GPU-BASED VOLUMETRIC RECONSTRUCTION OF TREES FROM MULTIPLE IMAGES

D. M. M. Vock [a, b, *], S. Gumhold [a], M. Spehr [a], P. Westfeld [b], H. G. Maas [b]

[a] Computer Science Faculty, Dresden University of Technology, Noethnitzer Street, 01067 Dresden, Germany
[b] Institute of Photogrammetry and Remote Sensing, Dresden University of Technology, Helmholtz Street, 01067 Dresden, Germany

Commission V, WG V/4

KEY WORDS: 3D Reconstruction, Image-based Modelling, Tomographic Reconstruction, Visualisation, Hardware Acceleration

ABSTRACT:

This paper presents a new hardware-accelerated approach on volumetric reconstruction of trees from images, based on the methods introduced by Reche Martinez et. al [Rec04]. The shown system applies an adapted CT procedure that uses a set of intensity images with known interior and exterior camera parameters for creating a 3D model of a tree, while requiring considerably less images then standard CT. At the same time, the paper introduces a GPU-based solution for the system. As tomographic reconstructions are rather complex tasks, the generation of high-resolution volumes can result in very time-consuming processess. While the performance of CPUs grew in compliance with Moore's law, GPU architectures showed a significant performance improvement in floating-point calculations. Regarding well parallelizeable processes, today's end-user graphics-cards can easily outperform high-end CPUs. By improving and modifying the existing methods of volumetric reconstruction in a way, that allows a parallelized implementation on graphics-hardware, a considerable acceleration of the computation times is realized. The paper gives an overview over the single steps from the acquisition of the oriented images displaying the tree till the realization of the final system on graphics processing hardware.

1. INTRODUCTION

The three-dimensional reconstruction of real life objects for virtual representations gains more and more importance in different fields of scientific research, industry and multimedia entertainment. While current methods such as laser scanning or structured light projection are able to generate satisfying results for most artificial structures, they are only of limited use for the acquisition and replication of complex objects like plants. The multifaceted geometric shape of plants (especially treetops), which is characterized by a large number of occlusions and a huge amount of self similarities, complicates the creation of 3D tree-models. A possible solution to circumvent these complications has been introduced by Reche Martinez (Reche et. al 2004) realizing the tree-models as volumetric grids that contain an opacity value for every voxel. As this approach is based on the principles of x-ray computed tomography, the model is created from projections from object to image space instead of estimating discrete points of the tree surface. Thus geometric difficulties such as the mentioned self similarities have no effect on the actual reconstruction.
While Reche intends to generate low resolution volumes that use view dependent microfacette billboards (Yamazaki et. al 2002) for visually detailed models, this paper extends and alters the approach in a way that realizes high resolution grids with precise measurements. However, as tomographic procedures are rather time-consuming and as the amount of data calculated grows cubically in the resolution of the volume, the generation of higher resolutions can lead to very high computation times. To speed up reconstruction, the presented approach shows a solution for a GPU-based out-of-core implementation, enabling large grids to be computed on low budget graphics cards in less than 1/100th of the time as needed by a CPU implementation.

2. IMAGE ACQUISITION

The fundamental input to the system is a series of images with known orientation and camera calibration parameters. For a detailed reconstruction, the precision of the image orientation is essential. Small deviations lead to wrong projections between the image and the volume space so that details as thin twigs and branches can easily disappear from the model.
Among the examined systems, the AICON-3D-Studio (AICON 3D Systems 2009) has proven to be able to accomplish the required precision in combination with an on-the-job calibration of the used camera. For the actual image acquisition, a larger number of markers was placed within the tree's surrounding area, which were used to locate positions and rotations of the images by bundle adjustment. AICON-3D-Studio (AICON 3D Systems 2008) uses a set of encoded markers that can automatically be found inside the images and recognized by their cyclic code.

Figure 1: Detail view of markers (encoded and uncoded) placed around a tree

* Corresponding author. Contact: dmmvock@gmail.com

During favourable conditions (little or slight wind), the marker positions within the measurement environment could be pinpointed with an average precision below 0.5mm for a tree of 10m height, which allows also thin branches of the trees to be preserved within the volume.
For a complete reconstruction of a tree, about 20 exposures from different sides were sufficient for good visual results.

