Enhanced EWA filtering

Kalle Rutanen

March 3, 2009

1 Abstract

This paper discusses the Elliptically Weighted Average (EWA) filtering algorithm along with enhancements that further increase image quality. We derive all the mathematical results that are needed in the algorithm and generalize the algorithm as follows. First, we allow filters with varying radii of support. Second, we propose a technique to embed differing filters for the minification and magnification. This enhances the image quality by allowing to use those filters in reconstruction that do not create excessive blurring on magnification. The transition between filters is made invisible by linearly interpolating between the filters. The original EWA algorithm is recovered by setting both filters to a Gaussian filter with radius 1. Third, our derivations are coordinate-free which we believe is the best representation for the formulae in the algorithm. This also enables generalizing the algorithm to higher dimensions (e.g. filtering voxel data in 3d), although this would be very costly. We shall demonstrate that using a filter radius of 1 for the gaussian filter as in the original EWA filtering algorithm leads to artifacts when magnifying.

2 Aliasing

EWA filtering was developed by Heckbert and Greene [1] [2]. It is one of the highest quality filtering techniques that are practical for image warping. The need for filtering can be seen from Figure 1a which shows a checkerboard viewed in perspective, with severe aliasing artifacts. Figure 1b shows the same image with proper filtering using the enhanced EWA filtering as desribed in this paper.

In principle we visualize there being an image function $I : \mathbb{R}^2 \to \text{Color}$ that maps positions on the image plane to colors. This function can be

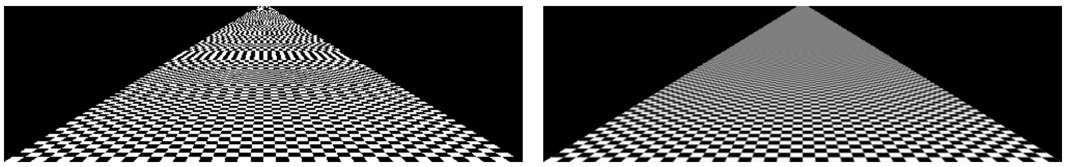

(a) No filtering (b) Enhanced EWA filtering

Figure 1: An example of aliasing and antialiasing.

arbitrary and it is not uncommon for the function to have frequency content extending to infinity. Examples of this kind of frequency content are sharp edges and a checkerboard pattern extending infinitely to the horizon. To show this continuous image on the computer screen, we can only use a finite amount of pixels to represent it. This means that there is an upper limit to the frequency content that can be displayed on the screen. On the other hand, the sampling theorem from signal processing theory tells us that to reconstruct a continuous function f from point samples of f arranged on a regular grid we need a sampling frequency F that is at least twice the highest frequency in f. This F is known as the *Nyquist limit.* Since we know that in some cases the image's maximum frequency tends towards infinite, we can be assured that there is in general no way to represent images on the display in their original form.

Naive point sampling of the image function with a sampling frequency below the Nyquist limit results in an artifact called *aliasing*, which Figure 1a demonstrates. When the continuous function is reconstructed from these point samples, the function can be significantly different from the original one: aliasing causes the higher frequencies to sum up to the lower frequencies. Since the result of aliasing is in general meaningless noise, we would like to avoid it. This is done by removing the higher frequencies of the image function *before* point sampling it. This process is called anti-aliasing or filtering, an example of which can be seen in Figure 1b.

Rather than considering the problem in its all generality, the approach usually taken in computer graphics is to isolate the sources of aliasing and handle these cases specially. An example of such an approach is when we wish to warp an image (called a *texture*) on the screen. Such image warps are responsible for much of the high frequency content in computer generated imagery. Fortunately they can be anti-aliased very efficiently as we shall see. The additional information we have here is that the data is itself of finite frequency. It is the image warp which causes the possible higher frequencies in the screen image function.

3 Texture antialiasing

Let us define an image warping function f as an arbitrary C^1-diffeomorphism from the texture space to the screen space. What this means is simply that f is continuous, invertible, differentiable, and that its inverse f^{-1} is all of these too. When rendering the warped texture on the screen, one usually loops over the screen pixels (x, y) and uses the inverse warp f^{-1} to get to the texture space to find out the color for the pixel. Naively one could take this color to be the texture pixel (texel) nearest to $(u, v) = f^{-1}(x, y)$. However, as discussed above, this approach leads to aliasing. Rather, the approach usually taken is to consider an infinitesimally small point set around the sampling point. In this set, the inverse image warping function f^{-1} looks like a plane (a simple affine function) which can be completely described by the two partial derivative vectors at that point. This observation is extended as an approximation over the finite area of a single screen pixel. This approximation is called a local affine (linear) approximation and is a recurrent theme when considering problems containing non-affine (non-linear) functions. Strictly speaking, pixels do not have an area since they are point samples. More formally then, by the 'area' of the pixel we mean the support set of the pixel filter (the collection of those points that map to non-zero values under the filter) that is used to filter the screen image function.

