Compression of Raster Images Using Vertex Representations

Karl Döhm Claudio Esperança

December 9, 1993

Abstract

The application of vertex-based representations for storing images and its potential as a compression technique is discussed. Vertex representations are data structures that have been successfully used in the domain of VLSI circuit design by virtue of being very compact. A review of vertex representations and of standard image compression schemes is presented. An algorithm for transforming pixel rasters into vertex rasters is described. This algorithm was applied to a series of test images and the resulting vertex structures were further compressed using two popular data compression techniques: Huffman encoding and the Lempel-Ziv-Welsh algorithm. Strengths and weaknesses of vertex rasters are inferred from these experiments and directions for further research are suggested.

1 Introduction

It is difficult to overestimate the need for faster and more economical ways of storing and processing images. The search for efficient image compression schemes has been particularly intense in the last decade. Image compression schemes can be loosely divided in two 0 - *lossy* and *lossless*. Lossy compression schemes may compromise a fraction of the overall visual quality of the image in order to achieve high compression rates – this approach is adequate for most applications. When information must be exactly preserved, however, lossless compression techniques are still able to significantly reduce memory requirements.

An image, in order to be processed by computers, must be converted to a rectangular array of cells, which are called *pixels* (short for "picture element"). A pixel is a binary number representing the color of the image at that particular location. The mapping between the value of the pixel and the color it represents is what is called a *color table*. In terms of memory storage, the key factors are:

1. The *resolution* of the cell array (also called *raster*). The more cells in it, the more memory space it is likely to take.

2. The size of the color table, i.e., the number of different colors in the image.

Most compression schemes are applied on images represented as rasters. The compression process starts by analyzing these rasters in order to discover patterns of shapes and colors. The compression is ultimately achieved by representing recurring patterns in a compact way. Consider for instance an image where all pixels are black: instead of scanning the raster row by row and storing the color value for each pixel, it would be more compact to store the number of consecutive pixels with the same color and its value – this last scheme is called *run-length encoding* [7]. This example illustrates a typical expedient used in image compression, namely, that of using a different way of describing an image. In addition to run-length encoding, we may also cite the *discrete cosine transform* [1] and hierarchical coding employed in the *JPEG* image format [5].

Vertex-based representations [8, 3] provide still another way of describing images. Generally speaking, vertex representations extend the concept of run-length encoding to n-dimensional rasters, that is, they try to cluster regions with same value together and represent only the changes in value and the positions where these changes occur. This property makes vertex representations of potential use in the field of image compression.

The rest of this paper is divided as follows: Section 2 reviews vertex representations and describes an algorithm for converting a pixel-based raster to a vertex-based raster; Section 3 presents basic concepts of image compression; Section 4 describes two image-compression programs using vertex encoding; Section 5 presents results obtained by applying these as well as other data-compression programs on a series of test images; Section 6 draws some conclusions from the experiments and suggests other ideas for the use of vertex representations on the image-compression problem.

2 Vertex Representations

The pioneering work on vertex representations was done by J. Schechtman [8] and was targeted for use in the design of VLSI circuit masks. His work demonstrated that two-dimensional vertex structures are very compact and presented algorithms for performing set-theoretic operations, region expansion and region contraction on them.

Vertex structures are used to describe mathematical entities called scalar fields. A scalar field is a function that maps each point of the space to a number (*scalar*). For instance, the function

$$f(x, y) = x + y$$

is a scalar field that maps points with coordinate values (x, y) to the number given by $x + y$. An image can be regarded as a visualization of a scalar field, where the value of the field at each point is interpreted as a color. A rendition of the scalar field given above is shown in Figure 1, for the portion of the plane given by $0 \leq x, y \leq 1$ and with values mapped to shades of gray.

The term *vertex* refers to a point with an associated scalar value called *weight*. A vertex with a weight of α placed at a point (x_0, y_0) of the plane increments by α the value of the scalar field at all points (x, y) such that $x \geq x_0$ and $y \geq y_0$ - those points are

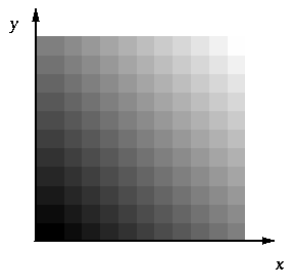

Figure 1: Scalar field given by $f(x, y) = x + y$ plotted for the region defined by $0 \leq x, y \leq 1$. Points where $f(x, y) = 0$ are painted in black and points where $f(x, y) = 2$ are painted white, with intermediate values painted in shades of gray

said to lie in the *cone* of the vertex. In general, any raster image can be constructed by placing vertices of proper weights in proper positions of the plane. Figure 2 illustrates how the placement of a few vertices affects the value of the scalar field.

