REIHE INFORMATIK
26/95

VIRIM, A Real-Time Volume Rendering System for Medicine

A. Gröpl, T. Günther, J. Hesser, J. Kröll,
R. Männer, C. Poliwoda, C. Reinhart
Universität Mannheim
Seminargebäude A5
D-68131 Mannheim

VIRIM, A Real-Time Volume Rendering System for Medicine [1]

A. Gröpl, T. Günther, J. Hesser, J. Kröll, R. Männer, C. Poliwoda, C. Reinhart
Lehrstuhl für Informatik V, Universität Mannheim, Germany
email: hesser@mp-sun1.informatik.uni-mannheim.de

Abstract

VIRIM, a real-time direct volume rendering system is presented. The system is freely programmable and supports models like α-compositing, front-to-back (back-to-front) techniques, and the slab method. The hardware system is divided into two units, a geometry unit and a raycast unit. The geometry unit performs resampling and gradient estimation and is mapped directly into hardware. It supports different resampling filters in order to reduce resampling artifacts. The raycast unit consists of 16 digital signal processors that perform the programmable ray-casting.
The software of VIRIM is layered and provides manipulation tools for the data during real-time visualization like arbitrary gray-value mapping and setting the region of interest.
The system is under test and will be available as prototype 1995.

Introduction

Various imaging systems like computer tomography (CT), magnetic resonance imagery (MRI), ultrasound tomography (UST), and other systems produce large amounts of data in short time, the physician has to work with and analyze for diagnostic and therapy planning purposes [1]. This huge amount of data forces the physician to switch from investigating one slice after the other to viewing 3D reconstructions.
The current acceptance of 3D reconstruction is severely hampered by limited visualization quality of currently used surface oriented visualization algorithms [2]. Determining surfaces of objects in medical volume data sets can be very difficult and requires detailed segmentation. In other cases however, surfaces do not exist, e.g. for a tumor ramifying into the surrounding tissue.

[1]This project is supported by the German Ministry for Education and Research (BMBF) under grant 01 IR 406 A 8.

Direct volume rendering algorithms overcome the problem of finding surfaces in the data volume by calculating the interaction light matter throughout the volume. Semi-transparency and the visualization of small structures, while posing problems for current algorithms, turn out to be realizable very naturally. Semi-transparency is obtained by objects with opacity values less than one. Small structures are visible because they reflect some part of incoming light which is projected on the viewing plane.
A high computational demand has to be taken into account when using these algorithms. For a class of front-to-back algorithms interactive visualization rates have been achieved by preprocessing and the use of multiprocessors or graphics workstations [3]. Especially pre-calculations are of limited use since interactive classification and modification of data properties like opacity or movements of the viewer are not possible. Real-time rates and user interaction seem therefore to be restricted to special purpose rendering systems.
Among all volume rendering architectures, VIRIM is nearest to realization. The main design issues have been programmable real-time volume ray-casting as well as modularity and scalability. In the first section the most widespread volume rendering algorithms are introduced, section 2 is concerned with the VIRIM hardware, section 3 describes the user-interface, and the software structure.

Visualization Methods Supported by VIRIM

Direct volume rendering algorithms have in common that the interaction light-matter is calculated throughout the data volume. Their differences originate from the exactness the physical process of illumination is realized.
α-compositing does not require complex shading and can be realized efficiently in texture mapping hardware [4]. To each volume element α, R, GB values are assigned in a preprocessing step. The data volume is first resampled in a new coordinate system, then each slice parallel to the projection plane is blended in a frame buffer for each RGB color:

$$\text{slice}(i+1) = (1-\alpha)*\text{slice}(i) + \text{voxel\_color}*\alpha,$$

The connected physical process is self-lumination, which however does not allow to see the orientation of local surfaces.
For higher quality demands, the local orientation of the surfaces has to be visible which requires the introduction of the shading process into the algorithm. The methods are known as front-to-back or back-to-front techniques, which are in widespread use [5]. The shading process is usually realized by Phong models [6]:

$$I_\lambda = I_{a\lambda} k_a O_{d\lambda} + f_{att} I_{p\lambda} \left[k_d O_{d\lambda} (\mathbf{N} \cdot \mathbf{L}) + k_s O_{s\lambda} (\mathbf{R} \cdot \mathbf{V})^n \right]$$

where I_λ is the intensity reflected to the viewer direction **V**, $I_{a\lambda}$ is the spectral intensity for ambient light, $I_{p\lambda}$ is the spectral light source intensity, k_a, k_d, k_s are the coefficients for ambient light, diffuse, and specular reflection, $O_{d\lambda}$ is the object's diffuse, and $O_{s\lambda}$ is the object's specular color, f_{att} takes into account that the light intensity decreases with the distance to the light source, n is a constant defining the pointness of the reflection, **N** is the normal direction of the surface, **V** is the direction of the viewer, and **R**, the direction of reflection, is given by

$$\mathbf{R} = 2\mathbf{N}(\mathbf{N} \cdot \mathbf{L}) - \mathbf{L},$$

where **L** is the direction of the light source.

