

DISTRIBUTED VOLUME RAYCASTING ON MASPAR MP-I IN AN INTEGRATED ENVIRONMENT

Matthias Hopf Roberto Grosso Thomas Ertl

Universität Erlangen, IMMD IX, Graphische Datenverarbeitung

Am Weichselgarten 9, D-91058 Erlangen, Germany

Email: {mshopf,grosso,ertl}@informatik.uni-erlangen.de

KEYWORDS

Graphics
Distributed processors
Multiprocessors
Interactive

ABSTRACT

Volume rendering is an important tool in scientific visualization. However, due to the fact that it is computationally very expensive, applications on standard workstations are not capable of providing interactive frame rates. These are essential to understand the three dimensional nature of the data. Standard visualization software packages are either specialized for volume rendering or general-purpose solutions with inefficient and/or incomplete volume rendering techniques. In this paper we present a visualization environment based on Iris Explorer of SGI/NAG, which integrates a high performance massively parallel computer such as the Maspar MP-I for efficient and flexible volume rendering, including different techniques such as X-ray, MIP and emission-absorption. The volume rendering algorithm is based on a shear-warp factorization of the viewing matrix, which we proposed for an efficient use of the XNet topology of the MP-I. The algorithm resulting from the proposed factorization fits the network properties of the target machine very well and optimal scalability can be reached. Furthermore, the Explorer interface allows a flexible and intuitive handling of the different parameters and visualization techniques of the volume renderer.

1 INTRODUCTION

The problem of visualizing three-dimensional volumetric data is one of the major areas of research in scientific visualization (Rosenblum 1994). Although computational much more expensive than the indirect visual-

ization of scalar fields by means of isosurfaces, direct volume rendering has attracted much attention because of its ability to present the full volumetric structure of the dataset through the use of transparency and various mapping techniques.

Volume rendering has in principle a complexity of $O(n^3)$. Thus, to get interactive frame rates, which is important to understand the three dimensional nature of the data, extremely high computational power and very efficient algorithms are required. To achieve this goal many different approaches can be found in the literature. One approach is to use special purpose hardware, for example 3D-texture hardware as described in (Cabral, Cam, and Foran 1994). A different approach is to implement specialized algorithms for massively parallel computers (Challinger 1993; Hsu 1993; Ma, Painter, Hansen, and Krogh 1994; Corrie and Mackeras 1993; Vezina, Fletcher, and Robertson 1992; Wittenbrink and Somani 1993). Another very interesting approach is based on data and image coherence (Lacroute 1995; Lacroute and Levoy 1994), achieving nearly interactive rates even on standard workstations.

In this work we are concerned with volume rendering algorithms which are suitable for massively parallel SIMD architectures. Our target machine is a Maspar MP-I, which is characterized by its communication structure. Due to this fact, the class of shear-warp algorithms (Krüger and Schröder 1994) fits best for this kind of architecture.

A typical working session in scientific visualization requires the use of many different visualization algorithms. Thus, a major goal of our work was to integrate the massively parallel computer into a visualization application environment. We decided to use the Iris Explorer package of SGI/NAG.

In Section 2 all information about the target architecture Maspar MP-I relevant for the implementation phase is presented. Section 3 discusses the theoretical aspects of the algorithm used in this work. The imple-

mentation itself is described in Section 4 and Section 5. Finally, we conclude this paper with an analysis of the achieved performance in Section 6.

2 SIMD MASPAR MP-I

One major aim of this work is to implement the shear-warp algorithm on a massively parallel SIMD computer like the Maspar MP-I. This machine consists of a Unix frontend, an array control unit (ACU) for singular expressions, the processor array consisting of 1024 to 16384 processor elements (PEs) and two separate processor networks, the so called XNet for local regular communication and a less efficient global router for irregular communication.

The PEs are ordered in a two-dimensional array and all of them can perform only exactly the same action at a time, but they may all work on different data. For this data parallelism each PE has its own 16 KByte or 64 KByte local memory which can be addressed very fast in parallel. PEs' access to ACU memory is very slow and should be avoided in parallel instructions.

