Practical Course: GPU Programming in Computer Vision

Mathematics 1: Basics

Björn Häfner, Benedikt Löwenhauser, Thomas Möllenhoff

Technische Universität München
Department of Informatics
Computer Vision Group

Summer Term 2017
September 11 - October 8

Outline

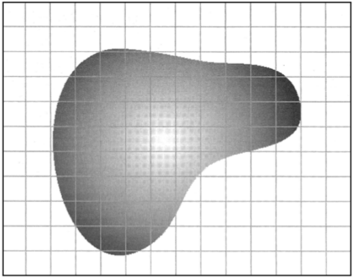
1. images
2. differential operators and convolution
3. discretization

Outline

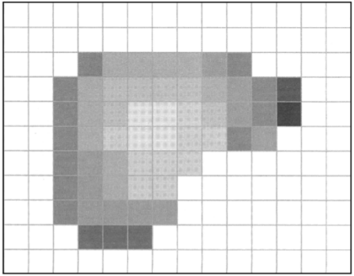
continuous setting

We think of images as (possibly vector-valued) functions that are defined on a *continuous* domain $\Omega \subset \mathbb{R}^d$:

$$u : \Omega \to \mathbb{R}^n$$



continuous setting



discrete setting

examples

domain $\Omega \subset \mathbb{R}^d$ (usually rectangular):

- $d = 2$: image (rectangle)
- $d = 3$: volume, movie (cuboid)

range $\mathbb{R}^n$:

- $n = 1$: greyscale images, ...
- $n = 2$: $2D$-vector fields, ...
- $n = 3$: RGB images, ...
- $n = 4$: RGB-D images, ...

Each dimension of the range is called a *channel*. We can represent an image with n channels as n stacked single-channel images.

examples

domain $\Omega \subset \mathbb{R}^d$ (usually rectangular):

- $d = 2$: image (rectangle)
- $d = 3$: volume, movie (cuboid)

range $\mathbb{R}^n$:

- $n = 1$: greyscale images, ...
- $n = 2$: $2D$-vector fields, ...
- $n = 3$: RGB images, ...
- $n = 4$: RGB-D images, ...

Each dimension of the range is called a *channel*. We can represent an image with n channels as n stacked single-channel images.

examples

domain $\Omega \subset \mathbb{R}^d$ (usually rectangular):

- $d = 2$: image (rectangle)
- $d = 3$: volume, movie (cuboid)

range $\mathbb{R}^n$:

- $n = 1$: greyscale images, ...
- $n = 2$: $2D$-vector fields, ...
- $n = 3$: RGB images, ...
- $n = 4$: RGB-D images, ...

Each dimension of the range is called a *channel*. We can represent an image with n channels as n stacked single-channel images.

Outline

partial derivatives

Let's assume we have a two-dimensional domain $\Omega \subset \mathbb{R}^2$.

The partial derivatives of a scalar ($d = 1$) image $u : \Omega \to \mathbb{R}$ at $(x, y) \in \Omega$ is defined in the following way:

$$\partial_x u : \Omega \to \mathbb{R}, \quad \partial_x u(x, y) = \lim_{h \to 0} \frac{u(x + h, y) - u(x, y)}{h}$$

$$\partial_y u : \Omega \to \mathbb{R}, \quad \partial_y u(x, y) = \lim_{h \to 0} \frac{u(x, y + h) - u(x, y)}{h}$$

gradient

The *gradient* combines all partial derivatives into a vector:

$$\nabla u : \Omega \to \mathbb{R}^2, \nabla u(x, y) = \begin{pmatrix} \partial_x u(x, y) \\ \partial_y u(x, y) \end{pmatrix}$$

The gradient of a function at a point (x, y) always points in the direction of the *steepest increase* of u.

