Gradient Domain Imaging

Final Presentation

Submitted by:
Dolniak, Oliver / 220678 / Informatik DIP
Klaghstan, Merza / 323179 / Informatik MA

Tutor: Mathias Eitz

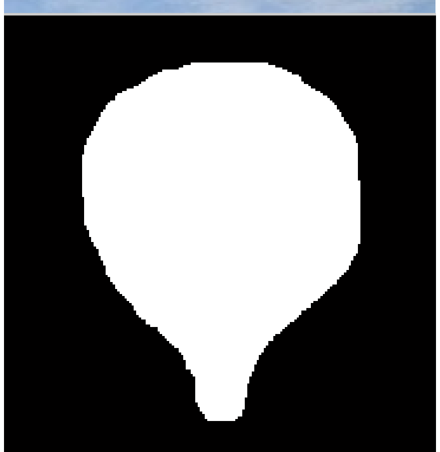
TU Berlin
Department of Computer Graphics
Winter Semester 2010-2011

Achievement

- Demo ..
  - Balloon
  - House

What's going on !!

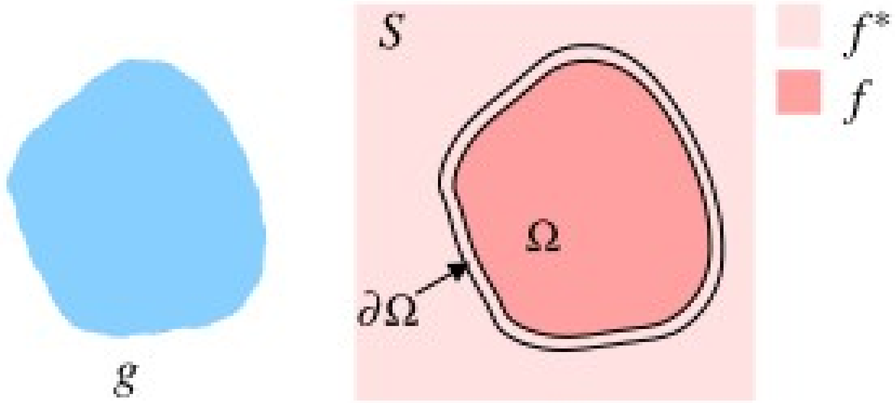
- Simply: replace N pixels from the target, with N *processed* ones from source



In Details ..

- f* known function in target in domain S
- f unknown function in target

  in domain Ω
- g source
- ∂Ω boundary



- Solution idea is to find f whose gradient is the closest to G; gradient of source g .. mathematically formulated :

$$min \iint ||\nabla f - G||^2$$

- Whose solution is the solution of Poisson equation

$$\Delta f = div\ G = \Delta g \qquad (*)$$

Poisson Equation

- $div\ G$ is the Divergence
  - Property of gradient field
  - $div\ G = \partial Gx/\partial x + \partial Gy/\partial y$
- Δ is the Laplacian operator

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

- Condition : Dirichlit-boundary : known pixels at boundaries
  - Fulfilled
  - Pixels at boundaries are read from target

Linear Equation

- Applying to (*) equation

$$-4\, f(x,y) + f(x-1,y) + f(x,y-1) + f(x+1,y) + f(x,y+1) = div\ G$$

- We turn into a classical linear equation

$$Ax = b$$

- # Unknowns = N pixels
- A: NxN matrix , b: N-elements vector
- Problem
  - Build A
  - Build b

Vector b

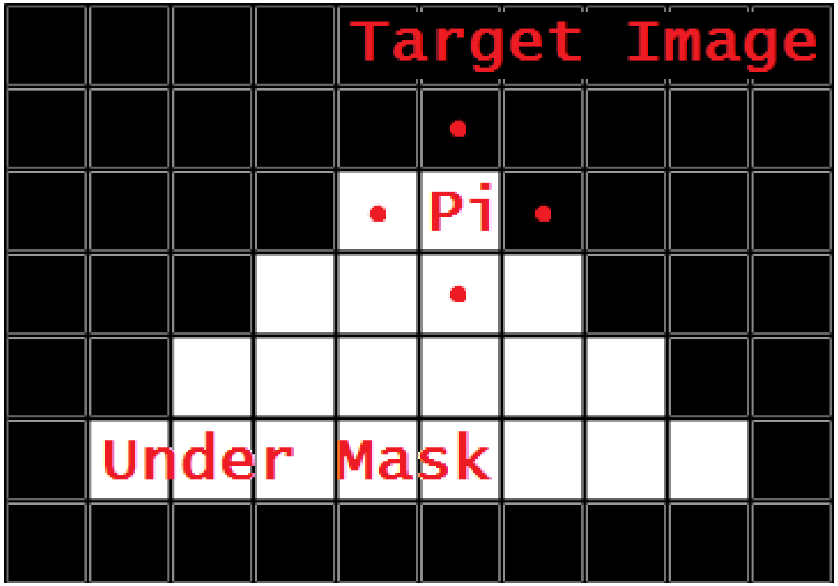
- Suppose Pi denotes the unknown pixel (i) ; i = 1 .. N
- And Si denotes the corresponding pixel (i) in source image
- *b[i] = div (G(Si)) + Neighbor(Pi)*
- Where *Neighbor(Pi)* is

  subset of the 4 neighbors of Pi

  (top, bottom, right, left)