In many cases shading can be simplified considerably. Omitting color, ambient light and setting n=1, f_{att}=1 reduces the amount of computations significantly and nevertheless gives acceptable visualization results.

However, lacking shadows is an unphysical process and comes from the neglect of the absorption of incident light. Shadows can be generated by calculating the interaction of incident light with matter. At least two light sources are required for evading dark shadow areas in the image. The slab method [7] is one example of this more physical visualization model. It operates with two light sources at 0° and 45° from the viewer using parallel or point light sources and relies on Phong shading for the light-matter interaction.

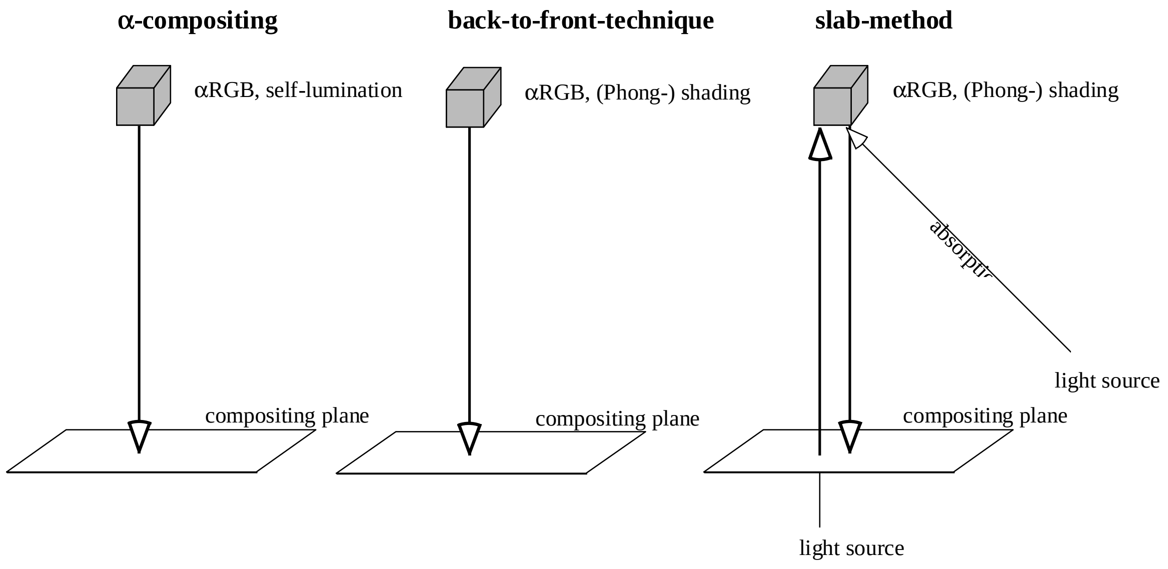

Fig.1 Schematic of three different volume visualization algorithms.

VIRIM Hardware

Direct volume rendering algorithms have in common that the data set has to be resampled in new coordinates and except for α-compositing local gradients are required for shading using the slab method. These operations consume 80% of the computation time and have been mapped directly into hardware for the VIRIM system. After volume resampling and gradient estimation shading and compositing must be flexible and programmable in order to

implement the different visualization models mentioned above. Due to their high I/O and computing resources VIRIM uses digital signal processors to carry out these operations.
VIRIM has a modular design where each module consists of one geometry unit and a raycast unit with 16 DSPs. Four modules are required for real-time visualization of 256x256x128 data volumes.

