Eurographics Symposium on Parallel Graphics and Visualization (2006)
Alan Heirich, Bruno Raffin, and Luis Paulo dos Santos (Editors)

Accelerated Volume Rendering with Homogeneous Region Encoding using Extended Anisotropic Chessboard Distance on GPU

A. Es[1,2] ,H.Y. Keleş[1,2],V. İşler[2]

[1]Tübitak-Bilten METU, Ankara, Turkey
[2]Department of Computer Engineering METU, Ankara, Turkey

Abstract

Ray traversal is the most time consuming part in volume ray casting. In this paper, an acceleration technique for direct volume rendering is introduced, which uses a GPU friendly data structure to reduce traversal time. Empty regions and homogeneous regions in the volume is encoded using extended anisotropic chessboard distance (EACD) transformation. By means of EACD encoding, both the empty spaces and samples belonging to the homogeneous regions are processed efficiently on GPU with minimum branching. In addition to skipping empty spaces, this method reduces the sampling operation inside a homegeneous region using ray integral factorization.The proposed algorithm integrates the optical properties in the homogeneous regions in one step and leaps directly to the next region. We show that our method can work more than 6 times faster than primitive ray caster without any visible loss in image quality.

Categories and Subject Descriptors (according to ACM CCS): I.3.1 [Computer Graphics]: Parallel Processing, I.3.3 [Computer Graphics]: Viewing Algorithms,Bitmap, I.3.3 [Computer Graphics]: And Framebuffer Operations

1. Introduction

Increasing programmability of commodity graphics processing units enabled visualization of 3D scalar density fields in high quality at interactive frame rates. Despite many successes achieved in this field, there still exists accuracy and efficiency problems in visualization of real world scientific data. Rendering the whole volume content accurately takes considerable amount of computational time.

Early acceleration methods use hierarchical data structures such as Kd trees [SS90] and octrees [Lev88] to skip empty regions of volume data. Afterwards, several works extended the usage of hierarchical data structures so as to provide acceleration of rendering homogenous regions as well as empty regions [DH92], [LH91]. However, usage of complex data structures has its own cost (i.e. octrees). Recent techniques explored other forms of encoding schemes such as look-aside buffers, proximity clouds [CS94], and shell encoding [UO93] to skip empty spaces. On the other hand, more recent approaches proposed new algorithms and data structures in order to take advantage of internal parallelism and efficient programmability of the dedicated graphics hardware utilities [LK03], [LMK03], [KW03]. In this work, we focus on the advantages of using both the distance encoding schemes and the internal parallelism of the new generation programmable graphics hardware to accelerate rendering.

GPUs are parallel stream processors and they favor simple localized data access, exploiting instruction level parallelism and arithmetically intensive kernels for maximum efficiency [PF05]. Thus GPU friendly data structures and algorithms should conform to the stream processing model for efficient processing. 3D regular grid acceleration structures are ideal for GPUs due to the simplicity of the structure. Many acceleration techniques have been proposed in the past for ray casting. Refer to [Hav00] for detailed discussion and comparison of different acceleration methods for ray traversals. Some of the fastest known grid based traversal algorithms use distance transformations or macro regions to accelerate ray casting [ZKV92], [CS94], [SK00], [Dev89].

Essentially, distance based methods utilize distance fields, which are calculated during a preprocessing stage. Proximity information to the nearest objects are stored in these fields. Distance values are calculated using Euclidian, city block, chessboard or chamfer metrics. Distance based algorithms accelerate traversals by skipping empty macro regions with the encoded information. In volume rendering, in addition to skipping of the empty spaces for acceleration, it is also possible to make use of coherency and process homogeneous regions in one step [FH98]. We define a homogenous region as a group of neighboring voxels sharing the same or very similar optical properties.

