Uniform Frequency Images: Adding Geometry to Images to Produce Space-Efficient Textures

Adam Hunter Jonathan D. Cohen
The Johns Hopkins University
{fezzik, cohen}@cs.jhu.edu
http://www.cs.jhu.edu/~cohen/Research/UFI

Abstract:
We discuss the concept of uniform frequency images, which exhibit uniform local frequency properties. Such images make optimal use of space when sampled close to their Nyquist limit. A warping function may be applied to an arbitrary image to redistribute its local frequency content, reducing its highest frequencies and increasing its lowest frequencies in order to approach this uniform frequency ideal. The warped image may then be downsampled according to its new, reduced Nyquist limit, thereby reducing its storage requirements. To reconstruct the original image, the inverse warp is applied.

We present a general, top-down algorithm to automatically generate a piecewise-linear warping function with this frequency balancing property for a given input image. The image size is reduced by applying the warp and then downsampling. We store this warped, downsampled image plus a small number of polygons with texture coordinates to describe the inverse warp. The original image is later reconstructed by rendering the associated polygons with the warped image applied as a texture map, a process which is easily accelerated by current graphics hardware. As compared to previous image compression techniques, we generate a similar graceful space-quality tradeoff with the advantage of being able to "uncompress" images during rendering. We report results for several images with sizes ranging from 15,000 to 300,000 pixels, achieving reduction rates of 70-90% with improved quality over downsampling alone.

CR Categories and Subject Descriptors: I.4.2. **[Image Processing and Computer Vision]:** Compression (Coding) — Approximate methods, I.3.3 **[Computer Graphics]:** Picture/Image Generation — Display algorithms.
Additional Key Words and Phrases: texture mapping, Fourier analysis, sampling, parameterization, visualization.

1 Introduction

Images play an integral role in today's 3D computer graphics pipeline. They appear not only as the output of the rendering process, but as input to the process as well. In the form of texture maps, images allow high-frequency variation in surface color over either a single polygon or a mesh of polygons – more complex than is practical with geometry alone.

We can think of an image as a uniform sampling of an underlying continuous image function. The simple image representation has the advantage of being easily stored and accessed in either general-purpose memory or special-purpose texture memory on the graphics hardware. Because it is uniform, however, it has the potential disadvantage of using space inefficiently. Portions comprising low spatial frequency content require as much space as those with much higher spatial frequencies. If we apply standard compression algorithms, such as JPEG, to such images to reduce wasted space, we may lose the nice properties of the simple 2D representation.

This wasted space may not matter to applications which rely primarily on small, repeated textures to enhance the appearance of a simple environment. However, the production of realistic environments often requires the use of large textures which are uniquely applied to individual surfaces. For example, realistic urban simulations and terrain visualizations [8] may involve real photographic imagery applied to the geometry of outdoor environments. Indoor environments may be mapped with illumination data resulting from pre-computed radiosity simulations [2]. Arbitrary shading parameters may be stored as images and mapped to surfaces for the computation of arbitrary BRDFs [7]. In fact, by storing arbitrary per-vertex data as images, we can apply simplification algorithms to convert mesh complexity into textures[5]. All of these applications require significant storage and bandwidth resources to manage the associated image data. To reduce this demand, we need an image representation which is compatible with 2D texture-mapping hardware but permits better use of storage without unacceptable data loss.

1.1 Main Contribution

We present a new conceptual approach to efficient image representation: the *uniform frequency image*. By representing an image uniformly in the frequency domain, UFIs enable graceful, uniform resizing. Although the ideal UFI is not fully realizable, we present an algorithm which targets it, automatically reducing the maximum frequency content of local areas of an image to allow downsampling of the image with minimal data loss.

Given an input image, we automatically generate an invertible warping function which downshifts the image's highest spatial frequencies in exchange for upshifting some of its lowest spatial frequencies, producing a concentration of mid-range frequencies. After warping the input image to produce a UFI, we take advantage of the reduction in high-frequency content by downsampling it. We store the inverse of the warping function with the downsampled, warped image for later reconstruction.

We choose a piecewise-linear warping function because it is both efficient to represent and fast and intuitive to invert. In fact, we can think of our scheme as an "image compression" algorithm for which the decompression algorithm is simply the rendering of texture-mapped polygons. Our "compressed" image is just the warped, downsampled image plus a small set of polygons with texture coordinates.

Our approach has the following desirable properties:

1. **Fast decompression using graphics hardware:** The original image is reconstructed by rendering texture-mapped polygons, a feature available on most current graphics hardware.
2. **Transparency to application:** UFIs are rendered as easily as rendering a normal textured quadrilateral. The procedure for unwarping the textures is simply the rendering of certain specific geometry, which can be treated as a "black box" operation.
3. **Gracefully degradation under downsampling:** As UFIs are squeezed into smaller and smaller portions of texture memory, their loss of content initially occurs in a slow, graceful fashion. The tradeoff is that past a certain very compressed point, the degradation begins to accelerate.
4. **Preservation of high-frequency detail:** As opposed to most image compression techniques, UFIs value and maintain high-frequency detail. Portions of an image which make it identifiable are retained longer under increased downsampling, reducing the overall blurred or pixelated effect.

Another interesting aspect of this research is that it explores the symbiosis between irregular representations (e.g. a triangle mesh) and regular representations (e.g. a 2D image). Whereas typical texturing applications augment an irregular representation with a regular one to enhance its appearance or reduce its complexity, our algorithm augments a regular representation with an irregular one, preserving scarce texture memory and bandwidth through the use of a small amount of geometry.

The rest of the paper is organized as follows: First we introduce the concept of uniform frequency images. We then discuss our high-level algorithm and its components. Next we describe our implementation of these components. Finally, we present results and discuss future work.

