Фотограмметрия: построение 3D-двойника города при 16 ГБ памяти

No OOM. Just results.

@UnicornGlade
@PolarNick239
polarhare@gmail.com
Николай Полярный

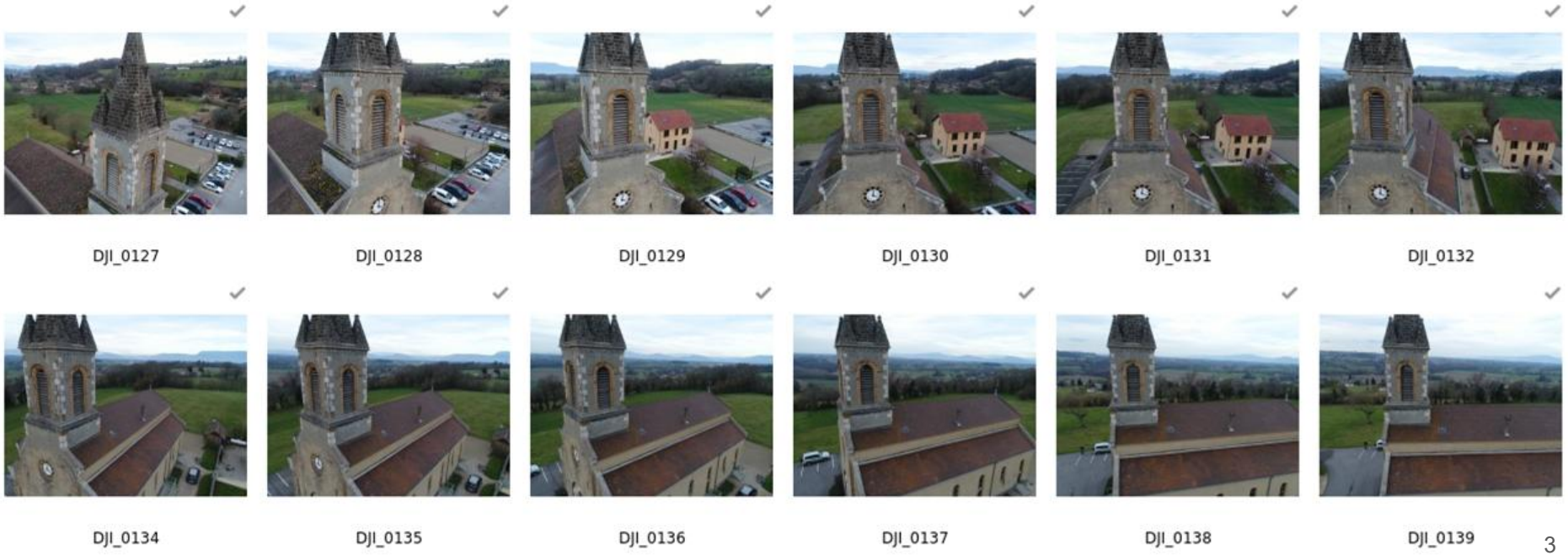
Agisoft Metashape

C++ Russia 2026


Задача

Фотограмметрия

По множеству фотографий одной и той же сцены получить цифровую модель



Данные предоставил Stéphane Prodent

Фотограмметрия

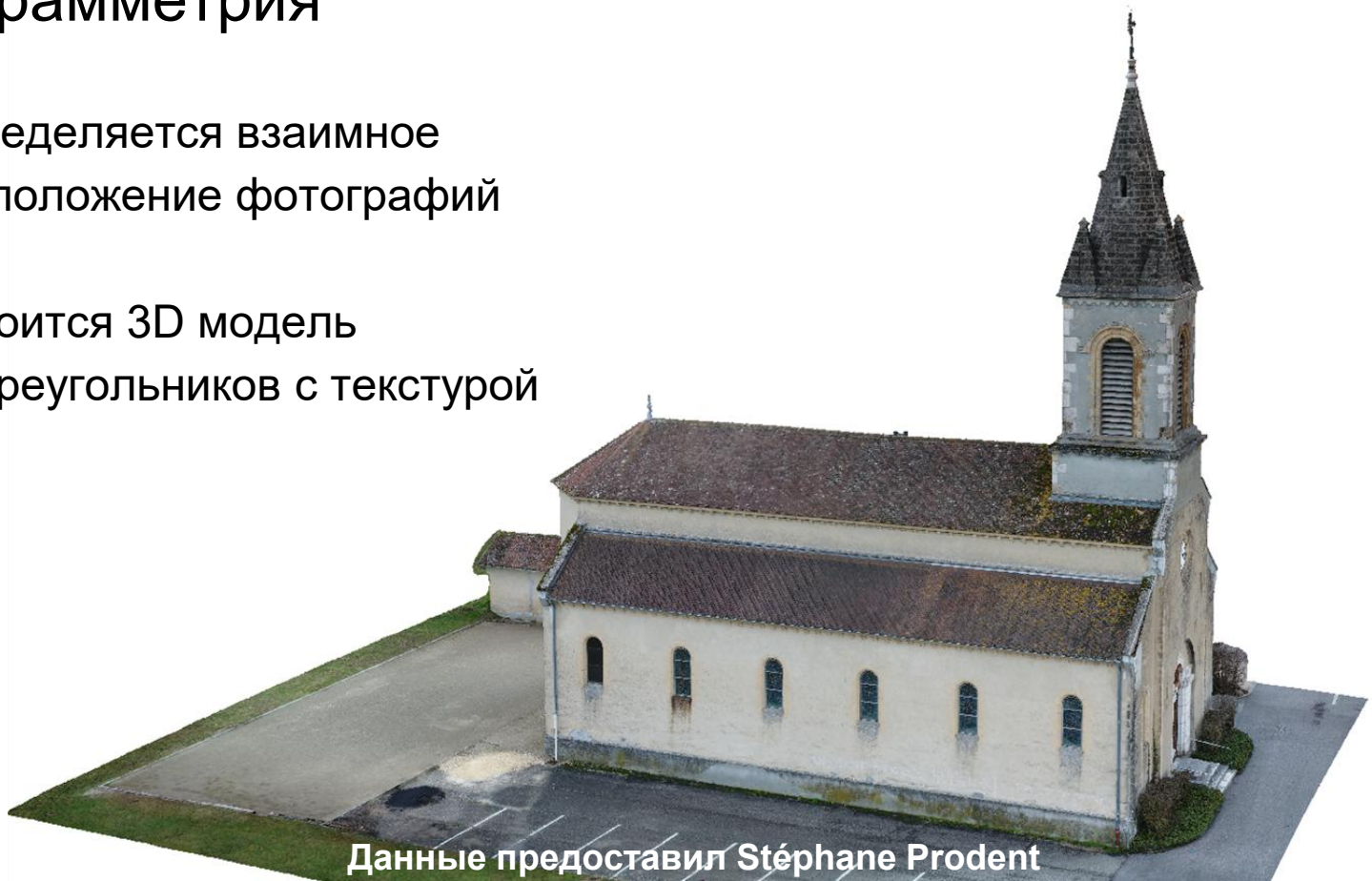
1) Определяется взаимное расположение фотографий

Данные предоставил Stéphane Prodent

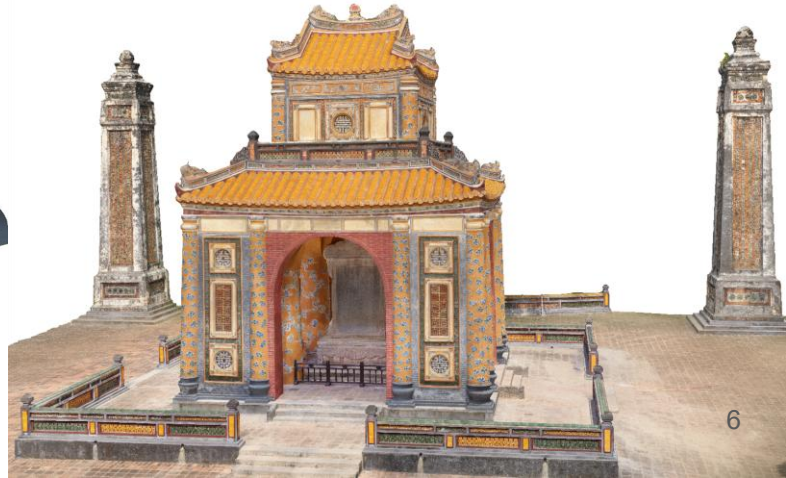
Фотограмметрия

1) Определяется взаимное расположение фотографий

2) Строится 3D модель из треугольников с текстурой



Данные предоставил Stéphane Prodent



is_copter-20180820-1-0760

is_copter-20180820-1-0761

is_copter-20180820-1-0762

is_copter-20180820-1-0763

is_copter-20180820-1-0764

is_copter-20180820-1-0765

is_copter-20180820-1-0766

is_copter-20180820-1-0884

is_copter-20180820-1-0885

is_copter-20180820-1-0886

is_copter-20180820-1-0887

is_copter-20180820-1-0888

is_copter-20180820-1-0889

is_copter-20180820-1-0890

is_copter-20180821-0-4041

is_copter-20180821-0-4042

is_copter-20180821-0-4043

is_copter-20180821-0-4044

is_copter-20180821-0-4045

is_copter-20180821-0-4046

is_copter-20180821-0-4047

is_copter-20180823-0-0811

is_copter-20180823-0-0812

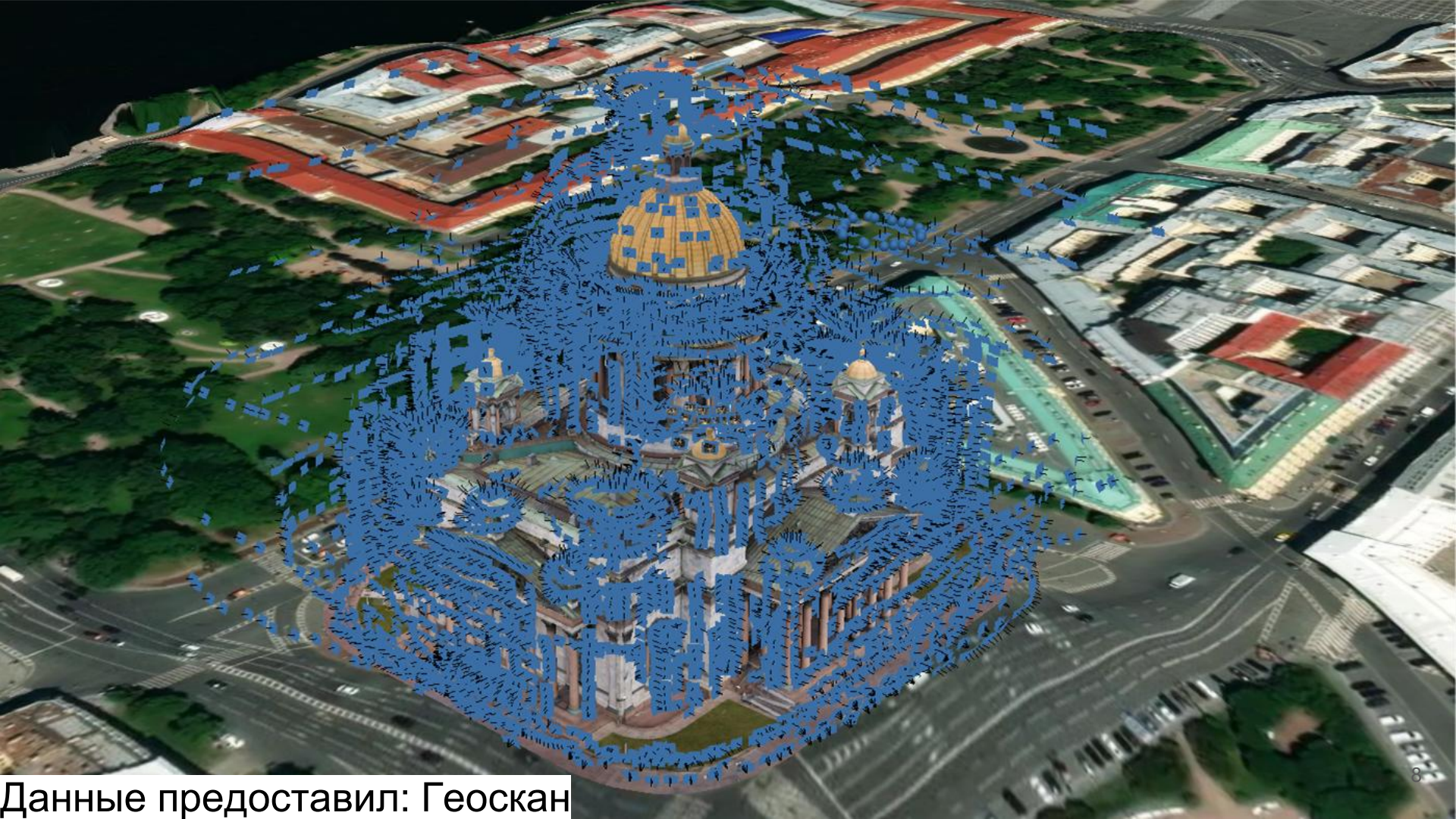
is_copter-20180823-0-0813

is_copter-20180823-0-0814

is_copter-20180823-0-0815

is_copter-20180823-0-0816

is_copter-20180823-0-0817



Данные предоставил: Геоскан

Данные предоставил: Геоскан

Данные предоставил: Геоскан

Данные предоставил: Геоскан

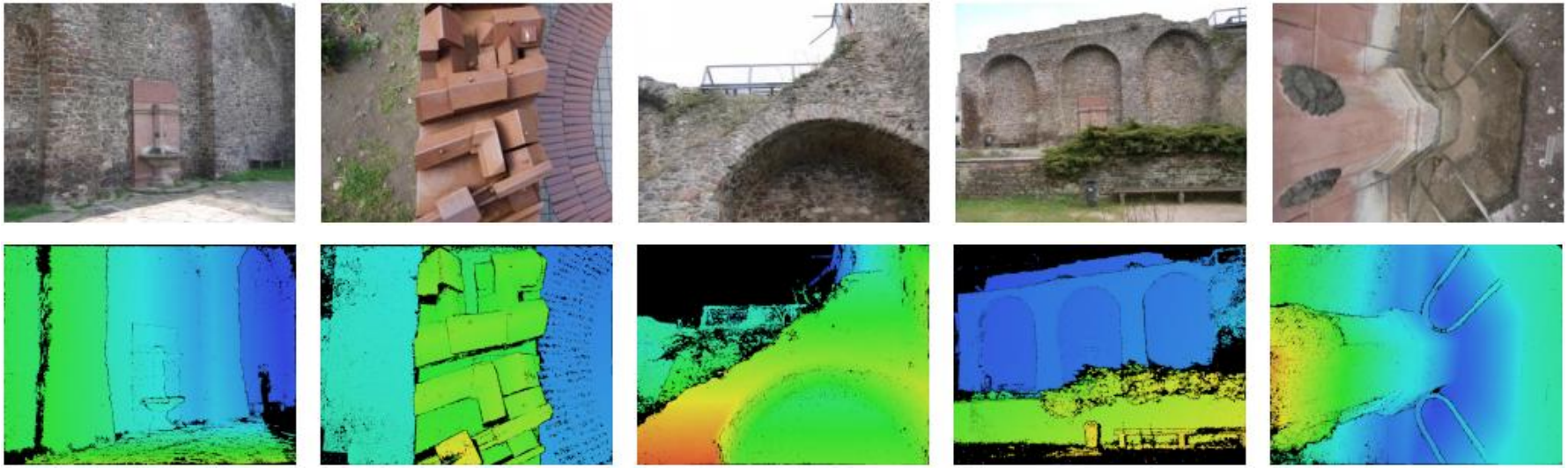
Данные предоставил: Геоскан

На вход: точное положение фотографий

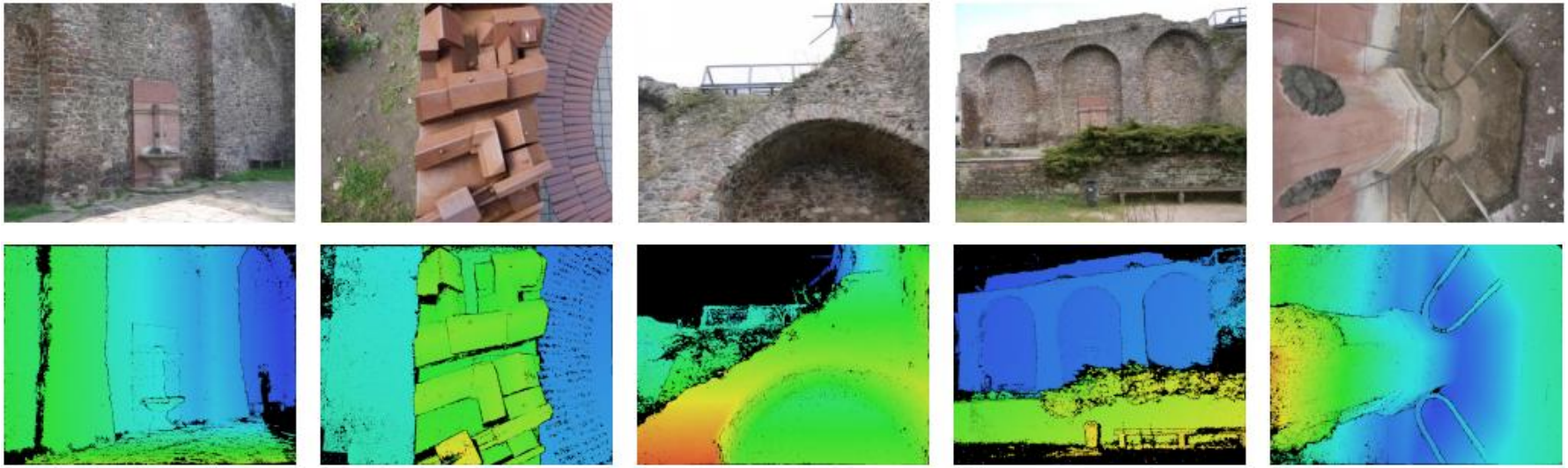




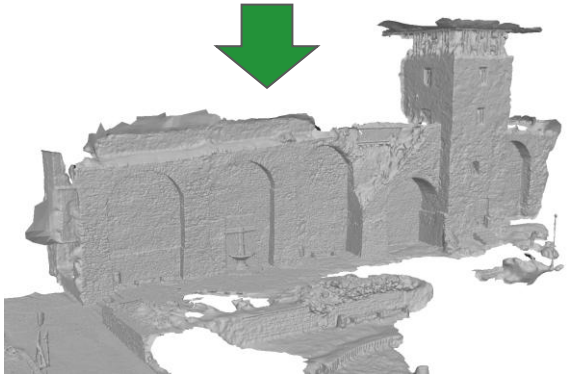
На вход: точное положение фото + карты глубины



На вход: точное положение фото + карты глубины



На выход:
полигональная
3D модель



2016 год, Словакия

RealityCapture

2016 год, Словакия

RealityCapture

WIKIPEDIA

Meshing, coloring and texturing are completely out-of-core in RC, which is intended to avoid RAM performance loss during these processes.

2016 год, Словакия

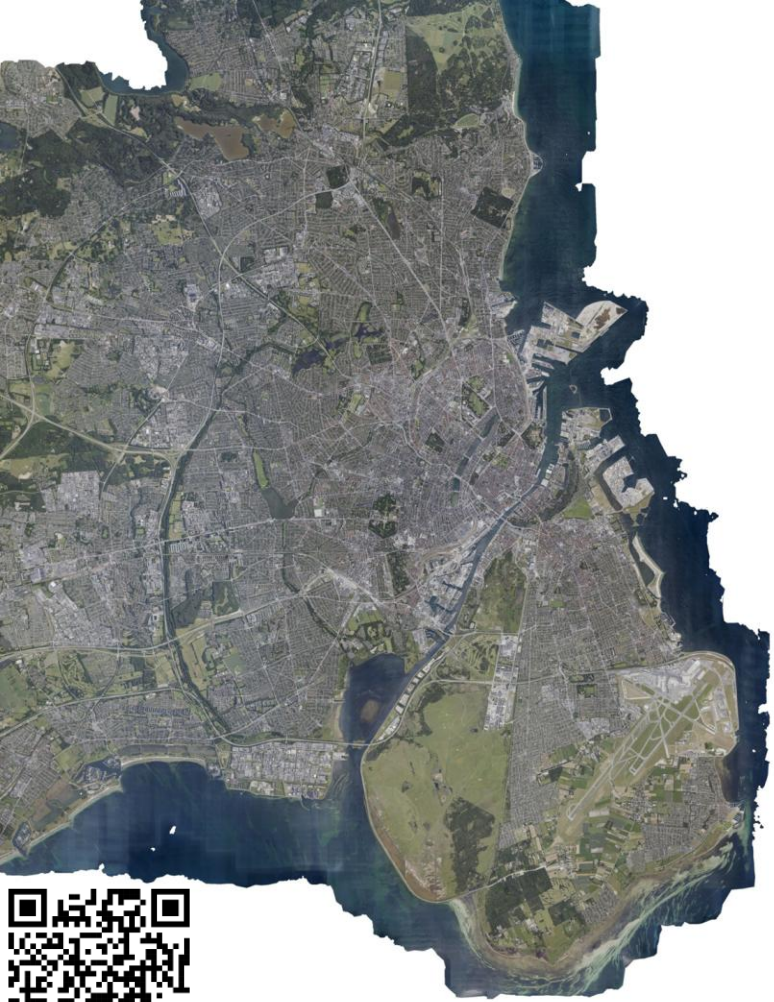
RealityCapture

2021 год, куплены EPIC GAMES

2024 год, бесплатная лицензия (если < $1 млн)

WIKIPEDIA

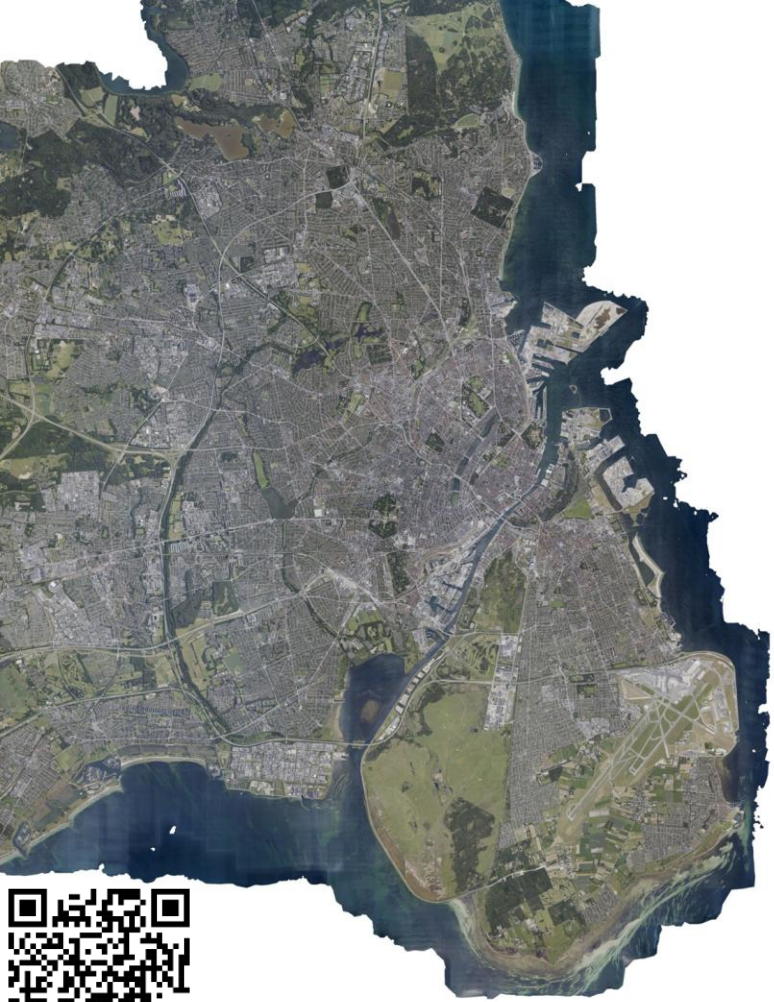
Meshing, coloring and texturing are completely out-of-core in RC, which is intended to avoid RAM performance loss during these processes.



Building Copenhagen in a Day*

- 425 km²
- 27472 фотографии (566 GB, jpeg)

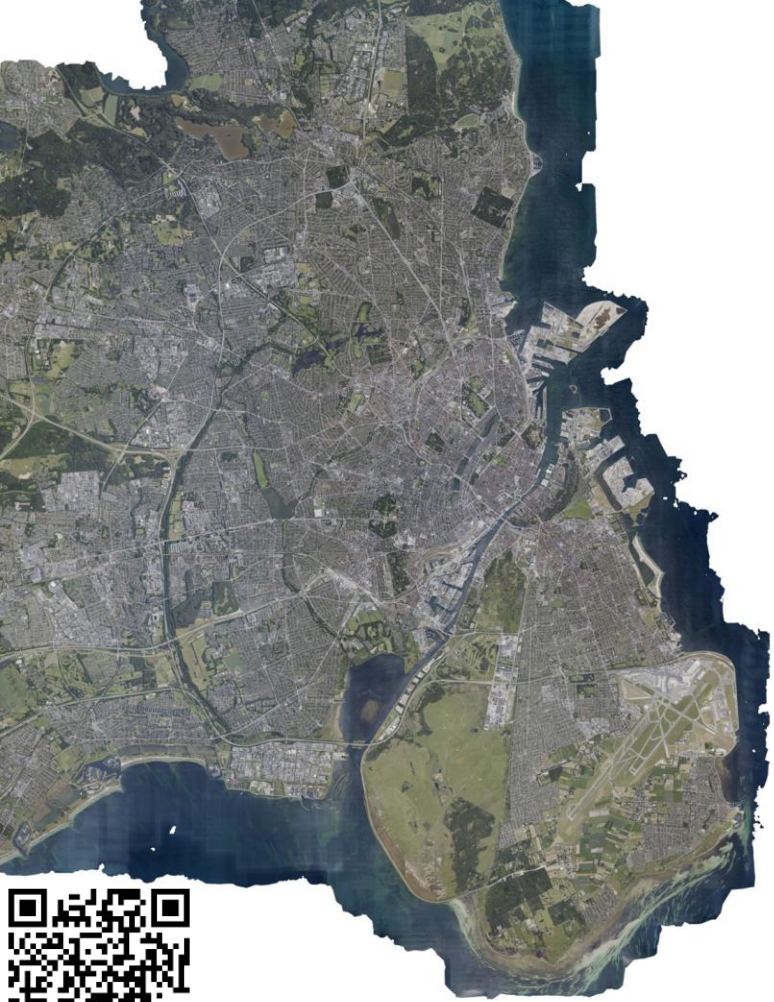
Out-of-Core Surface Reconstruction via Global TGV Minimization, **Poliarnyi, 2021**



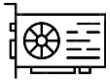
Building Copenhagen in a Day*

- 425 km²
- 27472 фотографии (566 GB, jpeg)
- 28 миллиардов “точек”
- 7.5 миллиардов треугольников

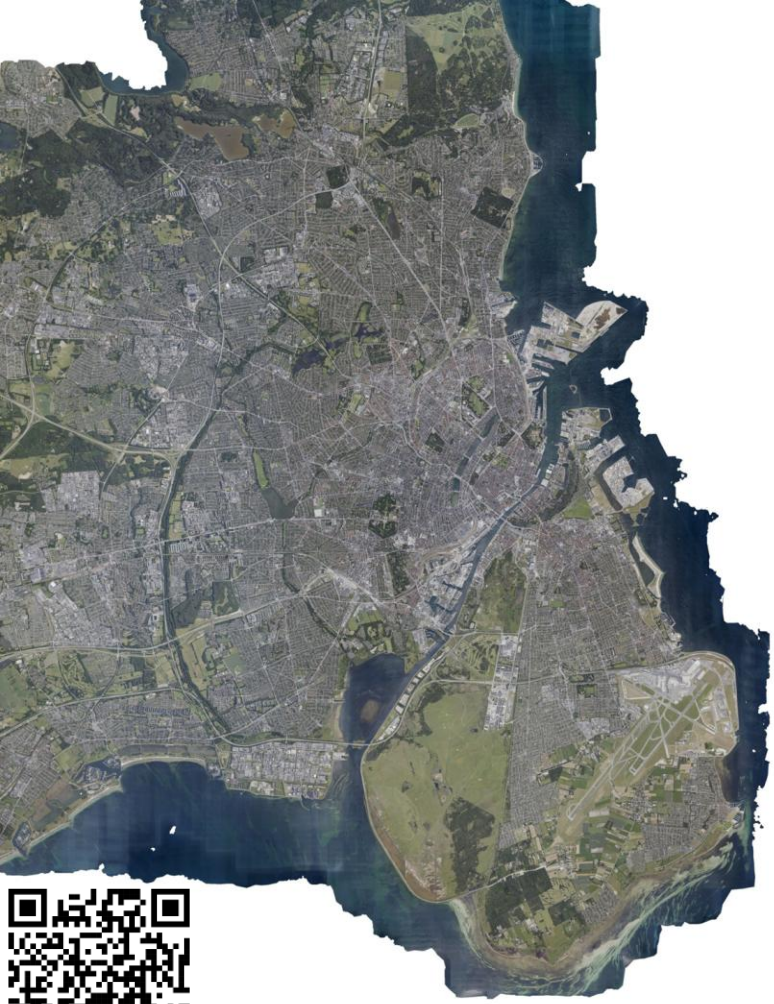
Out-of-Core Surface Reconstruction via Global TGV Minimization, **Poliarnyi, 2021**



Building Copenhagen in a Day*

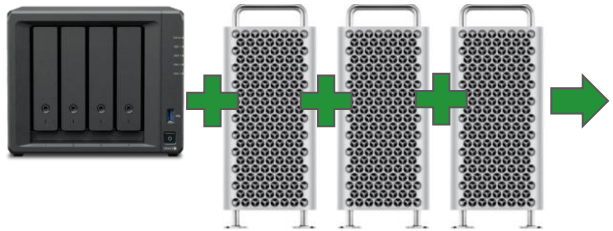
- 425 km²
- 27472 фотографии (566 GB, jpeg)
- 28 миллиардов “точек”
- 7.5 миллиардов треугольников
- **Пиковая RAM: 14 GB (не сервер!)**
- *на кластере за 29 часов:
  7 consumer grade компьютеров
  по 8 ядер + GTX 1080 
  + NAS как способ «коммуникации»

Out-of-Core Surface Reconstruction via Global TGV Minimization, **Poliarnyi, 2021**

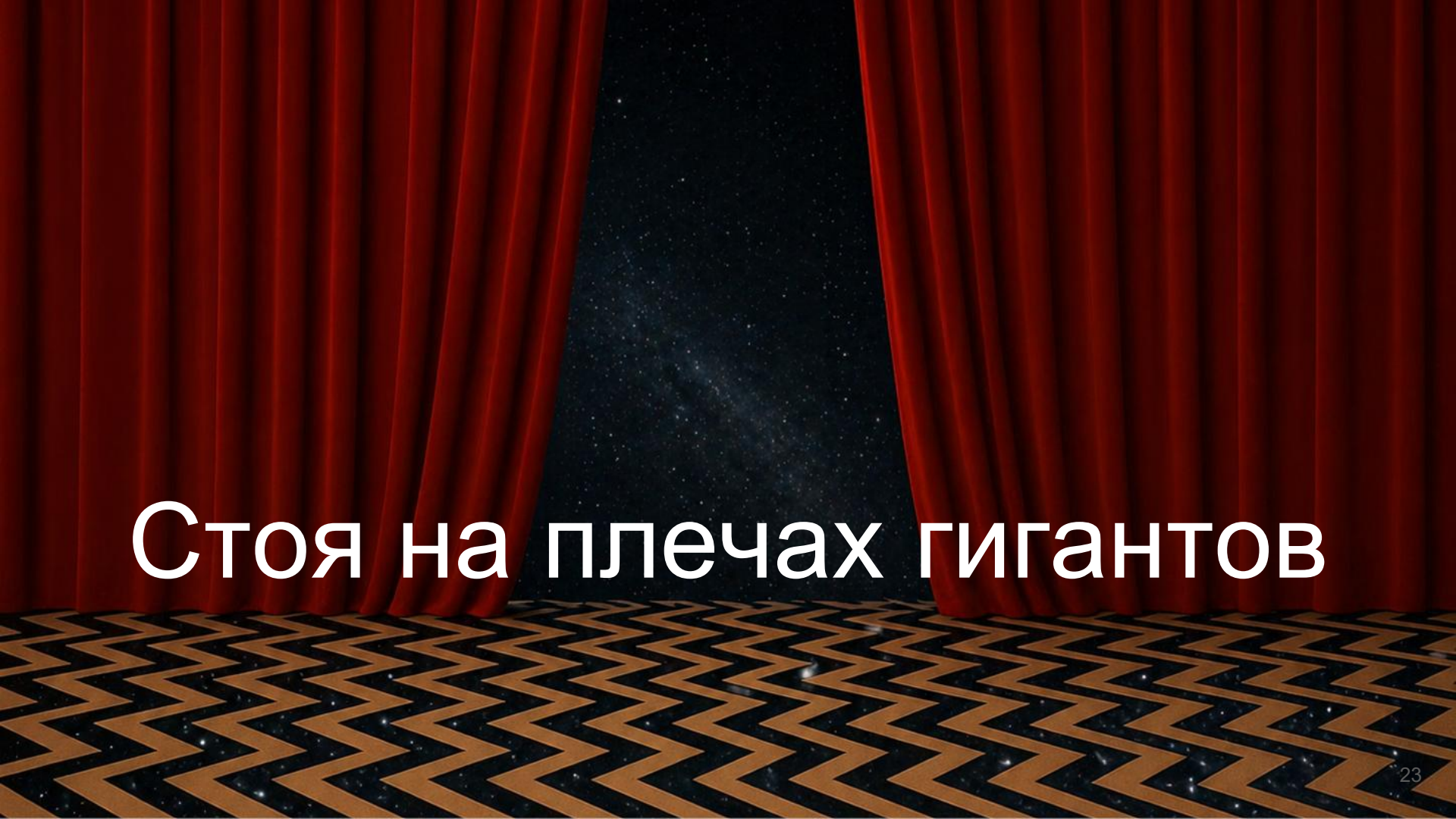


Building Copenhagen in a Day*

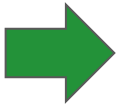
- 425 km²
- 27472 фотографии (566 GB, jpeg)
- 28 миллиардов “точек”
- 7.5 миллиардов треугольников
- **Пиковая RAM: 14 GB (не сервер!)**
- *на кластере за 29 часов:
  7 consumer grade компьютеров
  по 8 ядер + GTX 1080
  + NAS как способ «коммуникации»



Out-of-Core Surface Reconstruction via Global TGV Minimization, **Poliarnyi, 2021**


Стоя на плечах гигантов

Удаление шума с картинки



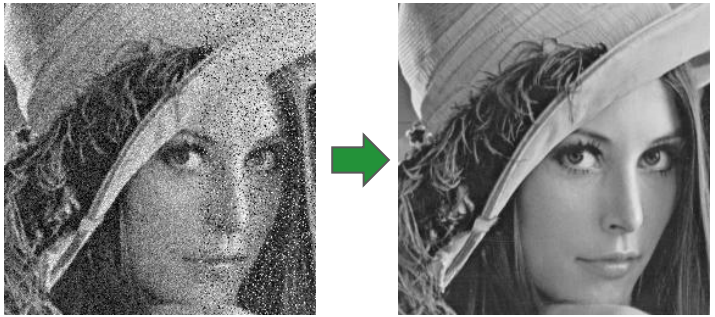
Шумная картинка

Эталон

TV-L2 Image Denoising (**ROF**)

Rudin-Osher-Fatemi model (ROF):

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



f - данные $\longleftrightarrow$ x - искомое

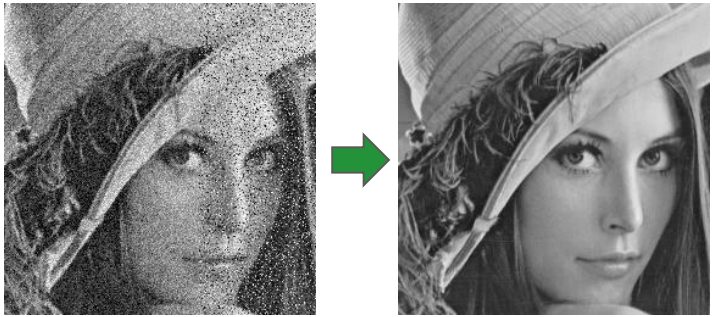
Где:

- $\| x - f \|^2$ - L^2 тяготение к данным (наблюдаемому f)

TV-L2 Image Denoising (**ROF**)

Rudin-Osher-Fatemi model (ROF):

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



f - данные $\longleftrightarrow$ x - искомое

Где:

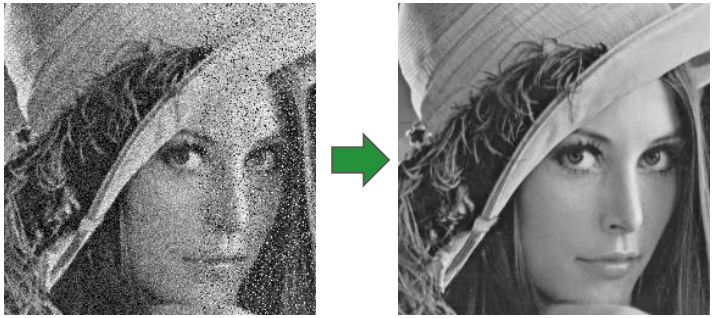
- $\| x - f \|^2$ - L^2 тяготение к данным (наблюдаемому f)

За счет чего можно удалить шум и выбросы?

TV-L2 Image Denoising (**ROF**)

Rudin-Osher-Fatemi model (ROF):

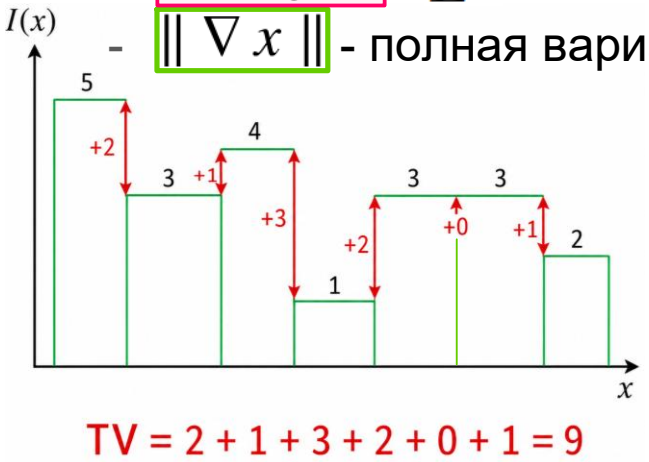
$$\min_x \| \nabla x \| + \frac{\lambda}{2} \| x - f \|^2$$



f - данные $\longleftrightarrow$ x - искомое

Где:

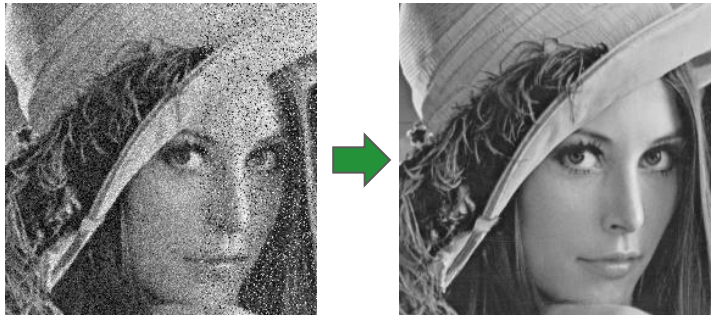
- $\| x - f \|^2$ - L^2 тяготение к данным (наблюдаемому f)
- $\| \nabla x \|$ - полная вариация (регуляризация)



TV-L2 Image Denoising (**ROF**)

Rudin-Osher-Fatemi model (ROF):

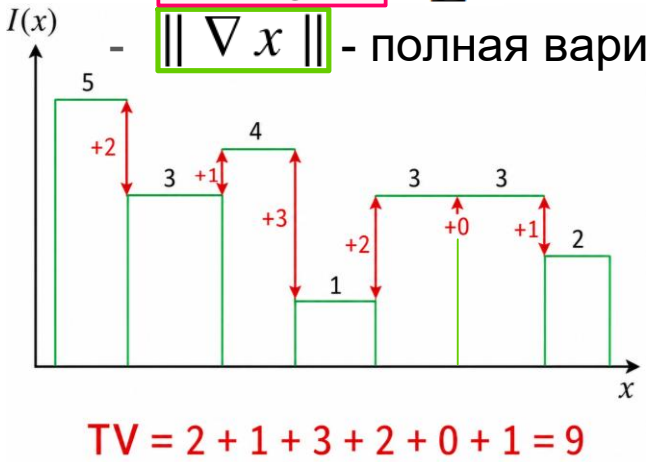
$$\min_x \| \nabla x \| + \frac{\lambda}{2} \| x - f \|^2$$



f - данные $\longleftrightarrow$ x - искомое

Где:

- $\| x - f \|^2$ - L^2 тяготение к данным (наблюдаемому f)
- $\| \nabla x \|$ - полная вариация (регуляризация)



Выпуклая оптимизация

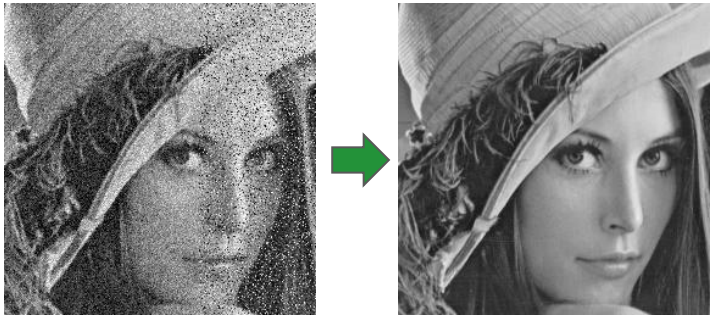
Много итераций (локальные вычисления)

OpenCL™

TV-L2 Image Denoising (**ROF**)

Rudin-Osher-Fatemi model (ROF):

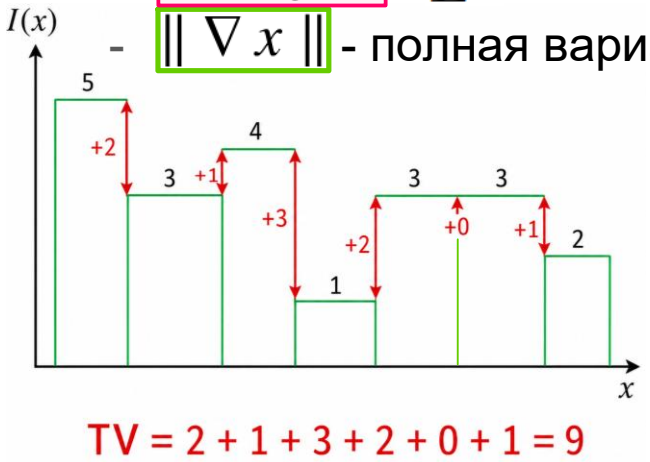
$$\min_x \| \nabla x \| + \frac{\lambda}{2} \| x - f \|^2$$



f - данные $\leftrightarrow$ x - искомое

Где:

- $\| x - f \|^2$ - L^2 тяготение к данным (наблюдаемому f)
- $\| \nabla x \|$ - полная вариация (регуляризация)



Что будет если убрать второе слагаемое?

Выпуклая оптимизация

Много итераций (локальные вычисления)

OpenCL™

Rudin–Osher–Fatemi model

Минимизируем полную вариацию и одновременно держимся за исходный сигнал f.

data fidelity λ

0.45

λ: насколько сильно держимся за f

меньше — TV сильнее · больше — ближе к f

сильное сглаживание — сильная привязка к f

ROF energy

0.309

TV(x)

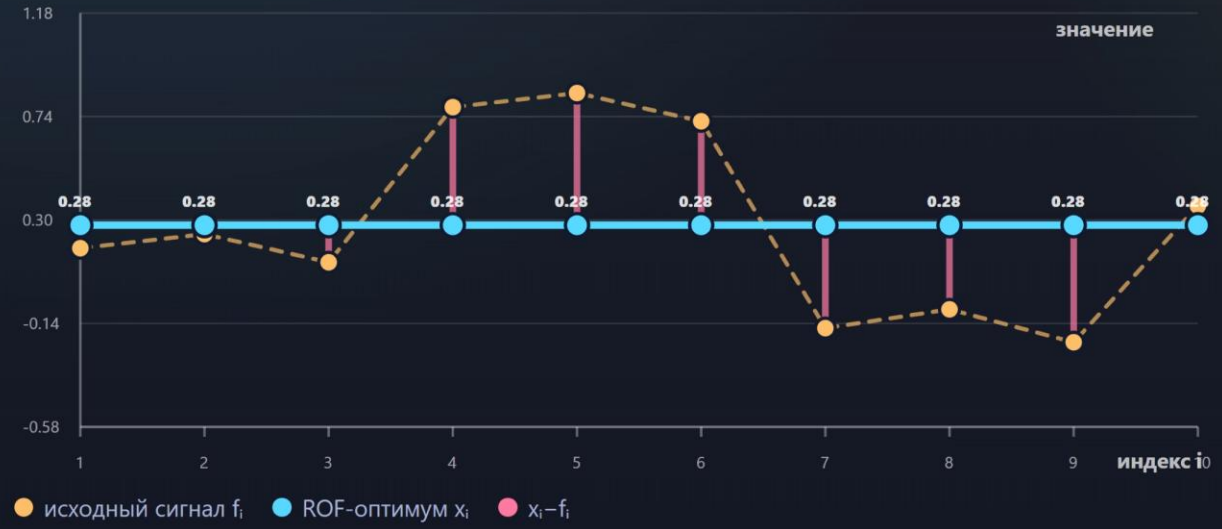
0.00

$\lambda/2 \cdot ||x-f||^2$

0.309

Оптимум x для текущего λ

розовые отрезки — отличие x от исходного f



ROF functional

$$x^* = \operatorname{argmin}_x E_{ROF}(x)$$

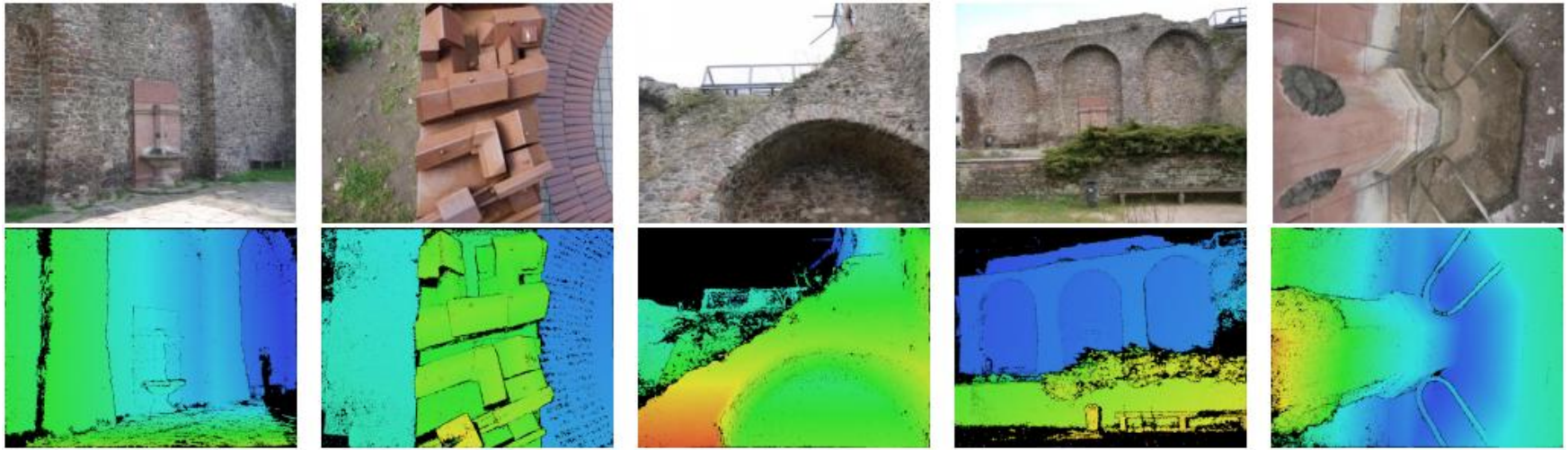
$$E_{ROF}(x) = TV(x) + \lambda/2 \cdot ||x - f||^2_2$$

$$TV(x) = \Sigma_i |x_i - x_{i-1}|$$

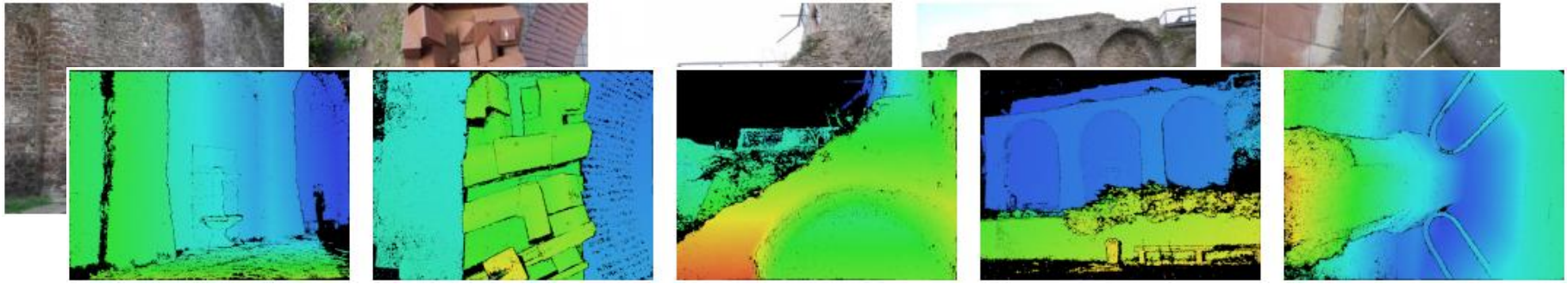
$$||x - f||^2_2 = \Sigma_i (x_i - f_i)^2$$

TV(x) делает сигнал кусочно-постоянным.
$\lambda/2 \cdot ||x-f||^2$ штрафует уход от данных.

