UNIVERSITY OF WATERLOO
COMPUTER SCIENCE DEPARTMENT

Using B-Splines for Re-Sizing Images

R. Victor Klassen
Richard H. Bartels

CS-86-55

November, 1986

Using B-Splines for Re-Sizing Images

R. Victor Klassen

Richard H. Bartels

University of Waterloo
Department of Computer Science
Computer Graphics Laboratory
Waterloo, Ontario
N2L 3G1

Abstract

Raster-image pixels are used as control vertices to construct B-spline surfaces. The resulting surfaces may be sampled at different densities to provide raster images at different resolutions. This can be made efficient, and it has good filtering behaviour. When the re-sampling density is non-uniform, various distortion effects can be achieved.

Contents

Introduction

Consider the problem of re-sizing an image, which we assume to be presented as a raster file of some width and height that is to be reproduced at some new width and height. One could merely use pixel replication for enlargement – this is implemented in hardware on some frame buffers – and subsampling for reduction. The problem with such simple techniques is that they may leave the new image much more aliased than was the original. It would be preferable to provide some filtering while changing the size of the original image.

One can think of the pixel values in the original raster file as a discrete sampling, $V_{i,j} = V(x_i, y_j)$, of a height field, $V(x, y)$, which for lack of other information will be assumed continuous between the sample points. In effect, by making this assumption we ignore the existence of edges in the image, but as our examples will show, the consequences of doing so are not serious. The basic idea to be presented is that of constructing a spline surface from the pixel data to "recover," approximately, the original height field as a spline surface $S(x, y)$. This will provide us with a representation that can be resampled at any density, giving a new picture of greater or lesser detail than the original raster image.

It is possible to use interpolation to construct the spline surface. One problem with doing this, however, is that interpolation is computationally intensive. Another is that maxima and minima of an interpolating spline are not necessarily bounded by the maxima and minima of the sample points, which can cause problems when pixel values exist at the upper and lower limits of the displayable range. In addition to this, for any sufficiently smooth interpolation, the value of any given pixel has an influence on the entire surface, which may not be desirable.

The alternative that we propose involves using the pixel values as control vertices to construct the surface from tensor-product B-splines. Not only is this a fast method of approximation, but it also is *local*, *variation diminishing*, and possesses the *convex hull property*. The locality guarantees that individual pixel values have an influence over a restricted area of the spline surface; the convex hull property guarantees that each surface value will fall between the extremal pixel values, and the variation-diminishing nature of this method of approximation will provide some damping of high-frequency components in the image during re-sizing.

More precisely, the surface can be constructed and sampled in time proportional to the sum of the sizes of the input and output images. The time required to compute a sample value in a region of the surface influenced by a given number of pixel values is proportional to that number. The number of pixels of the original raster that influence a region is given by the orders of the B-splines making up the tensor product. For the typical choice of bicubic splines, for example, no point on the surface is influenced by more than 16 pixels, arranged in a 4×4 pattern. The variation-diminishing property implies that high frequency information in the original image is reduced. Indeed, if the image is "enlarged" by a factor of one, the result is the same as filtering using the repeated box kernel of [Heckbert1986].

As in the work just cited, the method presented here can be generalised to arbitrary filter kernels of finite size, in place of using B-splines as filter kernels, though some of the properties of B-splines will be sacrificed. The summed-area method used in the cited work reduces the cost slightly over that in our proposed technique, however space limitations can make that method intractable. In rendering and processing very large (4096×4096) images, scanline algorithms are the only choice, since the entire image does not fit in the memory of most processors in common use.

Splines

A B-spline curve of order k with knots at integral values $\lfloor \bar{u} \rfloor$ of the parameter $\bar{u}$ may be computed from the following formula:

$$S(u) = \sum_{i=0}^{k-1} b_{-i}(u)\, V_{\lfloor \bar{u} \rfloor - i + 1}, \qquad (1)$$

where

$$u = \bar{u} - \lfloor \bar{u} \rfloor,$$

and where the functions b_{-i} are the segment polynomials of the uniform B-splines of order k ; e.g., in the case when $k = 4$ (cubics):

$$b_{-0}(u) = \frac{1}{6} u^3$$

$$b_{-1}(u) = \frac{1}{6} (1 + 3u + 3u^2 - 3u^3)$$

$$b_{-2}(u) = \frac{1}{6}(4 - 6u^2 + 3u^3)$$

$$b_{-3}(u) = \frac{1}{6}(1 - 3u + 3u^2 - u^3).$$

The notation follows that used in [Bartels1987]. In two dimensions the above is simply iterated for the second dimension to produce a tensor-product surface. A different order, ℓ, may be used for the second dimension, yielding:

