A hardware architecture for video rate smooth shading of Volume data

Michael C. Doggett* and Graham R. Hellestrand
School of Computer Science and Engineering
The University of New South Wales †

Abstract

This paper describes a new architecture for generating smoothly shaded two dimensional images of volume data. This architecture fits into an image synthesis pipeline and uses only simple arithmetic operations and a look-up table to generate two dimensional images in real time. The shading algorithm is an extension of the grey-level gradient algorithm for shading volume data. The shading technique produces smooth images for voxelized geometrical data and sampled volume data. Image synthesis from volume data in real time is an important technique in visualization and graphics systems.

Additional Key Words and Phrases: real-time shading, volume visualization, graphics hardware, image generation

1 Introduction

The generation of two dimensional (2D) images from volumetric data sets containing either measured data or voxelized geometrical primitives is creating a new alternative for image synthesis techniques [8]. The ability to synthesize 2D images in real time, that is at video presentation rate, will allow the interactive exploration of these data sets through several interfaces including Virtual Reality. Many aspects of real time volume imaging require improved technology, including processing power for image analysis and synthesis, and data sampling techniques for rapid image data acquisition in modalities, such as magnetic resonance imaging (MRI). Improvements in processing power can be achieved through the creation of special-purpose graphics imaging hardware [9].

The ability to generate two dimensional images from volumetric data, based on voxelized geometrical primitives, provides an alternative to surface based graphics. Image synthesis from voxelized data is independent of the original scene complexity once voxelization has been completed. This paper is concerned with the surface shading of voxel data in real time using a graphics pipeline.

Three dimensional sampled magnetic resonance (MR) data of the human body is used daily in the diagnosis of medical conditions. Real time MRI of the moving internal organs of the body will further assist medical treatment, in particular, heart conditions. Imaging technology of this nature will generate large amounts of data to be rendered for human inspection in real time. The technology described in this paper is intended for this type of application.

2 Shading techniques for Volume data

Image generation involves the calculation of interaction between light rays and surfaces to create recognisable three dimensional objects [3]. Shading of Volume data is accomplished in a similar fashion with the construction of surfaces being the key factor. Due to the large volume of data, processing speed is limited by the complexity of calculating the normals to the constructed surfaces.

Direct volume rendering involves shading volume data without producing intermediate image states and numerous algorithms exist for this purpose [10]. In this paper we will deal with the shading technique called gradient shading [4] and an extension which uses data contained in each voxel [6]. This extension is commonly referred to as grey-level gradient shading and produces images with smooth shading of high quality, but is limited due to its inability to display thin objects and failure where the grey-level is inappropriate for locating the surface in volume data.

The generation of images using volume data in real time is made practical by low complexity shading algorithms and application specific hardware [10, 12]. A hardware algorithm called congradient shading, which is capable of shading volume data in real time using

*Email: miked@vast.unsw.edu.au
†Sydney 2052, AUSTRALIA

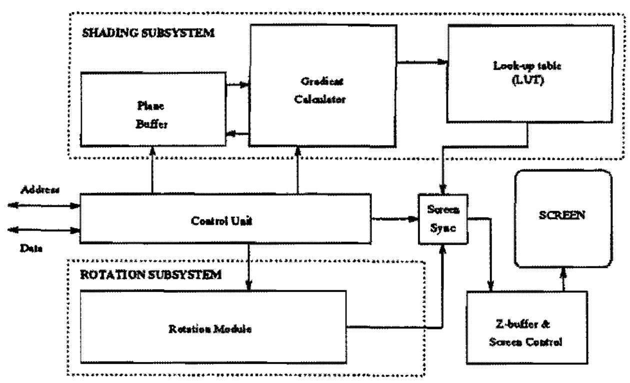

Figure 1: Top level diagram of system architecture

gradients and look-up tables is presented in [1]. This architecture only uses the x and y gradients and not the z gradient, and hence the quality of normals generated is impaired. The Verve system [11] uses gradient shading to determine the normals for shading and is capable of generating images in real time by employing several units in parallel to create Phong shaded surfaces. The Verve system casts a viewing ray through the volume for each pixel on the view plane and has a long arithmetic pipeline for vector normalisation.

