Самая лучшая лекция: вариационные методы

Вычисления на видеокартах. Лекция 9

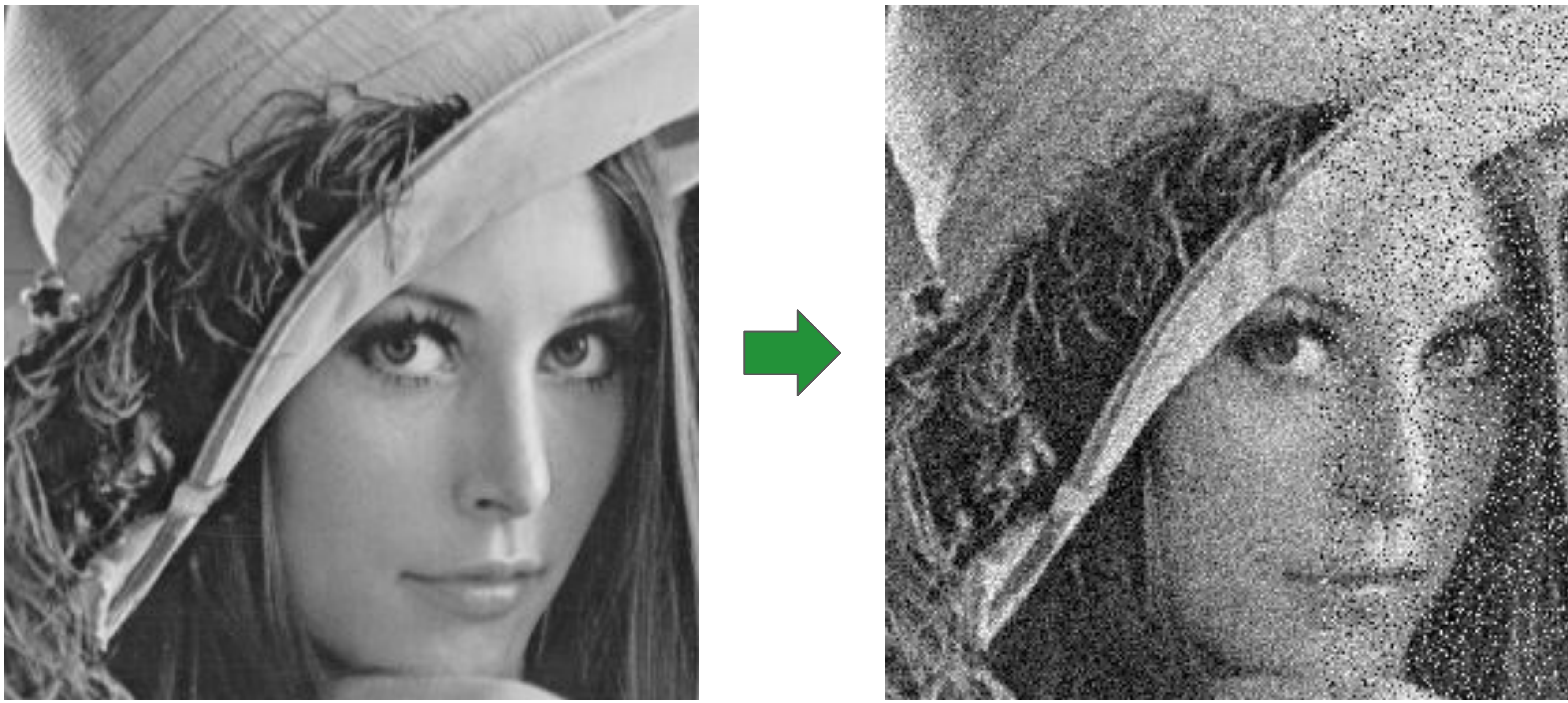
1. Image Denoising via Total Variation Minimization (TV-L2, TV-L1)
2. Image Super Resolution (TV-L1, Huber model)
3. 2.5D Surface Reconstruction (TGV-Huber model)
4. 3D Surface Reconstruction on regular grid (TV-L1)
5. 3D Surface Reconstruction on adaptive octree (TV-L1)

Полярный Николай

polarnick239@gmail.com

Пусть есть шумное изображение

- Гауссовый шум по всей поверхности изображения
- Salt-and-pepper шум на правой части изображения



ROF Image Denoising (TV-L2)

Rudin-Osher-Fatemi model (ROF):

$$\min_x \| \nabla x \| + \frac{\lambda}{2} \| x - f \|^2$$

Где

- f - наблюдаемое изображение с шумом
- x - искомое очищенное от шума изображение
- $\| \nabla x \|$ - полная вариация (регуляризация)
- $\| x - f \|^2$ - L^2 тяготение к данным (наблюдаемому f)
- λ - параметр регуляризации

Как найти x?

Primal-Dual algorithm

Primal-Dual алгоритм минимизации в общем виде решает:

$$\min_x F(Kx) + G(x)$$

Где: F, G - выпуклые функции, K - линейный оператор.

Тогда итеративная схема (модель - выпуклая):

$$\begin{cases} p_{n+1} = (I + \sigma\partial F^*)^{-1}(p_n + \sigma K x_n) \\ \hat{x}_{n+1} = (I + \tau\partial G)^{-1}(x_n - \tau K^T p_{n+1}) \\ x_{n+1} = \hat{x}_{n+1} + \theta(\hat{x}_{n+1} - x_n) \end{cases}$$

Primal-Dual algorithm (TV-L2, ROF)

Primal-Dual алгоритм минимизации в общем виде решает:

$$\min_x F(Kx) + G(x)$$

Где: F, G - выпуклые функции, K - линейный оператор.

Тогда итеративная схема (модель - выпуклая):

$$\begin{cases} p_{n+1} = (I + \sigma\partial F^*)^{-1}(p_n + \sigma K x_n) \\ \hat{x}_{n+1} = (I + \tau\partial G)^{-1}(x_n - \tau K^T p_{n+1}) \\ x_{n+1} = \hat{x}_{n+1} + \theta(\hat{x}_{n+1} - x_n) \end{cases}$$

$$(I + \sigma\partial F^*)^{-1}(p) = \mathbf{project}_P(p)$$

$$(I + \tau\partial G_{ROF})^{-1}(x) = \frac{x + \lambda\tau f}{1 + \lambda\tau}$$

$$\mathbf{project}_P(p) = \frac{p}{max(\|p\|, 1)}$$

$$K x_n = \nabla x_n$$

$$K^T p_{n+1} = \nabla^T p_{n+1}$$

Подробнее: [IPython notebook with ROF and TVL1 denoising (with math!)]

