Shear-Warp Volume Rendering Algorithm

- So far we have seen ray-casting, an image-order algorithm; and splatting, an object-order algorithm
- Today we will see a *hybrid* volume rendering algorithm that exhibits characteristics of both image-order and object-order algorithms
- We will see the *shear-warp* algorithm, which is a very efficient algorithm for volume rendering that can be implemented in software (and hardware) and generate several frames per second
- The downside is that we lose a small amount of quality
- The idea is to render the volume onto an image plane parallel to one of the volume faces, and then *warp* or distort this intermediate image to generate the desired image
- It avoids the computational expense involved in rendering volumes at arbitrary rotations
- This process is often referred to as a "shear-warp factorization of the viewing matrix" in the literature

Motivation of this Approach (1994)

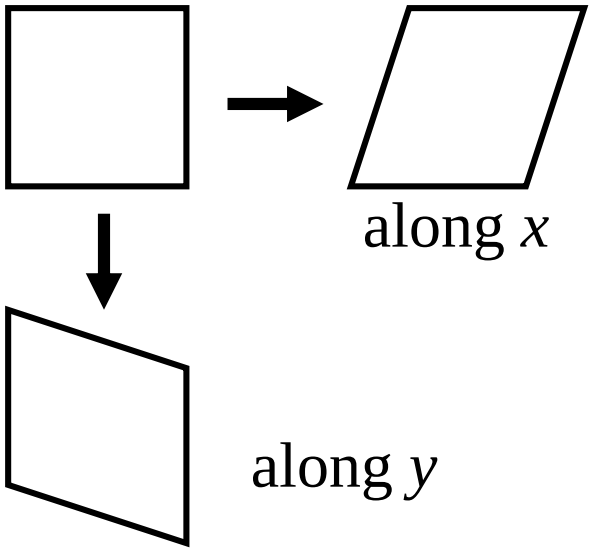
- Image-order algorithms involve traversing the volume and processing voxels encountered along the ray
- Hence, each voxel is visited many times
- This means that if a hierarchical data structure is used to accelerate the ray traversal (to skip transparent voxels), we wind up performing the same computations over and over
- Object-order algorithms (as of 1994) could not perform the ray-casting equivalent of early ray termination
- Hence, although splatting was faster than ray-casting, it still could not generate images at real-time frame rates
- The shear-warp algorithm was proposed to avoid these two problems and to be an efficient software-only solution
- We will look at hardware-based rendering next week

2D Shear

- In 2D, shearing can take place along the *x* direction or along the *y* direction
- Note that *x* shearing is proportional to the *y* coordinate

$$\begin{bmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

shear in *x* direction



$$\begin{bmatrix} 1 & 0 & 0 \\ b & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

shear in *y* direction

3D Shear

- In 3D, shearing can take place along x, y or z
- (x,y)-shear, (x,z)-shear, (y,z)-shear

$$\begin{bmatrix} 1 & 0 & sh_x & 0 \\ 0 & 1 & sh_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

(x,y) shear

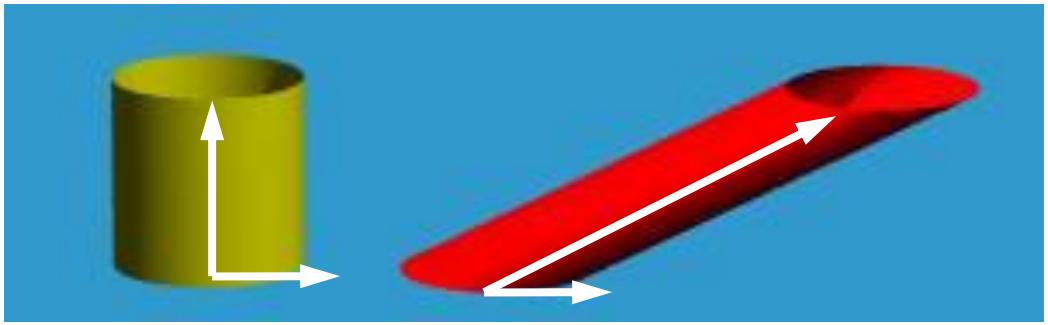
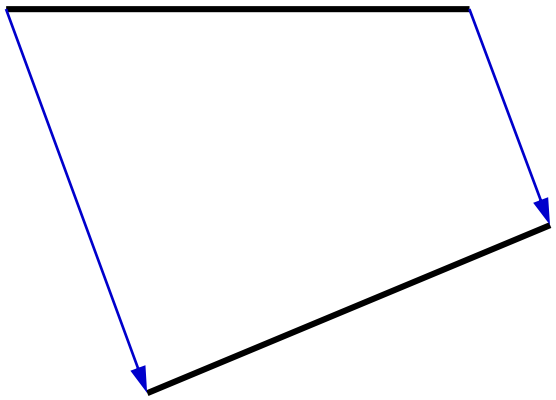
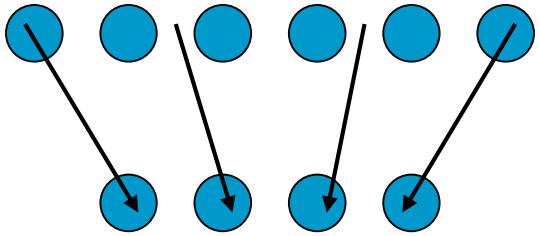