Multi-channel images $u : \Omega \to \mathbb{R}^n$: one gradient per channel

$$\nabla u : \Omega \to \mathbb{R}^{2\times n}, \quad \nabla u(x, y) = (\nabla u_1(x, y), \dots, \nabla u_n(x, y))$$

gradient

The *gradient* combines all partial derivatives into a vector:

$$\nabla u : \Omega \to \mathbb{R}^2, \nabla u(x,y) = \begin{pmatrix} \partial_x u(x,y) \\ \partial_y u(x,y) \end{pmatrix}$$

The gradient of a function at a point (x, y) always points in the direction of the *steepest increase* of u.

Multi-channel images $u : \Omega \to \mathbb{R}^n$: one gradient per channel

$$\nabla u : \Omega \to \mathbb{R}^{2\times n}, \quad \nabla u(x,y) = (\nabla u_1(x,y), \dots, \nabla u_n(x,y))$$

gradient

The *gradient* combines all partial derivatives into a vector:

$$\nabla u : \Omega \to \mathbb{R}^2, \nabla u(x, y) = \begin{pmatrix} \partial_x u(x, y) \\ \partial_y u(x, y) \end{pmatrix}$$

The gradient of a function at a point (x, y) always points in the direction of the *steepest increase* of u.

Multi-channel images $u : \Omega \to \mathbb{R}^n$: one gradient per channel

$$\nabla u : \Omega \to \mathbb{R}^{2\times n}, \quad \nabla u(x, y) = (\nabla u_1(x, y), \ldots, \nabla u_n(x, y))$$

gradient norm

the pointwise magnitude of the gradient of an image,

$$|\nabla u(x,y)| = \sqrt{\partial_x u(x,y)^2 + \partial_y u(x,y)^2},$$

may serve as an edge detector.

Multi-channel images $u : \Omega \to \mathbb{R}^n$: Norm over all partial derivatives:

$$|\nabla u(x,y)| = \sqrt{\sum_{i=1}^{n} |\nabla u_i(x,y)|^2} = \sqrt{\sum_{i=1}^{n} \partial_x u_i(x,y)^2 + \partial_y u_i(x,y)^2}$$

gradient norm

the pointwise magnitude of the gradient of an image,

$$|\nabla u(x,y)| = \sqrt{\partial_x u(x,y)^2 + \partial_y u(x,y)^2},$$

may serve as an edge detector.

Multi-channel images $u : \Omega \to \mathbb{R}^n$: Norm over all partial derivatives:

$$|\nabla u(x,y)| = \sqrt{\sum_{i=1}^{n} |\nabla u_i(x,y)|^2} = \sqrt{\sum_{i=1}^{n} \partial_x u_i(x,y)^2 + \partial_y u_i(x,y)^2}$$

divergence

The *divergence* of a vector field $u : \Omega \to \mathbb{R}^2$ is defined as

$$\operatorname{div} u : \Omega \to \mathbb{R}, \quad \operatorname{div} u(x, y) = \partial_x u_1(x, y) + \partial_y u_2(x, y)$$

Multi-channel 2D-vector fields $u : \Omega \to \mathbb{R}^{2\times n}$: divergence per channel

$$\operatorname{div} u : \Omega \to \mathbb{R}^n, \quad \operatorname{div} u = (\operatorname{div} u_1, \ldots, \operatorname{div} u_n)$$

divergence

The *divergence* of a vector field $u : \Omega \to \mathbb{R}^2$ is defined as

$$\operatorname{div} u : \Omega \to \mathbb{R}, \quad \operatorname{div} u(x,y) = \partial_x u_1(x,y) + \partial_y u_2(x,y)$$

Multi-channel 2D-vector fields $u : \Omega \to \mathbb{R}^{2\times n}$: divergence per channel

$$\operatorname{div} u : \Omega \to \mathbb{R}^n, \quad \operatorname{div} u = (\operatorname{div} u_1, \ldots, \operatorname{div} u_n)$$

