EG UK Theory and Practice of Computer Graphics (2009)
Wen Tang, John Collomosse (Editors)

Accelerating Raycasting Utilizing Volume Segmentation of Industrial CT Data

S. Frey[1] and T. Ertl[1]

[1]Visualisierungsinstitut der Universität Stuttgart, Germany

Abstract

We propose a flexible acceleration technique for raycasting targeted at industrial CT data and the context of material deficiency checking. Utilizing volume segmentation that is typically employed for object analysis, GPU raycasting can be accelerated significantly using a novel data structure that is integrated into the volume to improve the responsiveness for the interactive, visual inspection of high-resolution, high-precision data.
Our acceleration approach is designed to cause no extra texture lookups and to produce only marginal computational and storage overhead. Despite the fact that the data structure is integrated into the volume, the graphics card's hardware can still be used for trilinear interpolation of density values without producing incorrect results. The presented method can further easily be utilized in combination with out-of-core approaches and distributed volume rendering schemes.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Viewing Algorithms – I.3.8 [Computer Graphics]: Applications – I.4.6 [Image Processing and Computer Vision]: Region Growing

1. Introduction

Computed Tomography (CT) is widely used not only for medical purposes, but also in the field of industrial material testing and quality assurance. In this area, the analysis of component parts is usually accomplished by a combination of interactive visual inspection (e.g. with a raycaster) and the (semi-)automatic extraction and comparison of characteristics and geometry. These advanced processing steps are usually executed on the basis of a segmentation step that partitions the examined object. The focus of this paper is to use this segmentation information additionally for raycasting acceleration. This is important because even when using modern high-end GPUs for visual analysis, raycasting of big volumes that barely fit into graphics memory and beyond still only achieves low frame rates. In this context, the bottleneck are the texture accesses in the raycasting loop that are needed to sample the volume, apply a transfer function, perform lighting and so forth. For this reason, a typical approach to accelerate raycasting is to reduce their amount by decreasing the number of raycasting loop iterations per ray. This is frequently accomplished by only sparsely sampling regions that have no contribution to the pixel color or that do not contain interesting information (e.g. homogeneous regions). In order to provide a basis for these algorithms, volume data needs to be segmented in such a way, that no critical information is contained in an area that is only sparsely sampled. For example, when checking an engine block, one is typically not interested in the regions homogeneously filled with metal, but in blowholes or other material deficiencies, which should not be in the same segmented region as metal. Like in the example of the engine block, industrial volume data often consists of only a few materials. Small but significant changes like blowholes or bubbles in a structure need to be identified, while large homogeneous areas are negligible. Therefore the proposed volume acceleration technique guarantees that deficiencies are not skipped. Furthermore, no additional texture fetches are needed in the raycasting loop and only minor computational overhead is induced compared to a standard raycaster in order to achieve high rendering performance even in areas that have to be sampled densely. Economic utilization of graphics memory is important as well because modern industrial CT scanners allow for high-resolution volumes with 16 bit precision which results in large data sizes. On this account it is also important that the acceleration algorithm can be combined with out-of-core approaches and distributed volume rendering schemes.

© The Eurographics Association 2009.

delivered by EUROGRAPHICS DIGITAL LIBRARY www.eg.org diglib.eg.org

DOI: 10.2312/LocalChapterEvents/TPCG/TPCG09/033-040

2. Related Work

The GPU raycasting [SSKE05] acceleration method introduced in this paper belongs to the class of techniques that skip samples in regions that are considered to not contain critical information. Much work has already been done in this area in order to be able to handle the high computational complexity of volume rendering. However, implementations of previously published techniques on the GPU have certain shortcomings which the approach presented in this paper targets to resolve.

One of the first acceleration techniques developed is empty space leaping, which skips empty or transparent regions of the volume and uses a precomputed data structure to determine these regions. Realizing this concept, Cohen and Shefer [CS94] and Zuiderveld et al. [ZKV92] introduced proximity clouds. Proximity clouds employ a distance transform of the object to accelerate the rays in regions far from object boundaries. Distance transform values are stored in place of density values in this empty regions, and therefore storage is not increased. Freund [FS97] additionally utilized the fact that medical imaging scanners only produce twelve bit intensity values for each voxel. So four bits are available to store leap values besides the density values in regions containing critical data while the whole 16 bit are used to store leap values in empty regions. This allows for leaping not only in empty space but also in occupied regions. However, this is not applicable in the context of this paper as modern industrial CT scanners produce 16 bit data. Yet the main problem with proximity clouds and similar approaches that were primarily developed for CPUs is that they have no means to correctly deal with trilinear interpolation that is usally employed in graphics hardware when sampling with a GPU raycaster. The interpolation between leap values and standard density information is undefined and leads to deceptive results. This can be overcome by doing an extra lookup to check for leap values, but it is not eligible because it causes a texture fetch overhead.