3 System Overview

An architecture has been proposed for the creation of video rate images, and is described in [2]. The architecture was designed as an integrated processing element for generating screen pixels and the organisation of the processing element is shown in Figure 1. In this figure, the rotation and shading subsystems are independent which allows them to operate in parallel. Processing of the volume data set is done in a plane by plane, scanline fashion. This style of processing allows the system to be adapted to work from a standard memory arrangement or a stream of input data from a variety of sources. The plane by plane, scanline processing style also reduces the calculation complexity for projections. The output from these two concurrently operating subsystems is synchronized prior to display by a control unit which also controls the operations in both the rotation and shading subsystems. The authours are investigating the possibilities of supporting translucency by incorporating image compositing calculations into the shading subsystem and screen control unit.

3.1 Shading Subsystem

The shading subsystem is made up of several sections including, the plane buffer, the gradient calculator, and the Look Up Table (LUT). The plane buffer stores voxel values to be fed into the cubic window inside the gradient calculator. The cubic window is a buffered set of registers representing a three dimensional $3 \times 3 \times 3$ window through which voxel data is streamed. The gradient is calculated for the voxel located at the centre of this 26 connected neighbourhood and represents a surface normal at the centre voxel which is used as the index for the LUT. The result from the LUT is used to generate the final value for the screen. This paper is concerned with the gradient calculator and uses an architecture not previously reported.

3.2 Rotation Subsystem

The rotation subsystem projects three dimensional coordinates to a two dimensional viewing plane, using an axonometric orthographic projection [3] which requires a three dimensional rotation of the data set to the viewing plane. The mathematical formula for the X component of the projection is: $X = ax_v + by_v + cz_v$, where X is the x value in the viewing space, a, b, c are constants associated with the current viewing position, and x_v, y_v, z_v are the positions of the current voxel being processed in the volume data. The rotation subsystem takes advantage of the traversal of voxel memory by only having to recalculate the increment along one axis, the scanline direction, for each voxel processed, since values of the data set only change along the scanline. The critical calculation per voxel for a parallel projection is reduced to three parallel multiplys and adds per voxel.

3.3 Light sources and shadows

A limitation in this system is the restriction on the placement of light sources, which can be located at an infinite distance from the object, and only on one of the three cartesian axes. This placement means occlusion calculations can be performed while the volume data is processed and up to three light directions can be used. The shading calculation for 3 light directions can take advantage of the pre-calculated LUT by rearranging the gradient components to create a new LUT index for the other two light sources [2]. If a single light source is used and no shadowing is calculated, the system supports an arbitrary light direction for shading calculations in the LUT. The feasibility of a parallel implementation that uses shadowing and limited multiple light sources has not been investigated.

4 Surface Shading

The shading of three dimensional volume data is done using the grey-level gradient technique which calculates the gradient at a voxel on the surface, using local neighbourhood voxels, and then uses that gradient as a surface normal. For a surface voxel at location i, j, k, within the volume data, the calculations for the orthogonal components of the gradient G are :

$$G_x = g_{(i+1,j,k)} - g_{(i-1,j,k)}$$

$$G_y = g_{(i,j+1,k)} - g_{(i,j-1,k)}$$

$$G_z = g_{(i,j,k+1)} - g_{(i,j,k-1)}$$

g represents the grey scale value at the coordinate specified by the coordinate indices. The light intensity is calculated using standard lighting calculations.

4.1 Lighting calculations

The calculation of light intensity reflected from a particular surface in a three dimensional scene is dependent on many factors [3]. To simplify this calculation in hardware, and also to allow other shading calculations to be used, an LUT is used in the shading subsystem. The following uses the gradient as a surface normal to calculate the light intensity reflected by a surface :

$$I = I_p k_d(\bar{G} \cdot \bar{L})$$

where I_p is the light source intensity
k_d is the diffuse-reflection coefficient
$\bar{G}$ is the normalised gradient
$\bar{L}$ is the direction toward the light source.

This calculation is completed for each possible gradient and stored in the LUT. The LUT is also capable of storing more realistic values such as those for specular reflection [3] :

$$I = I_a k_a + I_p[k_d(\bar{G} \cdot \bar{L}) + k_S(\bar{R} \cdot \bar{H})^n]$$

where I_a is the ambient light
k_a is the ambient coefficient
I_p is the light source intensity
k_d is the diffuse-reflection coefficient
$\bar{G}$ is the normalised gradient
$\bar{L}$ is the direction toward the light source
k_S is the material specular-reflection coefficient
$\bar{R}$ is the direction of reflected incident light
$\bar{H}$ is the halfway vector
(between light source and viewer)
n is the material specular reflection exponent