3. ALPHA MATTING

Each pixel of an image represents a projection from the actual surrounding onto the image plane. Thus, each pixel can be regarded as a projection of the tree displayed in the image or its background. Same as presented in (Reche et. al 2004), masks are required that segment the pictures into those two areas, the tree and it's background. Those masks are generally known by the term alpha-mattings. Usually, trees situated in natural and urban areas are surrounded by further objects with similar appearance and color, that can prevent the successful segmentation of the images. To reconstruct every plant regardless of it's environment, an alpha matting algorithm stable against such influences is required. Experiments with various algorithms (Chuang, et al. 2001) (Grady, et al. 2005) (Sun, Jia und Tang 2004) have shown, that the closed form matting as introduced by (Levin et. al 2008) generated the best results in most cases while requiring only little user-input.

Figure 2: Alpha matting of an image using the closed form matting algorithm by (Levin et. al 2008). Left: the original image; Middle: image with scribbles by user-input; Right: resulting alpha image, separating the image in fore- and background.

The algorithm is based on a local smoothness assumption on the fore- and background colors of the images. Regarding the segmentation of an image into two regions, each pixel can be interpreted as a combination of the local fore- and background colors:

$$I_i = \alpha_i F_i + (1-\alpha_i) B_i \quad (1)$$

where I = image
i = pixel i
α = alpha value
F = foreground colors, B= background colors

Given this assumption, F_i and B_i are constant within a window w enclosing a pixel i. As the assumption is only locally introduced, F and B may vary in different areas of the image. At the same time, sharp edges in the image are preserved, as only the color spaces are smoothed while the spatial affiliation stays untouched.
For the algorithm to know which part of the image belongs to the foreground, additional information about the images is required. By simple user-input (scribbles and strokes) areas that belong entirely to the fore- or background are defined. Based on this input, the algorithm estimates the alpha value of the yet undefined pixels.
Experiments have shown, that depending on the amount of interfering textures within an image, more or less user-input is required. How much the alpha-matting is influenced by the image content depends on the contrast and regional distance between fore- and background color spaces on the image plane. Thus, the segmentation of an picture, containing an isolated tree only needs a minimum of user-input. Referring to the amount of interferences and complexity of the trees structure, the time, necessary to prepare an image for the alpha matting varied between 1 and 5 minutes. In most instances, satisfying matting results were obtained. Only in cases where the tree-top area was covered by similar trees in the background, good mattings were difficult to achieve.

4. GENERAL PURPOSE COMPUTATION ON GRAPHICS PROCESSING UNITS

Over the last few years, the processing abilities of graphics cards developed from simple graphic processing units with fixed computation pipelines, to flexible, programmable multiprocessor units. (NVidia 2009a) These changes made it possible to use graphics cards for jobs apart from general rendering tasks and formed the term "general purpose computation on graphics processing units" (GPGPU).
Each GPU consists of several multiprocessor units able to execute arbitrary calculations, similar to CPUs. The multiprocessors consist of several scalar-value processors sharing a common instruction unit.
Nevertheless, GPUs underly certain restrictions that have to be taken into account for a successful implementation of the system on GPU hardware. While it is possible to assign arbitrary threads to the cores of a CPU, the scalar-processors within the GPU's multiprocessor are only able to execute multiple threads if those perform the same instructions at the same time. Otherwise, the threads have to run sequentially. Therefore, GPUs achieve best performance, when all threads follow the same order of instructions, allowing best parallelization of the tasks. Another limitation of GPUs is not being able to execute recursive functions and the lack of atomic instructions for writing to a certain memory location consecutively. Only certain GPUs offer atomic instructions in their instruction sets, yet until now, these are still limited to simple computations on integer values. Because of the architecture of graphic cards that runs parallel threads in so called warps (NVidia 2009b) on a multiprocessor, the usage of mutex structures is difficult and can easily lead to deadlocks. Yet, the established approach by (Reche et. al 2004) requires sequential write instructions to be feasible in this context.
Taken these limitations into account, the following chapters introduce a solution for the volumetric reconstruction on graphics processing units.

