

Введение в фотограмметрию

3D модель из карт глубины (Out-of-Core)

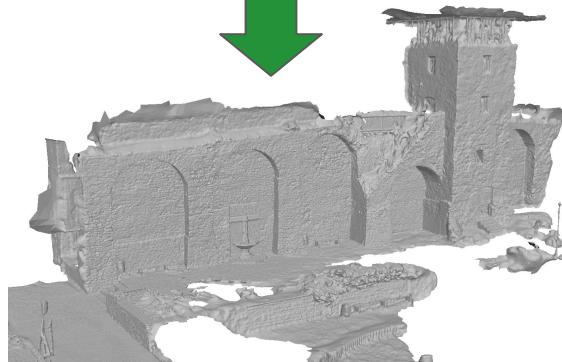
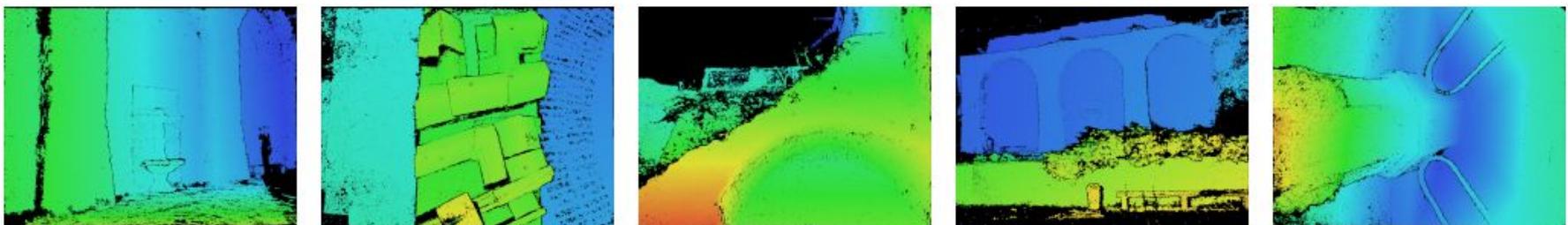
Фотограмметрия. Лекция 19

- Out-of-Core K-way merge sort
- QSLim упрощение модели - edge collapse
- Out-of-Core метод реконструкции



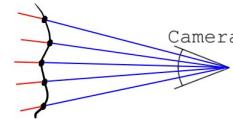
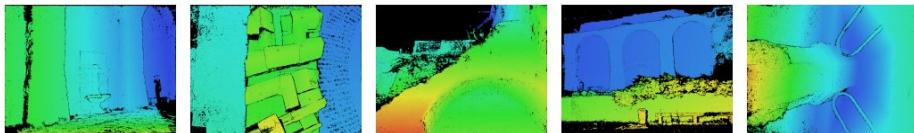
Полярный Николай
polarnick239@gmail.com

Входные данные: карты глубины



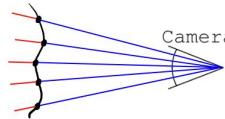
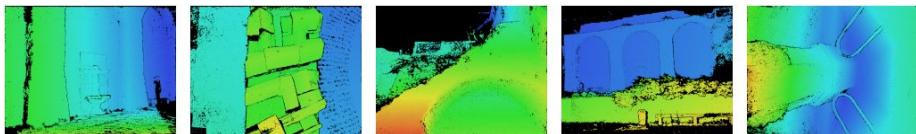
3D модель (TV-L1 / TV-Huber / TGV)

1) Есть множество карт глубины



3D модель (TV-L1 / TV-Huber / TGV)

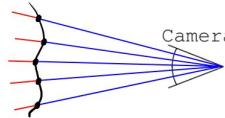
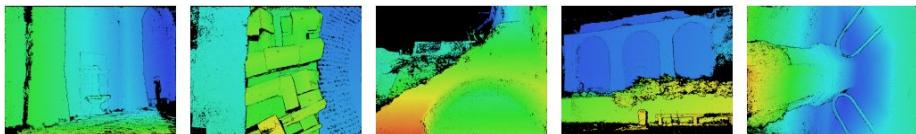
- 1) Есть множество карт глубины



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

3D модель (TV-L1 / TV-Huber / TGV)

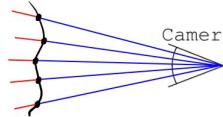
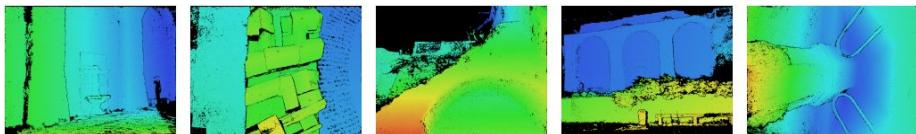
- 1) Есть множество карт глубины

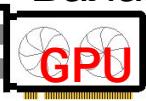


- 2) Множество точек всех карт глубины - порождает **адаптивное** октодерево (дискретизация пространства)
- 3) Балансируем октодерево 2:1 (по каждой стороне макс. два соседа)

