Volume xx (*200y*), *Number z, pp. 1–13*

Image Interpolation by Pixel Level Data-Dependent Triangulation

Dan Su, Philip Willis

Department of Computer Science, University of Bath, Bath, BA2 7AY, U.K.
mapds, P.J.Willis@bath.ac.uk

Abstract

We present a novel image interpolation algorithm. The algorithm can be used in arbitrary resolution enhancement, arbitrary rotation and other applications of still images in continuous space. High resolution images are interpolated from the pixel level data-dependent triangulation of lower resolution images. It is simpler than other methods and is adaptable to a variety of image manipulations. Experimental results show that the new "mesh image" algorithm is as fast as the bilinear interpolation method. We assess the interpolated images' quality visually and also by the MSE measure which shows our method generates results comparable in quality to slower established methods. We also implement our method in graphics card hardware using OpenGL which leads to real-time high-quality image reconstruction. These features give it the potential to be used in gaming and image processing applications.

1. Introduction

Digital image interpolation is the recovery of a continuous intensity surface from discrete image data samples. It is a link between the discrete world and the continuous one. In general, almost every geometric transformation requires interpolation to be performed on an image, e.g. translating, rotating, scaling, warping or other applications. Such operations are basic to any commercial digital image processing software.

There are several issues which affect the perceived quality of the interpolated images: sharpness of edges, freedom from artifacts and reconstruction of high frequency details. We also seek computational efficiency, both in time and in memory. Classical techniques, such as pixel replication, bilinear or bicubic interpolation have the problem of blurred edges or artifacts around edges. Although these methods preserve the low frequency content of the sample image, they are not able to recover the high frequencies which provide a picture with visual sharpness.

Standard interpolation methods are often based on attempts to generate continuous data from a set of discrete data samples through an interpolation function. These methods attempt to improve the ultimate appearance of re-sampled images and minimise the visual defects arising from the inevitable resampling error.

Traditionally, interpolation is accomplished through convolution of the image samples with a single kernel – typically a bilinear, bicubic[1], or cubic B-spline[2]. A number of algorithms have been proposed to improve the magnification results. PDE-based approaches[3] apply a nonlinear diffusion process controlled by the local gradient. POCS (Projection-Onto-Convex-Set) schemes[4] formulate the interpolation as an ill-posed inverse problem and solve it by regularised iterative projection. Orthogonal transform methods focus on the use of the discrete cosine transform (DCT)[5, 6]. Directional methods[7, 8] examine an image's local structure around edge areas to direct the interpolation. Variational methods formulate the interpolation as the constrained minimisation of a functional[9, 10].

It has been recognised that taking edge information into account will improve the interpolated image's quality [11, 12, 13, 14] and it is known that the human visual system makes significant use of edges[18]. Instead of approaching interpolation as simply fitting the interpolation function, these methods consider also the geometry of the image. Li[11] asserts that the quality of an interpolated image mainly de-

pends on *the sharpness across the edge* and *the smoothness along the edge*.

Li et al.[11] attempted to estimate local covariance characteristics at low resolution and used them to direct interpolation at high resolution (NEDI - New Edge Directed Interpolation) while Allebach et al.[12] generated a high resolution edge map and used it to direct high-resolution interpolation (EDI - Edge Directed Interpolation). Battiato et al.[13] proposed a method by taking into account information about discontinuities or sharp luminance variations while doing the interpolation. Morse et al.[14, 15] presented a scheme that uses existing interpolation techniques as an initial approximation and then iteratively reconstructs the isophotes using constrained smoothing. They emphasise the importance of the "smoothness" quality, if the isophotes are not to be visually intrusive. As will shortly become clear, we too accept this need to fit the visual geometry.

The above schemes demonstrate improved visual quality (in terms of sharpening edges or suppressing artifacts) by using a model to preserve the edges of the image and to tune the interpolation to fit the source model. However they are complex compared to traditional methods and thus computationally expensive.

Another approach is triangulation modelling. Triangulation has been an active research topic during the past decade. It is popular in geometric modelling. However, image reconstruction using triangles isn't widely used, probably because of the large number of triangles needed. Yu et al.[16] modelled images as data dependent triangulation meshes and reconstructed images from the triangulation mesh. Their approach adapted traditional data-dependent triangulation[17] (DDT) with their new cost functions and optimisations. The data dependent triangulation thus matches the edges in the image and improves the reconstructed image. Their method is relatively complex and computationally expensive.

We develop a new edge-directed method for image interpolation. We call this an *image mesh* DDT. We do not assume knowledge of the low-pass filtering kernel or attempt to find a statistical rule about the local geometry. Our approach is related to that of Yu but is simpler and faster because it does not involve any cost function or repeating optimisation process. Our mesh is very simple and completely regular. We avoid the complexity of a full DDT method while keeping the feature of DDT that improves the reconstruction quality. We will demonstrate our algorithm used in arbitrary magnification of still images and other applications.

2. Image Mesh Data-Dependent Triangulation

2.1. Principle of the Algorithm

We first consider the case that there is an edge passing between a square of four pixels. If this edge cuts off one corner, one pixel will have a value substantially different to the other three. We call this pixel the *outlier*. Imagine that we represent the brightness of the pixel as the height of a terrain. In effect, the three similar pixels define a plateau, relatively flat, while the outlier value is at the bottom of the cliff (if smaller) or the top of a peak (if higher) (Figure 1). This gives us a hint that if we want to interpolate a high resolution pixel within the relatively flat region we should not use the outlier. Classical interpolation methods like bilinear interpolation suffer from edge blurring because they use all four pixels to do interpolation. We only use three.

