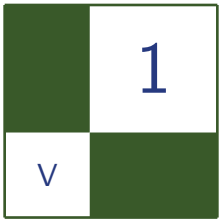


Anisotropic Kuwahara Filtering on the GPU

Jan Eric Kyprianidis, Henry Kang, and Jürgen Döllner

1.1 Introduction

Photorealistic depictions often contain more information than necessary to communicate the intended information. Artists therefore typically remove detail and use abstraction for effective visual communication. A typical approach to

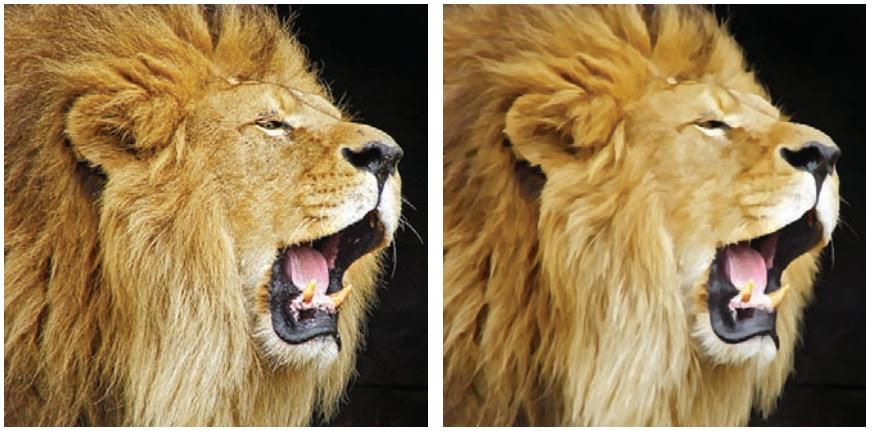


Figure 1.1. Original image (left). Output of the anisotropic Kuwahara filter (right). A painting-like flattening effect is generated along the local feature directions, while preserving shape boundaries.

automatically create stylized abstractions from images or videos is the use of an *edge-preserving* filter. Popular examples *of* edge-preserving filters used for image abstraction are the *bilateral* filter [Tomasi and Manduchi 98] and *mean shift* [Comaniciu and Meer 02]. Both smooth low-contrast regions while preserving high-contrast edges. Therefore they may fail for high-contrast images, where either no abstraction is performed or relevant information is removed because of the thresholds used. They also often fail for low-contrast images, where typically too much information is removed.

An edge-preserving filter that overcomes this limitation is the *Kuwahara filter* [Kuwahara et al. 76]. Based on local area flattening, the Kuwahara filter properly removes details even in a high-contrast region, and protects shape boundaries even in low-contrast regions. Hence it helps to maintain a roughly uniform level of abstraction across the image, while providing an overall painting-style look. Unfortunately the Kuwahara filter is unstable in the presence of noise and suffers from block artifacts. Several extensions and modifications have been proposed to improve the original Kuwahara filter. A discussion can be found in [Papari et al. 07].

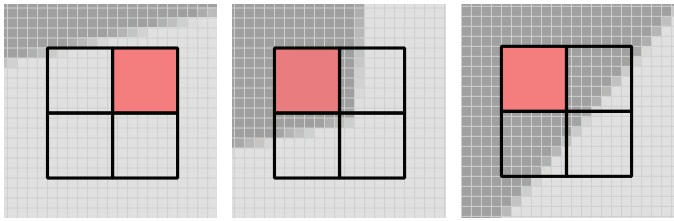
In this chapter we present an implementation of the anisotropic Kuwahara filter [Kyprianidis et al. 09]. The anisotropic Kuwahara filter is a generalization of the Kuwahara filter that avoids artifacts by adapting shape, scale, and orientation of the filter to the local structure of the input. Due to this adaption, directional image features are better preserved and emphasized. This results in overall sharper edges and a more feature-abiding painterly effect.

1.2 Kuwahara Filtering

The general idea behind the Kuwahara filter is to divide the filter kernel into four rectangular subregions which overlap by one pixel. The filter response is then defined by the mean of a subregion with minimum variance (Figure 1.2).

Let $f\colon \mathbb{Z}^2 \longrightarrow \mathbb{R}^3$ denote the input image, let $r > 0$ be the radius of the filter and let $(x_0, y_0) \in \mathbb{Z}^2$ be a point. The rectangular subregions are then given by

$$
\begin{aligned}
W_0 &= [x_0 - r, x_0] \times [y_0, y_0 + r], \\
W_1 &= [x_0, x_0 + r] \times [y_0, y_0 + r], \\
W_2 &= [x_0, x_0 + r] \times [y_0 - r, y_0], \\
W_3 &= [x_0 - r, x_0] \times [y_0 - r, y_0].
\end{aligned}
$$



(a) Flat region (b) Corner (c) Edge

Figure 1.2. The Kuwahara filter divides the filter kernel into four rectangular subregions. The filter response is then defined by the mean of a subregion with minimum variance.

