The Visual Computer

Edge-preserving texture filtering for real-time rendering

Yuan-Chung Lee,
Chein-Wei Jen

Department of Electronics Engineering, National Chiao Tung University, Hsinchu, Taiwan, ROC
E-mail: {yzlee, cwjen}@twins.ee.nctu.edu.tw

Published online: 28 January 2003

Texture filtering is essential in enhancing the visual quality of real-time rendering. Conventional schemes do not consider the characteristics of texture content, thus the sharpness of edges in texture images cannot be retained. This paper proposes a novel texture-filtering algorithm, which consists of edge-preserving interpolation and edge-preserving MIP-map prefiltering. The memory bandwidth requirement is kept the same as in conventional schemes by dynamically adjusting the interpolation kernel. Hardware implementation is also provided to show the real-time processing capability.

Key words: Filtering – Resampling – Texture mapping

1 Introduction

Texture mapping is an important operation to enhance photorealism in high-quality computer graphics. It maps image patterns onto 3-D surfaces to improve visual details. After the mapping transformation, texture filtering resamples the texels to prevent aliasing artifacts. Heckbert (1986) provided a comprehensive survey of various texture filtering algorithms for 2-D texturing. The direct convolution filters (Blinn and Newell 1976; Feibush et al. 1980; Gangnet et al. 1982) use a weighted average of the texels corresponding to each screen pixel. Greene and Heckbert (1986) proposed the elliptical weighted average (EWA) filter to reduce the computational cost while maintaining quality. These algorithms, however, suffer from high memory bandwidth consumption, because the texture region mapped to the corresponding screen pixel may be arbitrarily large.

Prefiltering is an effective means of reducing the runtime memory access. Summed-area tables (Crow 1984) store two-dimensional preintegrated results, but take two to four times as much memory. MIP-map image pyramids (Williams 1983) combined with bilinear and trilinear interpolation are the most widely used schemes. MIP-map prefiltering serves as precomputed texture minification and is very efficient for real-time rendering. However, bilinear and trilinear filtering is limited in that the filter kernels are square only and cannot simulate an anisotropic effect. In recent years, the development of texture filtering has been focused on anisotropic filtering. NIL-maps (Fournier and Fiume 1988) store the preconvolution of basis functions to yield constant-time filtering, but the memory and computational cost is extremely high. The clustering technique (Demirer and Grimsdale 1994) uses precalculated cluster maps to decrease the access time. Footprint assembly (Schilling et al. 1996) approximates an anisotropic region by a number of square and MIP-mapped texels. The technique called Feline (McCormack et al. 1999) computes a more appropriate length for the sampling line of an anisotropic region. Fast footprint MIP-mapping (Huttner and Strasser 1999) provides a scalable footprint to respect the memory bandwidth of a graphics system. Texture potential MIP-mapping (Cant and Shrubsole 2000) presums texels on one axis and accumulates them dynamically on the other axis. A low-cost anisotropic filter may also be implemented by averaging multiple isotropic regions (Ewins et al. 2000). These filter kernels match the

Digital Object Identifier (DOI) 10.1007/s00371-002-0169-8

desired projection of the screen pixels in texture space more accurately than bilinear and trilinear filtering.

The above methods attempt to approximate the ideal sampling theory to avoid aliasing; nevertheless, they still result in blurred edges. Their finite-duration kernels with band-limited input signals cannot reconstruct images with very sharp edges. However, it is known that edges carry the most perceptible characteristics in human vision (Hubel 1988). The above methods do not consider the content of texture images, so they involve no special process to handle the edges. Moreover, closer observation of a 3-D surface reveals that the selected MIP-map level is out of the MIP-map pyramid. In such a case, the above methods degenerate into the traditional bilinear or simplified bicubic interpolation, which produces excessively blurred or jagged edges. Although data-dependent triangulation (Yu et al. 2001) breaks texture images into a mosaic of triangles according to the texture contents, it generates a huge amount of triangles, which are not efficient for real-time rendering.

In order to develop an edge-preserving texture-filtering algorithm for real-time rendering, several requirements should be met.

1. Arbitrary resizing. The filtering should be able to resample the texture into an arbitrary size based on texture coordinates.
2. Small memory bandwidth consumption. The consumption should be as small as possible since memory bandwidth is the current bottleneck of graphics hardware.
3. Real-time processing. The hardware should cope with the pixel rate demanded by state-of-the-art graphics hardware at moderate cost.

A novel edge-preserving texture-filtering algorithm, which meets the above requirements for real-time rendering, is proposed in this paper. This filtering consists of real-time edge-preserving interpolation and off-line edge-preserving MIP-map generation. The interpolation can perform arbitrary up-sampling as well as produce less-jagged edges. The interpolation kernel can be adjusted dynamically according to the texture cache status. The MIP-map preminification is also conducted by edge-preserving method to enhance visual sharpness. The hardware design has been implemented to prove its real-time capability.

2 Edge-preserving interpolation

2.1 Previous works

Several edge-preserving interpolation algorithms have been developed for image processing and video de-interlacing. In image processing, Ting and Hang (1997) and Michaud et al. (1997) utilized fuzzy inference. Carrato and Tenze (2000) applied a rational operator with optimization of parameters. Jensen and Anastassiou (1995) exploited sub-pixel edge localization. Cubic spline-under-tension (Xue et al. 1992), a quadratic Volterra filter (Thurnhofer and Mitra 1996), and rank-order filtering (Algazi et al. 1991) have also been introduced to perform edge-preserving interpolation.

In de-interlacing, the edge-based line average (ELA) algorithm (Doyle and Looymans 1990) reconstructs the missing lines by interpolating along the highest correlation of three directions in a 2×3 window. Salonen (1994) extended the ELA algorithm to a $2 \times N$ interpolation kernel for detecting more directions. Kuo et al. (1996) improved the quality of horizontal edges in the ELA algorithm. Chen et al. (2000) and Lee et al. (2000) incorporated the ELA algorithm with some checking rules to increase the accuracy of interpolation. In addition, a directional-correlation dependent interpolation filtering (DIF) algorithm (Lee et al. 1994) solves the defect of the ELA algorithm that occurs when the signals have components with a high horizontal frequency. The DIF algorithm simply splits the frequency bands into two parts. The ELA-like interpolation is used for the low-frequency part, and the simple line duplication is adopted for the high-frequency part. Kang (2000) also recommended this frequency division concept.

