Towards Real-Time Voxel Coloring

Andrew C. Prock **Charles R. Dyer***
Department of Computer Science
University of Wisconsin
Madison, WI 53706
{prock|dyer}@cs.wisc.edu
http://www.cs.wisc.edu/~{prock|dyer}

Abstract

Techniques for constructing three-dimensional scene models from two-dimensional images are often slow and unsuitable for interactive, real-time applications. In this paper we explore three methods of enhancing the performance of the voxel coloring reconstruction method. The first approach uses texture mapping to leverage hardware acceleration. The second approach uses spatial coherence and a coarse-to-fine strategy to focus computation on the filled parts of scene space. Finally, the multi-resolution method is extended over time to enhance performance for dynamic scenes.

1 Introduction

Construction of 3D scene information from sets of images has long been an active research topic in computer vision. In recent years several systems have evolved that allow 3D scene construction from video [Kanade *et al.*, 1997, Moezzi *et al.*, 1997]. Reconstruction from video allows realistic 3D movies to be recorded and played back from arbitrary vantage points.

For many applications scene reconstruction must be done in real time, delivering to the user a steerable gaze that is updated dynamically as the scene changes. Producing reconstructions in real time would allow interactive navigation through real scenes as events transpired. This would be ideal for immersing a user into environments that are not easily accessible. Examples of such environments include battlefields, contaminated environments, and restricted areas such as secured buildings.

Reconstruction from video is a computationally expensive process. The virtualized reality system at CMU uses 51 cameras and a bank of VCRs to record video input. The data is processed off-line using stereo methods for reconstruction [Kanade *et al.*, 1997]. The multiple perspective interactive (MPI) video efforts at UCSD also use prerecorded video as input to their system, however with many fewer cameras [Moezzi *et al.*, 1997].

In this paper we take a structurally simple algorithm, voxel coloring, and recast it's solution as an efficient process for creating virtual scenes from video input streams. The variants run 2 to 40 times faster than the original algorithm allowing scenes to be generated at rates of 1-2 Hz.

1.1 Voxel Coloring

Voxel coloring is a scene-space-based algorithm for creating a model of a scene based on widely distributed, calibrated views [Seitz and Dyer, 1997]. As the scene space is traversed, each voxel of the space is projected into the input images and analyzed to determine whether it represents a surface or non-surface region of space. The traversal is done in a depth order with vox-

*This work is sponsored in part by the Defense Advanced Research Projects Agency (DARPA) and Rome Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-97-1-0138.

els nearer to the cameras colored first. The voxels are segregated into planar slabs called layers, and processed on a layer by layer basis. This allows a map to be maintained to account for occlusions in any particular view. With this occlusion map only surfaces are reconstructed.

Voxel coloring, as described in [Seitz and Dyer, 1997], takes tens of seconds to tens of minutes to reconstruct a scene depending on the spatial resolution being modeled. In this paper we describe three methods for speeding up the voxel coloring process. First, texture mapping is used to project the input images onto the voxel layers so as to use hardware acceleration. Second, a coarse-to-fine approach is used. Since voxel coloring produces a 2D surface approximation with voxels, nearly all the space, in most scenes, is empty in the sense that voxel coloring will not color it. The coarse-to-fine approach allows computation time to be focused on the regions in the scene that represent surfaces, thus reducing the overall computation time dramatically. Third, assuming temporal coherence – that is, the scene at successive points in time is similar – we can use the previous time's coloring as input for the next time's coloring. The remainder of the paper describes these three approaches in more detail.

2 Texture Mapping

The projection of millions of voxels into images is computationally expensive. In particular, for a given layer every voxel must be projected into each image. If we approximate the layer with a plane, this process corresponds to mapping the plane of voxels onto the images, or, inversely, projecting the images onto the plane of voxels. This projection can be implemented on conventional graphics workstations using hardware texture mapping.

For texture mapping to work correctly, the input images must be prewarped so that the transformation from world coordinates to image coordinates can be modeled by a pinhole camera. Also, the overall structure of the algorithm must be modified to change the focus from a per-voxel computation to a per-image computation. Because the images are projected onto the voxel plane, the occlusion information must be stored in the images. This requires updating image pixels on a per-layer basis. The pseudocode below presents the main loop of the algorithm. Experimental results are summarized in Section 4.

```
prewarp all images
foreach voxel layer i do
    foreach image j do
        texture map layer i with image j
        foreach voxel v in layer i do
            record color value of v
        od
    od

    foreach voxel v in layer i do
        if v's colors are correlated
            color voxel v
            update image pixels to
                reflect occlusions
    od
od
```

3 Coarse-to-Fine Coloring

Coarse-to-fine methods allow processing to be focused on important regions by using relatively low resolution information as input when creating higher resolution solutions. The application of coarse-to-fine methods to voxel coloring allows most of the computation to concentrate on locations in the scene that contain colored voxels. A similar octree strategy was applied in [Szeliski, 1993], however that method is not suitable here.

