TEKNILLINEN KORKEAKOULU
Tietotekniikan ostasto

Andreas Gustafsson

# Interactive Image Warping

Diplomityö, joka on jätetty opinnäyttenä tarkastettavaksi
diplomi-insinöörin tutkintoa varten Espoossa 31.5.1993.

| | |
|---|---|
| Työn valvoja | Heikki Saikkonen |
| Työn ohjaaja | Ken Rimey |

HELSINKI UNIVERSITY OF TECHNOLOGY MASTER'S THESIS ABSTRACT

*Author:* Andreas Gustafsson

*Name of the thesis:* Interactive Image Warping

*Date:* May 31, 1993 *Number of pages:* 59

*Faculty:* Faculty of Information Technology *Professorship:* Tik-76 Computer Science

*Supervisor:* Prof. Heikki Saikkonen

*Instructor:* Ph.D. Ken Rimey

This thesis addresses the subject of *digital image warping*, the process of geometrically transforming digital images.

Warping digital images involves a resampling process which may introduce aliasing and other forms of image quality degradation. To minimize the loss of quality, filtering is necessary. The thesis discusses the theory of resampling filters and surveys a number of well-known methods for performing image filtering, ranging from simple interpolation and decimation filters to the high-quality space-variant Elliptical Weighted Average (EWA) filter of Greene and Heckbert. A variant of the EWA method suited for warping tasks involving simultaneous magnification and minification is proposed and contrasted with Heckbert's method based on Gaussian filter kernels.

The thesis also describes the design and implementation of a new kind of highly interactive image warping system aimed at creative graphic design applications. The system allows a complex warp to be built incrementally from a series of simple, local transformations. A graphical user interface lets the user specify each transformation with a single mouse stroke, during which the warped image is displayed in real time. Performance results from a prototype implementation are presented.

TEKNILLINEN KORKEAKOULU DIPLOMITYÖN TIIVISTELMÄ

*Tekijä:* Andreas Gustafsson

*Työn nimi:* Digitaalisten kuvien vuorovaikutteiset geometriset muunnokset

*Päivämäärä:* 31.05.1993 *Sivumäärä:* 59

*Osasto:* Tietotekniikan ostasto *Professuuri:* Tik-76 Tietojenkäsittelyoppi

*Työn valvoja:* Prof. Heikki Saikkonen

*Työn ohjaaja:* Ph.D. Ken Rimey

Tässä työssä käsitellään geometristen muunnosten tekemistä digitaalisille kuville.

Digitaalisen kuvan geometrisen muuntamisen yhteydessä kuva näytteistetään uudelleen (engl. resampling), mikä voi johtaa aliasointiin ja muihin kuvan laatua heikentäviin ilmiöihin. Työssä käsitellään kuvien näytteistyksen ja suodatuksen teoria ja kuvataan siihen pohjautuvia käytännön suodatusmenetelmiä yksinkertaisista interpolointi- ja desimointisuotimista Greenen ja Heckbertin Elliptical Weighted Average (EWA) -menetelmään. Esitetään myös EWA-menetelmän muunnelma niitä kuvamuunnoksia varten joissa kuvaa yhtä aikaa suurennetaan ja pienennetään. Tätä EWA-menetelmän muunnelmaa verrataan Heckbertin vastaavaan Gauss-muotoisia suotimia käyttävään menetelmään.

Työssä esitellään myös uudenlainen vuorovaikutteinen, luovaan graafiseen suunnitteluun tarkoitettu kuvamuunnosjärjestelmä. Tässä järjestelmässä monimutkainenkin kuvamuunnos voidaan rakentaa asteittain sarjasta yksinkertaisia, paikallisia perusmuunnoksia. Graafisen käyttöliittymän avulla käyttäjä määrittelee jokaisen perusmuunnoksen yhdellä hiiren liikkeellä, jonka aikana muunnettu kuva näytetään reaaliajassa. Järjestelmän prototyypin toteutus kuvataan ja sen suorituskyvystä esitetään mittaustuloksia.

## Acknowledgements

This work was done at Unda Oy. I wish to thank Unda Oy and all of its personnel for supporting this work and for creating a very stimulating working environment. Special thanks go to my advisor Ken Rimey, who has provided much sound advice and many helpful comments. Tapio Takala provided additional feedback. I also thank my thesis supervisor, Prof. Heikki Saikkonen, for his support and for initially putting me in touch with Unda.

Espoo, May 31, 1993

Andreas Gustafsson

## *Table of contents*

# Chapter 1. Introduction

This thesis concerns itself with the field of *image warping*. The following sections serve as a short introduction to the field and its applications.

The term “image warping” refers to the process of geometrically transforming two-dimensional pictures, or *images*. Although the word “warp” may seem to suggest a radical distortion, the term “image warping” encompasses the whole range of transformation from simple ones such as scaling or rotation to complex, irregular warps.

Geometric image transformations performed by optical or mechanical means could conceivably be considered cases of image warping, but in practice the term is used specifically for transformation performed by electronic, and especially by digital means. This thesis mainly addresses methods for performing image warping using general-purpose digital computers.

## 1.1. Applications of image warping

The applications of digital image warping are many and diverse. Almost any application of digital image processing involves at least an occasional need to change the location, scale or orientation of the image. All these operations are simple instances of image warping.

One of the oldest and still one of the most important applications of digital image warping is in remote sensing. Photographs taken from aircraft or satellites suffer from a number of geometric distortions caused by lens imperfections, perspective, curvature of the earth, etc. Digital image warping is used in correcting these distortions and in aligning multiple overlapping images.

In the field of three-dimensional computer graphics, a technique called *texture mapping* [4, 16] is used to give models of three-dimensional objects a more natural appearance by mapping two-dimensional textures onto their surfaces to simulate features like wood grain or surface bumpiness. When these three-dimensional objects are then rendered on a two-dimensional display, the overall effect is that of a mapping from the two-dimensional texture space to the two-dimensional screen space, i.e., an image warp.

Image warping is also increasingly being used as an artistic tool. Most software packages for painting and photo retouching contain at least some kind of image warping facilities and/or warping-based special effects.

Recently, much attention has focused on a special effects technique known as *morphing* (from *metamorphosis*, although the technique apparently predates the term) [2, 13]. In morphing, image warping techniques are combined with key-frame animation and cross-fading to create a convincing illusion of one object transforming smoothly into another. Morphing has been used as an element in many recent motion pictures, music videos and television commercials.

As an example of a more exotic application, image warping techniques have found use in research on the human visual system. Here image warping is used in modeling the approximately conformal mapping of images from the retina to the visual cortex [9].

## 1.2. Key literature

Digital image warping is an area of active research, and its literature is extensive although scattered among a large number of publications in different fields including computer graphics, image processing, digital signal processing, and medical imaging.

The most comprehensive survey of the field to date is George Wolberg's book *Digital Image Warping* published in 1989 [28]. This book attempts to bring together the scattered work on image warping in a single coherent framework. It gives particular attention to two-pass warping methods and contains a very extensive bibliography.

Wolberg's book received a critical review in an IEEE CG&A article by Paul Heckbert [29], which in turn caused a heated debate published in the form of letters to the editor in a subsequent issue [30].

Heckbert has himself published a significant amount of survey-type material on image warping, including a survey of texture mapping techniques in 1986 [16] and his master's thesis *Fundamentals of Texture Mapping and Image Warping* in 1989 [17].

## 1.3. This thesis

This thesis treats image warping from both theoretical and practical viewpoints. As an introduction, Chapter 2 presents some basic classes of geometric transformations and strategies for implementing them.

Chapter 3 discusses sampling and filtering, beginning with a short treatment of the basic digital signal processing theory necessary for understanding the need for filtering as a part of the image warping process. Section 3.2 discusses issues particular to two-dimensional sampling and filtering. Section 3.3 treats the theory of *unified resampling filters*, where the two stages of filtering needed for a general warping task are combined into a single filter capable of handling arbitrary combinations of magnification and minification. The remainder of Chapter 3 (sections 3.4 to 3.7) discusses practical filtering techniques. A number of commonly used, simple filtering techniques are surveyed in Section 3.4, and the high-quality Elliptical Weighted Average (EWA) filter of Greene and Heckbert is discussed in Section 3.5. A method for using the EWA as a unified resampling filter with an arbitrary circularly symmetric filter kernel is proposed in Section 3.6 and contrasted to Heckbert's Gaussian EWA filter in Section 3.7.

Chapter 4 presents a new, highly interactive approach to image warping for creative graphic design applications, demonstrated in an implementation. This implementation also applies several of the filtering methods discussed in Chapter 3 in a practical context, including the proposed variation of the EWA method.

# Chapter 2. Fundamentals of image warping

## 2.1. Mapping functions

An image warp is defined by a mapping from the coordinate space of a *source image* $(u, v)$ to the coordinate space of a *destination image* $(x, y)$. If the destination coordinates are specified as functions of the source coordinates, the mapping is called a *forward mapping*:

$$x = x(u, v), \quad y = y(u, v)$$

If the source coordinates are specified as functions of the destination coordinates, the mapping is called a *inverse mapping*:

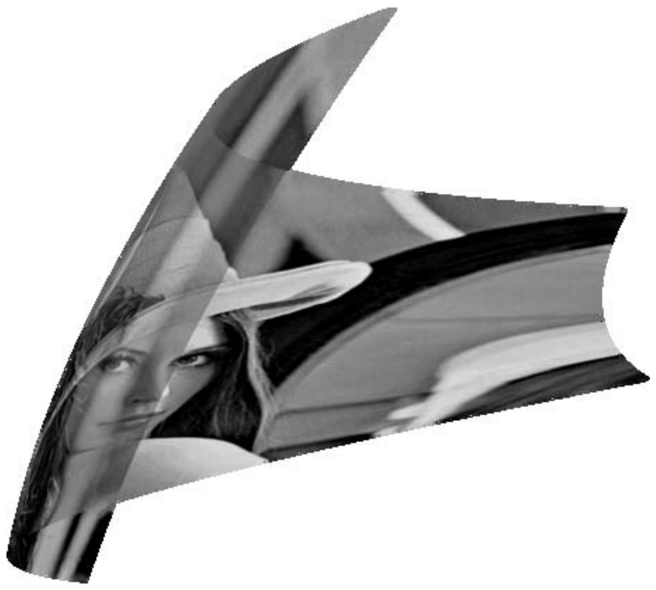
$$u = u(x, y), \quad v = v(x, y)$$

Forward and inverse mappings actually describe different classes of warps: a forward mapping is a many-to-one mapping and it can therefore model warps where several points in the source image map to the same point in the destination image. On the other hand, an inverse, or one-to-many, mapping can model warps where several points in the destination image map to the same point in the source image. An example of the former class of of warps is imaging a texture on a folded surface; of the latter, a reflection in an irregular, curved mirror. Figure 1 illustrates these two cases.

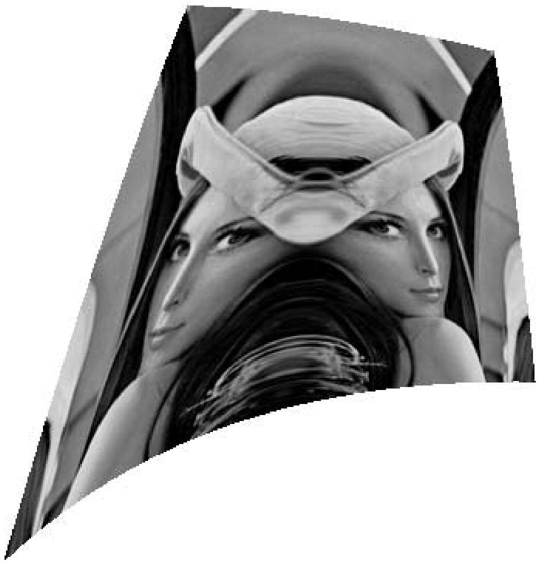
Many practical mappings are of course bijections and may therefore be specified arbitrarily as either forward or inverse mappings.

In addition to this conceptual difference, forward and inverse mappings also suggest different implementation techniques: A warp described by a forward mapping is most naturally performed by scanning the source image pixel by pixel, calculating the corresponding location in the destination image by evaluating the mapping function, and painting that location in the destination image with the color of the source pixel. On the other hand, a warp described by an inverse mapping would be performed by scanning the destination image pixel by pixel, calculating the corresponding location in the source image by evaluating the mapping function, and painting the destination pixel with the color of the calculated source location.

Regardless of whether a forward or inverse transformation is used, the spatial transformation may map the area corresponding to a pixel onto



*a. Forward mapping*



*b. Inverse mapping*

*Figure 1: Non-one-to-one warps of a rectangular image*

an area that is not centered on integer pixel coordinates and may be of a different size or shape than the mapped pixel. In forward mapping this means that the color of the single source pixel has to be distributed over several destination pixels; in inverse mapping, the destination pixel has to be interpolated from several source pixels. If this is done improperly, e.g. by simply rounding the mapped coordinates to the nearest pixel, the result will be nonuniform intensity and/or holes in the destination image in the forward mapping case, and jagged edges or other aliasing artifacts in the inverse mapping case. Chapter 3 discusses the sampling and filtering theory that is central in avoiding such degradation of image quality, and presents some practical methods for implementing necessary resampling filters.

## 2.2. *Classes of spatial transformations*

This section presents some important classes of spatial transformations used in image warping.

### 2.2.1. *Affine transformations*

A simple class of spatial transformations is that of the *affine* transformations, which consists of translation, rotation, scaling, shear, and combinations of the above.

The class of affine transformations is closed under composition, and the inverse of a (non-degenerate) affine transformation is another affine transformation. Mathematically, the affine transformations form a group.

An affine transformation can be fully described by six parameters, which are usually represented by a 3 by 2 transformation matrix. This means that an affine transformation can be unambiguously defined by establishing a correspondence between three points in the source image and three points in the destination image.

### 2.2.2. *Bilinear and perspective transformations*

The *bilinear* and *perspective* (or *projective*) transformations both have eight parameters, and can therefore be used to map an arbitrary quadrilateral in the source image onto another arbitrary quadrilateral in the destination image.

When the bilinear transformation is used to map the unit square to a general quadrilateral, it will preserve any straight lines parallel to the $x$ and $y$ axis and relative distances along such lines (e.g., the midpoint is mapped onto the midpoint). However, diagonal lines in the source image will generally be mapped to curved shapes. The bilinear transformation is *not* closed under composition or inversion.

Perspective transformations preserve lines in all directions, but distance relationships are not preserved. Like the affine transformations, the class of perspective transformations is closed under composition and inverse.

A complete treatment of the properties of affine, bilinear and perspective transformations can be found in Heckbert's master's thesis [17].

### 2.2.3. Polynomial transformations

A polynomial image transformation is one where the source image coordinates are defined as bivariate polynomials in the destination image coordinates [28]:

$$u = \sum_{i=0}^{N} \sum_{j=0}^{N-i} a_{ij} x^i y^j, \quad v = \sum_{i=0}^{N} \sum_{j=0}^{N-i} b_{ij} x^i y^j$$

It is also possible to define the destination image coordinates as polynomials in the source image coordinates, giving rise to a class of transformations different from the above. Examples of both forward and inverse second-degree polynomial mappings were shown in Figure 1.

Polynomial transformations are commonly used for image registration and correction of distortions in remote sensing applications. The polynomial coefficients are often inferred from known positions of a few points in the image (e.g., landmarks) using least-squares methods.

