Interner Bericht

Efficient 3-D Visualization of Hybrid Medical Data Sets

Rolf H. van Lengen

Jörg Meyer

Universität Kaiserslautern

257/94

Fachbereich Informatik

Universität Kaiserslautern · Postfach 3049 · D-67653 Kaiserslautern

Efficient 3-D Visualization of Hybrid Medical Data Sets

Rolf H. van Lengen

Jörg Meyer

Universität Kaiserslautern

257/94

Universität Kaiserslautern
AG Computergraphik
Postfach 30 49
67653 Kaiserslautern

Oktober 1994

Herausgeber: AG Graphische Datenverarbeitung und Computergeometrie
Leiter: Professor Dr. H. Hagen

Efficient 3-D Visualization of Hybrid Medical Data Sets

Rolf H. van Lengen and Jörg Meyer
University of Kaiserslautern
Department of Computer Science
P. O. Box 3049
D-67653 Kaiserslautern
Germany

Abstract

Visualization of large data sets, especially on small machines, requires advanced techniques in image processing and image generation. Our hybrid raytracer is capable of rendering volumetric and geometric data simultaneously, without loss of accuracy due to data conversion. Compound data sets, consisting of several types of data, are called "hybrid data sets".
There is only one rendering pipeline to obtain loss-less and efficient visualization of hybrid data. Algorithms apply to both types of data. Optical material properties are stored in the same data base for both volumetric and geometric objects, and anti-aliasing methods appeal to both data types.
Stereoscopic display routines have been added to obtain true three-dimensional visualization on various media, and animation features allow generation of recordable 3-D sequences.

Keywords: Scientific Visualization, Medical Imaging, Volumetric Data, Geometric Data, Hybrid Data, Volume Rendering, Surface Rendering, Hybrid Rendering, BSP-Trees, Stereoscopic Visualization.

1 Introduction

Progress in scientific visualization and in medical imaging allows more complex views of interior characteristics and hidden structures of objects than any other technique before. Modern tomography, such as CT, MRI *(Magnetic Resonance Imaging)*, and ultrasonic scans, deliver large data sets with high accuracy. Advanced technologies are used to explore and evaluate such complex data sets without loss of detail.

Applications in cancer recognition, radiological therapy planning, surgical operation planning, minimal invasive surgery (MIS), etc. require precise visualization techniques to locate

desired elements in a data set. Using computer tomography, it is possible to scan a human body slice-by-slice to obtain an overview of interior structures (figure 1).

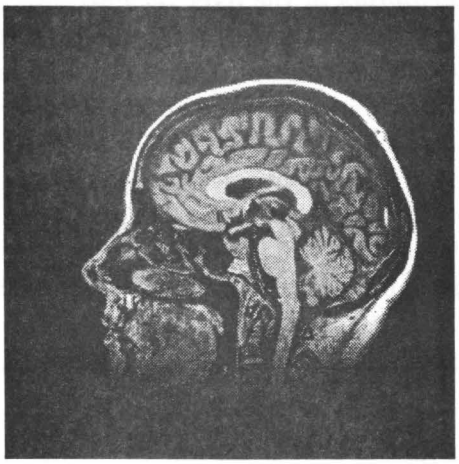


Figure 1: *Slice from MRI scan*

If all slices are put together again and arranged in a regular grid, we can build up a volumetric data set, which consists of an array of 2-D slices. Regarding the nodes carrying a data value, we have a three-dimensional arrangement of volume elements *(voxels)*. Figure 2 emphasizes the reconstruction of a volume from an array of slices.

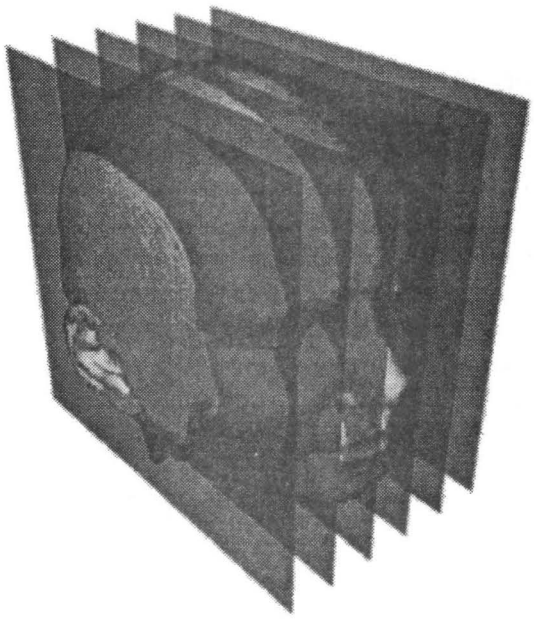


Figure 2: *Reconstruction from array of slices*

Based on the data delivered by the scanning device the volumetric data set consists of voxels with a scalar value attached to each vertex. To obtain a more differentiated visualization, it is

necessary to classify this raw data into different materials, which can be visualized separately. Optical properties can be assigned to each material to simulate transparent skin or bone for example. This process is called *classification.*

Therefore we need a *segmentation* of the data set into several classes of materials. This can be done using edge detection algorithms and thresholding techniques. Since there is no uniform correspondency between scalar values, e. g. obtained from a CT scan, and materials, classification to some extent must be done interactively, but segmentation may be assisted by computer graphics methods.

To enable image processing on small machines not capable of storing the whole volumetric data set in system memory, we have developed a technique to render these data sets in several stages. A volumetric data set is subdivided in one direction into blocks of voxels. Each block consists of two neighboring slices that make up a two-dimensional array of voxels (figure 3, [6]).

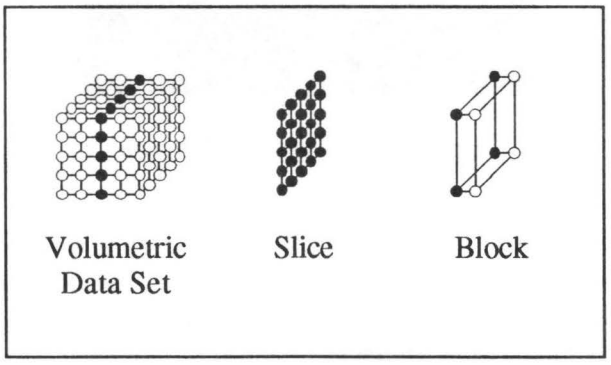


Figure 3: *Volumetric data set*

To support a greater variety of objects, we added geometric objects to the scene. These objects can consist of triangles, polygons, spheres, or, more commonly, CSG objects. The program provides an extendable interface for adding new object classes. Geometry can be used to create additional objects, or to remove some parts of the scene (e. g. volumetric data). An object can be defined as a cutting object that only allows specific materials in it's interior. If the object is completely transparent, we can create an invisible cutting object. Thus we have implemented a universal method for cutting operations on volumetric data.

Similar to volumetric data, we use a subdivision technique to partition geometric data into several buckets. If volumetric data is present in the scene, we use the same dimensions from the block structure to define the bucket size for geometric objects. Subdivision is especially useful when running a distributed version of the program on several machines. Each machine only requires those parts of the scene that appear in a specific part of the image. The size of image portions depends on each machine's computing power.

We use ray-casting techniques to render the scene [7]. In contrast to parallel projection methods [12], which require a complete resampling of the whole scene if the camera position changes, a lot of preparation can be done in a pre-processing stage. Results, such as evaluation of system parameters, hierarchical subdivision, shadow maps, etc., can be reused for each frame of an animation sequence. Besides, resampling always implies loss of detail and diminuition of image quality.

To render volumetric data as well as geometric objects, we need a method for *hybrid rendering* of both data types. Existing methods often fail in preserving image quality due to

necessity of data conversion. The method presented in this paper works on an unaltered set of original data and avoids conversion of raw data.

2 Hybrid Data Sets

Different rendering methods have been established and widely studied in the last decades ([12], [13], [14]). The most researched techniques are *volume rendering* and *surface rendering*. Both methods are limited to a specific type of data, either volumetric or geometric data [17].