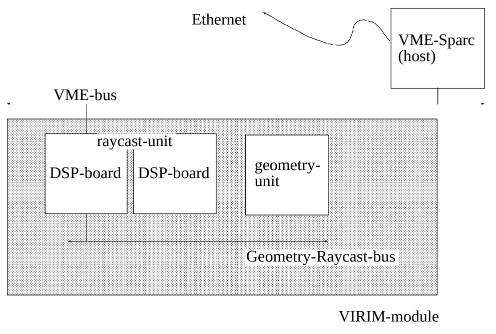

Fig.2 Sketch of VIRIM-hardware

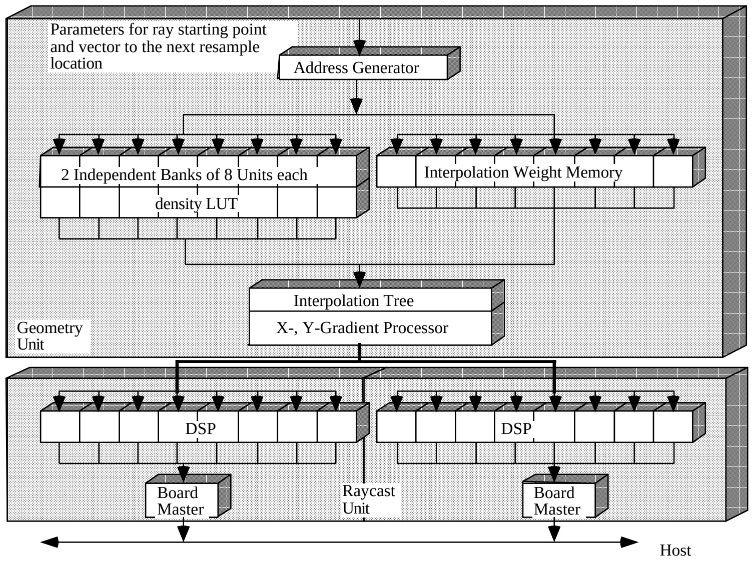
Parameters for ray starting point and vector to the next resample location
Address Generator
2 Independent Banks of 8 Units each
density LUT
Interpolation Weight Memory
Interpolation Tree
X-, Y-Gradient Processor
Geometry Unit
DSP
DSP
Board Master
Raycast Unit
Board Master
Host

Fig.3 Sketch of hardware architecture

The system has the following technical data:

- modular and scalable system
- programmable
- real-time change of the viewer position
- real-time gray-value segmentation
- perspective projection
- real-time setting of the region of interest

High Accuracy Resampling

Mapping rotation directly into hardware fixes the rotation algorithm used; which has therefore be chosen very carefully. In other words, rotation with the least amount of artifacts has to be realized.

The starting point for each rotation algorithm is the given discrete 3D-image. The Fourier image of the original data, which is of limited size, is periodic. Rotation in the Euclidean space is realized by sampling in rotated coordinates. Due to the spatial limitation of the resulting image, its Fourier transform has to be periodic again.

In the Fourier space the following happens (see fig. 4). Sampling in rotated coordinates means that the image area in the Fourier space is rotated with regard to the original Fourier transform. Periodicity means that all surrounding areas fold their information into the Fourier image; which leads to artifacts.

Box filters allow the exclusion of these artifacts by removing all information around the interesting image region. Its Euclidean counterpart is a sin(x)/x filter that is not bounded in space. Hardware realization however is only feasible with small (2x2x2) filters. Consequently we have chosen a local approximation of sin(x)/x (fig. 5).

The rotation algorithm proceeds as follows:

1 the distance of the sample point to its 8 neighbors in the original image is determined,

2 the distance is considered as address value of a mask memory that stores the weights,

3 the gray values of the 8 neighbors are interpolated using the weights from mask memory

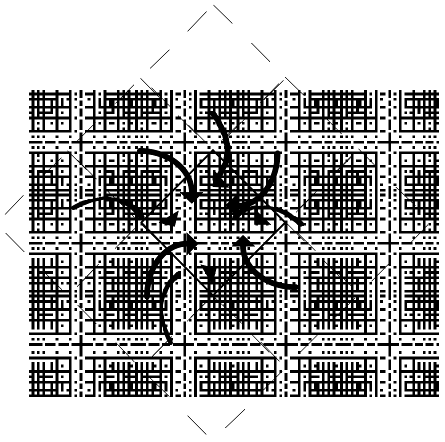

Fig.4 Folding operation in the Fourier space.

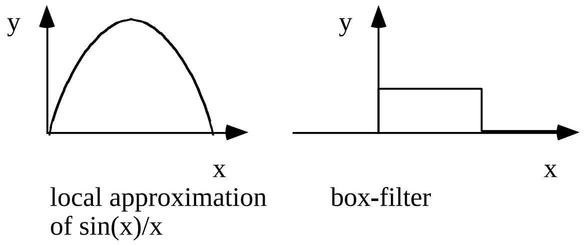

Fig.5 Image filters: box(x) in the Fourier space equal Sin(x)/x in the real space.

Geometry Processor

Rotations are the most important geometry operations and they pose two problems for hardware architectures, data access and data distribution.