Let $|W_k| = (r+1)^2$ be the number of pixel in each subregion. The mean (average) of a subregion W_k is then defined as:

$$m_k = \frac{1}{|W_k|} \sum_{(x,y) \in W_k} f(x,y).$$

The variance is defined as the average of the square of the distance of each pixel to the mean:

$$\begin{aligned} s_k^2 &= \frac{1}{|W_k|} \sum_{(x,y) \in W_k} \big(f(x,y) - m_k\big)^2 \\ &= \frac{1}{|W_k|} \sum_{(x,y) \in W_k} f^2(x,y) - m_k. \end{aligned}$$

Assuming that the variances of the color channels do not correlate, we define the variance of the subregion W_k to be the sum of the squared variances of

Figure 1.3. Output of the Kuwahara filter.

```
uniform sampler2D src;
uniform int radius;

void main (void) {
  vec2 src_size = textureSize2D(src, 0);
  vec2 uv = gl_FragCoord.xy / src_size;
  float n = float((radius + 1) * (radius + 1));

  vec3 m[4];
  vec3 s[4];
  for (int k = 0; k < 4; ++k) {
    m[k] = vec3(0.0);
    s[k] = vec3(0.0);
  }

  struct Window { int x1, y1, x2, y2; };
  Window W[4] = Window[4](
    Window( -radius, -radius,      0,      0 ),
    Window(       0, -radius, radius,      0 ),
    Window(       0,       0, radius, radius ),
    Window( -radius,       0,      0, radius )
  );

  for (int k = 0; k < 4; ++k) {
    for (int j = W[k].y1; j <= W[k].y2; ++j)  {
      for (int i = W[k].x1; i <= W[k].x2; ++i)  {
        vec3 c = texture2D(src, uv + vec2(i,j) / src_size).rgb;
        m[k] += c;
        s[k] += c * c;
      }
    }
  }

  float min_sigma2 = 1e+2;
  for (int k = 0; k < 4; ++k) {
    m[k] /= n;
    s[k] = abs(s[k] / n - m[k] * m[k]);

    float sigma2 = s[k].r + s[k].g + s[k].b;
    if (sigma2 < min_sigma2) {
      min_sigma2 = sigma2;
      gl_FragColor = vec4(m[k], 1.0);
    }
  }
}
```

Listing 1.1. Fragment shader implementation of the Kuwahara filter.

the color channels:

$$\sigma_k^2 = s_{k,r}^2 + s_{k,g}^2 + s_{k,b}^2. \tag{1.1}$$

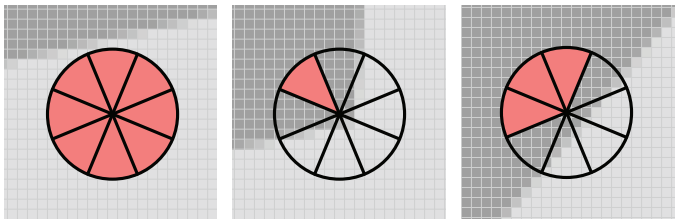
Now, the output of the Kuwahara filter is defined as the mean of a subregion with minimum variance:

$$F(x_0, y_0) := m_i, \quad i = \text{argmin}_k \, \sigma_k.$$

In Figure 1.3 the output of an image processed with the Kuwahara filter is shown. Clearly noticeable are the artifacts in the output. These are due to the use of rectangular subregions. In addition, the subregion selection process is unstable if noise is present or subregions have the same variance. This results in randomly chosen subregions and corresponding artifacts. A more detailed discussion of limitations of the Kuwahara filter can be found in [Papari et al. 07]. An implementation of the Kuwahara filter is shown in Listing 1.1.

1.3 Generalized Kuwahara Filtering

Several attempts have been made to address the limitations of the Kuwahara filter. In this section we present an implementation of the generalized Kuwahara filter, which was first proposed in [Papari et al. 07]. To overcome the limitations of the unstable subregion selection process, a new criterion is defined. Instead of selecting a single subregion, the result is defined as the weighted sum of the means of the subregions. The weights are defined based on the variances of the subregions. This results in smoother region boundaries and fewer artifacts. To improve this further, the rectangular subregions are replaced by smooth weighting functions over sectors of a disc (Figure 1.4). As can be seen in Figure 1.5, this significantly improves the quality of the output.



(a) Flat region (b) Corner (c) Edge

Figure 1.4. The generalized Kuwahara filter uses weighting functions defined over sectors of a disc. The filter response is defined as a weighed sum of the local averages, where more weight is given to those averages with low standard deviation.