3D модель (TV-L1 / TV-Huber / TGV)

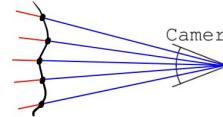
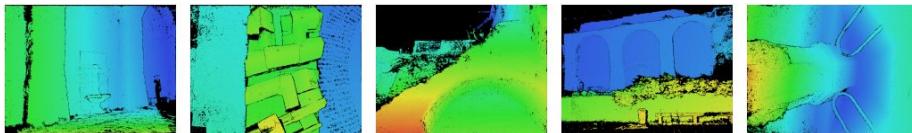
- 1) Есть множество карт глубины



- 2) Множество точек всех карт глубины - порождает **адаптивное** октодерево (дискретизация пространства)
- 3) Балансируем октодерево 2:1 (по каждой стороне макс. два соседа)
- 4)  Каждая карта глубины вносит индикаторные f_i голоса за voxели

3D модель (TV-L1 / TV-Huber / TGV)

- 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) GPU Каждая карта глубины вносит индикаторные f_i голоса за voxели
- 5) Оптимизировали индикаторное поле (**много итераций + Coarse-to-Fine**)

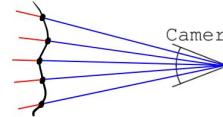
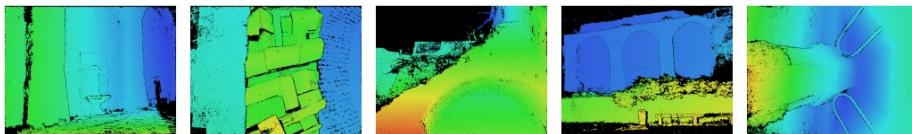


TV- L^1 :
$$E = \int_{\Omega} \left\{ |\nabla u| + \lambda \sum_{i \in \mathcal{I}(\vec{x})} w_i(\vec{x}) |u - f_i| \right\} d\vec{x}$$

The equation represents the TV- L^1 energy functional. It consists of two terms: a total variation term $|\nabla u|$ and a data-fitting term. The data-fitting term is weighted by indicator functions $w_i(\vec{x})$ and absolute differences $|u - f_i|$. The indicator functions are highlighted with red and blue circles.

3D модель (TV-L1 / TV-Huber / TGV)

- 1) Есть множество карт глубины



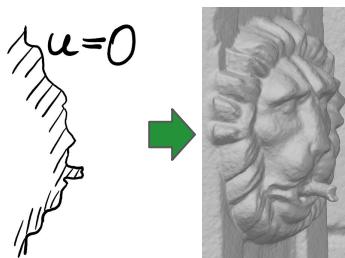
- 2) Множество точек всех карт глубины - порождает **адаптивное** октодерево (дискретизация пространства)
- 3) Балансируем октодерево 2:1 (по каждой стороне макс. два соседа)
- 4) GPU Каждая карта глубины вносит индикаторные f_i голоса за voxели
- 5) Оптимизировали индикаторное поле (**много итераций + Coarse-to-Fine**)

