Fast Progressive Transmission of Images Using Wavelets With Sorted Coefficients

by

Allan G. Rempel

B.Sc., The University of Saskatchewan, 1993

B.Comm. (Computational Science), The University of Saskatchewan, 1993

AN ESSAY SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES

(Department of Computer Science)

I accept this essay as conforming
to the required standard

The University of British Columbia

August 1997

Abstract

Wavelets provide a convenient mechanism for encoding image data and transmitting it progressively. This is useful in many applications such as Geographic Information Systems, in which users interactively obtain images from a central server. By sorting the coefficients in decreasing order of magnitude, further gains may be realized in that the important features of the image may be transmitted sooner rather than later. This essay documents software written to do this, and shows the results of these techniques.

Contents

Acknowledgements

First and foremost, I would like to thank my research supervisor, Alain Fournier, without whose guidance this work would not have been completed. I would also like to thank Gerry Furseth of Facet Decision Systems, Inc., who provided a convenient problem and context for my work, and Jack Snoeyink, who brought Facet and I together. And I would be remiss if I did not thank the members of Imager who provided friendship and assistance along the way, NSERC and BC ASI for providing the funding which enabled me to pursue this degree, and the Computer Science Department at UBC for providing the breadth Masters option as well as additional funding.

I would also like to thank my family, especially my mother, Doris Rempel, who is responsible for much of what I am. Thanks to Angela and Bret RempelEwert, who keep me in touch with reality, lest I lose myself in the ivory tower, and to Stephen Rempel, for uncountably many conversations and *gedankenexperimenten* on the nature of the universe, which have fueled the flame that makes hot the water of science in which I swim.

Finally, I would like to thank the residents of Green College at UBC, a bastion of interdisciplinarity (of which my breadth Masters degree is yet another symptom) which provided the cold water of art in which I also swim. I would especially like to thank my best friend and fellow inmate Dawna Tong, who encouraged me to think in new ways and changed my mind on many topics, who taught me much about sociology, law, postmodernism, and French, who gave me a greater appreciation for the plight of the disenfranchised, and who has made my life better by her presence.

Allan G. Rempel

The University of British Columbia
August 1997

To Nick Rempel, who unfortunately could not live to see his son grow up,

and will be forever missed

Chapter 1

Introduction: The Problem of Communication

Since the beginning of humanity, people have been concerned with knowledge. People learned by observing the physical world and performing experiments with it, thus acquiring knowledge, and this knowledge became useful in predicting the outcomes of future actions. But in order for one person's knowledge to become useful to another, such knowledge would have to be communicated in some way.

Communication is the transmission of knowledge or information from one entity to another. When people talk, they encode the knowledge they're trying to convey into a format whereby it can be communicated, such as words. This is necessary, since it is not well known how to directly transmit thoughts from one person to another. Similarly, digital information must be encoded or translated into a format appropriate to the transmission medium to be sent from one entity to another.

The translation process is concerned with two key features: correctness and

efficiency. When communicating knowledge, we want to ensure that what is being communicated is as true or correct as possible, since the communication of falsehoods is not useful in the same way as the communication of truth, and the goal of communication is to share the usefulness of true knowledge. But there are also costs to communication and translation in that they require time, which is a limited resource and which can be used in many other ways. As a result, it is advantageous to communicate knowledge as quickly or efficiently as possible.

Sometimes, though, there is a tradeoff between correctness and efficiency, in that a more efficient encoding of the knowledge can be achieved by altering it somewhat. In many cases, this alteration might not change the effective correctness of the knowledge in a specific context, because that context does not have the resolution to make use of the difference between the true encoding and the more efficient almost true encoding. However, the task is to determine just what the resolution of the context is, and how much truth can be sacrificed for the sake of expediency.[1] There is also the concern that truth in one context may be taken to another context which may have a higher resolution, in which case the truth may cease to be truth, and become erroneous. This problem is particularly endemic to natural languages, in which words, even in the same language, are coloured by the experiences and cultural context of the person uttering them, which may not match those of the receiver.

This problem arises in numerous fields, including many areas of computer science, computer graphics, and scientific computing, where a quick approximate solution is often better than an time-consuming exact one. However, the concern here is how it arises in the field of Geographic Information Systems (GIS). Geographic

[1] For more on truth and resolution, see [7]. For more on this theme as applied to computer science, see [6].

Information Systems are systems that dispense geographic information interactively from a server to a client over some transmission line. In particular, I am interested in the case where a raster image with a resolution of 512 by 512 greyscale pixels whose values range from 0 to 255, is transmitted over a slow communication line such as a 28,800 bps modem. Such an image could be, for instance, a digitized satellite photograph, or an image of terrain with thematic polygons denoting regions of forest, water, etc. These images can take prohibitively long to transmit over such a medium, and so it is desirable to find ways of encoding the image so that it can be transmitted faster, perhaps even at the expense of some detail.

Chapter 2

Previous Work

The most obvious and naive approach to transmitting an image such as the one described in the previous chapter is to just do it. A quick calculation indicates that there are 262,144 bytes to transmit. The modem typically requires approximately 12 bits (including start and stop bits) to transmit a single byte, so it can transmit about 2400 bytes per second. Hence, transmission of this image would take about 109 seconds, nearly 2 minutes, not counting any transmission delays that could occur. This is clearly inadequate for any interactive application. However, to its credit, the image can be displayed as it comes in. This can be useful if the part of the image that the user is most interested in arrives first. Unfortunately, if it arrives last, or if the user is looking for a general overview of the image instead of a small piece of it, the user has a long wait ahead.

However, several encoding schemes exist which can cut down on the time required for transmission. Many of these do so by compressing the image data so that it requires less bytes, usually exploiting some kind of redundancy in the image, and hence can be transmitted faster. Unfortunately, these schemes are dependent on the nature of the image data. It is mathematically impossible, as can be proven

by a simple counting argument[4], for a compression scheme to compress any and every file by some amount without any loss of data. Lossless compression schemes will make some files smaller while making other files larger, in the hopes that the files that need to be compressed are the of the former. Lossy compression schemes, on the other hand, are not restricted by this theoretical limit and can often achieve very high compression while losing detail that, it is hoped, will not be noticed by the human visual system. This is an example of sacrifice of extraneous detail when the message is transmitted to a receiver with less resolution than the sender, as mentioned in the previous chapter.