The MP-I is programmed in MPL (Maspar 1991) which is in fact C with parallel extensions. The main extension is the introduction of parallel data types (called *plural*), including plural pointers. Using plural `if` and `while` statements the processor working set can be altered. As only one statement can be executed at a time, both cases of an `if/else` statement have to be executed whenever the `if` expression is true for some processors and false for others. Therefore, no parallelization by case division is possible.

By using the shear-warp factorization of the view all irregular communication needed for accessing the partitioned volume can be reduced to regular communication utilizing the XNet on the Maspar for maximum efficiency. This network connects all PEs with their neighbors in eight directions including the diagonal ones with toroidal wraparound. Consequently a volume data partitioning has to be selected which totally exploits the available communication hardware.

3 SHEAR-WARP ALGORITHM

In this work we are interested in a factorization allowing an optimal use of the network topology of our target architecture. The main goal is to achieve regular communication patterns between neighboring processors.

The main idea behind *multipass transformations* is to factorize the viewing matrix into a number of matrix operations, which can be interpreted as simple transformations, each in contrast to the original complex viewing transformation. The most general form of these transformations that can be applied to volumes is given in (Hanrahan 1990).

Following a similar strategy as proposed in (Lacroute 1995), we assume that the volume is discretized in a uniform grid. We search for a factorization of the viewing matrix which consists of a *shear* perpendicular to the projection axis and a final *warp* to obtain the corresponding viewing transformation. In order to obtain the factorization we further assume, that the z-axis is the major projection axis. Otherwise, the volume can be transposed using a permutation. We assume for the viewing matrix \mathbf{V} the decomposition

$$\begin{aligned} \mathbf{V} &= \mathbf{P} \cdot \mathbf{R} \\ &= \mathbf{P} \cdot \mathbf{W} \cdot \mathbf{S} \end{aligned} \quad (1)$$

where \mathbf{P} is the parallel projection matrix along the z-axis, \mathbf{R} is a general rotation matrix

$$\mathbf{R} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$$

and \mathbf{W} and \mathbf{S} are the matrices corresponding to the warp and shear transformations, respectively:

$$\mathbf{W} = \begin{pmatrix} w_{11} & w_{12} & 0 \\ w_{21} & w_{22} & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad \mathbf{S} = \begin{pmatrix} 1 & 0 & s_x \\ 0 & 1 & s_y \\ 0 & 0 & 1 \end{pmatrix}$$

The solution to the matrix equation (1) can be easily computed. After some simple algebraic operations one obtains:

$$\begin{aligned} w_{11} &= a, \\ w_{12} &= b, \\ w_{21} &= d, \\ w_{22} &= e, \\ s_x &= \frac{fb - ec}{ae - bd}, \\ s_y &= \frac{af - dc}{ae - bd} \end{aligned} \quad (2)$$

The denominator in the expression for s_x and s_y may become zero. To interpret this fact geometrically, one can use the Euler angles α , β and γ to represent the

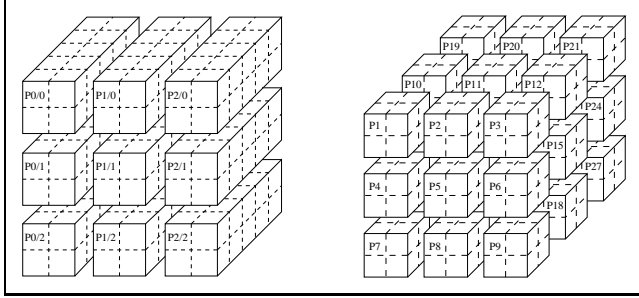


Figure 1: Volume representation by columns or blocks

coefficients of the rotation matrix \mathbf{R} . The denominator in the expression for s_x and s_y can now be written as:

$$ae - bd = \cos\beta$$

which means that rotations around the second rotation axis should not be $\frac{\pi}{2}$, which actually is not a restriction. The rendering algorithm based on this factorization of the viewing matrix works as follows:

- computation of the shear transformation matrix from the viewing matrix coefficients,
- ray integration from front to back parallel to the processor array to obtain an intermediate image. Here we have to shift the rays between processors according to the shear values of \mathbf{S} as described in Section 4,
- warping of the intermediate image to the final image using the matrix \mathbf{W} .