4.1 CUDA

For the implementation of the algorithms on GPUs, different APIs have been analyzed regarding their advantages for the given tasks. In this context, nVidia's Compute Unified Device

Architecture (CUDA) has proven to be a good solution. CUDA is a parallel programming model which is suitable for outsourcing complex algorithms for parallel processing of large data streams to the GPU. It presents a new way of multithreading, allowing the user to implement the algorithm for a single thread which is used as a template for a multidimensional field of threads generated and executed during runtime. E.g. to process an image with 320x240 pixel, a thread can be assigned to each pixel, resulting in a field of 768000 threads that can be executed concurrently. The main advantage of CUDA over most other GPU-APIs is flexible access to the GPUs memory, allowing direct access of the global memory as well as the multiprocessors shared memory for reading and writing instructions.
While most APIs only allow limited access to the global memory, marking memory locations as read only or write only, CUDA enables both, read and write access to the same memory block. For the reconstruction of the trees, this implicates that only one representation of the voxel grid is required in the GPUs global memory instead of one read only input and one write only output data grid.

5. RECONSTRUCTION

For the reconstruction of the tree, a volume is specified within the world space defined by the positions of the calibrated images and the measurement fields encoded and uncoded markers. The system introduced in this paper makes direct usage of the marker locations to determine a fitting volume position and size.
To simplify the implementation of the algorithm for GPUs, a regular grid containing the voxels is used to represent the volume. The actual algorithm for the reconstruction is divided into 2 steps:

5.1 Initialization

At first, an initial volume is created by applying a space carving algorithm which determines the photo consistency of the voxels and generates a first approximation of the 3D model. For this purpose, every voxel of the volume is projected into all image spaces to calculate the alpha value of the voxel from every given line of sight.

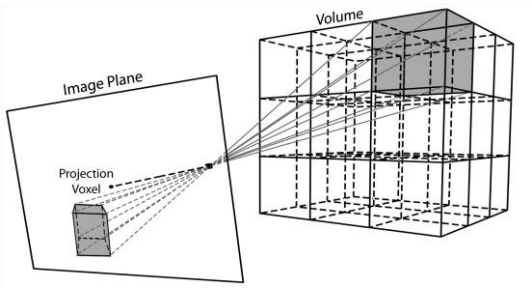

Figure 3: Projection of a voxel into the image plane of an exposure.

Comparing the obtained alpha-values and storing only the minimum value into the corresponding voxel results in a good representation of the outer hull and the general structure of the tree.
However, the model shows a significantly too high number of non-transparent voxels, especially within the area of the tree-top. This is due to the fact, that during the space carving, only those voxels opacity is set to transparent, where the actual projections are transparent in at least one matting. In contrast, voxels that should be transparent in an optimal representation, but are covered by opaque elements in all mattings, are also erroneously initialized to an opaque value. As consequence, the model will appear overfilled from different angles.

Conventional CT-methods are able to deal with this problem, as the density information from x-ray exposures allows a direct reconstruction of the parts of the object that are hidden in the alpha images. (Peters 2002)
However, the alpha mattings only represent a silhouette of the trees and thus, the use of a CT-approach to reduce the number of non-transparent hidden voxels in the presented model is not feasible.

Nevertheless, a visually convincing result can be achieved from the alpha mattings by adjusting the voxel opacities in a way, that a raycasted rendering through the volume results in an intensity image which resembles the according alpha-matting with the same orientation.

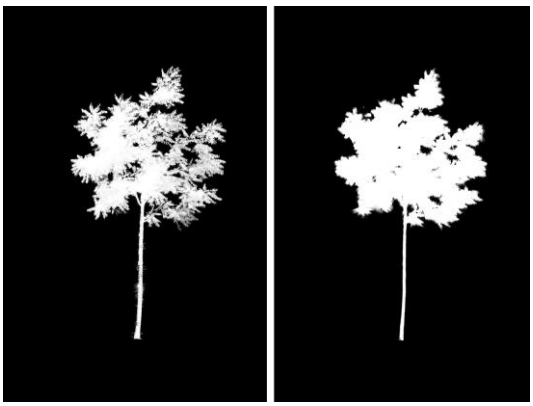

Figure 4: Comparison between an alpha-matting of a tree (left) and the according visual result of a raycasted rendering through the initialized volume, given the same orientation (right).