ROF Image Denoising (TV-L2)

Rudin-Osher-Fatemi model (ROF):

$$\min_x \| \nabla x \| + \frac{\lambda}{2} \| x - f \|^2$$

$\lambda = 4$

$\lambda = 8$

Image Denoising (TV-L1)

TV-L1 model:

$$\min_x \| \nabla x \| + \lambda \| x - f \|$$

Т.е. тяготение к шумам гораздо слабее.

Primal-Dual algorithm (TV-L1)

Primal-Dual алгоритм минимизации в общем виде решает:

$$\min_x F(Kx) + G(x)$$

Где: F, G - выпуклые функции, K - линейный оператор.

Тогда итеративная схема (модель - выпуклая):

$$\begin{cases} p_{n+1} = (I + \sigma \partial F^*)^{-1}(p_n + \sigma K x_n) \\ \hat{x}_{n+1} = (I + \tau \partial G)^{-1}(x_n - \tau K^T p_{n+1}) \\ x_{n+1} = \hat{x}_{n+1} + \theta(\hat{x}_{n+1} - x_n) \end{cases}$$

$$(I + \sigma \partial F^*)^{-1}(p) = \mathbf{project}_P(p)$$

$$(I + \tau \partial G_{TV-L1})^{-1}(x) = \mathbf{shrink}(x,\ f,\ \lambda\tau)$$

$$\mathbf{project}_P(p) = \frac{p}{max(\ \| p \|\ ,\ 1)}$$

$$\mathbf{shrink}(x,\ f,\ \lambda\sigma) = \begin{cases} x - \lambda\sigma & x > f + \lambda\sigma \\ x + \lambda\sigma & x < f - \lambda\sigma \\ f & |x - f| \le \lambda\sigma \end{cases}$$

$$K x_n = \nabla x_n$$

$$K^T p_{n+1} = \nabla^T p_{n+1}$$

Подробнее: [IPython notebook with ROF and TVL1 denoising (with math!)]

Image Denoising (TV-L1)

TV-L1 model:

$$\min_x \| \nabla x \| + \lambda \| x - f \|$$

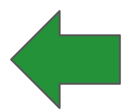
$\lambda = 1$

$\lambda = 1$

Image Denoising (TV-L1, **много наблюдений**)

TV-L1 model:

$$\min_x \| \nabla x \| + \lambda \sum_i \| x - f_i \|$$



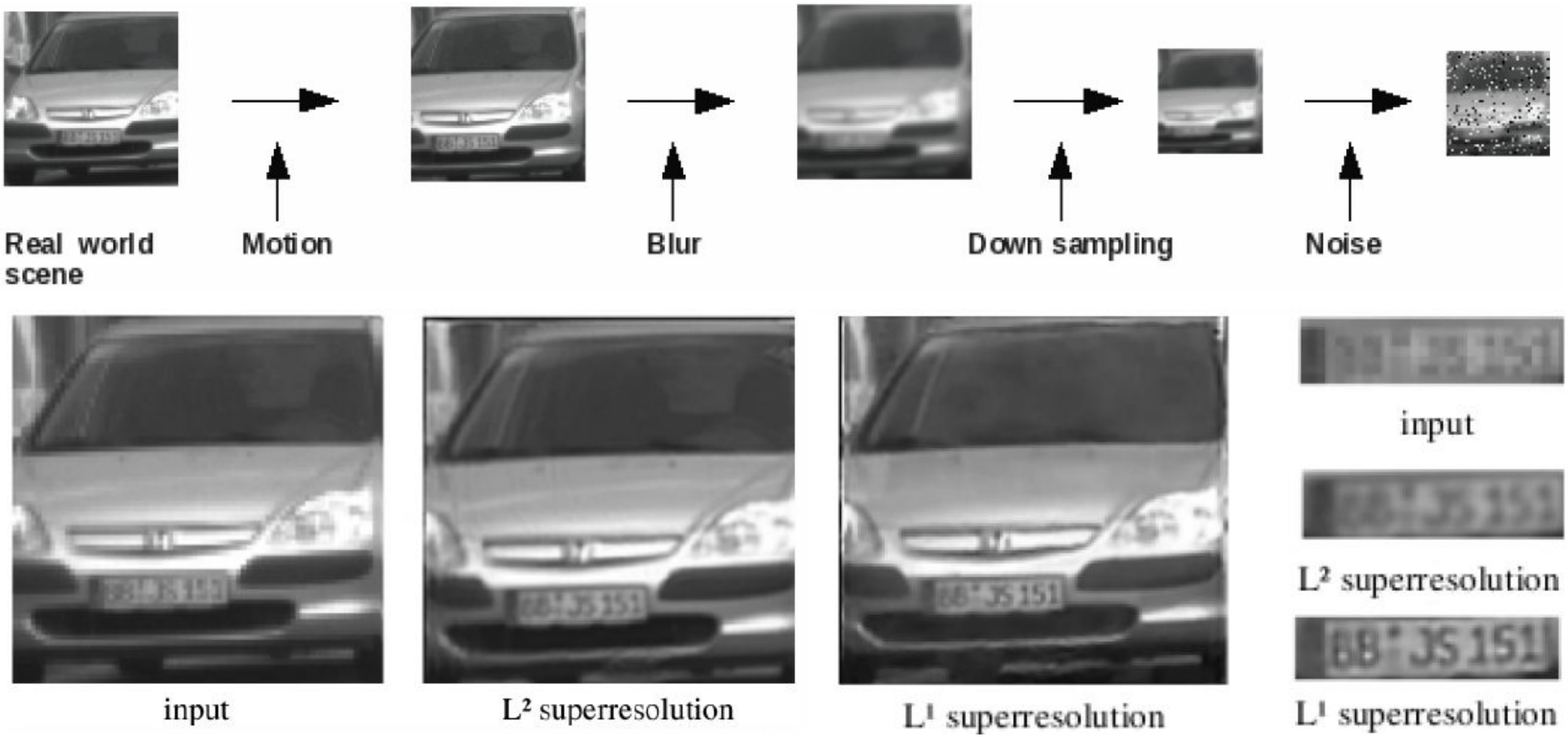
5 очень шумных наблюдений:

Подробнее: [IPython notebook with ROF and TVL1 denoising (with math!)]