2 Previous Work

Many of the standard compression algorithms, such as JPEG compression [17] are not directly applicable to the problem of compressing textures. Such algorithms often require significant processing to decompress, thus requiring additional hardware in the graphics pipeline. Also, such compression schemes generally do not allow random access to the pixel data. However, the Talisman architecture [16] proposes a blockwise JPEG variant called TREC and includes hardware support for both compression (to be used on intermediate, retained images) and decompression of these small image blocks.

JPEG compression has also been applied to the compression of synthetic images [10] by combining the rendering of low quality triangles (to preserve sharp features) with JPEG-compressed refinement images to achieve high-quality images on low end graphics platforms.

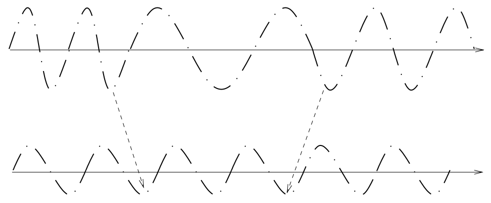


Figure 1: *Creating UFIs. Note the irregularity of the top function makes uniform sampling wasteful in the middle region. By shifting the peaks labeled by arrows, we spread out the frequencies and create a nice signal which is easily uniformly sampled.*

Vector quantization, such as color cell compression [4], is a compression scheme which allows fast, random access to image data. It has been applied in software for both texture maps [3] and light fields [11]. It has also been proposed for hardware texture decompression [9].

The most straightforward ways to reduce texture complexity on today's graphics hardware are to reduce either the color depth or the spatial resolution of the texture images. Spatial resolution must, in fact, be reduced as part of the minification process, whereby several samples from the texture are mapped to a single pixel in the rendered image. Such minification is typically performed using a mip-map image hierarchy [18].

Traditionally, entire levels of the mip-map were either stored in dedicated texture memory or communicated to the graphics hardware as needed. More recently, it has become possible to cache subsets of the mip-map hierarchy in texture memory [14] or even to fetch texture samples as needed from general application memory [1]. It is also possible to willfully trade texture quality for performance at the application level using the OpenGL texture level of detail commands [19] to modify the selection of mip-map levels.

Although we have not thoroughly explored all the ramifications, we believe our compression scheme, which downsamples the original texture after reducing the spatial frequency of the data, should cooperate well with such minification algorithms.

A similar approach was proposed by Sloan [13], however the focus of this work was on optimizing textures for use on existing meshes. The context was that of painting applications, in which the user could even guide the process by specifying a custom importance function along with the painted texture. Although he did briefly discuss the creation of "atlas" meshes, his approach used simple uniform spatial divisions and precluded the use of mip-mapping. Terzopoulos and Vasilescu, in their 1991 paper on adaptive meshes [15] discuss the use of adaptive meshes for reconstructive work, but their approach is very general and does not use the Fourier transform. In addition, by using meshes of constant topology they do not adequately solve the problem of shear distortion.

3 Uniform Frequency Images

We have discussed in section 1 how the standard 2D representation of an image represents a uniform sampling of an underlying image function. Before imposing a uniform sampling on an image, however, one could consider making the underlying *image function* uniform by elongating the high frequencies and compressing the low frequencies (see figure 1). This makes uniform sampling much more appropriate by lowering the Nyquist limit of the function, at the expense of increasing image density in areas of low frequency concentration. In one dimension, at least, it is not a stretch to assert that this could be done perfectly, given a transform of sufficient complexity. We call the resulting image a *uniform frequency image*.

Such an image has the characteristic that, for any two subregions R and S, their Fourier representations, $\hat{R}$ and $\hat{S}$, respectively, are equivalent to within some tolerance. We will consider later how to produce such images, but for now let us examine some of their characteristics. It is apparent that this characteristic of uniformity has nothing to do with spatial similarity or any statement about exact frequency characteristics. Indeed, both a completely solid-colored image and an image with totally random data qualify as uniform frequency images. Instead, this property examines the local *change in frequency*, or first derivative of the Fourier transform as it travels across the image. This first derivative describes the local change in *density of frequency information*.

In the following sections, we describe the desirable properties of uniform samplings and uniform frequency images, the effect of resizing on such images, the properties of warping functions used to create these images, and the relationship of our image reductions to traditional compression schemes.

3.1 Uniform Sampling

A uniform sampling of an image has many advantages, most notably that it can be stored natively in texture memory. Unfortunately, such sampling does not cope well with images that have large frequency variations. To adequately sample the image function uniformly, a sampling rate above the Nyquist limit of the function is necessary. Because images are rarely uniform in frequency density, such a sampling rate will almost certainly be wasteful in many areas. What is really desired is an adaptive sampling, which is the basis of some image compression techniques. One obvious example is run-length encoding which in the extreme samples exactly one time in an area of zero frequency. More complicated variations on this theme, involving entropy coding, give rise to PNG or GIF compressed image formats.

We will go in the opposite direction, however. Instead of forcing our image sampling method to be adaptive, we will make a uniform sampling *be* the correct adaptive choice, by making our image have a uniform Nyquist limit. Once done, our images respond as well to a uniform sampling as normal images do to an adaptive one. We get all the benefits of an adaptive image-sampling algorithm, while still leveraging our hardware for doing uniform sampling.