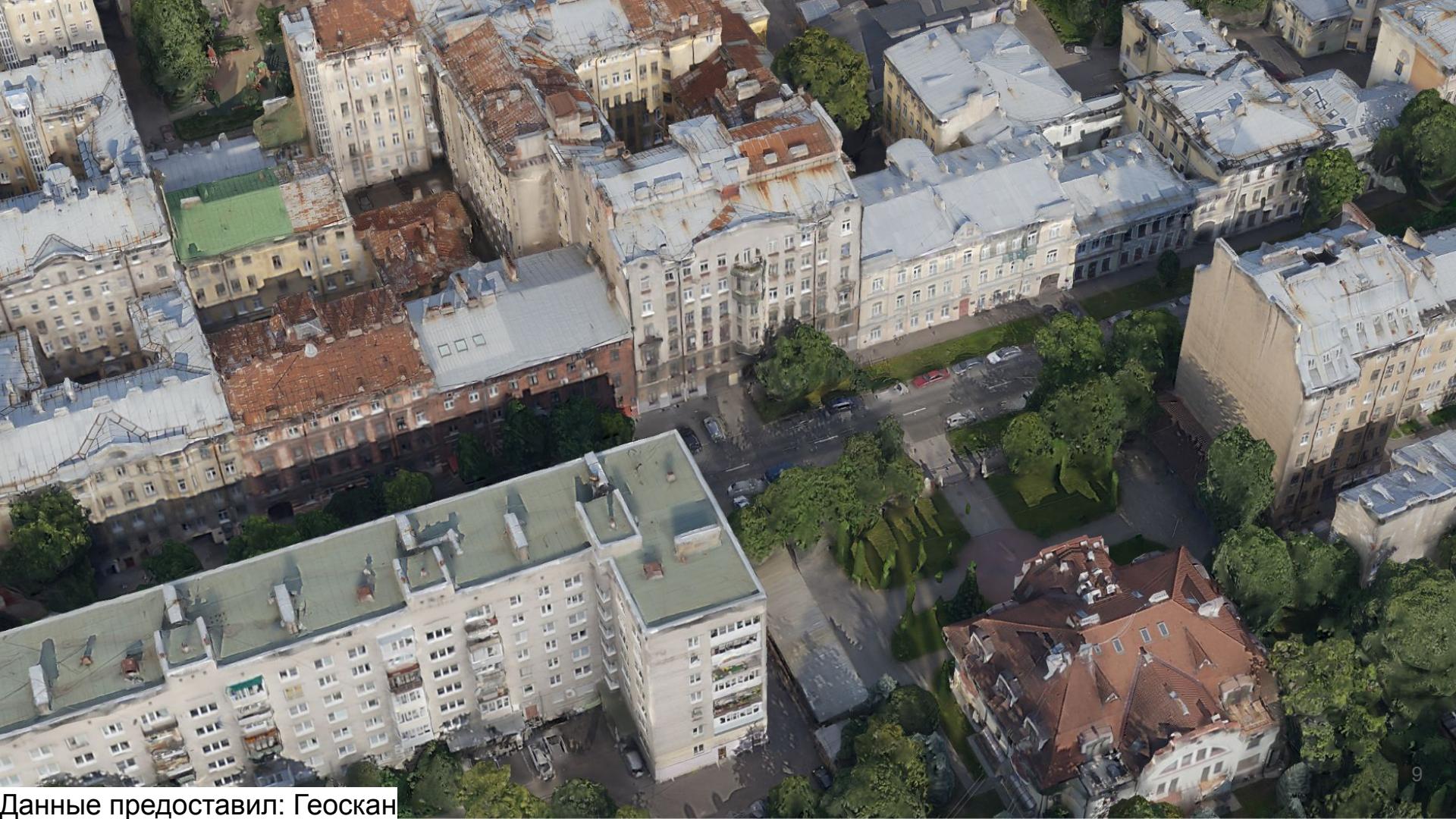
GPU

TV- L^1 :