3.1 Naive Approach

The main work in voxel coloring is determining which voxels should be colored. The set of colored voxels represent an approximation of the surfaces in the scene. A voxel that contains a small patch of a surface projects to a superset of the pixels which correspond to the actual patch. When the surface only intersects a small fraction of the voxel that is being colored, most pixels that the voxel projects to will not represent the surface. At coarse resolutions this can cause voxels that contain surface patches to go undetected. Voxels that remain uncolored at lower resolutions, but actually contain small opaque sub-regions, should not be eliminated from consideration at higher resolutions. If these regions are lost at a low resolution pass, the resulting coloring will contain noticeable gaps.

Figure 1 shows an example of this problem. The left coloring, (a), was generated by coloring a scene at low resolution. Then, only the colored voxels were then subdivided and subsequently recolored. This process was repeated until the final resolution was reached. Over 13% of the voxels are missing when compared to the correct coloring in (b). This lower density of voxels shows up as gaps in the voxel surface. The gaps and missing voxels are illustrated in (c).

Figure 1: (a) Naive subdividing of colored voxels. (b) Correct coloring. (c) Projected difference of the two voxel colorings.

3.2 Missing Voxels

As resolution is reduced it becomes more and more likely that voxels with smaller occupied sub-volumes will be missed as false negatives. To understand why, consider a simplified version of voxel coloring.

Here voxels are assumed to be spheres, and the sub-volumes that represent occupied space are also spheres wholly contained in the voxel. Let r be the radius of a particular voxel and let r' be the radius of the sub-volume which represents filled space. Let I represent the set of input images.

To determine whether a voxel should be colored, the pixels to which the voxel projects are analyzed. Let P be the set of pixels in I to which a voxel projects, $c(p)$ be the color of pixel $p \in P$, and $\hat{c}$ be the mean color of the voxel over all pixels in P. We can express the occupation likelihood test, λ, as the average 1-norm from the mean pixel color over P as

$$\lambda = \frac{1}{|P|} \sum_{p \in P} |c(p) - \hat{c}|$$

Now if we denote the set of pixels that correspond to solid space as P', and the set of pixels that correspond to empty space as $\bar{P}$, we can write the summation above as

$$\sum_{p \in P} |c(p) - \hat{c}| = \sum_{p' \in P'} |c(p') - \hat{c}| + \sum_{\bar{p} \in \bar{P}} |c(\bar{p}) - \hat{c}|$$

Thus we can write λ as

$$\begin{aligned} \lambda = \frac{|P'|}{|P|} \left(\frac{1}{|P'|} \sum_{p' \in P'} |c(p') - \hat{c}| \right) + \\ \frac{|\bar{P}|}{|P|} \left(\frac{1}{|\bar{P}|} \sum_{\bar{p} \in \bar{P}} |c(\bar{p}) - \hat{c}| \right) \\ = \frac{|P'|}{|P|} \lambda_{v'} + \frac{|\bar{P}|}{|P|} \lambda_{\bar{v}} \end{aligned}$$

where $\lambda_{v'}$ is the quantity in the first set of parentheses above and $\lambda_{\bar{v}}$ is the second quantity. These can be thought of as the occupation likelihoods for the opaque and non-opaque subregions, respectively. For the continuous case the ratio $\frac{|P'|}{|P|}$ is equal to the ratio of the projected area of the solid sub-volumes with respect to the volume. This quantity is simply $\delta^2 = \frac{\pi r'}{\pi r}$. Thus we can approximate the occupation likelihood for the discrete case as

$$\lambda = \delta^2 \lambda_{v'} + (1 - \delta^2) \lambda_{\bar{v}}$$

The quantities $\lambda, \lambda_{v'}$, and $\lambda_{\bar{v}}$ express the color stability of a set of voxels. The occupation likelihood is simply a convex combination of the stability of the two sub-volumes that correspond to solid and empty space.