5.2 Iterative Solution

One possible solution to improve the visual appearance of the volume is to improve the volume transparencies by minimizing the differences between the renderings of the volume and the input alpha-mattings.
This aim can be achieved by calculating the weighted mean difference between the actual voxel values and their expected correct values. The mean differences can then be used to improve the current voxel transparencies. By iterating this procedure, the visual appearance of the volume converges, resulting in a minimum deviation from the alpha-mattings.

Regarding the implementation on GPUs, the approach by (Reche et. al 2004) is suitable only to a limited extend, as it requires large amounts of memory and atomic instructions on floating point data. The approach, presented in this paper provides a new solution, separating the task into 2 sequential tasks, well suited for today's GPU hardware.

At first, an adjustment image is calculated for every input alpha-matting. These images are generated from the deviation between the input images and the projections of the current volumes with the same orientation and camera calibration parameters. They contain correcting values that can be used to improve the voxel transparencies for a better visual appearance.

The current transparency values of the projected images are obtained by applying a ray-casting algorithm that gathers the voxel values along a ray through every pixel. All parameters, required for the ray-casting, are taken from the interior and exterior orientation of the alpha-mattings to render the new images from the same point of view. The raycasting is implemented, assuming a simple absorption model for the volume, based on the volume rendering equation as shown in (Reche et. al 2004):

$$\log(\tau_p) = \sum_{i=1}^{n} \log(\tau_i) \quad (2)$$

where τ_p = transparency value of image at pixel p

τ_i = transparency value of the volume at voxel i

1..n = voxel that were sampled by the ray

This way, the raycasting can be simplified to a mere addition of the values sampled along the ray from the entry- to the exit-point of the volume (Figure 5).

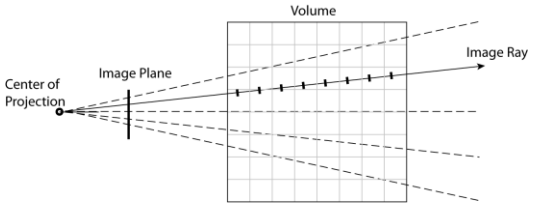

Figure 5: 2D example for a raycasting through the volume, gathering the voxel values along the ray.

The rendered images represent the actual appearance of the current volume. By interpreting the alpha-mattings as the targeted appearances, the adjustment images can be calculated directly:

$$corr_p = \log(\alpha_p) / \log(\tau_p) \quad (3)$$

where $corr_p$ = *adjustment image pixel p*

$\log(\alpha_p)$ = alpha matting image *pixel p*

$\log(\tau_p)$ = raycasted projection image *pixel p*

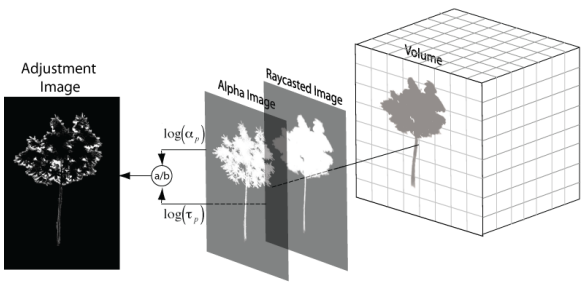

Figure 6: Creation of an adjustment image from the given alpha-matting and the via raycasting rendered volume.

The resulting images contain a correcting value for every pixel. This procedure is repeated for all alpha-images used in the reconstruction to generate the required set of adjustment images.

By projecting a voxel onto the image planes once again, a corrected transparency value can be calculated for every pixel in the projections:

$$\log(\tau_i') = \log(\tau_i) \cdot corr_p \quad (4)$$

where τ_i' = corrected transparency value τ_i

For the estimation of the weighted mean deviation of the voxel values, an additional value pair σ and ω is specified. σ defines the total deviation of the actual voxel value from the targeted value, weighted by the influence of the respective image ray on the voxel and ω defines the total weight of all image-rays that intersect the voxel.