This work is about acceleration of volume ray casting using distance based techniques on the GPU. The main contribution of this paper is the introduction of a GPU based homogeneous region skipping algorithm using (EACD) fields [Es05]. We extend anisotropic chessboard distance fields [SK00] and devised a GPU friendly ray casting algorithm. Additionally, previously known distance based space skipping and primitive ray casting algorithms are adapted to GPU in order to compare performances. The adapted space skipping methods include Cohen and Sheffer's [CS94] proximity clouds (PC) and Sramek and Kaufman's [SK00] anisotropic chessboard distance based ray traverser (ACD). A primitive ray caster sweeps the volume through the rays by taking constant intervals in each turn even if the region being traversed is empty. Distance field methods (PC, ACD, EACD) on the other hand, skip range of empty voxels in big steps. We also extend PC and ACD traversals to facilitate the skipping of not only empty spaces but also any homogeneous regions in the volume. During the classification of homogeneous regions, we impose a predefined error threshold. This threshold is determined experimentally such that human eye can not capture the errors caused by this minor difference.

The remainder of the paper is organized as follows. The next section discusses the volume ray casting and factorization of ray integral that we use during homogeneous space leaping. The third section explains the distance based homogeneous region leaping technique. Next, our GPU based EACD volume renderer is explained. The test results are presented and discussed in the following chapter. Finally, the conclusion is given.

2. Volume Ray Casting

Raw 3D volume data contains scalar density values in each voxel grid. Optical parameters have to be determined for each voxel in order to display light interaction with the volume densities and obtain realistic-looking rendering results. This is performed in a preprocessing step by defining a transfer function, which maps density values in each voxel to opacity and color values. For this work we used the classification method proposed by Mark Levoy [Lev88]. In addition to opacity and color information, approximate surface gradients are determined during this classification method. In this work, we use opacity and approximate surface gradients as for the optical properties.

During the traversal, ray is sampled trough the ray direction from front-to-back viewing order with a constant step size. At each sample point the effects of the optical properties are integrated with the effects of incoming sample's optical properties to obtain the accumulated color and opacity values trough that ray. This is achieved with the well known numerical solution for the ray integral (equation (1)).

$$\begin{aligned} C_r &= C_1\alpha_1 + (1-\alpha_1)C_2 \\ \alpha_r &= \alpha_1 + (1-\alpha_1)\alpha_2 \end{aligned} \tag{1}$$

2.1. Homogeneous Space Ray Integration

The sampling and reconstruction operation is very time-consuming during the ray traversal. In order to minimize this cost, our method groups the maximum number of voxels with similar optical properties as belonging to the same region, namely homogeneous region with EACD encoding. There is no need for multiple sampling and composition operations in a ray segment inside a homogeneous region. Our method exploits the ray integral factorization method [FH98]. In this method, we make only one fetch operation to obtain the optical properties of the sample points belonging to the homogeneous ray segment. These parameters are then used to calculate accumulated color and opacity values for the entire segment. Ray integral factorization along a ray is expressed as in equation (2).

$$\begin{aligned} C_r &= \sum_{i=1}^{n}\left[C_i\alpha_i\prod_{j=1}^{i-1}(1-\alpha_j)\right] \\ &= \sum_{i=1}^{m}\left[C_i\alpha_i\prod_{j=1}^{i-1}(1-\alpha_j)\right] + \\ &\left[\prod_{k=1}^{m}(1-\alpha_k)\right]\sum_{i=m+1}^{n}\left[C_i\alpha_i\prod_{j=m+1}^{i-1}(1-\alpha_j)\right] \\ &where, m\in[1,n] \end{aligned} \tag{2}$$

According to this factorization, when a ray enters into a homogeneous region and passes n samples until exiting the region, we calculate the accumulated color and alpha values inside the region according to equation (3).

$$\begin{aligned} C_{r_i} &= C(1-(1-\alpha)^n) \\ \alpha_{r_i} &= 1-(1-\alpha)^n \end{aligned} \tag{3}$$

where, C_{r_i} and α_{r_i} are the accumulated color and opacity values respectively for the homogeneous region i, with n samples. The final accumulated color of the ray is computed using regular ray integration formula (4).

$$C_{final} = \sum_{i=1}^{k} \left[C_{r_i} \prod_{j=1}^{i-1} (1 - \alpha_{r_j}) \right] \quad (4)$$

According to equation (4), k is the number of regions located in sequence through the ray direction.