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

- 6) Извлекли поверхность маркировкой кубов



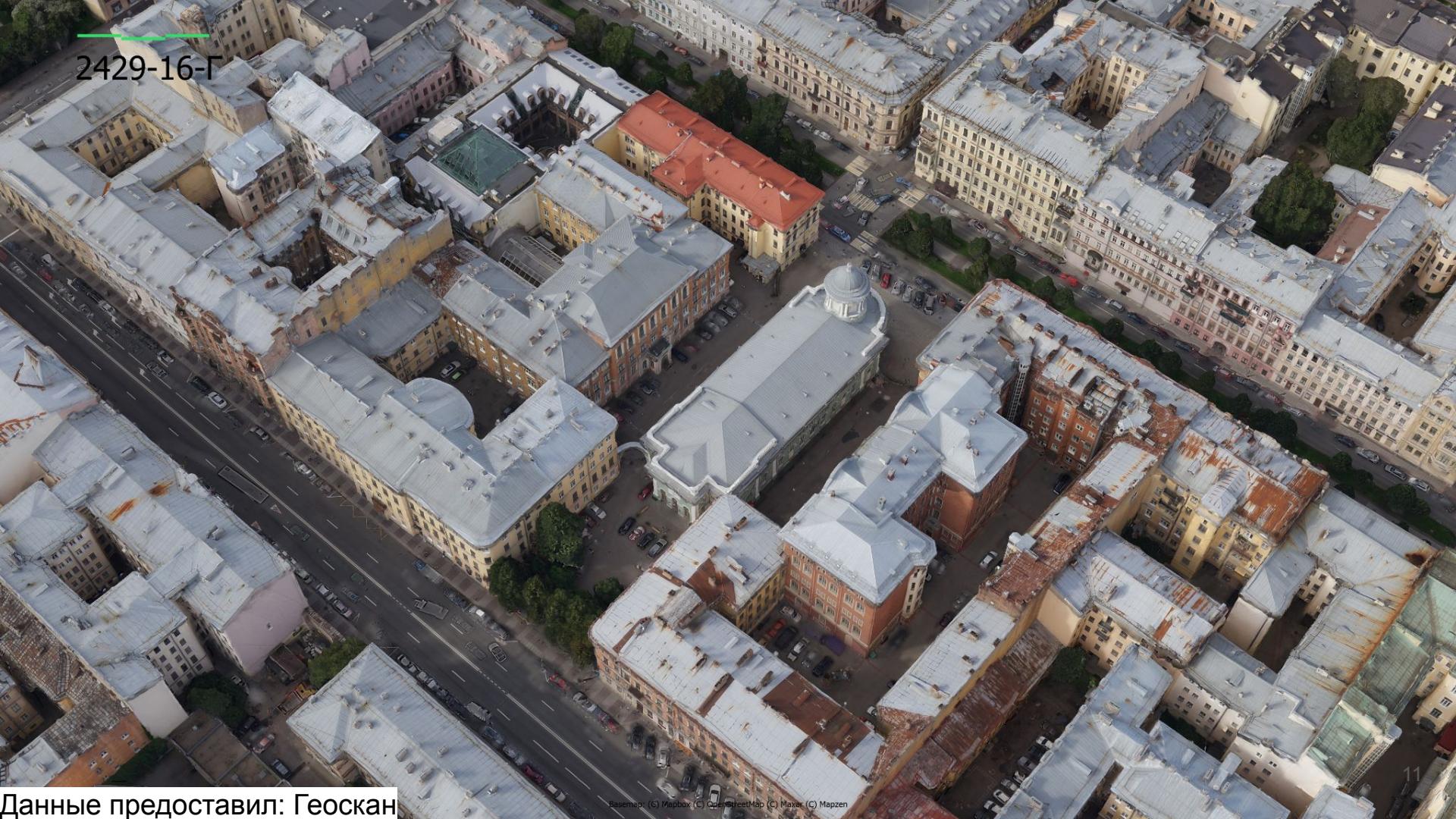


Данные предоставил: Геоскан



Данные предоставил: Геоскан

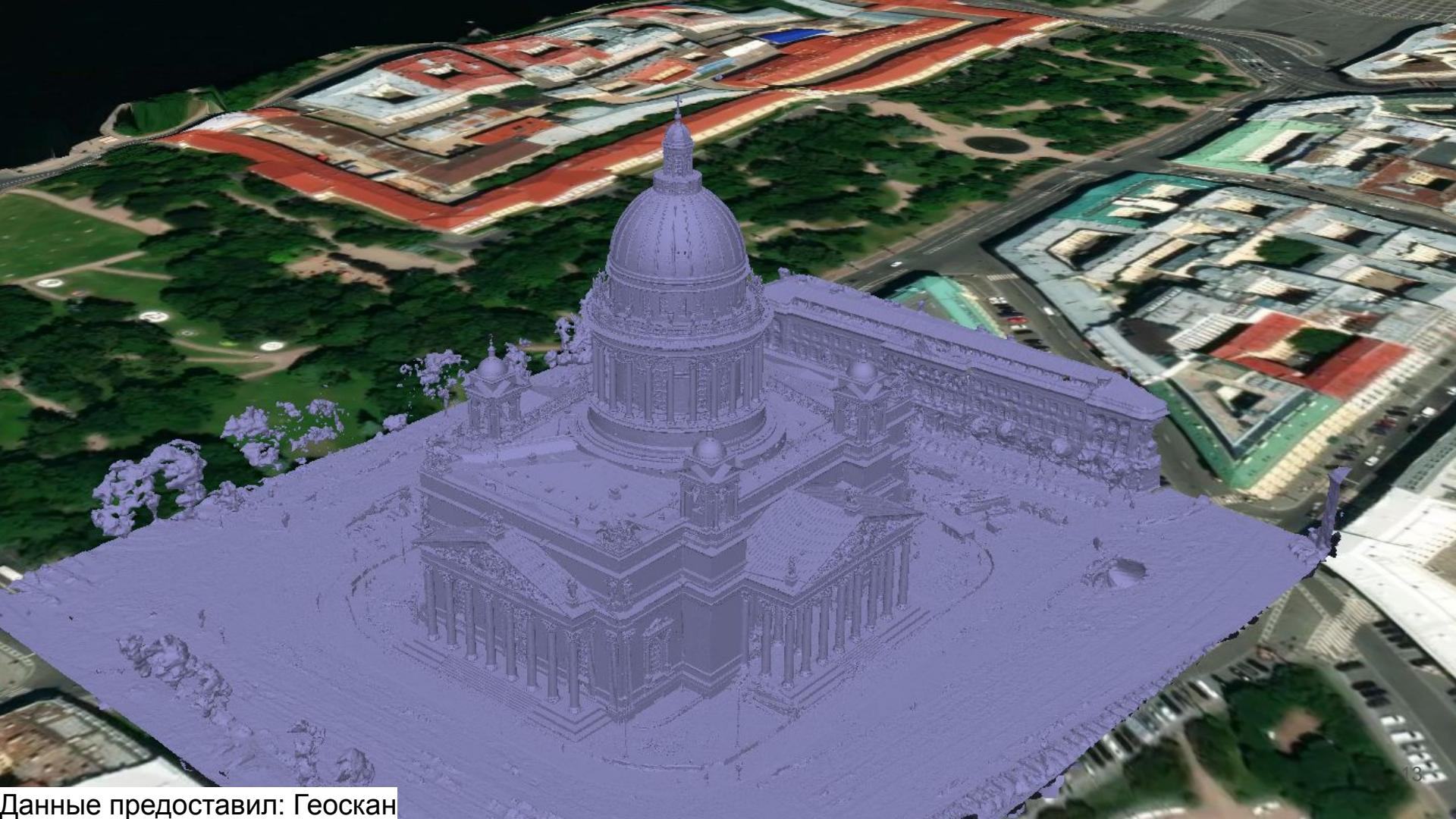
2429-16-Г



Данные предоставил: Геоскан



Данные предоставил: Геоскан