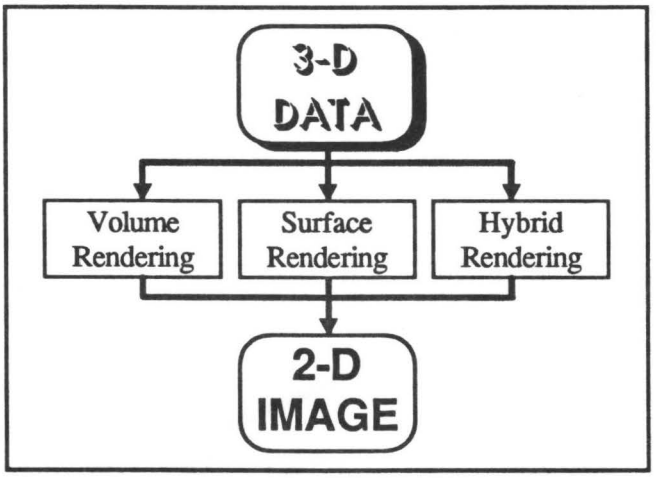


Figure 4: *Rendering methods (alternatives)*

If hybrid visualization [15] is required, it is necessary either to convert geometric data into a volumetric representation using scan conversion algorithms (figure 5), or to extract surfaces from volumetric data to change them into geometric objects (figure 6). Several kinds of algorithms for surface extraction have been published, such as edge detection (iso-contouring), iso-surfacing, and "Marching Cubes", which depends on thresholding [11].

All these methods require a conversion from one data type into the other, causing a loss of accuracy [10]. While positioning geometric data into a volumetric grid (figure 5), the algorithm applies a discrete scan conversion to the original object. This results in a loss of details due to discrete sampling, and requires approximations, if data does not fit exactly into the regular volumetric grid.

On the other hand, if volumetric data is to be converted into a geometric representation (figure 6), a surface reconstruction algorithm must be used to create geometric information, causing problems if there is no real surface information inherent in the data set, or if weak contrasts disable thresholding methods from finding a valid surface [4].

Therefore we introduce a new approach to render volumetric and geometric data simultaneously using the same visualization techniques. The method applies to partial data sets and does not require to keep the whole data set in system memory. Simultaneous visualization of different kinds of data sets is called *Hybrid Rendering*. The main idea is to leave both data sets in their original representation during all preprocessing stages, apply the same projection and transformation algorithms to both data sets, and render them simultaneously.

For each pixel, a ray is cast into the scene, scanning volumetric data as well as geometric data. We use a hierarchical subdivision algorithm to accelerate the scanning procedure,

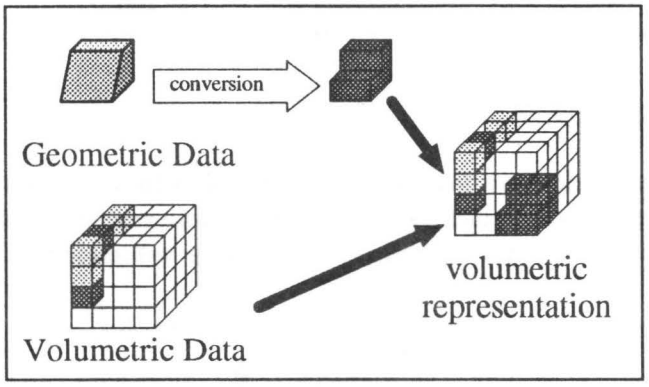


Figure 5: *Volumetric representation*

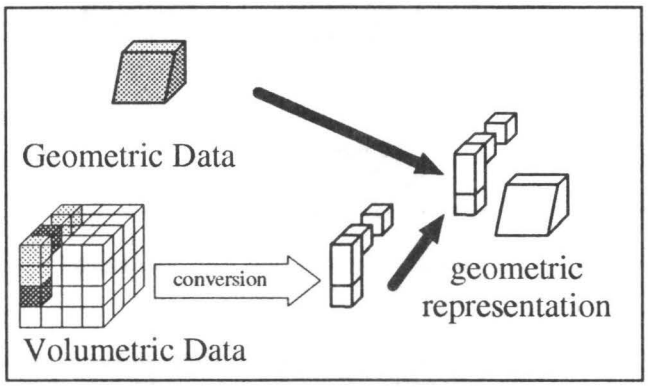


Figure 6: *Geometric representation*

introducing a BSP-tree structure for hybrid data sets (see chapter 3.5). Volume data is sampled at equidistant sampling positions along the ray, and geometric data is inserted at the intersections of geometric surfaces and the ray. A list of sampling positions for volumetric and geometric data is created and sorted in ascending order (figure 7).

At each sample position, we read a material label, which is assigned to both volumetric and geometric data. Different interpolation methods, which can be selected using interpreter commands, are used to determine the best volume label closest to the sampling position. Since we have a common representation now, we apply the same visualization techniques to both volumetric and geometric data.

Our hybrid renderer evaluates the corresponding optical properties at each sampled position from a data base and applies one of several shading models to calculate pixel intensities. The same data bank is used for both types of raw data, and the same optical properties can be assigned to volumetric and geometric data. The combination of both data types is called *hybrid data.*

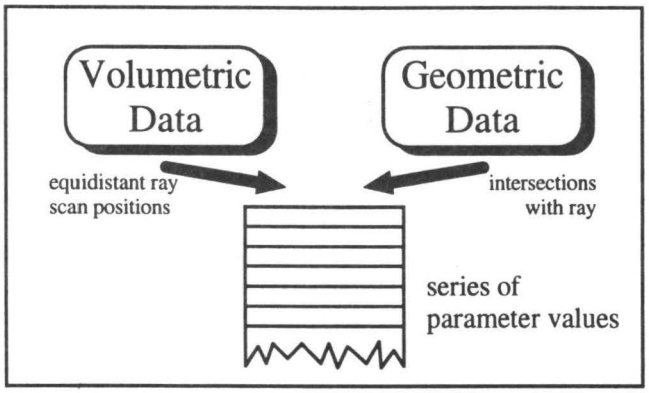


Figure 7: *Hybrid rendering: combination of volumetric and geometric data*

3 Hierarchy of the Scanning Procedure

The hybrid raytracer is controlled by an interpreter that allows us to modify various parameters and to initiate the visualization procedure. It enables us to define data structures to store points, vectors, counters, etc., and it supports control structures similar to high-level programming languages (e. g. *'C'*) in order to create conditional statements or loops for animation sequences.

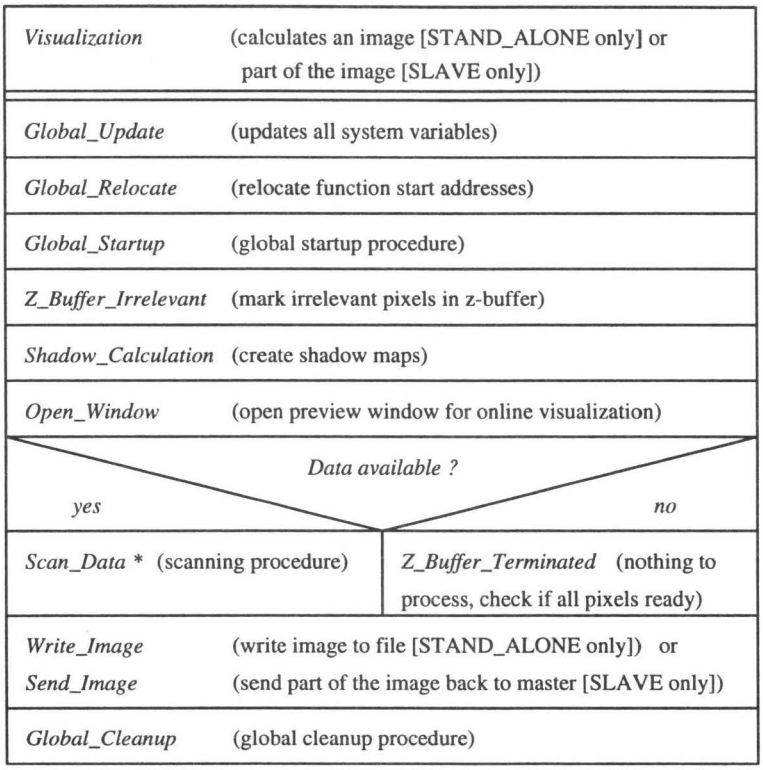
Most of the commands control parameters for the rendering stage, such as camera position, light sources, optical parameters for material properties, shading models, and shadow calculation. There are other commands to determine the output image format, size and quality. The image can be shown on a flat display or as a stereoscopic view. Different display technologies for mono and stereo animations are supported. Finally, we added some commands to control memory management, which is especially important for the distributed version of the hybrid raytracer. All parameters are preset to default values and can be modified any time using easy-to-handle interpreter instructions.