4.2 Neighbourhood grey-level gradient shading

The shading in this paper uses all of the 26 neighbours of the centre voxel in a $3 \times 3 \times 3$ window containing voxel data. This neighbourhood of voxels is referred to as the cubic window and contains the voxels used for surface shading in the architecture described in this paper. The use of a $3 \times 3 \times 3$ neighbourhood of voxels surrounding a surface voxel for shading is also described in [13] where a biquadratic surface is interpolated through the neighbourhood. The grey-level gradient shading algorithm has an extension which uses a 3×3 neighbourhood of each voxel adjacent to the centre voxel to calculate the gradient. We call this algorithm extension cubic shading. The inclusion of all 26 neighbours in the calculation of the gradient is described in [6] and used in [7], where it was used for non real time shading. The hardware described in this paper deals with the application of 26 neighbour grey-level gradient shading to a video rate hardware system.

5 Hardware design

This section deals mainly with the shading subsystem and particular attention is drawn to the pipeline for gradient calculation. The gradient calculator is an integral part of the shading subsystem of the architecture depicted in Figure 1, where it calculates a value which is used as an index to the LUT, which stores shading information. The volume data used in the calculation of the gradient is reused by storing it in the plane buffer, which holds approximately two adjacent planes of the original data.

The calculation of the gradient involves all of the elements in the cubic window. Figure 2 shows a three dimensional perspective of the flow of data into and out of the cubic window, where the coordinate axes are used to describe the positions of voxels, rows, columns and sides. To calculate the LUT index, which is a similar calculation to the gradient calculation described previously, the sums of the sides of the cubic window, made up of either three rows or three columns, are computed according to the following equations :

$$S_{0yz} = C_{00z} + C_{01z} + C_{02z}$$

$$S_{2yz} = C_{20z} + C_{21z} + C_{22z}$$

$$S_{x0z} = C_{20z} + C_{10z} + C_{00z}$$

$$S_{x2z} = C_{22z} + C_{12z} + C_{02z}$$

$$S_{xy0} = R_{x20} + R_{x10} + R_{x00}$$

$$S_{xy2} = R_{x22} + R_{x12} + R_{x02}$$

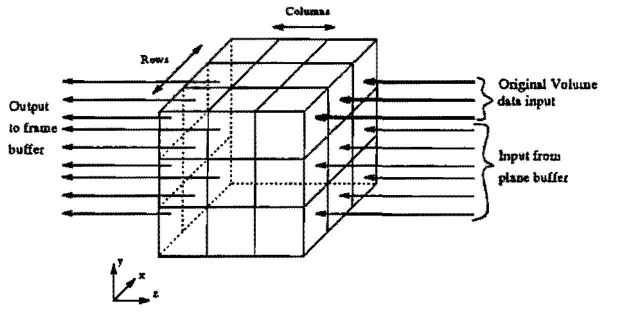

Figure 2: Cubic window showing input and output points and the coordinate axis used for indices.

where S_{xyz} is the summation of a side
S_{iyz} is the side where $x = i$
S_{xiz} is the side where $y = i$
S_{xyi} is the side where $z = i$
$C_{ijz} = g_{ij1} + g_{ij2} + g_{ij3}$
is the column where $x = i, y = j$
$R_{xij} = g_{1ij} + g_{2ij} + g_{3ij}$
is the row where $y = i, z = j$
g_{xyz} is the grey scale value of the voxel

The coordinate values in the variable subscripts are taken from the axis for the cubic window shown in Figure 2. The summation of each side is the sum of nine voxels in a 3 × 3 two dimensional window of voxels situated adjacent to the surface voxel. There are six of these windows and they represent the six sides of the three dimensional cubic window depicted in Figure 2.

The x, y and z components of the modified gradient equation are now calculated as :

$$G_x = S_{2yz} - S_{0yz}$$

$$G_y = S_{x2z} - S_{x0z}$$

$$G_z = S_{xy2} - S_{xy0}$$

This gradient value is normalised and used as the surface normal in the lighting calculations that are stored in the LUT.