Image Warp

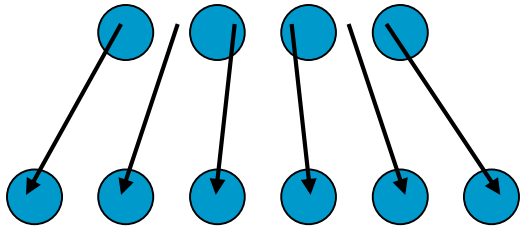


- An image warping is simply a mapping from the pixels in one image to pixels in another
- If the two images are not of exactly the same resolution (as is the case above) we need to use interpolation

- Bilinear interpolation is one possibility
- If high-res to low-res, we perform some kind of image shrinking

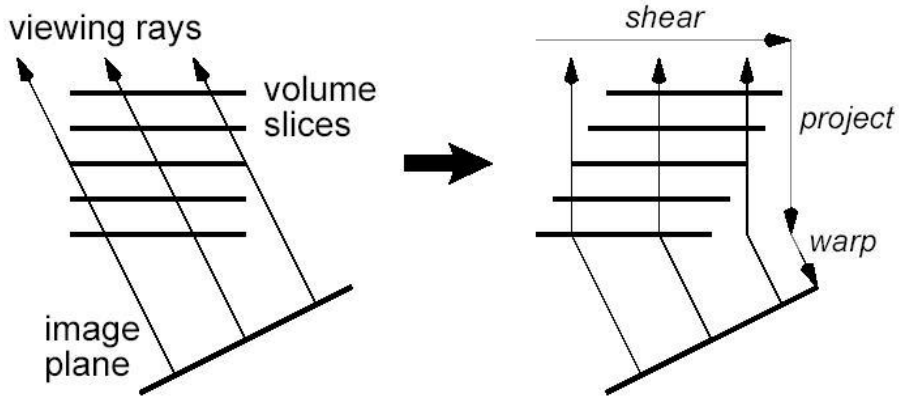


- Otherwise, it's image stretching



Shear-Warp Algorithm Overview

- The main process in the shear-warp algorithm is to simulate rotations by *shearing* the volume, rendering this sheared volume onto an intermediate image plane parallel to the near face, and then *warping* the intermediate image to the final image plane
- Sheared object space

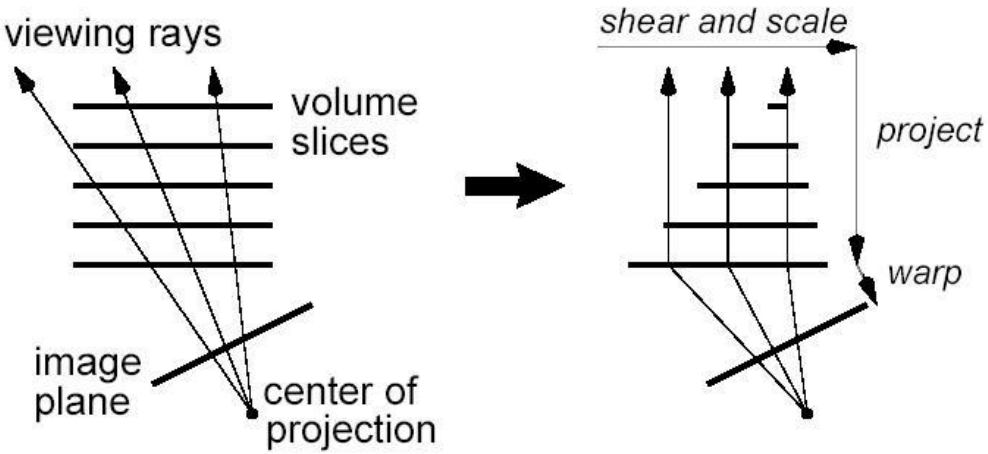


Overhead view (orthographic projection)

