2. Previous Work

In the next two sections (2.1 - 2.2), the previous research on rendering volumes is discussed and grouped into uniprocessor methods and parallel algorithms. The last section (2.3) focuses on reconstruction error studies.

The Introduction presented an algorithmic model of volume rendering; many existing rendering algorithms deviate from that model for motivations of simplicity and efficiency. The object lattice data is often shaded and classified prior to resampling. In this case, the resampling process interpolates the color and opacity of the object lattice points rather than the scalar field. Because of the nonlinearity of the shading and classification process, this approach is more likely to produce errors and artifacts in the final images. This is analogous to the choice between using Gouraud or Phong lighting of polygons. Shading before resampling is more efficient since the normal vectors used in lighting may be precomputed and used for any view direction and because a change in view direction does not require recomputing the classification function for the whole volume. Although the result is a lower image quality, shading before resampling is a common trade-off in volume renderer implementations. Rather than dwell on the merits of either approach, shading and classification before resampling is regarded as a speedup technique that can be applied to any of the algorithms discussed here. To be consistent, all rendering approaches are introduced in the form with resampling performed before classification and shading.

2.1. Uniprocessor Methods

The Introduction outlined three common methods for viewing volumes: ray casting, splatting, and volume shearing. These methods are reviewed here focusing on how the reconstruction and resampling process is performed. There are other approaches to volume rendering. These usually involve a polyhedral decomposition of the volume [Max$^+$90] [Shirley$^+$90] [Upson$^+$88] [Wilhelms$^+$91] and are frequently used for irregular mesh data sets and to leverage polygon rendering hardware. Reconstruction in these approaches is performed by linear interpolation, so the results are no different than those obtained with direct rendering. There is no advantage to using these methods with general purpose computers since the intermediate polygonal representation is costly to produce and to render.

2.1.1. Ray Casting

Ray casting is perhaps the most commonly used volume rendering method [Levoy88] [Sabella88] [Upson$^+$88]. Rays are sent into the volume from the view point passing through each pixel in the image plane. The scalar field's value along each ray is usually sampled at a regular interval. Figure 2.1 depicts the object lattice as dark, dashed lines

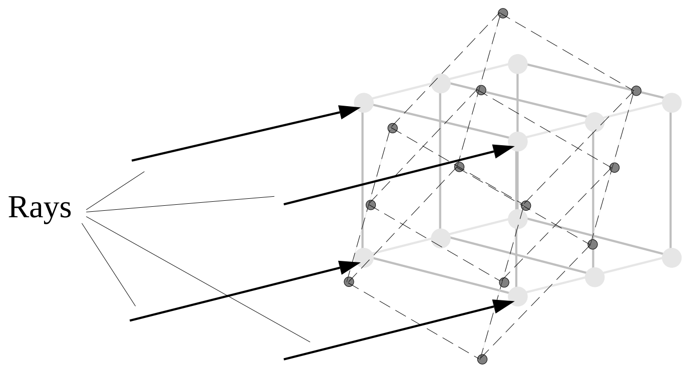


Fig. 2.1 - Ray casting

and the image lattice as gray solid lines. Four typical rays are shown passing through image lattice points in the volume.

Trilinear interpolation is used to reconstruct the field data at the image lattice points. The computation proceeds ray-by-ray as shown in the program fragment below. New sample points are shaded, classified, and composited (front-to-back) with the accumulated color along the ray to that point.

```
for each pixel {
    for each z-step along ray {
        reconstruct field $F$ at new sample point by trilinear interpolation;
        shade and classify new sample to obtain $\Omega$ and $E$;
        compute new segment $I$ and $T$ from averaged $\Omega$ and $E$ of new and last sample;
        composite new ray segment $I$ and $T$ behind accumulated ray color;
}  }
```

This code sequence computes Ω and *E* values from the reconstructed field value *F* at the new sample points along the rays. This approach is used by Upson and Keeler [Upson+88] and for the comparison tests in this work. Other implementations evaluate the the Ω and *E* functions on the object lattice points and reconstruct those values on the image lattice [Levoy88], thereby gaining efficiency by using a precomputed data gradient when shading the object lattice. This approach trades some image quality for efficiency, but it may be appropriate in some applications. The shade-before-reconstruction computation sequence is shown below to contrast the reconstruct-before-shade approach shown above.

```
shade and classify object lattice data f to obtain $\Omega$ and $E$;
for each pixel {
    for each z-step along ray {
        reconstruct $\Omega$ and $E$ at new sample point by trilinear interpolation;
        compute new segment $I$ and $T$ from averaged $\Omega$ and $E$ of new and last sample;
```

```
            composite new ray segment $I$ and $T$ in front of accumulated ray color;
    }  }
```