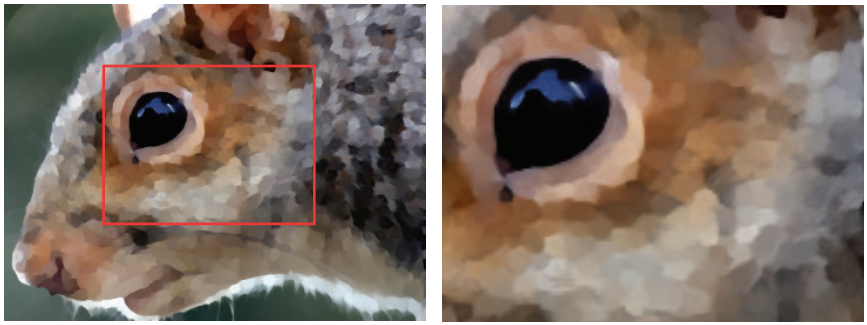


Figure 1.5. Output of the generalized Kuwahara filter.

We begin with the construction of the weighting functions. We divide the plane into N equal sectors by defining characteristic functions which are 1 over the sector and 0 otherwise:

$$\chi_k(x,y) = \begin{cases} 1 & \frac{(2k-1)\pi}{N} < \arg(x,y) \leq \frac{(2k+1)\pi}{N} \\ 0 & \text{otherwise} \end{cases} \qquad k = 0, \ldots, N-1.$$

Let

$$G_\sigma(x,y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$

be the Gaussian with standard deviation σ. To define smooth weighting functions, the characteristic functions of the different sectors χ_k are first convolved and then multiplied with a Gaussian:

$$w_k = \left(\chi_k \star G_{\sigma_s}\right) \cdot G_{\sigma_r} \tag{1.2}$$

The convolution smooths the characteristic functions such that they slightly overlap. The multiplication achieves a decay with increasing radius. Since $\sum_k w_k(x,y) = G_{\sigma_r}(x,y)$ for $(x,y) \in \mathbb{Z}^2$ the sum of the w_k is equivalent to a Gaussian filter.

Let f denote the input image. The weighted mean at a point $(x_0, y_0) \in \mathbb{Z}^2$ is then defined by

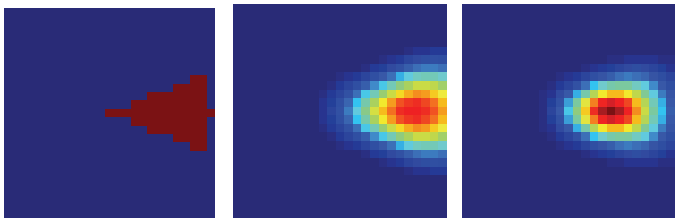
$$m_k = \sum_{(x,y)\in\mathbb{Z}^2} f(x,y)\, w_k(x-x_0, y-y_0),$$

and the weighted variance is given by

$$\begin{aligned} s_k^2 &= \sum_{(x,y)\in\mathbb{Z}^2} \big(f(x,y) - m_k\big)^2\, w_k(x-x_0, y-y_0) \\ &= \sum_{(x,y)\in\mathbb{Z}^2} f^2(x,y)\, w_k(x-x_0, y-y_0) - m_k. \end{aligned}$$

```
uniform sampler2D src;
uniform sampler2D K0;
uniform int N;
uniform int radius;
uniform float q;

const float PI = 3.14159265358979323846;

void main (void) {
  vec2 src_size = textureSize2D(src, 0);
  vec2 uv = gl_FragCoord.xy / src_size;

  vec4 m[8];
  vec3 s[8];
  for (int k = 0; k < N; ++k) {
    m[k] = vec4(0.0);
    s[k] = vec3(0.0);
  }

  float piN = 2.0 * PI / float(N);
  mat2 X = mat2(cos(piN), sin(piN), -sin(piN), cos(piN));

  for ( int j = -radius; j <= radius; ++j ) {
    for ( int i = -radius; i <= radius; ++i ) {
      vec2 v = 0.5 * vec2(i,j) / float(radius);
      if (dot(v,v) <= 0.25) {
        vec3 c = texture2D(src, uv + vec2(i,j) / src_size).rgb;
        for (int k = 0; k < N; ++k) {
          float w = texture2D(K0, vec2(0.5, 0.5) + v).x;

          m[k] += vec4(c * w, w);
          s[k] += c * c * w;

          v *= X;
        }
      }
    }
  }

  vec4 o = vec4(0.0);
  for (int k = 0; k < N; ++k) {
    m[k].rgb /= m[k].w;
    s[k] = abs(s[k] / m[k].w - m[k].rgb * m[k].rgb);

    float sigma2 = s[k].r + s[k].g + s[k].b;
    float w = 1.0 / (1.0 + pow(255.0 * sigma2, 0.5 * q));

    o += vec4(m[k].rgb * w, w);
  }
  gl_FragColor = vec4(o.rgb / o.w, 1.0);
}
```

Listing 1.2. Fragment shader implementation of the generalized Kuwahara filter.



