GPU Programming in Computer Vision

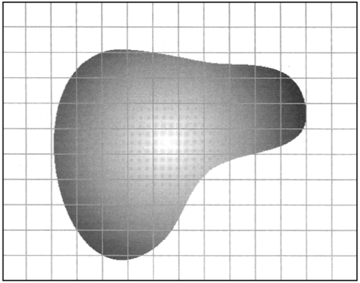
Summer Semester 2015

Thomas Möllenhoff, Robert Maier, Caner Hazirbas

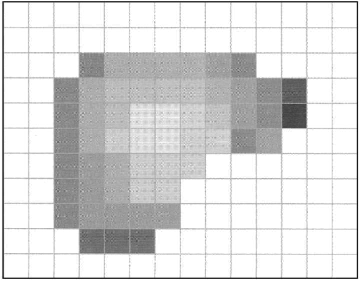
Continuous Setting

Continuous setting
We view images as being defined on a *continuous* demain Ω.
Images are *functions*

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



continuous setting



discrete setting

Representing Images as Functions

Image are functions

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

Domain Ω (a rectangular subset of $\mathbb{R}^d$)
$\Omega \subset \mathbb{R}^1$: signal (1D)
$\Omega \subset \mathbb{R}^2$: image (2D)
$\Omega \subset \mathbb{R}^3$: volume (3D)

Range $\mathbb{R}^n$
$\mathbb{R}^1$: grayscale images, ...
$\mathbb{R}^2$: 2D-vector fields, ...
$\mathbb{R}^3$: RGB images, HSV values, normals, ...
$\mathbb{R}^4$: matrix valued images, ...

We will represent *multi-channel* images by *n single-valued images*:

$$u = (u_1, \dots, u_n), \qquad u(x) = \big(u_1(x), \dots, u_n(x)\big) \in \mathbb{R}^n$$

Differential Operators

We assume a two-dimensional domain: $\Omega \subset \mathbb{R}^2$.

Partial derivative w.r.t. x **of a scalar image** $u : \Omega \to \mathbb{R}$

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

Partial derivative w.r.t. y **of a scalar image** $u : \Omega \to \mathbb{R}$

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

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

Differential Operators

Gradient of a scalar image $u : \Omega \to \mathbb{R}$
The gradient combines all partial derivatives into a vector:

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

This vector is the direction of the *fastest increase* of u.

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

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

Differential Operators

Divergence of a 2D-vector field $u : \Omega \to \mathbb{R}^2$
This operator needs a vector field as input. The result is a scalar function:

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

Multi-channel 2D-vector fields $u : \Omega \to (\mathbb{R}^2)^n$*:* Divergence per channel:

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

Differential Operators

Gradient magnitude of a scalar image
Pointwise absolute value of ∇u: $|\nabla u| : \Omega \to \mathbb{R}$,

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

This often serves as an edge detector: big values $|(\nabla u)(x, y)|$ indicate an edge at (x, y).

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

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

Differential Operators

Laplacian of a scalar image $u : \Omega \to \mathbb{R}$
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*:

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

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

The Laplacian is useful in *physical models*. For example, if $u(x, y)$ is the temperature at each point (x, y), then $\Delta\, u$ is the rate of local temperature decrease: $(\partial_t u)(x, y) = a(\Delta\, u)(x, y)$ for some $a > 0$.

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

Convolution

Convolution computes a weighted sum of the image values.

Convolution

Convolution
Given a *kernel* $K : \mathbb{R}^2 \to \mathbb{R}$ and a multi-channel image $u : \Omega \to \mathbb{R}^n$:

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

(channel-wise). This sums up the u values around (x, y), weighted by K.

