Incremental Voxel Colouring by Ray Traversal

O. Batchelor, R. Mukundan, R. Green
University of Canterbury, Dept. Computer Science& Software Engineering.
owb13@student.canterbury.ac.nz, {Mukundan, Richard.Green}@canterbury.ac.nz

Abstract

Image based reconstruction from multiple views is an interesting challenge. Recently methods of optimisation based voxel colouring have appeared, which make use of incremental visibility updates.

We present an alternative incremental voxel colouring algorithm in the mould of GVC-LDI [2] which directly associates each visible voxel with the set of rays in it's visible projection, using ray-voxel traversal as a mechanism for updates.

We make some time comparisons using calibrated photographs and synthetic images, as well looking at optimisation based voxel colouring and level of detail. Results show that our method compares favourably to GVC-LDI.

Keywords— **Voxel Colouring, Ray Tracing, Reconstruction, Computer Vision**

1 Introduction

We introduce a derivative method of using ray traversal for determining visibility incrementally. Like Voxel Colouring [6], Space Carving [4] and GVC-LDI [2], the algorithm works on reconstructing a volumetric scene using colour consistency primarily.

Unlike earlier works, we have looked at using ray traversal as the primary method of updating visibility. Our algorithm works on an arbitrary collection of rays rather than a set of images. Though we convert calibrated images to a set of rays for input. Our primary reasons for looking into ray traversal are the potential of optimisation colour consistency and benefits associated with incremental visibility – fast convergence and flexible carving order.

Due to the nature of the problem we refer to rays and pixels interchangeably, where a ray can be generated from the inverse projection (of the calibrated image's projection matrix).

2 Background

Voxel Colouring [6] and derivatives are methods for reconstructing a scene from a set of calibrated images. A scene is represented by voxels, which are defined to be a unit of volume, in our case a cube. The primary characteristics of voxel colouring are it's explicit handling of visibility and it's local decision function for opaqueness, colour consistency.

Space carving [4] introduced the idea of conservative carving and defined the photo hull. This entails progressively carving from an estimate of the true scene, while using the estimate as an approximation for visibility. The space carving algorithm performs reconstruction with arbitrary view configurations by using a plane sweep method where a mask is used in each image to mark occlusion.

2.1 Generalised Voxel Colouring

Generalised Voxel Colouring [2] built on this idea, extending it to support full use of all scene views symmetrically, with arbitrary view placement. The key data structure used by GVC is an *item buffer* which stores the closest visible voxel for every pixel in an image (shown in 1).

The second (GVC-LDI) uses layered depth images of surface voxels as item buffers, as the voxel model is updated the layered depth images are updated incrementally. A LDI consists of surface voxel intersections, shown in figure 1).

The basic GVC algorithm performs updates in parallel across surface (SVL) voxels, and suffers from recalculation as the algorithm becomes close to converging, where as GVC-LDI is a serial process.

2.2 Reprojection error

A useful measure for evaluating the results of reconstructions, as well as a measure of colour consistency is the reprojection error. Where the output image of voxels are reprojected and the difference taken with input images on a pixel by pixel basis. A common distance used is the square of differences per component, equation 1, where $C1_\alpha$ correspond to a colour component, $\alpha = \{r, g, b\}$ for a RGB image.

$$dist(C1, C2) = \sum_{\alpha=1} (C1_\alpha - C2_\alpha)^2 \qquad (1)$$

2.3 Colour consistency

A colour consistency function is a binary function to determine if a voxel is opaque. It's purpose is to determine if it is possible for rays emitted from a scene point to form the colours which exist in the input images. For a scene of purely diffuse reflectors, this amounts to ensuring the colours of the scene point are approximately equal.

One consistancy function is testing the reprojection error of a voxel (equation 2) against a threshold. proj(V) is the set of pixels in the projection of a voxel (to all images), vis(V) is a subset of proj(V) where each pixel is visible. C_p is the colour of pixel P, and C_v is the colour of voxel V.

$$error(V) = \sum_{P \in vis(V)} dist(C_p, C_v) \quad (2)$$

Another simple function involves a threshold on (mean) differences between groups (where groups are the set of pixels projected to each image). We have used this simple test extensively for purposes of comparison, it is shown in equation 3. rays(V, i) is a subset of vis(V) for rays from image i, nrays refers to the number of rays. C_i and C_v are the mean colour of rays(V, i), and the mean colour of vis(V).

$$between(V) = \frac{\sum_{i=1} dist(C_i, C_v) * rays(V, i)}{\sum_{i=1} nrays(V, i)} \quad (3)$$

2.4 Voxel colouring as optimisation

Optimisation based colour consistency involves minimising some global function (such as the global reprojection error). Two approaches are minimising reprojection error [7], and maximising probability with a statistical model [3]. Both methods use GVC-LDI to evaluate visibility.

