Practical Course: GPU Programming in Computer Vision

Mathematics 3: Diffusion

Björn Häfner, Benedikt Löwenhauser, Thomas Möllenhoff

Technische Universität München
Department of Informatics
Computer Vision Group

Summer Semester 2017
September 11 - October 8

Image Evolutions

Image evolutions

Consider images which evolve over time

$$u : \Omega \subset \mathbb{R}^2 \times [0, T] \to \mathbb{R}^n, \tag{1}$$

i.e. the image u is now depending on three parameters (x, y, t) and with indicate this writing $u(x, y, t)$.

Discretized view

Generate a sequence of images $u^k : \Omega \to \mathbb{R}^n$ for $k = 0, \ldots, n$:

$$u^0, u^1, u^2, u^3, \ldots \tag{2}$$

Example

Algorithm generating an image sequence where each newly generated image u^{k+1} is based on the previous image(s) $u^k, u^{k-1}, \ldots$

Diffusion

as Image Evolution

For simplicity let us assume $u : \Omega \subset \mathbb{R}^2 \times [0, T] \to \mathbb{R}^n$ are grayscale images, i.e. $n = 1$. Later we will generalize to multi-channel images, i.e. $n \geq 1$.

Diffusion

Continuous time update equation

$$\partial_t u = \operatorname{div}\left(D \nabla_x u\right), \tag{3}$$

where ∇_x is the gradient operator, but only w.r.t. the spatial variables x, y.

$D : \Omega \times [0, T] \to \mathbb{R}^{2\times 2}$ is called the *diffusion tensor* and is a symmetric, positive definite matrix. Note that D is depending on (x, y, t), i.e. $D = D(x, y, t)$. Thus, $D(x, y, t)$ may be different $\forall (x, y, t)$.

Diffusion

Intuition

$$\partial_t u = \text{div}\left(D \nabla_x u\right), \tag{3}$$

Diffusion tries to locally cancel out any existing color differences, thus the evolution of the image u gradually becomes more and more smooth while time passes. Varying D one can change how the image will be smoothed. More about this later...

Diffusion

Discretization and Implementation steps

Temporal derivative

Use forward differences ∂_t^+ with a time step $\tau > 0$

$$\left(\partial_t^+ u\right)(x, y, t) = \frac{u(x, t, t+\tau) - u(x, y, t)}{\tau} \tag{4}$$

Spatial derivative

Use forward differences for ∇_x and backward differences for div,

$$\operatorname{div}^- \left(D \nabla_x^+ u\right) = \partial_x^- \left(D_{11} \partial_x^+ u + D_{12} \partial_y^+ u\right) + \partial_y^- \left(D_{21} \partial_x^+ u + D_{22} \partial_y^+ u\right). \tag{5}$$

Diffusion

Discretization and Implementation steps

Thus, using (4) and (3) we get the **Gradient Descent algorithm**

$$u(x, t, t+\tau) = u(x, y, t) + \tau \operatorname{div}\left(D \nabla_x u(x, y, t)\right). \qquad (6)$$

Implementation steps

1. Compute the gradient of u per pixel: $G = \nabla_x^+ u \in \mathbb{R}^2$
2. Compute the diffusion tensor per pixel: $D = \cdots \in \mathbb{R}^{2\times 2}$
3. Compute the product of D with G per pixel: $P = D \cdot G \in \mathbb{R}^2$
4. Compute the divergence P per pixel: $d = \operatorname{div}^-(P) \in \mathbb{R}$
5. Use d to calculate (6) per pixel: $u^{t+1} = u^t + \tau \cdot d$

Diffusion

Diffusion Tensor

Depending how D is chosen, different types of diffusion can occur

1. Linear or non-linear
   - Linear: D does not depend on u
   - Non-linear: D does depend on u
2. Isotropic or Anisotropic
   - Isotropic: D is a multiple of the identity matrix, i.e.

     $$\exists g(x, y, t) \in \mathbb{R} : D(x, y, t) = g(x, y, t) \cdot \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

   - Anisotropic: Any diffusion which is not isotropic.