Please note that the volume data is not moved at all during the rendering. Only the image is shifted between neighbor processors as it is shown in Section 4. This algorithm benefits from all the advantages of the XNet network topology of the parallel computer, because during the shifting of the image only local communication between neighbor processors is needed. The final warp transformation is only a two dimensional operation and therefore it needs much less computation time.

4 IMPLEMENTATION

There are two major ways how to distribute large volumes on several processors as shown in Figure 1. Because of the implemented shear-warp algorithm, the use of a column based distribution requires a volume transposition every time the viewing angle exceeds $\frac{\pi}{4}$. For

greater angles the shift values exceed one, which would complicate programming, and memory requirements for the intermediate image can exceed every given finite limit. On the other hand communication costs will go down remarkably with this model as the XNet network on the MP-1 can be used during the shear step, because its topology matches exactly the needed communication directions.

Using a SIMD machine for the implementation makes it impossible to speed up rendering times by early ray termination or volume run length encoding. These techniques were used for MIMD approaches by Lacroute, Levoy, Amin and others in (Lacroute 1995; Lacroute and Levoy 1994), (Amin, Grama, and Singh 1995). For volume data with low compression, memory requirements can be much higher, as run length encoding information has to be precomputed and stored for all three coordinate axes, and this for every voxel cell. In these cases the performance of the mentioned algorithms suffers whenever the transfer functions are continuously changed, which is the case in a typical working session. Data-parallel approaches like ours are of course independent of the volume data values.

Before the integration process can be started, shearing and final warping values are computed and the pixel array is cleared. The pixel array, which will contain the final picture, is implemented as a two-dimensional distributed array. Every PE has four parts of this array, one major working area and three shift-areas. The total size of every part equals the (eventually transposed) volume dimensions in x - and y -directions. For the lookup and shading process a temporary array is allocated as well as for the final warping step.

In the integration process, the volume is resampled for every ray belonging to a pixel in the working area of the pixel array. The resampled value is transformed into a RGBA color used for integration. After all pixels in the working area have been processed, the volume has to be shifted according to the shear values.

However, shifting the volume has two disadvantages: First, resampling with the use of bilinear interpolation would destroy the volume data and accumulate errors while going through all slices of the volume. Second, shifting the volume has a total cost of $O(n^4)$ since all volume elements are shifted in average at least $\frac{n}{4}$ times in an intelligent implementation. So we decided to shift the coordinate system instead of the volume, like Schröder, Krüger and Stoll did in (Schröder and Stoll 1992; Schröder and Krüger 1993; Krüger and Schröder 1994). This implies a shift of the pixel array in the opposite direction, which yields a total cost of $O(n^3)$. We

also do not shift anything as long as the difference between the current and the actual position does not exceed one. The remainder is instead used during volume data lookup for bilinear interpolation. No communication occurs in that step because all column edges are duplicated on the neighbors' PEs.

Since we do not move the volume data at all, voxel neighborhood is obtained and therefore no normal vectors have to be precomputed and stored for shading purposes. So far we have only implemented a simple emission absorption model which does not need normals at all; for more sophisticated models see (Levoy 1988).

Almost all parallel volume raycasting implementations we have encountered so far seem to use only one fixed implementation for voxel interpolation, shading and integration. However, it would be nice to be able to change those implementations on the fly. For an average user it is much easier to view a volume with different integration methods by changing a simple rendering parameter than by restarting the volume renderer and trying to get the same view as before. For this we encapsulated the algorithms into so-called *handlers*. These handlers can be seen as instances of a handler class which provides several methods for initialization, invocation and termination. Thus they can all be activated and deactivated on the fly. At the moment, three different integration handlers (X-ray, MIP, emission-absorption), several lookup and warping handlers with and without bilinear interpolation, and several color lookup handlers are implemented. Two examples for different integration handlers are shown in Figure 4 with activated X-ray and MIP handlers.