There is a compromise between these two extremes. To keep the efficiency high, the normals are precomputed on the object lattice, while as much processing as possible is deferred until after reconstruction. The object lattice points are shaded in every frame based on the precomputed normals, the current view, and light direction. If the classification is deferred until after reconstruction, the benefit of more accurate classification is obtained with little or no additional expense. This compromise approach is shown below and has been used here at UNC by Timothy Cullip and others with dramatically better results than those obtained with the pre-classification method; in particular, surfaces are clearer and more well-defined than the blurry surfaces often obtained with the pre-classification method.

```
shade object lattice data $f$ to obtain $E$;
for each pixel {
    for each z-step along ray {
        reconstruct field $F$ and emittance $E$ at new sample by trilinear interpolation;
        classify the reconstructed $F$ value to obtain $\Omega$;
        compute new ray segment $I$ and $T$ as averaged $\Omega$ and $E$ of new and last sample;
        composite new ray segment $I$ and $T$ in front of accumulated ray color;
}  }
```

2.1.1.1. Ray Casting Speedups

Many techniques exist to speed up ray casting of volumes. The shade-before-reconstruction technique was described above. Others are adaptive ray termination, adaptive sampling, progressive refinement, hierarchical data structures, and distance transformations. This abundance of optimizations is unique to ray casting due to its pixel-by-pixel processing order. This processing sequence is known as an *image order* algorithm.

Adaptive ray termination [Upson$^+$88] [Levoy90] [Danskin$^+$92] may only be applied when ray samples are computed and composited front-to-back. As samples are taken along a ray, the ray accumulates color and opacity. When the opacity exceeds a threshold, further samples are not taken since their contribution to the final ray color is occluded by the opaque material already accumulated. For images that are rendered with a classification function that produces high opacity, this technique can save significant work.

Adaptive sampling is the technique of sampling only a subset of the screen pixels unless the variance of neighboring samples is above a threshold [Whitted80]. In the areas of high variance, additional pixels are sampled. This is recursively done until either the variance is below the threshold, or the sampling has progressed to the pixel level. The color of pixels not sampled in low variance areas is interpolated from neighboring sampled pixels. Pixel sampling is done in a recursively applied pattern. Marc Levoy used a rectangular subdivision whose recursion is similar to that in quadtrees [Levoy90b]. Shu and Liu show that recursive triangular subdivision patterns are generally more efficient than rectangular patterns since such patterns adapt more locally and thus result in fewer total pixels sampled [Shu+91].

Progressive refinement is the technique of producing "rough" images at high frame rates when a user is modifying view parameters and progressively better images when user input ceases [Bergman+86]. With this technique users get the benefit of high frame rates while navigating the data and high quality images when users stop to inspect an image in more detail. With a ray casting renderer, a rough image is computed quickly by undersampling the image lattice. Undersampling may be done in several ways:

1 - regular but sparse sampling in all three dimensions,
2 - raising the threshold on the variance allowed in adaptive sampling,
3 - lowering the alpha cutoff threshold of accumulated opacity.

Any or all of these may be used to lower the number of ray samples computed. Image pixels that are not sampled directly have their colors interpolated from neighboring computed pixels. To minimize aliasing artifacts, the object data should be low-pass filtered proportionally with the degree of undersampling. By precomputing a pyramid of multiple resolution filtered data sets, the most appropriate set can be rendered for a given level of undersampling. By using the filtered data sets, the undersampled images will appear blurry but contain fewer aliasing artifacts. It is a subjective argument as to which a user finds more desirable. Marc Levoy and Ross Whitaker used pyramidal data to subsample a volume based on the gaze direction of the user [Levoy+90].

Another ray casting speedup involves the use of octrees. Octrees are hierarchical spatial decomposition graphs. Each parent node has eight children representing the eight octants of space into which a volume is decomposed. Volume data is preprocessed so that its associated octree representation contains a descriptor at each node. The descriptor conveys a summary of the data in the object lattice subvolume associated with the octree node. At sample points along a ray, the octree is traversed to access the descriptors for the point to be sampled. The descriptors allow empty subvolumes to be skipped since they will not contribute to the image [Levoy90]. They may also indicate the appropriate sample density for a subvolume [Danskin+92].

