A Real-Time Volume Rendering Architecture Using an Adaptive Resampling Scheme for Parallel and Perspective Projections

Masato Ogata[*] TakaHide Ohkami[†] Hugh C. Lauer[†] Hanspeter Pfister[†]
Mitsubishi Electric Information Technology Center America [‡]

Abstract

This paper describes an object-order real-time volume rendering architecture using an adaptive resampling scheme to perform resampling operations in a unified parallel-pipeline manner for both parallel and perspective projections. Unlike parallel projections, perspective projections require a variable resampling structure due to diverging perspective rays. In order to address this issue, we propose an adaptive pipelined convolution block for resampling operations using the level of resolution to keep the parallel-pipeline structure regular. We also propose to use multi-resolution datasets prepared for different levels of grid resolution to bound the convolution operations. The proposed convolution block is organized using a systolic array structure, which works well with a distributed skewed memory for conflict-free accesses of voxels. We present the results of some experiments with our software simulators of the proposed architecture and discuss about important technical issues.

CR Categories and Subject Descriptors: I.3.1 [Computer Graphics]: Hardware Architecture - Graphics Processors; I.3.3 [Computer Graphics]: Picture/Image Generation - Display Algorithms.
Additional Keywords: Volume Graphics, Volume Rendering, Raycasting, Raytracing, Parallel Projection, Perspective Projection, Scientific Visualization, Real-Time, Systolic Array.

1 INTRODUCTION

Volume visualization techniques are becoming available and popular. This is due to the increasing availability of scientific data generated by a variety of computer simulations, medical data obtained by MRI and CT scanners, and geological, oceanographic, and meteorological data collected from various sensors, and also due to the decreasing costs of high-performance computing required for volume rendering.

One of the notable characteristics shared by these volume data is the sheer amount of data elements to be processed in rendering. This requires a huge amount of computing resources for animated visualization, which is essential to observe some physical phenomena. Another characteristic of the data is that they can not be represented by surfaces as in the conventional polygon-based graphics; the volume data may include complicated internal structures and shapeless features. Because of these characteristics, fast direct volume rendering methods are in high demands.

The most popular volume rendering algorithm is the raycasting algorithm, which casts rays from the center of projection into a volume, computes samples along the rays, and accumulates the sampled values to determine the pixel values on the screen. In many cases, each sample is computed from eight voxels surrounding the sample point by linear interpolations. Each resampling operation is relatively simple, but the total number of resampling operations is very large, and the time spent for the operations is dominant in the rendering time. Because of this, a raycasting-based volume rendering system could be considered a resampling machine.

The rendering operations for parallel projections are very regular and amenable to parallel-pipeline processing. The operations for perspective projections, however, are variable due to diverging perspective rays. This processing variability adversely affects the parallel-pipeline structure for parallel projections and has been a major obstacle for hardware implementation of a perspective projection system.

In this paper, we propose a real-time volume rendering architecture using an adaptive resampling scheme for both parallel and perspective projections. The proposed architecture is structured as a *sample-parallel* machine to perform resampling operations in a unified parallel-pipeline manner for both types of projections. The processing variability issue is addressed by an adaptive pipelined convolution block for resampling voxels using the level of grid resolution. The convolution area can be arbitrarily large because of diverging perspective rays. Multi-resolution datasets are prepared for different resolution levels to address this issue.

This paper is organized as follows; section 2 describes related work, section 3 presents some issues for real-time perspective projections and the key ideas for the proposed architecture, section 4 shows a proposed hardware structure, section 5 presents some experiments with the architecture simulator, section 6 describes future work, and section 7 concludes the paper.

2 RELATED WORK

2.1 Rendering Methods

Rendering methods are categorized into two groups: image-order and object-order. Each method has advantages and disadvantages for structuring a real-time volume rendering architecture based on some form of parallel processing.

The image-order method casts rays from a screen into a volume. The number of rays to cast is determined by the screen size and resolution. The ray-parallel scheme parallelizes rendering operations on a ray basis [4]. The major disadvantage of this scheme is that one voxel is accessed by multiple rays for resampling, increasing the total number of memory accesses. There are some optimization techniques available for this scheme. Early ray termination and coherence encoding are two examples to reduce the number of memory accesses [6]. However, they require a variable resampling

[*] ogata@merl.com, visiting from Mitsubishi Electric Corporation
[†] {ohkami,lauer,pfister}@merl.com
[‡] 201 Broadway, Cambridge, MA 02139, U.S.A.

structure, making the pipeline structure complicated.

The object-order method, on the other hand, maps a volume onto a screen. The splatting scheme splats each voxel onto the screen [9]. One of its major disadvantages is that filter kernels of different sizes are required for perspective projections, The voxel-parallel scheme reads a voxel once and retains it until all the samples requiring the voxel are computed [8]. The major advantages of this scheme are the reduction of the memory accesses and the fixed resampling structure. The disadvantage is that there are no major optimization techniques available for this scheme. The optimization techniques for the ray-parallel scheme can not be used for this object-order scheme, because they break the fixed resampling structure.

2.2 Shear-Warp Algorithm

The *shear-warp* algorithm provides a unified framework for formulating the rendering operations for both parallel and perspective projections [5].

Given a viewing vector, the algorithm finds the principal viewing axis, the axis most parallel to the viewing vector, to permute a volume so that the principal viewing axis be the z axis. Raycasting is performed on the permuted volume. The resulting image is an intermediate image formed on the base plane, the plane perpendicular to the viewing vector, which includes the front face of the volume. It is warped to produce the final screen image.

Raycasting is effectively a shear operation that shears the voxel grid of the volume to parallelize the rays at the base plane. The parallelized rays are all perpendicular to the base plane. In parallel projections, the rays proceed in the sheared voxel grid. In perspective projections, the rays proceed in the sheared and progressively scaled voxel grid. The grid scaling depends on the distance between the center of projection and the resampling position in the z direction; the grid becomes finer as the z position increases.

Each pixel in the base plane image is computed by compositing the samples taken along the perspective ray starting at the pixel position on the base plane. A sample is estimated from the voxels in its neighborhood. Because of the progressively scaled grid, the samples along a perspective ray have to be computed from progressively larger groups of voxels.

The shear-warp algorithm can be implemented by both software and hardware. For hardware implementation, however, there are several critical issues to be addressed, such as how to compute samples along perspective rays in the progressively scaled voxel grid with a fixed amount of hardware resources and how to perform resampling operations in a unified pipeline manner for both parallel and perspective rays.