Because, in general, filtering in continuous space does not have a closed form solution, and the computer can't be used to actually compute results in a continuous space, we need to somehow convert the filtering problem to a discrete one. The way to do this in this case is to inverse warp the pixel filter to the texture space with f^{-1}. However, instead of using the f^{-1} itself, we will use it's local affine approximation at that point. This way the pixel filter ends up being transformed by an affine transformation.

4 EWA algorithm

In addition to the local affine approximation, the EWA algorithm is based on the assumption that the filtering of the screen image function is done with a radially symmetric filter. Such a radially symmetric filter necessarily has a circular support: thus the pixel 'area' is circular. If we now transform the filter to the texture space using the inverse of the local affine transformation, we necessarily end up with an ellipse. This explains the name of the algorithm. We can now easily describe the algorithm:

1. Find out the bounding box B of the ellipse E in the texture space.

2. Loop over the texels in B. If the texel is contained in E, multiply the texel value with the filter value and add the result to a running sum named 'imageSum'. In addition, keep up a running sum of the filter weights named 'weightSum'.

3. Return 'imageSum / weightSum'.

Note that the texture is thought to be extended to infinity by some extension mechanism. That is, if the texel is physically outside the texture stored in the memory, one chooses some texel in the texture according to some preferred strategy. The common ones include choosing the closest boundary texel, returning some fixed value, tiling the texture repeatedly, and tiling the texture by successive mirroring.

Two problems immediately arise. First, the affine transformation can expand the filter support without a limit: this is what happens when the texture is viewed at increasing distances. While object sizes decrease, the work we do for the filtering increases. We would like the filtering to always be a constant time operation. Second, the affine transformation can shrink the filter support without a limit making the ellipse fall between texels: this is what happens when the texture is viewed at increasing magnifications. This is because we haven't yet taken into account the reconstruction of the texture function. We would like to include the effect of the reconstruction filter to the antialiasing filter. After this, the ellipse always intersects some texels. We shall later give solutions to both of these problems.

What follows is the derivation of the computation of the bounding box, and the derivation of the forward differences that are used to effectively trace where we are with respect to the ellipse.

5 Implicit representation of ellipsoids

We will be working with origin-centered ellipsoids in $\mathbb{R}^n$. We define an origin-centered ellipsoid as an invertible linear transform of an origin-centered unit sphere. From now on we will leave the 'origin-centered' qualification off and assume it for brevity.

All ellipsoids can be described by the level sets of quadratic forms as follows. Let S be a symmetric and positive-definite $n \times n$ real matrix. Let f be a function such that:

$$f : \mathbb{R}^{n\times 1} \to \mathbb{R} : f(x) = x^T S x.$$

Then the set $\{x : f(x) = 1\}$ is an ellipsoid. The proof is as follows. Because S is symmetric, its eigen-decomposition exists. So let

$$S = QDQ^T,$$

where Q is an orthogonal matrix containing the unit eigenvectors as its columns and D is a diagonal matrix containing the corresponding eigenvalues. Now

$$\begin{aligned}
\{x : x^T Sx = 1\} &= \{x : x^T QDQ^T x = 1\} \\
&= \{x : x^T Q\sqrt{D}\sqrt{D}^T Q^T x = 1\} \\
&= \{(\sqrt{D}^T Q^T)^{-1} x : x^T x = 1\} \\
&= \{Q^{-T}\sqrt{D^{-T}} x : x^T x = 1\} \\
&= \{Q\sqrt{D^{-1}} x : x^T x = 1\}.
\end{aligned}$$

What we have shown is that each ellipsoid can be formed by a two-pass process. First one scales a unit sphere independently in each dimension giving an axis aligned ellipsoid. Then one applies a rotation (and possibly a reflection) to that ellipsoid. Because a rotation does not change lengths, we conclude that the lengths of the principal axes are given by the diagonal of $\sqrt{D^{-1}}$. The principal axis vectors themselves are given by the product $Q\sqrt{D^{-1}}$. At the same time, we have found the linear transformation that transforms the unit sphere to the ellipsoid with the given quadratic form matrix. This proves that all of the sets $\{x \mid x^T Sx = 1\}$ are ellipsoids.