The problem of computing a set of vertices equivalent to a n-dimensional raster has been solved in [3] . We will present here a somewhat different version of that algorithm restricted to two-dimensions. The pseudo-code for algorithm $pixel_to_vertex$ is given below. It takes as input a raster image P with pixels organized in m rows and n columns and outputs in V an equivalent vertex representation. V is also organized as a raster, that is, a cell of V in row y, column x contains the weight of a vertex located at (x, y) – a weight of zero means that no vertex exists at that point. We use the notation $P[x, y]$ to refer to the cell at row y, column x of P (and similarly for V). Figure 3 shows the conversion of a 4×4 raster.

```
procedure pixel_to_vertex(P, V, n, m)
    for y varying from 1 to m do
        for x varying from 1 to n do
            if y = 1 then
                if x = 1 then V[x, y] ← P[x, y]
                else V[x, y] ← P[x, y] − P[x − 1, y]
            else
                if x = 1 then V[x, y] ← P[x, y] − P[x, y − 1]
                else V[x, y] ← P[x, y] + P[x − 1, y − 1] − P[x − 1, y] − P[x, y − 1]
```

3 Basic concepts of image compression

Electronic images typically consume large amounts of storage. Image compression deals with the algorithms which are used to reduce the data needed to recreate the electronic

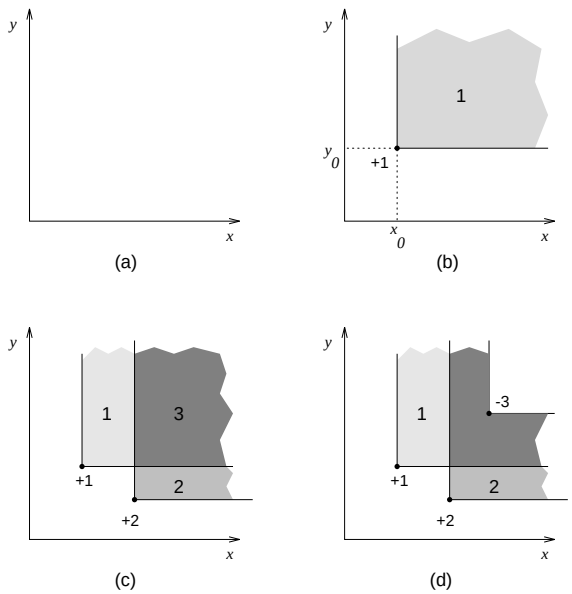

Figure 2: Influence of the placement of vertices in the value of a scalar field. Scalar field in (a) maps all points to 0; placement of a vertex with weight $+1$ at (x_0, y_0) as shown in (b) increments the value of the scalar field for all points (x, y) where $x \geq x_0$ and $y \geq y_0$; other vertices are placed with values $+2$ and -3 changing the field as shown in (c) and (d) respectively.

image, or a likeness of it, into as compact a form as possible. For typical complex images, the large quantity of pixels and colors represent a classic large scale encoding/compression problem for algorithm designers.

This section attempts to bridge the reader who is less experienced with image representation and compression to the subject matter at hand. First, a review of basic image representation is presented, followed by an overview of basic image compression.

Image representation is driven by the hardware implementation of graphics adapters and displays. The basic addressable unit of information on the raster display is the pixel, and a pixel can be associated with one color. There are typically many pixels associated with an image, for example over 300K pixels are needed to display a standard 640x480 image.

A pixel color is represented in most computer systems by a combination of three standardized versions of the colors red, blue and green. Most but not all possible colors can be represented by combining the standard three colors at varying intensities. A color value for a pixel is specified by a three-tuple of intensities for standard red, blue, and green. The standard colors are calibrated so that if the three are displayed at the same intensity for a given pixel, that pixel takes on a shade of grey. By varying the intensities uniformly across the three standard colors, an entire range of grey colors can

P

0	1	2	4
0	1	1	2
1	2	3	1
1	1	2	2

V

0	0	+1	+1
-1	0	-1	+3
0	+1	0	-1
+1	0	+1	0

Figure 3: Example of a call to procedure $pixel_to_vertex$. P is a pixel-based raster and V is an equivalent vertex-based raster.

be displayed. This is referred to as the grey scale. One can uniquely identify a grey scale value for a pixel by specifying exactly one intensity value.