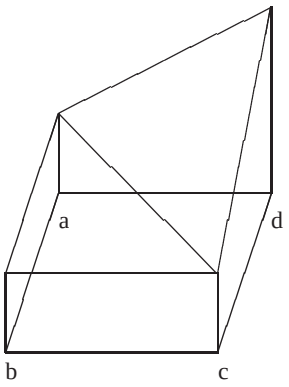


Figure 1: *Triangulation in a four-pixel square*

The strength of employing triangles in this way is that we model edges in the image. In effect we tune the interpolator to match edges. In Figure 1, when interpolating the high-resolution pixel falling in triangle *abc*, the interpolator won't use the value of *d* which is very different to this plateau. For two pixels falling in different triangles, the height of the vertices will be quite different and thus the sharpness of the edge is kept. It is easy to see that in very smooth regions, the interpolator keeps smoothness as well, even across triangle boundaries.

This simple geometry suggest a way to guide the interpolation so that smoothness within the regions and sharpness between the flat region and cliff region can both be kept. If the diagonal is to correspond to the edge in the image, the diagonal should be the one which does *not* connect to the outlying pixel value, the one most different to the other three.

Suppose pixels *a*, *b* and *c* are the same height while *d* is higher than these three. Obviously *a*, *b* and *c* define a flat region while *d* is the most different pixel to the other three. Thus we connect diagonal *ac* and get the triangles *abc* and *adc*. In general, if *b* or *d* is the most different pixel, the edge should be *ac*, otherwise *bd* will be the edge. There are other situations if *a* and *d* are very different to *b* and *c*; or *a* and *b* are very different to *c* and *d*. In these cases it makes little difference which diagonal is chosen. The edge is roughly either horizontal (*ad* are different to *bc*) or vertical (*ab* are

different to cd) and the triangle will always cross the edge. It is similar to bilinear interpolation in these cases.

Obviously, using the diagonal to triangulate the four-pixel square cannot correspond to edges of arbitrary angle. The diagonal can only roughly represent the orientation of the edge. We could use sub-pixel triangulation to represent arbitrary angles, but that would add more complexity to the algorithm. Our aim is to keep the algorithm as simple as possible. We will demonstrate in this paper that triangulation by diagonal is enough in most situations and can provide excellent results. It is the direction-selection method that is the key.

Our method thus fits the finest triangular mesh to the source pixels. This "image mesh" is completely regular except that the diagonals are locally selected to run in the same general direction as any visible edge. To generate a new image, possibly at higher resolution, the target pixels are located in the source mesh. We then evaluate each target pixel from the triangle in which it sits. It is interpolated using only the information from the three triangle vertices. In edge areas, the interpolator won't interpolate any two pixels that fall in different triangles. In other words, the new high-resolution image has the edges sharp and the smooth areas smooth.

2.2. Implementation and Optimisation

Suppose the low-resolution image is X and the high-resolution image to be generated is Y. Our algorithm can be expressed as two steps. We first scan the sample image X to initialise a 2D array which records the edge direction of all four-pixel squares. In the second step we scan Y. For each y_{ij} we inverse map to the sample image X and use the array to identify the triangle in which the point falls. Then we interpolate within that triangle to get the value of y_{ij}.

In the first step, the algorithm has to determine the outlier pixel. This has to be done repeatedly, so speed is important. Instead of finding the outlier directly, we compare the difference $|a-c|$ with the difference $|b-d|$ and connect the pair with smaller difference. The proof that this is equivalent to finding the outlier pixel is in Appendix A. This saves computing time, needing only two subtractions and a comparison. Doing it directly would require sorting four pixels and then comparing the highest and lowest pixels with the average value.

We use inverse mapping in the interpolation step because it has a number of benefits. First it can be used at arbitrary resolution. We are not constrained in any way by the resolution of the source data. Second, there is no requirement to align the target grid parallel to the source grid, so arbitrary rotation is possible at no additional cost. Third, sampling can be irregular to provide warps, although the sampling rate must not be too low because this would cause break-up. Finally it is a single-step method.

We use linear interpolation within the triangles. However there is some confusion of terminology in the literature, which we need to clarify before proceeding. "Bilinear interpolation" strictly refers to interpolating between four values and we will use the term only in that sense. In the graphics community, three-value interpolation, as used in Gouraud shading, is also called bilinear interpolation, although it is only a degenerate case. We will distinguish this by calling it "triangle interpolation".

Figure 2 shows a flower image with the magnified view of the tip of the lowest stamen and the pixel level data dependent triangulation mesh of that stamen. (We only show the diagonals of the triangles for a clearer view.) We represent the triangulation in two diagrams, each one only containing a specific direction. The stamen and a black edge near the stamen both roughly have NW-SE orientation. It is clear to see that the corresponding triangles also cluster in the NW-SE direction, which matches the edges of the image. In particular, note the absence of NE-SW diagonals near these linear features.

2.3. Extended Method

Some problems still remain in our basic model. For example, close study of the triangulation of the stamen (Figure 2) reveals a problem. The actual local edge goes in the NW-SE direction while a few diagonals in the lowest stamen areas give the NE-SW direction. This leads to some small deterioration of edge quality. These diagonals contradict the local edge orientation because our basic method only considers the four-pixel square, ignoring the surrounding values. This only catches the micro-geometry (pixel level), not the local geometry due to edges passing through several pixels. To correct this we have developed an extended model where we consider this extra information.

We assume the image is locally stationary. That is to say, the intensity of a pixel is dependent on its spatial neighbourhood while independent of the rest of the image. The neighbourhood of a pixel can be modelled as a window around this pixel. Instead of a normal least-square adaptive edge prediction scheme, we simply consider the neighbourhood window's edge direction. Our basic method considers four pixels arranged in a square. Our extended method considers 16 pixels arranged as 3×3 squares. To predict the edge direction in the central square, we consider all of them (Figure 3). If most of these squares have their diagonals in one particular direction, then we impose that direction on the central square. In our case we do this if at least 6 of the 9 squares have the same direction. All decisions are made on the original data so that changes do not influence nearby decisions taken later.

Obviously our extended model increases complexity, but very marginally. It is worth noting that this additional complexity is only in preparing the diagonals, not in using the mesh to interpolate an image.