We will now prove that all linear transformations of the unit sphere can be represented in the quadratic form representation. First, we note that a unit sphere Q is given by the set:

$$Q = \{x : |x| = 1\}$$

However, this is equivalent to:

$$\begin{aligned}
Q &= \{x : |x|^2 = 1\} \\
&\Leftrightarrow \\
Q &= \{x : x^T x = 1\} \\
&\Leftrightarrow \\
Q &= \{x : x^T Ix = 1\}
\end{aligned}$$

Now, transform Q by a linear transformation L:

$$\begin{aligned} L(Q) &= \{Lx : x^T I x = 1\} \\ &= \{x : (L^{-1}x)^T I (L^{-1}x) = 1\} \\ &= \{x : x^T L^{-T} L^{-1} x = 1\} \end{aligned}$$

Clearly $L^{-T}L^{-1}$ is symmetric:

$$(L^{-T}L^{-1})^T = L^{-T}L^{-1} = L^{-T}L^{-1}$$

And positive semi-definite:

$$\forall x : x^T L^{-T} L^{-1} x = (L^{-1}x)^T (L^{-1}x) = |L^{-1}x|^2 \geq 0$$

However, because L is invertible, $x = 0$ is the only vector for which $L^{-1}x = 0$ and thus the matrix is positive definite. Thus any ellipsoid can be given by a quadratic form representation as stated. Let there be two ellipses with quadratic form matrices $L^{-T}L^{-1}$ and $M^{-T}M^{-1}$. Then:

$$\begin{aligned} \{ML^{-1}x : x^T L^{-T} L^{-1} x = 1\} &= \{x : (LM^{-1}x)^T L^{-T} L^{-1} (LM^{-1}x) = 1\} \\ &= \{x : x^T M^{-T} L^T L^{-T} L^{-1} L M^{-1} x = 1\} \\ &= \{x : x^T M^{-T} M^{-1} x = 1\} \end{aligned}$$

Thus any two ellipsoids are related by a linear transformation.

6 Computation of the ellipsoid bounding box

We wish to compute the bounds of the ellipsoid along an arbitrary unit vector n, so that we can bound the ellipsoid with a box. That is, we wish to maximize the length of the projection to n:

$$f(x) = n^T x,$$

under the constraint that the vector lies on the ellipsoid:

$$x^T S x = 1.$$

To solve this, we transform the problem to an unconstrained problem of maximizing (this is the method of Lagrance multipliers)

$$h(x, t) = n^T x + t(x^T S x - 1).$$

The gradient of h w.r.t to x is given by

$$\nabla_x h(x,t) = n + 2tSx,$$

and the derivative of h w.r.t. t is:

$$\frac{\partial h}{\partial t}(x,t) = x^T Sx - 1.$$

We set these derivatives to zero to find the maximum:

$$\begin{cases} n + 2tSx = 0 \\ x^T Sx - 1 = 0 \end{cases}$$

From the first equation we obtain

$$x = -\frac{S^{-1}n}{2t}.$$

Substitute this into the second equation to solve for t:

$$\begin{array}{rcl} \frac{1}{4t^2} n^T S^{-T} S S^{-1} n - 1 & = & 0 \\ & \Leftrightarrow & \\ \frac{1}{4t^2} n^T S^{-T} n & = & 1 \\ & \Leftrightarrow & \\ \frac{n^T S^{-T} n}{4} & = & t^2 \\ & \Leftrightarrow & \\ \frac{n^T S^{-1} n}{4} & = & t^2 \\ & \Leftrightarrow & \\ t & = & \pm \frac{\sqrt{n^T S^{-1} n}}{2}. \end{array}$$

Finally, substitute t back to the first equation to solve for x:

$$x = \pm \frac{S^{-1}n}{\sqrt{n^T S^{-1} n}}.$$

We are only interested in the length of vector x along the vector n (remember that n is a unit vector). This is computed as

$$|x_{\|n}| = n^T x = \pm \frac{n^T S^{-1} n}{\sqrt{n^T S^{-1} n}} = \pm \sqrt{n^T S^{-1} n}.$$

which completes our derivation. Let us apply the formula for the standard basis axes. In this case:

$$|x_{\|e_i}| = e_i^T x = \pm\sqrt{e_i^T S^{-1} e_i} = \pm\sqrt{S_{ii}^{-1}},$$

6.1 Example: Computation of an axis aligned bounding box in $\mathbb{R}^2$

Let

$$S = \begin{bmatrix} a & b \\ b & d \end{bmatrix}.$$

By Cramers rule

$$S^{-1} = \frac{1}{ad - b^2} \begin{bmatrix} d & -b \\ -b & a \end{bmatrix},$$

from which we get:

$$\begin{cases} x_{max} = \frac{\sqrt{d}}{ad-b^2} \\ x_{min} = -x_{max} \\ y_{max} = \frac{\sqrt{a}}{ad-b^2} \\ y_{min} = -y_{max} \end{cases}.$$