If one were to limit the amount of colors which could be simultaneously displayed to some small subset of all possible permutations of the standard colors and intensities (the palette), then the compression size for a color version of an image can become comparable to the size of the grey scale version.

The vertex representation approach lends itself to encoding the pixel to value relationship, which is in its most basic form in gray scale representations. We chose to perform our investigation with grey scale images and leave the extension to color imagery as a potential future investigation.

One popular format for expressing grey scale information for an entire image is the Portable Grey Map (*PGM*). It simply contains a general header and one byte of intensity information for each pixel. The intent is to be portable for conversion to multiple formats. The vertex compression encoding in our investigation uses *PGM* format as input.

An image compressor applies one or several encoding schemes to the basic straight pixel for pixel image representation. Algorithms to do so can be termed lossy or lossless depending on whether information is lost during compression. Lossy compression systems, like *JPEG/JFIF*, work by storing an image which highly resembles the original one but can be represented much more compactly. These systems rely on a statistical analysis of the original image, and the degree of trade-off between compression rate and fidelity can typically be controlled by the user.

Lossless encoding systems are so named because information is not lost during the compression process. Instead of being statistically based, the algorithm relies on string compression schemes such as Huffman, Adaptive Huffman, Lempel-Ziv-Welsh, and arithmetic coding. *GIF* is a common lossless compression format and *JPEG* standard also offers a lossless compression mode of operation, albeit somewhat uncommon.

Our investigation involved using vertex representations as an additional tool for lossless image compression. Further investigation may reveal usefulness in lossy compression algorithms as well.

Due to its popularity, we chose *GIF* as a baseline comparison for compression performance in our investigation. One notable limitation of *GIF* is that it only works with images consisting of 256 or less colors.

Different image compression schemes can have widely different results when encoding image data. The variance is due to the strengths and weaknesses of the associated algorithms coupled with the unique properties and fluctuations of data across pixels. Section 6 will demonstrate some of the performance differences across lossless compression methods.

4 Compressing vertex-based rasters

We have implemented two programs for compressing gray-scale images. Both consist of transforming the original pixel-based raster into a vertex-based raster (as per procedure *pixel_to_vertex*) followed by the application of a string compression algorithm. The first program, called *vtx_huff* uses a non-adaptive Huffman code compressor [4], and the second, called *vtx_lzw* uses the Lempel-Ziv-Welsh compression algorithm [10]. In order to have a standard for comparison, these two string compression algorithms were also implemented without the pixel-to-vertex conversion step – we will refer to these programs as *huff* and *lzw*, respectively. A block diagram illustrating these four ways of compressing a gray scale image is shown in Figure 4

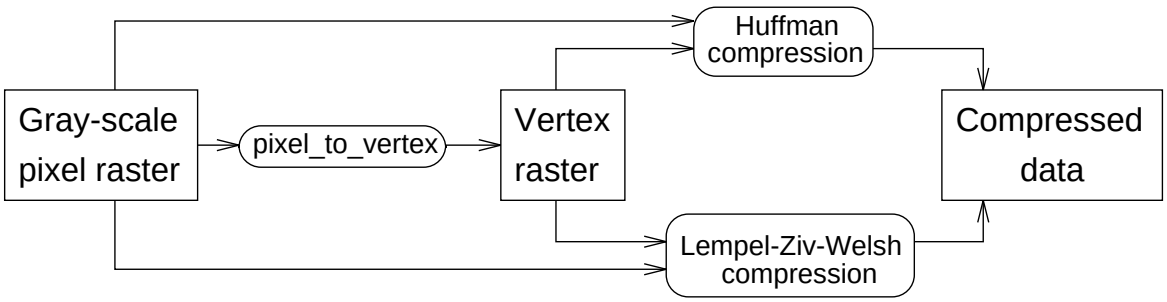

Figure 4: Four schemes for compression of gray scale images.

