CSC320

Introduction to Visual Computing

Sinan Li

2024

CONTENTS

Part I

Foundations

CHAPTER

1 IMAGE GEOMETRY

1.1 Linear 2D Transforms

1.1.1 Homogeneous 2D Point Coordinate

Basic Notational Conventions

- Column-Vector Representation

$$\begin{bmatrix} x \\ y \end{bmatrix}$$

- Row-Vector Representation

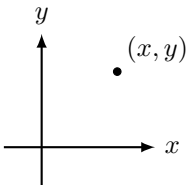
$$\begin{bmatrix} x & y \end{bmatrix}$$

- Matrix Transpose Operation

$$\begin{bmatrix} x \\ y \end{bmatrix}^\top = \begin{bmatrix} x & y \end{bmatrix}$$

Homogeneous Coordinates

In the standard Euclidean coordinate system, a point is represented by a pair of coordinates (x, y), where x and y are the coordinates of the point along the x and y axes, respectively.



In homogeneous coordinates, a point is represented by a triple of coordinates (x, y, w), where x and y are the coordinates of the point along the x and y axes, respectively, and w is a scaling factor that is normally set to 1. The homogeneous coordinates of a point are not unique, since any multiple of the coordinates (x, y, w) represents the same point. For example, $(2, 3, 1)$ and $(4, 6, 2)$ represent the same point. The homogeneous coordinates of a point are unique only up to a scaling factor.

Remark

Two vector of homogeneous coordinates (x, y, w) and (x', y', w') represent the same point if and only if there exists a non-zero scalar λ such that

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \lambda \begin{bmatrix} x' \\ y' \\ w' \end{bmatrix}$$

Example. The homogeneous coordinates of the point $(2, 3)$ are $(2, 3, 1)$, $(4, 6, 2)$, $(6, 9, 3)$, etc. ♢

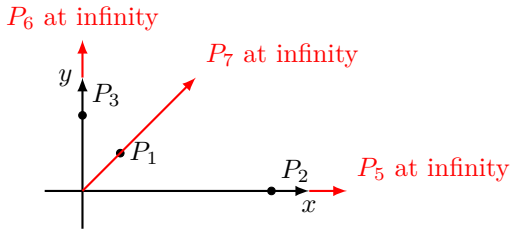
Homogeneous to Euclidean Coordinat

To convert a homogeneous coordinate (x, y, w) to a Euclidean coordinate (x', y'), we divide the first two coordinates of the homogeneous coordinate by the third coordinate, i.e.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x/w \\ y/w \end{bmatrix}$$

Example. Plot the following points in the Euclidean plane:

$$\begin{array}{lll} P_1 = (2, 2, 2) & P_2 = (10, 0, 2) & P_3 = (0, 8, 4) \\ P_4 = (1, 0, 0.01) & P_5 = (1, 0, 0) & P_6 = (0, 1, 0) \\ P_7 = (1, 1, 0) & & \end{array}$$



Note that P_5, P_6, and P_7 are at infinity. This is a very important property of homogeneous coordinates. ♢

Remark

Points infinitely far away from the origin in the Euclidean plane have a finite representation in homogeneous coordinates (i.e. $(x, y, 0)$). These points are sometimes called **ideal points**.

1.1.2 Homogeneous 2D Line Coordinates

Homogeneous 2D Line Coordinates

Remark

The general equation of a line in the Euclidean plane is

$$ax + by + c = 0$$

where a, b, and c are real numbers and a and b are not both zero.
In homogeneous coordinates, using the matrix notation, the general equation of a line is

$$\begin{bmatrix} a & b & c \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} = 0$$

Or, equivalently,

$$\ell^\top p = 0$$

where ℓ is the vector holding line coefficients and p is the vector holding point coordinates.

Definition 1.1.1

The **homogeneous coordinates of a line** with the equation

$$ax + by + c = 0$$

is the vector

$$\ell = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

Example. The homogeneous coordinates of the line $y = x$ is

$$\ell = \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix}$$

Line Passing Through Two Points

In this case, ℓ must satisfy the following equations:

$$\ell^\top P_1 = 0 \qquad \ell^\top P_2 = 0$$

where p_1 and p_2 are the homogeneous coordinates of the two points.

We can take the cross product of p_1 and p_2 to obtain ℓ:

$$\ell = P_1 \times P_2$$

> **Remark**
>
> To compute the cross product of two vectors, we can use the following two methods.
>
> $$\ell = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} \times \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix}$$
>
> As matrix multiplication:
>
> $$P_1 \times P_2 = \begin{bmatrix} 0 & -z_1 & y_1 \\ z_1 & 0 & -x_1 \\ -y_1 & x_1 & 0 \end{bmatrix} P_2$$
>
> As determinant:
>
> $$P_1 \times P_2 = \begin{vmatrix} i & j & k \\ x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \end{vmatrix}$$

Intersection of Two Lines

In this case, p must satisfy the following equations:

$$\ell_1^\top p = 0 \qquad \ell_2^\top p = 0$$

where ℓ_1 and ℓ_2 are the homogeneous coordinates of the two lines.

We can take the cross product of ℓ_1 and ℓ_2 to obtain p:

$$p = \ell_1 \times \ell_2$$

> **Remark**
>
> For parallel lines, we cannot compute its standard Euclidean intersection point. Instead, we can compute the homogeneous intersection point, which will be at infinity.

1.1.3 Affine 2D Transformations

- **Identity**

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} \cong \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- **Stretching** (along x)

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} \cong \begin{bmatrix} sx \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

  Stretching (along y)

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} \cong \begin{bmatrix} x \\ sy \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- **Shearing** (along y)

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} \cong \begin{bmatrix} x \\ hx + y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ h & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- **Rotation** (about the origin bt angle ϕ)

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} \cong \begin{bmatrix} (\cos\phi)x - (\sin\phi)y \\ (\sin\phi)x + (\cos\phi)y \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- **Translation** (along x)

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} \cong \begin{bmatrix} x + d \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & d \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

  Translation (along y)

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} \cong \begin{bmatrix} x \\ y + d \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & d \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

We can combine these transformations to form more complex transformations. For example, we can combine a rotation and a translation to form a **rotation about a point** transformation.

Definition 1.1.2 Affine Transformation (Intuitive)

An **affine transformation** is any combination of scaling, shearing, rotation, and translation.

$$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & g \end{bmatrix}$$

where $g \neq 0$ is a scaling factor that does not affect the transformation.

Remark

An affine transformation is a transformation that preserves parallelism and ratios of distances along parallel lines.

In the original image, two parallel lines would intersect at some infinity $(x, y, 0)$. After an affine transformation, the two lines would still intersect at some infinity $(x', y', 0)$.

$$\begin{bmatrix} x' \\ y' \\ 0 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & g \end{bmatrix} \begin{bmatrix} x \\ y \\ 0 \end{bmatrix}$$

Definition 1.1.3 Affine Transformation

An affine transformation is any invertible 3×3 matrix that preserves the points at infinity.

But what if the last row of the matrix is not $(0, 0, 1)$?

1.1.4 Homographies

Homographies are also called **projective transformations** or **perspective transformations**. They still preserve linearity, but they do not preserve parallelism.