$$S(u,v) = \sum_{j=0}^{\ell-1} b_{-j}(v) \sum_{i=0}^{k-1} b_{-i}(u) V_{\lfloor \bar{u} \rfloor - i + 1, \lfloor \bar{v} \rfloor - j + 1}, \quad (2)$$

where $v = \bar{v} - \lfloor \bar{v} \rfloor$. In this way the bivariate spline, S, is defined over a rectangular domain of much the same extent as the domain of V. S approximates the pixel values, $V_{i,j}$, at integral values of its arguments, and it fills in the gaps between the pixel values in a smooth manner (C^{k-2} with respect to $\bar{u}$ and $C^{\ell-2}$ with respect to $\bar{v}$). Changing the size of an image is achieved by sampling S at any grid-oriented sampling of values for $\bar{u}$ and $\bar{v}$, yielding a new raster image $V'_{i,j}$.

Equation (2) can be viewed as a one-dimensional formula with "derived" control vertices:

$$\sum_{i=0}^{k-1} b_{-i}(u) W_{\lfloor \bar{u} \rfloor - i + 1}$$

where

$$W_{\lfloor \bar{u} \rfloor - i + 1} = \sum_{j=0}^{\ell-1} b_{-j}(v) V_{\lfloor \bar{u} \rfloor - i + 1, \lfloor \bar{v} \rfloor - j + 1}.$$

If the usage of this formula is on a scanline-by-scanline basis, as it is in our proposal, these derived control vertices need be computed only once per scanline.

Domains

Control-vertex approximation by B-splines allows some freedom in selecting end conditions; e.g., see [Bartels1987]. We have chosen to fix the intensities around the border of the new image to the same values as in the original image. This dictates that the bordering pixels be treated as if they were replicated outward to infinity (or as far as the spline of the chosen order requires). Using these end conditions, it is desirable that the input and output images be aligned at both ends (symmetrically). Arranging to get the end conditions met correctly is subtle enough that it is worth discussing here in detail. To simplify, we will use the one-dimensional setting and splines of order k . The discussion will be incorrect when $k = 1$; i.e., for constant splines (step functions). However, since this corresponds to subsampling and pixel replication, there is no need to use spline techniques.

Suppose that the input raster consists of m pixel values

$$V_0, \ldots, V_{m-1} .$$

The raster is considered to have an extension by k-1 pixels to the left of the range, for which the pixel value V_0 will be replicated, and to have an extension by k-1 pixels to the right of the range, for which the pixel value V_{m-1} will be replicated. On each interval $i \leq \bar{u} < i+1$, the value of S will have the value

$$b_{-0}(\bar{u}-i)\, V_{i+1} + \cdots + b_{-k+1}(\bar{u}-i)\, V_{i-k+2} .$$

When $\bar{u} = i$, b_{-0} will be zero, and the spline will be an average of the values $V_i, \ldots, V_{i-k+2}$. Similarly, when $\bar{u} = i+1$, b_{-k+1} will be zero, and the spline will be an average of the values $V_{i+1}, \ldots, V_{i-k+3}$. Because of the extensions to the left and right, this will have the result of making S equal to V_0 when $\bar{u} = 0$ and equal to V_{m-1} when $\bar{u} = m+k-3$. This determines the range of $\bar{u}$ values over which the spline S will approximate the original m pixels. The parameter $\bar{u}$ is considered to vary from 0 on the left to $m+k-3$ on the right to cover the distance from V_0 to V_{m-1}. In Figure 1 below we take the specific case of $m = 7$ and $k = 4$ (a cubic spline). Note that the range of $\bar{u}$ corresponding to $V_0, \ldots, V_6$ is $0 \leq \bar{u} \leq 8$.

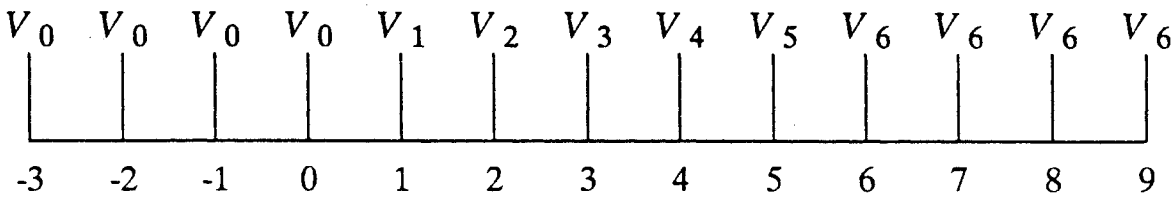

Figure 1

Precomputing the basis functions

Since S is only to be evaluated at a finite number of discrete values of its arguments, clearly b_i and b_j may be precomputed at those values of $u = u^{(r)}$, $v = v^{(s)}$ for which S is required to produce the new image V'. If b_j or b_i were only required once for a particular value of $u^{(r)}$ or $v^{(s)}$, no saving would be obtained. For each row and column of the output image, however we expect the same values of the basis polynomials to be re-used, so we consider how they may be precomputed.