В этой форме ROF: чем больше λ, тем ближе x к f.

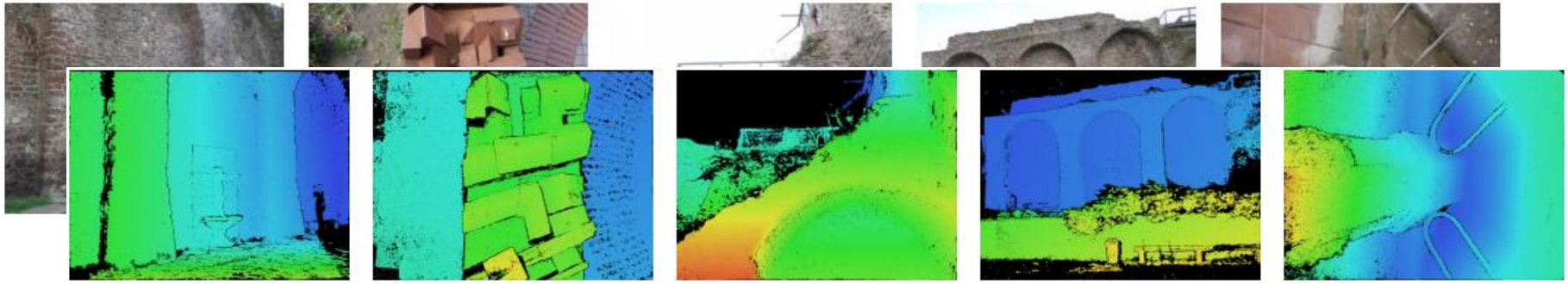


На вход: карты глубины



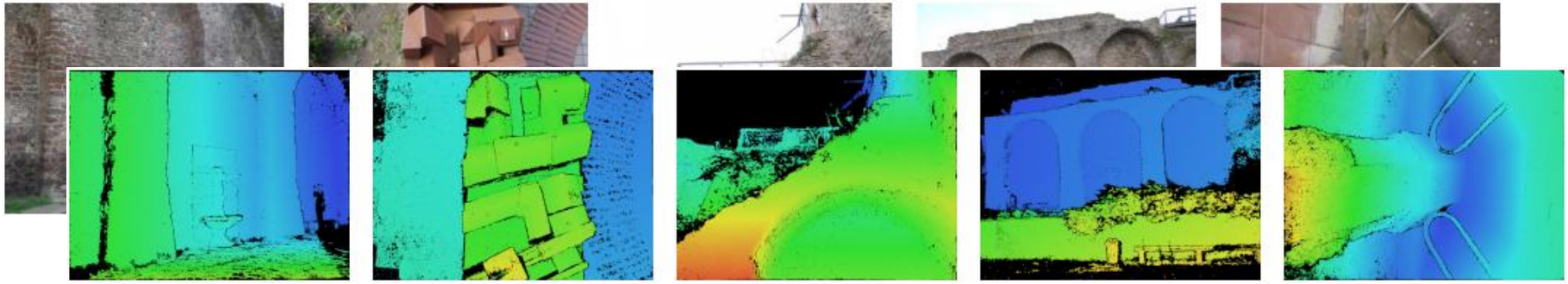
На вход: карты глубины

На выход: полигональная модель?



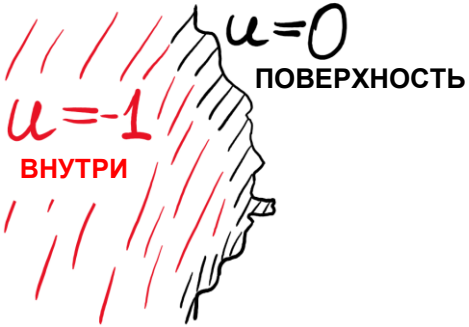
На вход: карты глубины

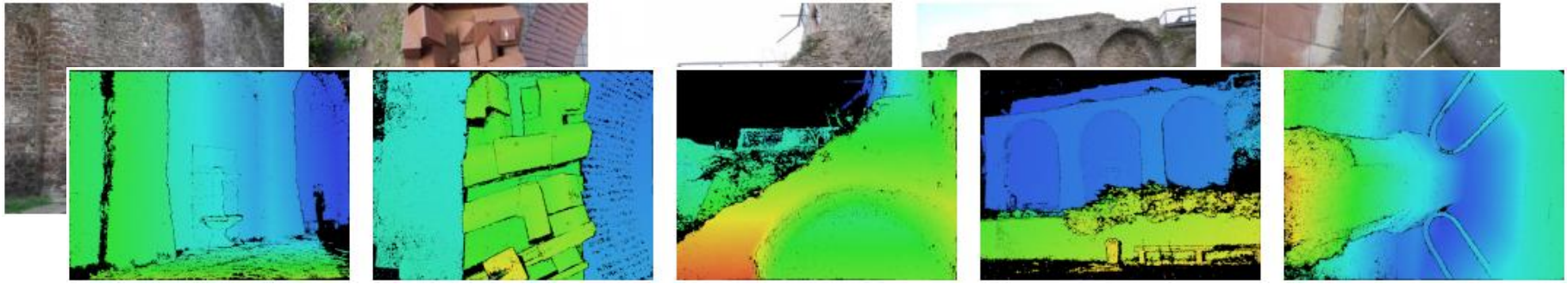
На выход: индикаторное скалярное поле



На вход: карты глубины

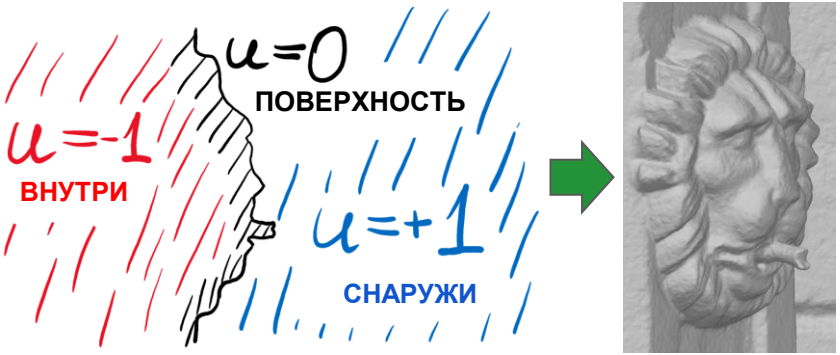
На выход: индикаторное скалярное поле

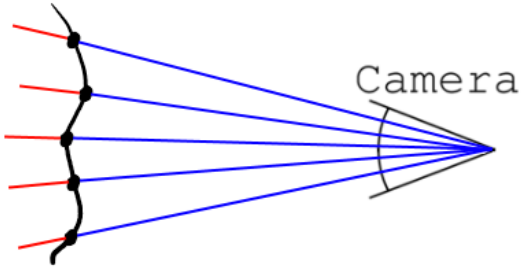




На вход: карты глубины

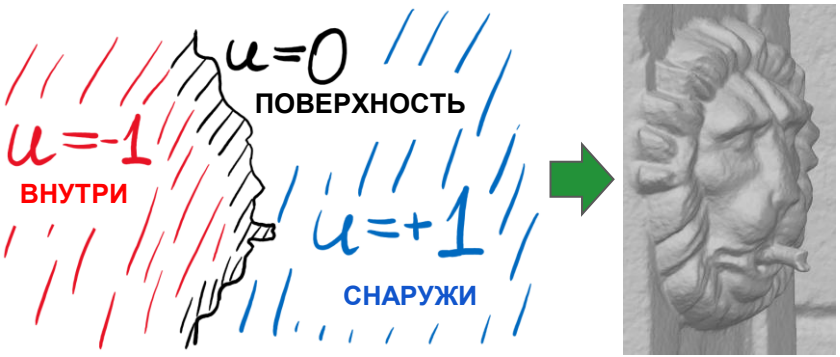
На выход: индикаторное скалярное поле

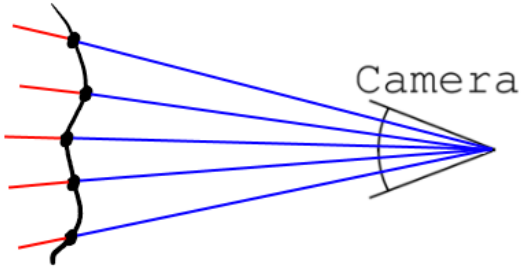


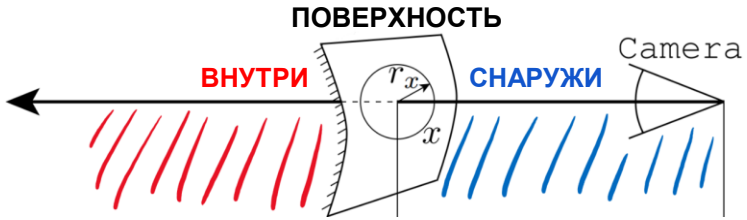


На вход: ~~карты глубины~~
индикаторное скалярное поле

На выход: индикаторное скалярное поле

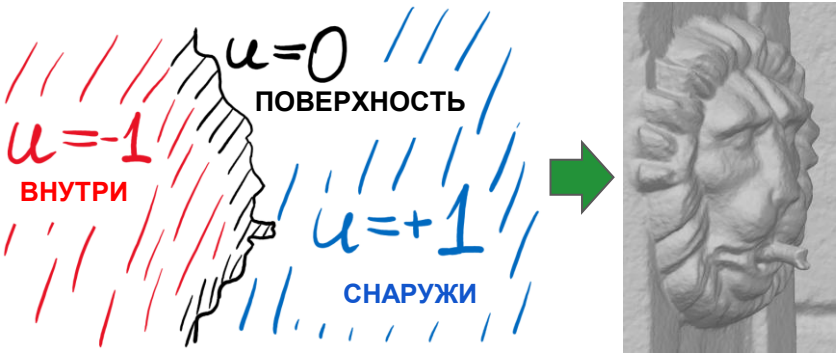


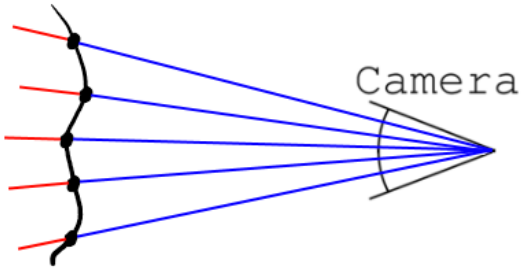


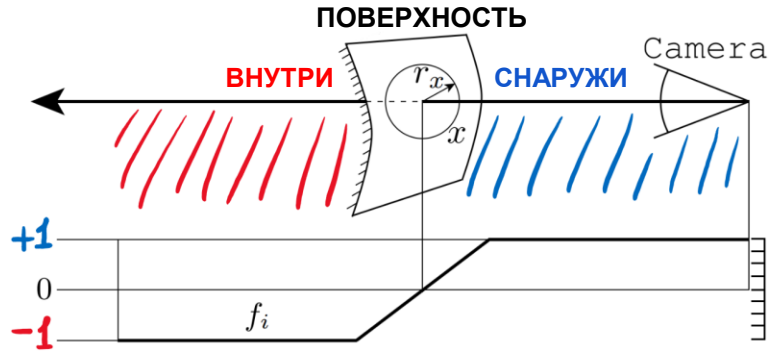


На вход: ~~карты глубины~~
индикаторное скалярное поле

На выход: индикаторное скалярное поле

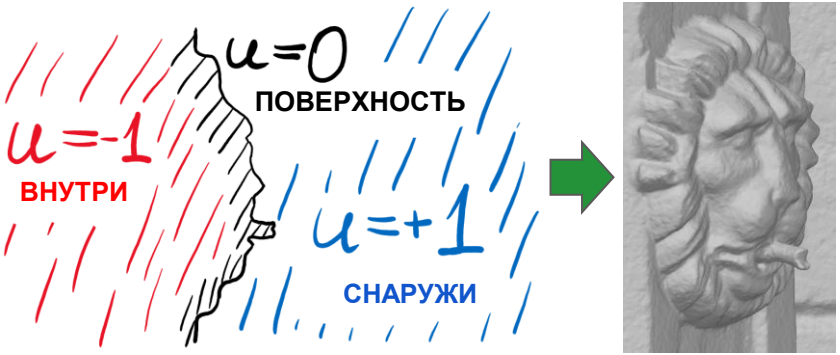


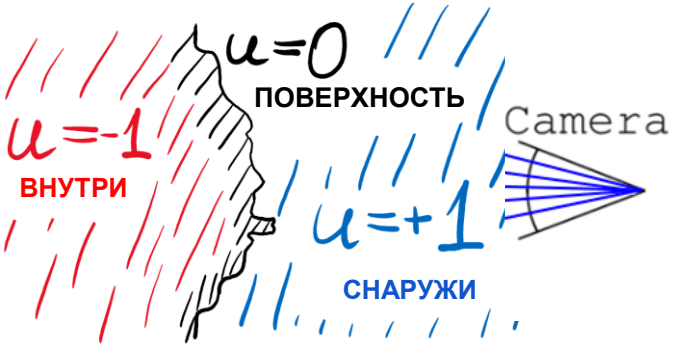


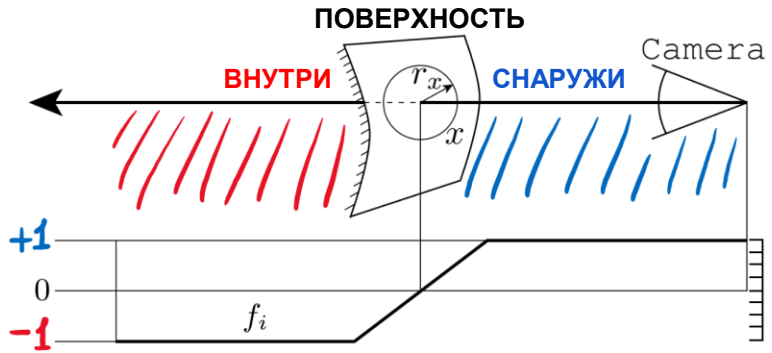


На вход: ~~карты глубины~~
индикаторное скалярное поле

На выход: индикаторное скалярное поле

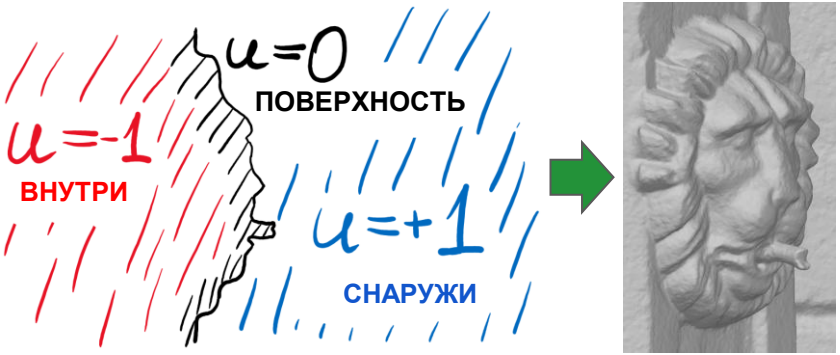


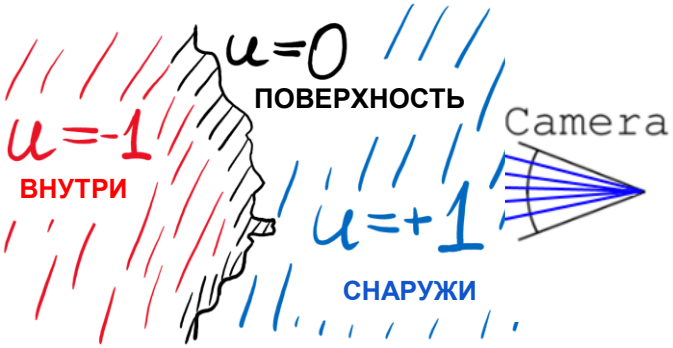


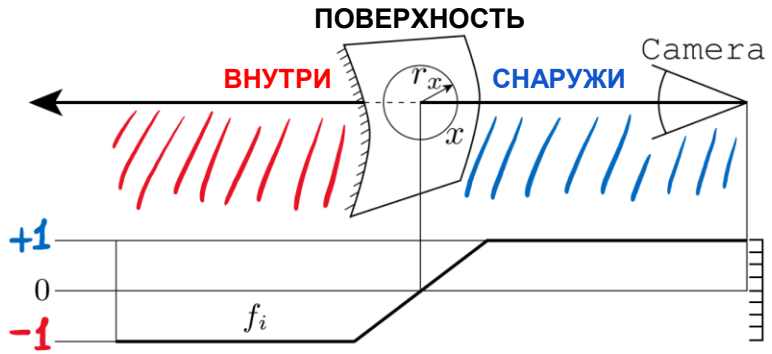


На вход: ~~карты глубины~~
индикаторное скалярное поле

На выход: индикаторное скалярное поле

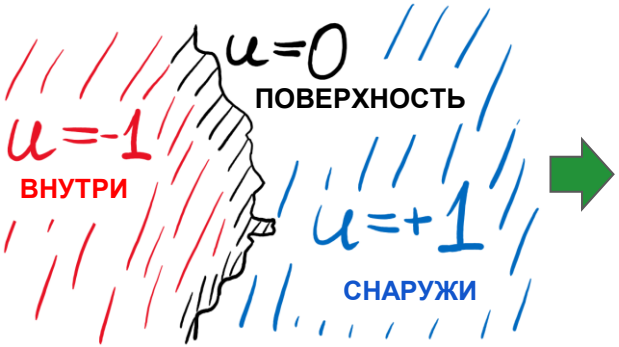




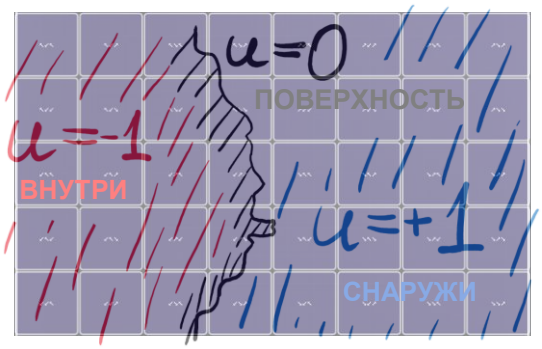


На вход: ~~карты глубины~~
индикаторное скалярное поле

На выход: индикаторное скалярное поле

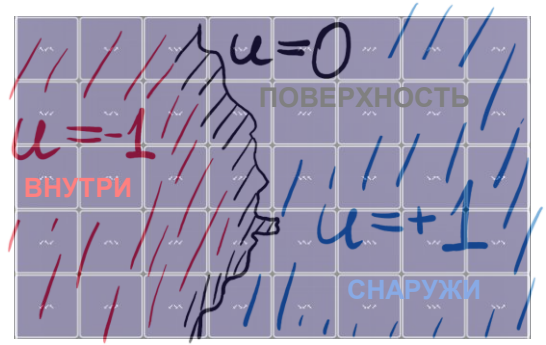


Как хранить индикаторное поле?
Как дискретизировать?

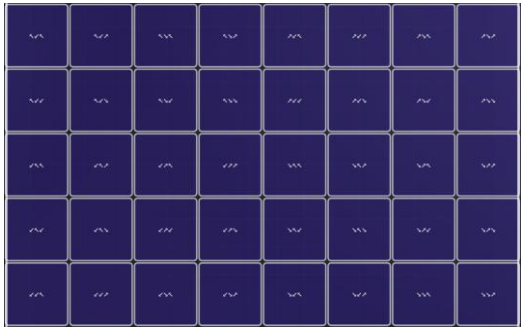


На вход: ~~карты глубины~~
индикаторное скалярное поле

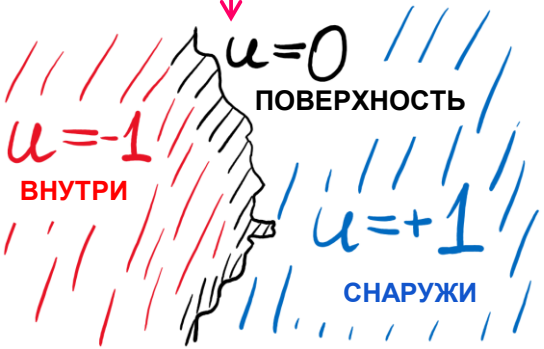
На выход: индикаторное скалярное поле



Как хранить индикаторное поле?
Как дискретизировать?
Регулярная решетка?



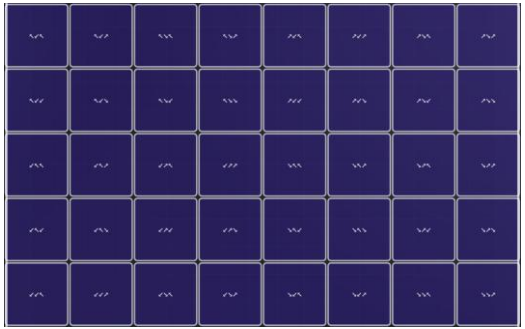
a teapot in



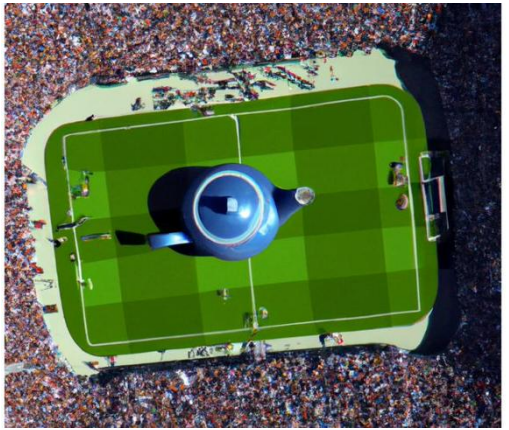
Как хранить индикаторное поле?

Как дискретизировать?

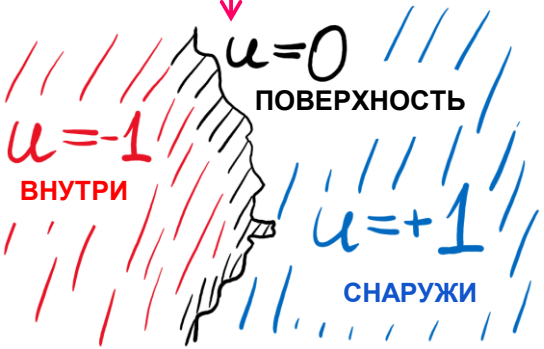
~~Регулярная решетка?~~



a teapot in



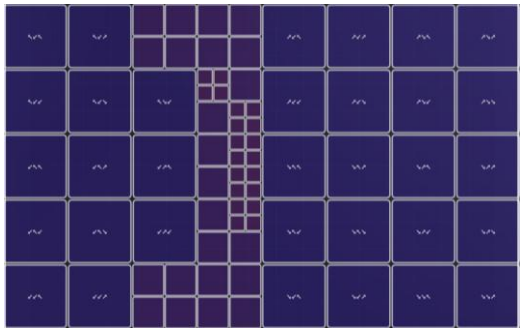
the stadium



Как хранить индикаторное поле?

Как дискретизировать?

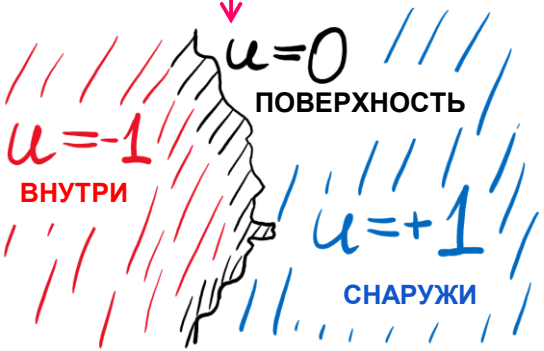
~~Регулярная решетка?~~



a teapot in

the stadium

Октодерево! Хранит индикаторное скалярное поле



Как хранить индикаторное поле?

Как дискретизировать?

~~Регулярная решетка?~~

Квадродерево

Интерактивное разбиение области на четыре дочерних квадрата.

В 2D это квадродерево. Октодерево — тот же принцип, но в 3D: куб делится на 8 кубов.

Сбросить | Разбить всё

Показать Z curve | Провести Z curve

Продолжить | Скрыть Z curve

Показать несбалансированные стороны

Максимальная глубина:

7

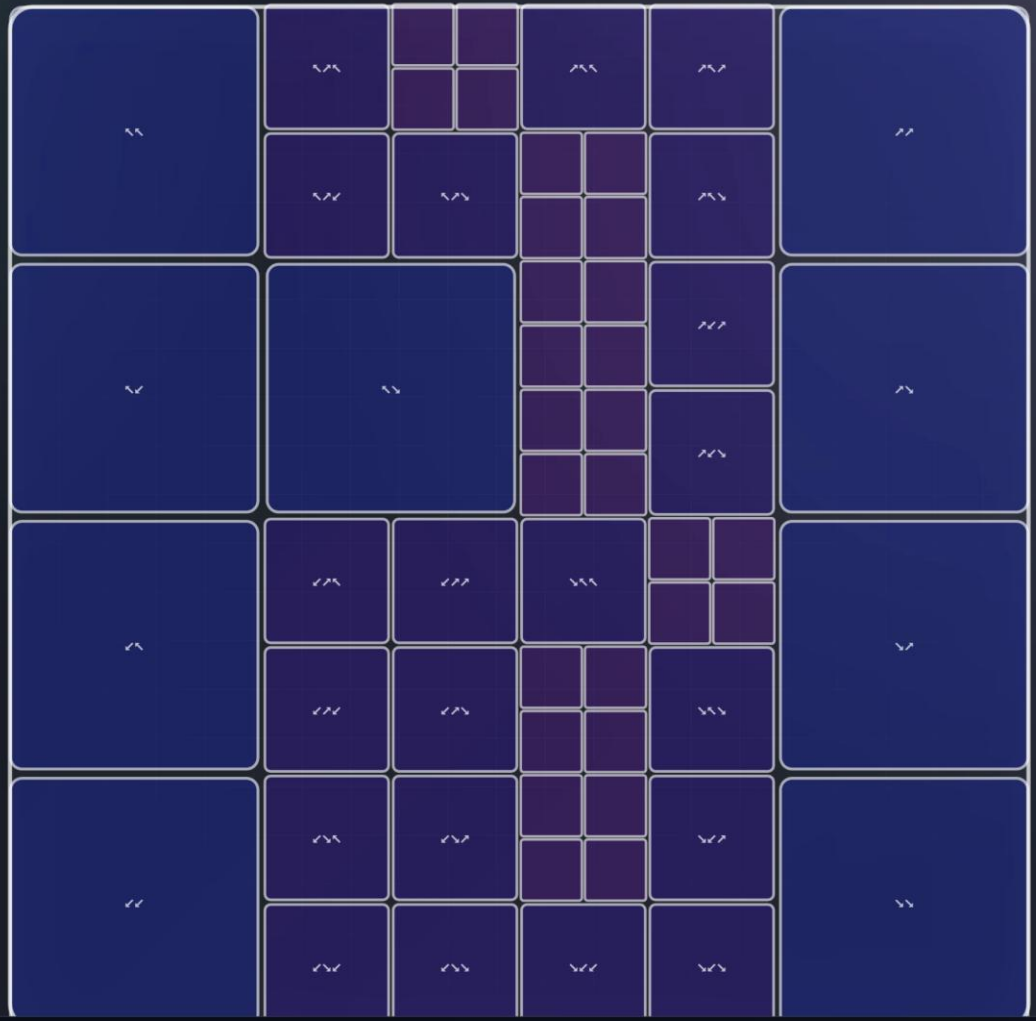
Листьев	58
Узлов всего	77
Текущая глубина	4
Точек Z-curve	58
Несбаланс. сторон	6

ЛКМ по квадрату — разбить его на 4 части.

ПКМ по квадрату — схлопнуть его и 3 соседних брата в родителя.

Z curve соединяет центры текущих листьев в Morton-порядке: ↖ → ↗ → ↙ → ↘.

Красные стороны — границы смежных листьев с разницей глубин больше 1.



Квадродерево

Интерактивное разбиение области на четыре дочерних квадрата.

В 2D это квадродерево. Октодерево — тот же принцип, но в 3D: куб делится на 8 кубов.

Сбросить | Разбить всё

Показать Z curve | Провести Z curve

Продолжить | Скрыть Z curve

Показать несбалансированные стороны

Максимальная глубина:

7

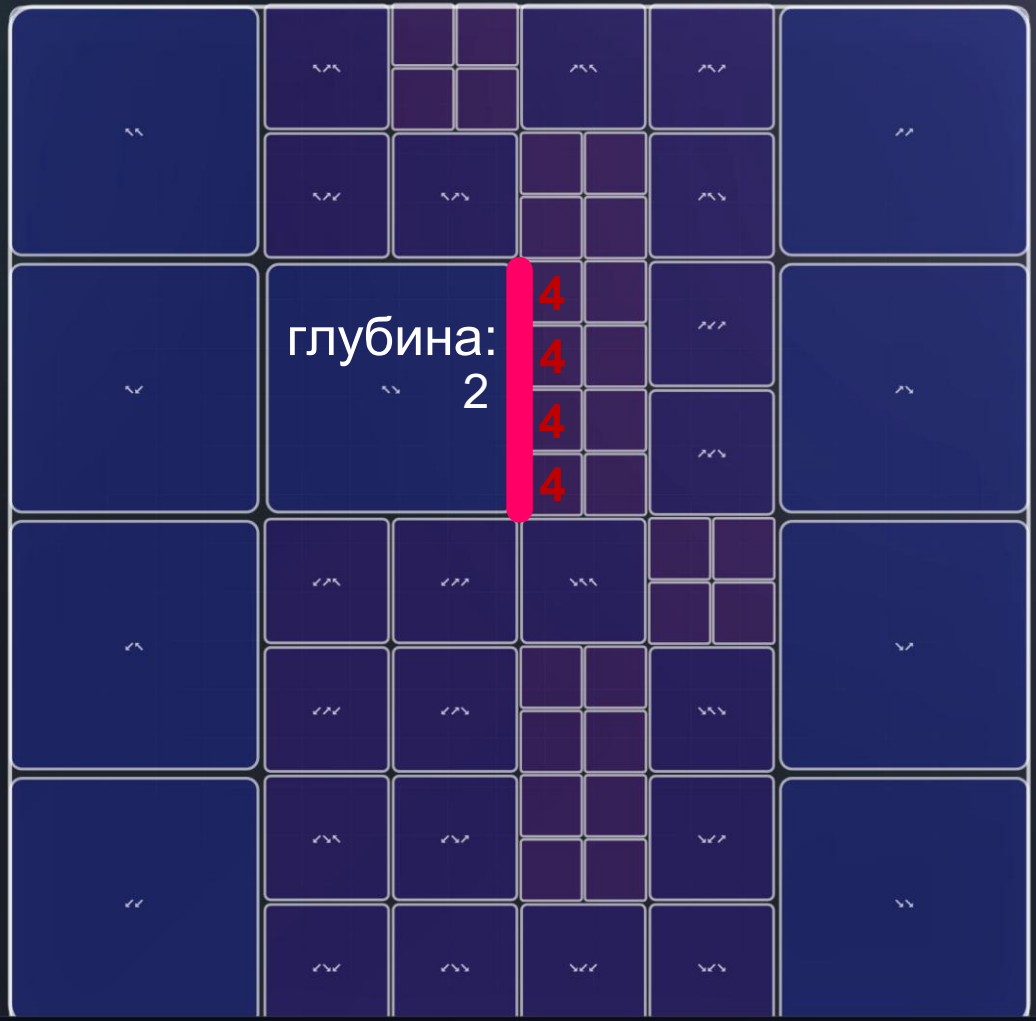
Листьев	58
Узлов всего	77
Текущая глубина	4
Точек Z-curve	58
Несбаланс. сторон	6

ЛКМ по квадрату — разбить его на 4 части.

ПКМ по квадрату — схлопнуть его и 3 соседних брата в родителя.

Z curve соединяет центры текущих листьев в Morton-порядке: ↖ → ↗ → ↙ → ↘.

Красные стороны — границы смежных листьев с разницей глубин больше 1.



Квадродерево

Интерактивное разбиение области на четыре дочерних квадрата.

В 2D это квадродерево. Октодерево — тот же принцип, но в 3D: куб делится на 8 кубов.

Сбросить | Разбить всё

Показать Z curve | Провести Z curve

Продолжить | Скрыть Z curve

Показать несбалансированные стороны

Максимальная глубина:

7

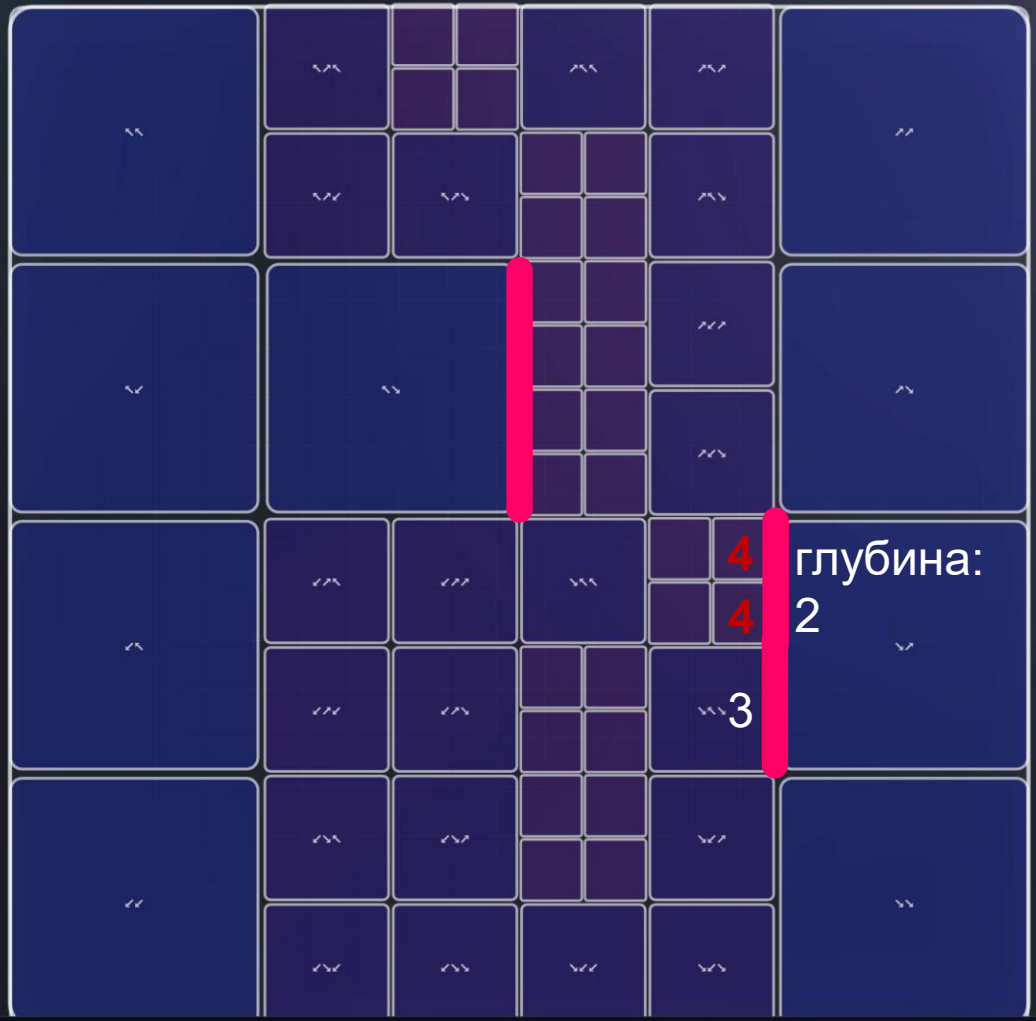
Листьев	58
Узлов всего	77
Текущая глубина	4
Точек Z-curve	58
Несбаланс. сторон	6

ЛКМ по квадрату — разбить его на 4 части.

ПКМ по квадрату — схлопнуть его и 3 соседних брата в родителя.

Z curve соединяет центры текущих листьев в Morton-порядке: ↖ → ↗ → ↙ → ↘.

Красные стороны — границы смежных листьев с разницей глубин больше 1.



Квадродерево

Интерактивное разбиение области на четыре дочерних квадрата.

В 2D это квадродерево. Октодерево — тот же принцип, но в 3D: куб делится на 8 кубов.

Сбросить | Разбить всё

Показать Z curve | Провести Z curve

Продолжить | Скрыть Z curve

Показать несбалансированные стороны

Максимальная глубина:

7

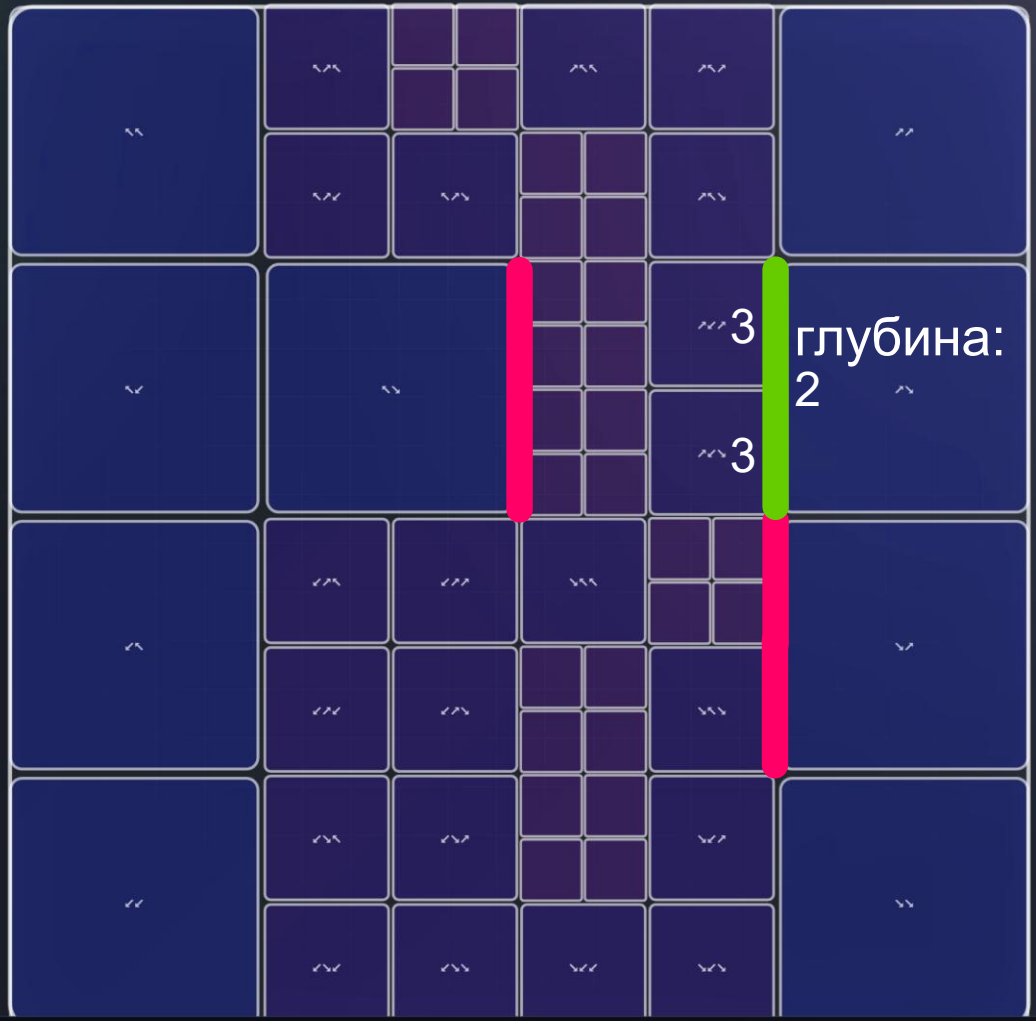
Листьев	58
Узлов всего	77
Текущая глубина	4
Точек Z-curve	58
Несбаланс. сторон	6

ЛКМ по квадрату — разбить его на 4 части.

ПКМ по квадрату — схлопнуть его и 3 соседних брата в родителя.

Z curve соединяет центры текущих листьев в Morton-порядке: ↖ → ↗ → ↙ → ↘.

Красные стороны — границы смежных листьев с разницей глубин больше 1.



Квадродерево

Интерактивное разбиение области на четыре дочерних квадрата.

В 2D это квадродерево. Октодерево — тот же принцип, но в 3D: куб делится на 8 кубов.

Сбросить | Разбить всё

Показать Z curve | Провести Z curve

Продолжить | Скрыть Z curve

Показать несбалансированные стороны

Максимальная глубина:

7

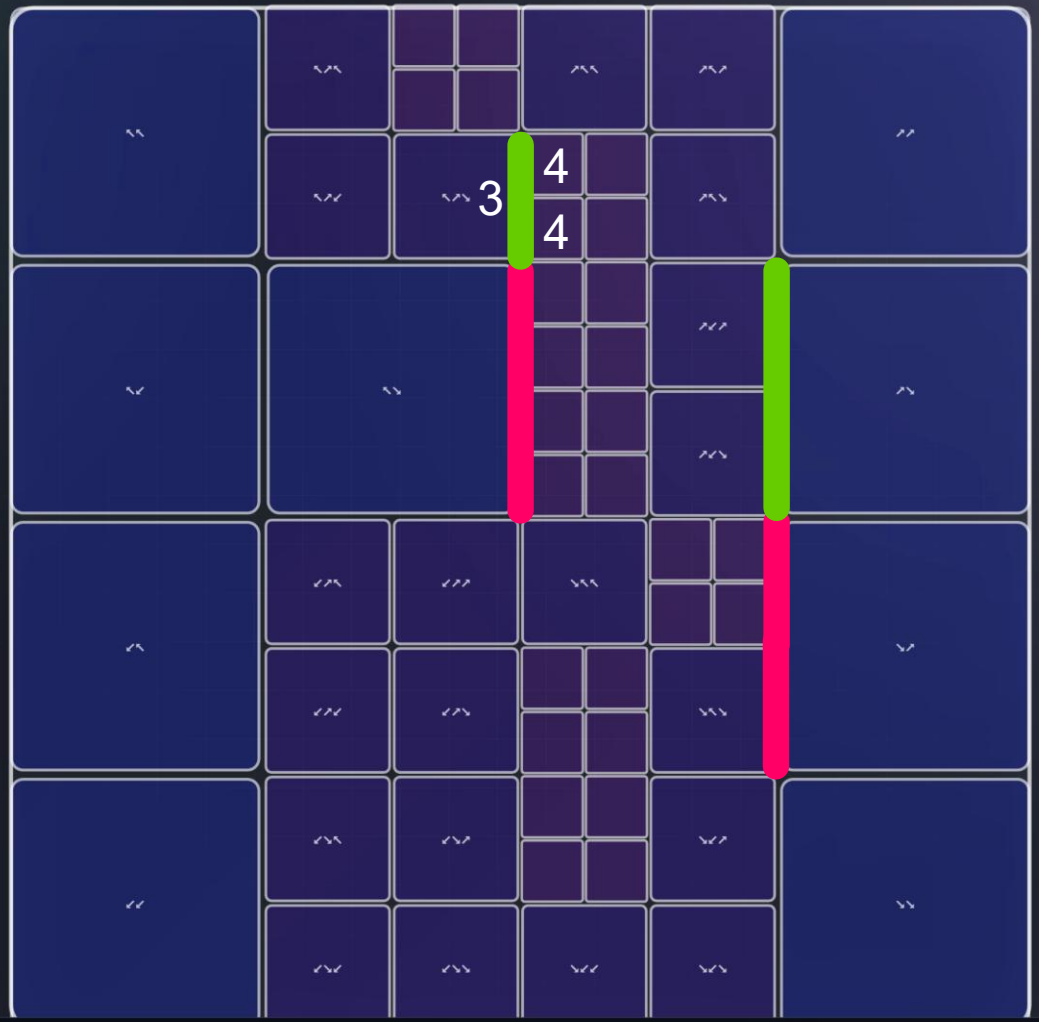
Листьев	58
Узлов всего	77
Текущая глубина	4
Точек Z-curve	58
Несбаланс. сторон	6

ЛКМ по квадрату — разбить его на 4 части.

ПКМ по квадрату — схлопнуть его и 3 соседних брата в родителя.

Z curve соединяет центры текущих листьев в Morton-порядке: ↖ → ↗ → ↙ → ↘.

Красные стороны — границы смежных листьев с разницей глубин больше 1.



Квадродерево

Интерактивное разбиение области на четыре дочерних квадрата.

В 2D это квадродерево. Октодерево — тот же принцип, но в 3D: куб делится на 8 кубов.

Сбросить | Разбить всё

Показать Z curve | Провести Z curve

Продолжить | Скрыть Z curve

Показать несбалансированные стороны

Максимальная глубина:

7

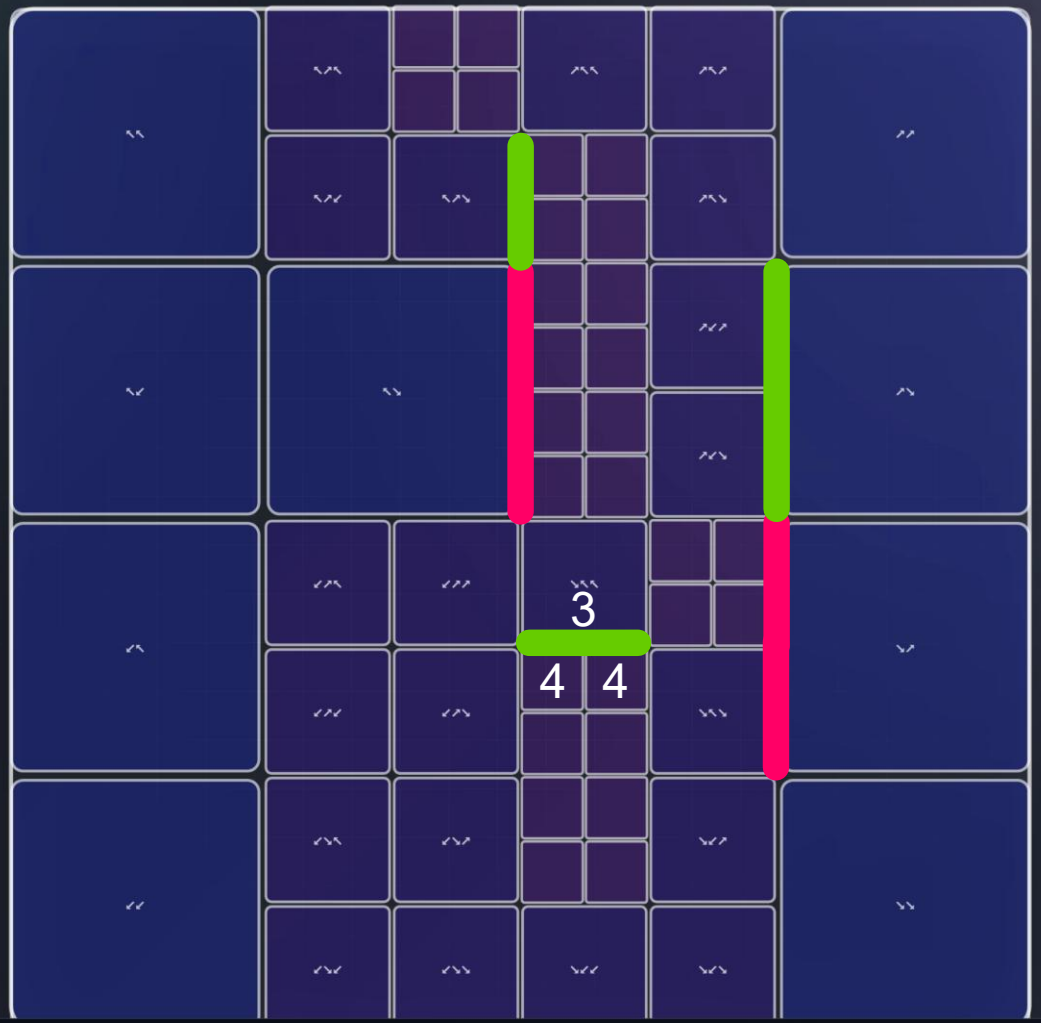
Листьев	58
Узлов всего	77
Текущая глубина	4
Точек Z-curve	58
Несбаланс. сторон	6

ЛКМ по квадрату — разбить его на 4 части.