Most of the edge-preserving interpolation schemes described above only double the resolution of images and cannot perform arbitrary resizing. Hence, we propose a novel edge-preserving interpolation scheme to overcome this problem. Our algorithm utilizes the band-split concept of the DIF algorithm but modifies the interpolation techniques for each frequency band. This band-split approach is excellent at preventing high-frequency interference, which many other edge-preserving interpolation schemes fail to do.

2.2 The proposed interpolation

In the processing of texture mapping, the texture coordinates of each pixel inside a polygon are in-

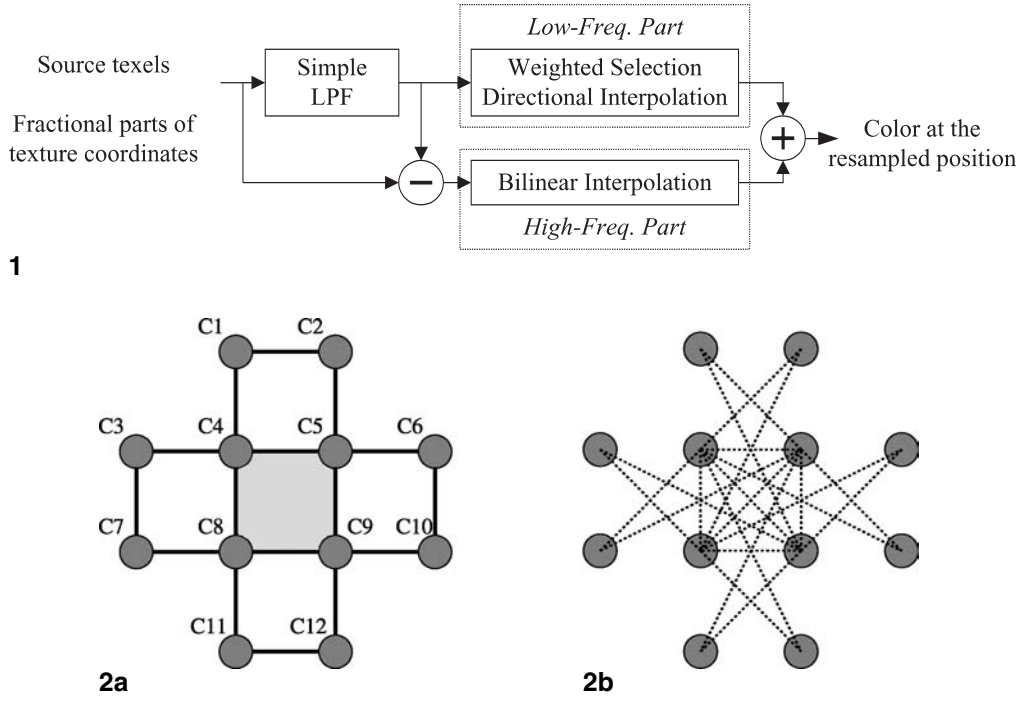


Fig. 1. Block diagram of the edge-preserving interpolator
Fig. 2a,b. Interpolation kernel style: **a** source grid; **b** directional correlations

terpolated in a perspective-projection manner. The integer parts of the texture coordinates indicate the texel address where the screen pixel is mapped in the texture source grid. The decimal fraction parts control the resampling procedure. The interpolation generates the output color at the resampled position based on the decimal fractions, just as in the bilinear interpolation of the conventional scheme.

The block diagram of the proposed edge-preserving interpolator is shown in Fig. 1. The source texels located around the resampled position are sent to the interpolator. The simple low-pass filter first extracts the low-frequency signals from the source texels. These signals are subsequently interpolated by the weighted selection directional interpolation module. The high-frequency signals are obtained by subtracting the low-frequency signals from the source texels. Bilinear interpolation is adopted for high-frequency signals. Finally, the color at the resampled position is generated by merging the results of these two parts.

Figure 2a shows the interpolation kernel style, which comprises 12 source texels. The sample to be interpolated is located in the center square of the kernel. The fractional parts of texture coordinates control the exact resampled position. Figure 2b illustrates the directional correlations, which are used by the weighted selection directional interpolation module. The measured directions are shown as dashed lines. In this module, the interpolation takes place along the direction of the smallest color difference, i.e., along the edge. The choice of interpolation direction should also take into account the fractional parts of the texture coordinates. The influence of this direction is less significant if the resampled position is farther from the measured direction. Hence, we attach a weighting to each measurement of color differences based on the distances.

Let x_f and y_f be fractions of texture coordinates, and ΔP_i be the color differences in the measured directions. The interpolation direction is chosen as

$$\min(w_i \times \Delta P_i), \quad i \in \text{all measured directions}, \qquad (1)$$

where the weighting is

$$w_i = \frac{1}{\left\| 1 - \frac{d_i}{ROI_i} \right\|}. \quad (2)$$

d_i is the Euclidean distance from the resampled position to the line of each measured direction and is calculated according to x_f and y_f. The *ROI* (region of influence) is the maximum distance that the corresponding measured direction can affect. The symbol $\|x\|$ means that w_i should be set as a very large number if the value x is negative or zero, so that this direction is not chosen for interpolation. A very small threshold should be adopted for implementing the $\|x\|$ operation to prevent overflow due to finite precision. As a result, the corresponding measured direction is not taken into consideration if d_i is larger than ROI_i.