Carving a voxel V changes the colour of a set of voxels, the Changed Visibility Set CVS(V). The pixels/rays for which the reprojection error may change is the union of vis(U) for each voxel U in CVS(V), we term this the Changed Visibility Ray Set CVRS(V) shown in 4.

$$CVRS(V) = \bigcup_{U \in CVS(V)} vis(U) \quad (4)$$

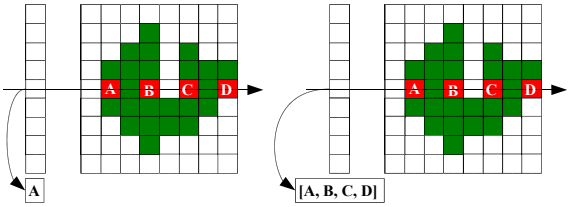

Figure 1: An item buffer, and a LDI

A binary colour consistancy function can be found by assuming the pixels in projections of each voxel are independent, defined as comparing the total sum of reprojection errors in CVSR(V) before and after carving. Equation 6 describes such a function for reprojection error.

$$reproj(V) = \sum_{P \in CVRS(V)} distSq(C_v, C_p) \quad (5)$$

$$consist(V) = reproj(V) \leq carved(reproj(V)) \quad (6)$$

One of the problems with this approach is that it contains many local minima. Far from the global minimum reprojection error is rather noisy, on it's own no reasonable output is produced. Slabaugh et al. [7] treat optimisation as a post process from an existing voxel colouring algorithm, as well as looking at simulated annealing with an adding/removing process.

2.5 Voxel traversal

Voxel traversal on a regular grid consists of line drawing. The voxel traversal algorithm of Amanatides and Woo [1] computes exactly the intersections of all voxels along the path of a ray which is precisely the information required for our purposes. It is particularly efficient, it involves just 2 floating point comparisons, one addition and an integer comparison.

We use ray traversal to find the next solid voxel along a ray, this is shown in figure 2, where voxels [A, B, C, D, E, F, G] are traversed and the algorithm stops at voxel H.

3 Approach

The common factor required for colour consistency tests is vis(V). Generalised Voxel Colouring [2] uses rasterization to determine proj(V) and vis(V) contains each element proj(V) equal to it's corresponding entry in an item buffer.

Instead of looking to efficiently evaluate vis(V), we look at an approach where vis(V) is *updated* incrementally

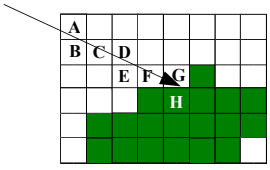

Figure 2: Ray traversal.

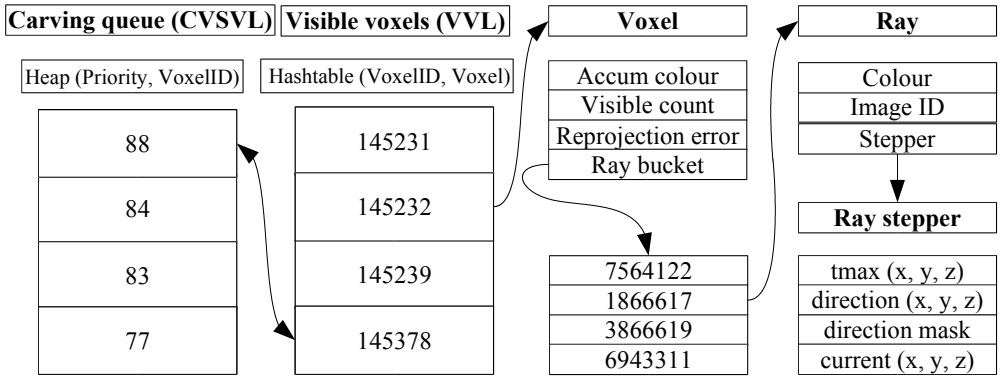

Figure 3: The major data structures and their connections.

for all visible voxels (we call this the Visible Voxel list (VVL)), as a result when a voxel is to be evaluated Vis(V) is available immediately to pass to a colour consistency function. The VVL is a subset of the SVL from [2], as there may be surface voxels which are not visible.

