Images as Functions
Foundations of Computer Vision

Jason Corso
University of Michigan
September 8, 2017

Images are the primary data source that we focus on in this course. However, we do not restrict ourselves to images of the natural world, i.e., photographs, or single images (rather than sequences of images). We develop an abstraction of images and develop ideas for representing them under this abstraction.

Examples of images that we will discuss either directly or in examples include, indeed, photographs, as well as depth images, radar/lidar images, color-and-depth images, video streams, medical images, and cartoon-drawn images. Additionally, we sometimes will consider 3D points in Euclidean space during discussions on the applications involving geometry and reconstruction.

There are three primary topics in the representation of images that we will focus on:

1. Images as Functions;
2. Images as Coefficients / Images as Points;
3. Images as Graphs.

These three representations of images afford transformations between each other that retain all information. Instead, we primarily focus on these three types in terms of the distinct processing that each affords.

1 Defining The Classes of Images

Let us begin with the most natural representation of images: functions. Consider the physical process of an image being captured by a sensor. Energy passes through the environment—visible light, x-rays, etc.—undergoes various transformations such as refraction by a lens and finally impacts a sensor.

1.1 The Perfect Image

Definition 1. A *perfect image* is the continuous image generated by a physical process. We denote them with $\mathcal{I}$. They are represented as maps from points on the plane to the real numbers, $\mathcal{I}\colon \mathbb{R}^2 \mapsto \mathbb{R}$.

Remark 1.1. $0 < \mathcal{I}(\cdot) < \infty$ since the range of the function is proportional to the energy radiated by a physical source.

Remark 1.2. In our abstraction, we can alternatively define their domain as points on a hyperplane of some dimension greater than two, for increased generality. Likewise, we can define them as maps to some alternative space, depending on the underlying physical process.

Remark 1.3. Perfect images exist in abstraction only; we can not sample them.

Example 1 — Lambertian Images A classical model for reflectance of a surface is that the amount of light reflected off the surface is varies with the cosine of the angle of incident light. This type of surface is called a Lambertian surface; this model of reflectance captures diffuse characteristics of the surface but not ambient lighting, specular reflections, or other more complex radiometric functions such as sub-surface scatter.

Consider a point $\mathbf{x} \in \mathbb{R}^3$ on the Lambertian surface with a corresponding surface normal $\mathbf{n}(\mathbf{x})$. The reflectance at $\mathbf{x}$ for an incident light source with direction $\boldsymbol{\ell}(\mathbf{x})$ is written

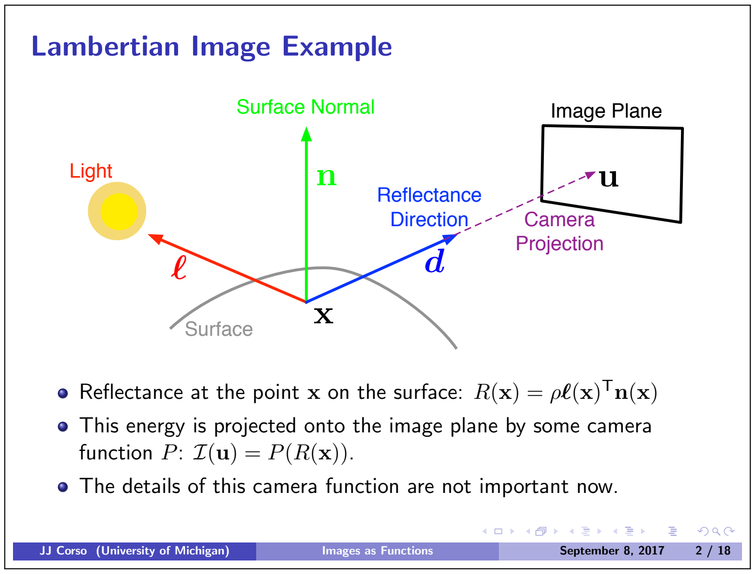
$$R(\mathbf{x}) = \rho \boldsymbol{\ell}(\mathbf{x})^{\mathsf{T}} \mathbf{n}(\mathbf{x}) \quad , \tag{1}$$

where ρ is a scalar constant that describes the surface material properties.

Any reflected light intersecting our image plane, by way of a camera projection function P, would induce our *perfect* image:

$$\mathcal{I}(\mathbf{u}) = P(R(\mathbf{x})) \quad . \tag{2}$$

The details of this camera projection are not relevant to our current discussion.



Lambertian Image Example

- Reflectance at the point $\mathbf{x}$ on the surface: $R(\mathbf{x}) = \rho \boldsymbol{\ell}(\mathbf{x})^{\mathsf{T}} \mathbf{n}(\mathbf{x})$
- This energy is projected onto the image plane by some camera function P: $\mathcal{I}(\mathbf{u}) = P(R(\mathbf{x}))$.
- The details of this camera function are not important now.

1.2 The Digital Image

We unfortunately cannot acquire perfect images directly. Instead, we have access to electronics that will transduce the perfect images into a digital representation. In doing so, it will sample and quantize these perfect images to yield *digital images*.

Definition 2. A *digital image* is a sampled and quantized perfect image. They are represented as maps from points at non-negative natural pixels (picture elements, induced by the sampling hardware) to the naturals, $\mathbf{I}\colon \mathbb{N}_0^2 \mapsto \mathbb{N}_0$.

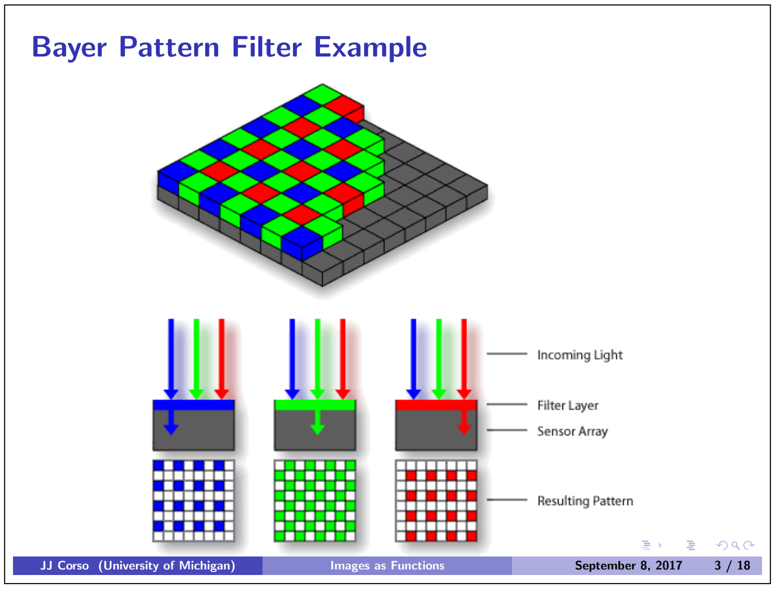
Remark 2.1. Digitization is the projection $(\mathbb{R}^2 \mapsto \mathbb{R}) \longrightarrow (\mathbb{N}_0^2 \mapsto \mathbb{N}_0)$.

Remark 2.2. "Sampling" refers to the digitization of the coordinate values whereas "quantization" refers to the digitization of the intensity values. Sampling is generally uniform, but quantization can vary over the range based on properties of the sensor such as gain, white balance, etc.

Remark 2.3. Sampling density is constrained by the physical limitations of the sensor.

Remark 2.4. The quantization depth depends on the particular hardware in use. 8-bit quantization (values ranging from 0 through 255) are most common for digitized natural images, like photographs.

Remark 2.5. In certain digitization scenarios the perfect image is manipulated to tease out signals of interest. The most common example of this is the Bayer pattern that is overlaid atop the photosensitive circuity. In some cases, the output dimensionality may be higher.



Bayer Pattern Filter Example

JJ Corso (University of Michigan) Images as Functions September 8, 2017 3 / 18

1.3 Modeling the Digital Image

We define two mathematical interpretations on top of the digital image with which we can work. First, we consider the digital image as a discrete function from pixels to values. Second, we generalize domain and range of this function to once again consider a continuous image, albeit an approximate one to our original perfect image.

Definition 3. A *discrete image* is the natural mathematical generalization of the digital image. It captures integer pixel locations and maps them to integer pixel values, $\mathbf{I}\colon \mathbb{Z}^2 \mapsto \mathbb{Z}$.

Remark 3.1. The stated function is the simplest we will consider. It, however, encompasses higher dimensional domains, such as in video $\mathbb{Z}^3$, and higher dimensional ranges, such as in color $\mathbb{Z}^3$ or even in quantized feature images $\mathbb{Z}^k$.

Remark 3.2. We will generally denote pixel coordinates with scalar tuples (s, t) or vectors s in discrete representations.

