Chapter 7

Fast algorithms for the volumetric method

The creation of detailed, complex models requires a large amount of input data to be merged into high resolution voxel grids. The examples in the next chapter include models generated from as many as 70 scans containing up to 12 million input vertices with volumetric grids ranging in size up to 160 million voxels. Clearly, time and space optimizations are critical for merging this data and managing these grids. In this chapter, we begin by describing a run-length encoding scheme for efficient storage of the volumetric data (Section 7.1). Next, we develop a number of optimizations designed to improve execution time when merging range data into the volume (Section 7.2). In Section 7.3, we summarize an efficient isosurface extraction method. Finally, we show that the storage and execution optimizations lead to significant improvements in asymptotic complexity (Section 7.4).

7.1 Run-length encoding

The core data structure is a run-length encoded (RLE) volume with three run types: empty, unseen, and varying. The varying fields are stored as an explicit stream data, rather than runs of constant value. Typical memory savings vary from 10:1 to 20:1. In fact, the space required to represent one of these voxel grids is usually *less* than the memory required to represent the final mesh as a list of vertices and triangle indices.

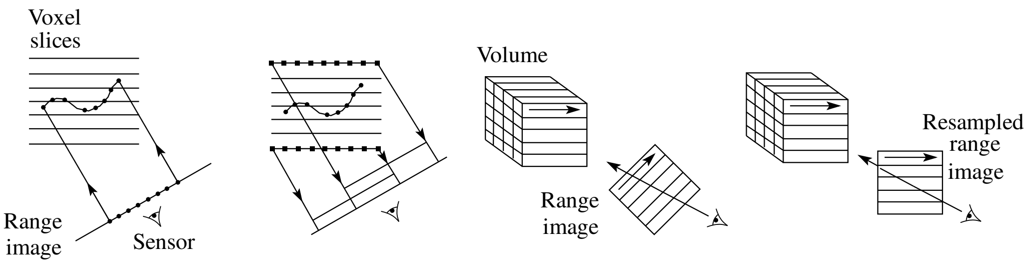

Figure 7.1: Overview of range image resampling and scanline order voxel updates. (a) Casting rays from the pixels on the range image means cutting across scanlines of the voxel grid, resulting in poor memory performance. (b) Instead, we run along scanlines of voxels, mapping them to the correct positions on the resampled range image. (c) Range image scanlines are not in general oriented to allow for coherently streaming through voxel and range scanlines. (d) By resampling the range image, we can obtain the desired range scanline orientation.

7.2 Fast volume updating

Updating the volume from a range image may be likened to inverse volume rendering: instead of reading from a volume and writing to an image, we read from a range image and write to a volume. Research in volume rendering has shown that the manner of traversing the volume and image is critical to the efficiency of the algorithm [Lacroute & Levoy 1994]. In particular, streaming through the image and following each line of sight through the volume leads to poor performance, because it requires random access to voxels that are not adjacent in memory, as shown in Figure 7.1a. We can instead step through the volume in scanline order, but this requires random access of the *image* pixels, leading to a similar memory access inefficiency, as shown in Figure 7.1c. The solution is to pre-warp the image in a manner that forces voxel and image scanlines to map onto each other.

In this section, we develop the geometric intuition for how the range image plane must be oriented to ensure ideal scanline alignment for both the orthographic and the perspective case. Next we describe methods of performing the resampling and subsequent updating of the volume. Obeying the more restrictive image plane requirement imposed by the perspective case, we show how we can develop a shear-warp method for resampling the range image and updating the volume. This shear-warp method leads to a fast algorithm using binary depth trees to update only relevant portions of the volume. Finally, we describe a fast

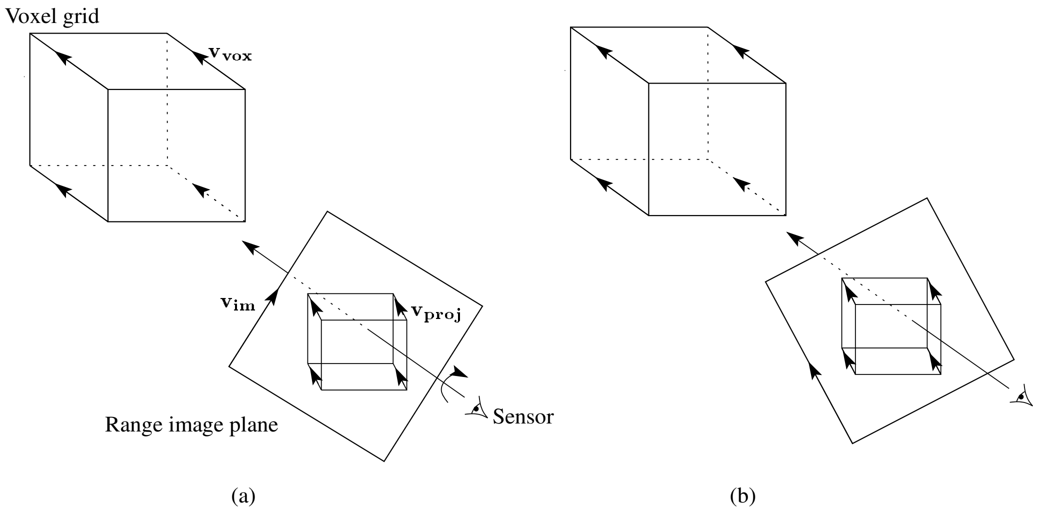

Figure 7.2: Orthographic range image resampling. As shown in (a), the voxel scanlines run in the direction $\mathbf{v_{vox}}$. The projections of these scanlines onto the range image plane run along the direction $\mathbf{v_{proj}}$, while the range image scanlines run in the direction $\mathbf{v_{im}}$. By rotating the range image plane as shown in (b), we can align the directions of the projected voxel scanlines and the image scanlines. This affords coherent memory access when streaming through the voxel grid and the range image.

transpose method for ensuring that voxel scanlines run parallel to image scanlines.