Each diffusion is either linear or nonlinear, and either isotropic or anisotropic:

	isotropic	anisotropic
linear	linear isotropic	linear anisotropic
non-linear	non-linear isotropic	non-linear anisotropic

Diffusion

Diffusion Tensor

Isotropic diffusion spreads the values u equally in *x*- and *y*-direction.

- Laplace diffusion
- Huber diffusion

Whereas anisotropic diffusion can selectively suppress information flow in certain directions

- Structure Tensor Diffusion (only smooth *u* along potential edges, and not across.)

Diffusion

Laplace Diffusion

Using

$$D_L(x, y, t) := \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \tag{7}$$

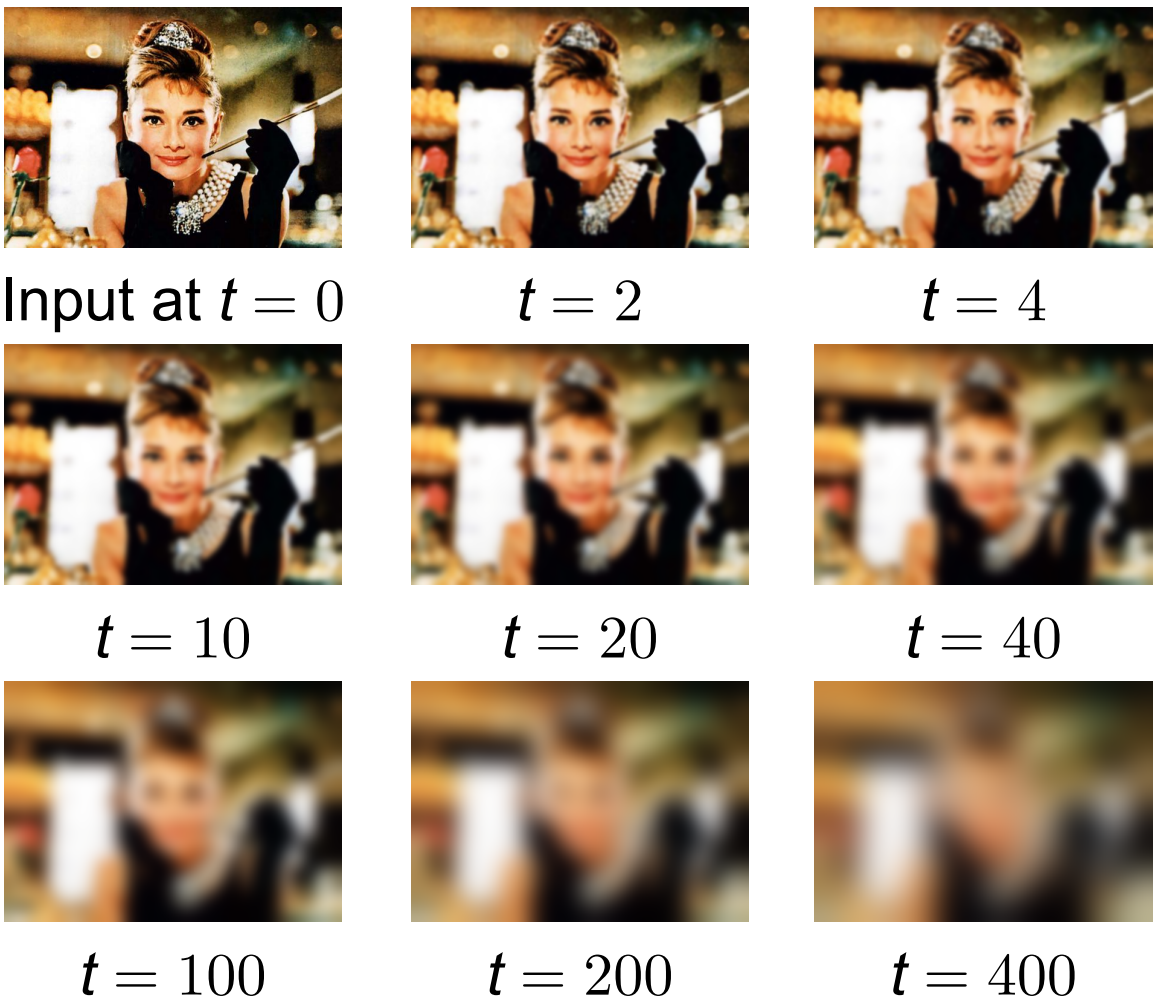
results in so called *Laplace Diffusion*. It is linear isotropic diffusion and the resulting equation is