3. Distance Based Homogeneous Region Leaping

Distance based methods compute the distance values to the nearest non-empty voxel per voxel basis. In homogeneous region encoding, the values represent the distance to the nearest voxel belonging to a different region. Thus there is no simply empty or non-empty type classification. This type of classification essentially affects the computation of distance fields and the traversal algorithm itself.

Previous works on distance fields based volume ray casting basically relies on proximity clouds. Cohen et al.'s PC [CS94] utilize isotropic 3D distance field. Isotropy comes from the fact that there is a single distance value per voxel, representing the maximum interval that rays can advance regardless the direction. When Euclidian metric is used, the distance value represents the radius of the homogeneous region centered in the voxel. In the original algorithm, distance values are stored in background (empty) voxels. On the other hand we do not classify voxels as empty or non-empty. Instead, we treat all homogeneous regions as the same (including non-empty and transparent voxels). Therefore a secondary 3D grid is necessary for keeping the values of the distance field. Each voxel has a corresponding distance value. Both the volume data and the distance field are represented by 3D textures on GPU. In homogeneous regions, a ray moves forward as long as its distance value. The ray integration within the homogeneous segment of a ray is performed using the factorization method explained in Section 2.1. Number of sample points spanned by the ray in this region (n), is calculated as the ceiling of the ratio of distance value to the constant step size. In the vicinity of a different region, the traversal switches to the primitive ray casting mode with constant steps.

More recently Sramek and Kaufmann [SK00] utilized anisotropic chessboard distance fields to further accelerate the ray casting. Their chessboard distance traversal method does not need to switch between the stepping modes. In the original algorithm empty regions can be skipped fully. The algorithm can also work not only for regular grids but also for rectilinear grids with some additional cost. They observed that in distance based traversals, ray steps get shorter as it gets closer to the objects, and many small steps are taken until the ray gets far away from the close vicinity. To alleviate this problem, they propose using anisotropic chessboard distances depending on the ray directions. Rays are classified by the component sign of their directions ($\pm x$, $\pm y$, $\pm z$) giving eight direction octants. Thus in ACD, instead of

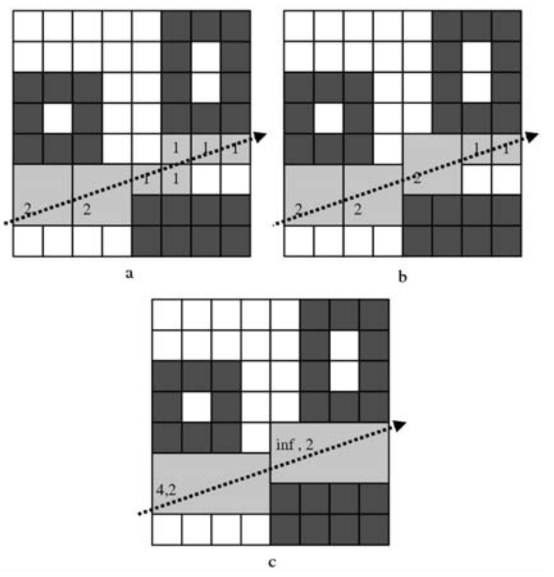

Figure 1: *Isotropic distance field has single isotropic distance value per voxel, while ACD filed stores a distance value for each direction quadrant. EACD on the other hand stores different distance values for each primary axis. (a),(b) and (c) depict example traversals based on the isotropic, ACD and EACD fields respectively. EACD traversal significantly reduces the number of traversal steps in this situation.*

a single isotropic distance, one of the eight distance values are utilized based on the ray direction. The appropriate distance value to be used is determined by the component signs of the ray direction. The original ACD based traversal algorithm has many conditional execution paths which cause major performance hit for the current generation of graphics processors. Therefore we use a more GPU friendly ray casting algorithm based on [Es05]. Both ACD and EACD methods use the same algorithm with a minor difference. The details of the algorithm are given in the next section. In this work, we use the same homogeneous ray integration as in PC method for both ACD and EACD algorithms.

4. EACD Ray Casting

Note that although eight distance values form an anisotropic shape around the voxel in ACD, the homogeneous regions defined by each of these values are cubic (assuming unit voxel dimensions). EACD extends this structure in such a way that it allows the definition of non-cubic homogeneous regions. We observe that non-cubic regions may reduce the number of traversal steps considerably as shown in Figure 1.

