Multiprocessor Iso-surface Volume Rendering

Mark J.S. van Doesburg[a] and Jaap Smit[a]

[a] Univ. of Twente/Network Theory Lab. EL/S&S,
Postbus 217, NL-7500 AE Enschede, The Netherlands

ABSTRACT

The rendering of iso-surfaces in a scalar 3D dataset can be performed with a new algorithm, called iso su
rendering. This algorithm does not introduce sampling artifacts or artifacts due to triangularization.
skip very small details by insufficient re-sampling is also eliminated. Another advantage is its speed c
conventional volume rendering. So far we achieved speeds in the order of ten frames per second on advar
The multiprocessor implementation of this new algorithm uses a division of the voxel data into multiple c
cubes are the basis for distributing the workload onto several processors. A scheduler process is running
the distribution of the workload. During the distribution of the workload the scheduler also eliminates
render invisible parts of the dataset. This reduces the part of the dataset which must be processed t
of the original dataset for typical applications. Another major advantage of the scheduling algorithm
communication overhead is reduced by a factor of ten to twenty, which allows for the efficient use of many

Keywords: 3D, volume rendering, volume visualization, multi processor

1. INTRODUCTION

As the capabilities of medical scanning equipment, like CT and MRI scanners, increase it is becomin
difficult to interpret the results by just looking at the gray-scale slices. For this reason it is desirable t
view the data in a three or even four dimensional manner. Unfortunately the most commonly used m
those given by Lorensen[2] and Levoy,[1] of three dimensional visualization are too complex for normal
This results in very low speed rendering or extremely expensive computers. The problem with low spee
is that in a single image only a subset of the original data is presented, in many cases it is also very
interpret a single three dimensional image correctly. For these reasons we want to be able to render at a
to give the user a way to interact with the dataset, such as rotating the object or even moving inside th

1.1. The render algorithm

The render algorithm must give very high quality images. With high quality we not only mean that th
beautiful but they must be easy to interpret, realistic and provide very detailed information. In mar
information is in the small details, it is important that these details are not skipped for speed. The alg
Bosma[4] combines a very accurate image and low computational complexity.

1.2. Computers

A major factor in the rendering speed is, of course, the computer. Fortunately computers are becomi
time, this doesn't mean we can wait for a fast enough computer to arrive at our desks however. As is al
before the scanners are also increasing in the amount of data they produce, this data needs more processi
create an image. If nothing is done it is not unlikely that the rendering speed will drop despite the faster

Another trend that can be seen in the computer industry is the use of multiple processors insteac
processor. An affordable dual Pentium 3 can be bought in almost every computer-shop.

More recent developments are even integrating multiple processors into a single chip. An example is
from IBM which has 2 processors. Another method is going to be used in the Alpha 21464 chip, there
processor but it has multiple contexts. These contexts will be switched on a cache miss or even a pipeli

Multiprocessor Iso-surface Volume Rendering

Mark J.S. van Doesburg[a] and Jaap Smit[a]

[a] Univ. of Twente/Network Theory Lab. EL/S&S,
Postbus 217, NL-7500 AE Enschede, The Netherlands

ABSTRACT

The rendering of iso-surfaces in a scalar 3D dataset can be performed with a new algorithm, called iso su
rendering. This algorithm does not introduce sampling artifacts or artifacts due to triangularization.
skip very small details by insufficient re-sampling is also eliminated. Another advantage is its speed
conventional volume rendering. So far we achieved speeds in the order of ten frames per second on adva
The multiprocessor implementation of this new algorithm uses a division of the voxel data into multiple
cubes are the basis for distributing the workload onto several processors. A scheduler process is runnin
the distribution of the workload. During the distribution of the workload the scheduler also eliminates
render invisible parts of the dataset. This reduces the part of the dataset which must be processed
of the original dataset for typical applications. Another major advantage of the scheduling algorithm
communication overhead is reduced by a factor of ten to twenty, which allows for the efficient use of man

Keywords: 3D, volume rendering, volume visualization, multi processor

1. INTRODUCTION

As the capabilities of medical scanning equipment, like CT and MRI scanners, increase it is becomin
difficult to interpret the results by just looking at the gray-scale slices. For this reason it is desirable
view the data in a three or even four dimensional manner. Unfortunately the most commonly used
those given by Lorensen[2] and Levoy,[1] of three dimensional visualization are too complex for norma
This results in very low speed rendering or extremely expensive computers. The problem with low spe
is that in a single image only a subset of the original data is presented, in many cases it is also ver
interpret a single three dimensional image correctly. For these reasons we want to be able to render at
to give the user a way to interact with the dataset, such as rotating the object or even moving inside t

1.1. The render algorithm

The render algorithm must give very high quality images. With high quality we not only mean that
beautiful but they must be easy to interpret, realistic and provide very detailed information. In ma
information is in the small details, it is important that these details are not skipped for speed. The al
Bosma[4] combines a very accurate image and low computational complexity.

1.2. Computers

A major factor in the rendering speed is, of course, the computer. Fortunately computers are becom
time, this doesn't mean we can wait for a fast enough computer to arrive at our desks however. As is a
before the scanners are also increasing in the amount of data they produce, this data needs more proces
create an image. If nothing is done it is not unlikely that the rendering speed will drop despite the fast

Another trend that can be seen in the computer industry is the use of multiple processors instea
processor. An affordable dual Pentium 3 can be bought in almost every computer-shop.

