8

Hardware Support for the Display and Manipulation of Binary Voxel Models

G.J. Jense, and D.P. Huijsmans

We describe some of our experiences with the implementation of a 3D reconstruction system for the visualization of the shapes and structural development of biological objects. We use a binary voxel model as volumetric representation of the reconstructed objects.

The manipulation and display of volumetric representations involve the processing of huge amounts of data, making hardware support a virtual necessity. Instead of attempting to design special purpose hardware, we decided to try and exploit readily available image processing hardware.

We use one of the available frame buffers for storage and direct display of the binary voxel data set. The other frame buffer holds either a surface normal view, a depth-shaded pre-image or the binary voxel data set of a secondary object. Altering the light direction or shading function is performed by manipulating the hardware output lookup tables. An additional frame processor is employed for running various filter operators over pre-images, computing bitwise logical functions on two binary voxel data sets and for pan and zoom operations.

1. Introduction

We will first give an overview of some applications for volume visualization and present a brief description of a reconstruction method to obtain three-dimensional object representations from two-dimensional data.

1.1. Application Area

Medical and biology researchers are often interested in the 3D spatial structure of various, sometimes microscopically small, objects. For several reasons, the spatial structure of these objects may not be obtained by direct three-dimensional sampling, but has to be reconstructed from series of parallel, cross-sectional slices through the object. In our case, these slices are physical cuts through the specimen, made *in vitro*. However, input to the reconstruction system may also consist of series of virtual slices, obtained in vivo from MRI or CT scanners, or of images that are acquired through optical sectioning by a confocal scanning laser microscope.

1.2. The Reconstruction Process

The reconstruction process consists of several steps. The first step, input preparation, consists of embedding the specimen in paraffine or plastic, adding reference markers, cutting the slices or *coupes*, putting the coupes on microscope slides and staining them with specific dyes.

The second step is the digitizing process. This may be accomplished in two ways. In the manual input procedure the slides are photographed and the resulting photos enlarged. Subsequently, the reference markers and boundary contours of structural details are traced manually on a graphics tablet.

In the automated input procedure the slides are digitized by a video camera on the microscope. After this, image subtraction, thresholding, area-smoothing and contour detection algorithms are applied to find the boundary contours of the area of interest.

The last step is the actual reconstruction of the 3D information. This involves the closing and smoothing of the boundary contours followed by aligning the slices by the reference markers and building a contour stack [10, 11].

2. Object Representation

This section describes possible representations for three-dimensional objects. After evaluating our experiences with a previous reconstruction and visualization system, we motivate our reasons for choosing a volume description for object representation in a new system.

2.1. Spatial Representations

In this section we will briefly discuss a number of spatial representation schemes. Data structures for the representation of three dimensional objects may be classified, in order of increasing topological dimension of the descriptive elements, as follows.

- Point collection: the descriptive element is a 0 dimensional point.
- 1. Wireframe: built from 0-dimensional elements (nodes) and 1-dimensional line segments (edges).
  2. Contourstack: 0-dimensional elements are chained together to form planar, closed contours, which in turn are stacked in the third dimension.
- Boundary surface representation: the boundary surface of the object is tessellated with 2-dimensional elementary surface patches. These may be either planar, for example triangular or polygonal, or parametrically defined, e.g. Bézier- or B-spline surface patches.
- Volume description: an object is decomposed in three dimensional volume elements. Each volume element may contain information about some attribute (spatial occupation) or physical quantity (density).

Which representation is chosen depends on the desired manipulations that are to be performed on the object, since not all representations are equally suitable for all manipulations [12, 13].

2.2. Earlier Attempts

A previous implementation of a 3D reconstruction system used a contour stack for object representation. After reconstruction, the contour stack may be displayed. First, a geometric transformation is applied to the contours to reflect the specified viewing parameters. Then, the area enclosed within the nested inner and outer contours is re-calculated to serve as a mask for the removal of hidden lines: when the contours are displayed in a back-to-front fashion, the enclosed area masks out the invisible parts of the contours displayed previously.