7.2.1 Scanline alignment

The type of viewing frustum for a rangefinder imposes constraints on how to choose the range image resampling that guarantees volume and image scanline alignment. For an orthographic frustum, the desired alignment can be attained by simply rotating the image plane about the axis defined by the viewing direction. The amount of rotation can be determined by projecting the direction of the voxel scanlines onto the image plane, followed by rotating the image plane until the image scanlines align with the projected viewing directions (see Figure 7.2). Because the orthographic projection of the voxel scanlines is a set of parallel lines, it is always possible to rotate the range image about the view direction to assure mutual alignment of image and voxel scanlines.

Perspective viewing frustums impose additional constraints. The first requires that the new image plane be parallel to the voxel scanlines. Figure 7.3 shows how this constraint

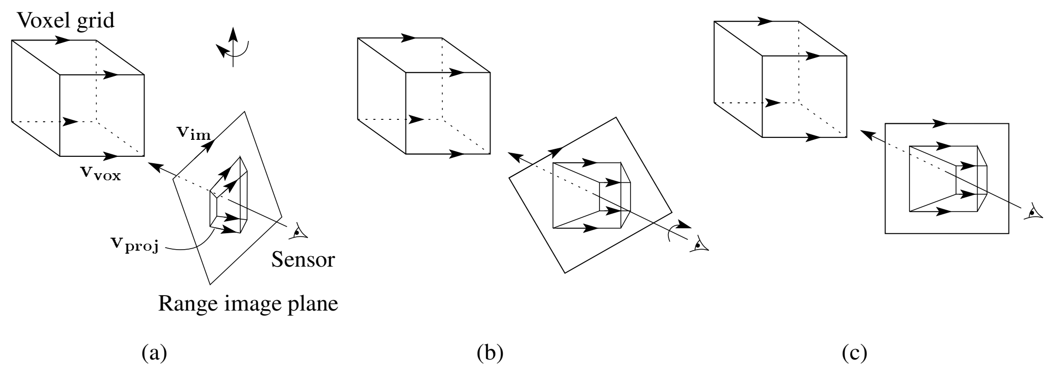

Figure 7.3: Perspective range image resampling. The notation follows that of Figure 7.2. In (a), the range image plane is at an angle to the front face of the voxel grid. As a result, the projected voxel scanlines are not parallel. By rotating the image plane so that the image plane is parallel to the voxel scanlines (b), we force the projected voxel scanlines to be parallel. As in the orthographic case, we must additionally rotate the image plane about the viewing axis to complete the mutual alignment.

applies. When coupled with rotating the image about the viewing axis, the resulting image scanlines, viewed as lines segments in three dimensions, now run exactly parallel to the voxel scanlines. An additional restriction applies to the direction of the *voxel* scanlines. Consider the case shown in Figure 7.4. As we follow the voxel scanlines away from the viewpoint, the projections of these scanlines progress toward the center of the image. Thus, no rotation of the image plane is suitable to align voxel and image scanlines. On the other hand, by reorganizing the voxel scanlines so that they run parallel to the image plane (i.e., by transposing the data structure), we obtain the desired alignment.

More complex viewing frustums can make it impossible to assure precise alignment of voxel and image scanlines, as is the case with the line perspective of the Cyberware scanner for traditional triangulation. In this case, we can choose an image orientation that is aligned with scanlines for one voxel slice, but the scanlines at different slices will project to curves that cross image scanlines. Nevertheless, if the projection is nearly orthographic, i.e., the rays are not severely divergent, then an image orientation can be chosen to minimize the amount that projected scanlines cross range image scanlines. If rays are highly divergent, then other methods, such as partitioning the range image resamplings into manageable sections may be applicable.

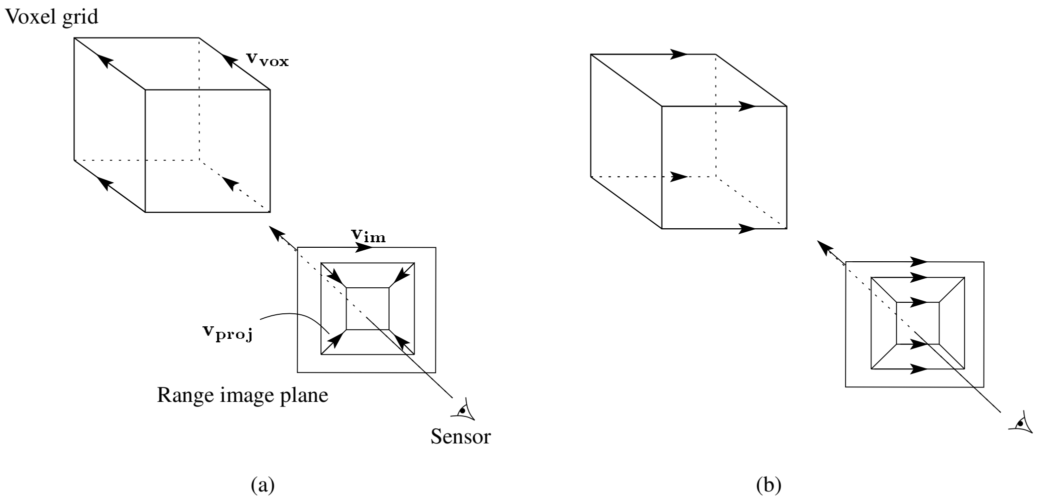

Figure 7.4: Transposing the volume for scanline alignment. The notation follows that of Figure 7.2. The projected voxel scanlines in (a) converge to the perspective vanishing point on the image plane. In this case, it is not possible to rotate the image plane to align with projected voxel scanlines. By transposing the volume (b), we see that the projected scanlines are now parallel.