Figure 4 shows the diagonals resulting from our extended

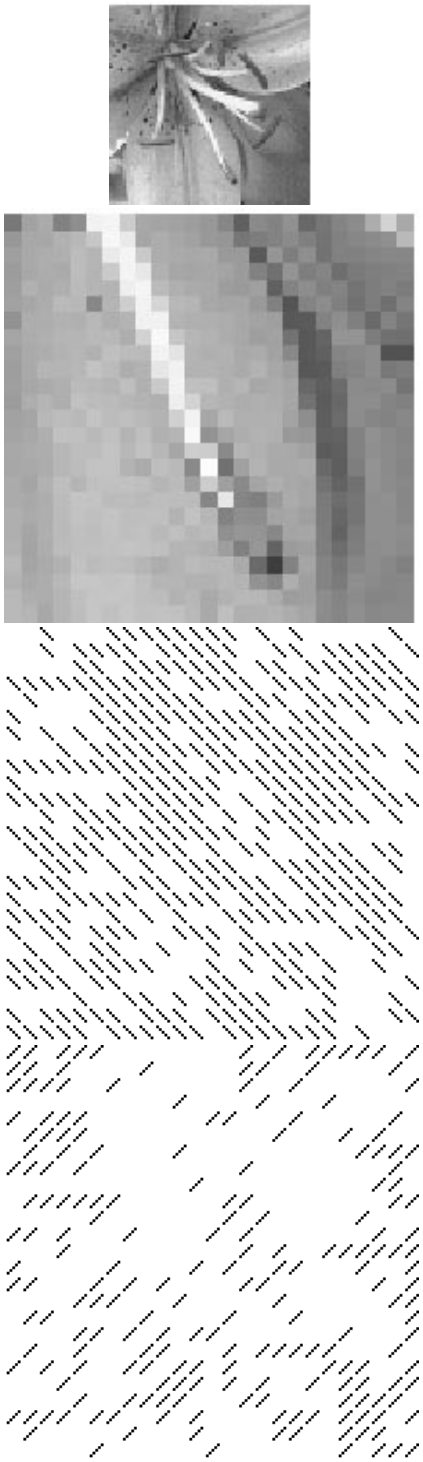


Figure 2: *Top: a part of a flower image. Second: a magnified view of the bottom stamen. Third: the pixel level data dependent triangulation of the stamen (NW-SE direction) Bottom: NE-SW direction*

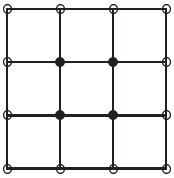


Figure 3: *Neighbourhood of 3 × 3 squares*

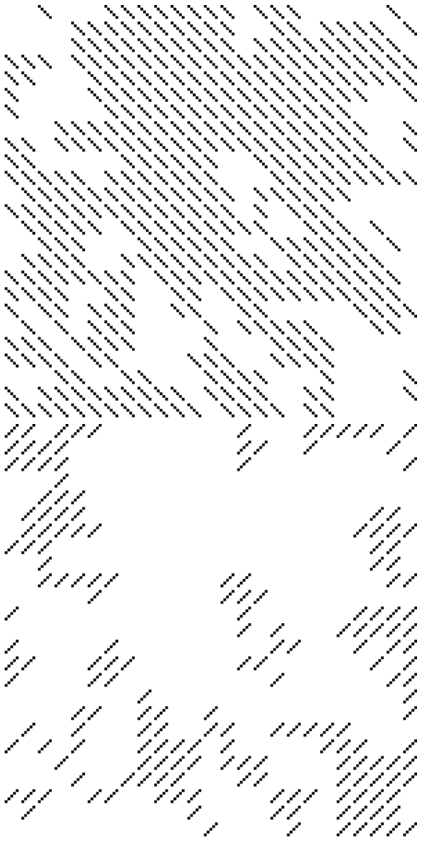


Figure 4: *The triangulation mesh of the extended method. Top: NW-SE direction. Bottom: NE-SW direction*

method. The stamen of Figure 2 has 625 diagonals. Our basic method generates 418 diagonals in the NW-SE direction and 207 diagonals in the NE-SW direction while our extended method produces 438 and 187 diagonals respectively. They differ only in 20 diagonals, mainly along the stamen and the black edge: the extended method better preserves the local geometry.

2.4. Algorithm Analysis

We analyse the complexity of the basic method and the extended method in this section. Suppose the image I has width and height m, so the number of pixels is $n = m^2$. The number of triangles in the triangulation is then $(m-1) \times (m-1) \times 2$. In our implementation, we use a table to record the orientation of the diagonal in each square. As there are only two possible diagonal directions for each square of two traingles, we use one bit to store this information. Thus, the total memory requirement for the triangulation mesh is $(m-1)^2 \approx n$ bits. In other words, we need one bit per pixel. For a normal image with size 1024×1024 the memory requirement is $128KB$. Compared to the standard memory in current PCs, this is very small. Moreover, the memory requirement n is linear with the number of pixels n.

In our basic method, each triangle needs two subtractions and one comparison, so the total computation is $(m-1) \times (m-1) \times 2 \times 3 \approx 6n$.

Our extended method has two steps in preparing the mesh. In the first step, we calculate just like the basic method and set each triangle's diagonal direction. In the second step, each triangle needs a sum of eight surrounding squares and a comparison to decide if there is an overriding edge orientation in local area. Thus, the computation for each triangle needs two extra computations, and the whole image needs $10n$ computation which is still linear with image size n.

Then follows the interpolation step. It is easy to see that the triangle interpolation has the same complexity as bilinear interpolation which is linear with n. Thus, both the basic method and the extended method have a time complexity of $O(n)$.

Our method is thus efficient in both memory and time, and is suitable for handling large images with a linear dependency on the image size.

2.5. Algorithm Comparison

Yu et al.[16] propose an image reconstruction method using data dependent triangulation. They use a new cost function and an improved optimisation algorithm to generate an optimised triangulation mesh. Their method is able to model an image effectively. It is complex to implement and is computationally slow. It takes several iterations to get an optimised triangulation and each iteration takes "between 0.5 and 5 seconds" even for a small image (80×80) on a consumer-grade PC. Another limitation of the method is it cannot catch single-pixel and small features.