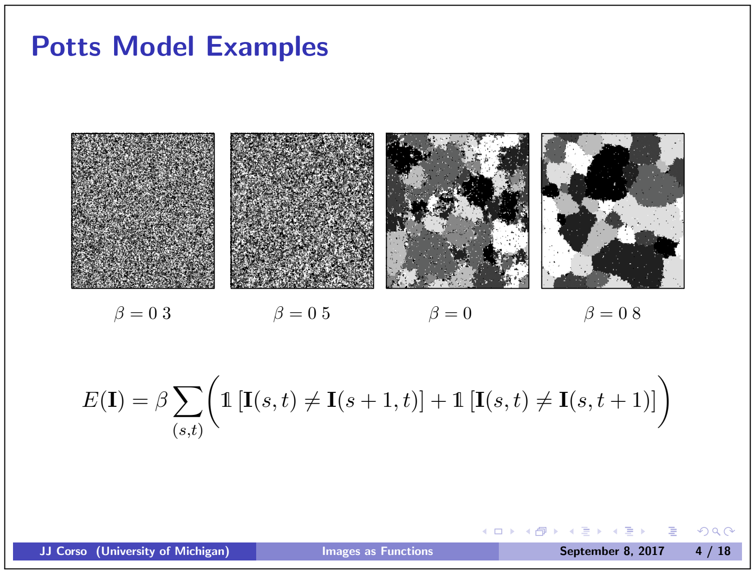
Remark 3.3. We denote the subset of the integers corresponding to the domain of an images as Λ.

Example 2 — Potts Model: Piecewise Constant Images A straightforward classical example of a discrete image model is the Potts Model. Originally derived in statistical physics as a generalization of the two-state Ising model, the Potts model assumes a piecewise constant image signal. We write it as the energy functional, for an image of size $n \times n$

$$E(\mathbf{I}) = \beta \sum_{s=1}^{n-1} \sum_{t=1}^{n} \Big(\mathbb{1}\left[\mathbf{I}(s,t) \neq \mathbf{I}(s+1,t)\right] \Big) + \beta \Big(\sum_{s=1}^{n} \sum_{t=1}^{n-1} \mathbb{1}\left[\mathbf{I}(s,t) \neq \mathbf{I}(s,t+1)\right] \Big)$$

where β is a modeling constant originally related to the physical properties of the material in study, and we have neglected boundary conditions on the domain. $\mathbb{1}$ is the indicator function which takes the value 1 if its argument is true and 0 otherwise.

Inspecting the Potts model definition, we find that the model has energy in proportion to the amount of changes in both horizontal and vertical adjacent pixels in the image. When the image $\mathbf{I}$ has large constant regions, it will have commensurately low energy.



Potts Model Examples

$\beta = 0\ 3$ $\beta = 0\ 5$ $\beta = 0$ $\beta = 0\ 8$

$$E(\mathbf{I}) = \beta \sum_{(s,t)} \Big(\mathbb{1}\left[\mathbf{I}(s,t) \neq \mathbf{I}(s+1,t)\right] + \mathbb{1}\left[\mathbf{I}(s,t) \neq \mathbf{I}(s,t+1)\right] \Big)$$

JJ Corso (University of Michigan) Images as Functions September 8, 2017 4 / 18

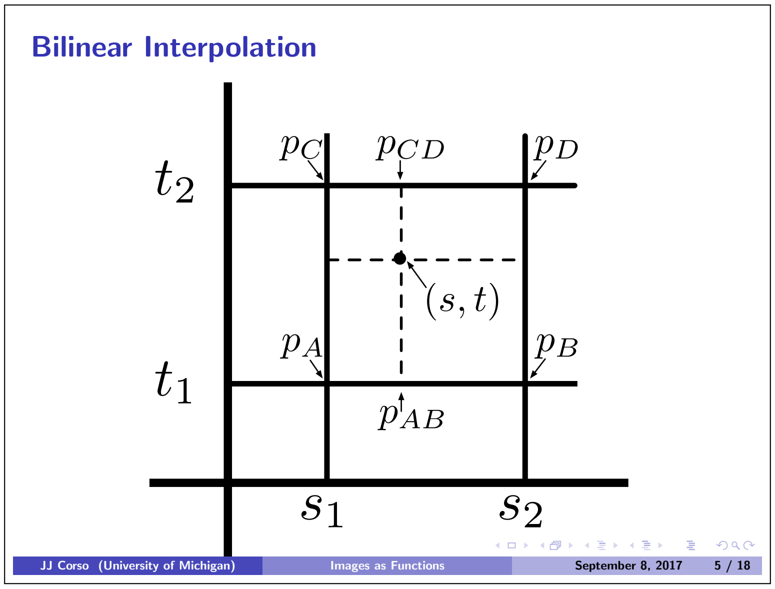
Question 1. Although we can realize many possible discrete images for a certain domain and range, the Potts model organizes these images into a set of equivalence classes. What are these equivalence classes?

Definition 4. A *continuous image* further generalizes the discrete image to a continuous domain and range. Continuous images bring our discussion full-circle back to perfect images, but they are at best approximations thereof. We generally consider them maps $I\colon \mathbb{R}^2 \mapsto \mathbb{R}$.

Remark 4.1. Similar generalizations of domain and range as those discussed in the discrete image—e.g., video—are plausible for continuous images.

Remark 4.2. Hybrid continuous-discrete representations are also common, such as a discrete domain but continuous range.

Remark 4.3. Interpolation Through the digitization process, the perfect (continuous) initial images have been sampled at integer locations. Yet, for various analyses on these continuous images, we will inevitably need to sample at non-integer coordinates, i.e., between two pixels. This sampling requires interpolation.



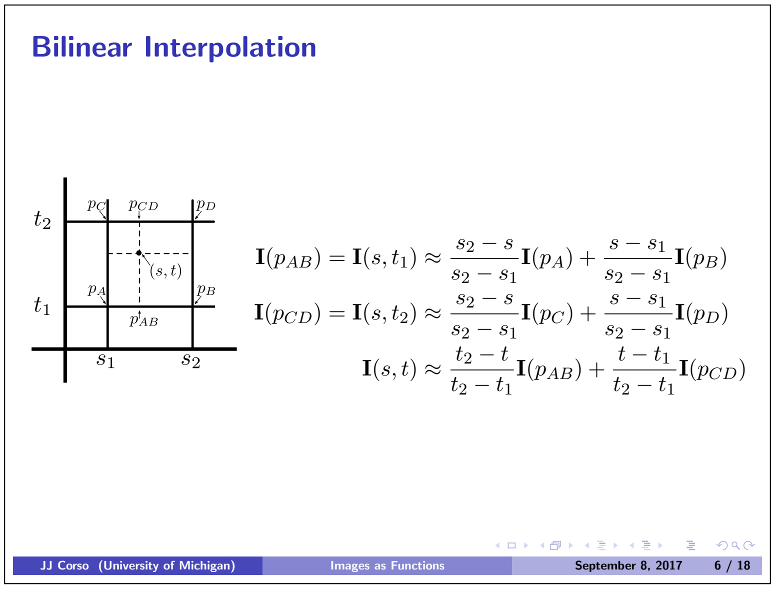
Bilinear Interpolation

JJ Corso (University of Michigan) Images as Functions September 8, 2017 5 / 18

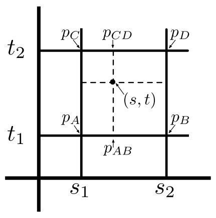
$$\mathbf{I}(p_{AB}) = \mathbf{I}(s, t_1) \approx \frac{s_2 - s}{s_2 - s_1}\mathbf{I}(p_A) + \frac{s - s_1}{s_2 - s_1}\mathbf{I}(p_B) \tag{3}$$

$$\mathbf{I}(p_{CD}) = \mathbf{I}(s, t_2) \approx \frac{s_2 - s}{s_2 - s_1}\mathbf{I}(p_C) + \frac{s - s_1}{s_2 - s_1}\mathbf{I}(p_D) \tag{4}$$

$$\mathbf{I}(s, t) \approx \frac{t_2 - t}{t_2 - t_1}\mathbf{I}(p_{AB}) + \frac{t - t_1}{t_2 - t_1}\mathbf{I}(p_{CD}) \tag{5}$$



Bilinear Interpolation



$$\mathbf{I}(p_{AB}) = \mathbf{I}(s,t_1) \approx \frac{s_2 - s}{s_2 - s_1}\mathbf{I}(p_A) + \frac{s - s_1}{s_2 - s_1}\mathbf{I}(p_B)$$
$$\mathbf{I}(p_{CD}) = \mathbf{I}(s,t_2) \approx \frac{s_2 - s}{s_2 - s_1}\mathbf{I}(p_C) + \frac{s - s_1}{s_2 - s_1}\mathbf{I}(p_D)$$
$$\mathbf{I}(s,t) \approx \frac{t_2 - t}{t_2 - t_1}\mathbf{I}(p_{AB}) + \frac{t - t_1}{t_2 - t_1}\mathbf{I}(p_{CD})$$

JJ Corso (University of Michigan) Images as Functions September 8, 2017 6 / 18

1.4 Review

Our discussion developed an abstraction of images as functions beginning with the physical process underlying the acquisition of various types of images. These perfect images, however, evade acquisition, which has significant limitations from sampling and quantization during transduction into digital images.

We then described a pair of mathematical interpretations of these digital images: both discrete and continuous with the latter of the two returning to a similar mindset as the perfect image abstraction. Depending on the specific problem or application, either of these two interpretations, or even a hybrid between them, can be used.

