Eurographics/ IEEE-VGTC Symposium on Visualization (2006)
Thomas Ertl, Ken Joy, and Beatriz Santos (Editors)

Multiresolution Interblock Interpolation in Direct Volume Rendering

Patric Ljung[†], Claes Lundström[‡] and Anders Ynnerman[†]

[†]Norrköping Visualization and Interaction Studio, Linköping University
[‡]Center for Medical Image Analysis and Visualization, Linköping University and Sectra Imtec AB

Abstract

We present a direct interblock interpolation technique that enables direct volume rendering of blocked, multiresolution volumes. The proposed method smoothly interpolates between blocks of arbitrary block-wise level-of-detail (LOD) without sample replication or padding. This permits extreme changes in resolution across block boundaries and removes the interblock dependency for the LOD creation process. In addition the full data reduction from the LOD selection can be maintained throughout the rendering pipeline. Our rendering pipeline employs a flat block subdivision followed by a transfer function based adaptive LOD scheme. We demonstrate the effectiveness of our method by rendering volumes of the order of gigabytes using consumer graphics cards on desktop PC systems.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Viewing algorithms; I.4.10 [Image Processing and Computer Vision]: Volumetric;

1. Introduction

Direct volume rendering [Kau91] has been shown to be a very valuable tool in the visualization and exploration of volumetric data sets. In many areas, including medical imaging and scientific simulations, the size of the generated data sets are causing severe performance limitations in the volume rendering pipeline. Methods that can smoothly adapt to the available resources in the rendering system are therefore highly desired. Currently the primary means to achieve this is through level-of-detail (LOD) techniques for volumetric data. Reducing the LOD in different local regions reduces memory, transfer bandwidth and processing requirements.

LOD schemes are typically based on a block subdivision of the volume. The different local resolutions then cause discontinuities in the rendered image. Several variations of sample replication between blocks have been proposed to minimize or avoid these artifacts but replication has a number of unwanted side effects. The major drawbacks are that the redundancy introduced significantly decreases the data reduction achieved by the LOD selection and that significant block interdependency is introduced in the LOD construction process. Furthermore, there are potential issues with rendering quality due to incorrect sample positioning, and the block interdependence makes effective parallelization harder.

In this paper we introduce an interblock interpolation technique that removes the need for replication and supports $\mathcal{C}^0$ continuity between arbitrary resolution levels. The main benefit of this approach is that the full data reduction is kept throughout the rendering pipeline without increasing rendering error. The interblock interpolation has been implemented in a 3D texture slicing volume ray caster using fragment programs. We summarize our contribution to be:

- Provision of high quality rendering without discontinuities arising from blocking.
- Permitting high LOD adaptivity through smooth interpolation between arbitrary resolutions.
- Avoiding data replication such that data reduction rates can be maintained.
- Supporting highly parallel LOD preprocessing since the proposed method does not impose any interblock dependencies.

[†] {plg,andyn}@itn.liu.se
[‡] clalu@cmiv.liu.se

© The Eurographics Association 2006.

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

Both hierarchical (octree) and flat block subdivision schemes have been used for LOD implementations. A drawback of hierarchical schemes is that low-resolution blocks cover a large spatial region. This means that the granularity of the LOD selection is low, which leads to poor data reduction in some cases. In contrast, flat schemes are highly spatially adaptive, which is why we have chosen this approach. To demonstrate the effectiveness of our method, we render large, out-of-core, uncompressed data sets using a Transfer Function (TF) based LOD selection [LLYM04]. Data is dynamically cached from disk when the LOD selection is modified and the reduced volume data is fitted to the available texture memory on consumer graphics cards.

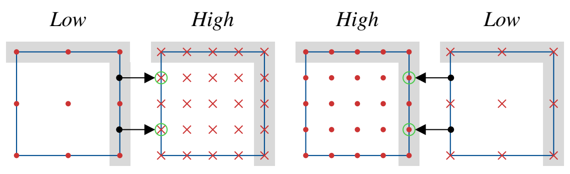
2. Related Work

Multiresolution techniques have been used in a wide range of applications. Currently, the most common scheme in hardware-based volume rendering is a hierarchical scheme (octree) with recursive subdivision and increasing resolution. It was first introduced in volume rendering by LaMar et al. [LHJ99] and extended by Weiler et al. [WWH*00] to minimize discontinuities between blocks of different level-of-detail. The criterion for LOD selection is based on region of interest or distance to viewer.

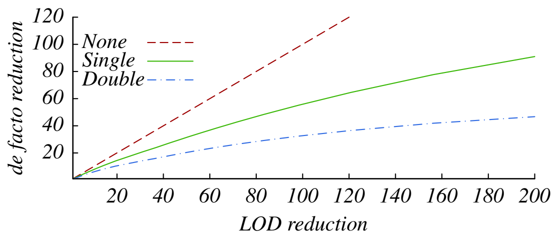
LOD selection based on data error between levels was proposed by Boada et al. [BNS01], and Guthe et al. [GWGS02] combine this with wavelet based compression. LaMar et al. [LHJ03] further propose an efficient method to compute the level error by means of frequency tables and they derive a grey scale Transfer Function (TF) error for the interior octree nodes. Guthe and Strasser [GS04] present a LOD selection through approximation of the screen-space error from the maximum voxel deviation, which is used to overestimate a TF based difference in the separate RGB color channels. Gao et al. [GHSK03] compute a Plenoptic Opacity Function (POF) for each block, based on a set of TF basis functions.