7.2.2 Resampling the range image

To resample the range image using the newly oriented image plane, we first reconstruct the range surface. Next, we can cast a set of rays from the sensor through the image pixels and intersect them with the range surface. The distances along these rays are the range values to be stored in the new range image. By interpolating the weights at the vertices on the range surface, we can associate a new weight with each sample.

Alternatively, we can resample the range image using an incremental scan-conversion algorithm, a method know to be very efficient. In this case, we first transform the vertices of the range surface using the viewing transformation specific to the lines of sight of the range sensor and the image plane. The weights are then assigned as vertex "colors" to be linearly interpolated during the rendering step, an approach equivalent to Gouraud shading of triangle colors [Foley et al. 1992]. This resampling can be performed with a traditional z-buffered rendering algorithm, though no depth comparisons are necessary when entering values into the z-buffer (i.e., there must be exactly one depth per line of sight if the viewing rays are properly calibrated).

7.2.3 Updating the volume

To merge the range data into the voxel grid, we stream through the voxel scanlines in order while stepping through the corresponding scanlines in the resampled range image. We map each voxel scanline to the correct portion of the range scanline as depicted in Figure 7.1b, and we resample the range data to yield a distance from the range surface. Using the combination rules given by Equations 6.3 and 6.4, we update the run-length encoded structure. To preserve the linear memory structure of the RLE volume (and thus avoid using linked lists of runs scattered through the memory space), we read the voxel scanlines from the current volume and write the updated scanlines to a second RLE volume; i.e., we double-buffer the voxel grid.

7.2.4 A shear-warp factorization

In Section 7.2.1, we showed that forcing the range image scanlines to run parallel to the voxel scanlines is necessary for achieving good memory performance for perspective projections. If we require that this restriction holds in all cases (including orthography), then we can develop a shear-warp factorization of the viewing transformation, similar to the one described for volume rendering in [Lacroute & Levoy 1994]. The resulting algorithm utilizes a z-buffered scan-conversion algorithm and allows for computation of signed distances by comparing a scanline's constant z value against the resampled z values from the range image. This method of establishing signed distance will be exploited for more efficient voxel traversal using the binary depth trees described in the next section.

Figure 7.5 illustrates the shear warp procedure for an orthographic projection in two dimensions. We first transpose the voxel data structure so that the scanlines run as close to perpendicular to the viewing direction as possible. Then we shear both the range surface and voxel grid so that the viewing rays run perpendicular to the voxel slices. Note that shearing the voxel grid simply amounts to changing the origins of the voxel slices; no resampling of the voxels is ever performed. Next, we resample the range surface with respect to an image plane parallel to the voxel slices. As before, we stream through voxel scanlines and resample depths stored in the resampled range image. The difference between the z-depth of each voxel (constant within a scanline) and the resampled range depth corresponds to the signed

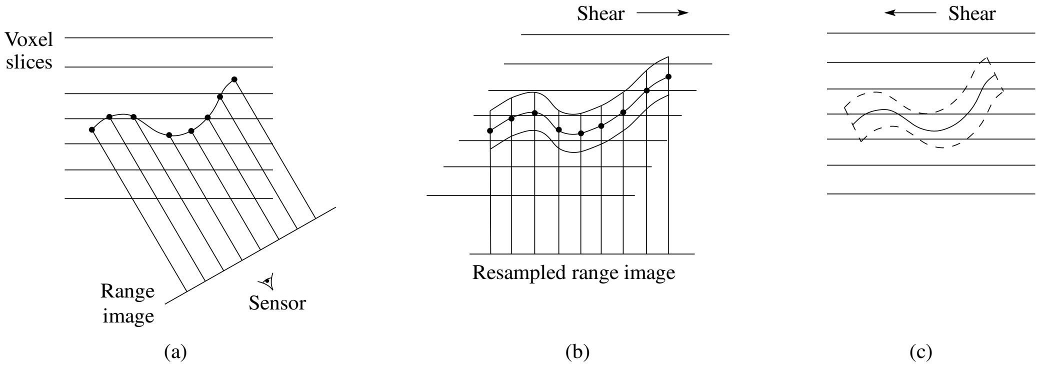

Figure 7.5: A shear warp method for parallel projection. (a) The range image and sensor lines of sight are not initially aligned with the voxel grid. (b) By shearing the grid and range image, we "straighten" the sensor rays with respect to the voxel grid. This shear performs the same function as the rotation in Figure 7.3. By resampling the range image, we align its pixels with voxels on the sheared grid. We then lay down the signed distance and weight functions. (c) After updating the grid, we shear it back to the original space. Note that shearing the voxel grid is simply a matter of offsetting the origin of each slice; *no voxel resampling* is performed.