3.1 Initialisation

Initialisation is performed by accumulating all generated onto the initial voxel volume. In practice this occurs by generating one ray image at a time, pairing it with an input image and extracting each ray one by one then accumulating it to the VVL hashtable. This is done image by image so that only a small part of the dataset need be in memory twice at any one time. Afterwards the images needn't remain in memory as the information (colour, image number etc.) has been stored with each ray.

3.2 Carving process

Voxel ray traversal is used to update the VVL structure. Each ray stored (each member of vis(V)) also stores ray stepping information. When a voxel is carved, rays in Vis(V) are accumulated to other voxels which are or will become members of the VVL, the particular voxels is the CVS(V), the set of voxels found by ray traversal for rays from vis(V).

A queue is used to maintain the voxels which require evaluation, only those which change visibility during a carving operation require re-evaluation. This is the same role as Changed Visible Surface Voxel List from GVC-LDI [2]. The data structures used are outlined in figure 3.

The VVL is implemented as a hashtable mapping Voxel IDs to a structure containing vis(V). We use the term *bucket* for this structure, as it behaves as water poured into a bucket, when the voxel is carved the water is tipped out and "falls" into other buckets. The bucket may also be used to update useful statistics, for example current reprojection error of a voxel.

3.3 Algorithm

At the top level, the process involves iterating the following steps:

1. Remove a voxel from the carving queue (CVSVL)
2. Evaluating the visibility change of carving voxel
3. Evaluate colour consistency
4. If inconsistent then make visibility changes from (2)
5. Add changed visibility voxels to the carver queue

The algorithm given makes the update process more explicit. A set of "candidates" are generated at each iteration which are voxels which change visibility given the carving of the voxel in question. The structure used for Candidates(U) is the same "bucket" structure of that used in the VVL(V).

When candidates are committed to the main VVL, their priority is also updated in the carving queue - we use a heap which is doubly linked into the VVL hashtable, when priorities are updated elements in the heap are moved about and links are updated.

```
initialise Solid

Generate rays from input images
For each ray R {
    Find first solid voxel V
    Accumulate R to VVL(V)
}

Set CVSVL to voxels in VVL

while(CVSVL is not empty) {
    remove voxel V from head of CVSVL
    lookup vis(V) from VVL

    for ray R in Vis(V) {
```

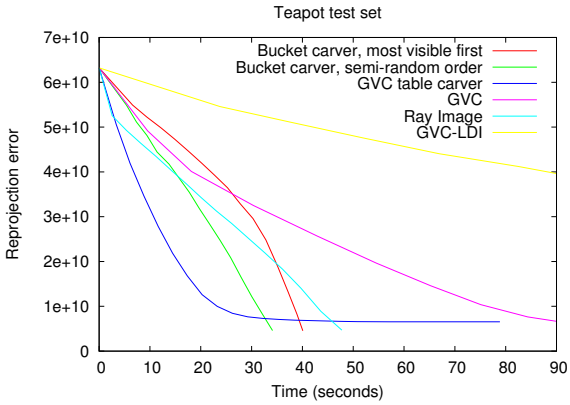

Figure 4: Reprojection vs. time for several algorithms

Figure 5: Reconstructed teapot scene.

```
            step ray R
            U = next Solid voxel on ray R
            if not null(U)
                add R to Candidates(U)
        }

        if(not Consist(vis(V), Candidates) {
            carve V from Solid
            remove V from VVL

            for each Voxel U in Candidates {
                Accumulate rays in
                    Candidates(U) to VVL(U)
                Update priority of U in CVSVL
            }
        }
}
```

3.4 Consistency Inputs

Different colour consistency functions require different input parameters, of the two classes (threshold based, optimisation based) threshold based functions (equations 3, 2) require primarily vis(V) as inputs. Optimisation based colour consistency (equations 4, 5), requires the changed visibility (voxel) set CVS(V), and changed visibility ray set CVRS(V) (equation 4).

For a threshold based colour consistency function, the generation of candidates and evaluation of colour consistency may be swapped without changing the result (to improve efficiency).