ПКМ по квадрату — схлопнуть его и 3 соседних брата в родителя.

Z curve соединяет центры текущих листьев в Morton-порядке: ↖ → ↗ → ↙ → ↘.

Красные стороны — границы смежных листьев с разницей глубин больше 1.



Квадродерево

Интерактивное разбиение области на четыре дочерних квадрата.

В 2D это квадродерево. Октодерево — тот же принцип, но в 3D: куб делится на 8 кубов.

Сбросить | Разбить всё

Показать Z curve | Провести Z curve

Продолжить | Скрыть Z curve

Показать несбалансированные стороны

Максимальная глубина:

7

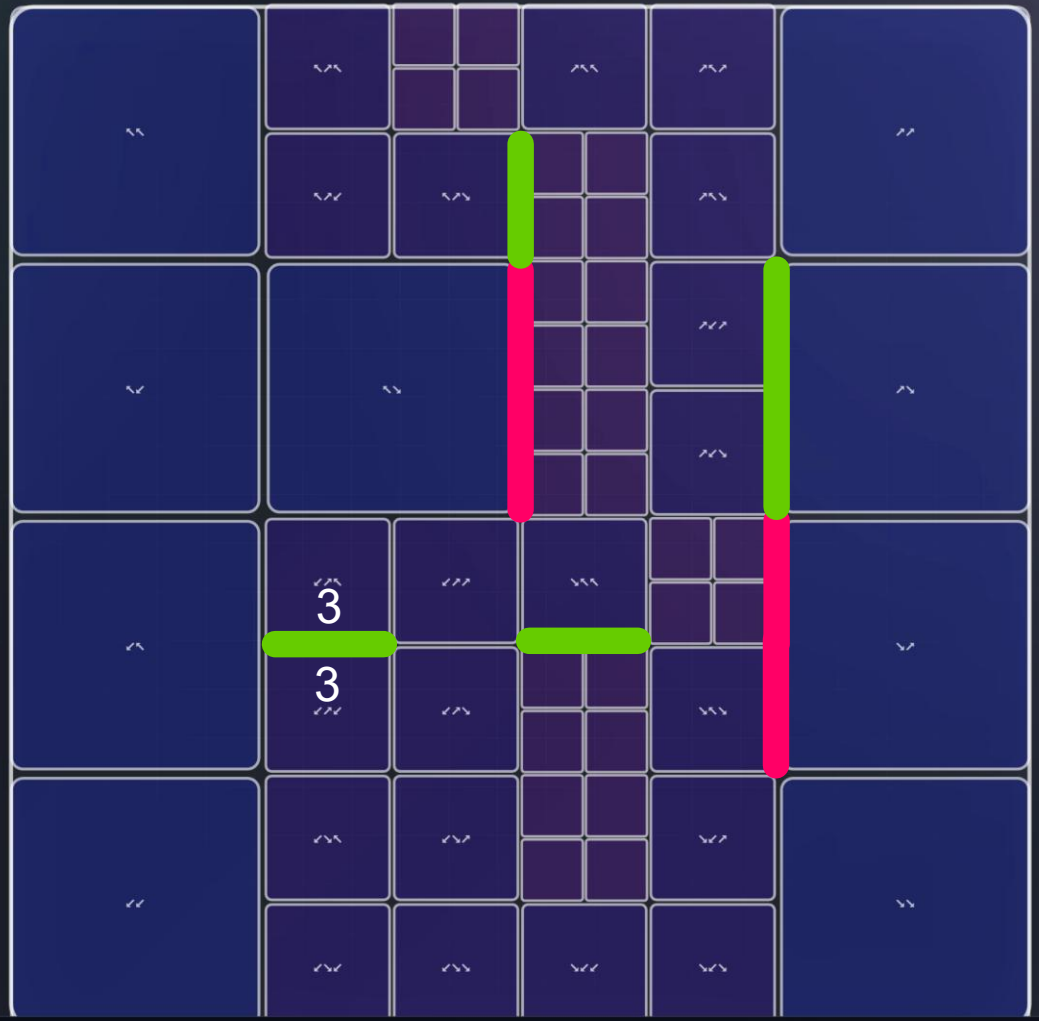
Листьев	58
Узлов всего	77
Текущая глубина	4
Точек Z-curve	58
Несбаланс. сторон	6

ЛКМ по квадрату — разбить его на 4 части.

ПКМ по квадрату — схлопнуть его и 3 соседних брата в родителя.

Z curve соединяет центры текущих листьев в Morton-порядке: ↖ → ↗ → ↙ → ↘.

Красные стороны — границы смежных листьев с разницей глубин больше 1.



Квадродерево

Интерактивное разбиение области на четыре дочерних квадрата.

В 2D это квадродерево. Октодерево — тот же принцип, но в 3D: куб делится на 8 кубов.

Сбросить | Разбить всё

Показать Z curve | Провести Z curve

Продолжить | Скрыть Z curve

Показать несбалансированные стороны

Максимальная глубина:

7

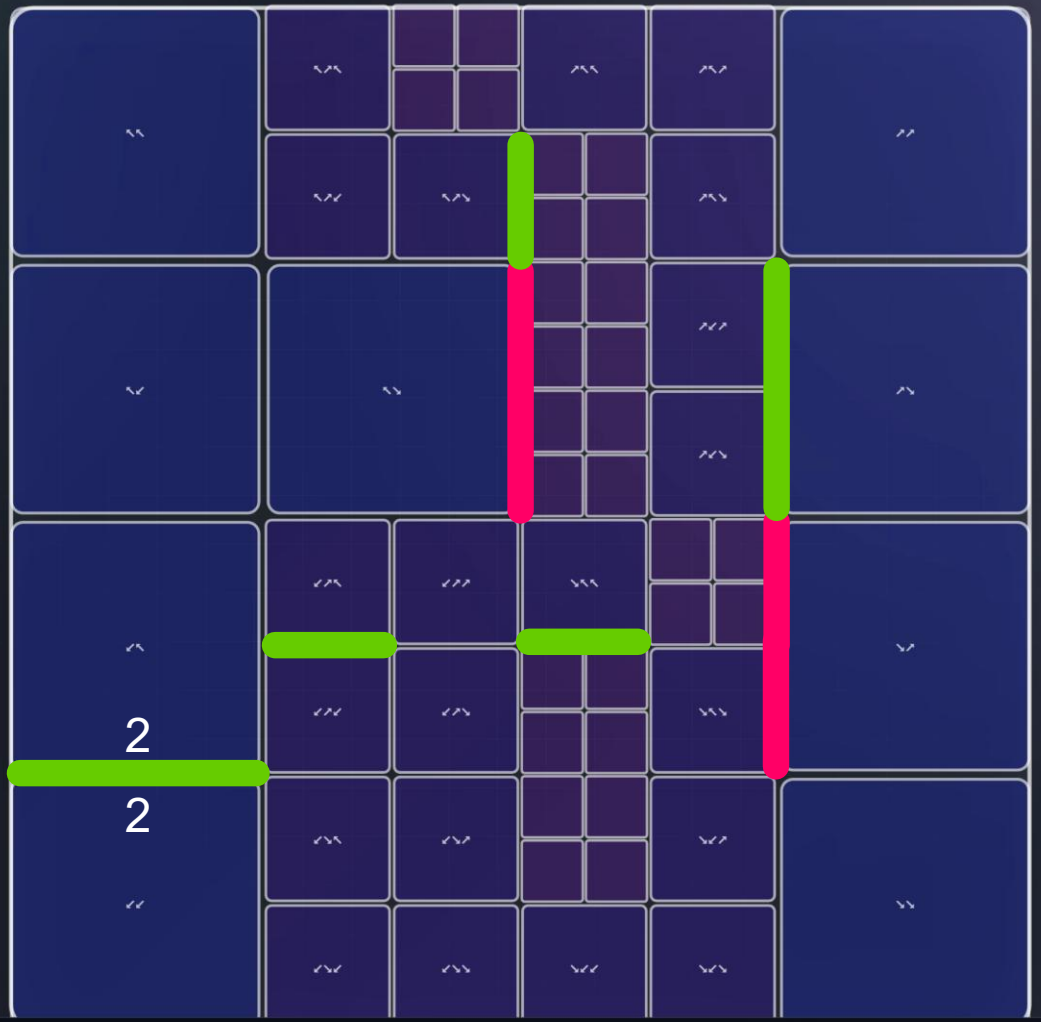
Листьев	58
Узлов всего	77
Текущая глубина	4
Точек Z-curve	58
Несбаланс. сторон	6

ЛКМ по квадрату — разбить его на 4 части.

ПКМ по квадрату — схлопнуть его и 3 соседних брата в родителя.

Z curve соединяет центры текущих листьев в Morton-порядке: ↖ → ↗ → ↙ → ↘.

Красные стороны — границы смежных листьев с разницей глубин больше 1.



Квадродерево

Интерактивное разбиение области на четыре дочерних квадрата.

В 2D это квадродерево. Октодерево — тот же принцип, но в 3D: куб делится на 8 кубов.

Сбросить | Разбить всё

Показать Z curve | Провести Z curve

Продолжить | Скрыть Z curve

Показать несбалансированные стороны

Максимальная глубина:

7

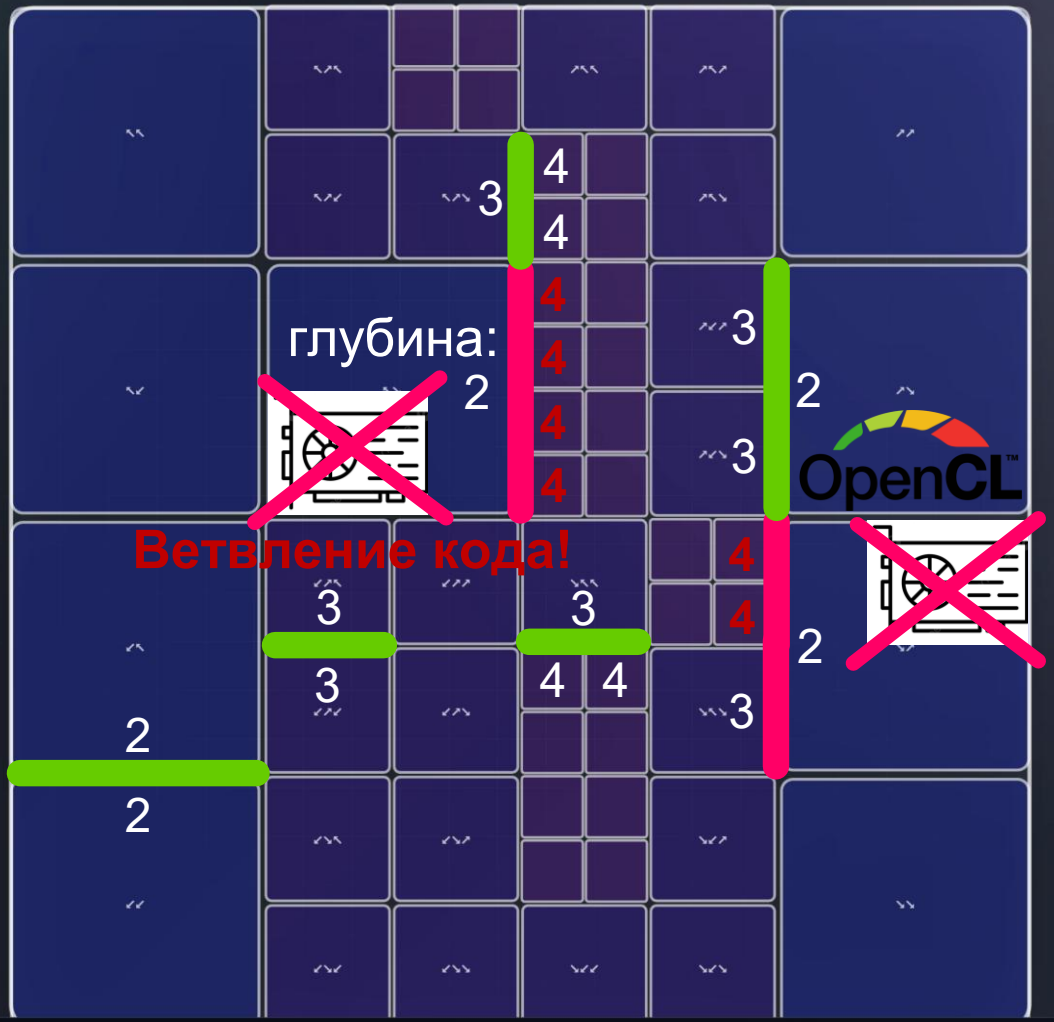
Листьев	58
Узлов всего	77
Текущая глубина	4
Точек Z-curve	58
Несбаланс. сторон	6

ЛКМ по квадрату — разбить его на 4 части.

ПКМ по квадрату — схлопнуть его и 3 соседних брата в родителя.

Z curve соединяет центры текущих листьев в Morton-порядке: ↖ → ↗ → ↙ → ↘.

Красные стороны — границы смежных листьев с разницей глубин больше 1.



Квадродерево

Интерактивное разбиение области на четыре дочерних квадрата.

В 2D это квадродерево. Октодерево — тот же принцип, но в 3D: куб делится на 8 кубов.

Сбросить | Разбить всё

Показать Z curve | Провести Z curve

Продолжить | Скрыть Z curve

Показать несбалансированные стороны

Максимальная глубина:

7

Листьев	64
Узлов всего	85
Текущая глубина	4
Точек Z-curve	64
Несбаланс. сторон	0

ЛКМ по квадрату — разбить его на 4 части.

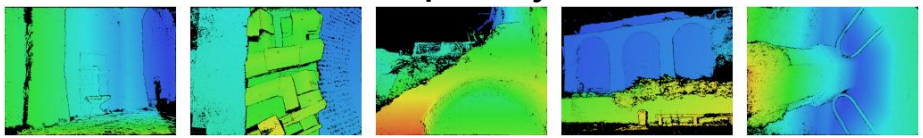
ПКМ по квадрату — схлопнуть его и 3 соседних брата в родителя.

Z curve соединяет центры текущих листьев в Morton-порядке: ↖ → ↗ → ↙ → ↘.

Красные стороны — границы смежных листьев с разницей глубин больше 1.

Итого

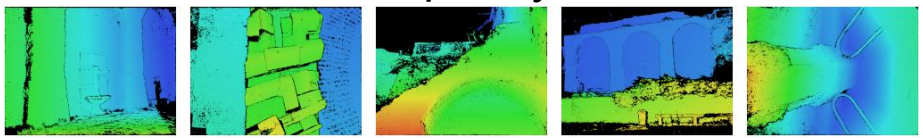
1) Есть множество карт глубины

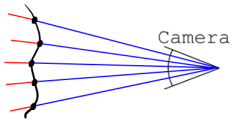


A Globally Optimal Algorithm for Robust TV-L1 Range Image Integration, **Zach et al., 2007**

Итого

1) Есть множество карт глубины

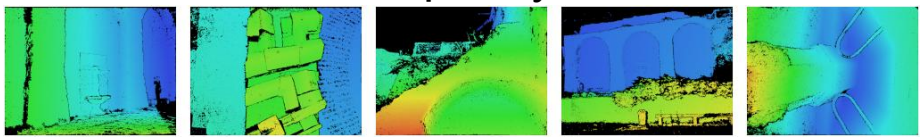


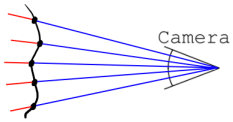


A Globally Optimal Algorithm for Robust TV-L1 Range Image Integration, **Zach et al., 2007**

Итого

1) Есть множество карт глубины



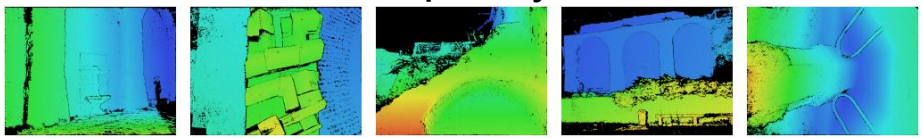


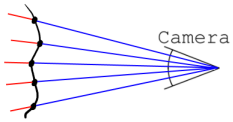
2) Множество точек всех карт глубины - порождает **адаптивное** октодерево (дискретизация пространства)

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

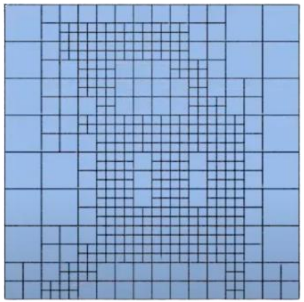
Итого

1) Есть множество карт глубины





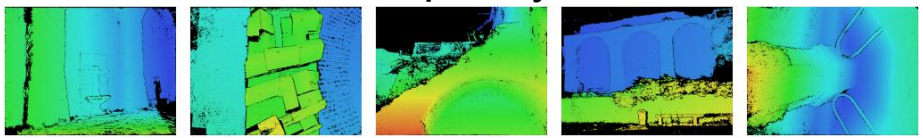
2) Множество точек всех карт глубины - порождает **адаптивное** октодерево (дискретизация пространства)

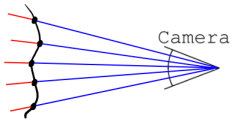


A Globally Optimal Algorithm for Robust TV-L1 Range Image Integration, **Zach et al., 2007**

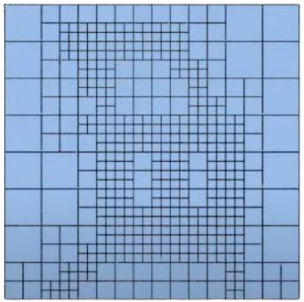
Итого

1) Есть множество карт глубины





2) Множество точек всех карт глубины - порождает **адаптивное** октодерево (дискретизация пространства)



3) Оптимизировали индикаторное поле (много итераций)

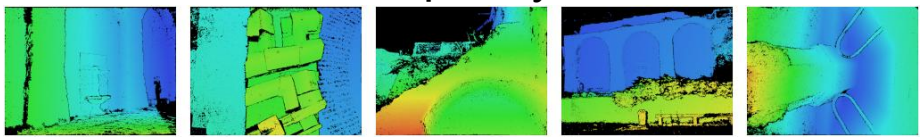
TV-L^1:

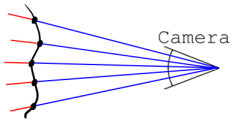
$$E = \int_\Omega \left\{ |\nabla u| + \lambda \sum_{i \in \mathcal{I}(\vec{x})} w_i(\vec{x}) |u - f_i| \right\} d\vec{x}$$

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

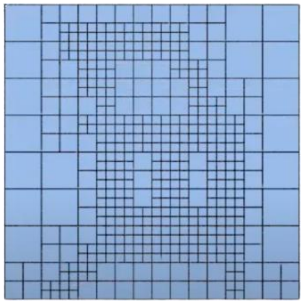
Итого

1) Есть множество карт глубины





2) Множество точек всех карт глубины - порождает **адаптивное** октодерево (дискретизация пространства)

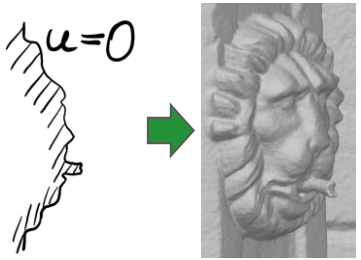


3) Оптимизировали индикаторное поле (много итераций)

TV-L^1:

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

4) Извлекли поверхность маршировкой кубов

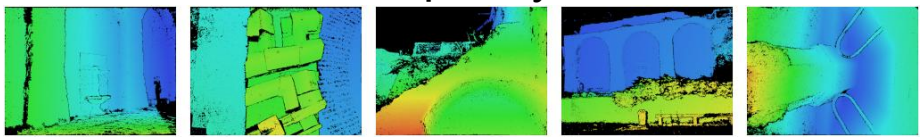


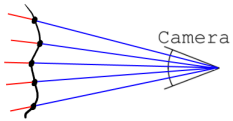
A Globally Optimal Algorithm for Robust TV-L1 Range Image Integration, **Zach et al., 2007**

Как ускорить?

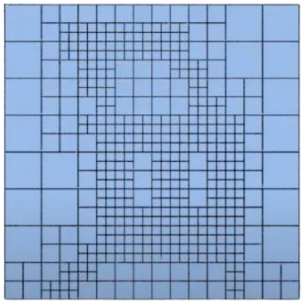
Итого

1) Есть множество карт глубины





2) Множество точек всех карт глубины - порождает **адаптивное** октодерево (дискретизация пространства)

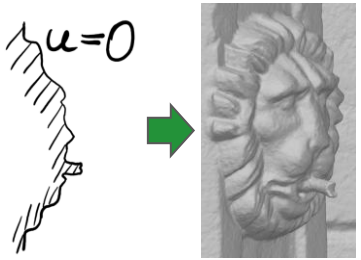


3) Оптимизировали индикаторное поле (**много итераций**)

TV-L^1:

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

4) Извлекли поверхность маршировкой кубов

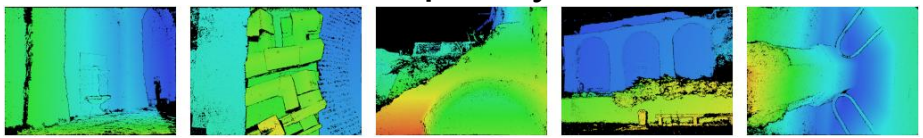


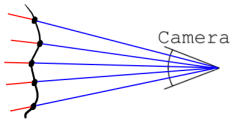
A Globally Optimal Algorithm for Robust TV-L1 Range Image Integration, **Zach et al., 2007**

Итого

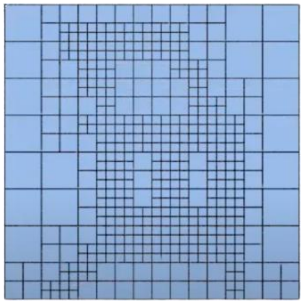
Как ускорить?

1) Есть множество карт глубины





2) Множество точек всех карт глубины - порождает **адаптивное** октодерево (дискретизация пространства)

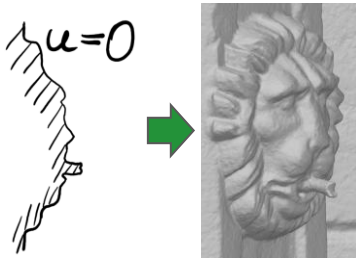


3) Оптимизировали индикаторное поле (**много итераций**)

TV-L^1:

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

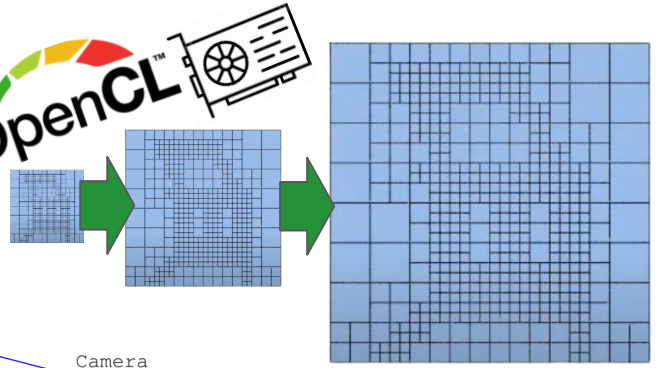
4) Извлекли поверхность маршировкой кубов



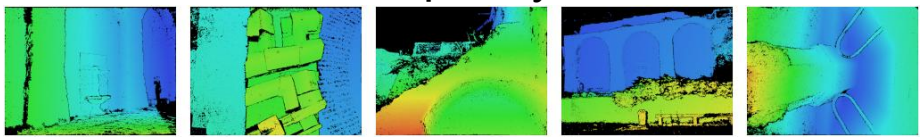
A Globally Optimal Algorithm for Robust TV-L1 Range Image Integration, **Zach et al., 2007**

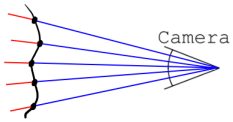
Итого

Ускоряем: GPU + Coarse-to-Fine

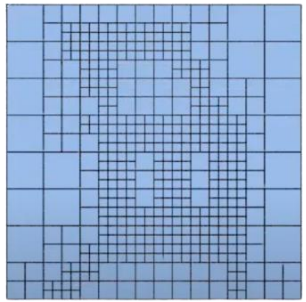


1) Есть множество карт глубины





2) Множество точек всех карт глубины - порождает **адаптивное** октодерево (дискретизация пространства)

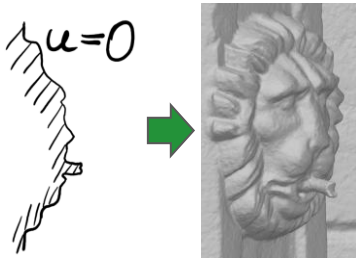


3) Оптимизировали индикаторное поле (**много итераций**)

TV-L^1:

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

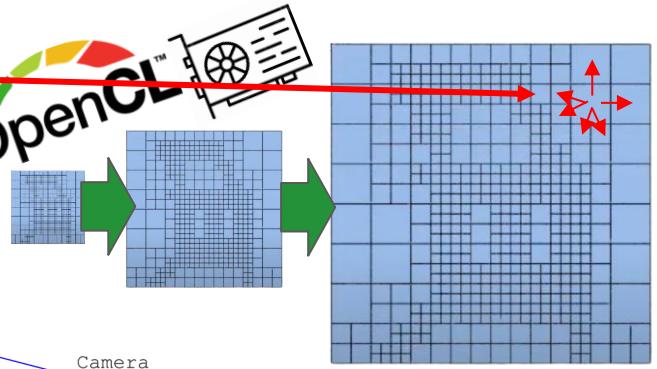
4) Извлекли поверхность маршировкой кубов



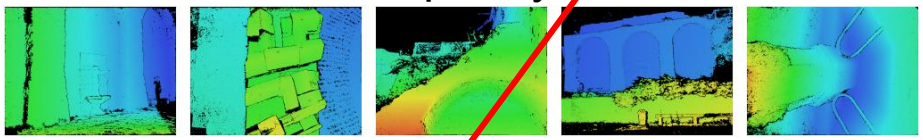
A Globally Optimal Algorithm for Robust TV-L1 Range Image Integration, **Zach et al., 2007**

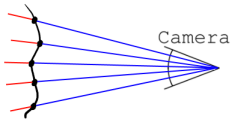
Итого

Нужно обращаться к соседним вокселям

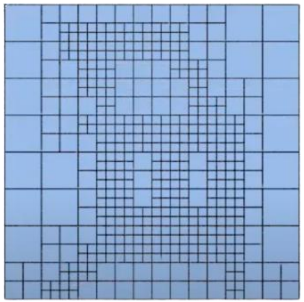


1) Есть множество карт глубины





2) Множество точек всех карт глубины - порождает **адаптивное** октодерево (дискретизация пространства)

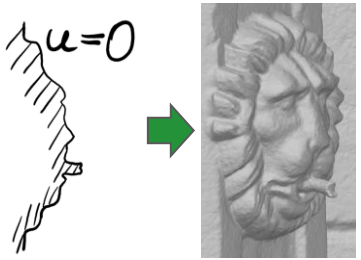


3) Оптимизировали индикаторное поле (**много итераций**)

TV-L^1:

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

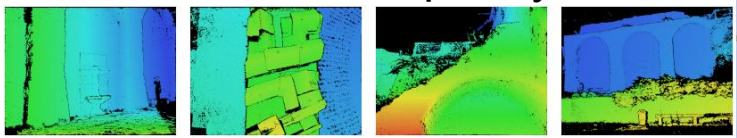
4) Извлекли поверхность маршировкой кубов

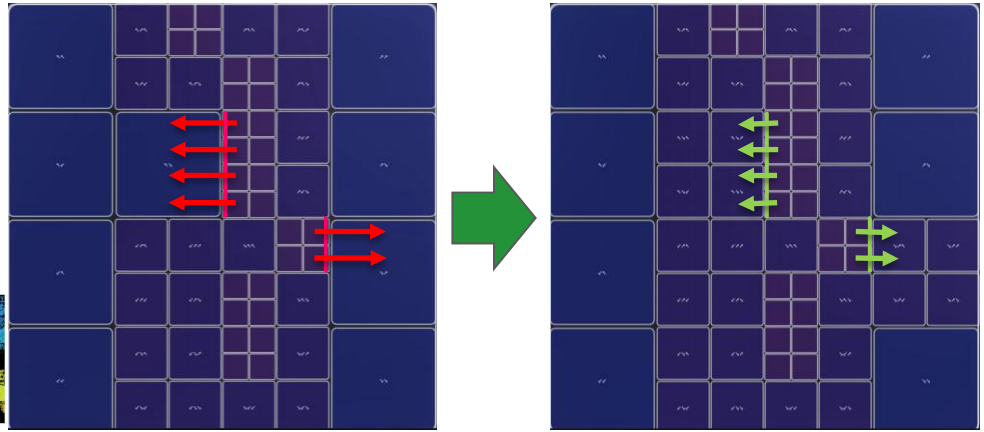


A Globally Optimal Algorithm for Robust TV-L1 Range Image Integration, **Zach et al., 2007**

Итого

1) Есть множество карт глубины



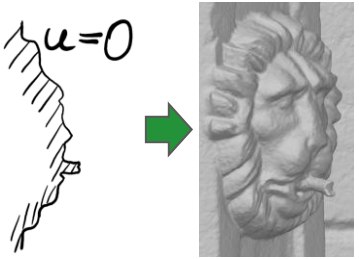


2) Множество точек всех карт глубины - порождает **адаптивное** октодерево (дискретизация пространства)
3) Балансируем октодерево 2:1 (по каждой стороне макс. два соседа)
4) Оптимизировали индикаторное поле (**много итераций + Coarse-to-Fine**)

TV-L^1:

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

5) Извлекли поверхность маршировкой кубов

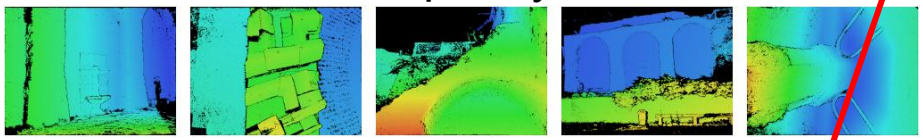


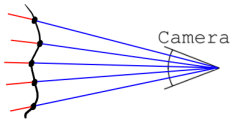
[A Globally Optimal Algorithm for Robust TV-L1 Range Image Integration, **Zach et al., 2007**]

Итого

Как компактно хранить карты глубины?
При этом чтобы быстро с ними сверяться!

1) Есть множество карт глубины





2) Множество точек всех карт глубины - порождает **адаптивное** октодерево (дискретизация пространства)
3) Балансируем октодерево 2:1 (по каждой стороне макс. два соседа)
4) Оптимизировали индикаторное поле (**много итераций + Coarse-to-Fine**)

TV-L^1:

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

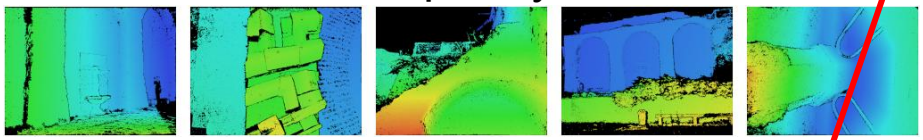
5) Извлекли поверхность маршировкой кубов

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

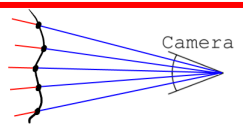
Итого

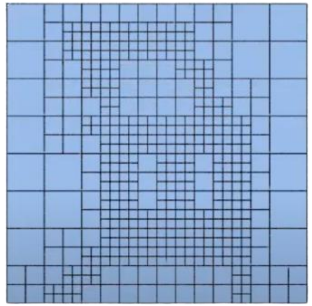
Как компактно хранить карты глубины?
При этом чтобы быстро с ними сверяться!

1) Есть множество карт глубины



OpenCL
Преобразуем в гистограммы!





2) Множество точек всех карт глубины - порождает **адаптивное** октодерево (дискретизация пространства)
3) Балансируем октодерево 2:1 (по каждой стороне макс. два соседа)
4) Оптимизировали индикаторное поле (**много итераций + Coarse-to-Fine**)

TV-L^1:

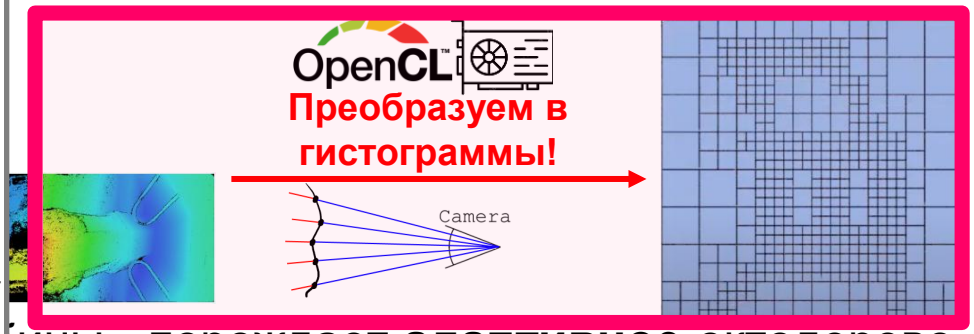
$$E = \int_\Omega \left\{ |\nabla u| + \lambda \sum_{i \in \mathcal{I}(\vec{x})} w_i(\vec{x}) |u - f_i| \right\} d\vec{x}$$

5) Извлекли поверхность маршировкой кубов

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

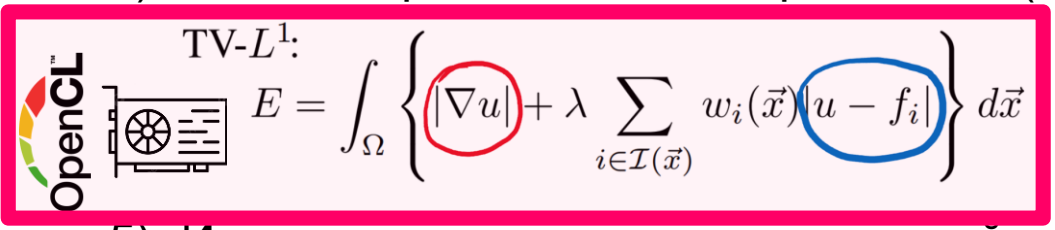
		Duration
Octree generation	Density estimation	74.6 min
	Balancing	7.9 min
17% →	Histograms	782.4 min
Surface comp. 80%	Dual grid generation	19.9 min
	Energy minimization	3678.0 min
	Dual contouring	16.3 min
Other		23.5 min
Total		4602.9 min

Global, Dense Multiscale Reconstruction for a Billion Points, **Ummenhofer et. al., 2017**

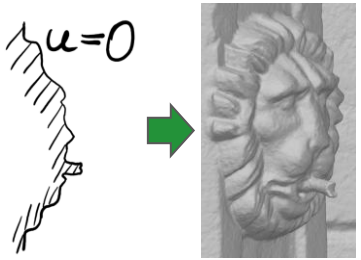


ины - порождает **адаптивное** октодерево

3) Балансируем октодерево 2:1 (по каждой стороне макс. два соседа)
4) Оптимизировали индикаторное поле (**много итераций + Coarse-to-Fine**)



5) Извлекли поверхность маршировкой кубов



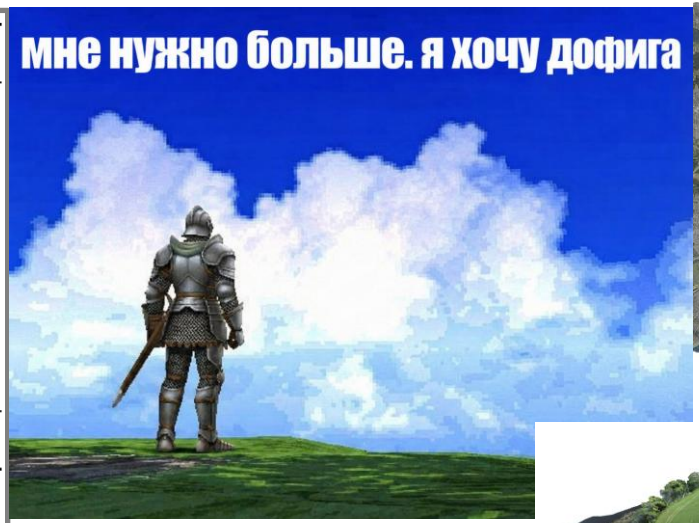
A Globally Optimal Algorithm for Robust TV-L1 Range Image Integration, **Zach et al., 2007**

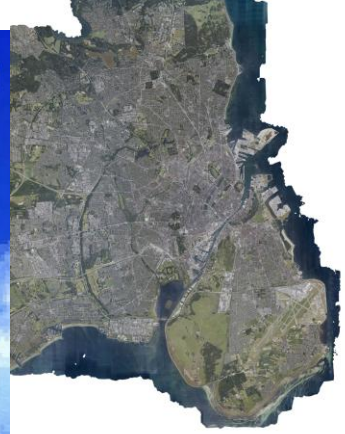
		Duration
Octree generation	Density estimation	74.6 min
	Balancing	7.9 min
	Histograms	782.4 min
Surface comp.	Dual grid generation	19.9 min
	Energy minimization	3678.0 min
	Dual contouring	16.3 min
Other		23.5 min
Total		4602.9 min

17%

80%

Global, Dense Multiscale Reconstruction **for a Billion Points**, Ummenhofer et. al., 2017





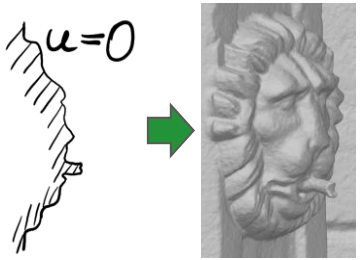
Мало! Нужно out-of-core!

3) Балансируем октодерево 2:1 (по каждой стороне макс.
4) Оптимизировали индикаторное поле (**много итераций**

OpenCL

TV-L^1:

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



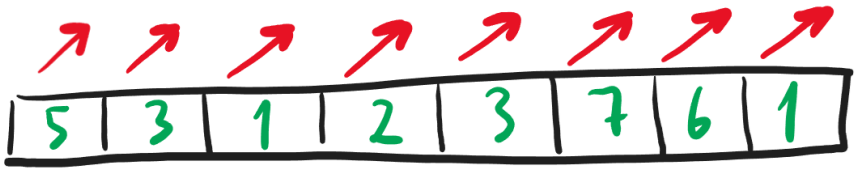
5) Извлекли поверхность маршировкой кубов

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

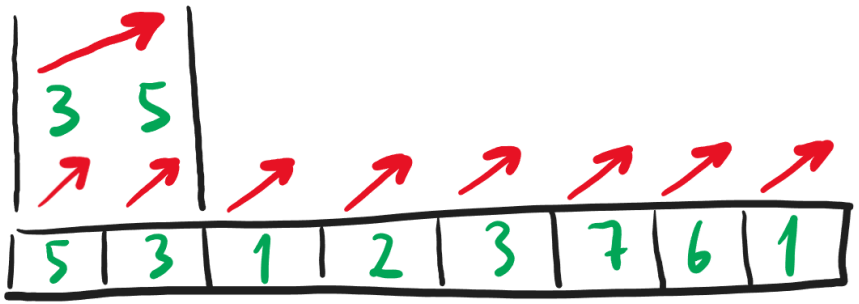

Алгоритм во внешней памяти
(out-of-core свойство)
K-way merge sort

Out-of-Core сортировка (**k-way merge sort**)

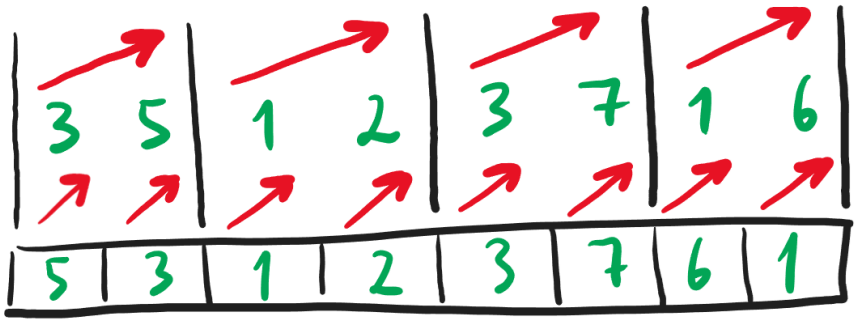
Out-of-Core сортировка (**k-way merge sort**)



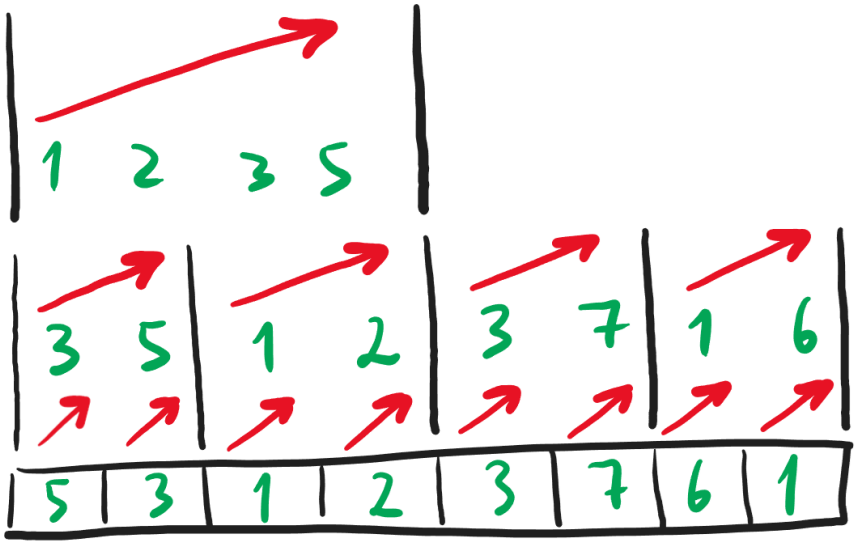
Out-of-Core сортировка (**k-way merge sort**)



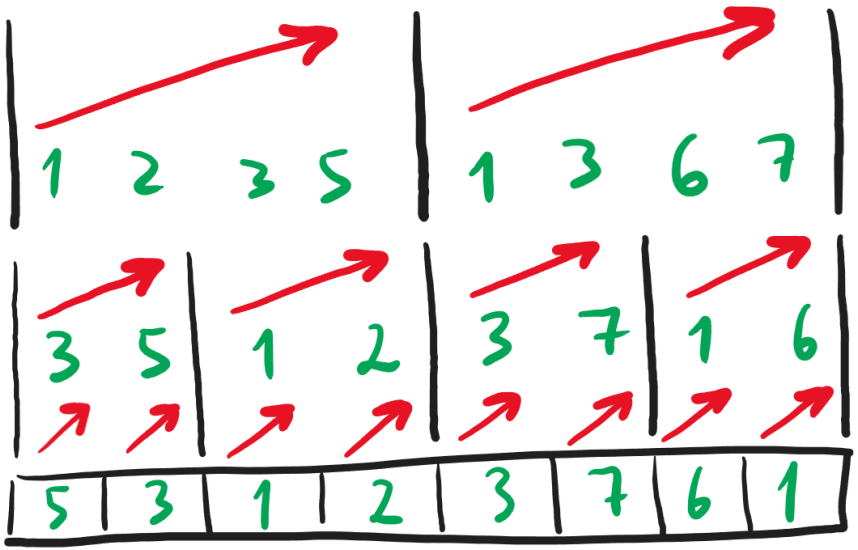
Out-of-Core сортировка (**k-way merge sort**)



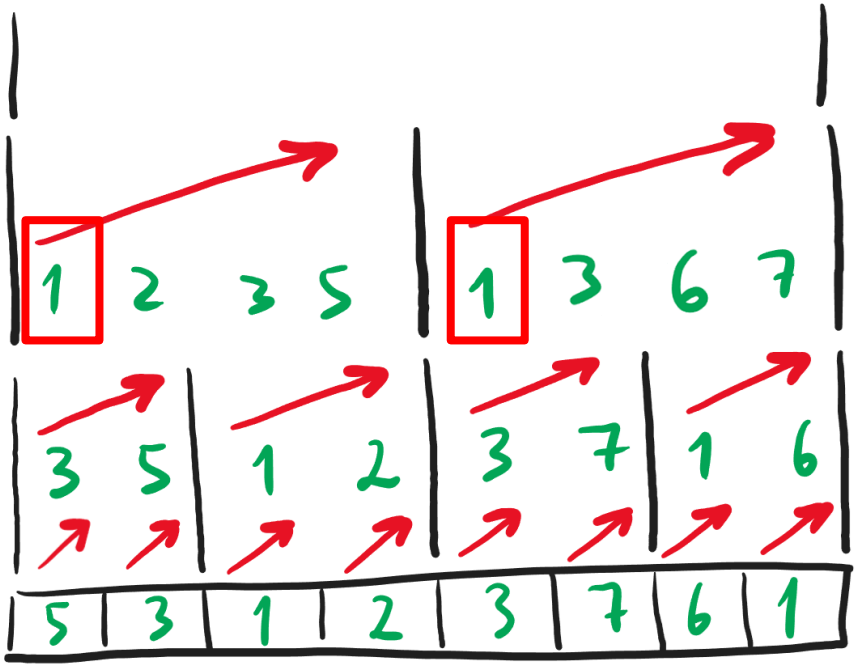
Out-of-Core сортировка (**k-way merge sort**)



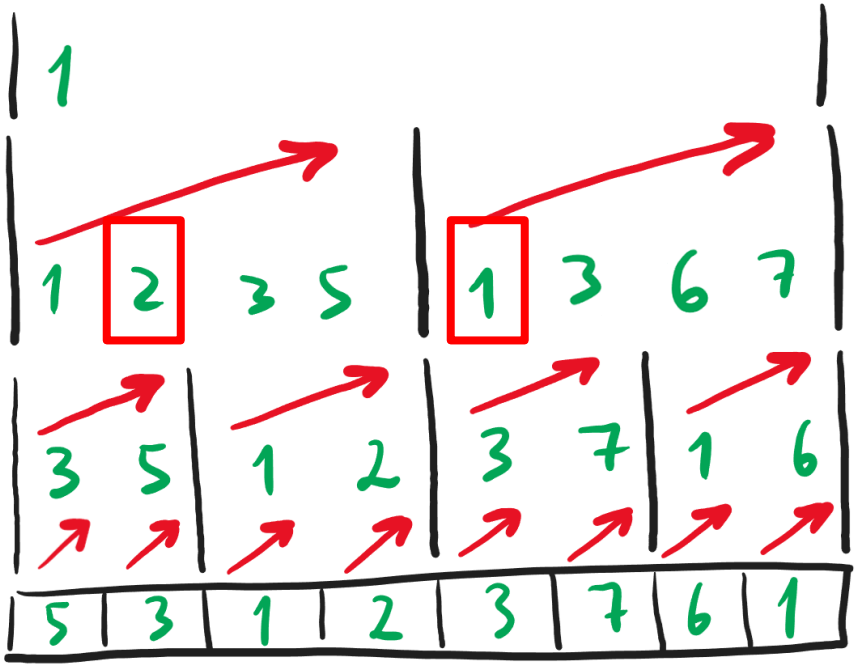
Out-of-Core сортировка (**k-way merge sort**)