2 Operations on Images

With our functional interpretation of images, we can bring to bear the full-range of mathematical aspects of functions both in terms of practice and theory. For example, we can consider arithmetic operations on images; we can embed images in a composition of functions; we can study properties of certain models on images; and so on. In this section, we will enumerate the three core types of operations on images as functions:

1. Spatial range operations such as computing the summation of all intensity values over an image;
2. Range map operations such as computing the difference of two images;
3. Domain operations or geometrical transformations like translation and rotation.

We will discuss these three types of operations and provide various examples of them in the subsequent sections.

2.1 Spatial Range Operations

Spatial range operations integrate information over a region in the image (or the whole image). Define a region in the image as $W \subseteq \Lambda \subseteq \mathbb{Z}^2$; we use W since such regions are often called *windows* into the image.

Definition 5. A *spatial range operation* f is a function mapping the image to a real value, $f\colon W \times (\mathbb{Z}^2 \mapsto \mathbb{Z}) \mapsto \mathbb{R}$, or more simply, $f\colon W \times \mathbf{I} \mapsto \mathbb{R}$.

Remark 5.1. We use a shorthand $\mathbf{I}[W]$ to denote the new image whose domain Λ is induced by W. This new image is often called a sub-image, or an image *patch*. In these notes, we refer to it as a patch. These are the pixel values whereas the window is the pixel locations in the original image $\mathbf{I}$.

Remark 5.2. How we represent W is a practical nuance. We can consider W is a function that maps pixels to $\mathbb{B} = \{0, 1\}$, as in $W\colon \Lambda \mapsto \mathbb{B}^\Lambda$ to act as a mask. Then we compose the image with the mask before applying the spatial range operation f. Alternatively, we can implement specific domain indexing inside of the function f based on the window W. These interpretations are mutually interesting and useful; we make no distinction in the current discussion and will make the choice explicit when necessary.

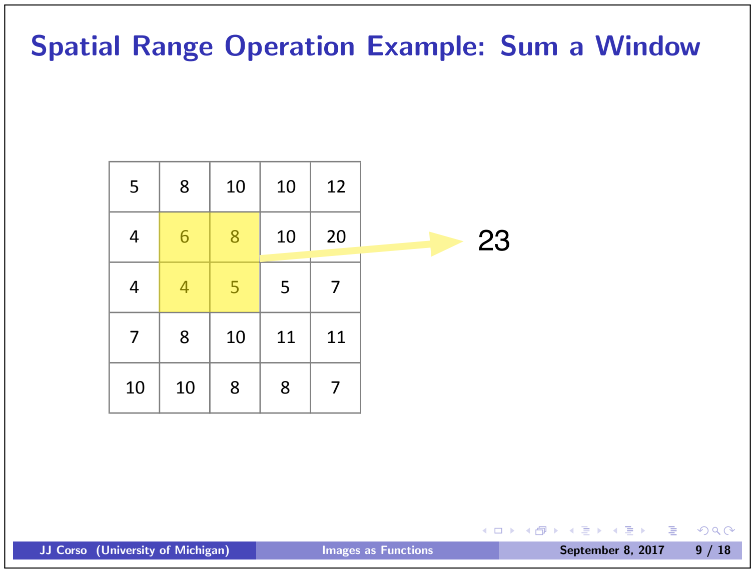
Remark 5.3. A spatial range operation is said to be *linear* if

$$f_W\left(\alpha_i \mathbf{I}_i + \alpha_j \mathbf{I}_j\right) = \alpha_i f_W\left(\mathbf{I}_i\right) + \alpha_j f_W\left(\mathbf{I}_j\right) \tag{6}$$

where α_i and α_j are arbitrary scalar constants.

Example 3 — Summing pixels in a window One simple linear spatial range operation is the summation of all pixels in the region W:

$$\text{sum}(\mathbf{I}, W) = \sum_{(s,t)\in W} \mathbf{I}(s,t) \ . \tag{7}$$



Spatial Range Operation Example: Sum a Window

5	8	10	10	12
4	6	8	10	20
4	4	5	5	7
7	8	10	11	11
10	10	8	8	7

23

On the other hand, the max and min operators, which outputs the maximum and minimum, respectively, values are not linear spatial range operators.

Question 2. Can you construct a simple example of a pair of images that show the maximum operator is not linear?

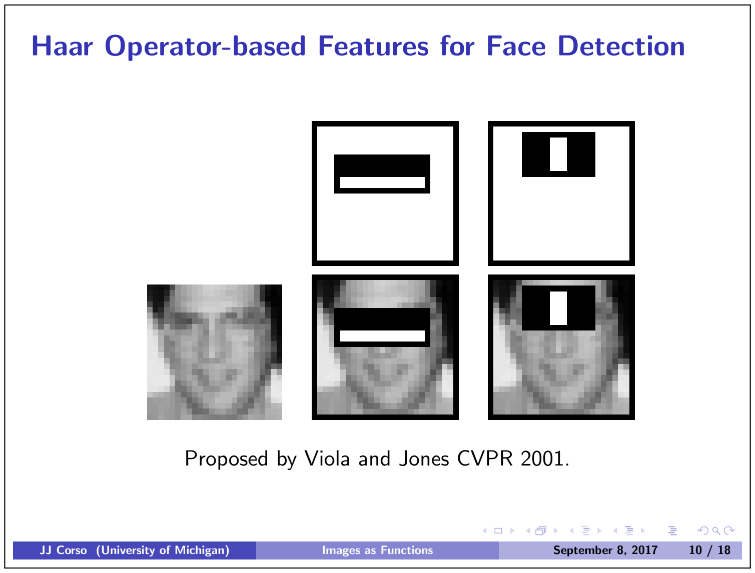
Remark 5.4. Spatial range operations can be combined to form increasingly complex operators on image regions.

Remark 5.5. Certain spatial range operators will require parameters for processing. These are generally denoted by lowercase Greek letters after a semi-colon, e.g., $f(W, \mathbf{I}; \theta)$, bolded for vectors or matrices, e.g., $\boldsymbol{\theta}$.

Example 4 — Haar Operator We can further make use of the operations resulting from even simple spatial range operations like sum. For example, region boundaries can be characterized by the difference of the summed pixel values within those regions. This is called the Haar operator due to its similarity to the computation performed in the Haar wavelet decomposition of images, which we will discuss later in the semester (Images as Points).

Here, we take a combination of two or more separate summed regions $\{W_1, \ldots, W_n\}$ where parameters (weights) $\{\alpha_1, \ldots, \alpha_n\}$ are from $\{-1, 1\}$. The combination is

$$\mathrm{haar}(\mathbf{I}, \{W\}; \{\alpha\}) = \sum_i \alpha_i \,\mathrm{sum}(\mathbf{I}, W_i) \tag{8}$$

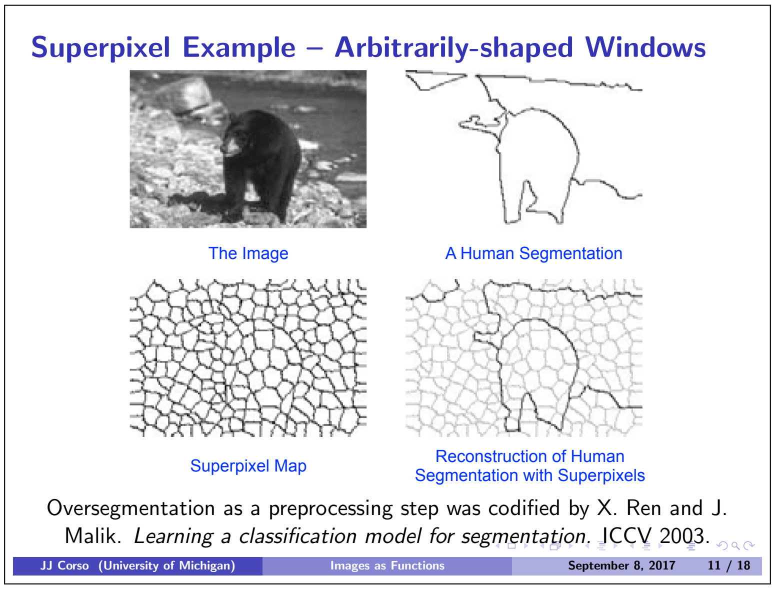


Haar Operator-based Features for Face Detection

Proposed by Viola and Jones CVPR 2001.

JJ Corso (University of Michigan) Images as Functions September 8, 2017 10 / 18

Remark 5.6. Spatial range windows are subsets of the lattice in a general sense. A single pixel is such a window, albeit a degenerate one. They do not need to be rectilinear subsets. They do not need to be connected subsets. One common example of non-rectilinear subsets are *superpixels*, or compact, connected sets of roughly homogeneous pixels.



Superpixel Example – Arbitrarily-shaped Windows

The Image
A Human Segmentation
Superpixel Map
Reconstruction of Human Segmentation with Superpixels

Oversegmentation as a preprocessing step was codified by X. Ren and J. Malik. *Learning a classification model for segmentation.* ICCV 2003.

JJ Corso (University of Michigan) Images as Functions September 8, 2017 11 / 18

Remark 5.7. Spatial range operations are applicable to both discrete and continuous image representations with the obvious care taken to define appropriate windows or intervals.