When local control over the mapping is required, piecewise polynomial methods can be used. The mapping $u = u(x, y)$ can be seen as defining a surface in $xyu$-space, and similarly for $v = v(x, y)$, so in this case the problem of inferring the parameters of the mapping becomes one of fitting two smooth spline surfaces to their respective sets of control points [28].

A freely distributable implementation of an interactive polynomial-based image warper is available as part of the Khoros image processing system from the University of New Mexico [24].

## 2.3. Multi-pass methods

In Section 2.1 it was noted that a warping system might scan the input image in a fixed order (e.g., scanline order), producing output in arbitrary order defined by the forward mapping. Alternatively, it might generate the output image in a fixed order, accessing the input image in an arbitrary order defined by the inverse mapping.

A third possibility is that of using *multi-pass* methods, where the mapping is decomposed into two or more simpler mappings where both input and output scanning order can be more closely controlled. For instance, a typical two-pass method will first warp each horizontal line of the input image into a temporary image, and then warp each vertical line of the temporary image into the destination image.

This has the advantage that filtering can be done using one-dimensional rather than two-dimensional filters, which leads to a simpler implementation and reduces the computational complexity of filtering with a filter kernel of radius $r$ from $O(r^2)$ to $O(r)$ multiplications per output pixel. The predictable, scanline-oriented memory access patterns of the multi-pass methods also ease the implementation of pipelined special-purpose warping hardware. This makes multi-pass warping the method of choice in high-speed applications such as real-time warping of video images.

On the other hand, multi-pass warping also suffers from disadvantages relative to single-pass methods:

- It accesses the image in both row and column order, even in the case where the mapping doesn't include any rotational components. The column-wise accesses can cause a severe performance degradation on general-purpose computers by causing numerous cache misses or page faults.
- Nontrivial mapping functions can be difficult to decompose, and the decomposed mapping can be more expensive to calculate than the original one.
- In two-pass warping, a large area in the source image may sometimes map to a small area in the intermediate image, and this small area is then mapped to a large area in the destination image. This causes information in the image to be lost. This phenomenon is called the *bottleneck problem*. It can be overcome by carefully considering the order in which the passes are performed and possibly transposing the input or output image before warping.
- Filtering is restricted to separable filter kernels.

The advantage in computational complexity is also somewhat questionable in practice because typical filter kernels are quite small. The algorithm's asymptotic performance may be less important than the constant of proportionality.

Although multi-pass methods are usually superior to single-pass methods when implemented on special-purpose hardware, they are not necessarily faster when implemented in software on general-purpose computers, and they are certainly more complex. This led the author to choose the single-pass, inverse mapping method for the warping system described in Chapter 4.

# Chapter 3. *Image reconstruction and filtering for image warping*

A *continuous image* such as a photograph can be considered as a real-valued light intensity which is a function of two real-valued coordinates: $I = f(x, y)$. To process such an image by means of a digital computer it must first be converted into a *discrete*, or *digital*, image where both the coordinates and the intensity are represented as discrete values with a finite precision.

The discretizing of the coordinates is referred to as *sampling* and involves replacing the continuous intensity function by one defined only at a finite set of points, called picture elements or *pixels*. The pixels are usually arranged in a uniformly spaced rectangular grid, but other arrangements such as hexagonal grids are sometimes used. In the case of moving images, the additional dimension of time also needs to be discretized through time sampling.

The discretizing of the intensity values, or *quantization*, involves representing the intensity of each pixel by a finite-precision number.

Various combinations of continuous and discrete images also occur. For instance, television uses images that are continuous in intensity and in the horizontal spatial dimension, but discrete in the time and vertical dimensions.

Because image warping deals more with the spatial relationships between pixels than their actual intensities, it is mostly concerned with the effects of the spatial sampling rather than intensity quantization. Therefore, the theory of image warping can be built on an abstraction where the pixel values are considered to be real numbers.

## 3.1. *Sampling and filtering in one dimension*

This section presents the fundamentals of sampling, borrowing some theory and terminology from the field of digital signal processing, where sampling plays a central role. We will first consider the sampling of one-dimensional signals and then extend the results to two dimensions.

In its simplest form, signal processing deals with signals that are one-dimensional functions of time. Sampling a *continuous signal* at a finite number of points in time yields a *discrete-time* signal, and a signal that is discrete in both time and amplitude is called a *digital signal.*

To understand the effects of sampling, it is helpful to consider the signal's Fourier transform:

$$F(u) = \int_{-\infty}^{\infty} f(x)\, e^{-2\pi i u x}\, dx$$

The Fourier transform $F(u)$ of a signal $f(x)$ is also called the *spectrum*, or *frequency domain* representation (because if the dimension of $x$ is time, the dimension of $u$ is 1/time, i.e., frequency). In the case of image processing, the signal is a function of space rather than time, and in that case the domain of the Fourier transform is called *spatial frequency*.

Table 1 summarizes some important properties of the one-dimensional Fourier transform. The symbol $*$ is used to denote convolution:

$$(f * g)(x) = \int_{-\infty}^{\infty} f(t)\, g(x - t)\, dt$$

| | Function | Fourier transform |
|---|---|---|
| Definition | $f(x)$ | $F(u) = \int_{-\infty}^{\infty} f(x)\, e^{-2\pi i u x}\, dx$ |
| Inverse | $f(x) = \int_{-\infty}^{\infty} F(u)\, e^{2\pi i x u}\, du$ | $F(u)$ |
| Convolution | $(f * g)(x)$ | $F(u)\, G(u)$ |
| Impulse train | $f(x) = \sum_{n=-\infty}^{\infty} \delta(x - nT)$ | $F(u) = \frac{1}{T} \sum_{n=-\infty}^{\infty} \delta(u - n\frac{1}{T})$ |

*Table 1: Some properties of the Fourier transform*

One way to model the sampling process mathematically is to think of it as multiplying the signal with an infinite train of Dirac impulses $\delta(t)$. Sampling the continuous signal $f_c(x)$ at regular intervals $T$ (i.e., with the frequency or *sampling rate* $f_s = 1/T$), we get the sampled, or *discrete* signal

$$f_d(x) = f_c(x) \sum_{n=-\infty}^{\infty} \delta(x - nT)$$

Because multiplication in the time domain corresponds to convolution in the frequency domain, and the Fourier transform of an infinite impulse train with period $T$ is another infinite impulse train with period $1/T$, the Fourier transform of the sampled signal is the Fourier transform of the continuous signal convolved with an impulse train of period $1/T$:

$$F_d(u) = F_c(u) * (\frac{1}{T} \sum_{n=-\infty}^{\infty} \delta(u - n\frac{1}{T}))$$

This can also be expressed as

$$F_d(u) = \sum_{n=-\infty}^{\infty} \frac{1}{T} F_c(u - n\frac{1}{T})$$

i.e., the spectrum of the sampled signal is the sum of an infinite number of scaled copies of the original spectrum shifted by multiples of $1/T$. This is illustrated in Figure 2.

As long as the replicated spectra don't overlap, all the information in the original spectrum is preserved, but any amount of overlap causes information to be lost. Such spectral overlap is called *aliasing* because any spatial frequency component $f_s/2 + \delta$ becomes indistinguishable from, or *aliases to*, the frequency $f_s/2 - \delta$.

This is a manifestation of the well-known *sampling theorem*: a sampled signal can be reconstructed exactly if the sampling rate is at least twice the highest frequency component in the signal.

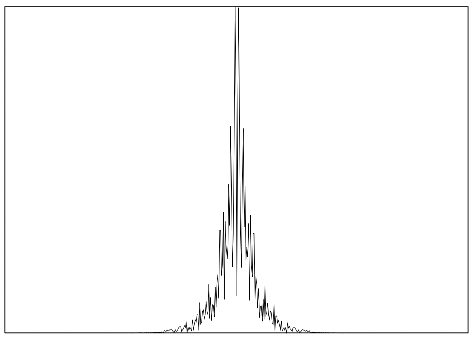
### 3.1.1. Prefilters

Most real-world signals contain arbitrarily high frequency components, and therefore cannot be reconstructed exactly when sampled with a finite sampling frequency. But in practical applications like processing of images or sound, there is no need for an exact reconstruction; rather, we only need an approximation that is perceived as equal to the original by a human viewer or listener. This means that we only need to sample the signals at a sufficiently high rate to accurately reproduce the smallest details (highest spatial frequencies) perceptible by the human eye or, in the case of audio, the highest frequencies perceptible by the human ear. Specifically, we must sample at no less than twice the highest perceptible frequency.

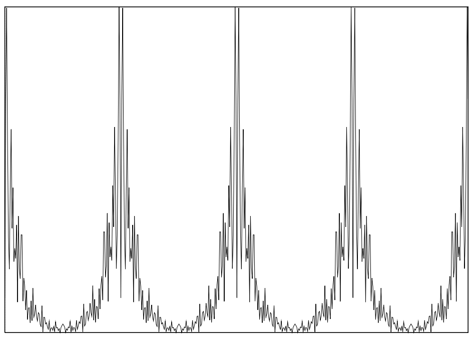
Even if this criterion is fulfilled, the signal needs to be bandlimited to $f_s/2$ before sampling to avoid aliasing. Otherwise, higher (imperceptible) frequencies in the sampled signal will alias to lower (thus perceptible) frequencies, causing distortion of the signal.

This removal of high-frequency components is called *antialiasing* and is performed by an *antialiasing filter*, or *prefilter*. It *must* be done *before* sampling, because after sampling the aliases of high-frequency components are indistinguishable from the signal itself.

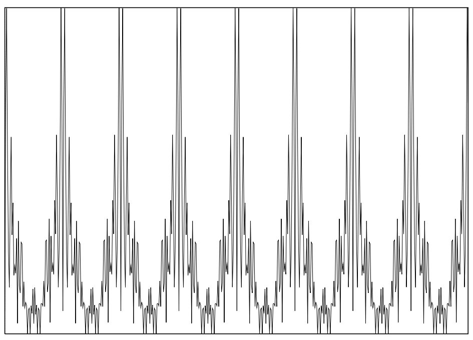
A *filter* is the general term for a device which selectively emphasizes or attenuates different frequency components in a signal according to a desired



*a. Original spectrum*



*a. Effect of sampling at adequate sampling rate (negligible aliasing)*



*c. Effect of sampling at inadequate sampling rate (considerable aliasing)*

*Figure 2: Spectra of sampled signals, with aliasing*

*frequency response*. In effect, the signal is multiplied in the frequency domain by the filter's frequency response. This corresponds to a convolution in the time domain with the inverse Fourier transform of the frequency response, the *filter kernel* or *impulse response* (so called because it is the output produced when the input to the filter is an impulse).

A filter which removes high-frequency components while passing through low-frequency components, such as is needed for antialiasing, is called a *lowpass filter*. In the one-dimensional case, the ideal lowpass filter is one which completely removes all frequency components above its *cutoff frequency* $f_c$ without affecting the rest of the signal, giving the frequency response

$$F(u) = \begin{cases} 1, & |u| \leq f_c; \\ 0, & |u| > f_c. \end{cases}$$

The corresponding impulse response is

$$f(x) = \frac{1}{f_c} \operatorname{sinc}(f_c\, x)$$

where

$$\operatorname{sinc}(x) = \frac{\sin \pi x}{\pi x}$$

The realization of the antialiasing filter varies according to the kind of signal that is being sampled, and in practice the ideal sinc($x$) shape can never be achieved but only approximated, often crudely. For instance, when sampling continuous signals in electronic form, electronic lowpass filters of varying quality are used (although many modern systems use *oversampling* techniques where part of the filtering is performed digitally).

When sampling images using an array of optical sensors such as a CCD camera or scanner, blurring in the optics usually provides a sufficient lowpass filtering. A lens has the side effect of convolving the image with its *point spread function*, which typically has an approximately Gaussian shape, and a well-designed image sampling system takes advantage of this blurring to attenuate high spatial frequencies that could otherwise cause aliasing.

### 3.1.2. Postfilters

After processing, the digital signals must usually be converted back into continuous signals. This is done by passing the series of impulses representing the samples through another lowpass filter, the *reconstruction*

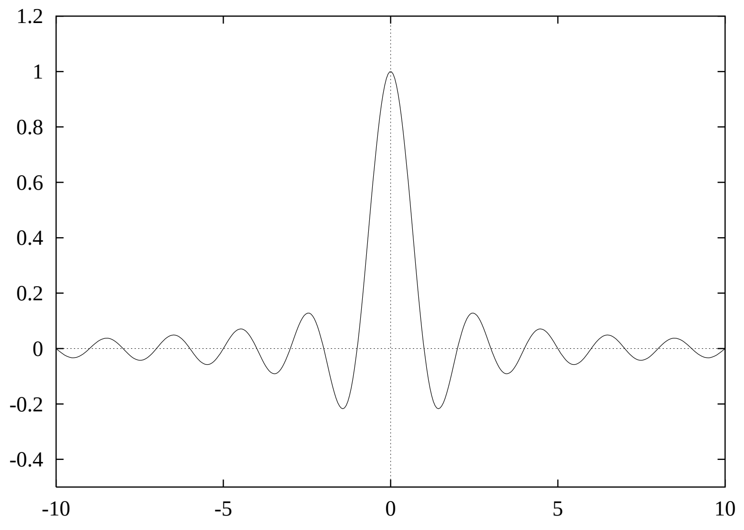


*Figure 3: The sinc function*

*filter* or *postfilter*. The purpose of this filter is to remove all the replicated spectra except the one around $f = 0$, leaving just the original signal; in the time domain it can be seen as smoothing the series of impulses into a continuous function.

Like the prefilter, the postfilter should ideally have a $\mathrm{sinc}(x)$ impulse response. Again, practical filters are often crude approximations. A typical example is displaying a digital image on a CRT display, where there usually is no explicit reconstruction filtering but the image is in effect convolved with the display's spot shape.

Another example of a crude reconstruction filter is the use of pixel replication when magnifying digital images. This is equivalent to using a box-shaped reconstruction filter kernel. The familiar "big blocky pixels" effect that results from this can be avoided by using a reconstruction filter that is a better approximation of the ideal sinc filter.

## 3.2. Sampling and filtering in two dimensions

When a two-dimensional signal, such as an image, is sampled on a uniform rectangular grid of frequency $f_s$, it is possible to reproduce, without aliasing, the spatial frequencies $(u, v)$ that lie within the square where $-f_s/2 < u < f_s/2$ and $-f_s/2 < v < f_s/2$.

This implies that a system using a square sampling geometry in fact has greater resolution by a factor of $\sqrt{2}$ in the diagonal directions than in the horizontal and vertical directions. Because practical two-dimensional signals usually are either isotropic or even dominated by horizontal and vertical frequency components, the additional resolution outside the circle with radius $f_s/2$ around the origin is often of no practical use and a corresponding part of the bandwidth of the sampled signal is wasted. Sometimes a hexagonal sampling geometry is used to reduce this waste [6].