2.3 Specialized Hardware

The EM-Cube architecture [7] is a real-time volume rendering architecture under development. It is derived from the Cube-4 architecture [8] with significant modifications and extensions for VLSI implementation. The architecture is based on the shear-warp algorithm, but supports parallel projections only. It is a voxel-parallel architecture with the skewed voxel memory for conflict-free voxel accesses. Each rendering pipeline computes a sample at each slice of voxels; a slice is defined as a group of voxels with the same z coordinates in the volume space permuted by the viewing vector. A rendering pipeline computes samples for different rays at different slices. The compositor accumulates samples for each ray and stores the resulting pixels in the pixel memory to generate the base plane image, which is warped to produce the final screen image.

The rendering operations are performed in a parallel-pipeline structure with multiple rendering pipelines. The architecture computes samples slice by slice. It is designed to exploit the geometrical regularity in rendering operations for parallel projections; all the samples at a slice have the same offsets from the reference points, since all the rays are parallel and proceed from the points with the same offsets from the reference points toward the same direction with the same steps.

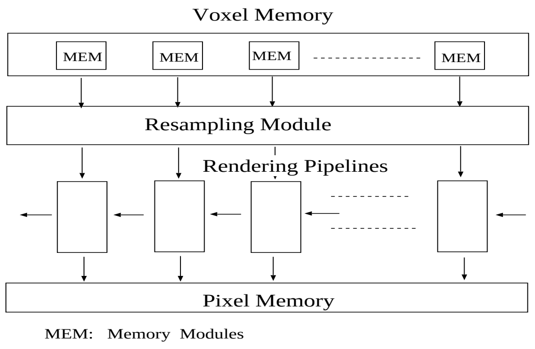


Figure 1: Proposed sample-parallel architecture.

The architecture is well organized for parallel projections. It is, however, not obvious to facilitate perspective projections in this architecture, because the variable resampling structure for the perspective rays does not fit well in its parallel-pipeline structure.

The ray-slice-sweep algorithm is proposed for parallel and perspective projections in the Cube-4L architecture [1]. It is still under investigation for improvements.

VIRIM is a real-time parallel rendering system for perspective projections [2]. It uses a Gaussian filter mask to compute samples from $8 \times 8 \times 8$ neighboring voxels. The resampling operations are performed in the rotator board and the resulting samples are sent to the multi-DSP board for raytracing. It is a parallel system, but not organized in a complete parallel-pipeline structure for VLSI implementation.

3 PROPOSED ARCHITECTURE

3.1 Sample-Parallel Architecture

The EM-Cube architecture is a voxel-parallel architecture that provides continuous streams of voxels from the voxel memory and feeds them to multiple rendering pipelines, each performing resampling operations. Its parallel-pipeline structure is tuned for parallel projections. The architecture's basic assumption is that a sample can always be computed from two neighboring voxels in one dimension, a notable characteristic of the parallel rays. This assumption does not hold for perspective projections, because the number of voxels required for resampling increases as the distance of the resampling point from the center of projection increases. It is not easy to include in the architecture the variable resampling structure required for perspective projections.

In order to address this issue, we shift a focus from voxels to samples and reorganize the rendering architecture as a *sample-parallel* architecture to provide a unified parallel-pipeline structure for both parallel and perspective projections. As shown in Fig. 1, the proposed architecture places a resampling module between the voxel memory and rendering pipelines. This module is specialized for resampling with a variable number of voxels in the resampling area. All the resampling functions are moved from the rendering pipelines to this resampling module to organize the rendering pipelines in a fixed parallel-pipeline structure for the other rendering operations.

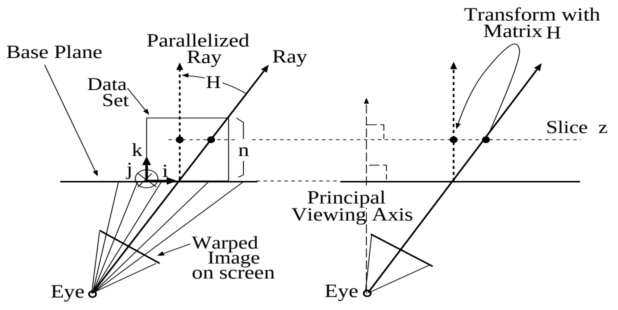


Figure 2: Shearing and scaling in perspective projections.

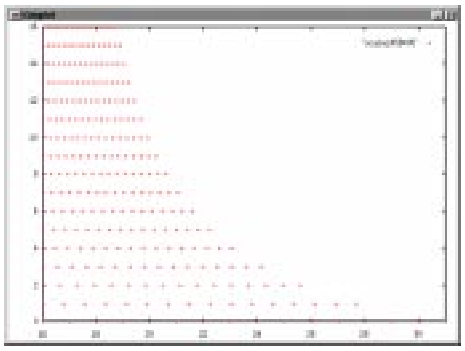


Figure 3: Sheared and scaled grid of a 16^3 dataset.

3.2 Perspective Projections

The shear-warp algorithm produces a base plane image as an intermediate image, which is warped to produce the final screen image. The warping operations are different from those for parallel projections due to diverging perspective rays. Fig. 2 shows the perspective rays parallelized at the base plane by a shearing matrix H. They are parallel to the principal viewing axis and perpendicular to the base plane. The base plane image is produced by the parallelized perspective rays.

The shearing transformation shears and progressively scales the voxel grid as shown in Fig. 3, where the voxel grid becomes finer as the distance from the base plane increases. Starting from a position in the first slice, a parallelized perspective ray proceeds in the progressively scaled grid to compute a sample at each slice. The computed samples are accumulated by a compositor to produce the final pixel value in the base plane image. One of the simple resampling operations is to average the values of the voxels in the neighborhood of a sample point. A more general operation is convolution over the voxels in the resampling area.

There are two important issues for the pipeline implementation of the convolver: convolution area and structure. The convolution area is determined by the resolution of the scaled voxel grid at the resampling point. The convolution area size W in one dimension is computed by:

$$W = 1 + k/Z_0, \tag{1}$$

where k is the slice number or the distance from the base plane and Z_0 the distance between the eye position (the center of projection) and the base plane. The size of convolution area is equivalent to the distance between the two neighboring perspective rays at slice k. It can be arbitrarily large with large values of k and/or small values of Z_0. This is illustrated in Fig. 4(a). Since the hardware implementation can only use a fixed amount of resources for convolution, it has to ignore the voxels outside the convolution area, producing a low-quality or aliased base plane image.