3.2 Resizing

Now we turn to the problem of compressing uniformly sampled images. Conventional texturing, which motivated our decision to sample uniformly, now also motivates our method of compression. Although compressing an image is normally thought of as an arbitrary transformation taking one set of data to another, in this context we will consider compressing an image solely as resizing the image to take up less space. Whichever of the many possible resizing methods we use, such a downsampling acts as a *low-pass filter*; it operates globally in the spatial domain with a bias towards low-frequency data. When we cannot give preference to one area of an image over another, high-frequency decimation is then spread equally to high and low-content areas. Such decimation will naturally will have proportionally

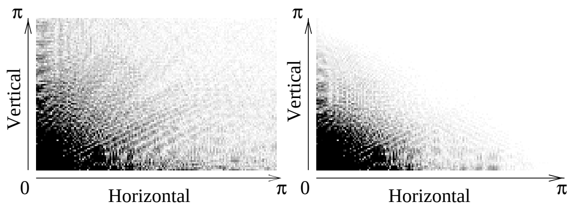


Figure 2: *Frequency Truncation – downsizing an image by a factor of 2 truncates data which is above the Nyquist limit of the smaller image.*

greater impact on high-frequency areas, creating uneven distortion across the image – unless, that is, the image does not *have* proportionally higher-frequency areas, which is the case with a uniform frequency image.

In addition, if the resizing operation involves (as is commonly the case) some sort of Gaussian blend operation, high-frequency data is even more strongly impacted. The effect is one of *frequency truncation* (see figure 2). The image resizing method we consider is a standard Gaussian filter which decimates high frequency data by blending nearby pixel values. This decimation is linear in the degree to which the image is resized (i.e. greater downsampling implies more high-frequency data is erased).

3.3 Warping Functions

An issue we have not discussed so far is the creation of these uniform frequency images. As we saw in figure 1, the key is pushing and pulling on the image using our resizing operation to produce areas of uniform frequency. Conceptually, such pushing and pulling can be thought of as a *warping function* $W : \mathcal{R}^2 \to \mathcal{R}^2$ which takes an image and produces a uniform frequency image. This, along with a resizing operator $R : \mathcal{R}^2 \to \mathcal{R}^2$, will form the basis of our algorithm. Given an input image, I, its corresponding UFI is $W(I)$ and its downsampled UFI is $R(W(I))$. Our reconstruction process recovers I by computing $W^{-1}(R^{-1}(R(W(I))))$.

Let us consider some desirable properties of such operators:

- W is *approximately invertible*, i.e. $\left|I - W^{-1}(W(I))\right| < W_\epsilon$, the *warping error*, which may be dependent on the size of I. W itself should work on domains of any size, and is generally defined as $W : [0,1)^2 \to [0,1)^2$.
- R should also be approximately invertible with an error R_ϵ corresponding to the input and output size. R should also have the property that it operates equally on all subimages of I
- The two transforms should be composable as follows: $W^{-1}(R^{-1}(R(W(I)))) \approx W^{-1}(W(I)) \approx I$, and $W^{-1}(R(W(I))) \approx R(I)$. These approximations have definable errors related to resizing and warping.
- The warp W should *facilitate resizing*, defined as $\left|I - W^{-1}(R^{-1}(R(W(I))))\right|_M < \left|I - R^{-1}(R(I))\right|_M$ for some metric M. This metric is achieved by making W a *flattening warp*, for which $W(I)$ is a more uniform image than I itself.

For practical reasons we also desire two additional properties:

- W should be easy to represent, so as not to add overly to storage and communication overhead.
- W^{-1} should be easy to compute by the system performing our reconstruction computation.

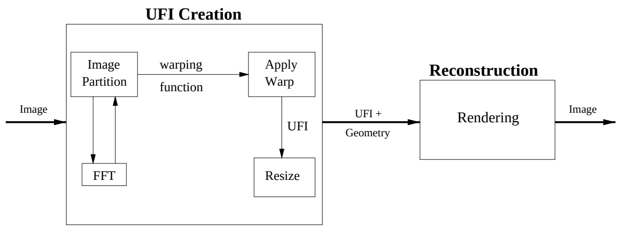


Figure 3: *Our algorithm as a flow diagram. The UFI creation occurs off-line; the only thing communicated to the rendering pipeline is one image (the UFI) and geometry.*

3.4 Traditional Image Compression

Our reduction scheme operates in quite a different fashion from traditional frequency-based compression techniques. Such techniques attempt to *coalesce* data in the frequency domain into tightly packed frequency groupings. Thus the compressed results do not constitute an image in the spatial domain and require significant processing to convert back to a spatial image. In addition, because of the low-pass filter property of texture resizing discussed earlier, coalescing data is the wrong approach for two reasons:

- Preserving low-frequency data is not a high priority. This data will be well-preserved by the resizing operation.
- Since the filter is the same everywhere in the spatial domain, we want our data to be similar in nature across the spatial domain, not to capture and distill the variability of the data.

In contrast to these approaches, our approach uses information from the frequency domain to construct an appropriate warp in the spatial domain which retains continuity across the image, and allows for efficient traditional 2d image representation. Thus our compressed result may still be represented as an image in the spatial domain and functions as such in the graphics pipeline.

4 Algorithm Overview

As depicted in figure 3, our uniform frequency image scheme has two phases: UFI creation, and reconstruction of the original image. We can think of the former as an off-line pre-process, while the latter can be performed quite rapidly, possibly as part of an interactive graphics application.

Our high-level algorithm for generating a warping function takes a top-down, divide-and-conquer approach. Initially, the entire image is our region of interest. We first partition the region into two or more sub-regions. Next we measure the frequency content of each sub-region using a fast Fourier transform. Based on the relative frequency content of these sub-regions, we compute a local warp (see figure 4) that equalizes their frequency contents as much as possible. Finally, we recurse into each of the sub-regions, repeating our procedure until we reach a termination condition. Typically we terminate when the frequency content of a region is too low (or too uniform) or when a region or its warped counterpart become too small.

This algorithm has the nice property that it naturally operates at multiple scales. Our initial warp operation takes place at the largest scale and has the greatest potential to dramatically affect the frequency content of the entire image. Each successive warp operates at a smaller and smaller scale, refining the frequency content of increasingly local regions.