During the time the system was actually used by embryologists, a number of disadvantages of this representation became evident:

- the original direction in which the object was sectioned remains clearly visible;
- the contour-stack representation is not suitable for numeric analysis, both of surface and volume parameters;
- after reslicing the contourstack along a different direction, the large errors that arise from sectioning and handling the coupes become apparent;
- for a more realistic display than a wire frame, a conversion from contour-stack to either surface or volume representation is needed.

A definite advantage of the contour stack model is that it facilitates manual input procedures.

2.3. Conversion Between Representations

As stated before, the choice of object representation depends on the desired manipulations. An important issue is the conversion between representations. For display purposes, a surface representation would be suitable: such a surface representation may be rendered through a conventional graphics pipeline.

Several algorithms exist to build a tessellation with elementary, mostly polygonal, surface patches, but when planar contours are used as input, these algorithms have difficulties handling topologically non-trivial cases [7]. To avoid this, other surface construction algorithms start from a volume representation [1, 16].

A reasonable approach therefore would be to employ a volume description as main representation and convert from/to either boundary surface representation or contour model when needed.

2.4. Volume Representation

Because of the advantages in the areas of numerical analysis, re-slicing the object in any direction, projection and performing other spatial selections, a volume description was chosen for a new implementation of the reconstruction system. The representation used is a spatial enumeration of elementary volume elements or *voxels*. Each voxel may assume the value 0 or 1 depending on whether it is located outside or inside the object. We call this representation a *binary voxel model*. Conceptually, the binary voxel model of an object is stored as a three-dimensional array of bits.

3. Hardware Description

The image processing hardware consists of two printed circuit boards that plug into an IBM PC/AT backplane. We will give a brief description of the hardware itself and mention some of the traditional image processing applications it was designed for originally.

3.1. Frame Grabber and Buffer

The first board is a Data Translation DT2851 High Resolution Frame Grabber [5]. Figure 1 is a block diagram, showing the most relevant functional units.

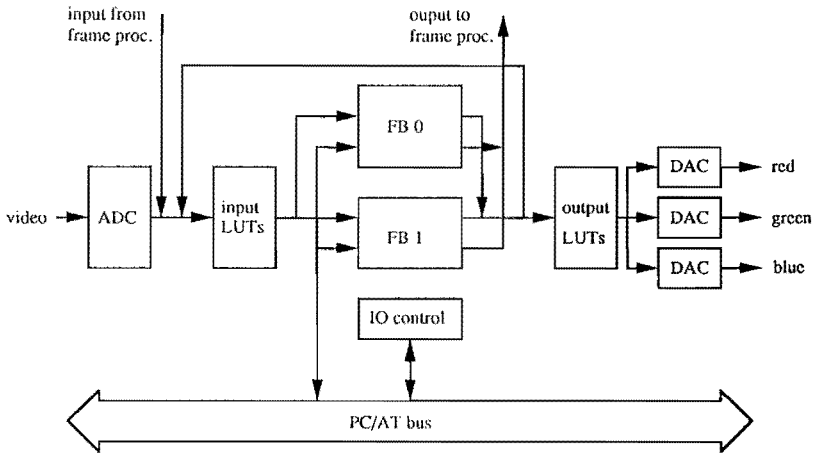


Figure 1: Block diagram of the frame buffer/grabber.

The frame grabber is capable of digitizing analog video signals into 8-bit samples in real time, which in this case means one video frame in $\frac{1}{25}$ of a second. There are eight 256×8-bit RAM input look-up tables, normally used for operations such as input data conversion or thresholding. The digitized images may be stored in one of two 512×512

by 8-bit frame-store memory buffers. The two framebuffers are mapped into the IBM PC/AT memory space and can be accessed using normal memory instructions. Write protection of image data may be enabled for several combinations of bit planes.

A feed-back loop from the output of the frame buffers to the input lookup tables allows image data to be passed through any of these tables and re-written in either frame buffer. This permits several arithmetic or logical functions to be performed on images.

Eight 256×24-bit RAM output look-up tables are available for pseudo-colour or gray-level conversion of image pixels. There are also two external video I/O ports. These ports can be used to transfer data in and out of the framebuffers at video speeds to other image processing devices such as the frame processor that we describe below.