Definition 1.1.4

A **homography** is any 2D transformation of homogeneous coordinates that is represented by an invertible 3×3 matrix.

$$\begin{bmatrix} a & b & c \\ d & e & f \\ l & m & g \end{bmatrix}$$

All homographies are invertible. This is because Homographies map unique points to unique points.

Remark

Lines remain straight after a homography.
Let $\ell^T p = 0$ be a line. It suffices to show that all points on this line are mapped to points on a line.

Example. A naive image warping algorithm is forward mapping.

```
input: src_image, H
output: dest_image

for c=1 to num_columns                // cycle over all source pixels
   for r=1 to num_rows
        x, y = pixel_xy(r,c)          // get the source pixel's (x,y) coordinates
        p = homogeneous_coords(x, y) // convert to vector homogeneous coords
        p_prime = H*p                 // apply homography
        x_prime, y_prime = euclidean_coords(p_prime)  // convert to Euclidean coords
        r_prime, c_prime = pixel_rc(x_prime, y_prime) // calculate the dest pixel
```

```
dest_image(r_prime, c_prime) = src_image(r,c) // copy pixel color
```

This algorithm has some problems:

- Magnification causes "holes" in destination image
- Minification causes overwriting of pixel contents

Backward-Mapping Algorithm

This is also known as the **inverse mapping algorithm**. This algorithm does not create "holes" in the destination image.

```
input: src_image, H
output: dest_image

for c_prime=1 to num_columns                          // cycle over all dest px
   for r_prime=1 to num_rows
       x_prime, y_prime = pixel_xy(r_prime,c_prime)   // get dest px's (x,y)
       p_prime = homogeneous_coords(x_prime, y_prime) // convert to homo. coords
       p = inverse(H)*p_prime                         // apply inverse homography
       x, y = euclidean_coords(p)                     // convert to Euclidean
       r, c = pixel_rc(x, y)                          // calculate source px
       dest_image(r_prime, c_prime) = src_image(r,c)  // copy px color
```

1.2 Perspective Viewing

1.2.1 Aperture

Without an aperture, every sensor receives every light, so the image is very blurry. We need an aperture to control the amount of light that enters the camera. The aperture acts as a diaphragm, allowing us to regulate the size of the opening through which light passes. By adjusting the aperture size, we can control the amount of light that reaches the sensor, resulting in a properly exposed image.

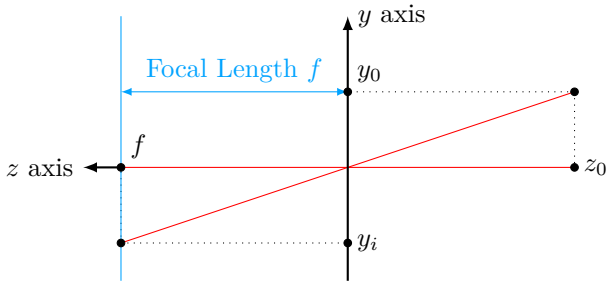
The image we get is up-side-down, and as we increase the aperture size, the image becomes more blurry.

1.2.2 Perspective Projection

Geometry of Perspective Projection

Definition 1.2.1 Focal Length

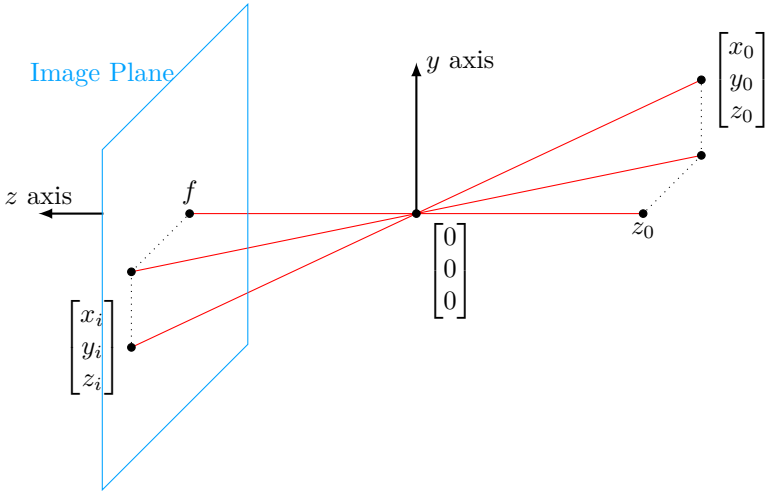
The **focal length** of a lens is the distance from the lens to the point where light rays converge to a point.



From similar triangles, we get

$$\frac{y_i}{y_0} = \frac{f}{z_0} \implies y_i = \frac{f}{z_0} \cdot y_0$$

The same applies to 3-dimensional points.



We have

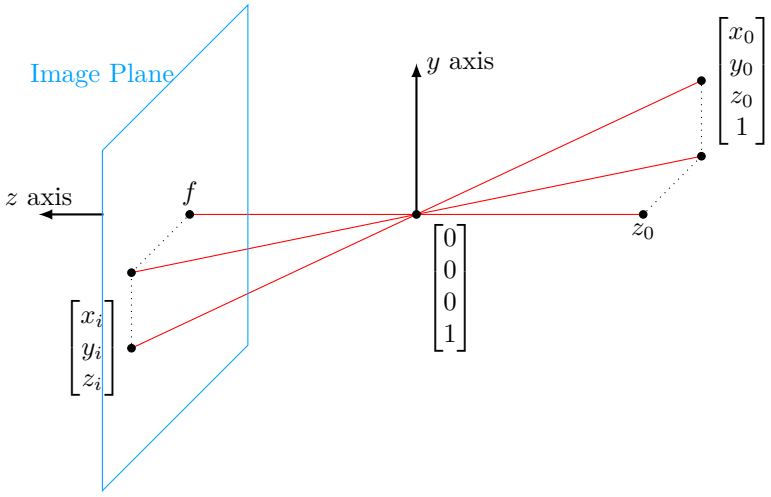
$$\begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} = \frac{f}{z_0} \cdot \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}$$

Interpretation of homogeneous equality tells us that "all 3D points have the same projection",

$$\begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} \cong \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}$$

In other words, $\begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix}$ represents a ray through the origin.

Homogeneous 3D Coordinates



Similar to 2D homogeneous coordinates, we can represent 3D points in homogeneous coordinates

$$\begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} \cong \begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{bmatrix}$$

To convert from the homogeneous coordinates to the Euclidean coordinates, we divide the first three coordinates by the fourth coordinate, i.e.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix} / w_0$$

Remark

The point

$$\begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ 0 \end{bmatrix}$$

is at infinity in 3D space, in the direction of the vector $\begin{bmatrix} x_0 \\ y_0 \\ z_0 \end{bmatrix}$.

Homogeneous coordinates provide us a convenient way to express the projective of a 3D point into a 2D image,

$$\begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{f}{z_0} \cdot x_0 \\ \frac{f}{z_0} \cdot y_0 \\ 1 \end{bmatrix} \cong \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{bmatrix}$$

where $\begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$ is the homogeneous coordinates of the 2D projection of the 3D point $\begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{bmatrix}$, and the only difference between the 2D and 3D projection matrices is the scaling factor $\frac{f}{z_0}$.