5.1 Gradient Calculator

Figure 3 shows the top level of the architecture used to calculate gradient values at voxel data rate in the shading unit. The cubic window is made up of 27 registers which are laid out in a horizontal line, where the line is subdivided into groups of 3 registers representing columns in the cubic window. The volume data passes into the start of a column, through the column and leaves the cubic window at the other end of the

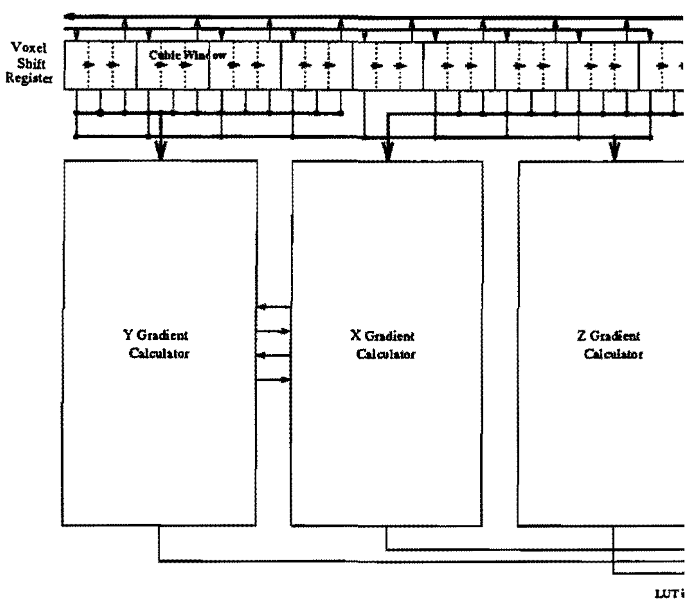

Figure 3: Gradient Module

same column. The data flow through the cubic window is shown in the voxel shift register at the top of Figure 3. The voxel values are passed from the registers in the cubic window into the appropriate calculating section for each of the X, Y and Z components, as determined by the gradient equations.

Each component calculator consists of a three stage pipeline with registers separating each stage. The organisation of the X, Y and Z component pipelines is shown in Figure 4. To show how the pipeline works, the stages of the Y gradient calculator are taken separately and shown in Figure 5. The first stage of the pipeline consists of values from the cubic window passing through a Triple Input Adder (TIA) which generates a result that is placed in a column total register. In the Y gradient calculator, the first column on the left is taken from the cubic window, the components added together and the result placed in a column total register, as shown in Figure 5(a). The second stage uses the values from the column total registers as inputs to a second TIA which passes its output to a side total register, as shown for the Y gradient in Figure 5(b). The final stage takes the values from the side total registers and finds the difference and stores the result in one of the component registers. For the Y gradient calculator a side total calculated in the Y gradient calculator and a side total from the X gradient calculator are used to find the difference for the Y gradient register as shown in Figure 5(c).