More recent developments are even integrating multiple processors into a single chip. An example i
from IBM which has 2 processors. Another method is going to be used in the Alpha 21464 chip, ther
processor but it has multiple contexts. These contexts will be switched on a cache miss or even a pipe

2. CHOOSING THE HARDWARE

The render algorithm chosen is not yet implemented in any hardware accelerator. Most 3D hardware is surface rendering and is almost completely useless for volume visualization. There are some hardware ac for volume rendering available like the Volume Pro 500.[3] However, these hardware accelerators tend t inflexible in their use, and even worse they use a rendering algorithm that gives an inferior image qua compared to the Iso Surface Volume Rendering algorithm.

We could have decided to make dedicated hardware for the Iso Volume Rendering algorithm as well. Unfo the algorithm is, due to its computational efficiency, very complex to implement in hardware. Another prol creating dedicated hardware is that the development cycle tends to be much longer, by the time the design i the hardware is usually obsolete. Also minor modification in the algorithm may lead to large modificatic hardware design. The biggest problem is however the lack of flexibility. This inflexibility makes it impossi the hardware to implement additional features, unless of course the whole design is over engineered and ha a general purpose computer.

For these reasons the decision has been made to run the Iso Surface Volume Rendering algorithm o purpose microprocessors.

3. MULTI PROCESSOR COMPUTER ARCHITECTURE

When using general purpose micro processors we can choose many different designs from many different turers. Another design decision to make is how many processors are going to be used. And in the case of n one processor, we also have to choose a method to connect the processors with each other. As can be se title of this paper we have chosen to use several processors. The reason if of course that, if done correctly processors will be faster than one fast processor.

There are many ways to build a multi processor computer. It can be done as simple as connecting single processor computers with Ethernet, to creating a custom build multiprocessor machine with a high b interconnect.

Using an Ethernet connection to connect multiple computers is the cheapest and easiest solution. Unfo the bandwidth is too low and the communication overhead is large to be useful in a fine grain comm application like the multi processor volume render engine. A more optimal solution would be to provid standard PC that can be upgraded when needed.

3.1. Symmetrical Multi Processor boards

The most commonly used technique to create a multi processor computer with very high speed and lo communication is the Symmetrical Multi Processor (SMP) method. In these system several processors shar memory bus as is shown in figure 1. One reason these systems tend to work very well is that most software relatively small amount of data. This allows for efficient use of the cache memories. In the case of volume however this will not work. For every frame a lot of data needs to be fetched from main memory. As so amount of data that a processor needs to fetch for a single frame is larger than the cache size of that proc cache will need to be refilled on the next frame. As cache memory is expensive the caches are unlikely to than four Mega bytes, which will not be enough for any useful volume rendering application. The result data needs to be fetched from main memory for each frame. As the main memory bus is shared betwe processors this will lead to a severe bottleneck. An analysis of how such a system would perform showed th system with more than four processors would be inefficient for Iso Volume Rendering algorithm.

As long as no more than four processors are used they are an attractive option however, as there a two systems available in almost every computer shop. It is also possible to buy a fourfold SMP system, thes tend to be much more expensive however (more than four times as expensive as a twofold SMP system) development of the Multi Processor Iso Surface Volume Rendering algorithm a twofold SMP system wa contained two Intel Pentium 3 processors.

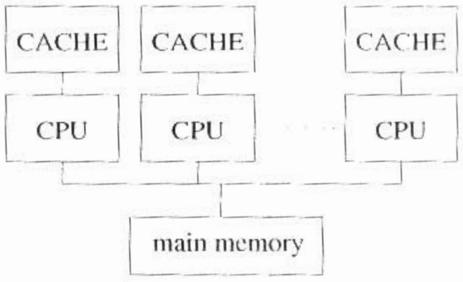

Figure 1. Symmetrical Multi Processor computer.

.2. The StrongARM card

o overcome the main memory bandwidth problem it was decided to create a computer which had much mo
emory bandwidth, which lead to the design of a computer where each processor had its own main memo
omputer was based on a PC with additional processors on PCI boards. These PCI boards contained eight pr
the SA110 type. The SA110 is a member of the StrongARM processor family and has exceptionally lo
onsumption for its performance. This combined with an already available PCI interface chip allowed for
eation of a board as is shown in figure 2

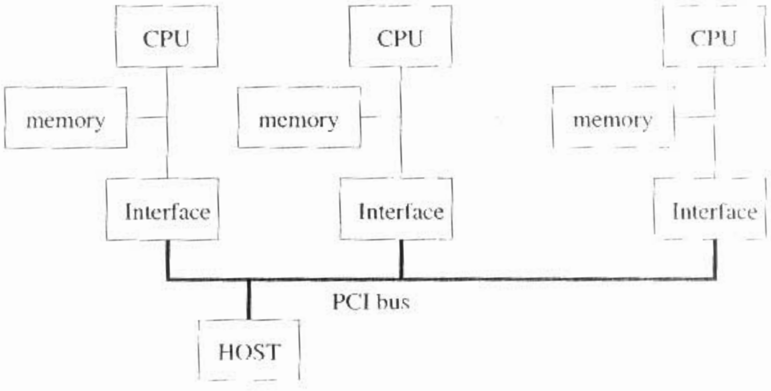

Figure 2. The StrongARM PCI card.

When the project was nearing its completion major performance problems became apparent. The proble
ofold, the first being the system interface of the SA110. This system interface lacked a cache coherency
ich made communication inefficient. Due to the large datasets involved when volume rendering, no single p
the system could have the entire dataset in its main memory. For this reason the processors had to be
vn-load parts of the dataset to their main memory from other processors. There was also a second a
re problematic shared memory area called the ready buffer, which will be explained later. The lack of
erency protocol on the bus interface made it necessary to flush the cache of the SA110 whenever the
mory segments were to be read from another processor. Unfortunately the SA110 was not very efficient
; and the overhead was enormous.

