Eurographics Symposium on Parallel Graphics and Visualization (2004), pp. 1–7
Dirk Bartz, Bruno Raffin and Han-Wei Shen (Editors)

Hierarchical Visualization and Compression of Large Volume Datasets Using GPU Clusters

Magnus Strengert[1], Marcelo Magallón[1], Daniel Weiskopf[1], Stefan Guthe[2], and Thomas Ertl[1]

[1]Institute of Visualization and Interactive Systems, University of Stuttgart, Germany
[2]WSI/GRIS, University of Tübingen, Germany

Abstract

We describe a system for the texture-based direct volume visualization of large data sets on a PC cluster equipped with GPUs. The data is partitioned into volume bricks in object space, and the intermediate images are combined to a final picture in a sort-last approach. Hierarchical wavelet compression is applied to increase the effective size of volumes that can be handled. An adaptive rendering mechanism takes into account the viewing parameters and properties of the data set to adjust the texture resolution and number of slices. We discuss the specific issues of this adaptive and hierarchical approach in the context of a distributed memory architecture and present solutions for these problems. Furthermore, our compositing scheme takes into account the footprints of volume bricks to minimize the costs for reading from framebuffer, network communication, and blending. A detailed performance analysis is provided and scaling characteristics of the parallel system are discussed. For example, our tests on a 16-node PC cluster show a rendering speed of 5 frames per second for a $2048 \times 1024 \times 1878$ *data set on a* 1024^2 *viewport.*

Categories and Subject Descriptors (according to ACM CCS): I.3.2 [Graphics Systems]: Distributed/network graphics I.3.3 [Picture/Image Generation]: Viewing algorithms

1. Introduction

Often volume rendering has to be applied to large data sets. For example, the increasing resolution of medical CT scanners leads to increasing sizes of scalar data sets, which can be in the range of gigabytes. Even more challenging is the visualization of time-dependent CFD simulation data that can comprise several gigabytes for a single time step and several hundred or thousand time steps. Parallel visualization can be used to address the issues of large data processing in two ways: Both the available memory and the visualization performance are scaled by the number of nodes in a cluster computer.

In this paper, we follow an approach that combines the "traditional" benefits of parallel computing with the high performance that is offered by GPU-based techniques. Our contributions are: First, hierarchical wavelet compression is adapted to the distributed-memory architecture of a cluster computer to increase the effective size of volumes that can be handled. Second, we present an adaptive, texture-based volume rendering approach for a PC cluster. Third, an advanced compositing scheme is described to take into account the footprints of volume bricks to minimize the costs for reading from framebuffer, network communication, and blending. Fourth, we document performance numbers for different combinations of parameters to clarify the performance and scaling characteristics. Results are discussed for both a mid-price system with 16 GPU/dual-CPU nodes and Myrinet, and a low-cost system with standard PCs connected by Gigabit Ethernet. We think that our findings are useful for working groups that have to visualize large-scale volume data.

2. Previous Work

This work builds up on that of Guthe et al. [GWGS02], who represent a volumetric data set as an octree of cubic blocks, to which a wavelet filter has been applied. By recursively applying these filters, a hierarchical multi-resolution structure is generated. Rendering is accomplished by computing a quality factor to select for which block the higher or lower resolution representations should be used. The decompres-

sion of the texture data happens then in software. Binotto et al [BCF03] recently presented a system that also uses a hierarchical representation, but is oriented towards the compression of time-dependent highly sparse highly temporal-coherent data sets. Their algorithm uses fragment programs in order to perform the decompression of the data sets in the graphics card, with a reported performance of over 4 fps for an image size of 512^2 pixels and a texture data set of 128^3 voxels.

Rosa et al [RLMO03] presented a system specifically developed for the visualization of time-varying volume data from thermal flow simulations for vehicle cabin and ventilation design. The system is based on the work of Lum et al [LMC02], which quantizes and lossy compresses the texture data by means of a discrete cosine transformation and stores the result as indexed textures. Textures stored in this way can be decoded in hardware by just changing the texture palette. The disadvantage of this method is that hardware support for paletted textures is being phased out by hardware vendors. This could be replaced by dependent texture look-ups, but these have a different behavior with respect to interpolation of the fetched data. In comparison to the other methods mention before, this approach achieves much lower compression ratios.