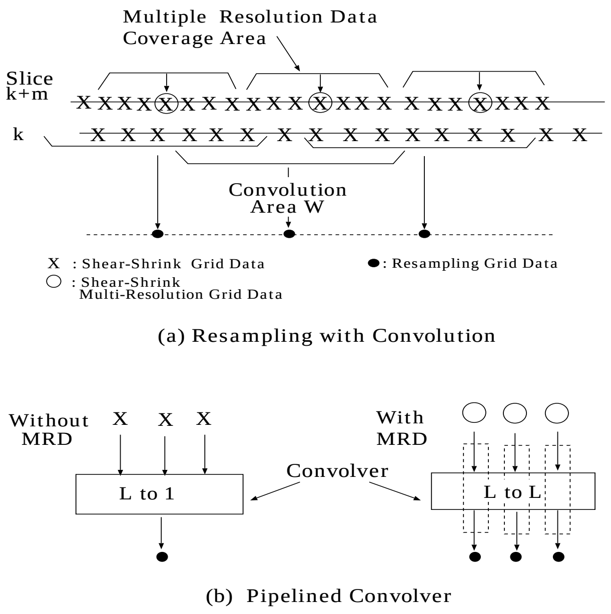

Multiple Resolution Data Coverage Area
Slice k+m
k
Convolution Area W
X : Shear-Shrink Grid Data
● : Resampling Grid Data
○ : Shear-Shrink Multi-Resolution Grid Data
(a) Resampling with Convolution
Without MRD
With MRD
Convolver
L to 1
L to L
(b) Pipelined Convolver

Figure 4: Convolution with multi-resolution datasets.

The convolution structure is another issue. The underlining architecture pairs a memory module and a rendering pipeline so that each pipeline can take one voxel along with a neighboring voxel through a sideway communication and produce one sample computed from the two voxels in each dimension. In this structure, the number of inputs (voxels) is equal to the number of outputs (computed samples). This holds for parallel projections, but not for perspective projections. Because of the progressively scaled grid, the number of outputs is equal to or less than the number of inputs, requiring a variable convolution structure, as illustrated in Fig. 4(b).

3.3 Multi-Resolution Datasets

The control of the variable convolution area is a critical issue for the hardware implementation of the convolver. In order to address the issue, we propose to use *multi-resolution datasets* prepared for different levels of grid resolution. It is a 3D version of the mip-mapping scheme for texture mapping [10]. As shown in Fig. 4(a), a data in a multi-resolution dataset represents the area covered by a certain number of original voxels. The covered area is larger in a coarse dataset than in a finer dataset. The use of multi-resolution datasets can reduce the number of data required for convolution. By selecting an appropriate multi-resolution dataset depending on the resolution of the scaled voxel grid, the architecture can always use a bounded number of data for resampling regardless of the number of original voxels covering the same convolution area. It also makes the pipeline convolver simple because the number of outputs is equal to the number of inputs as shown in Fig. 4(b). Fig. 5 shows the effect of using the multi-resolution datasets for the volume of the same size in Fig. 3.

It is not necessary, but very practical, to use powers of 2 for multi-resolution data, as illustrated in Fig. 6, where L0 is the finest, and L3 the coarsest. The memory overhead to store multi-resolution datasets for a volume of size n^3 is less than $n^3/7$, which is not considered a very large overhead.

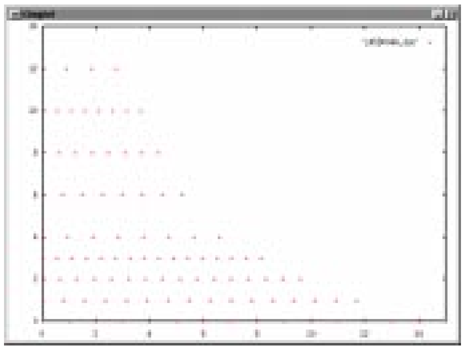


Figure 5: Sheared and scaled grid with multi-resolution datasets.

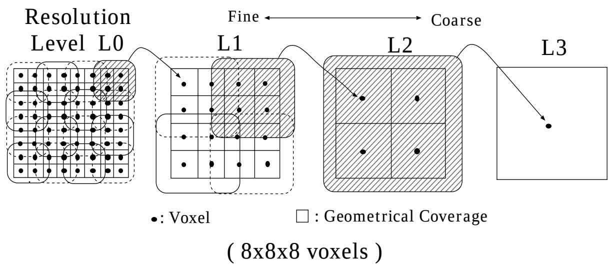


Figure 6: Power-of-2 multi-resolution datasets.

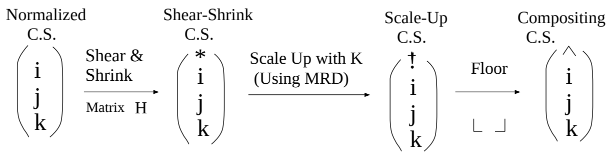


Figure 7: Four coordinate systems.

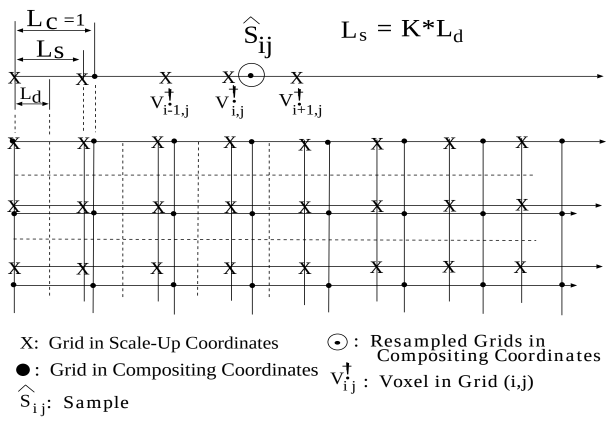

Lc =1
Ls
Ld
Ŝij
Ls = K*Ld
X: Grid in Scale-Up Coordinates
●: Grid in Compositing Coordinates
Ŝij: Sample
⊙: Resampled Grids in Compositing Coordinates
V†ij : Voxel in Grid (i,j)

Figure 8: Grids in scale-up and compositing coordinates.

3.4 Skewed Memory

The skewed memory organization is a technique to store voxels in separate memory modules so that voxels in a slice can be accessed in parallel without any memory conflict regardless of the viewing direction [3]. It does not require multiple volume copies. The proposed architecture uses it to store multi-resolution datasets.

Consider a system with N_p rendering pipelines for a volume of size n^3. Since each pipeline is connected one-to-one to a memory module, the number of memory modules is equal to the number of pipelines. Let (x, y, z) be a position index set of a data. Then the memory module number n_p and the memory module address i_p for the data are computed as follows:

$$m = (x + y + z) \bmod n, \quad (2)$$
$$n_p = m \bmod N_p, \quad (3)$$
$$i_p = \lfloor m/N_p \rfloor + yn/N_p + zn^2/N_p, \quad (4)$$

where n is a multiple of N_p. It guarantees that voxels in any slice denoted by $(*, y, z)$, $(x, *, z)$, or $(x, y, *)$ can be accessed in parallel with no memory conflict.