The lack of a floating point unit was a second problem. This was already known when the project was
a new generation of the StrongARM family was already announced. This new generation was supposed
ector floating point unit which would have been perfect for the application. Unfortunately we were unab
icient information on the availability of these devices, and these StrongARMs with a vector floating p
to the authors knowledge still not available.

This led us to abandon the StrongARM based project.

. The G4 boards

SH family of Hitachi. Unfortunately it suffered from the same problem as the StrongARM, the pro an announced next generation which would be almost perfect but without any indication of when it v available. Having learned from the StrongARM project we decided it was not a very good choice. As v on we looked at the Motorola G4 processor.

The G4 is not an extremely low power processor, it consumes 5 Watt compared to 450 mW for the S still possible to build a relatively low power computer with it. The G4 is faster than the SA110, so fewer can be used to achieve the same performance. If one only looks at the power consumption of the proc SA110 would still outperform the G4 however. A more important factor is the power consumption of th system. When these figures are compared to each other the G4 wins. This is mainly due to the power co of the bridge and the memory. A complete cluster of an SA110 with an 21285 and 64MB of SDRAM consu If we create a cluster of four G4 processors we would consume 20W in the processors and ≈10W in th and bridge. If we then compare the 30W of four G4 processors with ten SA110 processors, the G4 proc with a very large margin, as our benchmark have shown the G4 to be more than eight times as fast as t This comparison is not even including the tremendous performance win that comes from the much more ef interface of the G4.

3.4. The bus-interface of the G4

Unlike the SA110, the bus-interface of the G4 is capable of providing cache coherency. This makes it possibl an SMP system with G4 processors. Unfortunately we already came to the conclusion that an SMP sy more than 4 processors would be inefficient. Obviously we do not want to limit our system to only four p The G4 bus interface, is flexible enough to create larger efficient computers. We can create four way SM and put multiple clusters into a large shared cache coherent memory system as is shown in figure 3.

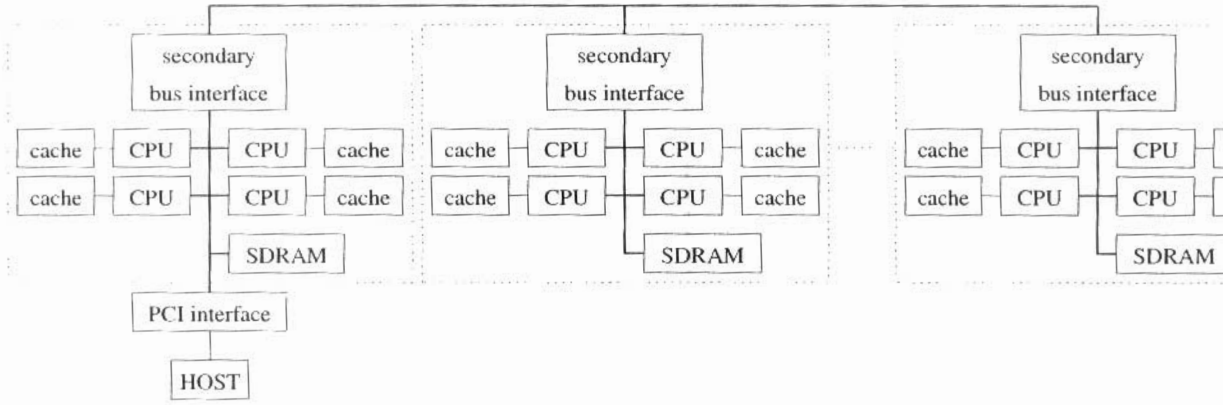

Figure 3. A multi processor G4 system.

3.5. Altivec

he G4 processor has a vector instruction set called Altivec. With this instruction set it is possible to spe ender algorithm. This work has already been done on a single processor machine by M.K. Bosma at the f Twente and can be used in the multi processor G4 system. A similar instruction set is available on the , where it is called SSE. During the development of the multi processor rendering algorithm on a dual Pe peedup of a factor four was achieved by using SSE. The availability of Altivec also lead to the choice of t

4. THE RENDER ALGORITHM

n this section a more detailed overview is given of the Iso Surface Volume Rendering(ISVR) algorithn nowledge is necessary to determine a good method of making a multi processor version of it.

There are mainly two known approaches for volume [illegible] ıt to be computed the ready buffer is checked to see if the computations are really necessary, if the ray y terminated the computations are skipped. This speeds up the rendering process by a large amount.

: rendering is by its very nature very data intensive. For this reason the ISVR algorithm uses a binary binary shell is also checked to avoid unnecessary computations. When the iso surface is reconstructed an ›r is used to find the exact location of the surface. If based on the values in the dataset we can determine is no iso surface between eight neighboring voxels, we can skip that particular space. These small spaces ght neighboring voxels will be called a cell for the rest of this paper. Generating the binary shell can be fast compared to the rest of the computations and therefore using the binary shell is very efficient. Another of the binary shell is that it only needs to be generated when the iso value is changed. This allows the to skip many cells whose voxels will no longer be fetched. This reduces the amount of data that must be ›m the main memory, which also speeds up the rendering process.