In our simulation, *ROI* is set to 0.5 for the horizontal and vertical measured directions, $\sqrt{2}/4$ for the diagonals, and $1/2\sqrt{5}$ for the 2×1 directions (26.6°, 63.4°, 116.6° and 153.4°). These *ROI* settings are determined by the halves of distances from the lines of measured directions to the nearby corners of the center square. This means that more important directions (horizontal, vertical and diagonals) have a greater influence. The total number of measured directions is 22. The diagonals of the four outer squares of the kernel grid are also included, because they influence the regions near the four corners of the center square. However, the 3×1 directions (18.4°, 71.6°, 108.4° and 161.6°) are not utilized, because their texels are too widely spaced, possibly resulting in high-frequency distortion. Finally, after the decision of interpolation direction is made, the color at the resampled position is generated by linear interpolation along the selected direction. As shown in Eq. (3), C_p and C_q are the low-frequency colors of the source texels on the line of the selected direction. The interpolated color of the low-frequency part is the linear blending of these two colors according to one of its fractional coordinates that ranges from 0 to 1 in the selected direction.

$$C_{\text{Low}} = (1-t) \times C_p + t \times C_q, \quad t \in x_f \text{ or } y_f. \quad (3)$$

The simple low-pass filter is realized with an asymmetrical median filter. Three-point median filters are applied to generate the low-frequency signals of the eight boundary texels, while five-point median filters are applied to generate the low-frequency signals of the four central texels. Thus, no extra memory access is imposed for the low-pass filtering beyond the original 12 source texels. Furthermore, the median filter is superior to weighted convolution in low edge blurring. The detailed equation is shown in Eq. (4).

$$\begin{aligned}
C1_{\text{low}} &= median(C1, C2, C4)\\
C2_{\text{low}} &= median(C1, C2, C5)\\
C3_{\text{low}} &= median(C3, C4, C7)\\
C4_{\text{low}} &= median(C1, C3, C4, C5, C8)\\
C5_{\text{low}} &= median(C2, C4, C5, C6, C9)\\
C6_{\text{low}} &= median(C5, C6, C10)\\
C7_{\text{low}} &= median(C3, C7, C8)\\
C8_{\text{low}} &= median(C4, C7, C8, C9, C11)\\
C9_{\text{low}} &= median(C5, C8, C9, C10, C12)\\
C10_{\text{low}} &= median(C6, C9, C10)\\
C11_{\text{low}} &= median(C8, C11, C12)\\
C12_{\text{low}} &= median(C9, C11, C12).
\end{aligned} \quad (4)$$

The high-frequency source signals are the differences between source texels and the low-frequency signals generated by Eqs. (4). These differences can be either positive or negative. Bilinear interpolation is the operation used for high-frequency processing, as shown in Eq. (5). The four weighting coefficients are first calculated according to (x_f, y_f). The four high-frequency signals are then multiplied by the weighting coefficients and summed up as the high-frequency result.

$$\begin{aligned}
C_{\text{High}} = {} & (1-x_f)(1-y_f)H4 + x_f(1-y_f)H5\\
& + (1-x_f)y_fH8 + x_fy_fH9.
\end{aligned} \quad (5)$$

The overall interpolation I is the combination of low- and high-frequency results.

$$I = C_{\text{Low}} + C_{\text{High}}. \quad (6)$$

2.3 Refinement for specific issues

For color texture images with RGB components, some operations should be handled specifically. The sorting process of the median filter is determined by the luminance (Y) component, so an RGB-to-Y conversion is initially performed. The color difference operation uses the sum of absolute values of the three component differences. The linear interpolation is then carried out in a component-wise manner.

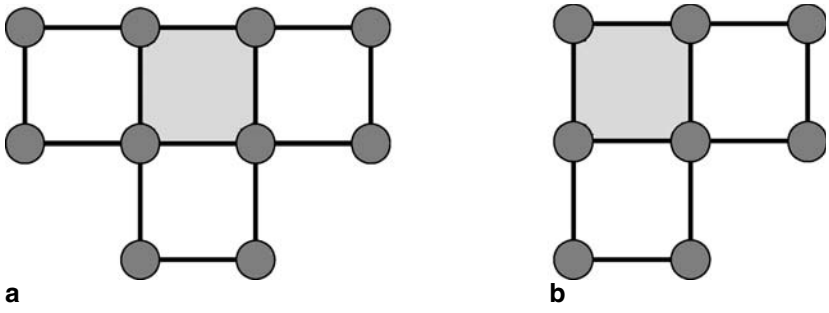


Fig. 3a,b. Two additional kernel styles for dynamic kernel adjustment: **a** T-shape; **b** L-shape

Some kinds of texture images, such as light-maps, do not benefit from edge-preserving filtering. The color deviation of a light-map is continuous over the whole map, so no edge exists. The light-map simulation results of edge-preserving filtering are almost identical with that of bilinear filtering. In this situation, bilinear filtering is more suitable for light-maps due to its lower computational cost.

When the interpolation is combined with MIP-map prefiltering, the two results of adjacent levels can be linearly blended, just as in trilinear filtering, to eliminate the level-switching jittering. Possible temporal aliasing during animation is also reduced. Although the image sharpness of this method is slightly inferior to that of the method using only one level, edges are still much sharper than with trilinear filtering.

3 Dynamic kernel adjustment

An advantage of our interpolation algorithm is that the number of the measured directions is adjustable. Hence, a dynamic kernel adjustment approach, based on the texture cache status, is presented to eliminate the extra off-chip texture access. The use of texture caches is prevalent for minimizing bandwidth demands by integrating a small amount of high-speed, on-chip memory. Hakura and Gupta (1997) analyzed the effects on the performance of caches by varying the cache organization and rasterization order. A texture cache is in a blocked representation to increase the spatial locality of texture access. The block size usually ranges from 4×4 to 8×8, and even above. Our interpolation kernel is dynamically adjusted according to the relationship between current texture position (memory address) and the cache block boundaries.

In addition to the original cross-shaped kernel, two kinds of kernel styles, T-shape and L-shape, are included, as shown in Fig. 3. The T-shape kernel comprises 10 source texels and 18 measured directions, while the L-shape kernel comprises 8 source texels and 14 measured directions. Each of these two styles has four transformations by orthogonal rotations.

Take Fig. 3a for example. The source texels *C1* and *C2* are not available. The simple low-pass filter for $C4_{\mathrm{Low}}$ is adjusted to the median of *C3*, *C4*, *C5*, *C8*, and *C9*. Similarly, $C5_{\mathrm{Low}}$ is adjusted to the median of *C4*, *C5*, *C6*, *C8*, and *C9*. The measured directions related to *C1* and *C2* are ignored.