The major aim of this research is to determine whether a vertex raster representation is more amenable to compression than an equivalent pixel raster. It is known that a string compression scheme usually performs better, that is, yields a bigger compression ratio, if the size of the input alphabet is small. In our case, this means that we expect images with a small number of different colors to compress better. Similarly, vertex rasters with a small number of different vertex weights should also give the best results. Thus, a key factor is, for a given pixel raster with N colors, how many different vertex weights will the equivalent vertex raster have. By analyzing procedure *pixel_to_vertex* we can estimate this number to be $O(N^2)$. This seems hopelessly high; however, if we restrict the N pixel values to be *consecutive* numbers instead of *any* numbers, the worst-case estimate for the number of different vertex weights drops to $4 \cdot N + 1$. To see this,

consider the way the vertex weights are computed in *pixel_to_raster*: a vertex at position (x, y), for x and y greater than 1, has its weight computed as

$$(P[x, y] + P[x - 1, y - 1]) - (P[x - 1, y] + P[x, y - 1]).$$

If a pixel value is an integer number between 1 and N, then $(P[x, y] + P[x - 1, y - 1])$ is a number between 1 and $2 \cdot N$, and so is $(P[x, y] + P[x - 1, y - 1])$. Thus, the result of the subtraction is a number between $-2 \cdot N$ and $+2 \cdot N$, for a total of $4 \cdot N + 1$ possible values.

A simple expedient was used to enforce this requirement on the pixel values: first, the whole pixel raster is examined to determine which and how many pixel values are present; a color table is then constructed where each color (or rather, gray tone) value is mapped to a number between 1 and N; finally, each pixel value in the raster is replaced by its index in the color table. It is clear that the color table must be stored somehow in the compressed file – the additional storage required for that is usually small compared with the size of the raster itself.

It should be added at this point that the construction of the color table can be done in many ways, since there are $N!$ ways of ordering N different values. In our implementation, we used an *ad hoc* approach of placing the gray tones in the color table in order of intensity, that is, number 1 is assigned to the darkest gray and number N is assigned to the lightest.

5 Experimental results

The programs described in the earlier section were used in the compression of ten test images. The scarcity of space does not allow us to show these images in the present document; however, they can be downloaded by anonymous ftp from Internet "site" `kodak.cs.umd.edu`, under directory `images`. The following compression methods were utilized in our investigation:

1. Graphics Interchange Format (*GIF*)
2. Huffman encoding of *PGM* format (*huff*)
3. Lempel-Ziv-Welsh encoding of *PGM* format (*lzw*)
4. Huffman encoding of vertex representation (*vtx_huff*)
5. Lempel-Ziv-Welsh encoding of vertex representation (*vtx_lzw*)

The *GIF* format represents an industry wide image compression standard against which the vertex encoding scheme can be judged. One goal was to see whether the vertex method could outperform *GIF* compression for some or all images we tested. The *GIF* compression is based on the Lempel-Ziv-Welsh (LZW) algorithm, and *GIF* compression

results are therefore very similar to those obtained with *lzw*. Differences in the two can be attributed to different startup configuration of the LZW compressor, details about how often the hash table is cleared, and some non-essential data embedded in the *GIF* file such as a color mapping table. See Table 1 for a comparison of *GIF* and *lzw* encoding for our 10 test files.

Since the compression performances obtained with *lzw* and *GIF* are nearly identical, we chose to compare *vtx_lzw* to *lzw* in our analysis. This comparison removes some small bias that would be present with a comparison of *vtx_lzw* to *GIF*. The LZW algorithm is implemented identically for both the *lzw* and *vtx_lzw* programs.

The *huff* and *vtx_huff* programs provide a second means of comparison between vertex and pixel-based rasters. By using a different string compression method, we can more confidently estimate the entropy reduction effect obtained by using vertices instead of pixels. Table 1 illustrates raw compression data for the five methods described above against our 10 test images.

file name	#pixels	*PGM*	*GIF*	*huff*	*lzw*	*vtx_huff*	*vtx_lzw*
parrot	64000	64015	54222	55886	50996	42588	51268
pepsi	64000	64015	4219	21473	4240	3807	2844
asterix	64000	64056	8127	30346	8407	11594	10916
fashion	64000	64056	28134	39620	25657	47400	43514
monalisa	190080	190095	161537	169143	152828	107727	123899
p2tutd	256000	256015	168953	220080	152734	135148	149837
leaf	263175	263231	243283	215690	229414	194413	236924
spidersmo	344745	344760	351780	305428	324163	216701	257820
spiderman	344745	344801	195975	203962	186948	285196	349623
calvin3	442476	442532	223901	243977	196342	246372	282756

Table 1: Compression results – Absolute file sizes in bytes

Table 2 illustrates a comparison of performance between *huff* and *vtx_huff* showing some parameters describing different aspects of the vertex compression.