5. MULTI PROCESSOR RENDERING

ndering with multiple processors on a general purpose computer is not only a matter of making the software ıltiple processors simultaneously. The properties of the hardware should be taken into account as well. sidering the hardware we assume an architecture as is given for the StrongARM of the G4 boards. This is sentative for most larger computers. The rendering software is also expected to run efficiently on an SMP nce the SMP systems have a much simpler architecture they are far more easy to program efficiently and re developed for a large machine is also close to optimal for an SMP machine.

ꞓ software architecture

lete software system will exist of several processes. These processes will communicate with standard Inter ommunication (IPC) methods. The first method of communication is through the use of network sockets. kets will enable the various processes to send each other packets of information. Another feature of sockets ɛy can be multiplexed with the `select`[5] function call. This function will also put the process to sleep when to be done, which allows the computer to be used for other tasks.

large amounts of data need to be shared, shared memory segments will be used. These large amounts of he dataset, the ready buffers and the output buffer. The whole software structure is shown in figure 4, ; the number of clusters and n is the number of processors in each cluster.

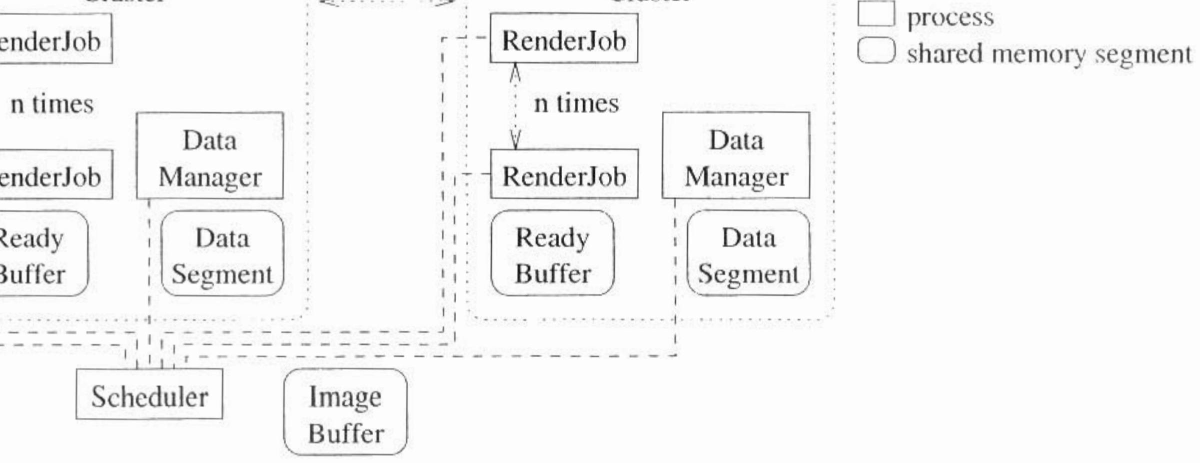

Figure 4. The software architecture.

responsible for the actual rendering. The scheduler will assi

Distributing the workload

are two basic ways to distribute the rendering workload on several processors. It can be done either i space or the output space, where the input space is the voxel dataset and the output space is the image. ne like the G4 board it is much more attractive to take the input space as a basis for the workload distribu ll be explained discussing how to store the dataset.

take the input space as a basis for workload distribution makes it necessary to make a division of the i nto several smaller pieces. These pieces will be cubical subsets of the original dataset and will be called cu cubes will be the smallest unit that can be rendered on a single processor.

nen scheduling the cubes to be rendered on the render processors we can take several properties of the I hm into account. These properties are:

Back to front rendering.

Each ray is terminated only once.

A binary shell gives the parts of the dataset which need to be rendered.

A ready buffer indicates which parts of the image are already finished.

ensure back to front rendering and still use multiple processors we will schedule only cubes that are complet . The sequence in parallel view is shown in figure 5. As can be seen we can render only one cube in the fi This cube will thus be rendered on a single processor. After this cube is done we get three completely visi which will be rendered on three different processors. The amount of available cubes will grow rapidly a most cases soon exceed the number of processors in the system. The amount of cubes available for renderi n in equation 1. Where k is the rendering step and s is the number of cubes along every axis, which is cation as the s may be different for each dimension. Obviously a slight modification is needed when renderi pective, but the principle stays the same.

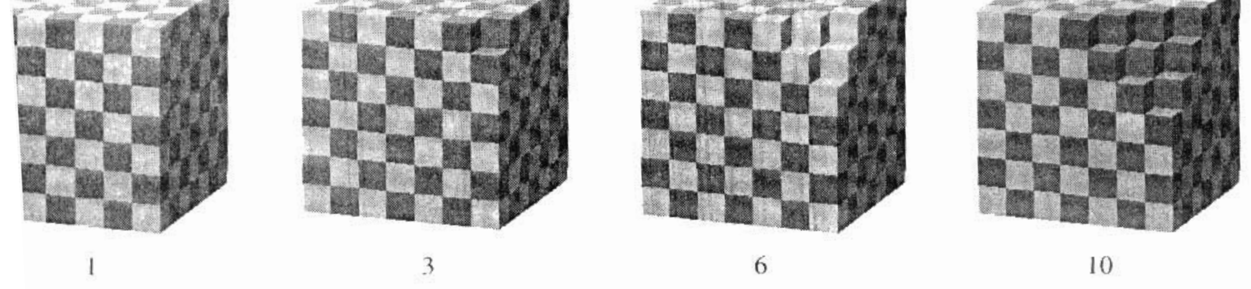

Figure 5. The first four steps in rendering cubes.