Figure 4 depicts the usage of these three kernel styles in a cache block. The borders represent the boundaries of the block. The interpolator employs the original cross-shaped kernels inside the cache block, the T-shape kernels on the boundaries, and the L-shape kernels at the corners. When the center square of the kernel is located between two or four neighboring cache blocks, the conventional bilinear interpolation has to fetch all of these blocks. In this situation, our interpolator can utilize the same fetched blocks and apply the cross-shaped or the T-shape kernel across these neighboring blocks. As a result, our interpolation consumes the same off-chip memory bandwidth as the conventional bilinear or trilinear interpolation. Moreover, our interpolation makes better use of the texels already in the cache by considering their correlations.

The reduction in quality from that of the full cross-shaped style is hardly visible, since sufficient directions are measured in the whole operation. Notably,

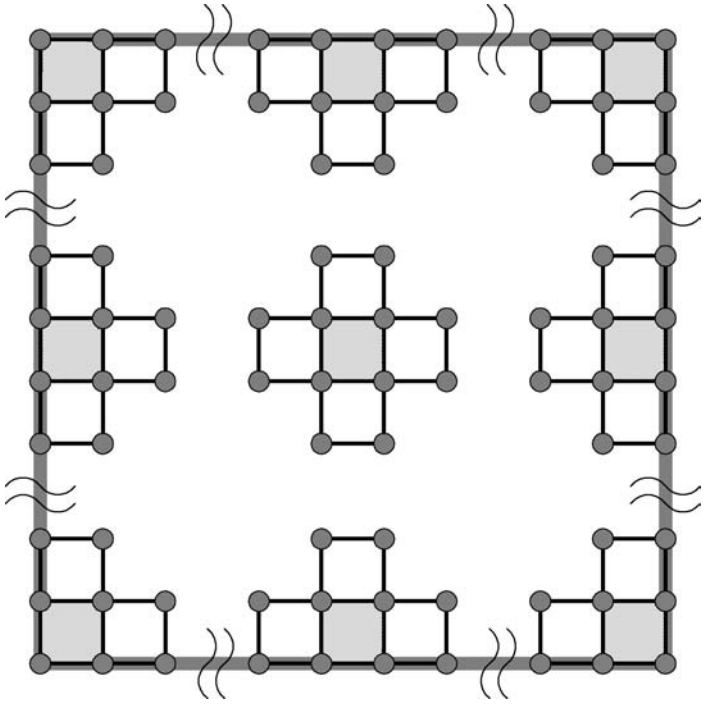


Fig. 4. Dynamic kernel adjustment in a cache block

the utilization of the cross-shaped style can be increased by the texture-prefetching technique or examining whether the neighboring blocks are already in the cache.

4 Edge-preserving MIP-map generation

Image pyramids are widely used in image coding, progressive transmission, and pattern recognition. A pyramid is typically generated by successively filtering and down-sampling by a factor of 2 in both dimensions. The common down-sampling filters are convolution-based, such as the box filter, Gaussian filter, and windowed sinc filters. However, some studies have indicated that nonlinear methods potentially produce more visually pleasing results.

Defee and Neuvo (1991) described three classes of median-type filters for preserving details. A multilevel median filter stacks median operations, which are performed over cross- and diamond-shaped masks in a 3×3 window. A FIR-median hybrid filter cascades averaging operations with a multilevel median filter. A recursive predicting FMH filter includes averaging filters and backward/forward optimal ramp predictors with half of the structure operating in a recursive loop. The authors' analyzes showed that these median-type filters with high-frequency periodic component removal best fit the pyramid scheme.

You and Kaveh (1996) demonstrated anisotropic diffusion for low-pass filtering. Anisotropic diffusion is derived from a specific partial differential equation with a carefully chosen diffusion coefficient. Their simulation showed that this method possesses the good property of preserving edges and their locations.

Decenciere et al. (2001) developed content-dependent down-sampling techniques. A reference image that specifies the importance given to each pixel is initially constructed. The construction uses the morphological Laplacian and the tophat operator. The recursive procedure to build a MIP-map is then based on the reference image but is modified to preserve small details during only one MIP-map level.

The above algorithms can be employed in edge-preserving MIP-map generation; otherwise, our interpolation algorithm can be modified to achieve

down-sampling by 2. The basic idea is to use the edge-preserving interpolation results at the centers of the even squares of the texture grid as the enhancement of the conventional convolution-based down-sampling. Let $M_n(x, y)$ be the nth level MIP-map image, and level 0 be the original texture image. The down-sampling operation is the weighted blending of a convolution function G with our interpolation I at specific locations. The formula is defined as follows:

$$M_{n+1}(x, y) = (1-\alpha)G(M_n) + \alpha I(M_n(2x+0.5, 2y+0.5)). \quad (7)$$

If the convolution function is the box filter, the function G is expressed as

$$G(M_n) = \frac{1}{4}[M_n(2x, 2y) + M_n(2x+1, 2y) + M_n(2x, 2y+1) + M_n(2x+1, 2y+1)]. \quad (8)$$

A Gaussian filter can also be used as a function G and outperforms the box filter. The positions interpolated by I are at the centers of the cross-shaped kernels on the even source squares of the texture grid. The weighting, α, controls the degree of edge preserving. Since a MIP-map is usually generated offline, α can be chosen carefully by trading off aliasing and blurring.

One possible advantage of extending the interpolation to the MIP-map generation is that the generation may be hardware accelerated. First, the convolution results are put into an accumulation buffer. Then, the interpolated results at specific locations are blended into the accumulation buffer as down-sampled texture images. This acceleration is helpful in real-time created textures, such as environment maps.

5 Simulation results

The proposed algorithm has been implemented using the Mesa 3D graphics library (Paul 2001) to observe the visual quality. The proposed algorithm is only compared to bilinear and trilinear filtering, because these algorithms consume the same memory bandwidth. Future work should extend the proposed algorithm to anisotropic filtering.