Consider a solid volume and a background for which both $\lambda_{v'}$ and $\lambda_{\bar{v}}$ are fixed. Then λ is simply a function of δ^2. If $\delta = 1$, the entire voxel is solid, and we have $\lambda = \lambda_{v'}$, as expected. But if $\delta = \frac{1}{2}$, then $\lambda = \frac{1}{4}\lambda_{v'} + \frac{3}{4}\lambda_{\bar{v}}$. By halving the resolution the occupation likelihood becomes dominated by the background. A voxel that would be detected at a given resolution would most likely go undetected if it was the only solid octant of a super voxel.

As a result of the above argument, the naive multi-resolution approach misses a significant number of voxels. To compensate for these missing voxels, some kind of search strategy must be implemented that finds the missing voxels.

3.3 Searching for False Negatives

Without knowing which of the voxels have been mistakenly left uncolored, the best we can do is to use some kind of heuristic to locate those voxels. We can take advantage of the spatial coherence of the surfaces we are trying to extract by considering only the neighborhood around previously colored voxels.

The search strategy we chose was a nearest neighbor search. All voxels within some 1-norm neighborhood of the original low resolution set were added to the set. These voxels were then subdivided into octants. This new set of voxels was then traversed in the standard layered order and colored according to the original algorithm.

Figure 2 illustrates the process used with a neighborhood size of one in a two-dimensional scene. First, all four nearest neighbors are added at the current resolution. Then these voxels are subdivided to create the next higher resolution set of voxels that are candidates for coloring. In the three-dimensional case the neighborhood size is increased to two.

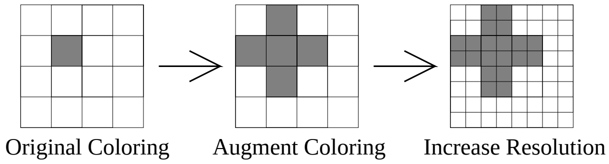

Figure 2: 2D illustration of search strategy.

4 Static Scene Experiments

This section describes experiments evaluating the performance of the two methods described in Sections 2 and 3. The dataset consisted of eight views of a human figure evenly spaced above the scene (see Figure 3). All of the experiments were run on a 200 MHz R5000 SGI O2. Throughout this section the term *scene resolution* refers to the largest of the three dimensions of the voxel scene space being colored.

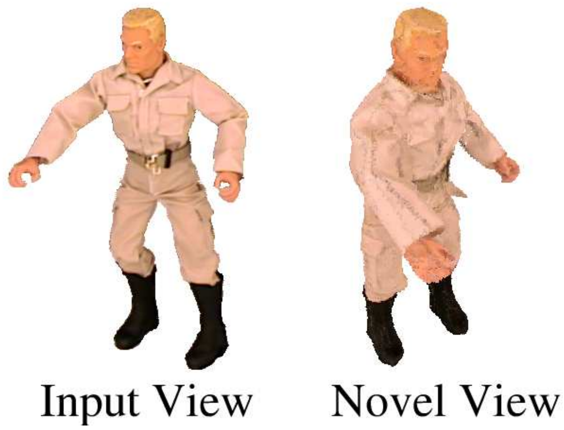

Figure 3: Sample input image, and novel reconstructed view.

4.1 Input Data

Voxel coloring requires widely distributed views of a scene, and corresponding camera calibration information as input. For the experiments in this paper Tsai's method was used to obtain calibration information [Tsai, 1987].

Preprocessing of the input can result in dramatic performance gains. Preprocessing is independent of, and can occur separately from, voxel coloring. Thus it can be implemented in hardware or pipelined with voxel coloring.

Prewarping the input images greatly enhances the performance of the algorithm. Most cameras introduce some amount of radial distortion in images. Tsai's camera calibration method models this radial distortion [Tsai, 1987]. However this method is slow; by prewarping the image we can use a pinhole camera model instead of Tsai's camera model.

Performance of voxel coloring is also enhanced by segmentation of the foreground from the background. This can be done automatically with a staging area using chroma key techniques, or by more elaborate techniques such as motion tracking and snakes. For now we make the assumption that automatic segmentation is available and robust enough for our purposes. For the data presented here the images were segmented manually.

4.2 Texture Mapping Results

In order to perform texture mapping in hardware, the input images were scaled from 640x480 down to 128x128. Because of the reduced resolution, colorings were only performed at scene resolutions up to 256 voxels.