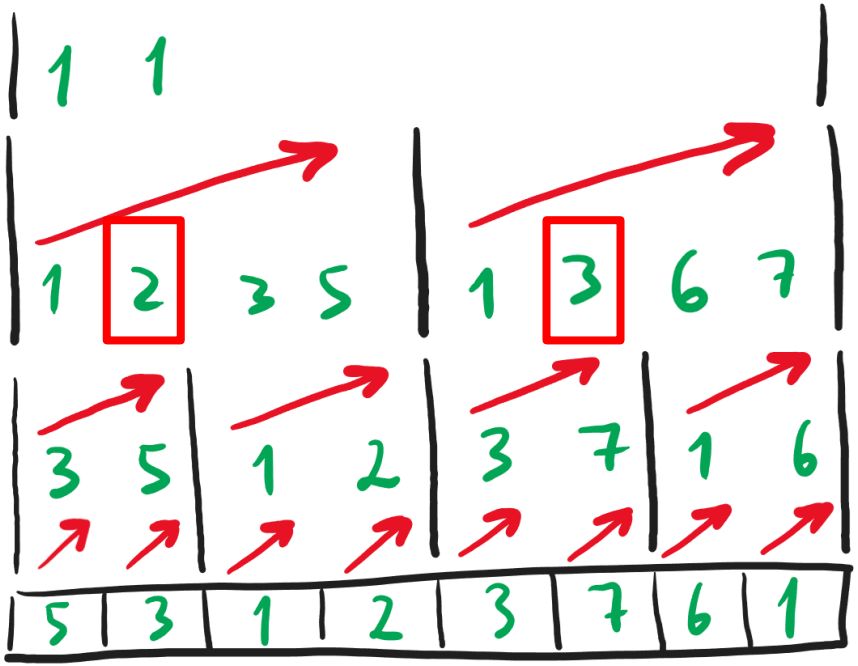
Out-of-Core сортировка (**k-way merge sort**)



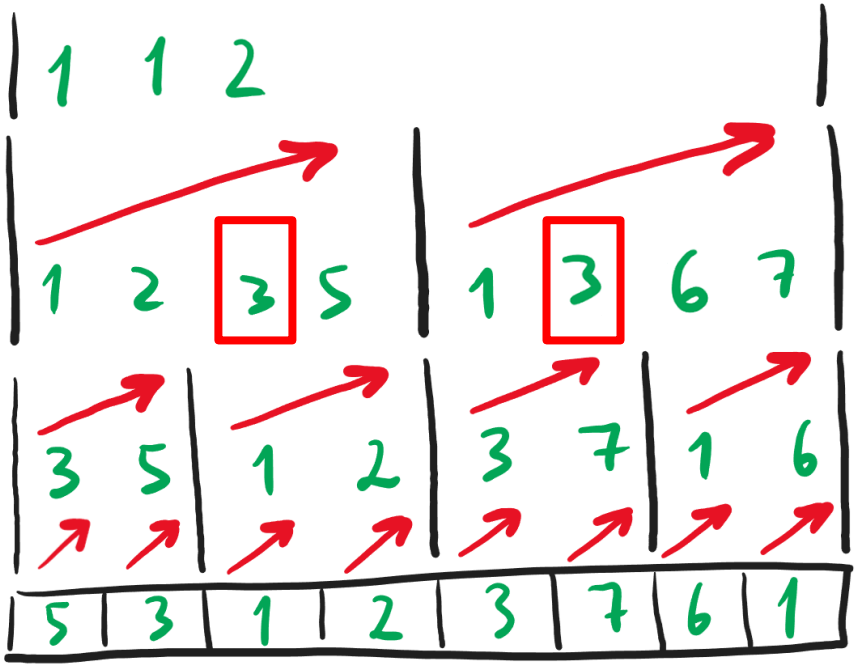
Out-of-Core сортировка (**k-way merge sort**)



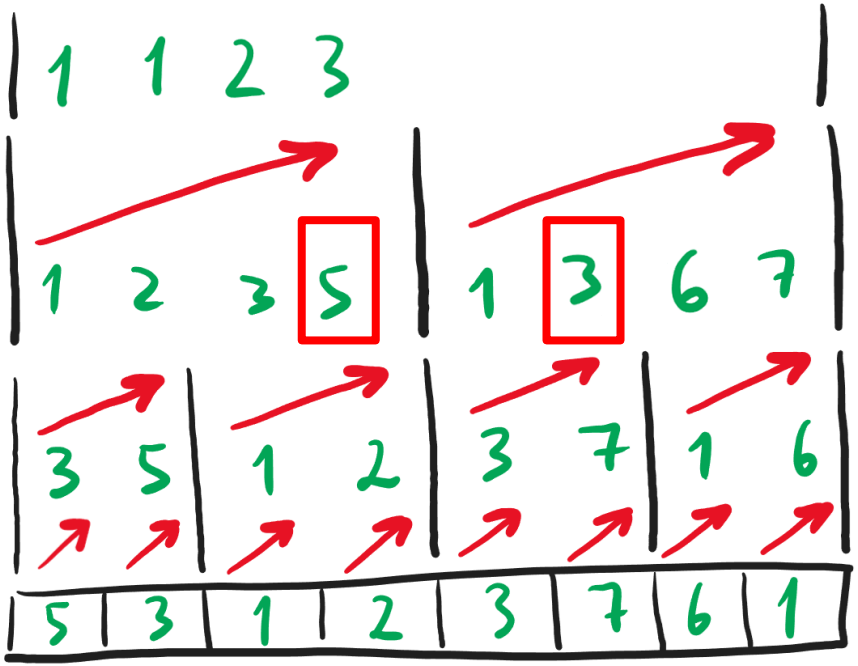
Out-of-Core сортировка (**k-way merge sort**)



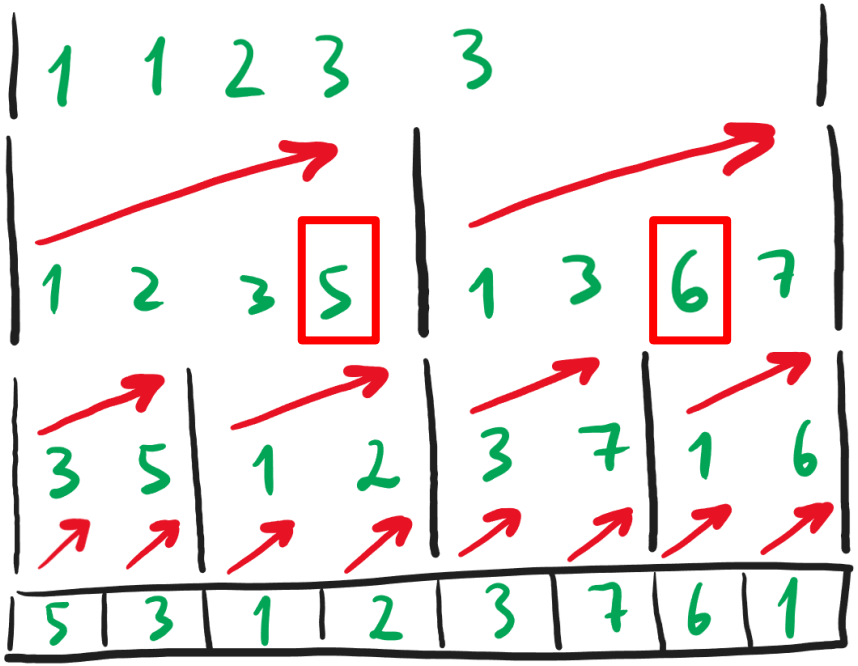
Out-of-Core сортировка (**k-way merge sort**)



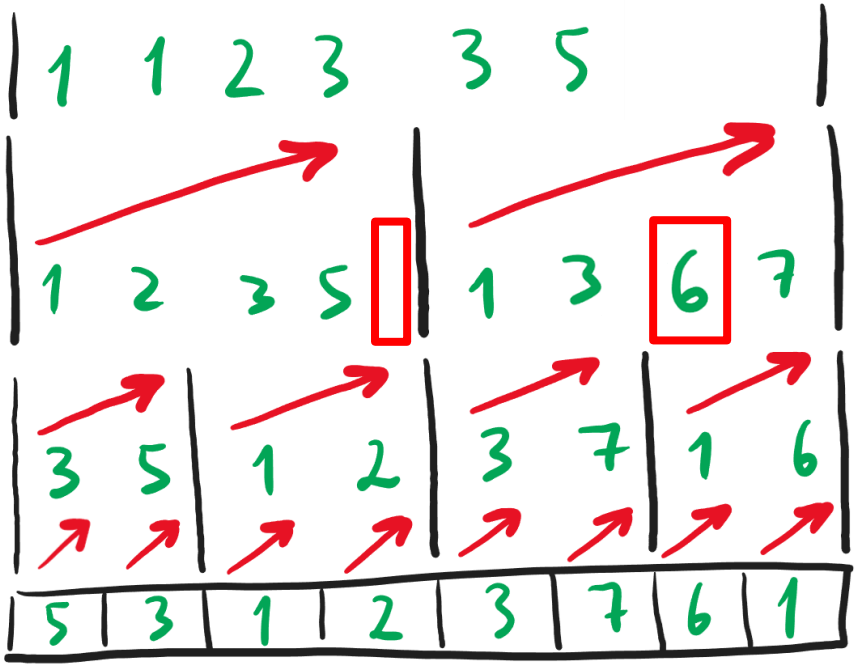
Out-of-Core сортировка (**k-way merge sort**)



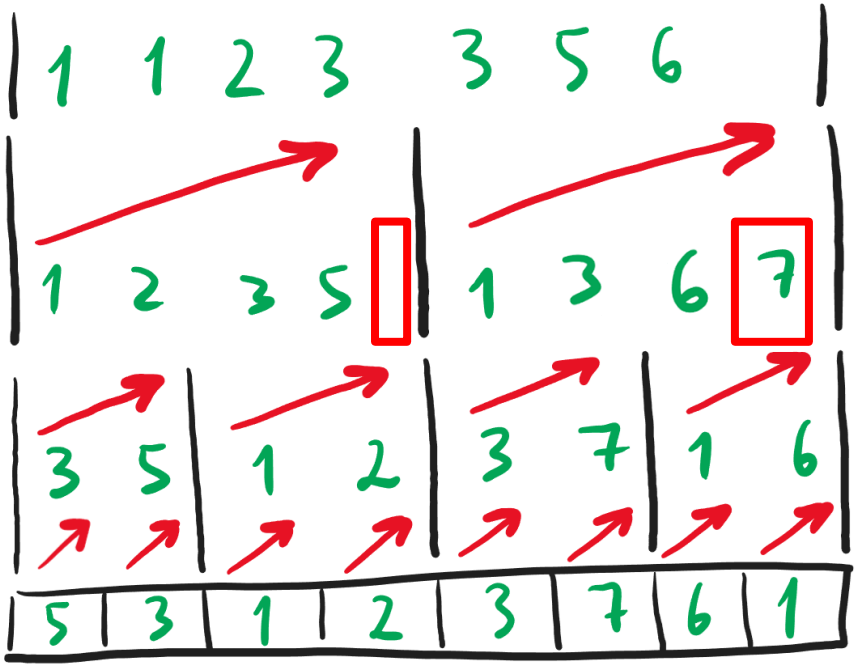
Out-of-Core сортировка (**k-way merge sort**)



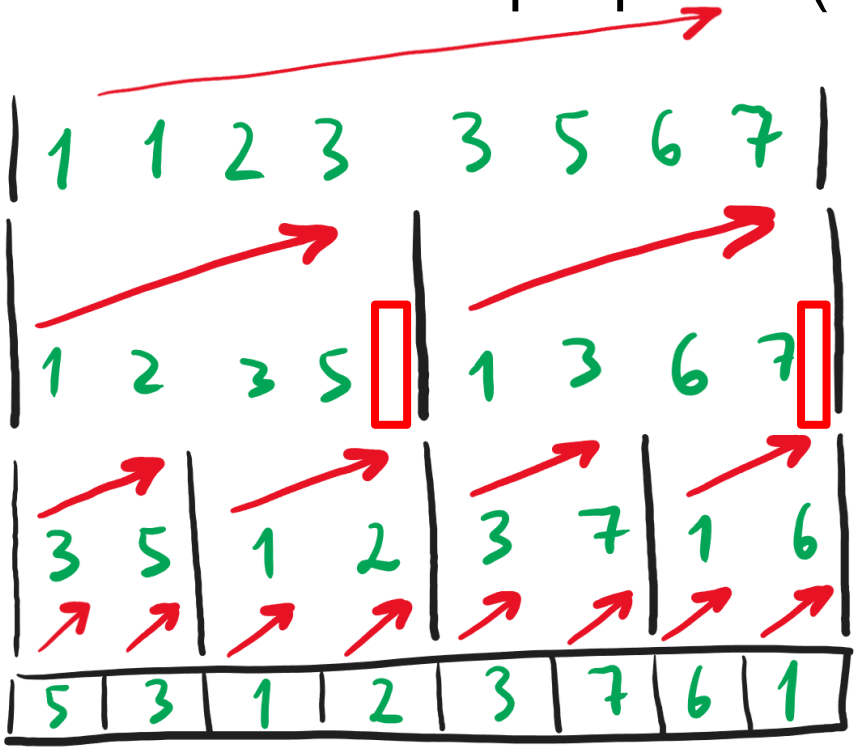
Out-of-Core сортировка (**k-way merge sort**)



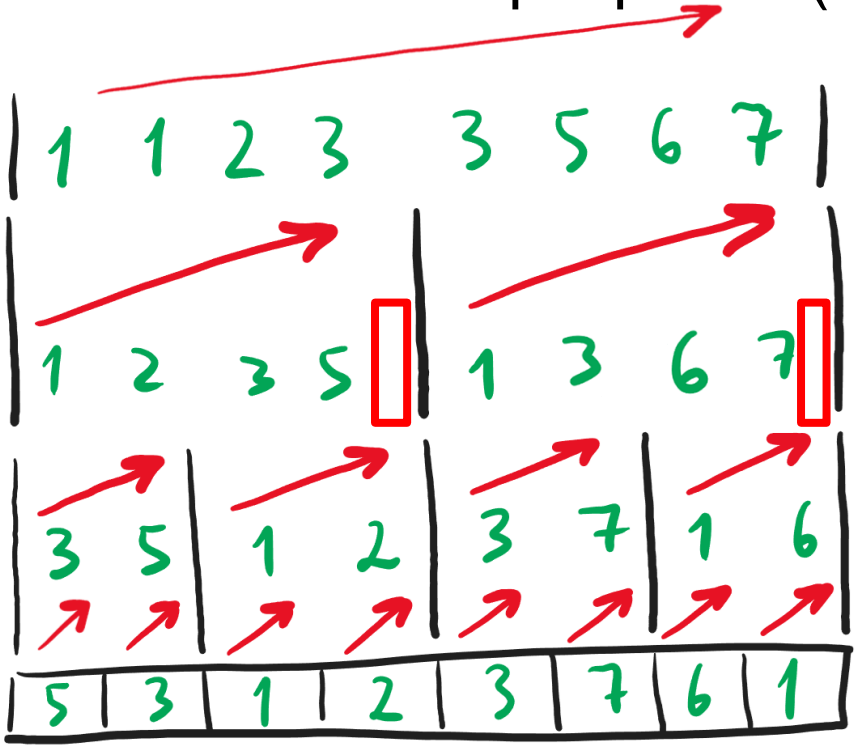
Out-of-Core сортировка (**k-way merge sort**)



Out-of-Core сортировка (**k-way merge sort**)

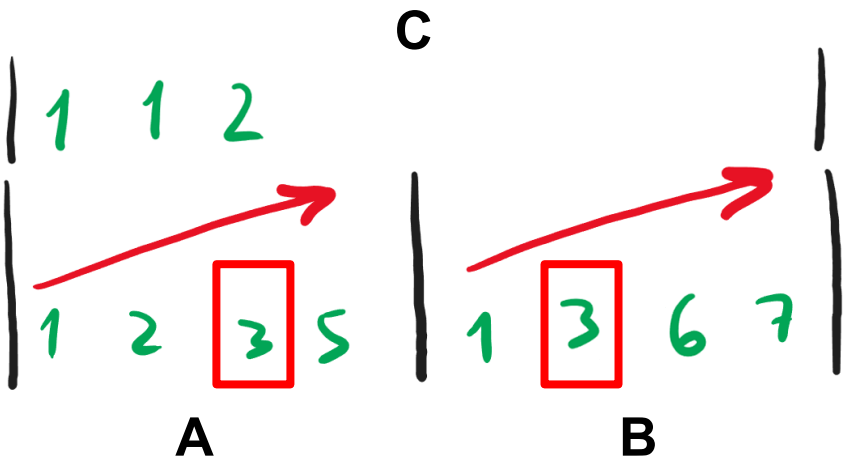


Out-of-Core сортировка (**k-way merge sort**)



Как отсортировать 10 терабайт чисел?

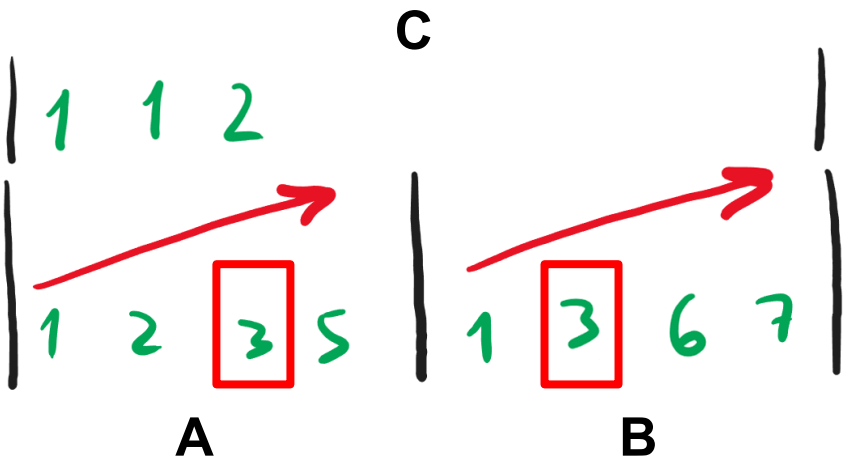
Out-of-Core сортировка (**k-way merge sort**)



Как отсортировать 10 терабайт чисел?

Достаточно научиться выполнять
C = merge(A, B) в **Out-of-Core** манере!

Out-of-Core сортировка (**k-way merge sort**)

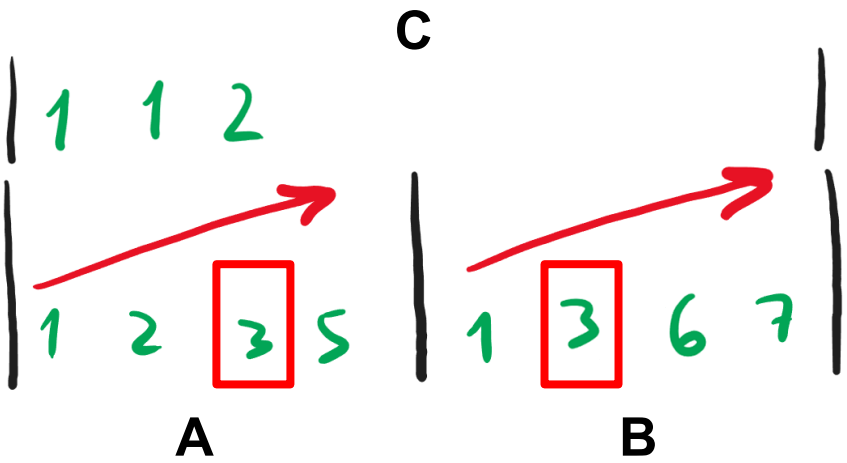


Как отсортировать 10 терабайт чисел?

Достаточно научиться выполнять
C = merge(A, B) в **Out-of-Core** манере!

Как выполнить **merge** двух 5 терабайтных файлов **отсортированных чисел**?

Out-of-Core сортировка (**k-way merge sort**)



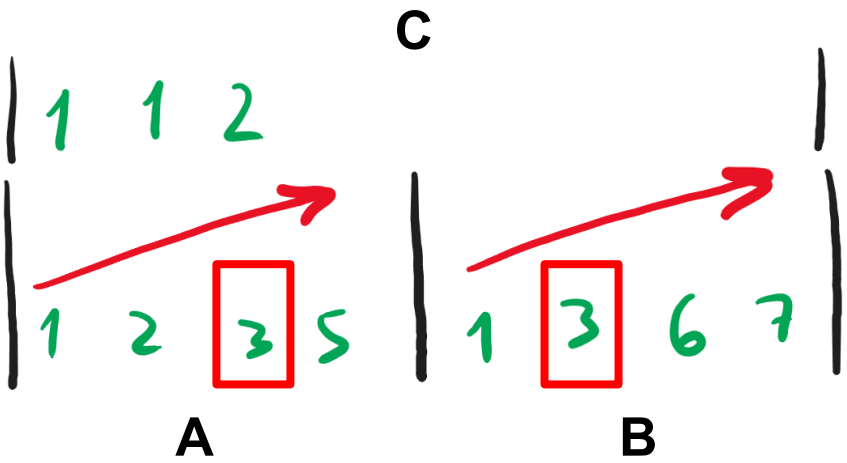
Как отсортировать 10 терабайт чисел?

Достаточно научиться выполнять
C = merge(A, B) в **Out-of-Core** манере!

Как выполнить **merge** двух 5 терабайтных файлов **отсортированных чисел**?

1) Создали пустой файл для **C**

Out-of-Core сортировка (**k-way merge sort**)



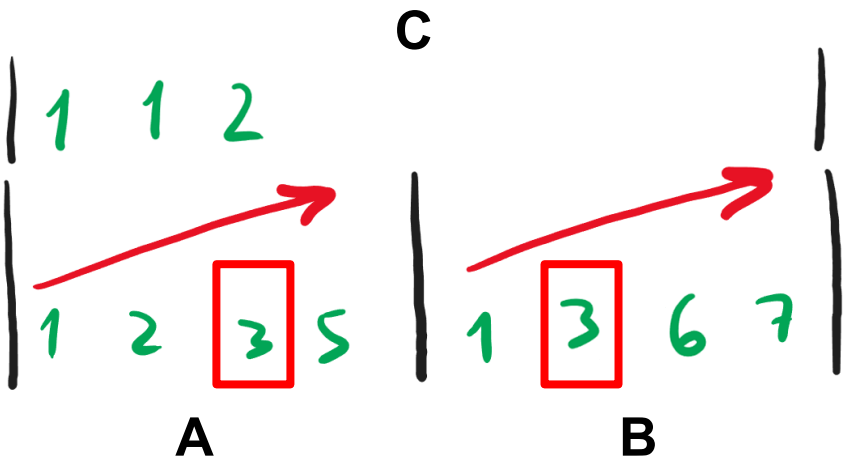
Как отсортировать 10 терабайт чисел?

Достаточно научиться выполнять
C = merge(A, B) в **Out-of-Core** манере!

Как выполнить **merge** двух 5 терабайтных файлов **отсортированных чисел**?

1) Создали пустой файл для **C**
2) Два указателя - позиция чтения **A, B**

Out-of-Core сортировка (**k-way merge sort**)



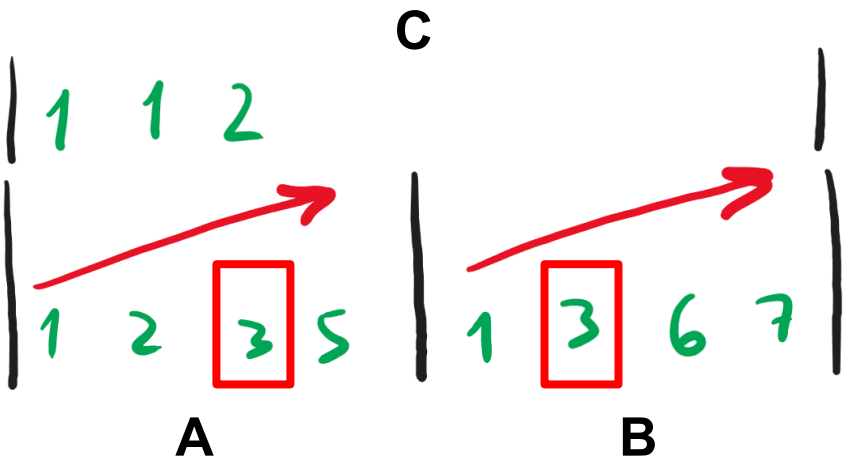
Как отсортировать 10 терабайт чисел?

Достаточно научиться выполнять
C = merge(A, B) в **Out-of-Core** манере!

Как выполнить **merge** двух 5 терабайтных файлов **отсортированных чисел**?

1) Создали пустой файл для **C**
2) Два указателя - позиция чтения **A, B**
3) В памяти только 2 числа-кандидата

Out-of-Core сортировка (**k-way merge sort**)



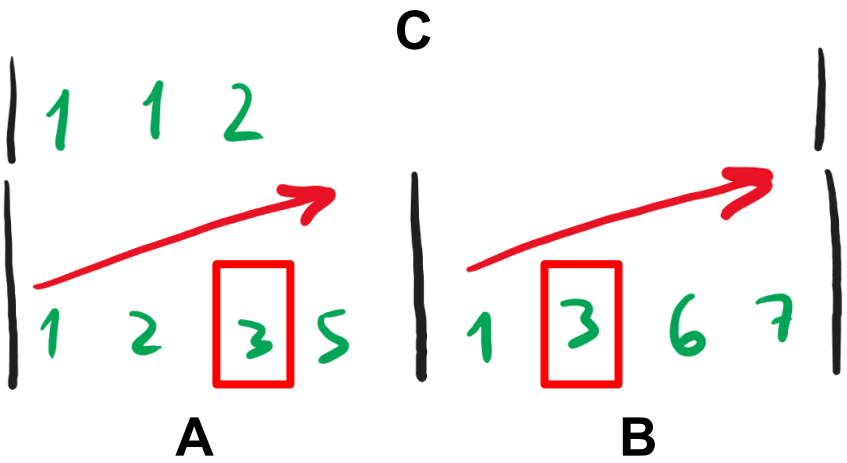
Как отсортировать 10 терабайт чисел?

Достаточно научиться выполнять
C = merge(A, B) в **Out-of-Core** манере!

Как выполнить **merge** двух 5 терабайтных файлов **отсортированных чисел**?

1) Создали пустой файл для **C**
2) Два указателя - позиция чтения **A, B**
3) В памяти только 2 числа-кандидата
4) Минимальный из них - **пишем** в **C**

Out-of-Core сортировка (**k-way merge sort**)



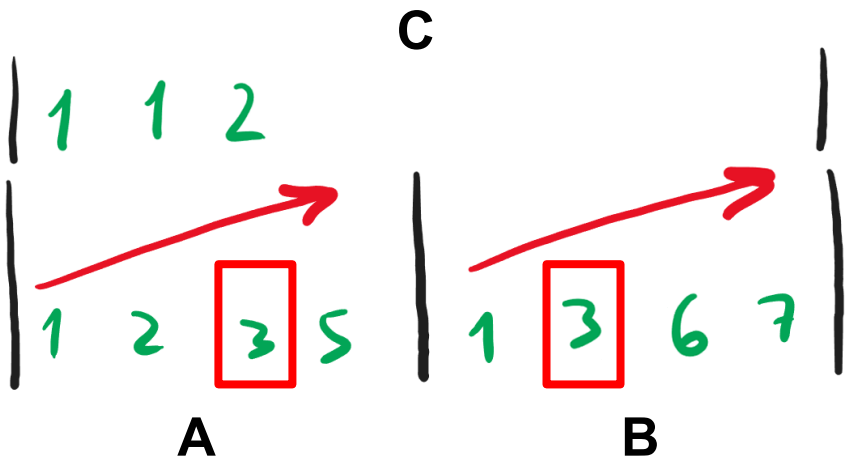
Как отсортировать 10 терабайт чисел?

Достаточно научиться выполнять
C = merge(A, B) в **Out-of-Core** манере!

Как выполнить **merge** двух 5 терабайтных файлов **отсортированных чисел**?

1) Создали пустой файл для **C**
2) Два указателя - позиция чтения **A, B**
3) В памяти только 2 числа-кандидата
4) Минимальный из них - **пишем** в **C**
5) **Считываем** новое число-кандидат

Out-of-Core сортировка (**k-way merge sort**)



Какое суммарное **IO merge**(5 TB + 5 TB)?
Какое суммарное **IO merge-sort**(N TB)?
Как ускорить?

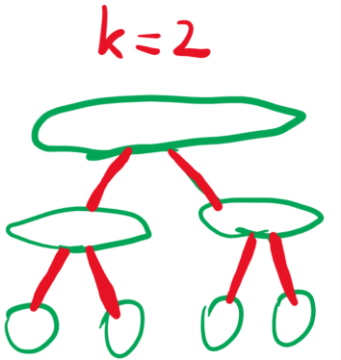
Как отсортировать 10 терабайт чисел?

Достаточно научиться выполнять
C = merge(A, B) в **Out-of-Core** манере!

Как выполнить **merge** двух 5 терабайтных файлов **отсортированных чисел**?

1) Создали пустой файл для **C**
2) Два указателя - позиция чтения **A, B**
3) В памяти только 2 числа-кандидата
4) Минимальный из них - **пишем** в **C**
5) **Считываем** новое число-кандидат

Out-of-Core сортировка (**k-way merge sort**)



Как отсортировать 10 терабайт чисел?

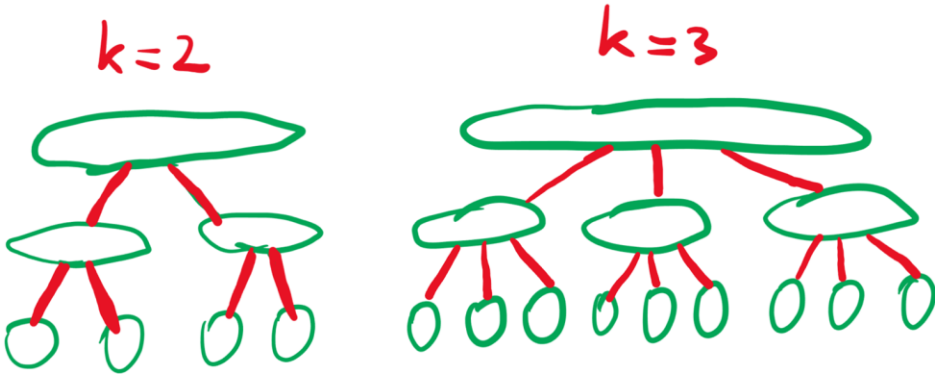
Достаточно научиться выполнять
C = merge(A, B) в **Out-of-Core** манере!

Как выполнить **merge** двух 5 терабайтных файлов **отсортированных чисел**?

Какое суммарное **IO merge**(5 TB + 5 TB)?
Какое суммарное **IO merge-sort**(N TB)?
Как ускорить?

1) Создали пустой файл для **C**
2) Два указателя - позиция чтения **A, B**
3) В памяти только 2 числа-кандидата
4) Минимальный из них - **пишем** в **C**
5) **Считываем** новое число-кандидат

Out-of-Core сортировка (**k-way merge sort**)



Как отсортировать 10 терабайт чисел?

Достаточно научиться выполнять
C = merge(A, B) в **Out-of-Core** манере!

Как выполнить **merge** двух 5 терабайтных файлов **отсортированных чисел**?

Какое суммарное **IO merge**(5 TB + 5 TB)?
Какое суммарное **IO merge-sort**(N TB)?
Как ускорить?

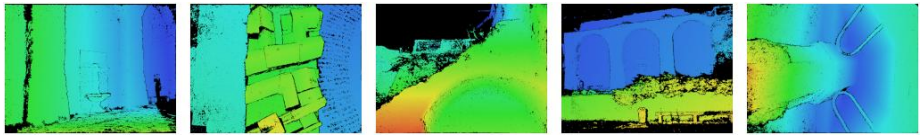
1) Создали пустой файл для **C**
2) **K указателей** - позиция чтения
3) В памяти только K чисел-кандидатов
4) Минимальный из них - **пишем** в **C**
5) **Считываем** новое число-кандидат

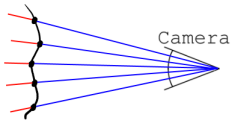
Алгоритм во внешней памяти (out-of-core свойство)

Дискретизация пространства октодеревом

Out-of-Core реконструкция: **строим октодерево**

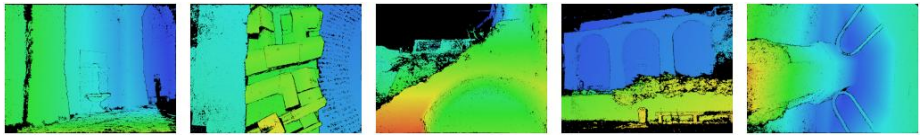
1) Есть множество карт глубины

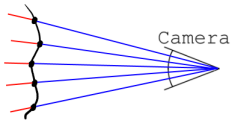




Out-of-Core реконструкция: **строим октодерево**

1) Есть множество карт глубины

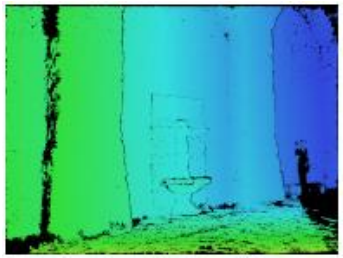




2) Множество точек всех карт глубины - порождает **адаптивное** октодерево (дискретизация пространства)

Как построить октодерево не упав по памяти?

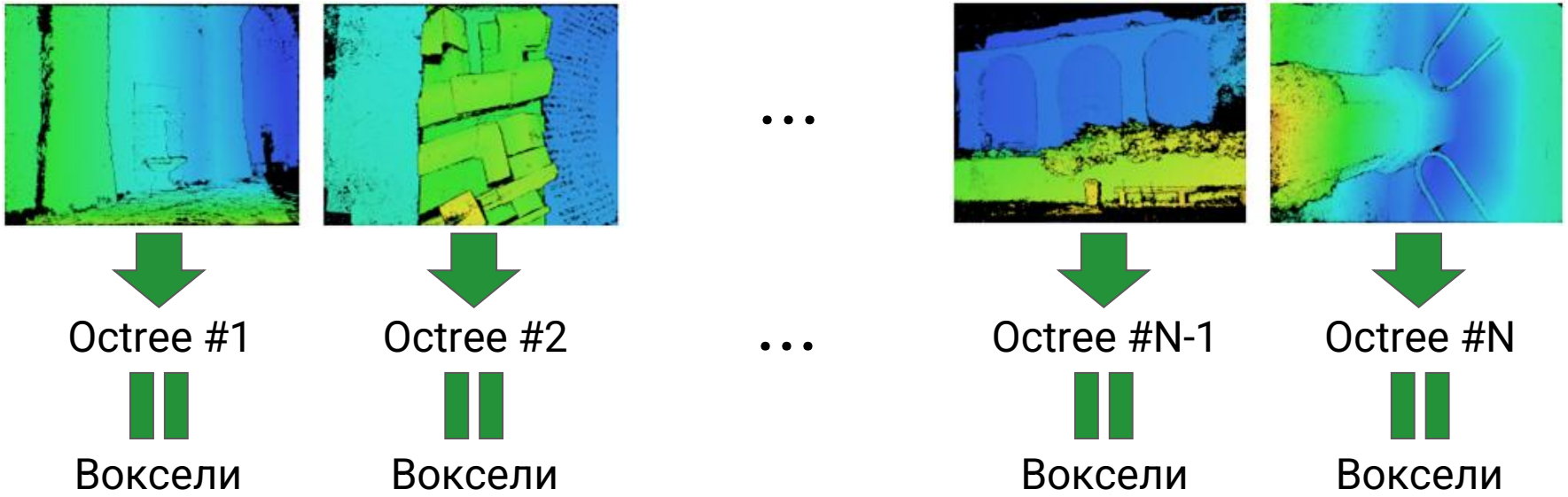
Out-of-Core реконструкция: **строим октодерево**



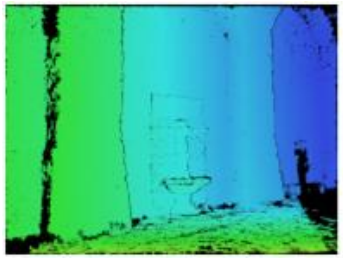
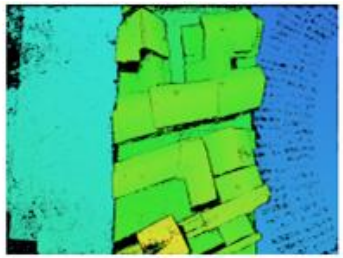
Octree #1

Воксели

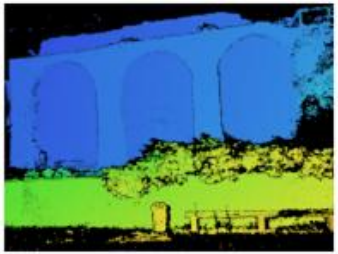
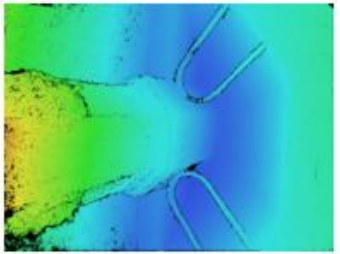
Out-of-Core реконструкция: **строим октодерево**



Out-of-Core реконструкция: **строим октодерево**

...

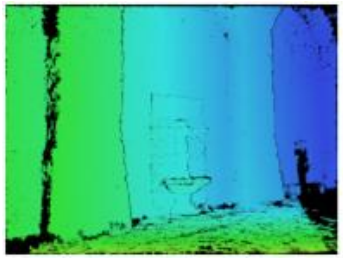



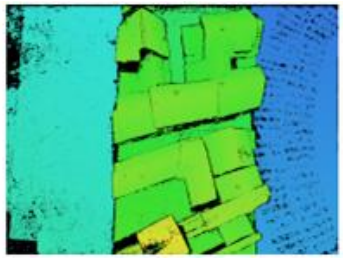
Octree #1 | Octree #2 | ... | Octree #N-1 | Octree #N

Вокселли | Вокселли | Вокселли | Вокселли

96-bit Morton Codes | 96-bit Morton Codes | 96-bit Morton Codes | 96-bit Morton Codes

Out-of-Core реконструкция:





...

Octree #1

Octree #2

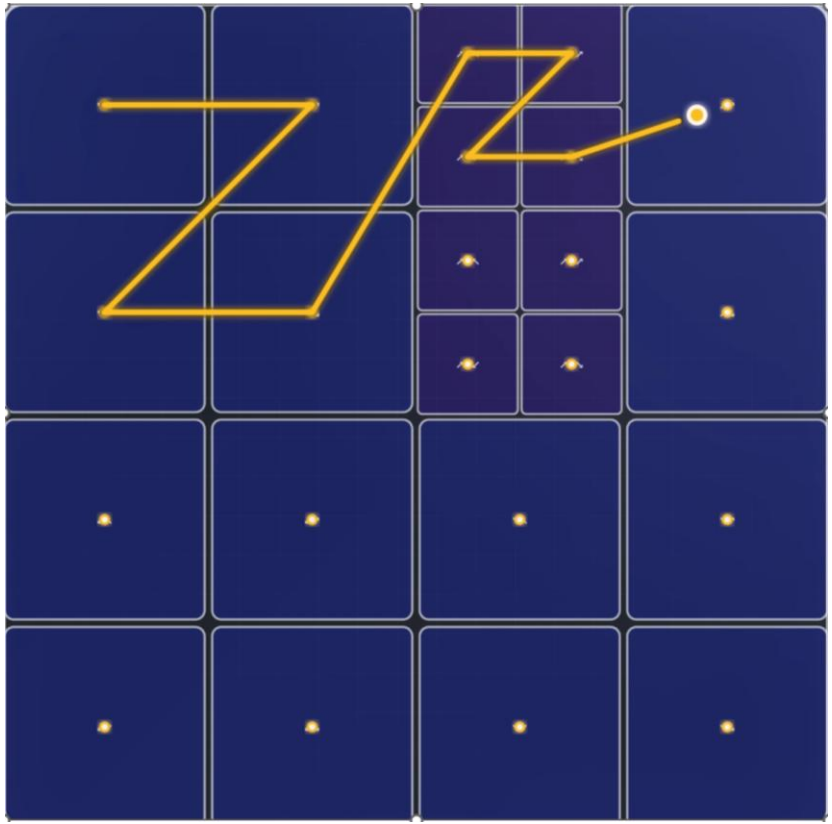
...

~~Воксели~~

~~Воксели~~

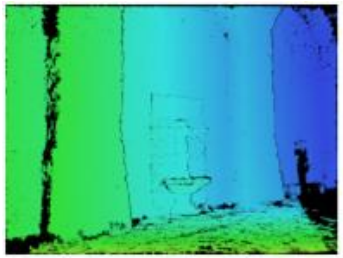
96-bit
Morton Codes

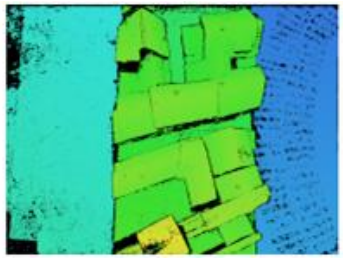
96-bit
Morton Codes



Z-curve: ввели линейный порядок на вокселях.
Есть пространственная локальность!
Morton Codes: биекция воксель ➡ число,
такая что числа соблюдают порядок Z-curve.

Out-of-Core реконструкция:





...

Octree #1

Octree #2

...

~~Воксели~~

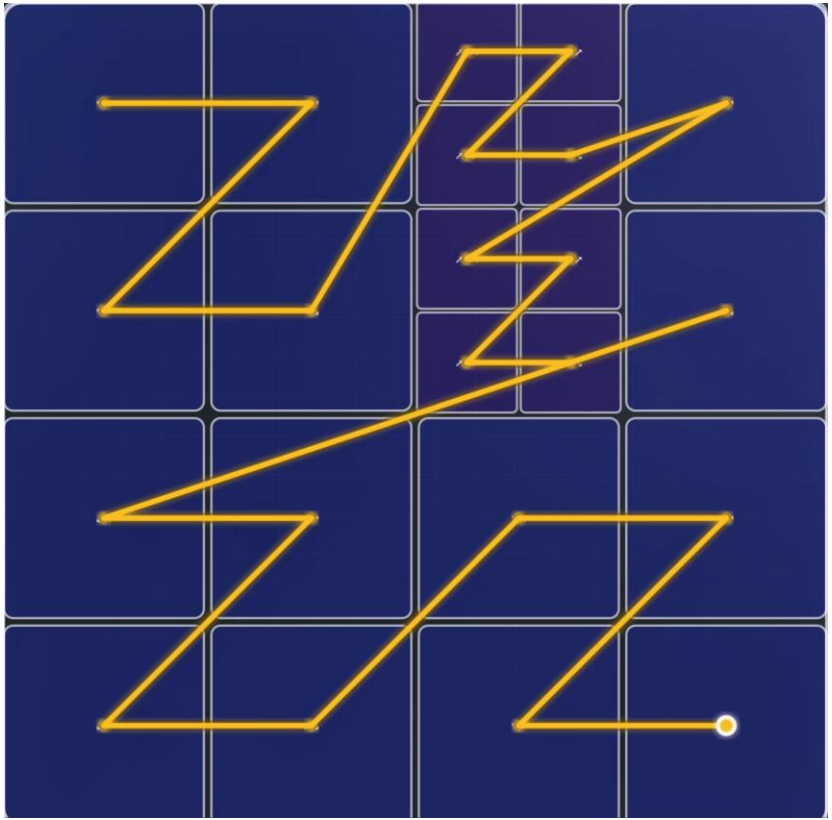
~~Воксели~~

96-bit
Morton Codes

96-bit
Morton Codes

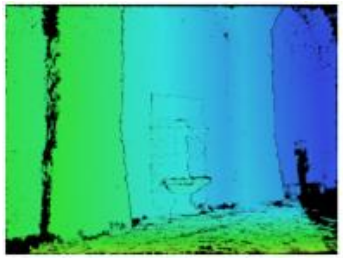
Shuffled xyz Morton Code:

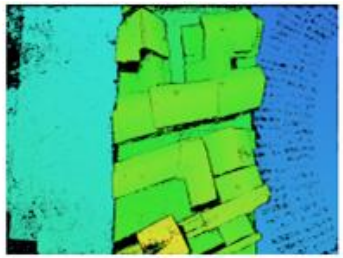
$x_1 y_1 z_1$



Z-curve: ввели линейный порядок на вокселях.
Есть пространственная локальность!
Morton Codes: биекция воксель ➡ число,
такая что числа соблюдают порядок Z-curve.

Out-of-Core реконструкция:





...

Octree #1

Octree #2

...

~~Воксели~~

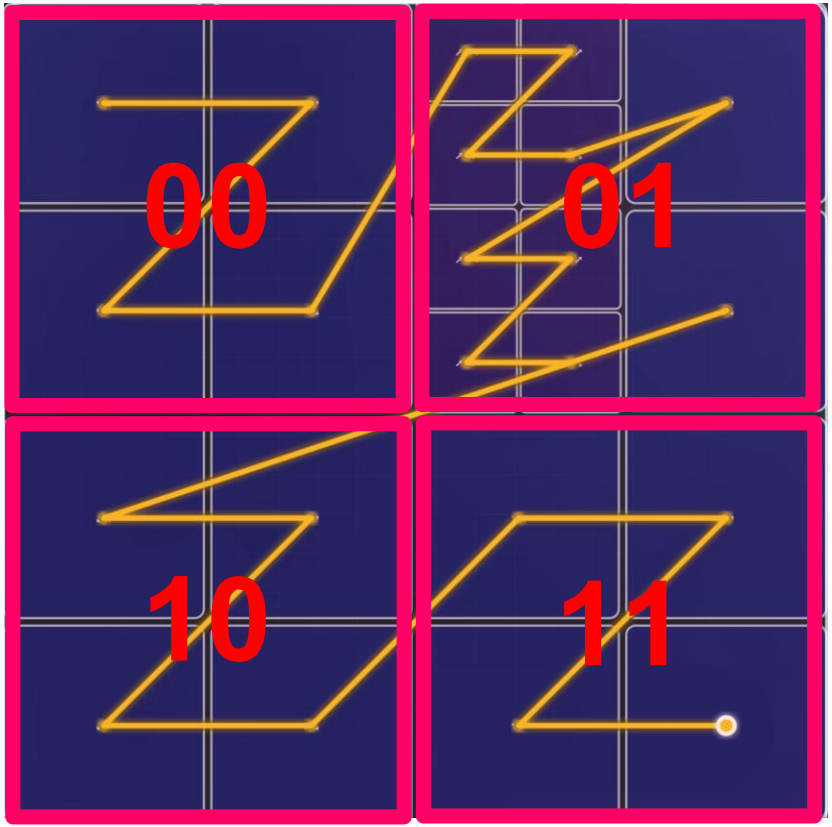
~~Воксели~~

96-bit
Morton Codes

96-bit
Morton Codes

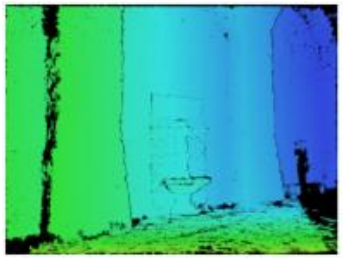
Shuffled xyz Morton Code:

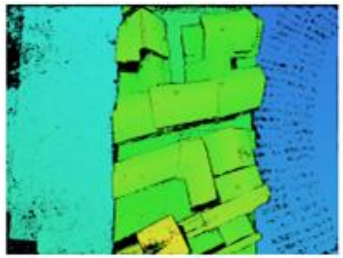
$$x_1 y_1 z_1$$



Z-curve: ввели линейный порядок на вокселях.
Есть пространственная локальность!
Morton Codes: биекция воксель ➡ число,
такая что числа соблюдают порядок Z-curve.

Out-of-Core реконструкция:





...

Octree #1

Octree #2

...