The original traversal algorithm we developed in [Es05] essentially addresses ray tracing. In this work the algorithm is adapted for volume ray casting. In the original algorithm, rays skip empty regions fully through the border of the next

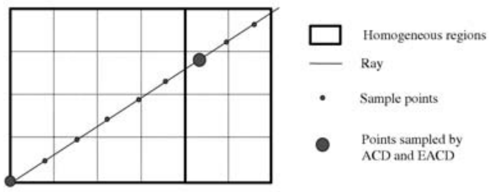

Figure 2: *ACD and EACD sample points are aligned with the primitive ray caster's sample points. In the figure, homogeneous region leaping requires 2 samples as opposed to 9 samples needed by a primitive ray caster.*

region. In order to find next position, the ray is intersected with the border planes defined by the homogeneous region. The intersection point giving the minimum parametric distance is selected as the next ray position. On the other hand, volume ray casting with homogeneous region leaping require some adjustments to this scheme. The image rendered with EACD should be no different than the image of primitive ray caster. For that reason contrary to [Es05], the next position of a ray during the traversal is aligned with the primitive ray caster's sample points as shown in Figure 2. The aligned position can be computed by dividing the homogeneous region distance to the constant ray step size. The ceiling of this division gives how many samples we can safely skip. The ray is advanced by the number of ray step times the constant step size. This scheme works even if the homogeneous region is only one voxel. Note that, we use point sampling (nearest neighbor) instead of tri-linear filtering. In case of tri-linear filtering, opacity and normal values may change in the border voxels due to the interpolation. Since the optical properties of inner voxels are very similar, fetching one sample from this region is still sufficient for the ray integral factorization.

4.1. Implementation

The implementation is done using OpenGL 2.0 and Cg 1.4 toolkit [MGAK03] with FP40 profile. Our GPU allows for dynamic looping and branching. Therefore the whole ray casting operation can be executed within a single fragment program.

The volume data itself contains a normal and opacity value for each voxel. Volume is stored in a 4-component 16-bit floating point formatted 3D texture. Similarly, the distance field is stored in another 3D texture. In order to reduce memory requirements 16- bit packed color format (3×5 bits for RGB, 1 bit for alpha) is used for EACD texture. Since EACD and ACD utilize different distance values for each direction octant, the distance field texture is enlarged by a factor of 2 along each direction. As a result, for each volume voxel, there are 8 corresponding distance field voxels.

Prior to ray casting, a ray generator program is run to cre-

```
float4 RayCast ( Sampler2D $Tex_{direction}$, $Tex_{origin}$ ,
                 Sampler3D $Tex_{volume}$, $Tex_{distance}$ ,
                 float2 rayIdx ,
                 float  $t_{step}$ ,
                 shadingParameters )
{
  // read ray direction and origin
  float3 $D_{ray}$ = Fetch( $Tex_{direction}$, rayIdx )
  float3 $O_{ray}$ = Fetch( $Tex_{origin}$, rayIdx )
  // initialize final color and alpha
  float3  $C_{final}$ = (0,0,0)
  float   $A_{final}$ = 0
  // loop while the ray is not fully opaque and is inside the volume
  while ( $A_{final} < 1$ ) and InsideScene( $O_{ray}$ )
  {
    // read the voxel data
    float3 voxcoords  = floor( $O_{ray}$ )
    float3 Normal = Fetch( $Tex_{volume}$, voxcoords ).rgb
    float  $A_{sample}$  = Fetch( $Tex_{volume}$, voxcoords ).a
    // read distance value from the distance texture based on ray
      position and direction
    float3 distance = Fetch( $Tex_{distance}$,
                       ComputeDistanceTexCoords ($O_{ray}$, $D_{ray}$) )
    // define the intersection borders for the region
    float3 regionBorder = ComputeRegionBorders( voxcoords,
                                                 distance )
    // compute the intersection borders for the region
    float3 $t_{intersections}$ = IntersectRayWithRegionBorders( $O_{ray}$, $D_{ray}$,
                                                 regionBorder )
    // find the parametric distance of the nearest intersection point
    float  $t_{distance}$ = min( $t_{intersections}$.x, $t_{intersections}$.y, $t_{intersections}$.z )
    // compute the number of ray steps that can be leaped
    int    n = ceil( $t_{distance}$ / $t_{step}$ )
    // compute the region color and alpha using Eq. (3)
    float $A_{region}$ = (1-pow( (1-$A_{sample}$), n))
    float3 $C_{region}$ = $A_{region}$ * Phong( $O_{ray}$, $D_{ray}$, Normal,
                                    shadingParameters )
    // accumulate the final color and alpha with the region color and
      alpha using Eq. (4)
    $C_{final}$ = $C_{final}$ + $C_{region}$*(1-$A_{final}$)
    $A_{final}$ = $A_{final}$ + $A_{region}$*(1-$A_{final}$)
    // move ray
    $O_{ray}$ = $O_{ray}$ + $D_{ray}$*(n* $t_{step}$)
  }
  return float4($C_{final}$ , $A_{final}$)
}
```