For the multi-resolution skewed memory, a memory address is specified by (L, x, y, z), where L is the resolution level. Each multi-resolution dataset can be stored in the skewed memory as if it were an original volume dataset. Multi-resolution data are accessed by the following addressing scheme:

$$m = (x' + y' + z') \bmod n', \quad (5)$$
$$n_p = m \bmod N_p, \quad (6)$$
$$i_p = \lfloor m/N_p \rfloor + y'n'/N_p + z'n'^2/N_p, \quad (7)$$

where

$$K = 2^L,\ x' = \lfloor \frac{x}{K} \rfloor,\ y' = \lfloor \frac{y}{K} \rfloor,\ z' = \lfloor \frac{z}{K} \rfloor,\ n' = \lfloor \frac{n}{K} \rfloor. \quad (8)$$

3.5 Convolution Area

The resolution level is an indicator of the convolution area size at each slice in the progressively scaled voxel grid to choose an appropriate multi-resolution dataset. The power-of-2-based resolution level is defined by

$$L = \lfloor \log_2 W \rfloor, \quad (9)$$

where W is the convolution area size in equation (1).

We use four 2D coordinate systems and transformations between them to determine the convolution area for a given resolution level. The four coordinate systems are the normalized, shear-shrink, scale-up, and compositing coordinate systems, as illustrated in Fig. 7. The normalized coordinate system defines the original voxel grid at slice 0. It is equivalent to the base plane coordinate system. The shear-shrink coordinate system defines the voxel grid sheared and scaled by a shear-shrink matrix. The scale-up coordinate system defines a grid scaled up by a scaling factor K. This is the effect of using multi-resolution datasets; a dataset covering a larger area effectively scales up the grid. The compositing coordinate system defines a pixel grid which coincides with the original voxel grid in the normalized coordinate system.

The sequence of transformations from the normalized coordinate system to the compositing coordinate system entails the sequence of grid changes. The procedure to determine the convolution area is described as a sequence of transformations between the coordinate systems as follows: 1) transform a grid point at (i, j, k) in the normalized coordinate system into a grid point at (i^*, j^*, k^*) by the shear-shrink matrix; 2) scale up the grid point at (i^*, j^*, k^*) using the scaling factor K to a grid point at $(i^\dagger, j^\dagger, k^\dagger)$; and 3) apply the floor operation to the grid position $(i^\dagger, j^\dagger, k^\dagger)$ to get the final position $(\hat{i}, \hat{j}, \hat{k})$. The scaling factor K is given in equation (8).

The multi-resolution data $v^\dagger_{i,j,k}$ for convolution are addressed by $(L, \lfloor i/K \rfloor, \lfloor j/K \rfloor, \lfloor k/K \rfloor)$, as shown in equations (5)-(8). Note that their geometrical positions are given by $(i^\dagger, j^\dagger, k^\dagger)$.

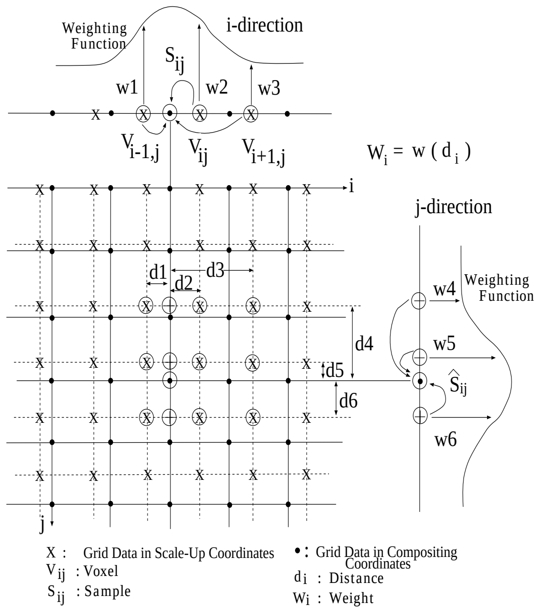


Figure 9: 2D convolution.

The grids in the scale-up and compositing coordinate systems are shown in Fig. 8, where L_s and L_c are the grid spacings in the scale-up and compositing coordinate systems, respectively. Note that L_c is equal to the original voxel spacing. Because of the power-of-2 multi-resolution datasets, the following condition holds:

$$L_s \leq L_c < 2L_s. \tag{10}$$

3.6 Sample Estimation

Sample values are estimated by convolution over voxels or multi-resolution data in the convolution area. We assume that the convolution kernel or the set of weights is symmetric for i, j, and k directions and separable; that is, $w_{ijk} = w_i w_j w_k$.

Consider $M \times M$ data for sample estimation. We apply a 1D convolution to i and j directions independently, as illustrated in Fig. 9. For simplicity, we assume that $M = 3$. In 1D, a sample value s_i can be computed by

$$s_i = w_1 v^\dagger_{i-1} + w_2 v^\dagger_i + w_3 v^\dagger_{i+1}, \tag{11}$$

where $v^\dagger_{i-1}$, $v^\dagger_i$, and $v^\dagger_{i+1}$ are data in the convolution area and w_m are weight functions defined by

$$w_m = w_m(\hat{i} - i^\dagger_m), \quad \text{for } m = 0, 1, 2 \tag{12}$$

where $\hat{i}$ is the sample position and $i^\dagger_m$ the data position.