~~Воксели~~

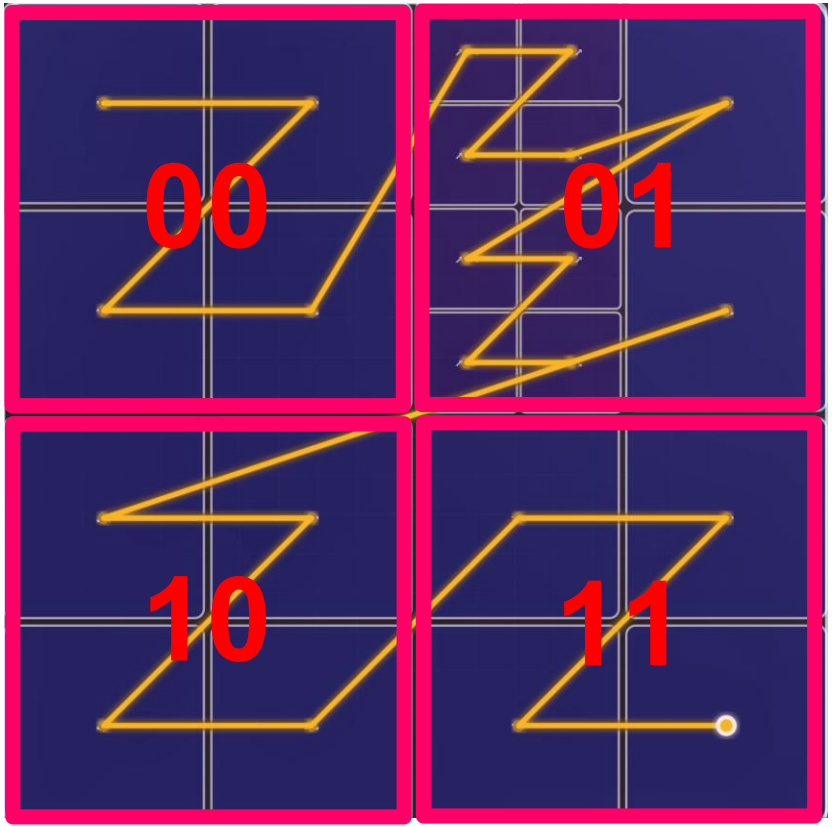
~~Воксели~~

96-bit
Morton Codes

96-bit
Morton Codes

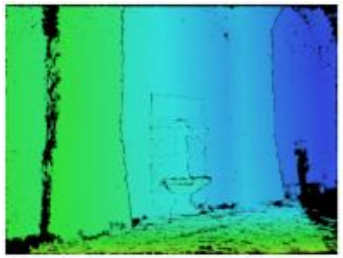
Shuffled xyz Morton Code:

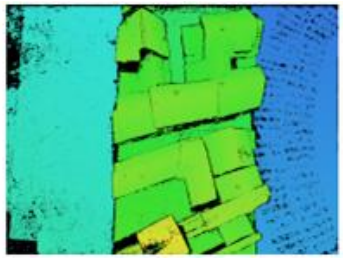
$$x_1 y_1 z_1 \; x_2 y_2 z_2 \cdots x_D y_D z_D$$



Z-curve: ввели линейный порядок на вокселях.
Есть пространственная локальность!
Morton Codes: биекция воксель ➡ число,
такая что числа соблюдают порядок Z-curve.

Out-of-Core реконструкция: **строим октодерево**





...

Octree #1

Octree #2

...

Octree #N-1

Octree #N

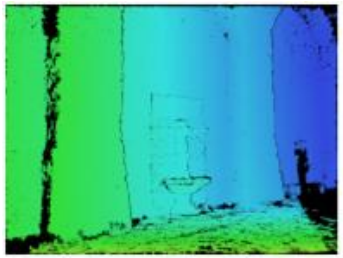
96-bit
Morton Codes

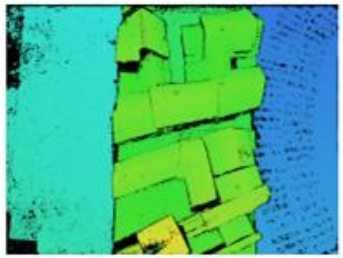
96-bit
Morton Codes

96-bit
Morton Codes

96-bit
Morton Codes

Out-of-Core реконструкция: **строим октодерево**





...

Octree #1

Octree #2

...

Octree #N-1

Octree #N

96-bit
Morton Codes

96-bit
Morton Codes

96-bit
Morton Codes

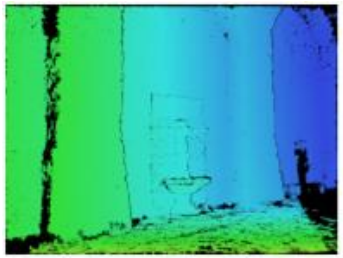
96-bit
Morton Codes

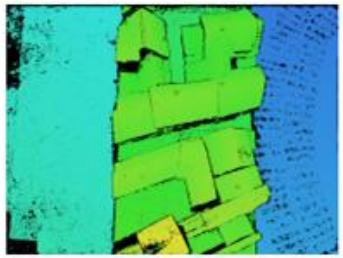
Как сделать это **Out-of-Core**?

96-bit
Morton Codes
(без дупликатов)

Глобальное Октодерево

Out-of-Core реконструкция: **строим октодерево**





...

Octree #1

Octree #2

...

Octree #N-1

Octree #N

96-bit
Morton Codes

96-bit
Morton Codes

96-bit
Morton Codes

96-bit
Morton Codes

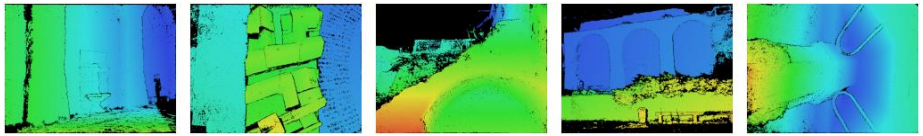
Out-of-Core k-way merge sort

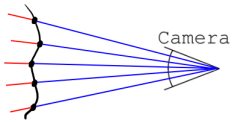
96-bit
Morton Codes
(без дупликатов)

Глобальное Октодерево

Out-of-Core реконструкция: **строим октодерево**

1) Есть множество карт глубины

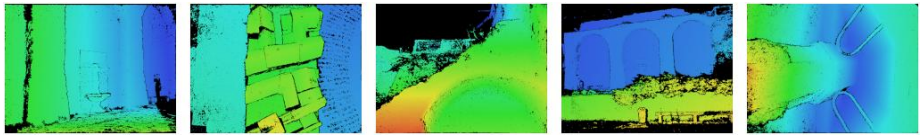


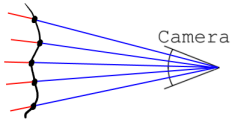


2) Множество точек всех карт глубины - порождает **адаптивное** октодерево (дискретизация пространства)

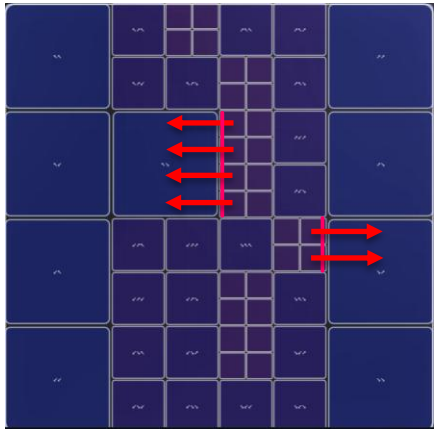
Out-of-Core реконструкция: **строим октодерево**

1) Есть множество карт глубины



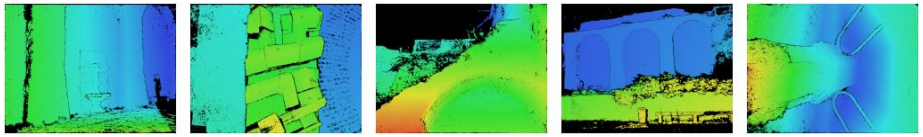


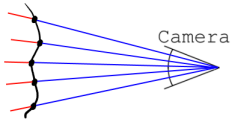
2) Множество точек всех карт глубины - порождает **адаптивное** октодерево (дискретизация пространства)
3) Балансируем октодерево 2:1 (по каждой стороне макс. два соседа)



Out-of-Core реконструкция: **строим октодерево**

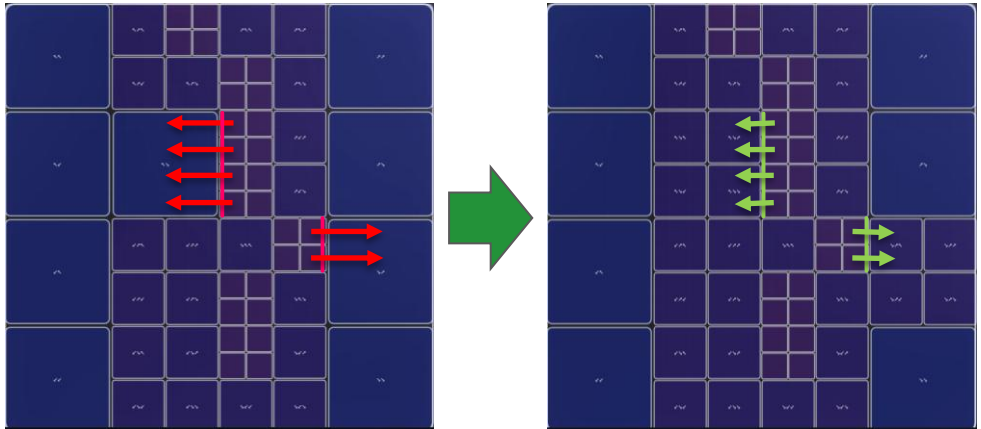
1) Есть множество карт глубины





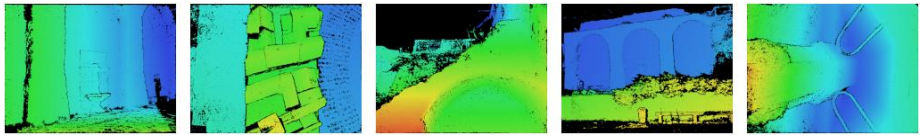
2) Множество точек всех карт глубины - порождает **адаптивное** октодерево (дискретизация пространства)

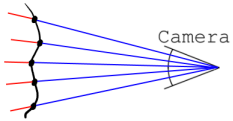
3) Балансируем октодерево 2:1 (по каждой стороне макс. два соседа)



Out-of-Core реконструкция: **строим октодерево**

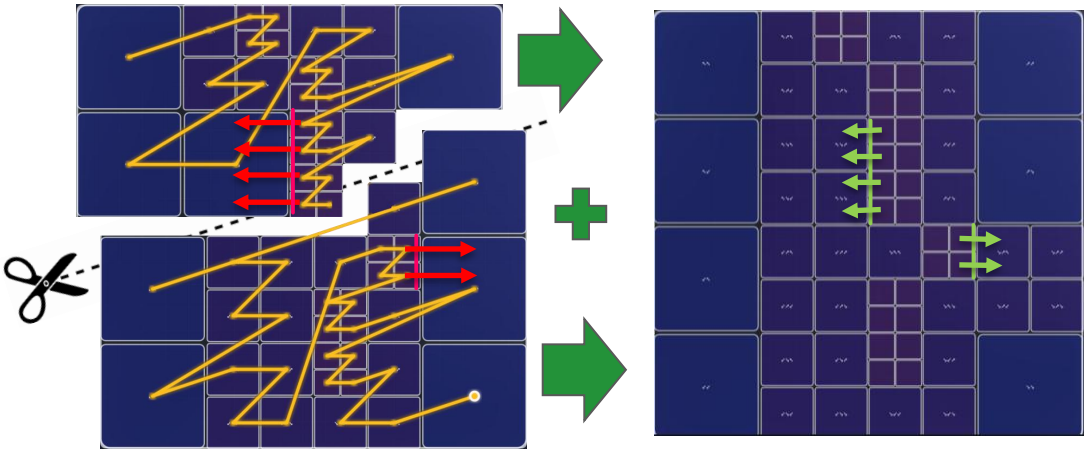
1) Есть множество карт глубины



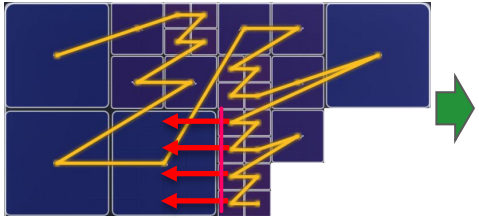


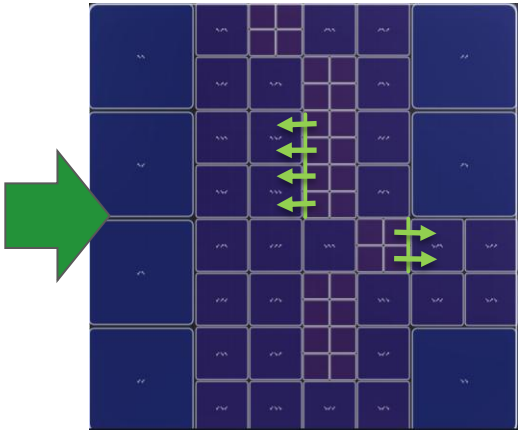
2) Множество точек всех карт глубины - порождает **адаптивное** октодерево (дискретизация пространства)

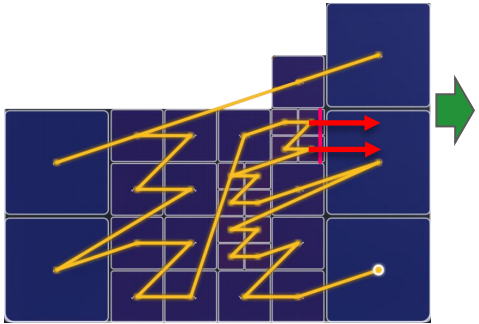
3) Балансируем октодерево 2:1 (по каждой стороне макс. два соседа)



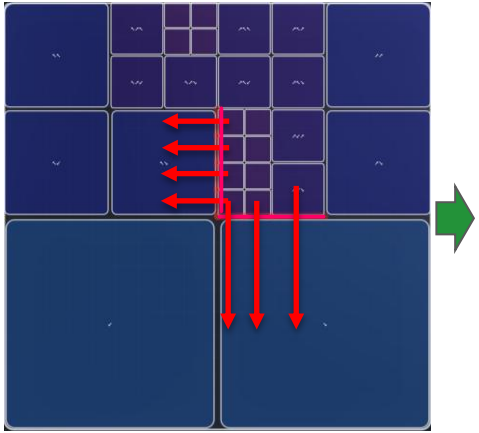
Out-of-Core реконструкция: **2:1 балансировка**

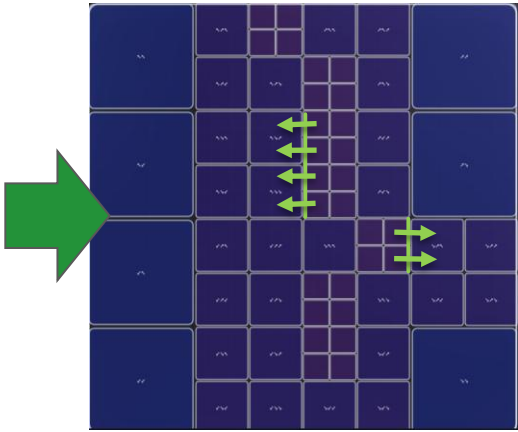


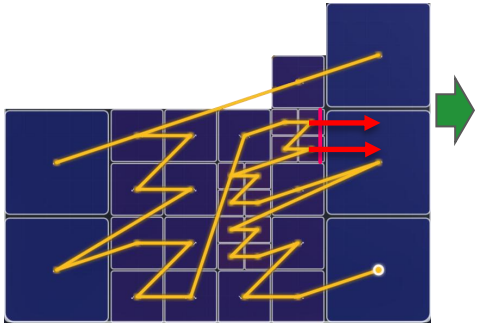




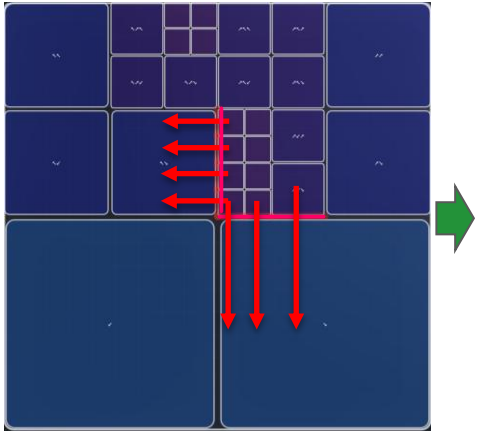
Out-of-Core реконструкция: **2:1 балансировка**

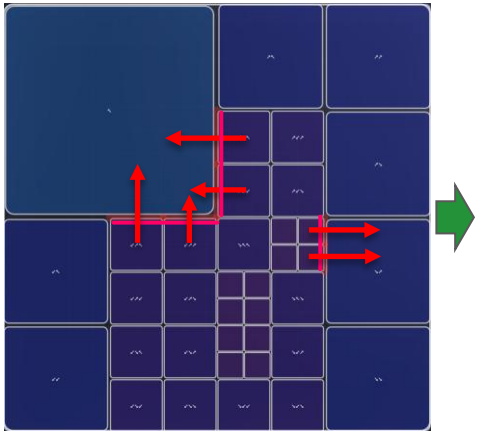


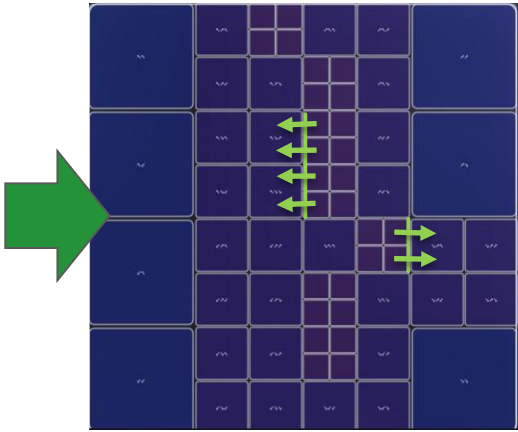




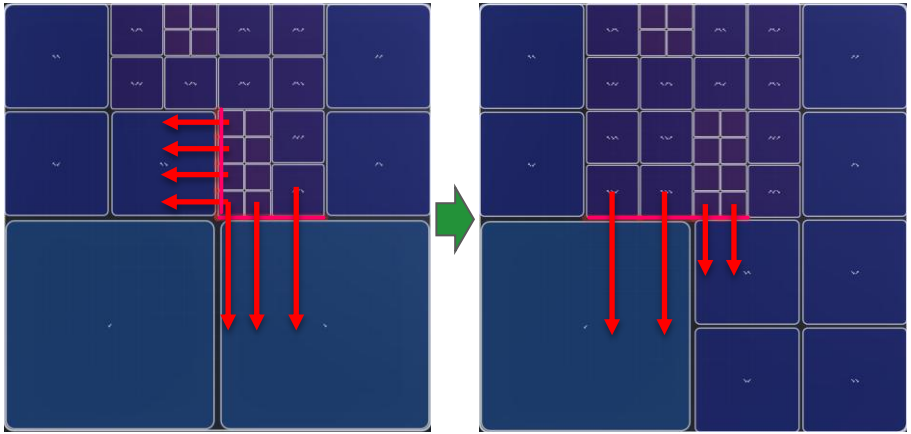
Out-of-Core реконструкция: **2:1 балансировка**

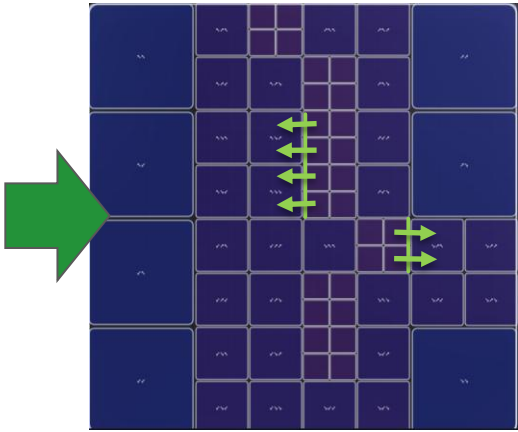


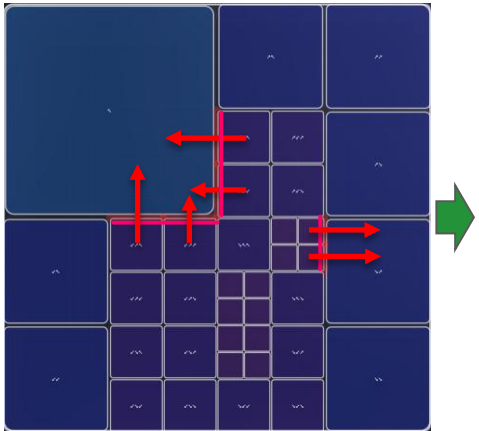




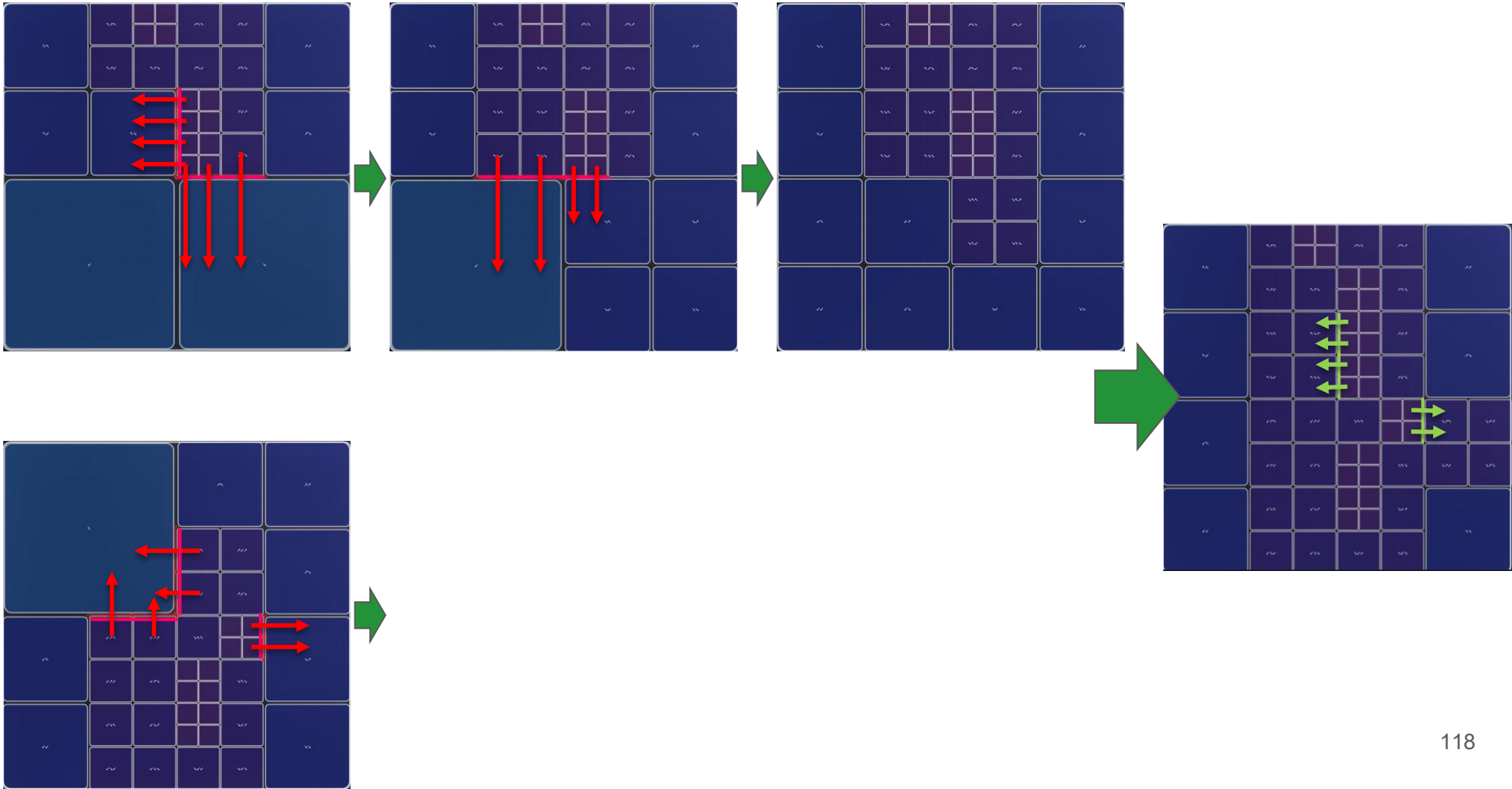
Out-of-Core реконструкция: **2:1 балансировка**



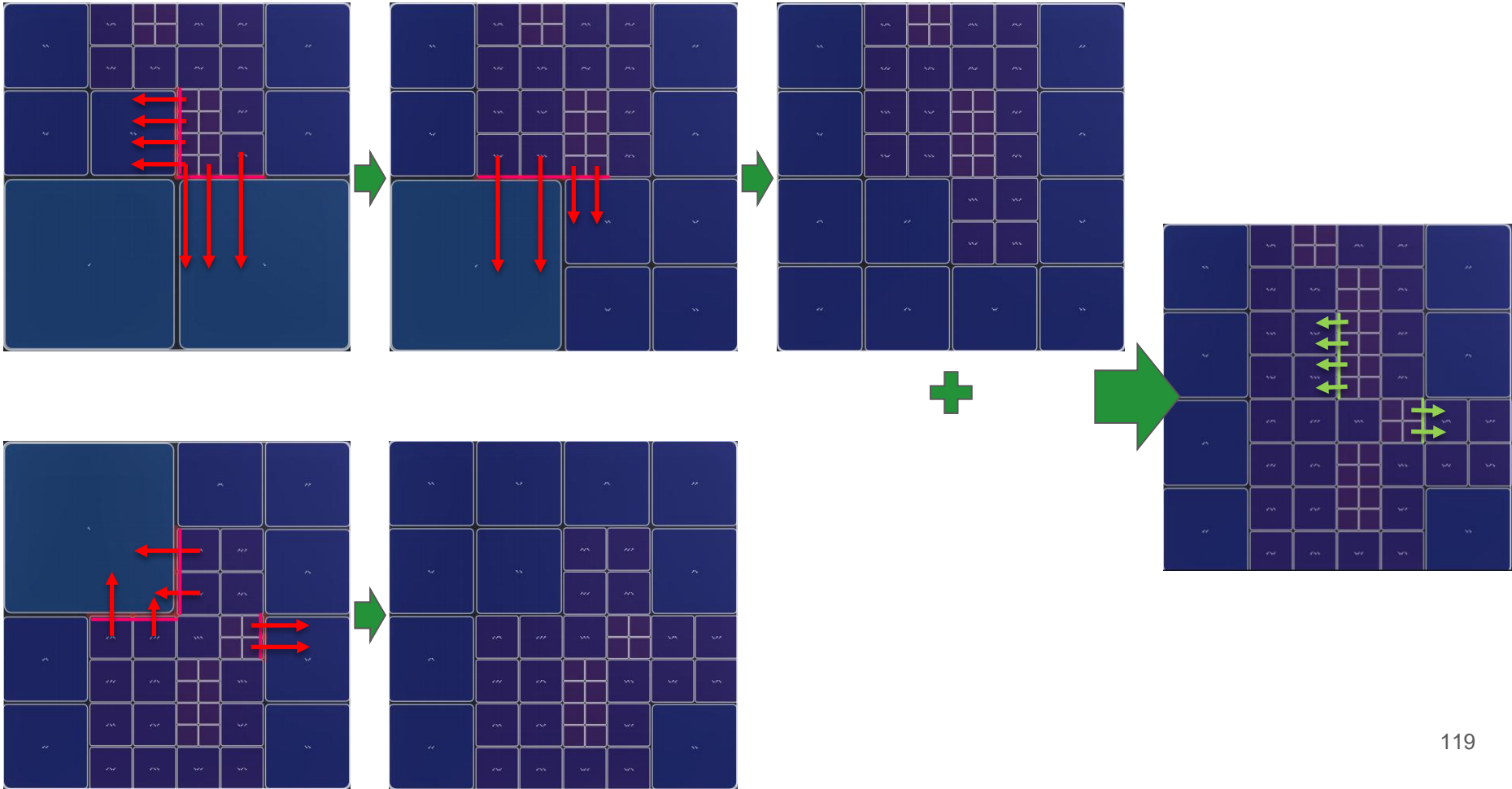




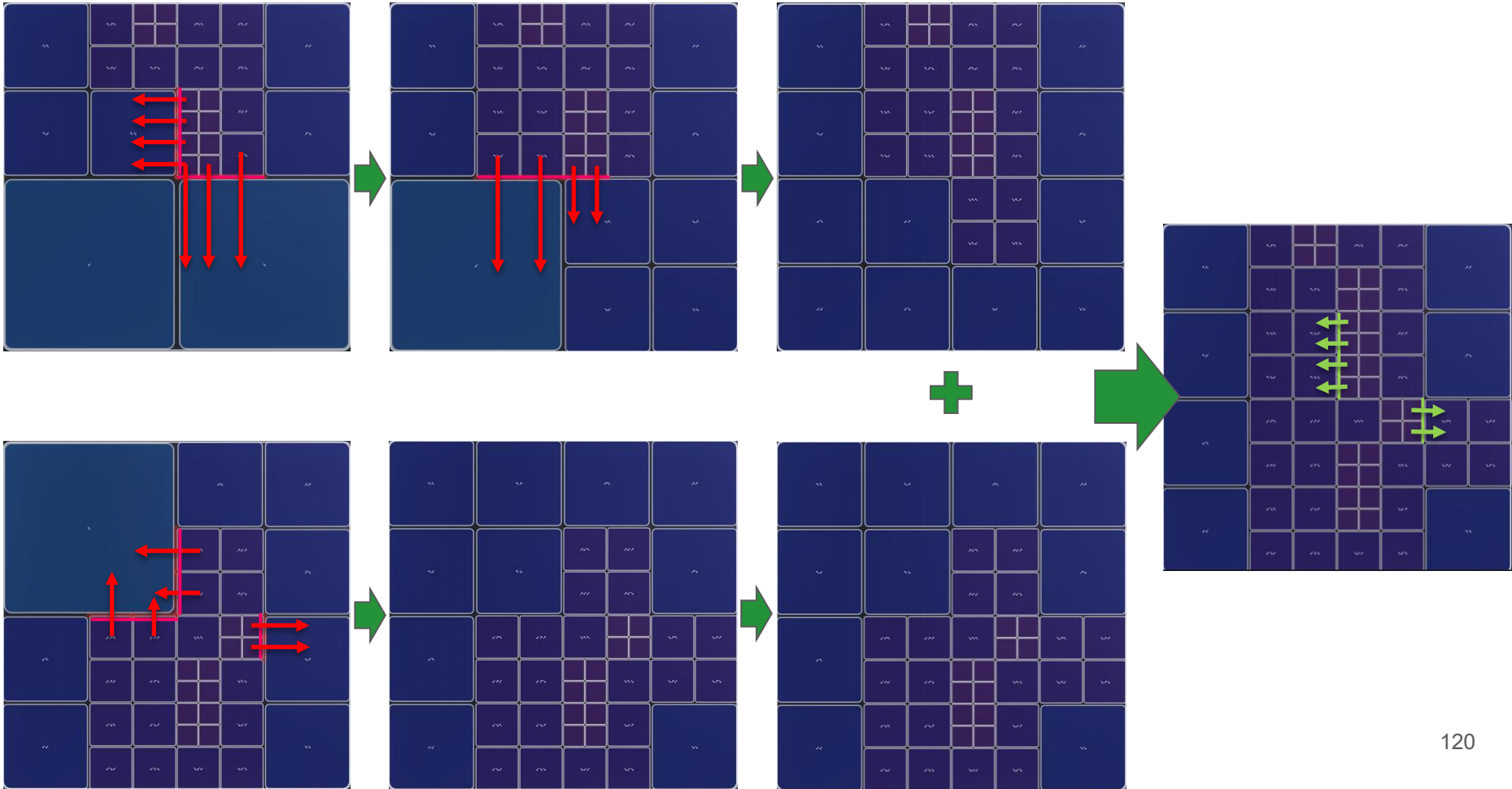
Out-of-Core реконструкция: **2:1 балансировка**



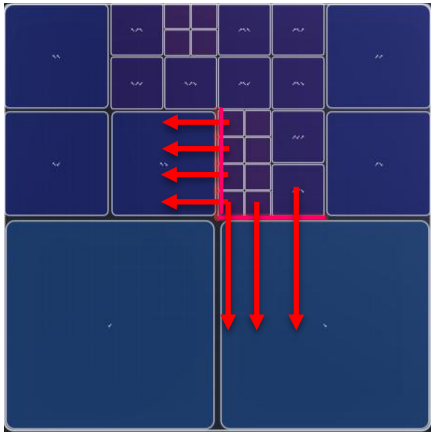
Out-of-Core реконструкция: **2:1 балансировка**

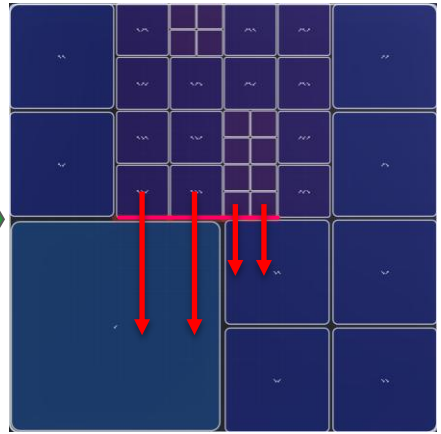


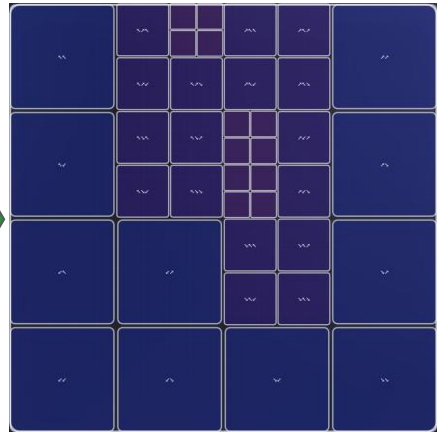
Out-of-Core реконструкция: **2:1 балансировка**



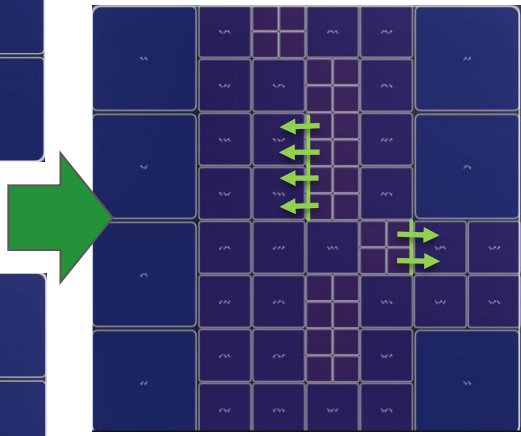
Out-of-Core реконструкция: **2:1 балансировка**

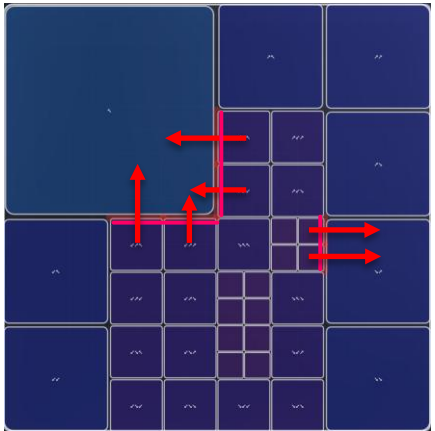


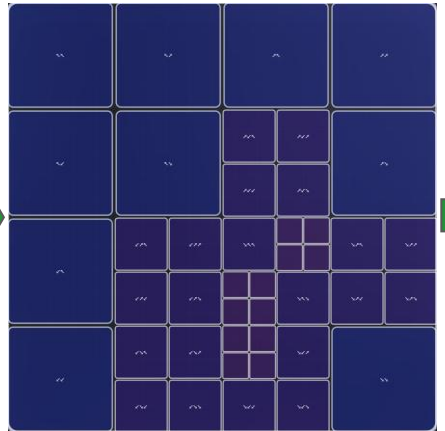


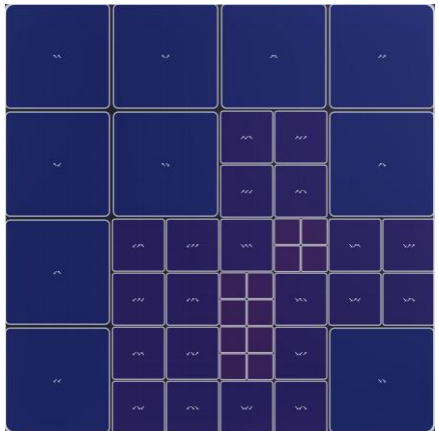


Как объединить не упав по памяти?

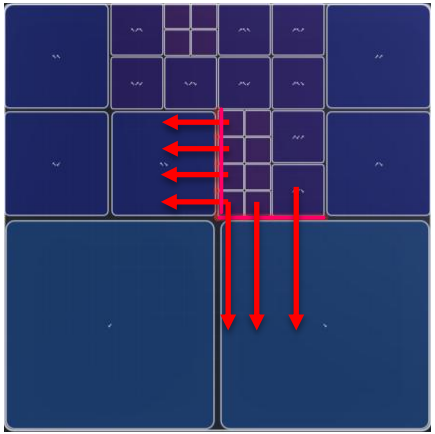


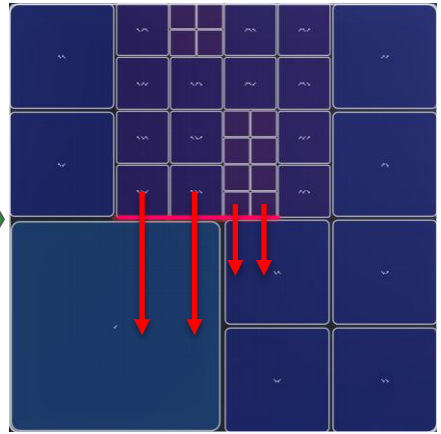


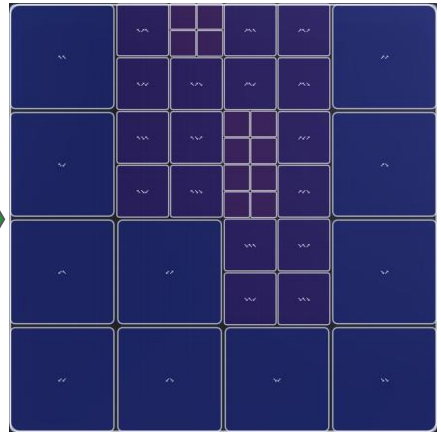




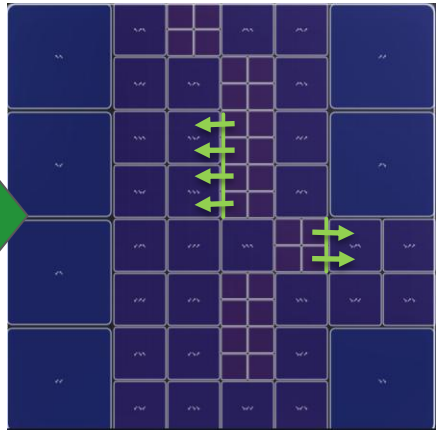
Out-of-Core реконструкция: **2:1 балансировка**

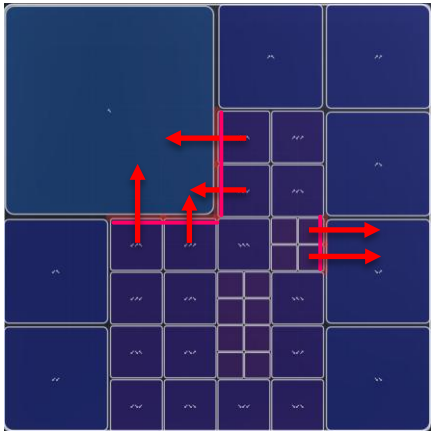


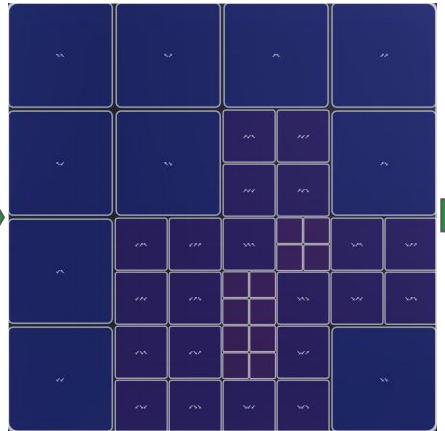


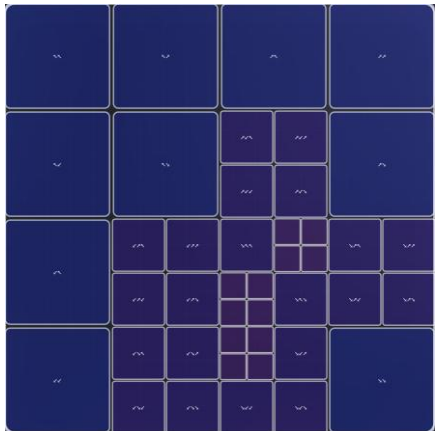


Out-of-Core k-way merge sort ✚







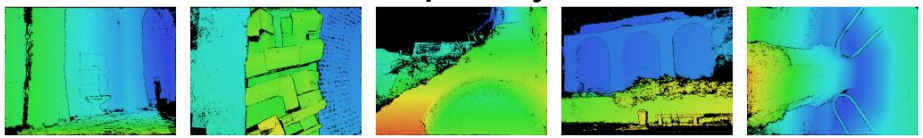


Алгоритм во внешней памяти (out-of-core свойство)

Обработка октодерева

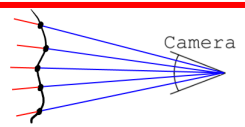
Итого

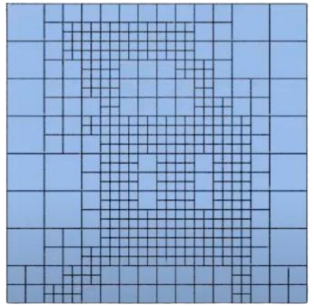
1) Есть множество карт глубины



OpenCL

Преобразуем в гистограммы!





2) Множество точек всех карт глубины - порождает **адаптивное** октодерево (дискретизация пространства)

3) Балансируем октодерево 2:1 (по каждой стороне макс. два соседа)

4) Оптимизировали индикаторное поле (**много итераций + Coarse-to-Fine**)

OpenCL

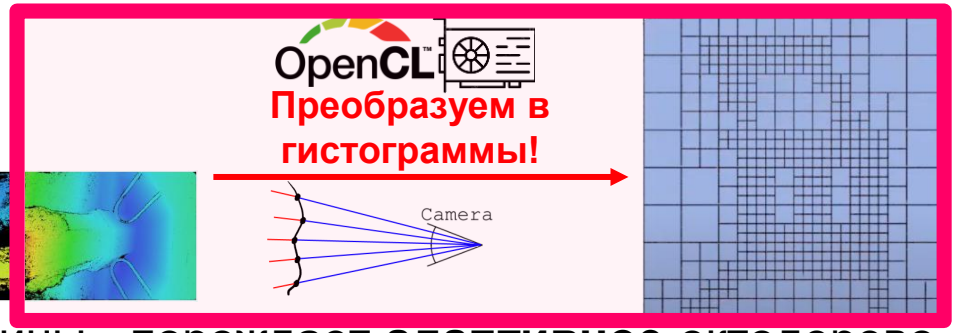
TV-L^1:

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

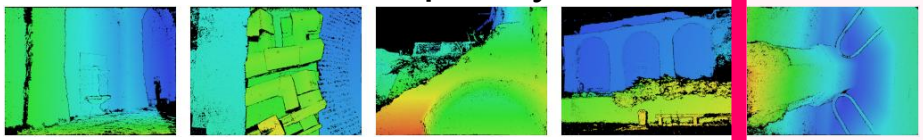
5) Извлекли поверхность маршировкой кубов

Итого

Как обработать все октодерево не упав по памяти?