7 Computation of the forward differences

The function f that gives the ellipsoid as its level set is given by:

$$f(x) = x^T S x$$

We note that f is a n-variate quadratic polynomial in x. Rather than evaluating this polynomial directly at each point, we can make use of forward differences. The forward difference of f is simply $\delta f : \delta f(x; \delta x) = f(x+\delta x) - f(x)$. Similarly to the derivative, the forward difference of an n-degree polynomial is an $(n-1)$-degree polynomial. The advantage of using forward differences is that we can evaluate the polynomial on a uniform grid by simply using the forward difference to step forward on the polynomial: $f(x+\delta x) = f(x) + \delta f(x; \delta x)$. What really make the forward differences useful is that we can repeat this process: we can take take the forward difference of a forward difference to end up at a $(n-2)$-degree polynomial and so on. This process ends up when then forward difference gives a 0-degree polynomial, that is, a constant. Where this leads to is that given the values of the forward differences and the function value at a single point, we can find out all the other values of the function on the grid by using addition exclusively. Since we avoid the multiplications (other than at the setup at the beginning point), this is very efficient. We will now proceed with their derivation:

$$\begin{aligned}
\delta f(x;\delta x) &= f(x+\delta x) - f(x) \\
&= (x+\delta x)^T S(x+\delta x) - x^T Sx \\
&= x^T Sx + 2\delta x^T Sx + \delta x^T S\delta x - x^T Sx \\
&= 2\delta x^T Sx + \delta x^T S\delta x.
\end{aligned}$$

Because we wish to visit all texels inside a box, the only forward differences we need are those that are in the directions of the standard basis axes and of one-texel magnitude. Let us simplify the forward differences for this situation:

$$\begin{aligned}
\delta_i f(x) &= \delta f(x;e_i) \\
&= 2e_i^T Sx + e_i^T Se_i \\
&= 2S_i x + S_{ii},
\end{aligned}$$

where S_i denotes the i:th row of S. This is clearly a 1-degree polynomial in x. To get to the constant forward differences, we need to repeat the process.

$$\begin{aligned}
\delta^2 f(x;\delta x_1,\delta x_2) &= \delta f(x+\delta x_2;\delta x_1) - \delta f(x;\delta x_1) \\
&= 2\delta x_1^T S(x+\delta x_2) + \delta x_1^T S\delta x_1 - 2\delta x_1^T Sx - \delta x_1^T S\delta x_1 \\
&= 2\delta x_1^T S\delta x_2.
\end{aligned}$$

Again, we are only interested in those directions δx_1 and δx_2 that are along the standard basis axes. Thus we get:

$$\begin{aligned}
\delta_{ij}^2 f(x) &= \delta^2 f(x;e_i,e_j) \\
&= 2e_i^T Se_j \\
&= 2S_{ij}.
\end{aligned}$$

Note that

$$\delta^2 f(x;\delta x_1,\delta x_2) = \delta^2 f(x;\delta x_2,\delta x_1).$$

In general, with forward differences of arbitrary order, the differenciation order does not matter.

7.1 Example: Computation of the forward differences in $\mathbb{R}^2$

Let

$$S = \begin{bmatrix} a & b \\ b & d \end{bmatrix}.$$

Then the forward differences in the standard basis axes directions are

$$\begin{cases} \delta_x f(x,y) = 2S_0[x,y]^T + S_{00} = 2ax + 2by + a \\ \delta_y f(x,y) = 2S_1[x,y]^T + S_{11} = 2bx + 2dy + d \\ \delta^2_{xx} f(x,y) = 2S_{00} = 2a \\ \delta^2_{yy} f(x,y) = 2S_{11} = 2d \\ \delta^2_{xy} f(x,y) = 2S_{01} = 2b = 2S_{10} = \delta^2_{yx} f(x,y) \end{cases}$$

8 Constant-time filtering

Constant-time filtering can be achieved by using mipmaps. A *mipmap* is a list M_i of downsampled images of a texture. As an overload of terminology, the invidual images M_i are also called mipmaps. For familiarity, we will restrict the discussion to two dimensions. The M_0 contains the original image upsampled to the nearest width and height such that they are equal powers of two. For $i > 0$, M_i contains a downsampled version of M_0 with both dimensions divided by 2^i. If the width and height of M_0 are D, then the number of mipmaps in M is $log_2(D)$. Thus M_D contains only a single pixel, since $D/2^{log_2(D)} = 1$.

Mipmaps are strongly associated with trilinear (in higher dimensions, *multilinear*) filtering. However, more correctly mipmaps are simply prefiltered versions of an image. Which technique is used to filter over these images is an independent decision. Trilinear filtering works by finding a mipmap level where the pixel filter contains at most two pixels. The texture value at this level is retrieved by filtering with a separable triangle filter that is oriented on the standard basis axes in the texture space (this is often called bilinear filtering). However, to make transitions between mipmaps less evident, one also retrieves the texture value with bilinear filtering from one coarser level and then interpolates between these two filtering results, giving 'trilinear filtering'.

While fast, trilinear filtering has serious problems with quality. It does prevent antialiasing, but unfortunately also cuts away detail unnecessarily, resulting in loss of detail. The reason for this is that the filter used in texture space is always shaped and oriented as a square axis aligned box. In particular, pixel filters that transform to skinny diagonal shapes in texture space are heavily overblurred.