A distance function may be used to speed up ray casting [Zuiderveld+92]. This method computes a volume that contains the radial distance from each point in the data volume to the closest "interesting" point, where "interesting" means above a threshold. This distance value allows sample points in empty regions to be skipped - just as an octree does. This technique also helps avoid sub-sampling artifacts that occur with adaptive sampling since small features will not be missed.

2.1.2. Splatting

This method, in its general form, convolves each object lattice point with a 3D reconstruction filter and accumulates the contribution of the filtered points on the image lattice. In practice, this is not done due to the computational expense of three-dimensional convolution. Implementations prior to this date have approximated the ideal by limiting the spatial domain of the convolution to two dimensions [Westover91]. Using this approximation, object lattice slices are projected onto image lattice planes. The object lattice is cut into slices along the plane most perpendicular to the view direction. The 2D slice image is reconstructed from the points in each object lattice slice and projected onto an image lattice plane. The reconstruction is often done with a truncated Gaussian filter since its bandwidth $\times$ area product is minimal. This property means that for a given spatial extent, a Gaussian low pass filter passes the least possible high frequency energy. This is desirable since it minimizes aliasing during the reconstruction process. The slice image is resampled and accumulated on a plane of the image lattice. Figure 2.2 illustrates two points in a slice projecting onto an image lattice plane. The textured ellipses in the figure represent the projected filter kernels. The kernels shown here have a radial extent of one object lattice coordinate. Typical filter extents range from one to two so their overlap is considerable, resulting in each image lattice pixel accumulating energy from many object lattice points. Since the filter is position-invariant for affine transformations, software lookup table methods are often used to quickly approximate it [Westover89]. Polygon rendering hardware can generate filter kernels for perspective projections [Laur91] [Neumann92]. The reconstructed field values on the image lattice are shaded and classified prior to being composited to produce an image. This method constrains the number of image lattice planes to be equal to the number of object lattice slices. Each image lattice plane is a reconstructed, shaded, classified, and projected image of a single slice of data. A volume image is created by compositing the projected slice images together. Since each slice image is independent, as the view direction changes, the slices just "slide" relative to each other. Both ray casting and splatting

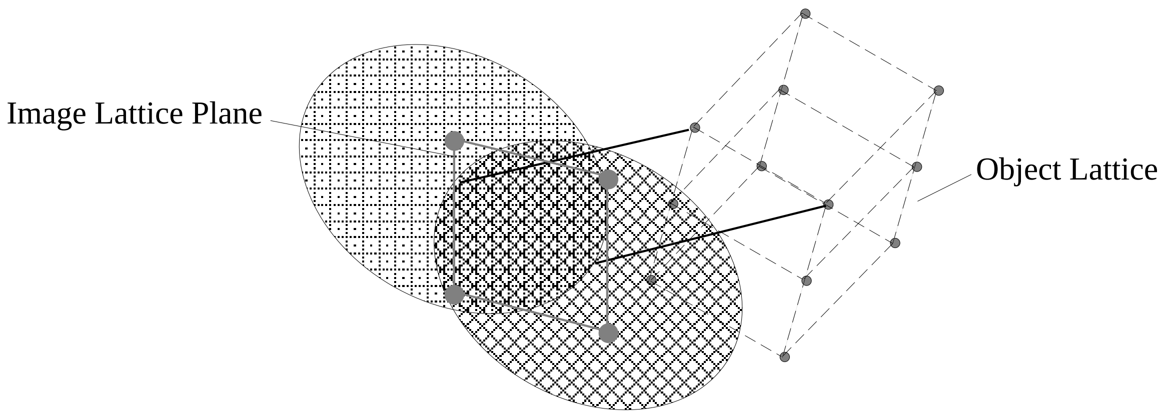


Fig. 2.2 - Splat kernels

produce multiple two-dimensional images and composite them. In ray casting the images lie perpendicular to the view direction. In splatting the images lie in a plane of the object lattice and are projected onto the image plane. Both methods are equally valid approaches to approximating samples along the intensity integral of equation 1.3. The resample-before-shade sequence of the splatting approach is given below.

```
for each slice of the object lattice {
   for each point in a slice {
      filter and project a point ($f * K$) onto an image lattice plane;
      accumulate energy at image lattice points to produce $F$;
   }
   shade and classify image lattice points to obtain $\Omega$ and $E$;
}
compute segment $I$ and $T$ from averaged $\Omega$ and $E$ of two adjacent slices;
composite all segment $I$ and $T$ values to produce final image;
```