5.1 Texture magnification

Edge-preserving filtering exhibits a significant visual improvement when texture images are displayed with magnification. Contrarily, trilinear filtering degenerates into bilinear interpolation in magnification. Many anisotropic filtering algorithms do not deal with texture magnification, and also perform bilinear interpolation. Cubic convolution requires a 4×4 array of texels, which is more bandwidth demanding than the proposed algorithm. Thus it is omitted from this comparison.

Figure 5 is a portion of the Lena image that is scaled up by three. Figure 6 is a Chinese word on a quadrilateral in a 3-D environment. Figure 7 is a simple pattern on a teapot model. The staircase effects of trilinear filtering are clearly visible, while the proposed algorithm produces visually pleasing edges. The smooth regions are similar in either algorithm, but the details are much sharper with the proposed algorithm.

5.2 Texture minification

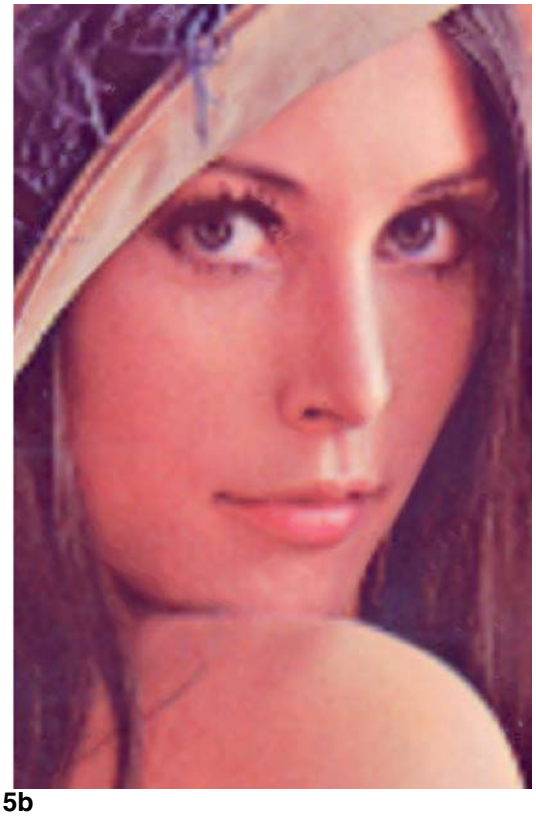
Figure 8 shows texture-mapped text on a cubic Bezier patch. The text, whose size is 1024×1024, is shrunk to smaller than half its original size when displayed on the screen. The MIP-map generation filter is the box filter as in Eq. (8) for the trilinear filtering and uses $\alpha = 0.5$ for the proposed algorithm. The proposed algorithm preserves the details and produces less blurry results. The anisotropic situation is also investigated, as shown in Fig. 9. The effect of varying the parameter α is also explored. With the proposed algorithm, the patterns in the foreground area have less-jagged edges, while the patterns in the background area are much clear. However, the result is not free from aliasing. The algorithm should be transformed into an anisotropic style, when the normal of the polygon is nearly perpendicular to the view direction.

6 Hardware implementation

This study aims to design high-performance interpolation hardware suitable for real-time texture filtering. The hardware should operate at a high clock rate with low latency and be fully pipelined for burst-mode processing. Parallel processing is employed to meet the requirement. Some special designs are described as follows.

A table look-up technique is adopted to implement the complex weighting calculation of Eq. (2). For each measured direction, ROI_i is a constant and d_i is