The independent application of the 1D convolution to i and j directions simplifies the 2D convolution structure. First, three 1D convolutions in the i direction are computed:

$$\begin{bmatrix} s_{i,j-1} \\ s_{i,j} \\ s_{i,j+1} \end{bmatrix} = \begin{bmatrix} v^\dagger_{i-1,j-1} & v^\dagger_{i,j-1} & v^\dagger_{i+1,j-1} \\ v^\dagger_{i-1,j} & v^\dagger_{i,j} & v^\dagger_{i+1,j} \\ v^\dagger_{i-1,j+1} & v^\dagger_{i,j+1} & v^\dagger_{i+1,j+1} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}. \tag{13}$$

Then a 1D convolution for the j direction is performed to compute the final sample value $\hat{s}_{i,j}$:

$$\hat{s}_{i,j} = w_4 s_{i,j-1} + w_5 s_{i,j} + w_6 s_{i,j+1}. \tag{14}$$

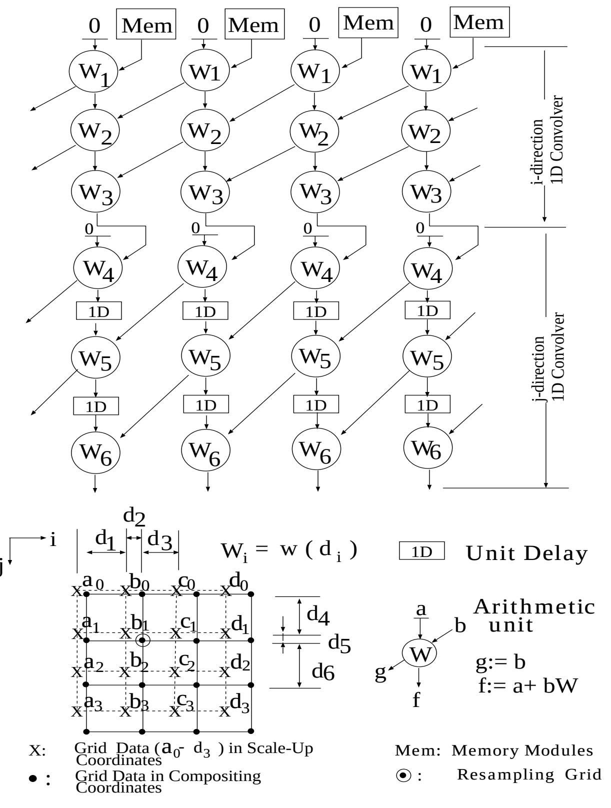


Figure 10: Pipelined 2D convolver.

The set of weights depends on the filter function chosen for convolution. Depending on the viewing direction, the distance between sample positions varies from 1 to $\sqrt[3]{3}$ in both parallel and perspective projections. To make corrections for this variable distance, an opacity table can be used to provide corrected opacity values by using the distance as an address, as described in [5].

4 HARDWARE STRUCTURE

4.1 Pipelined 2D Convolver

Fig. 10 illustrates an implementation of a pipelined 2D convolution for a 3×3 convolution area. The operations are divided into two groups: one group for the i direction and the other for the j direction. In this example, the dataset size in one dimension and the number of pipelines are both 4.

Table 1 shows several snap shots of the pipeline operations of the 1D convolution for the i-direction. In this example, one slice of 4×4 voxels is processed with the skewed memory organization. The 1D convolution structure for the j direction is the same, but has different time delays due to the different timings of neighboring voxels in the j direction.

4.2 Convolution Kernels

The previous pipelined 2D convolver works fine for parallel projections. However, it does not work for perspective projections because of the misalignment of the position of the voxel group for convolution with the position of the kernel center, as shown in Fig. 11(a). Because of this misalignment, the convolver does not produce the correct samples for perspective projections. Fig. 11(b)

Table 1: Snap shots of pipeline data flow.

Out	Pipe 0	Pipe 1	Pipe 2	Pipe 3
Mem	a_0	b_0	c_0	d_0
Mem	d_1	a_1	b_1	c_1
W_1	$(a_0, 0, 0)$	$(b_0, 0, 0)$	$(c_0, 0, 0)$	$(d_0, 0, 0)$
Mem	c_2	d_2	a_2	b_2
W_1	$(d_1, 0, 0)$	$(a_1, 0, 0)$	$(b_1, 0, 0)$	$(c_1, 0, 0)$
W_2	$(a_0, b_0, 0)$	$(b_0, c_0, 0)$	$(c_0, d_0, 0)$	$(d_0, a_0, 0)$
Mem	b_3	c_3	d_3	a_3
W_1	$(c_2, 0, 0)$	$(d_2, 0, 0)$	$(a_2, 0, 0)$	$(b_2, 0, 0)$
W_2	$(d_1, a_1, 0)$	$(a_1, b_1, 0)$	$(b_1, c_1, 0)$	$(c_1, d_1, 0)$
W_3	(a_0, b_0, c_0)	(b_0, c_0, d_0)	(c_0, d_0, a_0)	(d_0, a_0, b_0)
Mem	a_4	b_4	c_4	d_4
W_1	$(b_3, 0, 0)$	$(c_3, 0, 0)$	$(d_3, 0, 0)$	$(a_3, 0, 0)$
W_2	$(c_2, d_2, 0)$	$(d_2, a_2, 0)$	$(a_2, b_2, 0)$	$(b_2, c_2, 0)$
W_3	(d_1, a_1, b_1)	(a_1, b_1, c_1)	(b_1, c_1, d_1)	(c_1, d_1, a_1)

$$(p, q, r) = pW_1 + qW_2 + rW_3$$

shows a pipelined convolver that produces correct samples. Table 2 also shows the two convolvers. In this example, 1D convolutions over 3 voxels are performed for 4 pipelines.

In the worst case, the grid spacing L_s in the scale-up coordinate system is half the grid spacing L_c in the compositing coordinate system, when we use power-of-2 multi-resolution datasets. In this case, the convolver taking N voxels produces only $N/2$ samples.

Given the kernel center position, which corresponds to the base plane memory address, and the slice number, it is possible to compute the positions of voxels in the convolution area in advance or on the fly. The order of voxel reads cannot be controlled, but the position of the kernel center can be controlled by choosing one of the voxel positions in the convolution area as the position of the kernel center. For example, the leftmost voxel position can be the kernel center position. Since each voxel position corresponds to a rendering pipeline, the rendering pipeline corresponding to the kernel center position receives a valid sample; the other pipelines receive invalid samples. This is illustrated in Table 2. To control this situation, a valid/invalid flag is attached to each sample, indicating the validity of the sample to the receiving pipeline. Invalid samples are discarded at the composition stage.

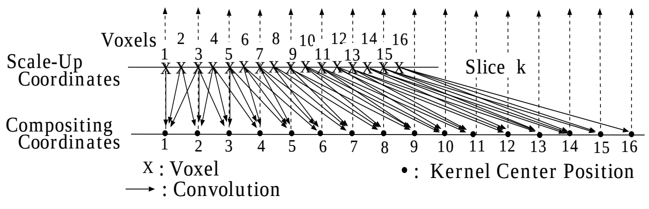
Convolution centers are not necessarily aligned with the sample positions. It causes an error if a fixed set of weights is used. This misalignment error can be corrected by computing convolution weights using a look-up table addressed by the offsets of sample positions.