Laplacian

The gradient $\nabla u : \Omega \to \mathbb{R}^2$ is a 2D-vector field and divergence div operates on 2D-vector fields. Thus we can concatenate these two operators. The result is the *Laplacian*:

$$\Delta u : \Omega \to \mathbb{R}, \quad \Delta u := \operatorname{div}(\nabla u) = \operatorname{div} \begin{pmatrix} \partial_x u \\ \partial_y u \end{pmatrix}$$

$$\Delta u(x, y) = \partial_{xx} u(x, y) + \partial_{yy} u(x, y)$$

There is a physical interpretation of the Laplacian: For example, if $u(x, y)$ denotes the temperature at point (x, y), then $\Delta u(x, y)$ is the rate of local temperature decrease/increase: $\partial_t u(x, y) = a \Delta u(x, y)$ for some constant $a > 0$.

Multi-channel images $u : \Omega \to \mathbb{R}^n$: channel-wise

Laplacian

The gradient $\nabla u : \Omega \to \mathbb{R}^2$ is a 2D-vector field and divergence div operates on 2D-vector fields. Thus we can concatenate these two operators. The result is the *Laplacian*:

$$\Delta u : \Omega \to \mathbb{R}, \quad \Delta u := \operatorname{div}(\nabla u) = \operatorname{div}\begin{pmatrix} \partial_x u \\ \partial_y u \end{pmatrix}$$

$$\Delta u(x, y) = \partial_{xx} u(x, y) + \partial_{yy} u(x, y)$$

There is a physical interpretation of the Laplacian: For example, if $u(x, y)$ denotes the temperature at point (x, y), then $\Delta u(x, y)$ is the rate of local temperature decrease/increase: $\partial_t u(x, y) = a\Delta u(x, y)$ for some constant $a > 0$.

Multi-channel images $u : \Omega \to \mathbb{R}^n$: channel-wise

Laplacian

The gradient $\nabla u : \Omega \to \mathbb{R}^2$ is a 2D-vector field and divergence div operates on 2D-vector fields. Thus we can concatenate these two operators. The result is the *Laplacian*:

$$\Delta u : \Omega \to \mathbb{R}, \quad \Delta u := \operatorname{div}(\nabla u) = \operatorname{div} \begin{pmatrix} \partial_x u \\ \partial_y u \end{pmatrix}$$

$$\Delta u(x, y) = \partial_{xx} u(x, y) + \partial_{yy} u(x, y)$$

There is a physical interpretation of the Laplacian: For example, if $u(x, y)$ denotes the temperature at point (x, y), then $\Delta u(x, y)$ is the rate of local temperature decrease/increase: $\partial_t u(x, y) = a \Delta u(x, y)$ for some constant $a > 0$.

Multi-channel images $u : \Omega \to \mathbb{R}^n$: channel-wise

convolution

Convolution computes a weighted 'sum' of the image values.

 *  = 

 *  = 

convolution

Given a *kernel* $k : \mathbb{R}^2 \to \mathbb{R}$ and an image $u : \Omega \to \mathbb{R}$, the *convolution* between k and u is defined by:

$$k * u : \Omega \to \mathbb{R}^n, \quad (k * u)(x, y) = \int_{\mathbb{R}^2} k(a, b) u(x - a, y - b) \,\mathrm{d}a \,\mathrm{d}b$$

For *multi-channel images* the convolution is computed channel-wise.

definition of u outside of Ω:
To compute a convolution, we need values of u outside of the image domain Ω. There are a few different ways to extend u:

- *clamping* of (x, y) back to Ω (Neumann boundary conditions)
- *periodic* intensity
- *mirrored* intensity

convolution

Given a *kernel* $k : \mathbb{R}^2 \to \mathbb{R}$ and an image $u : \Omega \to \mathbb{R}$, the *convolution* between k and u is defined by:

$$k * u : \Omega \to \mathbb{R}^n, \quad (k * u)(x, y) = \int_{\mathbb{R}^2} k(a, b) u(x - a, y - b) \, \mathrm{d}a \, \mathrm{d}b$$