One then calls for a filtering method that can adapt to differently oriented shapes. This is where EWA filtering steps in: the ellipses can fit the the pixel filter shapes and orientations exactly under the local affine approximation and radial filter assumption. To use this with mipmaps, we increase the number of pixels that the filter is required to cover in the chosen mipmap level. Specifically, we require that the minor axis of the ellipse covers at least x amounts of pixels *per filter radius* in the coarser image. The emphasized words are what generalizes the technique to arbitrary filter radii. This means that the filtering time becomes dependent on the filter radius. Indeed this is how it should work: for example, increasing the radius of a gaussian filter (while keeping its variance constant) should not produce a more compact transformed pixel filter.

The work that must be done for the filtering is still not limited in any way: the ellipse can be of arbitrary size and eccentricity. *Eccentricity* of an ellipse is defined as the ratio of the lengths of its major and minor axis. It is the arbitrary eccentricities that allow for large filtering areas. We can achieve constant-time filtering by setting a maximum to eccentricity. Whenever the ellipse has an eccentricity above the allowed eccentricity, we scale up the length of the minor axis such that the ellipse has the maximum allowed eccentricity. This results in excessive blurring, but we can not allow for aliasing that would happen if we shrinked the major axis. This blurring is not something to be afraid of in terms of quality: since the filtering area is large anyway, the difference isn't probably noticeable to the eye. One just has to set the maximum eccentricity high enough: 30 seems to be a good value.

As with trilinear filtering, one should linearly interpolate between the EWA filtering results of the two mipmap levels that are adjacent to the computed detail level to make transitions invisible.

9 Computing ellipsoid eccentricity

We need to compute the ellipsoid eccentricity to be able to bound the work that must be done for the filtering. For this, we need the principal axes of the ellipsoid. In some texts, the authors incorrectly take the texture space derivatives as the ellipsoid's principal axes. However, this is incorrect: the derivatives correspond to the ellipsoid's principal axes if and only if they are orthogonal to each other. Using the lengths of the derivative vectors for the eccentricity computation leads to underestimating the eccentricity, sometimes to the point that it doesn't approximate the real eccentricity at all. For a pathological example, if the derivative vectors are close to being equal, the approximated eccentricity is close to 1, and the real eccentricity can be close to anything. Thus, using the derivatives even as an approximation to the principal axes is unjustified.

Instead, a full eigen-decomposition must be performed on the ellipsoid quadratic form matrix. The eigenvectors then give the principal axis directions and the eigenvalues are related to the lengths of the principal axis vectors. Let

$$x^T S x = 1,$$

where S is symmetric. Let x be an eigenvector of S corresponding to the eigenvalue λ with the restriction that it lies on the ellipsoid. Then:

$$\begin{aligned} x^T S x &= \lambda x^T x = \lambda |x|^2 = 1 \\ &\Rightarrow \\ |x| &= \frac{1}{\sqrt{\lambda}} \end{aligned}$$

Thus the lengths of the ellipse's principal axes can be computed from eigenvalues. There are several standard methods for computing eigen-decompositions, see for example [3].

9.1 Eigen-decomposition of a 2×2 matrix

For the 2d case, finding the eigen-decomposition for a symmetric 2×2 matrix is trivial. Let

$$S = \begin{bmatrix} a & b \\ b & d. \end{bmatrix}$$

We can solve the eigenvalues λ directly from the characteristic equation:

$$\begin{aligned} det \begin{bmatrix} a - \lambda & b \\ b & d - \lambda \end{bmatrix} &= 0 \\ &\Rightarrow \\ (a - \lambda)(d - \lambda) - b^2 &= 0 \\ &\Rightarrow \\ ad - \lambda(a + d) + \lambda^2 - b^2 &= 0 \\ &\Rightarrow \\ \lambda^2 - \lambda(a + d) + (ad - b^2) &= 0, \end{aligned}$$

which is a quadratic equation in λ and trivially solved. Theoretically, this equation should always have two real solutions (we take double-roots as two distinct roots) and thus the discriminant should be non-negative. However, practically the discriminant will sometimes be computed as negative due to rounding errors (especially when the ellipse is close to being a circle). In this case the discriminant should simply be rounded to zero. Given the eigenvalues λ_1 and λ_2, where $\lambda_1 \leq \lambda_2$:

$$\text{major axis length} = \frac{1}{\sqrt{\lambda_1}},$$
$$\text{minor axis length} = \frac{1}{\sqrt{\lambda_2}},$$

$$\begin{aligned} \text{eccentricity} &= \frac{\text{major axis length}}{\text{minor axis length}} \\ &= \sqrt{\frac{\lambda_2}{\lambda_1}}. \end{aligned}$$

In the case that eccentricity is above the maximum allowed eccentricity, we must also compute the eigenvectors. Let

$$\begin{aligned} \begin{bmatrix} a-\lambda & b \\ b & d-\lambda \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} &= 0 \\ &\Rightarrow \end{aligned}$$
$$\begin{cases} (a-\lambda)x + by = 0 \\ bx + (d-\lambda)y = 0 \end{cases}$$