The *vh/ph* ratio indicates how well the vertex compression algorithm fared against straight Huffman compression of the *PGM* image. In most images we tested, *vtx_huff* did a better compression job than *huff*.

The *c/vv* ratio indicates information about the transition from color to color through the image. A higher value indicates generally smoother transitions across gray scales, i.e., less abrupt color changes. Based on theory and trending observed from our test runs, it is believed that this ratio is a partial indicator of the efficiency of vertex encoding.

The *v/p* ratio indicates how many pixels worth of vertex values were required to store color data for all pixels. This ratio indicates both the complexity of the picture and to some degree the efficiency of vertex encoding. Unfortunately, it is difficult to separate

file name	vh/ph	col	vv	c/vv	vert	v/p	holes	hs	1–2%	norm
pepsi	0.18	26	46	0.57	4382	0.07	1289	149	0.63	0.78
asterix	0.38	31	108	0.29	9649	0.15	4386	150	0.12	0.20
p2tutd	0.61	180	151	1.19	182675	0.71	40259	26	0.33	0.65
monalisa	0.64	197	154	1.28	170374	0.90	16796	6	0.40	0.74
spidersmo	0.71	190	297	0.64	313540	0.91	26546	11	0.30	0.47
parrot	0.76	152	201	0.76	55181	0.86	5715	20	0.23	0.43
leaf	0.90	168	275	0.61	249309	0.95	11911	15	0.19	0.35
calvin3	1.01	92	214	0.43	297837	0.67	50216	102	0.29	0.41
fashion	1.20	98	163	0.60	59448	0.93	2259	23	0.02	0.18
spiderman	1.40	86	276	0.31	321244	0.93	18706	14	0.10	0.19

Key to terms

vh/ph	ratio of file sizes obtained with *vtx_huff* and with *huff*
col	number of colors in image
vv	number of vertex values used in vertex raster
c/vv	ratio of number of colors to number of vertex values
vert	number of vertices needed in vertex raster
v/p	ratio of number of vertices to number of pixels
holes	number of consecutive sequences of pixels without vertex values
hs	number of different sizes of hole sequences
nullv	number of pixels not holding a vertex value
1–2%	percentage of vertices which hold values -2, -1, 1, 2
norm	addition of normalized c/vv and 1–2% values

Table 2: Compression results sorted by *vtx_huff / huff*

the two constituents to this ratio.

The data concerning *holes* indicates information about how pixels not associated with vertex values were distributed in the image. This data, like the v/p ratio, is tied to both image complexity and efficiency of vertex encoding.

The factor *1–2%* is another indicator of the smoothness of transition between gray scales in the image. It indicates the percent of vertex values which are a transition in value of one or two from the previous vertex. It is correlated to how radically the gray scales change value. As with the c/vv ratio, we believe that this statistic is also a partial indicator of the efficiency of vertex encoding.

The last column, *norm*, represents a statistic comprised of the normalized versions of c/vv and *1–2%* added together. Its intent is to capture both factors related to smoothness of transition across gray scales in one data item.

Table 3 illustrates a comparison of performance between *vtx_lzw* and *lzw* showing some parameters relevant to the vertex compression. Some items common to table 2

have been removed for clarity purposes. *Vtx_lzw* performed a more efficient compression than *lzw* in 4 of our 10 test cases.

file name	vl/pl	c/vv	v/p	1–2%	norm
pepsi	0.67	0.57	0.07	0.63	0.78
spidersmo	0.80	0.64	0.91	0.30	0.47
monalisa	0.81	1.28	0.90	0.40	0.74
p2tutd	0.98	1.19	0.71	0.33	0.65
parrot	1.01	0.76	0.86	0.23	0.43
leaf	1.03	0.61	0.95	0.19	0.35
asterix	1.30	0.29	0.15	0.12	0.20
calvin3	1.44	0.43	0.67	0.29	0.41
fashion	1.70	0.60	0.93	0.02	0.18
spiderman	1.87	0.31	0.93	0.10	0.19

Table 3: Compression results sorted by *vtx_lzw / lzw*

Some variations on the images were performed to see if any notable changes occurred in the efficiency of the vertex encoding. The first variation was to examine whether reducing the number of gray scales in an image would impact the performance of *vtx_huff* and *vtx_lzw*. We chose to quantize the monalisa image into three additional versions. The original version had 197 gray scales. The image was quantized into versions with 32, 4 and 2 gray scales. Comparison data is shown in table 4.