For *multi-channel images* the convolution is computed channel-wise.

definition of u outside of Ω:

To compute a convolution, we need values of u outside of the image domain Ω. There are a few different ways to extend u:

- *clamping* of (x, y) back to Ω (Neumann boundary conditions)
- *periodic* intensity
- *mirrored* intensity

convolution

Given a *kernel* $k : \mathbb{R}^2 \to \mathbb{R}$ and an image $u : \Omega \to \mathbb{R}$, the *convolution* between k and u is defined by:

$$k * u : \Omega \to \mathbb{R}^n, \quad (k * u)(x, y) = \int_{\mathbb{R}^2} k(a, b) u(x - a, y - b) \, \mathrm{d}a \, \mathrm{d}b$$

For *multi-channel images* the convolution is computed channel-wise.

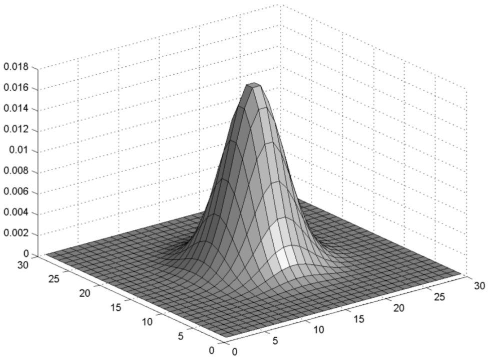
definition of u outside of Ω:
To compute a convolution, we need values of u outside of the image domain Ω. There are a few different ways to extend u:

- *clamping* of (x, y) back to Ω (Neumann boundary conditions)
- *periodic* intensity
- *mirrored* intensity

kernel

2D-Gaussian kernel with standard deviation $\sigma > 0$

$$k(a,b) = G_\sigma(a,b) = \frac{1}{2\pi\sigma^2} \mathrm{e}^{-\frac{a^2+b^2}{2\sigma^2}}$$



Outline

discretization: images

The image domain $\Omega \subset \mathbb{R}^2$ is discretized into a 2D-grid of $w \times h$ pixels.
Caution: in a slight abuse of notation we denote both the continuous image and its discretization with u.

Linearized storage for scalar images $u : \Omega \to \mathbb{R}$
The wh values $u(x, y)$ are arranged into a *single one-dimensional array* u in a row-major order:

$$\begin{aligned} u = \big(&u(0,0), u(1,0), u(2,0), \ldots, u(w-1,0), \\ &u(0,1), u(1,1), u(2,1), \ldots, u(w-1,1), \\ &\ldots, \\ &u(1,h), u(2,h), \ldots, u(w-1,h)\big) \end{aligned}$$

Linearized access

$$u(x, y) = u[x + w \cdot y]$$

discretization: images

The image domain $\Omega \subset \mathbb{R}^2$ is discretized into a 2D-grid of $w \times h$ pixels.
Caution: in a slight abuse of notation we denote both the continuous image and its discretization with u.

Linearized storage for scalar images $u : \Omega \to \mathbb{R}$
The *wh* values $u(x, y)$ are arranged into a *single one-dimensional array* u in a row-major order:

$$
\begin{aligned}
u = \big(& u(0,0), u(1,0), u(2,0), \dots, u(w-1,0), \\
& u(0,1), u(1,1), u(2,1), \dots, u(w-1,1), \\
& \dots, \\
& u(1,h), u(2,h), \dots, u(w-1,h)\big)
\end{aligned}
$$

Linearized access

$$u(x, y) = u[x + w \cdot y]$$

discretization: images

The image domain $\Omega \subset \mathbb{R}^2$ is discretized into a 2D-grid of $w \times h$ pixels.
Caution: in a slight abuse of notation we denote both the continuous image and its discretization with u.