Run-length encoding (RLE) is a lossless compression scheme that takes advantage of simple redundancy in an image by collecting strings of identical pixels (bytes) and replacing them with pairs of bytes where the first byte is a count of the number of pixels in a string and the second is the value of those pixels. Unfortunately, it tends not to work well for the images that this essay is concerned with because the images seldom have long strings of pixels with the same value. In fact, RLE often *increases* the size of the file to be transmitted.

Many other schemes such as GIF and JPEG also exist and are in common use for compressing images. For more information on these and other encoding schemes, see [5]. However, these are usually not completely suitable for the GIS application, either because the compression ratio is not particularly good, or they require all the data in the image to be downloaded before display can begin, or they lose too much detail.

Another development in coding theory which holds considerable promise for applications such as this is *wavelets*. Wavelets provide a means of decomposing an image into different levels of detail, which can later be combined again to recon-

struct the image. This has numerous applications from querying images at multiple resolutions to editing them. It can also be used for compression, in that a good approximation of an image can be gained with a very small number of wavelet coefficients. But perhaps most importantly for our application, it is ideally suited to progressive transmission.

Progressive transmission simply means displaying the image continuously as it comes in. As such, the term could be applied to the naive transmission scheme at the beginning of this section since the image can be continuously updated line by line as the pixels come in. However, it is perhaps best applied to schemes in which successive better approximations to the whole image are sent, one after the other.

One way to achieve this is to create a set of images which are downsampled versions of the original image. For example, from a 512x512 image, one can create 256x256, 128x128, 64x64, 32x32, 16x16, 8x8, 4x4, 2x2, and 1x1 images. Each image has only 1/4 of the data of the previous image, and taken together, they comprise only 1/3 of the data of the original image. By transmitting the 1x1 image first, then the 2x2, and so on up to 256x256 and eventually 512x512, the user is able to see successive approximations of the image very quickly, and from that data may gain enough information to quickly decide whether to abort the transfer (saving a lot of time) or continue. In applications such as GIS, the user may need to view a number of images before finding the right one, and thus would save a lot of time by utilizing progressive transmission. However, in exchange for this luxury, the price for downloading the image that the user eventually settles on is 4/3 of what it originally was, which, while efficient from an algorithmic analysis point of view in that the time complexity in order notation has not increased, is still not insignificant.

However, the above scheme transmits redundancy in that each of the first

nine images sent before the final one was generated by and is subsumed by the final one. This redundancy can easily be eliminated by using wavelets, as follows: A wavelet decomposition of a flat image file yields an "image pyramid"[3] with a number of levels that is proportional to the logarithm of the number of pixels in the image, just as in the above example. But this time, instead of resending a new version of the image each time, the only data sent is incremental data which can be added to the existing data to create a newer higher resolution image. This is the essence of progressive transmission using wavelets.

Chapter 3

Wavelets

Stollnitz, DeRose, and Salesin[8] provide a good description of Haar wavelets and their use in standard and nonstandard decomposition and reconstruction of images. I will not reproduce that here, except to recall the relevant formulae. Recall that any wavelet is comprised of a set of *scaling functions* and a set of *wavelet functions*. The scaling functions for Haar wavelets are

$$\phi_i^j(x) = \phi(2^j x - i) \qquad\qquad i = 0, \ldots, 2^j - 1$$

where

$$\phi(x) = \begin{cases} 1 & \text{for } 0 \leq x < 1 \\ 0 & \text{otherwise} \end{cases}$$

while the wavelet basis functions are

$$\psi_i^j(x) = \psi(2^j x - i) \qquad\qquad i = 0, \ldots, 2^j - 1$$

where

$$\psi(x) = \begin{cases} 1 & \text{for } 0 \leq x < 1/2 \\ -1 & \text{for } 1/2 \leq x < 1 \\ 0 & \text{otherwise} \end{cases}$$

These can then be *normalized* as desired, by replacing the previous equations with

$$\phi_i^j(x) = \sqrt{2^j}\phi(2^j x - i)$$
$$\psi_i^j(x) = \sqrt{2^j}\psi(2^j x - i)$$

In the case of 2-dimensional images, new scaling and wavelet functions need to be defined based on the above functions, depending on how the decomposition and reconstruction is done. Stollnitz et al define the 2-dimensional scaling function,

$$\phi\phi(x, y) = \phi(x)\phi(y)$$

and the three wavelet functions for the *standard construction*:

$$\phi\psi(x, y) = \phi(x)\psi(y)$$
$$\psi\phi(x, y) = \psi(x)\phi(y)$$
$$\psi\psi(x, y) = \psi(x)\psi(y)$$

and for the *nonstandard construction*:

$$\phi\psi_{kl}^j(x, y) = 2^j\phi\psi(2^j x - k, 2^j y - l)$$
$$\psi\phi_{kl}^j(x, y) = 2^j\psi\phi(2^j x - k, 2^j y - l)$$
$$\psi\psi_{kl}^j(x, y) = 2^j\psi\psi(2^j x - k, 2^j y - l)$$

The nonstandard construction uses the same scaling function as the standard, $\phi\phi_{0,0}^0(x, y) = \phi\phi(x, y)$. Stollnitz et al also provide algorithms which use these functions to decompose and reconstruct 2-dimensional images.

For simplicity of coding, I have chosen to use slightly different decomposition and reconstruction algorithms which correspond to the following wavelet functions:

$$\phi\psi_{kl}^j(x, y) = \phi\psi(2^{j+1} x - k, 2^j y - l)$$
$$\psi\phi_{kl}^j(x, y) = \psi\phi(2^j x - k, 2^j y - l)$$

Again, the scaling function is the same as that used in the standard construction, $\phi\phi_{0,0}^0(x, y) = \phi\phi(x, y)$.

My algorithms are derived from the nonstandard decomposition and reconstruction algorithms given by Stollnitz et al, which, for convenience, are reproduced in Figure 3.1.