Bandlimiting a two-dimensional signal prior to sampling can be done using a filter whose frequency response is either square or circular, although many practical filters are so crude that the distinction is of no consequence. The ideal square two-dimensional lowpass filter has a *separable* impulse response, i.e. it is the product of a horizontal and a vertical impulse response:

$$f(x, y) = \mathrm{sinc}(x)\,\mathrm{sinc}(y)$$

*Figure 4: Square 2-D ideal lowpass filter kernel and frequency response*

On the other hand, the ideal circularly symmetric lowpass filter has a cylinder-shaped frequency response and a circularly symmetric impulse response. Surprisingly, many standard computer graphics and image processing textbooks [7, 11] completely neglect this filter even though they devote much attention to its square counterpart. It is, however, mentioned briefly in Pratt [23] and in Dudgeon and Mersereau [6].

The impulse response of this filter has the somewhat non-obvious form

$$f(r) = 2\pi \frac{J_1(2\pi r)}{2\pi r}, \quad r = \sqrt{x^2 + y^2},$$

where $J_1$ is a Bessel function of the first kind. This expression is derived in Appendix A.

Note that this filter kernel is *not* simply sinc($r$) (sinc($r$) is actually a fairly strong highpass filter), and that this kernel falls off at a rate of $r^{-1.5}$ asymptotically in all directions, while the rectangular sinc($x$)sinc($y$) kernel falls off only as $r^{-1}$ along the horizontal and vertical axis.

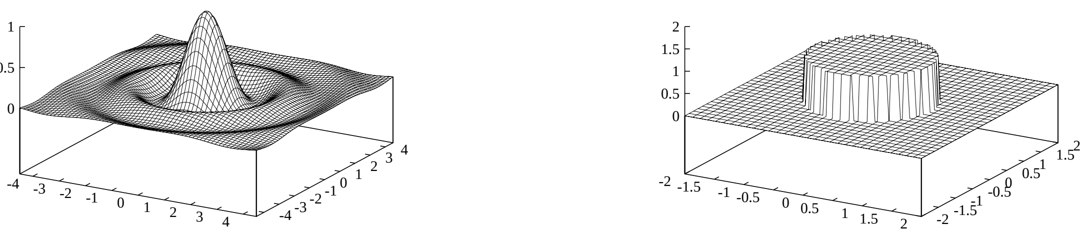


*Figure 5: Circular 2-D ideal lowpass filter kernel and frequency response*

## 3.3. *Warping sampled images*

In Section 2.1, spatial transformations were defined as mappings from one continuous coordinate space to another. A digital image warping system, however, needs to work with discrete coordinates. In theory, warping of a digital image could be achieved by performing the following separate steps:

1. Reconstruct a continuous image from the sampled source image.
2. Warp this continuous image.
3. Resample the warped continuous image (with appropriate prefiltering).

(Note that the prefilter comes *after* the reconstruction filter; the *pre-* refers to "before sampling", not "before warping").

In practice, a direct implementation of the above is not possible because there is no way to represent the continuous image exactly in a digital system. Schemes may be devised where the intermediate continuous image is approximated by a sampled image at a higher sampling rate than the final image (*supersampling*), possibly adapting the sampling rate depending on the local behavior of the mapping as in the work by Gagnet et al. [10].

More commonly, the calculations are rearranged such that the reconstruction filter and the prefilter can be combined into a single *unified resampling filter*, saving computational cost and making the intermediate image unnecessary.

The remainder of this section describes the rearrangement of the computations and derives an expression for the unified filter. It is mainly based on Heckbert's treatment of the subject [17], but the final observation about viewing the unified filter as the intersection of two shapes in the frequency plane is my own.

Assume we have an $n$-dimensional discrete input signal $f(u)$, $u \in Z^n$. This signal is mapped from the source coordinate space $u$ to the destination coordinate space $x$ by the forward mapping $x = m(u)$, corresponding to the inverse mapping $u = m^{-1}(x)$. The reconstruction filter kernel is $r(u)$, and the prefilter kernel is $h(x)$. To compute the discrete, warped output $g(x)$ using the step-by-step process outlined above, we proceed as follows:

| | |
|---|---|
| *Discrete input signal* | $f(u)$, $u \in Z^n$ |
| *Continuous input signal* | $f_c(u) = (f * r)(u) = \sum_{k \in Z^n} f(k)\, r(u-k)$ |
| *Warped continuous signal* | $g_c(x) = f_c(m^{-1}(x))$ |
| *Prefiltered signal* | $g'_c(x) = (g_c * h)(x) = \int_{R^n} g_c(t)\, h(x-t)\, dt$ |
| *Discrete output signal* | $g(x) = g'_c(x)\, i(x)$ |

Expanding this into a formula for $g(x)$, we get

$$\begin{aligned} g(x) &= g'_c(x) \\ &= \int_{R^n} f_c(m^{-1}(t))\, h(x-t)\, dt \\ &= \int_{R^n} h(x-t) \sum_{k \in Z^n} f(k)\, r(m^{-1}(t) - k)\, dt \\ &= \sum_{k \in Z^n} f(k)\, s(x,k) \end{aligned}$$

where

$$s(x,k) = \int_{R^n} h(x-t)\, r(m^{-1}(t) - k)\, dt.$$

Here $s(x, k)$ is the kernel of the unified resampling filter specifying the weight of the input sample at location $k$ for the output sample at location $x$. For non-affine mappings, this unified filter is *space variant*, meaning that not only its position but also its shape varies with $x$.

By the substitution $t = m(u)$ we arrive at an alternative form, integrating in the source image space rather than the destination image space:

$$s(x, k) = \int_{R^n} h(x - m(u))\, r(u - k) \left| \frac{\partial m}{\partial u} \right| du$$

where $|\partial m / \partial u|$ is the determinant of the Jacobian matrix of the mapping function. For a one-dimensional mapping,

$$\left| \frac{\partial m}{\partial u} \right| = \frac{dm}{du},$$

and for a two-dimensional mapping,

$$\left| \frac{\partial m}{\partial u} \right| = \begin{vmatrix} x_u & x_v \\ y_u & y_v \end{vmatrix},$$

where $x_u = \partial x / \partial u$, etc.

For any given output sample $x_0$, this resampling filter can be expressed as a convolution of the (mirrored) reconstruction filter with a warped and weighted prefilter

$$\begin{aligned} s_{x_0}(k) &= \int_{R^n} h'_{x_0}(u)\, r'(k - u)\, du \\ &= (h'_{x_0} * r')(k) \end{aligned}$$

where

$$\begin{aligned} h'_{x_0}(u) &= \left| \frac{\partial m}{\partial u} \right| h(x_0 - m(u)) \\ r'(u) &= r(-u) \end{aligned}$$

To simplify the expression for $h'_{x_0}(u)$, we replace the mapping $m$ with its local affine approximation in the neighborhood of $u_0 = m^{-1}(x_0)$:

$$m_{u_0}(u) = x_0 + (u - u_0) J_{u_0}$$

where

$$J_{u_0} = \frac{\partial m}{\partial u}(u_0)$$

is the Jacobian matrix evaluated at $u_0$. This leads to the approximation

$$\begin{aligned} h'_{x_0}(u) \approx& |J_{u_0}|\, h(x_0 - m_{u_0}(u)) \\ =& |J_{u_0}|\, h(-(u - u_0) J_{u_0}) \end{aligned}$$

which is an amplitude scaled, affine warp of the prefilter kernel. The translational portion of the warp can be expressed as a convolution by a shifted impulse

$$h'_{x_0}(u) \approx |J_{u_0}|\, h(-u\, J_{u_0}) \, * \, \delta(u - u_0)$$

Thus, the resampling filter can be approximated by a translated resampling filter that is the source image space convolution of the reconstruction filter kernel with a linear warp of the prefilter kernel:

$$\begin{aligned} s_{x_0}(u) &\approx \delta(u - u_0) * |J_{u_0}|\, h(-u\, J_{u_0}) * r(-u) \\ &= \delta(u - u_0) * s'_{x_0}(-u) \end{aligned}$$

where

$$s'_{x_0}(u) = |J_{u_0}|\, h(u\, J_{u_0}) * r(u)$$

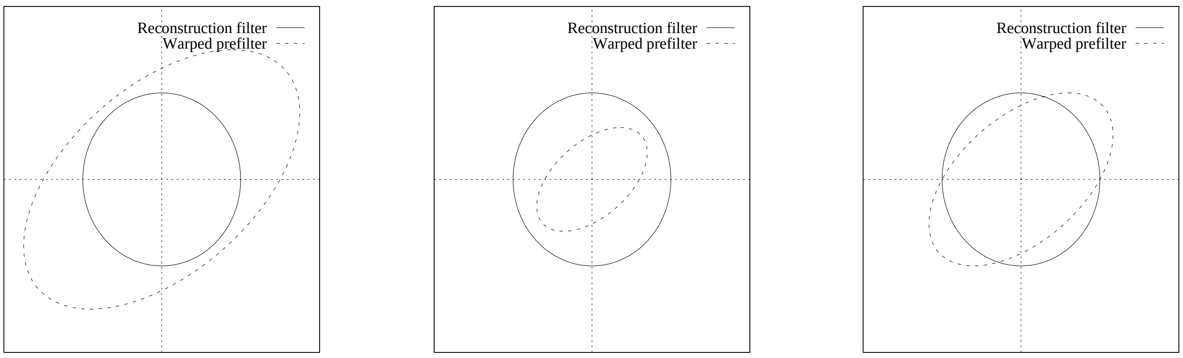
Denoting the Fourier transform of $r(u)$ by $R(w)$ and the Fourier transform of $h(u)$ by $H(w)$, the Fourier transform of $h(u\, J_{u_0})$ is $\frac{1}{|J_{u_0}|}\, H(w\, (J_{u_0}^T)^{-1})$, and the Fourier transform of $s'_{x_0}(u)$ is

$$S'_{x_0}(w) = H(w\, (J_{u_0}^T)^{-1})\, R(w)$$

This means that the frequency response of the unified resampling filter can be approximated simply by *multiplying* the frequency response of the reconstruction filter with the linearly transformed frequency response of the prefilter. Because the frequency response of an ideal lowpass filter is unity inside and zero outside a given shape (typically a square or a circle for two-dimensional filters) the product of the two frequency responses simply becomes the *intersection* of the reconstruction filter shape with the warped prefilter shape.

The case where both the reconstruction filter and the prefilter are ideal, two-dimensional circularly symmetric filters is of particular interest. A linear transformation of a circle is an ellipse, so the frequency response of the unified resampling filter in this case is the intersection between a circle and an ellipse (see Figure 6).

If the mapping magnifies the image in all directions, the ellipse falls entirely within the circle and the resampling filter is equal to the reconstruction filter. If the mapping minifies in all directions, the ellipse falls entirely outside the circle and the resampling filter is equal to the warped prefilter. However, for mappings that magnify in some directions and minify in others, the ellipse and circle boundaries will actually intersect and the resampling filter has a frequency response whose shape is neither a circle nor an ellipse (Figure 6*c*). Sections 3.6 and 3.7 will discuss approximations to this somewhat awkward frequency response shape for use in practical filters.



*a. Magnification: reconstruction filter dominates*

*b. Minification: prefilter dominates*

*c. Filter boundaries intersect*

*Figure 6: Different cases of unified resampling filters derived from circularly symmetric filters, in the frequency domain*

## 3.4. Practical digital image filtering

This section presents some important methods used in practical image filtering, and some of the problems that appear due to the non-ideal behavior of those methods.

### 3.4.1. Simple interpolation filters

When warping an image according to a mapping where the warped prefilter frequency response completely covers the reconstruction filter frequency response, as in uniform magnification, the prefilter has no effect (the input to the prefilter contains only frequencies within its passband), and the resampling filter is therefore equal to the reconstruction filter. In this case no information is lost in the warp. This case is commonly referred to as image *interpolation*, because each result pixel can be calculated directly by estimating the value of the warped continuous function at a given point given its samples, i.e., interpolating it from the samples. For a detailed discussion about the intimate relationship between interpolation and digital filtering, see the classic paper by Schafer and Rabiner [25].

A crude way to estimate the value of a continuous signal at a given point is to simply use the value of the nearest sample. This is called ***nearest-neighbor interpolation*** and corresponds to using a reconstruction filter whose impulse response has a box shape and a width equal to the sampling interval. This filter is widely used because of its simplicity and speed, despite its poor quality.

A somewhat more sophisticated method is that of linear interpolation, corresponding to convolution with a triangular, or "tent", filter kernel with a width of two samples. The two-dimensional generalization of linear interpolation, ***bilinear interpolation***, uses the four nearest samples and corresponds to a filter kernel which is the product of a horizontal and a vertical triangle (see Figure 7).

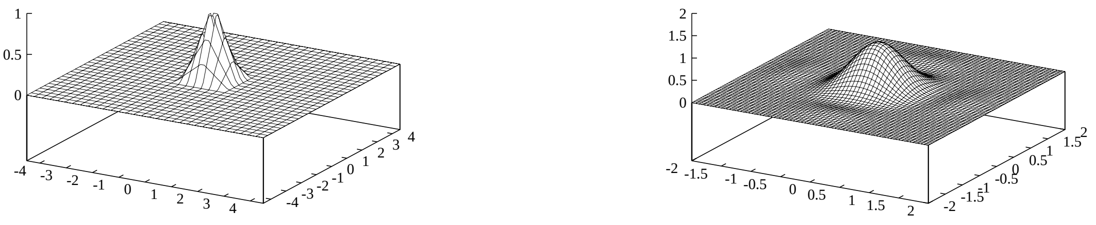


*Figure 7: Bilinear interpolation filter kernel and frequency response*

Separable filter kernels defined by piecewise cubic polynomials have also been used for image interpolation. These are discussed in detail by Mitchell and Netravali [18].

Convolution with a Gaussian filter kernel is also commonly used. The Gaussian has several properties that make it interesting as a filter kernel: it is positive everywhere, it is its own Fourier transform, it exhibits exponential falloff in both the spatial and the frequency domain, and the two-dimensional Gaussian is unique in having the property of being both circularly symmetric *and* separable.

### 3.4.2. Simple decimation filters

When warping an image according to a mapping that minifies the image (at least in some direction), the warped continuous image will contain spatial frequencies that cannot be reproduced in the discrete result image. These frequencies need to be removed by the prefilter to avoid aliasing. In this case information is lost as the warping in effect reduces the sampling rate of the image. This process of sampling rate reduction is known as *subsampling* or *decimation*.

While interpolation is performed using lowpass filters with a fixed cut-off frequency of $f_s/2$, decimation requires a lowpass filter whose cutoff frequency is inversely proportional to the decimation factor, and a filter kernel whose size is directly proportional to the decimation factor.

Performing a naïve convolution with such a large filter kernel can lead to a high computational cost per output pixel (although the cost per input pixel remains constant, because the number of output pixels is reduced in proportion to the decimation factor).

Reducing this cost is of particular interest in texture mapping applications, where a large texture image is often warped onto many different small areas in the final rendered image (or images). This can be achieved using a number of specialized methods based on precalculating an alternative representation of the input image, making the convolution of large areas more convenient. These methods include filtering by repeated integration [15], *summed-area tables* [5], *resolution pyramids*, and *MIP maps* [8, 27].

### 3.4.3. Windowed sinc filters

Filters of a higher quality than the simple filters above are often based on directly convolving the input signal with a filter kernel based on the ideal sinc filter kernel. These filters are useful for both interpolation and decimation tasks.

Unfortunately, the sinc is not suited for direct implementation as a digital filter because it is infinitely wide (i.e., it has *infinite support*). To derive a practical digital filter kernel from the sinc kernel, it must first be given a finite region of support. This is done through *windowing*: the ideal kernel is multiplied by another function having finite support (i.e., one which is zero outside some finite interval around the origin). A function used for this purpose is called a *window function*.