$$\partial_t u = \operatorname{div}(\nabla_x u) = \Delta u \tag{8}$$

It has the effect of blurring the input image similar to a Gaussian convolution.

Diffusion

Laplace Diffusion



Diffusion

Huber Diffusion

Using

$$D_H(x, y, t) := g(x, y, t) \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \tag{9}$$

with

$$g(x, y, t) := \frac{1}{\max\left(\varepsilon, \left\|\nabla u\left(x, y, t\right)\right\|\right)}. \tag{10}$$

results in so called *Huber Diffusion*. It is non-linear isotropic diffusion as g is depending on u and the resulting equation is

$$\partial_t u = \operatorname{div}\left(g(x, y, t) \nabla_x u\right) \tag{11}$$

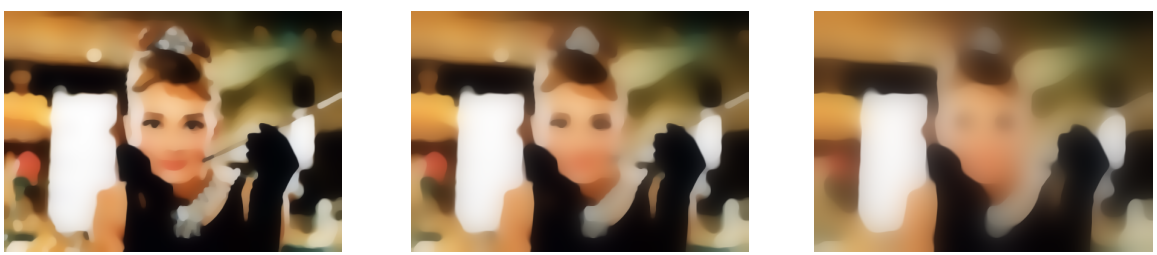
It has the effect of smoothing the input image where no large gradients of u occur and smooths less where large gradients of u occur.

Diffusion

Huber Diffusion

Input at $t = 0$ $t = 2$ $t = 4$

$t = 10$ $t = 20$ $t = 40$



$t = 100$ $t = 200$ $t = 400$

Diffusion

Structure Tensor Diffusion

Using

$$D_S(x, y, t) := G(x, y, t) = \mu_1 e_1 e_1^T + \mu_2 e_2 e_2^T \in \mathbb{R}^{2\times 2}, \tag{12}$$

with e_1 and e_2 being the eigenvectors of the structure tensor (cp. yesterdays maths slides) and

$$\mu_1 = \alpha, \tag{13}$$

$$\mu_2 = \alpha + (1-\alpha)\exp\left(-\frac{C}{(\lambda_1-\lambda_2)^2}\right), \tag{14}$$

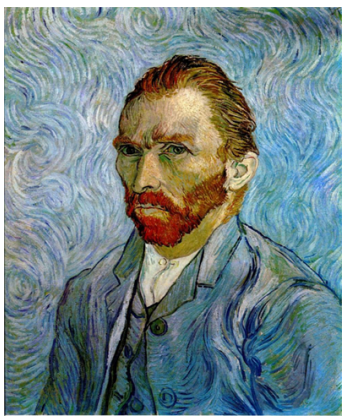
$\alpha \in (0,1)$, $C > 0$, results in a linear anisotropic diffusion with

