ShadeTree Image Compression for Embedded Computing

Ruben Gonzalez
NICTA, 300 Adelaide St, Brisbane, Australia
and
Institute for Intelligent Integrated Systems
Griffith University, PMB 50, Gold Coast Mail Centre, QLD, 4217
Email: R.Gonzalez@gu.edu.au

Abstract: This paper presents preliminary results of a new image compression method. The 'ShadeTree' method provides compression performance approaching the standard JPEG format but with greatly reduced computational complexity. It works by tessellating an image into a set of smooth shaded polygons that can be directly rendered using standard computer graphics algorithms without further processing.

Keywords: Image compression, embedded computing

1. Introduction

In very low power and embedded computing environments performance constraints often limit the use of computationally complex image compression standards. Decoding a 1.35 MB JPEG image on a 16.5 MIPS microcontroller for example takes 49 seconds. [1]. Alternatives such as quadtree based image compression have been investigated for many years [2, 3]. One of its main attractions is its decoding simplicity and good overall performance. It provides a mechanism to perform adaptive or non-uniform sampling of an underlying image based on its local statistics. In a quadtree, each leaf is used to represent the colour of uniform regions in tessellated images. Due to this point-sampling characteristic, quadtrees typically split up non-flat shaded regions into smaller and smaller regions until their variation falls below a given threshold. A number of adaptations have been proposed to improve quadtree based coding in the context of smooth image shading [4]. These are based on approximating the colour between leaf nodes using first or higher order interpolation. They may also use iterative optimisation methods together with node pruning and merging strategies improve coding performance. These methods have been reported to achieve performance on par with if not better than the wavelet based JPEG2000 image compression standard [5].

The various enhancements made to quadtree schemes to match the performance of transform and wavelet based approaches have come at the cost of increased complexity. Similar coding performance means similar coding complexity and more importantly decoding complexity. In contrast the work reported in this paper was based on developing a compressed image format met the following objectives:

(a) Reduce computational complexity for low power computing.

(b) Reduced algorithm complexity for ease of implementation.

(c) Compression performance same as or similar to existing compression standards.

2. Interpolating Quadtrees

A traditional quadtree recursively divides an image into quadrants. Each quadrant is described as either a node if it has children or a leaf that containing the colour value of the region it represents as shown in Figure 1 below.

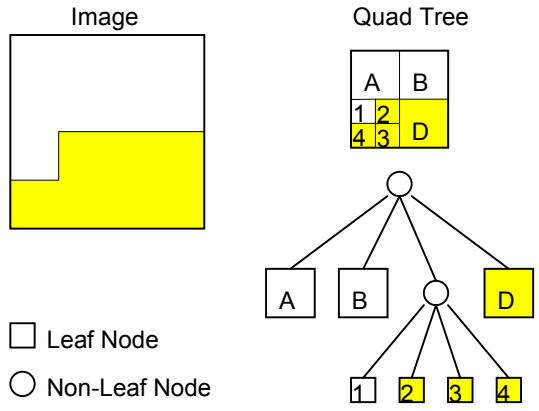

Figure 1 Typical Quadtree Structure

This approach can be extended to handle smooth shaded regions by using the leaves to represent

point samples at the centre of each region instead of area samples and interpolating pixel values between leaves as shown below in Figure 2

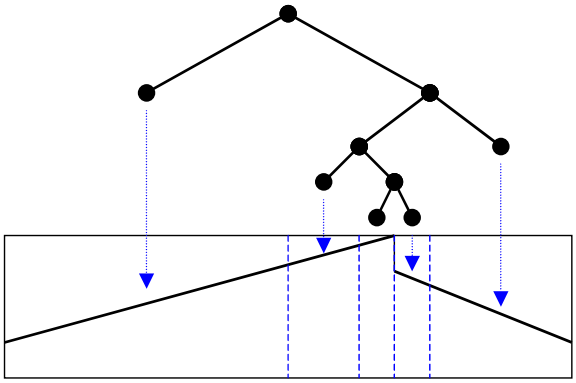

Figure 2 Interpolating leaf values

In the case of two-dimensional images, the non-uniform sample distribution created by the quadtree makes direct bilinear interpolation impossible. Difficulties arise with treatment of image boundaries, and non-equidistant spaced leaf values. These problems are normally overcome by using methods such as cubic spline surface fitting with assumed smoothness constraints. With this and similar approaches, the interpolated pixel values in each quadrant are dependent on usually at least four leaf values in each direction of neighbouring quadrants. These methods are reasonably effective but tend to be as computationally complex as standard compression methods such as JPEG being of the order $O(n^2 \log n)$.