The total deviation of a voxel i is calculated as followed:

$$\sigma_i = \sum_p \left((\tau_{i,p}' - \tau_p) \cdot \omega_{i,p} \right) \quad (5)$$

where $\tau_{i,p}$ = corrected transparency value of voxel I for the image ray through pixel p

$\omega_{i,p}$ = weight factor of the influence of the image ray through pixel p on the voxel i

The total weight:

$$\omega_i = \sum_p (\omega_{i,p}) \quad (6)$$

An essential factor for the weight of a deviation is the length the respective image ray intersects the voxel. This can be determined by a simple bounding box test, that returns the entry- and exit point of the ray through the voxel.

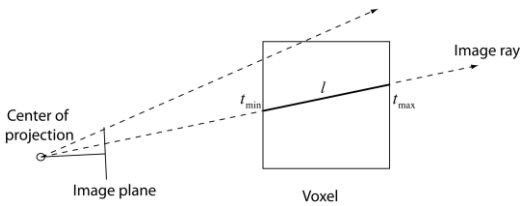

Figure 7: 2D example for an intersection of an image ray and a voxel. The ray intersects the image for the length of l from t_min to t_max

Finally, the new correcting value for the voxel transparencies can be estimated:

$$\tau_i += \sigma_i / \omega_i \quad (7)$$

If the resulting transparency value of an element is above a chosen level, the voxel is assumed to be completely transparent and excluded from the calculations of the following iterations. The iterative process is terminated when all correcting values of an iteration are below a certain threshold.

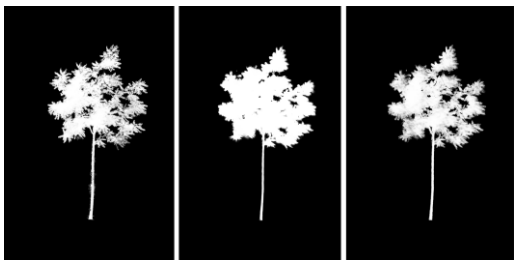

Figure 8: Left: Alpha-matting original; Center: Raycasted image of the initialized volume; Right: Raycasted image of the converged volume after 4 iterations.

6. IMPLEMENTATION ON GPU

As mentioned earlier, for realization using CUDA, the algorithm has to be implemented as a template for a single thread that will be assigned to a multitude of threads during runtime.

6.1 Initialization

For the initialization of the volume, it's suitable to assign every voxel to a thread of its own and thus, define a three dimensional grid of threads that is distributed amongst the multiprocessors of the GPU. As the number of threads in a 3D-grid is currently limited to a maximum of 64 in direction of the z-axis, a two dimensional array of threads is defined instead with one thread for every x,y index of the volume that iterates over all voxels along the respective z-axis (Figure 9).

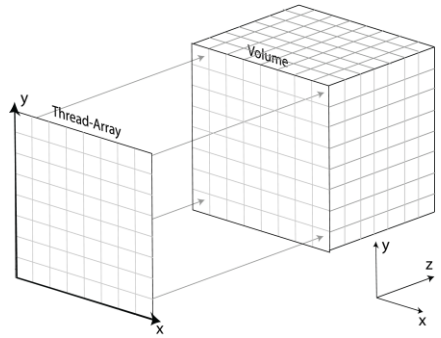

Figure 9: CUDA thread-array for the initialization of the voxel values.

The data-grid that represents the content of the volume is copied directly as a whole grid onto the graphics-card, while the alpha-mattings are loaded sequentially into and removed from the GPU-memory after being processed. In this way, the amount of memory required on the GPU is dramatically reduced, allowing for larger grids to be computed at once. When an image is loaded onto the GPU, the voxels are projected into the image plane and the maximum transparency value is computed for every voxel.
After the initialization is complete, the resulting volume data is copied back into the computers main storage.