Besides techniques that directly integrate skip values into the volume, numerous approaches employing hierarchical data structures like octrees were presented [Lev90]. In octrees, non-leaf nodes usually store an entropy metric of its children to be able to traverse occupied space faster when the entropy is low. Employed metrics include standard deviation [DH92] and the minimum-maximum range of values [WV90]. Guthe and Straßer [GS01] store in each non-leaf node a measure of the error that is committed when its children are not rendered. However, the big flexibility of these approaches induces a lot of additional texture fetches for tree traversal on the GPU which can significantly impair the performance benefit gained from sampling sparsely.

Instead of using a dedicated data structure, [KSSE05] utilize the spatial coherence of consecutively rendered images to determine the initial sampling point of a ray on the GPU to skip empty regions.

In this paper, the proposed acceleration technique is discussed in the context of a standard workflow of material deficiency checking with industrial CT data. Numerous visualization and information extraction techniques targeted at industrial CT data were presented [Hei09] [HMMW03], including feature extraction and visualization techniques utilizing homogeneous industrial workpiece segmentation amongst others. Bullitt and Aylward [BA02] also present a raycaster that is able to embrace segmented volume data for enhanced visual display in a medical context. However, none of these works use the regions gained from segmentation for acceleration purposes.

3. Basics

In this section, an introduction is given on the basics of the proposed acceleration approach and its modular implementation considering the context of industrial material deficency checking and advanced processing (Figure 1).

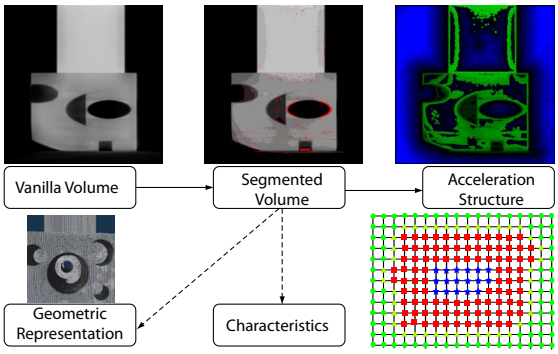

Figure 1: *Exemplary scheme of volume analysis and advanced processing for industrial CT data.*

3.1. Acceleration Approach

In order to allow for a more responsive interactive visual inspection by the user, the amount of iterations of the raycasting loop is reduced by leaping through volume regions that do not contain information of interest. These so-called homogeneous regions are coherent and contain voxels with density values in a certain scope called segmentation range. They are identified in a fully automatical segmentation step that can be configured by the user according to his interests. After that, for each region leap values are determined, which denote how many samples can safely be skipped by a ray without leaving the region so that no tiny critical areas like blowholes can be missed.

In order to avoid additional lookups for leap values, a sample value requested from the volume texture can either be a density or a leap value. This consequently requires the acceleration data structure to be integrated into the volume. To

© The Eurographics Association 2009.

assure that no leap voxels can be mistaken for density voxels and vice versa, the most significant bit of each voxel value is reserved for classification.

The integration of the data structure into the volume leads to another issue: trilinear interpolation must exclusively be used between density voxels as the result is undefined when leap voxels are involved. Thus interpolation needs to be switched off when necessary and a nearest neighbor access scheme must be used instead. This is triggered using a layer of voxels called guards that enclose the leap voxels of their region. When passing through these guards from the outside, the interpolation mode is switched to nearest neighbor. Trilinear interpolation is switched back on again when first encountering a voxel that does not belong to the current region. A region voxel is either a leap voxel containing a skip value or a guard voxel containing the regions average density (Figure 2).