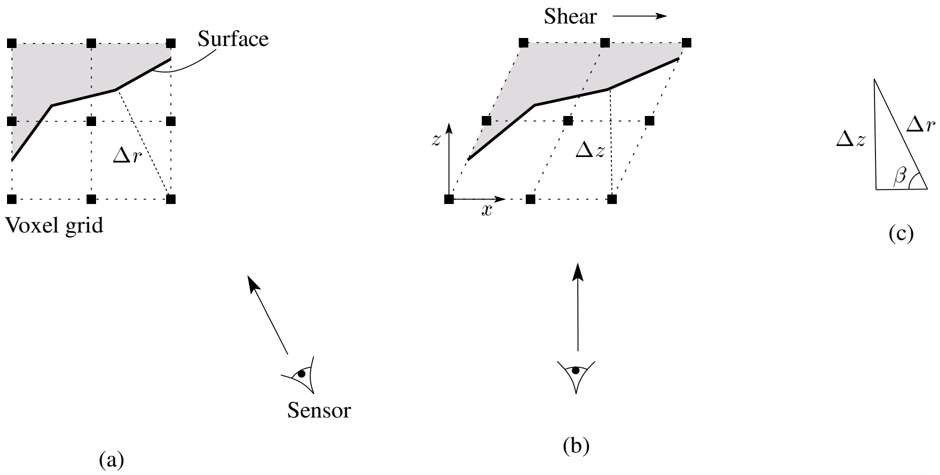

Figure 7.6: Correction factor for sheared distances. The signed distance ($d = \Delta r$) to a voxel should be taken along the line of sight from the sensor as shown in (a). After shearing the voxel grid and range image, the signed distance corresponds to the differences in z-depth for the voxel and range sample as shown in (b). The error may be corrected by a constant factor. In two dimensions, (c) indicates the correction factor would be $\Delta r = \Delta z / \sin \beta$, where β is the angle between the original view direction and the sheared view direction.

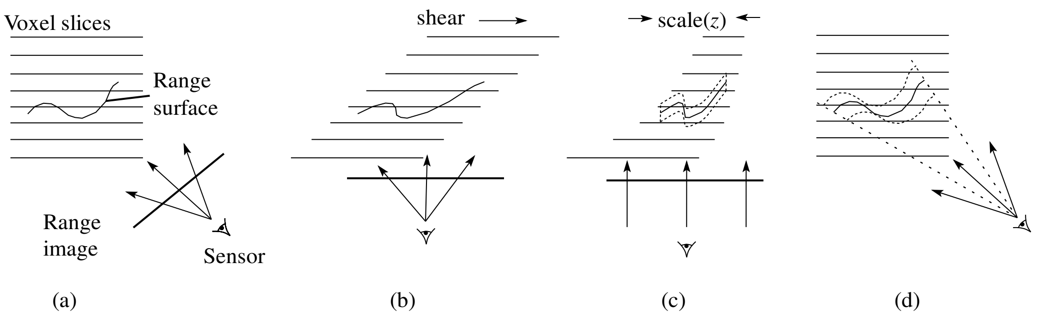

Figure 7.7: Shear-warp for perspective projection. (a) The range surface and sensor lines of sight are not initially aligned with the voxel grid. By shearing the voxel grid and range surface (b), followed by a depth dependent scale (c), we “straighten” the sensor rays with respect to the voxel grid. (d) After updating the grid, we “unscale” and “unshear” it back to the original space.

distance between the voxel and the range surface. Due to the shear, this distance must be corrected by a multiplicative factor as depicted in Figure 7.6.

Shear warp factorizations may be adapted to other range imaging frustums as well. Figure 7.7 shows how a perspective transformation can be decomposed into a shear and a scale. Again, the voxel lattice is never resampled; the shear and scale only serves to indicate how voxels map onto the resampled range image plane. The differences between voxel depths and resampled range depths again correspond to signed distances between the voxel and the range surface. As with the orthographic case, a correction factor is necessary due to the differences between the projection direction and viewing rays. This correction factor is not constant over all viewing rays, but it is constant along each viewing ray. Thus, the correction factor may be stored at each resampled range image pixel.

More complex imaging frustums such as the line perspective of Figure 5.2f will require more complex transformations, but the principle of doing modified z comparisons is still valid and leads to an efficient algorithm with the help of the binary depth tree described in the next section.

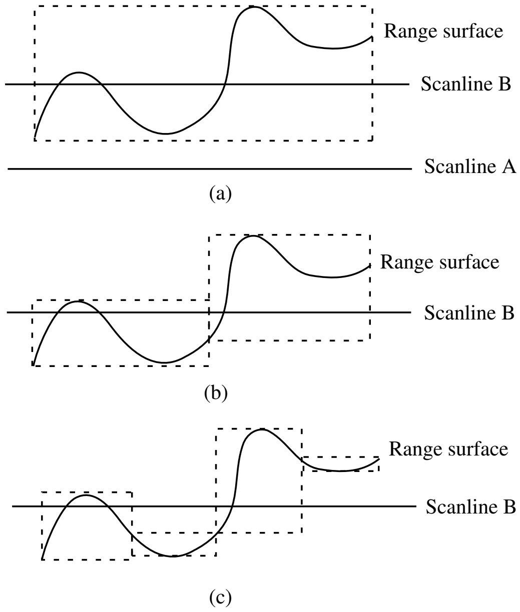

Figure 7.8: Binary depth tree. Each pair of resampled range image scanlines is entered into a binary depth tree. (a) Level 0: a bound is set on the whole range surface. Scanline A is well in front of the bounds and is not processed. Scanline B intersects and forces further traversal of the tree. (b) Level 1: the surface is split in two and yields tighter bounds. Still, both sides must be traversed further. (c) Level 2: the surface is quartered and scanline B is found to intersect the first and third quarters.