4.3 Adaptive Pipelined Convolvers

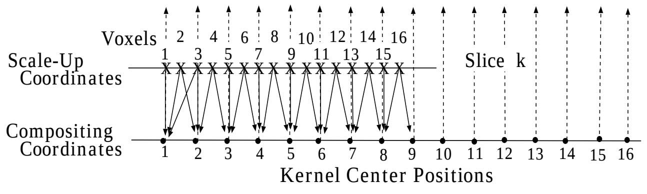
Fig. 12 shows a block diagram of an adaptive 2D convolver. Let n be the size of a dataset in one dimension, which is generally greater than the number of pipelines N_p. The selector is controlled by the status of pipeline 0, that is, whether or not the pipeline is processing the leftmost voxel in the current group of voxels. The delay of n/N_p is used to delay the operation for the time for one scanline of voxels. It is actually configured as a variable delay element for different resolution levels. For a given scaling factor K representing a resolution level, this delay element causes a delay of $(n/K)/N_p$.

Fig. 13 illustrates an adaptive 3D convolver. The structure is a direct extension of the adaptive 2D convolver with the k-direction 1D convolver added at the output of the 2D convolver with a delay of n^2/N_p. This delay element is also a variable delay element that causes a delay of $(n/K)^2/N_p$ for the scaling factor K.

The variable delay element can be implemented using a FIFO memory addressed in a circular manner by a single pointer for both read and write operations. The cycle time from one location to the same location determines the delay time, which can be easily changed by changing the maximum address value.



(a) Filtering area of Naive Pipeline Convolver



(b) Filtering area of Ideal Pipeline Convolver

Figure 11: Voxel positions vs. kernel centers.

Table 2: Kernel centers and voxel groups.

Convolver 1 (naive)		Convolver 2 (correct)	
Kernel center	voxels	kernel center	voxels
1	v_1, v_2, v_3	1	v_1, v_2, v_3
2	v_2, v_3, v_4	2	v_2, v_3, v_4
3	v_3, v_4, v_5	-	
4	v_4, v_5, v_6	3	v_4, v_5, v_6
5	v_5, v_6, v_7	-	
6	v_6, v_7, v_8	4	v_6, v_7, v_8
7	v_7, v_8, v_9	-	
8	v_8, v_9, v_{10}	5	v_8, v_9, v_{10}
9	v_9, v_{10}, v_{11}	-	
10	v_{10}, v_{11}, v_{12}	6	v_{10}, v_{11}, v_{12}
11	v_{11}, v_{12}, v_{13}	-	
12	v_{12}, v_{13}, v_{14}	7	v_{12}, v_{13}, v_{14}
13	v_{13}, v_{14}, v_{15}	-	
14	v_{14}, v_{15}, v_{16}	8	v_{14}, v_{15}, v_{16}
15	v_{15}, v_{16}	-	
16	v_{16}	9	v_{15}, v_{16}

4.4 Rendering Timings

The proposed rendering architecture is organized with the resampling module consisting of the pipelined convolvers, the multi-resolution skewed memory, the rendering pipelines, and the pixel memory. Since the real-time processing capability of the proposed architecture depends on its rendering performance, we estimate the timings for rendering volumes of practical sizes. The rendering time is directly related to the number of resampling operations to perform, which can be reduced by the use of multi-resolution data. It is upper-bounded by the total number of resampling operations without multi-resolution datasets, that is, only with original voxels.

Since the resampling and other rendering operations can be fully pipelined, the pipeline cycle time can be equal to the memory access time T_m. For a given set of rendering parameters is chosen, accesses to the voxel memory are regular and deterministic. A double

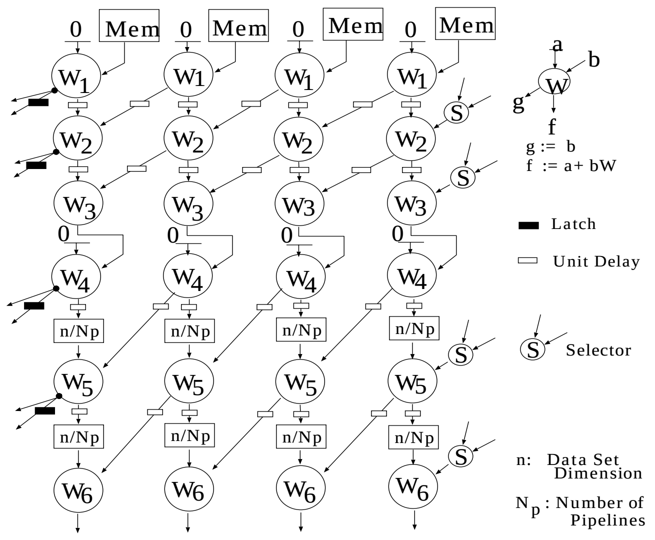


Figure 12: Adaptive pipelined 2D convolver.

buffering technique can be used to provide continuous data streams from the voxel memory to the resampling module and rendering pipelines. The pixel memory does not require much bandwidth on average, because the pixel write operations are bursty but intermittent. A simple pixel buffering technique with a FIFO memory will be enough for pixel write operations. The voxel and pixel memories can be implemented by SDRAM (Synchronous DRAM) chips to exploit their burst access mode.

Let n^3, N_p, and N_f be the volume size, the number of rendering pipelines, and the number of image frames generated per second. The total number of samples N_s to compute in each pipeline for one second is given by

$$N_s = n^3 N_f / N_p. \tag{15}$$

For each second,

$$N_s T_m \leq 1. \tag{16}$$

For a given set of parameters T_m, N_f, and N_p, the maximum dimension of volume that can be rendered is given by

$$n \leq \sqrt[3]{N_p/(N_f T_m)}. \tag{17}$$

Assuming that $T_m = 8$ ns as in a 125-MHz SDRAM chip and $N_f = 30$ frames/second, the volume dimensions computed for several values of N_f and N_p are shown in Table 3. These values verify that the proposed architecture can render volumes of practical sizes in real-time.

Voxel loading, rendering, and pixel reading can be pipelined using a double-buffering technique for the voxel and pixel memories. This allows the proposed architecture to render time-varying volume datasets in real-time.

Table 3: Maximum volume dimensions.