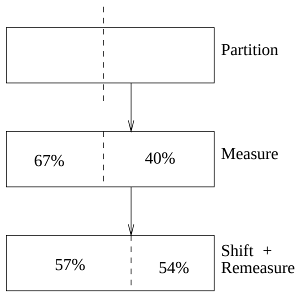


Figure 4: *Frequency Balancing. Evaluation of frequency data leads to an attempt to shift frequencies and even out content. If the process were perfect, both sides would end up equal. The maximum frequency content area (in this case the left with 57%) is related to the image's resizability. Of course the regions are two-dimensional, but only dimension is considered at a time.*

After generating the warping function, we apply it to the source image to generate a UFI. This UFI is then typically resized to reduce space consumption. The inverse warping function is represented as geometry and this geometry, along with the UFI, is stored as our augmented texture. At runtime, image reconstruction occurs by the rendering of textured polygons (see figure 5).

From this general algorithm we can identify the components needed to specify a complete implementation. We first need a data structure to represent the warp. The recursive nature of the algorithm makes a hierarchical representation natural. We require the structure to partition the space into regions which are decomposable into simple geometry.

We also need some concrete way of measuring the frequency content of an image region. Our measure should easily identify areas of high and low content. It should also be spatially additive: the sum of content of two areas should be reasonably related to the density of the areas together. In addition, it should linearly depend on the size of the area. If we have these conditions, then computing the destination shift is easy: Let f_l be the frequency content on the left side of the divide ,f_r the right and s in $[0, 1)$ be the source point. Then d, the destination point, is

$$d = \frac{2sf_l}{f_l + f_r} \tag{1}$$

to make the two sides equal (assuming an inverse linear relationship between the size and the frequency content).

In the next section, we will fill in the implementation details of these primary components of our general algorithm.

5 Implementation

Before proceeding with the details of our implementation, let us pose the problem in a concrete fashion. Our input data is simply an RGB image of arbitrary dimension. The output of our UFI creation process is another image along with a data structure. The image we generate is not meaningful in and of itself, having been warped, but together with the data structure it represents the image in a (potentially) size-independent fashion (although it is optimized for the particular size of image we began with). This data is stored for later use and can be conceptually regarded together with the image to form an *augmented texture*.

If our application supports it, we then communicate this augmented texture to a hardware board using, for example, standard

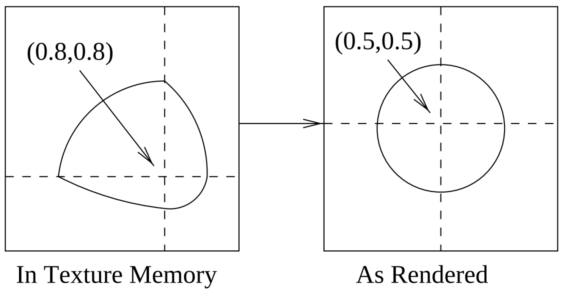


Figure 5: *Hardware unwarping through manipulation of texture coordinates. The k-d tree contains source coordinates, which are used as geometry, and destination coordinates, which are used as texture coordinates. This "unwarps" the pre-warped UFI.*

OpenGL texture-mapping calls. The result is the unwarped version of the texture, generated on-the-fly by the hardware. This unwarped version never actually resides in texture memory; only the warped version, which is displayed on triangles as though it were the original image, does (see figure 5).

We now present the details of our space-partitioning data structure, frequency content computation, and maintenance of pixel-level accuracy through the reconstruction process.

5.1 Space Partitioning Using a K-d Tree

The space-partitioning data structure represents our warping function and provides a simple decomposition of the image into a set of standard, easily renderable geometry. It must also support the use of the Discrete Fourier Transform, or DFT, to generate data about the importance of portions of the image, and it must allow for easy, localized partitioning and warping of the image. Based on these requirements, a *k-d hierarchical decomposition*, or k-d tree emerged as the natural choice. The k-d tree works well for several reasons:

- Its rectilinear partitions easily support the DFT.
- Its binary partitioning of the image at every step simplifies decisions about warping (as seen in figure 4)
- It is easily represented and translated into a small number of quadrilaterals.

Each node of the tree stores pointers to its children along with an xy-location of both the source (unwarped) division point and the destination (warped) division point. Storage requirements for these nodes are inconsequential (less than six percent of the base image size for a standard recursion depth of 5-6). Source and destination locations are measured in the $[0, 1)$ space to keep the tree independent of the size of the image. This allows us to use the tree at any size, and also allows us to use the coordinates of the tree directly as OpenGL texture coordinates. A *boundary* (a buffer zone typically in the .1-.15 range) is used to ensure that the effect of warping at any given level is not too dramatic. This, along with a stopping criterion based on destination quad size, keeps the quads from degenerating. Another stopping criterion terminates subdivision when the total information content dips below a certain extremely small value (to avoid pointless work).

Note the adaptive behavior of the tree, which highlights edges and other areas of high frequency. By terminating the subdivision when quads in the destination image reach a certain minimum size (typically 8-12 pixels), we end up only subdividing those areas where enough space has been allocated at higher levels to make subdivision a worthwhile exercise. These are exactly the areas which have high information content, thus focusing effort where it generates the most benefit.

There are several limitations to this approach. First, irreversible decisions are made at the higher levels which affect the amount of space available at lower levels. Several heuristics attempt to minimize the chance of making poor decisions at the top levels, but because the Fourier transform is inherently not additive, this may result in making poor decisions.

Second, the source point in our current implementation is heuristically chosen to be the midpoint of the current image region. However, other approaches have been tried, and some will discussed later. The advantage of choosing the source in the middle is that it is easier to make appropriate decisions and measure results given just one degree of freedom (the position of the destination point).

Finally, the k-d tree does not guarantee zero-order continuity of the transform. The partitioning created by a k-d tree has *T-junctions*, which translate to visible "cracks" in the output image. We have attempted to mitigate this by rendering every slice overlapping its neighbors with alpha-blending around the edges, however this does not totally eliminate the problem. This seems to be a fundamental problem with any rectangle-based partitioning scheme of sufficient flexibility and will need to be addressed by future work to increase the usefulness of the overall algorithm.