Results sorted by *vtx_huff / huff*

file name	vh/ph	col	vv	c/vv	vert	v/p	holes	hs	1–2%	norm
monalisa.2	0.18	2	4	0.50	7004	0.04	3937	334	1.00	1.13
monalisa.4	0.33	4	5	0.80	27061	0.14	13613	177	1.00	1.21
monalisa.32	0.55	32	33	0.97	122509	0.64	30404	54	0.78	1.04
monalisa	0.64	197	154	1.28	170374	0.90	16796	6	0.40	0.74

Results sorted by *vtx_lzw / lzw*

file name	vl/pl	col	vv	c/vv	vert	v/p	holes	hs	1–2%	norm
monalisa	0.81	197	154	1.28	170374	0.90	16796	6	0.40	0.74
monalisa.32	0.89	32	33	0.97	122509	0.64	30404	54	0.78	1.04
monalisa.2	1.04	2	4	0.50	7004	0.04	3937	334	1.00	1.13
monalisa.4	1.04	4	5	0.80	27061	0.14	13613	177	1.00	1.21

Table 4: Quantized *monalisa* compression results

The *vtx_huff / huff* results appear to indicate weaknesses in the Huffman algorithm when compressing the large amounts of redundant data in the *PGM* format. This ex-

ample shows the vertex raster helping the Huffman algorithm substantially when data is highly redundant.

The *vtx_lzw / lzw* results appear to indicate that the LZW algorithm is strong when converting large amounts of pixels with the same value. Vertex encoding appears to interfere with its ability to compress highly redundant data. However, as the redundancy is reduced (more gray scales are added to the image), benefits of vertex encoding appear to outweigh the LZW efficiency on the remaining data redundancy.

The second variation on images involved smoothing. This experiment resulted from our recognition that more radical deltas in adjacent gray scales in the image seemed to impact vertex encoding performance. It was decided to apply a smoothing algorithm to the 10 images and compare results to the original runs.

Smoothing is a standard image processing operation where the value of each pixel is replaced by the average of its neighbors. In our experiment, we used the original value of the pixel plus its 8 immediate neighbors divided by 9. The net effect of this practice is to reduce the number of different vertex values and skew their distribution so that more vertices have low absolute values. Tables 5 and 6 illustrate comparisons between compression of original images and compression of their smoothed cousins.

file name	vl/pl (original)	vl/pl (smooth)	% improvement
pepsi	0.67	0.70	-04
spidersmo	0.80	0.61	+23
monalisa	0.81	0.52	+36
p2tutd	0.98	0.51	+48
parrot	1.01	0.58	+43
leaf	1.03	0.69	+33
asterix	1.30	0.99	+24
calvin3	1.44	0.72	+50
fashion	1.70	0.71	+58
spiderman	1.87	0.80	+59

Table 5: Comparison of *vtx_lzw / lzw* ratio for original vs smoothed image

Positive values for *% improvement* indicate that the transition to a smoothed image helped the compression of vertex rasters more than that of pixel rasters. The improvement appears substantial. 85% of our test cases showed improvement in vertex based encoding efficiency when the smoothed images were used. Those images showing negative percent improvement were the strongest performers in their respective categories on the initial run. It is likely that these images lost some unique property through the smoothing process which adversely effected vertex based compression performance.

Another item which is apparent from Tables 5 and 6 is that for the smoothed images, *vtx_lzw* outperformed *lzw* and *vtx_huff* outperformed *huff* in every case. Figures 5 and 6

file name	vh/ph(original)	vh/ph (smooth)	% improvement
pepsi	0.18	0.23	-28
asterix	0.38	0.47	-23
p2tutd	0.61	0.39	+36
monalisa	0.64	0.35	+45
spidersmo	0.71	0.56	+21
parrot	0.76	0.54	+29
leaf	0.90	0.54	+40
calvin3	1.01	0.62	+39
fashion	1.20	0.63	+47
spiderman	1.40	0.71	+49

Table 6: Comparison of *vtx_huff / huff* ratio for original vs smoothed image

help in visualizing this result. It appears likely that vertex encoding affects favorably the compression of smoothed versions of images.