Block-based compression is widely used in image and video compression techniques, such as JPEG and MPEG. Flat blocking schemes have also been used for wavelet based volume compression in [IP99, BIP01, NS01] where the authors present random access techniques for data samples. Ljung et al. [LLYM04] use a TF based LOD selection that exploits the multiresolution properties of the wavelet transform on a per block basis. A simplified block histogram is used to derive an error in the perceptually adapted CIELUV color space. A coarse value histogram has also been used by Gao et al. [GHJA05].

In the majority of the presented hierarchical schemes, blocks are rendered individually and whenever sampling discontinuities are considered, for example in [WWH*00, GWGS02], it is addressed by sample replication (including interpolation to derive the replication sample). This requires access to neighbor data, at specific resolution levels, which may be resident in memory [LHJ99, WWH*00] or decoded and cached as needed [GWGS02]. Guthe et al. [GWGS02] use a single sided replication scheme that is C^0 continuous only under condition that higher resolution samples are replaced with interpolated samples from lower resolution neighbors (figure 1a). While Weiler et al. [WWH*00] propose a method that preserves original samples it comes with considerable replication cost, figure 1b shows the cost for both single and double sided replication. Both of these methods require access to higher resolution neighbor blocks in the LOD creation process for replicated samples, in the double sided scheme up to 26 blocks have to be considered. Furthermore, these methods also suffer from restricted sample placement within the block since they have to be aligned with the block boundary, as seen in figure 1a. Which in turn limits the set of applicable downsampling schemes.



a) Left: Single sided replication (grey area) fails to support continuous interpolation unless the higher resolution block is adjusted (green circles) and accurate data is lost. Right: Missing samples (green circles) are interpolated to provide continuous interpolation.



b) Comparison of de facto data reduction versus LOD reduction for none, single, and double sided sample replication.

Figure 1: *Single sided replication has issues with continuity (a - left). The de facto reduction is significantly lower for replication applied to flat blocking (b).*

3. Direct Multiresolution Interpolation

With direct multiresolution interpolation we mean a method that can directly, without intermediate reconstruction and sample replication, interpolate between sample values within and between blocks of arbitrary resolution levels. First, we establish how sampling within a block is performed, intrablock sampling, and then our scheme for direct multiresolution interpolation is presented.

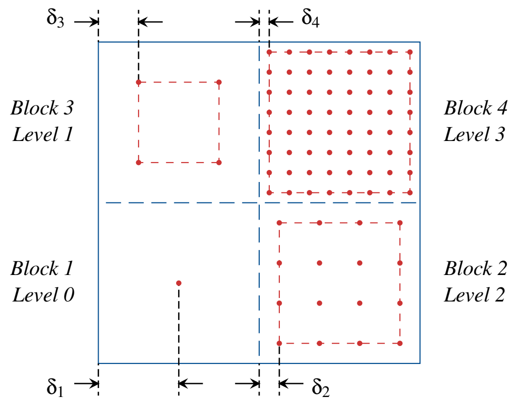


Figure 2: *A 2D illustration of a four-block neighborhood. Red points show the location of block samples, the red dashed lines are the sample boundaries. The sample boundary distance, δ_b, is indicated for each block, b.*

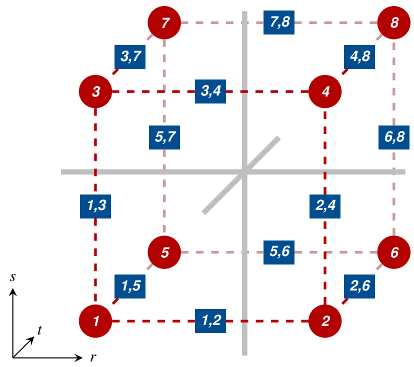


Figure 3: *Center of eight-block neighborhood configuration with block and edge labeling. The gray solid lines indicate the block intersections. The edge labeling is used to define the edge weights, $e_{i,j}(\rho)$, where ρ is the position along the edge in the local coordinate system, r, s, t, with origin at the block intersection center.*

3.1. Intrablock Volume Sampling

Intrablock sampling defines the sampling within the interior of a block, with clamping to the sample boundaries. Commonly, a lower resolution sample is situated at the center of its higher resolution source samples. Figure 2 illustrates in 2D a set of four blocks with different resolutions. The sample placement used in this paper is justified for resampling using averaging but, in general, the placement depends on the downsampling method. We define the *sample boundary* of a block to be the smallest box spanning all samples: for a block of resolution level l, a box inset by $\delta(l)$ from all edges of the block boundary (eq. 1).

$$\delta(l) = \frac{1}{2^{1+l}} \tag{1}$$

The sample to be retrieved is given by the normalized intrablock coordinates $u', v', w' \in [0,1]$. The computation of local block texture coordinates u, v, w, shown for u in equation 2, ensures that samples are not taken outside the boundary.

$$u = \mathbf{C}_{\delta}^{1-\delta}(u'), \tag{2}$$

where $\mathbf{C}_{\alpha}^{\beta}(\gamma)$ clamps the value γ to the $[\alpha, \beta]$ interval.