Our method can be thought of as a simplified data dependent triangulation (DDT). It generates the triangulation mesh simply by inserting diagonals. This leads to some degradation in quality since a normal DDT can model the edge at arbitrary angles. However our method provides a notable trade-off between quality and speed. Although the DDT method can in principle give higher quality, ours is very easy to implement and much faster. Also our method needs only a small byte array to store the triangulation mesh while a full DDT requires a more complicated structure and more storage space. An advantage of our extended method is it is able to catch small and local features.

Other researchers[19] also use DDT for data interpolation, aiming at a better optimisation of DDT according to their cost functions and optimisation processes. Our method avoids this. We will now demonstrate that the method is effective and that it does provide high-quality reconstructed images compared to conventional methods.

3. Experimental Assessment

3.1. Implementations

We implemented several interpolation methods. The images from bilinear interpolation and bicubic interpolation were produced from Matlab 5 built-in functions. The NEDI method was tested from a Matlab program kindly provided by its originator. We used a C++ program and our own graphics library to implement our methods.

Greyscale images were processed exactly as already described. When selecting edge directions in colour images, we converted the RGB components of each pixel into luminance using the following formula[16] where L stands for luminance:

$$L = 0.21267R + 0.71516G + 0.07217B$$

The edge direction was determined by these luminance values. Interpolation was performed in the R,G,B planes independently.

3.2. Visual Assessment

We performed preliminary tests both to check the implementations and to permit a visual assessment of the methods. We wanted to use an image in both in greyscale and in colour. The flower image we have used has well-defined edges (to test edge sharpness), thin linear features and small details (to ensure they are retained) and smoothly varying areas (to reveal any discontinuity).

Figures 5 and 6 show the comparison results. All the images in Figures 5 and 6 are magnified from the flower image of Figure 2 by a factor of 4.

Figure 5: *Detail of image magnified by 4. Top: bilinear interpolation. Middle: bicubic interpolation. Bottom: NEDI*

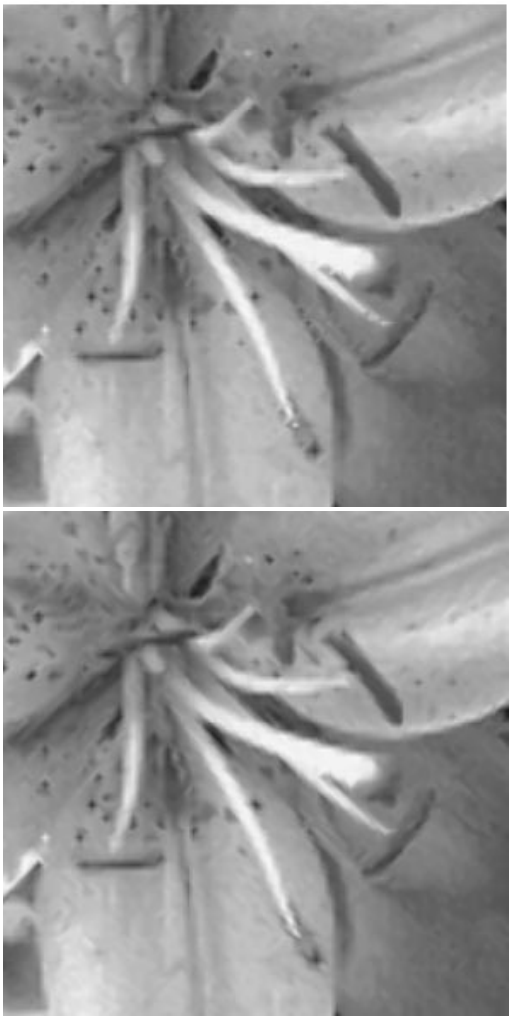


Figure 6: *Detail of image magnified by 4. Top: our basic method. Bottom: our extended method*

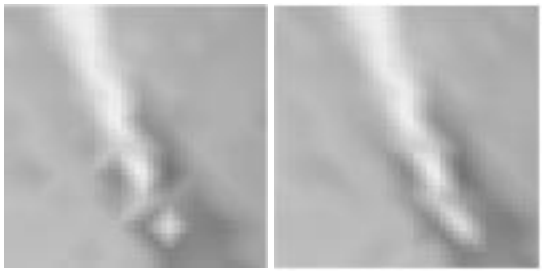


Figure 7: *Magnified view of the stamen. Left: our basic method. Right: our extended method*

Figure 7 shows a close-up view of the stamen using our basic and extended method. This illustrates that the basic method has some artifacts along the stamen which are reduced in the extended method.

Figure 8 shows the various methods used to magnify the colour flower image by a factor of 3.5.

From visual inspection our method produces better images than bilinear and bicubic interpolation, while the NEDI method is better still (Figures 5 and 6). However, it seems NEDI's weighting algorithm changes the contrast of the image. The bilinear interpolation suffers from blurring of the edges. The bicubic method introduces sharper edges but more artifacts.

We next performed analytical testing.

3.3. Quality Assessment

To perform analytical assessment of the the interpolated images, we need a quality measure. The degradation based method[20] is not able to report the "jagged" artifacts related to the orientation of edges. Daly's visible differences predictor[21] produces an error image which characterises the regions in the test image that are visually different from the original image. It is relatively complex to implement and it is not really feasible to use error images for ranking, as Daly mentioned in his paper. Therefore we used mean-square error (MSE) as our assessment tool. The MSE is the cumulative squared error between the reconstructed and the original image. It is widely used in image processing to evaluate reconstructed image fidelity.

Our method aims at improving edge quality on magnified images and retaining a good overall quality as well. Thus we produced one sample image set of five "edge" images with size 200×200 (Figure 9) and used twenty 768×512 real nature images as a more general test set.