Texture mapping gives modest performance gains for scene resolutions up to 160 voxels. Figure 4 shows a comparison of texture mapping and the original algorithm with prewarping. For scene resolutions of at least 192 voxels, the texture resolution becomes smaller than the resolution of the voxel layer onto which it is projected. Expansion of textures may be performed more efficiently at this point on the SGI O2 architecture and thus there is a significant speed up.

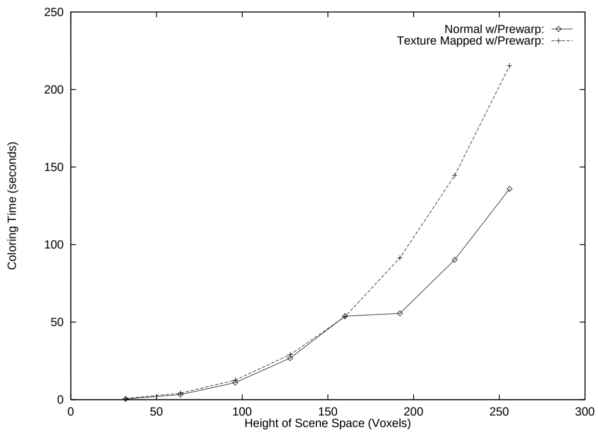

Figure 4: Texture mapping compared to original algorithm with prewarped input.

The colors of voxels produced by texture mapping tend to be mixed with the background color because when the texture is projected the colors are interpolated, causing background pixels to mix with the foreground pixels. This degradation is most noticeable at voxels which correspond to the occluding contours of objects in the input images.

4.3 Coarse-to-Fine Results

Using this strategy the number of voxels traversed is reduced considerably. The total number of voxels traversed was reduced by 80% to 99% depending on the scene resolution. All of the colorings produced by the coarse-to-fine strategy were identical to the colorings produced by the original algorithm.

Table 1 summarizes the execution times of voxel coloring as the scene resolution (in voxels) increases. The original algorithm is compared to the multi-resolution method applied to prewarped input images. For high scene resolutions (512 voxels), the speedup was over forty times. For lower scene resolutions (128 voxels), the speedup was more modest.

Resolution	*Orig. (sec)*	*M/P (sec)*	*Speedup*
32	0.9291	0.7509	1.23
64	5.777	1.512	3.82
128	48.33	4.093	11.8
256	335.8	15.70	21.4
512	2671	64.98	41.1

Table 1: Comparison of original voxel coloring (Orig.) vs. multi-resolution coloring with prewarped input images (M/P).

Figure 5 compares the running time of the original algorithm with the multi-resolution variant as well the effect of prewarping the input images.

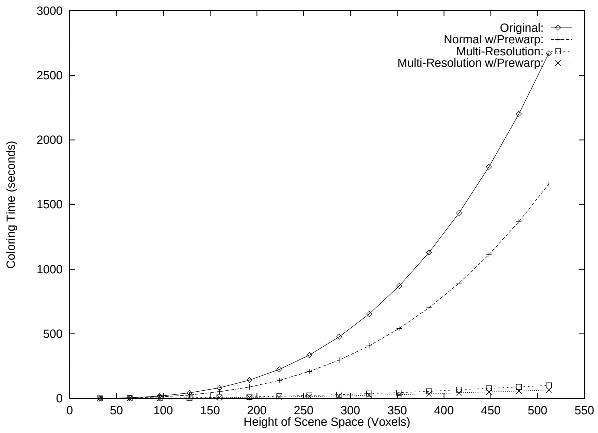

Figure 5: Running time vs. scene complexity for original and multi-resolution variant. Times are with and without prewarped input.

5 Dynamic Voxel Coloring

If we have video of a dynamic scene, we can take advantage of temporal coherence to avoid analyzing regions of scene-space that were determined to contain empty space at the previous time. This will work as long as the scene does not change too quickly and no new objects suddenly appear.

5.1 Using Temporal Coherence

To take advantage of the fact that the scene will be similar between two successive points in time, the lowest resolution coloring from time t_k can be used as the starting point for the next time t_{k+1}, thus eliminating the need to visit every voxel at the lowest resolution. However, rapid motion in the scene may cause this assumption to be locally violated. Again, some sort of search strategy must be used to locate regions of colored space that lie outside the seed coloring.