Stompel et al [SML*03] have recently presented a new compositing algorithm which takes advantage of the fact that in a configuration of n processing elements, there are on average $n^{\frac{1}{3}}$ partial images which are relevant for any given pixel of the final image. They report promising results using a 100 Mbps Ethernet network as the underlying communications fabric. The efficiency of the algorithm is naturally highly dependent on the viewing direction, but it reportedly compares favorably to the direct send and binary swap algorithms which are commonly used for this task.

3. Distributed Visualization

We use a *sort-last* [MCEF94] strategy to distribute the visualization process in a cluster environment. With increasing size of the input data set, this sorting scheme is favorable, since the input data becomes larger than the compositing data and hence a static partitioning in object space avoids communication regarding the scalar field during runtime. The basic structure of our implementation follows the approach by Magallon et al. [MHE01].

During a preprocessing step object-based partitioning is performed to split the input data set into multiple, identically sized sub-volumes, depending on the number of nodes in the cluster configuration. To overcome possible memory limitations in connection with large data sets, this step is executed using the same set of nodes as the following render process. Once all sub-volumes are created and transferred to their corresponding nodes, the render loop that can be split into two consecutive tasks is entered. The first task is to render each brick separately on its corresponding node. An intermediate image is generated by texture-based direct volume visualization. We employ screen-aligned slices through a 3D texture with back-to-front ordering [CCF94, CN93]. By adapting the model-view matrix for each node, it is assured that each sub-volume is rendered at its correct position in image space. Since the partitioning is performed in object space, the rendering process of different nodes can produce output that overlaps each other in image space. The second task blends the intermediate images and takes into account that multiple nodes can contribute to a single pixel in the final image. The distributed images are depth sorted and processed through a compositing step based on alpha blending. To this end, each node reads back its framebuffer, including the alpha channel, and sends it to other nodes. To take advantage of all nodes for the computational expensive alpha blending, direct send is usedas communication scheme [Neu93]. Each intermediate result is horizontally cut into a number of slices matching the total number of nodes. All these slices are sorted and transferred between the nodes in a way that each node receives all stripes of a specific area in the image space. Then each node computes an identically sized part of the final image.

The alpha blending of the intermediate images is completely performed on the CPU. Although the GPU was highly specialized for this task, the additional costs for loading all new stripes into texture memory and reading back the information after blending would lead to a lower overall performance. Instead, an optimized MMX [PW96] code is used to determine the result of the blend function for all four channels of one pixel in parallel. In order to implement blending using MMX operations it is necessary to express the operation

$$r = a + \frac{(1 - a_{\text{alpha}}) * b}{255}$$

in terms of bit-shifts operations, which can be done as:

$$\frac{x}{255} = \frac{x + 128 + \frac{x+128}{256}}{256}$$

This expression is correct for the range $0..255^2$ when compared with the floating point version rounded up and truncated to integer results. The actual implementation using MMX operations is given in Appendix A.

Without major changes this approach can also handle time-dependent scalar fields. During the bricking process a static partitioning scheme is used for all time steps, i.e., each sub-volume contains the complete temporal sequence for the corresponding part of the input volume. To synchronize all nodes the information regarding the current time step is communicated to the render nodes.

4. Accelerated Compositing Scheme

Three limiting factors for overall performance are: The process of reading back the results from the framebuffer, the data transfer between nodes, and the compositing step. In the following we address these issues by minimizing the amount of image data to be processed. The key observation is that the image footprint of a sub-volume usually covers only a fraction of the intermediate image. For the scaling behavior, it is important that the relative size of the footprint shrinks with increasing number of nodes. For simplicity, we determine an upper bound for the footprint by computing the axis-aligned bounding box of the projected sub-volume in image space. Since the time needed to read back a rectangular region from the framebuffer is nearly linearly dependent on the amount of data, reducing the area to be retrieved leads to a performance increase of this part of the rendering process. Similarly, the communication speed also benefits from the reduction of image data.