3.2. Frame Processor

The second board is a Data Translation DT2858 Auxiliary Frame Processor [6]. See Figure 2 for a block diagram of this device. The frame processor is connected to the frame grabber via the external video I/O ports.

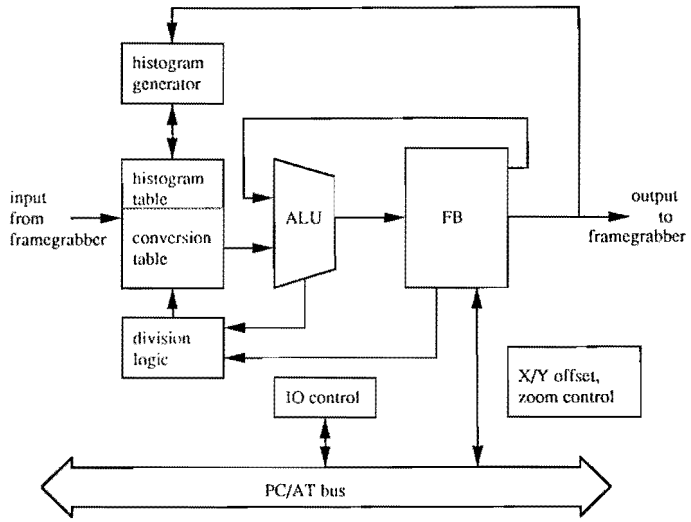


Figure 2: Block diagram of the frame processor.

The six 8-bit to 16-bit look-up conversion tables, are used to perform operations such as pixel offsetting, multiplication or thresholding. For instance, when calculating a convolution, the tables are used to multiply or divide data by a constant.

A histogram generator is capable of calculating a 256-bin by 32-bits histogram of the pixel values in the frame buffer. The resulting histogram is stored in (part of) the conversion table RAM.

The division logic employs a successive approximation method to divide 16-bit numbers from the frame buffer by an 8-bit divisor, giving 8-bit results. A division is performed by the ALU, along with the successive approximation register and the conversion table.

The 16-bit ALU can perform arithmetic and logic operations on both incoming data and data in the frame buffer. The incoming data may be added to, subtracted from, logical ANDed, ORed or XORed with data from the frame buffer. Various combinations of these functions are also possible.

The 512×512 by 16-bit framebuffer stores video frames sent from the frame grabber/buffer, intermediate computational results or image data that are to be sent back to the frame grabber/buffer. Data may also be looped back into the frame buffer via the ALU.

Finally, there are two offset registers controlling the start address for the scan-out sequence of the frame buffer. Programming these registers allows panning and scrolling of images. An important use of these registers is the computation of convolutions of image data with some convolution kernel matrix. Here, in several passes, all pixels of the image are subsequently multiplied by the convolution matrix coefficients and added to the intermediate result. In each pass, the pixels are offset by an amount that corresponds to the place of the coefficient in the convolution kernel matrix.

During scanout of the frame buffer, pixel values may be replicated in both x and y direction either 1, 2, 4 or 8 times. When used in conjunction with the offset registers, this feature allows a general zoom operation on stored images.

Total hardware cost is in the order of $ 10,000, giving an excellent price/performance ratio. In the following section we will describe how we use the hardware in a novel way to support display and manipulation of 3D objects.

4. Using the Hardware

Although originally designed for the processing of two-dimensional images, a number of the manipulations of the binary voxel model as well as several display facilities may be supported by the hardware:

- direct display of the binary voxel data-set
- interactive display of shaded surface views
- running a gradient operator over a depth shaded pre-image
- enlarging and translating the displayed image
- interactive editing of reconstructed objects
- reslicing reconstructed objects along an arbitrary plane
- modelling objects by combining primitive objects through set operations
- calculation of the volume of an object

4.1. Displaying the Data-set

By choosing a special storage scheme for the 3D voxel array, we can display the entire data-set on the screen. A three-dimensional binary array of size 128^3 occupies 256Kbytes of memory. Our image processing hardware has 2 framebuffers of 512×512 8-bit pixels. Each of these framebuffers is therefore also 256Kbytes in size. We use one of these to store the binary voxel model. For this, the 512×512 framebuffer is subdivided in 16 (4×4) sections of 128×128 8-bit pixels. Each section holds 8 consecutive slices through the object.