Linearized storage for scalar images $u : \Omega \to \mathbb{R}$
The *wh* values $u(x, y)$ are arranged into a *single one-dimensional array* u in a row-major order:

$$
\begin{aligned}
u = \big(&u(0,0), u(1,0), u(2,0), \ldots, u(w-1,0),\\
&u(0,1), u(1,1), u(2,1), \ldots, u(w-1,1),\\
&\ldots,\\
&u(1,h), u(2,h), \ldots, u(w-1,h)\big)
\end{aligned}
$$

Linearized access

$$u(x, y) = u[x + w \cdot y]$$

discretization images

Linearized storage of multi-channel images $u : \Omega \to \mathbb{R}^n$
The *nwh* values $u_i(x, y)$ are arranged into a single one-dimensional array.
The n channels u_i are stored directly one after another

$$u = (u_1, u_2, ..., u_n).$$

and, as previously, each channel u_i is stored in row-major order.

This is called *layered storage* in contrast to *interleaved storage*, where one save thes n values $u_i(x, y)$ pixel-by-pixel.

Linearized access for layered storage

$$u_i(x, y) = u[x + w \cdot y + wh \cdot i]$$

discretization images

Linearized storage of multi-channel images $u : \Omega \to \mathbb{R}^n$
The *nwh* values $u_i(x, y)$ are arranged into a single one-dimensional array.
The n channels u_i are stored directly one after another

$$u = (u_1, u_2, ..., u_n).$$

and, as previously, each channel u_i is stored in row-major order.

This is called *layered storage* in contrast to *interleaved storage*, where one save thes n values $u_i(x, y)$ pixel-by-pixel.

Linearized access for layered storage

$$u_i(x, y) = u[x + w \cdot y + wh \cdot i]$$

discretization images

Linearized storage of multi-channel images $u : \Omega \to \mathbb{R}^n$
The *nwh* values $u_i(x, y)$ are arranged into a single one-dimensional array.
The n channels u_i are stored directly one after another

$$u = (u_1, u_2, ..., u_n).$$

and, as previously, each channel u_i is stored in row-major order.

This is called *layered storage* in contrast to *interleaved storage*, where one save thes n values $u_i(x, y)$ pixel-by-pixel.

Linearized access for layered storage

$$u_i(x, y) = u[x + w \cdot y + wh \cdot i]$$

partial derivatives and gradient

In the discrete setting, we approximate the partial derivatives using *forward differences* with Neumann boundary conditions

$$\partial_x^+ u(x,y) = \begin{cases} u(x+1,y) - u(x,y), & \text{if } x+1 < w \\ 0, & \text{else} \end{cases}$$

$$\partial_y^+ u(x,y) = \begin{cases} u(x,y+1) - u(x,y), & \text{if } y+1 < h \\ 0, & \text{else} \end{cases}$$

Thus we obtain a discretization of the *gradient*:

$$\nabla^+ u(x,y) = \begin{pmatrix} \partial_x^+ u(x,y) \\ \partial_y^+ u(x,y) \end{pmatrix}$$

partial derivatives and gradient

In the discrete setting, we approximate the partial derivatives using *forward differences* with Neumann boundary conditions

$$\partial_x^+ u(x,y) = \begin{cases} u(x+1,y) - u(x,y), & \text{if } x+1 < w \\ 0, & \text{else} \end{cases}$$

$$\partial_y^+ u(x,y) = \begin{cases} u(x,y+1) - u(x,y), & \text{if } y+1 < h \\ 0, & \text{else} \end{cases}$$