The shade-before-resample variation of splatting is shown below. It was used by Westover, Laur and Neumann in their implementations.

```
shade and classify object lattice points to obtain $\Omega$ and $E$;
for each slice of the object lattice {
   for each point in a slice {
      filter and project a point ($\Omega * K$ and $E * K$) onto an image lattice plane;
      accumulate energy at image lattice points to incrementally produce $\Omega$ and $E$;
}  }
compute segment $I$ and T from averaged $\Omega$ and $E$ of two adjacent slices;
composite all segment $I$ and $T$ values to produce final image;
```

Little has been discovered to enhance the speed of this technique. Most effort has gone into optimizing the reconstruction process. There are no analogous methods to those available with ray casting. Splatting is an object-order method due to the algorithm's sequential traversal of the data set. Object-order approaches do have one important optimization - data values below a threshold of interest may be ignored. For many data sets, more than fifty percent of the data points are uninteresting. This percentage is clearly dependent on the data and classification; data sets with eighty and ninety percent of their points below the threshold of interest are not uncommon.

One efficiency enhancement for splatting that uses a hierarchical representation of a volume was presented by David Laur and Pat Hanrahan [Laur$^+$91]. The method constructs an octree representation of the volume where each node contains *I* and *T* values that approximate all that node's children and the error associated with the approximation. This structure is recomputed every time the classification changes. The shading function is restricted to be view-independent. The octree is traversed in view-order to generate an image. At each node the node representation error is compared to an allowable error threshold. If the threshold is greater than the node representation error, the node is splatted and none of that node's children are visited. If the threshold error is exceeded, then the node's children are recursively visited, to the leaf nodes if necessary.

2.1.3. Volume Shearing

This approach applies a sequence of three 1D transformations to the object lattice, transforming it one dimension at a time into the image lattice. An affine view transformation may be decomposed into three sequential 3D shear operations [Hanrahan90]. Perspective may be applied through two additional 1D transformations. A 3D shear is effected by a 1D transformation of the form $x' = Ax + B$ where A and B are constant for points along the transformation axis. Since these transformations may use only a 1D reconstruction filter, cubic splines are commonly used to facilitate the resampling. Once resampled, the volume lies on the image lattice and is ready for shading, classification, and compositing. Figure 2.3 illustrates the shearing steps. The gray grid is a plane of the image lattice. The image lattice x-axis is oriented roughly left to right and the y-axis runs roughly vertically. The object lattice starts out unaligned with respect to the image lattice. The first shear in this example resamples the data along the object axis that lies closest to parallel to the image x-axis. The result is that all the new object lattice points now lie on integer image lattice x-coordinates. The second shear is along the object axis closest to parallel to the image y-axis resulting in the new object lattice points lying on integer image x and y-coordinates. The final shear fully aligns the points with the image lattice by resampling along the object axis closest to parallel to the image z-axis. In practice the sequence of shears is determined by view direction such