Ссылки

- An introduction to Total Variation for Image Analysis, Chambolle et al., 2009
- IPython notebook with ROF and TVL1 denoising (with math!)
- Google Scholar: Thomas Pock (lots of research with Primal Dual method)

Image super resolution from multiple images
TV-L2, TV-L1



Подробнее: [Video Super Resolution using Duality Based TV-L1 Optical Flow, Mitzel et al., 2009]()

Huber loss

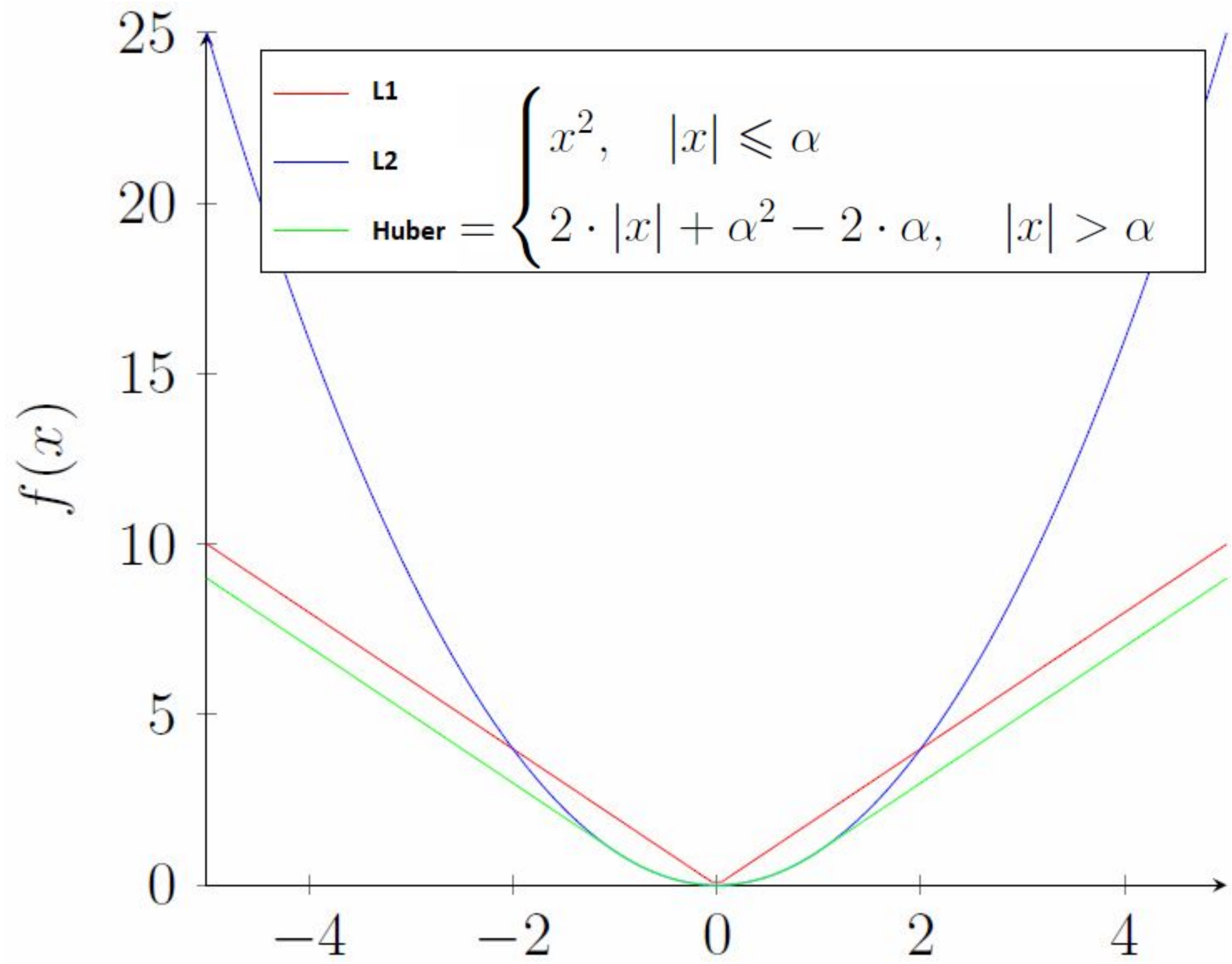

L1
L2
Huber $= \begin{cases} x^2, & |x| \leqslant \alpha \\ 2 \cdot |x| + \alpha^2 - 2 \cdot \alpha, & |x| > \alpha \end{cases}$
$f(x)$
25
20
15
10
5
0
−4
−2
0
2
4

Image super resolution from multiple images

Huber model:

$$\min_x \| \nabla x \|^h + \lambda \| x - f \|^h$$

Подробнее: [A Convex Approach for Variational Super-Resolution, Unger et al., 2010]