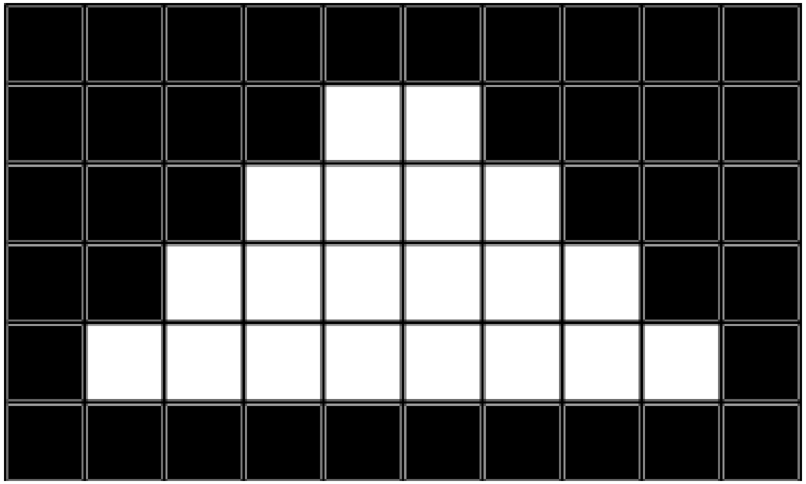
  that belongs to the boundary.

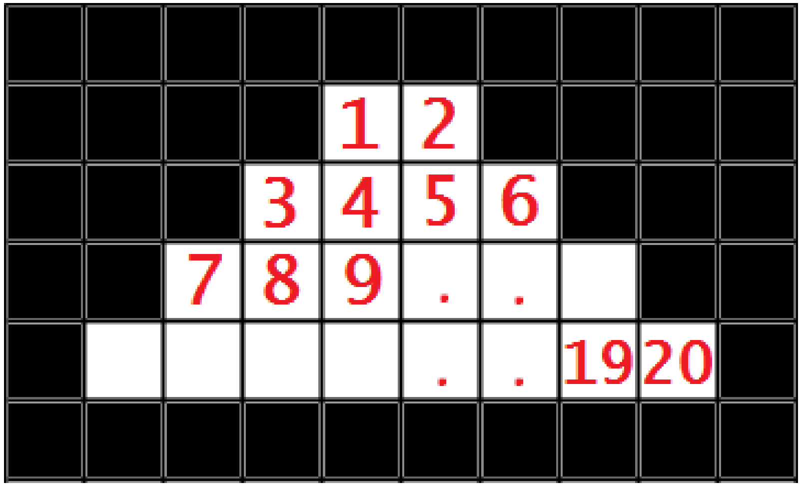
  Its value is read from target.



Matrix A

- First, we build an index for pixels under mask
    - Numerating pixels



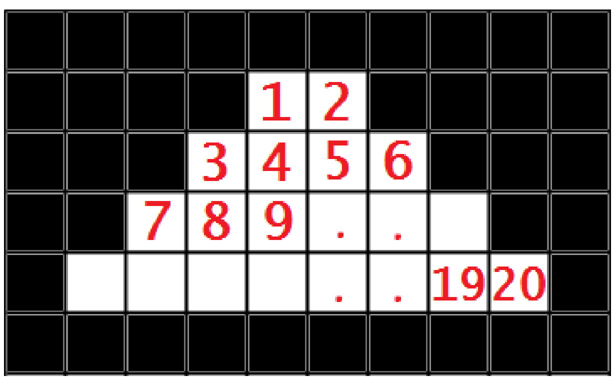


Matrix A

- Then we build an NxN matrix A as follows :

  for each pixel denoted by row number

  - Put -4 in its column
  - Put 1 in columns referring to its neighbors



	1	2	3	4	5	6	7	8	9	..	20
1	-4	1	0	1	0	0	0	0	0	..	0
2	1	-4	0	0	1	0	0	0	0	..	0
3	0	0	-4	1	0	0	0	1	0	..	0
4	1	0	1	-4	1	0	0	0	1	..	0
..	..	..	..	..	..	..	..	..	..	..	..
20	0	0	0	0	0	0	0	0	0	..	-4

- So, we get a symmetric matrix with diagonal = -4

Matrix A

- Problem, A is huge
- Usually number of pixels to be processed is about 30,000

  => A : 30,000 x 30,000 = 900,000,000 elements
- How to store and deal with it ?!
- Notice that A is a sparse matrix
  - populated primarily with zeros
- Benefit from *sparse* notation in Matlab!
  - Sparse matrix is stored using Coordinated-list compression method

Coordinated List

- Represent a sparse matrix (A) with 3 vectors referring to non-zero elements
  - *Row* : row index
  - *Column* : column index
  - *Value* : value at the index A(row, column)
- Example :

Row = [1, 1, 1, 2, 2, ..]

Col = [1, 2, 4, 1, 2, ..]

Val = [-4, 1, 1, 1, -4, ..]

	1	2	3	4	5	6	..	20
1	-4	1	0	1	0	0	..	0
2	1	-4	0	0	1	0	..	0
..	..	..	..	..	..	..	..	..
20	0	0	0	0	0	0	..	-4

- Number of non-zero elements = O(5xN) : pixel itself + 4 neighbors

  => store 3x5xN instead of NxN

Equation Solve

- Once A and b are there, x can be easily computed

$$x_k = A \setminus b_k \, ; \, k = R, G \; and \; B$$

- Combine the 3 channels and re-arrange the vector into mask-form

Examples

Examples

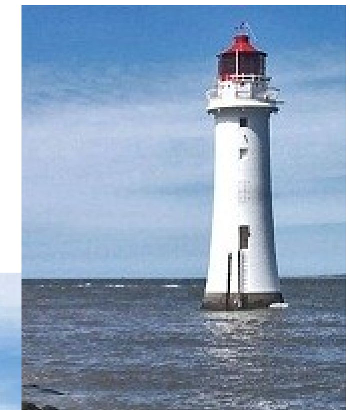
Examples

Examples

Examples

Examples

Examples



Examples

Thank you !