3.1 Visualization Procedure

When all parameters are set as desired, the visualization procedure is called as an initial stage of the procedural hierarchy. First, this procedure checks if any parameter has changed since the last visualization and updates all system variables, if necessary. Sometimes, if the scene changes completely, complex recalculations must be executed, but most of the time values from prior stages can be used.

A z-buffer is used to store color values, transparencies, and other values which must be preserved for subsequent passes. After the relocation procedure, which is used for the distributed version only, and after initialising some local variables, which is done by a procedure that globally checks all modules for initialization services (similar to the *Global_Update* and *Global_Startup* procedure), we can prepare the z-buffer by marking all irrelevant pixels that need not be calculated due to selected scanning modes (e. g. interleaved scanning with interpolation), or because they are covered by a non-transparent logo insert. This feature has been included in order to speed up the visualization procedure.

In the precalculation stage we create shadow maps for fast and efficient shadow generation. A preview window can be opened, if desired, to monitor the progress of visualization, and then



* for more details see chapter 3.2

Figure 8: *Visualization procedure*

we start the scanning procedure (see chapter 3.2). If data is available, the image is calculated and written to a file. In the distributed version only a part of the image is calculated by each slave process and must be sent back to the master process to rebuild the whole image. A global cleanup procedure follows the rendering stage to free unused system resources. Figure 8 shows a flow chart for the main visualization procedure.

3.2 Scanning Procedure

The scanning procedure first determines the appropriate scan direction, which depends on camera position and viewing direction. Since volumetric data is arranged in slices, blocks of data must be loaded in the appropriate order (left-to-right, center-to-left and center-to-right, or right-to-left).

For example if the camera is positioned left to the data set, the scanning procedure first needs the left-most slice to be loaded into system memory, and then continually reloads subsequent blocks of data. It is necessary to keep at most four slices of raw data in system memory to permit interpolation between neighboring voxel fields. When new data is loaded, all slices are shifted one position to the left in a buffer that contains four slices at each time. Remaining slices on the left are removed from system memory and replaced by new ones on the right. Figure 9 shows the structure of the scanning procedure.

Scan_Data	(scan hybrid data)
Hybrid_Box	(calculate bounding box for hybrid data)
Loading_Direction	(determine scan direction, depends on camera position relative to bounding box of hybrid data)
while	
*Block_and_Bucket_Management***	(scan blocks, depending on scan direction)
Z_Buffer_Terminated	(check if all pixels ready)
! ready	(any pixels left ?)

** for more details see chapter 3.3

Figure 9: *Scanning procedure*

3.3 Block and Bucket Management

Hybrid rendering requires handling of two different data types, i. e. volumetric and geometric data. As mentioned before, volume data is arranged in blocks, and similarly geometric data is arranged in buckets. A management routine is necessary to coordinate the different areas in object space.

This routine initially determines the positions where volumetric and geometric data commences and then enters a loop over the whole hybrid data set. If data of one of both types is present, the appropriate block or bucket is loaded into a system buffer and scanned using the hybrid scanning routine. Figure 10 illustrates block and bucket management.

Block_and_Bucket_Management	(scan blocks, depending on scan direction)
Init_Position	(calculate start position for volumetric and geometric data)
do	
First_Block	(get first block[s] of volumetric data, if necessary #)
Sort_Geometry	(sort geometric data, depending on scan direction, if necessary #)
First_Bucket	(get first bucket, if necessary #)
Hybrid_Scanning ***	(scan region of hybrid data)
Next_Block	(get next block, if necessary ##)
Next_Bucket	(get next bucket, if necessary ##)
while (! last_block_or_bucket)	(any blocks or buckets left ?)

\# data valid and start position reached
\## data valid and region not empty
*** for more details see chapter 3.4

Figure 10: *Block and bucket management*

3.4 Hybrid Scanning

A region of hybrid data contains areas of volumetric and geometric data respectively, which may be congruent, disjunct, or overlapping. A block of volumetric data or a bucket of geometric data is projected onto the screen, and the bounding box of the convex hull is used to determine the scanning region. Two different 2-D polygons are calculated to cover these regions.

The polygon scanning routine scans these polygons for volumetric and geometric data. Before scanning, data is divided into subregions using a BSP-tree structure *(BSP: binary spaced partion)* to avoid time-consuming search processes and scanning of empty regions. With this method, hybrid data is prepared for more efficient scanning. The BSP-tree subdivision method is described in more detail in the next paragraph.

If there is only one type of data (volumetric *or* geometric data), the polygon scanning routine has to scan a single polygon only. If there is hybrid data, the scanning area is subdivided into regions of volumetric and geometric data respectively as well as a hybrid region, where both types overlap. To ensure fast and correct scanning, the polygon edges are generated using a modified Bresenham algorithm. Up to 9 regions are scanned line by line, following the polygon edges (figure 11).

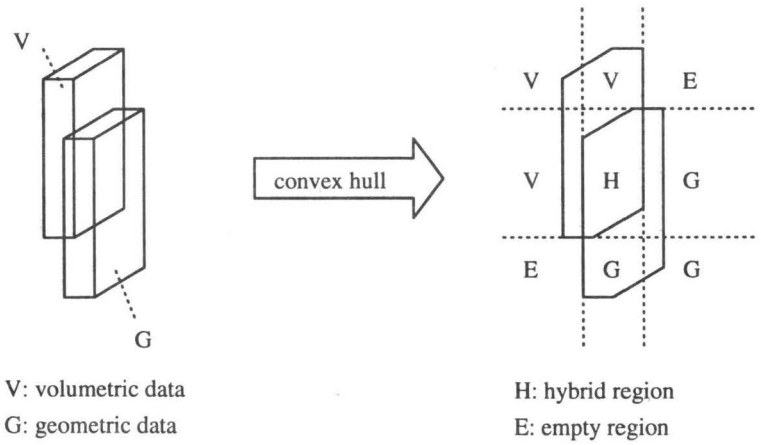


Figure 11: *Scanning regions*

Figure 12 shows the hybrid scanning procedure. When all regions are scanned using the BSP-tree partitioning method, the BSP-tree can be erased from system memory and the next block or bucket is loaded.

3.5 BSP-Tree: A Subdivision Algorithm for Hybrid Data

The BSP-tree subdivision algorithm recursively partitions a hybrid data set into regions of lower complexity. It alternatively subdivides the data set in horizontal and vertical direction and reduces the number of data elements in each specific region [9].

The subdivision process is arranged as a binary tree. The root contains a description of the whole data set of a specific area, i. e. the current bucket for geometric data, and the block boundary indices for volumetric data. The left successor denotes the upper or left half of the data set, and the right successor describes the lower or right part.