4 Comparisons

We perform three comparisons. We evaluate time and space complexity between various voxel colouring algorithms and our buckets algorithm with the teapot dataset using threshold colour consistency. We then evaluate it's use for level of detail and for optimisation colour consistency. We make use of three datasets, "Teapot" a set of rasterized images, calibrated photo sets of Cactus and Gargoyle, courtesy of Pr. Kyros Kutulakos (University of Toronto). We use one system throughout comparisons, an Athlon 2600+, with 768MB of RAM and a Geforce FX 5200 graphics card.

4.1 Threshold colour consistency

We have compared our algorithm to several others (at a common level) using threshold colour consistency test using equation 3. We compare time vs. reprojection error, as well as runtime statistics of memory use. The implementations we compare are our implementations of GVC, GVC-LDI, an version of optimised GVC using OpenGL for rasterization (GVC table), and earlier work using ray traversal with item buffers (Ray–image). We endeavoured to use optimised versions of each algorithm, most sharing significant common code for rasterization and volume representation.

The first test set consists of a teapot scene with 15 800×600 generated images (by rasterization). The reconstructed resolution of the teapot scene was at 120^3.

All six methods successfully reconstruct the teapot to a similar quality shown in figure 5, there are minor differences due to carving order. In addition vis(V) differs slightly between rasterization, OpenGL rasterization and ray traversal. Figure 4 shows that the GVC table method faster than the others (though converges slowly), followed closely by the Bucket method, GVC and GVC-LDI significantly slower.

To show the differences resulting from ordering we have shown two carving orders for the Bucket carver, the reason for the hashtable ordering (Also used by GVC-LDI and Ray–image) being faster is simply that it creates more surface noise and uses less rays per voxel.

Of the two classes of algorithm, it can be seen that incremental methods $\{$GVC-LDI, Ray–image, Bucket$\}$ carve at a nearly constant rate, but GVC based methods trail off rapidly towards convergence. All algorithms have a total


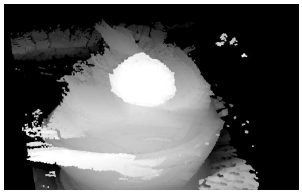
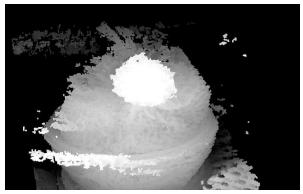
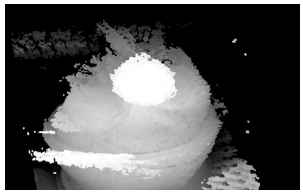

Figure 6: Original. Figure 7: Threshold. Figure 8: Reprojection error. Figure 9: Probabilistic.

running time approximately on the order of voxels $\times$ rays per voxel. For each voxel, calculating vis(V), stepping rays and evaluating colour consistency are all $O(n)$ for the number or rays.

Method	Memory use (MB)
Images alone	54.5
GVC table	70.7
GVC	87.6
Bucket	225.0
Ray–image	311.8
GVC-LDI	538.3

Memory use was significantly higher for incremental methods. Memory use for the Buckets method is dominated by an order of the number of rays, which diminishes as carving proceeds. GVC-LDI has a problem where memory use is not entirely deterministic throughout the process. A surface with more noise creates more surface intersections which increases the list size per pixel. This could be resolved by using a priority queue for ordering.

4.2 Optimisation colour consistency

We looked at application of the bucket method to optimisation voxel colouring as [7, 3]. We have looked only at a carving method as opposed to [7] which involved a process both adding and removing voxels. Adding voxels would complicate the bucket algorithm because it would involve removing rays from existing buckets, as well as forcing a mapping from images back to rays. This is one area where GVC-LDI is more flexible.

We have tested both implementations on the cactus test set, a set of calibrated photos of a complex cactus plant with very fine needles in places. We first ran a thresholding carver over the data set to reduce time taken, though using level of detail is also equally useful for this purpose. We used a combination of threshold and optimisation to reduce local minima which seem to be the main problem with this type of consistency function.

Method	Time (s)	Reduced reproj.
Reprojection error	82.0	39.7%
Probabilistic	221.7	28.1%

Both reduced reprojection error without destroying the integrity of the model as happens by directly reducing the threshold. Noisy voxels are not greatly reduced, in fact they both seem to make a considerably noiser surface - as there are no surface constraints. Results of each are shown in figures (8, 9). Both take considerably longer than threshold consistency functions, but show a small improvement in reconstruction quality.