Thus we obtain a discretization of the *gradient*:

$$\nabla^+ u(x,y) = \begin{pmatrix} \partial_x^+ u(x,y) \\ \partial_y^+ u(x,y) \end{pmatrix}$$

rotationally robust gradient

A *more rotationally robust* discretization of the partial derivatives:

$$\partial_x^r(x,y) = \frac{1}{32}\big(3u(x+1,y+1) + 10u(x+1,y) + 3u(x+1,y-1) \\ - 3u(x-1,y+1) - 10u(x-1,y) - 3u(x-1,y-1)\big)$$

$$\partial_y^r(x,y) = \frac{1}{32}\big(3u(x+1,y+1) + 10u(x,y+1) + 3u(x-1,y+1) \\ - 3u(x+1,y-1) - 10u(x,y-1) - 3u(x-1,y-1)\big)$$

If values $u(x,y)$ in pixels outside of Ω are needed, clamp (x,y) back to Ω.

rotationally robust gradient

A *more rotationally robust* discretization of the partial derivatives:

$$\begin{aligned}\partial_x^r(x,y) =& \frac{1}{32}\big(3u(x+1,y+1) + 10u(x+1,y) + 3u(x+1,y-1)\\ &- 3u(x-1,y+1) - 10u(x-1,y) - 3u(x-1,y-1)\big)\end{aligned}$$

$$\begin{aligned}\partial_y^r(x,y) =& \frac{1}{32}\big(3u(x+1,y+1) + 10u(x,y+1) + 3u(x-1,y+1)\\ &- 3u(x+1,y-1) - 10u(x,y-1) - 3u(x-1,y-1)\big)\end{aligned}$$

If values $u(x,y)$ in pixels outside of Ω are needed, clamp (x,y) back to Ω.

rotationally robust gradient

A *more rotationally robust* discretization of the partial derivatives:

$$\partial_x^r(x,y) = \frac{1}{32}\big(3u(x+1,y+1) + 10u(x+1,y) + 3u(x+1,y-1) \\ - 3u(x-1,y+1) - 10u(x-1,y) - 3u(x-1,y-1)\big)$$

$$\partial_y^r(x,y) = \frac{1}{32}\big(3u(x+1,y+1) + 10u(x,y+1) + 3u(x-1,y+1) \\ - 3u(x+1,y-1) - 10u(x,y-1) - 3u(x-1,y-1)\big)$$

If values $u(x,y)$ in pixels outside of Ω are needed, clamp (x,y) back to Ω.

divergence

We discretize the divergence using *backward differences*:

$$\operatorname{div}^- u(x,y) = \partial_x^- u_1(x,y) + \partial_y^- u_2(x,y)$$

With the backward differences ∂_x^- and ∂_y^- defined as:

$$\partial_x^- u(x,y) = \begin{cases} u(x,y) - u(x-1,y), & \text{if } x > 0 \\ 0, & \text{else} \end{cases}$$

$$\partial_y^- u(x,y) = \begin{cases} u(x,y) - u(x,y-1), & \text{if } y > 0 \\ 0, & \text{else} \end{cases}$$

divergence

We discretize the divergence using *backward differences*:

$$\operatorname{div}^- u(x,y) = \partial_x^- u_1(x,y) + \partial_y^- u_2(x,y)$$

With the backward differences ∂_x^- and ∂_y^- defined as:

$$\partial_x^- u(x,y) = \begin{cases} u(x,y) - u(x-1,y), & \text{if } x > 0 \\ 0, & \text{else} \end{cases}$$

$$\partial_y^- u(x,y) = \begin{cases} u(x,y) - u(x,y-1), & \text{if } y > 0 \\ 0, & \text{else} \end{cases}$$

Laplacian

Using the discretizations ∇^+ and div^- we obtain a discretization of the *Laplacian*:

$$\Delta u = \operatorname{div}^-(\nabla^+ u) = \partial_x^-(\partial_x^+ u) + \partial_y^-(\partial_y^+ u)$$