1.3 Alignment and Stitching

1.3.1 Estimating Homography from Point Correspondence

Claim. Given sufficiently many point correspondences we can compute tha homogeneous transformation H that aligns two images. We need at least 4 correspondences for a unique solution.

Given a point correspondence $\begin{bmatrix} {x_i}' \\ {y_i}' \\ 1 \end{bmatrix} \cong \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$, we can write the following equation

$$\begin{bmatrix} {x_i}' \\ {y_i}' \\ 1 \end{bmatrix} \cong H \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \iff \begin{bmatrix} {x_i}' \\ {y_i}' \\ 1 \end{bmatrix} \cong \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

which allows us to establish a system of linear equations

$$\begin{bmatrix} {x_i}' \\ {y_i}' \\ 1 \end{bmatrix} \cong \begin{bmatrix} ax_i + by_i + c \\ dx_i + ey_i + f \\ gx_i + hy_i + 1 \end{bmatrix} \iff \begin{bmatrix} {x_i}' \\ {y_i}' \\ 1 \end{bmatrix} \cong \begin{bmatrix} ax_i + by_i + c/gx_i + hy_i + 1 \\ dx_i + ey_i + f/gx_i + hy_i + 1 \\ 1 \end{bmatrix}$$

That is, one correspondence gives us two linear equations. We have 8 unknowns, so we need 4 correspondences to solve the system of linear equations.

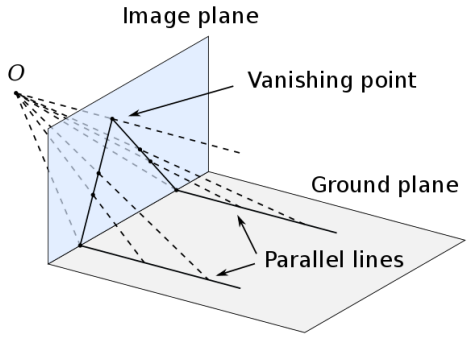
1.4 3D Metrology

1.4.1 3D Reconstruction from 2D Images

1.4.2 Vanishing Points

Definition 1.4.1

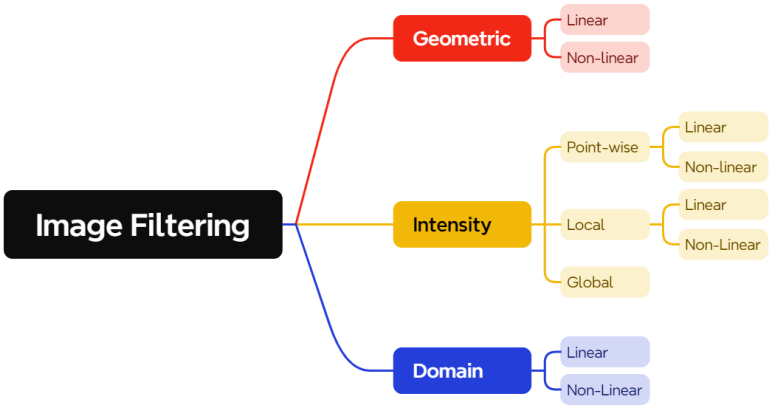
A **vanishing point** is a point on the image plane of a perspective rendering where the two-dimensional perspective projections of mutually parallel lines in three-dimensional space appear to converge.



Every direction in space has a unique vanishing point. These vanishing points are the intersection of the image plane with the projective space, and they all lie on the horizon line.

CHAPTER

2 IMAGE FILTERING



2.1 Linear Shift-Invariant Filters

2.1.1 Definition

A filter transforms one signal into another. Filters are used to describe image formation (lens burring, sensor noise, etc.), as well as to implement operations on images (edge detection, noise reduction, etc.).

Definition 2.1.1 Linear Transformation

A transformation T is **linear** if and only if it satisfies

$$T[a_1 f_1(x) + a_2 f_2(x)] = a_1 T[f_1(x)] + a_2 T[f_2(x)]$$

for any $a_1, a_2 \in \mathbb{R}$ and continuous functions f_1, f_2.

The Superposition Integral

Every linear transformation between functions f, g can be expressed as an integral of the form

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

where the $h(x,t)$ is the contribution of position t of the input to position x of the output.

In other words, the value of $g(x)$ depends on the entire input $f(t)$, but the contribution of $f(t)$ to $g(x)$ is given by the scaling factor $h(x,t)$. A linear transformation is **completely determined** by the function $h(x,t)$.

Definition 2.1.2 Shift-Invariant Filter

A filter $h(x,t)$ is **shift-invariant** if and only if shifted inputs produce identical but shifted outputs.

Imagine we have the shifted input

$$f'(x') = f(x - x_0)$$

and the shifted output

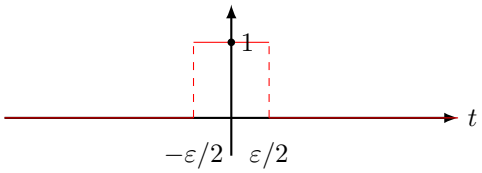
$$g'(x') = g(x - x_0),$$

the **shift invariance property** states that

$$T[f(x - x_0)] = g(x - x_0).$$

2.1.2 The Impulse Function

Imagine we have a function $\text{box}_\varepsilon(t) = \begin{cases} 1 & |t| < \frac{\varepsilon}{2}, \\ 0 & \text{otherwise.} \end{cases}$



Now consider the function

$$\frac{\text{box}_\varepsilon(t)}{\varepsilon}.$$

We see that regardless of the value of ε, the area under the curve is always 1. As $\varepsilon \to 0$, the function becomes more and more concentrated at $t = 0$. This is the **impulse function**,

$$\delta(t) = \lim_{\varepsilon \to 0} \frac{\text{box}_\varepsilon(t)}{\varepsilon}.$$

Definition 2.1.3

The **impulse function** $\delta(x)$ is defined as

$$\delta(x) = \begin{cases} 0 & x \neq 0, \\ \infty & x = 0. \end{cases}$$

The impulse function has the property that

$$\int_{-\infty}^{\infty} \delta(x) f(x)\, dx = f(0) \quad \text{for any integrable function } f(x).$$

The Impulse Response

The **impulse response** of a filter is the output of the filter when the input is the impulse function. The impulse response is a complete description of the filter.

Definition 2.1.4 Impulse Response

The **impulse response** of a filter $h(x,t)$ is the output of the filter when the input is the impulse function,

$$g(x) = \int_{-\infty}^{\infty} h(x,t)\delta(t)\, dt = h(x,0).$$

The filter "response" to the impulse tells us how the filter responds to a single point of input. The response of the filter to any input can be found by integrating the impulse response over the input.

Shift-Invariance

When responding to a shifted impulse, a shift-invariant filter $h(x,t)$ will produce a shifted output.

$$g_2(x) = \int_{-\infty}^{\infty} h(x,t)\delta(t - t_0)\, dt = h(x,t_0).$$

This means that if $g_1(x) = h(x,0)$, then $g_2(x) = h(x,t_0) = g_1(x - t_0)$.