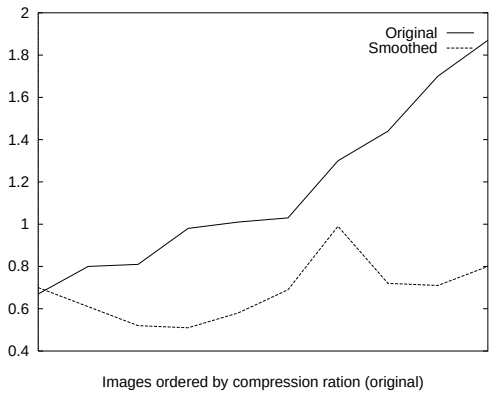

Figure 5: Plot of improvement in *lzw* compression achieved by smoothing

In this section we presented data which compares vertex based compression to straight Huffman encoding of *PGM* data and Lempel-Ziv-Welsh encoding of *PGM* data. The latter is essentially identical to the *GIF* encoding mechanism. Results showed that for some images, vertex encoding provided additional compression not realized by the other methods. However, vertex encoding sometimes resulted in worse performance. Several statistics related to the encoding were surmised to have a positive correlation with vertex compression performance. These statistics generally dealt with how radically gray scale values changed throughout the image. When a smoothing operation was applied to the test images, the performance of the vertex based compression increased dramatically relative to the GIF-like compression.

Figure 6: Plot of improvement in *huff* compression achieved by smoothing

6 Conclusions and suggestions for future research

The approach we adopted for using vertex representations on the image compression problem was a little too simplistic and, arguably, naïve. Our goal, however, was not to build a finished image compression system, but rather to investigate under what circumstances the encoding of a picture as a vertex raster is preferable over its encoding as a pixel raster.

The fact that a vertex representation consists essentially of a differential encoding of a scalar field suggested that the best results would be obtained for images with large regions of the same color. This turned out not to be always the case, as evidenced by the results obtained on images *monalisa.2 monalisa.4*, *calvin3* and *asterix* (we can tell that these pictures have large regions with the same color by the high number of null vertices). The reason for this seems be that the string compression algorithms handle very well long sequences of equal pixel values.

Another hypothesis suggested by the differential nature of vertex encoding is that this scheme would benefit smooth pictures most. This thought appears to be justified by the results obtained on *pepsi*, *monalisa* and all other artificially smoothed images.

This research is still in a preliminary stage. A more consistent investigation will require the extension of our implementations to handle color images. Also of importance is the utilization of more powerful string compression algorithms such as arithmetic coding [6] and the comparison with more recent image compression systems [9].

Although vertex rasters do not offer a generally better initial encoding for images than pixel rasters, they enjoy some properties that can be explored in an image compression system. One alternative that seems to be worth investigating is whether hierarchical coding, such as described in [2], can benefit from vertex representations. Another possible area of application is in the compression of video (motion pictures).

References

[1] N. Ahmed, T. Natarajan, and K. R. Kao. Discrete cosine transform. *IEEE Trans. Computers*, C:2390–2393, January 1974.

[2] P. J. Burt and E. H Adelson. The laplacian pyramid as a compact image code. *IEEE Trans. Comm.*, COMM-31:552–540, April 1983.

[3] C. Esperança and H. Samet. Representing orthogonal multidimensional objects by vertex lists. to appear in (...).

[4] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proc. IRE*, 9(40):1098–101, September 1952.

[5] W. B. Pennebaker and J. L. Mitchell. *JPEG still image data compression.* Van Nostrand Reinhold, 1993.

[6] J Rissanen. Generalized kraft inequality and arithmetic coding. *IBM J. Res. Develop.*, 20:198–203, 1976.

[7] D. Rutovitz. Data structures for operations on digital images. In G. C. Cheng et al., editor, *Pictorial Pattern Recognition*, pages 105–133. Thompson Book Co., Washington DC, 1968.

[8] J. Shechtman. Processamento geométrico de máscaras VLSI. Master's thesis, Eng. Elétrica, Universidade Federal do Rio de Janeiro, Rio de Janeiro, August 1992.

[9] J. A. Storer. *Image and Text Compression.* Kluwer Academic Publishers, 1992.

[10] T. A. Welch. A technique for high-performance data compression. *IEEE Computer*, 17(6):8–19, 1984.