3.2. Interblock Interpolation

The task of interblock interpolation is to retrieve a sample value, φ, for a position between the sample boundaries of neighboring blocks. The overall structure of the method is given below:

1. Determine the current eight-block neighborhood, r_0, s_0, t_0, and setup the local coordinates, r, s, t.
2. Take samples, φ_b, from the blocks using the intrablock method described above.
3. Compute edge weights, $e_{i,j}$, between side-facing neighbors.
4. Compute block weights, ω_b, from three (in 3D) edge-weights.
5. Compute the normalized weighted sum of $\omega_b \varphi_b$ yielding the sample value, φ.

A block neighborhood is illustrated in figure 3 where block 1 is in the left-lower-front and block 8 is in the right-upper-back. We also define the volume domain to be in block units and the volume thus covers the cubical range $\langle 0,0,0 \rangle$ to $\langle N_r, N_s, N_t \rangle$ where N_ρ is the number of blocks along dimension ρ. The local coordinates, r, s, t, for block 1 and 8 centers are consequently $\langle -0.5, -0.5, -0.5 \rangle$ and $\langle 0.5, 0.5, 0.5 \rangle$, respectively. The eight-block neighborhood is determined by translation and clamping of the global volume coordinates, r_g, s_g, t_g, to yield the block location, r_0, s_0, t_0, (eq. 3) of the left-lower-front block (block 1 in figure 3). With the neighborhood origin determined, the local coordinates, r, s, t, are easily computed (eq. 4). The other two dimensions, s and t, are computed analogously.

$$r_0 = \lfloor \mathbf{C}_0^{N_r-1}(r_g - 0.5) \rfloor, \tag{3}$$
$$r = r_g - r_0 - 1.0. \tag{4}$$

A sample, φ_b, is then taken from each of the blocks using r, s, t as the intrablock coordinates u', v', w' adjusted with unit offsets specific to each block's location in the local eight-block neighborhood.

For each of the twelve edges in the neighborhood, the edge weights $e_{i,j} \in [0,1]$ are computed, as described later. For convenience, three edge sets E_r, E_s, E_t, are introduced for edges of equal orientation (eq. 5). Using the labeling in

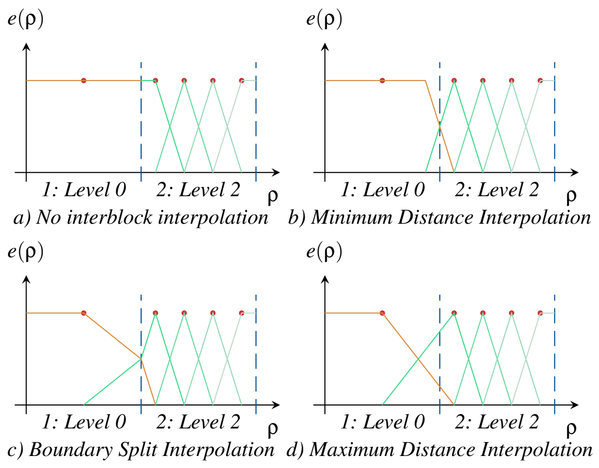


Figure 4: *One-dimensional illustrations of the three interblock interpolation variants (b–d) and nearest neighbor sampling (a). Each graph contain two blocks, the leftmost with resolution level 0 and the rightmost with level 2.*

figure 3 these are:

$$\begin{aligned} E_r &= \{(1,2),(3,4),(5,6),(7,8)\}, \\ E_s &= \{(1,3),(2,4),(5,7),(6,8)\}, \\ E_t &= \{(1,5),(2,6),(3,7),(4,8)\}. \end{aligned} \tag{5}$$

The edge weights, $e_{i,j}$, determine the block weights, ω_b, as shown below (eq. 6).

$$\begin{aligned} \omega_1 &= (1-e_{1,2})\cdot(1-e_{1,3})\cdot(1-e_{1,5}) \\ \omega_2 &= e_{1,2}\cdot(1-e_{2,4})\cdot(1-e_{2,6}) \\ \omega_3 &= (1-e_{3,4})\cdot e_{1,3}\cdot(1-e_{3,7}) \\ \omega_4 &= e_{3,4}\cdot e_{2,4}\cdot(1-e_{4,8}) \\ \omega_5 &= (1-e_{5,6})\cdot(1-e_{5,7})\cdot e_{1,5} \\ \omega_6 &= e_{5,6}\cdot(1-e_{6,8})\cdot e_{2,6} \\ \omega_7 &= (1-e_{7,8})\cdot e_{5,7}\cdot e_{3,7} \\ \omega_8 &= e_{7,8}\cdot e_{6,8}\cdot e_{4,8} \end{aligned} \tag{6}$$

The sample value, φ, is then computed as a normalized sum of all block samples, φ_b (eq. 7). Normalization is required since the edge weights do not always sum to one, as when block neighbors are of different resolutions.

$$\varphi = \frac{\sum_{b=1}^{8}\omega_b\varphi_b}{\sum_{b=1}^{8}\omega_b} \tag{7}$$

We present three variations of interblock interpolation: *Minimum Distance*, *Boundary Split*, and *Maximum Distance*. The variations stem from how the edge weights, $e_{i,j}$, are computed. Figure 4 shows how edge weights are computed for the variants and a comparison and illustration of the block weights for the methods is shown in figure 5.