1) Есть множество карт глубины

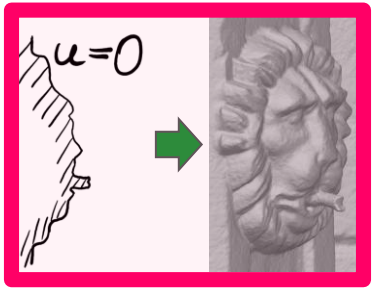


2) Множество точек всех карт глубины - порождает **адаптивное** октодерево (дискретизация пространства)
3) Балансируем октодерево 2:1 (по каждой стороне макс. два соседа)
4) Оптимизировали индикаторное поле (**много итераций + Coarse-to-Fine**)

OpenCL

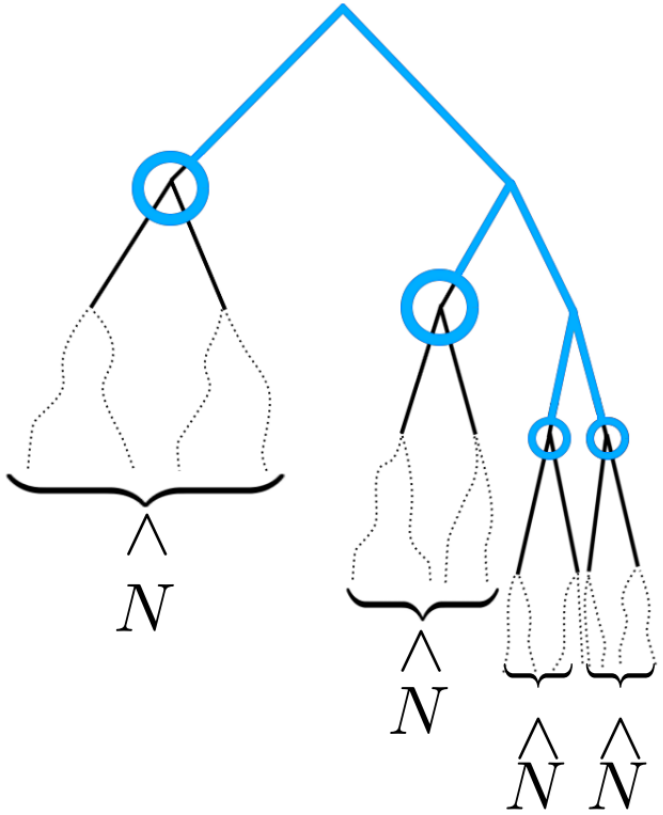
TV-L^1:

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

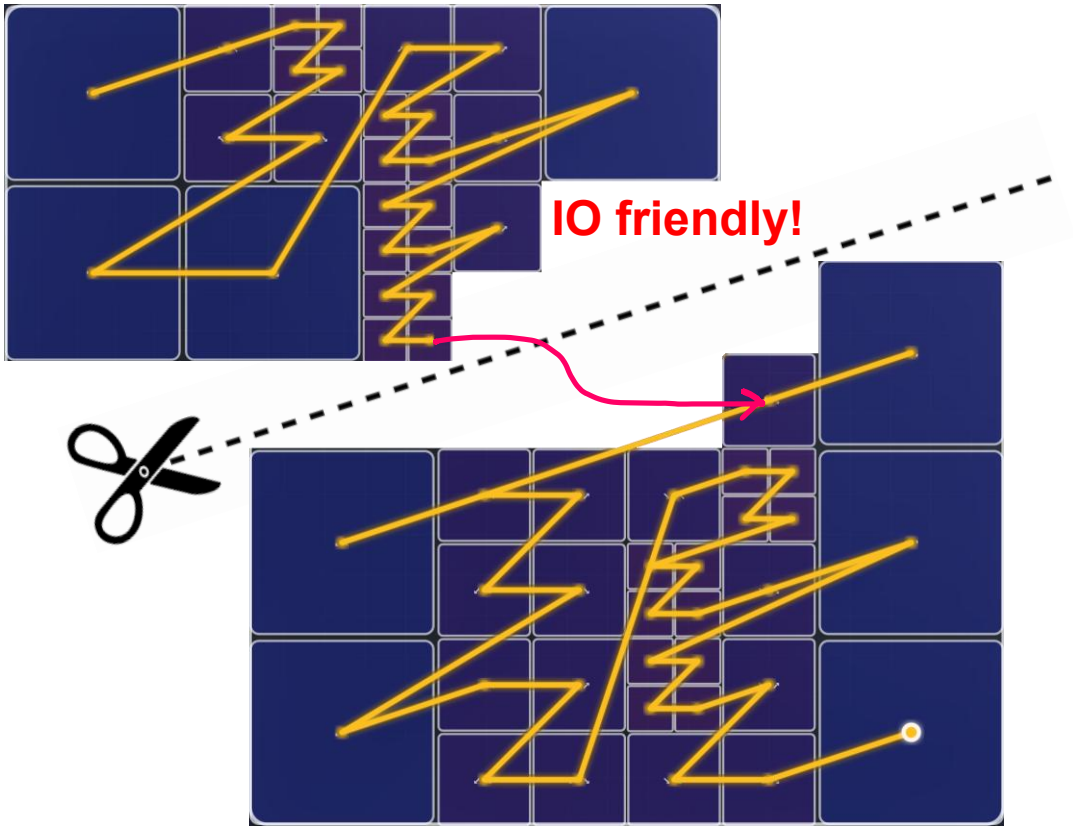


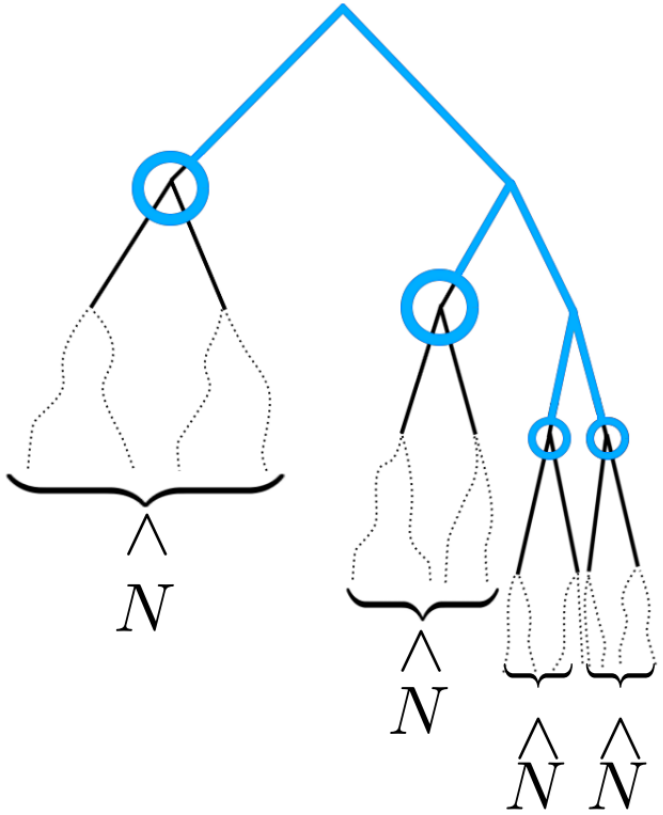
5) Извлекли поверхность маршировкой кубов

Обработка октодерева по частям

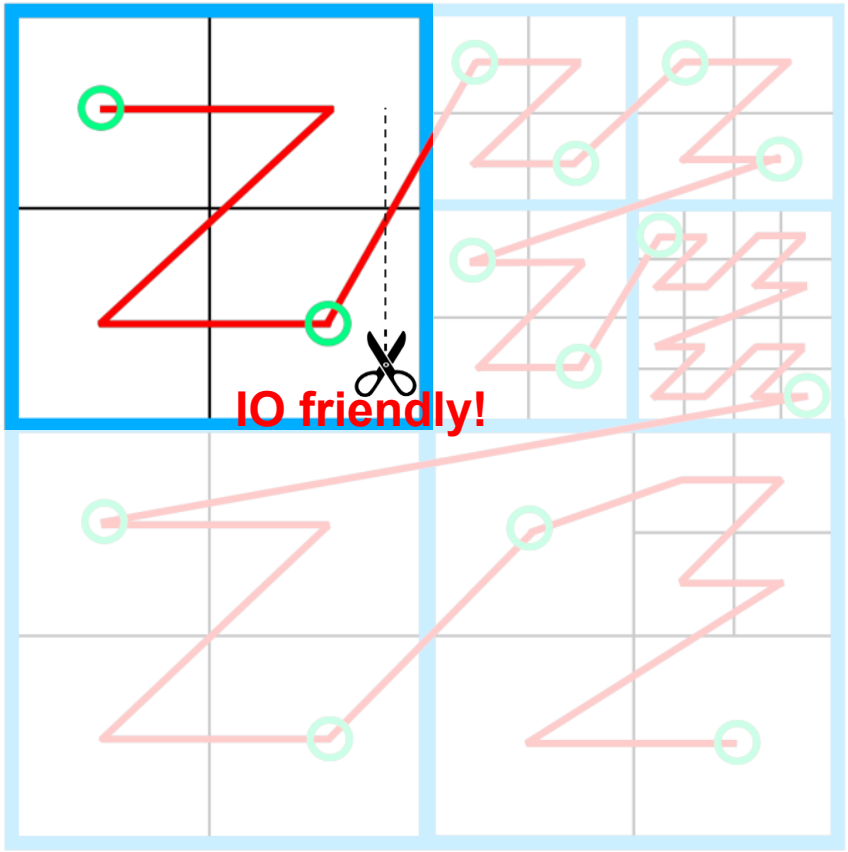


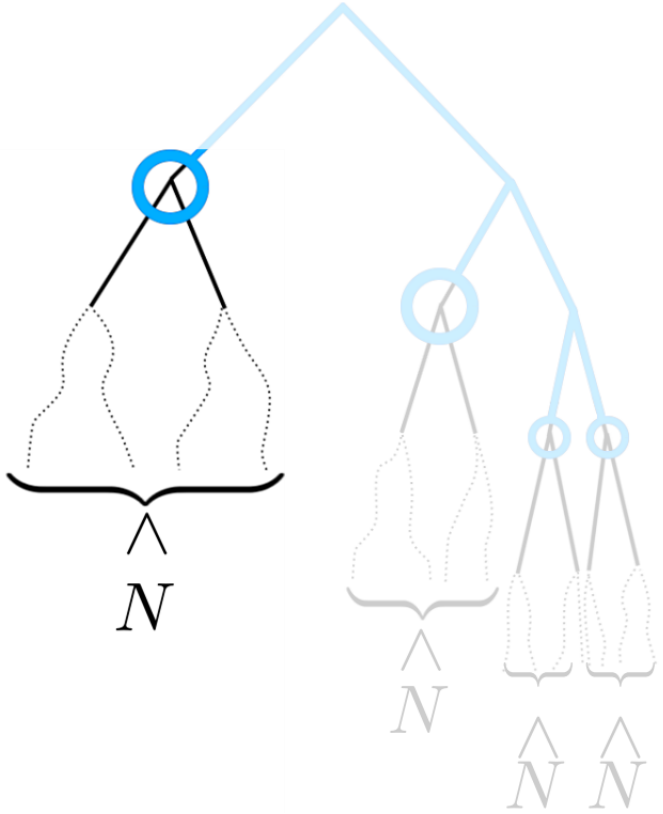
Обработка октодерева по частям



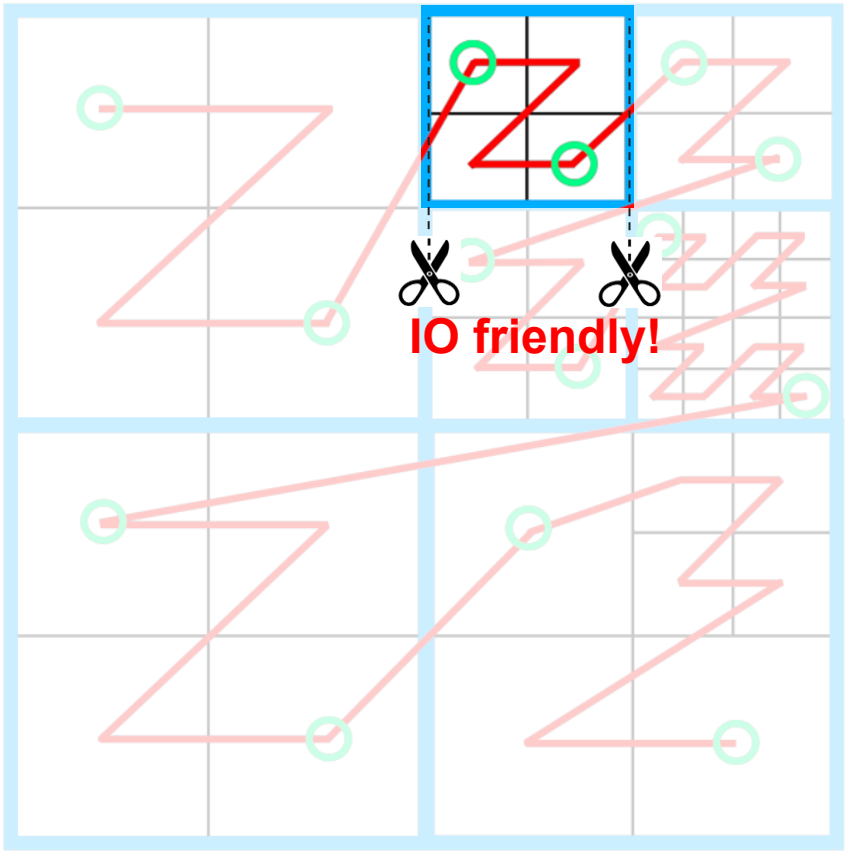


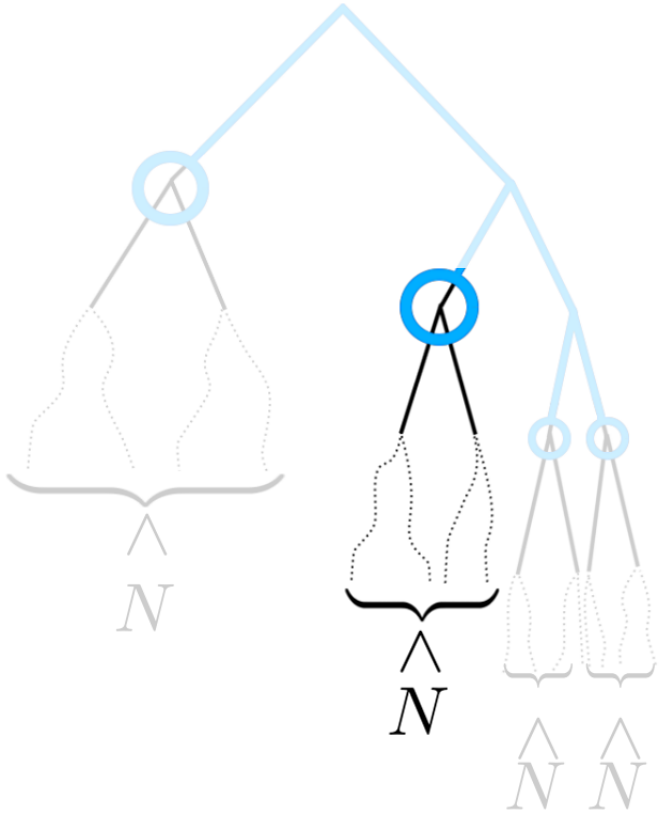
Обработка октодерева по частям



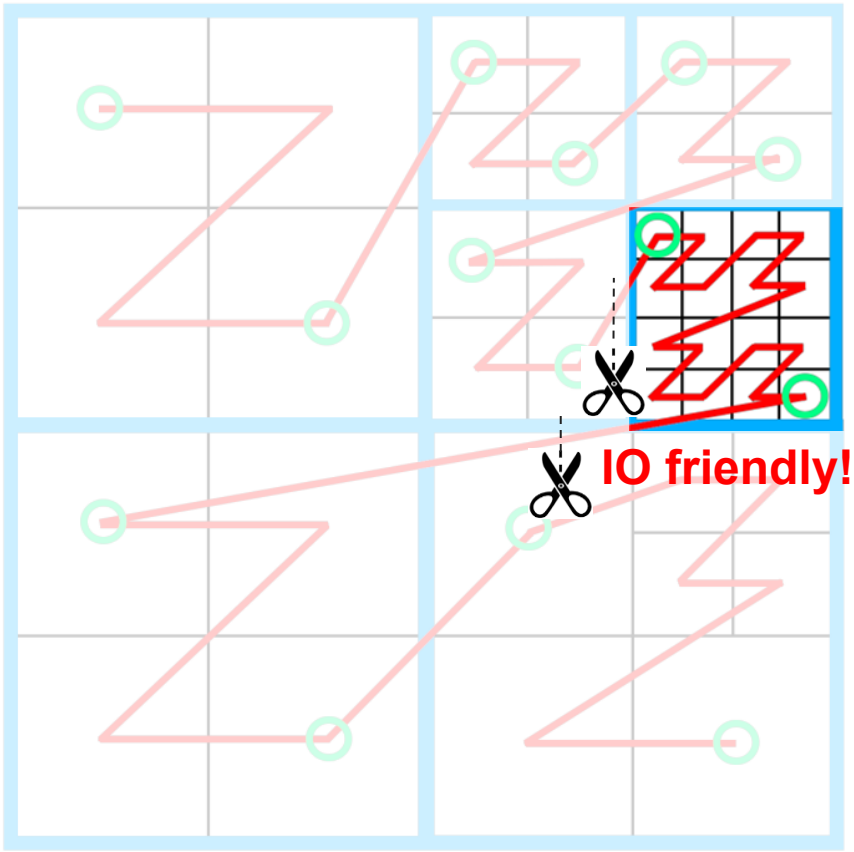


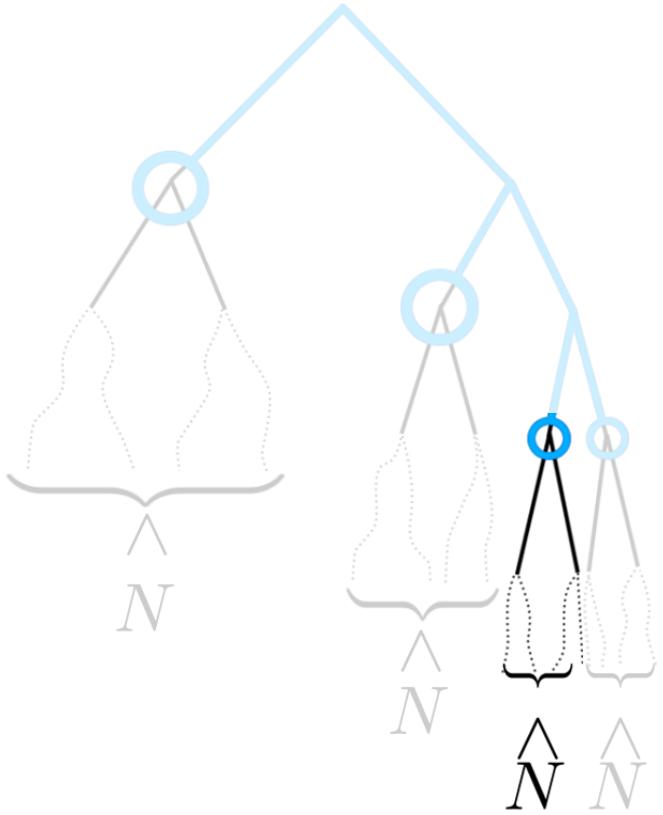
Обработка октодерева по частям



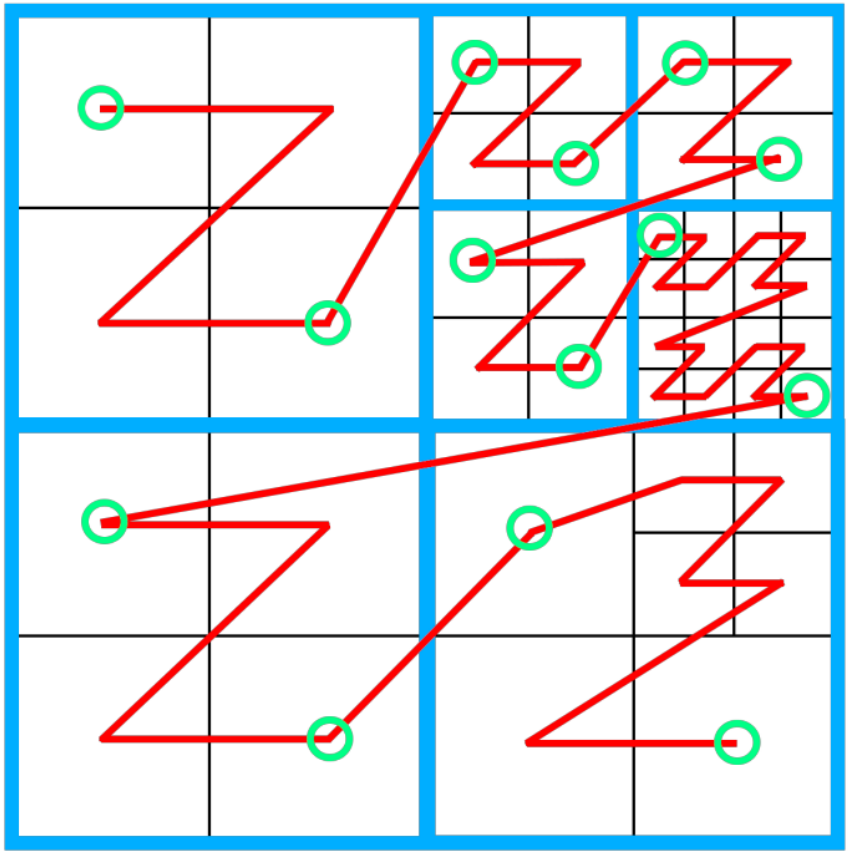


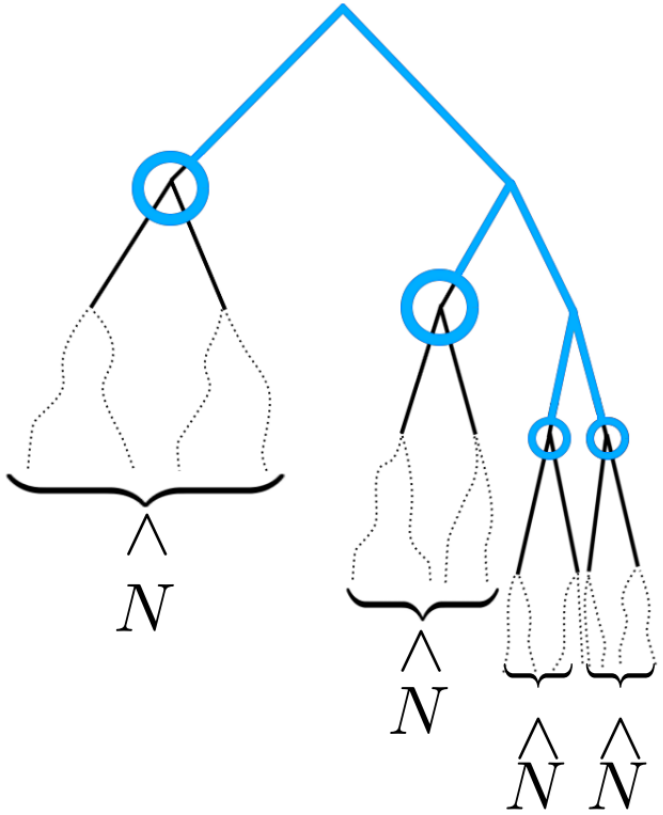
Обработка октодерева по частям





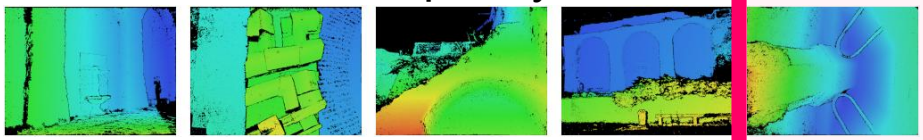
Обработка октодерева по частям

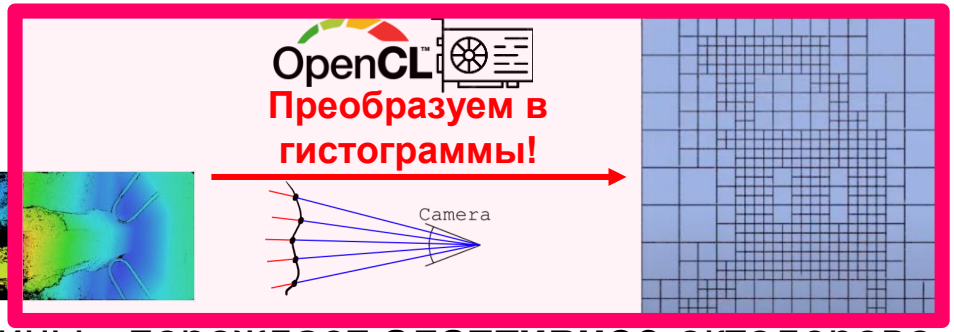




Итого

1) Есть множество карт глубины





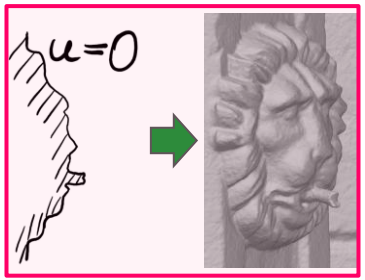
2) Множество точек всех карт глубины - порождает **адаптивное** октодерево (дискретизация пространства)
3) Балансируем октодерево 2:1 (по каждой стороне макс. два соседа)
4) Оптимизировали индикаторное поле (**много итераций + Coarse-to-Fine**)

OpenCL

TV-L^1:

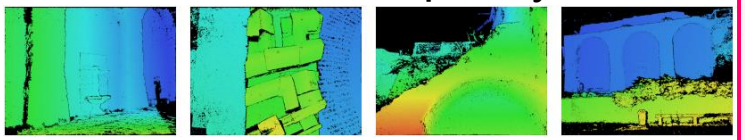
$$E = \int_\Omega \left\{ |\nabla u| + \lambda \sum_{i \in \mathcal{I}(\vec{x})} w_i(\vec{x}) |u - f_i| \right\} d\vec{x}$$

5) Извлекли поверхность маршировкой кубов



Итого

1) Есть множество карт глубины



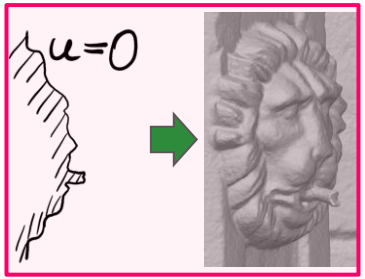
2) Множество точек всех карт глубины - порождает **адаптивное** октодерево (дискретизация пространства)
3) Балансируем октодерево 2:1 (по каждой стороне макс. два соседа)
4) Оптимизировали индикаторное поле (**много итераций + Coarse-to-Fine**)

OpenCL

TV-L^1:

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

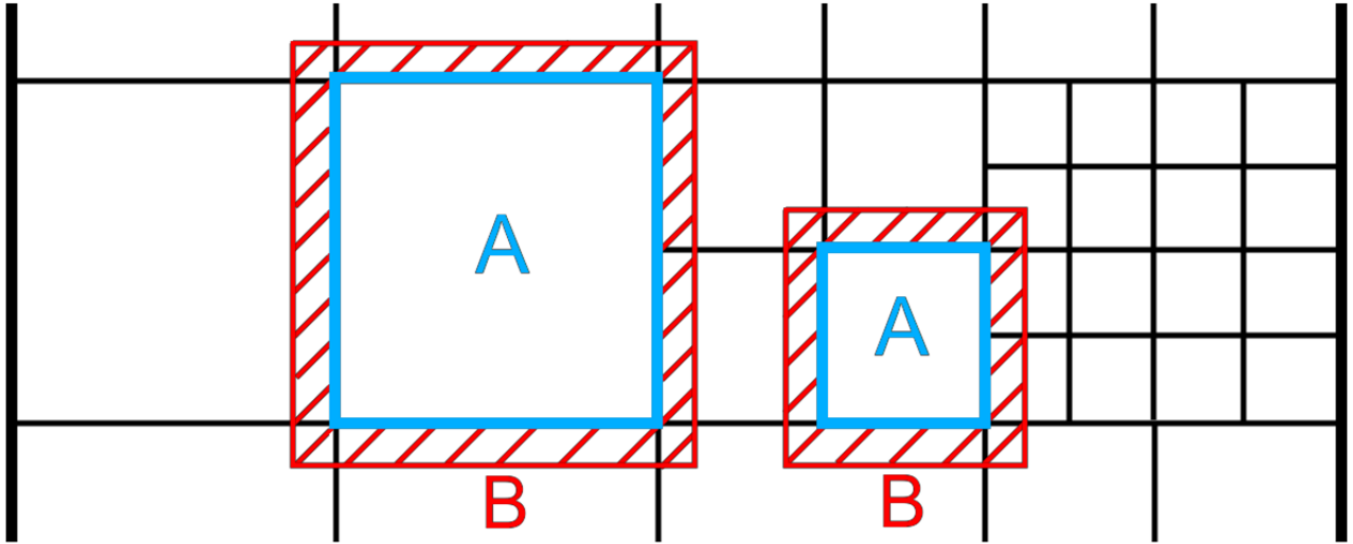
5) Извлекли поверхность маршировкой кубов



Обработка октодерева по частям

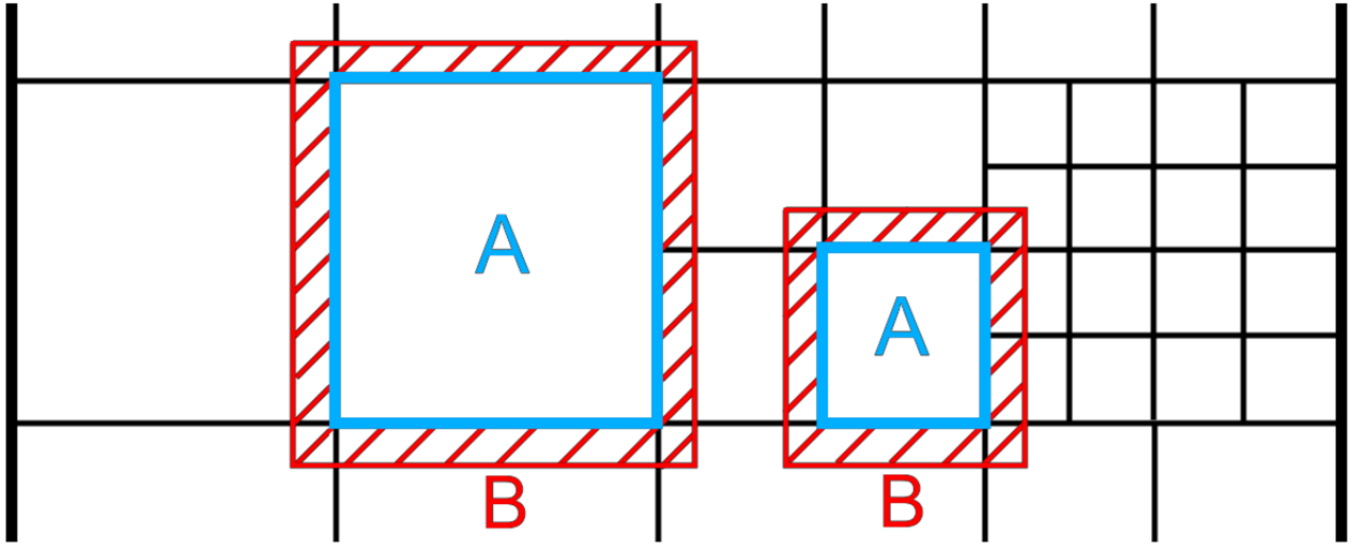
Обработка октодерева по частям

Обработка октодерева по частям



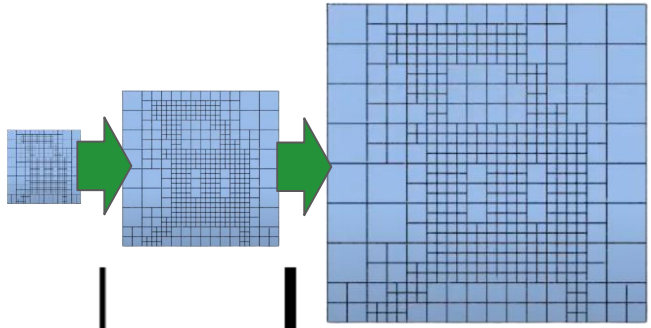
Заморозка границ!
Т.е. добавили в нашу минимизацию граничное условие.

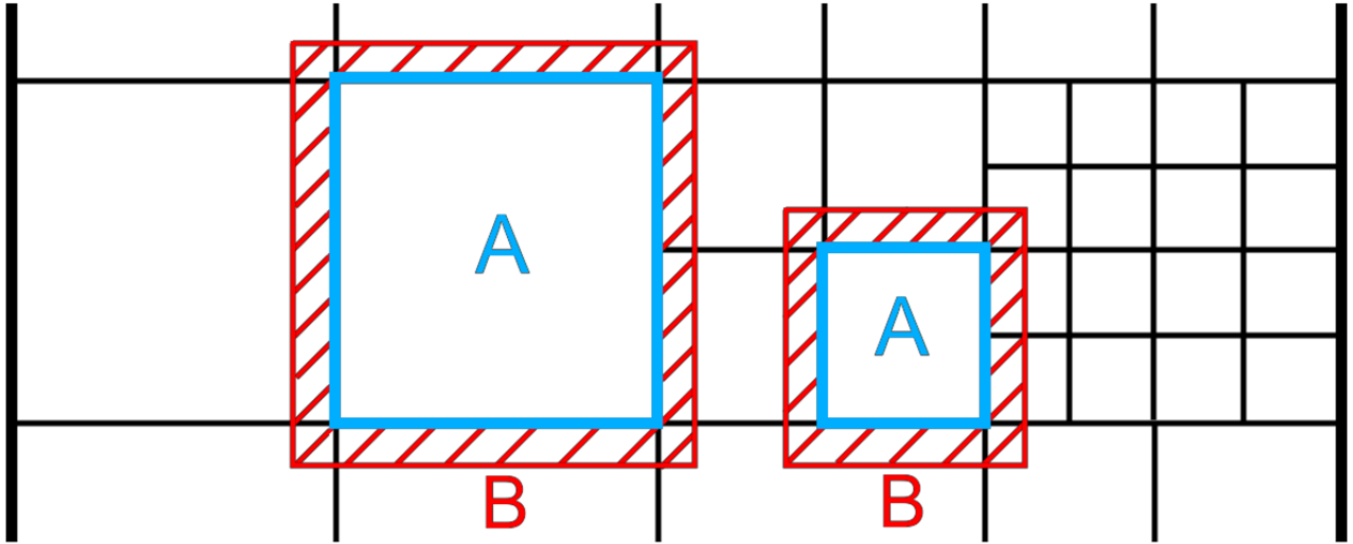
Обработка октодерева по частям



Заморозка границ!
Т.е. добавили в нашу минимизацию граничное условие.
Проблема курицы и яйца - мы не знаем результат соседа!
Откуда взять индикатор по периметру для заморозки?

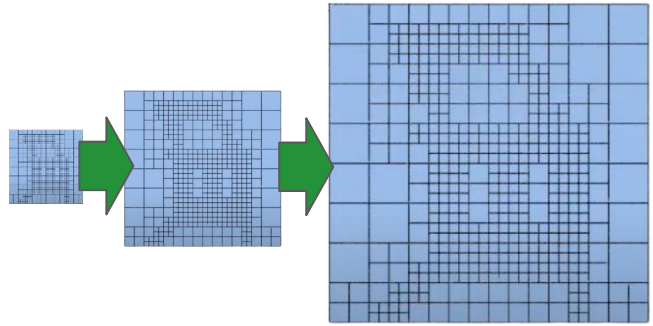
Обработка октодерева по частям

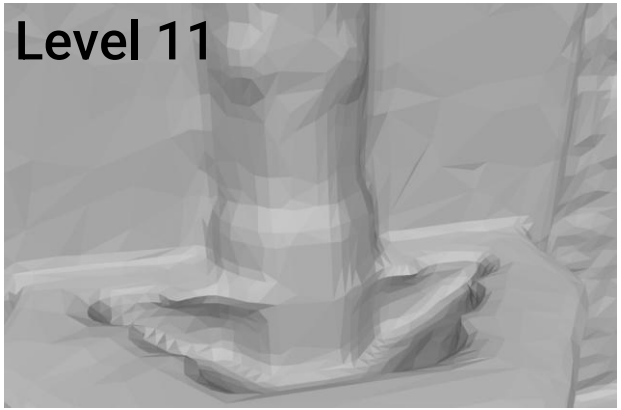




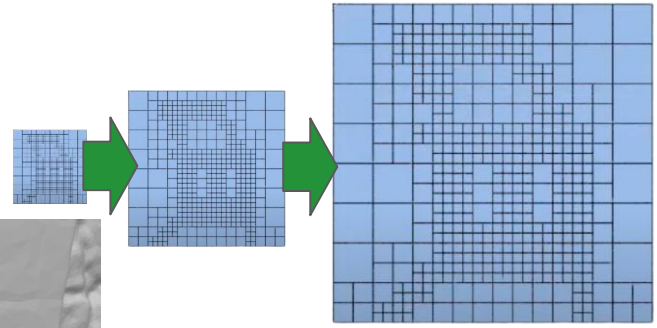
Заморозка границ!
Т.е. добавили в нашу минимизацию граничное условие.
Проблема курицы и яйца - мы не знаем результат соседа!
Откуда взять индикатор по периметру для заморозки?

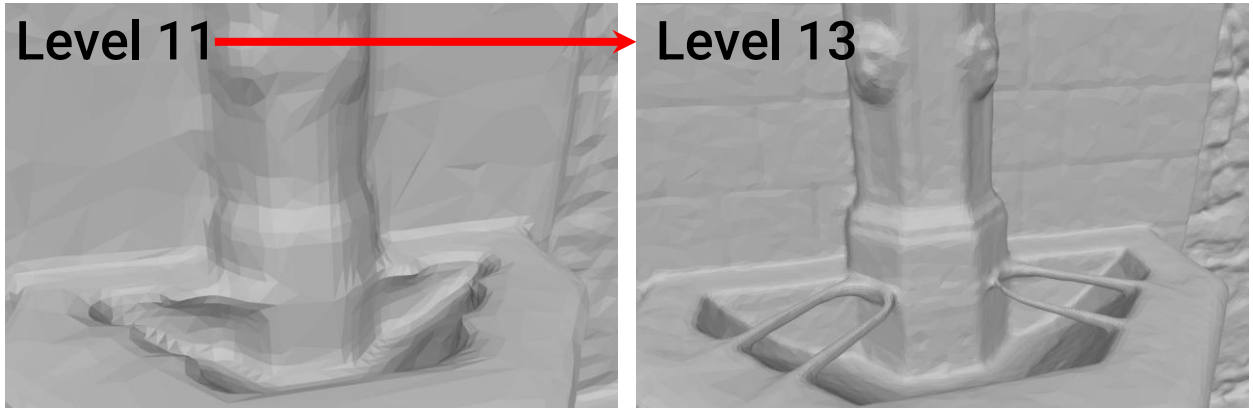
Обработка октодерева по частям



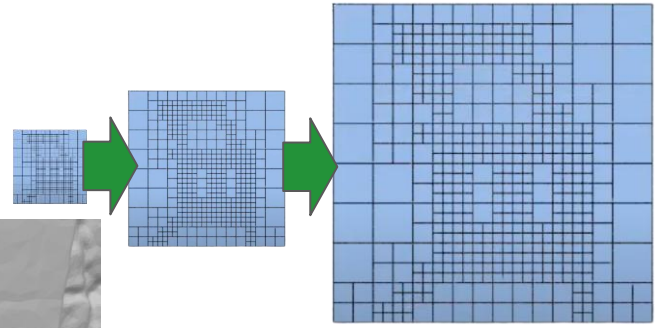


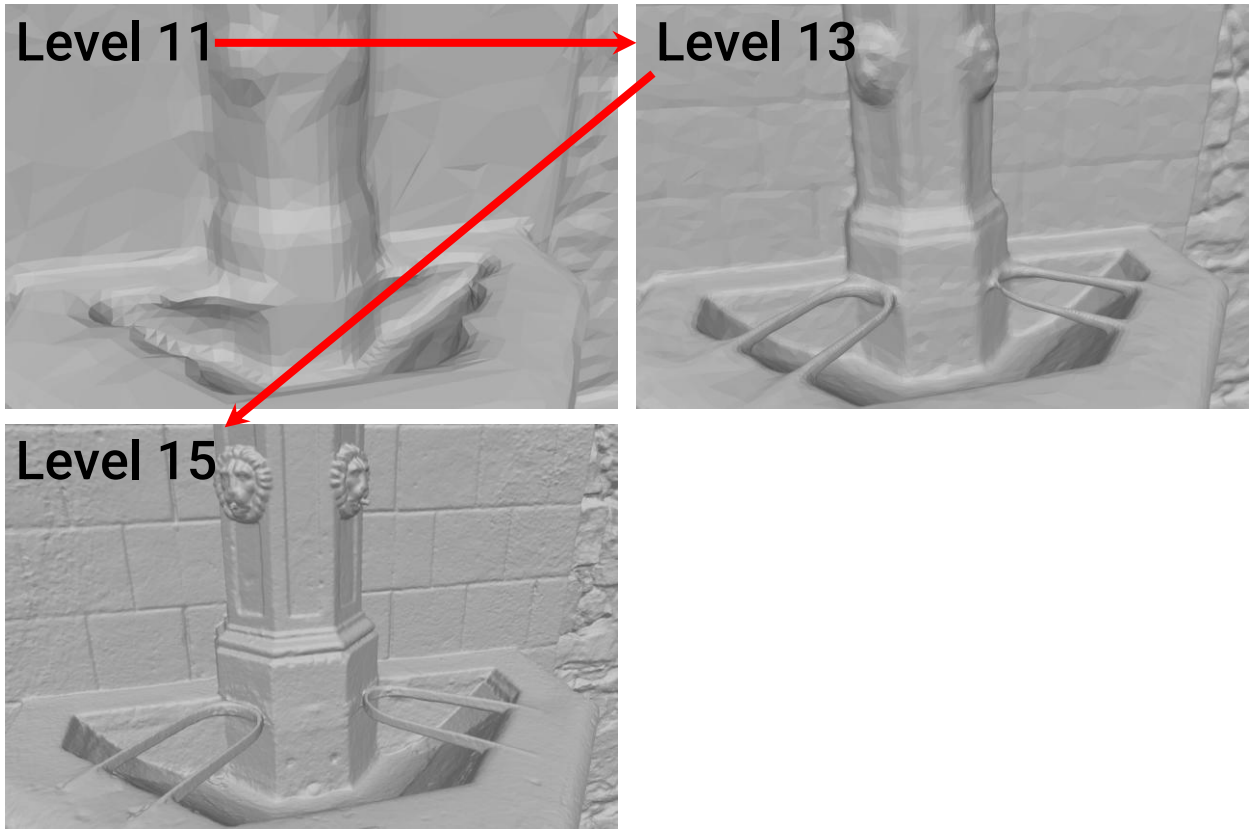
Обработка октодерева по частям



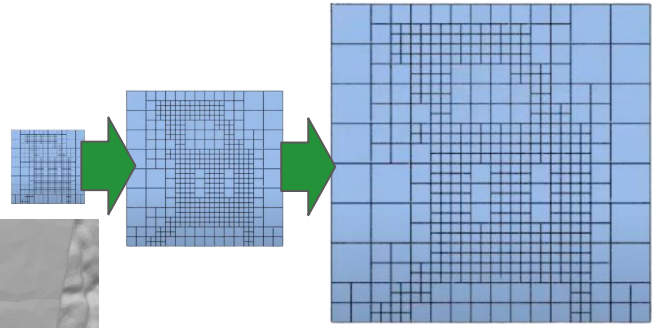


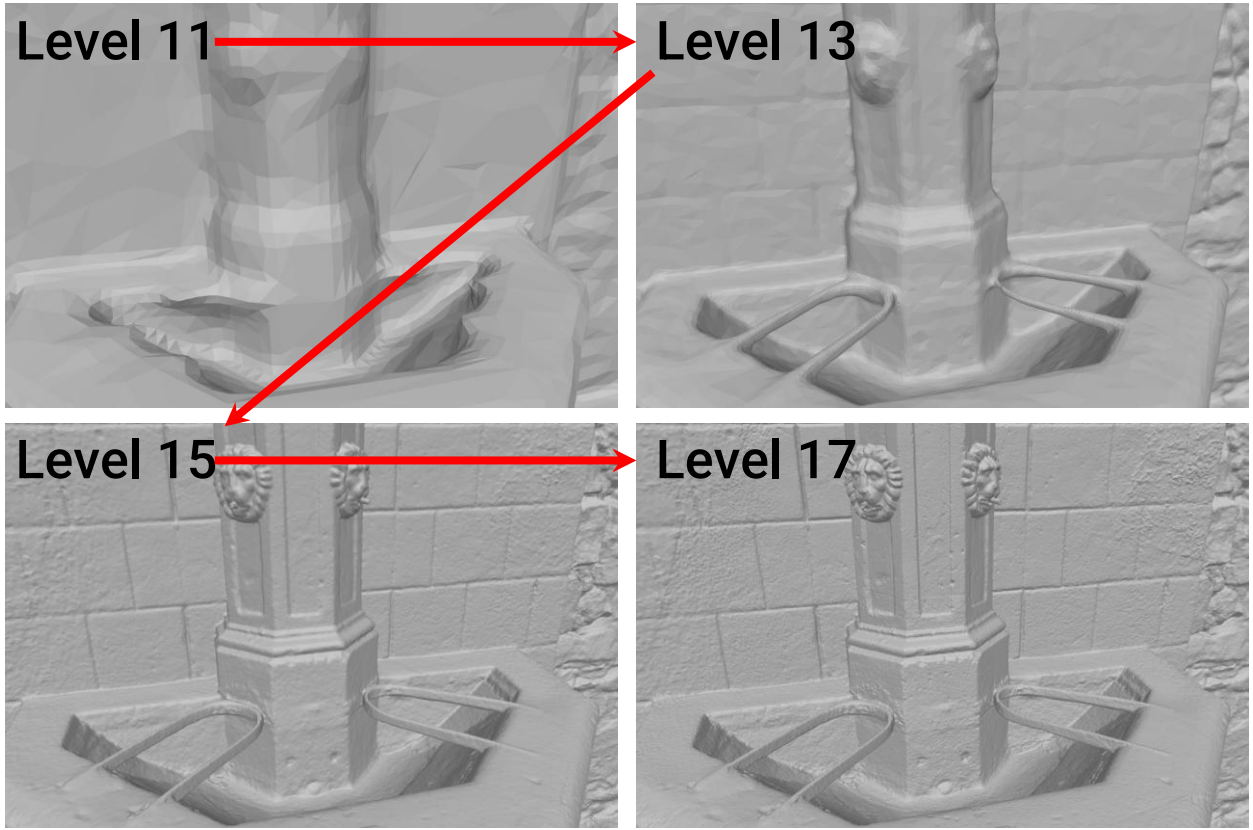
Обработка октодерева по частям





Обработка октодерева по частям

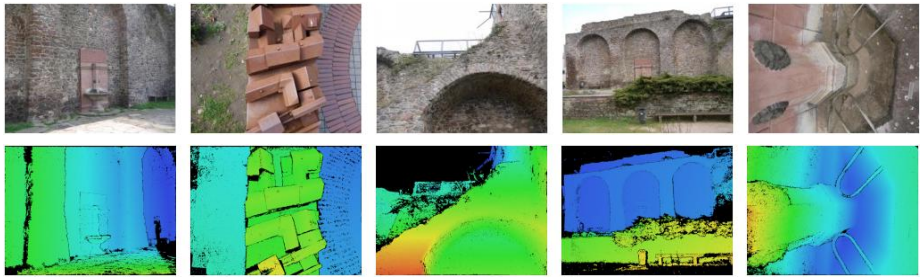




Извлечение поверхности ввиде треугольников

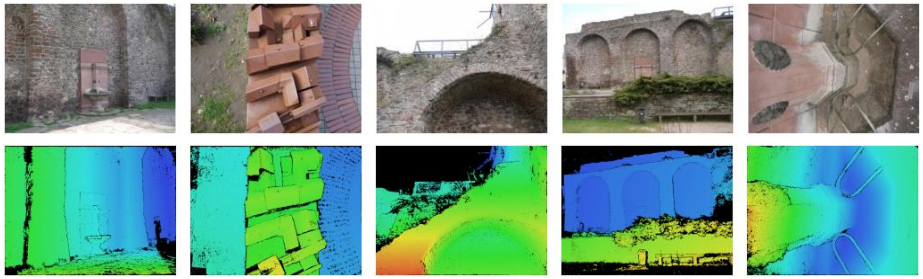
Маршировка кубов, QSlim упрощение геометрии

3D модель



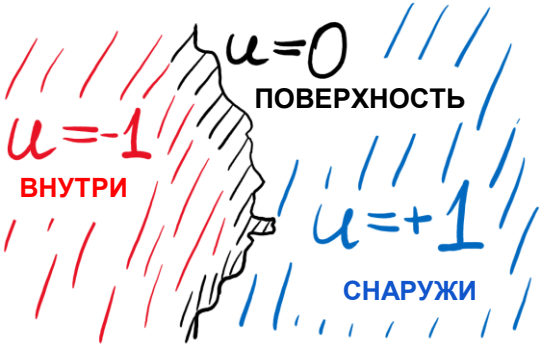
На вход: карты глубины

3D модель

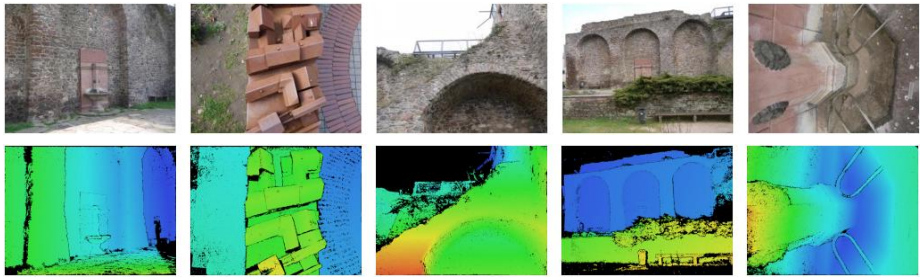


На вход: карты глубины

На выход: индикаторное скалярное поле

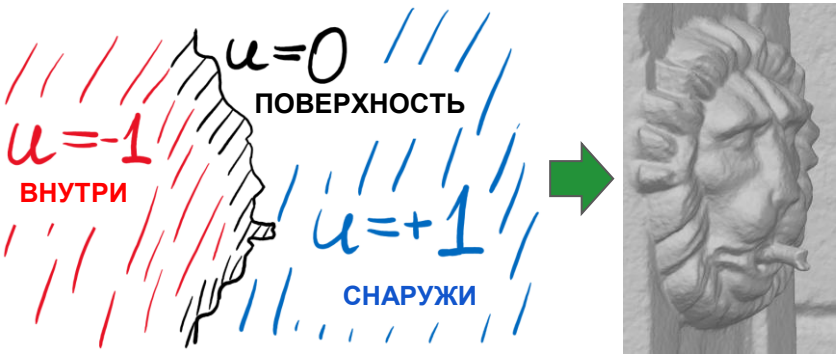


3D модель

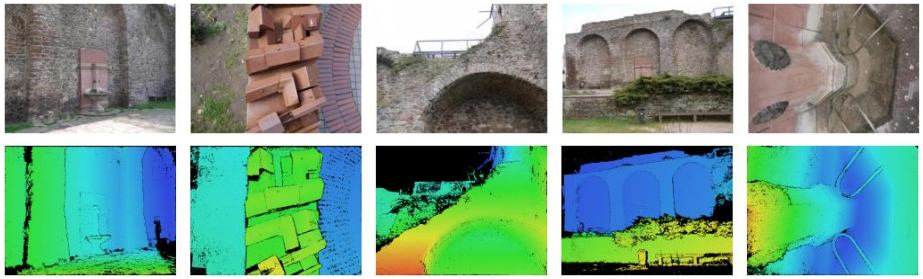


На вход: карты глубины

На выход: индикаторное скалярное поле

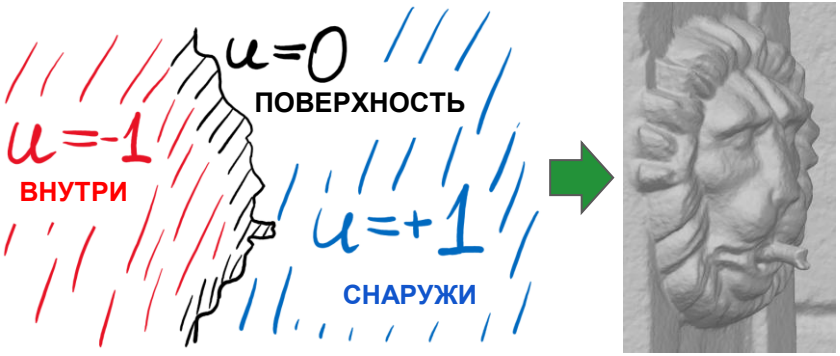


3D модель

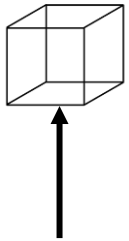


На вход: карты глубины

На выход: индикаторное скалярное поле

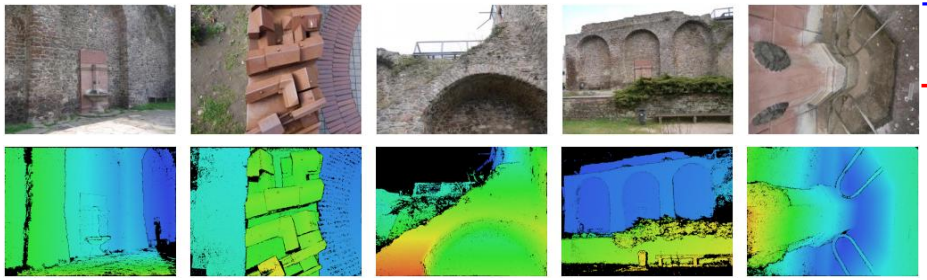


Маршировка кубов



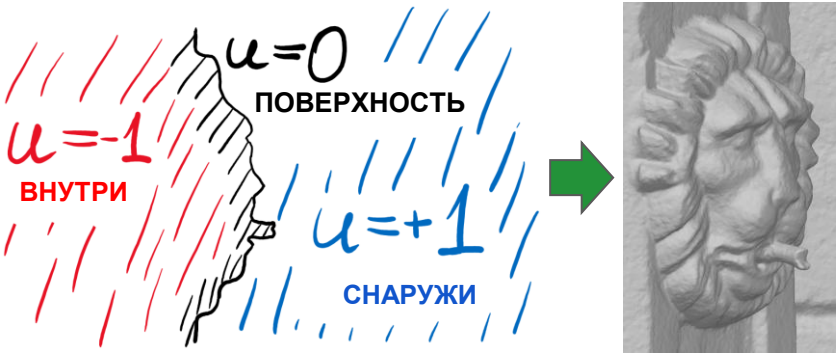
рассмотрим воксель нашего пространства

3D модель

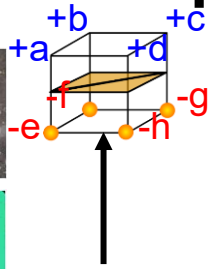


На вход: карты глубины

На выход: индикаторное скалярное поле



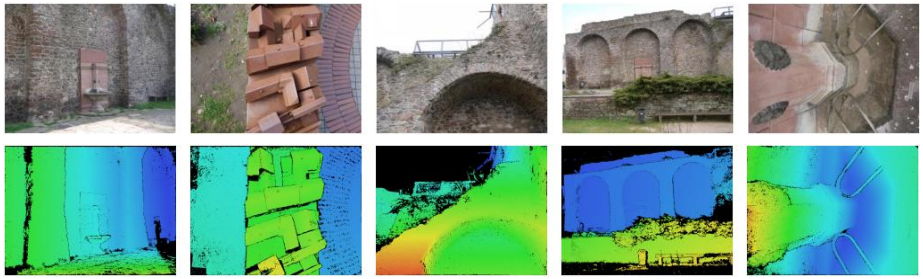
Маршировка кубов



рассмотрим воксель нашего пространства

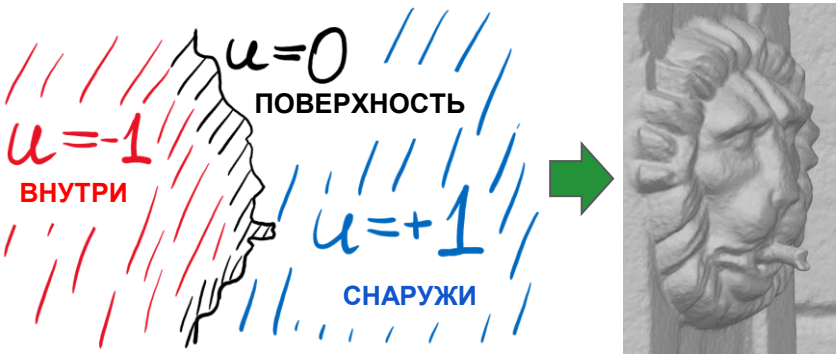
как зная индикаторное поле на его 8 углах понять где проходит поверхность?