All handler specific parameters are also treated in a unique way. Thus it is possible to implement a new handler needing new parameters and the user can alter these without recompiling the user interface module. Introducing abstraction always slows down computation processes. However, in this case the implied overhead is very small compared to rendering times as shown in Section 6.

5 A DISTRIBUTED VOLUME RENDERING ENVIRONMENT

In most cases users have no direct access to massively parallel computers, so a user interface running on workstations is required. Direct volume rendering is also only one step in a scientific visualization process and thus we wanted to integrate our volume renderer into the Iris Explorer visualization packet. We have developed two

explorer modules with which the user can transfer volume and picture data as well as change programming options and rendering parameters.

Figure 2 shows a typical view of the map editor of the Iris Explorer visualization packet. The visualization process is divided into several modules (SGI 1993) written in C or C++ which perform one specific action each and are connected by streams. The map shown above contains a reader module, an image displayer, a colormap editor and two self-developed modules described below.

The communication module *RaycastMaspar* is used to start the main program on a remote parallel computer and to build up a socket connection. It sends volume data and parameter options to the main program and receives the rendered image for viewing. It automatically detects any data changes on its input streams and starts new rendering steps as needed.

With the *RayOptions* module all handlers can be activated or deactivated while the main program is running. Every handler can have its own set of parameters which can be altered by the user. The options module receives the list of known handlers and their parameters from the main program via the communication module on program startup. In this way the module does not have to be recompiled when new handlers or parameters are added to the renderer. Since a user can activate handlers and alter parameters on the fly, he or she may easily take a look at several aspects of the same volume, for example by selecting the X-ray or the MIP integration handler. Without this approach several additional tools and render modules would be necessary.

Using modules to integrate our raycaster in the Explorer visualization package we are able to combine all the powerful visualization techniques of Explorer modules with a very efficient and complete volume rendering tool.

6 RESULTS AND CONCLUSION

The MP-I we used is a relatively old massively parallel computer and not much faster than a high performance workstation today. Faster parallel computers used for implementations like (Lacroute 1995) are now available. But the implemented algorithm will scale well with better processors like in the MP-II and we plan to port it to the CM-5 which is also available in the computer science department. Thus the most interesting timing results are speedup scaling values and handler calling