From the first equation one gets:

$$y = \frac{(\lambda - a)x}{b},$$

giving a family of eigenvectors of

$$\begin{bmatrix} 1 \\ \frac{\lambda-a}{b} \end{bmatrix}.$$

The singularity can be removed by multiplying by b:

$$\begin{bmatrix} b \\ \lambda - a \end{bmatrix}.$$

From the second equation one gets:

$$y = \frac{bx}{\lambda - d},$$

giving a family of eigenvectors of

$$\begin{bmatrix} 1 \\ \frac{b}{\lambda - d} \end{bmatrix}.$$

The singularity can again be removed by multiplying by $(\lambda - d)$:

$$\begin{bmatrix} \lambda - d \\ b. \end{bmatrix}.$$

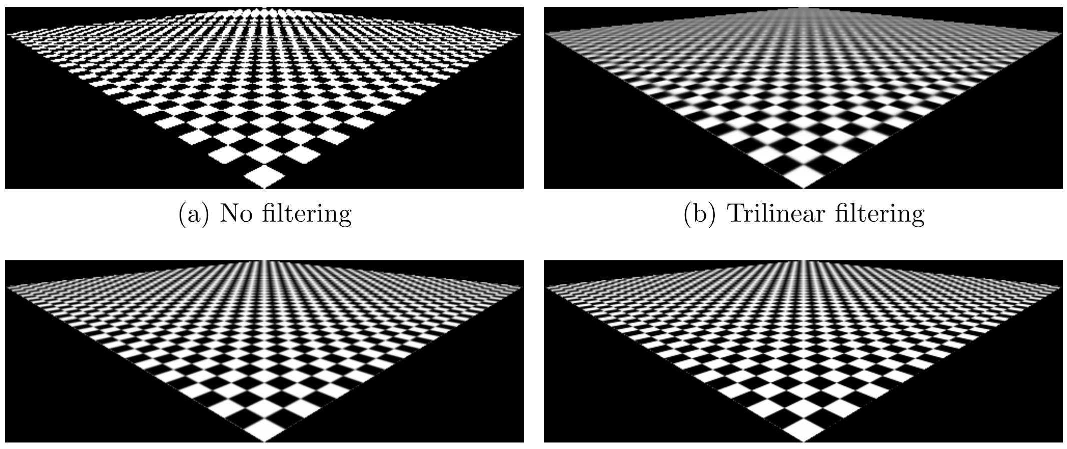
Note that we have used only one eigenvalue here, and thus the obtained vectors should be multiples of each other. However, we need to consider both of them because one of the vectors can be the zero vector. Both of the vectors being zero is equivalent to the matrix being a multiple of the identity matrix (then the ellipsoid is a sphere). However, if we compute the eigenvectors only when we have already checked that the eccentricity exceeds some threshold, this can't be the case. Out of these two vectors then, we should pick the one that has greater norm (for example, manhattan norm, for efficiency).

It can be shown that the eigenvectors of a real symmetric matrix are orthogonal to each other. Thus it suffices to compute a single eigenvector and obtain the other eigenvector by taking its perpendicular.

10 Visual comparison of filtering techniques

We shall now compare the results of different filtering techniques visually. We chose four filtering techniques for this task. These are 'no filtering', 'trilinear filtering', 'EWA filtering', and 'enhanced EWA filtering'. The original EWA filtering algorithm uses a gaussian filter with radius 1. For the enhanced EWA filtering algorithm I chose a Lanczos filter with radius 2 for magnification and a triangle filter with radius 1 for minification: I found these to produce the best results visually. For the checker images I have used repeated tiling to extend the texture: this is why there are visible edges at the filtered images. For the other images I have used clamping instead.

The Figure 2 image is a stress-test for the antialiasing capabilities of the filtering technique as well as its ability to adapt to anisotropic conditions. With no filtering, the image shows severe aliasing. With trilinear filtering, the image is overblurred. With EWA filtering, the image is nicely antialiased,


(a) No filtering (b) Trilinear filtering

(c) EWA filtering (d) Enhanced EWA filtering

Figure 2: Quality of anisotropic antialiasing with diagonal filter footprints.

but is a bit overblurred at the front. With enhanced EWA filtering, the image is both nicely antialiased and sharply magnified.

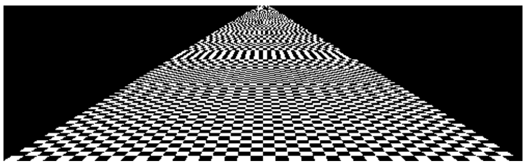
The Figure 3 is another stress-test for antialiasing and anisotropy, showing much greater frequency content. The results are similar to the previous image.

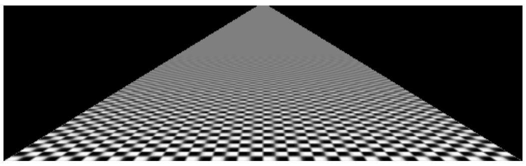
The Figure 4 demonstrates the quality of image in a normal viewing situation. In this case there is not that much frequency content and thus with no filtering the image looks acceptable, although one can see the blockiness of magnification. Trilinear filtering has severe overblurring. Both EWA filtering and enhanced EWA filtering perform antialiasing in a pleasing manner, but the enhanced EWA filtering preserves detail better.