N_f	N_p (# pipelines)					
(frames/sec)	2	4	8	16	32	64
30	203	255	322	405	511	644
20	232	292	368	464	585	737
10	292	368	464	585	737	928

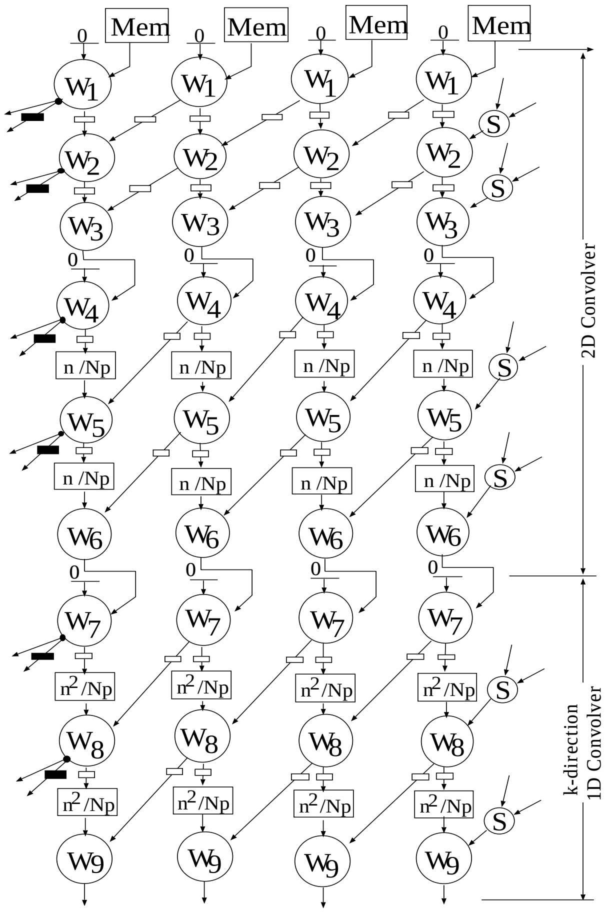


Figure 13: Adaptive pipelined 3D convolver.

Multi-resolution datasets are generated before a volume dataset is loaded into the voxel memory. They can be generated off-line by software. The set of operations to compute a single data from eight data at a lower level includes seven additions, one division (shift), and eight memory reads, and one memory write; $n^2(n-1)^2/4$ sets of operations are required for a volume of n^3. Although the number of operations can be reduced by using some optimization techniques, such as buffering and pipelining, it seems still difficult to perform these operations on the fly with the current technologies. It may be reasonable, however, to perform these operations using several frame periods for practical applications.

4.5 Scalability

Adding rendering pipelines increases the volume size for a fixed frame rate or the frame rate for a volume of fixed size. There are no architectural problems in adding rendering pipelines, because 1) in the voxel memory interface, each memory module in the voxel memory is connected one-to-one to a pipeline in the resampling module; 2) in the resampling module, the resampling pipelines communicate only with the left and right pipelines; 3) in the interface between the resampling module and rendering pipelines, each rendering pipeline is connected to one resampling pipeline; 4) in inter-pipeline communications, each pipeline communicates only with the left and right pipelines; and 5) in the pixel memory interface, the number of pixels to write is independent of the number of pipelines. Therefore, the proposed architecture is scalable.

5 EXPERIMENTS

We built a software simulator to simulate the pipeline data flow of the proposed architecture and verify the concept for both parallel and perspective projections. To compare images, we also built a screen-to-object raycasting renderer to simulate a texture mapping method that computes slices of samples perpendicular to the viewing vector and accumulates them to produce the final image. We conducted several rendering experiments with these simulators.

Fig. 14 shows a perspective image rendered from an opaque cube of size 64^3 to verify the perspective projection. The filter kernel based on the $2 \times 2 \times 2$ Lagrange formula is used in resampling.

Figs. 15 and 16 show two perspective images rendered from an opaque checker-board cube of size 128^3 (spatial frequency of 64 Hz) to explore the aliasing problem; a fully opaque dataset gives the worst case for aliasing. The image in Fig. 15 is generated by using the nearest neighbor voxel values in resampling, showing the aliasing problem clearly. The image in Fig. 16 is generated by using a $3 \times 3 \times 3$ box filter kernel in resampling, showing the antialiasing effect by convolution using multi-resolution datasets.

Figs. 17, 18, and 19 show the images rendered from the engine block of size 256^3 used in Lacroute's rendering experiments [5] with a manually adjusted opacity table. Fig. 17 shows a perspective-projection image, and Fig. 18 shows a parallel-projection image for a comparison purpose. These two images are generated by using a kernel based on the $3 \times 3 \times 3$ Lagrange formula.

Fig. 19 is a perspective-projection image generated by the screen-to-object raycasting renderer with interpolations using multi-resolution datasets. The number of the slices taken for this image is 258, about the same number of slices (256) used in Fig. 17. The two images in Figs. 17 and 19 look comparable in quality.

6 FUTURE WORK

The proposed architecture provides a freedom for adjusting the convolution kernel. Choosing a convolution kernel for the best image is a challenging problem. Especially, it is an important topic to find the kernel to maximize the antialiasing effect.

The current architecture simulator computes samples only at integral slice positions. Subslicing, that is, taking samples between integral slice positions, is expected to improve the image quality. How much can the subslicing improve the image quality? What is the limitation of subslicing?

Similarly, the image quality can be improved by additional sampling in the x and y directions. Since the proposed architecture casts a fixed number of rays, images generated by rays starting at different offsets need to be blended. This is an interesting technique worth exploring for the improvement of image quality.

Error analysis is an essential work for hardware implementation, which is most likely to use fixed-point arithmetic. The application of resampling by convolution to rendering a class of irregular volumes is another topic for study.

7 CONCLUSION

We have proposed a real-time volume rendering architecture using an adaptive resampling module for both parallel and perspective projections. It is organized as a sample-parallel architecture with a unified parallel-pipeline structure. The architecture can be easily organized in a systolic array structure for implementation on an ASIC chip. The resampling module is the key feature of the proposed architecture to address the processing variability problem caused by the diverging perspective rays. It is placed between the voxel memory and the rendering pipelines for resampling by convolutions over groups of data to make the parallel-pipeline structure regular. The use of multi-resolution datasets is another key feature to reduce the number of resampling operations. We have demonstrated that it also contributes to antialiasing.

We have shown that pipelined 2D and 3D convolvers can be implemented with $2M$ and $3M$ elements, respectively, by exploiting the properties of the sheared and scaled grids. They are a contrast to M^2 and M^3 elements required to implement general 2D and 3D convolvers. We have also described the adaptive convolvers with variable delays.

ACKNOWLEDGEMENTS

We would like to thank M. Maeda, K. Muroi, and H. Takayama of the Vision-21 project in Mitsubishi Electric Corporation for giving us an opportunity of this research and all the members of the volume graphics project at Mitsubishi Electric Information Technology Center America for their support. Especially, many thanks to Beverly Schultz for her valuable comments on the paper.

References

[1] I. Bitter and A. Kaufman. A ray-slice-sweep volume rendering engine. In *Proceedings of the 1997 SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, pages 121–130, August 1997.