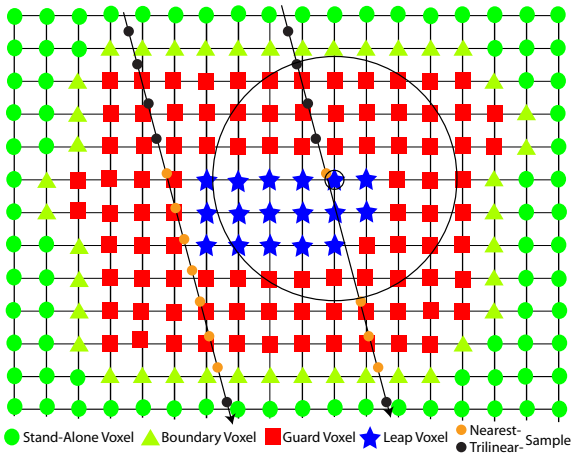

Figure 2: *Two rays passing a small region. The right one leaps and the associate leap value is displayed by the big circle around the respective leap voxel. Note that boundary voxels do not belong to a region and are not modified in value or used in the actual raycasting, but only their position is utilized for efficient leap value calculation.*

3.2. Modules

The components needed for the proposed acceleration approach are split into modules for enhanced flexibility. The first step for creating the acceleration data structure is the *Region Segmentation* that seperates the volume into regions. After that, the *Classification of Region Voxels* is executed to determine whether a voxel is a guard voxel or a leap voxel. Finally, the *Determination of Leap Values* computes for each leap voxel how many samples can safely be skipped. The *Volume Rendering Core* extends a standard raycaster with the functionality required to utilize the integrated data structure for leaping in regions.

4. Building the Data Structure

In this section, the modules are discussed that are employed for the construction of the data structure: region segmentation, classification of region voxels and the determination of leap values.

4.1. Region Segmentation

The basis for the proposed acceleration approach is the segmentation of the volume in homogeneous regions. A typical criteria for segmentation is the maximal range of values present in a region, but often also more specialized constraints are applied when prior knowledge about the data is available. This way, the user can focus on areas he considers as critical.

In this work, we use a fully automatic variant of the 3D flood fill algorithm, but any region segmentation procedure could be used as well like semi-automatic histogram-based methods. Every voxel in the volume that does not already belong to a region is considered as a potential seed point and attempts to grow. Which voxels are added to the region is controlled by user-defined region criteria. Voxels that were considered for a region but did not meet these criteria are saved in a boundary list that finally forms a one voxel thin layer around the region. This list is subsequently used for leap value determination.

4.2. Voxel Classification

Density values are sampled with trilinear interpolation, but there is no interpolation allowed between leaping and density values as this leads to undefined results. It is thus critical for the proposed acceleration technique that no leap voxel can be reached without switching to nearest neighbor interpolation. In our technique, this is effected when the ray collects exactly the same density value for two consecutive samples. This provides an indication that guard voxels have been passed and leap values might be sampled in the next step. Guard voxels contain the average value of the region and are situated in a layer around the leap voxels at the border inside of the region (Figure 2). The thickness of the layer depends on the maximum step size that the user wants to sample the volume with. Before the actual voxel classification, a list of neighborhood voxels is compiled based on this standard sampling distance. The list includes the relative coordinates of surrounding voxels that have to be checked for classification. If at least one of them does not belong to the region, the considered voxel is a guard voxel, otherwise it is a leap voxel (Listing 1).

The time complexity for the classification of all voxels in a region is $O(r \cdot s^3)$ with r denoting the amount of region voxels and s being the maximum user-defined standard step size, that is typically in the scale of the distance between two side by side voxels.

© The Eurographics Association 2009.

```
for r in regionVoxels
  // voxel classification
  leap = true
  for n in neighborList()
    if r+n not in regionVoxels
      leap = false
  // leap value determination
  if leap
    d = nearestD(boundaryVoxelTree, r)
    vol[r+n] = d | 0x8000
  else //guard
    vol[r] = regionVal & 0x7fff
```

Listing 1: *Integrated voxel classification and leap value determination.*

4.3. Determination of Leap Values

The values of the voxels classified as leap voxels are determined in this module. This could be done analogously to the voxel classification with a neighbor list by walking from the considered voxel spherically to the outside until a voxel is found that does not belong to the region and the distance to this voxel would be the leap value. While this is a good method for the classification as only a small and fixed distance has to be covered, this is very inefficient especially for large leap values, as potentially all region voxels have to be visited.