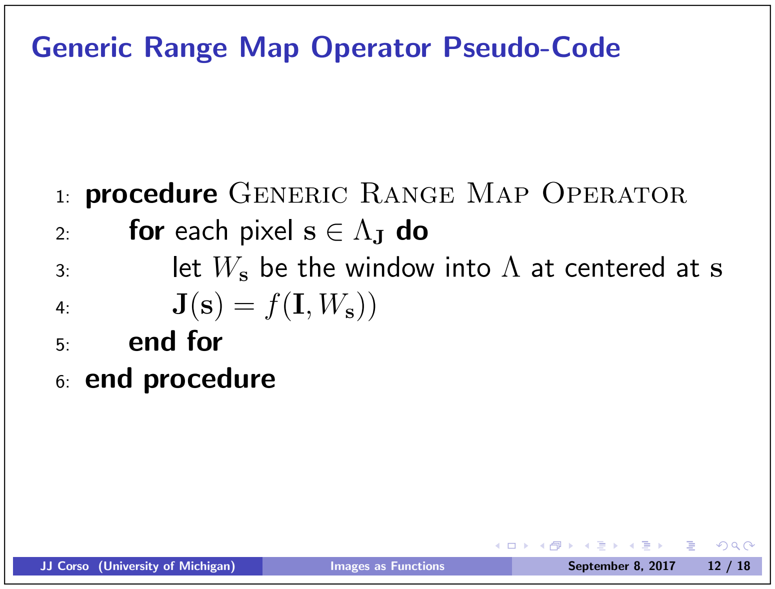
2.2 Range Map Operations

Range map operations continue to work in the image range but they *map* a single operation over the entire image domain Λ to yield a new image.

Definition 6. A *range map operation* $g\colon (W \times \mathbf{I} \mapsto \mathbb{R}) \times \mathbf{I} \mapsto \mathbf{J}$, for image $\mathbf{J}$, is a function that applies a spatial range operation f at every possible location in the image domain Λ to produce another image.

We can define the range map operation in a number of different ways. Here is a simple one, for output image $\mathbf{J}$ and spatial range operator $f\colon W \times \mathbf{I} \mapsto \mathbb{R}$:

```
1: procedure GENERIC RANGE MAP OPERATOR
2:     for each pixel s ∈ $\Lambda_\mathbf{J}$ do
3:         let $W_\mathrm{s}$ be the window into $\Lambda$ at centered at s
4:         $\mathbf{J}(\mathrm{s}) = f(\mathbf{I}, W_\mathrm{s})$
5:     end for
6: end procedure
```


Generic Range Map Operator Pseudo-Code

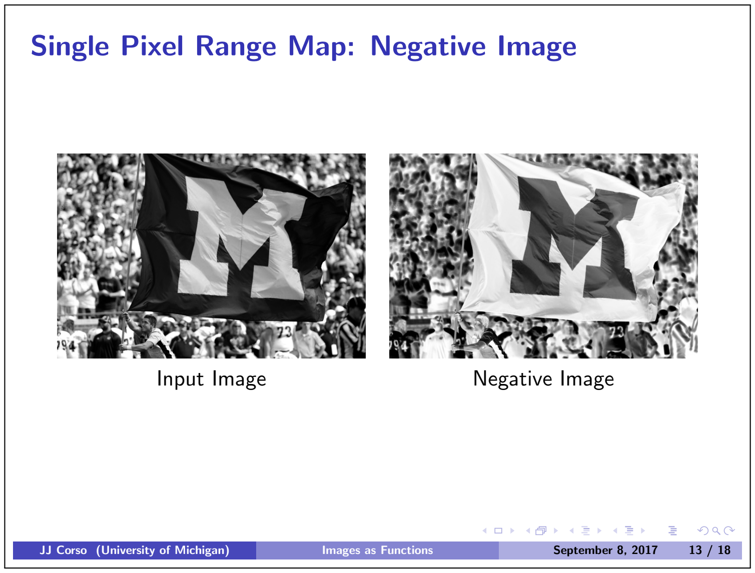
```
1: procedure GENERIC RANGE MAP OPERATOR
2:     for each pixel s ∈ $\Lambda_\mathbf{J}$ do
3:         let $W_\mathrm{s}$ be the window into $\Lambda$ at centered at s
4:         $\mathbf{J}(\mathrm{s}) = f(\mathbf{I}, W_\mathrm{s})$
5:     end for
6: end procedure
```

JJ Corso (University of Michigan) Images as Functions September 8, 2017 12 / 18

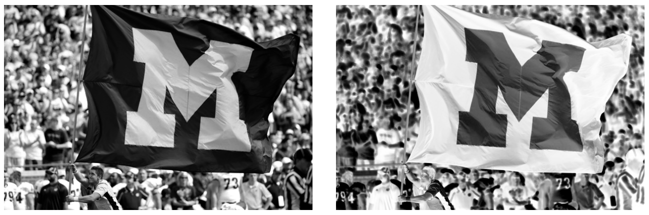
Remark 6.1. The output set of the spatial range operators have been generally defined as the real numbers. In practice, this is often relaxed to be the integers or some other range set.

Remark 6.2. Range map operations are commonly applied for single pixel windows in what are called *intensity transformations*. Very many possibly intensity transformations exist in the literature and include operators such as histogram equalization, or linear scaling, log-transforms and so on.

Example 5 — Single Pixel Range Map Intensity Transformation One example of an intensity transformation is the computation of a negative image. Each new pixel value is the negated input pixel value: $\mathbf{J}(\mathrm{s}) = -\mathbf{I}(\mathrm{s})$.



Single Pixel Range Map: Negative Image



Input Image Negative Image

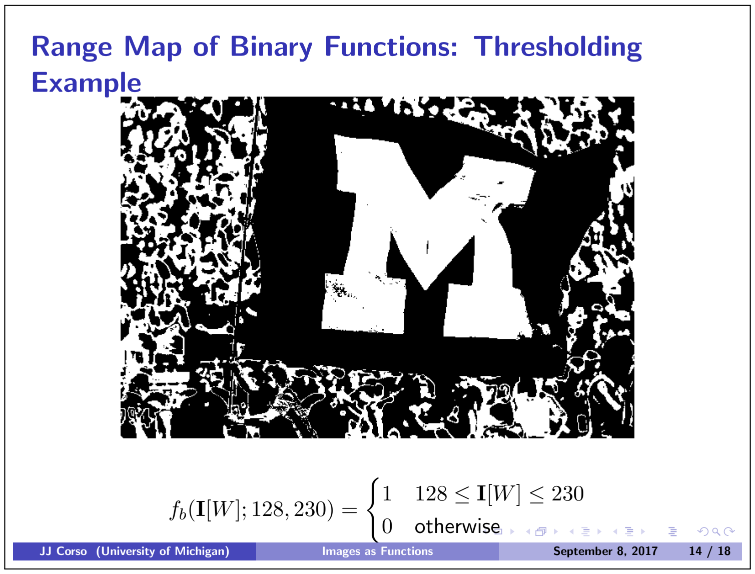
Remark 6.3. A special case of the range map is one that maps to binary pixel values rather than to the reals, $g\colon (W \times \mathbf{I} \mapsto \mathbb{B}) \times \mathbf{I} \mapsto \mathbf{B}$ for binary image $\mathbf{B}$.

Example 6 — Range Map of Binary Functions Consider a binary threshold operator, f_b that selects pixel values ($\tau^- \leq \mathbf{I}[W] \leq \tau^+$) within a certain range for individual pixel windows.

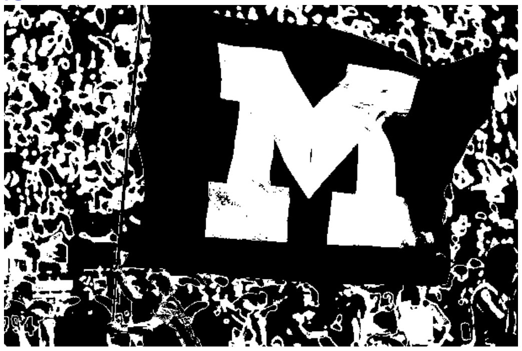
$$f_b(\mathbf{I}[W]; \tau^-, \tau^+) = \begin{cases} 1 & \tau^- \leq \mathbf{I}[W] \leq \tau^+ \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

or over spatial range operators on larger windows, such as sum:

$$f_s(\mathbf{I}, W; \tau^-, \tau^+) = \begin{cases} 1 & \tau^- \leq \mathrm{sum}(\mathbf{I}, W) \leq \tau^+ \\ 0 & \text{otherwise} \end{cases} \tag{10}$$



Range Map of Binary Functions: Thresholding Example

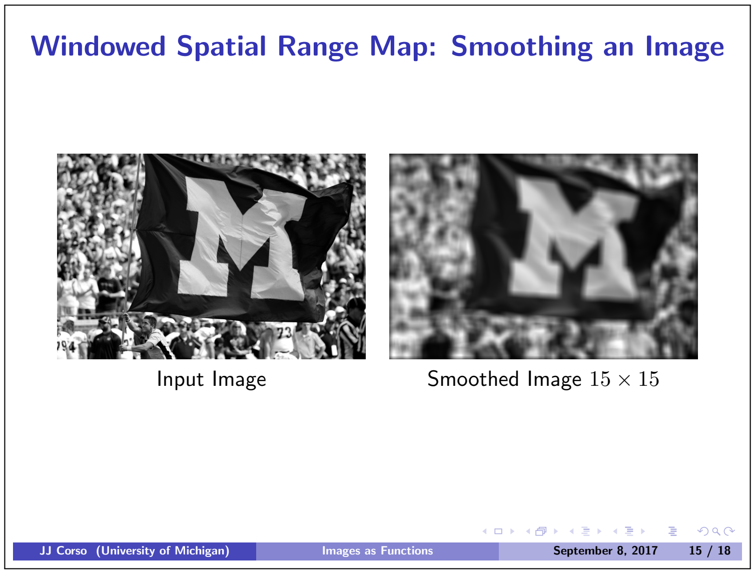


$$f_b(\mathbf{I}[W]; 128, 230) = \begin{cases} 1 & 128 \leq \mathbf{I}[W] \leq 230 \\ 0 & \text{otherwise} \end{cases}$$

Question 3. Consider converting the threshold operator into the magicwand operator: thresholding pixels within and certain range and spatially connected to a certain pixel location. This is non-trivial when we interpret images as functions. Why?