There are a number of numerical issues involved in this wavelet decomposition and reconstruction, some of which are dealt with by normalization. Each pass over a set of data produces 2 data sets, one with means and one with offset or *detail* coefficients. The means then become the inputs for the next pass. As successive passes produce successive sets of means, the means approach the one mean of all the pixel values in the original image. As that happens, the offset coefficients approach zero. This can result in loss of data if the number representation scheme (e.g. floating point) does not have the resolution to capture those numbers. The solution to this is to normalize the coefficients, so that they will all still have the same dynamic range that the original image did.

Another issue is that some of the coefficients will be negative numbers. This can be a problem if the wavelet-decomposed image is to be stored as a PNM file (for example) to be viewed. PNM files typically have pixel values ranging from 0 to 255, without negative values. My solution to this problem is to subtract 128 from every pixel, so that the dynamic range becomes -128 to 127, which is the same as that for the C type `signed char`. My variation of the nonstandard decomposition will then yield coefficients all in that same range. I then add 128 to all the coefficients to bring them back up to the original range, suitable for viewing.

There are also other significant differences between my decomposition, given in Figure 3.3, and the nonstandard decomposition. Where the nonstandard decomposition takes care of normalization by dividing means and detail coefficients by $\sqrt{2}$, mine divides by 2. Since I did not prenormalize the input by dividing all the pixels

```
void DecompositionStep( c: array[1..2^j] of reals )
{
    for i = 1 to 2^j/2
        c'[i] = (c[2i-1] + c[2i])/√2
        c'[2^j/2+i] = (c[2i-1] - c[2i])/√2
    c = c'
}

void NonstandardDecomposition( c: array[1..2^j, 1..2^j] of reals )
{
    c = c/2^j  // normalize input coefficients
    g = 2^j
    while g >= 2 do
        for row = 1 to g
            DecompositionStep( c[row, 1..g] )
        for col = 1 to g
            DecompositionStep( c[1..g, col] )
        g = g/2
}

void ReconstructionStep( c: array[1..2^j] of reals )
{
    for i = 1 to 2^j/2
        c'[2i-1] = (c[i] + c[2^j/2+i])/√2
        c'[2i] = (c[i] - c[2^j/2+i])/√2
    c = c'
}

void NonstandardReconstruction( c: array[1..2^j, 1..2^j] of reals )
{
    g = 2
    while g <= 2^j do
        for col = 1 to g
            ReconstructionStep( c[1..g, col] )
        for row = 1 to g
            ReconstructionStep( c[row, 1..g] )
        g = 2g
    c = 2^j c  // undo normalization
}
```

Figure 3.1: Nonstandard decomposition and reconstruction from [8].

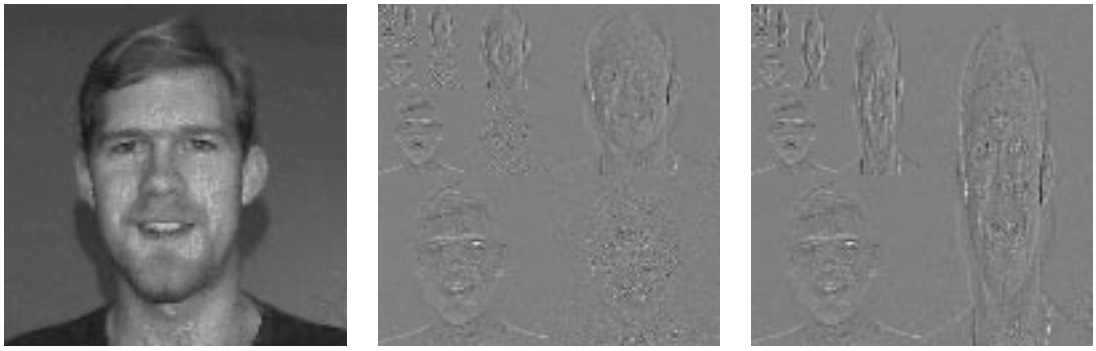

Figure 3.2: Original image, Nonstandard decomposition, Rempel decomposition.

by 2^j, this will guarantee that the resulting numbers stay in the same range as the numbers from which they were derived, which is useful for numerical reasons.

The other change affects the wavelet basis functions. The nonstandard decomposition, in its first loop, runs a pass on all the rows and then all the columns, so that both the means and the detail coefficients resulting from the pass across the rows are processed by the pass across the columns. In my decomposition, only the means from the previous pass are processed on the next pass. In the first loop, my decomposition runs a pass on all the rows just as the nonstandard decomposition did, but it then runs a pass on only the columns that contain the means, leaving the offset coefficients as they are. I believe this approach is conceptually simpler than the nonstandard decomposition, and it also makes the progressive reconstruction slightly simpler. Figure 3.2 shows a sample image, the nonstandard decomposition of it, and my decomposition of it.

Just as the nonstandard reconstruction is the simply reverse of the nonstandard decomposition, so is the Rempel reconstruction the reverse of the Rempel decomposition. The base coefficient (in the upper left corner of the image) is the mean of all the pixels in the original image, and is typically transmitted first. The

```
void DecompositionStep( c: array[ 1..2^j ] of reals )
{
    for i = 1 to 2^j/2
        c'[i] = ( c[2i−1] + c[2i] ) / 2
        c'[2^j/2+i] = c[2i−1] − c[2i]
    c = c'
}

void RempelDecomposition( c: array[ 1..2^j, 1..2^j ] of reals )
{
    c = c − 128
    g = 2^j
    while g >= 2 do
        for row = 1 to g
            DecompositionStep( c[row, 1..g] )
        for col = 1 to g/2
            DecompositionStep( c[1..g, col] )
        g = g/2
    c = c + 128
}

void ReconstructionStep( c: array[ 1..2^j ] of reals )
{
    for i = 1 to 2^j/2
        c'[2i−1] = c[i] + c[2^j/2+i]/2
        c'[2i] = c[i] − c[2^j/2+i]/2
    c = c'
}

void RempelReconstruction( c: array[ 1..2^j, 1..2^j ] of reals )
{
    c = c − 128
    g = 2
    while g <= 2^j do
        for col = 1 to g/2
            ReconstructionStep( c[1..g, col] )
        for row = 1 to g
            ReconstructionStep( c[row, 1..g] )
        g = 2g
    c = c + 128
}
```

Figure 3.3: Modified nonstandard decomposition and reconstruction.

pixel immediately below it divides the image into an upper and a lower half. The next two coefficients (to the right of the first two, completing the square) divide each of the halves in half again, resulting in a total of 4 squares within the square image. The next 4 coefficients divide each of those squares into an upper and a lower half. The next 8 coefficients divide those halves in half again, resulting in a 4x4 grid of square pixels. This process continues until the entire image is reconstructed. By this scheme, each pass requires the transmission of exactly twice as many coefficients as the previous pass (with alternating passes affecting the rows and columns), and the number of individual blocks in the image doubles with every pass as well.