We employ a much more efficient technique that utilizes the so-called boundary voxels, a one voxel thin layer around the volume which is generated in the segmentation step. By organizing the boundary voxels in a kd-tree, the leap value of a considered voxel can efficiently be determined by a nearest point request (Listing 1). The complexity of this technique is $O(r \cdot log_2(b))$ with b denoting the amount of boundary voxels, which again is in $O(r^{\frac{2}{3}})$. This is derived from the approximation of the region as cuboid or ellipsoid, whose volume is in $O(e^3)$ (with $e^3 \widehat{=} r$) and whose surface is in $O(e^2)$ (with $e^2 \widehat{=} b$) regarding the one-dimensional extent e. We measured that most of the total preprocessing time is spent on kd-tree nearest neighbor requests using this approach. In order to overcome this bottleneck, we developed a modification that does not calculate optimal leap values, although they are usually almost as good, but that cuts down the preprocessing time significantly. Note that the actual leap values must only be underestimated, as an overestimation would allow a ray to leap out of volume and thus potentially skip critical data.

The main idea is that a leap value is not only computed for a single voxel but that it can be reused for a whole group of voxels around a so-called center voxel. Leap values for group voxels are applied by subtracting the leap value determined for the center voxel with the distance of the considered group voxel to the center voxel. All visited voxels are thus implicitly classified as leap voxels and not explicitly classified or considered as center voxels afterwards. However, they can be part of another voxel group in which case the maximum of the already applied and the newly calculated leaping value is taken. It needs to be guaranteed though, that no potential guard voxels belong to such a group, as a voxel could be falsely classified as a leap voxel this way, and this would not be corrected in the further course of the algorithm. This is why a voxel group can only be spread up to a certain distance to the boundary (Listing 2). However, voxels that are classified as leap voxels only become center voxels with a certain, user-defined probability and are otherwise assigned a minimum leap value. Another option for fine-tuning is the sphere radius that is defined relative to the center voxel leap value in which voxels are considered as group voxels. In general, the bigger the center voxel probability or the smaller the relative radius is the better are the leap values but the more time is consumed by the classification and especially the leap value determination.

```
for r in regionVoxels
  // check whether voxel has already been classified
  if todo[r]
  ...
    if leap
      // voxel becomes center voxel with certain probability
      if random(probability)
        d = nearestD(kdtree, r)
        // determine the absolute radius of a voxel group
        maxRadius = min(relLeap*d, d − guardDist)
        // apply leap values to group voxels
        for n in longNeighborList
          if len(n) > maxRadius
            break
          todo[r+n] = false
          vol[r+n] = max(vol[r+n], (d−len(n)) | 0x8000)
      else
        vol[r] = guardDist | 0x8000
...
```

Listing 2: *Leap voxel group acceleration extension for leap value determination.*

Overall, this modified version reduces leap value determination time significantly because the amount of nearest neighbor requests is reduced from $O(r)$ to $O(\sqrt[3]{r})$ for a group voxel radius in order of the region size. Taking into account that every voxel is at most looked at a constant number of times (depending on the maximum group voxel radius) and is thus in $O(r)$, this leads to the total complexity of the technique $O(\sqrt[3]{r} \cdot log_2 b + r)$.

Note that only isotropic leap values are considered with this module, but directional leap values could also be gen-

© The Eurographics Association 2009.

```
while t < tmax
  // Take a sample
  pos = eyePos + t*rayDir
  sample = tex3D(volTex, condNearest(pos, homReg))
  // Calculate the leap value
  leap = ((sample & 0x8000) == 0x8000)
  leapValue = leap ? decLeapValue(sample, ...) : stdStep
  // Get color for the sample and composit
  col = tex1D(transTex, leap ? oldSample : sample)
  col.w = 1 − (1 − col.w)^min(leapValue,tmax−t)
  sum.xyz += sum.w*col.w*col.xyz
  sum.w *= (1−col.w)
  // Change ray mode and save sample
  homReg = (sample == oldSample) || leap
  oldSample = leap ? oldSample : sample
  t += leapValue
```

Listing 3: *Pseudocode of the raycasting loop of the accelerated GPU renderer.*

erated and encoded with minor modifications leading to a more optimal leaping behavior of the ray at the cost of more time-consuming preprocessing.

5. Leaping using the modified volume

The integrated acceleration structure computed in the preprocessing steps can be used by the raycaster presented in this section in order to achieve significant speedups.