After creating this tree, "warping" becomes simply rendering every leaf of the tree. Resizing the image is handled automatically by the texture-mapping hardware as well. The advantage of this approach is that no new abilities are required on the part of the hardware – there is only one pass and no texture data is read out of the buffer, permitting fully-pipelined performance.

5.2 Computing Frequency Content

To describe the frequency content of a particular region of the image as a single scalar, we choose a representative frequency to describe the region. We begin by performing a DFT using a fast, stable cross-platform implementation known as the FFTW [6]. This DFT contains data in two dimensions as the amplitudes of the frequency components - some samples are shown in figure 2. These amplitudes are imaginary, but we remap them to the real domain by taking their absolute values ($|a + bi| = \sqrt{a^2 + b^2}$), a standard practice when computing power transforms.

Now we require an information content metric. Desired properties of this metric are

- **Information Capture** - the metric should weight higher frequencies more heavily.
- **Comparability** - two measurements should be easily and, preferably, linearly comparable.
- **Truncation Sensitivity** - the metric should report information which somehow captures the response of an area under the frequency-space truncation operations seen earlier.

Rather than using the commonly-used power integral weighted by amplitudes and wavelengths (a valid measure of content, but one not linked directly to the decimation point), we compute a *percentile distribution measurement*, δ. This δ is defined as the point (in a one-dimensional frequency scale) where the sum of amplitudes of frequencies less than δ makes up a certain percentage of the total frequency amplitude. This corresponds to our notion of information content as the maximum frequency because if we set this $\delta = 1$, we get the maximum non-zero frequency, or point above which frequency decimation causes no data loss.

Practically, by setting $1 - \delta$ to some small value we get a decimation point which involves a data loss of only δ, where "data loss" is defined as the sum of available amplitude. In practice, this δ is only representative of the data loss. It is not an absolute measure, due to the inaccuracies of resizing in a discrete domain, and the natural frequency-decimation behavior of most resizing methods. Typical values of δ, which can be seen as the degree of information capture, were 85-90%.

5.3 Pixel-Level Accuracy

A decision we made earlier about our warping function now comes back to haunt us. The tree contains floating-point values in the range $[0, 1)$, but our image is a collection of discrete points. We obviously do not want to lay edges of our tree inside of pixels, but rather along pixel boundaries. Assigning pixel boundary values to floating-point numbers on-the-fly, however, poses some problems. Consider a simple image of size 16 by 16 and a division point which divides the image vertically at $x = \frac{1}{3}$. Where do we put this division point? Wherever we chose to put it, we must make a common decision at all scale levels. We cannot simply take the "ceil" or "floor" of these values, because we will create a situation where an image, of size 4 at a higher level, becomes an image of size 1 when reduced by 2. This causes many pixel-level accuracy problems, which produce unsightly "cracks" in the output. We solve this problem by storing offline, along with the tree, pixel-accurate integer boundary values for the *last* warping or resizing operation which involved that tree. Every node contains pixel-accurate values for the size of the region represented by that node and its children. In this way we make sure that, whatever decisions are made about placing pixel boundaries at step i of the algorithm, those decisions are available to fuel step $i + 1$. Of course, we may not have this level of control (depending on our rendering pipeline) over the eventual unwarping operation, but our texture pipeline, which can handle floating-point texture coordinates, will handle those decisions for us, as long as we have made consistent decisions up until that point. This increases the size of the tree somewhat, but we do not need to communicate this extra size to the graphics board.

6 Results

We tested our software on suite of 12 images with a range of qualitative characteristics (see figure 6). These images ranged in size from 262x52 to 640x480. We ran the images through the mesh generation algorithm with a stopping criterion of 8 pixels region width and $\delta = .85$.

Figure 7 shows the quadtrees for the "google" and "quake" images in both the source (unwarped) and destination (warped) domains. It is easy to see that high frequency regions such as the text in the google image and specular highlights and other edges of the quake image are expanded at the expense of solid or relatively low frequency regions.

In figure 8 we see the impact of the warp on the frequency content of the google and quake images. Each block in the pre- and post-warp images represents a leaf node of the k-d tree, with the intensity representing the frequency content (as defined by our percentage metric). Notice the spreading effect of the warp – black and white areas compress or expand to approach gray.

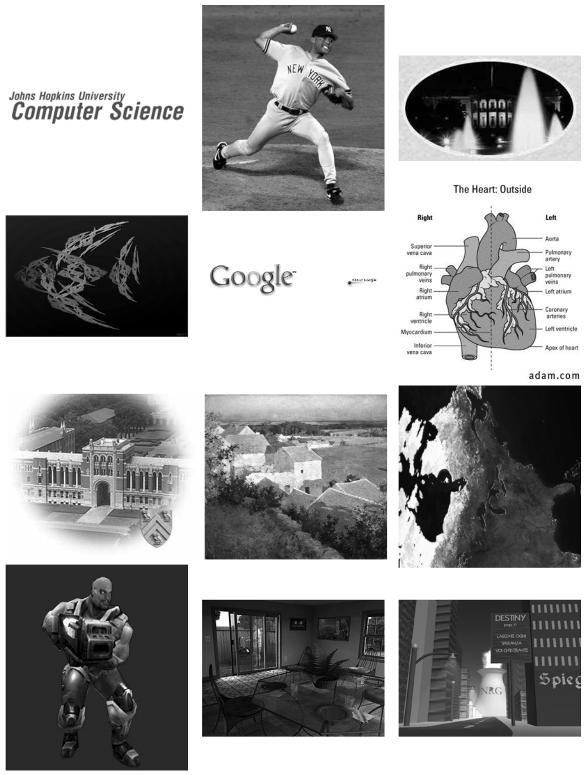


Figure 6: *Image Suite Thumbnails. The images, ordered from left to right and top to bottom are: 2s, baseball, bevel, fish, google, heart, lovett, metcalf, north america, quake, render, and sign.*

The histograms beneath the quake images show how many pixels lie in blocks at various frequency ranges. The bell curve of the warped image is characteristic of the shifting of frequencies towards the midrange.