Data distribution Problem

Geometry operations like rotation lead to bottlenecks for massively parallel processors. Given a certain part of the final projection, the required data for visualization can lie anywhere in the original data cube thus leading to data distribution and communication problems. Mainly the question how to distribute data over the local memories of processing elements (PEs) poses the most severe difficulties. Two main distribution strategies can be identified, original image data distribution, and final image data distribution, which are defined as follows:

For original image data distribution the data set is assigned to local memories of processing units once when loaded. No redistribution is necessary during visualization since high-speed channels between processing units guarantee the necessary transport of partial results between these units.

Final image data distrubtion assumes that visualization is performed in two steps, first a geometric transformation of the original data set into new coordinates, second the simulation of interaction of light with virtual matter. In these new coordinates the viewer looks into one of the main axis directions (in our case the y-axis). This transformation is performed by geometry units whereby each unit resamples one or more slabs of the rotated data set. Now, wih final image data distribution each geometry unit contains only that data of the original data set which is required to resample the slabs. After each rotation of the slabs data transfer is necessary to newly resample the rotated slabs.

Original image data distribution poses data transfer problems with the slab method for visualization. Since each digital signal processor (DSP) calculates one or more scanlines of the final projection it requires the corresponding slabs from the geometry units. For original image data distribution the slab data is calculated by several geometry units. It follows that each DSP must have access to all geometry units. As consequence an interconnection network is required with high flexibility and data transfer rates in the order of Gbytes/s; which is difficult to realize with current technology. Therefore, this solution has not been considered for VIRIM.

Final image data distribution in contrast demands that each geometry unit obtains only the data it requires for calculating the slabs that are sent to the DSPs (see. Table 1). Replication of data poses the least problems with communication bandwidth but requires high data storage. Sharing the volume memory to all geometry units, while reducing the demand on data storage, has to cope with high data rates and access conflicts when several geometry units want to access the same data; not to mention the problems to provide the high data rate from the volume memory.

Caching data seems to offer a good compromise in terms of data storage and communication bandwidth.

	replicated data	cached data	shared data
communication bandwidth	no	moderate	high
data storage	high	moderate	low

Table 1 Data distribution solutions

Considering that for VIRIM only 4 geometry units are required for achieving real-time frame rates of typical data sets with 8 M volume elements, concrete figures can be determined for communication and storage (s. Table 2).

Currently VIRIM can visualize data sets of size 256^3, where each volume element occupies 2 bytes. In total, the volume memory requires 32 MByte data storage.

Data replication requires no data communication but 128 MByte data storage (4 geometry units with 32 MBytes volume memory each). Shared data strategies require only one volume memory of 32 MByte but the necessary communication bandwidth is 16 MByte per data set * 10 Hz visualization rate * 8 interpolation neighbors for resampling = 1.28 GByte/s.

Concerning caching strategies only a static distribution of image data before visualization is considered. Redistribution of data during the visualization process depends on many factors like cache size, maximal data transfer rates of dynamic memory and the used bus architecture, as well as the time for bus arbitrartion. Optimization of all these figures under technical constraints will be an issue of redesigning VIRIM.

If the data set is rotated by 1° around its center the slabs are rotated out of their original position by up to 2 volume elements. The newly required data amounts to 256^2 * 2 volume elements. Each volume elements requires 2 bytes, i.e., for each geometry unit and each frame 500 kBytes data have to be exchanged, in total 4 (geometry units) * 10 (Hz) * 500k = 20 MBytes/s, which is achievable with bus systems of current technology. Static data distribution however demands that the cache size of all geometry units together is at least as large as the data volume. It follows that the cache memory in all geometry units plus the volume memory is 64 MByte.

From this point of view the advantage for a cache solution is marginal over the use of replicated data. But it has to be paid for with additional hardware on the board. An address translation buffer is required that translates the data set relative addresses into the address of the location of the related data block on the cache memory. Additionally a cache controller has to be implemented.

Using only 4 geometry units, replication of data seems to provide the least amount of complexity and is realized for the prototype. For higher parallelism a cache strategy will be imperative in the future and will be investigated using the first VIRIM prototype.

	replicated data	cached data	shared data
communication bandwidth	no	estimated: 16 MByte/s	1.28 GByte/s
data storage	4x 32 MByte	32 MByte for volume memory and 32 MByte for cache memory	32 MByte for volume memory
hardware overhead	no	high	high
cost memory	1-2 k ECU	no	no

Table 2 Data distribution solutions for VIRIM with 4 geometry units that require the access to the volume data

Data Access Problem

The high processing power of VIRIM is achieved by using a fast memory architecture. The data transfer rates underlines these requirements. 8 neighboring gray values of 16 bit each have to be read out at a rate of 20 to 40 MHz; which leads to a read-out rate of 320-640 MBytes/s. More important, for low cost, only commercial DRAM can be taken into consideration. How can this high data rate be achieved?