In the one dimensional case, if V is given by m pixel values, yielding a range of 0 to $M = m + k - 3$ for $\bar{u}$ as described above, and if V' is to be defined on N points, then $u^{(r)}$ will take on a well defined set of values:

$$u^{(r)} = \text{frac}\left[r \frac{(N-1)}{(M-1)} \right] \quad r = 0,1, \cdots N-1. \qquad (3)$$

Unless $N-1$ and $M-1$ are mutually prime, $u^{(r)}$ will take on less than $N-1$ distinct values. For the purposes of illustration, suppose that we continue using the above example with 7 input pixels and cubics, yielding the effective range of 9 pixels for the spline, S, and suppose 11 pixels are required for output, as in Figure 2 below. We require that the 0th output pixel be aligned with V_0, and that the 10th output pixel be aligned with V_6, which is taken on when $\bar{u} = 8$. It follows that output pixels 0,1,2,3... are aligned with values 0, 4/5, 8/5, 12/5... of $\bar{u}$.

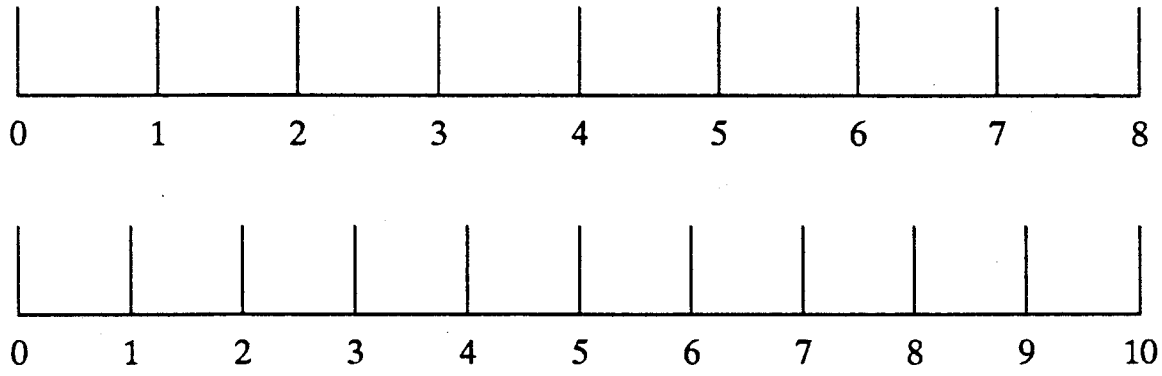

Figure 2

In this case the only values of $u^{(r)}$ which occur are 0/5, 4/5, 3/5, 2/5 and 1/5. In general, the values of $u^{(r)}$ may be found as follows: let $G = GCD\ (M-1\ ,\ N-1\ )$; in our example $G = 2$. Then there are $(N-1)/G$, in this case $10/2 = 5$ distinct values of $u^{(r)}$, evenly spaced through the $[\ 0\ ,\ 1\ )$ interval.

In the one-dimensional case this precomputation gives no improvement over evaluating the segment polynomials on the fly; however, in the two-dimensional case each value of $u^{(r)}$ will occur on every row, and every value of $v^{(s)}$ will occur for every column. Here $O(N_{\bar{u}}\ N_{\bar{v}}\ )$ multiplications will be spared for an $N_{\bar{u}} \times N_{\bar{v}}$ output image at the cost of $O(N_{\bar{u}} + N_{\bar{v}}\ )$ multiplications at the start.

At this point it is interesting to note that the above computation does not depend on M being less than N. This implies that the same computation may be applied to image reduction as is applied to enlargement. Aliasing artifacts may still result from the subsampling involved, but the reduction capability is available "free" and, if a spline of order higher than 1 is used, the aliasing is less than that obtained by using a simple subsampling technique, since the averaging window is more than one pixel wide. Aliasing may be further reduced using the stage-wise reduction method described later.

The Enlarging Algorithm

Determine the input and output dimensions and the spline order.
Compute the basis functions where required.

Initialise ring buffer of scanlines, including any replicated lines.

for(outRow ← 0; outRow < output height; outRow++)

Adjust ring buffer so the necessary input scanlines are in place.