Chapter 4

Progressive Transmission

Using wavelets to encode an image into a base coefficient and a large number of detail coefficients is an interesting way to encode an image, but it suffers from some problems as well. An nxn greyscale image has n^2 pixels, where each pixel ranges over [0..255] and hence takes up 1 byte. A wavelet transform of an image, as done by one of the above methods, also has n^2 numbers, but here each number has fractional value, and can be encoded as a floating point number or a more simple string of bits with a binary point. My implementation in C uses the former, where a `float` type number uses 4 bytes. Thus, the image has increased to four times its original size. It is quite possible to reduce this factor substantially by some clever encoding, but without using another compression technique on the coefficients, the coefficient data will still be larger than the size of the original image, which brings us no closer to our goal of fast image transmission.

However, the power of wavelets is that a substantial part of the original image can be reclaimed with very few coefficients, and it is this feature that we wish to take advantage of. In an interactive application such as a GIS, a user perusing images looking for the right one will not be interested in having each image be

perfectly reconstructed, as much as having a close approximation of the image be transmitted quickly so that a decision can be made quickly as to whether to get the rest of the image or move on to the next one. This power is particularly present in the nonstandard decomposition and my decomposition, as opposed to the standard one, since every pair of passes across the rows and columns yields an approximation of the original image which is 1/4 the size of the image from which it was generated.

The simple and obvious way to do progressive transmission is to send the coefficients in the reverse order of when they were generated. In other words, the last coefficient generated, the base coefficient (which happens to be the mean of all the pixels in the original image) is sent first, followed by the coefficients which turn that into a 2x2 approximation of the original image, then a 4x4, an 8x8, and so on until the whole image is reconstructed. At each stage, 3 times as many coefficients as have been transmitted to that point are required to complete the next stage.

This is similar to the scheme outlined in Chapter 2, where several smaller approximations to the image were sent before the whole image. However, while that scheme sent redundant data, this scheme always builds upon the previously sent data. Even with that, though, it can take longer to transmit a complete image under this scheme since the numbers can be up to 4 bytes each in size. Moreover, while the smaller images quickly generate a good approximation of the whole image, the user may be less interested in an overview of the whole image than in some finely detailed sections, which are not likely to be generated until the final stage. In addition, images with high-frequency elements or many sharp discontinuities may not have the relevant features adequately reconstructed until late in the process.

My solution to this problem, which forms the crux of the contributions of this essay, is to sort the coefficients and transmit them in an order independent of

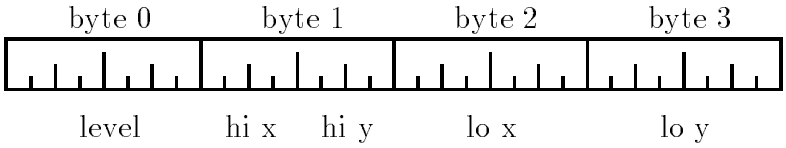

Figure 4.1: 4-byte wavelet coordinate.

when in the wavelet transform they were generated. Specifically, those coefficients that have the highest magnitude represent the greatest deviances (excluding from consideration the size of the area over which they act) between the original image and the approximation. Consequently, it makes sense to send them before other coefficients. This is not a particularly new idea; it is a relatively straightforward extension of wavelet reconstruction and has been alluded to in such papers as [1] and [2]. What is new is that I have implemented it and have results to show how useful it can be.

Unfortunately, this approach requires even more data overhead to be transmitted. The simple top-down reconstruction needs to send only the coefficients themselves because it is always clear at which level and where in the image a coefficient is to be applied. However, if these coefficients are sorted by magnitude, the level and position of the coefficient must be specified for each coefficient. My implementation uses a 4-byte coordinate, broken down as shown in Figure 4.1.

There are 3 values encoded in the coordinate, `level`, `x`, and `y`. `level` gets an entire byte to itself, though it only needs the lower 5 bits, thus leaving (or wasting, depending on one's point of view) 3 bits available for future definition, if necessary. It indicates the level at which the coefficient is to be applied. The number of levels possible in a 2^jx2^j image is $2j$, ranging from 0 to $2j-1$. The reason for $2j$ instead of j is that both horizontal and vertical passes need to be made, which must be differentiated from each other. At level i, a total of 2^i coefficients exist to be

transmitted by my decomposition scheme. The base coefficient is always transmitted first and hence needs no additional information.

`x` and `y` are each 12-bit values, where the lowest 8 bits are stored in bytes 2 and 3 respectively, and the highest 4 bits are stored in the upper and lower halves, respectively, of byte 1. That is,

$$\texttt{x} = \text{byte2} + (\text{byte1}/16) * 256$$
$$\texttt{y} = \text{byte3} + (\text{byte1}\&15) * 256$$

For even levels i, both `x` and `y` range between 0 and $2^{i/2} - 1$. However, for odd levels i, `x` ranges between 0 and $2^{i/2+1} - 1$ while `y` still ranges between 0 and $2^{i/2} - 1$. Note also that, in accordance with the third image of Figure 3.2, `x` refers to the row and `y` to the column.

This wavelet coordinate can be used to represent coefficients in images as large as 4096x4096 pixels, since that is the range of 12-bit numbers. At that size, there will be 24 levels, which fits easily within 5 bits. Larger images will need a different wavelet coordinate encoding.

Chapter 5

Results

The source image that I will be using for this section is a sample image from a GIS application. It consists of a number of thematic polygons which indicate regions of different terrain. It began as a colour 429x305 GIF file but was then resampled to a 512x512 PGM file for the sake of this experiment. It is reproduced in Figure 5.1.