7.2.5 Binary depth trees

We can take advantage of the fact that z values are constant at each voxel scanline in order to process the voxels efficiently. In the case of merging range data only in the vicinity of the surface, we try to avoid processing voxels distant from the surface. To that end, we construct a binary tree of minimum and maximum depths for every adjacent pair of resampled range image scanlines. The trees must span pairs of range scanlines, because voxel scanlines may fall between range scanlines, and linear interpolation requires simultaneously processing two neighboring scanlines. Figure 7.8 illustrates the binary tree for three levels in two dimensions. Before processing each voxel scanline, we query the binary tree to decide which voxels, if any, are near the range surface. In figure 7.8a, scanline A lies completely outside of the range tree and thus requires no processing. Scanline B on the other hand does intersect the tree, so we proceed down the levels of the tree until we establish which portions of the scanline need to be updated. In a similar fashion, the depth tree can be traversed in a manner that rapidly indicates what sections of a scanline are likely to be empty. The resulting speed-ups from the binary tree are typically a factor of 15 without carving, and a factor of 5 with carving.

When using the more complex viewing frustums such as line perspective, the voxel scanlines may map onto more than two range image scanlines. In this case, the domain of the binary tree is widened to span as many scanlines as necessary.

7.2.6 Efficient RLE transposes

As described in Figure 7.4, if the scanlines of the volume are badly aligned with the range image scanlines, it may be necessary to transpose the direction of the scanlines. Without optimizations, this transpose would require $O(n^3)$ operations for a volume n voxels on a side. To avoid this $O(n^3)$ cost, the solution is to work primarily on the varying voxels and to create the transposed volume directly in RLE form without ever expanding it in an intermediate step. This objective is very similar to the one for rotation of the RLE spacetime images described in Chapter 4. Indeed, the transpose is very much like a rotation by 90 degrees. The key differences are: (1) no reconstruction and resampling is required, and (2) there are two values for constant runs (unseen and empty) instead of one (zeroes). The first difference simplifies the task, while the second complicates it. In the RLE rotation algorithm, when a run of varying type began where there had been none before on a target scanline, then the prior run on the target scanline was deduced to be a constant run of zeroes, and could be updated accordingly. As a result, we could stream through only the varying runs in the source image, and ignore the constant runs. However, when more than one constant run type is possible, then the constant runs require further attention.

Fortunately, by tracking two source scanlines simultaneously, we can overcome this difficulty. As before, we construct the target scanlines as we stream through the source scanlines. This time, however, each target scanline must keep track of values (empty or unseen) when building constant runs. As long as the runs in the source scanline are the same as the current runs in the target scanlines, we would like to skip to the interesting voxels. When the type of run in the source scanline does differ from the type in the target scanline, then the target needs to be updated and a new run begun. By simultaneously streaming through

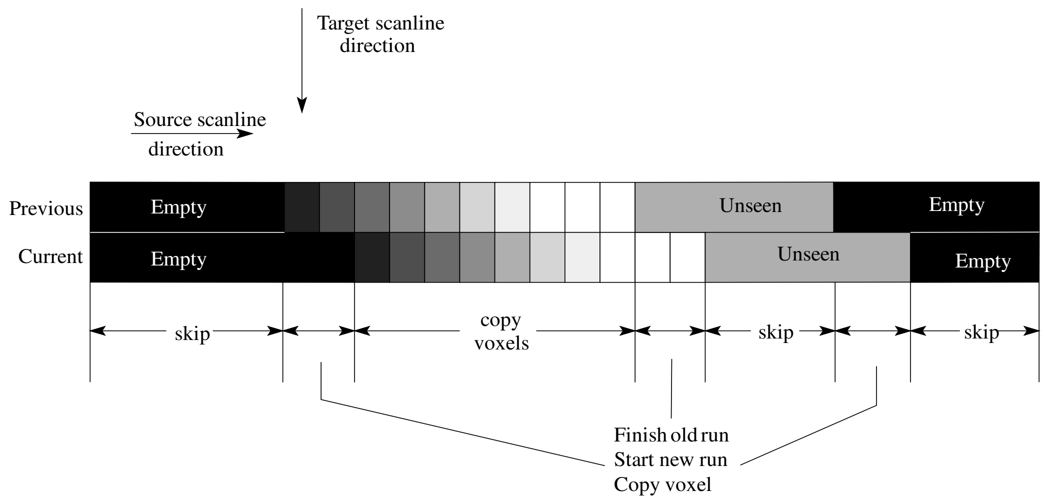

Figure 7.9: Fast transpose of a multi-valued RLE image. The source scanline direction runs horizontally and the target direction runs vertically. We stream through the current and previous source scanlines simultaneously. We can skip over intervals where scanlines have constant runs of the same value, as indicated with the first "empty" run. When the runs differ, then the target scanline must be finished with its run, so we compute the length of the concluded run, and begin a new one. When the runs are both of varying type, then the varying voxels are simply copied into the target runs.

the current and the *previous* source scanline, we can deduce when the source is different from the target, requiring some work to be done. Figure 7.9 illustrates the fast transpose algorithm.