Remark 6.4. Spatial range operations on larger windows W will apply the same operation over every region of the image in a range map operator.

Example 7 — Spatial Range Map: Smoothing



Windowed Spatial Range Map: Smoothing an Image

Input Image

Smoothed Image 15×15

JJ Corso (University of Michigan) Images as Functions September 8, 2017 15 / 18

Remark 6.5. A particularly important parameter in a spatial range operator is known as a *kernel*. A kernel $\boldsymbol{\kappa}$ is a matrix of the same size as W whose values are real numbers, $\boldsymbol{\kappa} \in \mathbb{R}^{|W|}$. In kernel operations, the element-wise product between the kernel $\boldsymbol{\kappa}$ and the image window $\mathbf{I}[W]$ is computed and summed. This operation is most easily represented by the dot-product of the *vectorized* kernel and image window, which we denote as $\overrightarrow{\boldsymbol{\kappa}}$ and $\overrightarrow{\mathbf{I}[W]}$, respectively. Vectorizing a matrix means concatenating all of the columns of the matrix into one long column vector.

When this kernel operation is applied in a range map over the entire image, we call the process *discrete convolution* and represent is with the symbol $\otimes$. We write this convolution for output image location $\mathbf{J}(s,t)$, a kernel of size $2m+1 \times 2n+1$, and hence a window W that indexes into $\mathbf{I}$ at $s-m,\ldots,s+m$ and $t-n,\ldots,t+n$ as follows:

$$\mathbf{J}(s,t) = \boldsymbol{\kappa} \otimes \mathbf{I}[W] = \sum_{k=-m}^{m} \sum_{l=-n}^{n} \boldsymbol{\kappa}(k,l)\mathbf{I}(s-k,t-l) \tag{11}$$

$$= \overrightarrow{\boldsymbol{\kappa}}^{\mathsf{T}} \overrightarrow{\mathbf{I}[W]} \ . \tag{12}$$

We denote the kernel map over the all locations, accounting suitably for the image borders, as simply $\boldsymbol{\kappa} \otimes \mathbf{I}$, which is a function creating a new image $\mathbf{J}$. There is a continuous analog of convolution, but we do not discuss it here. This operation can clearly generalize to higher dimension assuming the dimensionality of the image and the kernel match.

Question 4. Show that convolution is a linear operator.

Example 8 — Discrete Image Derivatives In many computer vision problems, computing the partial derivatives of an image with respect to various directions is necessary. With the function interpretation of images, such computation is natural. First, recall the definition of a partial derivative of a continuous function over two variables, $I(x, y)$:

$$\frac{\partial I(x,y)}{\partial x} = \lim_{h \to 0} \frac{I(x+h,y) - I(x,y)}{h} \tag{13}$$

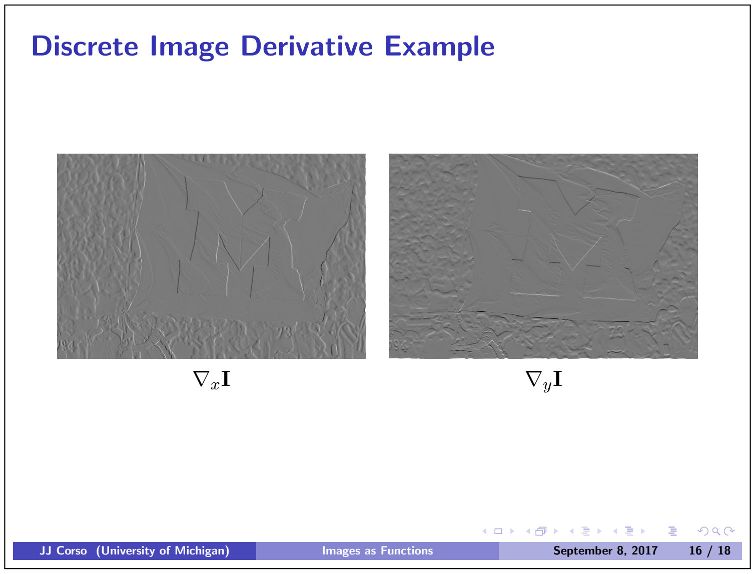
Now, considering a discrete image model, we have a fixed value of h which forces us to consider a finite difference interpretation of the partial derivative:

$$\frac{\mathrm{d}I(x,y)}{\mathrm{d}x} = \frac{I(x+h,y) - I(x,y)}{h} \quad . \tag{14}$$

Finally, considering our discrete image $\mathbf{I}$, we can set $h = 1$ to indicate one pixel difference. We denote this finite difference operator as ∇_x:

$$\nabla_x = \frac{\mathrm{d}\mathbf{I}(x,y)}{\mathrm{d}x} = \mathbf{I}(x+1,y) - \mathbf{I}(x,y) \quad . \tag{15}$$

We can write this operator by convolution with the row-vector kernel $\boldsymbol{\kappa} = \begin{bmatrix} 1 & -1 \end{bmatrix}$. And, similarly, this can be applied in the vertical direction simply by convolution with the transpose of the kernel: $\boldsymbol{\kappa} = \begin{bmatrix} 1 & -1 \end{bmatrix}^{\mathsf{T}}$.



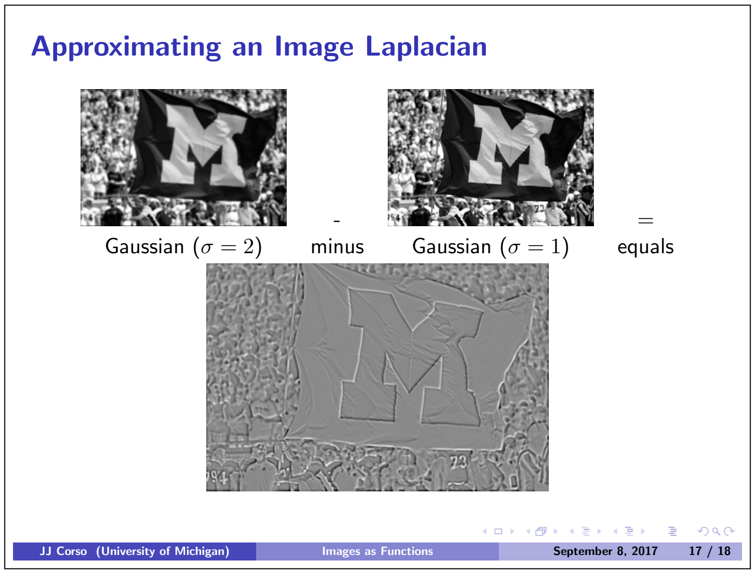
Discrete Image Derivative Example

$\nabla_x \mathbf{I}$ $\nabla_y \mathbf{I}$

JJ Corso (University of Michigan) Images as Functions September 8, 2017 16 / 18

Remark 6.6. Range operations of both types can also be binary, trinary or operate on arbitrary numbers of images concurrently. They hence can include the variety of arithmetic operations, like image addition and subtraction, to which we are accustomed.

Example 9 — Image Laplacians and Zero-Crossings Let us begin the example by considering the convolution of an image $\mathbf{I}$ by kernels $\boldsymbol{\kappa}_1$ and $\boldsymbol{\kappa}_2$, which are sampled Guassian functions of $\sigma = 1$ and $\sigma = 2$ respectively. Denote these convolved images as $\mathbf{G}_1$ and $\mathbf{G}_2$, respectively. Now, let us take the difference of these two images, $\mathbf{G}_2 - \mathbf{G}_1$, which is clearly a simple but important operation.