Table 1 presents some numerical results for each test image. We compare the effect of resizing the original image by a factor of 0.5 to resizing using our warping algorithm. To measure this effect, we count the number of pixels which lie in blocks whose frequency content is too high for the given resize operation and take this as a percentage of the total pixels. This is a rather approximate measure, because we know little about any pixel except for the characteristics of its block. However, it does give some indication of the benefit of our approach. The table also demonstrates that the auxiliary tree structure requires relatively little storage as a percentage of the original image size.

We examine the effectiveness measure over a range of scale factors for the quake image in Figure 9. According to this approximate measure, our algorithm is clearly superior for scale factors from 1.0 to 0.5. However, as shown in color plates 2d and 2e, our image retains more of the sharp features even at a scale of 0.25.

Color plates 1 through 5 justify our claims of producing higher uniformity in the frequency domain. We show the result of resizing each image without our uniform frequency algorithm as compared to resizing with our algorithm. In each case, we see some sharp features that our algorithm preserves, such as the "About Google" text in google, the specular highlights in quake, the carpet and chairs in render, the stripes and text in baseball, and the Great Lakes in north america. Unfortunately, we also see discontinuity artifacts due to our current k-d tree algorithm. These are especially noticeable in the upper right quadrant of

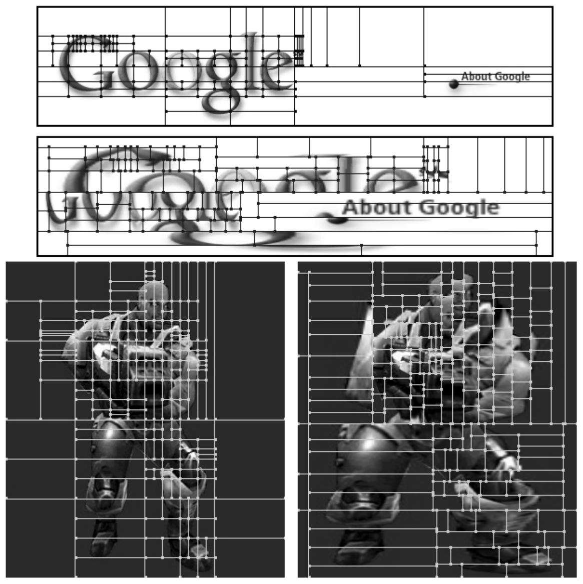


Figure 7: *K-d Trees. The google and quake images are displayed with their k-d trees, first unwarped, then after warping.*

5e. (For a detailed examination of these images, you may wish to view them in digital form, available on the proceedings CD-ROM and our web site.)

As expected, images which are fairly uniform to begin with, especially images which contain high frequency data, are resistant to being downsampled. Typical examples include small texture map patterns for modeling brick or cloth. Images with large first derivative of frequency content, such as images containing text over a solid background region, resize well.

Overall, the results of our test runs suggest that our algorithm outperforms standard texture mapping techniques by preserving detail with decreased image store.

7 Future Work

There are several limitations to our current approach, some of which we have already discussed. This paper presents a "proof of concept" implementation, but clearly more work is needed. We believe the following areas may be fruitful for improving and extending this research:

- A more flexible source mesh. The current implementation does not allow for an arbitrary choice of source point, but adding this extra degree of freedom to the warp computation may produce better warps.
- Using triangle meshes instead of quads. By using triangle meshes, we can maintain zero-order continuity across the image and avoid cracks. However, it may be difficult to perform frequency analysis on a triangular subset of an image.
- Applying to complex, multi-resolution, textured meshes. Our current work only applies directly to a simple image, but it may be extended to compute a new parameterization for a complex mesh whose current parameterization does not have uniform frequency properties.
- Applying to 3D textures. 3D textures require even greater resources than their 2D counterparts, so this research has even greater potential benefit in that domain.

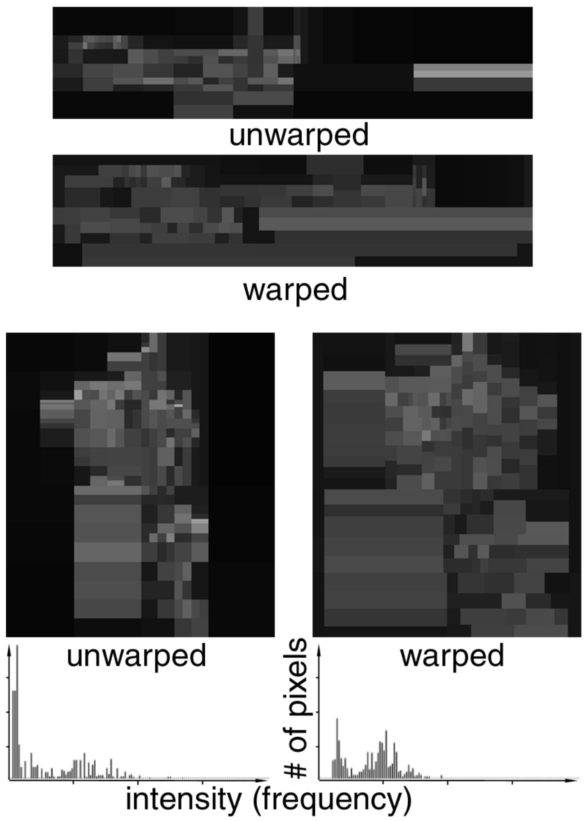


Figure 8: *Frequency Distribution, before and after warping. The google and quake histograms are shown with frequency content mapped to intensity. The quake distributions are also plotted as a histogram. Note the spreading effect. Whites and blacks approach medium gray. Also, shifting the frequencies towards the center produces a more uniform bell-curve.*