5a

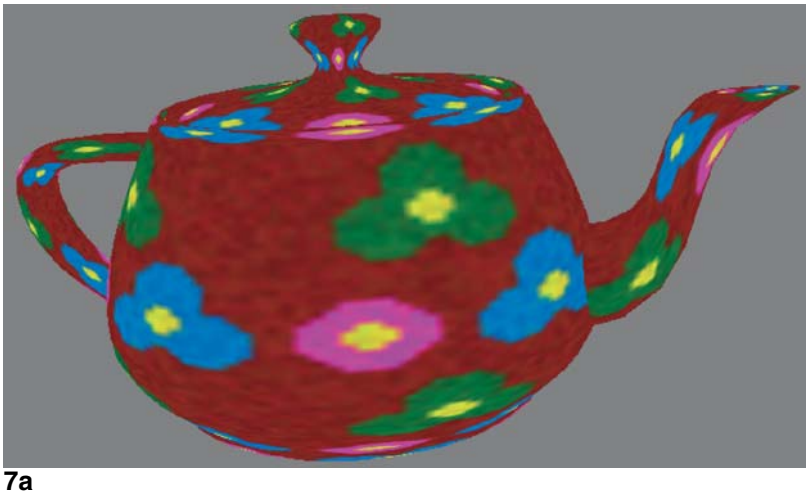


5b

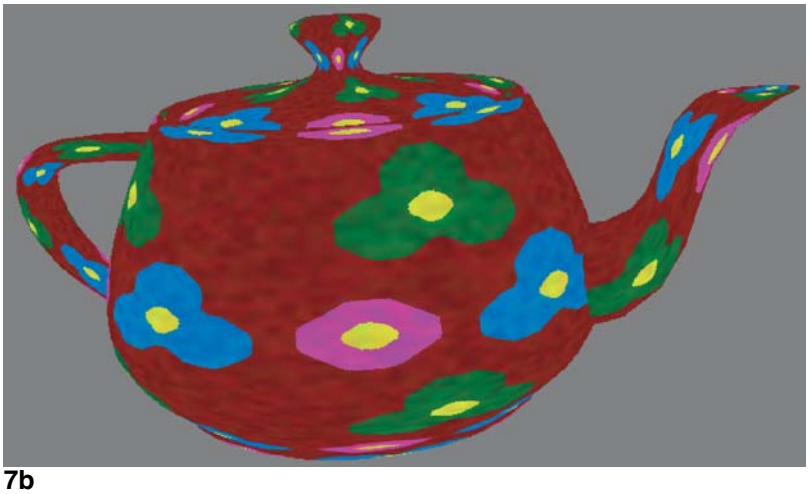
6a

6b

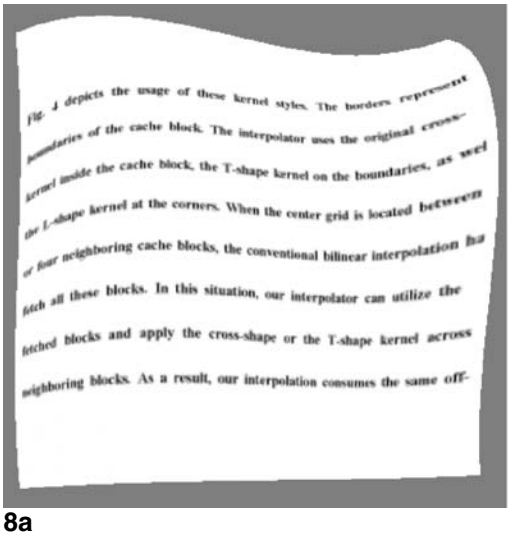
Fig. 5a,b. Texture magnification of Lena image: **a** trilinear filtering; **b** proposed algorithm
Fig. 6a,b. Texture magnification of a Chinese word: **a** trilinear filtering; **b** proposed algorithm



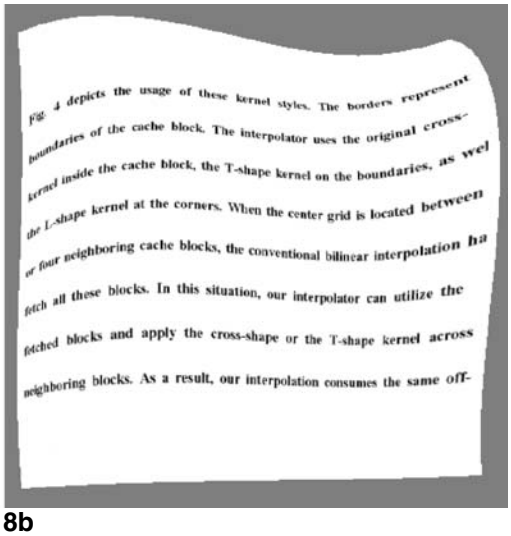
7a



7b



8a



8b

Fig. 7a,b. Texture magnification on a teapot model: **a** trilinear filtering; **b** proposed algorithm
Fig. 8a,b. Texture minification of the text: **a** trilinear filtering; **b** proposed algorithm

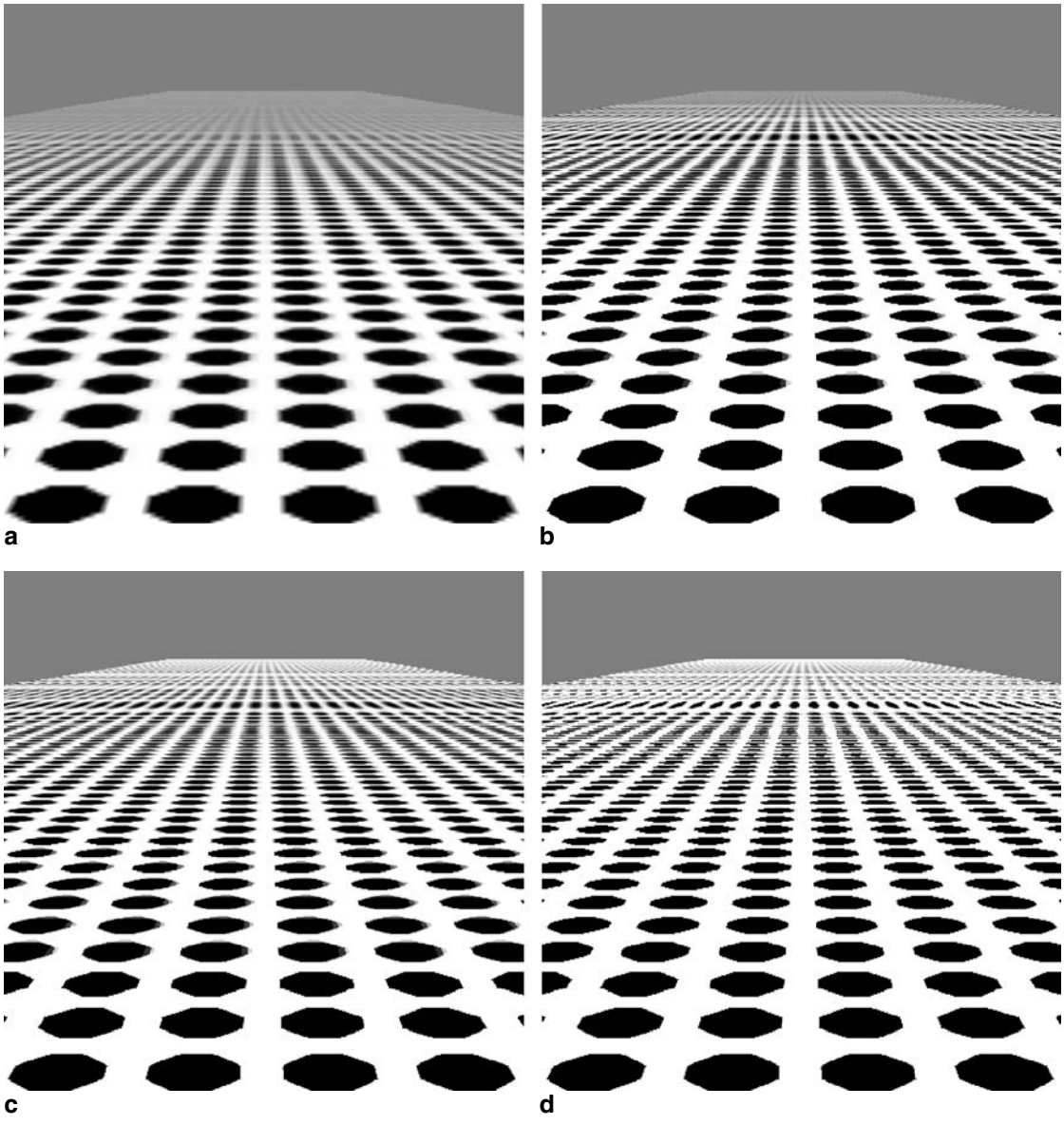


Fig. 9a–d. Anisotropic situation: **a** trilinear filtering; **b** proposed algorithm with $\alpha = 0$; **c** proposed algorithm with $\alpha = 0.5$; **d** proposed algorithm with $\alpha = 1$

a predefined function of (x_f, y_f). The inputs of each weighting table are x_f and y_f. Four-bit precision for each coordinate is enough. This implementation subdivides a sampling lattice into 256 sub-regions. The output of each table is w_i, which exhibits eight-bit precision. The gate count of each table is fewer than 550 after hardware synthesis, and the latency is 1.4 ns.