In theory, there is no perfect way to judge the magnification quality. Because the image we have is of fixed resolution, we don't know what the 'correct' magnified image is. In order to analyse error, we need to know or simulate this image. So we start with an original image, generate a lower resolution version, then use different methods to magnify it. Then we compare the magnified image with the original image. This is not perfect but it provides a reasonable reference against which to measure the reconstruction quality.

The down-sampled images could be obtained by averaging or sub-sampling. However, edge blurring and ringing are introduced by averaging, while sub-sampling breaks down the geometry and introduces artifacts. We chose a Gaussian filter as the point-spread function with its standard deviation representing the radius of the point-spread function. Each pixel at the target image (the down-sampled image) is considered as a point-spread function represented by a Gaussian distribution. It is down-sampled from some part of the

Figure 8: *A flower image magnified by a factor of 3.5 using: Top: bilinear interpolation. Middle: bicubic interpolation. Bottom: our extended method.*

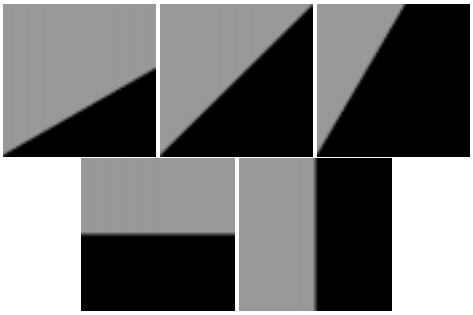


Figure 9: *Set of five edge images. The angles are 30, 45, 60, 0 and 90 degrees*

source image, represented by another point-spread function. In this case the radius of the point-spread in the source image is double that of the radius in the target image. Thus, we calculate the standard deviation of the target Gaussian distribution, then double this to get that of the source image. This is then used to down-sample, by convolution.

We used pixel replication, bilinear interpolation, bicubic interpolation, NEDI, our basic method and our extended method to obtain the reconstructed images. All reconstructed images are magnified by a factor of two and then compared to the original image.

3.3.1. Quality of edges

Our first test was to check the quality of well-defined edges. For the test set we generated five samples with a single edge of varying angle (30, 45, 60, 0 and 90 degrees). Each edge is black one side and white the other side (Figure 9). The down-sampled edge images are magnified by a factor of two and compared to the original edge images to get the MSE results which is reported in Table 1. The MSE is performed on 0 255 range for the grey-scale edge images. We put 0° and 90° in the same column because they give the same results for all methods. Our basic and extended methods have the same results in all these situations because our basic method is able to preserve the geometry well in these simple cases.

The MSE results report that our method gets the best (lowest) score in every case except at 0° and 90°. In these two cases pixel replication gets the best score, which it is trivially able to do. (In principle it should achieve zero MSE but the Gaussian sampling introduces some grey edge pixels.) Bicubic beats us here because its interpolation more sharply models these high-contrast edges. Our method is the equal of bilinear interpolation as we expect. Although our triangulation gives edges of 45°, it also performs well on 30° and 60°. Bicubic and bilinear interpolation are slightly worse because they suffer from artifacts or blurring on the edge. Pixel replication does not generally catch the geometry very well and NEDI suffers from the effects of its weighting algorithm.

	30°	45°	60°	$0^\circ, 90^\circ$
Our methods	28.8	28.9	28.8	26.0
Bicubic	29.7	31.5	29.3	22.2
Bilinear	34.0	38.4	34.0	26.0
Replication	41.8	45.4	41.5	9.2
NEDI	43.3	47.6	43.4	27.6

Table 1: *MSE results of edge images*

3.3.2. Quality of real images

In order to test the method on "smoother" and more typical images, we used twenty 24-bit 768×512 colour nature images as another test set. These images are down-sampled, magnified by different methods by a factor of two and compared to the original images. We perform the MSE comparison on R,G,B channels independently and Table 2 reported the averaged MSE values / standard deviations over 20 images from the test set. BC is the bicubic interpolation, EXT is our extended method, BL is bilinear interpolation, DDT is our basic method and PR is pixel replication.

	R	*G*	*B*
BC	109.4 / 85.1	119.4 / 106.7	123.8 / 121.2
EXT	117.6 / 92.8	127.8 / 115.2	132.7 / 131.3
BL	118.2 / 92.9	128.4 / 115.2	133.1 / 130.9
DDT	118.6 / 93.6	128.8 / 116.1	133.7 / 132.3
PR	126.1 / 99.7	134.8 / 120.8	138.7 / 137.2
NEDI	198.6 / 180.1	197.9 / 159.9	187.4 / 154.5

Table 2: *MSE results of real images*

However the standard deviations are very large. We also did a statistical t-test over the MSE results of the different methods and this confirmed that no significance could be read into these results. A full analysis would therefore need a much bigger set of pictures than we have available. Noting that these have to be of known provenance (for example, compressing a picture or simply reducing the pixel bit-depth introduces visually and statistically significant defects) it is not sufficient simply to analyse a large collection culled from the web. We therefore ceased any deeper analysis.

Visual inspection of our method shows that it produces good results, which we believe are due to its edge performance. We will now show that our method is intrinsically

very fast and easily made faster by using the PC graphics card.

3.4. Performance Assessment

We implemented bilinear interpolation, bicubic interpolation, our basic method and our extended method by C++ code and compared their computational performance. We used the real natural colour images test set again. We down-sampled every image to 384 $\times$ 256 pixels (using the method described earlier). Then we magnified the down-sampled images by a factor of 2 and also by a factor of 3.5. We used the bicubic interpolation proposed by Keys [22]. Table 3 shows the performance comparison on a machine with an Intel Pentium4 3G processor and 1G DDR system memory. Our extended method uses the 3 $\times$ 3 square window. All figures are in seconds.

	Bilinear	*Basic*	*Extended*	*Bicubic*
magnify 2	0.359	0.406	0.412	3.621
magnify 3.5	1.105	1.162	1.170	10.914

Table 3: *Performance comparison*

We can see from the table that our method is only slightly slower than bilinear interpolation. Importantly, bicubic is an order of magnitude slower than the other methods. The averaged times for calculating the triangle mesh are included in the above figures. For our basic and extended method these are 0.041 and 0.049 seconds respectively. Factoring these out reveals that our methods are linear with the number of pixels generated.