Search strategies for dynamic scenes can be more complex than those for static scenes. Besides problems of false negatives that arise at low resolutions, the search strategy must correct any false negatives due to object motion. Tracking methods could be used for this purpose. Also, the size of the search window could be varied as a function of the estimated velocity of surfaces.

If we employ the same search strategy as the coarse-to-fine coloring algorithm, rapid movement will cause the surface to be missed. However, as long as the surface does not move completely out of the search window, the algorithm will reconstruct the missing voxels at the next time step.

Assume the voxels used in the initial pass are roughly six inch cubes, and the input video rate is 30 Hz. If the nearest-neighbor search strategy is used, a single voxel would have to move roughly two voxels between frames to escape the search neighborhood in the next time step. This corresponds to a velocity of about 10 meters/sec. If the motion in the scene is less than this threshold, the scene will be reconstructed correctly. For the initial implementation of dynamic voxel coloring we made the assumption that all moving objects can be adequately tracked from frame to frame by simply using nearest-neighbor searches.

As each frame is colored, the seed coloring to be used for the next time needs to be updated to reflect any changes due to scene motion. After the seed coloring is augmented, the set of voxels is subdivided. While the subdivided voxels are being colored, the seed coloring for the next time instant is generated. If any voxel in the increased resolution space is colored, then the corresponding super-voxel of the new seed coloring is also colored.

The algorithm for dynamic scene coloring can be summarized as follows:

```
grab all images at time t = t_0
seedcoloring = low res coloring of scene

loop grab all images at current time t do
    augment = seedcoloring plus neighbors
    seedcoloring = emptyset
    for each voxel in augment do
        if voxel should be colored do
            mark supervoxel in
            seedcoloring as
            colored
        od
od
```

5.2 Results

The dynamic coloring algorithm was applied to a sequence of three time steps using images from four cameras (see Figure 6). The total time to color the sequence was 2.74 seconds. This compares to 3.70 seconds for coloring the three scenes separately. The performance gains are more dramatic if considered on a frame by frame basis. Table 2 summarizes the results and shows that the per frame speedup is more than two times.

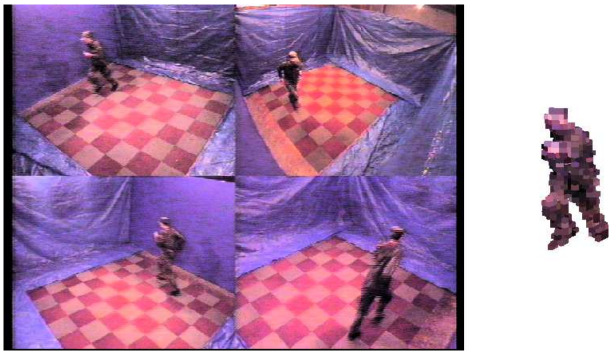

Figure 6: Four sample frames from input video and corresponding output.

Time Step	*Dynamic (sec)*	*Static (sec)*
Initial	0.945	n/a
Time 0	0.540	1.28
Time 1	0.644	1.24
Time 2	0.611	1.19

Table 2: Comparison of dynamic voxel coloring with the standard voxel coloring algorithm.

6 Conclusion

We have described three methods for improving the performance of voxel coloring. Taken together, speedups are significant and indicate that simple low resolution voxel scenes can be colored at rates of 1-2Hz on a medium-speed desktop computer. These rates were achieved by exploiting both spatial coherence and temporal coherence to avoid processing empty space.

References

[Kanade *et al.*, 1997] Takeo Kanade, Peter Rander, and P. J. Narayanan. Virtualized reality: Constructing virtual worlds from real scenes. *IEEE Multimedia*, 4(1):34–46, 1997.

[Moezzi *et al.*, 1997] Saied Moezzi, Li-Cheng Tai, and Philippe Gerard. Virtual view generation for 3d digital video. *IEEE Multimedia*, 4(1):18–26, 1997.

[Seitz and Dyer, 1997] Steven M. Seitz and Charles R. Dyer. Photorealistic scene reconstruction by voxel coloring. In *Proc. Computer Vision and Pattern Recognition Conf.*, pages 1067–1073, 1997.

[Szeliski, 1993] Richard Szeliski. Rapid octree construction from image sequences. *Computer Vision, Graphics, and Image Processing: Image Understanding*, 1(58):23–32, 1993.

[Tsai, 1987] R. Y. Tsai. A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf cameras and lenses. *IEEE Trans. Robotics and Automation*, 3(4):323–344, 1987.