(a) (b) (c)

Figure 1.6. Approximation of the weighting function w_0 for $N = 8$: (a) characteristic function χ_0; (b) characteristic function χ_0 convolved with Gaussian function G_{σ_s}; (c) finally, multiplication with Gaussian function G_{σ_r}.

Let σ be defined as in Equation (1.1). We set

$$\alpha_k = \frac{1}{1 + \left(255 \cdot \sigma_k^2\right)^{q/2}},$$

and define the output of the filter by

$$F(x_0, y_0) := \frac{\sum_k \alpha_k \, m_k}{\sum_k \alpha_k}. \tag{1.3}$$

The definition of the weighting factors α_k ensures that more weight is given to sectors with low standard deviation, i.e., those that are more homogeneous. This is similar to the approach of [Papari et al. 07], but avoids the indetermination, when some of the s_k are zero. The parameter q controls the sharpness of the output. We use $q = 8$ in our examples.

The weighting functions w_k are difficult to compute, because their computation requires convolution. A closed form solution is currently not known. Since the w_k do not depend on the pixel location, a straight forward approach would be to precompute them. We use a slightly different approach in our implementation, where all w_k are derived by bilinear sampling a texture map. We use this approach because it will easily generalize to anisotropic filtering which will be discussed in the next section. Let

$$R_\varphi = \begin{pmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{pmatrix}.$$

be the rotation matrix that performs a rotation by the angle φ in counter-clockwise order. Since $\chi_k = \chi_0 \circ R_{-2\pi k/N}$, and since Gaussian functions are rotational invariant, we have

$$\begin{aligned} w_k &= \left((\chi_0 \star G_{\sigma_s}) \cdot G_{\sigma_r}\right) \circ R_{-2\pi k/N} \\ &= w_0 \circ R_{-2\pi k/N}. \end{aligned}$$

Here, $\circ$ denotes composition of functions. For our implementation (Listing 1.2) we sample w_0 into a texture map K_0 of size 32×32. The sampling is performed using Equation (1.2) with

$$\sigma_r = \frac{1}{2} \cdot \frac{K_{\text{size}} - 1}{2} = 7.75,$$
$$\sigma_s = 0.33 \cdot \sigma_r,$$

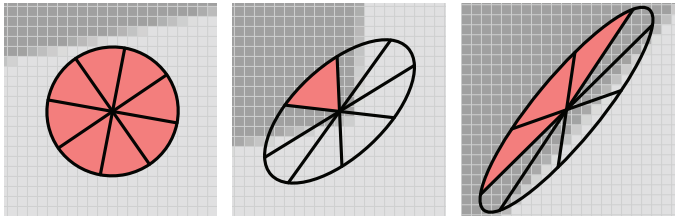
and the origin moved to the center of the texture map. Figure 1.6 illustrates the different steps of the computation. Now suppose that $r > 0$ denotes the desired filter radius. Then the weighting functions w_k can be approximated by

$$w_k(x, y) = K_0\left(\begin{pmatrix}0.5\\0.5\end{pmatrix} + \frac{R_{-2\pi k/N}(x, y)}{2r}\right).$$

1.4 Anisotropic Kuwahara Filtering

The generalized Kuwahara filter fails to capture directional features and results in clustering artifacts. The anisotropic Kuwahara filter addresses these limitations by adapting the filter to the local structure of the input. In homogeneous regions the shape of the filter should be a circle, while in anisotropic regions the filter should become an ellipse whose major axis is aligned with the principal direction of image features (Figure 1.7). As can be seen in Figure 1.8, this avoids clustering and moreover creates a painterly look for directional image features (Figure 1.1).

Figure 1.9 shows an overview of the algorithm. We begin with calculating the structure tensor and smooth it with a Gaussian filter. Local orientation and a measure for the anisotropy are then derived from the eigenvalues and eigenvectors of the smoothed structure tensor. Finally, the actual filtering is performed.



(a) Flat region (b) Corner (c) Edge

Figure 1.7. The anisotropic Kuwahara filter uses weighting functions defined over an ellipse, whose shape is based on the local orientation and anisotropy. The filter response is defined as a weighed sum of the local averages, where more weight is given to those averages with low standard deviation.

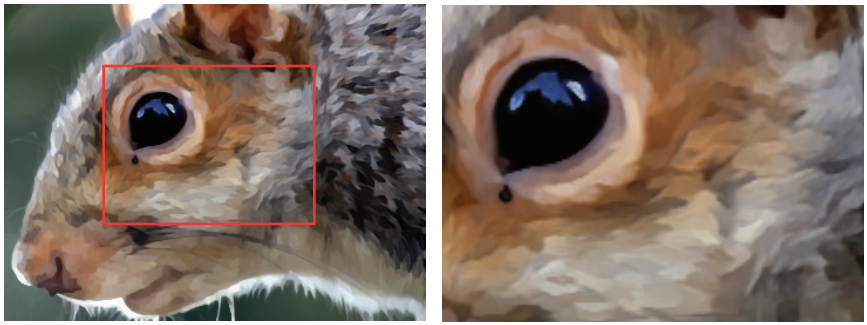


Figure 1.8. Output of the anisotropic Kuwahara filter.