Windowing changes the frequency response of the kernel: the frequency response is in effect convolved with the Fourier transform of the window function. Therefore, a good window function is one whose Fourier transform is concentrated in a narrow lobe around the origin, approximating an impulse function. Unfortunately, this is contradictory to the requirement that the window function itself should have a small region of support in order to minimize the amount of computation needed for the convolution. The Fourier transform of most practical window functions consists of a central lobe and some side lobes; different window functions offer different tradeoff between the width of the central lobe and the height of the side lobes. A wide central lobe causes the sharp edges in the ideal frequency response to be smoothed, while high side lobes cause ripples in the frequency response of the windowed filter.

The simplest window function is the *box*, or *rectangular window* of width $2w$:

$$\mathrm{Rect}_w(x) = \begin{cases} 1, & |x| \le w; \\ 0, & |x| > w. \end{cases}$$

Windowing a sinc with a rectangular window is equivalent to simply truncating the sinc by forcing it to zero outside some interval. Other commonly used windows include the triangular (tent) window, the Gaussian, and the *Hanning window*

$$\mathrm{Hanning}_w(x) = \begin{cases} \frac{1}{2}(\cos(\pi x/w) + 1), & |x| \le w; \\ 0, & |x| > w. \end{cases}.$$

Another popular window is the *Kaiser window*, which can be parametrized to allow different tradeoffs between central lobe width and side lobe level. The *N-lobe Lanczos window*

$$\mathrm{Lanczos}N(x) = \begin{cases} \mathrm{sinc}(x/N), & |x| \le N; \\ 0, & |x| > N \end{cases}$$

is itself a scaled, truncated (box-windowed) sinc function and is popular in computer graphics applications [3, 26].

Generally, filter kernels based on windowed sincs or other direct approximations to the sinc (such as most piecewise cubic filter kernels) share the sinc's property of having negative lobes. This causes some complications in implementing the filters: the range of the filter's output may exceed that of its input and the output can also become negative even though the input is always positive. Displaying the output from such a filter correctly may therefore require the production of "negative light", which is physically impossible, so the output of the filter must be either scaled or clipped; either choice will degrade image quality.

For a detailed treatment of windowing and a comprehensive list of window functions, see Harris [14].

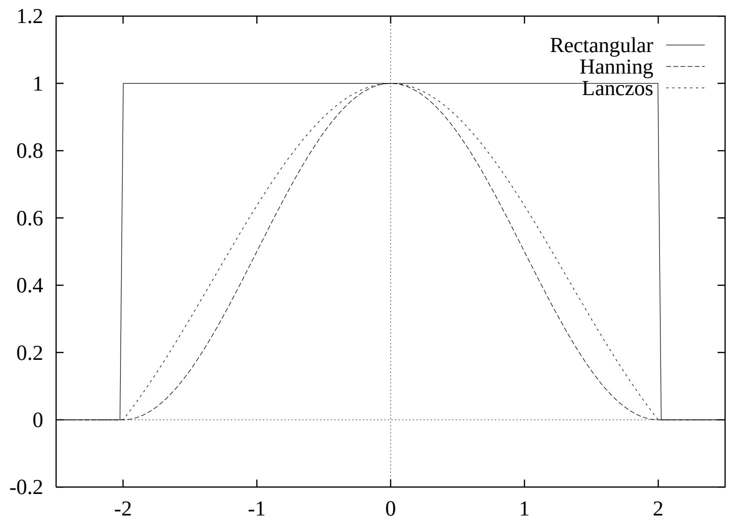


*Figure 8: Some common window functions*

### 3.4.4. A circular two-dimensional windowed filter

A square two-dimensional filter is easily derived from any one-dimensional windowed sinc filter simply as the product of a horizontal and a vertical filter kernel. It is also possible to design a circularly symmetric two-dimensional filter by windowing the ideal circularly symmetric filter with a circularly symmetric, two-dimensional window function. Such window functions are sometimes constructed by rotating a one-dimensional window around the origin [6].

For the image warping system described in Chapter 4, I have chosen a two-dimensional, circularly symmetric filter using a window that can be considered a two-dimensional equivalent of the two-lobe Lanczos window. While the Lanczos window consists of the central lobe of the ideal 1-D lowpass filter kernel, this window consists of the central lobe of the ideal circular 2-D lowpass filter kernel.

The filter, which I will refer to as the Lanczos2D filter, consists of the central lobe and first side lobe of the ideal circularly symmetric lowpass filter, windowed by the above function.

$$\text{Lanczos2D}(r) = \begin{cases} I(r)\, I(r\,\frac{r_1}{r_2}), & r \le r_2; \\ 0, & r > r_2. \end{cases}$$

where $I(r)$ is the ideal filter

$$I(r) = 2\pi \frac{J_1(2\pi r)}{2\pi r}$$

and $r_1$ and $r_2$ are the two first zero-crossings of the function $I(r)$

$$\begin{aligned} r_1 &\approx 1.220 \\ r_2 &\approx 2.233. \end{aligned}$$

For a one-dimensional Lanczos-windowed filter, the corresponding values are exactly $r_1 = 1$ and $r_2 = 2$. A cross-section of the Lanczos2D filter kernel, together with the ideal filter and the window function from which it was derived, is shown in Figure 9.

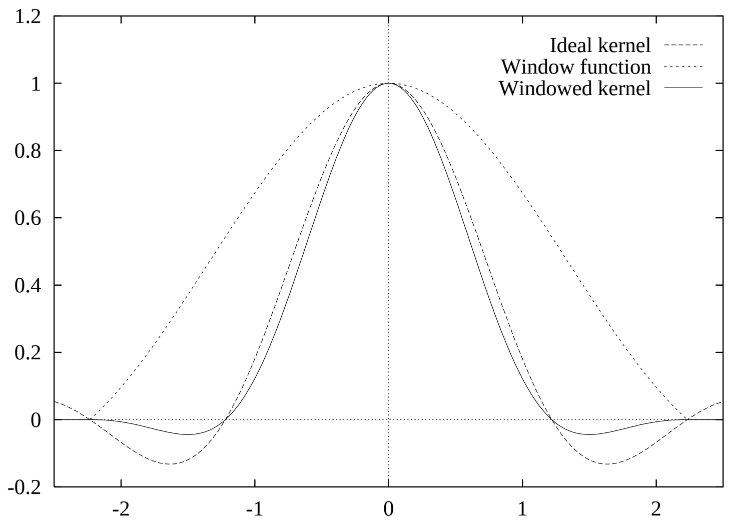


*Figure 9: Cross-section of Lanczos2D filter kernel*

A two-dimensional view of the Lanczos2D filter kernel and a frequency response calculated using the fast Fourier transform appear in Figure 10.

### 3.4.5. *Sampling-frequency ripple*

Because practical reconstruction filters are not ideal, some of the replicated spectra of the sampled signal will always remain after reconstruction. This is called *postaliasing*.

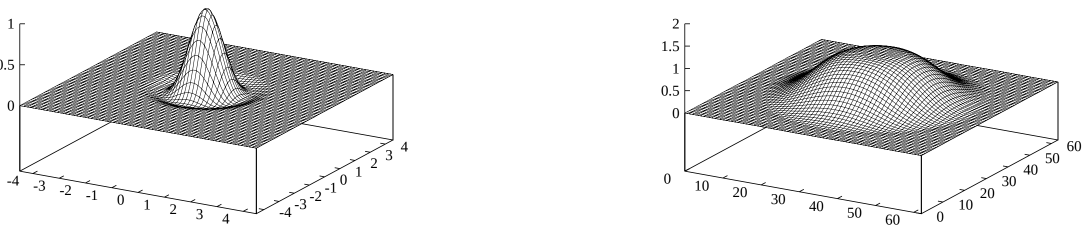


*Figure 10: Lanczos2D filter kernel and frequency response*

In particular, the frequency component $f = 0$, also known as the *DC component*, will alias to integer multiples of the sampling frequency. If the reconstruction filter fails to attenuate these components sufficiently, the sampling grid will become visible in the reconstructed image in the form of periodic intensity fluctuations. Mitchell and Netravali [18] refer to this defect as *sampling-frequency ripple*.

Sampling-frequency ripple can be avoided completely in the case of separable piecewise cubic reconstruction filters by choosing the parameters of the cubic polynomials appropriately [18, 20]. Bilinear interpolation is also inherently ripple-free. Other kinds of filter kernels, including circularly symmetric ones, can be designed with the specific goal of minimizing the amount of sampling-frequency ripple [21].

Another way to avoid this problem is to measure the actual filter output corresponding to a DC input, and compensate for any fluctuations in the output. In practice this means normalizing the result of the filtering by dividing each output pixel with the sum of all kernel samples contributing to it. This technique is used in the EWA filter described in Section 3.5. In an inverse mapping system like the EWA, the additional computational cost is $n$ adds and one divide per output pixel for a filter kernel covering $n$ pixels. If the technique is used in a forward mapping system, one also needs an additional accumulator array for the kernel sample values.

### 3.4.6. Filter kernel sampling and quantization

The purpose of reconstructing an image using a reconstruction filter is often to resample it on a different sampling grid. If the reconstruction filter is not ideal, its output will contain unwanted high-frequency components, and those will cause aliasing when the continuous image is resampled. For simplicity, we will assume throughout this section that the new sampling rate is no lower than the original sampling rate, so that no prefiltering is needed. Any aliasing that occurs in resampling is will thus be the result of imperfect reconstruction.[†]

As noted by Parker, Kenyon, and Troxel [20], the process of convolving the image with a continuous reconstruction filter kernel and then sampling the continuous result will in practice be implemented as the equivalent process of doing a discrete convolution of the image with a sampled reconstruction filter kernel.

Because of this equivalence, the abovementioned form of aliasing can also be thought of as the result of sampling the continuous filter kernel. It turns out that sampling a continuous filter kernel can change its frequency response significantly. This effect is especially noticeable when resampling an image at a sampling frequency that is close or equal to the original sampling frequency, such as when translating an image by a subpixel amount.

The change in frequency response associated with sampling a filter kernel depends on the alignment of the sample points with respect to the center of the kernel (the *sampling phase*). This is because sampling the kernel causes unwanted high-frequency components in the stopband to alias onto the passband, and the sampling phase determines whether the aliased stopband will be added to or subtracted from the passband response.

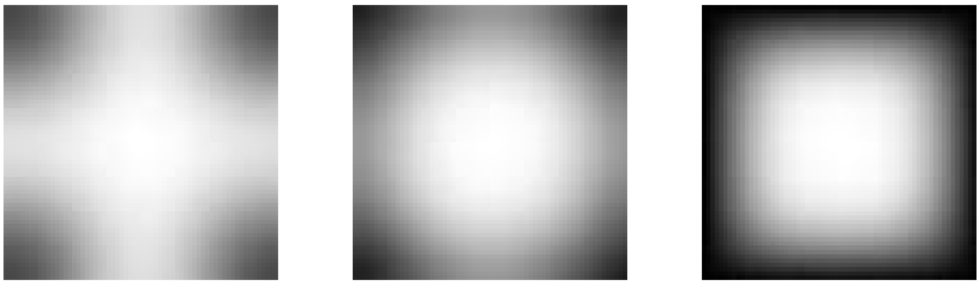
Therefore, in designing or comparing the performance of resampling filters it is often necessary to examine not only the Fourier transform of the continuous filter kernel but also that of the sampled kernel, at several different phases.

Consider, for example, the use of linear interpolation in resampling a (one-dimensional) sequence of samples at the original sampling rate but at positions that are offset from the original by some fraction of a sample. If the new samples coincide exactly with the original ones, the interpolated result will be identical to the original signal, but if the signal is resampled halfway between the original samples, each linearly interpolated output sample will be the average of the two neighboring samples of the original signal, leading to a considerable smoothing of the signal, more than would be expected from analyzing just the frequency response of the continuous kernel.

The effect of sampling a small two-dimensional filter kernel is illustrated in Figure 11. Here, the 2.233-pixels-wide Lanczos2D kernel of Section 3.4.4

[†] When a prefilter *is* needed, it can be combined with the reconstruction filter as in Section 3.3; what is said in this section also applies to the resulting unified filter.

has been sampled at three different phases: the sampling grid was offset by $(0,0)$, $(0.25,0.25)$, and $(0.5,0.5)$ pixels from the center of the continuous filter kernel. The resulting frequency responses were calculated using the fast Fourier transform and are shown as gray-scale images where the lightness of the image is proportional to the magnitude of the Fourier transform. Of these, only Figure 11*b* looks similar to the continuous frequency response plotted in Figure 10; the other two are visibly distorted.



*a.* $(0,0)$ *b.* $(0.25,0.25)$ *c.* $(0.5,0.5)$

*Figure 11: Frequency response of a lowpass filter sampled at different phases*

When a continuous filter kernel is used in a practical digital filter, it is not only sampled but the sampled filter coefficients are also quantized to a finite precision. If this precision is small, which may be the case especially in hardware implementations, the coefficient quantization may change the frequency response of the filter noticeably.

This phenomenon is particularly noticeable in the case of large filter kernels, because they have large fringe areas where the amplitude is small and therefore prone to quantization errors. High-quality filter implementations may therefore need to store the filter coefficients at a higher precision than the image pixel values, e.g. as floating-point values.

The effect of filter coefficient quantization is illustrated in Figure 12, where the frequency response of a sampled lowpass filter kernel with a radius of 20 has been calculated before and after quantizing the sampled coefficient values to 8-bit fixed-point numbers.

### 3.4.7. A subjective comparison of filter kernels

In Figure 13, an image has been rotated and resampled using several different filters so that the quality of the filters can be compared. The degradation of image quality has been exaggerated by rotating the image in several smaller steps, resampling after each step. Each of the rotated

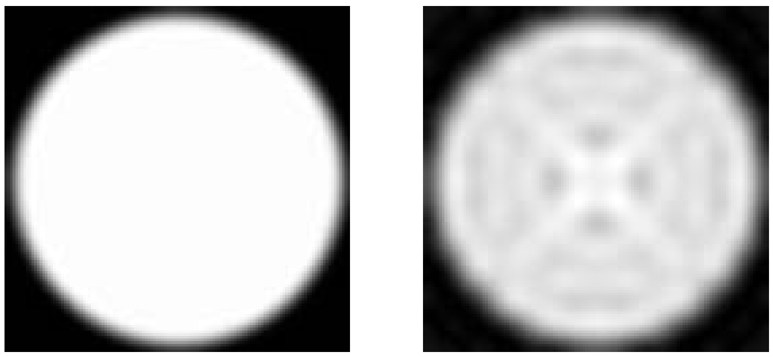


*Figure 12: Effect of filter coefficient quantization*

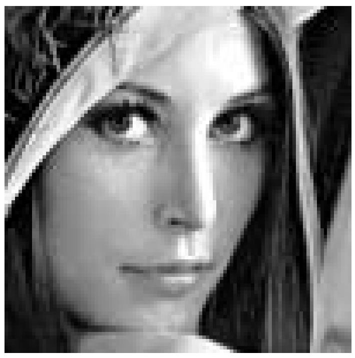
images is a 100 by 100 pixel detail from a larger image that was rotated a total of 15° clockwise in six steps of 2.5° each.