3D модель

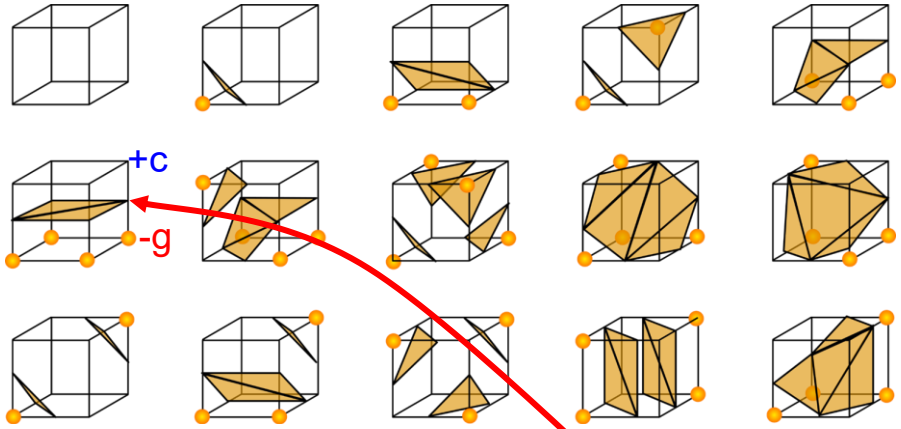


На вход: карты глубины

На выход: индикаторное скалярное поле



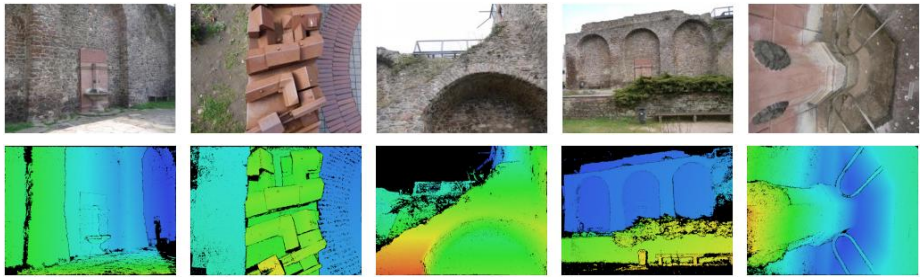
Маршировка кубов



А сколько всего случаев?

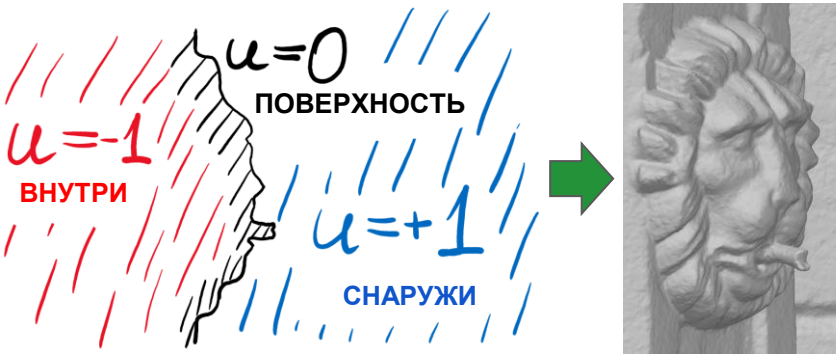
А где между углами вокселя ставить вершину?

3D модель

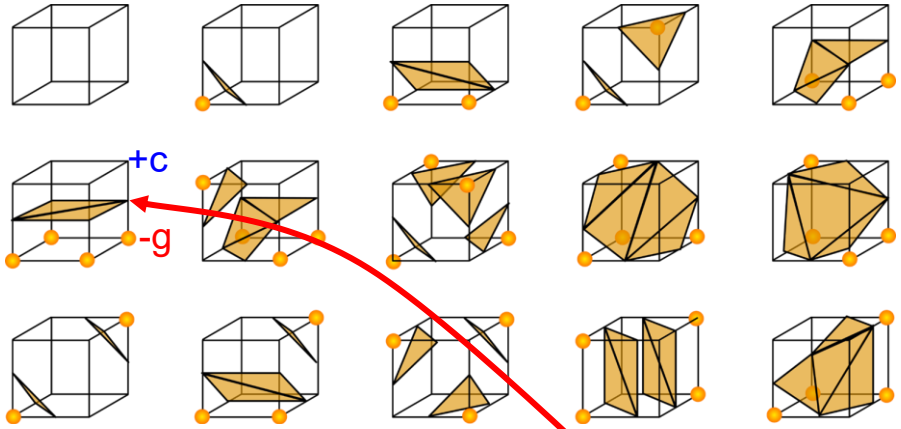


На вход: карты глубины

На выход: индикаторное скалярное поле



Маршировка кубов



А сколько всего случаев?

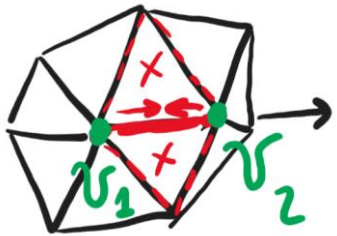
А где между углами вокселя ставить вершину?

А не много ли треугольников получится?
Не упадем ли по памяти на этапе объединения результата?

3D модель: упрощение геометрии (QSlim)

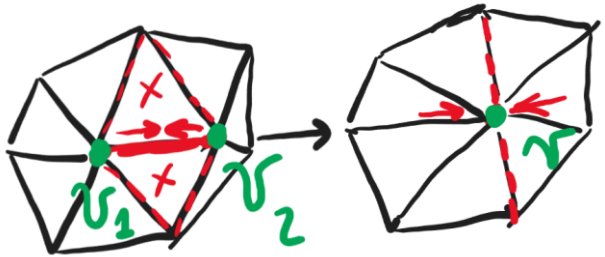
Simplifying surfaces with color and texture using quadric error metrics, **Garland et. al., 1998**

3D модель: упрощение геометрии (QSlim)



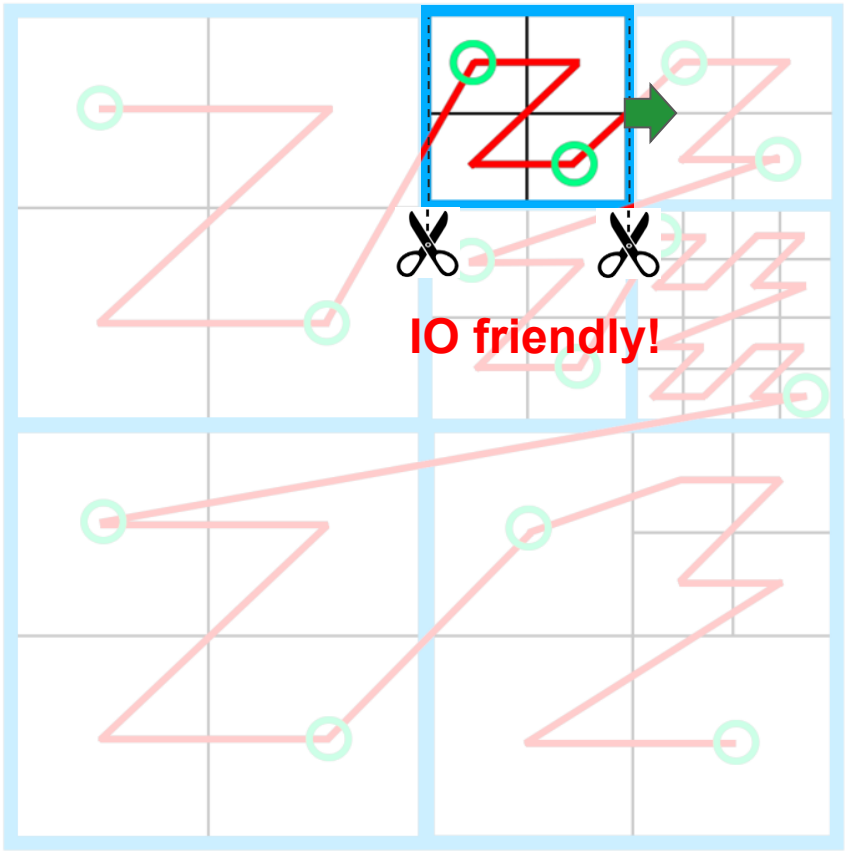
Simplifying surfaces with color and texture using quadric error metrics, **Garland et. al., 1998**

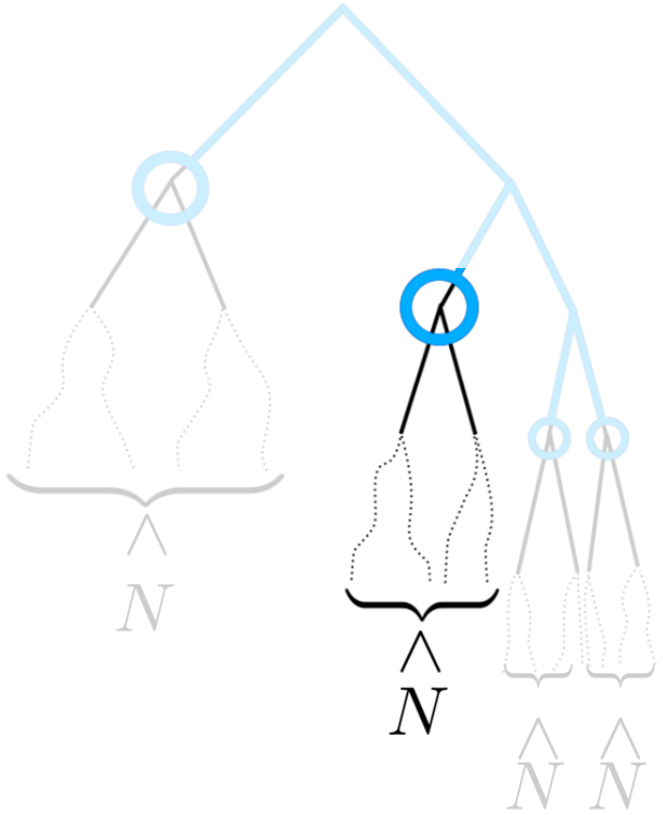
3D модель: упрощение геометрии (QSlim)



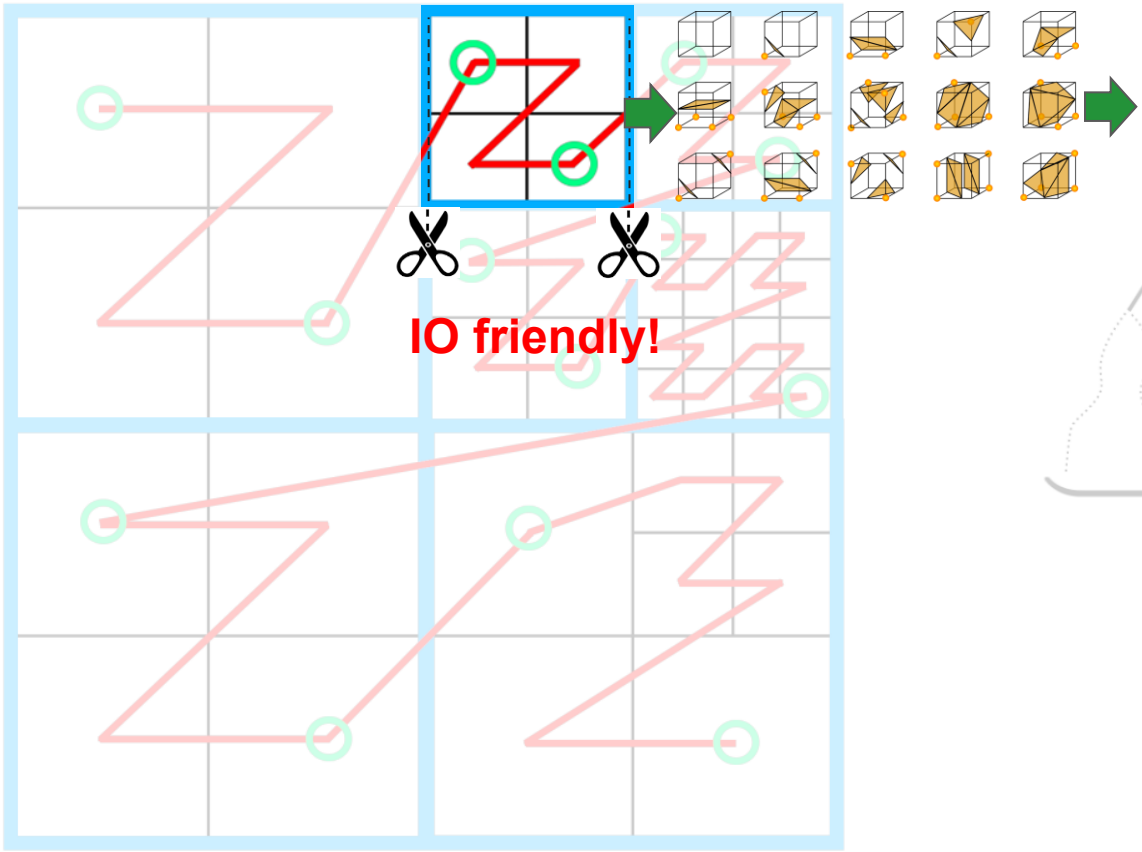
Simplifying surfaces with color and texture using quadric error metrics, **Garland et. al., 1998**

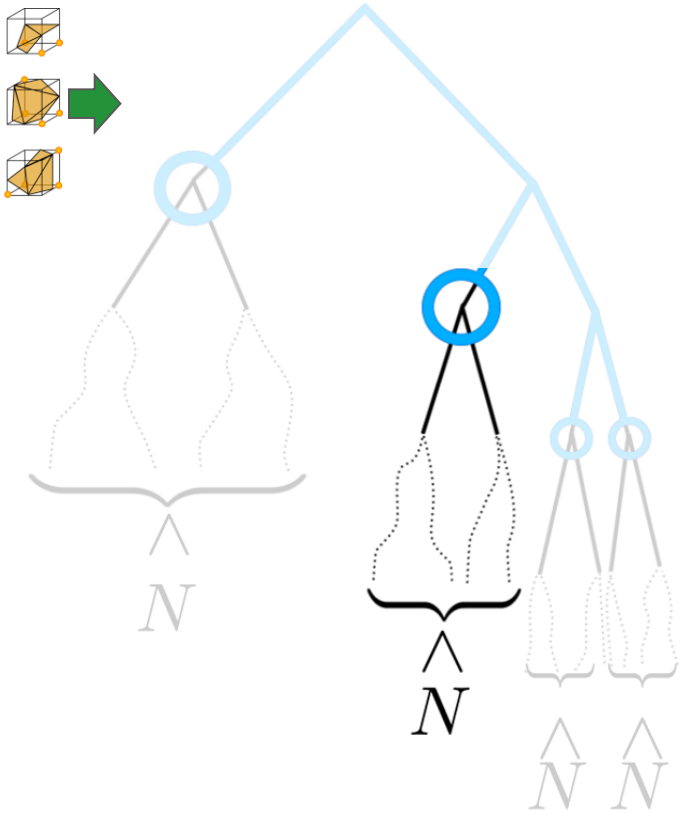
3D модель: упрощение геометрии (QSlim)



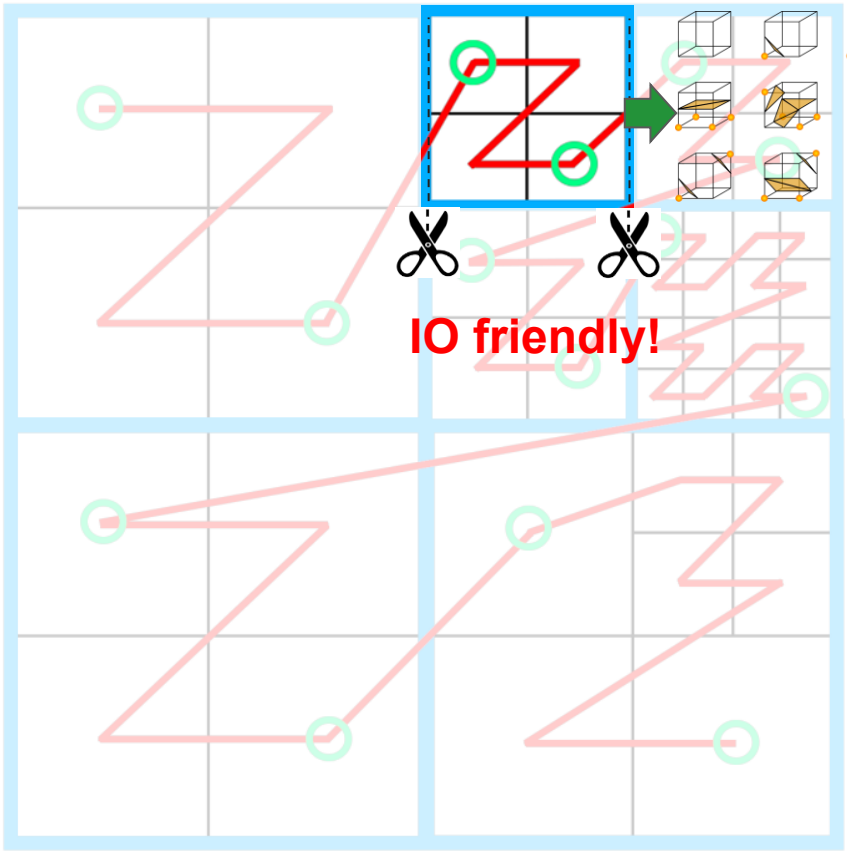


3D модель: упрощение геометрии (QSlim)

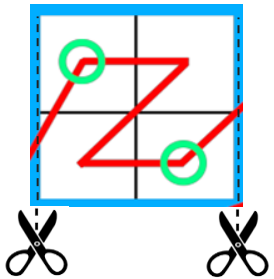


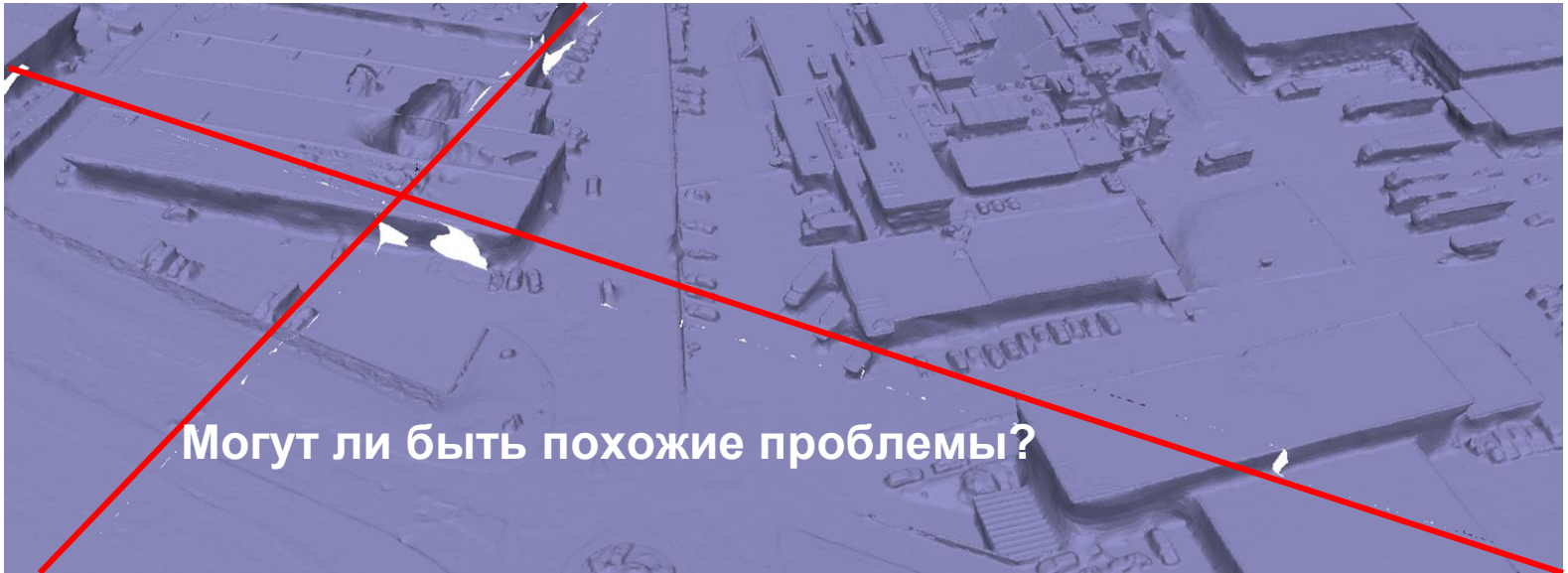


3D модель: упрощение геометрии (QSlim)

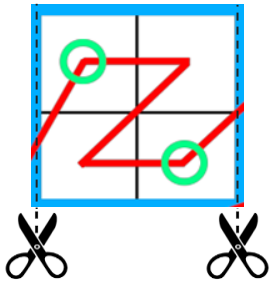


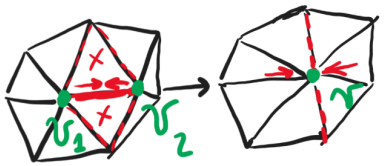
3D модель: упрощение геометрии (QSlim)



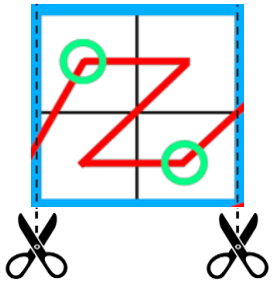


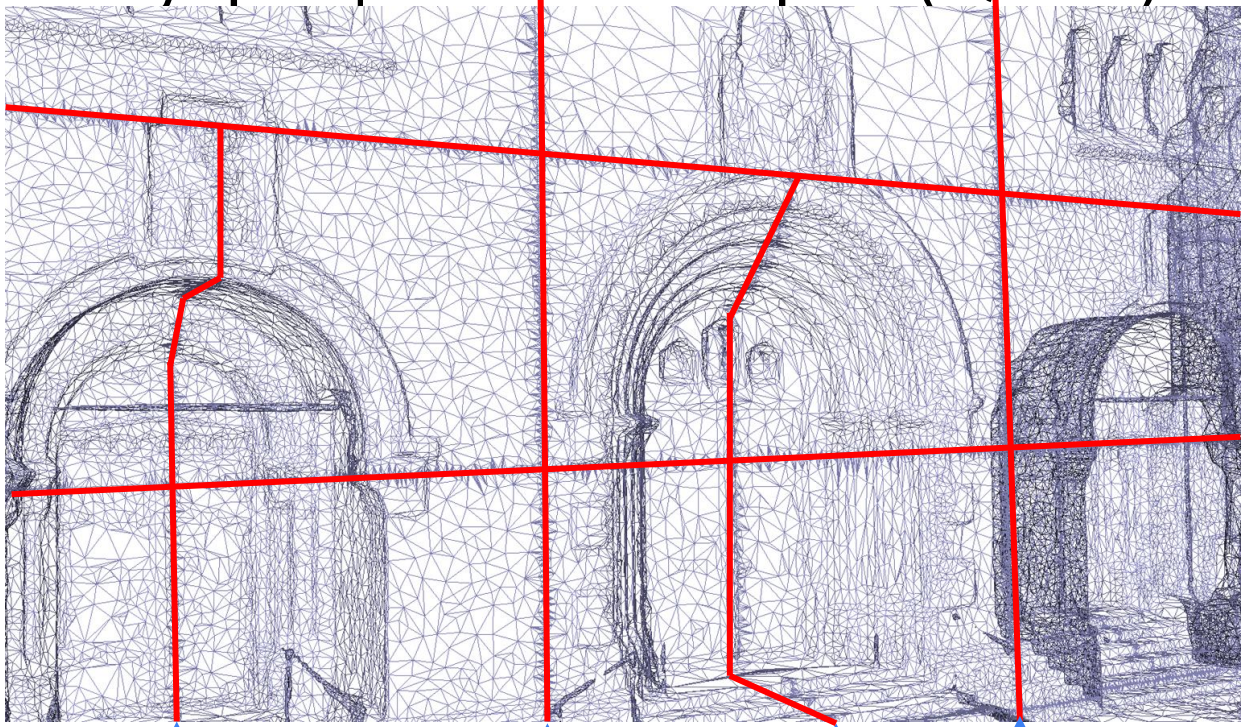
3D модель: упрощение геометрии (QSlim)

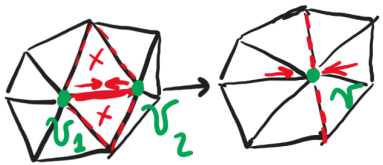




3D модель: упрощение геометрии (QSlim)

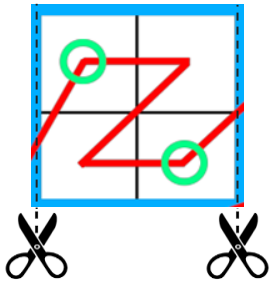


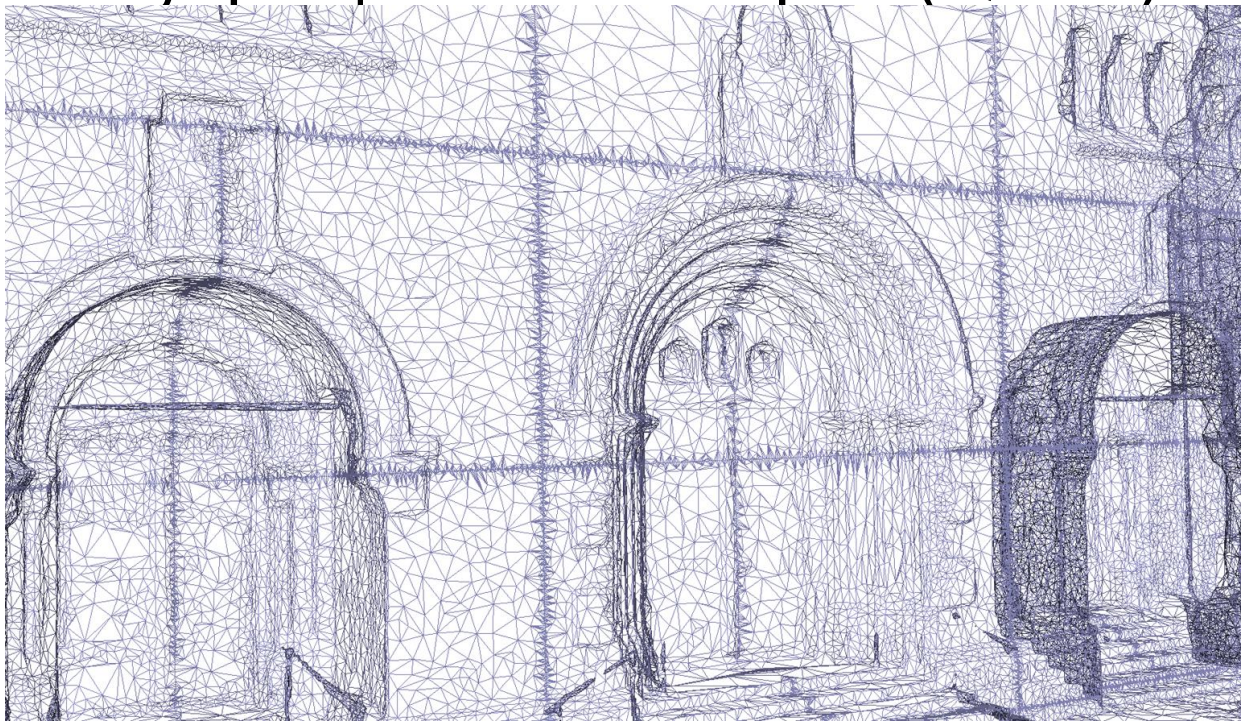


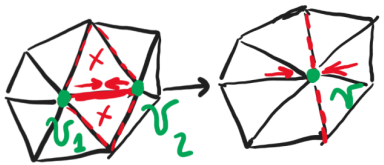


Заморозим ребра на границах! Запретим упрощать!

3D модель: упрощение геометрии (QSlim)

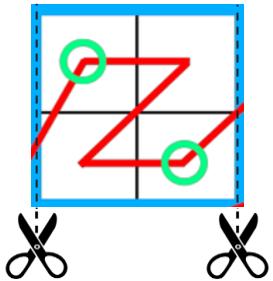


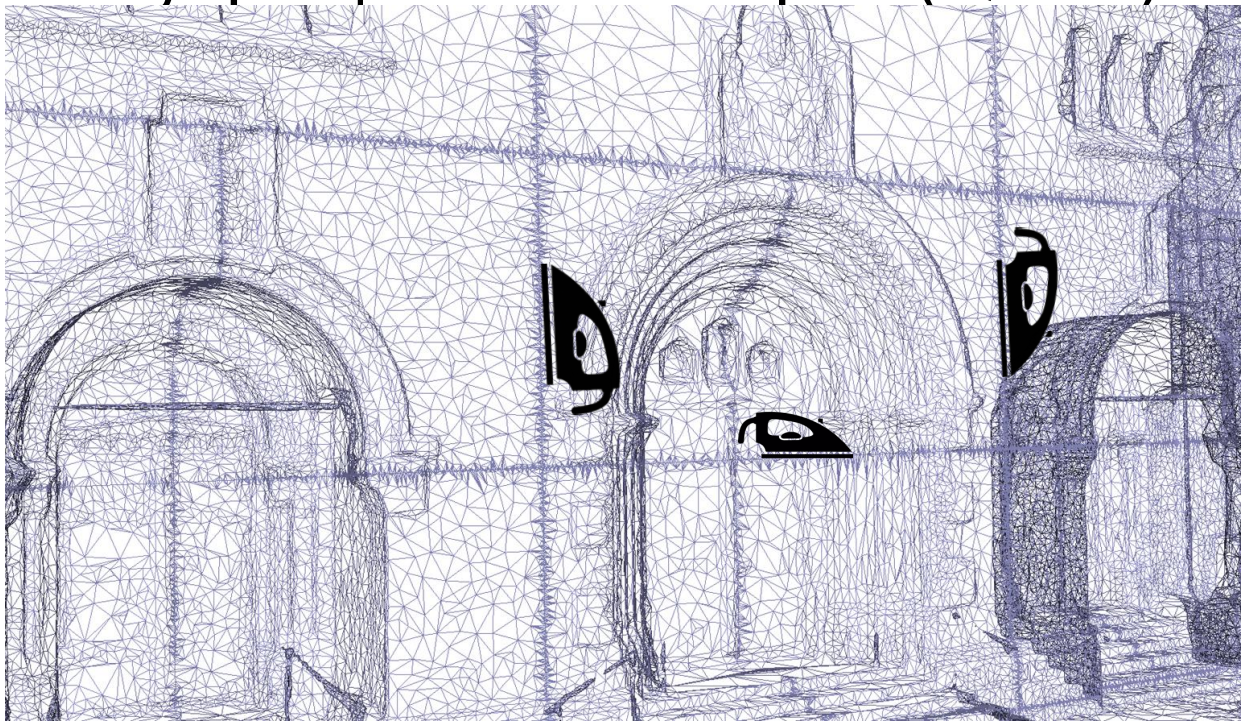


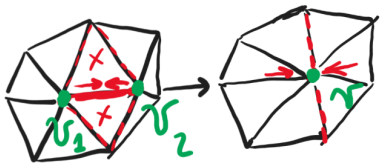


Как избавиться от этой корки?

3D модель: упрощение геометрии (QSlim)

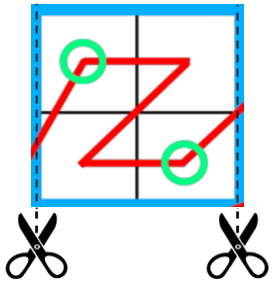


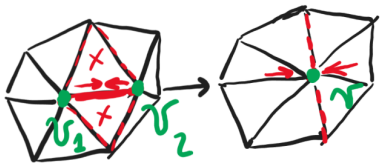




Как избавиться от этой корки?

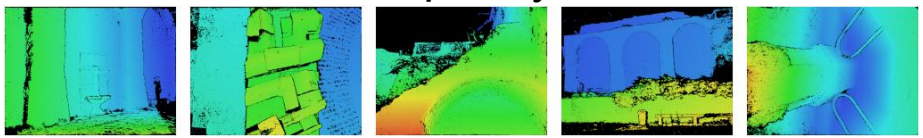
3D модель: упрощение геометрии (QSlim)





Итого

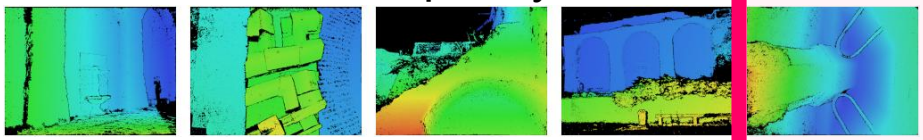
1) Есть множество карт глубины

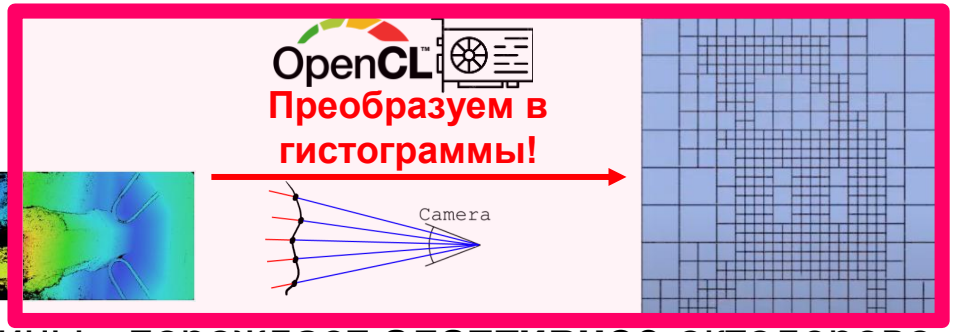


2) Множество точек всех карт глубины - порождает **адаптивное** октодерево (дискретизация пространства)
3) Балансируем октодерево 2:1 (по каждой стороне макс. два соседа)

Итого

1) Есть множество карт глубины

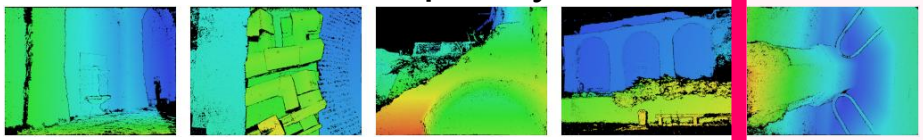


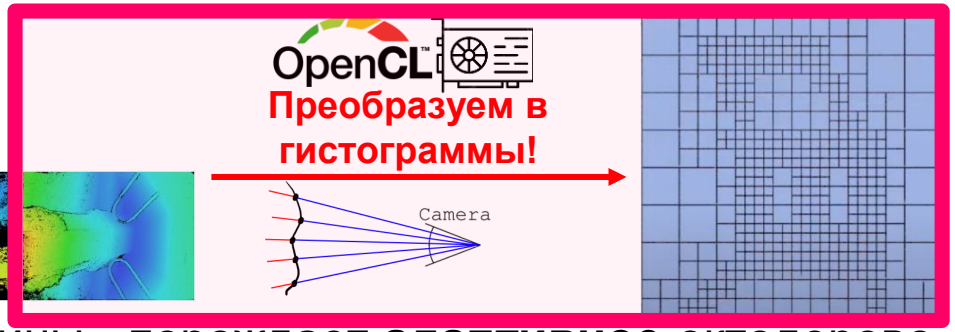


2) Множество точек всех карт глубины - порождает **адаптивное** октодерево (дискретизация пространства)
3) Балансируем октодерево 2:1 (по каждой стороне макс. два соседа)

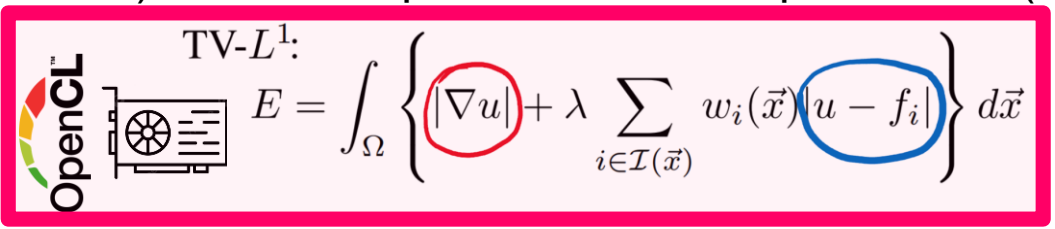
Итого

1) Есть множество карт глубины



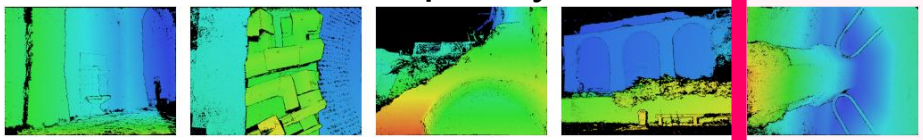


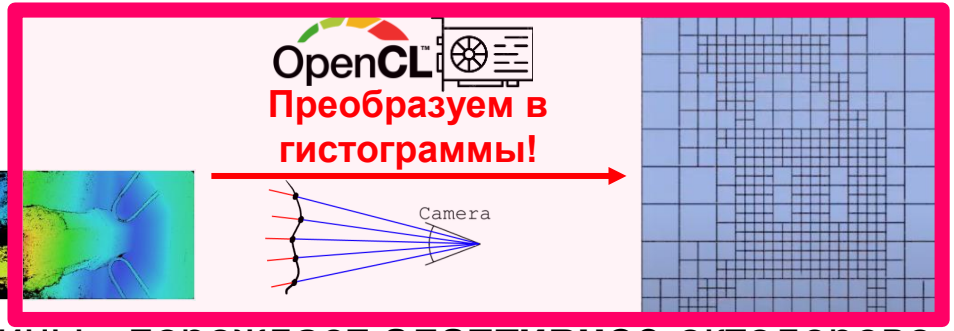
2) Множество точек всех карт глубины - порождает **адаптивное** октодерево (дискретизация пространства)
3) Балансируем октодерево 2:1 (по каждой стороне макс. два соседа)
4) Оптимизировали индикаторное поле (**много итераций + Coarse-to-Fine**)



Итого

1) Есть множество карт глубины





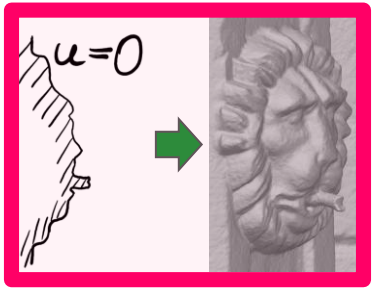
2) Множество точек всех карт глубины - порождает **адаптивное** октодерево (дискретизация пространства)
3) Балансируем октодерево 2:1 (по каждой стороне макс. два соседа)
4) Оптимизировали индикаторное поле (**много итераций + Coarse-to-Fine**)

OpenCL

TV-L^1:

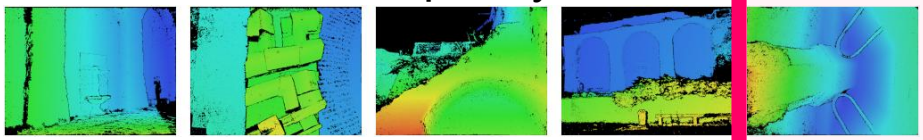
$$E = \int_\Omega \left\{ |\nabla u| + \lambda \sum_{i \in \mathcal{I}(\vec{x})} w_i(\vec{x}) |u - f_i| \right\} d\vec{x}$$

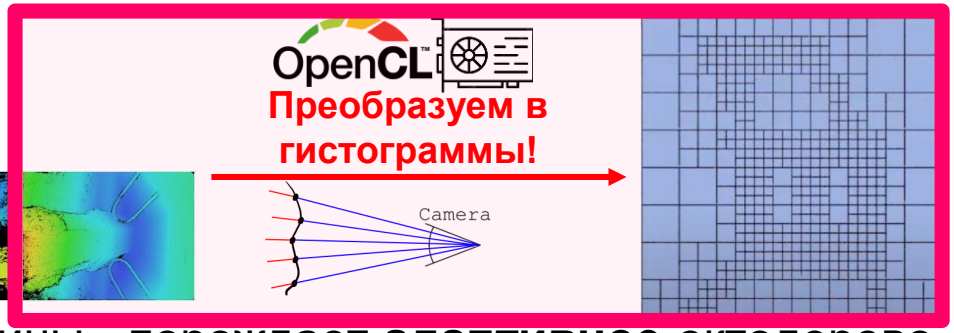
5) Извлекли поверхность маршировкой кубов



Итого

1) Есть множество карт глубины





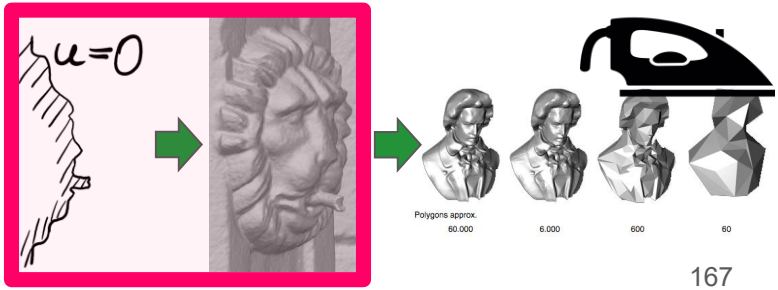
2) Множество точек всех карт глубины - порождает **адаптивное** октодерево (дискретизация пространства)
3) Балансируем октодерево 2:1 (по каждой стороне макс. два соседа)
4) Оптимизировали индикаторное поле (**много итераций + Coarse-to-Fine**)

OpenCL

TV-L^1:

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

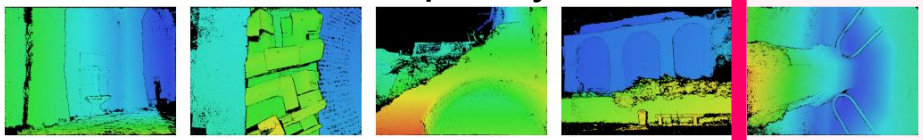
5) Извлекли поверхность маршировкой кубов

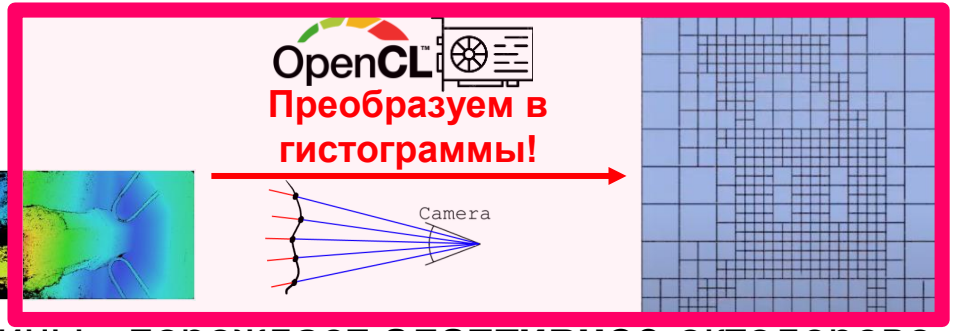


Итого

Out-of-core свойство идет рука об руку с параллелизуемостью на кластере!