- By construction, in sheared object space all viewing rays are parallel to the third coordinate axis, which we will assume is *z*

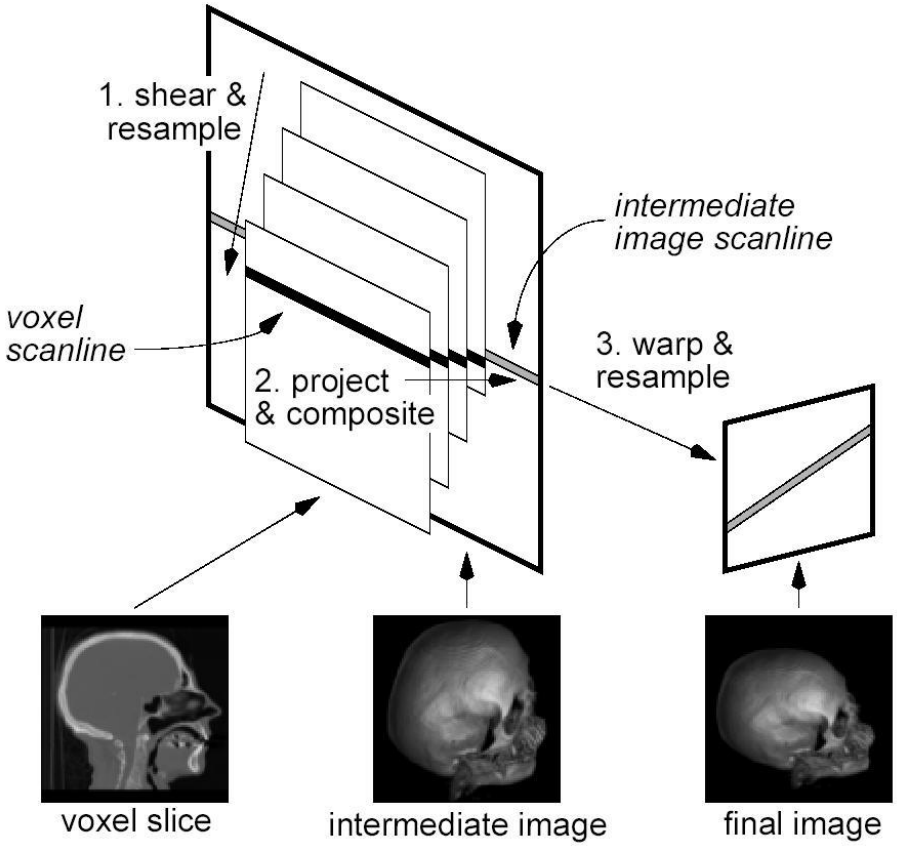
Perspective Projection

- It also works for perspective rendering if we also scale the distant slices



Shear-Warp Algorithm Overview

1. Transform the volume data to sheared object space by translating and resampling each slice. For perspective transformations, also scale each slice.
2. Composite the resampled slices together in front-to-back order. This step projects the volume into a 2D intermediate image in sheared object space.
3. Transform the intermediate image to image space by warping it. This second resampling step produces the correct final image.



Properties of Shear-Warp Algorithm

1. Scanlines of pixels in the intermediate image are parallel to scanlines of voxels in the volume data.
2. All voxels in a given voxel slice are scaled by the same factor.
3. For parallel projection, every voxel slice has the same scale factor, and this factor can be chosen arbitrarily. In particular, we can choose a unity scale factor so that for a given voxel scanline there is a one-to-one mapping between voxels and intermediate-image pixels.

- This third property initially avoids the problem of how to map multiple voxel scanlines to a single pixel scanline and vice versa (e.g., to specify an arbitrary image size).
- The warping step actually takes care of this issue, as we'll see later.