Figure 3: *Cg like pseudo-code for the EACD ray casting. Note that most of the operations works on vectors. These operations can be implemented using the SIMD instructions of the GPU.*

ate and clip rays to the bounding box of the scene. As for the output, ray generator creates ray origin and direction textures which have 4-component floating-point color format. Rays intersecting the scene are then transformed into the volume coordinate space. By this way, the voxel indices of a point can be easily computed by taking the floor of its coordinates during the ray casting.

The Cg like pseudo-code for the EACD ray casting is given in Figure 3. Variable types are explicitly given in order to reveal the vectoral nature of the algorithm. The code has no data dependent branching inside the main traversal loop. ACD traversal is almost identical to this one. The only difference is instead of three distance values, one value is fetched from the distance texture. In the pseudo-code, $Tex_{direction}$, Tex_{origin} are the 2D ray textures created by the ray generator. Tex_{volume} is the 3D volume data texture which contains

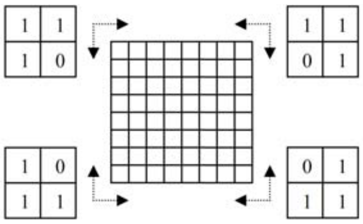

Figure 4: *ACD field creation in 2D. Four masks are applied in the shown directions. Four (anisotropic) distance fields are generated as the result.*

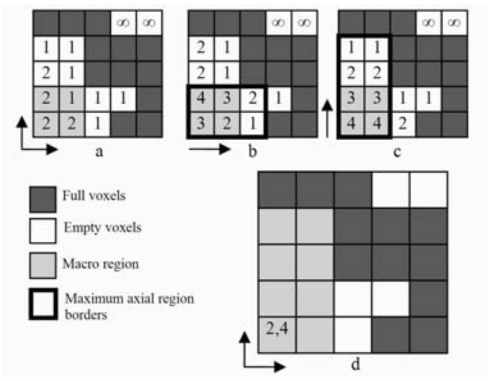

Figure 5: *Finding EACD region for the lower left cell. Only (+x,+y) direction quadrant is shown. Arrows denote the orientation of the distance values (a) is the base ACD grid. (b), (c) are the axial distance grids along +x and +y directions respectively. (d) is the resulting EACD distances.*

voxel normals and opacity values, while Tex$_{distance}$ is the 3D distance field texture as explained previously in this section. rayIdx is the ray index which is actually 2D texture coordinates to the ray textures. t$_{step}$ is the parametric distance for the constant ray step. A primitive ray caster always moves rays by this distance in each step. Finally shadingParameters is the structure keeping the shading parameters such as light position, diffuse and specular colors as well as the material colors. The output of the program is the final accumulated color of the ray. Note that this code needs some minor adjustments for tri-linear filtering. The computation of the number of ray steps should be altered in such a way that the ray should take constant step in border voxels. Border voxels can be easily determined by looking at the distance value (i.e. if any component of the distance vector is 0, it is considered as the border voxel). The resulting code should be like below:

n = IsBorderVoxel(distance) ? 1 : floor($t_{distance}$ / t_{step})