The compositing step is accelerated by avoiding unnecessary alpha blending operations for image regions outside the footprints. Similarly to SLIC [SML*03], a line-based compositing scheme is employed. For each line the span containing already blended data is tracked. With this information the new image data of the next compositing step can be split into two regions. The first part includes pixels that map into the region outside the marked span. These pixels need no further processing and can be copied into the resulting image. The remaining pixels map into an area where already other color information resides and an alpha blending operation has to be performed for each of those pixels. After one iteration the size of the span containing data needs to be updated and the next image stripe can be processed. In doing so only a minimal amount of blending operations must be carried out to obtain a correctly blended image.

5. Hierarchical Compression and Adaptive Rendering

Even with distributed rendering techniques the size of a data set can exceed the combined system memory of a cluster configuration and the already bricked data set is larger than one single node can handle. Another challenge is to further improve the rendering speed. We address the memory issue by using a hierarchical compression technique, and the performance issue by adaptive rendering.

5.1. Single-GPU Wavelet Compression

We adopt a single-GPU visualization approach that utilizes compression for large data sets [GWGS02]. The idea is to transform the input data set into a compressed hierarchical representation in a preprocessing step. With the help of wavelet transformations an octree structure is created. The input data set is split into cubes of size 15^3 voxels, which serve as starting point for the recursive preprocessing. Eight cubes sharing one corner are transformed at a time using linearly interpolating spline wavelets. The resulting low-pass filtered portion is a combined representation of the eight input cubes with half the resolution of the original data. The size of this portion is again 15^3 voxels. The wavelet coefficients representing the high frequencies replace the original data of the eight input blocks. After all cubes of the original data set are transformed, the next iteration starts using the newly created low-pass filtered cubes as input. The recursion stops as soon as the whole volume is represented through one single cube. This cube forms the root node of the hierarchical data structure and is the representation with the lowest quality. Except for the root node, all other nodes hold only high-pass filtered data, which are compressed through an arithmetic encoder [GS01]. While it is possible to increase the compression ratio by thresholding, we focus on lossless compression for best visualization results.

During rendering an adaptive decompression scheme depending on the viewing position and the data set itself is used. Starting at the root node of the hierarchical data structure, a priority queue determines which parts of the volume are decompressed next. Depending on the ratio between the resolution of a volume block and the actual display resolution, regions closer to the viewer are more likely decompressed than others. Additionally an error criterion describing the difference between two representations of varying quality is used to identify regions that can be rendered in low quality without noticeable artifacts. Having finished the quality classification all decompressed blocks are transferred to the graphics boards texture memory. Additionally a cache strategy is used to avoid the expensive decompression step for recently processed blocks and by tracking the already loaded textures unnecessary texture transfers are avoided.

5.2. Extension to Parallel Rendering

In a distributed visualization system, this approach leads to a problem concerning correct texture interpolation between sub-volumes rendered on different nodes. A typical solution is to create the sub-volumes with an overlap of one voxel. With multi-resolution rendering techniques it's necessary to know not only the border voxels of the original data set but also the data value at the border for all other used quality levels [WWH*00]. This information can be determined in the preprocessing step. After creating the sub-volumes and constructing the hierarchical data structure, each node transfers the information in all used quality levels to its appropriate neighbors border. But even with this information available on each node a correct texture interpolation cannot be generated easily. The remaining problem is to determine the quality level a neighboring node uses for rendering in order to choose the border information for the same quality level. Since the network communication between the nodes is rather slow, requesting this information from the neighboring node is not suitable. Another approach is to compute the

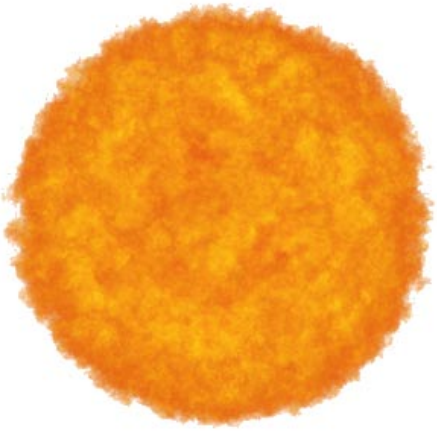

Figure 1: Radial distance volume combined with noise using a high frequency transferfunction.

Figure 2: Rendering result of upper 4 nodes showing anatomic cryosections through Visible Human Project male data set. The whole body is rendered with a total of 16 nodes.

quality classification on each node for an expanded area, but this is impractical, because of the classifications dependency on the volume data.