7.3 Fast surface extraction

To generate our final surfaces, we employ a Marching Cubes algorithm [Lorensen & Cline 1987] with a lookup table that resolves ambiguous cases [Montani et al. 1994]. To reduce computational costs, we only process voxels that have varying data or are at the boundary between empty and unseen. A practical consequence of using the hole filling approach is the generation of spurious components where the lines of sight carved around unknown regions that are not connected to the observed surface. It is a straightforward matter to remove these components. In practice, we typically work with models that consist of a single, large connected component. In this case, it suffices to extract the largest connected component.

m	Number of samples on a side for a range image (m^2 samples)
p	Number of range images
n	Number of voxels on a side for a cube shaped volume (n^3 voxels)
Δ	Voxel resolution in meters
b	Width of the signed distance ramp in meters
$\hat{b}$	Width of the signed distance ramp in voxels (b/Δ voxels)
A	Surface area of the visible portions of the object in square meters

Table 7.1: Symbol definitions for complexity analysis.

7.4 Asymptotic Complexity

The optimizations described in this chapter serve the purpose of improving memory performance, but they also significantly reduce the asymptotic complexity of both storage and computation. Before studying the algorithm's complexity, we need to define a few symbols as shown in Table 7.1. In the remainder of this section, we assume that the voxel resolution, Δ, and ramp widths, b and $\hat{b}$, are fixed, and that n grows only in proportion to the size of the object being reconstructed.

For comparison purposes, we first consider a non-incremental algorithm without optimizations. The non-incremental aspect means that all of the range images need to be processed at the same time. Since each range image contains m^2 samples and yields no more than $2m^2$ triangles after tessellation to create the range surface, the overall storage is $O(pm^2)$ for the range data. Without compression, another $O(n^3)$ storage is required to represent the volume, giving an overall storage cost of $O(pm^2 + n^3)$. As for computational complexity, a brute force algorithm without any spatial data structure for visiting relevant voxels would execute in time proportional to the number of voxels for each range image plus the number of samples in each range image, i.e., $O(m^2 + n^3)$, leading to a total time complexity of $O(pm^2 + pn^3)$.

In the remainder of this section, we show that the fast algorithm does significantly better than the brute force approach. We begin with the storage complexity of the algorithm followed by an analysis of the computational complexity.

7.4.1 Storage

To analyze storage complexity, we first consider the costs of storing the original range images as well the resampled range images and their depth tree data structures. Next we examine the storage requirements for the RLE volume with and without space carving. Finally, we compute the storage costs of the reconstructed surface and compare it to the volumetric storage costs.

As described above, each range image contains m^2 samples resulting in no more than $2m^2$ triangles after tessellation. Because the volumetric algorithm is incremental, we need not hold all range images in memory at once, so the storage cost for manipulating the range images is $O(m^2)$. Before merging the range surface into the volume, it is resampled at voxel resolution. This resampled range image requires $O(n^2)$ storage. In addition, a binary tree is associated with each scanline of the resampled range image. After summing over the nodes of one tree, we find that it holds no more than $2n$ nodes, leading to a total of $2n^2$ nodes over the whole resampled range image. Thus, the storage cost for the resampled range image and its binary trees is $O(n^2)$.

When reconstructing the observed portions of the surface without space carving, the surface is effectively "blurred" into the volume by the signed distance ramps. This blurred surface occupies a volume of roughly $V = bA$, which translates to bA/Δ^3 or $\hat{b}A/\Delta^2$ voxels that must be stored. The RLE representation requires additional storage for the run lengths. If the average number of intersections of the surface with each voxel scanline is $c_{\cap}^{surf}$, then $(1 + 2c_{\cap}^{surf})n^2$ run lengths are necessary. The overall storage cost for the volumetric data structure is then $O(\hat{b}A/\Delta^2 + (1 + 2c_{\cap}^{surf})n^2)$. For objects that do not have an unusually high surface area to volume ratio (i.e., objects without many folds or spiny projections), then $A/\Delta^2 \sim n^2$ and the number of voxels required is proportional to $\hat{b}n^2$. Thus, for non-pathological surfaces, the storage cost for regions near the observed surface is $O(n^2)$.

The addition of space carving to the algorithm has the potential of greatly increased storage costs. When a sequence of voxels are all labeled empty, they are compactly stored as a single run of empty voxels; however, the examples in Figure 7.10 illustrate some pathological cases that defy efficient storage. The objects in this figure have highly complex geometry (7.10a) or extreme variations in surface reflectance (7.10b) that can lead to the creation of