Данные предоставил: Геоскан



Данные предоставил: Геоскан



ИА ТА ГОСПОДИ УПОВАХОМЪ ДЛЯ ПОСТАДИСА ВО ЕЗЫКІ.



Данные предоставил: Геоскан



3D модель Кижского погоста (Agisoft)



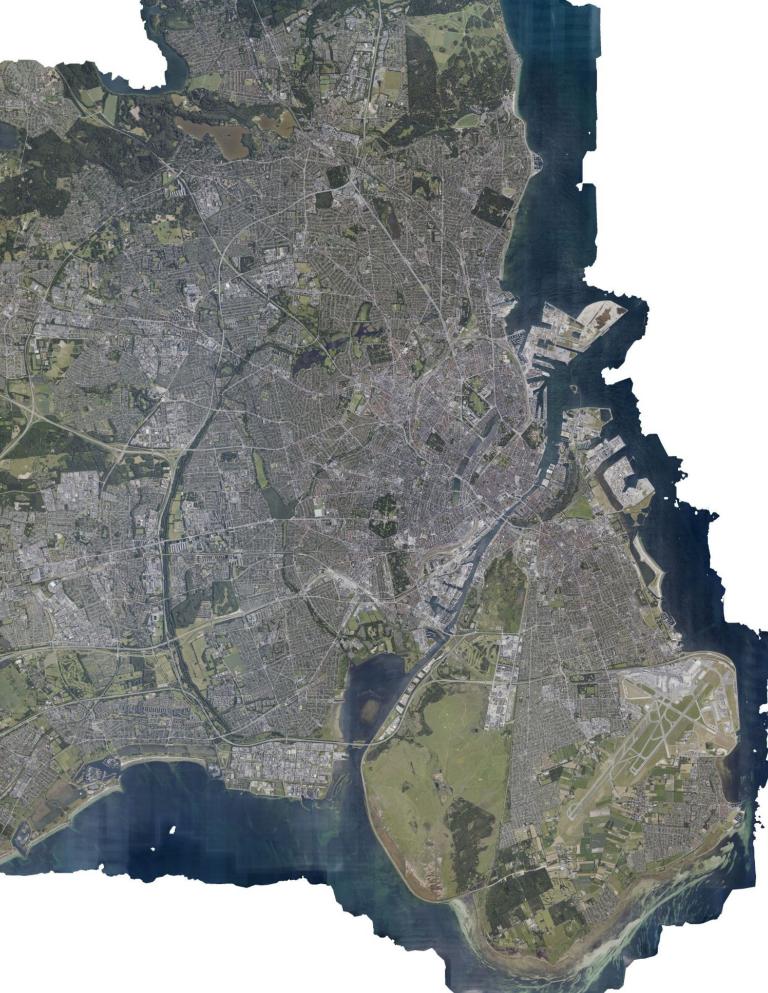
3D модель Кижского погоста (Agisoft)



3D модель Кижского погоста (Agisoft)