In conclusion, our extended method is comparable in speed to bilinear interpolation while providing excellent reconstruction results visually. In comparison to bicubic interpolation, our extended method is much faster and visually better, especially in edge reconstruction. Our method is fast, simple and modest in memory needs.

3.5. Hardware Implementation

More and more complex graphics operations have moved to the graphics co-processor or accelerator, including shading, texturing, anti-aliasing and bilinear interpolation. These features of graphics cards make it possible to create extremely realistic games and simulations.

However the only interpolation algorithms currently available on graphics cards are triangular and bilinear interpolation: the others are too complex. High quality image reconstruction in real-time still remains a difficult and unsolved problem. Our pixel level data dependent triangulation makes a step in this direction.

A graphics card can handle tens of millions of triangles per second and it can interpolate within triangles. This suggests that we convert any image to a triangle mesh and then pass the mesh to the graphics card. The card will deal with the mesh in real-time.

We have used OpenGL to explore the potential of our method in hardware implementation. We first generated a triangle mesh using our basic or extended model. Then we used OpenGL to pass the mesh to the graphics card so that it could manipulate the mesh, such as by scaling and rotating. These manipulations can be in 3D, at no extra cost. Our experimental results showed that high quality reconstructed images can be generated in real-time.

We used the OpenGL GL-TRIANGLE-STRIP to build the triangle mesh. This routine needs all of the triangles to have the same orientation. Thus we started a new GL-TRIANGLE-STRIP whenever the diagonal direction changes. All of these strips were saved in a display list which was then used to render the image.

The program flow of the OpenGL process is as follows:

1. Build a byte array to record the diagonals of the triangles.
2. Set up all the GL-TRIANGLE-STRIP and save them in a display list.
3. Render the image and call an OpenGL loop, waiting for keyboard response and doing manipulation corresponding to the key pressed.

We have tested several images with size 768 $\times$ 512 pixels, in the same machine: an Intel Pentium 4 3G processor and an NVidia GeForce 4 graphics card with 128M memory. Using our extended method, the time for preparing the mesh for an image with 768 $\times$ 512 pixels was under 0.2 seconds. Once the triangle mesh was loaded, the graphics card did all further manipulation. We used key presses for scaling or rotation, causing the appropriate updates to the transformation matrix.

The GeForce 4 graphics card specification claims a rendering speed of 136 million vertices per second. This equates to about 45M independent triangles per second. This latter *rate* could increase with triangle strips (due to vertex sharing), though of course the *number* of triangles which can be rendered at full speed is limited by the card memory. With our test image meshes having less than 1M triangles, the graphics card easily gives real-time zooms, translations and rotations.

4. Other Applications

Figure 10 shows our extended method applied to three colour images chosen to include edge, texture and smooth features, magnified by a factor of 2.

Due to the simplicity of our algorithm, it is easy to apply in other ways. For example, we can rotate the image by any

angle (Figure 11 – top). We inverse rotate each target pixel back to the sample image and interpolate the value. We can also generate a perspective transform of an image. On any given y scan line, we calculate the pixel at (x, y) by sampling the source image at (sx, ty) where s, t are scale factors which vary linearly with height (We are assuming the y axis is the centre of the screen). Figure 11 (middle) shows the result. We can produce a magnifying lens effect (Figure 11 – bottom). If the lens has radius R, then its disc is filled with the image from a smaller disc with radius r at the same centre. For any pixel inside R, we scale down to r, evaluate the original value at r and apply it at radius R.

In general, these are variants on the same technique: to evaluate the target pixel p, we evaluate pixel $F(p)$ where F is a simple inverse mapping to the original image. Then we interpolate in the triangle where it falls.

5. Discussion

In this paper we have presented a new method of image interpolation. We represent an image as a data-dependent triangulation mesh. Every four-pixel square is divided into two triangles with the diagonal corresponding to the local edge of the image. The desired pixel can then be interpolated from the triangle in which it falls, determined by inverse mapping.

Other variants of the diagonal choice procedure can also be tried. For example, a pair of suitable digital filters might be better at distinguishing the local edge direction; or the threshold could be different to the one we chose. Other variants of the sampling procedure can be used, the interpolation providing some security against sampling defects. These two procedures are independent and neatly correspond to the image modelling and image rendering phases.

The new interpolation approach generates images with better visual quality than traditional interpolation schemes. The error assessment also shows that our scheme produces good overall image accuracy. The complexity of the new method is similar to bilinear interpolation and much lower than the bicubic method. We avoid the time-consuming optimisations that others use but still produce good results very quickly.

Our method has several advantages. It requires no iteration. It achieves arbitrary factor magnification, rotation, perspective transform and warp through a single mechanism. Our scheme is very simple to implement and computationally fast. It requires little data structure overhead to generate the mesh image. Moreover, our meshes can be rendered on a graphics card which makes real-time image reconstruction possible. There is a potential for our method to be used in gaming and image manipulation generally. We have also extended our model to an important commercial application: demosaicing of colour images (the reconstruction of a full-resolution colour image from the *mosaiced* sample generated by current single-chip digital cameras)[23]. We are inves-

Figure 11: *Top:Flower image rotated by 27 degrees. Middle: a perspective view of the flower image. Bottom: a lens effect of the flower image*