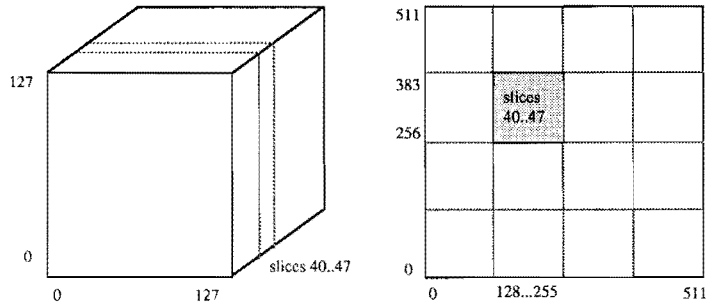


Figure 3: (a) 3D binary voxel array; (b) framebuffer storage scheme.

Because of spatial coherence, the changes between consecutive slices are only small. Therefore, when the content of the framebuffer is displayed, the screen shows 16 cross-sectional views of the binary voxel model, each of which in fact consists of 8 consecutive slices. By manipulating the output look-up table the display of individual bitplanes may be enabled or disabled. This allows us to display selected slices from each section.

4.2. Displaying a Shaded Surface View

A 3D model can only be rendered on a flat display screen as a 2D projection. However, we may use several depth cues to create an illusion of the third dimension. Such depth cues are:

- (simultaneous) projections from different viewpoints
- perspective projection
- hidden surface elimination
- stereo pairs
- depth shading (or depth cueing)
- surface shading and shadows
- real time rotation.

In our application, perspective projection is not relevant. Real time rotation of the object is a very effective depth cue, but usually not feasible without the support of special purpose hardware [8, 14, 15]. Several of these depth cues are available in our system.

Hidden surface display. Hidden surface elimination, or more accurately *visible surface detection*, is performed by a ray casting algorithm. This involves examining the voxels along a ray from the viewpoint through the voxel cube until the first 1-voxel is hit. Because of the way we store the cube, each byte in the binary voxel array holds eight voxels. This means that when we cast a ray along one of the main axes ($\pm X$, $\pm Y$, $\pm Z$), the algorithm may examine eight voxels at a time until a non zero byte is found. This byte holds the surface voxel that is subsequently retrieved by masking it out from the byte. This method results in a reasonably fast surface detection algorithm.

Depth shading. If we let the ray cast algorithm yield the depth coordinates of the surface voxels along the viewing axis, we obtain a kind of elevation map, that is often called a *depth shaded pre-image*. Strictly speaking though, at this point it does not yet represent a shaded image, but it may be used as input data for a depth shading algorithm. This image (or possibly several from different viewpoints) is stored in the second framebuffer. Displaying a shaded image is accomplished by calculating the light intensity as some function of the depth coordinate for each of it's 128 possible values and loading these function values in an output lookup table. Later, we will show how to employ the pre-image for other display facilities.

Binary gradient shading. For the calculation of the light intensity along the surface as a result of the diffuse or specular reflection of light, we must determine the direction of the surface normal vectors. The values of the normal vectors are approximated by examining the locally 6-connected (or face-adjacent) neighbours of the surface voxels. When a neighbouring voxel has value 1, the normal vector gets a component in the opposite direction. Thus, there are 26 possible values of the normal vectors.

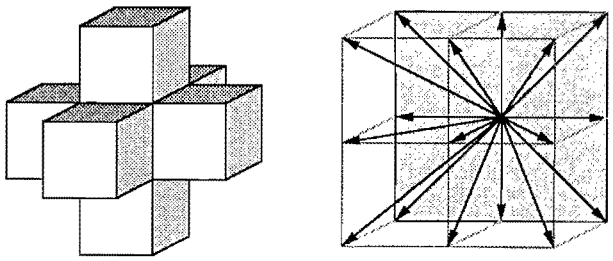


Figure 4: (a) A voxel and it's 6-connected neighbours; (b) 17 of the 26 possible surface normal vectors.