Theorem 2.1.1

A linear filter with impulse response h is shift-invariant if and only if for all, shifts $t_0 \in \mathbb{R}$,

$$h(x, t_0) = h(x - t_0, 0).$$

This tells us that filter's impulse response is actually a 1D function, and the 2D function $h(x, t)$ is just a way of representing the 1D function at different shifts. Thus, we can rewrite the superposition integral

$$\begin{aligned} g(x) &= \int_{-\infty}^{\infty} h(x, t) f(t)\, dt \\ &= \int_{-\infty}^{\infty} h(x - t, 0) f(t)\, dt \end{aligned}$$

The operation fully specifies the result of applying a linear shift invariant filter to a signal

$$g(x) = h * f(x) = \int_{-\infty}^{\infty} h(x - t, 0) f(t)\, dt.$$

This is the **convolution** of the filter h with the input f.

2.1.3 Convolution

Definition 2.1.5 Convolution

The **convolution** of two functions f, g is defined as

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

Here, h is called the **kernel** of the convolution. The convolution of f and h is a new function that describes the output of a filter with impulse response h when the input is f.

Remark

Substituting $x - t$ with u, we have

$$\begin{aligned} (f * g)(x) &= \int_{-\infty}^{\infty} f(x - t) g(t)\, dt \\ &= \int_{-\infty}^{\infty} f(u) g(x - u)\, du. \\ &= (g * f)(x). \end{aligned}$$

In other words, it doesn't matter which function is the input and which is the filter. The convolution is commutative.

- **Commutativity**:
$$f * g = g * f.$$
- **Associativity**:
$$f * (h_1 * h_2) = (f * h_1 * h.$$
- **Distributivity** (over addition):
$$f * (g + h) = f * g + f * h.$$

Convolutions in 2D

Given a continuous input signal $f(x, y)$ and a continuous filter $h(x, y)$, the convolution is defined as

$$g(x, y) = (f * h)(x, y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x - t, y - s)h(t, s)\, dt\, ds.$$

This is a filter that takes in a 2D input and produces a 2D output. The convolution is a linear shift-invariant operation, and it is completely determined by the filter's impulse response.

2.2 Basic Image Filters

2.2.1 Shift Filters and the Impulse Train

- Let $f(t)$ be an image, and δ the filter on the image,
$$g(x) = \int_{-\infty}^{\infty} \delta(x - t)f(t)\, dt = f(x).$$
The impulse filter δ is the identity filter.
- When the impulse filter is shifted, the output is also shifted,
$$g(x) = \int_{-\infty}^{\infty} \delta(x - x_0 - t)f(t)\, dt = f(x - x_0).$$
- Now, consider when the filter is multiple shifted impulses,
$$\delta(x) = \sum_{i=0}^{n} \delta(x - x_i).$$
The output is
$$g(x) = \sum_{i=0}^{n} f(x - x_i).$$
This is the **impulse train** filter.

Definition 2.2.1 Impulse Train

The **impulse train** is an infinite sum of identically shifted impulses,

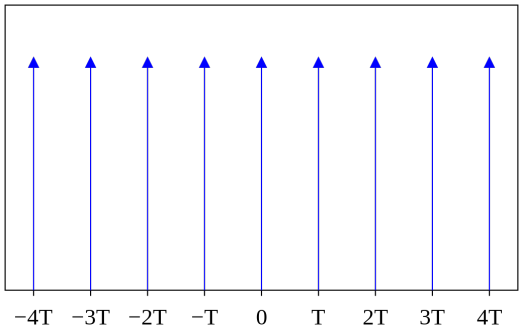
$$h(x) = \sum_{k=-\infty}^{\infty} \delta(x - k\Delta)$$

where Δ is the spacing between impulses (called the **period**), and k is the index of the impulse. The impulse train is also known as the **Shah function**.

Remark

A common notation for 1D impulse trains of period of Δ is

$$\delta_\Delta(x) = \sum_{k=-\infty}^{\infty} \delta(x - k\Delta).$$



2.2.2 Box and Pillow Filters

Recall that the box filter $\frac{1}{\varepsilon}\text{box}_\varepsilon(t)$ is a function that is 1 on the interval $(-\varepsilon/2, \varepsilon/2)$ and 0 elsewhere. The box filter is a simple filter that averages the input over a small interval. We now convolve an image $f(x)$ with the box filter,

$$\begin{aligned} g(x) &= \int_{-\infty}^{\infty} \frac{1}{\varepsilon}\text{box}_\varepsilon(x - t) f(t)\, dt \\ &= \frac{1}{\varepsilon}\int_{x-\varepsilon/2}^{x+\varepsilon/2} f(t)\, dt. \end{aligned}$$

This is the average of the input over the interval $(-\varepsilon/2, \varepsilon/2)$. We expect the output to be a smoothed version of the input.

The **pillbox filter** is a variant of the box filter that focus on a circular region with radius r.

$$h(x,y) = \begin{cases} \frac{1}{\pi r^2} & x^2 + y^2 \leq r^2, \\ 0 & \text{otherwise.} \end{cases}$$

We will get a similar result as the box filter, but the output will be smoothed over a circular region.

> **Remark**
> If we rotate then image, the box filter will produce a different output, but the pillbox filter will produce the same output.

2.2.3 Gaussian Filter

The **Gaussian filter** is a filter that is defined by the Gaussian function

$$G_\sigma(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-x^2/(2\sigma^2)}.$$

and the 2D Gaussian filter

$$\begin{aligned} G_\sigma(x,y) &= G_\sigma(x) G_\sigma(y) \\ &= \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/(2\sigma^2)}. \end{aligned}$$

The Gaussian filter computes the weighted average of the input over a range of values. The weights are determined by the Gaussian function, and the range of values is determined by the standard deviation σ. As we increase σ, the range of values increases, and the weights become more spread out (i.e., the filter becomes more blurred as we are averaging over a larger neighbourhood of pixels).

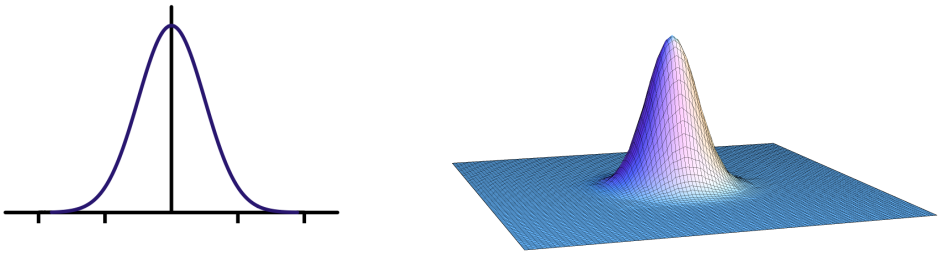


Figure 2.1: Shape of the impulse response of a typical Gaussian filter

The σ is also called the **scaling factor** of the Gaussian filter. The larger the scaling factor, the larger the range of values, and the more blurred the output.