The complete image as a PGM file takes up 262,159 bytes, but encoded as a GIF it takes up only 49,545 bytes. It thus seems reasonable that we should see useful results before we have transmitted 50,000 bytes or so.

As it happened, the regular top-down progressive reconstruction turned up some good results, as can be seen in Figure 5.2. Each coefficient takes up 4 bytes, and so the images there were generated from only 1024, 4096, and 16384 coefficients respectively. With more efficient representations of coefficients, further gains might be realized. These images are quite good and quickly converge to the actual image, making this a reasonable scheme to use in an interactive application such as a GIS.

Of course, the fine lines that run through the image are not very clear until the later stages of the reconstruction, and so for those who are interested in seeing those early in the reconstruction, we have the sorted reconstruction. As can be

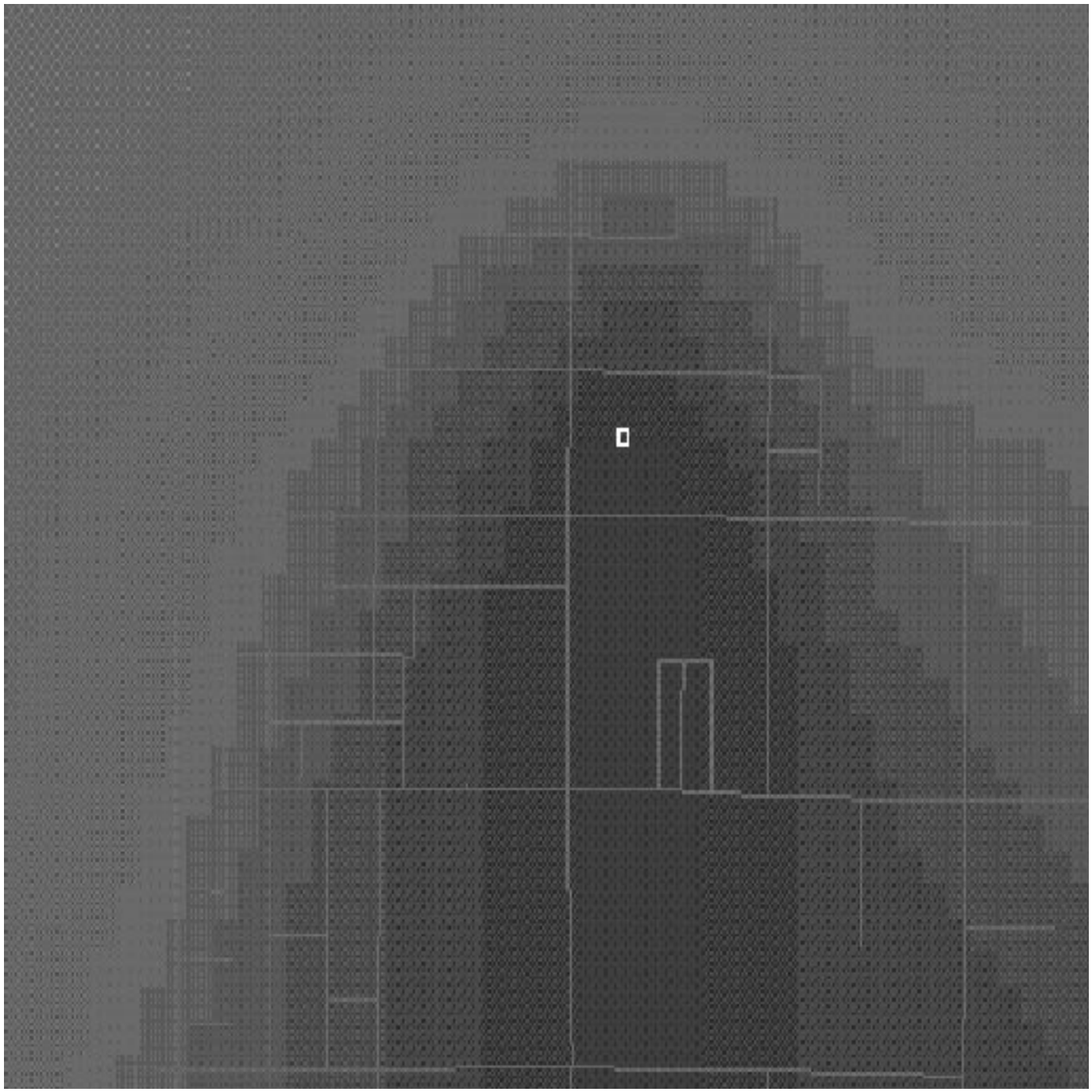

Figure 5.1: GIS image of thematic polygons.

Figure 5.2: Top-down reconstruction after 4096, 16384, 65536 bytes read.

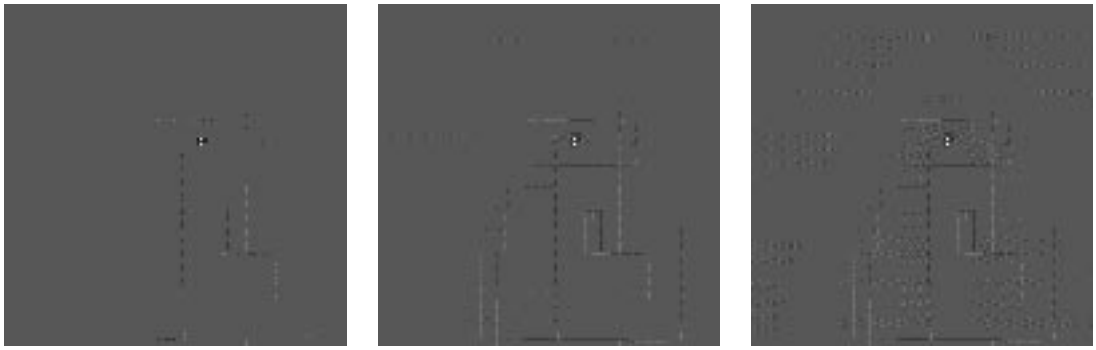

Figure 5.3: Sorted reconstruction after 4096, 16384, 65536 bytes read.