The third stage of the pipeline which calculates the

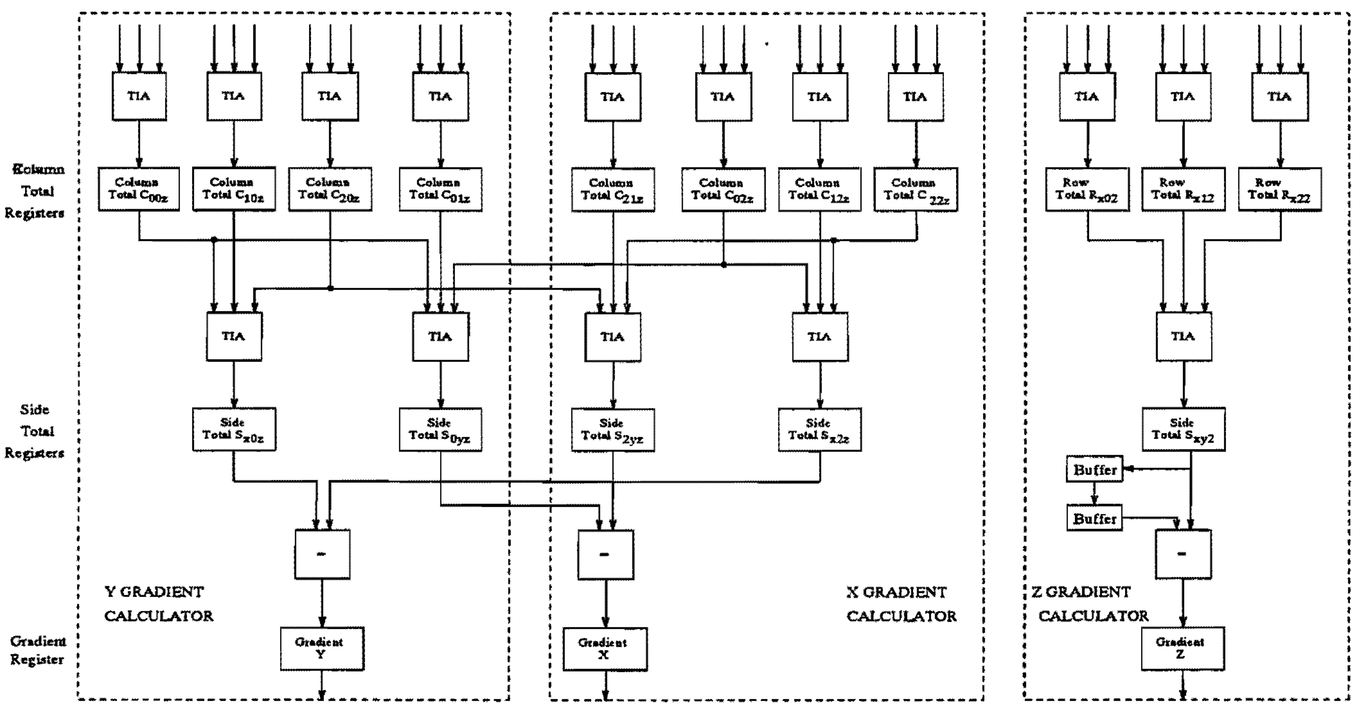

Figure 4: Gradient component pipelines

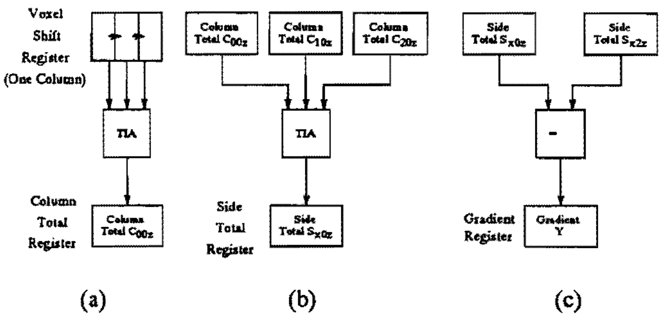

Figure 5: Pipeline stages in Y Gradient Calculator (a) Column total calculation stage (b) Side total calculation stage (c) Y gradient component calculation stage

Z component of the gradient is different from the X and Y component calculator pipelines. This difference is due to the manner in which data moves through the cubic window. The direction of the z axis is the same as the direction of data moving through the columns within the cubic window. Therefore the sum of the side on which data enters the cubic window will be equal to the sum of the side on which data leaves the cubic window after the cubic window has processed two centre voxels. To reuse this side total, when it arrives at the other side of the cubic window, the value is buffered twice. The difference between the current side total and the buffered side total is taken to calculate the final Z gradient component.

After calculation of the X, Y and Z components of the gradient, the gradient is used as the LUT index.

The number of bits used from the result of the gradient calculator determines the size of the LUT. Better shading is possible by using a full 8-bit result, but a large LUT is required. If the LUT has an index of 15 bits then the five most significant bits are used from each gradient component result. The size of the LUT can vary with requirements for memory size, speed and rendering quality for a particular application. A 15 bit LUT would require 2^{15} bytes or 32KB to store the precalculated shading values and would be required to operate at the same speed as the pipeline. The calculated values in the LUT use the shading equations described above.

6 Results

To test the image results from the design presented in this paper, a software simulation was written to inspect images derived from a geometrically constructed smooth object, a sphere, and sampled medical data. The sphere was voxelized into a volume size of $64 \times 64 \times 64$ with each voxel being divided into a smaller $16 \times 16 \times 16$ grid. Each discrete point on the smaller $16 \times 16 \times 16$ grid was tested to see if it was inside the sphere or outside the sphere. The number of discrete points at each voxel which are inside the sphere is calculated and stored as volume data. This volume data was reduced to eight bits and rendered using both typical grey-level gradient shading and cubic shading. The results are shown in Figure 6, where the top two spheres use double precision floating point calculations for shad-