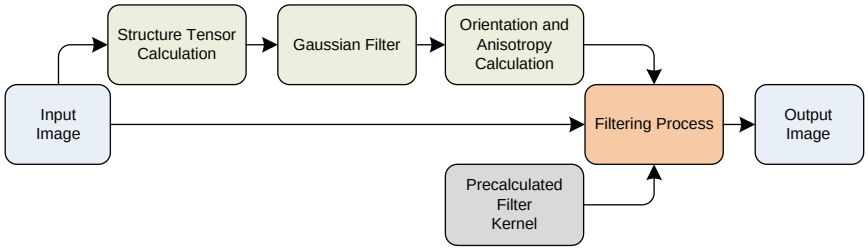


Figure 1.9. Schematic overview of the anisotropic Kuwahara filter.

1.4.1 Orientation and Anisotropy Estimation

The local orientation and anisotropy estimation is based on the *eigenvalues* and *eigenvectors* of the structure tensor [Brox et al. 06]. We calculate the structure tensor directly from the RGB values of the input [Kyprianidis and Döllner 08]. Let f be the input image and let

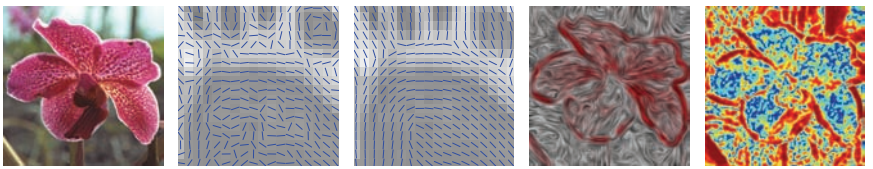
$$S_x = \frac{1}{4}\begin{pmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{pmatrix} \quad \text{and} \quad S_y = \frac{1}{4}\begin{pmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

be the horizontal and vertical convolution masks of the Sobel filter. Then, approximations of the partial derivatives of f can be calculated by

$$f_x = S_x \star f \quad \text{and} \quad f_y = S_y \star f,$$

where $\star$ denotes convolution. The structure tensor of f is then defined by

$$(g_{ij}) = \begin{pmatrix} f_x \cdot f_x & f_x \cdot f_y \\ f_x \cdot f_y & f_y \cdot f_y \end{pmatrix} =: \begin{pmatrix} E & F \\ F & G \end{pmatrix}.$$



(a) (b) (c) (d) (e)

Figure 1.10. (a) Original image. (b) Eigenvectors of the structure tensor. (c) Eigenvectors of the smoothed structure tensor. (d) Visualization of the eigenvectors of the smoothed structure tensor using line integral convolution. (e) Anisotropy (blue=low, red=high).

Here, $\cdot$ denotes the scalar product. The eigenvalues of the structure tensor correspond to the squared minimum and maximum rate of change of f. The eigenvectors correspond to the respective directions. Selecting the eigenvector corresponding to the minimum rate of change gives a vector field. As shown in Figure 1.10(b) this vector field has discontinuities. In order to smooth the vector field, smoothing of the structure tensor is performed. The result of applying a Gaussian filter is shown in Figure 1.10(c). Smoothing the structure tensor is a linear operation on the tensor, but the effect on the eigenvectors is highly nonlinear and corresponds geometrically to principal component analysis. In our examples, we use a Gaussian filter with standard deviation $\sigma = 2.0$. Note that we do not normalize the tensor. Therefore, structure tensors corresponding to edges with large gradient magnitude get more weight during smoothing. Hence, orientation information of edges is distributed into the neighborhood of the edges (Figure 1.10(d)).

The eigenvalues of the structure tensor are non-negative real numbers and are given by

$$\lambda_{1,2} = \frac{E + G \pm \sqrt{(E-G)^2 + 4F^2}}{2}.$$

The eigenvector oriented in direction of the minimum rate of change is given by

$$t = \begin{pmatrix} \lambda_1 - E \\ -F \end{pmatrix}.$$

We define local orientation by

$$\varphi = \arg t.$$

To measure the amount of anisotropy, we use the approach proposed in [Yang et al. 96]:

$$A = \frac{\lambda_1 - \lambda_2}{\lambda_1 + \lambda_2}.$$

The anisotropy A ranges from 0 to 1, where 0 corresponds to isotropic and 1 corresponds to entirely anisotropic regions (Figure 1.10(e)).

```
uniform sampler2D src;

void main (void) {
  vec2 src_size = textureSize2D(src, 0);
  vec2 uv = gl_FragCoord.xy / src_size;
  vec2 d = 1.0 / src_size;

  vec3 c = texture2D(src, uv).xyz;
  vec3 u = (
           -1.0 * texture2D(src, uv + vec2(-d.x, -d.y)).xyz +
           -2.0 * texture2D(src, uv + vec2(-d.x,  0.0)).xyz +
           -1.0 * texture2D(src, uv + vec2(-d.x,  d.y)).xyz +
           +1.0 * texture2D(src, uv + vec2( d.x, -d.y)).xyz +
           +2.0 * texture2D(src, uv + vec2( d.x,  0.0)).xyz +
           +1.0 * texture2D(src, uv + vec2( d.x,  d.y)).xyz
           ) / 4.0;

  vec3 v = (
           -1.0 * texture2D(src, uv + vec2(-d.x, -d.y)).xyz +
           -2.0 * texture2D(src, uv + vec2( 0.0, -d.y)).xyz +
           -1.0 * texture2D(src, uv + vec2( d.x, -d.y)).xyz +
           +1.0 * texture2D(src, uv + vec2(-d.x,  d.y)).xyz +
           +2.0 * texture2D(src, uv + vec2( 0.0,  d.y)).xyz +
           +1.0 * texture2D(src, uv + vec2( d.x,  d.y)).xyz
           ) / 4.0;

  gl_FragColor = vec4(dot(u, u), dot(v, v), dot(u, v), 1.0);
}
```

Listing 1.3. Fragment shader for calculating the structure tensor.

1.4.2 Filtering Process

The filtering is performed using the ideas from the previous section, but with redefined weighting functions.

We begin with calculating the bounding rectangle of an ellipse. An axis-aligned ellipse with major axis a and minor axis b is defined by

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1.$$

By rotating x and y by an angle φ we get the equation of a rotated ellipse:

$$\frac{(x\cos\varphi - y\sin\varphi)^2}{a^2} + \frac{(x\sin\varphi + y\cos\varphi)^2}{b^2} = 1.$$

This is a quadratic polynomial in two variables and by expanding and collecting terms it can be rewritten in normalized form:

$$P(x,y) = Ax^2 + By^2 + Cx + Dy + Exy + F = 0, \tag{1.4}$$

```
uniform sampler2D src;

void main (void) {
  vec2 uv = gl_FragCoord.xy / textureSize2D(src, 0);
  vec3 g = texture2D(src, uv).xyz;

  float lambda1 = 0.5 * (g.y + g.x +
      sqrt(g.y*g.y - 2.0*g.x*g.y + g.x*g.x + 4.0*g.z*g.z));
  float lambda2 = 0.5 * (g.y + g.x -
      sqrt(g.y*g.y - 2.0*g.x*g.y + g.x*g.x + 4.0*g.z*g.z));

  vec2 v = vec2(lambda1 - g.x, -g.z);
  vec2 t;
  if (length(v) > 0.0) {
    t = normalize(v);
  } else {
    t = vec2(0.0, 1.0);
  }

  float phi = atan(t.y, t.x);

  float A = (lambda1 + lambda2 > 0.0)?
    (lambda1 - lambda2) / (lambda1 + lambda2) : 0.0;

  gl_FragColor = vec4(t, phi, A);
}
```

Listing 1.4. Fragment shader for calculating the local orientation and anisotropy.

with

$$
\begin{aligned}
A &= a^2 \sin^2 \varphi + b^2 \cos^2 \varphi, \\
B &= a^2 \cos^2 \varphi + b^2 \sin^2 \varphi, \\
C &= 0, \\
D &= 0, \\
E &= 2(a^2 - b^2) \sin \varphi \cos \varphi, \\
F &= -a^2 b^2.
\end{aligned}
$$

The horizontal extrema are located where the partial derivative in the y-direction vanishes:

$$
\frac{\partial P}{\partial y} = 2By + Ex = 0 \quad \Leftrightarrow \quad y = \frac{-Ex}{2B}.
$$