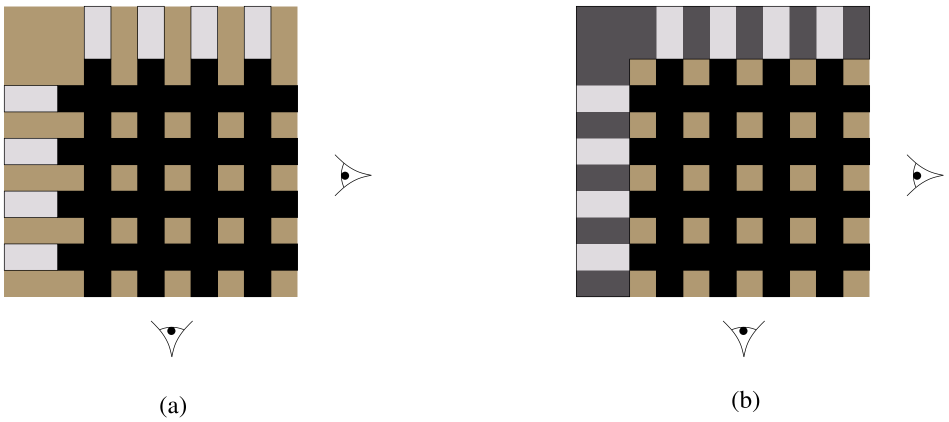

Figure 7.10: Carving from "difficult" surfaces. (a) The scene contains multiple objects shown in gray. Black regions are seen to be empty. After scanning from two viewpoints, many unseen "islands" form. (b) The same as (a), but now there is a single object with regions of negligible reflectance, shown in dark gray. The sensor cannot digitize these dark portions of the surface, and space carving from observed surfaces again leads to a fragmented volume.

many thin shafts of emptiness when carving away from the observed surface. The intersections of these shafts can lead to many separate unknown regions. This high degree of inhomogeneity leads to a worst case storage requirement of $O(n^3)$.

Nevertheless, many objects of interest do not exhibit such complexity in geometry or variations in surface reflectance. More "accessible" objects, i.e., objects having large connected visible portions, lead to a much more efficient representation. In practice, experiments with several objects of moderate complexity show that the storage requirements do not rise appreciably when including the space carving.

Note that the use of a backdrop can have an appreciable effect on the storage complexity. Consider the example in Figure 7.11a. Even though the scene being scanned is fairly simple, the number of unseen "islands" can increase rapidly with the number of scans. However, by performing only two scans with a backdrop, as shown in Figure 7.11b, these islands do not appear.

In the last step of the algorithm, we perform an isosurface extraction using an efficient Marching Cubes algorithm. By processing the run length encoded data structure without expanding it into the full volume, the cost of storing the volume remains the same as above. The resulting isosurface will have roughly A/Δ^2 vertices and $2A/\Delta^2$ faces. In practice,

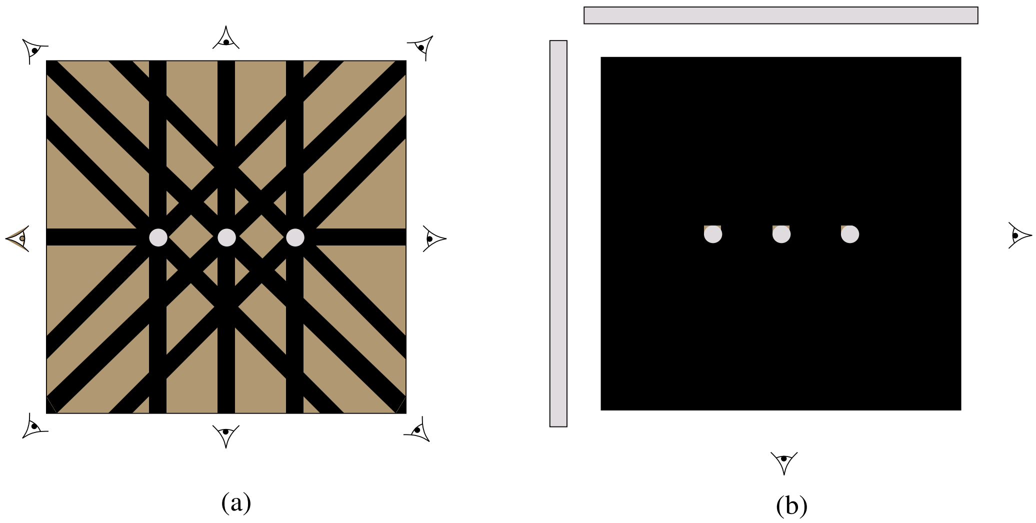

Figure 7.11: Storage complexity with and without backdrops in 2D. Three circles are being scanned indicated as the gray regions. (a) After taking 8 scans without a backdrop, a large number of unseen islands appear. These will result in high storage costs. In fact, more scans can actually increase the fragmentation. (b) Using a backdrop, only two scans are necessary to declare most of the space around the object to be empty.

we have found that the storage required for the RLE volume is comparable to the storage required for the extracted isosurface.

In summary, for non-pathological cases, the storage requirements for the fast volumetric algorithm are $O(m^2 + n^2)$. This complexity is clearly superior to the storage requirement of $O(pm^2 + n^3)$ for a non-incremental, uncompressed algorithm.

7.4.2 Computation

The key operations that influence the computational complexity of the volumetric method are:

- tessellating and resampling the range images
- building the binary depth trees
- performing volume transposes
- finding relevant voxels to update

- merging runs of voxels
- extracting the isosurface.

We address each of these in order.

In converting from a range image to a range surface, each sample is considered for creating triangles with its nearest neighbors. Thus, the tessellation requires time proportional to the number of samples, $O(m^2)$. Each range surface is resampled at the resolution of the voxel grid before merging into the volume. This step requires $O(m^2 + n^2)$ time to prepare the triangles for rasterization and $O(m^2 + n^2)$ to resample and the range image. Overall, the computational complexity is $O(m^2 + n^2)$ to tessellate and resample the range image.

After resampling the range image, we construct binary depth trees at each scanline. Constructing each depth tree requires work proportional to the number of nodes in the tree, i.e., $O(n)$. For n scanlines, the time complexity is $O(n^2)$.