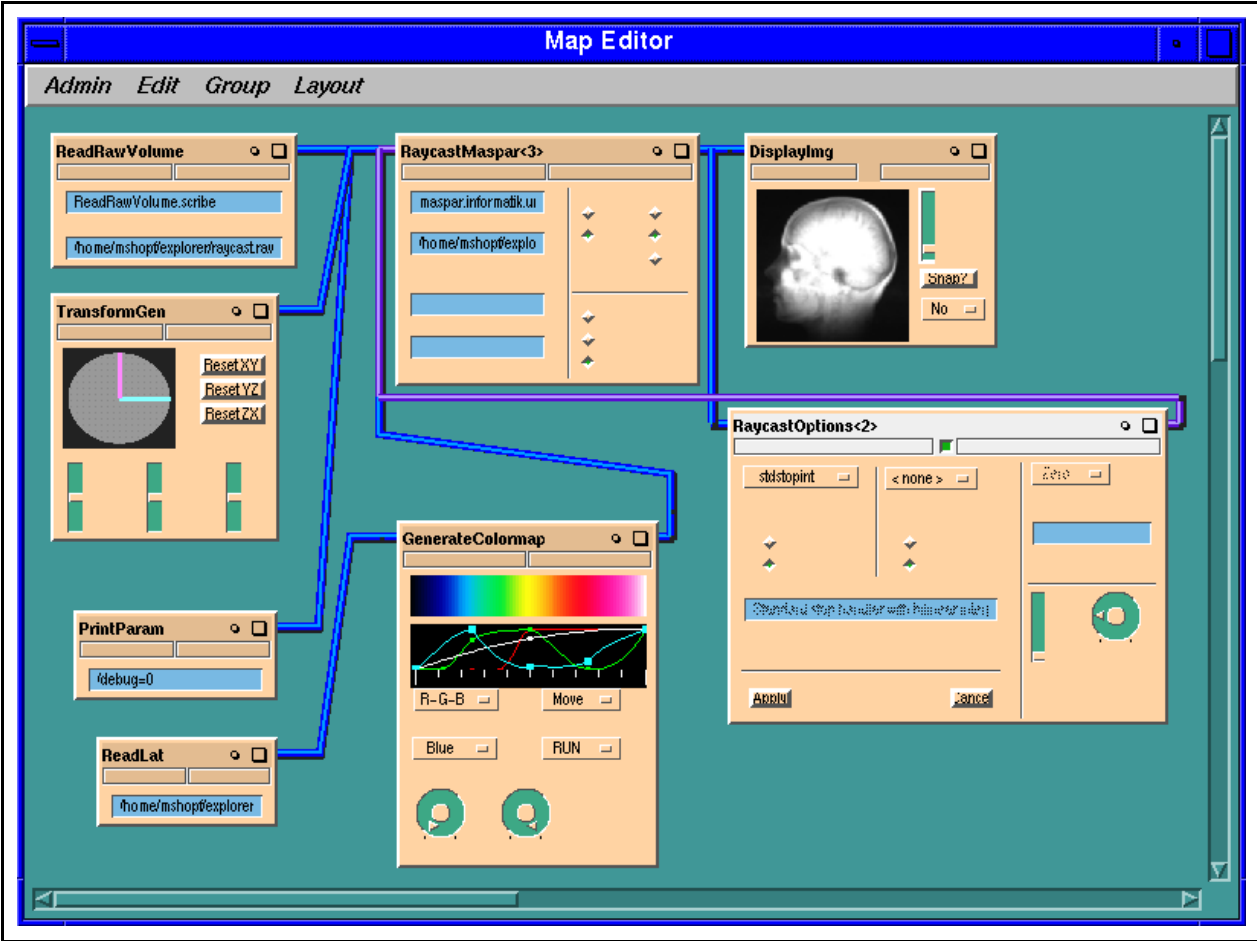


Figure 2: The *Map Editor* of Iris Explorer

overhead. Of course we can render a typical data set in interactive rates.

Execution times vary almost by a factor of two between the two extreme positions, the fastest being the frontal view (shear values are zero) and the slowest being a diagonal view (shear values are ± 1), both shown in Figure 3. The diagonal position could be rendered approximately 20 to 30% faster by using a plural `memmove` `libmpl` function, which is not yet implemented on the available system.

The times in table 1 were measured for two different handler configurations, one using lookup values and one using bilinear interpolation for every computation step. We tested two different processor arrays and varying volume sizes. The theoretical speedup scaling limit for the used PE array sizes is 16. The volume of size $256 \times 256 \times 109$ is a medical data set shown in Figure 5. The scaling factors are obviously very good, with

79% of the theoretical speedup limit on doubling the PE number available even in the worst examined case and 95% in the best case.

The images resulting from the pure lookup handlers using nearest neighbor interpolation were less informative and less appealing than those rendered with bilinear interpolation. However, the lookup handlers are faster and the difference is not as disturbing as mentioned in the literature. Thus, nearest neighbor interpolation should be sufficient for preprocessing, and bilinear handlers can be used for the final image.

Please note that our implementation sends 256^2 rays through a volume of size 256^3 . It thus renders pictures of size 256^2 scaled up to 362^2 in frontal view (best case for timing results) and pictures of size 512^2 in diagonal view (worst case). The algorithm is implemented not to scale down images in the final scaling step in order to preserve as much information as possible.