3. Shaded Quadtrees

A simpler solution would be to interpolate in from the corners of each quadrant rather than out from the centres. Each quadrant can then be processed independently of its neighbouring quadrants. In this case instead of providing a single sample at the centre of the region at each leaf we provide values of samples for each region corner as shown in Figure 3.

In this situation we can directly use bilinear interpolation without the need for any assumed constraints. As corners may be coded multiple times a simple prediction scheme may be used to avoid recoding an edge that has already been processed.

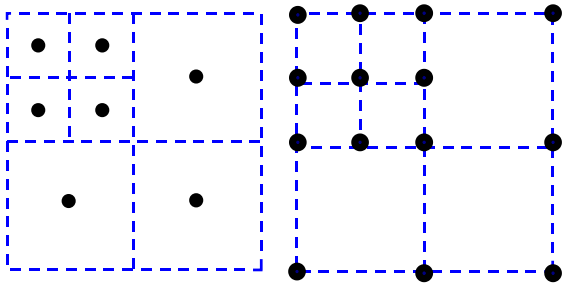

Figure 3 Corner Sampling

In the case of images, we start with the values at the four corners of the image. In a traditional quadtree structure splitting a quadrant generates four (area) leaf node values. In a ShadeTree, splitting a quadrant requires values for each of the corners of the quadrants that will be formed by the split that are not already available. In the example shown in Figure 4 the left side shows an image before being split. Splitting the image into the first four quadrants requires initialising the encoding by sending the corners of the original image, followed by the values 0…4 as shown. Next, splitting the top right quadrant requires sending values 5…9.

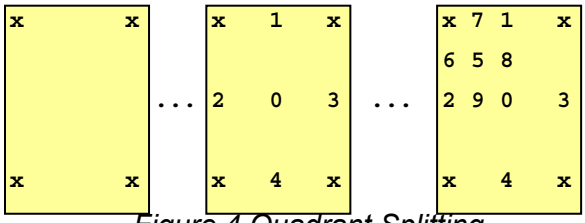

Figure 4 Quadrant Splitting

Statistical redundancy is then removed from the resulting linear tree structure through the use of an appropriate symbol encoding method.

The entire image is essentially transformed into list of shaded polygons. Decoding simply involves reading the corner values for each quadrant from the linear quadtree and passing them to a shaded polygon render to be drawn to the display buffer. Shaded polygon rendering is an algorithmically simple and very high-speed operation that is the core of all computer graphics systems, whether implemented in hardware or in software. The decoding algorithm is as follows:

```
Begin
    Read image corner values a,b,c,d from input
    Call FillQuadrant(a,b,c,d) for entire image
End

FillQuadrant (TL, TR, BL, BR)
Begin
    Read first (X) byte from input
    If value is zero then return
    FOR current quadrant Do
    Begin
        If we don't have L value read input
        If we don't have T value read input
        If we don't have R value read input
        If we don't have B value read input
        Call FillQuadrant (TL, T, L, X)
        RenderPolygon (TL, T, L, X)
        Call FillQuadrant (T, TR, X, R)
        RenderPolygon (T, TR, X, R)
        Call FillQuadrant (L, X, BL, B)
        RenderPolygon (L, X, BL, B)
        Call FillQuadrant (X, R, B, BR)
        RenderPolygon (X, R, B, BR)
End
```

4. *Experimental Results*

The performance of ShadeTree was compared against traditional quadtree and JPEG coders in terms of rate distortion and decoding speed. The decoding tests were performed using a 384x256 grey scale casa.png image and the rate distortion tests were performed using a 512x512, grey-scale LENA image.