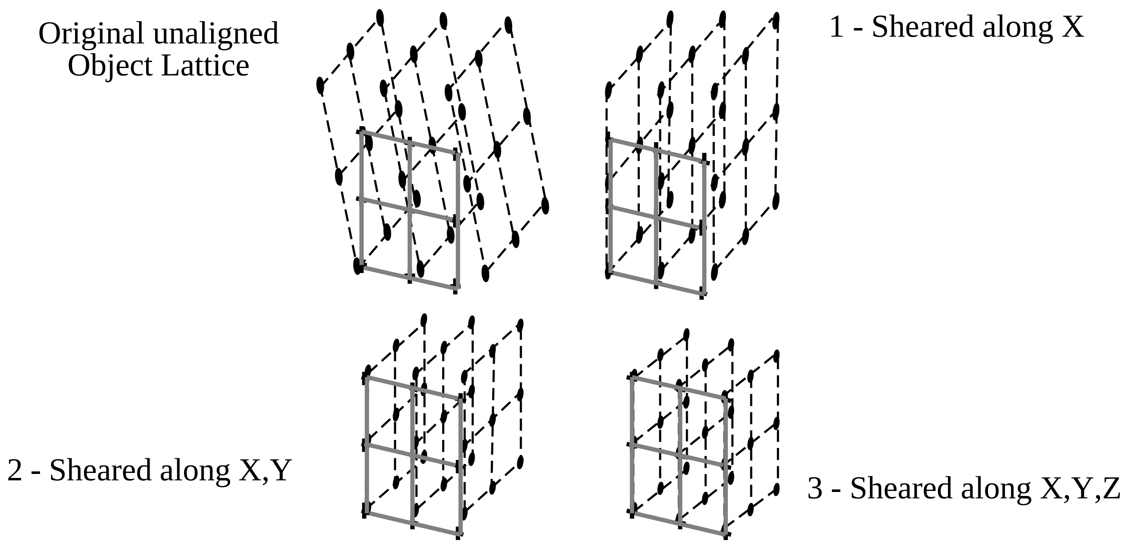


Fig. 2.3 - Volume Shearing

that minimal signal degradation occurs. For further detail, the reader is referred to Pat Hanrahan's ample description of the technique and its practical application [Hanrahan90].

The following code fragment conveys the simple sequencing of this method in the reconstruct-before-shade form. Its other forms are simple extensions of those shown for ray casting and splatting, and thus are not included here. We assume the appropriate order of shearing is the same as used in the previous example.

```
shear volume $f$ along x-axis;
shear volume $f$ along y-axis;
shear volume $f$ along z-axis;
shade and classify resampled volume to obtain $\Omega$ and $E$;
computer $I$ and $T$ as the average of successive points along the image lattice z-axis;
composite $I$ and $T$ of all segments along the image z-axis;
```

Volume shearing, like splatting, is an object order technique which can test points against a threshold of interest. To date there are no other published methods of optimizing this technique. In the implementation described in the next chapter, we show that images may be rendered by performing only two volume shears. This saves over one-third of the transformation and reconstruction time, and about one-half of the compositing time without incurring additional cost elsewhere in the algorithm.

2.2. Parallel Algorithms

Many previously published parallel approaches to direct volume rendering target application-specific hardware. Recently, work has been done for general purpose multicomputers. Where possible, these works are grouped according to two issues: whether an image or object-partition parallel algorithm is used, and whether an image or object-order rendering method is used. These distinctions are refined further in chapters three and five.

2.2.1. Object Partition, Object-Order Rendering

In the only comparative study of parallel volume rendering algorithms published to date, Judy Challinger compares the performance of a cell projection rendering method to ray casting on a BBN TC2000 multicomputer [Challinger91]. This system uses the shared memory communications model with a network that essentially provides a full N-to-N crossbar interconnection of the nodes and their local memory. The cell projection results are described here. Section 2.2.4. describes the ray casting results. The cell projection method [Max+90] [Wilhelms+91] is an object-order rendering method which integrates through projected volume cells. The integration at each pixel is based on front and rear cell-face values interpolated from the cell vertex values. The algorithm uses an object partition where interleaved cells are assigned to nodes on demand. Here increased communication costs arise from the need to composite into a globally accessible frame buffer for each pixel of each cell. This increased cost is not directly reported by the author, but it is consistent with the increased total projection cost measured as the the number of nodes increases.

2.2.2. Object Partition, Image-Order Rendering

An algorithm implemented on Pixel-Planes 5 introduced a two-level node organization where node groups, or clusters, act as a single "logical node" in a partition [Yoo$^+$91]. Considering each cluster as a single node makes this algorithm an object partition. Clusters are each assigned slab subsets of the data and are responsible for rendering a complete image of their data. Nodes within a cluster all have private copies of the same subset of data. The final images from each cluster are sent to a separate set of processors that composite them in the proper order and update the frame buffer. Within each cluster any partition may be used; in this case, an interleaved image partition with image order rendering by ray casting is used. Within a cluster, scan lines are distributed on demand to the nodes for ray casting. This provides dynamic load balancing within clusters. The load balance among clusters is set by the slab data distribution which is fixed in this implementation. There is no communication penalty for the interleaved image lattice since nodes only reference volume data within their local memory. This rendering program, called VVEVOL, will be referred to in later chapters.