$$c(k) = \begin{cases} \sum_{n=0}^{k} n & (k \in [0, s]) \\ \sum_{n=2s-k}^{s} n + \sum_{n=k-1-s}^{s-1} n & (k \in [s+1, 2s-1]) \\ \sum_{n=k-2s}^{s} n & (k \in [2s, 3s]) \end{cases}$$

5.3. Storing the dataset

When storing the dataset on a computer which has an architecture as is shown in figure 3 we must try to l access inside a SMP cluster. If possible we should store the entire dataset in the main memory of each minimize communication over the secondary bus. Unfortunately we will need much more RAM than is a necessary if we do that. The currently available dataset can already be over one hundred mega bytes and the datasets of the future to be much larger. Therefore we should not store the entire dataset in every clust memory but we should try to give each cluster a region in which it should work. The scheduler should available data in a cluster into account when selecting a processor for rendering a specific cube. If for some be explained, reason it is more attractive to render a cube on a cluster which does not have the data avai scheduler should instruct the render processor to down-load that cube into the cluster's main memory fror cluster.

This way most accesses to the dataset can be done locally and the memory bandwidth available for data increases with each cluster added to the system. It is very important to minimize communication over the s bus, as it is much slower than accessing local memory. It is also the only non scalable part of the compute

The data-segment of each cluster is managed by the data manager of that cluster. When the scheduler the data-manager to down-load a cube it will copy the cube from another cluster's data-segment to its necessary it will also delete cubes from its data-segment to make room for the new cube when instructed The scheduler has the responsibility to make sure every cube is present in at least one of the data segments, it may have to be fetched from hard-disk which is very slow.

5.4. Storing the ready buffer

Minimizing the use of the secondary bus for accessing the ready buffer presents another challenge to the s Each cluster will have its own copy of the ready buffer in local memory. The scheduler should instruct the re to fetch the parts of the ready buffer, that are necessary to render a specific cube, from the local memor clusters. This way only local memory access is used when repeatedly checking the ready buffer. Communic be lowered further by making sure that a cube is scheduled on a cluster which also rendered the cubes that the cube that is to be rendered. If a cube is rendered on a cluster which also rendered the obscuring ready buffer is already up to date and no communication over the secondary bus is necessary. It also ma to prefer rendering a cube on a processor which rendered the three obscuring cubes as it makes it more relevant parts of the ready buffer are still in the processor's cache, which reduces communication on the main memory interface.

5.5. Skipping cubes

In the single processor version of the ISVR algorithm we used a ready buffer and a binary shell to reduce th of work. This can also be taken to a coarse grain for the scheduler. Using the binary shell on a cube instea basis is relatively easy. If no bit is set in the entire binary shell of a cube, then the cube in its enterity w able to terminate any rays.

Using the ready buffer on a coarse grain is not so obvious. If we let the scheduler check the ready b time it is about to render a cube, we will use an enormous amount of secondary bus band width. A bette is to let the render-job check the ready buffer and make it report the unterminated ray-counts of its surfa scheduler.

Unfortunately this will force the scheduler to schedule cubes which are empty, since it needs the ray-c remove this unnecessary communication the ray propagation algorithm was developed.

6. RAY PROPAGATION

Ray propagation allows us to skip entire cubes if there is no pixel to be processed for this cube even though may contain an iso surface. In figure 6 a single slice of the binary shell of the MRbrain dataset is show

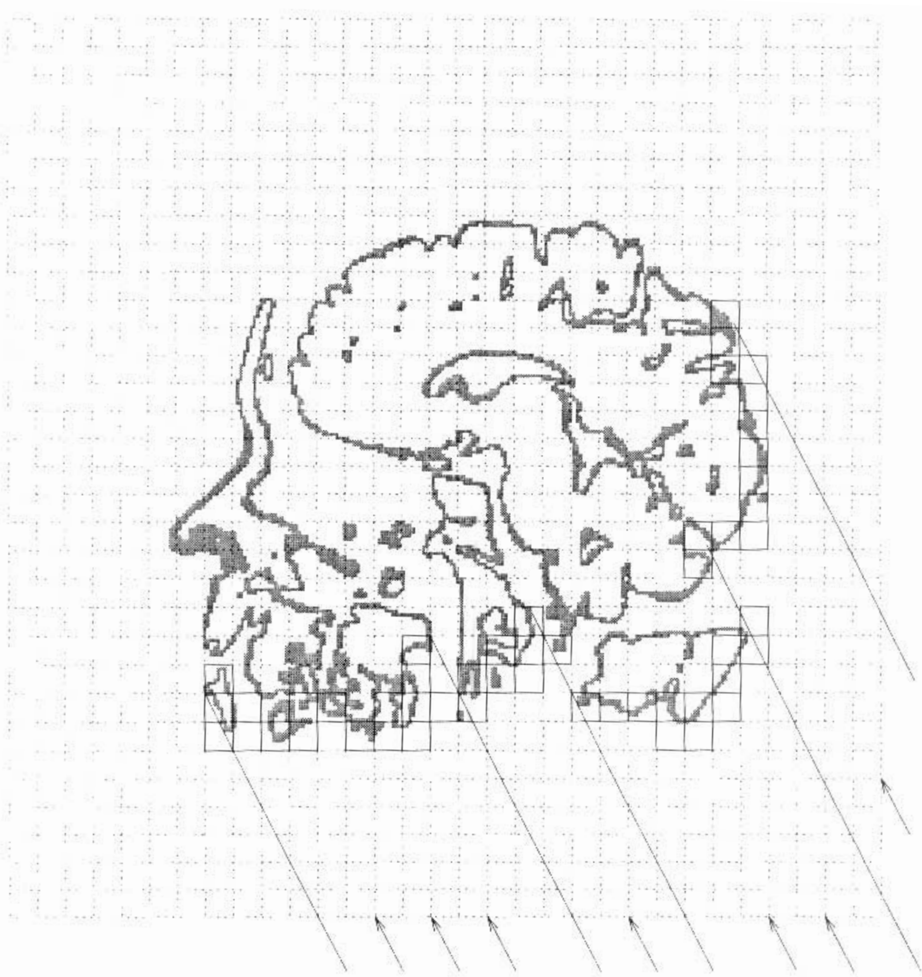

Figure 6. A single slice of the MRbrain dataset.