Hybrid_Scanning	(scan hybrid data)
Create_Vol_Polygon	(calculate 2-D polygon for volumetric data, if necessary ##)
Create_Geo_Polygon	(calculate 2-D polygon for geometric data, if necessary ##)
Merge_3D_Box	(calculate 3-D bounding box of hybrid data)
Create_BSP_Tree ****	(create BSP-tree of hybrid data)
Polygon_Scanning	(scan polygons for volumetric and geometric data)
Delete_BSP_Tree	(remove BSP-tree)

data valid and region not empty
****for more details see chapter 3.5

Figure 12: *Hybrid scanning*

Starting from the root, data is subdivided recursively. To simplify matters, we choose the center of a rectangular area for intersection, but it is also possible to use an adaptive method, depending on the amount of data remaining in the two subfields. Advanced partitioning methods are currently under development. Since we have binary classified materials for volumetric data it is possible to determine whether a voxel is filled with relevant data, which has been selected for visualization, or if it is empty. Two slices that make up a block are projected on one of the side walls of the block using a binary *'or'* function. Each region is checked whether it contains data or if it is empty. Non-empty regions are subdivided recursively until one region is empty. Processing both regions, the procedure enters a recursive loop to build up a tree. Recursion terminates if a pre-defined depth is exceeded or a minimum size of a subregion is reached (figure 14).

It is possible to use the same subdivision algorithm for geometric data. A list of all geometric objects of the current bucket is attached to the root. All these objects are checked if they come under the left or the right part, or both. When both successors are complete and two new lists are generated, the previous list may be deleted to save memory, and recursion continues with the next level.

Figure 13 shows the structure of the BSP-tree subdivision algorithm. The object from the root list is contained in region A as well as in region B. The next recursive step shows that after further subdivision region 3 is empty.

When scanning regions of hybrid data using the Hybrid Scanning routine, a ray must be calculated from the camera position through each pixel of the scan region (see previous chapter). Different anti-aliasing methods have been implemented, such as jittered ray-casting (monte-carlo method), adaptive oversampling and a discrete subsampling method ([4], [8]). Each ray is checked if it intersects the three-dimensional bounding box of a selected portion of hybrid data. Then the scene is traced to determine the pixel color and transparency. A pixel is terminated if all blocks and buckets are scanned, or if a transparency limit is exceeded.

The trace routine uses the precalculated BSP-tree structure to scan a selected part of the scene for hybrid data. For each part of the ray which is defined by the intersection of the ray with the data block the trace routine recursively descends the BSP-tree in front-to back order until it reaches a leaf containing an area of volumetric data and an object list. A two-dimensional DDA algorithm is used for traversing the voxel structure, starting from

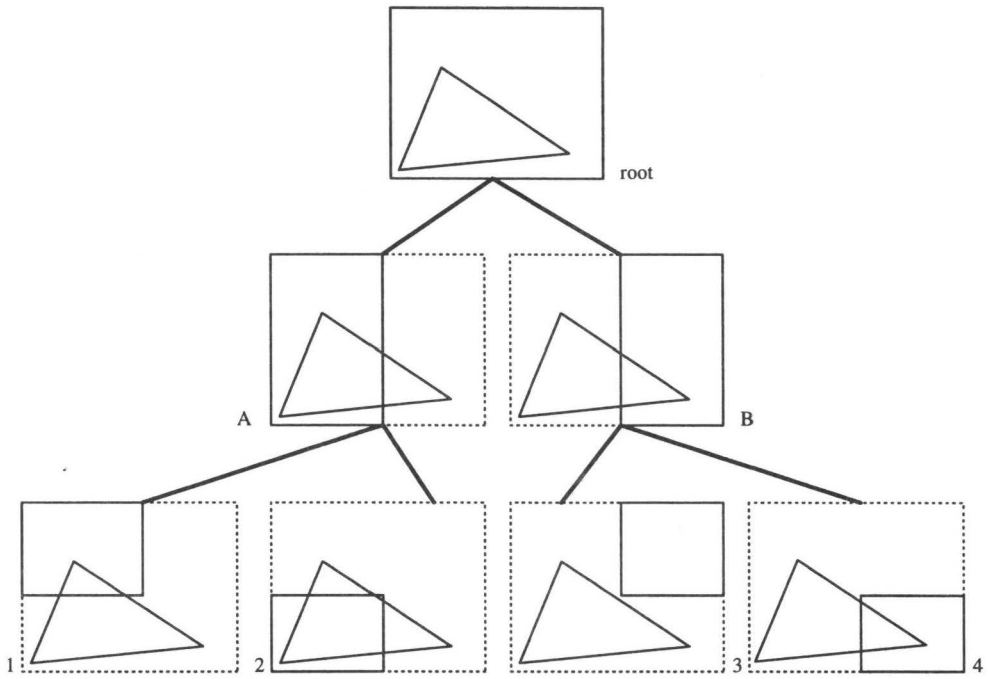


Figure 13: *BSP-tree: subdivision algorithm*

the first intersection between ray and block, and finishing at the projection of the second intersection onto the opposite side wall of the block. This is possible, because we have a block with a dimension of only one voxel in loading direction.

For geometrically defined objects, a list of all objects is created and sorted by camera distance in ascending order. To speed up the visualization procedure the pre-sort algorithm makes sure that those elements are tested first for intersection which appear to be nearer to the camera and therefore are more likely to be seen from the current camera position and not hidden by other non-transparent objects. A memory management routine avoids repeated removal and reallocation of list elements by overwriting existing data and allocating new elements only if necessary.

Spectral light intensity, transparency and texture values as well as shadows are evaluated for each intersection point between ray and geometrically defined object. Similarly for volumetric data the voxel area is sampled at equidistant positions along the ray. For hybrid data an interleaved scanning mode is activated regarding both intersections of geometry and the ray as well as equidistant sampling positions along the ray. Interleaved scanning depends on the order of appearance of intersections and scan positions.

Resulting values are combined with the existing z-buffer entry of the current pixel and stored for the next block to render. If all blocks and buckets are rendered or if a pixel terminates due to insignificant remaining transparency, z-buffer information is reduced to a color value in order to save memory. Also a memory management routine for large z-buffers on tiny machines has been implemented. If system memory is insufficient to keep the whole image, some pixels are marked to be left over for a second pass before a z-buffer entry is fully allocated. This enables us to process large images even on very small machines.

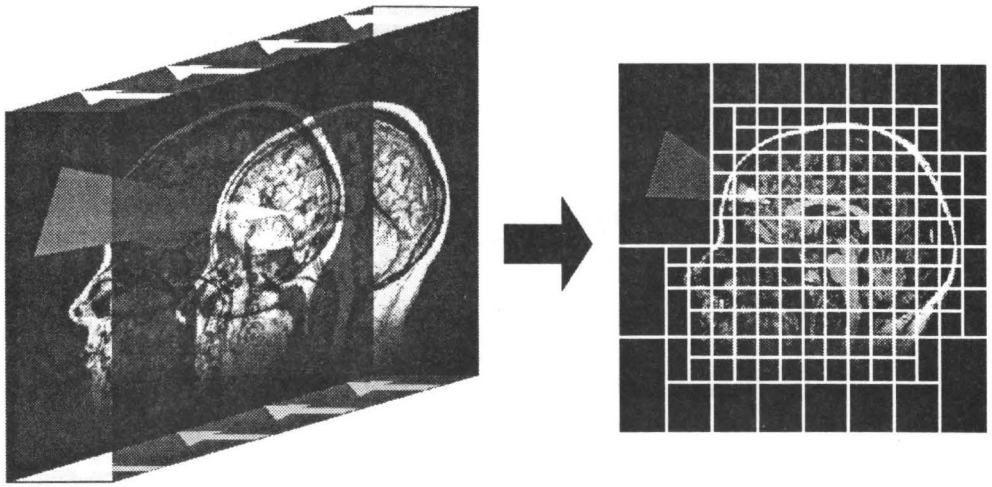


Figure 14: *BSP-tree: 2-D projection of hybrid data*

4 Stereoscopic Visualization of Hybrid Data Sets

One of the most important aims of our work was the extraction of as many features as possible from the the original data sets. To achieve this, we implemented different kinds of algorithms to improve image quality, i. e. several segmentation and classification methods, interpolation in data space, parameters for optical properties that can be assigned to different materials to control their manner of representation in the image, and some animation tools. This helps to explore concealed structures of a data set hidden by exterior layers [5].