Figure 5.4: Naive progressive transmission after 1365, 5461, 21845 bytes read.

seen in Figure 5.3, those features appear early, although without much background colouring to put them in a good context. The third image starts to bring in some background colour, but that's as good as it gets for this image until half the coefficients are transmitted and the image begins to closely resemble the original, as can be seen in Figure 5.5.

A useful baseline comparison is provided in Figure 5.4 which shows the progress of the naive progressive transmission algorithm described in Chapter 2. In this example, downsampled images have been created and transmitted, one after the other, beginning with a 1x1 image. The first image is 32x32, the second is 64x64, and the third is 128x128. As the pictures indicate, the scheme may be naive, but the images it generates make relatively efficient use of bandwidth, in comparison to the schemes presented here.

Those results are promising, but the original image in that test was somewhat sparse and the range of pixel values quite narrow. So for variety, I ran the same experiment with a different image, which is reproduced in Figure 5.6. This 512x512 image is an original, not upsampled from any smaller or colour photo.

The complete image as a PGM file again takes up 262,159 bytes, but this time, the GIF encoding takes up 214,857 bytes. It thus seems reasonable to allow more bytes to pass to see what the reconstruction routines can do.

Again, the regular top-down progressive reconstruction turned up some good results, as can be seen in Figure 5.7. And again, the sorted progressive reconstruction brought out the detail before the whole image, as seen in Figures 5.8 and 5.9. This time, however, the sorted reconstruction approached the real image a little faster than in the previous example, and each order of magnitude in the number of coefficients transmitted generated an appreciable improvement over the previous

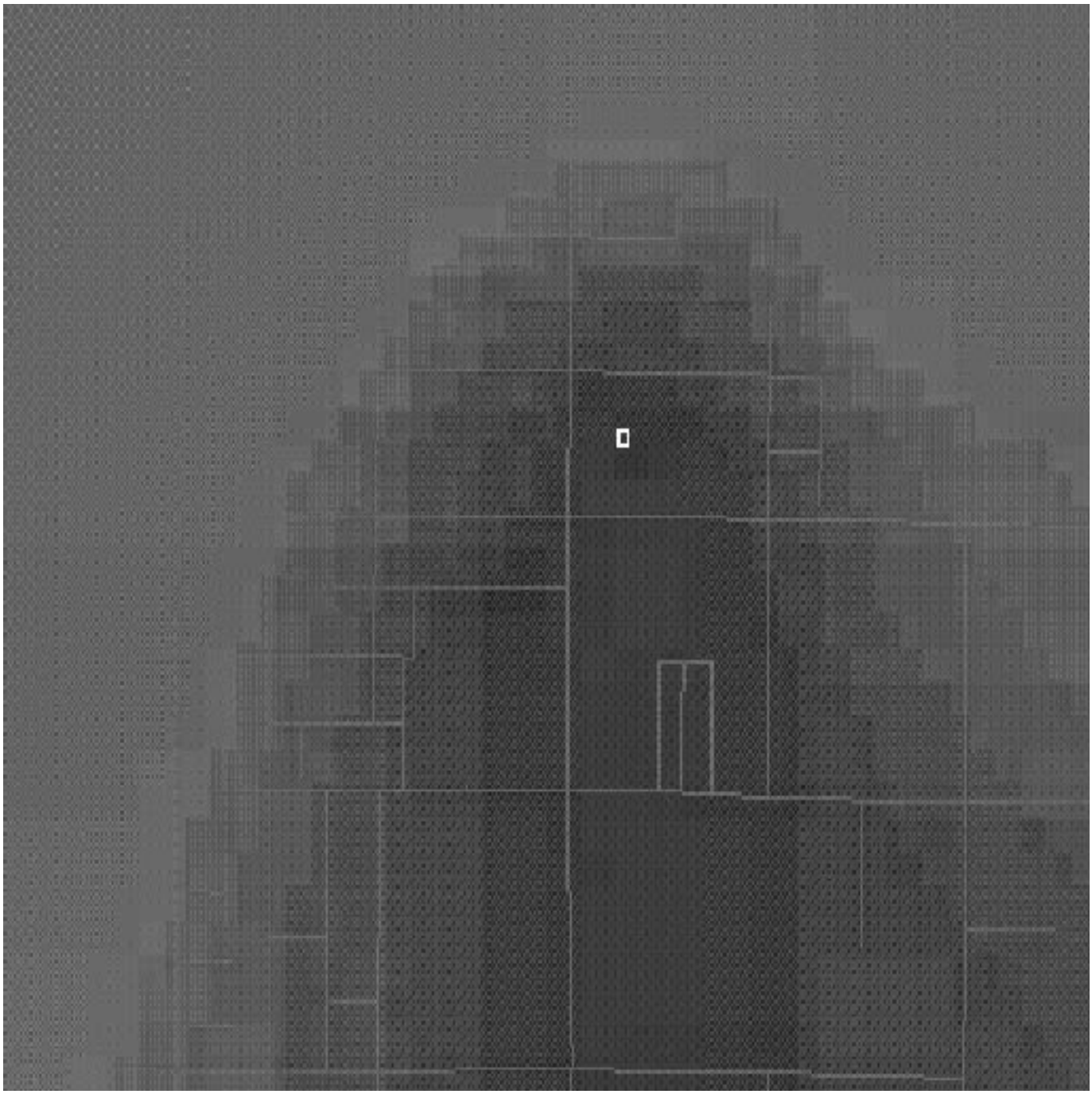

Figure 5.5: Sorted reconstruction after 1/2 the coefficients have been transmitted.

Figure 5.6: A greyscale photograph of a head, with a large dynamic range.

Figure 5.7: Top-down reconstruction after 4096, 16384, 65536 bytes read.