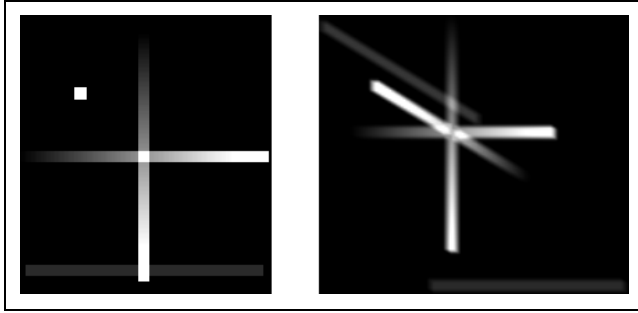


Figure 3: The most extreme views of a testing data set

As we divided the main program into several handlers it was interesting to measure processing times for individual handlers as shown in table 2. We also measured the overhead needed for handler calling. The total overhead varied between 5 and 50 ms depending on volume depth. This is not much though the MP-1 is quite slow when executing singular expressions.

ACKNOWLEDGMENTS

We want to thank the chair of computer architecture and performance evaluation of the University Erlangen-Nürnberg for providing us their Maspar MP-1. Timing measurements were made possible by the University of Stuttgart providing us an account on their Maspar MP-1 with 16384 processors.

REFERENCES

- Amin, M. B., Grama, A., and Singh, V. 1995. Fast volume rendering using an efficient, scalable parallel formulation of the shear-warp algorithm. In *Proceedings of the 1995 Symposium on Parallel Rendering*, pages 7–14. ACM SIGGRAPH.
- Cabral, B., Cam, N., and Foran, J. 1994. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In A. Kaufman and W. Krueger, editors, *Proceedings of the 1994 Symposium on Volume Visualization*, pages 91–98. ACM SIGGRAPH.
- Challinger, J. 1993. Scalable parallel volume raycasting for nonrectilinear computational grids. In T. Crockett, C. Hansen, and S. Whitman, editors, *Proceedings of the 1993 Symposium on Parallel Rendering*, pages 81–88. ACM SIGGRAPH.
- Corrie, B. and Mackerras, P. 1993. Parallel volume rendering and data coherence. In T. Crockett, C. Hansen, and S. Whitman, editors, *Proceedings of the 1993 Symposium on Parallel Rendering*, pages 23–26. ACM SIGGRAPH.
- Hanrahan, P. 1990. Three-pass affine transforms for volume rendering. *Computer Graphics*, 24(5), 71–78.
- Hege, H. C., Höllerer, T., and Stalling, D. 1993. Volume rendering. Technical Report TR 93–7, Konrad-Zuse-Zentrum für Informationstechnik, Berlin.
- Hsu, W. M. 1993. Segmented ray casting for data parallel volume rendering. In T. Crockett, C. Hansen, and S. Whitman, editors, *Proceedings of the 1993 Symposium on Parallel Rendering*, pages 6–14. ACM SIGGRAPH.
- Krüger, W. and Schröder, P. 1994. Data parallel volume rendering. In *Proceedings of the 1994 Scientific Visualization*, pages 37–52. Academic Press Ltd.
- Lacroute, P. 1995. Real-time volume rendering on shared memory multiprocessors using the shear-warp factorization. In *Proceedings of the 1995 Symposium on Parallel Rendering*, pages 15–22. ACM SIGGRAPH.
- Lacroute, P. and Levoy, M. 1994. Fast volume rendering using a shear-warp factorization of the viewing transformation. In A. Glassner, editor, *Proceedings of SIGGRAPH 1994*, pages 451–458. ACM SIGGRAPH.
- Levoy, M. 1988. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3), 29–37.
- Ma, K.-L., Painter, J. S., Hansen, C. D., and Krogh, M. F. 1994. Parallel volume rendering using binary-swap compositing. *IEEE Computer Graphics and Applications*, 14(5), 59–68.
- Maspar 1991. *MPL Reference Manual*. Maspar Computer Corporation, Sunnyvale, California 94086.
- Rosenblum, L. J. 1994. Research Issues in Scientific Visualization. *IEEE Computer Graphics and Applications*, 14(2), 61–85.
- Schröder, P. and Krüger, W. 1993. Data parallel volume-rendering algorithms for interactive visualization. In *The Visual Computer*, pages 405–416. Springer-Verlag.
- Schröder, P. and Stoll, G. 1992. Data parallel volume rendering as line drawing. *1992 Workshop on Volume Visualization*, pages 25–32.
- SGI 1993. *IRIS Explorer Module Writer's Guide*. Silicon Graphics Inc., Mountain View, California.
- Vezina, G., Fletcher, P. A., and Robertson, P. K. 1992. Volume rendering on the maspar MP-1. *1992 Workshop on Volume Visualization*, pages 3–8.
- Wittenbrink, C. M. and Somani, A. K. 1993. Permutation warping for data parallel volume rendering. In T. Crockett, C. Hansen, and S. Whitman, editors, *Proceedings of the 1993 Symposium on Parallel Rendering*, pages 57–60. ACM SIGGRAPH.