Approximating an Image Laplacian

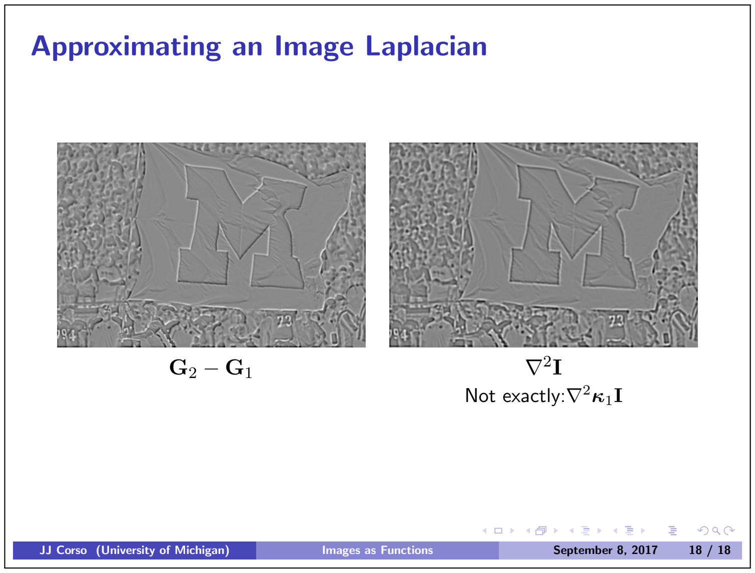
Gaussian ($\sigma = 2$) - minus Gaussian ($\sigma = 1$) = equals

JJ Corso (University of Michigan) Images as Functions September 8, 2017 17 / 18

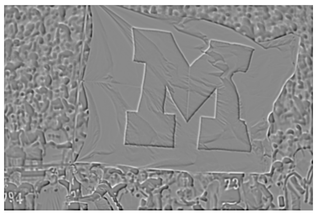
It is interesting that the edges of the image have been highlighted. Let us look closer at what is happening here. This requires that we look closer at intensity changes, or *edges*, in the image. “A sudden intensity change in the image will give rise to a peak or trough in the first derivative or, equivalently, to a zero-crossing in the second derivative.” (Marr 1982)

In Example 8, we saw the impact of the first derivative in each axis direction. Let us consider the second derivative operator, called the Laplacian,

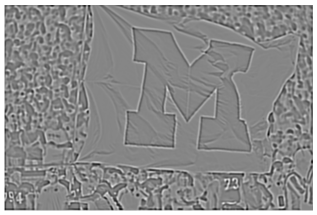
$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \quad . \tag{16}$$



Approximating an Image Laplacian



$\mathbf{G}_2 - \mathbf{G}_1$



$\nabla^2 \mathbf{I}$

Not exactly: $\nabla^2 \boldsymbol{\kappa}_1 \mathbf{I}$

JJ Corso (University of Michigan) Images as Functions September 8, 2017 18 / 18

These two processes are performing a similar extraction.

Question 5. What role does the Guassian play here?

Question 6. Show that the difference of Gaussian process is an approximation to the Laplacian of a Gaussian filter.

2.3 Domain Operations

Before we can intelligently analyze and manipulate images, we need to establish a vocabulary for describing the geometry of a scene. We also need to understand the image formation process that produced a particular image given a set of lighting conditions, scene geometry, surface properties, and camera optics. In this chapter, we present a simplified model of such an image formation process.

Section 2.1 introduces the basic geometric primitives used throughout the book (points, lines, and planes) and the *geometric* transformations that project these 3D quantities into 2D image features (Figure 2.1a). Section 2.2 describes how lighting, surface properties (Figure 2.1b), and camera *optics* (Figure 2.1c) interact in order to produce the color values that fall onto the image sensor. Section 2.3 describes how continuous color images are turned into discrete digital *samples* inside the image sensor (Figure 2.1d) and how to avoid (or at least characterize) sampling deficiencies, such as aliasing.

The material covered in this chapter is but a brief summary of a very rich and deep set of topics, traditionally covered in a number of separate fields. A more thorough introduction to the geometry of points, lines, planes, and projections can be found in textbooks on multi-view geometry (Hartley and Zisserman 2004; Faugeras and Luong 2001) and computer graphics (Foley, van Dam, Feiner *et al.* 1995). The image formation (synthesis) process is traditionally taught as part of a computer graphics curriculum (Foley, van Dam, Feiner *et al.* 1995; Glassner 1995; Watt 1995; Shirley 2005) but it is also studied in physics-based computer vision (Wolff, Shafer, and Healey 1992a). The behavior of camera lens systems is studied in optics (Möller 1988; Hecht 2001; Ray 2002). Two good books on color theory are (Wyszecki and Stiles 2000; Healey and Shafer 1992), with (Livingstone 2008) providing a more fun and informal introduction to the topic of color perception. Topics relating to sampling and aliasing are covered in textbooks on signal and image processing (Crane 1997; Jähne 1997; Oppenheim and Schafer 1996; Oppenheim, Schafer, and Buck 1999; Pratt 2007; Russ 2007; Burger and Burge 2008; Gonzales and Woods 2008).

A note to students: If you have already studied computer graphics, you may want to skim the material in Section 2.1, although the sections on projective depth and object-centered projection near the end of Section 2.1.5 may be new to you. Similarly, physics students (as well as computer graphics students) will mostly be familiar with Section 2.2. Finally, students with a good background in image processing will already be familiar with sampling issues (Section 2.3) as well as some of the material in Chapter 3.

2.1 Geometric primitives and transformations

In this section, we introduce the basic 2D and 3D primitives used in this textbook, namely points, lines, and planes. We also describe how 3D features are projected into 2D features.

More detailed descriptions of these topics (along with a gentler and more intuitive introduction) can be found in textbooks on multiple-view geometry (Hartley and Zisserman 2004; Faugeras and Luong 2001).

2.1.1 Geometric primitives

Geometric primitives form the basic building blocks used to describe three-dimensional shapes. In this section, we introduce points, lines, and planes. Later sections of the book discuss curves (Sections 5.1 and 11.2), surfaces (Section 12.3), and volumes (Section 12.5).

2D points. 2D points (pixel coordinates in an image) can be denoted using a pair of values, $\boldsymbol{x} = (x, y) \in \mathcal{R}^2$, or alternatively,

$$\boldsymbol{x} = \begin{bmatrix} x \\ y \end{bmatrix}. \tag{2.1}$$

(As stated in the introduction, we use the $(x_1, x_2, \ldots)$ notation to denote column vectors.)

2D points can also be represented using *homogeneous coordinates*, $\tilde{\boldsymbol{x}} = (\tilde{x}, \tilde{y}, \tilde{w}) \in \mathcal{P}^2$, where vectors that differ only by scale are considered to be equivalent. $\mathcal{P}^2 = \mathcal{R}^3 - (0, 0, 0)$ is called the 2D *projective space*.

A homogeneous vector $\tilde{\boldsymbol{x}}$ can be converted back into an *inhomogeneous* vector $\boldsymbol{x}$ by dividing through by the last element $\tilde{w}$, i.e.,

$$\tilde{\boldsymbol{x}} = (\tilde{x}, \tilde{y}, \tilde{w}) = \tilde{w}(x, y, 1) = \tilde{w}\bar{\boldsymbol{x}}, \tag{2.2}$$

where $\bar{\boldsymbol{x}} = (x, y, 1)$ is the *augmented vector*. Homogeneous points whose last element is $\tilde{w} = 0$ are called *ideal points* or *points at infinity* and do not have an equivalent inhomogeneous representation.

2D lines. 2D lines can also be represented using homogeneous coordinates $\tilde{\boldsymbol{l}} = (a, b, c)$. The corresponding *line equation* is

$$\bar{\boldsymbol{x}} \cdot \tilde{\boldsymbol{l}} = ax + by + c = 0. \tag{2.3}$$

We can normalize the line equation vector so that $\boldsymbol{l} = (\hat{n}_x, \hat{n}_y, d) = (\hat{\boldsymbol{n}}, d)$ with $\|\hat{\boldsymbol{n}}\| = 1$. In this case, $\hat{\boldsymbol{n}}$ is the *normal vector* perpendicular to the line and d is its distance to the origin (Figure 2.2). (The one exception to this normalization is the *line at infinity* $\tilde{\boldsymbol{l}} = (0, 0, 1)$, which includes all (ideal) points at infinity.)

We can also express $\hat{\boldsymbol{n}}$ as a function of rotation angle θ, $\hat{\boldsymbol{n}} = (\hat{n}_x, \hat{n}_y) = (\cos\theta, \sin\theta)$ (Figure 2.2a). This representation is commonly used in the *Hough transform* line-finding

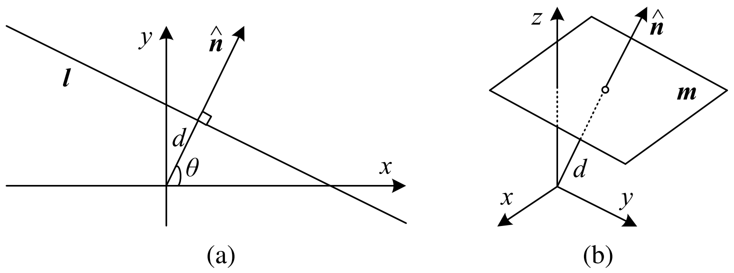


Figure 2.2 (a) 2D line equation and (b) 3D plane equation, expressed in terms of the normal $\hat{\boldsymbol{n}}$ and distance to the origin d.

algorithm, which is discussed in Section 4.3.2. The combination (θ, d) is also known as *polar coordinates*.