Codec	Ticks
RLE	0.14
ShadeTree'	0.55
JPEG	1.0

Table 1 Decoding Times

The normalised decoding performance of the ShadeTree algorithm as shown in Table 1 indicates that it is roughly twice as fast as JPEG and just over three times slower than simple run-length encoding. Being a work in progress a few caveats need to be taken with regard to the results. First the highly optimised IJG library was used to implement the JPEG encoder/decoder while the ShadeTree implementation was written for clarity rather than speed. The result for the ShadeTree coder didn't consider entropy-coding overheads, but does include the time for shaded polygon rendering in software.

The rate distortion results in Figure 5 demonstrate that the ShadeTree together with an entropy coder performs better than standard quadtree coders and close to but not as well as JPEG. (The decoded image is shown in Figure 6 and 7.) This result is largely due to not having used a bit allocation process to optimise the rate distortion performance. It has been shown that use of iterative bit allocation techniques in tree-structured codecs can achieve performance exceeding that of both JPEG and JPEG2000 [5]. We are confident that the performance of the ShadeTree method can significantly be improved by optimising bit allocation using appropriate bit allocation methods.

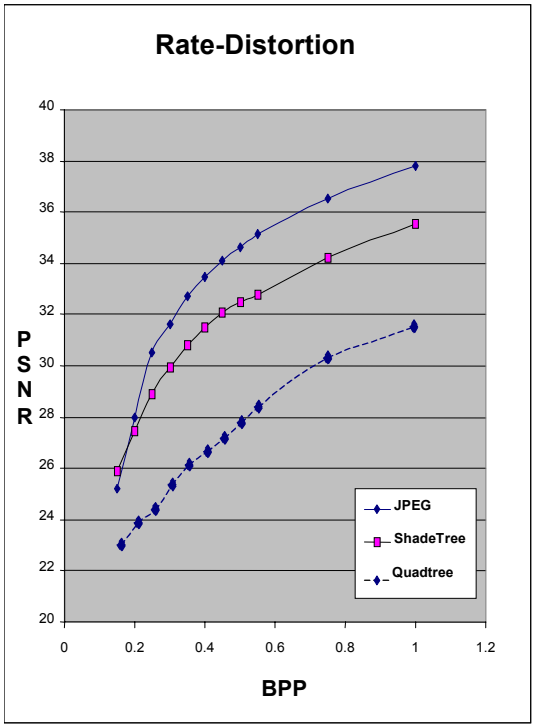

Figure 5 Rate Distortion Performance

Figure 6 JPEG (L) & ShadeTree (R) at 0.5 BPP

Figure 7 Casa image ShadeTree encoded at 0.45BPP

5. Conclusions

This paper has presented a novel variation on the traditional quadtree method for image compression. The ShadeTree approach represents images as a set of smooth shaded polygons. Decompression is both simple and fast while its rate distortion characteristic approaches that of the JPEG compression standard. A number of well known improvements can be made to the encoding process such as using an iterative bit allocation method to optimise its rate distortion performance. Improvements can also be made to the image quality in areas of high variations in the image by Haar transform encoding those regions. The method is particularly suited to low power microcontrollers or simple hardware implementation.

6. References

[1] Stefan Kuhr, "Implementation of a JPEG decoder on a 16-bit microcontroller" Masters Thesis, Department of Mathematics and Computer Science, Fachhochschule Stuttgart, Germany February 2002.

[2] G.J. Sullivan and R.L. Baker, "Efficient quadtree coding of images and video," IEEE Trans. Image Process. vol. 3, no. 3, pp. 327–331, Mar. 1994.

[3] M.Rabbani, P.W. Jones, "Digital Image Compression Techniques." Bellingham, WA: SPIE, 1991.

[4] Strobach, Peter, "Quadtree-structured recursive plane decomposition coding of images" IEEE Transactions on Signal Processing, vol. 39, No.6, June 1991, p. 1380-1397.

[5] Rahul Shukla, Pier Luigi Dragotti, Minh N. Do, and Martin Vetterli, "Rate-Distortion Optimized Tree-Structured Compression Algorithms for Piecewise Polynomial Images" IEEE transactions on image processing, vol. 14, no. 3, march 2005 p.343-358