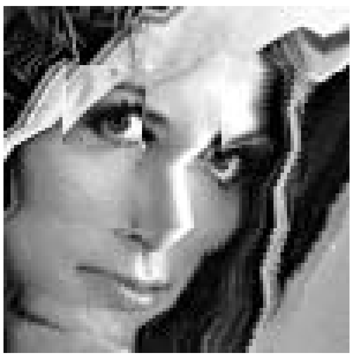
The original image is shown in Figure 13*a*. In Figure 13*b*, the image was resampled using nearest-neighbor sampling. The image is sharp, but the accumulated phase errors cause severe distortion of the image. In Figure 13*c*, bilinear interpolation was used. This avoids the most of the phase errors but causes excessive blurring. Convolution with a narrow truncated Gaussian filter kernel [26]

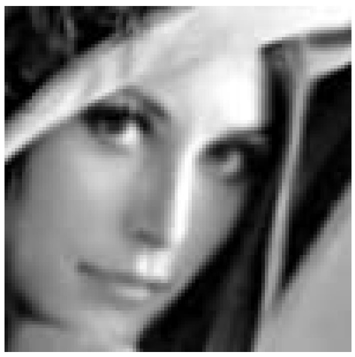
$$\text{Gaussian}_{\frac{1}{2}}(r) = \begin{cases} 2^{-4r^2}, & r \leq 1.5; \\ 0, & r > 1.5. \end{cases}$$

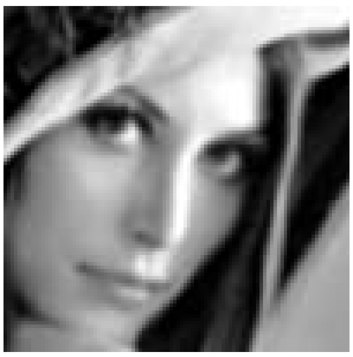
gives results similar to bilinear interpolation, as seen in Figure 13*d*. Wider Gaussians cause even more blurring.

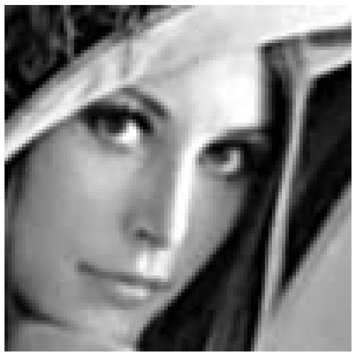
Figure 13*e* was calculated using the 2.233-pixels-wide Lanczos2D filter. It can be seen that this filter causes significantly less blurring than does bilinear interpolation. Finally, Figure 13*f* was calculated using a filter kernel with a radius of 8 pixels. This kernel was constructed by windowing the ideal circularly symmetric filter with the two-dimensional window obtained by rotating a Hanning window around the origin. This filter gives very good image quality, but is too slow to be of practical use on current hardware.


*a. Original image*


*b. Nearest-neighbor interpolation*


*c. Bilinear interpolation*


*d.* Gaussian$_{\frac{1}{2}}$ *filter*


*e.* Lanczos2D *filter*

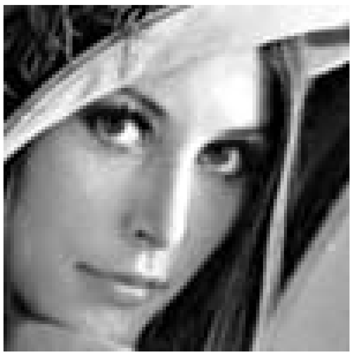

*f. Windowed filter with radius 8*

*Figure 13: Results of repeated rotation using different filter kernels*

## 3.5. *The Elliptical Weighted Average Filter (EWA)*

The Elliptical Weighted Average (EWA) filter of Greene and Heckbert [12] is an efficient method for implementing high-quality space-variant image filters with elliptically shaped filter kernels. It was originally developed for the purpose of warping perspective views into the fish-eye projection used in Omnimax wide-screen film.

In the EWA method, images are warped by scanning them in destination space order (i.e., an inverse mapping is used). The resampling filter has a circularly symmetric kernel with an arbitrary profile $f(r)$ in the destination space, corresponding to an approximately elliptical kernel in the source image space. Each output pixel is computed as a weighted average of all the source pixels falling inside the elliptical source-space filter kernel.

In order to efficiently find the extent of this ellipse and the value of the kernel function at each pixel within it, the ellipse is represented in an implicit form

$$Q(u,v) = Au^2 + Buv + Cv^2 = F$$

where $u, v$ are the source image coordinates and $A, B, C, F$ are the ellipse parameters. A pixel $(u,v)$ falls within the ellipse if $Q(u,v) < F$.

The parameters $A, B, C, F$ can be calculated directly from the Jacobian of the mapping:

$$\begin{aligned} A &= v_x^2 + v_y^2 \\ B &= -2(u_x v_x + u_y v_y) \\ C &= u_x^2 + u_y^2 \\ F &= (u_x v_y - u_y v_x)^2 \end{aligned}$$

where $u_x = \dfrac{\partial u}{\partial x}$, etc.

The biquadratic function $Q(u,v)$ is directly proportional to the square of the distance from the corresponding destination-space point $(x,y)$ to the point being evaluated. Therefore, the value of the warped filter kernel at $(u,v)$ can be expressed as $f(\sqrt{Q(u,v)})$.

One could store the filter kernel profile function $f(r)$ in a lookup table and evaluate $f(\sqrt{Q(u,v)})$ at every pixel inside the ellipse, but the square root calculation can be avoided by instead tabulating the function $f(\sqrt{Q})$ and indexing the table by $Q(u,v)$ directly.

The function $Q$ itself can be efficiently evaluated using the method of finite differences. The pixels within the bounding box of the source-space ellipse are visited in scanline order, and the $Q$ value at each pixel can be calculated incrementally at the cost of only two additions per pixel. Pixels falling inside the bounding box but outside the ellipse are efficiently eliminated by comparing $Q$ against $F$, and if the parameters $A$, $B$, $C$, and $F$ are scaled appropriately, the integer part of $Q$ can be used directly as an index into the lookup table giving the filter coefficient $f(Q)$. The coefficient is multiplied by the pixel value (or each component in the case of color images) and the result is accumulated. The coefficients themselves are also accumulated so that the final, filtered pixel values can be normalized by dividing them with the sum of the filter coefficients.

For a more detailed description of the EWA algorithm, including pseudocode, see either the original EWA paper [12] or Heckbert's master's thesis [17].

## 3.6. *A unified resampling filter based on the EWA*

In Section 3.3 it was shown that the frequency response of a 2-D unified resampling filter based on circularly symmetric prefilter and reconstruction filter kernels is the intersection of a circle and an ellipse. In order to use the EWA as a unified resampling filter, this shape must be approximated by a single ellipse. One way to do this is as follows: we represent the elliptical frequency response of the warped prefilter by its major axis length $A$, its minor axis length $B$, and a rotation angle $\alpha$, and we represent the reconstruction filter by its radius $R$. Then a reasonable approximation of the intersection is an ellipse with the same rotation angle $\alpha$, a minor axis of length $A' = \min(A, R)$, and a major axis of length $B' = \min(B, R)$.

In practice it is more convenient to perform this approximation in the spatial domain. The above ideal elliptical frequency responses will correspond in the spatial domain to elliptical filter kernels of infinite support (i.e., linear transformations of the ideal circular kernel). The axes of the elliptical filter kernels are oriented identically to those of the corresponding frequency-domain ellipse, but the dimension along each axis (the linear scale factor relative to the circular kernel) is inversely proportional to the length of the corresponding axis of the frequency domain ellipse (this follows from the rotational invariance and linear scaling properties of the two-dimensional Fourier transform).

Therefore, if the warped prefilter kernel has axes $a \propto 1/A$ and $b \propto 1/B$ and the reconstruction filter kernel has radius $r \propto 1/R$, the kernel of the approximated resampling filter will have the same orientation as the warped prefilter kernel and axes of length

$$a' = \frac{1}{\min(\frac{1}{a}, \frac{1}{r})} = \max(a, r)$$

$$b' = \frac{1}{\min(\frac{1}{b}, \frac{1}{r})} = \max(b, r)$$

This means that an approximate unified resampling filter can be obtained in the spatial domain simply by "fattening" the elliptical warped prefilter kernel along both axes to make it at least as wide as the reconstruction filter kernel. This is illustrated in Figure 14.

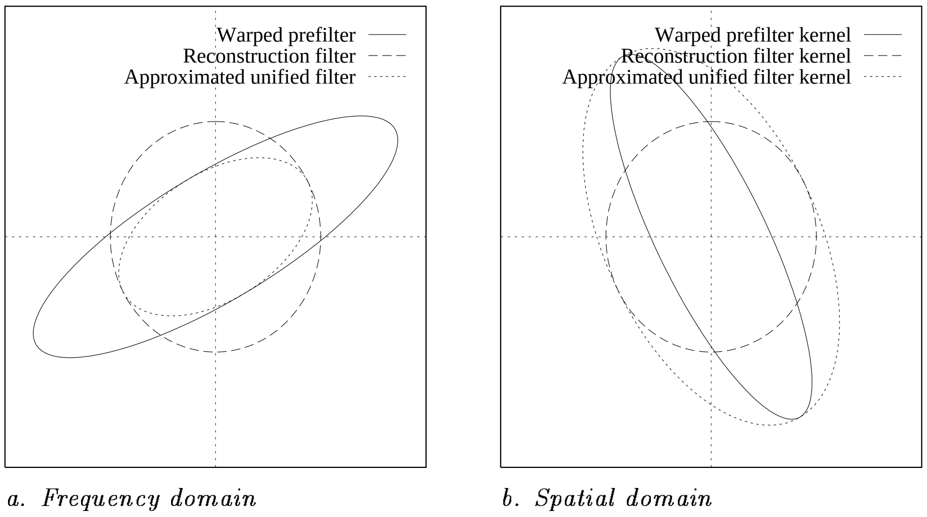


*Figure 14: Approximation of unified filter by elliptical filter*

A practical method to perform this "fattening" of an implicit ellipse of the form used in the EWA is by rotating the ellipse into alignment with the coordinate axes, setting any axis length less than $r$ to $r$, and rotating the ellipse back into its original orientation. The rotations can be done without actually performing any trigonometric operations, but the procedure does still require a substantial amount of computation, including a division and a square root.

Given an ellipse $ax^2 + bxy + cy^2 = f$, the aligned ellipse is $a'x^2 + c'y^2 = f$, where

$$a', c' = \frac{a + c \pm \sqrt{(a-c)^2 + b^2}}{2}.$$

The length of each axis is then forced to be at least $r$ by setting

$$a'' = \min(a', f/r^2)$$
$$c'' = \min(c', f/r^2)$$

Finally, the ellipse is rotated back into its original orientation:

$$
\begin{aligned}
a''' &= \frac{1}{2}(a'' + c'') + \frac{1}{2}(a'' - c'')\,(a - c)\,t \\
b''' &= (a'' - c'')\,b\,t \\
c''' &= \frac{1}{2}(a'' + c'') - \frac{1}{2}(a'' - c'')\,(a - c)\,t
\end{aligned}
$$

where

$$t = \frac{1}{\sqrt{(a - c)^2 + b^2}}$$

The degenerate case where the input ellipse is actually a circle must be handled as a special case to avoid dividing by zero. The algorithm can be optimized somewhat by avoiding the final rotation when the output is equal to one or the other of the input shapes.

## 3.7. *Heckbert's unified EWA*

In his master's thesis [17], Heckbert presents a different kind of EWA-based unified resampling filter. He chooses an elliptical Gaussian shape for both the prefilter and the reconstruction filter kernels. Because the class of elliptical Gaussians is closed under affine transformation and convolution, the local affine approximation of the warped prefilter is also an elliptical Gaussian, and the resampling filter can now be calculated exactly as the convolution of two elliptical Gaussians, which is itself an elliptical Gaussian. This calculation is trivial because two elliptical Gaussians can be convolved simply by summing their variances. The result is a filter that exhibits a smooth transition between magnification and minification, but which is considerably wider in the transition region than one derived using the method proposed in Section 3.6.

The difference between the two approaches can be summarized by noting that in Heckbert's method one calculates an exact convolution of two approximated filter kernels, while in the author's method one calculates an approximation of the convolution of two exact filter kernels. Heckbert's method has the advantages of simplicity and speed: calculating the parameter of the unified filter requires only two additions more than those for a minification-only filter. Calculating the ellipse parameters using the method of Section 3.6 is more complex and carries a significant computational cost. On the other hand, the method of Section 3.6 allows the use of higher-quality, non-Gaussian filter kernels, and the higher cost of calculating the ellipse parameters may be offset by savings in the actual filtering because the method results in a smaller filter kernel.

Figure 15 illustrates the difference in width of filter kernels derived using the two methods. The width of the reconstruction filter has been normalized to 1, and the unified filter width is plotted as a function of the prefilter width. Note that the difference between the two methods is at its greatest when the filters are of approximately the same width, i.e. when the magnification factor is approximately 1. This is the typical case in a local warping system such as that of Chapter 4.

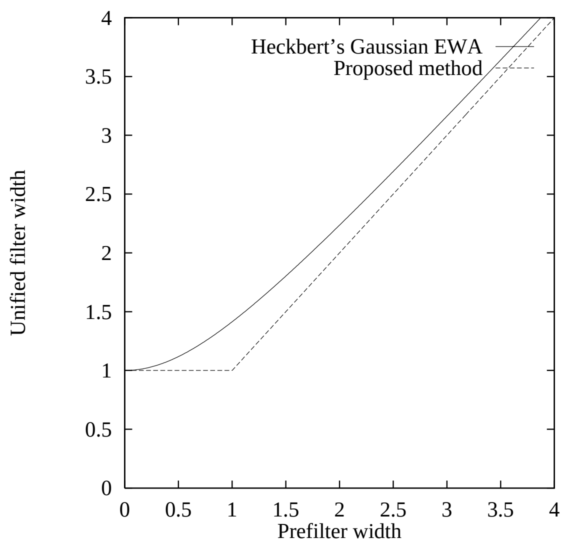


*Figure 15: Comparison of filter width in unified EWA methods*

Time did not permit a full comparison of the relative speed and subjective image quality of EWA filters constructed using the two methods. However, the results of the comparison of filter kernels for a space-invariant filtering task in Section 3.4.7 certainly suggest that the use of a non-Gaussian filter kernel can improve image quality.

# Chapter 4. Uwarp, an interactive image warping system

## 4.1. Existing methods for local warping

The image warping transformations described in Section 2.2 are global in nature: they are described by a set of coefficients such that a change in any one coefficient affects every pixel in the image.

In applications such as creative graphic design, there is frequently a need to perform local transformations on images: rather than warping the entire image, details of varying sizes may need to be moved, scaled, rotated, bent, or warped in other ways.

Traditionally, users of graphic design software have dealt with this need by using masking operations to separate the object to be warped from its environment, performing a global warp on the masked image, and finally pasting the warped image back into the original image. More complicated warps are created by dividing the image into several parts that are warped individually and then pasted together using alpha blending [22].

*Mesh warping* systems simplify the creation of local warps by overlaying the image with a rectangular grid of control points that can be moved individually. The warped position of each pixel in the image is determined by interpolation from the positions of neighboring control points. One such system was developed at Industrial Light and Magic by Douglas Smythe and is described in Wolberg's book [28]. This system is based on a two-pass warping algorithm and Catmull-Rom spline interpolation of control grid coordinates.