Compute derived control vertices.
Apply endpoint multiplicity to the control vertices.
Compute output scanline from the control vertices.
Write out the scanline.

The choice of the order of the spline is a compromise. Lower order splines are faster to compute, but produce more aliasing artifacts. High order splines take longer and have a tendency to blur the image. In most cases the best choice is either a quadratic (order 3) or cubic (order 4) spline in each dimension of the two. Figure 3 shows most of an enlargement of a 128x124 image to 170x170 using each of the first four splines. An order-one spline (corresponding to pixel replication) is almost never good enough, as is clear from the image in the top left. In this case the best choice is either linear (top right image) or quadratic approximation (bottom left). Cubic approximation is shown in the image on the bottom right. Figure 4 shows a section of each of four enlargements made by the same methods but with final resolution 512x512. Here linear interpolation is definitely not good enough, and the fourth order spline is probably the best choice.

Reducing

The most obvious way to reduce image size is to apply the enlarging algorithm with output dimensions smaller than input dimensions. Since the filter kernel averages together k pixels, this will result in some input data being totally ignored if the ratio of the output to the input dimensions is greater than k. To overcome this problem, the reduction should be performed in stages. To go from input width M to output width N ($< N / k$), compute $L = \log_k (M / N)$, then let f be k raised to the fractional part of L (f will be less than k). Reduce the original image by a factor of f, and then reduce the result (recursively) by a factor of k, L times. In this way, at no

time will an image be reduced by more than a factor of k. The smaller reduction should be performed first to minimise the loss of information. Figure 5 shows an original 420x420 image, along with successive reductions to 128x128 and 32x32 and 8x8. The 8x8 image of the cat is just discernable in the top left-hand corner.

Non-Uniform Sampling.

If, rather than being sampled on the grid, the surface is sampled at points that derive from some function of output grid position, a distorted image results. Within some restrictions, this may be done in nearly the same time as filtering or re-sizing the image.

Let $f : (x, y) \rightarrow (\bar{u}, \bar{v})$, the function mapping pixel locations to surface points be separable into $g(x)$, and $h(y)$. And further restrict h to be monotonicly increasing. With these restrictions on f, we can, with very little modification to the re-sizing algorithm, produce a distorting algorithm.

In each column the spline is evaluated at an arbitrary value of $\bar{u}$, but for each row the value of $\bar{u}$ is constant. Similarly, in each column $\bar{v}$ is held fixed. Pre-computing the basis functions and the knot indices as given by g and h can be done in time proportional to the output image perimeter. The economy of finding the greatest common divisor and thence the number of points at which the spline must be evaluated is lost, but the precomputation is still fairly inexpensive.

The requirement that h be monotonic and increasing preserves the scanline nature of the algorithm. Except for the computation of the knot indices and the basis functions the algorithm is exactly the same as that presented for enlarging and contracting images.

The requirement that f be separable into g and h is only necessary for the efficiency described here. Arbitrary distortions of the input image are possible if one is prepared to compute every B-spline value on the fly and keep the entire input image in memory.

Figure 6 shows the result of applying such a distortion to the image shown (contracted) in its upper left-hand corner. Figure 7 is a somewhat more extreme distortion of the same original.

Summary

An efficient method for re-sizing raster images has been presented. The complexity is of the order of the sum of the number of input and output pixels processed. The cost per pixel processed is that of a few multiplications and several additions, as well as the overhead of reading the input and writing the output. The images shown here were computed in approximately 35 microseconds per pixel processed, including approximately 10% overhead. Computation was performed on a VAX 8600.

The algorithm is not restricted to aspect-ratio preserving transformations, nor is it restricted to uniform size changes throughout an image. The use of non-uniform size changes can produce images reminiscent of those seen in distorted mirrors.

Acknowledgements

The re-sizing algorithm was implemented as another tool in the **im** package, [Paeth1986] most of which was written by Alan Paeth. His picture appears in Figure 6. The cat is courtesy of Stewart Kingdon. We have had helpful discussions with David Martindale of the French Animation section of the National Film Board of Canada. DEC hardware has been provided through the WATDEC Research Agreement between the University of Waterloo and the Digital Equipment Corporation. Other hardware has been acquired through a BILD Grant from the Province of Ontario. Work in the Computer Graphics Laboratory is supported by Operating, Infrastructure, and Strategic grants from the National Sciences and Engineering Research Council of Canada.

References

Bartels1987 — Bartels, R.H., Beatty, J.C., and Barsky, B.A., *An Introduction to the Use of Splines in Computer Graphics*, Morgan–Kaufmann, Palo Alto, California (1987).