- Applying to video. There are opportunities to exploit coherence between the video frames to achieve greater compression. We can find coherence in the 2D warping functions of successive frames or even consider an entire sequence of frames as a 3D image to exploit the coherence in a natural way.
- More advanced content analysis. Although our work on analyzing the content of subregions of the image was fruitful, we are exploring other methods, based on wavelet transforms and other content-based research.
- Proving error bounds. Although in theory our method should generate provable error bounds, in practice it has proven difficult to do so because of the difficulty of imposing a continuous warp on a discrete space, and inherent pixel-level error in texture-mapping routines.
- Designing for perceptual metrics. Our work has so far largely ignored the research in perceptual metrics. One paper we have considered is that of Rushmeier et al.[12] Preliminary analysis of our algorithm using these metrics has been positive.

The uniform frequency image representation we have presented allows compression of image data while maintaining the regularity that makes images convenient. The need for such compression is increasing with our ability to acquire large data sets and our desire to visualize them. We are optimistic that

Image Suite					
Name	Size (x,y)		Affected Pixels (warped, unwarped)		Tree Size (% of image)
2s	252	62	19%	43%	2.2%
baseball	398	450	6%	14%	1.7%
bevel	300	174	28%	34%	1.6%
fish	450	300	58%	63%	0.9%
google	556	130	7%	11%	0.9%
heart	353	405	36%	52%	2.0%
lovett	332	287	9%	29%	1.3%
metcalf	431	390	7%	15%	2.4%
n. america	512	512	9%	8%	1.3%
quake	300	340	7%	14%	1.1%
render	400	300	33%	50%	0.9%
sign	640	480	9%	17%	1.3%

Table 1: *Image test suite. The affected pixels column indicates the percentage of pixels which are predicted to be affected by more than 15% data loss (as measured in the frequency spectrum) if the image were to be shrunk by a factor of 2 in each dimension.*

improvements to the approach presented here will provide performance benefits to a wide array of visualization applications.

8 Acknowledgments

We would like to thank Id Software for the quake image, Michael Garland for the image of North America, and Google, Inc. for the google image. We would also like to thank Marc Olano, Greg Hager and Sarah Graham. This work was supported in part by a Department of Defense National Defense Science and Engineering Graduate Fellowship.

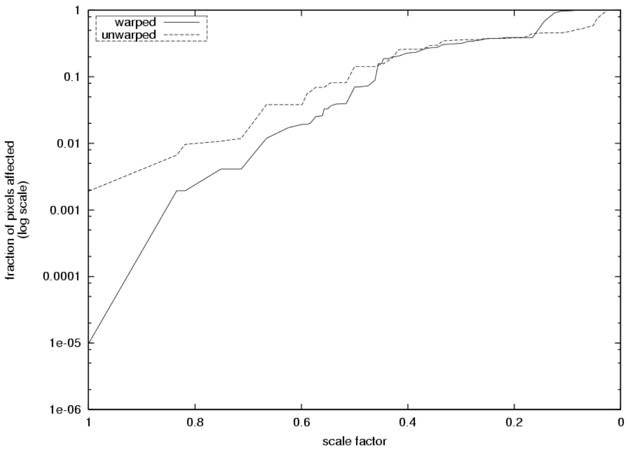


Figure 9: *Affected pixels for quake. The number of pixels affected by more than 15% data loss is much greater for the unwarped quake than for the warped quake, at least for the scales up to 0.5. Visual results look good beyond this range as well (see color plates 2d and 2e).*

References

[1] 3Dlabs. The virtual texture engine: Texture management on the Permedia 3 and GLINT R3. Technical report, 3Dlabs, 1999.

[2] Rui Bastos, Michael Goslin, and Hansong Zhang. Efficient radiosity rendering using textures and bicubic reconstruction. *1997 Symposium on Interactive 3D Graphics*, pages 71–74, April 1997. ISBN 0-89791-884-3.

[3] Andrew C. Beers, Maneesh Agrawala, and Navin Chaddha. Rendering from compressed textures. *Computer Graphics*, 30(Annual Conference Series):373–378, 1996.

[4] G. Campbell, T. A. DeFanti, J. Frederiksen, S. A. Joyce, L. A. Leske, J. A. Lindberg, and D. J. Sandin. Two bit/pixel full color encoding. *Computer Graphics*, 20(4):215–223, August 1986.

[5] Jonathan Cohen, Marc Olano, and Dinesh Manocha. Appearance-preserving simplification. *Proceedings of SIGGRAPH 98*, pages 115–122, July 1998. ISBN 0-89791-999-8. Held in Orlando, Florida.

[6] M. Frigo and S. G. Johnson. The fastest fourier transform in the west. Technical Report MIT/LCS/TR-728, Massachusetts Institute of Technology, September 1997. software available from http://www.fftw.org.

[7] Pat Hanrahan and Jim Lawson. A language for shading and lighting calculations. *Computer Graphics*, 24(4):289–298, August 1990.

[8] Kazufumi Kaneda, Fujiwa Kato, Eihachiro Nakamae, Tomoyuki Nishita, Hideo Tanaka, and Takao Noguchi. Three dimensional terrain modeling and display for environmental assessment. *Computer Graphics*, 23(3):207–214, July 1989.

[9] Anders Kugler. High-performance texture decompression hardware. *The Visual Computer*, 13(2):51–63, 1997. ISSN 0178-2789.

[10] Marc Levoy. Polygon-assisted JPEG and MPEG compression of synthetic images. *Computer Graphics*, 29(Annual Conference Series):21–28, November 1995.

[11] Marc Levoy and Pat Hanrahan. Light field rendering. *Computer Graphics*, 30(Annual Conference Series):31–42, 1996.