As can be seen only a small portion of the cubes which might contribute to the image will actual most typical application it will be around 30% of the binary shell. For special application like virtual may even be a much smaller percentage.

We can check the ready buffer to see if a cube has to be processed. A check of the ready bu cube *before* it is processed on a multiprocessor rendering engine implies however a substantial overhea communication. As the ready buffer local to the processor has to be updated for every cube, even if it entering it.

A far more attractive approach is to count the rays *after* the cube is rendered. This would then g counts on the three backside surfaces of the cube. The three backside surfaces of a cube are front-sic three other cubes. When a cube is unobscured by another cube it also receives the ray count from that all three obscuring cubes are rendered, the cube itself is ready to be rendered. Since all un-obscured c how many rays are entering this cube, it can be determined if there are in fact no rays entering this c are no rays entering this cube, this cube can be skipped.

Now we are left with the problem, what to do with the cubes we can skip. Of course we could de ray count as well by checking the ready buffer. This would, again, mean a lot of overhead however. be difficult to see that a least half the object could be skipped since the backside of the object would r The cubes which form this backside should not take much time to process. In most cases there also are [illegible] entirely empty. If we were not to count any rays in the ready buffer, these cubes would r

First an explanation is given on how ray propagation works in parallel view. In perspective view the
becomes more complex and the difference will be explained later.

6.1. Parallel view

Recording if any rays are coming into a cube is a straight forward process. Unfortunately it still leaves us w
cubes we have to check, as each cube with a non-zero incoming ray-count make three other cubes have a
incoming ray-count. We can also be more precise than this by using a matrix. The matrix is called the pr
matrix and it gives the relative overlap between the front side surfaces and the backside surfaces. It can als
as a matrix which gives the probability that a ray exits on a surface, given that it enters on another surfa

The propagation matrix is derived from the visualization matrix $\mathcal{M}$ (given in equation 2). This vis
matrix $\mathcal{M}$ transforms a coordinate from the dataset space $\vec{d}$ to the screen space $\vec{s}$. With the screen space
coordinate of the pixel and the distance from the screen of the point.

$$\vec{s} = \mathcal{M} \cdot \vec{d} + \vec{T}$$

If we have visualization matrix $\mathcal{M}$, we can find the propagation matrix $\mathcal{P}$. To do this we first find the f
point of a cell. This can be done by looking a the bottom row of $\mathcal{M}$. We call this front most point $\vec{c}_0$
vector is $\vec{0}$ by definition. We work in a three dimensional system, which means we will have eight points o
cell. We find $\vec{c}_1$ by taking $\vec{c}_0$ and change dimension 0. In a similar way we find $\vec{c}_2$ and $\vec{c}_4$. All the other d
are linear combinations of these vectors. This automatically leads to $\vec{c}_7$ being the back most point. This is
figure 7.

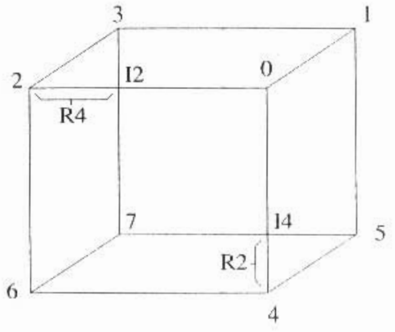

Figure 7. The vectors used for the propagation matrix.

To find the actual matrix $\mathcal{P}$ we do:

if $\frac{\vec{c}_1 \times \vec{c}_7}{\vec{c}_1 \times \vec{c}_2} > 0$ and $\frac{\vec{c}_2 \times \vec{c}_7}{\vec{c}_1 \times \vec{c}_2} > 0$ we use equation 3 to 9.

$$\begin{aligned}
\vec{I}_1 &= \left| \frac{\vec{c}_2 \times \vec{c}_7}{\vec{c}_1 \times \vec{c}_2} \right| \cdot \vec{c}_1 \\
\vec{I}_2 &= \left| \frac{\vec{c}_1 \times \vec{c}_7}{\vec{c}_2 \times \vec{c}_1} \right| \cdot \vec{c}_2 \\
\vec{I}_4 &= \vec{0} \\
\vec{R}_1 &= \vec{c}_2 - \vec{I}_2 \\
\vec{R}_2 &= \vec{c}_1 - \vec{I}_1 \\
\vec{R}_4 &= \vec{0}
\end{aligned}$$

$$
\begin{aligned}
\vec{I}_1 &= \left|\frac{\vec{c}_4 \times \vec{c}_7}{\vec{c}_1 \times \vec{c}_4}\right| \cdot \vec{c}_1 \\
\vec{I}_2 &= \vec{0} \\
\vec{I}_4 &= \left|\frac{\vec{c}_1 \times \vec{c}_7}{\vec{c}_4 \times \vec{c}_1}\right| \cdot \vec{c}_4 \\
\vec{R}_1 &= \vec{c}_4 - \vec{I}_4 \\
\vec{R}_2 &= \vec{0} \\
\vec{R}_4 &= \vec{c}_1 - \vec{I}_1 \\
\Delta &= \left|\vec{R}_1 \times \vec{R}_4\right|
\end{aligned}
$$

all other cases we use equation 17 to 23.