Calculation of the normal vectors may be done in two ways. In the first method the value is calculated 'within' the ray cast algorithm by examining the neighbours of a surface voxel

in the data set and returning an encoded normal vector per ray. The second method works via the pre-image: the convolution hardware runs a binary gradient operator over the pre-image.

Because we store the values of the surface normal vectors in the second frame buffer, we may again calculate the light intensity as a function of the direction of incident light for all possible values of the surface normal vectors and load the function values in a hardware output lookup table. Changing either the light direction or the lighting model (surface reflection coefficients etc.) is then a matter of altering the lookup table entries [2, 18].

The value of a surface normal vector may be encoded in 5 bits, as there are 26 possible values only. The remaining 3 bits of each byte may be used to hold an approximation of the surface depth coordinate. The lighting model may thus include both a part describing the reflection of light and depth cueing.

General gradient shading. The pre-image may be used as input data for a more general gradient shading method. Again we may use the image processing hardware to compute various gradient operators over the depth coordinates [3, 9].

Enlarging a pre-image. A pre-image may be enlarged by performing the following sequence of steps: first, the pre-image is zoomed by a factor of two. This means that each pixel (depth coordinate) is replicated 4 times in a 2×2 pixel square. If we then run a pixel averaging operator over the zoomed image, the replicated depth values are replaced by interpolated values, while the original values remain unchanged (see figure 5).

13	5	0	0
0	8	0	0
0	0	0	0
0	0	0	0

13	13	5	5
13	13	5	5
0	0	8	8
0	0	8	8

13	9	5	3
7	7	7	3
0	4	8	4
0	2	4	2

Figure 5: (a) original pixel values; (b) after zooming $\times$ 2; (c) after pixel averaging.

After this, we may again run any gradient operator (binary or other) over the enlarged pre-image to obtain a shaded surface view.

4.3. Manipulating Voxel Models

Manipulations of the voxel models are subdivided in two categories: local modifications and global modifications. Local modifications require local recalculations of surface normal vectors only. Global modifications require a re-display of the entire object.

Editing a voxel model. A voxel model may be edited in two ways: either individual voxels may be removed or added, which is a local modification, or a spatial selection may be applied on the voxel model by removing or adding voxels in specified x, y, z intervals. In this case we have to perform a global re-display of the object.

Reslicing a reconstructed object. A slight modification to the original ray-cast algorithm enables us to reslice a reconstructed object along an arbitrary plane. The equation describing a plane in three dimensions is

$$ax + by + cz + d = 0.$$

Assuming that we wish to view the object along the positive z-axis, we rewrite the equation as

$$z = -(\frac{a}{c})x - (\frac{b}{c})y - \frac{d}{c}.$$

The ray-cast is now performed by either starting the search for the first 1-voxel along the ray at the z-coordinate determined by this equation, or by continuing the search until this x-coordinate value. These two cases correspond to either removing the voxels for which the inequality $ax + by + cz + d < 0$ or the inequality $ax + by + cz + d > 0$ holds.

Modelling objects. Complex objects may be built from combinations of precomputed and stored primitive objects by applying the set operations union, intersection and difference in a CSG like way Also, parts may be removed from an object by computing the difference between the object and a secondary object. This is of interest when we want to examine a real-world object that has been reconstructed from physical data.

To implement set operations between objects, we make use of the correspondence between set operations and boolean operations: assuming we have two objects, represented by sets of boolean volume elements A and B, and volume elements v, then

$$A \cup B = \{ v \mid (v \in A) \text{ OR } (v \in B) \}$$

$$A \cap B = \{ v \mid (v \in A) \text{ AND } (v \in B) \}$$

$$A - B = A \cap B'$$

where B′ denotes the complement of B, which corresponds to the boolean operation NOT and the expression $v \in A$ corresponds to v's bit in the binary voxel model having value 1.

The ability of the image processing hardware to perform various bitwise logical operations between two framebuffers may be used for an efficient implementation of the set operations. The primary object's binary voxel model is loaded in frame buffer 1, while the secondary objects representation is loaded in frame buffer 2. Then, the frame processor calculates the result of the bitwise logical operation and stores it in either frame buffer, after which it can be displayed †.