Minimum Distance Interpolation: The motivation for this method is that a sample should not have impact outside its valid footprint, $2\delta(l)$ (figure 4b). A consequence is that lower resolution samples are extended towards the edge (eq. 8).

$$e_{i,j}(\rho) = \mathbf{C}_0^1(0.5+\rho/2\min(\delta_i,\delta_j)) \tag{8}$$

where $(i,j)\in E_\rho$ and ρ is one of r, s, t.

Boundary Split Interpolation: This variant is based on the idea that the steepness of the interpolation should not be influenced by neighboring blocks. Thus, the interpolation is divided into a two-segment linear function that is split at the spatial block boundary (figure 4c). Compared with Minimum Distance, high resolution samples have wider footprints and the constant part of low resolution samples is removed (eq. 9).

$$e_{i,j}(\rho) = \begin{cases} \mathbf{C}_0^1(0.5+\rho/2\delta_i) & \text{if} \quad \rho < 0 \\ \mathbf{C}_0^1(0.5+\rho/2\delta_j) & \text{if} \quad \rho \geq 0 \end{cases} \tag{9}$$

Maximum Distance Interpolation: A drawback of the other two variants is a discontinuity in the derivative of the interpolation weight within the interval. Maximum Distance avoids this problem, see figure 4d. The value is interpolated in one linear segment over the whole distance between neighboring sample boundaries (eq. 10).

$$e_{i,j}(\rho) = \mathbf{C}_0^1\Big((\rho+\delta_i)/(\delta_i+\delta_j)\Big) \tag{10}$$

The interblock interpolation variants all equate to traditional trilinear interpolation when the blocks have equal resolution. Let $\delta_i = \delta_j = \delta$, then equations 8, 9, and 10 each result in

$$e_{i,j}(\rho) = \mathbf{C}_0^1\Big(0.5+\rho/2\delta\Big) \tag{11}$$

which is a linear interpolation kernel. From the local nature of this method it follows that neighboring blocks do not affect the block sampling within the sample boundary.

The proposed method is $\mathcal{C}^0$ continuous, which follows from the continuity property of summation, product and quotient, given that $\sum_b \omega_b \neq 0$ (eq. 7) and the edge interpolators $e_{i,j} \in \mathcal{C}^0$ (eq. 8, 9, 10). These conditions are met as long as the pair-wise sample distance $\delta_i + \delta_j > 0$ for all $(i,j)\in E_\rho$ and $\delta_b \geq 0$ for all b, conditions which are all met.

4. Implementation

With the interblock interpolation sampling framework established we can proceed to implement a highly adaptive LOD selection scheme. We have incorporated the LOD selection scheme presented in [LLYM04] which uses a basic block histogram approximation together with the active TF to predict block significance. We also use cropping to affect the LOD selection. A memory budget is specified, such that the data fits in texture memory, and the method optimizes the LOD for each block with respect to its required size.

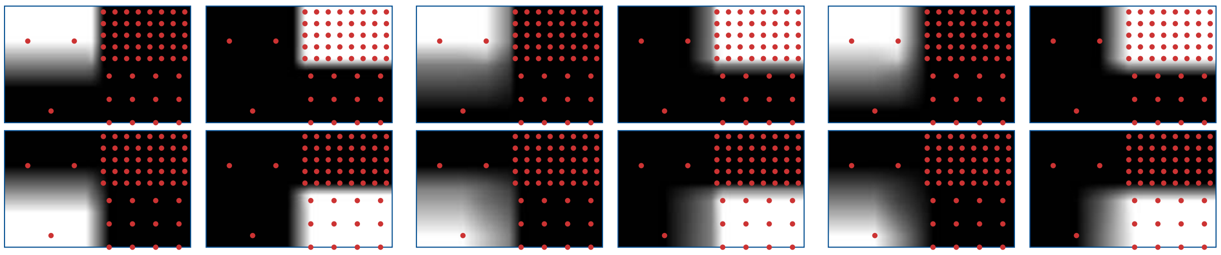

a) Minimum Distance Interpolation
b) Boundary Split Interpolation
c) Maximum Distance Interpolation

***Figure 5:** Block interpolation weights in 2D for a four-neighborhood. The interpolation weight for each specific block is illustrated by the four intensity images. Within each image the blocks resolutions are* 1×1, 4×4, 2×2 *and* 8×8, *from left-bottom to right-top. Images have been cropped vertically.*