6.2 Iterative Solution

For the refinement of the voxel values, the volume is copied onto the GPU as a CUDA 3D array. Together with the actual volume data, additional arrays are allocated for σ and ω as well as for an adjustment image.
Realizing the raycasting procedure on the GPU is easily done by assigning an own thread to every pixel of the output image. The resulting image is written back to the main storage and the adjustment image is created as presented in 5.2.
Finally, the total deviation and weight can be computed on the GPU using the same strategy as the one presented for the initialization.
The new transparency values of the voxels are calculated on the host computer and written back onto the GPU for the following iteration.

6.3 Out of core solution for large volumes

Another limitation to the computation of large volumes on GPUs is the amount of video memory available on today's end-user graphics-cards, which is small compared to the average computer's main-storage. The advantage of the introduced algorithm is that it is easily segmentable, allowing the volume to be split up into several smaller sub-volumes that can be calculated independently.
For the volume initialization, the segmentation is done as the voxels are independently projected onto the image planes of the alpha mattings. The volume is directly split up in smaller sub-volumes that fit onto the GPUs global memory. Every sub-volume is then independently initialized and can be recombined to a complete volume afterwards.
In contrast, the voxel values are largely dependent from each other during the iterative correction. This is based on the fact that the adjustment images are created from raycastings through the complete volume and contain the required information for the improvement of voxel values.
As taken from formula (2), all pixels of the raycasted images are calculated from the sum of all sampled $\log(\tau_i)$ values of each ray. Consequently, the rays can also be split up into several sections, containing partial results that can be added to the complete solution:

$$\begin{aligned}\log(\tau_p) &= \sum_{i=1}^{n\_n}\log(\tau_i) = \sum_{i=1}^{n\_1}\log(\tau_i) + \sum_{i=n\_1}^{n\_2}\log(\tau_i) \\ &+ \ldots + \sum_{i=n\_n-1}^{n\_n}\log(\tau_i) \\ &= \log(\tau_{p\_1}) + \log(\tau_{p\_2}) + \ldots + \log(\tau_{p\_n})\end{aligned} \tag{8}$$

Transferred to the volumetric reconstruction, the raycasting of a sub-volume corresponds to the calculation of such a section. By adding the partial renderings the required image can be calculated. The required adjustment image is obtained by:

$$diff_p = \frac{\log(\alpha_p)}{\log(\tau_{p\_1}) + \log(\tau_{p\_2}) + \ldots + \log(\tau_{p\_n})} \tag{9}$$

and can be used for the iterative solution as presented in 6.2.

7. RESULTS

The GPU-based implementation was realized for a significant boost of the computing time it takes to reconstruct a volume at high resolutions. To examine the actual performance growth, the volume initialization has been compared between GPU as well as CPU. As for the implementation on the CPU, a single-threaded solution was realized. Thus only one core was active during the reconstruction. All o the following tests were carried out by the same machine with the following specs:

CPU	Intel Core 2 Duo 2.4 Ghz, 3MB L2	RAM	4GB DDR3 1067Mhz
GPU	NVidia GeForce 9600M GT	VRAM	256MB

Table 1. System configuration of the test environment

For the evaluation, the ailanthus (Figure 2) was reconstructed at different resolutions of the volume as well as of the input matting images.

The resulting data (Figure 10 and Figure 11) shows a remarkable performance gain for the GPU implementation, of up to 75 times the speed of the CPU.

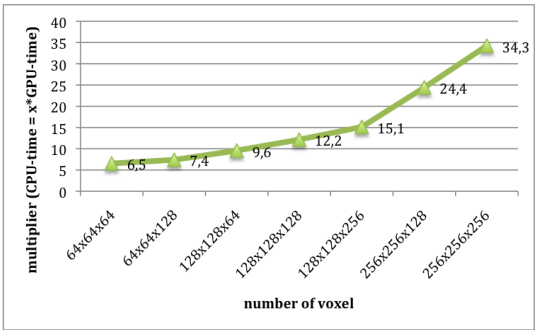

Figure 10: Evaluation of the performance growth of the GPU-based implementation. Used Dataset: Ailanthus, image resolution: 2376x1584.