Another important feature contained in a data set is depth information. Therefore we added 3-D visualization techniques to our system. Many people concerned with computer graphics consider raytraced images to be 3-D, though it is only a projection from a three-dimensional data set to a 2-D screen. We would call this $2\frac{1}{2}$-D. But real 3-D information can only be obtained if one delivers two separate images, one for each eye. This is called *Stereoscopic Visualization.*

Depth persistance depends on various factors. The brain plays an important role during the interpretation of two images projected onto the retina. In 1838 Wheatstone demonstrated that the mind fuses two planar images into one three-dimensional image. He separated this ability of the brain from other impressions and stated that there is a unique depth sense, called *stereopsis.*

There are a lot of other determinants that make up a realistic three-dimensional image. They can be separated from stereopsis, and are called *extrastereoscopic* or *monocular depth cues.* These cues include perspective, relative size, interposition, depth cuing (graduated reduction in brightness in proportion to the distance from the viewer), and motion parallax [1]. All these features have been implemented, some of them, e. g. perspective and interposition, are included in the raytracing approach, since this method implies solving the hidden surface problem.

Some others are not well suited to medical imaging applications, e. g. aerial perspective (diminution in visibility of distant objects caused by intervening bluish haze, because of the scattering of red light by the atmosphere), depth of focus (not implemented since it only reduces separation of fine details), etc. Additional monocular depth cues like textural

gradients (reduction of textural resolution in proportion to distance) and some algorithms to calculate shadows in an effective way are currently under development.

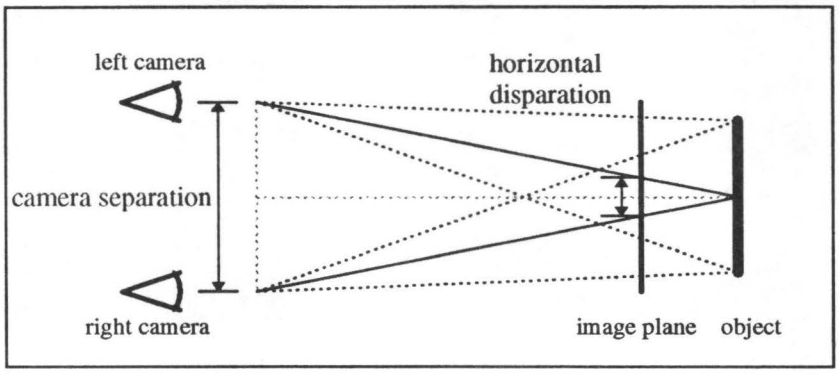


Figure 15: *Horizontal disparation*

To produce a stereoscopic image we have to calculate a pair of images from two slightly different positions (figure 15). Two cameras should be located with a certain offset in horizontal direction, which is called *horizontal parallax* (figure 16). If the object is located exactly in the image plane, we have *zero parallax* (figure 16a). If the object is in *CRT space*, there is no crossing between two rays in front of the screen, and we call this *positive parallax* (figure 16b). Diverging rays (figure 16c) should not be used to avoid viewer's distress. A negative parallax value means that the object is positioned between viewer and screen (*viewer space*, figure 16d).

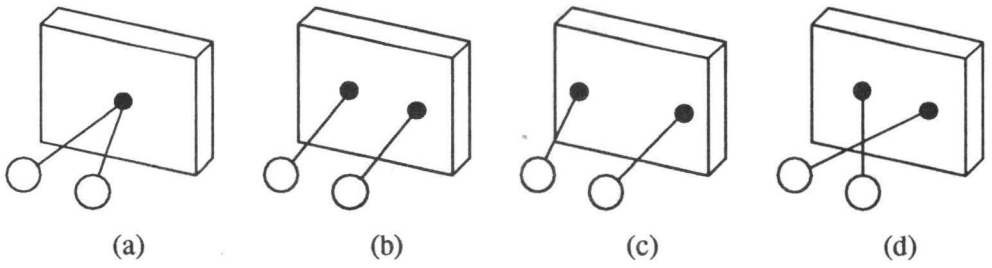


Figure 16: *Horizontal parallax*

Both cameras should point to the same direction, which is called *center of focus*. Each image should be presented only to the eye that it is calculated for. This will be discussed in more detail later.

First we define our viewing system. Let us assume that all objects are given in a world coordinate system. In this case it is defined by the structure of the volumetric data set, which is arranged in a regular three-dimensional grid. The resolution in each of the three orthogonal directions is not necessarily the same, so we scale our basis vectors:

$$\vec{e}_{iW} = \lambda_i \vec{e}_{iV}, \quad i \in \{1, \ldots, 3\} \tag{1}$$

where $\vec{e}_{iV}, \vec{e}_{iW}$ denote unity vectors of our volumetric and world coordinate system respec-

tively. λ_i is the distance between two neighbouring points along a scan axis.

$$B_W = \bigcup_{i=1}^{3} \{\vec{e}_{iw}\} \tag{2}$$

describes a basis for the world coordinate system. Geometric data is defined in the same coordinate system and can be adapted using affine transformations during data acquisition. Different types of geometry may be positioned separately by means of easy-to-handle interpreter commands during rendering stage. Thus we have defined a common world coordinate system for hybrid data sets.

To display data on the screen, we have to perform a projection from 3-D object space into 2-D screen space. This can be described as a basis transformation from B_W into B_S, where B_S denotes a basis for the screen coordinate system. So we are looking for a transformation

$$\varphi_{WS}: \mathbb{R}^3 \rightarrow \mathbb{R}^3, \quad \varphi_{WS}(P_W) = P_S; \quad P_W, P_S \in \mathbb{R}^3 \tag{3}$$

that puts any point P_W (in world coordinates) into P_S (in screen coordinates).

B_S is a left-handed orthogonal coordinate system with x- and y-axis in the screen plane (with the x-axis pointing to the right) and with the origin located in the center of the screen. Points with positive z values are defined to be in CRT space, points in negative z range are signified as existing in viewer's space. This is important for our stereo coordinate system that will be introduced later.

φ_{WS} is derived in several stages. To achieve coincidence between B_W and B_S we have to perform three rotations, one for each axis. Thus we need a common center of rotation, which can be obtained by moving all points in opposite direction to the camera, using the camera as center of rotation. This is described as matrix T_1.

The next step is to append three rotations R_x, R_y, R_z, where R_z can be utilized to rotate the image on the screen by adding a camera rotation angle. R_z can also be used to align volumetric data sets, which is necessary because of different scanning directions. Usually this is compensated using a camera rotation, but the rotation angle has to be recalculated for each new camera position. In particular this is important for animations and for stereoscopic images where we have more than one camera position. To achieve a common alignment we define a vector to be the "up" direction of a data set. If this is impossible due to the case that this is the viewing direction, we use a vector for the "right" direction for correct alignment.

In the last stage we have to move all points at a distance f towards the camera along the z-axis, where f denotes the focal length, which is the distance between the camera and the origin of the screen coordinate system. This is described as matrix T_2. Thus we have

$$M_{WS} = T_2 \circ R_z \circ R_y \circ R_x \circ T_1 \tag{4}$$

as a transformation matrix for φ_{WS} with

$$P_S = \varphi_{WS}(P_W) = M_{WS} \cdot P_W \tag{5}$$

A basis for the screen coordinate system is given as:

$$B_S = \bigcup_{i=1}^{3} \{\varphi_{WS}(\vec{e}_{iw})\} \tag{6}$$

We have only used bijective transformations, so the combination (4) also refers to a bijective transformation, and therefore the inverse transformation exists, too:

$$\varphi_{WS}^{-1} = \varphi_{SW}: \mathbb{R}^3 \rightarrow \mathbb{R}^3, \quad \varphi_{SW}(P_S) = P_W; \quad P_W, P_S \in \mathbb{R}^3 \tag{7}$$

This is the transformation from screen coordinates into world coordinates.