1) Есть множество карт глубины



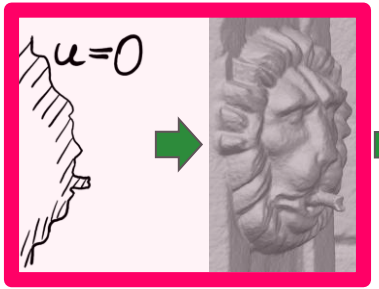


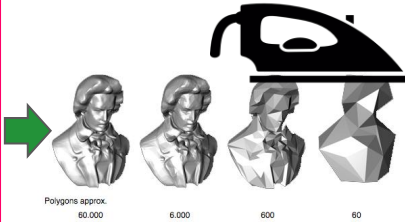
2) Множество точек всех карт глубины - порождает **адаптивное** октодерево (дискретизация пространства)
3) Балансируем октодерево 2:1 (по каждой стороне макс. два соседа)
4) Оптимизировали индикаторное поле (**много итераций + Coarse-to-Fine**)

OpenCL

TV-L^1:

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





5) Извлекли поверхность маршировкой кубов

Reality Capture 1.0.3	Metashape 1.6.0
1 hour 12 minutes	*43 minutes*

Reality Capture 1.0.3	Metashape 1.6.0
1 hour 12 minutes	*43 minutes*

Intel i9-7900x + GTX 1080 Ti

6 лет назад

Reality Capture 1.0.3	Metashape 1.6.0
1 hour 12 minutes	*43 minutes*

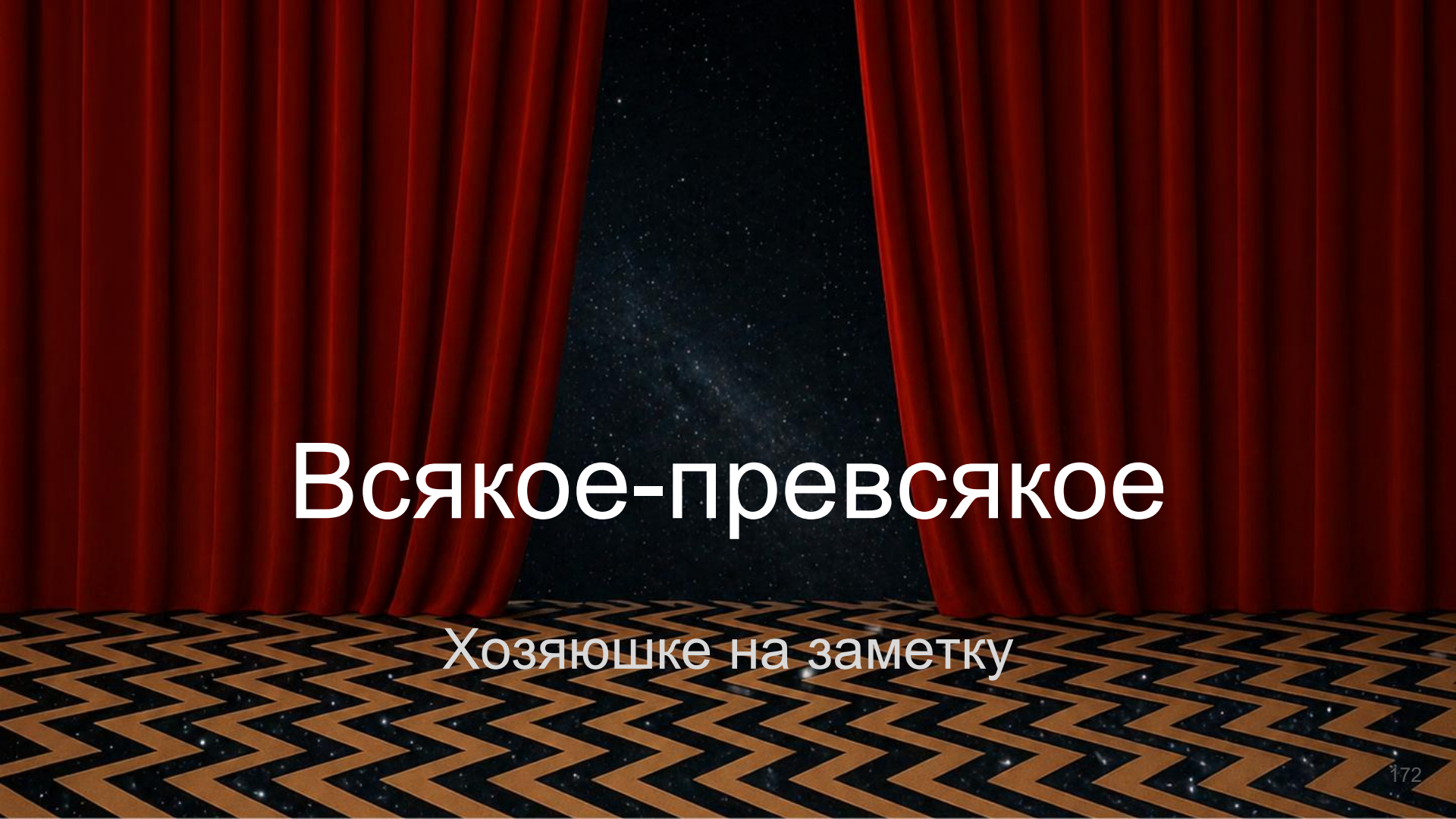
Intel i9-7900x + GTX 1080 Ti

6 лет назад

Metashape 2.3.1

5 minutes

AMD 9950X3D + RTX 5070 Ti


Всякое-превсякое
Хозяюшке на заметку


LEGO CITY
6-12
LEGO


LEGO CITY
6-12
LEGO


Еще и слабые тактильные ощущения…
LEGO CITY
6-12
LEGO


LEGO CITY
6-12
LEGO

Простой и удобный способ ловить ошибки

```
4182        rassert(less( a: cubes[i - from - 1], b: cubes[i - from]), 239101382);
```

предикат - всегда верен

Простой и удобный способ ловить ошибки

```
4182        ····rassert(less(a.cubes[i - from - 1], b.cubes[i - from]), 239101382);
```

~~предикат - всегда верен~~

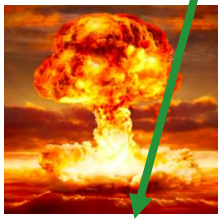
Простой и удобный способ ловить ошибки

```
4176    ListFileReader<tgv::CubeWithRadius> reader(linear_octree_cubes_path, params.ooc_compression_level);
4177    for (uint64_t i = from; i < to; ++i) {
4178        tgv::CubeWithRadius next = reader.next();
4179        rassert(next.radius_votes >= 1, 239101376);
4180        cubes[i - from] = next;
4181        if (i > from) {
4182            rassert(less(a: cubes[i - from - 1], b: cubes[i - from]), 239101382);
4183        }
4184    }
```

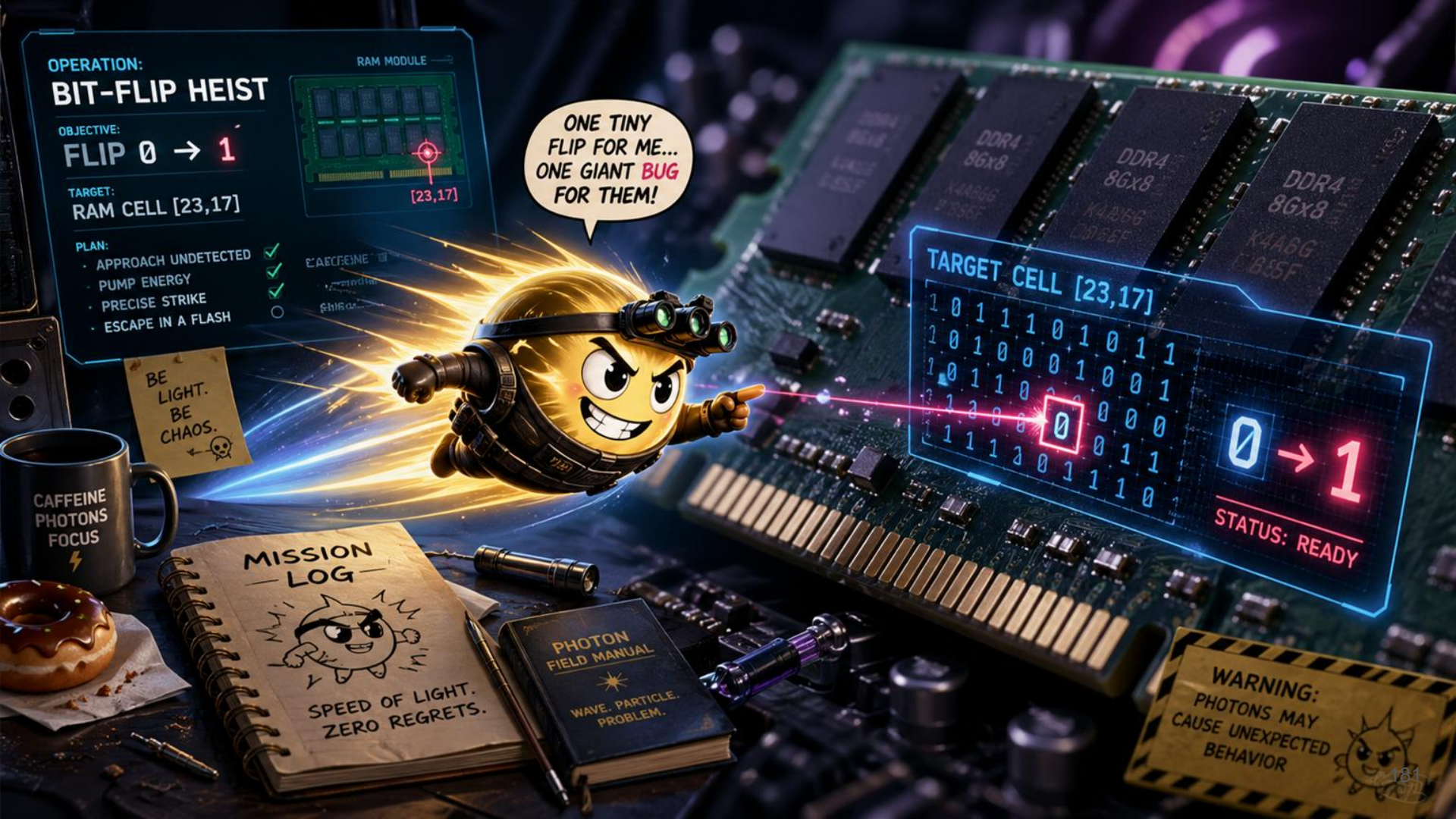
Простой и удобный способ ловить ошибки

```
3190    for (size_t i = 1; i < group_unqiue_cubes.size(); ++i) {
3191        rassert(cmp( a: group_unqiue_cubes[i - 1], b: group_unqiue_cubes[i]), 239101019);
3192    }
```

```
4176    ListFileReader<tgv::CubeWithRadius> reader(linear_octree_cubes_path, params.ooc_compression_level);
4177    for (uint64_t i = from; i < to; ++i) {
4178        tgv::CubeWithRadius next = reader.next();
4179        rassert(next.radius_votes >= 1, 239101376);
4180        cubes[i - from] = next;
4181        if (i > from) {
4182            rassert(less( a: cubes[i - from - 1], b: cubes[i - from]), 239101382);
4183        }
4184    }
```



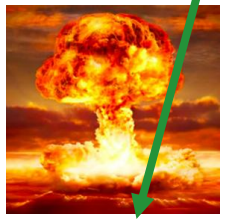
НО КАК?

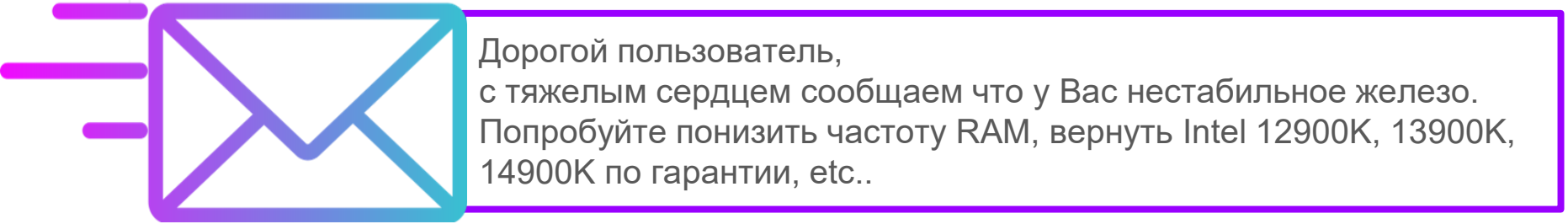

OPERATION:
BIT-FLIP HEIST
OBJECTIVE:
FLIP 0 → 1
TARGET:
RAM CELL [23,17]
PLAN:
· APPROACH UNDETECTED
· PUMP ENERGY
· PRECISE STRIKE
· ESCAPE IN A FLASH
RAM MODULE
[23,17]
ONE TINY FLIP FOR ME... ONE GIANT BUG FOR THEM!
BE LIGHT. BE CHAOS.
TARGET CELL [23,17]
0 → 1
STATUS: READY
CAFFEINE PHOTONS FOCUS
MISSION LOG
SPEED OF LIGHT. ZERO REGRETS.
PHOTON FIELD MANUAL
WAVE. PARTICLE. PROBLEM.
WARNING: PHOTONS MAY CAUSE UNEXPECTED BEHAVIOR

Простой и удобный способ ловить ошибки

```
3190    for (size_t i = 1; i < group_unqiue_cubes.size(); ++i) {
3191        rassert(cmp( a: group_unqiue_cubes[i - 1], b: group_unqiue_cubes[i]), 239101019);
3192    }
```

```
4176    ListFileReader<tgv::CubeWithRadius> reader(linear_octree_cubes_path, params.ooc_compression_level);
4177    for (uint64_t i = from; i < to; ++i) {
4178        tgv::CubeWithRadius next = reader.next();
4179        rassert(next.radius_votes >= 1, 239101376);
4180        cubes[i - from] = next;
4181        if (i > from) {
4182            rassert(less( a: cubes[i - from - 1], b: cubes[i - from]), 239101382);
4183        }
4184    }
```

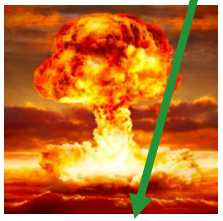




Простой и удобный способ ловить ошибки

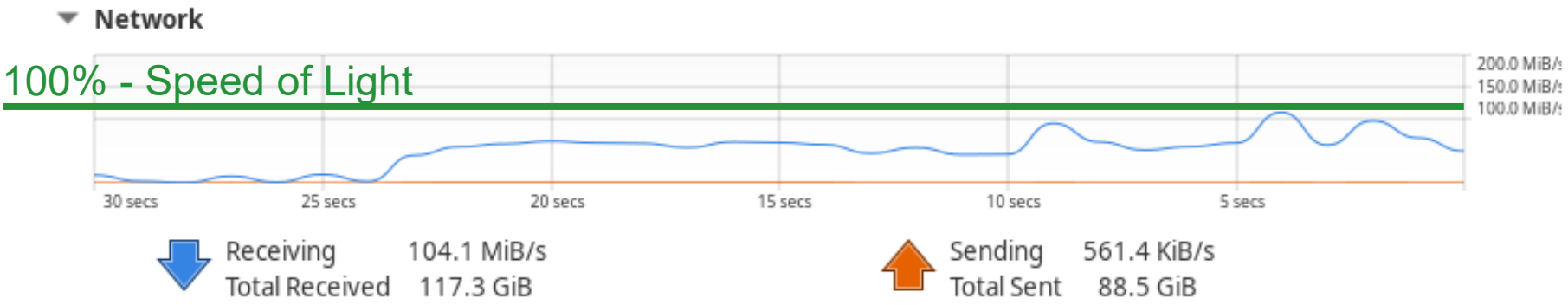
```
3190    for (size_t i = 1; i < group_unqiue_cubes.size(); ++i) {
3191        rassert(cmp(a: group_unqiue_cubes[i - 1], b: group_unqiue_cubes[i]), 239101019);
3192    }
```

```
4176    ListFileReader<tgv::CubeWithRadius> reader(linear_octree_cubes_path, params.ooc_compression_level);
4177    for (uint64_t i = from; i < to; ++i) {
4178        tgv::CubeWithRadius next = reader.next();
4179        rassert(next.radius_votes >= 1, 239101376);
4180        cubes[i - from] = next;
4181        if (i > from) {
4182            rassert(less(a: cubes[i - from - 1], b: cubes[i - from]), 239101382);
4183        }
4184    }
```

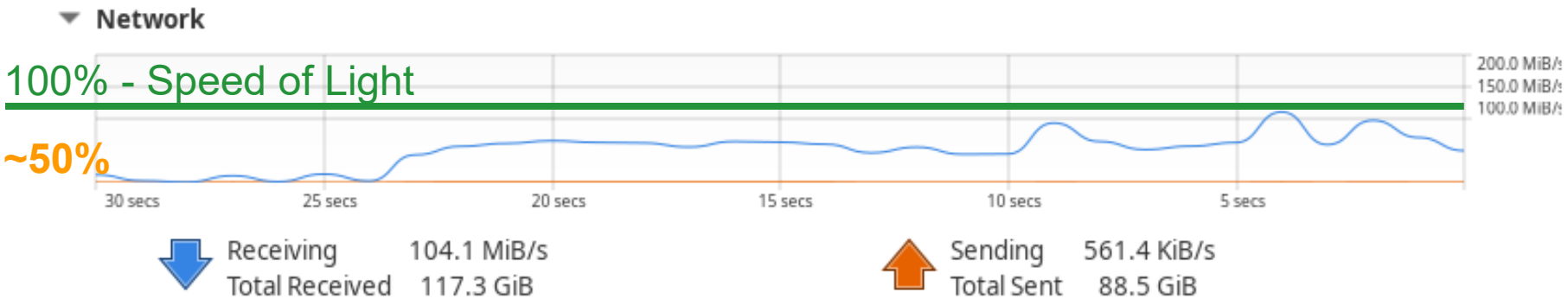


- Моментальная диагностика по коду (вместо строки + сверки ревизий)
- Накопление статистики частотности репортов
- Чем плотнее усеян код - тем ближе к первопричине (fail fast policy)
- Полезно добавлять контекст
- Может быть удобно проверять **rassert**-ы даже на **GPU** (в кернелах)

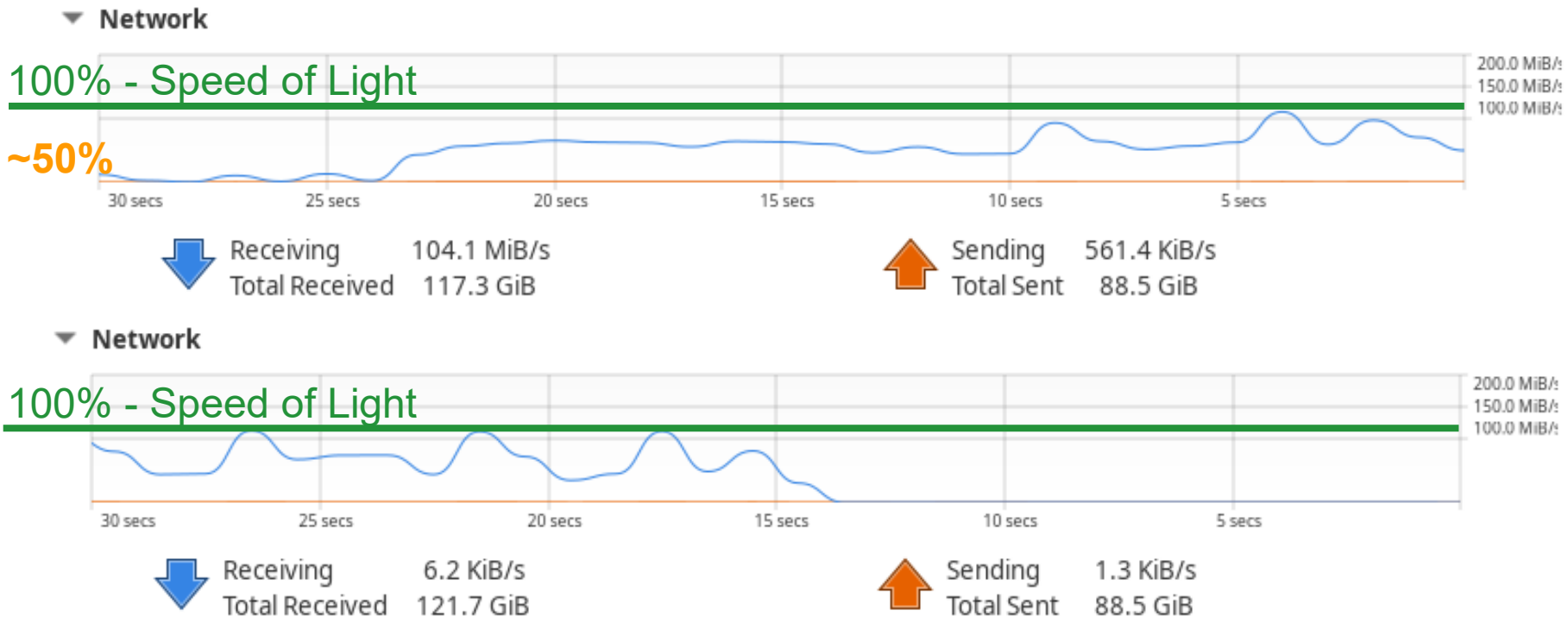
Speed of Light профилирование **(SoL)**



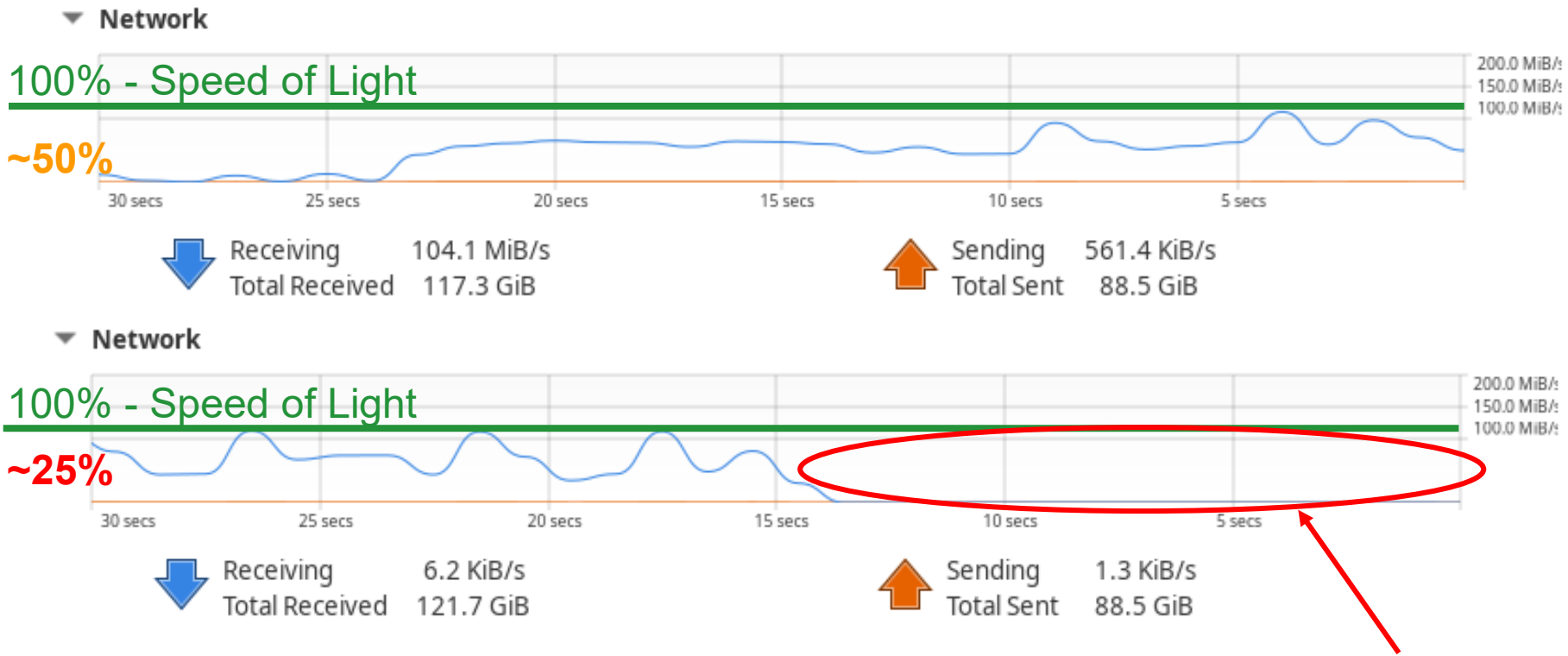
Speed of Light профилирование **(SoL)**



Speed of Light профилирование **(SoL)**

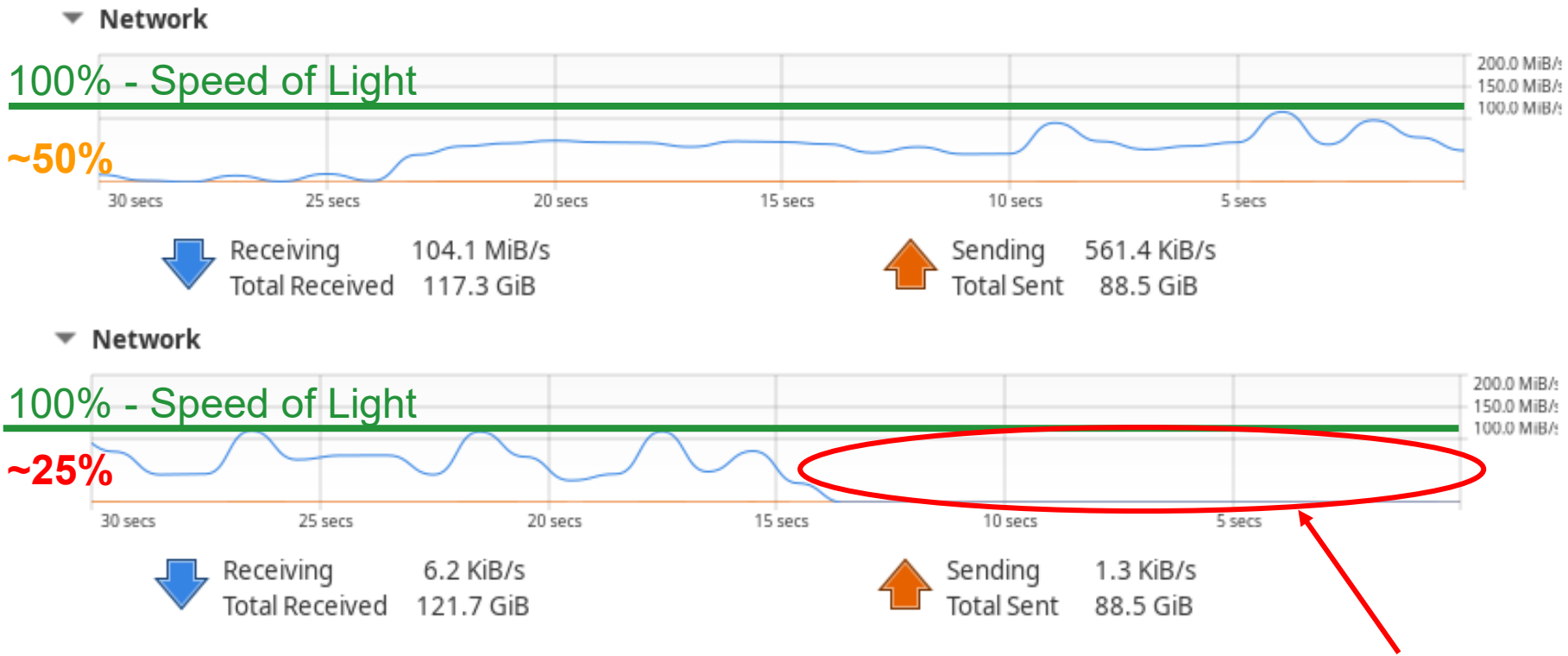


Speed of Light профилирование **(SoL)**



Аналогично если PCI-E шина - узкое место, очень хорошо и для LLM агента!

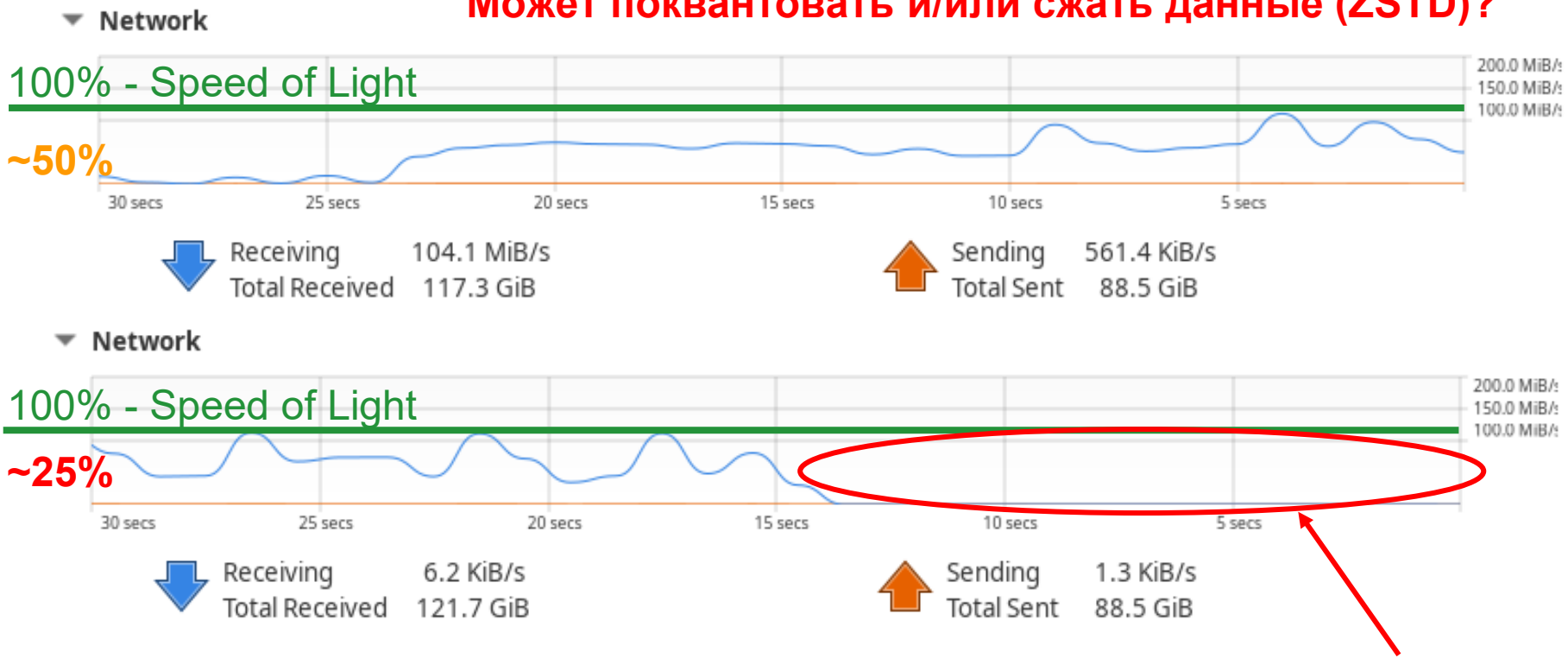
Speed of Light профилирование **(SoL)**



Аналогично если PCI-E шина - узкое место, очень хорошо и для LLM агента!

Speed of Light профилирование **(SoL)**

Может поквантовать и/или сжать данные (ZSTD)?



Встроенное самопрофилирование

```
pass #1, group #1/1: merging 5 files...
total input: 82465213 cubes, 1258.32 MB...
total output: 81823977 cubes, 1248.53 MB, 99%, 277.417 compressed MB - i.e. 22% compression
done in 3.62467 s, bandwidth: 4.53827e+07 cubes/s, 692.485 MB/s
```

Не тормозит ли IO?
Не стало ли CPU bound из-за сжатия?

Встроенное самопрофилирование

```
pass #1, group #1/1: merging 5 files...
total input: 82465213 cubes, 1258.32 MB...
total output: 81823977 cubes, 1248.53 MB, 99%, 277.417 compressed MB - i.e. 22% compression
done in 3.62467 s, bandwidth: 4.53827e+07 cubes/s, 692.485 MB/s
```

```
initing hists from depth maps or structured laser scans
Data: 2184.94 MB hists + 0 MB codes, 32.3348 MB pyramid
Found 1 GPUs in 0.006167 sec (CUDA: 0.004327 sec, OpenCL: 0.001332 sec, Vulkan: 0.000489 sec)
[GPU 1] Using device: NVIDIA GeForce RTX 5070 Ti, 70 compute units, free memory: 14078/15833 MB, compute capability 12.0
  driver/runtime CUDA: 13010/10010
  max work group size 1024
  max work item sizes [1024, 1024, 64]
Devices performance:
 - 100%      done by [GPU 1]  with timings: 10 s = 1% load hists + 30% load cams + 65% calcs
```

PCI-E **IO** **GPU compute bound**

Встроенное самопрофилирование

```
pass #1, group #1/1: merging 5 files...
total input: 82465213 cubes, 1258.32 MB...
total output: 81823977 cubes, 1248.53 MB, 99%, 277.417 compressed MB - i.e. 22% compression
done in 3.62467 s, bandwidth: 4.53827e+07 cubes/s, 692.485 MB/s
```

```
initing hists from depth maps or structured laser scans
Data: 2184.94 MB hists + 0 MB codes, 32.3348 MB pyramid
Found 1 GPUs in 0.006167 sec (CUDA: 0.004327 sec, OpenCL: 0.001332 sec, Vulkan: 0.000489 sec)
[GPU 1] Using device: NVIDIA GeForce RTX 5070 Ti, 70 compute units, free memory: 14078/15833 MB, compute capability 12.0
  driver/runtime CUDA: 13010/10010
  max work group size 1024
  max work item sizes [1024, 1024, 64]
Devices performance:
 - 100%      done by [GPU 1]  with timings: 10 s = 1% load hists + 30% load cams + 65% calcs
```

```
Level #14/14: part #1/1
Loading cubes...
    81307641 cubes, 2171.15 MB, 1 nodes, 100% from inner cubes, 100% from neighbors neighboring voxels ratio 100%
    71 blocks in 1.77409 s, 106.949 MB avg size, 559.062 MB max size, 3/71 big blocks, 119% neighbors
Found 1 GPUs in 0.002112 sec (CUDA: 0.001096 sec, OpenCL: 0.000458 sec, Vulkan: 0.00052 sec)
[GPU 1] Using device: NVIDIA GeForce RTX 5070 Ti, 70 compute units, free memory: 14011/15833 MB, compute capability 12.0
  driver/runtime CUDA: 13010/10010
  max work group size 1024
  max work item sizes [1024, 1024, 64]
Devices performance:
 - 100%      done by [GPU 1]  with timings: 15 s = 0% waiting + 0% fetching + 20% enumerating cubes + 3% bs + 7% preparing cubes buffers + 42% calcs + 14% saving
```

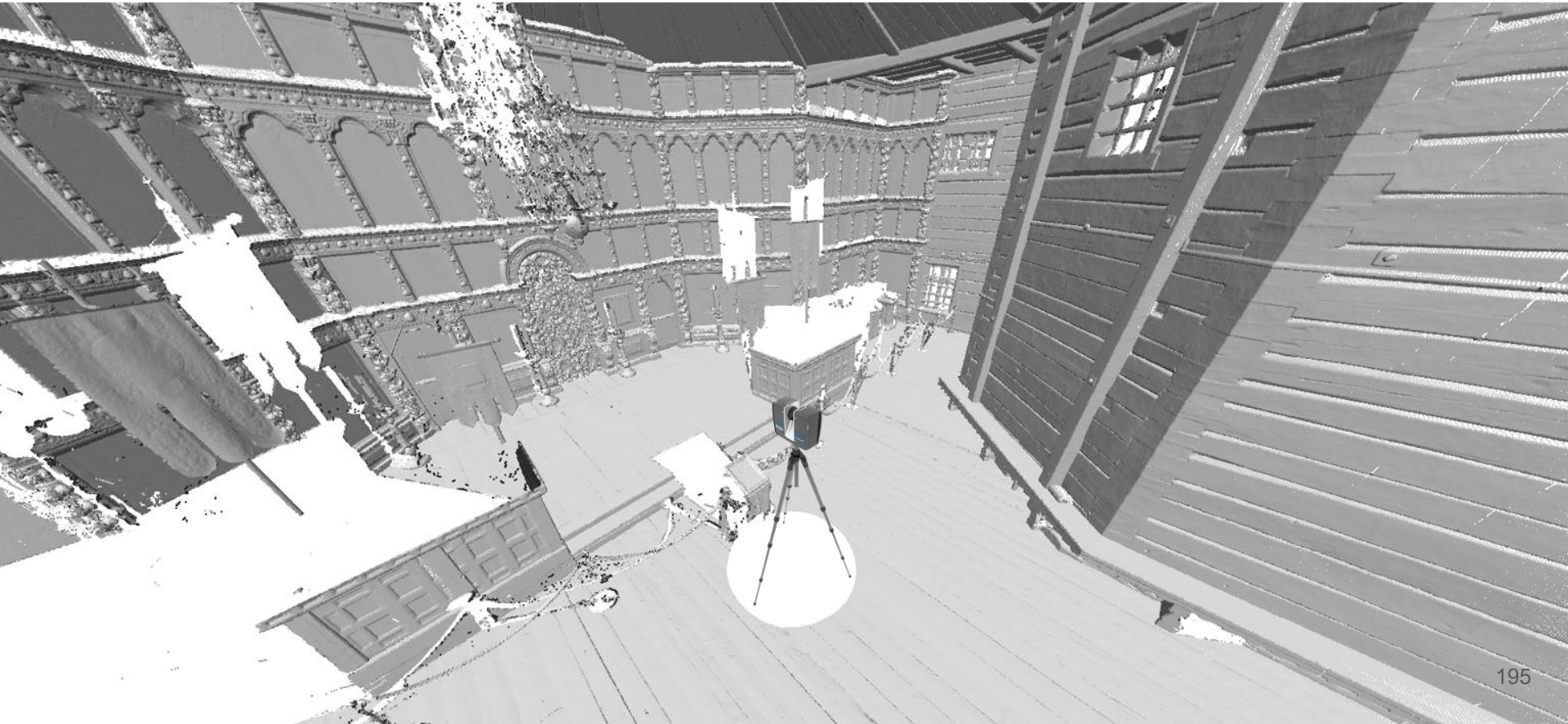
Встроенное самопрофилирование

```
total time: 717.338 sec
  - 45% - 323.608 sec - 15 x estimateWinnersPart, 10.64 GB peak
  - 14% - 100.519 sec - 15 x estimateQualityPart, 474.53 MB peak
  - 12% - 85.277 sec - 8 x prepareImagesPart, 3.42 GB peak
  - 12% - 84.191 sec - 15 x enblendPart, 1.19 GB peak
  - 6% - 45.157 sec - 8 x estimateWeightsPyrPart, 1.05 GB peak
  - 6% - 43.796 sec - 15 x estimateWinnersPackAtlasPart, 884.88 MB peak
peak memory used: 10.64 GB in subtask estimateWinnersPart
```

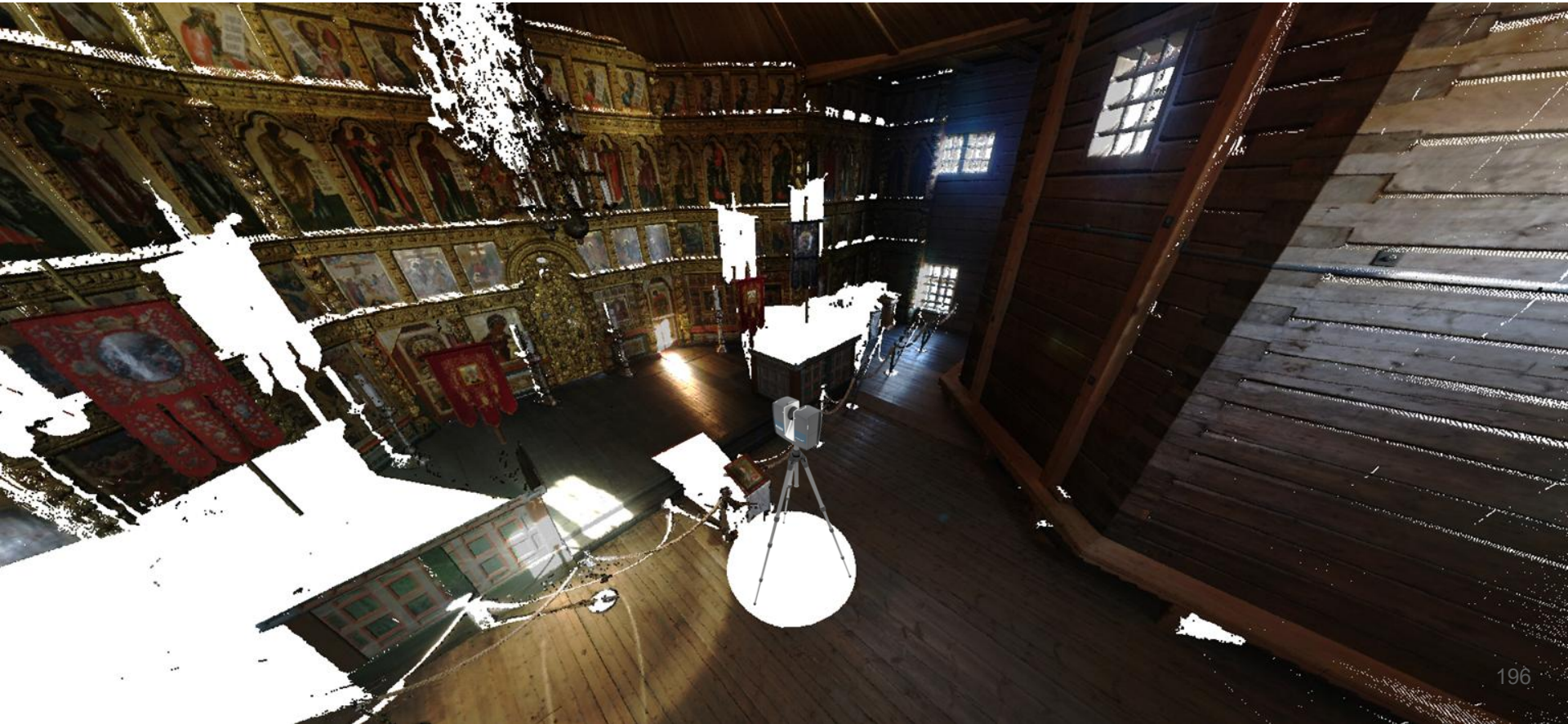
```
Level #14/14: part #1/1
Loading cubes...
    81307641 cubes, 2171.15 MB, 1 nodes, 100% from inner cubes, 100% from neighbors neighboring voxels ratio 100%
    71 blocks in 1.77409 s, 106.949 MB avg size, 559.062 MB max size, 3/71 big blocks, 119% neighbors
Found 1 GPUs in 0.002112 sec (CUDA: 0.001096 sec, OpenCL: 0.000458 sec, Vulkan: 0.00052 sec)
[GPU 1] Using device: NVIDIA GeForce RTX 5070 Ti, 70 compute units, free memory: 14011/15833 MB, compute capability 12.0
  driver/runtime CUDA: 13010/10010
  max work group size 1024
  max work item sizes [1024, 1024, 64]
Devices performance:
 - 100%      done by [GPU 1]  with timings: 15 s = 0% waiting + 0% fetching + 20% enumerating cubes + 3% bs + 7% preparing cubes buffers + 42% calcs + 14% saving
```

Terrestrial LIDAR

Terrestrial LIDAR

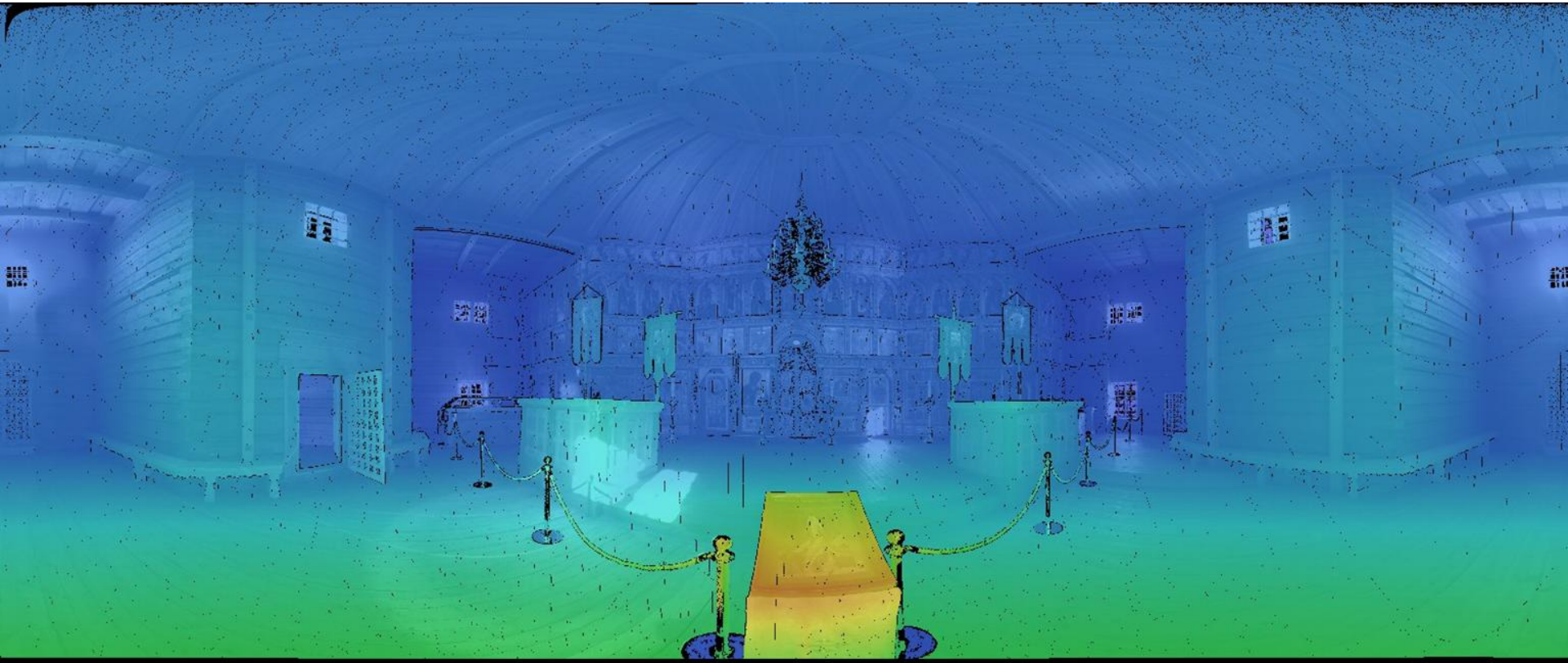


Terrestrial LIDAR



Terrestrial LIDAR = карта глубины сферической камеры

Terrestrial LIDAR = карта глубины сферической камеры



Ссылки

A Globally Optimal Algorithm for Robust TV-L1 Range Image Integration, **Zach et al., 2007**
Global, Dense Multiscale Reconstruction for a Billion Points, **Ummenhofer et al., 2017**
Out-of-Core Surface Reconstruction via Global TGV Minimization, **Poliarnyi, 2021**
Simplifying surfaces with color and texture using quadric error metrics, **Garland et. al., 1998**

Ссылки

A Globally Optimal Algorithm for Robust TV-L1 Range Image Integration, **Zach et al., 2007**
Global, Dense Multiscale Reconstruction for a Billion Points, **Ummenhofer et al., 2017**
Out-of-Core Surface Reconstruction via Global TGV Minimization, **Poliarnyi, 2021**
Simplifying surfaces with color and texture using quadric error metrics, **Garland et. al., 1998**

Лекции по **вычислениям на видеокартах (OpenCL, CUDA, Vulkan):**
- https://github.com/GPGPUCourse/
- youtube(“курс вычисления на видеокартах”)

Лекция про UE5: Nanite

Ссылки

A Globally Optimal Algorithm for Robust TV-L1 Range Image Integration, **Zach et al., 2007**
Global, Dense Multiscale Reconstruction for a Billion Points, **Ummenhofer et al., 2017**
Out-of-Core Surface Reconstruction via Global TGV Minimization, **Poliarnyi, 2021**
Simplifying surfaces with color and texture using quadric error metrics, **Garland et. al., 1998**

Лекции по **вычислениям на видеокартах (OpenCL, CUDA, Vulkan):**
- https://github.com/GPGPUCourse/
- youtube(“курс вычисления на видеокартах”)

Лекции по **фотограмметрии:**
- https://github.com/photogrammetryCourse/
- youtube(“курс по фотограмметрии”)

Лекция про UE5: Nanite

Ссылки

A Globally Optimal Algorithm for Robust TV-L1 Range Image Integration, **Zach et al., 2007**
Global, Dense Multiscale Reconstruction for a Billion Points, **Ummenhofer et al., 2017**
Out-of-Core Surface Reconstruction via Global TGV Minimization, **Poliarnyi, 2021**
Simplifying surfaces with color and texture using quadric error metrics, **Garland et. al., 1998**

Лекции по **вычислениям на видеокартах (OpenCL, CUDA, Vulkan):**
- https://github.com/GPGPUCourse/
- youtube(“курс вычисления на видеокартах”)

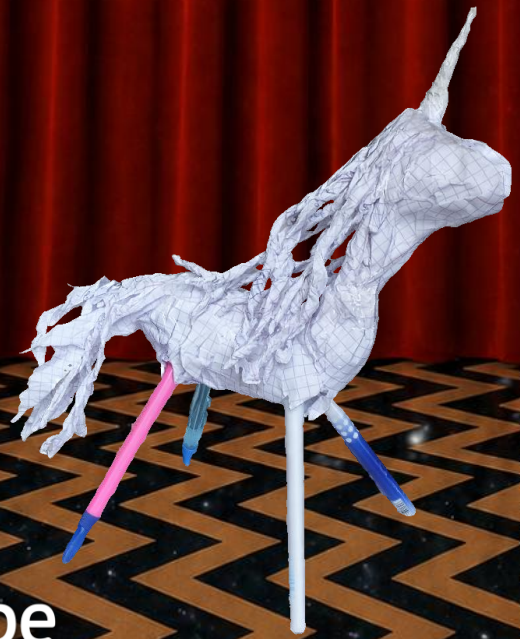
Лекции по **фотограмметрии:**
- https://github.com/photogrammetryCourse/
- youtube(“курс по фотограмметрии”)

Курсы и мероприятия **CSSpace:**
- https://t.me/csspace

Лекция про UE5: Nanite

Спасибо за внимание!

Вопросы?



@UnicornGlade
@PolarNick239
polarhare@gmail.com
Николай Полярный

Agisoft Metashape

C++ Russia 2026