4.2. Construction of the EACD Field

A heuristic with a simple greedy search is used in order to create EACD field. The heuristic is to find the largest homogeneous region per voxel basis. ACD fields representing the largest cubic homogeneous regions around the voxels can be constructed rather simply. On the other hand, finding the largest non-cubic homogeneous regions from scratch can be a very time consuming process. Therefore, we rely on the ACD field and extend regions along the main axes to construct the EACD field. As a result, building the acceleration structure involves two phases: The first phase is exactly the same as creating ACD field. The strategy to find the distance values is based on the idea of propagating local distances over the grid cells. Firstly, cell values of the distance field are initialized to infinity. Then a mask is overlaid onto each cell of the field in a specific direction (such as beginning from top-left to bottom-right). Each element of the mask is summed with the distance value of the underlying field element. The resulting value of the cell is the minimum of these sums and the initial distance value of the center element. In this procedure, the distance value of the overlaid element is computed using the following algorithm:

$$\text{Dist}(V_{i,j,k} \text{ due to } V_{x,y,z}) = \begin{cases} \text{Dist}(V_{i,j,k}) & \text{,if inSameGroup}(V_{x,y,z}, V_{i,j,k}, T_o, T_a) \\ 0 & \text{,otherwise} \end{cases}$$

Where V$_{x,y,z}$are the voxel coordinates of the current center voxel and V$_{i,j,k}$ are the voxel coordinates for the overlaid element. Function *inSameGroup* reads the opacities and the surface gradients of V$_{x,y,z}$ and V$_{i,j,k}$. V$_{i,j,k}$ is classified as belonging to a different region than V$_{x,y,z}$, if the opacity difference is greater than the opacity threshold, T$_o$, or the angle difference is greater than the angle threshold, T$_a$.

Generation of ACD field involves applying eight different masks (four for 2D), to the grid data beginning from one of the corners towards the opposite diagonal corner (Figure 4). Consequently, eight distance fields for each direction octant are created. These fields are then interleaved into a single big grid with eight times the size. The computational complexity of ACD transformation is O(n), where n is the number of voxels. In the second phase, we first create six axial distance grids representing the range of voxels belonging to the same group along the $\pm x$, $\pm y$ and $\pm z$ axis directions. In order to define EACD regions, by using the auxiliary axial distance grids and ACD field we determine how much the cubic homogeneous regions can be extended in a greedy manner. This operation is done for each voxel. The extension procedure is carried as follows: Border voxels of the empty regions are walked and maximum possible extension along the main axes is computed. The cubic region is then extended along the axis giving the maximum volume. This step is repeated once more, for one of the remaining two axes which is giving the maximum volume. Figure 5 depicts computing EACD macro region for a voxel in 2D.

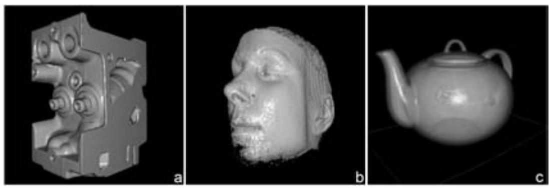

Figure 6: *(a)engine, (b) mrbrain, (c) teapot*

Table 1: *Performance results of the ray casting methods. Results are in milliseconds. Last column is the speedup achieved by EACD compared to primitive ray casting.*

Data	R.Cast	PC	ACD	EACD	Speedup
engine	183	83	42	33	554%
mrbrain	206	101	51	41	502%
teapot	346	121	62	49	706%

Distance field computation for tri-linear interpolation requires some modifications to this method: Border voxels are marked prior to the creation of ACD field. During the distance computation, voxels marked as the border voxels are always classified as belonging to a different region.