Instead, we propose an approximate solution that presumes that there are no changes in quality classification at the border of the sub-volumes. With this approach errors only occur if different qualities are used on each side of a sub-volume border. Due to the similar position of adjacent parts of the sub-volumes it is however likely that both regions are classified with the same quality.

For a correct solution of the interpolation problem, we propose another approach that separates the computation of the quality classification and the rendering process. In each frame an adaptive classification is determined, but the associated rendering is delayed for one frame. In doing so the information regarding the used quality levels can be transferred to the neighboring nodes at the time of distributing the intermediate results during the compositing step. Since at this time communication between all nodes must be performed anyway, the additional data can be appended to the image data. With transferred data the rendering process can produce a properly interpolated visualization during the next frame. The downside is that the latency between user interactions and the systems reaction is increased by one frame. To avoid this a hybrid technique, that exploits both described approaches, is possible. In case of user interaction the approximate solution is used to generate an image without increased latency times. In the other case the transferred quality classification is still valid and can be used for rendering the next frame until another user interaction occurs. Thus a fast user interaction is combined with a correct sub-volume interpolation in the static case.

6. Implementation and Results

Our implementation of the presented approach is based on C++ and OpenGL. Volume rendering adopts post-shading realized either through NVIDIAs register combiners or alternatively through an ARB fragment program depending on the available hardware support. MPI is used for all communication between nodes.

Two different cluster environments were used for developing and evaluation. The first one is a 16-node PC cluster. Each of these nodes runs a dual-CPU configuration with two AMD 1.6 GHz Athlon CPUs, 2 GB of system memory, and NVIDIA GeForce 4 Ti 4600 (128MB) graphics boards. The interconnecting network is a Myrinet 1.28GBit/s switched LAN providing low latency times. Linux is used as operating system, the *SCore* MPI implementation drives the communication [PC].

The second environment is built up by standard PCs using a Gigabit Ethernet interconnection with a maximum number of eight nodes. Each node has an Intel Pentium4 2.8GHz CPU and 4GB system memory. The installed graphics boards are a mixture of NVIDIA GeForce 4 Ti 4200 and GeForce 4 Ti 4600 both providing 128MB of video memory. Running Linux, the MPI implementation *LAM/MPI* is used for node management and communication [LAM].

We use three different large scaled data sets to evaluate the performance of the implemented visualization system. If not

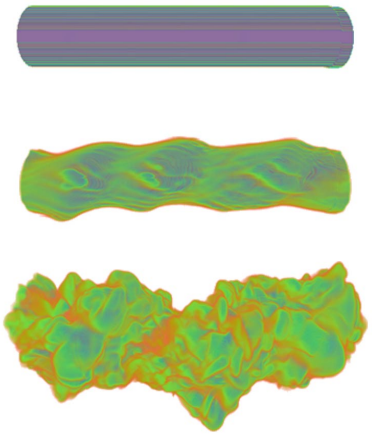

Figure 3: Visualization of the time-dependent CFD simmulation. From top to botton timesteps 0, 45, 89 are shown.

stated otherwise all measurements were performed on the cluster interconnected though Myrinet. The first data set is a generated scalar field showing a radial distance volume that is additionally combined with perlin noise (Figure 1). While arbitrary sized data sets can be produced, the typical characteristics of non-generated data sets is ensured by adding the distortion. For our testing purposes a 1024^3 sized volume is used. The second data set is derived from the anatomical RGB cryosections of the visible human male data set [The]. The slices are reduced to 8 bit per voxel and cropped to exclude external data like grey scale cards and fidual markers. The obtained data set has a resolution of $2048 \times 1024 \times 1878$ voxels (Figure 2). The third data set is a time-dependent CFD simulation of a flow field with increasing turbulence. The sequence is made up by a total of 89 time bins each sized 256^3 (Figure 3).

The Visible Human male data set can be visualized on a 1024^2 viewport using 16 nodes with 5 frames per second. The quality classification was set to use the original resolution for most regions. Due to the uniform characteristic of the surroundings, these areas were displayed in a lower resolution without any noticeable disadvantages. With a half sized viewport and the same settings the obtained framerate is increased to 8 frames per second.