Voxel Scanline Traversal

- Since we have a very strict procedure for traversing the voxels, we can exploit this knowledge and incorporate some optimizations to speed things up
- First optimization: use run-length encoding (RLE)
- Run-length encoding stores a sequence like

  44444999933222555788

  as

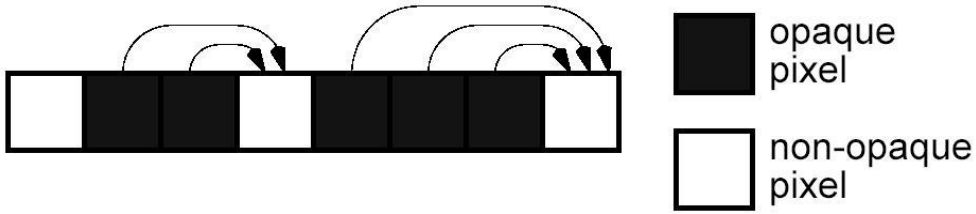
  54 49 23 32 35 17 28
- For very large data-sets that exhibit very high *spatial coherence*, RLE can save a lot of information
- Plus it is a form of *lossless compression* – no information is discarded during the compression process (as opposed to *lossy compression*)

Run-Length Encoding

- In particular, we will take a classified volume, and perform RLE to compute runs of transparent and opaque voxels
- Remembering that we will traverse the volume one scanline at a time, can you think of why it might be useful to run-length encode the voxels in this manner?
- So we can process only “interesting” voxels, those that will affect the final rendered image
- We are likely to have large regions that are opaque or transparent, so we should take advantage of spatial coherence whenever possible
- Aside: spatial coherence vs. temporal coherence – what’s temporal coherence?
- Where might we find temporal coherence in volume visualization?

Offsets

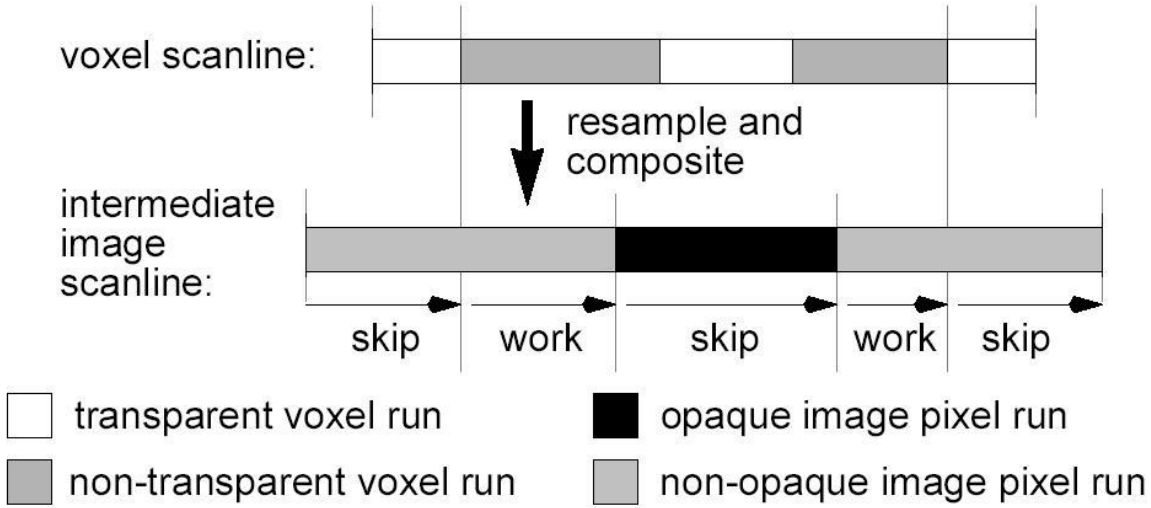
- In addition to performing RLE on the volume, we will also store a series of offsets that link non-transparent voxels together
- We classify each voxel before rendering begins
- Perform RLE
- Then we chain together all the opaque or mostly opaque voxels



- The combination of RLE and offsetting lets us process only the "important" voxels and skip through the scanlines very quickly
- These voxels are resampled and composited into the image scanline very easily then, as we will see now