Mesh warping systems give the user detailed control over the image, but they may be awkward in that the control point mesh has a rectangular geometry which does not necessarily correspond to the geometry of the objects the user wants to warp. The size of the mesh division is also critical: if too large, accurate positioning is not possible, and if too small, a large number of control points have to be moved to achieve even a simple warp.

A closer correspondence between control point geometry and image geometry is achieved in the *field morphing* system of Beier and Neely [2]. This system allows the user to position an arbitrary number of directed line segments over important image features such as the edges of objects, and then specify the corresponding line segment positions in the warped image. Each such pair of line segments defines an affine mapping between the two images. The combined mapping is determined as a weighted average of the individual mappings, such that the position of each pixel is influenced more strongly by nearby line segments than by those farther away.

In all the local warping systems mentioned above, the user has no explicit and convenient control over the size of the area influenced by moving a control point. In mesh warping the size of the area of influence is determined once and for all by the mesh size. In field morphing moving a single line segment always affects the whole image, but the exact effect depends on what other line segments are defined.

Also, those systems suffer from a lack of immediate feedback; the user must position a large number of control points before the actual warp can be performed and the results can be viewed. While this may be appropriate for precision work such as creating animated metamorphosis effects or correcting well-defined geometric distortions, applications involving a greater degree of spontaneous creativity could benefit from having the warped image displayed immediately after, or even during, each modification. Ideally, an artist should be able to apply geometric transformations to an image as quickly and easily as he applies paint.

An example of an existing image warping system where feedback is given in real time during the warping is the system for real-time interactive manipulation of three-dimensional texture-mapped surfaces by Oka et al. [19]. The system achieves an update rate of 30 frames per second through the use of specialized hardware.

## 4.2. The Uwarp system

The remainder of this chapter describes a new kind of image warper developed by the author in an attempt to achieve a high level of interactiveness in two-dimensional local warping using standard workstation hardware. This system, which will be referred to as Uwarp, allows the user to construct a complex warp interactively through successive refinement by combining a number of simple local warps. Each of these warps is defined by a single mouse stroke, during which the display is updated with the warped result in real time.

A prototype implementation of the system has been written in the C language and runs under Unix with the X window system.

## 4.3. The Uwarp user interface

In Uwarp, a complex warp is built by successively applying a number of simple, local warps, which are referred to as *primitive transformations*. Each primitive transformation affects a circular area, such that the central parts of the area are warped most strongly and the effect diminishes towards the edge of the circle. The user is given explicit control over both the radius of this circular "area of influence" and the strength of the warp by performing a two-part mouse stroke as illustrated in Figure 16.

Pressing a mouse button sets the center point of the warp. The radius of the area of influence is determined by the most distant point visited during the stroke; this means that the radius can only increase in size during the stroke, never decrease, and that the cursor is always either at the edge of or inside the area of influence. The strength of the warp is determined by the position of the cursor within the area of influence. The whole interaction thus consists of pressing the mouse button, moving the mouse outwards to set the radius, moving it back towards the center to set the strength, and finally releasing the button.

The example in Figure 16 shows a pair of local translation warps, where the parts of the image within the area of influence are translated relative to their original position. The area of influence is indicated by a circle. In the actual user interface, displaying this circle is optional.

The program provides a similar user interface for specifying local scaling and rotation, as illustrated in Figure 17.

The prototype Uwarp implementation has an OpenLook user interface with which the user can select the desired transformation mode by pressing buttons in a control panel above the image (see Figure 18).

## 4.4. *Uwarp's local mapping functions*

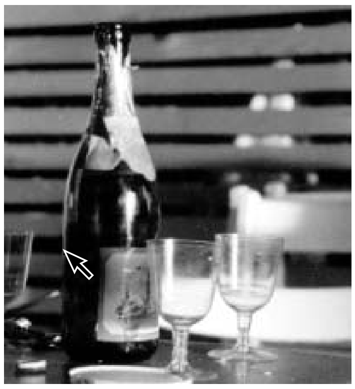
The following sections describe the parameters and mapping functions used in performing local translation, scaling, and rotation.

### 4.4.1. *Local translation warps*

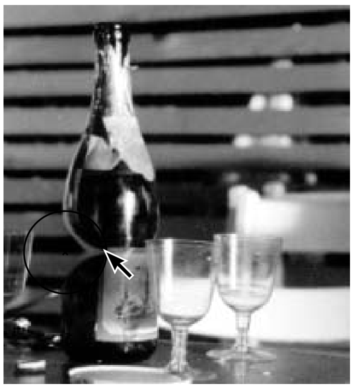
In local translation warping, the final mouse cursor position directly determines the warped position of the center point. The behavior of this user interface can be likened to that of an image printed on a rubber sheet glued onto a rigid surface: When a point on the rubber sheet is grabbed and dragged parallel to the surface, the tension overcomes the adhesive force of the glue and causes the rubber sheet to separate from the underlying surface over a circular area surrounding the grabbed point. This area gets larger as the point is moved farther away from its original location. When the separated area is of the desired size, the grabbed point can be positioned somewhere within it and reglued.

The source coordinate vector $\vec{u}$ corresponding to a destination pixel with coordinate vector $\vec{x}$ is calculated as follows:

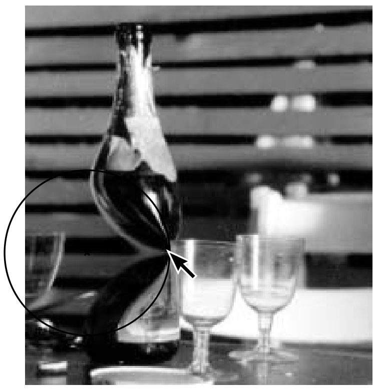
$$\vec{u} = \vec{x} - \left( \frac{{r_{\mathrm{max}}}^2 - |\vec{x} - \vec{c}|^2}{({r_{\mathrm{max}}}^2 - |\vec{x} - \vec{c}|^2) + |\vec{m} - \vec{c}|^2} \right)^2 (\vec{m} - \vec{c})$$



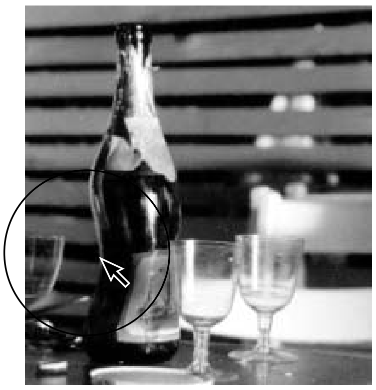
*a. Mouse button is pressed*



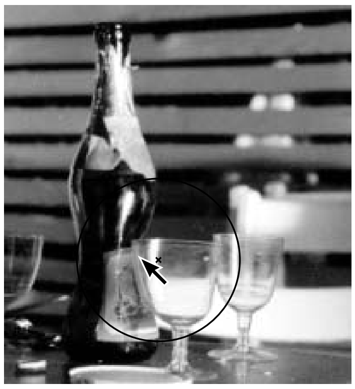
*b. Defining radius*



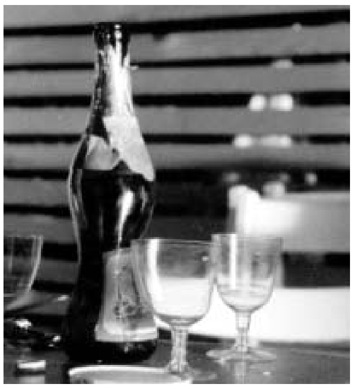
*c. Radius defined*



*d. Defining strength*

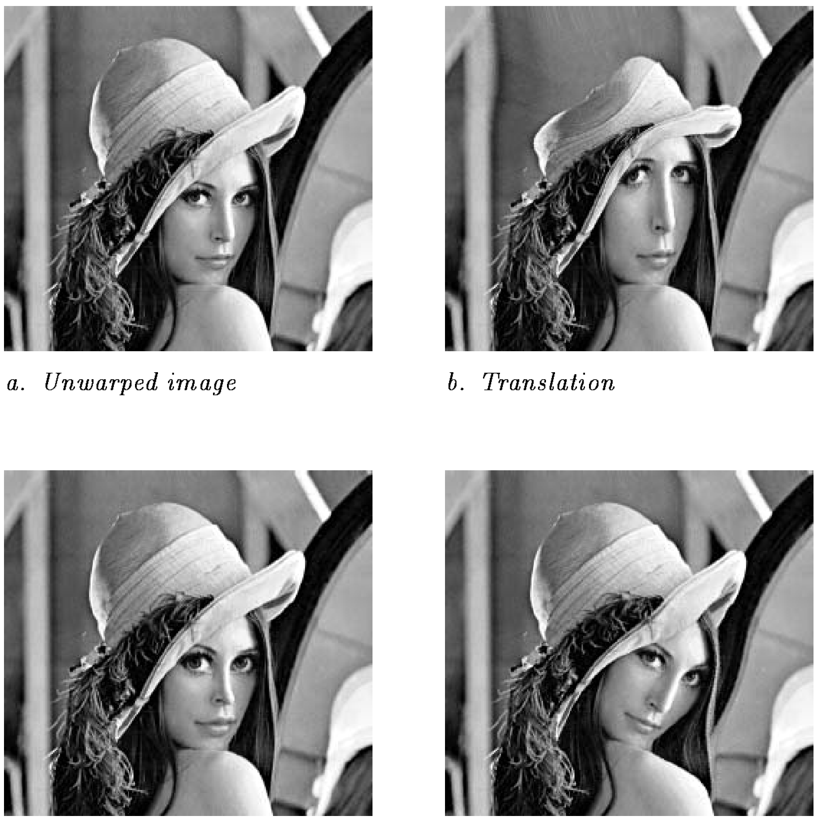


*e. Another warp*



*f. Final result*

*Figure 16: Interactive warping*


*a. Unwarped image* *b. Translation*

*c. Scaling* *d. Rotation*

*Figure 17: Local warps applied to a face*

where $\vec{c}$ is the center position, $\vec{m}$ is the current mouse cursor position, and $r_{\max}$ is the radius of the area of influence.

In Figure 19, the mapping function for points along the line through the initial and current position of the mouse has been plotted for a number of different mouse positions, with $r_{\max}$ normalized to 1.

### 4.4.2. Local scaling warps

The mapping used for a local scaling warp maps each point within the area of influence onto a point in the source image whose distance vector from the center of the area has the same orientation as that in the destination image, but whose length is a function $f_s(r)$ of the length $r$ of the vector.

The function chosen for $f_s(r)$ is

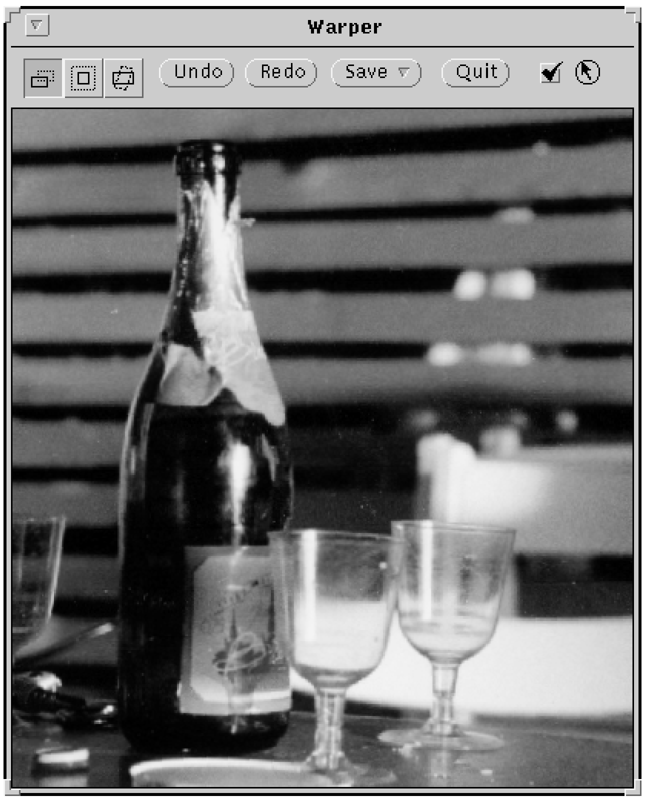


*Figure 18: OpenLook user interface*

$$f_s(r) = \left(1 - \left(\frac{r}{r_{\max}} - 1\right)^2 a\right) r$$

where the parameter $a$ is controlled by the current horizontal position of the mouse, ranging from $-1$ when the mouse is at the left edge of the area of interest to $+1$ when the mouse is at the right edge of the area of interest. Note that the function reduces to the identity function $f_s(r) = r$ when $a = 0$.

In Figure 20, the function $f_s(r)$ for $r_{\max} = 1$ has been plotted for a number of values of $a$ in the range $-1 \ldots 1$.

### 4.4.3. Local rotation warps

In local rotation warps, each point within the area of influence is mapped onto a point in the source image whose distance vector from the center of

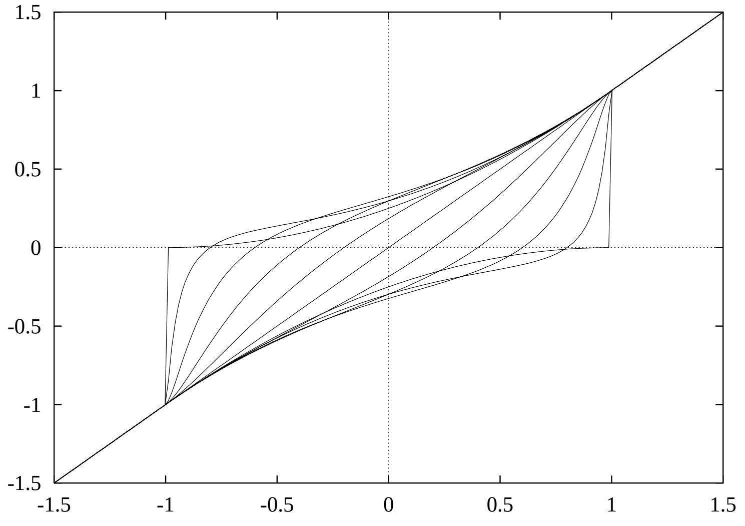


*Figure 19: A family of local translation warping functions*

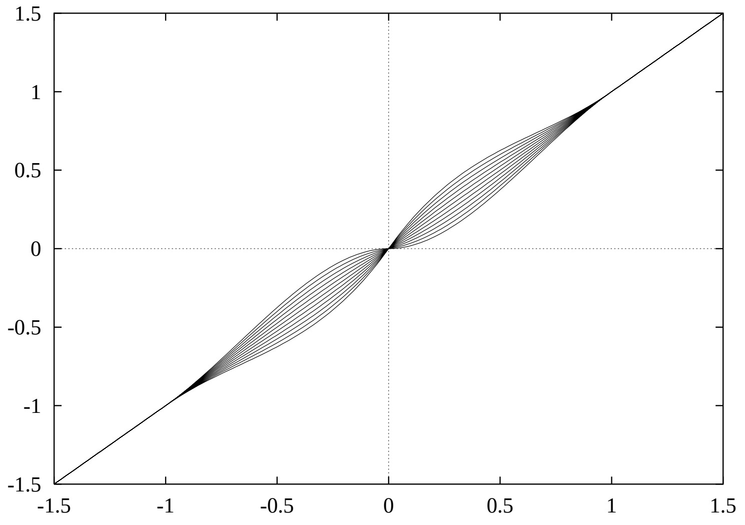


*Figure 20: A family of local scaling warping functions*

the area has the same length as in the destination image, but whose angle is a function $f_r(\theta)$ of the angle $\theta$ of the destination vector.