Digital **S**urface **M**odel reconstruction

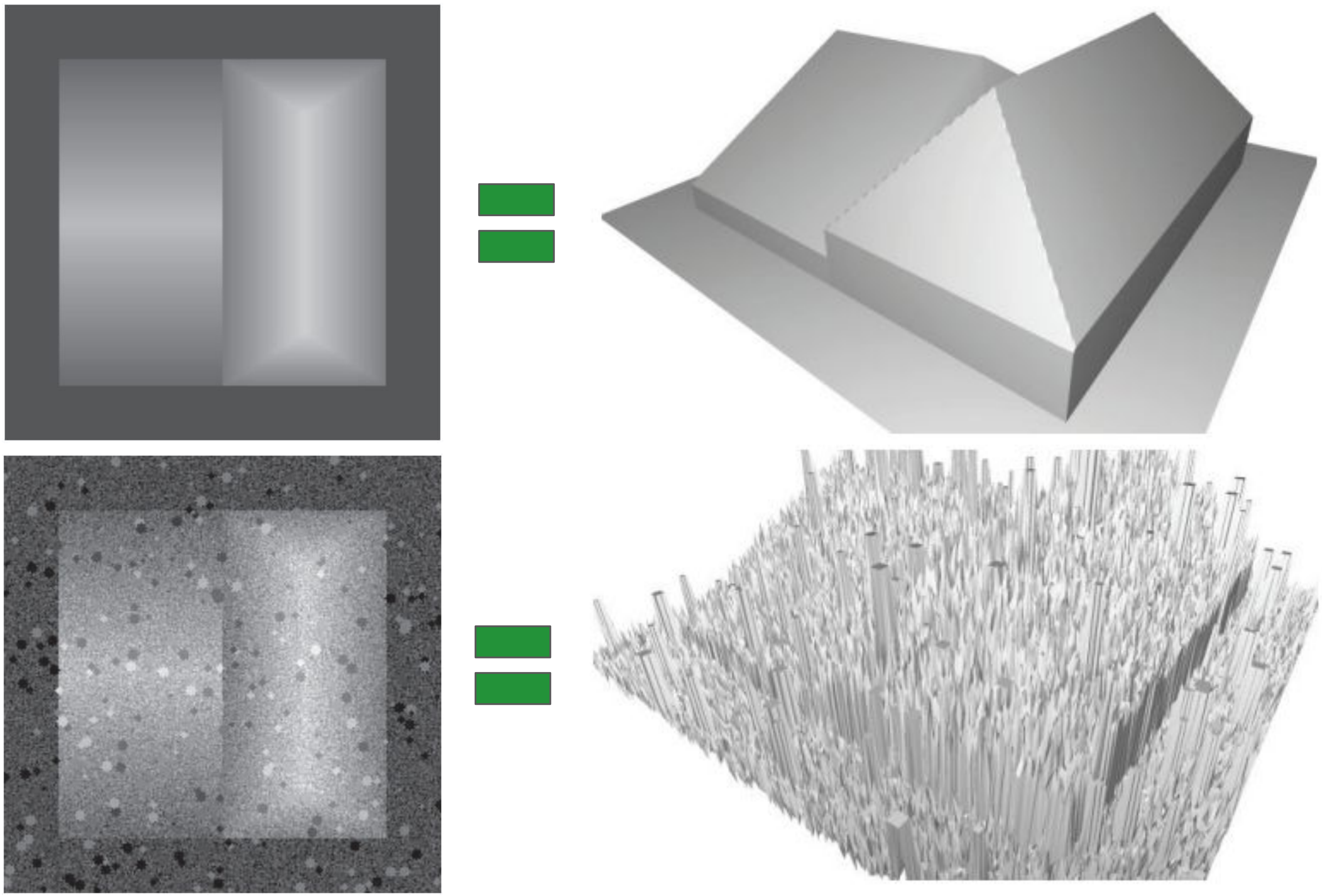
В image super resolution:

- Много шумных наблюдений - **много шумных картинок**
- Нужно найти одну наиболее правдоподобную - **четкую чистую картинку**

В случае **2.5D** реконструкции поверхности ландшафта (**DSM**):

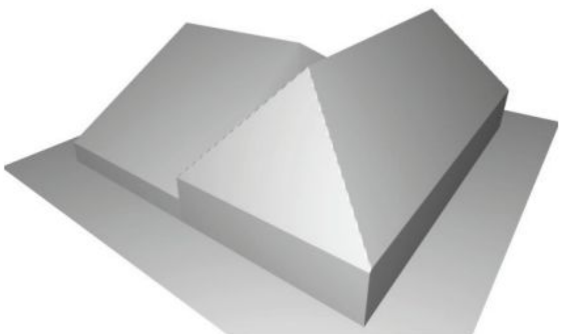
- Много шумных наблюдений - **шумные множества 2.5D точек**
  (на каждый сканер/фотографию - свое множество точек)
- Нужно найти одну наиболее правдоподобную - **2.5D карту высот**
  (т.е. как картинка, но в каждом пикселе - высота ландшафта в этой точке)

DSM (**D**igital **S**urface **M**odel, 2.5D height map)

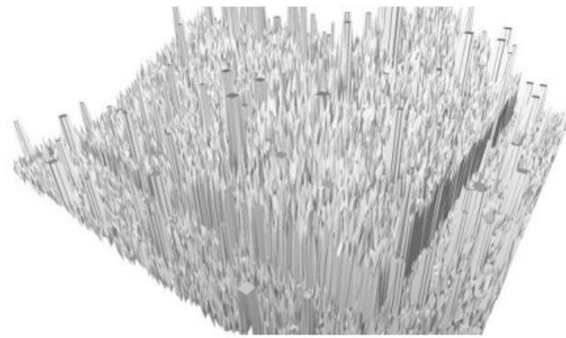


Gussian noise + outliers

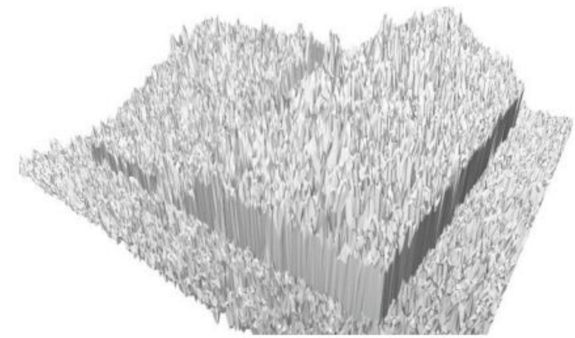
DSM from 5 observations (from 5 noisy height maps)