The schematic diagram of the asymmetrical median filter, serving as a simple low-pass filter, is shown in Fig. 10. For color images, the luminance (Y) is used for median selection. The RGB-to-

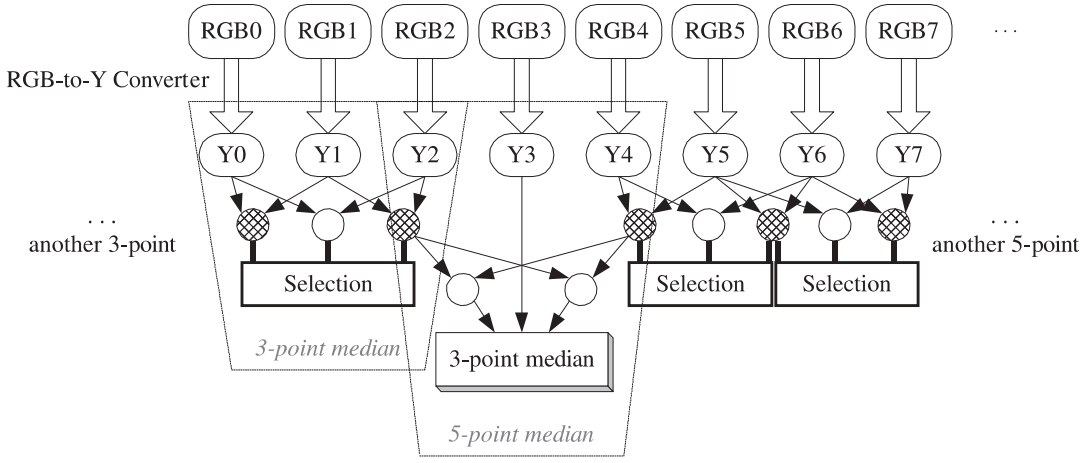


Fig. 10. Asymmetrical median filter module

Y converter is a carry-save adder tree with hard-wired preshift. The median selection consists of several two-input comparison cells, drawn as circles in Fig. 10. The comparison cells output the larger result from the right-hand side, the smaller result from the left-hand side, and a one-bit flag indicating comparison result. The three-point median filter has three comparison cells in parallel and a final selector. In the five-point median filter, the initial four comparison cells sort out the maximum and minimum from the five texels. Then, the remaining results pass through a three-point median filter to obtain the median of the five texels. In addition, circular operation sharing is exploited to merge the comparison cells that have the same inputs. Twelve comparison cells can be merged and are illustrated as filled circles in Fig. 10. The total number of comparison operations is reduced to 40.

The edge-preserving interpolation hardware was synthesized with the Avanti 0.35 μm cell library using the TSMC CMOS 1P4M process. The design can operate at 101.2 MHz with full pipelining and three-clock latency. The performance is sufficient for real-time rendering, since modern graphics chips with the same manufacturing process only operate at $66 \sim 90$ MHz. The total gate count is 87 431. This hardware area overhead is very small in comparison with millions gate count of state-of-the-art graphics chips.

7 Conclusion and future work

This paper has presented an edge-preserving texture-filtering algorithm. With this algorithm, edges and details are much sharper and more appropriate for human perception. The jaggedness of edges under texture magnification is greatly reduced. Text, in particular, benefits significantly. The algorithm requires no extra off-chip memory access, by combining texture caches and dynamic kernel adjustment. Our hardware design attests that the algorithm is capable of real-time rendering with low hardware cost. The same hardware can also be utilized for video resizing since current graphics chips all support video playback. Another potential benefit is that detailed but seldom used MIP-map levels can be dropped to save storage, because the algorithm retains good quality under higher-ratio magnification. However, temporal aliasing may sometimes occur in texture minification during animation. Edge preserving may not be advantaged for some texture images, such as light-maps, because no edge exists on them. The algorithm should be applied adaptively in consideration of whether the texture will profit or not.

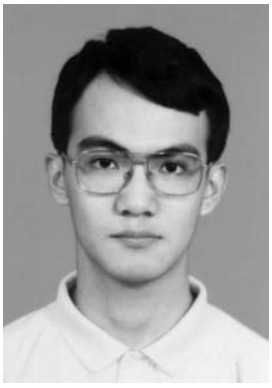
Several issues are still under research. For the purpose of transforming the algorithm into anisotropic filtering, one possible method is combination with the footprint assembly technique. Besides, the benefit of edge preserving for the transparency component or bump mapping is interesting. Although only

2-D texturing is considered in this paper, the algorithm may be extended further to 3-D texturing. The extension may result in surface-preserving volume filtering by grouping several measured directions on a surface.