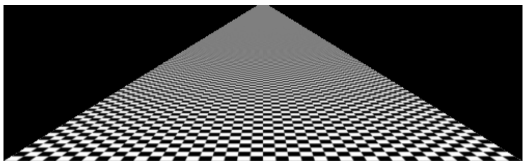
The Figure 5 demonstrates the quality of isotropic magnification. With no filtering, the result is blocky. With trilinear filtering, the result is still somewhat blocky and on the other hand blurry. EWA filtering results in overblurred image. Enhanced EWA filtering results in a sharp image.

The Figure 6 is another demonstration of the quality of magnification. Particularly, it demonstrates that in the original EWA algorithm the radius of 1 is not enough for the gaussian filter (of variance 0.5) to attenuate enough for the clamping step to be negligible. Let the gaussian filter $g : \mathbb{R} \to \mathbb{R}$ be given as $g(x) = ae^{-2x^2}$, where $a \in \mathbb{R}, a > 0$. Then the relative magnitude of the clamping step is

$$\frac{g(1)}{\max g} = \frac{g(1)}{g(0)} = \frac{ae^{-2}}{ae^{0}} = e^{-2} \approx 13.5\%.$$


(a) No filtering


(b) Trilinear filtering


(c) EWA filtering

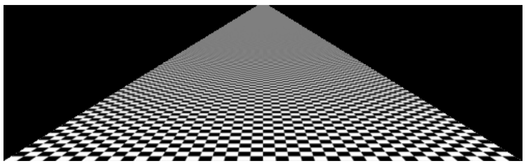
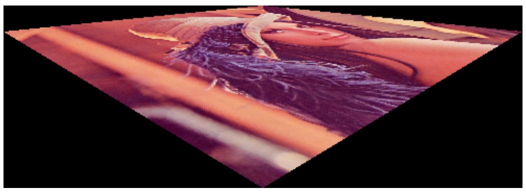
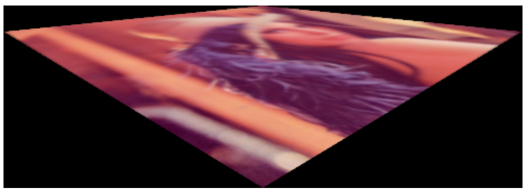
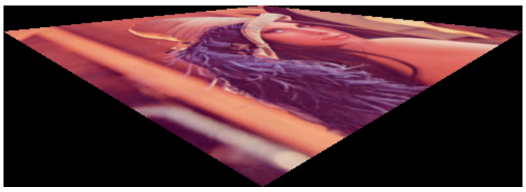

(d) Enhanced EWA filtering

Figure 3: Quality of anisotropic antialiasing with high frequency content.


(a) No filtering


(b) Trilinear filtering


(c) EWA filtering

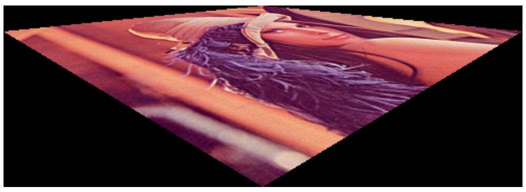

(d) Enhanced EWA filtering

Figure 4: A typical viewing situation.