Now we calculate the positions for the stereo camera. To obtain stereo vision two cameras have to be located according to the natural distance between the two eyes. Since we have only world and screen coordinates we introduce a new coordinate system, called *stereo coordinate system*, to describe all 'hardware-dependant' or 'exterior' measures that can not be expressed in world coordinates. In this metric system we define the distance between the eyes, the viewing distance (between viewer's eyes and screen surface), and the screen size. Also we can set the position of the object in depth relative to the screen surface. All these values may be given in centimetres, inches, or any other unit. Additionally we need the screen size in pixels for reference.

To define a stereo coordinate system for animations it is necessary to keep the start position and direction of the camera fixed in order to maintain a constant impression of depth. These values together with an angle that describes the viewing area can be kept constant throughout the animation. They are used as a reference for further camera movements.

After defining the stereo coordinate system using an interpreter command we calculate a new *stereo transformation* to replace the standard camera transformation. The stereo transformation provides two projection matrices, one for each eye. It is derived from the camera transformation in several stages, which are described later.

First, we have to determine the two positions of the stereo camera. They are derived from a horizontal shift of the mono camera in horizontal direction, symmetrically to the original position. A translation to the left and to the right (x-direction of the viewing system) is necessary according to the natural separation of the eyes. We use the screen coordinate system to perform this translation, because it has the same alignment as the stereo coordinate system. So the mono camera position and direction have to be transformed from world into screen coordinates first.

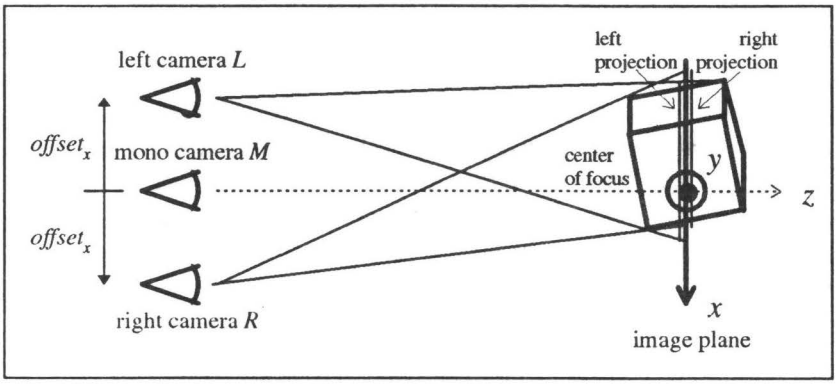


Figure 17: *Stereo coordinate system*

Both cameras have to be moved to the left and to the right respectively with a certain offset (figure 17). Therefore the distance between the eyes, which is given in stereo coordinates, has to be converted into screen coordinates.

$$offset_x = \frac{screen\_width[\text{pixels}]}{screen\_width[\text{units}]} \cdot \frac{eye\_distance[\text{units}]}{2} \tag{8}$$

The average distance of the eyes is 2.5 inch (6.4 cm), and usually the viewing distance is approx. 30 inch (76 cm). We only need the screen width in units of the stereo coordinate

system and in pixels. The height is irrelevant due to the fact that we only consider horizontal parallax.

The *object distance* determines the position of the object in relation to the screen surface at the beginning of the animation. This value is also defined in stereo coordinates. Thus we have the following conditions:

- *object_distance* > 0*:*

  The object is behind the screen. M is more apart from the origin; if the viewing angle is constant the object appears to be smaller.

- *object_distance* $= 0$*:*

  The object is located in the screen plane. Since the objects of the scene usually have three dimensions, this condition is valid only for some points on a plane orthogonal to the z-axis of our screen coordinate system. So we use the point that describes the viewing direction as a reference point. It is recommendable to choose the center of the hybrid data set as a reference point to make sure that some parts of the scene are in front of the screen, and some parts are behind.

- *object_distance* < 0*:*

  The object is in front of the screen *(viewer space)*. M is nearer to the origin; if the viewing angle is constant the object appears larger on the screen.

In the last case the follwing condition must be true:

$$(-object\_distance) < viewing\_distance \tag{9}$$

We discovered that it is advisable to restrict the object distance according to the equation:

$$|object\_distance| < \frac{viewing\_distance}{3} \tag{10}$$

Otherwise, horizontal disparity could exceed the range in which the brain fuses two images. This happens due to a breakdown of the accomodation/convergence relationship. Other sources recommend to use an object distance within -25% and +60% of the viewing distance [2]. But these values strongly depend on experience and physical constitution of the viewer.

The object distance has to be converted from stereo coordinates into screen coordinates. The camera offset results from

$$\frac{offset'_z}{D_z - M_z} = \frac{object\_distance[\text{units}]}{viewing\_distance[\text{units}]}$$

$$\Longrightarrow offset'_z = \frac{object\_distance[\text{units}]}{viewing\_distance[\text{units}]} \cdot (D_z - M_z) \tag{11}$$

where D denotes the point that describes the viewing direction. M_z is the z-coordinate of the camera.

During an animation sequence the distance between M and D or the viewing angle can be modified. This causes a magnification or reduction of the objects that are displayed on the screen. In this case the stereo projection has to be readjusted to prevent the object from remaining at a fixed distance from the screen. If not, the object would stay in a stationary position and only changes size.

As a solution we could change the object distance using the interpreter command, but it would be hard to coordinate this with the camera movement. An improved method to adapt the object distance automatically is to associate the offset along the z-axis with the camera movement. Therefore we scale the depth offset:

$$offset_z = offset'_z \cdot t_1 \cdot t_2 \tag{12}$$

where t_1 refers to the relation between the current camera position and the start condition:

$$t_1 = \frac{\|D - M\|_2}{\|D_0 - M_0\|_2} \tag{13}$$

D, M denote the point of the viewing direction and the camera position; D_0, M_0 refer to the start condition, which is valid at the beginning of an animation sequence. D_0, M_0 can be described as a reference for further camera movements.

Additionally, if the viewing angle changes, the focal length has to be adapted, too. The appropriate factor is:

$$t_2 = \frac{f}{f_0} \tag{14}$$

The values for D_0, M_0, and b_0 remain constant during an animation sequence and can be initially defined using the interpreter command that controls the stereo coordinate system.

Now, for the position of the stereo camera we have:

$$L = M + \begin{pmatrix} -offset_x \\ 0 \\ -offset_z \end{pmatrix}, \quad R = M + \begin{pmatrix} offset_x \\ 0 \\ -offset_z \end{pmatrix} \tag{15}$$

Finally, we recalculate the matrices for the transformation from world into screen coordinates and vice versa. To transform a point from the world coordinate system to the screen coordinate system we use the mono camera transformation matrix first. Then we move this point to the previously calculated distance $offset_z$ in z direction. The corresponding matrix is:

$$S_1 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & offset_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{16}$$

The translation in x direction depends on the distance of the point which has to be transformed from the origin of the screen coordinate system. This is equivalent to a shear transformation (figure 18).

Thus we have the following matrices:

$$S_{2_L} = \begin{pmatrix} 1 & 0 & \frac{-offset_x}{f} & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad S_{2_R} = \begin{pmatrix} 1 & 0 & \frac{offset_x}{f} & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{17}$$

Altogether we get the following matrices for our stereo transformation:

$$L_{WS} = S_{2_L} \circ S_1 \circ M_{WS}, \quad R_{WS} = S_{2_R} \circ S_1 \circ M_{WS} \tag{18}$$

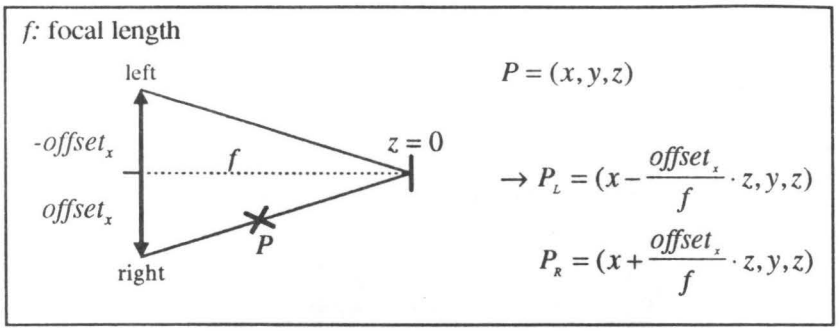


Figure 18: *Stereo projection: shear transformation*

Similar to equation 4, we can also calculate the inverse transformation L_{SW} and R_{SW} respectively. These new stereo camera positions and transformations are stored for further reference in order to switch between left, right, and mono camera intermediately.