$$
\begin{aligned}
\vec{I}_1 &= \vec{0} \\
\vec{I}_2 &= \left|\frac{\vec{c}_4 \times \vec{c}_7}{\vec{c}_2 \times \vec{c}_4}\right| \cdot \vec{c}_2 \\
\vec{I}_4 &= \left|\frac{\vec{c}_2 \times \vec{c}_7}{\vec{c}_4 \times \vec{c}_2}\right| \cdot \vec{c}_4 \\
\vec{R}_1 &= \vec{0} \\
\vec{R}_2 &= \vec{c}_4 - \vec{I}_4 \\
\vec{R}_4 &= \vec{c}_1 - \vec{I}_1 \\
\Delta &= \left|\vec{R}_2 \times \vec{R}_4\right|
\end{aligned}
$$

he propagation matrix can then always be found with equation 24.

$$
\mathcal{P} = \begin{pmatrix}
\left|\frac{\vec{I}_2 \times \vec{I}_4}{\vec{C}_2 \times \vec{C}_4}\right| & \left|\frac{\vec{R}_4 \times \vec{I}_4 + \Delta}{\vec{C}_1 \times \vec{C}_4}\right| & \left|\frac{\vec{R}_2 \times \vec{I}_2 + \Delta}{\vec{C}_1 \times \vec{C}_2}\right| \\
\left|\frac{\vec{R}_4 \times \vec{I}_4 + \Delta}{\vec{C}_2 \times \vec{C}_4}\right| & \left|\frac{\vec{I}_1 \times \vec{I}_4}{\vec{C}_1 \times \vec{C}_4}\right| & \left|\frac{\vec{R}_1 \times \vec{I}_1 + \Delta}{\vec{C}_1 \times \vec{C}_2}\right| \\
\left|\frac{\vec{R}_2 \times \vec{I}_2 + \Delta}{\vec{C}_2 \times \vec{C}_4}\right| & \left|\frac{\vec{R}_1 \times \vec{I}_1 + \Delta}{\vec{C}_1 \times \vec{C}_4}\right| & \left|\frac{\vec{I}_1 \times \vec{I}_2}{\vec{C}_1 \times \vec{C}_2}\right|
\end{pmatrix}
$$

e can now use P to estimate the ray count on the backside surfaces as given in equation 25. In this equatior ncoming ray count and $\vec{o}$ is the outgoing ray count. The three components of $\vec{o}$ are then added to the incor ount of the three obscured cubes. These cubes may then be skipped in the absence of rays entering these cu

$$
\vec{o} = \mathcal{P} \cdot \vec{i}
$$

n important feature of the propagation matrix is that it will contain many zero components. There ca one non-zero component on the diagonal for example. This will the algorithm to skip many more cubes th ble without the propagation matrix.

Perspective view

e preceding section we derived a matrix to be used for ray propagation. We saw that is was a 3×3 matrix ant components. If we try to derive a similar matrix for the perspective viewing method we will find out tha 6×6 matrix with place dependent components. To compute this matrix for every cube is far too complex

7. CONCLUSIONS

The multi processor version of the Iso Surface Volume Rendering algorithm has been developed and test esults on the dual Pentium 3 800 MHz test machine look very promising for the future implementation nachines. In figure 8 a rendering is shown of the CThead dataset. This dataset can be rendered with 14 fr econd on the test machine. Changing the iso value to be displayed hardly influences the rendering speed. an example of virtual endoscopy is given, the viewer is located inside the head and is looking to the ou he canal where the spinal cord is supposed to be.