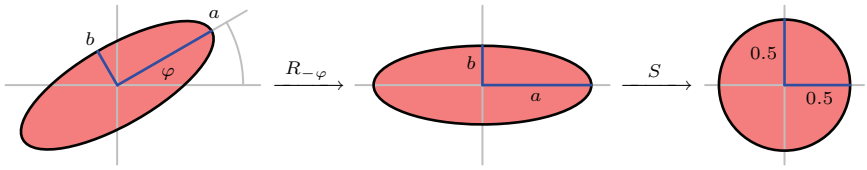


Figure 1.11. The mapping $SR_{-\varphi}$ defines a linear coordinate transform that maps an ellipse defined by major axis a, minor axis b and angle φ to a disc with radius 0.5.

```
vec4 t = texture2D(tfm, uv);
float a = radius * clamp((alpha + t.w) / alpha, 0.1, 2.0);
float b = radius * clamp(alpha / (alpha + t.w), 0.1, 2.0);

float cos_phi = cos(t.z);
float sin_phi = sin(t.z);

mat2 R = mat2(cos_phi, -sin_phi, sin_phi, cos_phi);
mat2 S = mat2(0.5/a, 0.0, 0.0, 0.5/b);
mat2 SR = S * R;

int max_x = int(sqrt(a*a * cos_phi*cos_phi +
                     b*b * sin_phi*sin_phi));
int max_y = int(sqrt(a*a * sin_phi*sin_phi +
                     b*b * cos_phi*cos_phi));

for (int j = -max_y; j <= max_y; ++j) {
  for (int i = -max_x; i <= max_x; ++i) {
    vec2 v = SR * vec2(i,j);
    if (dot(v,v) <= 0.25) {
      vec3 c = texture2D(src, uv + vec2(i,j) / src_size).rgb;
      for (int k = 0; k < N; ++k) {
        float w = texture2D(K0, vec2(0.5, 0.5) + v).x;

        m[k] += vec4(c * w, w);
        s[k] += c * c * w;

        v *= X;
      }
    }
  }
}.
```

Listing 1.5. Fragment shader implementation of the variance computation of the anisotropic Kuwahara filter.

Substituting y into Equation (1.4) yields

$$\left(A - \frac{E^2}{4B}\right)x^2 + F = 0.$$

The horizontal extrema of the ellipse are therefore given by

$$x = \pm\sqrt{\frac{F}{\frac{E^2}{4B} - A}} = \pm\sqrt{a^2\cos^2\varphi + b^2\sin^2\varphi}.$$

A similar calculation gives the vertical extrema:

$$y = \pm\sqrt{a^2\sin^2\varphi + b^2\cos^2\varphi}.$$

Suppose $r > 0$ is the desired filter radius. Let φ be the local orientation and let A be the anisotropy as defined in the previous section. We use the method proposed in [Pham 06] to define an elliptical filter shape. To adjust the eccentricity depending on the amount of anisotropy, we set

$$a = \frac{\alpha + A}{\alpha}\,r \quad \text{and} \quad b = \frac{\alpha}{\alpha + A}\,r.$$

The parameter $\alpha > 0$ is a tuning parameter. For $\alpha \to \infty$ the major axis a and the minor axis b converge to 1. We use $\alpha = 1$ in all examples, which results in a maximum eccentricity of 4. The ellipse defined by a, b and φ has its major axis aligned to the local image orientation. It has high eccentricity in anisotropic regions and becomes a circle in isotropic regions.

Now let

$$S = \begin{pmatrix} \frac{1}{2a} & 0 \\ 0 & \frac{1}{2b} \end{pmatrix},$$

then the mapping $SR_{-\varphi}$ maps points from the ellipse to a disc of radius 0.5 (Figure 1.11). Hence, the weighting functions over the ellipse can be defined by:

$$w_k(x,y) = K_0\left(\begin{pmatrix}0.5\\0.5\end{pmatrix} + R_{-2\pi k/N}SR_{-\varphi}(x,y)\right).$$

The filter response is defined as in the case of the generalized Kuwahara filter by Equation (1.3).

The implementation of the anisotropic Kuwahara filter is very similar to the implementation of the generalized Kuwahara filter (Listing 1.2). Therefore, only the variance computation of the anisotropic Kuwahara filter is shown in Listing 1.5. In Listing 1.6 an optimized variance computation for $N = 8$ is shown.

```
{
  vec3 c = texture2D(src, uv).rgb;
  float w = texture2D(K0123, vec2(0.5, 0.5)).x;
  for (int k = 0; k < N; ++k) {
    m[k] +=  vec4(c * w, w);
    s[k] += c * c * w;
  }
}

for (int j = 0; j <= max_y; ++j)  {
  for (int i = -max_x; i <= max_x; ++i) {

    if ((j !=0) || (i > 0)) {
      vec2 v = SR * vec2(i,j);

      if (dot(v,v) <= 0.25) {
        vec3 c0 = texture2D(src,uv + vec2(i,j)/src_size).rgb;
        vec3 c1 = texture2D(src,uv - vec2(i,j)/src_size).rgb;

        vec3 cc0 = c0 * c0;
        vec3 cc1 = c1 * c1;

        vec4 w0123 = texture2D(K0123, vec2(0.5, 0.5) + v);
        for (int k = 0; k < 4; ++k) {
          m[k] += vec4(c0 * w0123[k], w0123[k]);
          s[k] += cc0 * w0123[k];

          m[k+4] += vec4(c1 * w0123[k], w0123[k]);
          s[k+4] += cc1 * w0123[k];
        }

        vec4 w4567 = texture2D(K0123, vec2(0.5, 0.5) - v);
        for (int k = 0; k < 4; ++k) {
          m[k+4] += vec4(c0 * w4567[k], w4567[k]);
          s[k+4] += cc0 * w4567[k];

          m[k] += vec4(c1 * w4567[k], w4567[k]);
          s[k] += cc1 * w4567[k];
        }
      }

    }

  }
}
```

Listing 1.6. Fragment shader implementation of the variance computation of the anisotropic Kuwahara filter (optimized for $N = 8$).