2.2.4 The Gaussian Derivative Filters

First Derivative

The first derivative of the Gaussian filter is a filter that computes the gradient of the input. It is defined as

$$\begin{aligned}\frac{d}{d}x\left(f * G_\sigma\right) &= f * \left(\frac{d}{dx}G_\sigma\right) \\ &= f * \left(-\frac{x}{\sigma^2}G_\sigma\right).\end{aligned}$$

Remark

To derive the first derivative of the Gaussian filter, we first look at the derivative of a function f after convolve with a filter h,

$$\begin{aligned}\frac{d}{dx}(f * h)(x) &= \frac{d}{dx}\int h(x-t)f(t)\,dt \\ &= \int \left[\frac{d}{dx}h(x-t)\right] f(t)\,dt \\ &= f * \frac{d}{dx}h(x).\end{aligned}$$

We see that it is equivalent to f convolving with the derivative of the filter h.
Thus, the first derivative of the Gaussian filter is the derivative of the Gaussian function,

$$\begin{aligned}\frac{d}{dx}G_\sigma(x) &= \frac{d}{dx}\left(\frac{1}{\sigma\sqrt{2\pi}}e^{-x^2/(2\sigma^2)}\right) \\ &= -\frac{x}{\sigma^2}G_\sigma(x).\end{aligned}$$

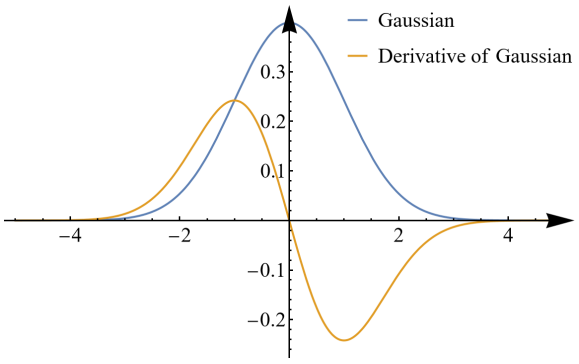


Figure 2.2: 1D Gaussian and its first derivative

The first derivative of the Gaussian filter enhances the edges of the input. The derivative along the x-axis is the gradient detects edges in the x-direction, and the derivative along the y-axis detects

edges in the y-direction. These two derivatives are sufficient to compute the gradient of the input in any direction.

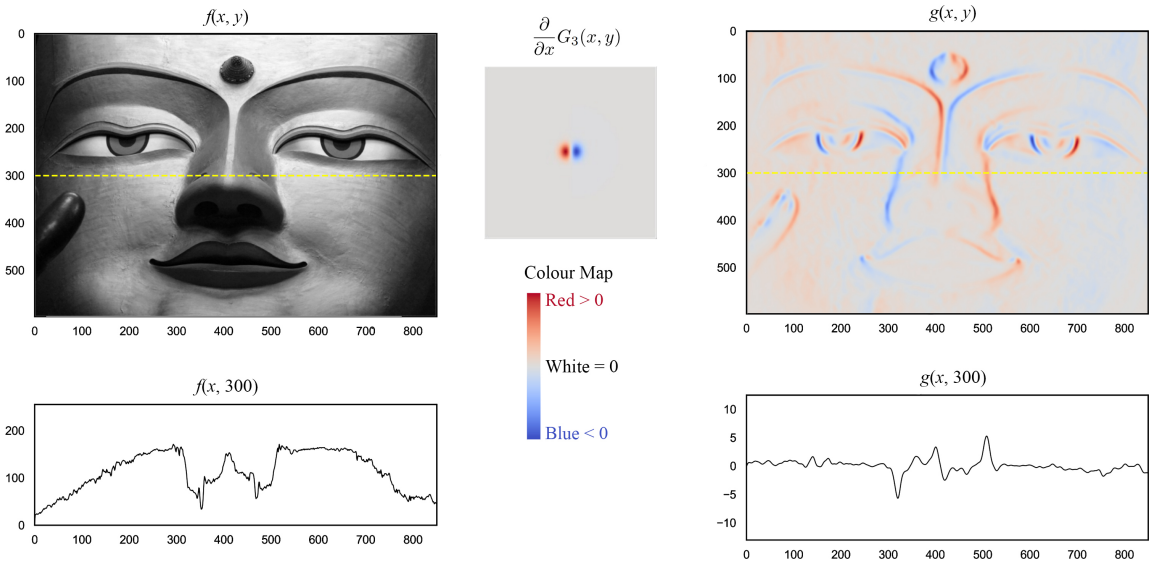


Figure 2.3: Applying the first derivative of the Gaussian filter along the x-axis

Second Derivative

The second derivative of the Gaussian filter is a filter that computes the Laplacian of the input. It is defined as

$$\begin{aligned}\frac{d^2}{d^2}x\,(f * G_\sigma) &= f * \left(\frac{d^2}{dx^2}G_\sigma\right) \\ &= f * \left(\frac{x^2-\sigma^2}{\sigma^4}G_\sigma\right).\end{aligned}$$

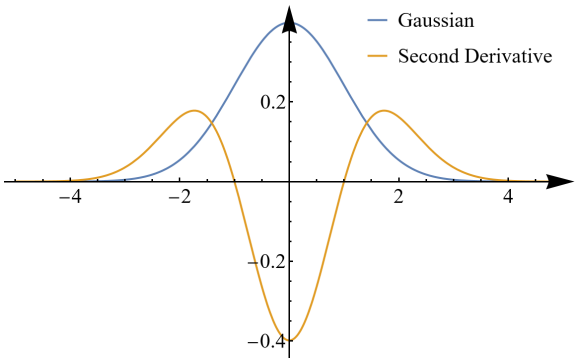


Figure 2.4: 1D Gaussian and its second derivative

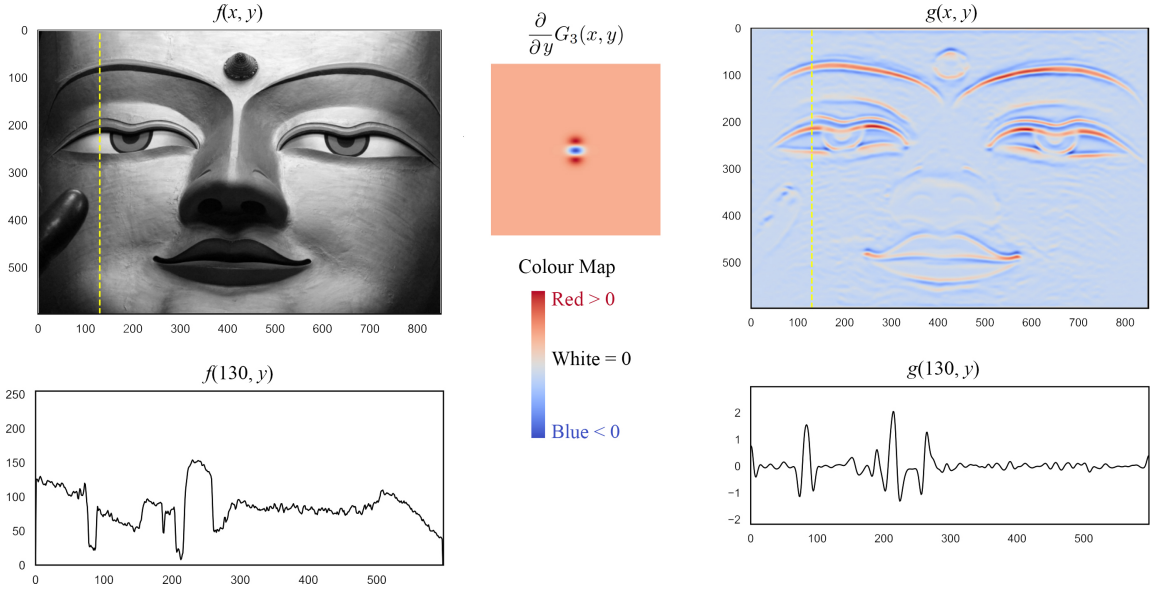


Figure 2.5: Applying the second derivative of the Gaussian filter along the y-axis