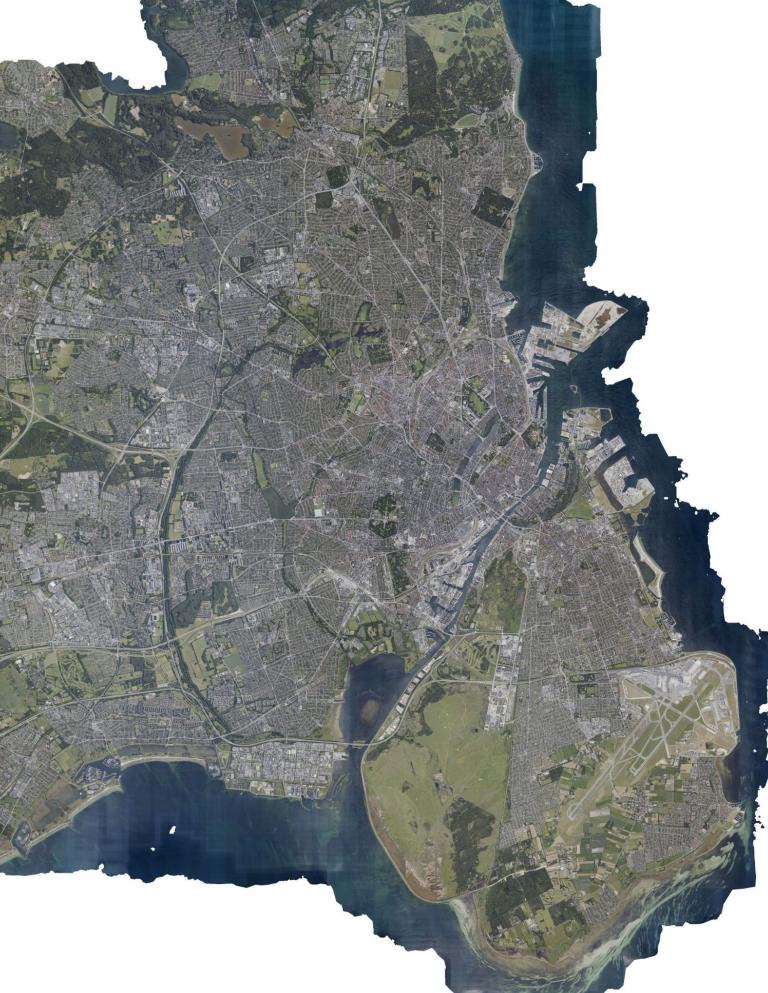


3D модель Кижского погоста (Agisoft)