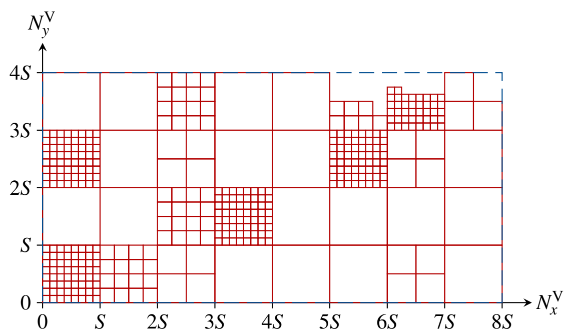


***Figure 6:** 2D illustration of block packing within a single texture object. Texture patches are allocated per level in chunks of the largest block size, S. Each block is represented by a square, smaller squares are lower resolution blocks. Multiple lower resolution blocks are packed tightly within the unit size* $S\times S$*, the largest squares.*

The specific resolution levels per block are requested from the volume data stored on disk and the data kept in main memory constitutes a volume data cache. From this cache a packed block data texture and a block metadata texture, describing block level and location, are created. The packing of the volume data texture is straightforward and conceptually similar to the method proposed in [KE02], see figure 6. Our method differs, primarily, in that we do not store replicated samples and that the level-of-detail is dictated by a LOD selection scheme.

For access to packed block samples in graphics hardware textures, the texture coordinates x,y,z need to be computed. This requires the scale, $\sigma(l) = 2^l$, and origin u_0,v_0,w_0 of each block. As in [KE02], this is looked up in an index data texture storing these metadata for all blocks in the volume. Since the texture coordinates reside in the $[0,1]$ interval, the side sizes of the allocated texture, $\kappa_r,\kappa_s,\kappa_t$, are also required. These are in block units, $S = \sigma(l_{\max})$, with $l_{\max}$ being the highest block resolution level. The complete expression is shown in equation 12.

$$\langle x,y,z\rangle = \langle \frac{\sigma u+u_0}{S\kappa_r}, \frac{\sigma v+v_0}{S\kappa_s}, \frac{\sigma w+w_0}{S\kappa_t} \rangle \tag{12}$$

The renderer then uses a texture-slicing rendering technique [CCF94] and our interpolation scheme is implemented using OpenGL ARB Fragment Programs. Each slice can be rendered in one or two passes. The reason for using two passes is to increase the performance of the renderer by minimizing the use of the interblock interpolation fragment program. The first pass draws all pixels from block-interior samples and zero level blocks. We use the zero level, a single sample value, to indicate completely transparent, non-visible blocks which are also rendered in their full spatial extent in the first pass. The zero level is a degenerate case without any interior domain. The fragment is killed if it is outside a block interior and not a zero level block. The second pass then exploits the early Z-termination feature and thus restricts the execution of the interblock interpolating shader to samples between the blocks.

To further speed up interaction, such as rotating, zooming, cropping and clip plane movements, the sampling density is lowered and/or interblock interpolation disabled while interaction takes place. Since the focus has been on the interpolation quality we have not yet applied more advanced acceleration methods such as empty space skipping, early ray termination and adaptive sampling density.

5. Results

When dealing with multiresolution data, it is important to consider the error introduced by the varying level-of-detail and the error introduced by interblock interpolation. In this paper we focus on the quality of the interblock interpolation schemes rather than of the level-of-detail scheme.

The first results are provided in figure 7, in which a slightly rotated linear gradient, from dark grey to white, is used as input. A linear gradient should only present rounding errors from the intrablock sampling and the choice of

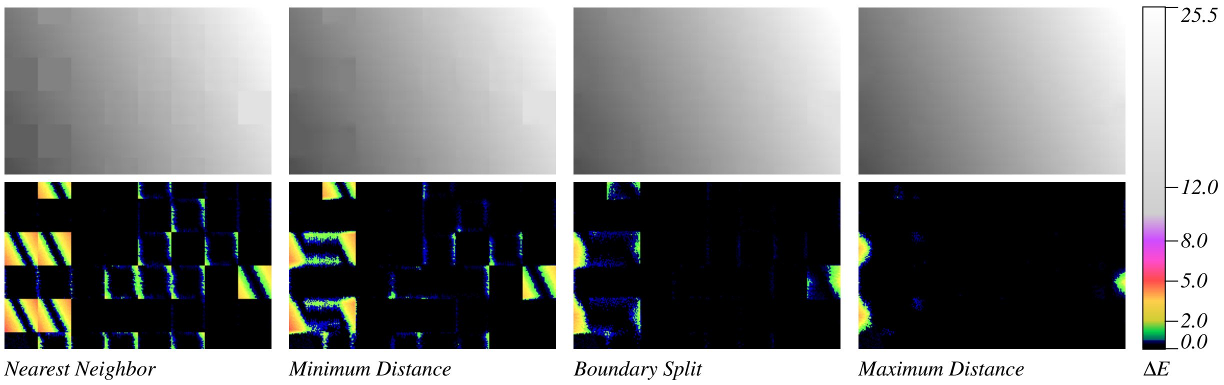


Figure 7: *A comparison of the interpolation methods on a slightly rotated linear gradient. The original image was a 256×256 grey-scale image (8×8 blocks) being reconstructed using random levels between 1×1 and the full resolution, 32×32. The bottom row shows color mapped images of the pixel-wise errors using the ΔE color difference from CIELUV.*