Configuration	Lookup	Lookup	Bilinear	Bilinear
Number of PEs	1024	16384	1024	16384
$128 \times 128 \times 54$	435–1016	82–171	1000–1519	126–222
128^3	977–2368	148–366	1963–3296	213–424
$256 \times 256 \times 109$	—	285–718	—	523–953
256^3	7147–15830 *	606–1626	13839–22366 *	1033–2054
Scaling factor for 128^3	6.56–6.4		9.28–7.84	
Scaling factor for 256^3	11.84–9.76		13.44–10.88	

All times in ms.

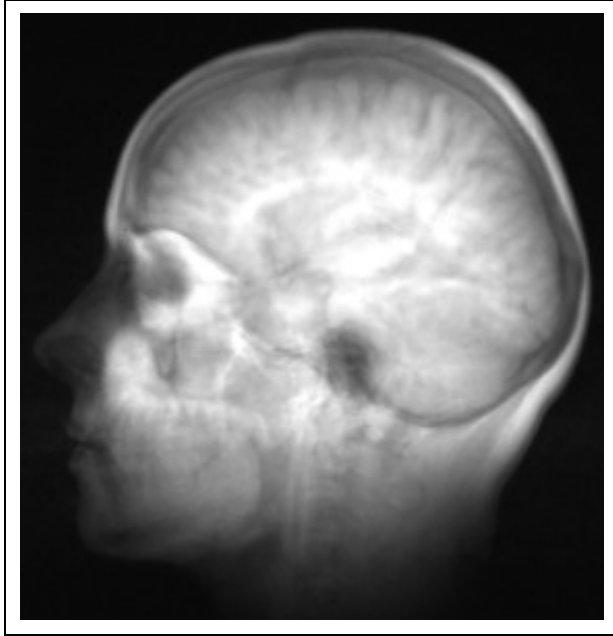
* The used MP-I with 1024 PEs did not have enough memory for the 256^3 volume. Thus we used a special version of the program which allocates memory for one voxel plane only. The calculated images looked strange, of course, but the measured times are correct.