Scanline Traversal

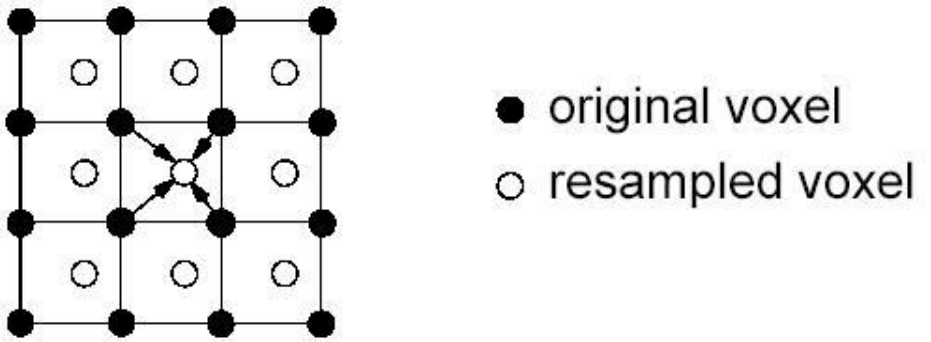
- With these two optimizations at our disposal, we can traverse the voxel and pixel scanlines very efficiently because we can substantially reduce the amount of addressing arithmetic we perform both in the volume and the intermediate image



- We perform our work only on those voxels *and pixels* that will affect the final image
- By skipping opaque pixels, we can perform early ray termination
- Pixel run-length encoding is computed on-the-fly

Parallel Projection with Shear-Warp Algorithm

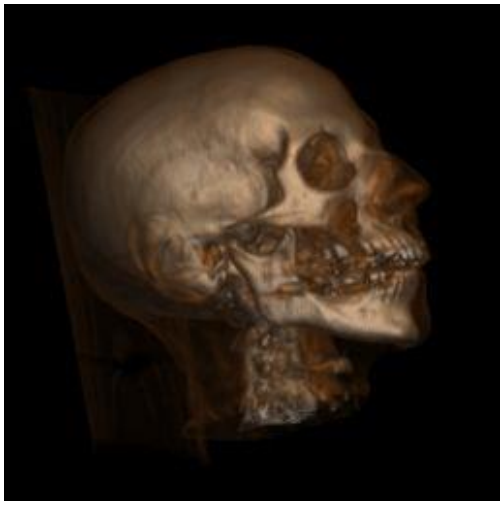
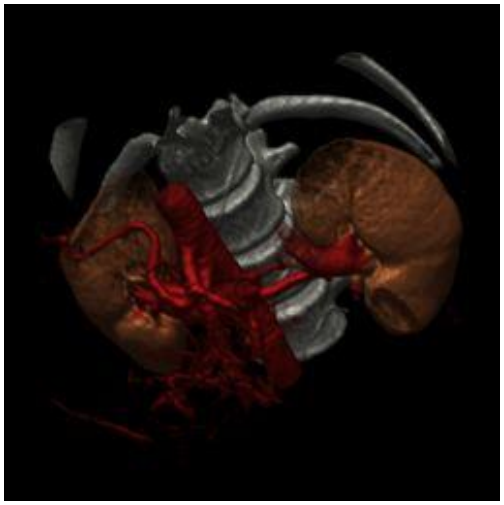
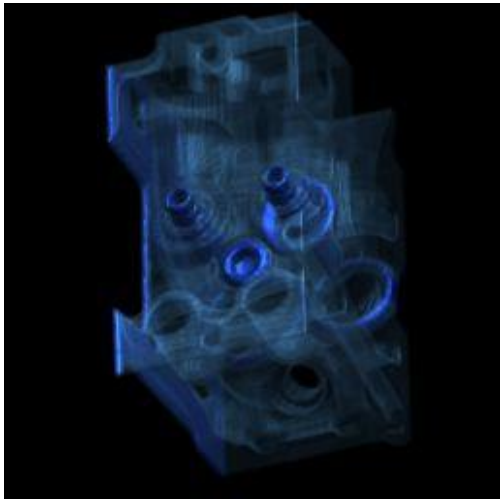
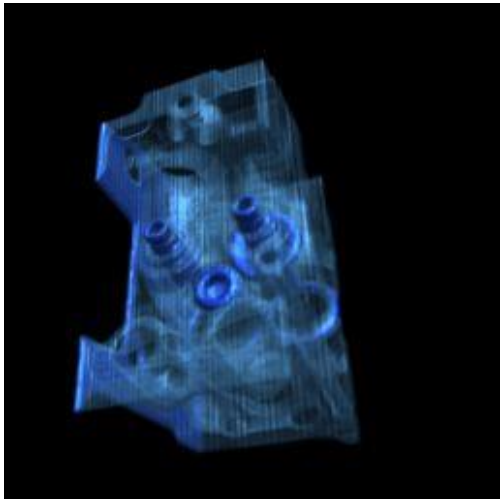
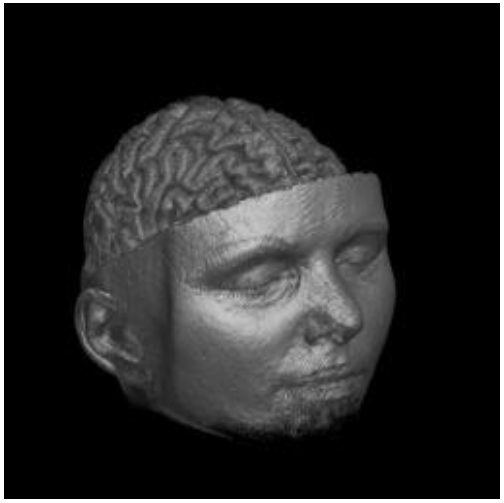
- We will assume that the voxels have been classified and shaded already (pre-shaded pipeline since we interpolate already-shaded samples)
- Since the voxel and pixel scanlines (in the intermediate image) are parallel and have a 1-to-1 correspondence, we will process them in tandem.
- Due to translation, however, the scanlines of the volume and intermediate image may not line up exactly.
- However, since each volume scanline will be displaced with respect to the intermediate image plane by the same amount, we can use the same bilinear interpolation weights for all resampled voxels (i.e., interpolated values that **do** line up exactly with the pixels).