References

1. Algazi VR, Ford GE, Potharlanka R (1991) Directional interpolation of images based on visual properties and rank order filtering. In: Chan YT (ed) Proceedings of ICASSP 1991. IEEE Signal Processing Society Press, Piscataway, pp 3005–3008
2. Blinn JF, Newell ME (1976) Texture and reflection in computer generated images. Commun ACM 19(10):542–547
3. Cant RJ, Shrubsole PA (2000) Texture potential MIP mapping, a new high-quality texture antialiasing algorithm. ACM Trans Graph 19(3):164–184
4. Carrato S, Tenze L (2000) A high quality $2x$ image interpolator. IEEE Signal Process Lett 7(6):132–134
5. Chen T, Wu HR, Yu ZH (2000) Efficient deinterlacing algorithm using edge-based line average interpolation. Opt Eng 39(8):2101–2105
6. Crow FC (1984) Summed-area tables for texture mapping. SIGGRAPH Comput Graph 18(3):207–212
7. Decenciere E, Marcotegui B, Meyer F (2001) Content-dependent image sampling using mathematical morphology: application to texture mapping. Signal Process Image Commun 16(6):567–584
8. Defee I, Neuvo Y (1991) Nonlinear filters in image pyramid generation. In: Shenoi BA (ed) Proceedings of IEEE International Conference on Systems Engineering. IEEE, Piscataway, pp 269–272
9. Demirer M, Grimsdale RL (1994) Texture mapping using clustering techniques. In: Birand T (ed) Proceedings of Mediterranean Electrotechnical Conference 1994. IEEE, Piscataway, pp 355–358
10. Doyle T, Looymans M (1990) Progressive scan conversion using edge information. In: Chiariglione L (ed) Signal Processing of HDTV II. Elsevier, North-Holland, pp 711–721
11. Ewins JP, Waller MD, White M, Lister PF (2000) Implementing an anisotropic texture filter. Comput Graph 24(2):253–267
12. Feibush EA, Levoy M, Cook RL (1980) Synthetic texturing using digital filters. SIGGRAPH Comput Graph 14(3):294–301
13. Fournier A, Fiume E (1988) Constant-time filtering with space variant kernels. SIGGRAPH Comput Graph 22(4):229–238
14. Gangnet M, Perny D, Coueignoux P (1982) Perspective mapping of planar textures. In: Greenaway DS, Warman EA (eds) Eurographics 82. Elsevier, North-Holland, pp 57–71
15. Greene N, Heckbert PS (1986) Creating raster omnimax images from multiple perspective views using the elliptical weighted average filter. IEEE Comput Graph Appl 6(6):21–27
16. Hakura ZS, Gupta A (1997) The design and analysis of a cache architecture for texture mapping. In: Pleszkun A, Mudge T (eds) Proceedings of the 24th International Symposium on Computer Architecture. ACM, New York, pp 108–120
17. Heckbert PS (1986) Survey of texture mapping. IEEE Comput Graph Appl 6(11):56–67
18. Hubel DH (1988) Eye, brain, and vision. Scientific American Library, New York
19. Huttner T, Strasser W (1999) Fast footprint MIPmapping. In: Knittel G, Pfister H (eds) Proceedings of the 1999 SIGGRAPH/Eurographics Workshop on Graphics Hardware. ACM, New York, pp 35–43
20. Jensen K, Anastassiou D (1995) Subpixel edge localization and the interpolation of still images. IEEE Trans Image Process 4(3):285–295
21. Kang DW (2000) Two-channel spatial interpolation of images. Signal Process Image Commun 16(4):395–399
22. Kuo CJ, Liao C, Lin CC (1996) Adaptive interpolation technique for scanning rate conversion. IEEE Trans Circuits Syst Video Technol 6(3):317–321
23. Lee MH, Kim JH, Lee JS, Ryu KK, Song DI (1994) A new algorithm for interlaced to progressive scan conversion based on directional correlations and its IC design. IEEE Trans Consumer Electron 40(2):119–129
24. Lee HY, Park JW, Bae TM, Choi SU, Ha YH (2000) Adaptive scan rate up-conversion system based on human visual characteristics. IEEE Trans Consumer Electron 46(4):999–1006
25. McCormack J, Perry R, Farkas KI, Jouppi NP (1999) Feline: fast elliptical lines for anisotropic texture mapping. SIGGRAPH Comput Graph 33(4):243–250
26. Michaud F, Dinh CTL, Lachiver G (1997) Fuzzy detection of edge-direction for video line doubling. IEEE Trans Circuits Syst Video Technol 7(3):539–542
27. Paul B (2001) The Mesa 3D graphics library. http://www.mesa3d.org/. Cited 3 Aug 2002
28. Salonen J (1994) Edge and motion controlled spatial up-conversion. IEEE Trans Consumer Electron 40(3):225–233
29. Schilling A, Knittel G, Strasser W (1996) Texram: a smart memory for texturing. IEEE Comput Graph Appl 16(3):32–41
30. Thurnhofer S, Mitra SK (1996) Edge-enhanced image zooming. Opt Eng 35(7):1862–1870
31. Ting HC, Hang HM (1997) Edge preserving interpolation of digital images using fuzzy inference. J Vis Commun Image Represent 8(4):338–355
32. Williams L (1983) Pyramidal parametrics. SIGGRAPH Comput Graph 17(3):1–11
33. Xue K, Winans A, Walowit E (1992) An edge-restricted spatial interpolation algorithm. J Electron Imaging 1(2):152–161
34. You YL, Kaveh M (1996) Pyramidal image compression using anisotropic and error-corrected interpolation. In: Hayes MH (ed) Proceedings ICASSP 1996. IEEE Signal Processing Society Press, Piscataway, pp 1946–1949
35. Yu X, Morse BS, Sederberg TW (2001) Image reconstruction using data-dependent triangulation. IEEE Comput Graph Appl 21(3):62–68

Photographs of the authors and their biographies are given on the next page.



YUAN-CHUNG LEE received the B.S. degree in electronics engineering from National Chiao Tung University in 1997, and the Ph.D. degree from National Chiao Tung University in 2002. His research interests include real-time rendering, graphics architecture, video processing, VLSI system, and digital IC design.

CHEIN-WEI JEN received the B.S. degree from National Chiao Tung University, Hsinchu, Taiwan, in 1970, the M.S. degree from Stanford University, Stanford, CA, in 1977, and the Ph.D. degree from National Chiao Tung University in 1983. He is currently with the Department of Electronics Engineering and the Institute of Electronics, National Chiao Tung University, as a Professor. During 1985–1986, he was with the University of Southern California at Los Angeles as a Visiting Researcher. His current research interests include VLSI design, digital signal processing, processor architecture, and design automation. Dr. Jen is a member of the IEEE and of Phi Tau Phi.