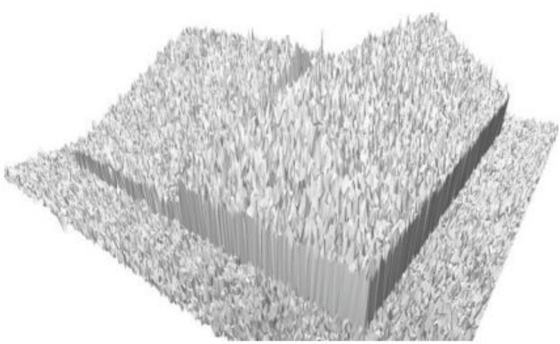
(a) Ground truth



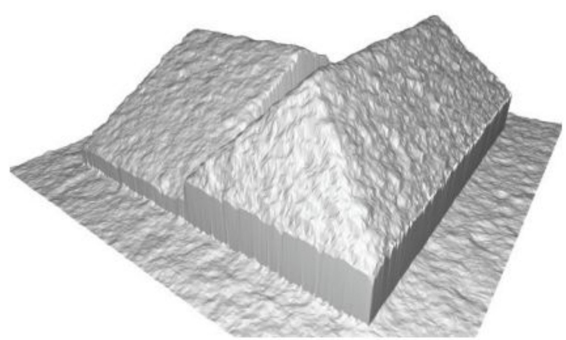
(b) Input image



(c) Average



(d) Median



(e) Huber

(f) TGV^2

TV-L1 -> Huber model -> TGV-Huber model

$\boldsymbol{TV\text{-}L^1}$ **model:**
$$\min_u \left\{ \alpha \int_\Omega |\nabla u| dx + \sum_{l=1}^{K} \int_\Omega |u - f_l|\, dx \right\}$$

Huber model:
$$\min_u \left\{ \alpha \int_\Omega |\nabla u|_\varepsilon \, dx + \sum_{l=1}^{K} \int_\Omega |u - f_l|_\delta \, dx \right\}$$

TGV^2 **model:**
$$\min_{u,v} \left\{ \alpha_1 \int_\Omega |\nabla u - v| dx + \alpha_0 \int_\Omega |\mathcal{E}(v)| dx + \sum_{l=1}^{K} \int_\Omega |u - f_l|_\delta \, dx \right\}$$

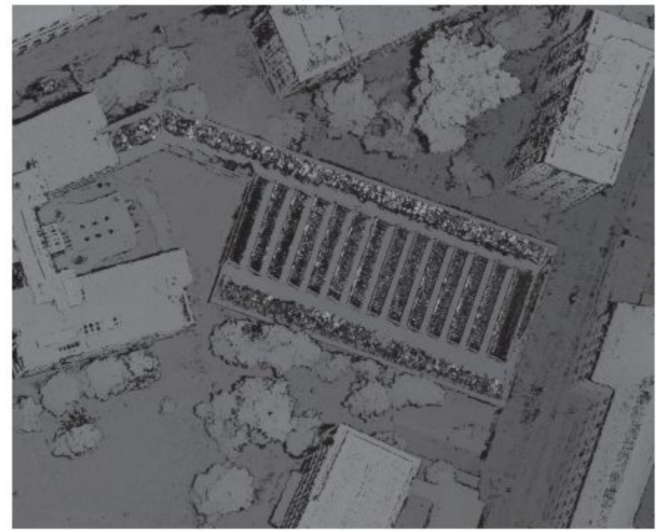
Т.е. регуляризация второго порядка:

$\alpha_1 \int_\Omega |\nabla u - v| dx$ => $v \approx \nabla u$

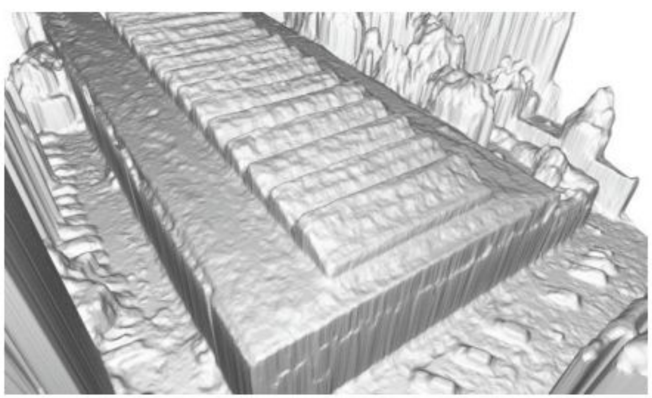
$\alpha_0 \int_\Omega |\mathcal{E}(v)| dx$ - полная вариация v ≈ полная вариация ∇u

Real-world examples

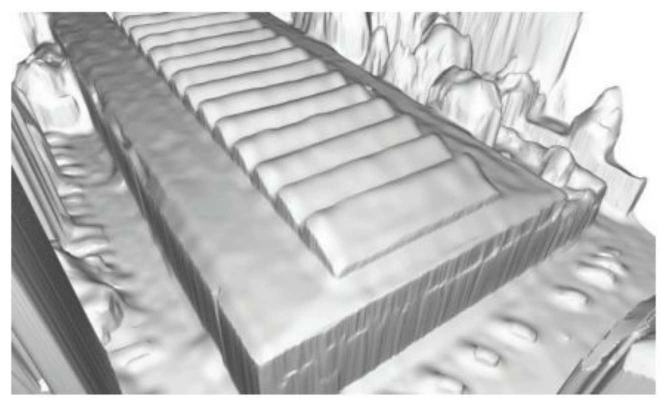
(a)



(b)