2.2.5 The DoG Filter

A **Difference of Gaussians (DoG)** filter is a filter that computes the difference between two Gaussian filters. It is defined as

$$\mathrm{DoG}_{\sigma_1,\sigma_2}(x) = G_{\sigma_1}(x) - G_{\sigma_2}(x).$$

In essence, we are getting information about what is lost when we blur the image with a larger Gaussian filter. Convolving the image with the DoG filter would **enhance the edges** of the input – the area where the difference between the two Gaussian filters is the largest.

The DoG filter is similar to the second derivative of the Gaussian filter, in a sense that it enhances the edges of the input. It is more robust to noise than the second derivative of the Gaussian filter, and it is also more efficient to compute.

2.2.6 Sharpening Filters

Boosting intensity variations in the image "sharpens" the image. Intuitively, this can be done by enhancing the edges of the image. We can convolve the image with the result of the DoG filter to sharpen the image,

$$\begin{aligned} g &= f + s\left(f * \left(G_{\sigma_1} - G_{\sigma_2}\right)\right) \\ &= f + f * \left(sG_{\sigma_1} - sG_{\sigma_2}\right) \\ &= f * \delta + f * \left(sG_{\sigma_1} - sG_{\sigma_2}\right) \\ &= f * \left(\delta + sG_{\sigma_1} - sG_{\sigma_2}\right). \end{aligned}$$

However, this method is not ideal. Since we are brightening the bright areas and darkening the dark areas, we can create halos around the edges of the image.

Figure 2.6: Halos around the edges of the image

2.2.7 Practical Considerations

Out-Of-Bounds Handling

Edge-Degrading Behaviour

The Bilateral Filter

Bilateral filtering reduces noise while preserving edges. It is a non-linear, edge-preserving, and noise-reducing smoothing filter for images. It replaces the intensity of each pixel with a weighted average of the intensities of nearby pixels. The weights are based on both the spatial distance and the intensity difference between the pixels,

$$g(x,y) = \int\int G_{\sigma_s}\left(\|(x-y)-(u-v)\right) G_{\sigma_r}\left(|f(x,y)-f(u,v)|\right) f(u,v)\,du\,dv.$$

2.3 Digital Imaging Basics

2.3.1 Convolution and Sampling

The camera sensors are arrays of microlenses, filters, and photodiodes.

- Microlenses concentrate all light falling onto a pixel's footprint to the pixel's photosensitive region.

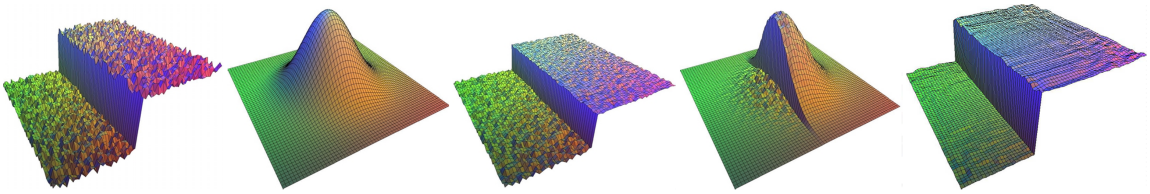


Figure 2.7: Bilateral filtering.Colors are used only to convey shape, [0].

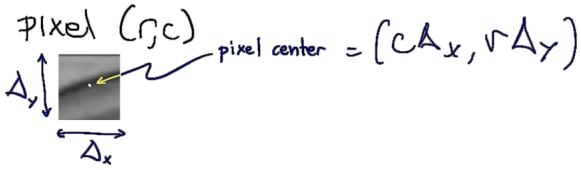
- Colour filters are for capturing colour information
- Each pixel outputs **one** intensity value
- These are processed to create full colour photos.

The light collected over a pixel's footprint produces just one intensity value. It can be thought of as the integral of the light intensity over the pixel's footprint (or, the average light intensity over the pixel's footprint).

> **Remark**
>
> It is important to remember that a pixel is **not a square** of constant intensity of the image. It is a *single measurement of the light intensity over a region of the image*. We can treat it as an impulse response whose height is the average intensity of the light over that region.
>
> The original continuous image was not constant in this footprint, and mathematically, the grid of pixels is most accurately represented as a grid of point samples.

The Pixel Filter

Consider a continuous image $f(x, y)$ formed on camera's sensor plane, and a discrete pixel grid that defines the locations and footprints of the pixels. Using the row and column coordinate, we look at pixel at location (r, c).



Ignoring noise, the value of pixel (r, c) will be the **average** of $f(x, y)$ over the pixel's footprint,

$$f_{r,c} = \frac{1}{\Delta_x \cdot \Delta_y} \int_{c\Delta_x - \frac{\Delta_x}{2}}^{c\Delta_x + \frac{\Delta_x}{2}} \int_{r\Delta_y - \frac{\Delta_y}{2}}^{r\Delta_y + \frac{\Delta_y}{2}} f(x, y)\, dx\, dy.$$

The averaging in the integral can be expressed a a convolution with a 2D box filter,

$$f * \left(\frac{1}{\Delta_x \Delta_y} \cdot \text{box}_{\Delta_x}(x) \cdot \text{box}_{\Delta_y}(y) \right),$$

which gives a blurred version of the original image. However, this **continuous** blurred image is **not** f_{rc} becaus ethe sensor measures the average only at the pixel centres.

We take the 2D impulse train

$$\delta_{\Delta_x}(x) \cdot \delta_{\Delta_y}(y)$$

of this blurry image gives us a function who is non-zero only at the pixel centres. This is the **sampling process**.

The full expression of a digital image is

$$\tilde{f}(x,y) = \left\{ \underbrace{f(x,y)}_{\text{Continuous image}} * \left[\frac{1}{\Delta_x \Delta_y} \cdot \underbrace{\text{box}_{\Delta_x}(x) \cdot \text{box}_{\Delta_y}(y)}_{\substack{\text{Averaging within} \\ \text{pixel footprint}}} \right] \right\} \cdot \underbrace{\delta_{\Delta_x}(x) \cdot \delta_{\Delta_y}(y)}_{\substack{\text{Sampling according to the} \\ \text{pixel grid, with period} \\ \Delta_x \text{ in } x \text{ and } \Delta_y \text{ in } y}}$$

Remark

Note that $\tilde{f}$ is expressed as a function over a continuous domain, but it is non-zero only at discrete set of points.

2.3.2 Resampling

Given a sampled image, we may want to display it in many ways. For example, we may want to display it on a monitor with a higher resolution than the camera sensor, or we may want to display it on a monitor with a different aspect ratio.