† The logical NOT operation is performed by moving all pixels via the feedback loop on the DT2851 frame grabber/buffer board through a lookup table containing 256 monotonically decreasing values, thereby inverting the pixels' bits.

4.4. Calculating the Volume of an Object

Determining the volume of an object involves counting all 1-voxels of the binary voxel model. This operation may be speeded up by using the histogramming hardware of the frame processor. The result of a histogramming operation is a table of 256 entries, each entry being the number of pixels having that entries pixel-value. Each 8-bit pixel-value represents 8 voxels, the number of one bits being equal to the number of 1-voxels. If we now multiply each entry by the number of 1-bits of the corresponding table-index (or *bin*), and sum all 256 terms, we end up with the total number of 1-bits in the frame buffer, which equals the total number of 1-voxels in the binary voxel model.

5. Results and Further Research

As an experimental extension to the earlier implementation of the reconstruction system, the capability to output the series of parallel 2D area masks, originally used for the hidden line display of the contour stack, was added. This gave us the possibility to obtain a binary voxel model of several reconstructed objects. Next, these voxel models were used as input data sets for the display routines of the system that is now under development.

We are now able to generate a binary gradient-shaded surface view along a major axis in 10 seconds. The interactive rotation of a light source around the object is possible in near real-time. Observing the changing surface-shading gives a very clear impression of the surface irregularities of the object. Computation of a depth shaded image also takes about 10 seconds. For comparison purposes: the earlier system generated a depth shaded image in 3 minutes.

Although a general gradient shading method has not yet been implemented, we expect to be able to generate images in about 10 s. This because the frame processor is able to compute a convolution of a depth shaded image with a 3×3 kernel matrix in 0.4 s, which is negligible compared to the 10 s for the computation of the depth-shaded image.

The development of the editing and object modelling operations is about to start. The performance of the modelling operations will be limited by the speed with which the primitive objects can be stored in the secondary frame buffer, as the frame processors performs logical operations on two framebuffers in real-time.

The storage scheme that we use for the binary voxel array favors the computation of orthographic projections of the object, because in these cases the ray-cast algorithm examines 8 voxels simultaneously. When computing general parallel projections, voxel-address calculations will probably cause display times to grow to unacceptable amounts. However, we are thinking of implementing isometric projections (i.e. with the projection plane perpendicular to one of the lines $\pm x = \pm y = \pm z$), since voxel address calculations are relatively easy in this case.

6. Conclusions

We are using conventional image processing hardware to implement a 3D reconstruction and display system. The hardware may not only be used in the input preparation and reconstruction steps, but also for the display and manipulation of the reconstructed 3D objects. The hardware support speeds up the display-processing significantly. Further improvements will most likely only be possible through the use of special purpose hardware.

7. References

[1] E. Artzy, G. Frieder, and G. T. Herman, "The theory, design, implementation and evaluation of a three-dimensional surface detection algorithm". *Computer Graphics and Image Processing*, vol. 15, pp. 1-24 (1978).

[2] D. H. Bass, "Using the video lookup table for reflectivity calculations: specific techniques and graphic results". *Computer Graphics and Image Processing*, vol. 17, pp. 249-261 (1981).

[3] S. Bright, and S. Laflin, "Shading of solid voxel models". *Computer Graphics Forum*. vol. 5, pp. 131-137 (1986).

[4] L. S. Chen, G. T. Herman, R. A. Reynolds, and J. K. Udupa, "Surface shading in the cuberille environment". *IEEE Computer Graphics and Applications*. vol. 5, pp. 33-43 (1985).

[5] "User Manual for DT2851 High Resolution Frame Grabber", Data Translation, Inc., Marlborough, Massachusetts (1986).

[6] "User Manual for DT2858 Auxiliary Frame Processor", Data Translation, Inc., Marlborough, Massachusetts, (1986).

[7] H. Fuchs, Z. M. Kedem, and S. P. Usselton, "Optimal surface reconstruction from planar contours". *Comm. ACM*, vol. 20, pp. 693-702 (1977).