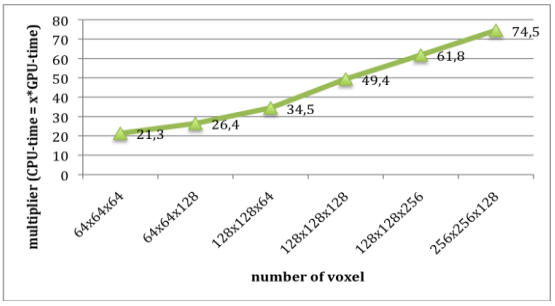

Figure 11: Evaluation of the performance growth of the GPU-based implementation. Used Dataset: Ailanthus, image resolution: 1188x792.

8. CONCLUDING REMARKS

In this paper, a new approach for the volumetric reconstruction from natural images has been realized that allows the generation of 3D tree models of within a short amount of time. The presented algorithms and techniques make it possible to sub-divide the process into many small independent tasks that can be easily parallelized for outsourcing the computations onto GPUs. At the same time, the maximum volume size is not limited by the amount of available VRAM as the introduced out-of-core solution splits the whole volume into swapable sub-volumes.

Together with the precise calibration and orientation of the images by the AICON 3D Studio and the detailed alpha-matting generated by using the closed form matting algorithm (Levin et. al 2008), high resolution volumes of detailed tree models are possible and can be generated in a minimum amount of time.

Figure 12: Sample rendering of reconstructed trees. Left: Ailanthus; Right: Rowan tree

9. REFERENCES

AICON 3D Systems. 2009. http://www.aicon.de/content/index.php?option=com_content&task=view&id=22&Itemid=37 (accessed 10 July 2009).

Chuang, Yung-Yu, Brian Curless, David H. Salesin, and Richard Szeliski, 2001. „A Bayesian Approach to Digital Matting.“ Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition.

Grady, Leo, Thomas Schiwietz, Shmuel Aharon, and Rüdiger Westerman, 2005. „Random walks for interactive alpha-matting.“ Proceedings of Visualization, Imaging and Image Processing 05. 423-429.

Halfhill, Tom R., 2008. „Parallel Processing With CUDA.“ *Microprocessor Report, January 2008*.

Kak, Avinash C., and Malcolm Slaney, 1988. *Principles of Computerized Tomographic Imaging.* IEEE Press, .

Levin, Anat, Dani Lischinski, and Yair Weiss, 2008. „A Closed Form Solution to Natural Image Matting.“ *IEEE Transaction on Pattern Analysis and Machine intelligence* 30, Nr. 2 : 228-242.

Luhmann, Thomas, 2003. *Nahbereichsphotogrammetrie - Grundlagen, Methoden, Anwendungen.* Wichmann, ISBN: 978-3879073986.

NVidia, 2009a. „CUDA Technical Training.“ *Volume I: Introduction to CUDA Programming.* NVidia. http://www.nvidia.com/docs/IO/47904/VolumeI.pdf. (accessed 12 July 2009)

NVidia, 2009b. „NVidia CUDA_Programming_GUIDE.“ http://developer.download.nvidia.com/compute/cuda/2_2/toolkit/docs/NVIDIA_CUDA_Programming_Guide_2.2.pdf (accessed 01 August 2009).

Patidar, Suryakant, and Shiben Bhattacharjee, 2007. „Exploiting the Shader Model 4.0 Architecture.“ Center for Visual Information Technology, International Institute of Information Technology Hyderabad, Hyderabad.

Peters, Terry, 2002. „CT Image Reconstruction.“ *American Association of Physicists in Medicine - 44th Annual Meeting.*

Reche, Alex, Ignacio Martin, and George Drettakis, 2004. „Volumetric reconstruction and interactive rendering of trees from photographs.“ ACM, 2004. 720-727.

Sun, Jian, Jiaya Jia, and Chi-Keung Tang, 2004. „Poisson Matting.“ Proceedings of ACM SIGGRAPH 2004, ACM Transactions on Graphics (TOG). ACM NY. 315-321.

Yamazaki, Shuntaro, Ryusuke Sagawa, Hiroshi Kawasaki, Katsushi Ikeuchi, and Masao Sakauchi, 2002. „Microfacet billboarding.“ ACM International Conference Proceeding Series, 2002. 169-180.