Parallel Projection with Shear-Warp Algorithm

- Given a particular view direction, we pick the face of the volume is most parallel to our image plane (or most perpendicular to the viewing direction)
- Usually it's not exactly parallel, which is why we warp the image
- To make things simple, suppose we implement our algorithm assuming our view direction is in the general direction of *z*
- If, after determining which volume face is most parallel, we find that our view direction is along *x* or *y* (or negative *x* or *y*), we actually need to re-order the voxel's by *transposing* the coordinate axes
- How does this affect the RLE? Our RLE algorithm assumes the scanlines run in a particular direction (along *x* or *y* or *z*)!
- To avoid this problem, we pre-compute *three* RLE volumes, one for each direction

Examples

- Movie time! (13)

Observations about Shear-Warp Rendering

- Advantage: fast (for 1994) when opacity transfer function is kept constant (to preserve the RLE volume)
- Disadvantages:
- Need to encode the volume after each opacity transfer function change
- Not good for interactive iso-surface modification (i.e. when opacity transfer function is varied)
- Need 3 encoded volumes, one for each main viewing axis
- True sampling rate in z varies between [1.0, 1.72] due to shearing
- Thus Nyquist theorem is almost always violated
- Leads to staircasing artifacts for diagonal views (e.g., at 45°)
- Blurring for magnified views, since volume is pre-shaded

Fast Classification

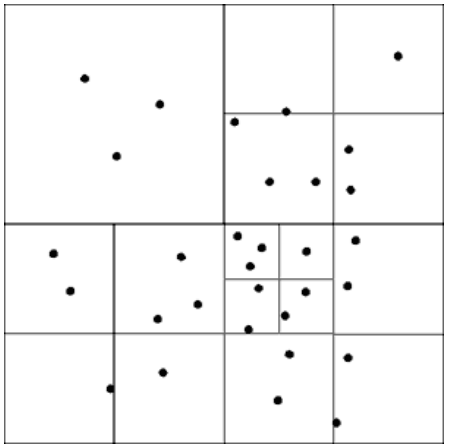
- The inability to modify the opacity transfer function is a serious limitation of the usefulness and applicability of the algorithm
- But there is hope!
- We will have to use something other than our RLE volume data structure and traverse the voxel scanlines in a different manner
- Let us assume we have some opacity transfer function $\alpha = f(p, q, \ldots)$ on a multi-dimensional scalar domain, such as $\alpha = f(d, |\nabla d|)$
- Using this function we want to figure out which portions of a scanline are opaque and which portions we can just ignore it
- This discernment will be made using a recursive algorithm