In order to show the scaling behavior of the visualization system configurations of 2 up to 16 render nodes are measured. The used data set for all these tests is the gigacube containing the distorted radial distance volume. The results are shown in figure 4. For a 16 node configuration the data set can be rendered in 174 ms, which corresponds to a refresh rate of 5.7 Hz.

For the time-dependent data set figure 5 shows the results

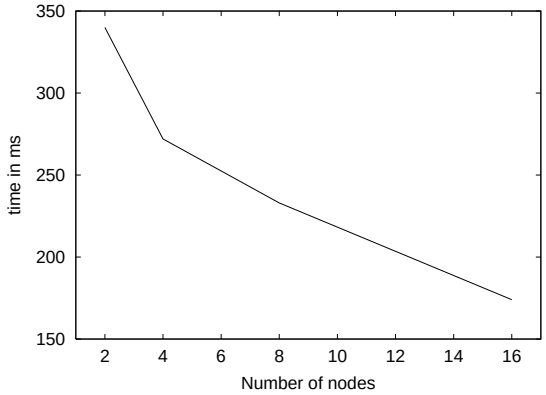

Figure 4: Scalability of the visualization system with the number of rendering nodes.

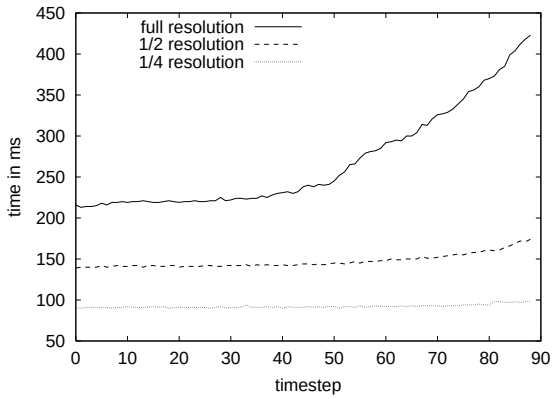

Figure 5: Performance rendering time-dependent data set.

for rendering each timestep in a row. The test was performed using three different quality levels. In case of the original quality the required time clearly increases towards the end of the sequence. The reason behind this is found in characteristic of the data set, which gets more and more turbulent over time leading to a higher amount of blocks that have to be decompressed. Furthermore with a progress in time the cache becomes invalid and all blocks have to be decompressed starting at the root node. Thats why the performance is rather slow for time-dependent data sets compared to the static ones.

Using the second cluster environment for rendering the distance volume with 8 nodes only 2 frames per second are achieved. Due to the similar configuration of each node this gap is solely caused by the Gigabit Ethernet in comparison to the Myrinet.

7. Conclusion and Future Work

We have presented a distributed rendering system for texture-based direct volume visualization. By adapting a hierarchical wavelet compression technique to a cluster envi-

ronment the effective size of volume data that can be handled is further improved. The adaptive decompression and rendering scheme allows a reduction of rendering costs depending on the viewing positing and the characteristics of the data set without leading to noticeable artifacts in the final image. The arising problem of texture interpolation at brick borders in connection with multi-resolution rendering is addressed and different solutions are provided. Parts of the rendering process crucial to the systems performance benefit from the applied reduction of the processed region in image space, especially with increasing numbers of rendering nodes.

The achieved performance is still primarily restricted by the capabilities of the used interconnection between the rendering nodes and the computation of blending operations during the compositing step. With viewports sized 1024^2 this upper bound is approximately 11 frames per second for our cluster configuration. To increase this upper limit the exact calculation of the footprints instead of using a bounding box could be helpful avoid remaining unnecessary blending operations and further reduce communication costs. In case of time-dependent data sets the performance is additionally bound by the decompression step, because the performed caching of decompressed blocks cannot be used in this context.

As part of our future work we would like to implement and test the SLIC algorithm from Stompel et al on Myrinet and 4x InfiniBand networks.