5. Results And Discussion

For testing, a number of well known volume datasets were used. The rendered images are shown in Figure 6. Tests were run on a 512MB GeForce7800 GTX graphics board. The frame size for the rendered images is 512×512. As seen in Table 1, the speedup compared to ACD is around 25%, while it is as much as 700% compared to primitive ray caster. EACD is especially advantageous if the volume is composed of many non cubic homogeneous regions. Since both ACD and EACD use the same algorithm essentially, their performances should be almost equal in the worst case. On the other hand, for EACD, maximum distance limit of 32 (5 bits) imposed by the low precision distance texture format may cause some performance penalty in very large homogeneous regions. But none of the volume datasets we experimented on revealed such a problem. As an empty space skipping technique, PC traversal does not perform as fast as EACD or ACD. This is largely the result of shorter ray steps caused by the isotropic regions. Although the loop body of the primitive ray caster is fairly short and efficient, it performs the worst compared to the region leaping methods.

Figure 7 illustrates the average number of loops performed by rays to finish the rendering. Brighter regions indicate higher loop counts. It is clearly seen from the image that ACD and EACD require considerably lower loop counts than PC. Among all methods, EACD can render the image with the least average number of loops; the traversal step counts are generally 10% to 30% lower than ACD.

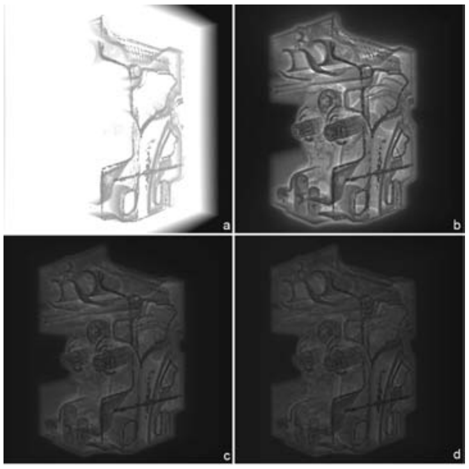

Figure 7: *Illustration of loop counts for (a)Ray caster, (b)PC, (c)ACD, (d)EACD. Brightness and contrast is adjusted for visual clarity. Darker regions indicate lower loop counts.*

The opacity and angle thresholds T_o and T_a, generates different homogeneous regions. As threshold values increase, larger homogeneous regions are formed and larger optical variability within the regions is allowed. Although the rendering speed may increase, this situation essentially causes errors on the rendered image. The T_o and T_a parameters can be adjusted to compromise between the image quality and rendering time. Note that if the volume data has many empty regions, huge speedup may be achieved even for the lower threshold values. Figure 8 shows the images rendered with several different T_o and T_a thresholds. Especially higher opacity thresholds cause more artifacts as seen in Figure 8.c. On the other hand increasing angle threshold did not create any noticable difference for this data. The reason is that most of the neighboring voxels in high curvature areas are already classified as different regions due to the chosen opacity threshold.

The problem of ACD and EACD is that, the acceleration structure is eight times as big as it is for PC. Since the maximum allowed 3D texture resolution is 512^3 in our hardware, the largest dimensions of the volume data can be 256^3. As the size of the local graphics memory enlarges, this will be less of a concern, but a better solution to this problem may be to explore hybrid partitioning structures.

6. Conclusion

In this work we have introduced an acceleration technique for volume ray casting by means of EACD transformations on GPU. In this technique, we exploited the coherency existing in most of the scientific volume data. By this way traversal through the empty regions and homogeneous regions is

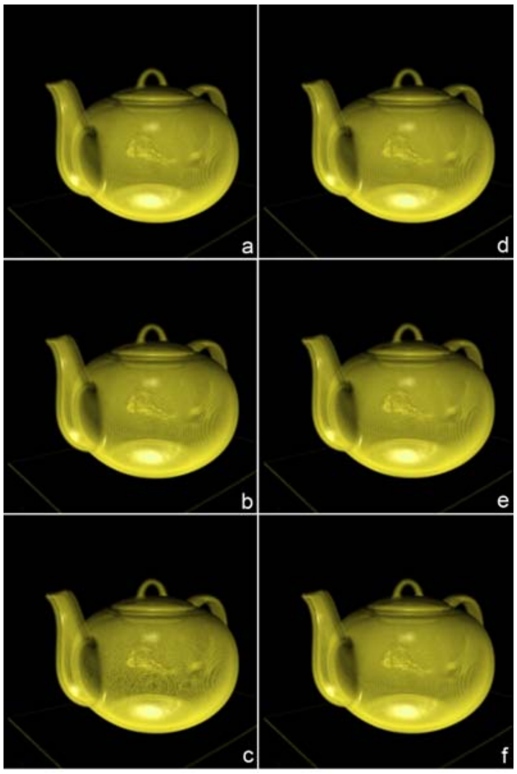

Figure 8: *Images rendered with ACD using different segmentation thresholds. (a)T_o:0.01,T_a:3, (b)T_o:0.02,T_a:3, (c)T_o:0.1,T_a:3, (d)T_o:0.01,T_a:1, (e)T_o:0.01,T_a:10, (f)T_o:0.01,T_a:20*