Heckbert1986 — Heckbert, P.S., Filtering by Repeated Integration, *Computer Graphics* 20(4) pp. 315-321 (August 1986).

Paeth1986 — Paeth, A.W., Design and Experience with a Generalized Raster Toolkit, *Proceedings of Graphic Interface '86*, pp. 91-97 Canadian Information Processing Society, (May 1986).

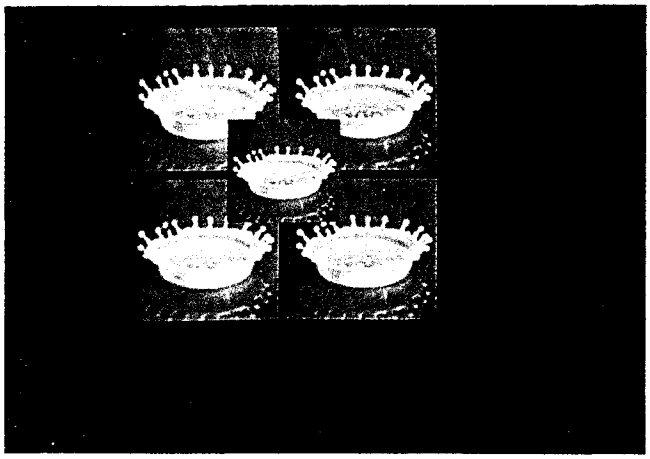

Figure 3

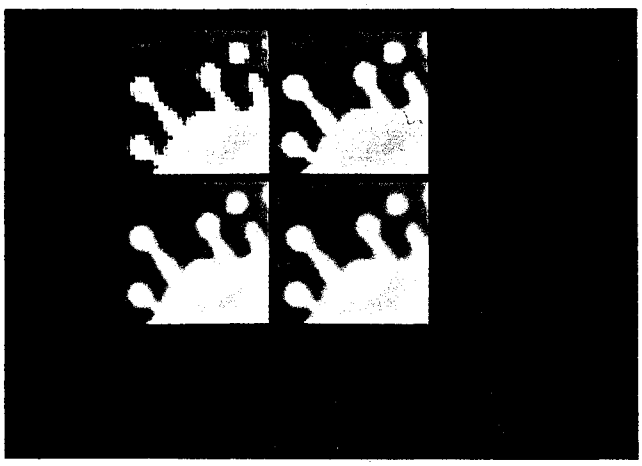

Figure 4

Figure 5

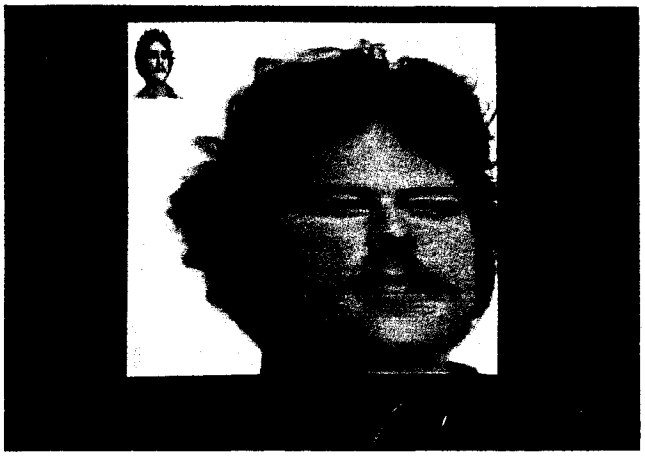

Figure 6

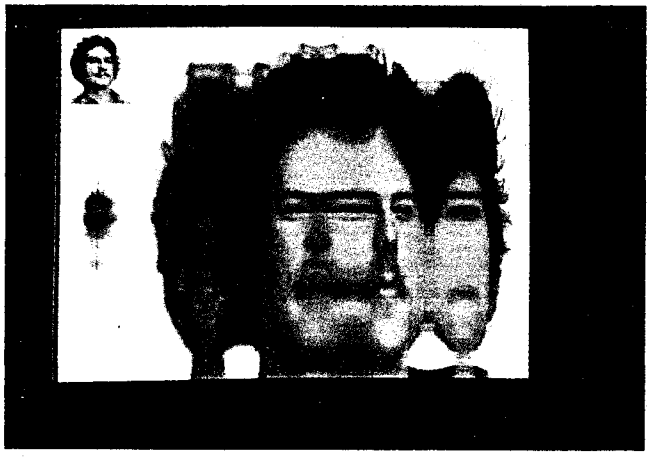

Figure 7