[8] S. M. Goldwasser, and R. A. Reynolds, "Real-time display and manipulation of 3-D medical objects: the Voxel Processor architecture". *Computer Vision, Graphics and Image Processing*, vol. 39, pp. 1-27 (1987).

[9] D. Gordon, and R. A. Reynolds, "Image space shading of 3-dimensional objects". *Computer Vision, Graphics and Image Processing*, vol. 29, pp. 361-376 (1985).

[10] D. P. Huijsmans, "Closed 2D contour algorithms for 3D reconstruction". *Proc. Eurographics*, pp. 157-168 (1983).

[11] D. P. Huijsmans, W. H. Lamers, J. A. Los, J. Smith, and J. Strackee, "Computer-aided three-dimensional reconstruction from serial sections". *Proc. Eurographics*, pp. 3-13 (1984).

[12] D. P. Huijsmans et.al., "Toward computerized morphometric facilities". *The Anatomical Record*, vol. 216, pp. 311-317 (1986).

[13] D. P. Huijsmans, and G. J. Jense, "Representation of 3D objects, reconstructed from series of parallel 2D slices". In: R. A. Earnshaw (ed), *Theoretical Foundations of Computer Graphics and CAD*, Springer-Verlag, Berlin (1988).

[14] D. Jackel, "The graphics PARCUM system". *Computer Graphics Forum*, vol. 4, pp. 21-35 (1985).

[15] A. Kaufman, and R. Bakalash, "CUBE - An architecture based on a 3D voxel map". In: R. A. Earnshaw (ed), *Theoretical Foundations of Computer Graphics and CAD*, Springer-Verlag, Berlin (1988).

[16] W. E. Lorensen, and H. E. Cline, "Marching Cubes: a high resolution 3D surface construction algorithm". *Computer Graphics* (proc. SIGGRAPH) vol. 21, pp. 163-169 (1987).

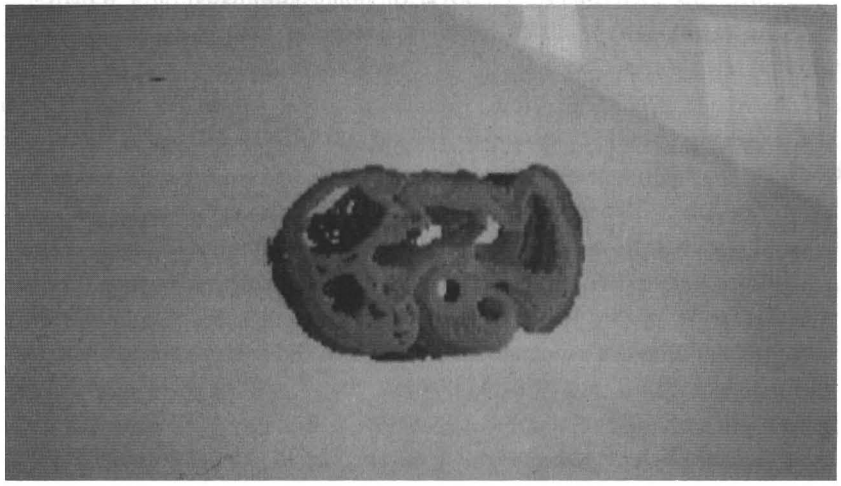
[17] A.A.G. Requicha, "Representations for rigid solids: theory, methods and systems". *Computing Surveys*, vol. 12, pp. 437-464 (1980).

[18] K. R. Sloan, and C. M. Brown, "Color map techniques". *Computer Graphics and Image Processing*, vol. 10, pp. 297-317 (1979).