In another Pixel-Planes 5 implementation called VOL, nodes produce a local image of their block data subsets by ray casting [Yoo$^+$91]. The local images are sent to the SIMD pixel-processors for compositing and transmission to the frame buffer. This algorithm's lack of a load balancing mechanism limited its performance. In later chapters a similar algorithm is described with a load balancing method which overcomes the performance limitations.

2.2.3. Image Partition, Object-Order Rendering

Lee Westover introduced two parallel splatting algorithms. The earlier design was implemented using multiple processors to traverse and shade their slab subset of data. The resulting interesting tuples are sent to a central "splat-server" where they are composited into a final image [Westover89]. As the number of traversal processors increases, the single splat-server becomes a bottleneck. The proposed later algorithm addresses this problem by having each node traverse a block of the volume and act as a splat-server for a subset of scan lines [Westover91]. Other nodes collect the splatted scan lines for compositing and writing to a frame buffer. The image lattice is interleaved in one dimension (scan lines) for load balancing. This interleaving raises the communication cost by a factor equal to the number of scan lines covered by the splat kernel. The target system has a shared memory communications model provided by a global-access bus.

A Pixel-Planes 5 implementation uses the SIMD pixel-processor arrays to generate the splat kernel as a "graphic primitive". Multiple MIMD traversal processors feed interesting points from their slab data subsets to the SIMD splat renderers [Neumann91]. Message passing communications support is provided by the operating system over an eight channel ring. This is an image partition algorithm using separate traversal and splat processors. In this algorithm the slabs are interleaved in one dimension (as slices) with no increase in communication cost. This shows that an image partition may interleave the object lattice without incurring a penalty. The converse of this is also true - an object partition algorithm may interleave the image lattice without penalty. This issue is explored further in chapters five and six.

2.2.4. Image Partition, Image-Order Rendering

The first volume rendering method implemented on Pixel-Planes 5 uses multiple SIMD shaders and MIMD ray casting processors interconnected by a ring network [Levoy89] [Yoo+92]. This is an image partition algorithm unique in its use of SIMD processor arrays to preform the classification and shading. Ray casting nodes are statically assigned interleaved image regions. The SIMD arrays store interleaved data blocks. High latency for data access is introduced by the message passing communication model and the decision to perform the shading in the SIMD arrays. The resulting latency of up to 4 ms. per data block access limits the performance and speedup attainable by this implementation.

Jason Nieh and Marc Levoy implemented an interleaved image partition with an optimized ray casting algorithm on the Stanford DASH [Nieh+92] . This system has a shared memory model and a mesh network topology [Dash]. The ray casting renderer uses adaptive termination, adaptive sampling, and an octree hierarchy to optimize the ray casting. The image lattice is interleaved in two dimensions with square screen regions to provide load balancing. The low latency communications network (~3μs per remote access) keeps the memory access overhead in the range of twenty to thirty-eight percent of the rendering time. For a 48 node system, speedups of forty and thirty are given for non-adaptive and adaptive sampling, respectively.

As the second part of a comparison study, ray casting was done on a BBN TC2000 multicomputer [Challinger91] using an image partition algorithm with interleaved scan lines or pixels assigned to nodes on demand. Since the image lattice is interleaved, a communications penalty for accessing data values from multiple nodes is exacted. This effect is not directly reported but may be inferred from the data presented in the case where interleaving is on a pixel basis. One conclusion drawn from this work is that ray casting scales better than projection methods on shared memory systems, demonstrating almost linear speedup from ten to one-hundred nodes when scan line interleaving is used.

Brian Corrie and Paul Mackerras use ray casting with an interleaved image partition algorithm on a Fujitsu AP1000 multicomputer with a two-dimensional torus network topology [Corrie+92]. A shared virtual-memory system is built on top of the message passing library provided by Fujitsu. They compare pixel, scanline, and rectangular-region image lattice interleaving methods and conclude that rectangular-region interleaving is an effective method of load balancing and incurs minimal communications penalty.

