconstruction.

Atsushi Nakazawa received the B.S. degree in 1997 and the Ph.D. degree in Engineering Science from Osaka University in 2001. Since 2001, he has been working for Institute of Industrial Science, University of Tokyo as a post-doctoral researcher. Since 2003, he has been working for Cybermedia Center, Osaka University as a lecturer. His research interests are Computer Vision and Robotics, in particular, estimating human posture from images, analysis human motion using motion capture and humanoid robot control. He is a member of IEEE, IPSJ, and RSJ.

Haruo Takemura received B.E., M.E., and Ph.D. degrees from Osaka University in 1982, 1984 and 1987 respectively. In 1987, he joined Advanced Telecommunication Research Institute, International. In 1994, he joined Nara Institute of Science and Technology, Nara as associate professor at Graduate school of Information Science and Technology. From 1998 to 1999, he was a visiting associate professor at University of Toronto, Ontario, Canada. In 2001, he became a full professor at Cybermedia Center, Osaka University, Osaka, Japan. Since August 2005, he also serves as a vice-director of Cybermedia Center. His research interest includes Interactive Computer Graphics, Human-Computer Interaction and Mixed Reality. He is a member of IPSJ, VRSJ, HIS, IEEE and ACM.