LOD should therefore not affect the results. The reconstructed output has the same sampling resolution as the input, 256×256 (the images shown in the figure are cropped), and we have assigned random resolution levels for each block, varying from level zero (1×1) to five (32×32). The bottom row shows color-coded error images based on ΔE, a perceptually adapted color error in the CIE 1976 $L^*u^*v^*$ (CIELUV) [Poy97]. The Maximum Distance variant clearly reconstructs the signal with the smallest error. In this image the largest errors come from the image boundary where interpolation is not possible.

The implementation has been tested on two different graphics adapters, an NVIDIA 7800 GTX PCI-express and an ATI X800XT AGPx8, both with 256 MB DDR3 memory. The non-interpolating shader uses 20 instructions including diffuse lighting and the corresponding Maximum Distance shader uses 171 instructions. Of the instructions, there are 4 and 25 texture lookup instructions, respectively. These figures refer to the NVIDIA card. Based on the number of instructions we could anticipate that the interblock interpolation scheme would require 8 times longer per frame but the texture lookups lead to a slowdown by a factor of 10 to 15 on the NVIDIA card. Using the two pass method described in section 4 the ATI graphics card can render volumes with interblock interpolation with considerably less performance drop, it being only about 3 to 4 times slower, by exploiting the early Z-termination feature.

We have used three data sets for 3D rendering, these are a female cadaver, a head, and a heart data set. All of the data sets are captured with CT. Details can be found in table 1. We have precomputed the gradients using the 3D Sobel operator in a preprocessing step and stored all the MIPMAP-levels on disk. The data sets have been converted to 8-bits per voxel and the gradients are stored in four bytes using three bytes for gradient direction and one byte for gradient magnitude.

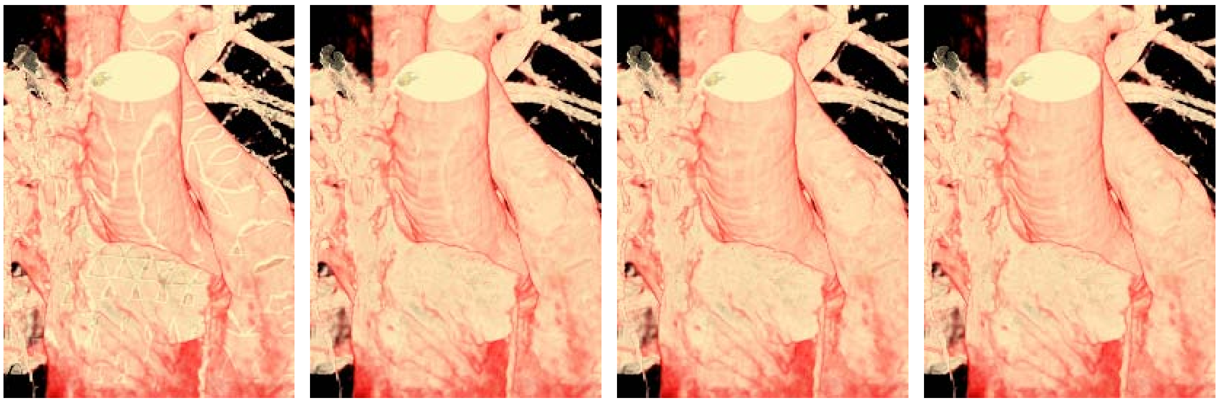
Table 1: *The data sets that have been used for the evaluation of interblock interpolation. All are converted to 8-bits precision. The heart data set is statically compressed without gradients and stored on disk.*

Data Set	**Dimensions**	**Size**	**Size on disk**
Female	512×512×2166	2.7 GB	3.1 GB
Skull	512×512×990	1.2 GB	1.5 GB
Heart	512×448×416	93 MB	2.9 MB

The heart data set has been statically compressed at 32:1 and stored on disk and it does not have gradients.