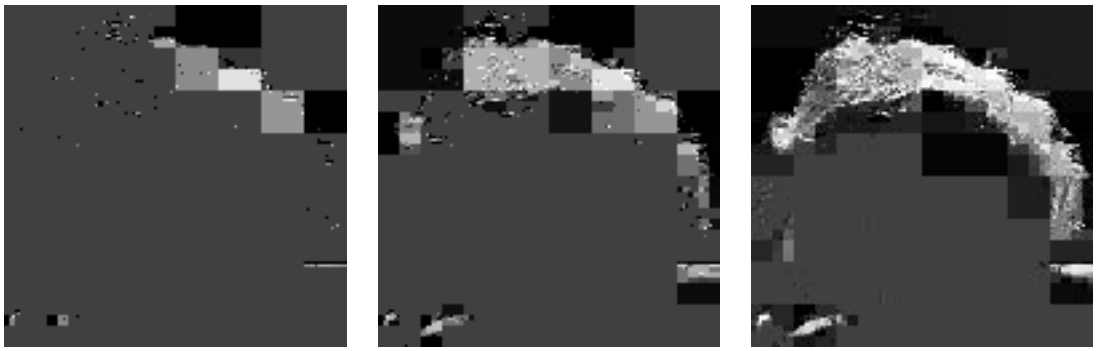

Figure 5.8: Sorted reconstruction after 4096, 16384, 65536 bytes read.

Figure 5.9: Sorted reconstruction after 262,144, 524,288, 1,048,576 bytes read.

image. The blocky artifacts which the wavelet decomposition generates persist noticeably in this image even when half the coefficients have been transmitted, as can be seen in the lower right-hand corner of Figure 5.10. However, at that resolution, the image is nearly indistinguishable from the original.

One comment that can be made on the blocky artifacts in these images is that they are images of a human head, which most humans are experts at recognizing and in which they are sensitive to deviations. Such artifacts appearing in a GIS image are much less likely to be noticeable. Furthermore, the artifacts only detract from an aesthetic appreciation of the image, which is not typically a concern in a GIS application.

Another application for which the sorted reconstruction performs reasonably well, and better for some purposes than top-down, is shown in Figure 5.11. This image started off as a 486x389 colour GIF image, which was then converted to greyscale rather well and without artifacts. Instead of scaling the image to 512x512, though, the scale of the features was preserved by duplicating the rightmost columns and the lower rows to fill out the image to 512x512.

Again one can see the difference between the top-down and sorted reconstructions in Figures 5.12 and 5.13.[1]

One salient feature of this image which we wish to have transmitted quickly is the text near the top of the image. Unfortunately, since the text consists of lines that are only 1 pixel wide, in the top-down case, it will never be adequately reconstructed until the very last level. The second last level, in which 1/4 of the

[1]It should be noted, however, for this and the previous examples, that the thumbnail 128x128 postscript images reproduced in this essay for the sorted reconstruction have artifacts not present in the 512x512 PGM images. These thumbnail images were provided for convenience of rough comparison, and where appropriate, 512x512 images have also been provided for a more accurate comparison.

Figure 5.10: Sorted reconstruction after 1/2 the coefficients have been transmitted.

Figure 5.11: Another GIS image of thematic polygons with text.

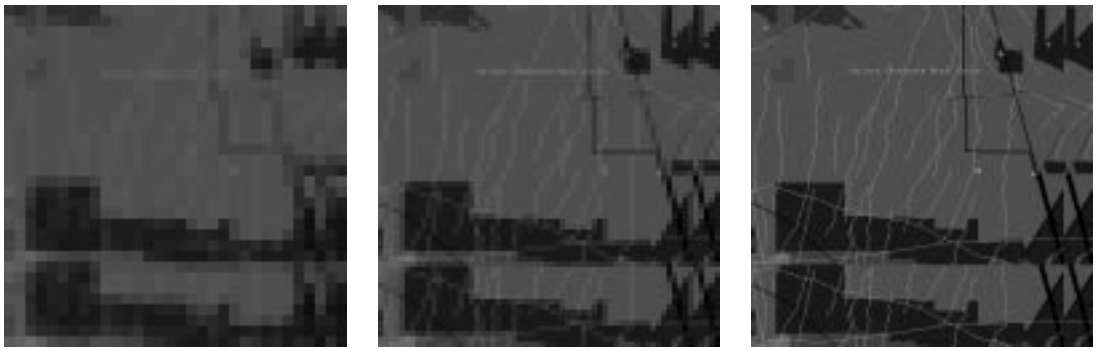

Figure 5.12: Top-down reconstruction after 4096, 16384, 65536 bytes read.

Figure 5.13: Sorted reconstruction after 4096, 16384, 65536 bytes read.

coefficients have been transmitted, is reproduced in Figure 5.14. It is only at this level that the text in the image begins to be readable.

However, when the coefficients are sorted, text that stands out significantly from the surrounding pixels can be transmitted quickly. The image in Figure 5.15 shows the sorted reconstruction after only 1/16 of the coefficients are sent, at which point the text is already readable. Further reconstruction to the point where 1/4 of the coefficients have been transmitted results in an image, in Figure 5.16, that is nearly indistinguishable from the original. The only unfortunate detail of this experiment is that the image is so sparse that the GIF encoding takes only 20703 bytes. Nevertheless, if a progressive transmission scheme is what is needed, the sorted coefficient scheme is an attractive option.

Figure 5.14: Top-down reconstruction with 1/4 of the coefficients.