Definition 2.3.1 Super-Sampling

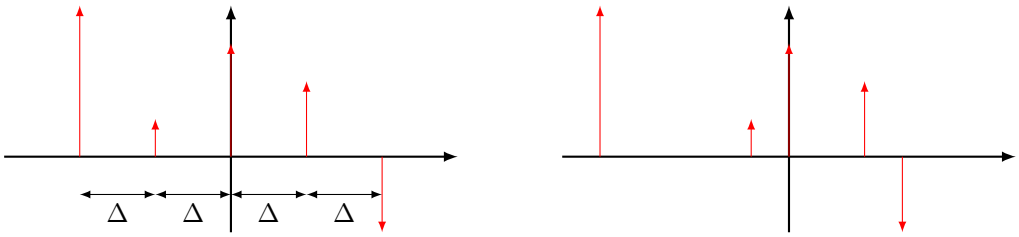
Super-sampling is the process of creating a higher-resolution image from a lower-resolution image.

Definition 2.3.2 Sub-Sampling

Sub-sampling is the process of creating a lower-resolution image from a higher-resolution image.

The resampling process is very simple, given a discrete collection of samples $\tilde{f}(x,y)$,

1. Convert $\tilde{f}(x,y)$ to a continuous function $\tilde{f}_{\text{int}}(x,y)$ via interpolation.
2. (Optional) Apply a filter to $\tilde{f}_{\text{int}}(x,y)$ to avoid aliasing.
3. Sample $\tilde{f}_{\text{int}}(x,y)$ on the new pixel grid to get $\tilde{f}_{\text{resampled}}(x,y)$.



(a) Uniform Interpolation

(b) Non-Uniform Interpolation

2.3.3 Interpolation

Definition 2.3.3 Function Interpolation

Given a discrete set of samples $f_k = f(x_k)$ of a function at $x_1, x_2, \ldots, x_k, \ldots$, we construct a new function $g(x)$ that satisfies $g(x_k) = f(x_k)$ and can be evaluated for any $x \in \mathbb{R}$.

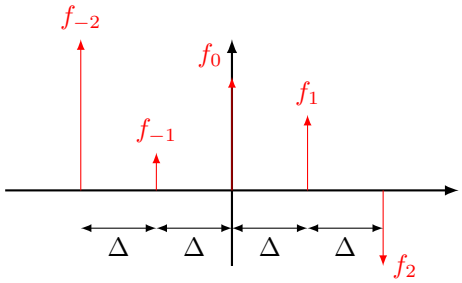
In **uniform interpolation**< the sample locations x_k are defined by an impulse train. In **non-uniform interpolation**, the sample locations are **not** on a grid.

There are some important considerations when interpolating a function, but **shift-invariance** is the key point. If we shift the samples $\tilde{f}$ and apply the same interpolation, we should get a shifted version of the result g.

Mathematically speaking, shifting the function would be equivalent to convolving with the impulse,

$$\tilde{f}(x, y) * \delta(x - \Delta_x, y - \Delta_y) = g(x, y) * \delta(x - \Delta_x, y - \Delta_y).$$

Consider a uniformly-sampled f with period Δ.



We observe that any interpolation methods that can be expressed as a linear shift-invariant transformation of the input samples must satisfy

$$g(x) = \sum_{k=-\infty}^{\infty} f_k h(x - k\Delta),$$

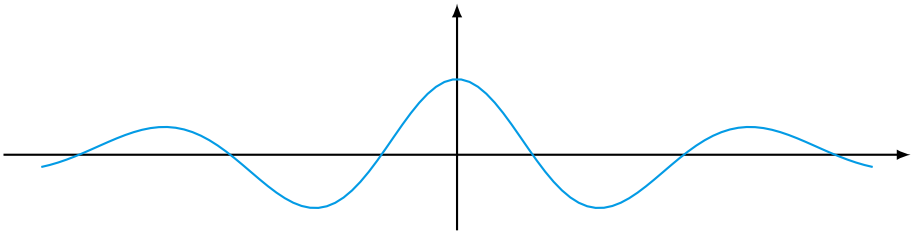
where $h(x)$ is the interpolation filter (i.e. the interpolation kernel).

Remark

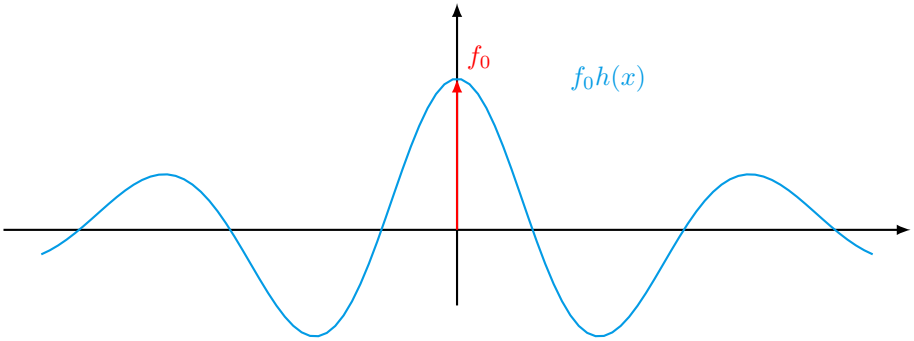
$h(x - k\Delta)$ is $h(x)$ shifted by $k\Delta$. What we are saying here is that the final function g will be a weighted sum of the shifted h functions.

In this course, we only consider *uniform sampling* and *LSI interpolations*.

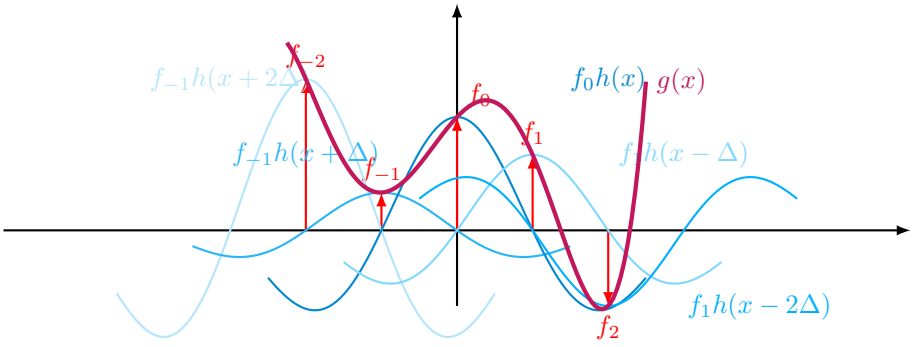
Example. Consider the interpolation filter h



The g function will be composed of many shifted copies of h.



$$g(x) = f_0 h(x) + \dots$$



$$g(x) = \dots + f_{-1}h(x+2\Delta) + f_{-1}h(x+\Delta) + f_0 h(x) + f_1 h(x-\Delta) + f_1 h(x-2\Delta) + \dots$$

♢

- Let $\tilde{f}$ be the function that defines the k samples,

$$\tilde{f} = \sum_{k=-\infty}^{\infty} f_k \delta(x - k\Delta).$$

  f is non-zero only at the sample locations, and it is zero elsewhere.