When the voxel scanlines are not oriented in a direction that is desirable for efficient memory traversal, it is necessary to transpose the volume. The fast transpose algorithm performs work in traversing the RLE data structures in both the current and the transposed directions, in addition to the work of copying over voxels. Managing the data structures takes $O(2(1 + c_{\cap}^{surf})n^2)$ time, while copying over the voxels requires $O(\hat{b}A/\Delta^2)$ time. The overall transpose time complexity is then $O(2(1+c_{\cap}^{surf})n^2 + \hat{b}A/\Delta^2)$. For non-pathological cases, the complexity of the transpose operation is $O(n^2)$.

Once the volume is transposed for optimal scanline traversal, we query the binary depth trees to decide which voxels require updating. In the case of reconstructing the observed portions of the surface, this amounts to deciding which voxels are near the range surface being merged. We can think of the surface as being sampled at voxel resolution and “blurred” along the sensor lines of sight by the signed distance ramps. Thus, as many as $\hat{b}n^2$ voxels may require updating. The cost of finding that a single voxel requires updating is $\log n$, i.e., the cost of traversing the binary tree down to a single leaf node. For a run of voxels spanning an interval of leaf nodes in the tree, the cost of finding the consecutive leaf nodes is amortized over the interval, since the tree is not re-traversed for each new voxel. Still, in the worst case, each voxel near the range surface requires $\log n$ time to be found, yielding an overall complexity of $O(n^2 \log n)$.

When performing space carving, the cost of locating which voxels should be marked as empty is a function of the complexity of the emptiness shafts. We define $c_{\cap}^{shaft}$ to be the average number of shaft intersections per scanline per range image. The cost of finding the interval of emptiness per scanline is then $O(c_{\cap}^{shaft} \log n)$, leading to an overall complexity of $O(c_{\cap}^{shaft} n^2 \log n)$. The constant, $c_{\cap}^{shaft}$, can actually be quite large as shown in Figure 7.10, but as indicated in the previous section, the shafts are usually small in number leading to small values of $c_{\cap}^{shaft}$ for non-pathological objects and an effective complexity of $O(n^2 \log n)$.

Once the relevant voxels have been identified with the binary depth tree, the runs of signed distances and emptiness must be merged with the existing runs in the volume. The work required for this step is proportional to the number of runs and the number of non-constant voxels that are already in the volume plus the number that require updating. If the average number of intersections of a range surface with each scanline is $c_{\cap}^{range}$, then the amount of work required to merge runs near the observed surface is $O(\hat{b}A/\Delta^2 + (2 + 2c_{\cap}^{shaft} + 2c_{\cap}^{range})n^2 + \hat{b}n^2)$. Note that because we are using a double-buffered volume, i.e., we copy over all untouched voxels to maintain memory locality, the merge step always requires at least $O(\hat{b}A/\Delta^2)$ time to execute. For a non-pathological surface, the overall time complexity for merging runs near the observed surface is $O(n^2)$.

When merging the runs of emptiness into the volume, the argument still holds that the work required is proportional to the number of runs and the number of non-constant voxels already in the volume and about to be merged. We define $c_{\cap}^{unseen}(i)$ to be the average number of intersections of unseen regions with each scanline after merging i scans. Merging the runs of emptiness derived from the ith range scan will then require $O((2c_{\cap}^{unseen}(i) + 2c_{\cap}^{shaft}(i) + 1)n^2 + \hat{b}A/\Delta^2)$ for each range image. If we let $c_{carve} = \mathrm{MAX}(2c_{\cap}^{unseen}(i) + 2c_{\cap}^{shaft}(i) + 1)$, then the total complexity of merging the empty regions is no worse than $O(c_{carve} n^2 + \hat{b}A/\Delta^2)$. As indicated in the previous section, c_{carve} can be very large, but for non-pathological objects and suitable use of backdrops, c_{carve} is typically small, leading to an overall time complexity of $O(n^2)$ for non-pathological objects when merging runs during the carving step.

All of the discussion of time complexity up to this point has been based on merging a single range image into the volume. For p range images, each of the complexities should be multiplied by p, except for the number of transposes which depends on the order of merging scans. If the object is scanned from all sides, then by selecting the order of merging the scans, the number of transposes need not exceed two: one transpose from the first scanline direction to the second plus one transpose from the second direction to the third. In the worst case, a transpose is required for every range image, which is still no more than p. Thus, the overall complexity for merging range images into the volume is $O(pm^2 + pn^2 \log n + pA/\Delta^2)$. For non-pathological objects, this becomes $O(pm^2 + pn^2 \log n)$.

The final step of the surface reconstruction algorithm is to extract an isosurface from the volume. By visiting only the voxels that are near the surface, this algorithm operates in $O(\hat{b}A/\Delta^2)$ time or $O(n^2)$ for the non-pathological case.

In summary, for non-pathological objects the overall time complexity for merging the observed portions of the surface is $O(pm^2 + pn^2 \log n)$. In practice, we have determined that the logarithmic term is typically overwhelmed by other factors, leading to an observed complexity of $O(pm^2 + pn^2)$. This complexity is a significant improvement over the worst case complexity of $O(pm^2 + pn^3)$ for the brute force algorithm. At the time of publication of this thesis, the computational optimizations for the space carving algorithm have not yet been implemented. However, as long as the volume does not become heavily fragmented, the optimized approach is expected to behave asymptotically as well as the algorithm that does not employ space carving.