The function $f_r(\theta)$ is defined as $f_r(\theta) = \theta + (1 - r^2 \,/\, r_{\mathrm{max}}^2)^2\, \alpha$, where $r$ is the length of the destination vector, $r_{\mathrm{max}}$ is the radius of the area of influence, and $\alpha$ is a parameter that defines the rotation angle depending on the current mouse position.

The rotation angle $\alpha$ depends on the direction of the vector from the center of the area to the current mouse position, such that moving the mouse circularly around the center point causes a corresponding local rotation of the central parts of the image. In extreme cases the accumulated rotation may exceed $180°$, bending the image into a spiral shape.

During the initial part of the mouse stroke, when the mouse position is just a few pixels away from the origin, the calculated value for the angle between the mouse position and the origin would fluctuate wildly because the mouse position is only known to pixel accuracy. To avoid this problem, the angle is ignored until the mouse has exceeded a certain fixed distance, say five pixels, from the origin. The angle at which this occurs is used as the reference point $\alpha = 0$, and all further mouse motion is interpreted relative to this angle.

Figure 21 illustrates how a a horizontal line passing through the center of the area of influence is warped by local rotations from $0°$ to $180°$.

## 4.5. *Combining the transformations*

In all of the local warping systems discussed above, the final warp can be considered as a combination of a number of simple primitive transformations, each defined by a few control points. In the case of field morphing the control points are the endpoints of the line segments, and in Uwarp they are the center, edge, and final mouse positions.

In grid warping and field morphing systems the user typically positions all the control points before the warped image is generated. All control point coordinates are therefore interpreted as coordinates either in the original, unwarped image or in the final, warped image, and the order in which the control points are placed is immaterial. In these systems, the combined transformation of a pixel may be determined as the average of the primitive transformations weighted by some measure of proximity between the pixel and the control points for each primitive transformation.

In Uwarp, however, the result of each primitive transformation is displayed in real time as the control points are being positioned. This affects the way in which the control point coordinates are interpreted. The user is looking at an image that may already have been partially warped, and will place the control points at the location of image features as they appear on the screen, in a partially warped coordinate space. Therefore the control

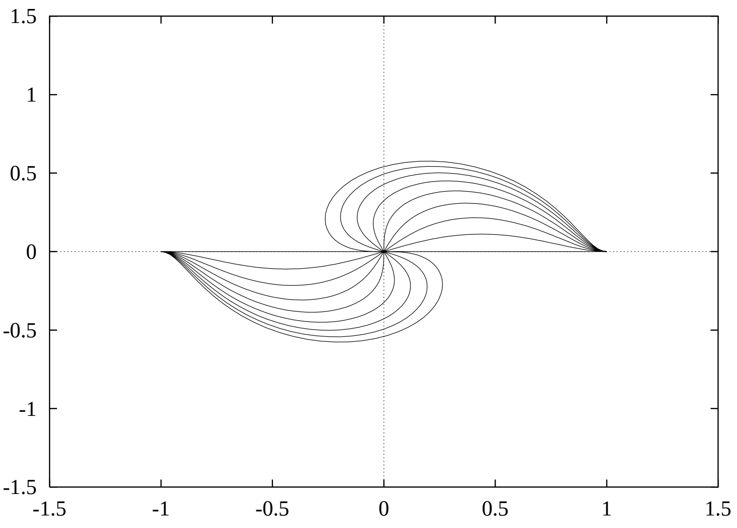


*Figure 21: Local rotations of a horizontal line*

point coordinates for each primitive transformation need to be interpreted in a coordinate space warped by all previous transformations, and simple averaging as above will not give intuitive results.

Instead, Uwarp combines the primitive transformations using functional composition. Because an inverse mapping is used, the coordinates of each pixel in the destination image are warped by applying each primitive transformation in turn, beginning with the most recent one:

$$u = f_1(f_2(\ldots f_n(x)))$$

Warping every pixel by every transformation may seem costly at first, but that is not really the case because the primitive transformations $f_n$ are local, i.e., $f(x) = x$ outside the transformation's area of influence. By comparing the pixel coordinates against the bounding box of each primitive transformation, actual evaluation of the mapping function can be avoided in most cases. In fact, in typical warps most of the pixels in the image are not affected by *any* of the primitive transformations. When the image is resampled using a high-quality resampling filter, the time spent in filtering usually exceeds the time spent evaluating the mapping functions.

## 4.6. On-demand calculation

Uwarp does not keep any warped image data in memory. Instead it just stores the original image and the coefficients for all the transformations that have been applied to it so far. Whenever there is a need to redisplay some part of the warped image, the needed pixels are simply recalculated on demand taking into account all the transformations.

During the interaction to specify a primitive transformation, dragging the mouse causes the parameters of the transformation to be continuously updated according to the current mouse coordinates. The mouse motion events also cause all pixels within the bounding box of the current transformation to be recalculated and displayed using the updated parameters. In order to avoid accumulating a large backlog of time-consuming redisplays when motion events arrive in rapid succession, a redisplay is initiated only when there are no further unprocessed motion events in the event queue.

Window exposure events also cause the exposed areas to be recalculated from the original image and warp parameters. In fact, the Uwarp prototype does *all* its display image calculations in the exposure event handler and performs redisplay during the interaction by sending exposure events to itself.

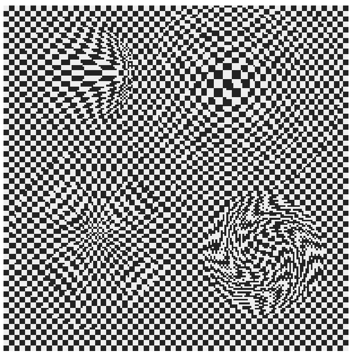
This on-demand calculation scheme makes it trivial to implement unlimited "undo" and "redo" operations. Each primitive transformation added is pushed on a stack; undoing the last operation amounts to popping the transformation from the stack, pushing it onto a separate "redo stack", and redisplaying the bounding box of the transformation (using those transformations that remain on the main stack). Redoing amounts to popping a transformation from the redo stack, pushing it on the main stack, and redisplaying the bounding box of the transformation. Whenever a new transformation is added to the main stack by means other than redoing, the redo stack is cleared.

## 4.7. *Filtering*

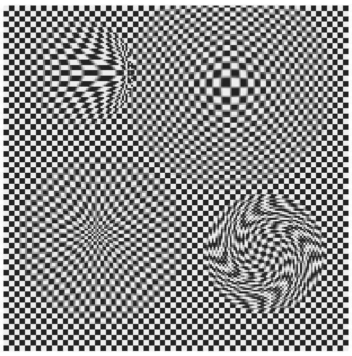
During user interaction, a fast interactive response is considered more important than flawless image quality. Therefore Uwarp calculates the warped image using nearest-neighbor sampling during the interaction stage. When the final warp parameters have been determined, the image is rewarped noninteractively using proper filtering.

The prototype implementation offers the user a choice between several filters offering different tradeoffs between quality and speed. These include (in order of increasing quality and decreasing speed): nearest neighbor interpolation (giving results identical to those on the screen), bilinear interpolation, and a unified EWA filter based on the principles described in Section 3.6, using the Lanczos2D filter kernel described in Section 3.4.4.

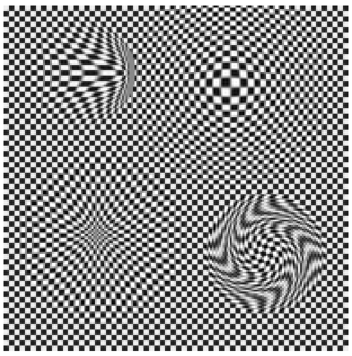
The local derivatives needed by the EWA filter are estimated using first-order differences between the warped coordinates of each pixel and its left and top neighbors. For this, a one-scanline buffer of warped coordinates is maintained. The topmost pixel row and leftmost pixel column are arbitrarily assigned local derivatives corresponding to a one-to-one warp.



*a. Nearest neighbor*



*b. Bilinear interpolation*



*c. EWA filter*

*Figure 22: Checkerboard image warped using different interpolation methods*

Figure 22 shows the results of applying some local warps to a checkerboard image using nearest neighbor interpolation (Figure 22*a*), bilinear interpolation (Figure 22*b*), and EWA interpolation (Figure 22*c*).

The warps applied are a translation (upper left in each subfigure), magnification (upper right), minification (lower left), and a rotation (lower right).

It can be seen that the bilinear interpolation works reasonably well for magnification, but the areas undergoing strong minification in the two left-hand warps show strong aliasing patterns when warped using bilinear interpolation. This is to be expected because the bilinear interpolation is a pure interpolation filter with a fixed-size kernel, so it doesn't even attempt to handle the decimation case properly.

## 4.8. Performance

This section presents some performance results for the prototype implementation of Uwarp. All measurements were made on a Sun SPARCstation 2 running SunOS version 4.1.1. When measuring the interactive timings, a VITec RasterFLEX-32 framebuffer and the VITec X server were used. The program makes use of the MIT shared memory extension to the X11 protocol to speed up the display of raster images. The program was compiled with the GNU C compiler, version 2.2.2.

The interactive response is quite good when warping areas of moderate size. For example, when warping a 24-bit color image by a translational warp with a radius of 100 pixels, the warped image within the 200 by 200 pixel bounding box is updated once every 350 milliseconds.

The time required for noninteractive warping depends on the type of resampling filter used. Timings for the 405 by 443 pixel image in Figure 16*f* are presented in Table 2.

| Interpolation | Time |
|---|---|
| Nearest neighbor | 1.5 s |
| Bilinear | 4.1 s |
| EWA | 26 s |

*Table 2: Timings for noninteractive warping*

The image calculated for Table 2 contains only translational warps. Scaling warps are somewhat slower because their mapping function includes a square root, and rotation warps are the slowest because their mapping function requires the calculation of a sine/cosine pair. The sine and cosine values need not be very precise, and an attempt was made to exploit this fact to speed up calculation by using lower-precision Chebyshev polynomial approximations of the sine and cosine functions. This turned out to be only slightly faster than the library `sin()` and `cos()` functions and about the same speed as the library `sincos()` function, so this approach was abandoned in favor of `sincos()`.

Both the evaluation of the mapping functions and the EWA filtering are done completely in floating point. Greene and Heckbert in their 1986 paper [12] suggest using integer arithmetic (or, more correctly, fixed-point arithmetic) as a way to speed up filtering, but recent trends in computer architecture have led to floating point being faster than fixed point on most new machines, especially when the calculations involve many multiplications. For example, on a Sun SPARCstation 2, which lacks an integer multiply instruction and has to perform integer multiplication by means of a library subroutine, a 32-bit integer multiply is several times slower than a 64-bit floating-point multiply.

The lack of an integer multiply instruction also had an unexpected side effect: an integer multiplication that was part of some array indexing calculations caused a major slowdown of the EWA filtering, not because of the actual multiplication time, but because the C compiler made the overly cautious assumption that the multiplication library routine might invalidate the floating point registers. This resulted in the compiler repeatedly saving and reloading a large number of double-precision values from memory. Replacing this one multiplication by a loop of repeated additions increased the EWA filtering speed by about 30 %.

Most of the time spent in EWA filtering appears to be in the multiplication of pixel component values with filter kernel values, despite the fact that the calculation of the ellipse parameters is quite heavy and involves two square roots and several divides. Although the SPARC processor has a fairly fast floating-point multiply, converting the 8-bit pixel component values to floating point is a serious bottleneck because they must first be loaded into an integer register and then transferred via a temporary memory location to a floating point register for conversion.

## 4.9. Future directions

The work presented in this chapter has explored only a small class of mapping functions acting on a circular environment around a point. Many other classes of mapping functions could conceivably be performed using interactive user interfaces similar to the ones described above.

For instance, one might warp the environments of line segments as in Beier and Neely [2], but use local mapping functions to reduce the amount of calculation needed for redisplay.

Uwarp's on-demand calculation scheme could also be applied to other local image transformations in addition to warping. For example, painting operations such as brush strokes could be treated similarly to warps, modifying the color of the underlying image rather than its geometry. An advantage of this scheme over the traditional approach of immediately modifying the pixels of the original image is that unlimited undoing and redoing of individual painting operations would be trivial to implement.

Work also remains to be done in the area of filtering. Although the EWA filtering method gives high-quality results, it does so at a considerable computational expense.

One fundamental problem with the EWA in the context of local warps is that the result of resampling according to an identity transformation may be slightly different from the original image. This would be the case even if the EWA filter were ideal, because the circular symmetry of the filter causes it to remove the spatial frequencies in the region falling between the circle with diameter $f_s$ and the square with side $f_s$ around $f = 0$.

Because of this, EWA filtering only the areas that are actually affected by the warps could cause visible seams between the filtered and the unfiltered areas. Therefore, the whole image must be filtered, which is wasteful because typical warps only affect a small part of the image. Filtering by bilinear interpolation does not suffer from this problem, but it is of lower quality and causes aliasing in image areas undergoing minification. A high-quality, space-variant filter which preserves the original image exactly in the case of an identity resampling would therefore be very useful in reducing the amount of calculation.

There are still performance problems to be solved. Although the system is very responsive when mapping areas of moderate size (up to a few hundred by a few hundred pixels) the time to update the warped area grows quadratically with the radius of the warped area and linearly with the number of warps already applied to the same area. The fast initial response of the system easily lures users into doing larger or more complicated warps than it can comfortably handle, and as the response time quickly increases, the users get confused due to lack of feedback.

One way to avoid the rapid increase in response time as more warps are added could be keeping some of the intermediate, partly warped images in memory and calculating subsequent display images from those intermediate images rather than from the original image. Because this would cause some additional degradation of image quality due to multiple resampling steps, the final, filtered image should still be calculated directly from the original image.

# Chapter 5. Conclusion

This thesis has reviewed the fundamental theory, methods, and applications of image warping. Chapter 3 showed that filtering is necessary to avoid aliasing and reconstruction artifacts, and reviewed the theory of space-variant resampling filters. It also presented a number of well-known methods for performing image filtering, ranging from simple interpolation and decimation filters to the high-quality space variant EWA filter of Greene and Heckbert.

A general image warp may involve magnification along one coordinate axis and minification along another. A method for handling this situation in the case of EWA filtering with an arbitrary circularly symmetric filter kernel was proposed and contrasted with Heckbert's solution, which involves Gaussian filter kernels. The method was implemented as part of the image warping system described in Chapter 4.

The main contribution of the thesis is the design of a new kind of highly interactive image warping system aimed at creative graphic design applications. This system, described in Chapter 4, allows a complex warp to be built incrementally from a series of simple, local transformations. A simple and intuitive graphical user interface lets the user specify each transformation with a single mouse stroke, during which the warped image is displayed in real time. High-quality filtered results are obtained in a separate non-interactive warping pass.

The feasibility of this approach is demonstrated in an implementation, and performance measurements indicate that current workstation computers are fast enough for real-time warping of areas several hundred pixels across.