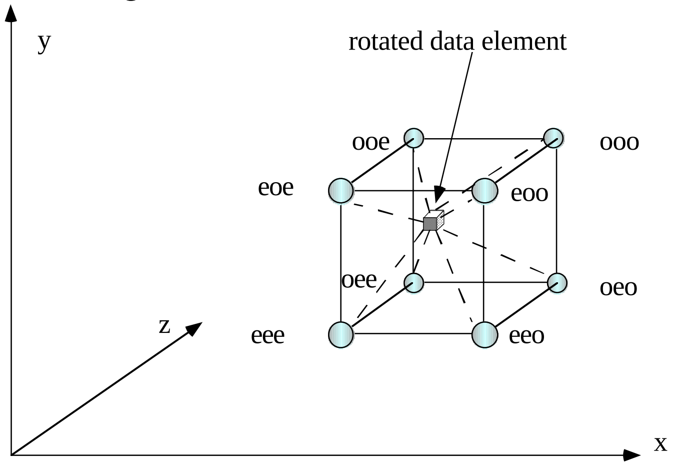

Fig.6 Distribution of volume data over memory units

The solution is the sampling along rays using three approaches.

The first approach relies on the observation that each neighbor has a different combination of even and odd x-, y-, and z-coordinates that allows the use of 8 separate memory units. Each unit contains data from one combination of even and odd x-, y-, and z-coordinates.
Next, DRAMs have the possibility to read out contiguous data much faster than for random access (fast page mode). Subdividing the data cube into many sub-cubes, that fit into one page each, makes use of this feature.
Both approaches have also been suggested independently by other authors [8] for high-speed data read-out architectures. However, the rate is still below 20 MHz and therefore not sufficient for our design. Further improvements are necessary.
Before discussing these improvements, a detailed sketch of the realization of one memory unit is shown in fig. 7. Data comes from the address generator and is buffered in the left FIFO. The comparator reads the addresses in the FIFO and sends it to the two memory banks associated with a controller. Data is read out from one memory bank and stored in a register (1-entry level cache), which has its own controller. Finally the data is written into a FIFO from which it can be read for the interpolation. The controllers are activated by the comparator using key-words.

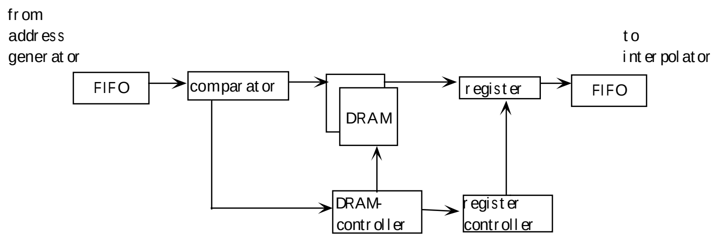

Fig.7 Sketch of the Memory Unit

The first method for increasing memory read-out speed is to bypass the memory whenever possible. This is the case when the data lies outside the original data cube and has gray value 0; the comparator produces a control word *out-of-bound* which invokes the register controller to reset the register.
Another case occurs when the same data is re-used. The comparator outputs a *same-data* control word and causes a hold-data on the register through its controller.
The second method is to use two memory banks which allow an alternate read-out when the subsequent data are stored in different banks. The following example clarifies the interplay of these two methods.

Example

Let us assume only a 2D rotation for simplicity. In fig. 8 one plane of the original data cube is shown. One memory unit is considered next, where data from that unit is marked by circles. White circles denote data stored in memory bank 0 and black circles denote data of memory bank 1. Data of all other memory units are on the other grid intersections and in other planes of the data cube.

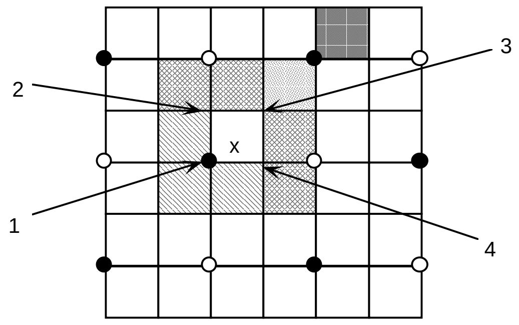

Fig.8 Example

Let us assume that we are sampling the point that is indicated by a cross. I.e., its density is obtained by interpolation from 1) data from the black circle, 2) three further memory units denoted by 2,3,4 respectively; and 3) additional four other memory units that are in the next plane.

Assume first, that the next point is sampled in one of the neighboring squares of the grid. Three different cases can be observed:

In the first case, the point falls within the hatched area. For interpolation of the gray value of the sample point, the same data (black circle, i.e., bank 1) is required from the considered memory unit. The *same-data word* is issued by the comparator allowing the availability of the data at a 40 MHz rate.