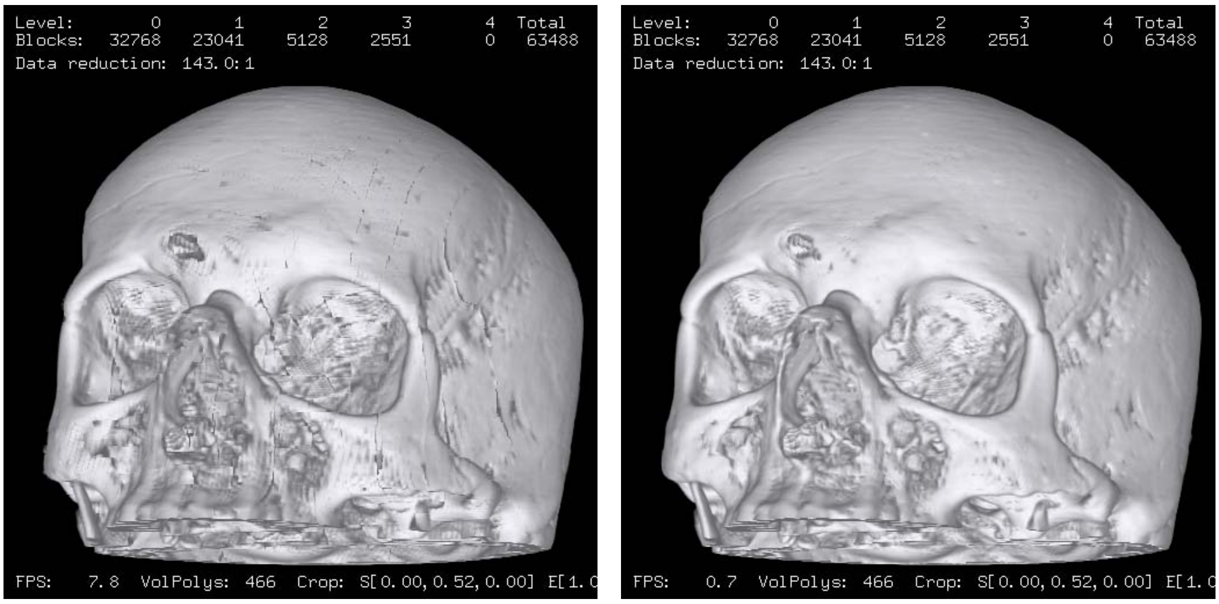
In figure 8 we show unlit renderings of the heart comparing all three variants and no interblock interpolation. The block boundary artefacts are clearly visible without interpolation and are reduced for the three variants, as expected from the 2D results above. It is hard to notice any visible artefacts from the Maximum Distance method.

Figure 10 shows renderings with a very thin iso-surface, the TF has been set to show the skin with some transparency and the bones in white. The blockiness is clearly visible without interpolation. Since all of the interpolation variants are equally expensive computationally we show only the Maximum Distance variant. The volume is rendered at a data reduction ratio of 35:1 in a window viewport of 1024×1024. Using 413 slicing planes the intrablock version renders at 4.3 frames per second and at 0.3 frames-per-second for interblock interpolation. The effect of the interpolation is less obvious in figure 9, which uses a highly opaque TF to show the bones. The degree of data reduction is significant, a ratio of 143:1. The intrablock frame-rate on the 7800-card was 7.8 and 0.7 for interpolation, for a 512×512 window with 466 slicing planes.



No interblock interpolation *Minimum Distance* *Boundary Split* *Maximum Distance*

Figure 8: *Comparison of the the presented block interpolations for a 32:1 data reduction of heart data set, original resolution 512×448×416. This rendering does not use any lighting and gradients are not present in the data set.*



```
Level:        0       1       2       3       4   Total
Blocks:   32768   23041    5128    2551       0   63488
Data reduction:  143.0:1

FPS:    7.8   VolPolys:  466   Crop:  S[0.00,0.52,0.00]  E[1.C
```

```
Level:        0       1       2       3       4   Total
Blocks:   32768   23041    5128    2551       0   63488
Data reduction:  143.0:1

FPS:    0.7   VolPolys:  466   Crop:  S[0.00,0.52,0.00]  E[1.C
```

No interblock interpolation *Maximum Distance*

Figure 9: *Comparison of interblock interpolation for an opaque lit surface. Original data set size 512×512×990 rendered at a ratio of 143:1. Block level distribution: 0, 2551, 5128, 23041, 32768 (highest to lowest resolution).*

6. Conclusions

In this paper we have introduced direct interblock interpolation for block-based level-of-detail schemes and demonstrated this technique on modern, commodity graphics hardware using fragment programs. The ability to directly and smoothly interpolate between blocks of arbitrary resolution levels removes the need for sample replication processing and does not require additional memory. The data reduction gained from the LOD scheme can thus be maintained down to the texture memory. We have used a transfer function based adaptive level-of-detail scheme [LLYM04] that works with a flat blocking structure and we have shown interactive, high quality renderings of data sets of the order of gigabytes.

Three interblock interpolation variants have been explored. The increase in rendering quality compared to nearest neighbor sampling is significant. Between the three variants the difference in performance is small. The Maximum Distance scheme tends to yield the highest subjective quality, but the desirable smoothing ability may also create exaggerated blurring into low resolution blocks when the resolution levels differ greatly, see figure 5. Due to the many texture