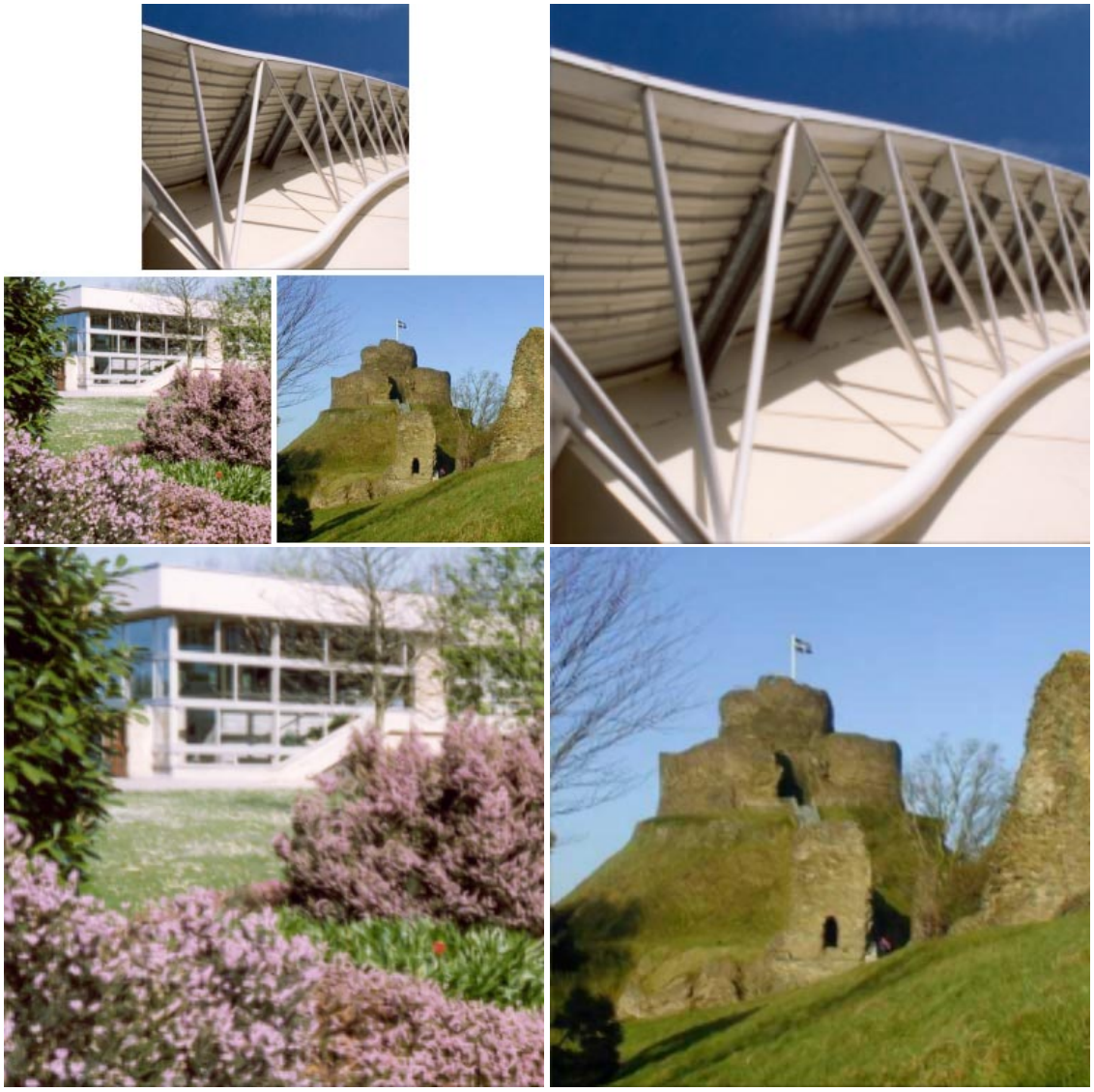


Figure 10: *Three sample images of* 192×192 *shown upper left. The corresponding larger images are magnified of a factor of 2, using our extended method.*

tigating its use in 4-colour separation for printing. Above all, we have demonstrated that a simple approach, sensibly used, can rapidly generate excellent results.

Acknowledgements

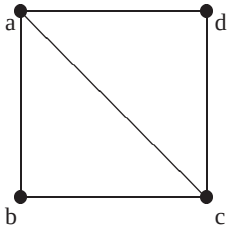
The authors would like to thank Dr. Xin Li at Sharp Labs of America at Camas, WA, USA for kindly providing his code for comparison tests and Dr. P.W. Wong at IDzap, USA for providing the flower image shown in this paper. We would also like to thank our colleagues Dr Peter Hall and Dr Man Qi for their suggestions and discussion. The anonymous referees gave excellent advice which has greatly improved the paper. We are grateful to Professor Ken Brodlie, at the University of Leeds UK, for long ago pointing out to one of us the distinction between true bilinear interpolation and what we here call triangular interpolation. We also thank Dr. Steven Ruzinsky for discussions and suggestions about interpolation. The work is funded under EPSRC project *Quasi-3D Compositing and Rendering* and a UK-funded *Overseas Research Scholarship* funded by Universities UK.