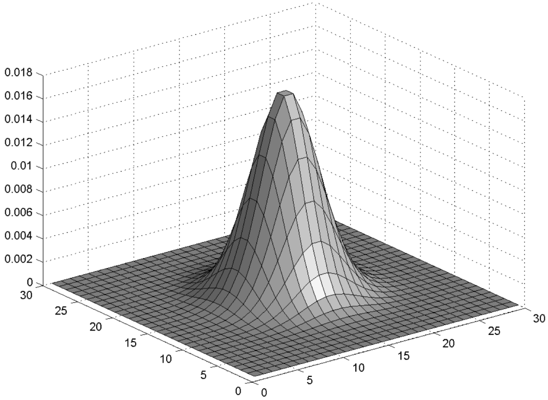
Definition at the boundary of image domain
The formula needs values of u outside of the definition domain Ω.
Common ways to resolve this:

- *Clamping* of (x, y) back to Ω (we will use this approach)
- *Periodic* boundary conditions (allows application of FFT)
- *Mirroring* boundary conditions

Convolution

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

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



Discretization: Images

The image domain $\Omega \subset \mathbb{R}^2$ is discretized into a 2D-grid of $W \times H$ pixels.

Linearized storage for scalar images $u : \Omega \to \mathbb{R}$
The WH values $u(x, y)$ are arranged as a *single one-dimensional array* u.
Usually, one uses a *row-by-row* order:

$$u = \Big(u(0,0),\ u(1,0),\ u(2,0),\ \ldots,\ u(W-1,0),\\ u(0,1),\ u(1,1),\ u(2,1),\ \ldots,\ u(W-1,1),\quad \ldots,\\ u(0,H-1),\ u(1,H-1),\ u(2,H-1),\ \ldots,\ u(W-1,H-1)\Big).$$

Linearized access

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

Discretization: Images

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

$$u = (u_1, u_2, \ldots, u_n)$$

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

This is called *layered* storage, and we will use this variant.
(Another possiblity is interleaved storage: save the n values $u_i(x, y)$ pixel-by-pixel. For example, this is used by OpenCV.)

Linearized access

$$u_i(x, y) = u\big[x + W \cdot y + WH \cdot i\big]$$

C/C++
To support potentially *very large* images, *always* compute the products using the `size_t` type: `x + (size_t)W*y + (size_t)W*H*i`.

Discretization: Differential Operators

Gradient

Forward differences:

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

Forward differences (with Neumann boundary conditions)

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

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

This assumes that u has slope 0 at the boundary: $\partial_{\text{normal}_\Omega} u = 0$.

Discretization: Differential Operators

Divergence
Backward differences:

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

Backward differences (with Dirichlet boundary conditions)

$$(\partial_x^{-} u)(x,y) := \left\{\begin{matrix} u(x,y) & \text{if } x+1 < W \\ 0 & \text{else} \end{matrix}\right\} - \left\{\begin{matrix} u(x-1,y) & \text{if } x > 0 \\ 0 & \text{else} \end{matrix}\right\}$$

$$(\partial_y^{-} u)(x,y) := \left\{\begin{matrix} u(x,y) & \text{if } y+1 < H \\ 0 & \text{else} \end{matrix}\right\} - \left\{\begin{matrix} u(x,y-1) & \text{if } y > 0 \\ 0 & \text{else} \end{matrix}\right\}$$

This assumes that u has zero values at the boundary.

Discretization: Differential Operators

Laplacian

According to ∇^+ and div^-:

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

This means

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

Here we define (and similarly for other factors):

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

Only compute $u(x+1,y)$ etc. if its factor is not zero!

Discretization: Differential Operators

Gradient

A *more rotationally invariant* discretization:

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

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

Neumann boundary conditions

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

Discretization: Convolution

Discretization

Finite weighted sum:

$$(K * u)(x, y) = \sum_{(a,b)\in S_K} K(a, b) \cdot u(x - a, y - b)$$

Windowing

S_K is the *support* of K: positions (a, b) with $K(a, b) \neq 0$.
It is assumed to lie entirely in a *small window* of size $(2r_x + 1) \times (2r_y + 1)$:

$$(K * I)(x, y) = \sum_{a=-r_x}^{r_x} \sum_{b=-r_y}^{r_y} K(a, b)\, u(x - a, y - b)\, da\, db$$

Discretized kernel

One often deals with small-support kernels K,
or the kernel is truncated artificially (e.g. *Gaussian kernel*).
Discretized K is stored row-by-row: $K(x, y) = K[x + (2r_x + 1) \cdot y]$.