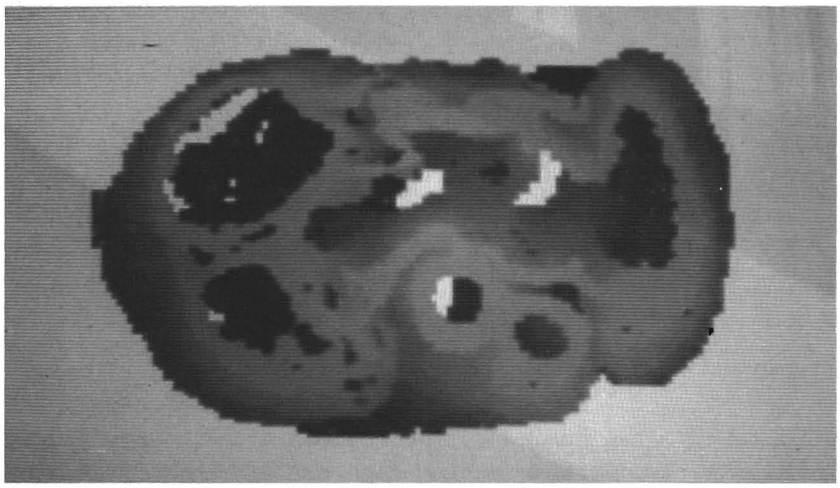
[19] S. S. Trivedi, "Interactive manipulation of three dimensional binary scenes". *The Visual Computer*, vol. 2, pp. 209-218 (1986).

Pictures

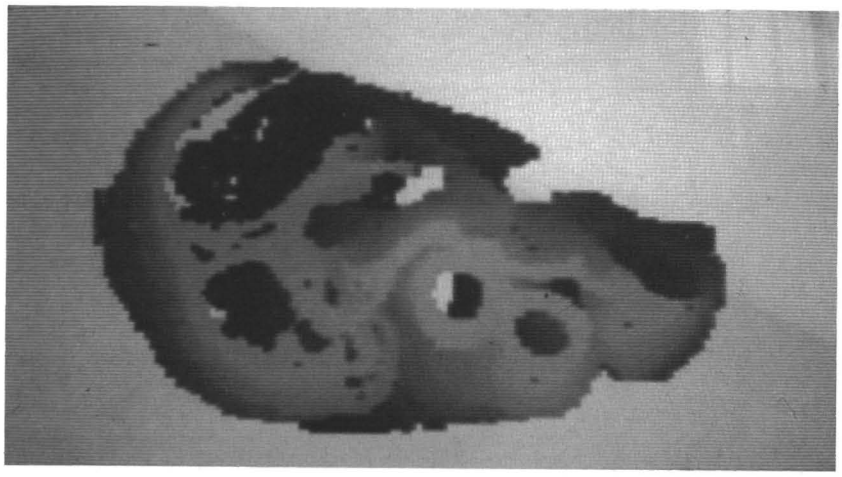
All images show an embryonic rat heart, reconstructed from about 70 serial sections. The diagonal band visible in some images is an artefact from interference between the vertical scanning of the display and the horizontal traversal of the camera shutter.



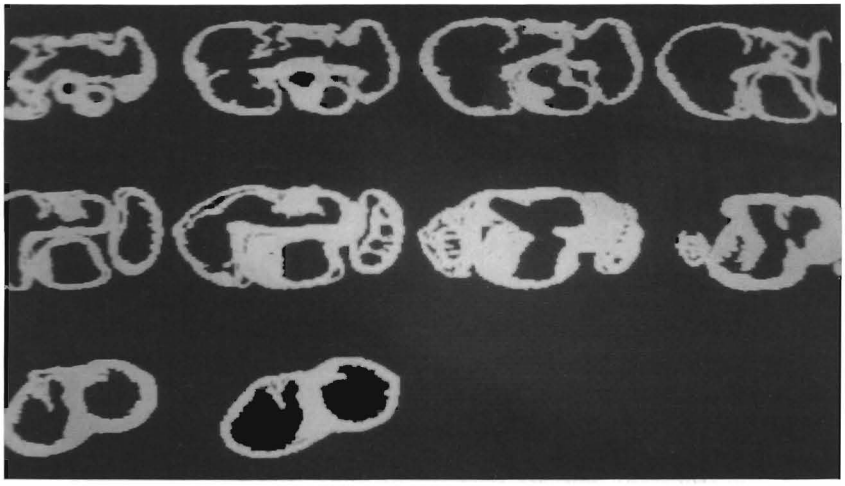
Picture 1: Binary gradient- and depth shaded.



Picture 2: Depth shaded, zoomed 2 ×, high pass filtered.



Picture 3: Same, but object resliced along a plane.



Picture 4: Voxel data set as stored in one of the frame buffers.