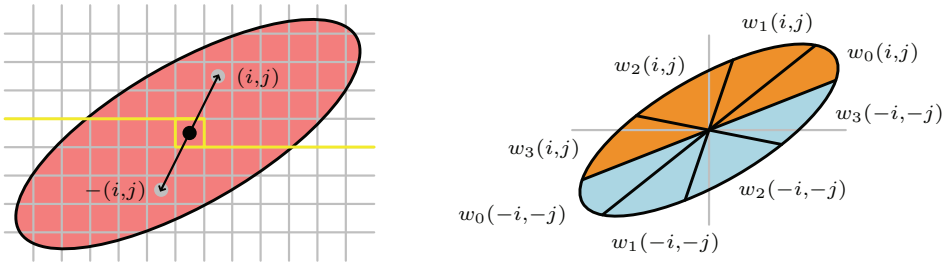


Figure 1.12. Symmetry about the origin of an ellipse (left). For $N = 8$ the values of the weighting functions can be fetched using two RGBA texture lookups (right).

In order to reduce the number of texture lookups, four weights are packed into a RGBA texture map. This texture map is constructed by sampling the weighting functions w_0,...,w_3 from section 1.3 as explained. Furthermore, the property of the ellipse being symmetric about the origin is used (Figure 1.12). Note that for $N = 8$ and $k = 0, \ldots, 3$ we have

$$w_{k+4}(x, y) = w_k(-x, -y).$$

1.5 Conclusion

In this chapter we have presented a GPU implementation of the anisotropic Kuwahara filter. Guided by the smoothed structure tensor, the anisotropic Kuwahara filter generates a feature-preserving, direction-enhancing look.

Unlike existing nonlinear smoothing filters, the anisotropic Kuwahara filter is robust against high-contrast noise and avoids overblurring in low-contrast areas, providing a consistent level of abstraction across the image. It also ensures outstanding temporal coherence when applied to video, even with per-frame filtering.

Acknowledgments

This work was supported by the German Research Foundation (DFG), Grant DO 697/5-1.

Original photographs from flickr.com kindly provided under Creative Commons license by Keven Law (Figure 1.3, 1.5, 1.8) and Tambako the Jaguar (Figure 1.1).

Bibliography

[Brox et al. 06] T. Brox, R. Boomgaard, F. Lauze, J. Weijer, J. Weickert, P. Mrázek, and P. Kornprobst. "Adaptive Structure Tensors and Their Applications." In *Visualization and Processing of Tensor Fields*, pp. 17–47. Springer, 2006.

[Comaniciu and Meer 02] D. Comaniciu and P. Meer. "Mean Shift: A Robust Approach Toward Feature Space Analysis." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24:5 (2002), 603–619.

[Kuwahara et al. 76] M. Kuwahara, K. Hachimura, S. Eiho, and M. Kinoshita. *Digital Processing of Biomedical Images*, pp. 187–203. Plenum Press, 1976.

[Kyprianidis and Döllner 08] J. E. Kyprianidis and J. Döllner. "Image Abstraction by Structure Adaptive Filtering." In *Proc. EG UK Theory and Practice of Computer Graphics*, pp. 51–58. Eurographics Association, 2008.

[Kyprianidis et al. 09] J. E. Kyprianidis, H. Kang, and J. Döllner. "Image and Video Abstraction by Anisotropic Kuwahara Filtering." *Computer Graphics Forum* 28:7. Special Issue on Pacific Graphics 2009.

[Papari et al. 07] G. Papari, N. Petkov, and P. Campisi. "Artistic Edge and Corner Enhancing Smoothing." *IEEE Transactions on Image Processing* 16:10 (2007), 2449–2462.

[Pham 06] T. Q. Pham. "Spatiotonal Adaptivity in Super-Resolution of Undersampled Image Sequences." Ph.D. thesis, Quantitative Imaging Group, Delft University of Technology, 2006.

[Tomasi and Manduchi 98] C. Tomasi and R. Manduchi. "Bilateral Filtering for Gray and Color Images." In *Proceedings International Conference on Computer Vision (ICCV)*, pp. 839–846, 1998.

[Yang et al. 96] G. Z. Yang, P. Burger, D. N. Firmin, and S. R. Underwood. "Structure Adaptive Anisotropic Image Filtering." *Image and Vision Computing* 14:2 (1996), 135–145.