An algorithm implemented on an nCUBE-2 model 6410 also uses the notion of node clusters [Montani+92]. The nCUBE has a message passing network model and a hypercube topology. Clusters are assigned interleaved scan lines of the image, making this algorithm an interleaved image partition. Within each cluster any partition may be used, and in this case the authors use an object partition with image order rendering by ray casting. Within a cluster a complete copy of the data set is distributed among the nodes. Each cluster node ray-casts its slab subset of the volume. Rays that pass into neighboring slabs are passed to the corresponding node within the same cluster for continuation. Rays stay within their cluster until completion when the final color is sent to a global access frame buffer. The hierarchical partitions localize communication within a cluster. This removes the penalty for the interleaved image lattice and provides a statically set load balance between clusters. Load balance within clusters is achieved by moving the slab boundaries between nodes based on a statistical sampling of the work-load from a subset of the rays to be traced. Results indicate a speedup of 102 for a 128 node system with a cluster size of two. Efficiency drops sharply as the cluster size increases due to greater communication and synchronization within each cluster.

2.2.5. SIMD, Vector, and Custom Hardware Approaches

Tim Cullip recently implemented a volume renderer using the SGI RealityEngine™ [Cullip93]. This system has custom VLSI pixel-processors that are capable of mapping 3D textures onto polygons. The volume is defined as a two-component 3D texture. One component is the original data value, and the other is a pre-computed shading intensity. Equally spaced polygons are embedded in the texture. The sampled textured data value at each polygon pixel is passed through classifier lookup tables to produce intrinsic color and opacity. Intrinsic color is multiplied by the shading intensity texture value to produce a final color. The colors and opacities of successive polygons are composited to produce the final image. This method is limited to using fixed, object-space lighting, and texture memory size is limited to one mega-point data sets. Larger data sets must be partitioned and loaded into texture memory in chunks, at the expense of about 0.1 seconds per load. A one mega-point data set is rendered on a 512^2 screen at about ten frames per second - the fastest performance reported for a commercially available system. This performance illustrates the benefit of fast reconstruction and resampling and will likely inspire further research into other VLSI methods of accelerating reconstruction.

One of the first published direct volume rendering algorithms uses a four-channel SIMD processor to project a volume by sequential 2D image transformations and compositing [Drebin+88]. Each channel operates on a separate element of a data point's ⟨R,G,B,Alpha⟩ tuple and has simultaneous access to a portion of a shared memory [Levinthal+84]. The reconstruction method is similar to that described for volume shearing. Each 2D transformation is performed by two 1D cubic interpolations. The per-tuple-element parallelism is uniquely suited to the CHAP hardware but not extendable to larger numbers of processors.

Peter Shröder and James Salem demonstrate a data-parallel volume shearing method on the Connection Machine CM2 [Shröder+91]. This algorithm scales well on the CM2 class of machines, but, as presented, it is limited to affine projections and rotation-only transformations.

Peter Shröder and Gordon Stoll used a line drawing algorithm to do ray tracing on the CM2 and Princeton Engine [Shröder+92]. The Princeton Engine is a special purpose ring of 2048 DSP processors. It is remarkably fast at performing this algorithm; the system achieves speeds in excess of thirty frames per second for 128^3 volumes.

The MasPar MP-1's indirect addressing is used with a modified volume shearing algorithm to perform arbitrary perspective transformations [Vénzia+92]. The authors demonstrate linear speedup for 1K and 16K node systems. A 128^3 volume is rendered in under 0.5 seconds using linear interpolation during the shearing reconstruction.

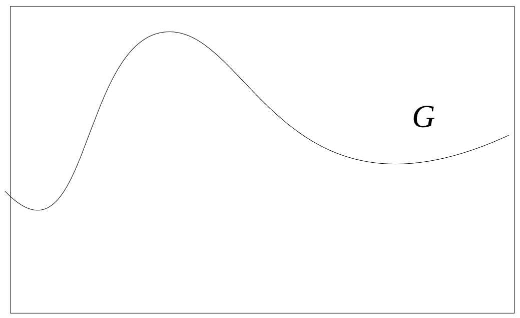
Vector and parallel processors are used to perform affine transformations efficiently on volumes [Machiraju+92]. Example implementations are compared on the Cray/Y-MP, the IBM Power Visualization System, and a Silicon Graphics multiprocessor workstation.

Several custom hardware systems utilize multiple traversal processors, each responsible for a sub-volume of data [Goldwasser+85] [Yazdy+90]. However, these schemes rely on a single compositing processor which becomes a bottleneck as more projection processors are added.