[2] T. Günther, C. Poliwoda, C. Reinhart, J. Hesser, R. Männer, H.-P. Meinzer, and H.-J. Baur. Virim: A massively parallel processor for real-time volume visualization in medicine. *Computers & Graphics*, 19(5):705–710, 1995.

[3] A. Kaufman and R. Bakalash. Memory and processing architecture for 3d voxel-based imagery. *IEEE Computer Graphics & Applications*, 8(6):10–23, November 1988.

[4] G. Knittel and W. Strasser. A compact volume rendering accelerator. In *Proceedings of the IEEE Symposium on Volume Visualization*, pages 67–74, October 1994.

[5] P. G. Lacroute. Fast volume rendering using a shear-warp facrorization of the viewing transformation. Technical Report CSL-TR-95-678 (Ph.D. Dissertation), Computer Systems Laboratory, Stanford University, September 1995.

[6] M. Levoy. Efficient ray tracing of volume data. *ACM Transactions on Graphics*, 9(3):245–261, July 1990.

[7] R. Osborne, H. Pfister, H. Lauer, N. McKenzie, S. Gibson, W. Hiatt, and T. Ohkami. Em-cube: An architecture for low-cost real-time volume rendering. In *Proceedings of the 1997 SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, pages 131–138, August 1997.

[8] H. Pfister. Architecture for real-time volume rendering. Ph.d. dissertation, Department of Computer Science, State University of New York at Stony Brook, 1996.

[9] L. Westover. Footprint evaluation for volume rendering. In *Proceedings of the ACM SIGGRAPH '94*, pages 144–153, October 1994.

[10] L. Williams. Pyramidal parametrics. In *Proceedings of the ACM SIGGRAPH '83 Conference*, pages 1–11, July 1983.

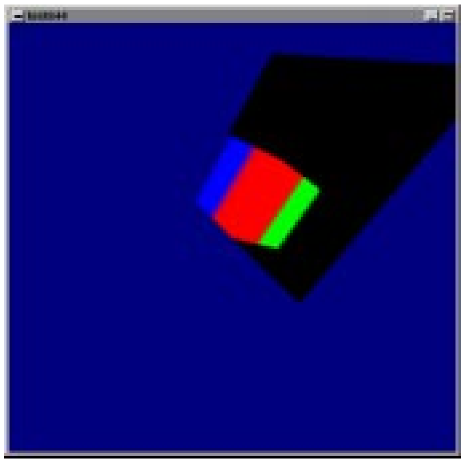


Figure 14: Perspective projection with 2 × 2 × 2 convolution.

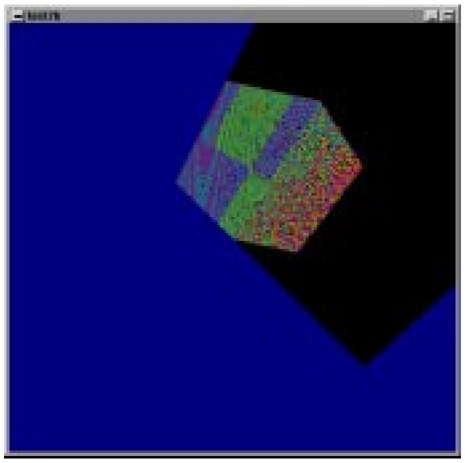


Figure 15: Perspective projection by resampling the nearest neighbor voxels.

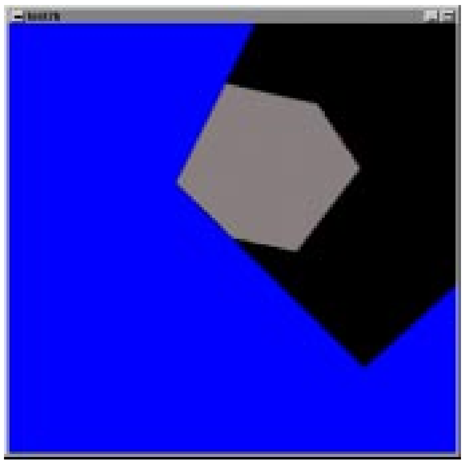


Figure 16: Perspective projection with 3 × 3 × 3 convolution.

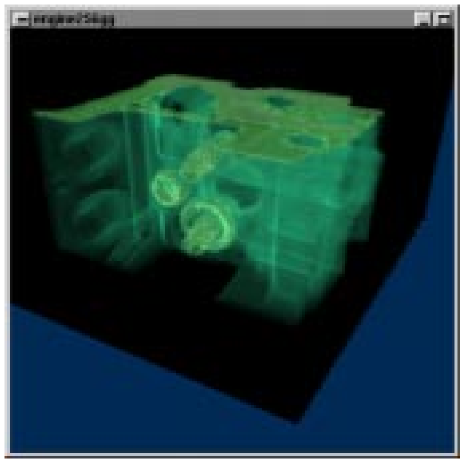


Figure 17: Perspective projection with 3 × 3 × 3 convolution.

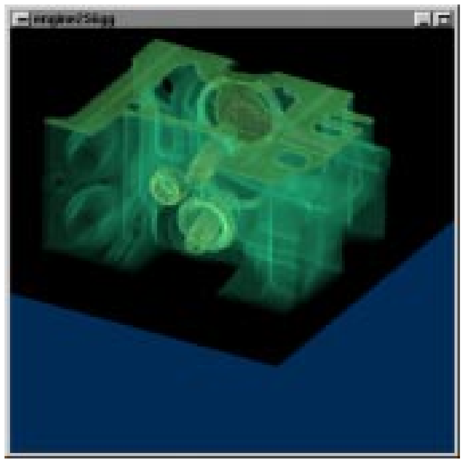


Figure 18: Parallel projection with 3 × 3 × 3 convolution.

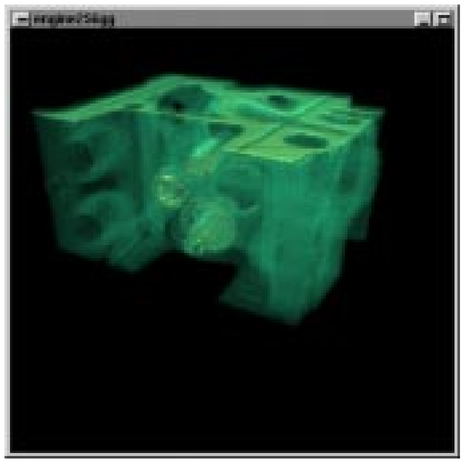


Figure 19: Screen-to-Object perspective projection by interpolations using multi-resolution datasets.