In the second case, the sample point falls into the meshed squares. Data from bank 0 (white circles) has to be read-out from the memory unit, i.e., a alternate read-out is possible. Again, the rate is 40 MHz.

Only in the dotted square, different data from the same bank of the memory unit has to be read, reducing the read-rate to 20 MHz. As sampling along a ray will yield the next point in the gray square, again the same data is required, i.e., case 1 occurs with a read-out rate of 40 MHz. In total, sampling along this direction gives an overall read-out rate of 30 MHz.

One could argue that if the next sample point lies in the gray square, different data from the same bank has to be read for the second sample point, which reduces the read-out rate to 20 MHz. However, in this example, the distance between the sample points is about three times the base length of the grid. Therefore it can be assumed that most of the sampled data cube lies outside the original one. Sampling in that outer region however is done at a 40 MHz rate that levels out the lower rate within the data cube.

The example that has just been discussed gives only the motivation why all hardware features are tightly connected for fast memory read-out. Simulations are necessary in order to obtain nearly real figures. Our simulation that has been carried out before designing the system was based on mean, worst, and best cases. The observed rate has been in the range of 26 to 36 MHz with an average read-out rate of 30 MHz.

Raycast Unit

While the geometry unit has been designed for maximal data throughput the raycast unit has to be flexible. 16 DSPs per geometry unit, 8 per board, are used for rendering where each DSP calculates one or more lines of the final image.
Data from the geometry unit enters the board over an geometry-raycast-interface. The geometry unit selects that DSP which should receive the information and sets the write enable signal for the respective DSP-FIFO before writing the data to it. Asynchronously the DSP accesses its FIFO which automatically maps integer data from the geometry unit into floating-point data used by the DSP. The DSP calculates the absorption, Phong shading, and compositing. A cut register is used to determine if the light is reflected to the viewer. It returns 0 for negative input and leaves the input unchanged otherwise.
When the DSP has finished one line of the projection, it invokes the local board master by interrupt to read the result and transfer it to the host system.

VIRIM System Software

One mayor design issue has been the embedding of VIRIM into existing clinic information systems and networks. VIRIM uses the possibilities of the UNIX environment for integration into a local area network (LAN) which allows the use of VIRIM from any possible X-terminal or UNIX workstation that are attached to the LAN. Ideally all data processing devices in the clinic are attached to the network, the diverse scanners, data servers, workstations and X-terminals, as well as VIRIM. One or more physicians can log in the visualization system for inspection of the image volumes or for detailed analysis of possible diagnosis or therapies.
The layered software model used for VIRIM ideally adapts to these demands. It basically consists of four layers. In the lowest layer (layer 1), the system is programmed to perform the visualization task, i.e., software for the DSP executing the raycast operations —determining the visualization model used.
In the next layer 2, raycast units and geometry units each are controlled over a local board master CPU. Its purpose is the control of the visualization process like reset, start, and the feeding of the registers with visualization parameters like orientation of the data set, its magnification etc.
Layer 3 is the communication layer, which runs on the host system. It has mainly the task to map the data from external sources like user interfaces or other devices into visualization parameters, and to transfer these parameters to the respective units. Additionally this layer

does the synchronization of the visualization process so that a continuous flow of visualized data is represented on the screen.

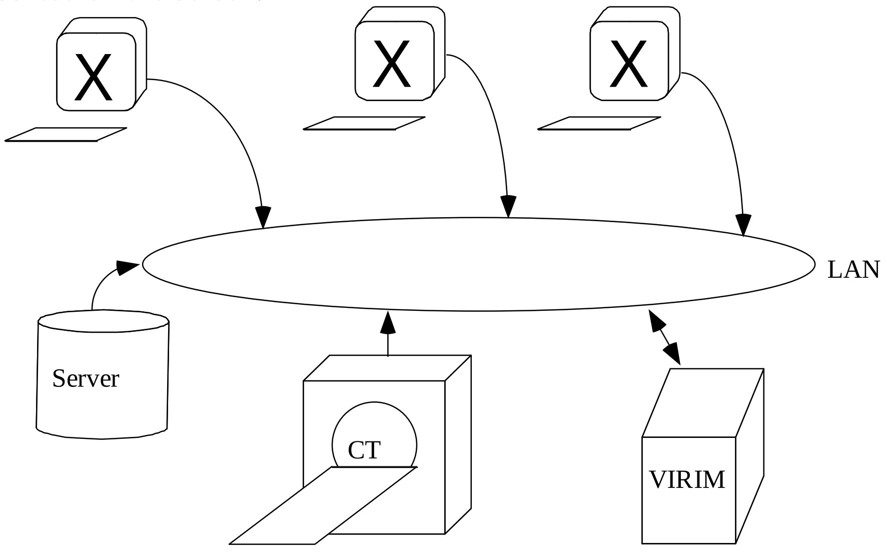

Fig.9 Embedding of VIRIM into a clinic network

In the fourth layer, the user interface is operated, which will be described next.