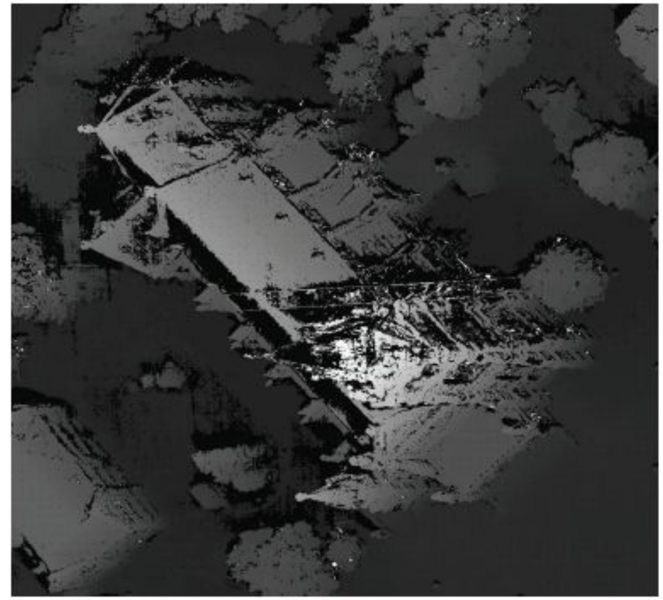


(c) Huber model

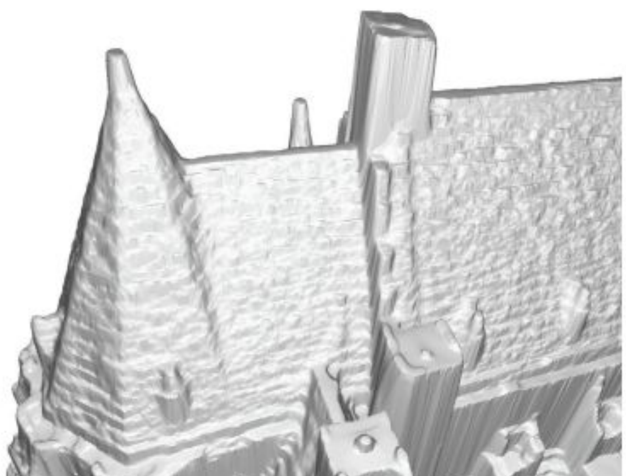


(d) TGV^2

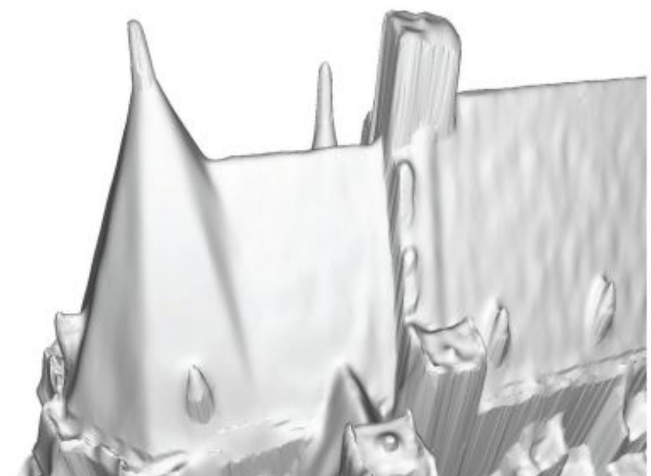
(a)



(b)



(c) Huber model



(d) TGV^2

Итого

Плюсы:

- Локальное решение, поэтому легко обрабатывать потайлово
- Регулярная решетка и независимые попиксельные вычисления
- Легко ускорить сходимость coarse-to-fine стратегией

Минусы:

- Размер тайла обработки зависит от числа наблюдений данного региона (т.е. зависит от степени перекрытия наблюдений)
- Неадаптивный размер ячейки регулярной решетки
- Это все еще 2.5D, а не произвольная 3D поверхность

Подробнее: TGV-Fusion, Pock et al., 2011

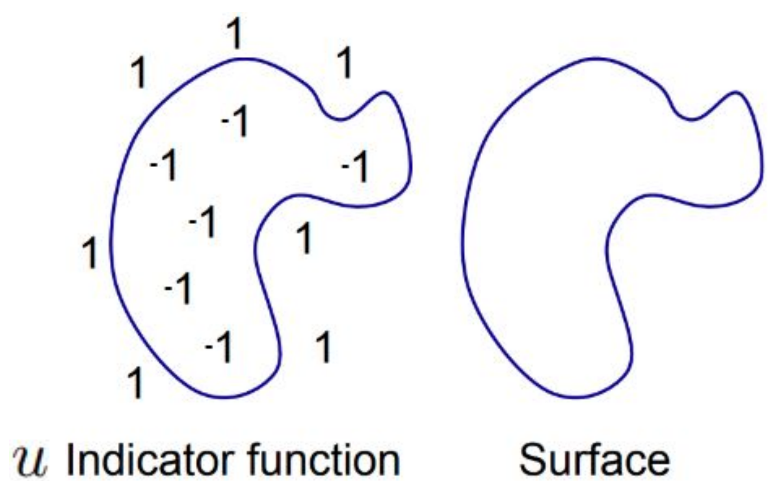
3D Model reconstruction

В случае **2.5D** реконструкции поверхности ландшафта (**DSM**):

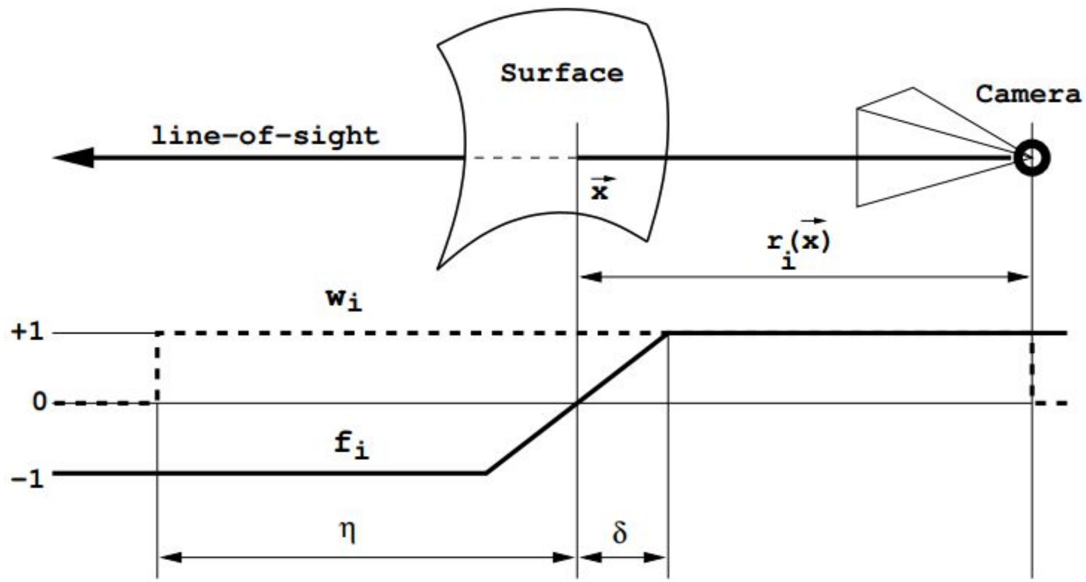
- Много шумных наблюдений - **шумные множества 2.5D точек**
- Нужно найти одну наиболее правдоподобную - **2.5D карту высот**