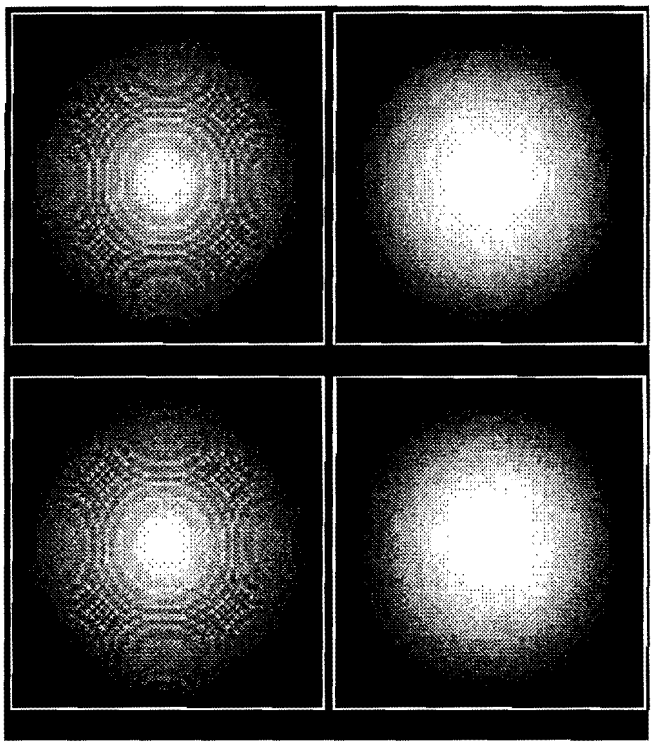
Figure 6: Comparison of shading techniques with look-up tables.

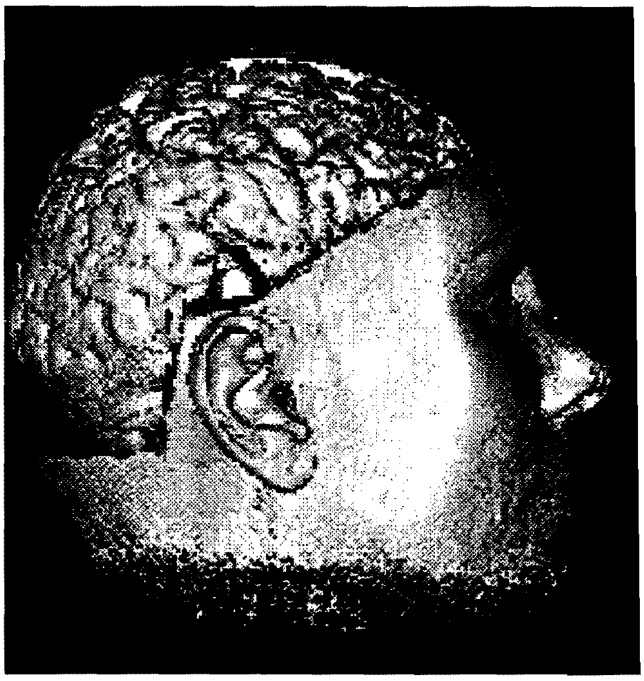
Figure 7: Rendering of MRI data set using gradient shading and a 21-bit LUT.

ing, and the lower two spheres use a 15-bit LUT for each shading algorithm. Visually, the cubic shading produces images which are closer to the expected look of rendered spheres.

To test the algorithms on sampled data, a common public domain MRI data set of 109 image slices through a human head, (each with a resolution of 256×256 pixels, 8 bits per pixel, and a pixel size of approximately $1mm \times 1mm$) was used. Figure 7 shows the results of gradient shading and the results of cubic shading are shown in Figure 8. While the cubic shaded image appears smoother in comparison with the gradient shaded image, it is difficult to determine whether fine detail has been lost in either or both shading processes.

This system offers the ability to incorporate original greyscale data into rendered surfaces as shown in Figure 9. This type of image generation is accomplished easily by setting volumes in the original data where voxels are transparent and planes where voxels are not to be rendered, but the original data is to be passed through to the final image. This process allows the inspection of the original data for finer detail that may have been lost in the shading process, and particularly, for a quantitative evaluation of the differences between the rendered and the original data.