Current painting and retouching programs allow the artist to add or modify color using a large number of tools, including both tools for drawing strict geometric shapes and airbrush-like tools for creative freehand drawing. So far the image warping facilities in such software have only provided tools of the former kind — there has been no such thing as a "warp brush". This system is an attempt to fill that gap. Thus, it is not a replacement for existing image warping methods, but rather a new addition to the toolbox of the graphics artist, enabling a new style of creative warping that has not been possible before.

# Appendix A. The ideal circularly symmetric lowpass filter

The frequency response of the ideal circularly symmetric lowpass filter with a cutoff frequency of 1 is

$$F(u,v) = \begin{cases} 1, & u^2+v^2 < 1 \\ 0, & u^2+v^2 \geq 1 \end{cases}$$

The impulse response of a filter is the inverse Fourier transform of its frequency response:

$$f(x,y) = \iint F(u,v)\, e^{2\pi i\,(ux+vy)}\, du\, dv$$

Because the frequency response is circularly symmetric around the origin, the impulse response is also circularly symmetric. Therefore it suffices to evaluate the impulse response along the $x$ axis:

$$\begin{aligned} f(x,0) &= \int_{-1}^{1} \left[ \int_{-\sqrt{1-u^2}}^{\sqrt{1-u^2}} e^{2\pi i u x}\, dv \right] du \\ &= \int_{-1}^{1} 2\sqrt{1-u^2}\, e^{2\pi i u x}\, du \\ &= \int_{-1}^{1} 2\sqrt{1-u^2}\, (\cos(2\pi u x) + i \sin(2\pi u x))\, du \\ &= 2\int_{-1}^{1} \sqrt{1-u^2}\, \cos(2\pi x u)\, du + 2i \int_{-1}^{1} \sqrt{1-u^2}\, \sin(2\pi x u)\, du, \end{aligned}$$

where the second term is zero because $\sin x$ is odd, giving

$$f(x,0) = 4\int_{0}^{1} \sqrt{1-u^2}\, \cos(2\pi x u)\, du$$

Abramowitz and Stegun [1] give the identity

$$J_\nu(z) = \frac{(\frac{1}{2}\, z)^\nu}{\pi^{\frac{1}{2}}\, \Gamma(\nu + \frac{1}{2})} \int_0^1 (1-t^2)^{\nu - \frac{1}{2}} \cos(zt)\, dt, \quad R\nu > -\tfrac{1}{2},$$

where $J_\nu(z)$ are the Bessel functions of the first kind. By substituting $\nu = 1$, $t = u$, $z = 2\pi x$, and using the identity $\Gamma(1+\frac{1}{2}) = \frac{1}{2}\sqrt{\pi}$[†] we get

$$\int_0^1 \sqrt{1-u^2}\,\cos(2\pi x u)\,du = \frac{\pi}{2}\frac{J_1(2\pi x)}{2\pi x}$$

and

$$\begin{aligned} f(x,0) &= 2\pi\,\frac{J_1(2\pi x)}{2\pi x} \\ &= \pi\,(J_0(2\pi x) + J_2(2\pi x)) \end{aligned}$$

By circular symmetry,

$$f(r) = 2\pi\frac{J_1(2\pi r)}{2\pi r} = \pi\,(J_0(2\pi r) + J_2(2\pi r)), \quad r = \sqrt{x^2+y^2}$$

[†] This follows from $\Gamma(\frac{1}{2}+z)\,\Gamma(\frac{1}{2}-z) = \pi\sec(\pi z)$ and $\Gamma(z+1) = z\Gamma(z)$ by substituting $z = 0$.

# Appendix B. Source code for local mappings

This appendix contains annotated source code for a few central functions in the interactive image warper implementation of Chapter 4, illustrating the algorithms for determining the warp parameters from a series of mouse positions and for evaluating the mapping defined by those parameters.

The main data structure of the warper is the stack of primitive transformations. Each primitive transformation is represented by a structure of type `warp_control`. This structure contains the type and parameters of the transformation, and also some additional values that are heavily used by the actual mapping code and therefore precalculated when the transformation is set up.

All three types of transformations require some common parameters, namely the center point and radius of the transformation. Additionally, the different types of transformations require some type-dependent parameters: a translation warp needs a displacement vector, a scaling warp needs a scale factor, and a rotation warp needs a rotation angle.

```
typedef struct warp_control
{
    struct warp_control *next;
    int warp_type;              /* TRANSLATE, SCALE, or ROTATE */
    double orig_x, orig_y;      /* center point */
    double max_dist;            /* radius */
    double mou_dx_norm, mou_dy_norm; /* normalized mouse delta */
    double max_dist_sq;         /* the square of max_dist */
    double mou_dx, mou_dy;      /* unnormalized mouse delta */
    double angle;               /* rotation angle */
    double angle_offset;        /* rotation reference angle */
} warp_control;
```

The state of the warper is encapsulated in a structure of type `warper`. The stack of primitive transformations is implemented as a linked list of `warp_control` structures, and the undo stack is a similar list.

```
typedef struct warper_struct
{
    warp_control *controls;     /* list of primitive warps */
    warp_control *redo_list;    /* undone warps saved for redoing */
} warper;
```

The mapping code repeatedly calculates squares of distances between points, so a convenience routine is defined for that operation:

```
#define sqr(x) ((x)*(x))
double hypotsq(x, y)
    double x, y;
{
        return x*x + y*y;
    }
```

When a new canvas interaction begins, a new primitive transformation is created and initialized with the initial mouse position and the transformation type:

```
    void warper_create_control(w, x, y, type)
        warper *w;
        int x, y;
        int type;
    {
        warp_control *c = (warp_control *) malloc(sizeof(warp_control));
        c->orig_x = (double) x;
        c->orig_y = (double) y;
        c->mou_dx_norm = c->mou_dy_norm = 0.0;
        c->mou_dx = c->mou_dy = 0.0;
        c->max_dist = 0.0;
        c->max_dist_sq = 0.0;
        c->angle = 0.0;
        c->angle_offset = NO_ANGLE;
        c->warp_type = type;
        c->next = w->controls;
        w->controls = c;
        /* clear the redo list */
        free_control_list(w->redo_list);
        w->redo_list = 0;
    }
```

Subsequent dragging of the mouse causes the topmost transformation on the transformation stack to be updated with parameters corresponding to the new mouse position:

```
    void warper_move_control(w, x, y)
        warper *w;
        int x, y;
    {
        warp_control *c = w->controls;
        double dx, dy;
        double cur_dist;
        double cur_dist_sq;
        dx = (double) x - c->orig_x;
        dy = (double) y - c->orig_y;
        cur_dist = hypot(dx, dy);
        /* keep track of maximum distance */
        if(cur_dist > c->max_dist)
            c->max_dist = cur_dist;
        cur_dist_sq = sqr(cur_dist);
        if(c->max_dist) /* avoid division by zero */
        {
            c->mou_dx_norm = dx / c->max_dist;
            c->mou_dy_norm = dy / c->max_dist;
        }
        else
        {
            c->mou_dx_norm = 0.0;
            c->mou_dy_norm = 0.0;
        }
        if(c->warp_type == ROTATE)
        {
            if(cur_dist_sq >= sqr(5.0))
            {
                double new_angle, delta_angle;
                new_angle = atan2(dy, dx);
                if(c->angle_offset == NO_ANGLE)
                    c->angle_offset = new_angle;
                delta_angle = (new_angle - c->angle_offset) - c->angle;
                while(delta_angle > pi)
                    delta_angle -= 2.0*pi;
                while(delta_angle < 0.0-pi)
                    delta_angle += 2.0*pi;
                c->angle += delta_angle;
            }
        }
        c->max_dist_sq = sqr(c->max_dist);
        c->mou_dx = c->mou_dx_norm * c->max_dist;
        c->mou_dy = c->mou_dy_norm * c->max_dist;
    }
```

After the parameters have been adjusted, all pixels within the bounding box of the updated transformation will be redisplayed by mapping their display coordinates to source image coordinates and point sampling the source image at the nearest pixel.

The actual evaluation of the combined mapping defined by the stack of primitive transformations is done by the function `mapping()`, which takes a pair of destination coordinates $(x, y)$ and maps them to source coordinates $(u, v)$.

```
    void mapping(controls, x, y, u, v)
        warp_control *controls;
        double x, y;
        double *u, *v;
    {
        warp_control *c;
        double fu, fv;
        fu = x;
        fv = y;
        for(c=controls; c; c=c->next)
        {
            double dx = fu - c->orig_x;
            double dy = fv - c->orig_y;
            /* bounding box check */
            if(dx > 0.0-c->max_dist &&
               dx < c->max_dist &&
               dy > 0.0-c->max_dist &&
               dy < c->max_dist)
            {
                /* rsq is the squared distance from the center of */
                /* the affected area to this pixel */
                double rsq = hypotsq(dx, dy);
                if(rsq < c->max_dist_sq)
                    switch(c->warp_type)
                    {
                    case TRANSLATE:
                    {
                        double msq =
                            hypotsq(dx - c->mou_dx, dy - c->mou_dy);
                        double edge_dist = c->max_dist_sq - rsq;
                        double a = edge_dist / (edge_dist + msq);
                        a *= a;
                        fu -= a * c->mou_dx;
                        fv -= a * c->mou_dy;
                    }
```

```
                        break;
                        case SCALE:
                        {
                            double rnorm = sqrt(rsq / c->max_dist_sq);
                            double a = 1 - c->mou_dx_norm * sqr(rnorm-1.0);
                            fu = c->orig_x + a * dx;
                            fv = c->orig_y + a * dy;
                        }
                        break;
                        case ROTATE:
                        {
                            double dx1, dy1, si, cs;
                            double a = 1 - (rsq / c->max_dist_sq);
                            double alpha = sqr(a) * c->angle;
                            si = sin(alpha); cs = cos(alpha);
                            dx1 = dx * cs + dy * si;
                            dy1 = dy * cs - dx * si;
                            fu = c->orig_x + dx1;
                            fv = c->orig_y + dy1;
                        }
                        break;
                    }
                }
            }
            *u = fu;
            *v = fv;
        }
```

The complete program of course contains a large amount of additional code implementing the OpenLook user interface, EWA filtering with various filter kernels, image file support, fast image display through the X shared memory extension, and various support routines.

## References

[1] Milton Abramovitz and Irene A. Stegun. *Handbook of Mathematical Functions*, page 360. Dover Publications, 1972.

[2] Thaddeus Beier and Shawn Neely. Feature-based image metamorphosis. *Computer Graphics*, 26(2):35–42, Jul 1992. (SIGGRAPH '92 proceedings).

[3] James F. Blinn. Return of the jaggy. *IEEE Computer Graphics and Applications*, 9(2):82–89, Mar 1989.

[4] James F. Blinn and Martin E. Newell. Texture and reflection in computer generated images. *Communications of the ACM*, 19(10):542–547, Oct 1976.

[5] Franklin C. Crow. Summed-area tables for texture mapping. *Computer Graphics*, 18(3):207–212, Jul 1984. (SIGGRAPH '84 proceedings).

[6] Dan E. Dudgeon and Russell M. Mersereau. *Multidimensional Digital Signal Processing.* Prentice-Hall, 1984.

[7] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics—Principles and Practice.* Addison-Wesley, second edition, 1990.

[8] Alain Fournier and Eugene Fiume. Constant-time filtering with space-variant kernels. *Computer Graphics*, 22(4):229–238, Aug 1988. (SIGGRAPH '88 proceedings).

[9] Carl Frederic and Eric L. Schwarz. Conformal image warping. *IEEE Computer Graphics and Applications*, 10:54–61, Mar 1990.

[10] M. Gagnet, D. Perny, and P. Coueignoux. Perspective mapping of planar textures. In *Eurographics '82*, pages 57–71, Sep 1982.

[11] Rafael C. Gonzalez and Paul Wintz. *Digital Image Processing.* Addison-Wesley, second edition, 1987.

[12] Ned Greene and Paul Heckbert. Creating raster Omnimax images from multiple perspective views using the elliptical weighted average filter. *IEEE Computer Graphics and Applications*, 6(6):21–27, Jun 1986.

[13] Valerie Hall. Introduction to morphing. Student paper, School of Computing Science, Curtin University of Technology, Australia,

Jul 1992. Available online from `marsh.cs.curtin.edu.au` in `/pub/graphics/bibliography/Morph`.

[14] Frederic J. Harris. On the use of windows for harmonic analysis with the discrete Fourier transform. *Proc. IEEE*, 66(1):51–83, Jan 1978.

[15] Paul Heckbert. Filtering by repeated integration. *Computer Graphics*, 20(4):315–321, Aug 1986. (SIGGRAPH '86 proceedings).

[16] Paul Heckbert. Survey of texture mapping. *IEEE Computer Graphics and Applications*, 6(11):56–67, Nov 1986.

[17] Paul Heckbert. Fundamentals of texture mapping and image warping. Master's thesis, Dept. of EECS, U. of California at Berkeley, Berkeley, California 94720, Jun 1989. Technical Report No. UCB/CSD 89/516.

[18] D.P. Mitchell and A.N. Netravali. Reconstruction filters in computer graphics. *Computer Graphics*, pages 221–228, 1988. (SIGGRAPH '88 proceedings).

[19] Masaaki Oka, Kyoya Tsutsui, Akio Ohba, Yoshitaka Kurauchi, and Taakashi Tago. Real-time manipulation of texture-mapped surfaces. *Computer Graphics*, 21(4):181–188, Jul 1987. (SIGGRAPH '87 proceedings).

[20] J. Anthony Parker, Robert V. Kenyon, and Donald E. Troxel. Comparison of interpolating methods for image resampling. *IEEE Transactions on Medical Imaging*, MI-2(1):31–39, Mar 1983.

[21] Mark J. Pavicic. Convenient anti-aliasing filters that minimize "bumpy" sampling. In Andrew S. Glassner, editor, *Graphics Gems*, pages 144–146. Academic Press, 1990.

[22] Thomas Porter and Tom Duff. Compositing digital images. *Computer Graphics*, 18(3):253–259, Jul 1984. (SIGGRAPH '84 proceedings).

[23] William K. Pratt. *Digital Image Processing.* John Wiley and Sons, New York, 1978.

[24] Tom Sauer and Danielle Argiro. *WARPIMAGE: Interactive Image Warping Application.* University of New Mexico, Mar 1992. Khoros User's Manual, Volume 1, Chapter 7.

[25] Ronald W. Schafer and Lawrence R. Rabiner. A digital signal processing approach to interpolation. *Proc. IEEE*, 61(6), Jun 1973.

[26] Ken Turkowski. Filters for common resampling tasks. In Andrew S. Glassner, editor, *Graphics Gems*, pages 147–165. Academic Press, 1990.

[27] Lance Williams. Pyramidal parametrics. *Computer Graphics*, 17(3):1–11, Jul 1983. (SIGGRAPH '83 proceedings).

[28] George Wolberg. *Digital Image Warping.* IEEE Computer Society Press, Los Alamitos, California, 1990.

[29] George Wolberg. Digital image warping. *IEEE Computer Graphics and Applications*, 11:114–116, Jan 1991. Book review by Paul Heckbert.

[30] George Wolberg, Mike McDonnell, and Paul Heckbert. Different views: Digital image warping. *IEEE Computer Graphics and Applications*, pages 4–5, May 1991. Letters to the Editor.