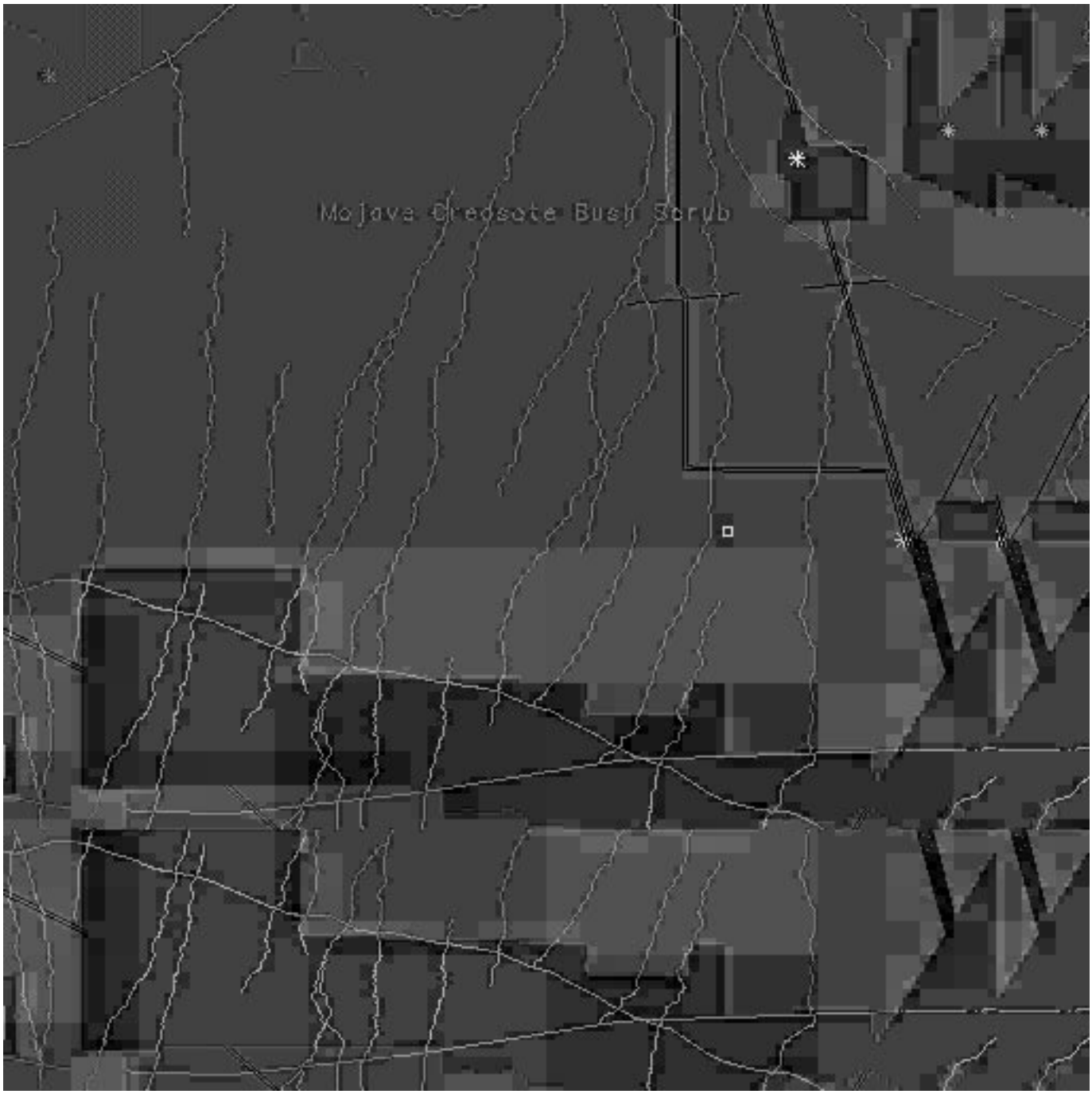

Figure 5.15: Sorted reconstruction with 1/16 of the coefficients.

Figure 5.16: Sorted reconstruction with 1/4 of the coefficients.

Chapter 6

Conclusions and Future Work

For this essay I have implemented routines to do standard and nonstandard wavelet decompositions and reconstructions of greyscale images. I have also implemented a variation on the nonstandard construction which I believe is simpler, both conceptually and in the implementation, and which retains the attractive features of the nonstandard construction. In addition, I have implemented routines to read and write PNM images as well as an OpenGL program to progressively display wavelet-encoded images in the form of floating point numbers which are fed to it, either by the top-down method or by my own sorted coefficient method.

I have used these programs to examine the efficacy of both the top-down and sorted progressive transmission schemes and have found that both are good in their own way. The top-down scheme quickly generates an overview of the image but takes some time to show the details, while the sorted coefficient scheme shows the details quickly but takes even more time to show the whole image. Neither are particularly good for lossless image reconstruction because they both take significantly longer to transmit the entire image than a naive pixel-by-pixel transmission. However, that effect can be mitigated significantly by a more efficient coefficient encoding.

Nevertheless, in both cases, significant parts of the image that a user may be very interested in are transmitted quickly, allowing the user to make a quick decision on whether to download the rest of the image or not. One can envision a system in which, when a user has had enough of an image progressively transmitted and has decided to request the complete image, the image is sent in a compressed non-progressive form such as GIF or JPEG. This gives the user interactivity as well as speed.

Future work could include extending these routines to colour RGB images, which may hold an answer to the problem of transmitting the images with coloured thematic polygons. Also, modifying the routines to handle nonsquare images, in particular those that are twice as wide as they are long (which should be easily handled by my modified nonstandard construction), might make these routines more flexible and useful. Modifying the sorted reconstruction to transmit, for every coefficient, all the coefficients above it in the image pyramid might give the sorted reconstruction the best of both worlds; fast transmission of detail, as well as the context in which that detail sits. In addition, the development of progressive JPEG encoding ([5]) might make for an interesting comparison. And finally, finding a more efficient representation of the coefficients than simple 4-byte floating point numbers could generate further improvements in interactivity by allowing even faster transmission than was presented here.

Bibliography

[1] Deborah F. Berman, Jason T. Bartell, and David H. Salesin. Multiresolution painting and compositing. In Andrew Glassner, editor, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 85–90. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

[2] Andrew Certain, Jovan Popović, Tony DeRose, Tom Duchamp, David Salesin, and Werner Stuetzle. Interactive multiresolution surface viewing. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 91–98. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[3] Alain Fournier. Introduction. In *SIGGRAPH '95 Course Notes, Wavelets and Their Applications in Computer Graphics*, 1995.

[4] Jean loup Gailly. Compression faq. `http://www.faqs.org/faqs/compression-faq/part1/section-7.html` accessed August 20, 1997.

[5] James D. Murray and William vanRyper. *Encyclopedia of Graphics File Formats*. O'Reilly & Associates, Sebastopol, California, 1994.

[6] Allan Rempel. Hacking the truth in linguistic translation: A survey of optimization in language processing. CPSC 511 class essay, University of British Columbia, 1997.

[7] Allan Rempel. Simulation and reality: An essay on ai, prolog, and the chinese room. IBM Writing Competition First Prize Essay, 1992.

[8] Eric Stollnitz, Tony DeRose, and David Salesin. *Wavelets for Computer Graphics: Theory and Applications*. Morgan Kaufmann, San Francisco, California, 1996.