Appendix A: Blending using MMX operations

The following code performs the operation $r = a + ((1 - a_{\text{alpha}}) * b)/255$ using MMX instructions. It uses the GNU Compiler Collection's (GCC) "extended assembly" notation, which means the operands are in AT&T syntax (source operand on the left side and destination operand on the right). `%0`, `%1` and `%2` are *r*, *a* and *b* respectively.

```
pxor          %mm2, %mm2

/* copy 128 to all words in mm4 */
mov           $128, %eax
movd          %eax, %mm4
pshufw    $0, %mm4, %mm4

/* copy a to mm0           */
movd          (%1), %mm0

/* copy b to mm3           */
movd          (%2), %mm3
/* 16-bit expand b         */
punpcklbw     %mm2, %mm3

/* fill mm1 with 1's       */
pcmpeqb       %mm1, %mm1
/* 1 - aalpha              */
pxor          %mm0, %mm1
/* 16-bit expand 1-aa      */
punpcklbw     %mm2, %mm1
/* copy 1-aa to all words */
pshufw    $0, %mm1, %mm1

/* x = (1-aalpha)*b        */
pmullw        %mm1, %mm3
/* x += 128                */
paddusw       %mm4, %mm3
/* y = x                   */
movq          %mm3, %mm1
/* y /= 256                */
psrlw           $8, %mm1
/* y = y + x               */
paddusw       %mm3, %mm1
/* y /= 256                */
psrlw           $8, %mm1

/* pack result             */
packuswb      %mm1, %mm1

/* add a and (1-aalpha)b  */
paddusb       %mm1, %mm0
/* copy result to memory  */
movd          %mm0,  (%0)
```

References

[BCF03] BINOTTO A. P. D., COMBA J. L. D., FREITAS C. M. D.: Real-time volume rendering of time-varying data using a fragment-shader compression approach. In *IEEE Symposium on Parallel and Large-Data Visualization and Graphics* (Oct. 2003), p. 10. 2

[CCF94] CABRAL B., CAM N., FORAN J.: Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *Proceedings of the 1994 Symposium on Volume Visualization* (1994), pp. 91–98. 2

[CN93] CULLIP T., NEUMANN U.: *Accelerating volume reconstruction with 3D texture mapping hardware*. Tech. Rep. TR93-027, Department of Computer Science at the University of North Carolina, Chapel Hill, 1993. 2

[GS01] GUTHE S., STRASSER W.: Real-time decompression and visualization of animated volume data. In *Proceedings of the Conference on Visualization '01* (2001), pp. 349–356. 3

[GWGS02] GUTHE S., WAND M., GONSER J., STRASSER W.: Interactive rendering of large volume data sets. In *Proceedings of the Conference on Visualization '02* (2002), pp. 53–60. 1, 3

[LAM] LAM/MPI PARALLEL COMPUTING: Web page: http://www.lam-mpi.org/. 4

[LMC02] LUM E. B., MA K.-L., CLYNE J.: A hardware-assisted scalable solution for interactive volume rendering of time-varying data. *IEEE Transactions on Visualization and Computer Graphics 8*, 3 (2002), 286–301. 2

[MCEF94] MOLNAR S., COX M., ELLSWORTH D., FUCHS H.: A sorting classification of parallel rendering. *IEEE Comput. Graph. Appl. 14*, 4 (1994), 23–32. 2

[MHE01] MAGALLÓN M., HOPF M., ERTL T.: Parallel volume rendering using PC graphics hardware. In *Pacific Graphics* (2001). 2

[Neu93] NEUMANN U.: Parallel volume-rendering algorithm performance on mesh-connected multicomputers. In *IEEE/SIGGRAPH Parallel Rendering Symposium* (1993). 2

[PC] PC CLUSTER CONSORTIUM: Web page: http://www.pccluster.org/. 4

[PW96] PELEG A., WEISER U.: MMX technology extension to the Intel architecture. *IEEE Micro 16*, 4 (1996), 42–50. 2

[RLMO03] ROSA G. G., LUM E. B., MA K.-L., ONO K.: An interactive volume visualization system for transient flow analysis. In *Proceedings of the 2003 Eurographics/IEEE TVCG Workshop on Volume graphics* (2003), pp. 137–144. 2

[SML*03] STOMPEL A., MA K.-L., LUM E. B., AHRENS J. P., PATCHETT J.: SLIC: scheduled linear image compositing for parallel volume rendering. In *IEEE Symposium on Parallel and Large-Data Visualization and Graphics* (2003), pp. 33–40. 2, 3

[The] THE NATIONAL LIBRARY OF MEDICINE'S VISIBLE HUMAN PROJECT: Web page: www.nlm.nih.gov/research/visible/. 5

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