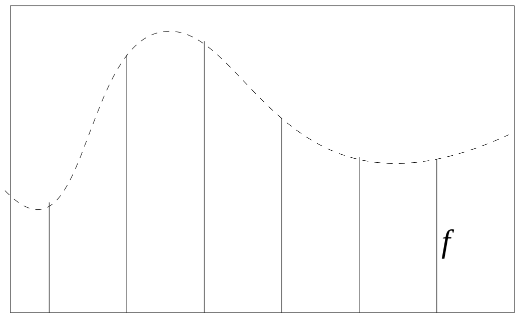
2.3. Reconstruction Methods

Reconstruction is the process of applying a filter to discrete sampled signal points, thereby creating a continuous function. There are numerous good references on the signal processing theory of reconstruction [Castleman] [Jain]. A simplified, one-dimensional illustration of signal processing in volume rendering is shown in figure 2.4. Using the notation of section 1.1, the original signal G is a real-world continuous function. It is uniformly sampled at a rate at least twice the highest frequency component in G. The discrete samples f are filtered by a reconstruction filter K to produce a continuous function *F* which ideally should be equivalent to *G*. Figure 2.4 shows reconstruction by a triangle filter (linear interpolation) in the lower image. The reconstructed function *F* is then resampled as necessary.

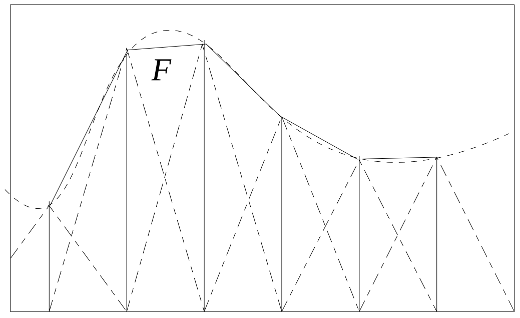
There are two possible error sources in this process. Real world signals are not infinite in extent; they are truncated or windowed to a size suitable for processing. This effectively adds high frequency energy to the signal spectrum. Realizable filters are not ideal. They always leak some energy beyond their cut-off point. Even if the windowed *G* function is filtered, it will contain some energy at frequencies beyond one-half the sample rate. Therefore the discrete samples will contain some aliased energy. This source of aliasing has been called *prealiasing* [Mitchell+88]. The second source of error, called *postaliasing,* is due to a poor choice of reconstruction filter. Since the reconstructed signal *F* will be resampled, it must contain no high frequencies that are above one-half the resampling rate. It is desirable to have a reconstruction filter *K* that minimizes these high frequencies while reconstructing *G* as accurately as possible. Three filter functions are compared in chapters three and four for their accuracy in reconstructing *G*. Below are summaries of previous efforts in evaluating the reconstruction errors produced by these filters.



Real-World Signal



Sampled Signal



Reconstructed Signal

Fig. 2.4 - Signal processing in volume rendering

Mitchell and Netravali published a comparative study of separable cubic filters applied to 2D image reconstruction [Mitchell+88] in which a family of two-parameter filters were subjectively evaluated by a test group. The use of the signal derivative to minimize postaliasing is also presented. Based on prior work by Robert Keys [Keys81], a one-parameter subset of the two-parameter family is shown to have quadratic convergence of the error (*F* - *G*) with respect to the sampling rate. This family of filters contains the Catmull-Rom spline which actually has cubic convergence and is one of the filters compared in chapters three and four.

Robert Keys derives the Catmull-Rom filter as the only exact-matching cubic filter with cubic convergence [Keys81]. He also shows that fourth-order convergence is the highest order possible from piecewise cubic polynomials but only obtained by increasing the support of the filter to six sample points. In his implementation timing tests, the cubic filter required approximately twice the computation time of linear interpolation. The Catmull-Rom filter has also been shown to be the optimal cubic filter by analysis in the Fourier domain [Park$^+$83].

Roy Hashimoto compares the two-parameter family of separable cubic filters applied to 3D volume reconstruction [Hashimoto90]. Using isosurface rendering, he concludes that blurring and ringing produced by cubic filters produce more objectionable artifacts than those produced by trilinear interpolation. He also points out the inaccuracies of linearly interpolating gradients during shading; he recommends cubic interpolation of those instead.

A method of projecting rectilinear cells and a comparative study of applicable reconstruction and integration methods was published by Wilhelms and Gelder [Wilhelms$^+$91]. This work also showed the effects of varying precision in the compositing process.