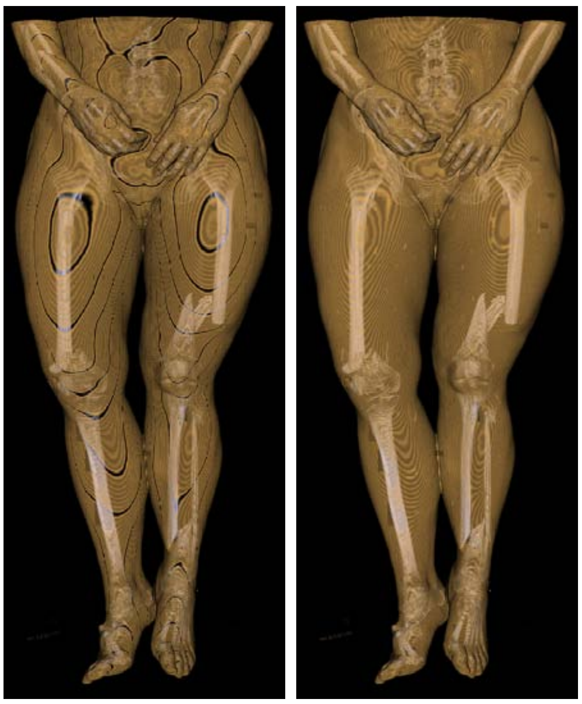


Figure 10: *Comparison of interblock interpolation for a thin iso-surface, skin, and bones on the cadaver. The block boundary artefacts without interpolation are clearly removed by our scheme. There are, however, sampling artefacts present in both images. Data reduction through LOD selection at 35:1. Original data set size 512×512×2166 fitted into a 256×256×256 texture.*

lookups and longer fragment program, the frame rates drop when using interblock interpolation. The early Z-termination feature on the ATI X800 minimizes the use of the more demanding interpolating shader and this two pass rendering then only runs three to four times slower than without interblock interpolation. The intrablock interpolation scheme is considerably faster than the interblock. A good combination of these two methods is to use intrablock during interactive use, interblock interpolation only being applied when the viewing parameters are not being changed.

Acknowledgements

This work has been funded by the the Swedish Research Council, grant 621-2001-2778 and 621-2003-6582, and the Swedish Foundation for Strategic Research, grant A3 02:116. The Center for Medical Image Science and Visualization (CMIV) and, in particular, Anders Persson is acknowledged for providing the medical data sets.

References

[BIP01] BAJAJ C., IHM I., PARK S.: 3D RGB image compression for interactive applications. *ACM Transactions on Graphics 20*, 1 (January 2001), 10–38.

[BNS01] BOADA I., NAVAZO I., SCOPIGNO R.: Multiresolution volume visualization with a texture-based octree. *The Visual Computer 17* (2001), 185–197.

[CCF94] CABRAL B., CAM N., FORAN J.: Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *VVS '94: Proceedings of the 1994 symposium on Volume visualization* (New York, NY, USA, 1994), ACM Press, pp. 91–98.

[GHJA05] GAO J., HUANG J., JOHNSON C. R., ATCHLEY S.: Distributed data management for large volume visualization. In *Proceedings IEEE Visualization 2005* (2005), IEEE, pp. 183–189.

[GHSK03] GAO J., HUANG J., SHEN H.-W., KOHL J. A.: Visibility culling using plenoptic opacity functions for large volume visualization. In *Proceedings IEEE Visualization 2003* (2003), IEEE, pp. 341–348.

[GS04] GUTHE S., STRASSER W.: Advanced techniques for high quality multiresolution volume rendering. In *Computers & Graphics* (2004), vol. 28, Elsevier Science, pp. 51–58.

[GWGS02] GUTHE S., WAND M., GONSER J., STRASSER W.: Interactive rendering of large volume data sets. In *Proceedings IEEE Visualization 2002* (2002), pp. 53–60.

[IP99] IHM I., PARK S.: Wavelet-based 3d compression scheme for interactive visualization of very large volume data. *Compute Graphics Forum 18*, 1 (1999), 3–15.

[Kau91] KAUFMAN A.: *Volume Visualization (Tutorial)*. IEEE Computer Society Press, 1991.

[KE02] KRAUS M., ERTL T.: Adaptive texture maps. In *Eurographics/SIGGRAPH Workshop on Graphics Hardware* (2002), pp. 7–15.

[LHJ99] LAMAR E. C., HAMANN B., JOY K. I.: Multiresolution techniques for interactive texture-based volume visualization. In *Proceedings IEEE Visualization 1999* (1999), pp. 355–362.

[LHJ03] LAMAR E. C., HAMANN B., JOY K. I.: Efficient error calculation for multiresolution texture-based volume visualization. In *Hierachical and Geometrical Methods in Scientific Visualization* (2003), Springer-Verlag, pp. 51–62.

[LLYM04] LJUNG P., LUNDSTRÖM C., YNNERMAN A., MUSETH K.: Transfer function based adaptive decompression for volume rendering of large medical data sets. In *Proceedings IEEE Volume Visualization and Graphics Symposium* (2004), pp. 25–32.

[NS01] NGUYEN K. G., SAUPE D.: Rapid high quality compression of volume data for visualization. *Computer Graphics Forum 20*, 3 (2001).

[Poy97] POYNTON C.: Frequently asked questions about color. http://www.poynton.com/PDFs/ColorFAQ.pdf, March 1997. Acquired January 2004.

[WWH*00] WEILER M., WESTERMANN R., HANSEN C., ZIMMERMAN K., ERTL T.: Level–of–detail volume rendering via 3d textures. In *Proceedings IEEE Volume Visualization and Graphics Symposium 2000* (2000), ACM Press, pp. 7–13.