- The interpolation process is LSI, so g can be rxpressed as a convolution with some filter h,

$$g = \tilde{f} * h = \left[\sum_{k=-\infty}^{\infty} f_k \delta(x - k\Delta) \right] * h.$$

- Thus, we can express g as

$$\begin{aligned} g &= \sum_{k=-\infty}^{\infty} f_k \left(\delta(x - k\Delta) * h \right) \\ &= \sum_{k=-\infty}^{\infty} f_k h(x - k\Delta). \end{aligned}$$

The interpolation filter is a **continuous function** over $x \in \mathbb{R}a$, but it **cannot be arbitrary**. To ensure $g(x)$ passes through all f_k, we need to ensure that $h(x)$ satisfies

- $h(0) = 1$ (interpolating the sample at $x = 0$)
- $h(k\Delta) = 0$ for all $k \neq 0$ (interpolating the sample at $x = k\Delta$)

To ensure that $g(x)$ is smooth, we need

- $h(x)$ is smooth everywhere (i.e. continuous derivatives of all orders)

Moreover, by having only few samples contributing to $g(x)$, efficiency can be ensured via local suppost:

- $h(x) = 0$ outside a small neighbourhood of $x = 0$

2.3.4 Common Interpolation Filters

Nearest-Neighbour Interpolation

Linear Interpolation

$$g(x) = \left(1 - \frac{x}{\Delta}\right) f_0 + \frac{x}{\Delta} f_1 \qquad \text{for } 0 \leq x < \Delta.$$

$$h(x) = \begin{cases} 1 - \frac{x}{\Delta} & 0 \leq |x| < \Delta, \\ \frac{x}{\Delta} & \text{otherwise.} \end{cases}$$

It is easy to show that

$$h(x) = \text{box}_\Delta(x) * \text{box}_\Delta(x) \cdot \frac{1}{\Delta}.$$

Bilinear Interpolation

$$g(x,y) = \tilde{f}(x,y) * [h(x) \cdot h(y)],$$

, where $h(x)$ and $h(y)$ are the 1D linear interpolation kernels.

Cubic Interpolation

Form the stated desiderata it is possible to also derive a 3rd degree polynomial interpolation filter,

$$h(x) = \begin{cases} \frac{3}{2}|s|^3 - \frac{5}{2}|s|^2 + 1 & 0 \leq |s| < 1, \\ -\frac{1}{2}|s|^3 + \frac{5}{2}|s|^2 - 4|s| + 2 & 1 \leq |s| < 2, \\ 0 & \text{otherwise.} \end{cases} \qquad \text{where } s = \frac{x}{\Delta}.$$

Remark **Why there are no second degree interpolation filters**

Recall that we require the property that

- They have value 1 at 0, and
- They go to 0 at $\pm\Delta$ (i.e. the previous / next sample location).

It is impossible for a second degree polynomial to satisfy both of these properties.

2.4 Fourier Transform

2.4.1 The Fourier Transform

Any periodic signal can be expressed as a sum of sines and cosines. The Fourier transform is a generalization of this idea to non-periodic signals.

In the **fourier domain**, a point in space no longer represents a value of the signal, but a *frequency* of the signal.

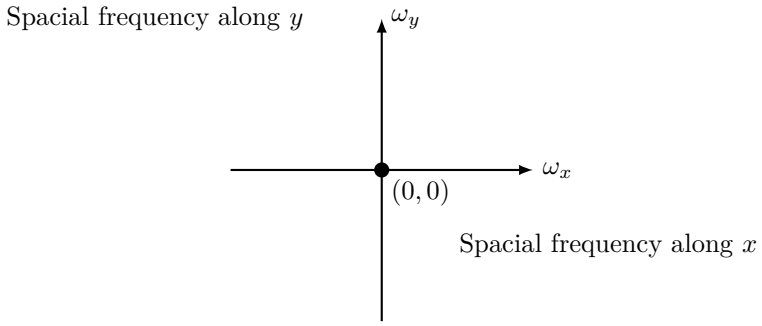


Figure 2.9: The Fourier domain of a constant-intensity image (the "DC component")

Example. The points in the Fourier domain represent the frequencies of the signal.

- The point $(5, 0)$ represents an image that varies sinusoidally along x with a frequency of 5 cycles per unit length.
- The point $(0, 5)$ represents an image that varies sinusoidally along y with a frequency of 5 cycles per unit length.

The larger the magnitude of the point, the higher the frequency of the sinusoid.

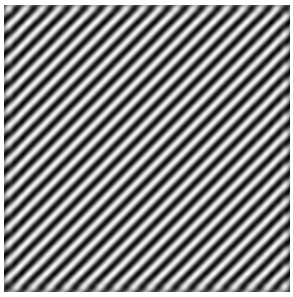


Figure 2.10: The image in the spatial domain corresponds to point $(17, 17)$ in the Fourier domain.

♢

Merely specifying the frequencies (ω_x, ω_y) is **not** sufficient to uniquely determine the image. We need the sinusoid to have a *phase* as well. The general expression for the intensity of pixel (x, y) represented by $(\omega_x, 0)\&(-\omega_x, 0)$ is given by

$$I(x, y) = \cos\left(\omega_x \frac{2\pi}{C} x + \varphi\right),$$

and similarly, the general expression for the intensity of pixel (x, y) represented by $(0, \omega_y)\&(0, -\omega_y)$ is given by

$$I(x, y) = \cos\left(\omega_y \frac{2\pi}{R} y + \varphi\right).$$

Thus, the general expression for image represented by (ω_x, ω_y) is given by

$$I(x, y) = \cos\left[\left(\omega_x \frac{2\pi}{C} x\right) + \omega_y \left(\frac{2\pi}{R} y + \varphi\right)\right].$$

To properly account for phase, (ω_x, ω_y) actualy represents an image whose intensities are *complex-valued*. The image will have a real component and an imaginary component, and the phase of the sinusoid is given by the angle of the complex number,

$$I(x, y) = \cos(2\pi \frac{x}{C} \omega + \varphi) + j \sin(2\pi \frac{x}{C} \omega + \varphi).$$

CHAPTER 3

MODEL FITTING

CHAPTER

COLOR IMAGING AND DISPLAYING 4

Part II

Image Representation for CV

CHAPTER

5 CONTINUOUS

CHAPTER

6 VECTOR-BASED

CHAPTER 7

MULTI-SCALE

Part III

Appendices

BIBLIOGRAPHY

[DD02] Frédo Durand and Julie Dorsey. "Fast Bilateral Filtering for the Display of High-Dynamic-Range Images". In: *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*. ACM SIGGRAPH. 2002. URL: https://people.csail.mit.edu/fredo/PUBLI/Siggraph2002/ (cited on page 30).

[Gla95] Andrew S. Glassner. *Principles of Digital Image Synthesis*. Morgan Kaufmann Publishers, Inc., 1995. ISBN: 1-55860-276-3. URL: https://www.realtimerendering.com/Principles_of_Digital_Image_Synthesis_v1.0.1.pdf.

[Sze22] Richard Szeliski. *Computer Vision: Algorithms and Applications*. 2nd edition. Springer, 2022. URL: https://szeliski.org/Book/.