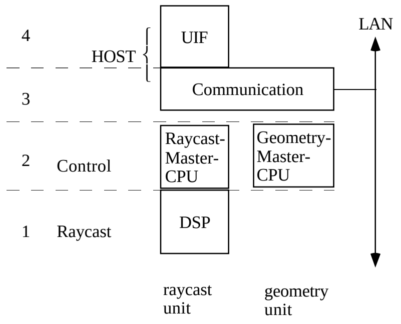

Fig.10 Software layers in VIRIM

In the layer structure the interface between the layers is specified so that changing the visualization algorithm or a new communication layer only requires the exchange of the respective layer while the other software remains unchanged. Ease of generalization and extensibility is therefore guaranteed.

layer #	Processor	Operations
1 Ray-casting	DSP	• ray-casting operations, • visualization algorithms
2 Control	master CPU on ray-cast/geometry board	control operations like • reset, • start, • load programs, • load parameters, • read-out results, • test modes
3 Communication	Sparc-VME-host (eventually other sy-stems)	• data transfer/communication • parameter mapping • synchronization • communication with other modules
4 UIF	any UNIX system accessible via LAN	• display of visualization result • user interactions • manipulation of data • i/o purposes

Table 3 Software layers of VIRIM

User Interface and Image Manipulation Tools

The user interface window of VIRIM is divided into two parts, an image presentation part on the left side, and an image manipulation part. The image presentation part shows the results of image manipulation and visualization.

Three different modes for manipulation are possible which are shown separately in their own sub-window.

In the first mode, the lighting parameters can be varied, i.e., diffuse light from 0° and 45°, and specular light form 45°. Higher proportions of diffuse light increases the cloudiness of objects, which is especially important for semi-transparent objects. The specular part emphasizes the surfaceness and produces light spots which ease the recognition of sharp boundaries and the 3D structure of surfaces.

In the second mode, the user can define the region he wants to visualize. By moving the edges and corners of the cube depicted on the manipulation part, the respective part of the data cube is cut out for visualization.

Finally, in the third mode, the user has the possibility to modify the density values of the original data cube arbitrarily. This mode is especially interesting for setting objects semi-transparent. Gray-value segmentation is possible in real-time. When using pre-segmented data by using a code for each object, organs can be made transparent by simple button clicks.
A 3D input device with 6 degrees of freedom like a 3D mouse will be used for 3D rotation and translation of the data cube.
A menu allows load/store of data cubes and LUTs, to set the size for the sampled data cube, and to report the results of the manipulations.