When using homogeneous coordinates, we can compute the intersection of two lines as

$$\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{l}}_1 \times \tilde{\boldsymbol{l}}_2, \tag{2.4}$$

where $\times$ is the cross product operator. Similarly, the line joining two points can be written as

$$\tilde{\boldsymbol{l}} = \tilde{\boldsymbol{x}}_1 \times \tilde{\boldsymbol{x}}_2. \tag{2.5}$$

When trying to fit an intersection point to multiple lines or, conversely, a line to multiple points, least squares techniques (Section 6.1.1 and Appendix A.2) can be used, as discussed in Exercise 2.1.

2D conics. There are other algebraic curves that can be expressed with simple polynomial homogeneous equations. For example, the *conic sections* (so called because they arise as the intersection of a plane and a 3D cone) can be written using a *quadric* equation

$$\tilde{\boldsymbol{x}}^T \boldsymbol{Q} \tilde{\boldsymbol{x}} = 0. \tag{2.6}$$

Quadric equations play useful roles in the study of multi-view geometry and camera calibration (Hartley and Zisserman 2004; Faugeras and Luong 2001) but are not used extensively in this book.

3D points. Point coordinates in three dimensions can be written using inhomogeneous coordinates $\boldsymbol{x} = (x, y, z) \in \mathcal{R}^3$ or homogeneous coordinates $\tilde{\boldsymbol{x}} = (\tilde{x}, \tilde{y}, \tilde{z}, \tilde{w}) \in \mathcal{P}^3$. As before, it is sometimes useful to denote a 3D point using the augmented vector $\bar{\boldsymbol{x}} = (x, y, z, 1)$ with $\tilde{\boldsymbol{x}} = \tilde{w}\bar{\boldsymbol{x}}$.

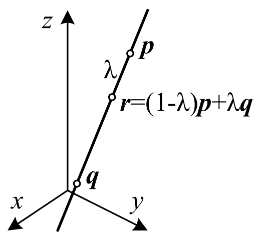


Figure 2.3 3D line equation, $\boldsymbol{r} = (1 - \lambda)\boldsymbol{p} + \lambda\boldsymbol{q}$.

3D planes. 3D planes can also be represented as homogeneous coordinates $\tilde{\boldsymbol{m}} = (a, b, c, d)$ with a corresponding plane equation

$$\bar{\boldsymbol{x}} \cdot \tilde{\boldsymbol{m}} = ax + by + cz + d = 0. \tag{2.7}$$

We can also normalize the plane equation as $\boldsymbol{m} = (\hat{n}_x, \hat{n}_y, \hat{n}_z, d) = (\hat{\boldsymbol{n}}, d)$ with $\|\hat{\boldsymbol{n}}\| = 1$. In this case, $\hat{\boldsymbol{n}}$ is the *normal vector* perpendicular to the plane and d is its distance to the origin (Figure 2.2b). As with the case of 2D lines, the *plane at infinity* $\tilde{\boldsymbol{m}} = (0, 0, 0, 1)$, which contains all the points at infinity, cannot be normalized (i.e., it does not have a unique normal or a finite distance).

We can express $\hat{\boldsymbol{n}}$ as a function of two angles (θ, ϕ),

$$\hat{\boldsymbol{n}} = (\cos\theta\cos\phi, \sin\theta\cos\phi, \sin\phi), \tag{2.8}$$

i.e., using *spherical coordinates*, but these are less commonly used than polar coordinates since they do not uniformly sample the space of possible normal vectors.

3D lines. Lines in 3D are less elegant than either lines in 2D or planes in 3D. One possible representation is to use two points on the line, $(\boldsymbol{p}, \boldsymbol{q})$. Any other point on the line can be expressed as a linear combination of these two points

$$\boldsymbol{r} = (1 - \lambda)\boldsymbol{p} + \lambda\boldsymbol{q}, \tag{2.9}$$

as shown in Figure 2.3. If we restrict $0 \leq \lambda \leq 1$, we get the *line segment* joining $\boldsymbol{p}$ and $\boldsymbol{q}$.

If we use homogeneous coordinates, we can write the line as

$$\tilde{\boldsymbol{r}} = \mu\tilde{\boldsymbol{p}} + \lambda\tilde{\boldsymbol{q}}. \tag{2.10}$$

A special case of this is when the second point is at infinity, i.e., $\tilde{\boldsymbol{q}} = (\hat{d}_x, \hat{d}_y, \hat{d}_z, 0) = (\hat{\boldsymbol{d}}, 0)$. Here, we see that $\hat{\boldsymbol{d}}$ is the *direction* of the line. We can then re-write the inhomogeneous 3D line equation as

$$\boldsymbol{r} = \boldsymbol{p} + \lambda\hat{\boldsymbol{d}}. \tag{2.11}$$

A disadvantage of the endpoint representation for 3D lines is that it has too many degrees of freedom, i.e., six (three for each endpoint) instead of the four degrees that a 3D line truly has. However, if we fix the two points on the line to lie in specific planes, we obtain a representation with four degrees of freedom. For example, if we are representing nearly vertical lines, then $z = 0$ and $z = 1$ form two suitable planes, i.e., the (x, y) coordinates in both planes provide the four coordinates describing the line. This kind of two-plane parameterization is used in the *light field* and *Lumigraph* image-based rendering systems described in Chapter 13 to represent the collection of rays seen by a camera as it moves in front of an object. The two-endpoint representation is also useful for representing line segments, even when their exact endpoints cannot be seen (only guessed at).

If we wish to represent all possible lines without bias towards any particular orientation, we can use *Plücker coordinates* (Hartley and Zisserman 2004, Chapter 2; Faugeras and Luong 2001, Chapter 3). These coordinates are the six independent non-zero entries in the 4×4 skew symmetric matrix

$$\boldsymbol{L} = \tilde{\boldsymbol{p}}\tilde{\boldsymbol{q}}^T - \tilde{\boldsymbol{q}}\tilde{\boldsymbol{p}}^T, \tag{2.12}$$

where $\tilde{\boldsymbol{p}}$ and $\tilde{\boldsymbol{q}}$ are *any* two (non-identical) points on the line. This representation has only four degrees of freedom, since $\boldsymbol{L}$ is homogeneous and also satisfies $det(\boldsymbol{L}) = 0$, which results in a quadratic constraint on the Plücker coordinates.

In practice, the minimal representation is not essential for most applications. An adequate model of 3D lines can be obtained by estimating their direction (which may be known ahead of time, e.g., for architecture) and some point within the visible portion of the line (see Section 7.5.1) or by using the two endpoints, since lines are most often visible as finite line segments. However, if you are interested in more details about the topic of minimal line parameterizations, Förstner (2005) discusses various ways to infer and model 3D lines in projective geometry, as well as how to estimate the uncertainty in such fitted models.

3D quadrics. The 3D analog of a conic section is a quadric surface

$$\bar{\boldsymbol{x}}^T\boldsymbol{Q}\bar{\boldsymbol{x}} = 0 \tag{2.13}$$

(Hartley and Zisserman 2004, Chapter 2). Again, while quadric surfaces are useful in the study of multi-view geometry and can also serve as useful modeling primitives (spheres, ellipsoids, cylinders), we do not study them in great detail in this book.

2.1.2 2D transformations

Having defined our basic primitives, we can now turn our attention to how they can be transformed. The simplest transformations occur in the 2D plane and are illustrated in Figure 2.4.

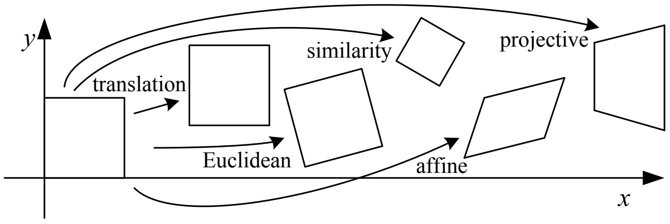


Figure 2.4 Basic set of 2D planar transformations.

Translation. 2D translations can be written as $\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{t}$ or

$$\boldsymbol{x}' = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{t} \end{bmatrix} \bar{\boldsymbol{x}} \tag{2.14}$$

where $\boldsymbol{I}$ is the (2×2) identity matrix or

$$\bar{\boldsymbol{x}}' = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{t} \\ \boldsymbol{0}^T & 1 \end{bmatrix} \bar{\boldsymbol{x}} \tag{2.15}$$

where $\boldsymbol{0}$ is the zero vector. Using a 2×3 matrix results in a more compact notation, whereas using a full-rank 3×3 matrix (which can be obtained from the 2×3 matrix by appending a $[\boldsymbol{0}^T\ 1]$ row) makes it possible to chain transformations using matrix multiplication. Note that in any equation where an augmented vector such as $\bar{\boldsymbol{x}}$ appears on both sides, it can always be replaced with a full homogeneous vector $\tilde{\boldsymbol{x}}$.