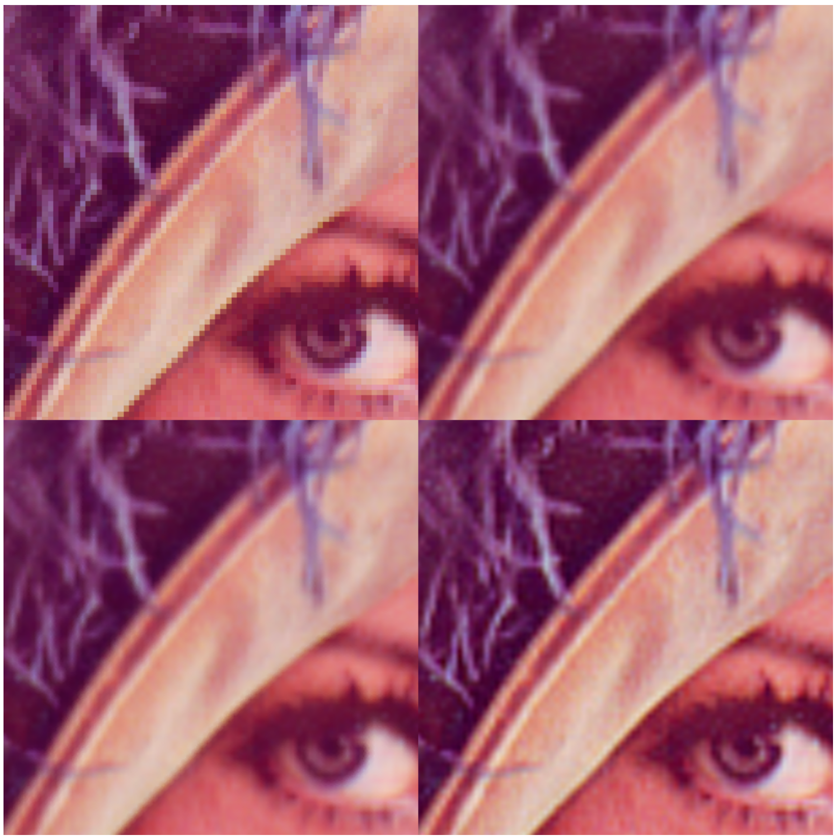
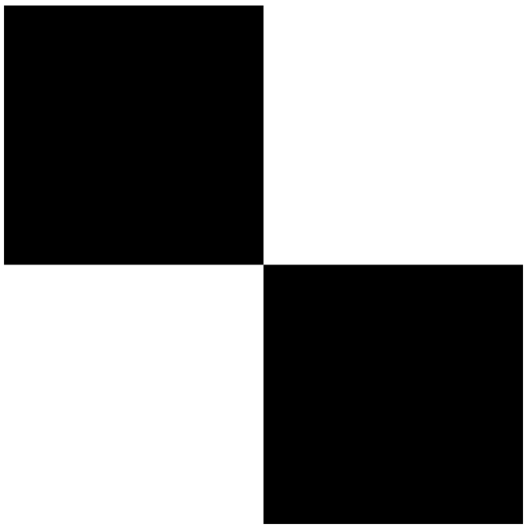
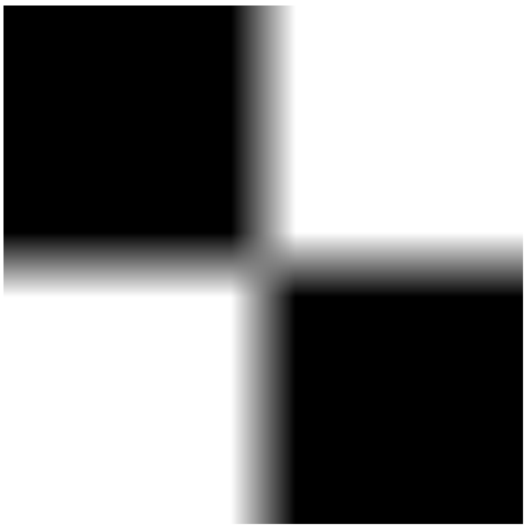


Figure 5: Quality of isotropic magnification. Top left: no filtering, top right: trilinear filtering, bottom left: EWA filtering, bottom right: enhanced EWA filtering.

This is quite high a magnitude and causes the visible artifacts shown.


(a) No filtering


(b) Trilinear filtering

(c) EWA filtering

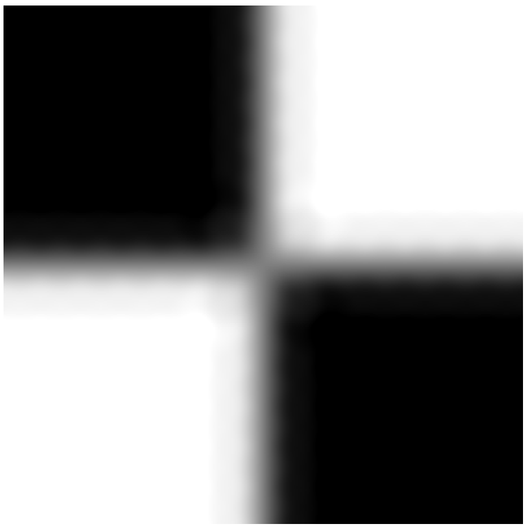

(d) Enhanced EWA filtering

Figure 6: Quality of isotropic magnification, magnification artifacts from Gaussian filter clamping.

11 Acknowledgements

I credit the poster 'Dave' from the newsgroup sci.math for the solution of the ellipsoid bound problem for projection axes along the standard basis vectors. Discussions with Dave Eberly in the newsgroup comp.graphics.algorithms resulted in the generalization of this result to arbitrary projection axes.

References

[1] Ned Greene and Paul Heckbert, *Creating Raster Omnimax Images from Multiple Perspective Views Using The Elliptical Weighted Average Filter*, IEEE Computer Graphics and Applications, June 1986, pp. 21-27.

[2] Paul Heckbert, *Fundamentals of Texture Mapping and Image Warping*, Master's thesis, UCB/CSD 89/516, CS Division, U.C. Berkeley, June 1989, 86 pp.

[3] William H. Press, Saul A. Teukolsky, William T. Vetterling, Brian P. Flannery, *Numerical Recipes, The Art of Scientific Computing*, 3rd. ed., 2007.