5.1. The Leaping Kernel

To be able to use the integrated data structure, a standard raycaster needs to be extended (Listing 3). It has to check whether a sample has potentially been taken from a region because it then has to switch the trilinear interpolation to a nearest neighbor address scheme. A sampling position is exactly then considered to be in a region if the ray either just leaped or consecutively sampled two identical density values (guard voxels). The ray is considered to have left a region when a sampled density value is not equal to the guard voxel's value that was retrieved when entering the region. When a sample is taken in a region, the sample value has to be classified as leap or density value using the most significant bit. In case of a leap value, the ray parameter must be added to the leaping value and correctly calculate the resulting color addition of the leaping step. Here, the opacity has to be corrected depending on the length of a leap [EHK*04].

Note that the leap value determination did not take volume boundaries into account as no boundary voxels were created even though naturally the region ends there. This allows for big leap values at the boundary of the volume, but it also means that leaps out of a volume are possible. This has to be accounted for to handle occupied regions at boundaries correctly. In order to assure that the calculated color is correct, the maximum leap value used for color calculation is trimmed to the exit parameter of the ray.

5.2. Supporting Slicing, Out-of-core and Distributed Raycasting

With no modification to the data structure and only minor modifications to the raycasting kernel, the proposed acceleration algorithm can be combined with slicing, out-of-core approaches and distributed volume rendering schemes [WGS04] to allow for the visualization of huge volumes exceeding the graphcis cards memory. We discuss the necessary modifications at the example of distributed raycasting approaches with object space partitioning [MSE07], that render volume blocks independently and finally blend the resulting images in a compositing step.

In contrast to the standard algorithm, the ray has to start as if it was inside a region with the nearest neighbor access scheme, as it is possible that the first sample taken is already in the context of leaping values. To check that, two samples with a standard step size are taken. If both sample values indicate an entering through guard voxels or if one of these samples contains leap values, the algorithm simply continues as usual. Otherwise, the ray starts over again like normally with trilinear inperolation switched on.

In the special case that a volume brick is completely inside a region, and thus the ray never hits a guard voxel and the density and thus the color information never get available, it sets a flag and writes out its length inside the brick respective to the standard step size instead of the color. The color for that brick pixel is then determined after the raycasting and before the compositing step. The color value of each sample along the ray is retrieved by a transfer function lookup with the average density of the concerning region and applying the amount of steps taken like in the raycasting kernel for leap values. What region the ray went through can efficiently be identified by taking any voxel that was visited by the ray and doing a lookup in a map that matches voxels with regions.

6. Volume Analysis for Deficiency Checking

The proposed raycasting acceleration approach fits conveniently into the standard workflow of volume analysis. Volume segmentation is used for both raycasting acceleration as well as for advanced processing and further helps the user with the visual exploration of the data set. He can select interactively whether he wants to see only (non-)region voxels or a mixed view. In addition, different transfer functions can be applied for voxels in and outside of regions. This enables the user for example to concentrate on inhomogeneous areas which are especially interesting when checking for defects.

© The Eurographics Association 2009.

Volume segmentation can further be used for the automatic extraction of characteristics without much additional effort. This way, characteristics like the volume, the surface area, the compactness, etc. of a region can be calculated. Additionally, the amount of holes in a region can easily be determined by flood filling its boundary voxels and counting the amount of not connected subsets. Using the segmentation, also a geometric representation can be extracted efficiently using a modified marching cubes algorihm. Instead of the whole volume, only the boundary voxel layer has to be visited to extract a geometric isosurface that is conformant with the volume segmentation.

The described approaches can efficiently be employed for random checking a series of components for certain types of defects. Region criteria are chosen once for a reference component and reused throughout the tests for automatical segmentation and subsequent characteristics extraction and comparison. A geometric representation can further be constructed based on the segmentation to support visual inspection and characteristics comparison as well as to provide input for advanced processing like a FEM simulation.

7. Results

We tested our approach on a machine equipped with an Intel Core2 Quad CPU 2.4 GHz, 4 GB of RAM and a NVIDIA GeForce GTX 280. We used a data set provided by our industry partner as well as publicly available data [Roe]. The publicly available data sets are the toy car ($559 \times 1023 \times 347$), the engine ($256^3$) and the ellipses ($256^3$). They are relatively noise free, materials have homogeneous values and transitions can clearly be distinguished. For providing a more challenging scenario, we further used the Zeiss512 (512^3) and the Zeiss768 (768^3) data set from our industry partner. They were reconstructed from the same noisy x-ray images without any smoothing, filtering or post-processing applied.

7.1. Preprocessing

We tested the performance of generating the acceleration structure at the example of the Zeiss512 data set with varying segmentation ranges, group voxel radii and probabilities in a series of 125 measurements. The classification of the voxels and the determination of the leap values utilize four CPU cores using OpenMP while the region segmentation employs a single core only.

