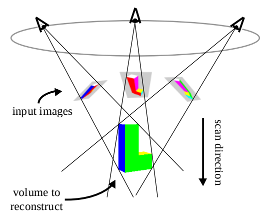
Voxel-coloring Algorithm*

```
for all images                                  /* clear occlusion bitmaps */
  set all pixels to not-occluded

for each layer of voxels along major axis {
  for each voxel V in layer {
     for each image i {
       find the set P(i) of pixels in the projection of V
       for all pixels of P(i) which are not-occluded
          add contribution to colorMean and colorStandardDeviation
     }
     if (colorStandardDeviation <  threshold) {  /* if colors match… */
       mark V opaque and color V with the colorMean
       for each image i
          set pixels in P(i) to occluded
     } else mark V transparent             /* if colors do not match… */
  }
}
```

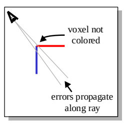
*Seitz and Dyer, "Photorealistic Scene Reconstruction by Voxel Coloring," 1997 IEEE Computer Society Conference on Computer Vision and Pattern Recognition

We start with a handful of images of a Lambertian scene, together with camera pose information for each. We use voxel-coloring to construct a volumetric model of the scene. Finally, we project the model to synthesize new images.



Synthesized images

Voxel-Coloring Enhancements

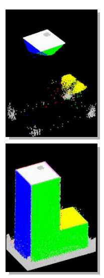


Problem: edges and other abrupt color variations

Where there is a large and abrupt color variation on a surface, for example at an edge, the corresponding voxel in the reconstruction projects onto pixels with a high color standard deviation. Hence, voxel-coloring fails to color the voxel for any reasonable color threshold. Worse, the occlusion bitmaps are not set correctly, so *no* voxels can be colored along the rays from the camera centers through the voxel. Thus, errors propagate.

Solution: treat edges specially

For reconstructing scenes with large, abrupt color variations, we modify the criteria for coloring a voxel. We color a voxel if either 1) the original criteria holds (standard deviation < threshold) or 2) for every image i, P(i) is either empty or includes a non-occluded pixel with a color significantly different from an adjacent pixel.



Problem: inconsistent color variation

Some surfaces in a scene can have a relatively high spatial color variation while others have a relatively low variation. If the color threshold is set low, voxel-coloring fails to color the areas with high variation. On the other hand, if the threshold is set high, small color variations are ignored, resulting in cusping in the areas with low variation. (Cusping is a distortion in which a surface in the reconstruction is warped toward the cameras, relative to the actual surface.)

Solution: use an adaptive color threshold

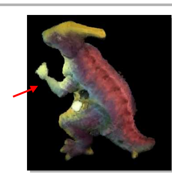
Compute the color variation among the non-occluded pixels in the individual P(i)s. Increase [decrease] the color threshold when these values are high [low]. For example, we have tried a threshold that is proportional to the mean of deviations in the individual images; we have also tried a threshold that is proportional to the minimum of the deviations.

Color Space Issues in Voxel-coloring

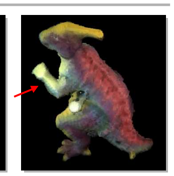
Our early experiments with voxel-coloring compared the color of voxels using Euclidean distance in the RGB cube. This caused two kinds of problems.

First, two colors that we can easily distinguish can be quite close together in RGB space if they are *dark*, whereas two *bright* colors that are relatively far apart can be hard to distinguish. Consequently, we performed comparisons in the perceptually uniform CIELab color space. There are areas of some scenes that fail to reconstruct accurately using RGB space and are significantly improved using CIELab.

Second, some image sets have inconsistent lighting, for example those obtained with a fixed light source and a subject rotating on a turntable. As a result, a point on a surface can project to different colors in different images and fail to be reconstructed. Using a color space that separates chromaticity and luminance, we can construct a color distance measure that weighs luminance less than chromaticity, thereby minimizing the problem of inconsistent lighting.



Reconstruction using RGB color space.

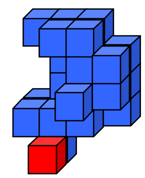


Reconstruction using CIELab color space. Note improved reconstruction of arm.

Blinking Cubes: Polygon-Based Scene Reconstruction

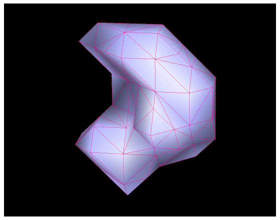
Binary Occupancy Volume

The algorithm maintains a binary occupancy volume that represents the current estimation of object shape. Each voxel in this space can be tested to determine if the object can be better represented with this region in space filled or vacant. The order for visiting this binary space can be the same as the original voxel-coloring algorithm or more sophisticated methods such as stochastic sampling can be employed.



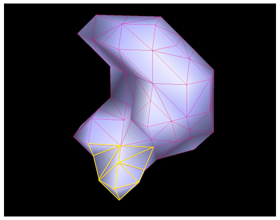
Surface Tessellation

A surface tessellation of the binary occupancy volume is computed via the *Marching Cubes* algorithm. The resultant set of polygons are projected to each image (using the occlusion information available via the current model) and colored with the average color sampled in all images. As an unexplored alternative, polygons could be assigned associated texture maps for higher detail.



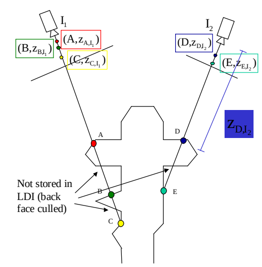
Differential Geometry

Testing a particular voxel in the binary occupancy volume implies an addition (and/or deletion) of new geometry. This differential geometry itself is projected to each source image, using the current model to provide accurate occlusion information. A simple measurement can be made to determine if the additional (or subtracted) geometry leads to an improved representation of the scene relative to the images captured of the scene. This information is used to accept or reject changes to the binary occupancy volume. Shown below are two images of a surface tessellation of the binary volume diagramed below left. Note the differential geometry has been highlighted.



Layered Depth Image

Note that a single evaluation of a voxel involves rendering the entire representation of the scene with and without the current voxel being tested. This is due to the fact that the occlusion information given by the current representation of the scene must be incorporated to determine which image pixels "see" the differential geometry. We can avoid a brute force implementation of this using a layered depth image per source image [Gortler, He, Cohen 97]. As shown in the accompanying figure this representation maintains depth and polygon ID information at each pixel, even for polygons that are not visible. In this manner, we need only update the layered depth image with the differential geometry being evaluated in each step.



Results:

* Original Images of Dinosaur model courtesy of Steve Seitz