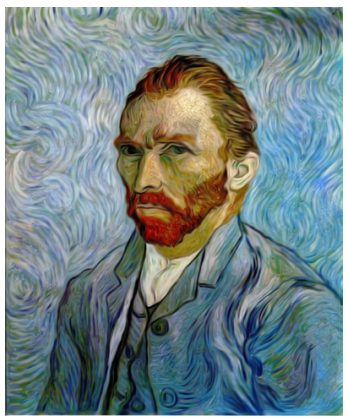
$$\partial_t u = \operatorname{div}\left(G(x, y, t)\nabla_x u\right) \tag{15}$$

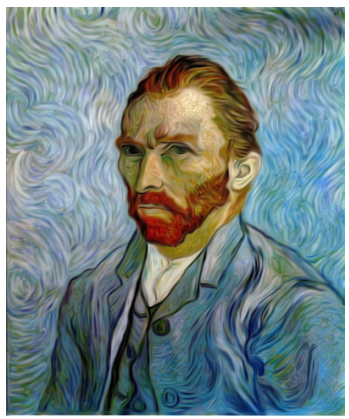
It has the effect of smoothing along the direction of the image structures.

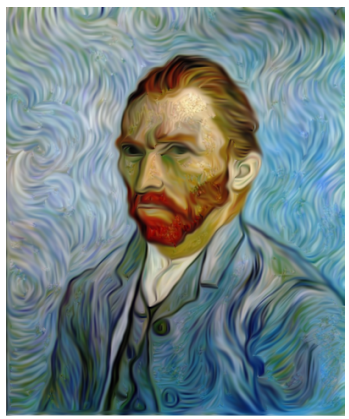
Diffusion

Structure Tensor Diffusion


Input at $t = 0$


$t = 2$


$t = 4$


$t = 10$


$t = 20$

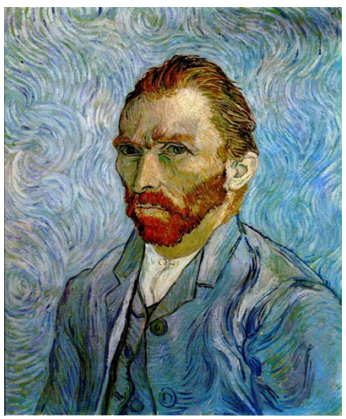

$t = 40$


$t = 100$


$t = 200$

Diffusion

Laplace Diffusion


Input at $t = 0$

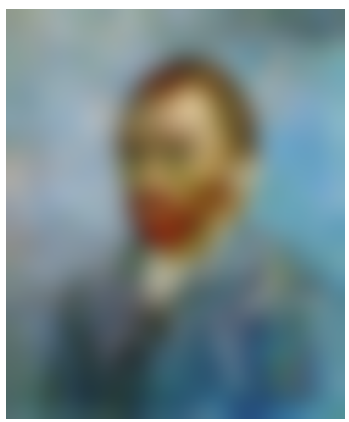

$t = 2$

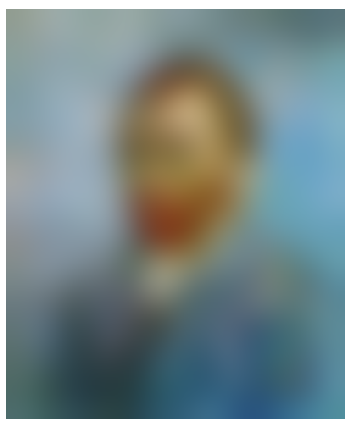

$t = 4$


$t = 10$


$t = 20$


$t = 40$


$t = 100$


$t = 200$