Without the center voxel modification, guard voxel classification takes five minutes and leap value determination one hour. When employing the modification, the time it takes for both tasks combined is reduced significantly to approximately 2.5 minutes and slightly varies with different group voxel radius and probability settings as well as the segmentation range. Contrary to the voxel classification and leap value determination, our exemplary region segmentation algorithm heavily depends on the segmentation range and takes from 1.5m for big segmentation range up to one hour for a tiny segmentation range.

7.2. Raycasting Acceleration

The leaping as well as the standard raycasting kernel were implemented using CUDA. The performance was measured for different region segmentation ranges with fixed center voxel probability 0.25 as well as fixed group voxel radius 0.25. The impact of different group voxel radii on the actual leap values is shown in Figure 3.

It can be seen from the rendering results in Figure 4 that the tested data sets benefit significantly from enabled leaping. Even with the very noisy Zeiss data set, a big speedup can be observed even when only considering a region range of only 3% or 7% with virtually no visible difference compared to a standard raycaster. It can also be seen in the diagram, that the segmentation range has a huge impact on the rendering performance as it influences how large regions grow, which again affects the leap values. However, an increase of segmentation range has different effects for different volumes. This is due to the fact that the additional range needed for a substantial increase of the regions might be much larger in one data set than in the other. For example the engine data set consists naturally of very homogeneous regions and distinct transitions, so that the segmentation does not vary much with increasing value ranges and the maximum leap potential is nearly already reached with a low range. Likewise, the visible difference between two renderings of the same volume with different range values is big or small depending on the data set. For the tested data sets, the results of the standard raycaster and the accelerated raycaster are virtually identical for a small segmentation range. However, big ranges potentially lead to significant differences depending on the data set (Figure 5).

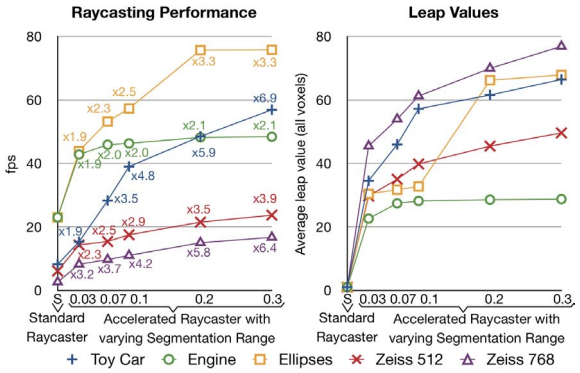

Figure 4: *The average leap values and thus the raycasting performance increase with a bigger region voxel value range. Speedup compared to standard raycaster noted.*

© The Eurographics Association 2009.