4.3 Level of detail

We have looked at the applicability of using level of detail to reduce the large reconstruction period at high resolutions, for both GVC and the bucket method. We have used the method first shown by Proc and Dyer in [5]. It begins with a low resolution voxel volume and repeats steps outlined below.

1. Carve voxels.
2. Augment voxel model.
3. Double resolution (along each axis)

We anticipated that the bucket method may be particularly suited to this technique because it is fast when convergence is close. We ran a comparison between GVC with LOD, the bucket method with LOD, and the bucket method without LOD as a control. Resolution began at 30^3 and increased in resolution 4 times to reach 240^3. A sequence of the intermediate reconstructions at each resolution is shown in figure 12, the original image from the same view 11, and a graph of reprojection error 10.

Results show that the technique worked better with GVC, the bucket method proved somewhat slower because of the extremely slow re-initialisation period involving ray-tracing the intermediate voxel model. This could be solved by using additional spacial acceleration structures. Clearly LOD pays off for both methods however, as they are much faster than the control. The only disadvantage is a possible loss of fine detail, however this also helps reduce noise.

Conclusions

Using ray traversal for updating global scene visibility for a voxel carving process seems to compare favourably with GVC-LDI greatly in time and space use. It also compares well with GVC, however the use of hardware rasterization is clearly beneficial resulting in faster running times

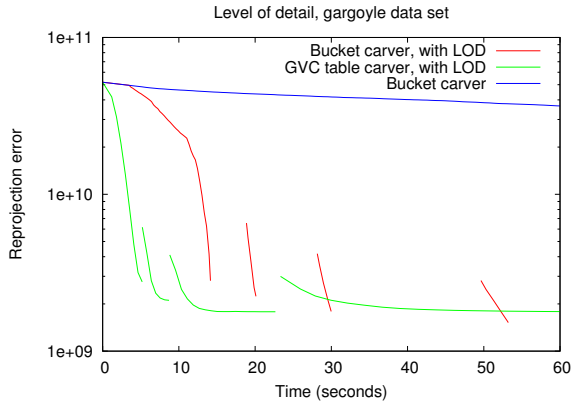

Figure 10: Reprojection vs. time for LOD sequence

Figure 11: Input image of gargoyle data set.

Figure 12: Level of detail sequence 30^3 to 240^3.

with GVC, in future perhaps a similar ray traversal algorithm can be accelerated using parallel graphics hardware.

In terms of flexibility it is tricker to implement a scheme allowing adding of voxels than GVC-LDI, however it has been shown that optimisation based colour consistency are feasible with the buckets method. It's use with level of detail is hampered by a slow initialisation period, though this could be solved by using spacial acceleration structures for initialisation.

Acknowledgements

We would like to thank Pr. Kyros Kutulakos for making the "Gargoyle" and "Cactus" datasets available for comparisons.

References

[1] John Amanatides and Andrew Woo. A fast voxel traversal algorithm for ray tracing. In *Eurographics '87*, pages 3–10. Elsevier Science Publishers, Amsterdam, North-Holland, 1987.

[2] W. B. Culbertson and T. Malzbender. Generalized voxel coloring. In *Proceedings of the ICCV Workshop*, volume Vision Algorithms Theory and Practice, September 1999.

[3] Ho-Won Kim and In So Kweon. Optimal photo hull recovery for the image-based modeling. In *The 6th Asian Conference on Computer Vision (ACCV), Jeju, Korea*, January 2004.

[4] K. N. Kutulakos and S. M. Seitz. What do n photographs tell us about 3d shape?, January 1998. Computer Science Dept. U. Rochester.

[5] Andrew C. Prock and Charles R. Dyer. Towards real-time voxel coloring. In *Proc. Image Understanding Workshop*, pages 315–321, 1998.

[6] Steven M. Seitz and Charles R. Dyer. Photorealistic scene reconstruction by voxel coloring. In *CVPR '97: Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97)*, page 1067. IEEE Computer Society, 1997.

[7] G. Slabaugh, B. Culbertson, T. Malzbender, and R. Schafer. Improved voxel coloring via volumetric optimization, 2000.