Fast Classification Algorithm

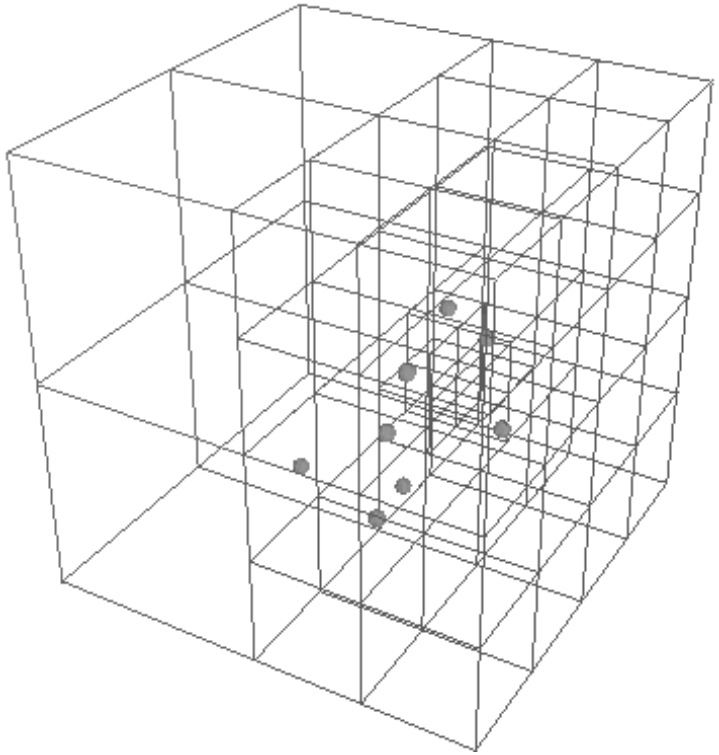
1. For some block of the volume that contains the current scanline, find the extrema of the parameters of the opacity transfer function ($d_{\min}$, $d_{\max}$, $|\nabla d|_{\min}$,$|\nabla d|_{\max}$). These extrema bound a rectangular region of the feature space.
2. Determine if the region is transparent, i.e., *f* evaluated for all parameter points in the region yields only transparent opacities. If so, then discard the scanline since it must be transparent.
3. Subdivide the scanline and repeat this algorithm recursively. If the size of the current scanline portion is below a threshold then render it instead of subdividing.

- In a nutshell, we replace pre-processing with a recursive process used during the rendering algorithm

Octree



- We partition the volume into a recursive data structure known as an *octree*
- Binary tree: each child has 2 nodes
- Quadtree: each child has 4 nodes
- Octree: each child has 8 nodes
- The octree is probably the single most important non-trivial data structure in all of volume visualization because…
- It can reduce certain algorithms from $O(n^3)$ to $O(\log_2 n^3)=O(3 \log_2 n) = O(\log_2 n)$!
- Volume/volume intersection, ray/volume intersection, etc.



Min-Max Octree

- To accommodate step 1, we will construct a *min-max octree*, an octree that contains in their nodes the extrema of the parameter values (opacities). Note that these values are independent of the view, or *view-independent*
- To accommodate step 2, we *integrate* the function *f* of opacities over each region in a recursive fashion. Note, if a large area is determined to have zero opacity, there is no need to check its 8 children (three cheers for recursion)
- During scanline traversal, we employ the octree and corresponding integrals to recursively determine which portions of a scanline we should process and which parts we can skip
- This whole algorithm is more complicated than using RLE, but is (believe it or not) substantially faster

Enhancements to Shear-Warp Algorithm

- Convert from pre-shaded to post-shaded pipeline
- Standard way in shear-warp is to classify and shade before rendering starts, which makes it a classic pre-shaded algorithm
- At run time, when we resample and composite voxel scanlines into the pixel scanlines, we perform bilinear interpolation on the colors and opacities
- Instead, we can interpolate the densities inside each voxel scanline during resampling, perform the classification and shading, and then composite the results into the intermediate image
- The same amount of work is done in terms of interpolations performed and does not degrade performance
- In retrospect it seems like an obvious solution, but no one seriously compared pre- and post-shading until a few years later

Perspective Projection

- Relatively easy to implement perspective projection using a shear-warp approach
- Voxels must be scaled as well as translated (as usual)
- This complicates the resampling process, which is no longer uniform across the volume since two or more voxel scanlines may project onto an image scanline
- However, it's still much faster than ray-casting since we can still march through the volume and image scanlines in a fairly rigid order that doesn't change much from one viewpoint to the next