One can check that

$$\begin{aligned}\Delta u(x,y) =& \mathbf{1}_{x+1<w} \cdot u(x+1,y) + \mathbf{1}_{x>0} \cdot u(x-1,y) \\ &+ \mathbf{1}_{y+1<h} \cdot u(x,y+1) + \mathbf{1}_{y>0} \cdot u(x,y-1) \\ &- (\mathbf{1}_{x+1<w} + \mathbf{1}_{x>0} + \mathbf{1}_{y+1<h} + \mathbf{1}_{y>0}) \cdot u(x,y),\end{aligned}$$

where we define (and similarly for other factors):

$$\mathbf{1}_{x+1<w} = \begin{cases} 1, & \text{if } x+1<w, \\ 0, & \text{otherwise.} \end{cases}$$

Laplacian

Using the discretizations ∇^+ and div^- we obtain a discretization of the *Laplacian*:

$$\Delta u = \mathrm{div}^-(\nabla^+ u) = \partial_x^-(\partial_x^+ u) + \partial_y^-(\partial_y^+ u)$$

One can check that

$$\begin{aligned}\Delta u(x,y) =& \mathbf{1}_{x+1<w} \cdot u(x+1,y) + \mathbf{1}_{x>0} \cdot u(x-1,y) \\ &+ \mathbf{1}_{y+1<h} \cdot u(x,y+1) + \mathbf{1}_{y>0} \cdot u(x,y-1) \\ &- (\mathbf{1}_{x+1<w} + \mathbf{1}_{x>0} + \mathbf{1}_{y+1<h} + \mathbf{1}_{y>0}) \cdot u(x,y),\end{aligned}$$

where we define (and similarly for other factors):

$$\mathbf{1}_{x+1<w} = \begin{cases} 1, & \text{if } x+1<w, \\ 0, & \text{otherwise.} \end{cases}$$

Laplacian

Using the discretizations ∇^+ and div^- we obtain a discretization of the *Laplacian*:

$$\Delta u = \mathrm{div}^-(\nabla^+ u) = \partial_x^-(\partial_x^+ u) + \partial_y^-(\partial_y^+ u)$$

One can check that

$$\begin{aligned}\Delta u(x,y) =& \mathbf{1}_{x+1<w} \cdot u(x+1,y) + \mathbf{1}_{x>0} \cdot u(x-1,y) \\ &+ \mathbf{1}_{y+1<h} \cdot u(x,y+1) + \mathbf{1}_{y>0} \cdot u(x,y-1) \\ &- (\mathbf{1}_{x+1<w} + \mathbf{1}_{x>0} + \mathbf{1}_{y+1<h} + \mathbf{1}_{y>0}) \cdot u(x,y),\end{aligned}$$

where we define (and similarly for other factors):

$$\mathbf{1}_{x+1<w} = \begin{cases} 1, & \text{if } x+1<w, \\ 0, & \text{otherwise.} \end{cases}$$

convolution

discretization

Let S_k be the *support* of k, that is positions (a, b) with $k(a, b) \neq 0$. Thus we write the convolution in the discrete setting as:

$$(k * u)(x, y) = \sum_{(a,b) \in S_k} k(a, b) \cdot u(x - a, y - b).$$

windowing

Often, the support of k lies within a *small window* of size $(2r_x + 1) \times (2r_y + 1)$. In this case we have:

$$(k * u)(x, y) = \sum_{a=-r_x}^{r_x} \sum_{b=-r_y}^{r_y} k(a, b) \cdot u(x - a, y - b).$$

convolution

discretization
Let S_k be the *support* of k, that is positions (a, b) with $k(a, b) \neq 0$. Thus we write the convolution in the discrete setting as:

$$(k * u)(x, y) = \sum_{(a,b) \in S_k} k(a, b) \cdot u(x - a, y - b).$$

windowing
Often, the support of k lies within a *small window* of size $(2r_x + 1) \times (2r_y + 1)$. In this case we have:

$$(k * u)(x, y) = \sum_{a=-r_x}^{r_x} \sum_{b=-r_y}^{r_y} k(a, b) \cdot u(x - a, y - b).$$