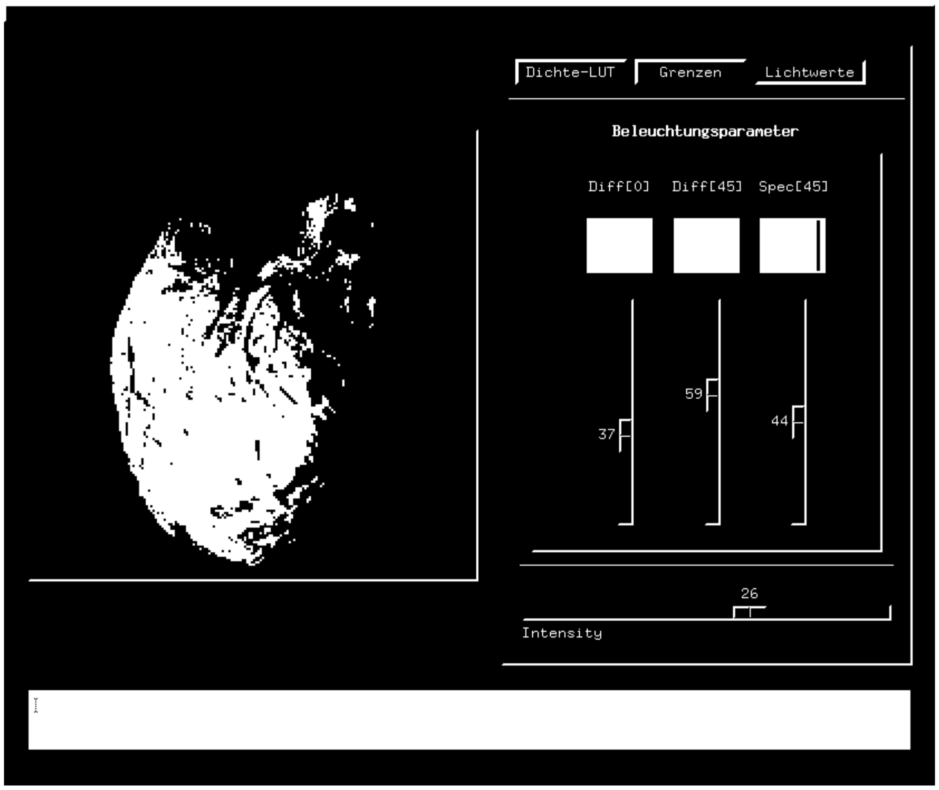

Fig.11 One module of the user interface for the modification of the light components.

Applications

VIRIM opens a wide field of applications in training, diagnosis, and therapy planning for physicians. Besides others, application areas for the system are operation planning in heart surgery, neuro endoscopy, and orthopedics.
In heart surgery [9] the detailed morphology of the heart is not known by the physician before operation. Detailed diagnosis and therapy have to be made at the open heart during the operation. Possible diagnosis is the insufficiency of the cardiac.
Modern ultra fast CT scanners allow the imaging of the beating human heart. The pre-segmented data set will be loaded onto the machine and the segmentation results will be

improved. E.g., we can map identical areas of oversegmented images into one gray value region using the gray value mapping feature. Next the surgeon will inspect the morphology of the heart, e.g., disfunctions of the cardiac. VIRIM allows the visualization of the heart in action; which is a decisive advantage since the functioning of the cardiac and possible other defects can be observed whereas during operation the heart is relaxed and its morphology can be estimated only hardly.
Endowed with this information the surgeon will be able to plan the surgery in detail, i.e., he can spend much more time for the operation itself. Benefits like shorter operation time and better diagnosis and therapy will follow.

Conclusions

VIRIM, due to its free programmability, provides many options for image manipulation, especially gray-value segmentation. Currently the segmentation is done by directly manipulating the mapping LUT leaving the problem of finding the appropriate mapping to the user. Segmentation tools will be developed which are tailored to the different applications areas and which allow the physician more easily to define grayscale windows for the different objects and to manipulate their density more naturally.
Another demand arised when only a small part of the image is interesting to the physician. Here he likes to select arbitrary regions of interest. For this task we will develop additionally tools to perform this operations as ergonomic as possible.
Real-time visualization of time dependent data is another feature that will be implemented by software on our system. At the moment the volume memory on the geometry unit is limited to 256^3 data sets. This allows to visualize up to eight 128^3 or sixty-four 64^3 data sets, which may either be different in modality (CT vs. MRI or ultrasound images) or data imaged at different times. An example of the latter case it the real-time visualization of a beating heart.

References

[1] Adam, J.A. *Medical Electronics.* IEEE Spectrum, Jan. 1995, pp. 80-83.

[2] T. Günther, C. Poliwoda, C. Reinhart, J. Hesser, R. Männer, H.-P. Meinzer, J.-J. Baur. VIRIM: A Massively Parallel Processor for Real-Time Volume Visualization in Medicine. W. Straßer, 9th Eurographics Workshop on Graphics HW, Oslo, Norway, 1994, pp. 103-108.

[3] P. Lacroute and M. Levoy. Fast Volume Rendering Using a Shear-Warp factorization of the Viewing Transform. Computer Graphics, Proc. of SIGGRAPH '94, Orlando, FL, 1994, pp. 451-457.

[4] Wilson, O., Van Gelder, A., Wilhelms, J. Direct Volume Rendering via 3D Textures. subm. to Proc. Vis. '94.

[5] M. Levoy. Display of Surfaces From Volume Data. EEE Comp. Graphics & Appl, Vol. 8, No. 5, 1988, pp. 29-37.

[6] B.-T. Phong. Illumination for Computer Generated Pictures. CACM, Vol. 18, No. 6, June 1975, 1975, pp. 311-317.

[7] J. Kajiya, B. v. Herzen. Ray tracing volume densities. Proc. Siggraph '84, pp 164-174.

[8] Knittel, G., Straßer, W. A Compact Volume Rendering Accelerator. Proc. IEEE 1994 Symp. Volume Vis., Washington, October 17-18, 1994, pp. 67-74.

[9] C.F. Vahl, H.-P. Meinzer, S. Hagl. Three-dimensional Presentation of Cardiac Morphology. Thorac. cardiovasc. Surgeon, Vol. 39 (Suppl.), 1991, pp. 198.