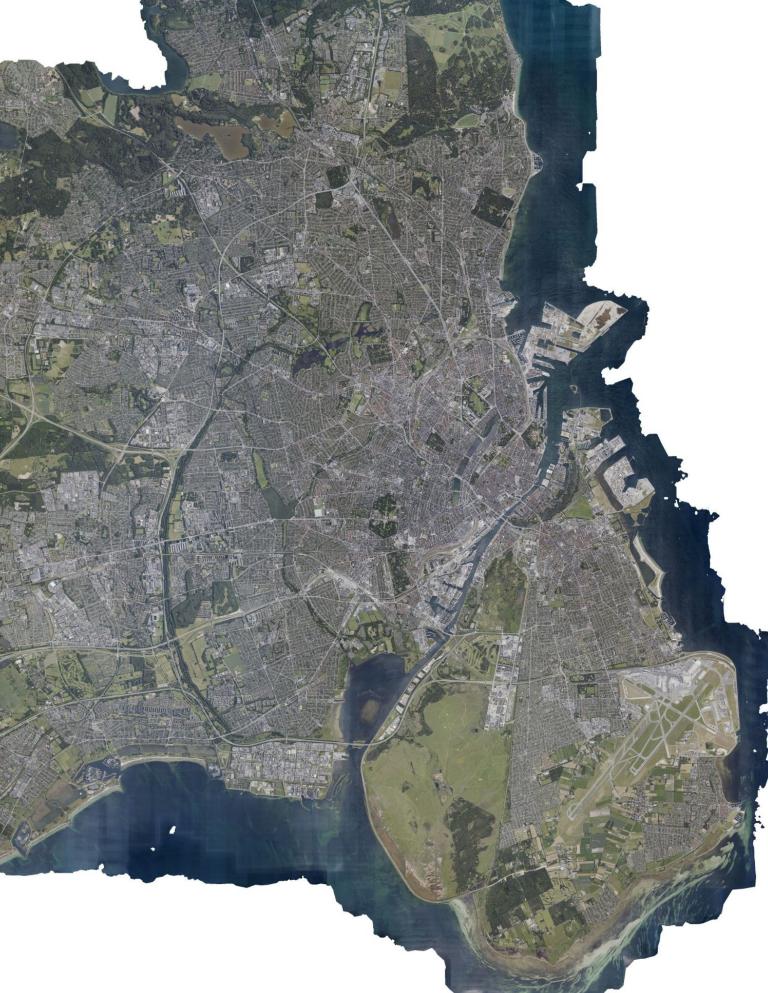
Building Copenhagen in a Day*

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



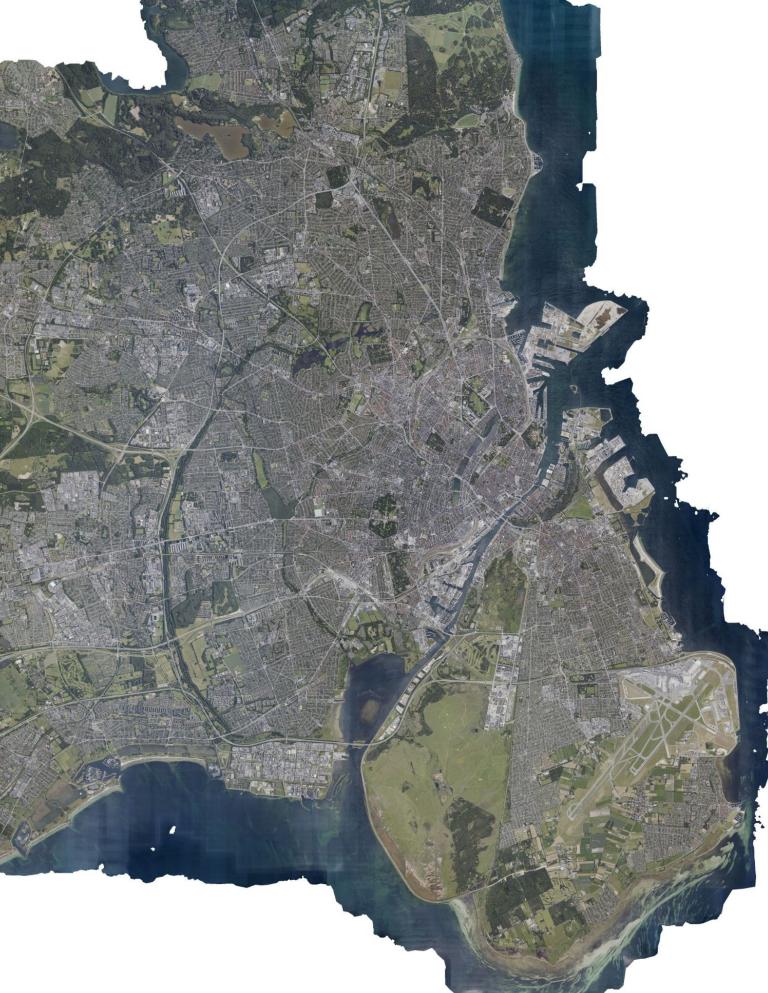
Building Copenhagen in a Day*

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



Building Copenhagen in a Day*

- 425 km²
- 27472 фотографии (566 GB, jpeg)
- 28 миллиардов “точек”
- 7.5 миллиардов треугольников
- **Пиковая RAM: 14 GB**



Building Copenhagen in a Day*

- 425 km²
- 27472 фотографии (566 GB, jpeg)
- 28 миллиардов “точек”
- 7.5 миллиардов треугольников
- **Пиковая RAM: 14 GB**
- *на кластере за 29 часов
(7 компьютеров по 8 ядер + GTX 1080)

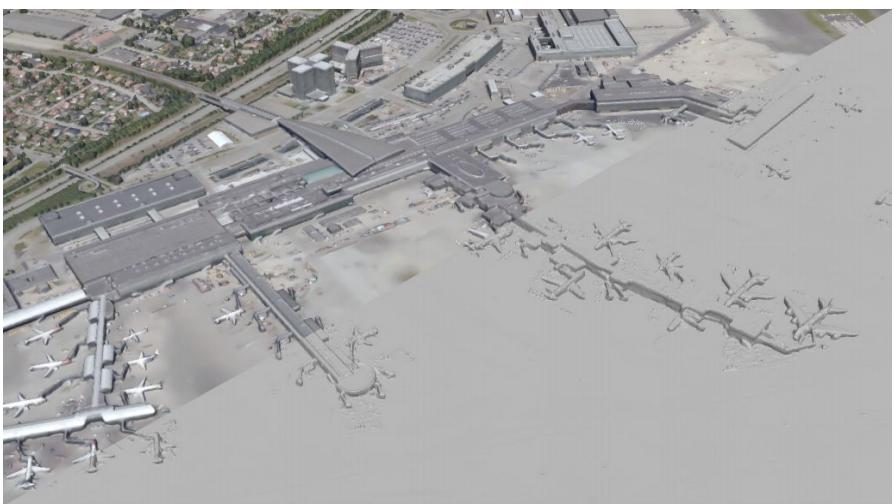


Figure 7. Airport closeup.

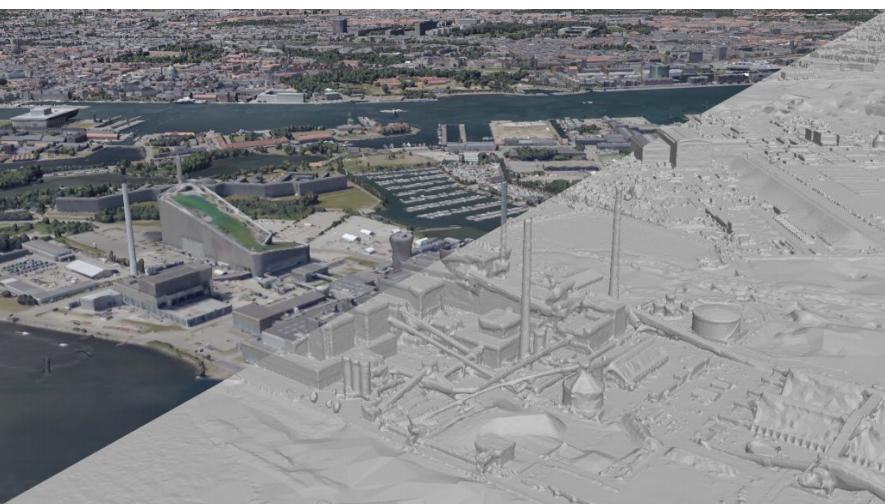


Figure 8. Hovor Amagerverket closeup.