achieved efficiently. Within a homogeneous region, ray integral is calculated in one step using a factorization method. In order to compare the performance, GPU versions of some of the previously known ray casting techniques are implemented. Our EACD traversal algorithm extensively uses instruction level parallelism and requires minimum number of dynamic branching, making it very efficient for the current generation of GPUs. It is shown that the introduced traversal algorithm is several times faster than a primitive ray caster, and considerably faster than other distance based homogeneous region leaping methods. In addition, our algorithm suits well to the modern pipelined super-scalar CPU architectures which support streaming parallel instructions. Therefore, presented methods can be ported to SIMD capable CPUs easily.

References

[CS94] COHEN D., SHEFFER Z.: Proximity clouds: an acceleration technique for 3d grid traversal. *The Visual Computer 10*, 11 (nov 1994), 27–38.

[Dev89] DEVILLIERS O.: The macro-regions : an efficient space subdivision structure for ray tracing. In *Proc. Of Eurographics'89* (1989), pp. 27–38.

[DH92] DANSKIN J., HANRAHAN P.: Fast algorithms for volume ray tracing. In *Workshop on Volume Visualization* (1992), pp. 91–98.

[Es05] ES A.: Acceleration of regular grid traversals using extended chessboard distance transformation on gpu. In *CAD/Graphics 2005* (2005).

[FH98] FUNG P., HENG P.: Efficient volume rendering by isoregion leaping acceleration. In *The Sixth International Conference in Central Europe on Computer Graphics and Visualization'98* (1998).

[Hav00] HAVRAN V.: *Heuristic ray shooting algorithms*. PhD Dissertation,The Faculty of Electrical Engineering, Czech Technical University, 2000.

[KW03] KRUGER J., WESTERMANN R.: Acceleration techniques for gpu-based volume rendering. In *IEEE Visualization'03* (2003), pp. 287–292.

[Lev88] LEVOY M.: Display of surfaces from volume data. In *IEEE Computer Graphics and Applications* (1988), vol. 8, pp. 29–37.

[LH91] LAUR D., HANRAHAN P.: Hierarchical splatting: A progressive refinement algorithm for volume rendering. In *Proc. of SIGGRAPH'91* (1991), pp. 285–288.

[LK03] LI W., KAUFMAN A.: Texture partitioning and packing for accelerated texture-based volume rendering. In *Graphics Interface* (2003), pp. 81–88.

[LMK03] LI W., MUELLER K., KAUFMAN A.: Empty space skipping and occlusion clipping for texture-based volume rendering. In *IEEE Visualization'03* (2003), pp. 317–324.

[MGAK03] MARK W. R., GLANVILLE R. S., AKELEY K., KILGARD M. J.: Cg: a system for programming graphics hardware in a c-like language. In *ACM Transactions on Graphics (Proc. ACM SIGGRAPH)* (2003), vol. 22, pp. 896–907.

[PF05] PHARR M., FERNANDO R.: *GPU Gems 2*. Addison Wesley Professional, 2005.

[SK00] SRAMEK M., KAUFMAN A.: Fast ray-tracing of rectilinear volume data using distance transforms. In *IEEE Transactions On Visualization And Computer Graphics* (2000), vol. 6, pp. 236–252.

[SS90] SUBRAMANIAN K. R., S.FUSSEL D.: Applying space subdivision techniques to volume rendering. In *Proc. of Visualization'90* (1990), pp. 150–158.

[UO93] UDUPA J., ODHNER D.: Shell rendering. In *IEEE Comp. Graph. and Applications(13)* (1993), pp. 58–67.

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