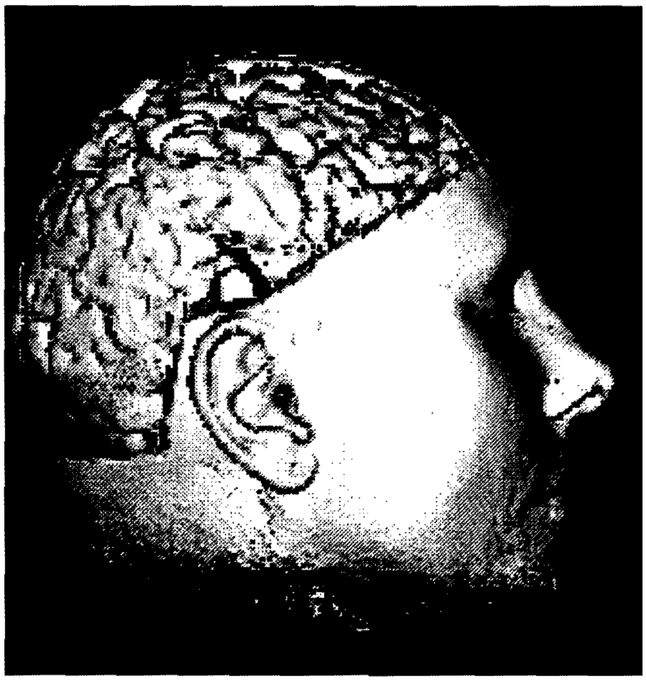
Figure 8: Rendering of MRI data set using cubic shading and a 21-bit LUT.

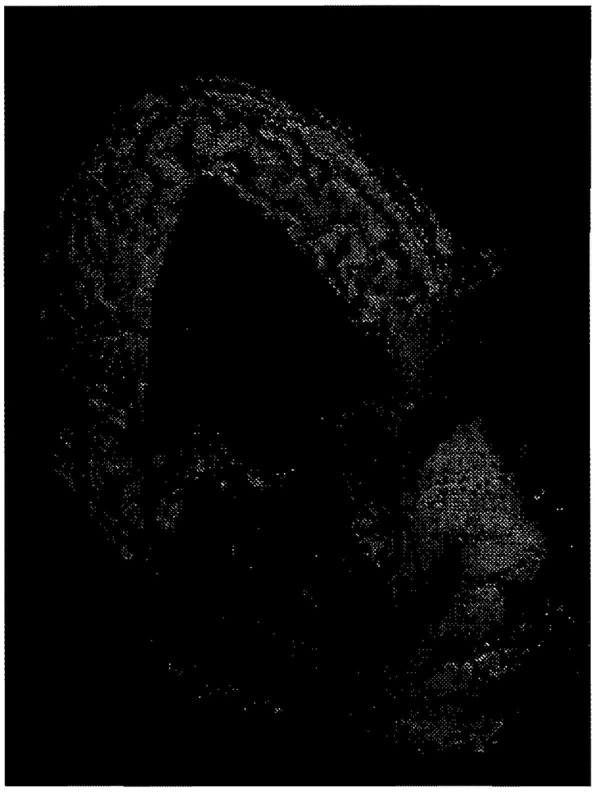

Figure 9: Rendering of MRI data with cut-away showing original scan.

6.1 Quantitative analysis

To compare the rendered results in a quantitative fashion the spherical images in Figure 6 are subtracted from an image of a sphere shown in Figure 10. The normals used in the shading of the sphere in Figure 10 are calculated using the equation of a sphere. Each voxel which the surface of the sphere passes through has its normal calculated from the spherical equation and shading is completed using the same double precision lighting calculations as used for the other spheres.

Once the difference between images is found, all pixel differences are used as a sample space to find means, variances and mean squares. These are shown in Table 1. The values in the table are based on grey-scale images which have pixels with values ranging from 0 to 256. The table shows there is a reduction in all statistical measures when using cubic shading instead of gradient shading. The reduction in mean, variance and mean square demonstrates that the normal construction used in cubic shading has final pixel values which are closer to the pixel values for a sphere using normals calculated from the sphere equation. When comparing the results without an LUT and the results with an LUT, the improvement due to cubic shading is not as significant with an LUT. This shows that in general shading is less accurate with an LUT and that an LUT reduces the effect of cubic shading.

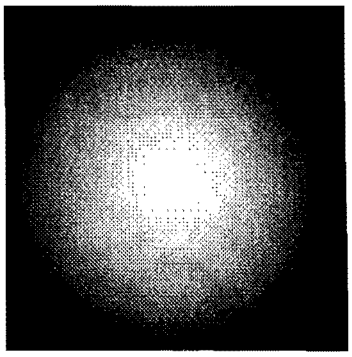

Figure 10: Rendering of a sphere with normals calculated directly from the equation of a sphere.