В случае **3D** реконструкции произвольной поверхности:

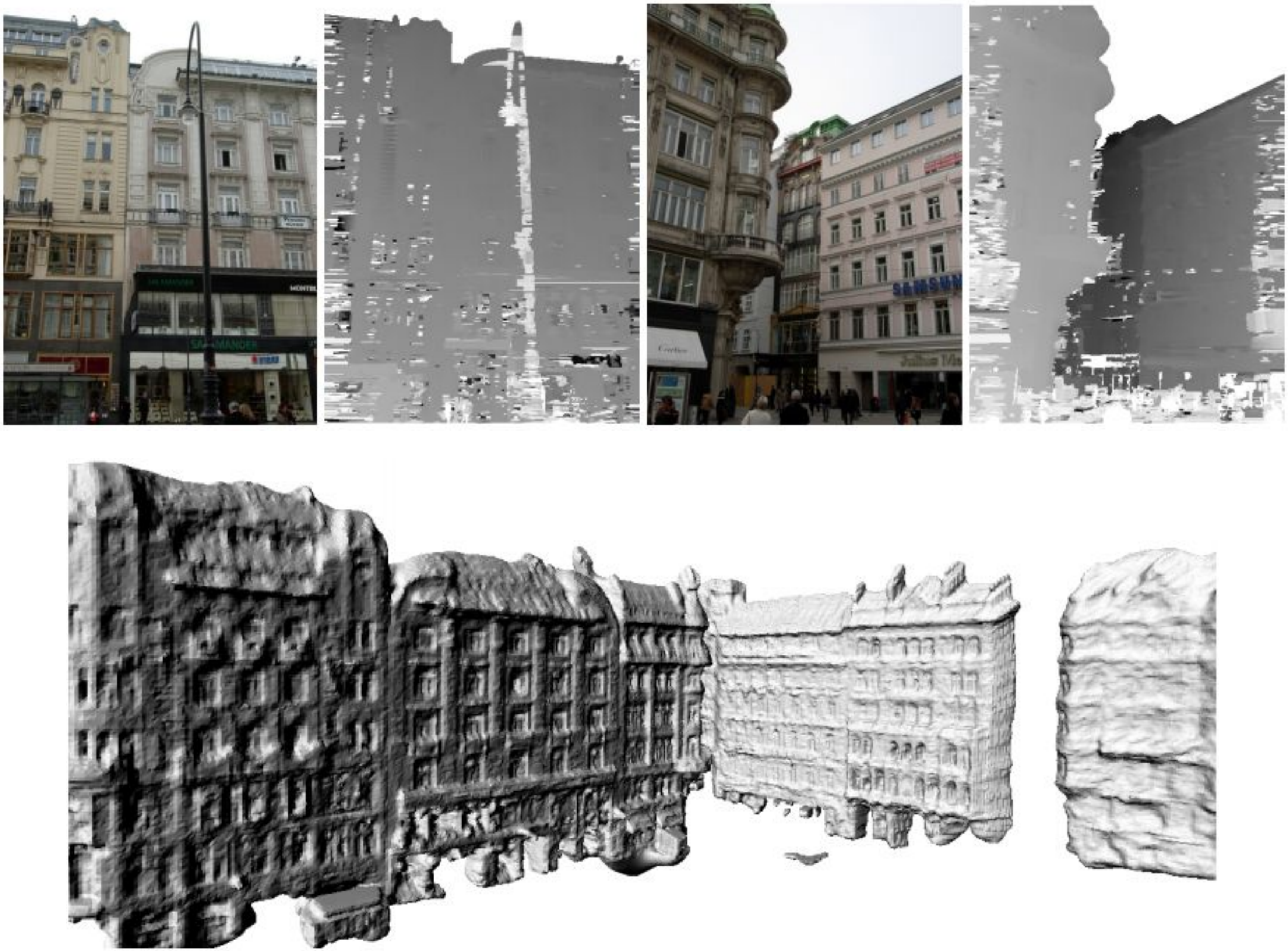
- Много шумных наблюдений - **шумные множества из 3D точек-лучей** (луч идущий из сканера/камеры-наблюдателя к наблюдаемой точке)
- Нужно найти наиболее правдоподообную - **3D поверхность,** сформулированную как изоповерхность в поле индикатор-функции:



TV-L1 for **3D** model



$$E = \int_\Omega \left\{ |\nabla u| + \lambda \sum_{i \in \mathcal{I}(\vec{x})} w_i(\vec{x}) |u - f_i| \right\} d\vec{x}$$



Итого

Плюсы:

- Локальное решение, поэтому можно обрабатывать по кубику
- Регулярная решетка и независимые попиксельные вычисления
- Легко ускорить сходимость coarse-to-fine стратегией

Минусы:

- **Размер кубика обработки зависит от числа наблюдений кубика** (т.е. зависит от степени перекрытия наблюдений)
- Неадаптивный размер ячейки регулярной решетки и от этого большое потребление памяти

Подробнее:

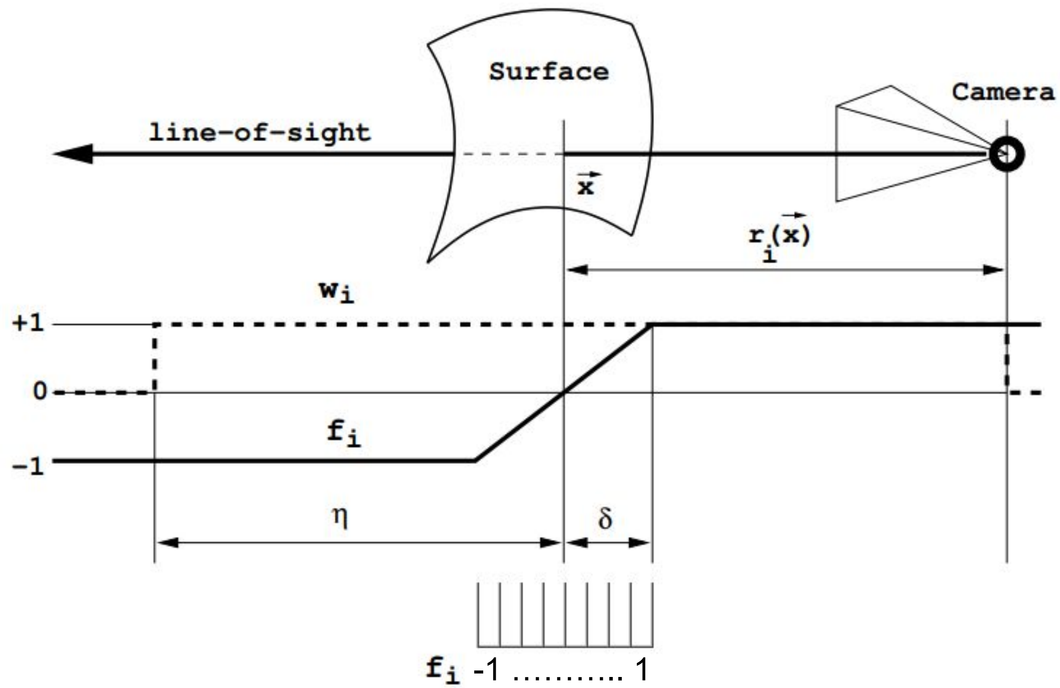
A Globally Optimal Algorithm for Robust TV-L1 Range Image Integration, Zach et al., 2007