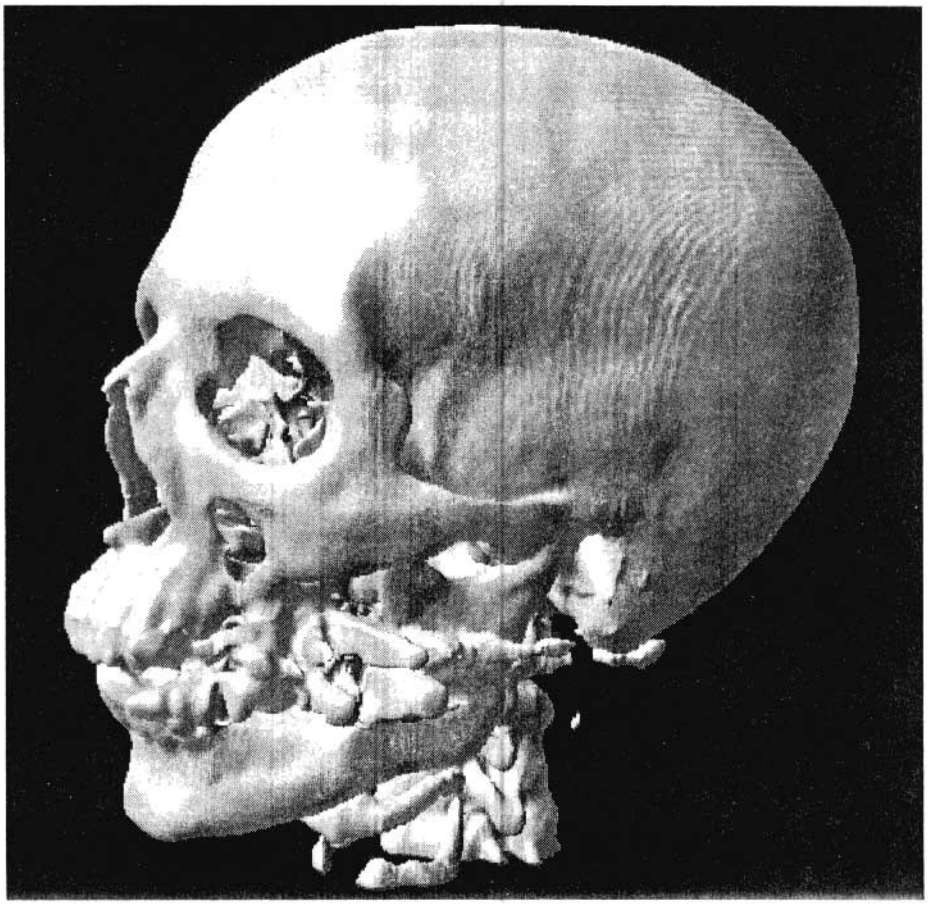

Figure 8. The CThead dataset.

Work on the new multiprocessor G4 machine is in progress and the algorithm has been tested on a nachine. The G4 compares very well to the Pentium 3, one major disadvantage is its low clock speed of New versions of the G4 running at 733 MHz are already available however. Once the machine is implemen 733 MHz G4's, a single cluster is expected to outperform the dual 800MHz Pentium 3 by a factor of 2 and PC can contain at least four of these clusters.

The render engine also needs more work. Currently only a single iso surface can be shown at any tin blending several iso surface should be shown in a single image. Cutting planes are already implemented, cutplanes can only cut along the axis of the dataset. It is our intention to implement these cut planes so can be placed in an arbitrary direction. Combining several dataset should be possible as well. This can by either showing the objects of different datasets in a single image or by color mapping a dataset on th generated from another dataset.

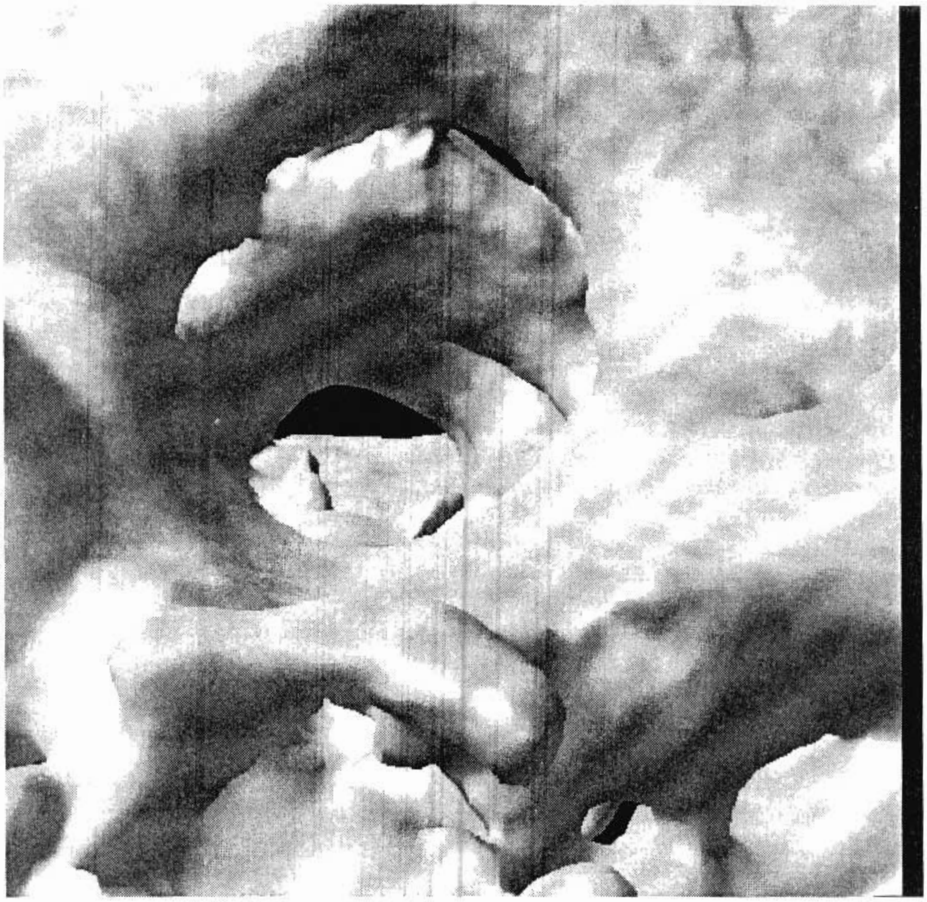

Figure 9. The interior of CThead.

2. W.E. Lorensen/H.E. Cline Marching cubes: *A High Resolution 3D Surface Construction Algorithr* Graphics,21,4,163-169, 1987.
3. H. Pfister, A. Kaufman and F. Wessels. *Towards a Scalable Architecture for Real-Time Volume F* proceedings of the 1995 Eurographics Workshop on Graphics Hardware, pp. 123-130, Maastricht, lands, August 1995.
4. M. Bosma/J. Terwisscha van Schelinga, *Efficient Super Resolution Volume Rendering*. Universit EL-BSC-079N95.
5. Any UNIX or Linux system, *Type: man 2 select.*