Original image	$\bar{x}$	s^2	x^2
Gradient_Shaded_Sphere	-7.6	214.6	272.1
Cubic_Shaded_Sphere	-2.1	102.3	106.6
Gradient_Shaded_using_LUT	-8.1	237.0	302.2
Cubic_Shaded_using_LUT	-6.2	199.0	237.2

Table 1: Sample mean ($\bar{x}$), sample variance (s^2) and mean square ($\bar{x^2}$) for image difference.

6.2 Hardware Simulation

To test the function and timing of the gradient calculator pipelines, the gradient calculator was described and simulated using a Hardware Description Language and an event driven simulator [5]. The slowest stage in the pipeline, the TIA, is simulated using an 8-bit ripple carry adder design with a resultant minimum clock cycle of $8ns$. This simulation result does not take account of interconnect delays, but considering the speed improvements of using faster carry techniques, a slightly faster clock speed would be possible. Memory reading is also required to run at this rate for the retrieval of volume data, and for the LUT. To process data sets of the size of 256 × 256 × 256 at 20 frames per second requires each subsystem to produce a result every $3ns$. A single processing element implemented in a non-aggressive technology may not be capable of video rate processing. In order to achieve this and higher speeds several processing elements may be used and run in parallel, with volume data being subdivided into equal volumes to achieve approximately linear speedup. Implementation of the design in a single element in 0.5 micron technology or as parallel processing elements would provide the video rate performance required.

7 Conclusion

This paper has presented an architecture for the shading module of a hardware system capable of producing video rate images with a quality suitable for interactive volume visualization. The shading of both voxelized spheres and MRI data has a natural appearance comparable with other volumetric shading techniques. The LUT design allows the changing of pixel values generated for various lighting situations, surface values and volume data processing applications. The processing method offers simple techniques for mixing original data with rendered surfaces and adds shadowing effects to direct rendered voxel systems.

Further work will be carried out on the investigation of animation techniques for multiple frame renderings and the application of this system to cardiac imaging.

References

[1] COHEN, D., KAUFMAN, A., BAKALASH, R., AND BERGMAN, S. Real time discrete shading. *The Visual Computer 6*, 1 (February 1990), 16–27.

[2] DOGGETT, M., AND HELLESTRAND, G. Video rate shading for volume data. In *Digital Image Computing: Techniques and Applications* (December 1993), Australian Pattern Recognition Society, pp. 398–405.

[3] FOLEY, J. D., VAN DAM, A., FEINER, S. K., AND HUGHES, J. F. *Computer Graphics: Principles and Practice*. Addison Wesley, 1989.

[4] GORDON, D., AND REYNOLDS, R. A. Image space shading of 3-dimensional objects. *Computer Vision, Graphics, and Image Processing 29* (1985), 361–376.

[5] HELLESTRAND, G. R. Modal: A system for digital hardware description and simulation. *Journal of Digital Systems 4*, 3 (1980), 241–303.

[6] HÖHNE, K. H., AND BERNSTEIN, R. Shading 3d-images from ct using gray-level gradients. *IEEE Transactions on Medical Imaging* (March 1986). MI-5.

[7] HÖHNE, K. H., BOMANS, M., POMMERT, A., RIEMER, M., SCHIERS, C., TIEDE, U., AND WEIBECKE, G. 3d visualization of tomographic volume data using the generalized voxel model. *The Visual Computer 6*, 1 (February 1990), 28–36.

[8] KAUFMAN, A., COHEN, D., AND YAGEL, R. Volume graphics. *IEEE Computer 26*, 7 (July 1993), 51–64.

[9] KAUFMAN, A., HÖHNE, K. H., KRUGER, W., ROSENBLUM, L., AND SCHROEDER, P. Research issues in volume visualization. *IEEE Computer Graphics and Applications 14*, 2 (March 1994), 63–67.

[10] KAUFMAN, A. E., Ed. *Volume Visualization*. IEEE Computer Society Press, 1990.

[11] KNITTEL, G. Verve : Voxel engine for real-time visualization and examination. *Computer Graphics Forum 12*, 3 (1993), 37–48.

[12] STYTZ, M. R., AND FRIEDER, O. Volume-primitive based three-dimensional medical image rendering: Customized architectural approaches. *Computers and Graphics 16*, 1 (1992), 85–100.

[13] WEBBER, R. E. Ray tracing voxel data via biquadratic local surface interpolation. *The Visual Computer 6*, 1 (February 1990), 8–15.