3D Model reconstruction

Как убрать потребность держать в видеопамяти все наблюдения для обрабатываемого региона?

Можно сохранить релевантную информацию обо всех наблюдениях в каждом кубике регулярной решетке (вокселе).

Как это сделать? Гистограммами:



Итого

Плюсы:

- Локальное решение, поэтому можно обрабатывать по кубику
- Регулярная решетка и независимые попиксельные вычисления
- Легко ускорить сходимость coarse-to-fine стратегией
- **Размер кубика обработки не зависит от числа наблюдений кубика**

Минусы:

- Неадаптивный размер ячейки регулярной решетки и от этого большое потребление памяти

Подробнее: [Fast and High Quality Fusion of Depth Maps, Zach, 2008]

Итого

Плюсы:

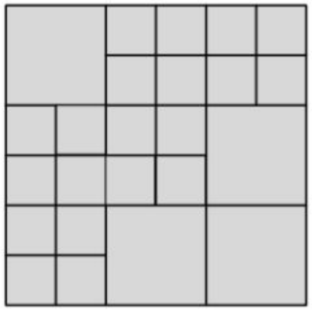
- Локальное решение, поэтому можно обрабатывать по кубику
- Регулярная решетка и независимые попиксельные вычисления
- Легко ускорить сходимость coarse-to-fine стратегией
- Размер кубика обработки не зависит от числа наблюдений кубика

Минусы:

- **Неадаптивный размер ячейки регулярной решетки и от этого большое потребление памяти**

Подробнее: Fast and High Quality Fusion of Depth Maps, Zach, 2008

3D Model reconstruction

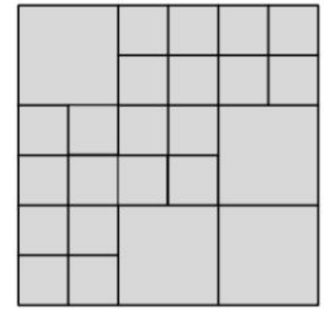


Как перейти от регулярной решетки к адаптивному октодереву?

Есть проблема:

Чтобы быстро выполнять численные итерации - нужно уметь быстро считать градиент. А значит нужно в каждом вокселе хранить индекс соседа.

3D Model reconstruction



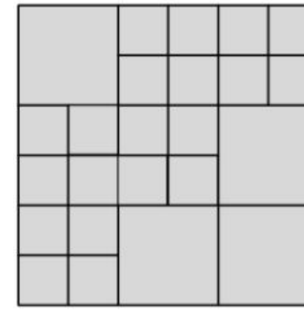
Как перейти от регулярной решетки к адаптивному октодереву?

Есть проблема:

Чтобы быстро выполнять численные итерации - нужно уметь быстро считать градиент. А значит нужно в каждом вокселе хранить индекс соседа.

Предварительно обойти октодерево и найти соседей не сложно - через **Look Up Tables (LUTs)**.

3D Model reconstruction



Как перейти от регулярной решетки к адаптивному октодереву?

Есть проблема:

Чтобы быстро выполнять численные итерации - нужно уметь быстро считать градиент. А значит нужно в каждом вокселе хранить индекс соседа.

Предварительно обойти октодерево и найти соседей не сложно - через **Look Up Tables (LUTs)**.

Но у каждого вокселя может быть **произвольное количество соседей**, а значит сохранить их в каждом вокселе для дальнейшего быстрого обращения с видеокарты так просто не выйдет.

Ограничим число соседей балансировкой

Пусть адаптивное октодерево сбалансировано - глубина соседей отличается не больше чем на один.

Теперь у каждого вокселя не больше чем 4*6=**24 соседа**.

Ограничим число соседей балансировкой

Пусть адаптивное октодерево сбалансировано - глубина соседей отличается не больше чем на один.

Теперь у каждого вокселя не больше чем 4*6=**24 соседа**.

Можно ли сэкономить еще?

Давайте хранить ссылку только на первый из четырех соседей, а для обращения к остальным трем соседям - использовать специальную **LUT**.

```
unsigned int LUTfromTheDeep[3][4] = {
        {0 + 0, 0 + 4, 2 + 0, 2 + 4},
        {0 + 0, 0 + 4, 1 + 0, 1 + 4},
        {0 + 0, 0 + 2, 1 + 0, 1 + 2},
};
```

Теперь каждый воксель хранит ссылки всего лишь на **6 соседей.**

Итого

- Вариационные методы шикарны, гибки и решают сложные вещи
- Большой объем численных операций компенсируется простотой и соответственно идеальной адаптацией на архитектуру видеокарт
- Данные можно сжимать в гистограммы (чтобы не зависеть от числа наблюдений, а зависеть лишь от обрабатываемого объема)
- Октодерево хуже регулярной решетки в смысле доступа к памяти, но лучше в смысле потребления памяти
- Обращаться к соседям на октодереве можно по предподготовленным указателям
- Для минимизации числа хранимых указателей нужно балансировать дерево и хранить ссылку лишь на один из соседей, а к остальным обращаться через LUT