[12] H. Rushmeier, G. Ward, C. Piatko, P. Sanders, and B. Rust. Comparing real and synthetic images: Some ideas about metrics. In *Eurographics Rendering Workshop 1995*. Eurographics, June 1995.

[13] Peter-Pike J. Sloan, David M. Weinstein, and J. Dean Brederson. Importance driven texture coordinate optimization. *Computer Graphics Forum*, 17(3):97–104, 1998. ISSN 1067-7055.

[14] Christopher C. Tanner, Christopher J. Migdal, and Michael T. Jones. The clipmap: A virtual mipmap. *Proceedings of SIGGRAPH 98*, pages 151–158, July 1998. ISBN 0-89791-999-8. Held in Orlando, Florida.

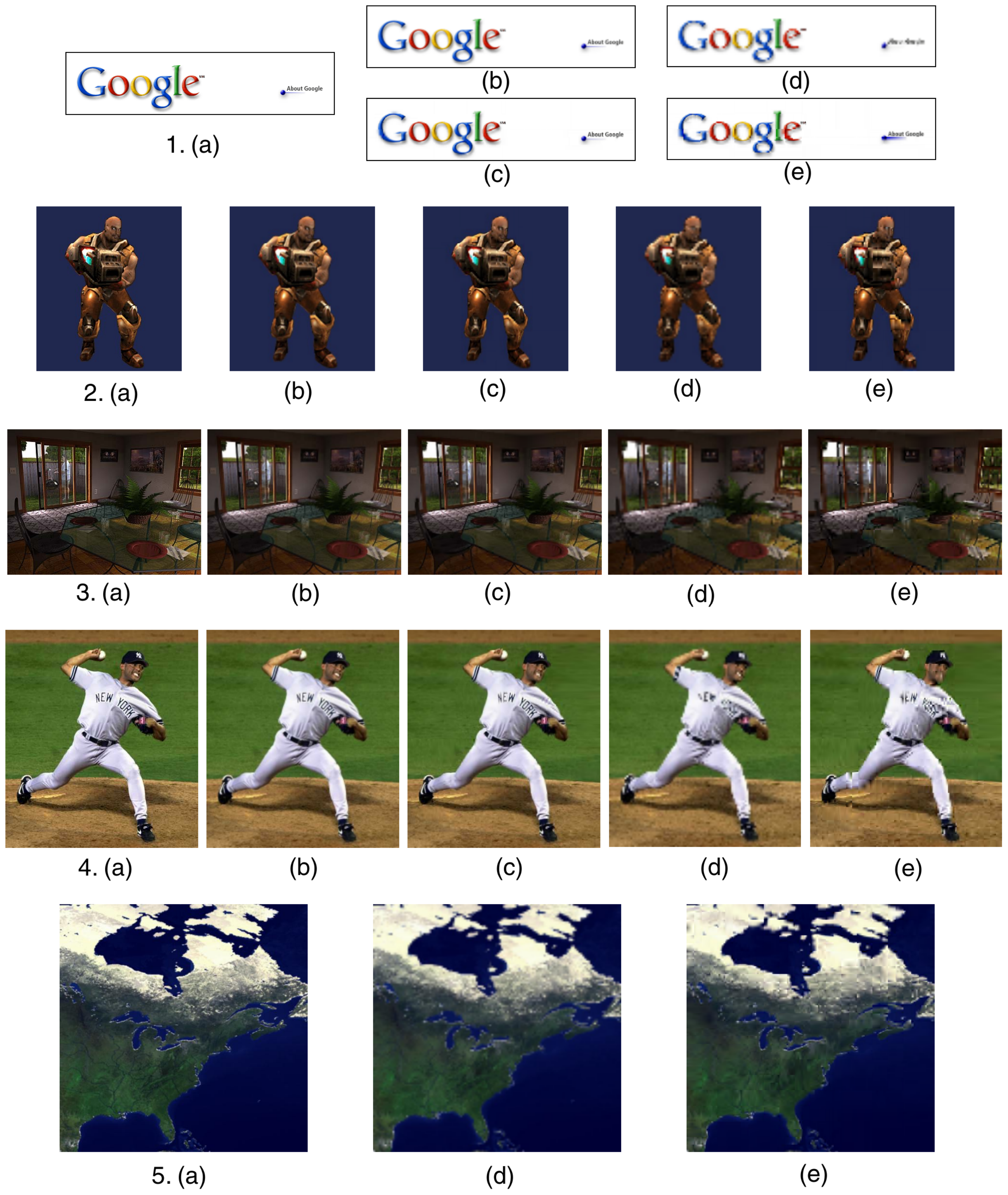
[15] D. Terzopoulos and M. Vasilescu. Sampling and reconstruction with adaptive meshes. In *CVPR91*, pages 70–75, 1991.

[16] Jay Torborg and Jim Kajiya. Talisman: Commodity real-time 3d graphics for the pc. *Proceedings of SIGGRAPH 96*, pages 353–364, August 1996. ISBN 0-201-94800-1. Held in New Orleans, Louisiana.

[17] Gregory K. Wallace. The JPEG still picture compression standard. *Communications of the ACM*, 34(4):30–44, April 1991.

[18] L. Williams. Pyramidal parametrics. *Computer Graphics*, 17(3):1–11, July 1983.

[19] Mason Woo et al. *OpenGL programming guide: the official guide to learning OpenGL, version 1.2*. Addison-Wesley, Reading, MA, USA, third edition, 1999.



5 UFI test images: 1 - Google, 2 - Quake, 3 - Render, 4 - Baseball, 5 - North America.
(a) Original, (b) Scaled by 0.5 then 2.0, (c) Scaled as (b) but using UFI warping,
(d) Scaled by 0.25 then 4.0, (e) Scaled as (d) but using UFI warping