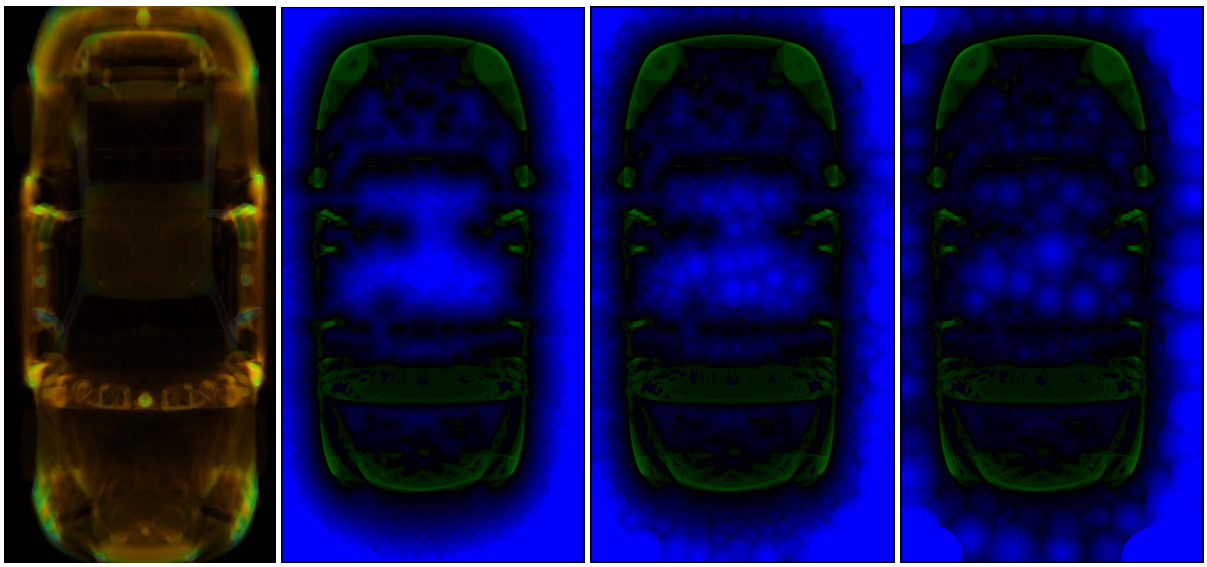

Figure 3: *Visualization of the toy car dataset and its datastructure. Left: Top view with the accelerated raycaster. Right: Volume slice featuring the integrated acceleration structure for the group voxel radii 0.25, 0.5 and 1.0 from left to right. Blue denotes leap values and green density values. A higher color intensity means larger leap or density values respectively.*

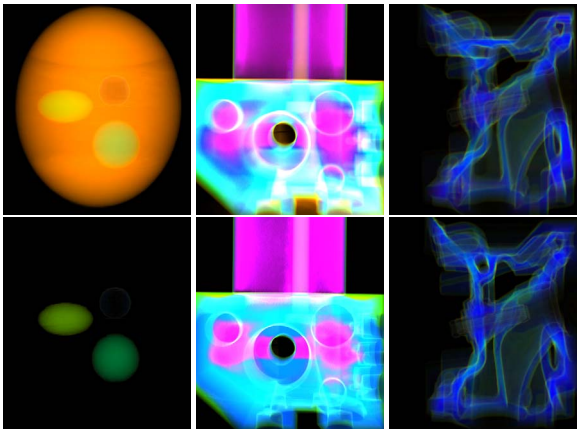

Figure 5: *A big segmentation range potentially but not compulsively results in a significant loss in visual detail (ellipses, Zeiss768 and toy car data sets, each with a region range of 3% at the top and 20% at the bottom).*

7.3. Volume Characteristics

For evaluating the component feature extraction, we counted the number of regions in a data set resulting from volume segmentation and calculated exemplarily for the biggest occupied (non-air) region its relative volume, the number of small holes it contains as well as its compactness employing the isoperimetric quotient $\frac{s^3}{v^2}$ as shape factor with s being the surface area and v denoting the volume of the region (Table 1). The smaller the shape factor is, the more compact is a region, with the minimum being 36π for a sphere.

	Volume	Biggest Occupied Region		
	#Regions	Vol.	Shape Factor	#Holes
Toy Car	21	26%	97659	154
Engine	8	30%	49554	407
Ellipses	5	91%	139	126
Zeiss512	23	26%	97659	18015
Zeiss768	32	24%	228052	31248

Table 1: *Volume characteristics extracted on basis of a segmentation range of 0.1. Vol. denotes the ratio of the regions voxels to the total amount of non-air voxels.*

Segmentation is also used to enhance visual analysis by excluding large homogeneous regions from the visualization and therewith focussing on small structures like defects (Figure 6). Also utilizing the segmented volume, a geometric representation can be extracted (Figure 6) by using a modification of a CUDA marching cubes implementation [NVI08].

8. Conclusions and future work

We proposed a flexible acceleration technique for raycasting based on volume segmentation that is targeted at industrial CT data and presented how it fits in the workflow of deficiency checking and advanced processing. Our approach

© The Eurographics Association 2009.

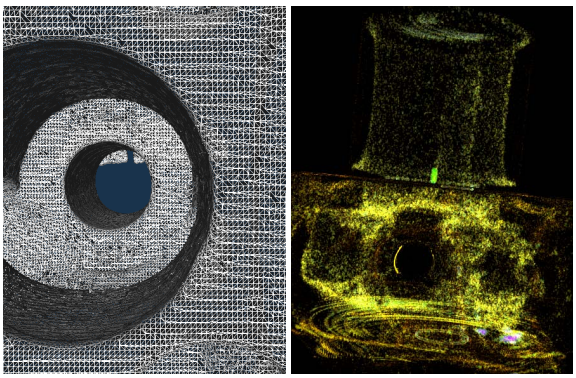

Figure 6: *Left: Triangles extracted from the Zeiss512 dataset. Right: Only voxels are displayed that do not belong to any region. Besides the noise in the data set, several areas with a different density than the surrounding material (i.e. green and purple) get visible that might be defects.*

leads to a significant speedup with only moderate preprocessing time required. While the targeted field of application is material checking, it is well-suited for all other areas with a demand in high rendering performance. For applications in which the segmentation parameters are not driven by the analysis, they can be configured solely to achieve the desired fluent frame rates with an acceptable loss of detail.