Table 1: Execution times for different configurations

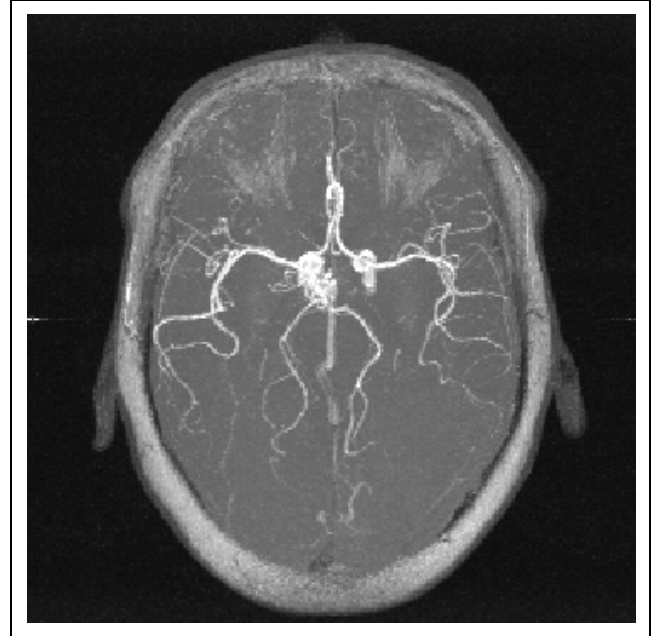
Configuration	Lookup	Bilinear	Worst case
<i>stdstart</i>	1 (0%)	15 (1%)	<i>collookup</i> active
<i>stdstop</i>	43 (3%)	—	frontal view
<i>stdstopint</i>	—	130 (6%)	frontal view
<i>posfast</i>	106 (6%)	—	constant
<i>posbin</i>	—	294 (14%)	constant
<i>scale</i>	215 (13%)	—	constant
<i>collookup</i>	—	203 (10%)	constant
<i>X-ray</i>	276 (16%)	—	constant
<i>emissabsorp</i>	—	388 (19%)	constant
<i>shiftpix</i>	1000 (59%)	1000 (48%)	diagonal view
Overhead	50 (3%)	50 (2%)	constant
Sum	1691 (100%)	2080 (100%)	
Timing	1040 (+62%)	1040 (+50%)	

All times are worst case times and were measured with handler time measurement enabled. *Timing* is the accumulated time needed for speed measurement, resulting from system call overhead.

Table 2: Handler times for volumes of size 256^3



Integration handler *X-ray*



Integration handler *MIP*

Figure 4: Examples for different integration handlers

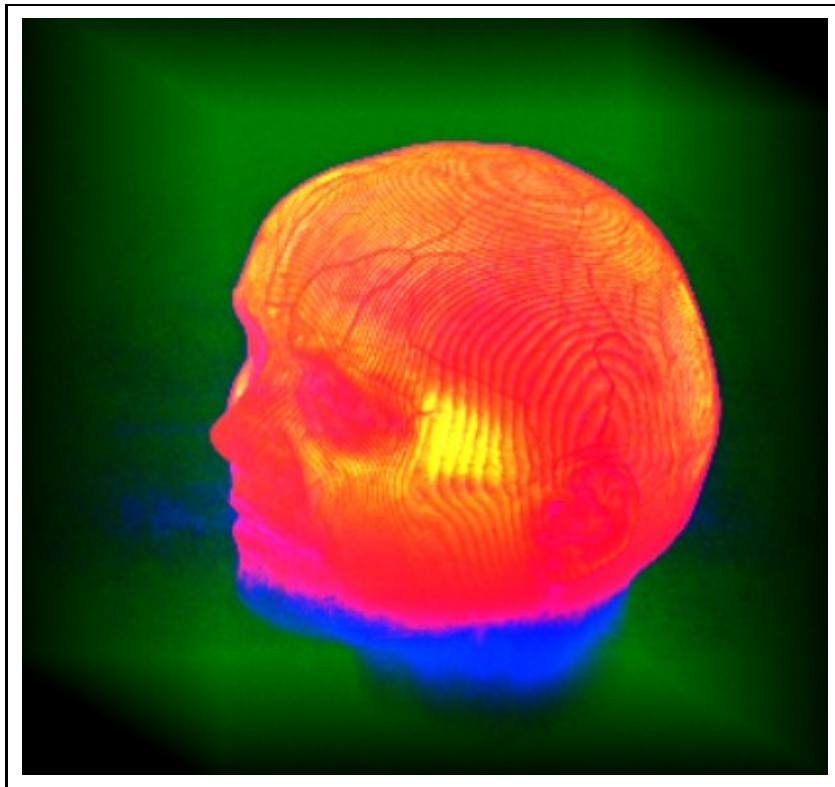


Figure 5: The rendered medical data set