We have defined a new stereo coordinate system which makes calculation of stereo perspectives independant from the size of objects in the scene. Visualization of molecular structures for example requires other parameters than objects of the real world. It is not very useful to use the natural distance of the human eyes as a basis for a calculation to obtain a stereo image. The virtual eyes have to be positioned in the scene so that they have dimensions in the same range as the objects in the scene. The same is true for astrophysical visualizations, where there are much larger dimensions. Our systems allows us to calculate stereo perspectives with no respect of the world coordinates defined in the scene. We only use parameters derived from hardware geometry and viewing habits ([18], [19]).

5 Results

Different display methods have been implemented for stereoscopic visualization. A standard method for 3-D reproduction is the combination of two slides into one anaglyph image. Both fields are converted to gray scale images following CIE[1] recommendations, and then superimposed as colored frames, one in red, and the other one in green. Special glasses with tinted plastic lenses in red and green must be worn to view these images.

Another method to create anaglyph images is to leave out the conversion to gray, and directly split the two fields into colors. For example if the red channel is used from the left image, combined with blue and green from the right image, as a result there will be a full-color image in those portions of the stereo image that are similar in both fields, and anaglyphic colors at the object borders. This does not disturb stereo impression too much, and the brain is able to recognize a three-dimensional color image. The object should have the same brightness in both color extracts, otherwise it would disappear in one of the two fields.

Two methods for black-and-white reproduction have been implemented, too: *wide-eyed stereo* and *cross-eyed stereo*. With a little practice it is possible to view both of them without special glasses. Depending on the image type, the eyes must be crossed in front or behind the image plane. Although the fix point is not in the image plane, the eyes must be focussed

[1]Commision Internationale L'Éclairage

on the paper. For a small percentage of people this is hard to achieve due to the accomodation/convergence breakdown, but with the help of optical devices it is much easier to view a wide-eye stereogram. Figure 19 shows a historic model of a stereoscope which was used to view stereo slides (paintings and photographs).

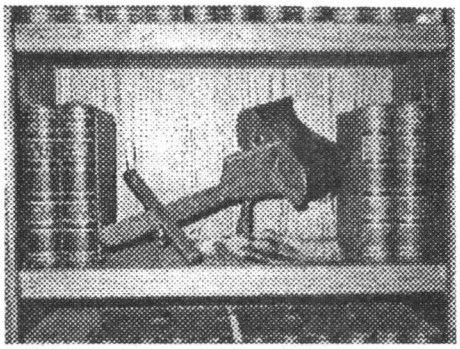


Figure 19: *Stereoscope*

A stereoscope uses lenses or mirrors to align the center of focus to the image plane, so that the eyes can watch two separate fields from a short distance. In 1849 David Brewster introduced a lenticular stereoscope [3]. Figures 20 and 22 show the principles of wide-eye and cross-eye stereograms. The examples (figures 21 and 23) show a human head with removed skull cranium. A geometric object has been used to define the intersection.

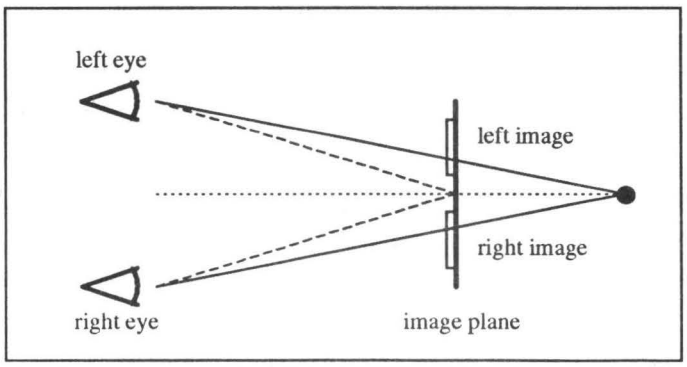


Figure 20: *Wide-eyed stereo (principle)*

All the methods mentioned above are well suited for printing, but we also developed some techniques for electronic media, such as shutter glasses, and our hybrid raytracer allows video recording of stereoscopic animation sequences.

Shutter technology requires a pair of LC *(liquid crystal)* glasses that can be controlled electronically. Both fields are displayed alternatively on the screen, and if one image is displayed, the shutter glasses open a window synchronously only for the corresponding eye, while the other eye is darkened. If the frequency is high enough (60 - 120 Hz), a flicker-free three-dimensional image can be seen.

The same technology not only applies to still frames, but also to animations. Special

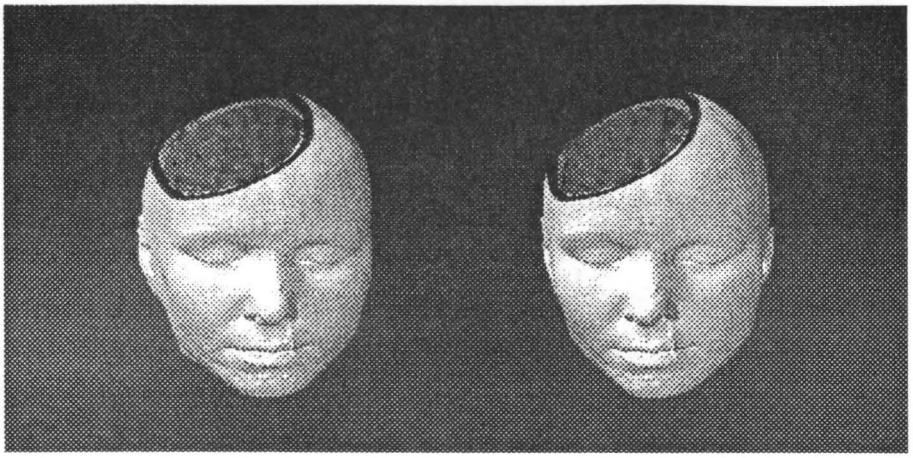


Figure 21: *Wide-eyed stereo image*

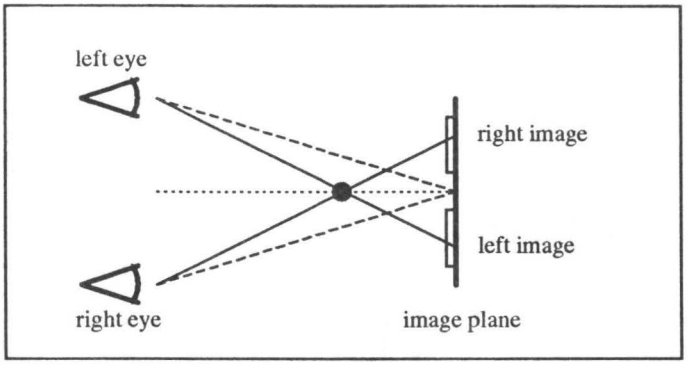


Figure 22: *Cross-eyed stereo (principle)*

efforts have been made to develop a hardware-independant interface for storage of animation sequences on video disks or tape and for replay on every CRT or projection screen. This is one of the best techologies for three-dimensional displays known so far, since it allows full-color reproduction of animated 3-D sequences.