For future work, we intend to take the ray direction into account and to allow for non-linear or region-dependant leap values by exploiting the transfer function fetches. Implementing our acceleration technique in a distributed raycaster for further evaluation also remains for future work.

Acknowledgements

The authors would like to thank the Deutsche Forschungsgesellschaft (DFG) for financial support within the Cluster Of Excellence in Simulation Technology at the Universität Stuttgart. The Zeiss data set is courtesy of Daimler AG.

References

[BA02] BULLITT E., AYLWARD S. R.: Volume rendering of segmented image objects. *IEEE Transactions on Medical Imaging 21* (2002), 200–2.

[CS94] COHEN D., SHEFFER Z.: Proximity clouds - an acceleration technique for 3d grid traversal. *The Visual Computer 11* (1994), 27–38.

[DH92] DANSKIN J., HANRAHAN P.: Fast algorithms for volume ray tracing. pp. 91–98.

[EHK*04] ENGEL K., HADWIGER M., KNISS J. M., LEFOHN A. E., SALAMA C. R., WEISKOPF D.: Real-time volume graphics. In *ACM SIGGRAPH 2004 Course Notes* (New York, 2004), ACM, p. 29.

[FS97] FREUND J., SLOAN K.: Accelerated volume rendering using homogeneous region encoding. In *VIS '97: Proceedings of the 8th conference on Visualization '97* (Los Alamitos, CA, USA, 1997), IEEE Computer Society Press, pp. 191–ff.

[GS01] GUTHE S., STRASSER W.: Real-time decompression and visualization of animated volume data. In *VIS '01: Proceedings of the conference on Visualization '01* (Washington, DC, USA, 2001), IEEE Computer Society, pp. 349–356.

[Hei09] HEINZL C.: Analysis and visualization of industrial ct data, 12 2009.

[HMMW03] HUANG R., MA K.-L., MCCORMICK P., WARD W.: Visualizing industrial ct volume data for nondestructive testing applications. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)* (Washington, 2003), IEEE Computer Society, p. 72.

[KSSE05] KLEIN T., STRENGERT M., STEGMAIER S., ERTL T.: Exploiting Frame-to-Frame Coherence for Accelerating High-Quality Volume Raycasting on Graphics Hardware. In *Proceedings of IEEE Visualization '05* (2005), C. Silva and E. Gröller and H. Rushmeier, (Ed.), IEEE, pp. 223–230.

[Lev90] LEVOY M.: Efficient ray tracing of volume data. *ACM Trans. Graph. 9*, 3 (1990), 245–261.

[MSE07] MÜLLER C., STRENGERT M., ERTL T.: Adaptive Load Balancing for Raycasting of Non-Uniformly Bricked Volumes. *Parallel Computing, Special Issue on Parallel Graphics and Visualization 33*, 6 (June 2007), 406–419.

[NVI08] NVIDIA: NVIDIA CUDA SDK code samples. http://developer.nvidia.com/object/cuda.html, 2008.

[Roe] ROETTGER S.: The volume library. http://www9.informatik.uni-erlangen.de/External/vollib.

[SSKE05] STEGMAIER S., STRENGERT M., KLEIN T., ERTL T.: A Simple and Flexible Volume Rendering Framework for Graphics-Hardware–based Raycasting. In *Proceedings of the International Workshop on Volume Graphics '05* (2005), pp. 187–195.

[WGS04] WANG C., GAO J., SHEN H.-W.: Parallel Multiresolution Volume Rendering of Large Data Sets with Error-Guided Load Balancing. In *Eurographics Symposium on Parallel Graphics and Visualization* (2004), pp. 23–30.

[WV90] WILHELMS J., VANGELDER A.: *OCTREES FOR FASTER ISOSURFACE GENERATION*. Tech. rep., Santa Cruz, CA, USA, 1990.

[ZKV92] ZUIDERVELD K., KONING A., VIERGEVER M.: Acceleration of ray-casting using 3d distance transforms. In *Visualization in Biomedical Computing II, Proc. SPIE 1808* (1992), pp. 324–335.

© The Eurographics Association 2009.