References

1. A.N. Netravali and B.G. Hasskell, "Digital Pictures: Representation, Compression and Standards", 2nd Ed., New York:Plenum Press, 1995 1
2. M. Unser, A. Aldroubi and M. Eden, "Fast B-Spline Transforms for Continuous Image Representationa and Interpolation", *IEEE TRans. Pattern Anal. Mach. Int.*, Vol. 13, No. 3, pp. 277-285, 1991 1
3. B. Ayazifar and J.S. Lim, "Pel-adaptive model-based interpolation of spatially subsampled images", *Proc. of Intl. Conf. on Acoust. Speech and Signal Processing*, Vol. 3653, pp. 181-184, 1992 1
4. K. Ratakonda and N. Ahuja, "POCS based adaptive image magnification", *Proc. of Intl. Conf. on Image Processing*, Vol.3, pp. 203-207, 1998 1
5. S.A. Martucci, "Image Resizing in the Discrete Cosine Transform Domain", *Proc. Int. Conf. Image Processing*, Vol.2, pp. 244-247, 1995 1
6. E. Shinbori and M. Takagi, "High Quality Image Magnification Applying the Gerchberg-Papoulis Iterative Algorithm with DCT", *Systems and Computers in Japan*, Vol. 25, No. 6, pp. 80-90, 1994 1
7. S.D. Bayrakeri and R.M. Mersereau, "A New Method for directional Image Interpolation", *Proc. Int. Conf. Acoustics, Speech, Sig. Process*, Vol.4, pp. 2383-2386, 1995 1
8. K. Jensen and D. Anastassiou, "Subpixel Edge Localization and the Interpolation of Still Images", *IEEE Trans. Image Process*, Vol. 4, No. 3, pp. 285-295, 1995 1
9. N.B. Karayiannis and A.N. Venetsanopoulos, "Image Interpolation Based on Variational Principles", *Signal Process*, Vol. 25, pp. 259-288, 1991 1
10. R.R. Schultz and R.L. Stevenson, "A Bayesian Approach to Image Expansion for Improved Definition", *IEEE Trans. Image Process*, Vol.3, No.3, pp. 233-242, 1994 1
11. X. Li and M. Orchard, "New Edge Directed Interpolation", *Proc. IEEE Int. Conf. Image Processing*, Vol. 2, pp. 311-314, 2000 1, 2
12. J. Allebach and P.W. Wong, "Edge-directed interpolation", *Proc. IEEE Int. Conf. Image Processing*, Vol. 3, pp. 707-710, 1996 1, 2
13. S. Battiato, G. Gallo, F. Stanco, "A locally-adaptive zooming algorithm for digital images", *Image and Vision Computing*, Vol. 20, No. 11, pp. 805-812, September 2002 1, 2
14. B.S. Morse and D. Schwartzwald, "Isophote-based interpolation", *Proc. IEEE Int. Conf. Image Processing*, Vol. 3, pp. 227-231, 1998 1, 2
15. B.S. Morse and D. Schwartzwald, "Level-Set Image Reconstruction", *Proc. Computer Vision and Pattern Recognition 2001 (CVPR'01)*, pp. 333-340, IEEE 2001 2
16. X. Yu, B. Morse, T.W. Sederberg, "Image Reconstruction Using Data-Dependent Triangulation", *IEEE Computer Graphics and Applications*, Vol. 21 No. 3, pp. 62-68, May/June 2001 2, 5
17. N. Dyn, D. Levin, S. Rippa, "Data Dependent Triangulations for Piecewise Linear Interpolation", *IMA Journal of Numerical Analysis*, Vol. 10, pp. 127-154, Institute of Mathematics and its Applications, 1990 2
18. Van Essen DC, Anderson CH, and Felleman DJ, "Information Processing in the Primate Visual System: An Integrated System perspective", *Science*, Vol. 255, No. 5043, pp. 419-423, 1992 1
19. E. Quak and L.L. Schumaker, "Cubic spline interpolation using data dependent triangulations", *Comput. Aided Geom. Design*, Vol. 7, pp. 293-301, 1990 5
20. N. Damera-Venkata, T.D. Kite, W.S. Geisler, B.L. Evans and A.C. Bovik, "Image Quality assessment based on a degradation model." *IEEE Transactions on Image Processing*, Vol. 9, pp. 636-650, 2000 7
21. S. Daly, "The visible differences predictor: An algorithm for the assessment of image fidelity." *Digital Images and Human Vision*, A.Watson, Ed. Cambridge, MA:MIT Press, 1993 7
22. Robert Keys, "Cubic Convolution Interpolation for Digital Image Processing", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. 29, No. 6, pp.1153-1160, 1981 9
23. D. Su and P.J. Willis "Demosaicing of Colour Images Using Pixel Level Data-Dependent Triangulation" *Proc. Theory and Practice of Computer Graphics (TPCG 2003)*, pp. 16-23, 2003, IEEE. 10

Appendix A: Proof

Consider a four pixel square *abcd*. We will first prove that, if pair *ac* has smaller difference than *bd*, then *b* or *d* is the outlier pixel and we should connect *ac*. That is to say, if $|a-c| < |b-d|$ then *b* or *d* is either the biggest or the smallest pixel.



Suppose $|a-c| < |b-d|$, and suppose $a \geq c$, then $a-c < |b-d|$.

1. Suppose $b > d$. Then $a-c < b-d$ $(b > d, a \geq c)$, hence $a-b < c-d$ $(b > d, a \geq c)$.
   We suppose $a > b$ and $c < d$, then $a-b > 0$ and $c-d < 0$, so we get $a-b > c-d$. However, we have the formula $a-b < c-d$ before which means our assumption that $a > b$ and $c < d$ is wrong.
   Because $a > b$ and $c < d$ is wrong, either $a < b$ or $c > d$ or $a < b, c > d$ with the condition $(b > d, a \geq c)$. In these cases, either b is the biggest pixel $(b > a, b > c, b > d)$ or d is the smallest pixel $(d < c, d < a, d < b)$.
2. Suppose $b < d$, then $a-c < d-b$ $(b < d, a \geq c)$, hence $a-d < c-b$ $(b < d, a \geq c)$.
   We suppose $a > d$ and $c < b$. Then $a-d > 0$ and $c-b < 0$, so we get $a-d > c-b$. However, we have the formula $a-d < c-b$ before which means our assumption that $a > d$ and $c < b$ is wrong.
   Because $a > d$ and $c < b$ is wrong, either $a < d$ or $c > b$ or $a < d, c > b$ with the condition $(b < d, a \geq c)$. In these cases, either b is the smallest pixel $(b < c, b < a, b < d)$ or d is the biggest pixel $(d > b, d > a, d > c)$.

We have proved that if pair *ac* has the smaller difference ($|a-c| < |b-d|$), there are two situations. One is that either b is the biggest pixel or d is the smallest pixel. The second is that either b is the smallest pixel or d is the biggest pixel. In either case the outlier is either b or d and *ac* should be the edge. Using the same method we can prove that if pair *bd* has the smaller difference ($|b-d| < |a-c|$), the outlier is either a or c and *bd* should be the edge.

So we can conclude that drawing the edge between the least-different diagonal pair gives the same result as drawing the edge which isolates the outlier.