Finally, another method for three-dimensional reproduction has been tested: autostereograms [16]. Although it is not so well suited for medical imaging applications, it allows to display depth information in a single frame without any special viewing devices. To create an autostereogram, both fields are rendered separately and projected onto the image plane. Pixels originated from the same object point obtain the same color or intensity, so that the brain is forced to fuse these points. The color of the pixel is irrelevant, it only must be the same for corresponding pixels. A random dot pattern or a colored texture may be used as a reference for assigning pixel values. If more than one object point is projected onto the same pixels, all pixels obtain the same color. Since we only consider horizontal disparity, the texture map width should be equivalent to the distance of two pixels, if a background point

Figure 23: *Cross-eyed stereo image*

is projected onto the image plane (equation 19, figure 24).

$$\frac{o}{d} = \frac{o+v}{e} \Longrightarrow d = \frac{o}{o+v} \cdot e \tag{19}$$

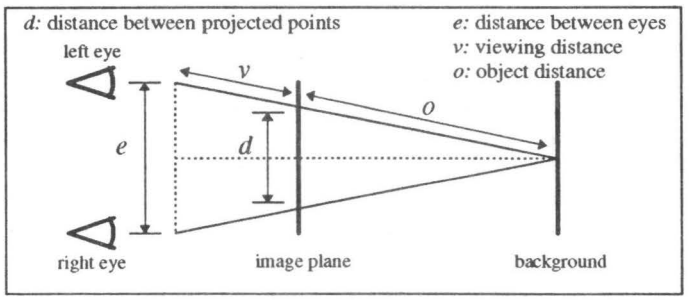


Figure 24: *Autostereogram (principle)*

Figure 25 shows a random dot stereogram of a human head. The depth information is taken immediately from the z-buffer. If you have no experience in free-viewing autostereograms, put your nose on the paper, focus behind the image, and then move your had slightly backwards, until you see sharp contours of the object.

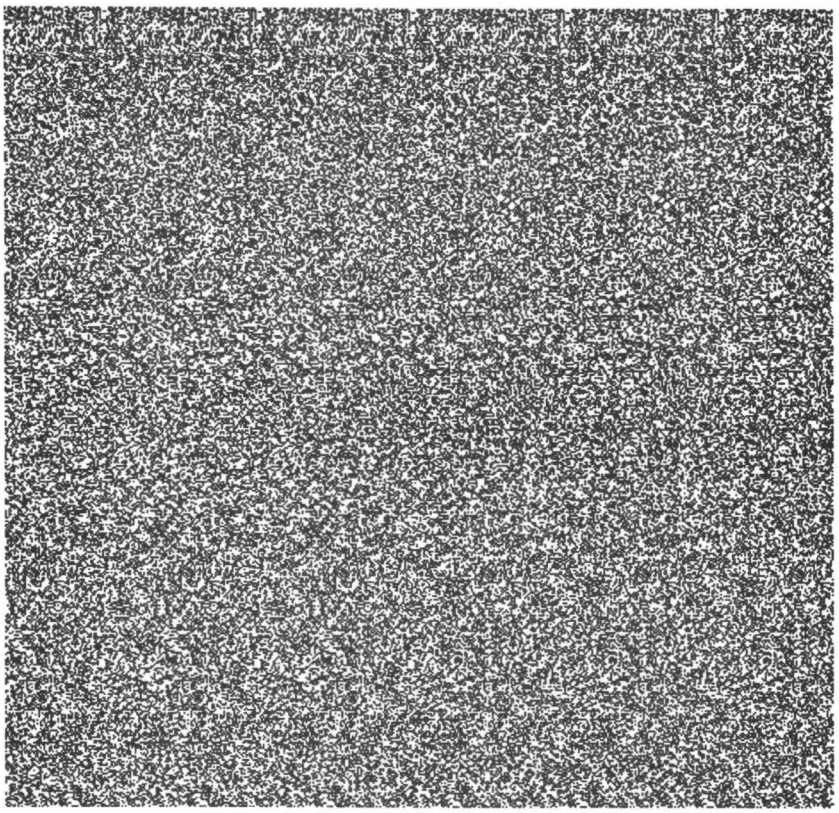


Figure 25: *Autostereogram of a human head*

References:

[1] Lipton, Lenny: *The Crystal Eyes Handbook*; StereoGraphics Corporation, San Rafael, CA, 1991

[2] Williams, Steven P.; Parrish, Russel V.: "New Computational Control Techniques and Increased Understanding for Stereo 3-D Displays"; *Proc. SPIE Vol. 1256*; Stereoscopic Displays and Applications, Washington, 1990

[3] Considine, Douglas M.: "Stereoscope"; *The Academic Encyclopedia (Electronic Version)*; Grolier, Inc.; Danbury, CT., 1992

[4] Meyer, Jörg: "Hybrider Raytracer mit Anti-Aliasing"; University of Kaiserslautern, Germany, 1993

[5] Meyer, Jörg: "Dreidimensionale Visualisierung und Animation hybrider Datensätze - Algorithmen und Displaytechnologien"; *master thesis*; University of Kaiserslautern, Germany, September 1994

[6] Lengen, Rolf H. van: "The Volume Priority Z-Buffer"; *Focus on Scientific Visualization*; Hagen, H.; Müller, H.; Nielson, G. M. (publ.), Springer Verlag, New York, October 1992

[7] Röhrig, Roger: "Theoretische und empirische Ansätze zur Visualisierung medizinischer Datensätze"; *master thesis*; University of Kaiserslautern, Germany, Juni 1994

[8] Heinrich, Stefan; Keller, Alexander: "Quasi-Monte-Carlo Methods in Computer Graphics, Part 1: The QMC-Buffer"; *Technical Report*; University of Kaiserslautern, Germany, 242/94

[9] Rodrian, Hans-Christian: "Schnelle Formfaktorberechnung zur Generierung realistischer Schattenverläufe"; *master thesis*; University of Kaiserslautern, Germany, 1993

[10] Frühauf, Martin: "Combining Volume Rendering with Line and Surface Rendering"; *Proc. Eurographics, 2nd European Computer Graphics Conference, North-Holland*; Amsterdam, 1991

[11] Lorensen, William E.; Cline, Harvey E.: "Marching Cubes: A High Resolution 3D Surface Construction Algorithm"; *ACM Computer Graphics Vol. 21*; SIGGRAPH '87, Anaheim, July 1987

[12] Rhodes, Michael L.; Stover, Henry S.; Glenn Jr., William V.: "True Three-Dimensional (3-D) Display of Computer Data: Medical Applications"; *Proc SPIE Vol. 318;* Picture Archieving & Communication Systems (PACS) for Medical Applications, Washington, 1982

[13] Wright, John R.; Hsieh, Julia C. L.: "A Voxel-Based, Forward Projection Algorithm for Rendering Surface and Volumetric Data"; *Proceedings Visualization '92, Boston, Massachussetts*; IEEE Computer Society Press, Los Alamitos, CA, October 1992

[14] Wixson, Steve: "Volume Visualization on a Stereoscopic Display"; *Proc. SPIE Vol. 1256*; Stereoscopic Displays and Applications, Washington, 1990

[15] Levoy, Marc: "A Hybrid Ray Tracer for Rendering Polygon and Volume Data"; *IEEE Computer Graphics and Applications 10 (3)*; 1990

[16] Tyler, Chistopher W.; Clarke, Maureen B.: "The Autostereogram"; *Proc. SPIE Vol. 1256*; Stereoscopic Displays and Applications, Washington, 1990

[17] Mosher Jr., C. E.; Johnson, E. R.: "Integration of Volume Rendering and Geometric Graphics"; *Proceedings of the Chapel Hill Workshop on Volume Visualization*; Chapel Hill, HC, May 1989

[18] Hodges, Larry F.: "Basic Principles of Stereographic Software Development"; *Proc. SPIE Vol. 1457*; Stereoscopic Displays and Applications II, Washington, 1991

[19] Wickens, Christopher D.: "Three-Dimensional Stereoscopic Display Implementation: Guidelines Derived from Human Visual Capabilities"; *Proc. SPIE Vol. 1256*; Stereoscopic Displays and Applications, Washington, 1990