Rotation + translation. This transformation is also known as *2D rigid body motion* or the *2D Euclidean transformation* (since Euclidean distances are preserved). It can be written as $\boldsymbol{x}' = \boldsymbol{R}\boldsymbol{x} + \boldsymbol{t}$ or

$$\boldsymbol{x}' = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \end{bmatrix} \bar{\boldsymbol{x}} \tag{2.16}$$

where

$$\boldsymbol{R} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{2.17}$$

is an orthonormal rotation matrix with $\boldsymbol{R}\boldsymbol{R}^T = \boldsymbol{I}$ and $|\boldsymbol{R}| = 1$.

Scaled rotation. Also known as the *similarity transform*, this transformation can be expressed as $\boldsymbol{x}' = s\boldsymbol{R}\boldsymbol{x} + \boldsymbol{t}$ where s is an arbitrary scale factor. It can also be written as

$$\boldsymbol{x}' = \begin{bmatrix} s\boldsymbol{R} & \boldsymbol{t} \end{bmatrix} \bar{\boldsymbol{x}} = \begin{bmatrix} a & -b & t_x \\ b & a & t_y \end{bmatrix} \bar{\boldsymbol{x}}, \tag{2.18}$$

where we no longer require that $a^2 + b^2 = 1$. The similarity transform preserves angles between lines.

Affine. The affine transformation is written as $\boldsymbol{x}' = \boldsymbol{A}\bar{\boldsymbol{x}}$, where $\boldsymbol{A}$ is an arbitrary 2×3 matrix, i.e.,

$$\boldsymbol{x}' = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{bmatrix} \bar{\boldsymbol{x}}. \tag{2.19}$$

Parallel lines remain parallel under affine transformations.

Projective. This transformation, also known as a *perspective transform* or *homography*, operates on homogeneous coordinates,

$$\tilde{\boldsymbol{x}}' = \tilde{\boldsymbol{H}}\tilde{\boldsymbol{x}}, \tag{2.20}$$

where $\tilde{\boldsymbol{H}}$ is an arbitrary 3×3 matrix. Note that $\tilde{\boldsymbol{H}}$ is homogeneous, i.e., it is only defined up to a scale, and that two $\tilde{\boldsymbol{H}}$ matrices that differ only by scale are equivalent. The resulting homogeneous coordinate $\tilde{\boldsymbol{x}}'$ must be normalized in order to obtain an inhomogeneous result $\boldsymbol{x}$, i.e.,

$$x' = \frac{h_{00}x + h_{01}y + h_{02}}{h_{20}x + h_{21}y + h_{22}} \text{ and } y' = \frac{h_{10}x + h_{11}y + h_{12}}{h_{20}x + h_{21}y + h_{22}}. \tag{2.21}$$

Perspective transformations preserve straight lines (i.e., they remain straight after the transformation).

Hierarchy of 2D transformations. The preceding set of transformations are illustrated in Figure 2.4 and summarized in Table 2.1. The easiest way to think of them is as a set of (potentially restricted) 3×3 matrices operating on 2D homogeneous coordinate vectors. Hartley and Zisserman (2004) contains a more detailed description of the hierarchy of 2D planar transformations.

The above transformations form a nested set of *groups*, i.e., they are closed under composition and have an inverse that is a member of the same group. (This will be important later when applying these transformations to images in Section 3.6.) Each (simpler) group is a subset of the more complex group below it.

Co-vectors. While the above transformations can be used to transform points in a 2D plane, can they also be used directly to transform a line equation? Consider the homogeneous equation $\tilde{\boldsymbol{l}} \cdot \tilde{\boldsymbol{x}} = 0$. If we transform $\boldsymbol{x}' = \tilde{\boldsymbol{H}}\boldsymbol{x}$, we obtain

$$\tilde{\boldsymbol{l}}' \cdot \tilde{\boldsymbol{x}}' = \tilde{\boldsymbol{l}}'^T \tilde{\boldsymbol{H}}\tilde{\boldsymbol{x}} = (\tilde{\boldsymbol{H}}^T \tilde{\boldsymbol{l}}')^T \tilde{\boldsymbol{x}} = \tilde{\boldsymbol{l}} \cdot \tilde{\boldsymbol{x}} = 0, \tag{2.22}$$

i.e., $\tilde{\boldsymbol{l}}' = \tilde{\boldsymbol{H}}^{-T}\tilde{\boldsymbol{l}}$. Thus, the action of a projective transformation on a *co-vector* such as a 2D line or 3D normal can be represented by the transposed inverse of the matrix, which is equivalent to the *adjoint* of $\tilde{\boldsymbol{H}}$, since projective transformation matrices are homogeneous. Jim

Transformation	**Matrix**	**# DoF**	**Preserves**	**Icon**
translation	$\left[\begin{array}{c\|c} \boldsymbol{I} & \boldsymbol{t} \end{array}\right]_{2\times 3}$	2	orientation	□
rigid (Euclidean)	$\left[\begin{array}{c\|c} \boldsymbol{R} & \boldsymbol{t} \end{array}\right]_{2\times 3}$	3	lengths	◇
similarity	$\left[\begin{array}{c\|c} s\boldsymbol{R} & \boldsymbol{t} \end{array}\right]_{2\times 3}$	4	angles	◇
affine	$\left[\begin{array}{c} \boldsymbol{A} \end{array}\right]_{2\times 3}$	6	parallelism	▱
projective	$\left[\begin{array}{c} \tilde{\boldsymbol{H}} \end{array}\right]_{3\times 3}$	8	straight lines	⏢

Table 2.1 Hierarchy of 2D coordinate transformations. Each transformation also preserves the properties listed in the rows below it, i.e., similarity preserves not only angles but also parallelism and straight lines. The 2×3 matrices are extended with a third $[\mathbf{0}^T\ 1]$ row to form a full 3×3 matrix for homogeneous coordinate transformations.

Blinn (1998) describes (in Chapters 9 and 10) the ins and outs of notating and manipulating co-vectors.

While the above transformations are the ones we use most extensively, a number of additional transformations are sometimes used.

Stretch/squash. This transformation changes the aspect ratio of an image,

$$\begin{aligned} x' &= s_x x + t_x \\ y' &= s_y y + t_y, \end{aligned}$$

and is a restricted form of an affine transformation. Unfortunately, it does not nest cleanly with the groups listed in Table 2.1.

Planar surface flow. This eight-parameter transformation (Horn 1986; Bergen, Anandan, Hanna *et al.* 1992; Girod, Greiner, and Niemann 2000),

$$\begin{aligned} x' &= a_0 + a_1 x + a_2 y + a_6 x^2 + a_7 xy \\ y' &= a_3 + a_4 x + a_5 y + a_7 x^2 + a_6 xy, \end{aligned}$$

arises when a planar surface undergoes a small 3D motion. It can thus be thought of as a small motion approximation to a full homography. Its main attraction is that it is *linear* in the motion parameters, a_k, which are often the quantities being estimated.

Transformation	**Matrix**	**# DoF**	**Preserves**	**Icon**
translation	$\left[\begin{array}{c\|c} \boldsymbol{I} & \boldsymbol{t} \end{array}\right]_{3\times 4}$	3	orientation	□
rigid (Euclidean)	$\left[\begin{array}{c\|c} \boldsymbol{R} & \boldsymbol{t} \end{array}\right]_{3\times 4}$	6	lengths	◇
similarity	$\left[\begin{array}{c\|c} s\boldsymbol{R} & \boldsymbol{t} \end{array}\right]_{3\times 4}$	7	angles	◇
affine	$\left[\begin{array}{c} \boldsymbol{A} \end{array}\right]_{3\times 4}$	12	parallelism	▱
projective	$\left[\begin{array}{c} \tilde{\boldsymbol{H}} \end{array}\right]_{4\times 4}$	15	straight lines	⏢

Table 2.2 Hierarchy of 3D coordinate transformations. Each transformation also preserves the properties listed in the rows below it, i.e., similarity preserves not only angles but also parallelism and straight lines. The 3×4 matrices are extended with a fourth $[\mathbf{0}^T \; 1]$ row to form a full 4×4 matrix for homogeneous coordinate transformations. The mnemonic icons are drawn in 2D but are meant to suggest transformations occurring in a full 3D cube.

Bilinear interpolant. This eight-parameter transform (Wolberg 1990),

$$\begin{aligned} x' &= a_0 + a_1 x + a_2 y + a_6 xy \\ y' &= a_3 + a_4 x + a_5 y + a_7 xy, \end{aligned}$$

can be used to interpolate the deformation due to the motion of the four corner points of a square. (In fact, it can interpolate the motion of any four non-collinear points.) While the deformation is linear in the motion parameters, it does not generally preserve straight lines (only lines parallel to the square axes). However, it is often quite useful, e.g., in the interpolation of sparse grids using splines (Section 8.3).

2.1.3 3D transformations

The set of three-dimensional coordinate transformations is very similar to that available for 2D transformations and is summarized in Table 2.2. As in 2D, these transformations form a nested set of groups. Hartley and Zisserman (2004, Section 2.4) give a more detailed description of this hierarchy.

Translation. 3D translations can be written as $\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{t}$ or

$$\boldsymbol{x}' = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{t} \end{bmatrix} \bar{\boldsymbol{x}} \tag{2.23}$$