Figure 9. Christiansborg Palace closeup.



Figure 10. Kastellet closeup.

Мечта

На компьютере с 16 GB RAM реконструировать объект любой сложности.

- **Out-of-Core** - в любой момент в памяти ограниченный объем данных

Dataset name	Input data	Initial cubes	Faces after marching cubes	Peak RAM (GB)	Processing time
Citywall [10]	564 depth maps	1205 mil	135 mil	13.17	63 min
Breisach [27]	2111 depth maps	2642 mil	558 mil	10.07	260 min
Tomb of Tu Duc (LIDAR) [6]	42 LIDAR scans	661 mil	672 mil	10.05	160 min
Palacio Tschudi [7]	13703 depth maps	16 billion	3159 mil	16.75	1213 min
Copenhagen city [8]	27472 depth maps	28 billion	7490 mil	13.35	1758 min

Мечта

На компьютере с 16 GB RAM реконструировать объект любой сложности.

- **Out-of-Core** - в любой момент в памяти ограниченный объем данных
- Преобразование карт глубины в гистограммы (голосование) на 

17%

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

Table 1. Runtime breakdown for the Breisach data set.

Мечта

На компьютере с 16 GB RAM реконструировать объект любой сложности.

- **Out-of-Core** - в любой момент в памяти ограниченный объем данных
- Преобразование карт глубины в гистограммы (голосование) на 
- Минимизация численным методом на 

17%

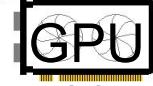
80%

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

Table 1. Runtime breakdown for the Breisach data set.

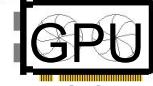
Мечта

На компьютере с 16 GB RAM реконструировать объект любой сложности.

- **Out-of-Core** - в любой момент в памяти ограниченный объем данных
- Преобразование карт глубины в гистограммы (голосование) на 
- Минимизация численным методом на 
- Фильтрация шумов на базе лучей видимости (просто точки - не подходят)

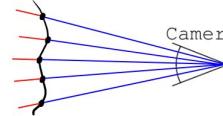
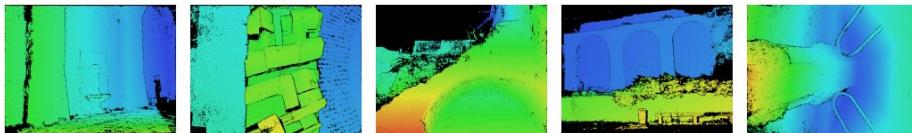
Мечта

На компьютере с 16 GB RAM реконструировать объект любой сложности.

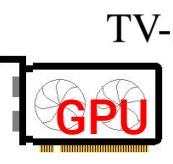
- **Out-of-Core** - в любой момент в памяти ограниченный объем данных
- Преобразование карт глубины в гистограммы (голосование) на 
- Минимизация численным методом на 
- Фильтрация шумов на базе лучей видимости (просто точки - не подходят)
- Возможность ускорять за счет вычисления на кластере

Out-of-Core реконструкция 3D модели

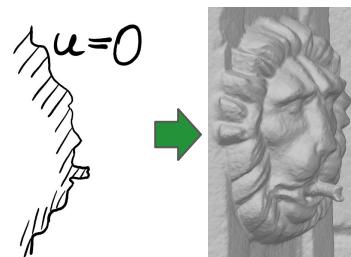
- 1) Есть множество карт глубины



- 2) Множество точек всех карт глубины - порождает **адаптивное** октодерево (дискретизация пространства)
- 3) Балансируем октодерево 2:1 (по каждой стороне макс. два соседа)
- 4) GPU Каждая карта глубины вносит индикаторные f_i голоса за voxели
- 5) Оптимизировали индикаторное поле (**много итераций + Coarse-to-Fine**)


$$TV-L^1: E = \int_{\Omega} \left\{ |\nabla u| + \lambda \sum_{i \in \mathcal{I}(\vec{x})} w_i(\vec{x}) |u - f_i| \right\} d\vec{x}$$

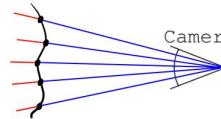
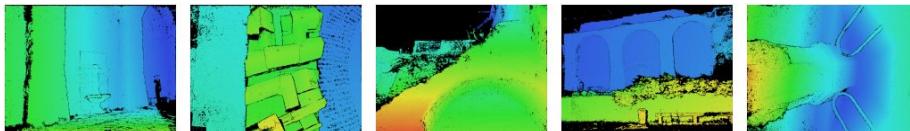
- 6) Извлекли поверхность маршировкой кубов



Out-of-Core реконструкция 3D модели

Можем ли мы просто
обработать пространство
по кусочкам?

- 1) Есть множество карт глубины

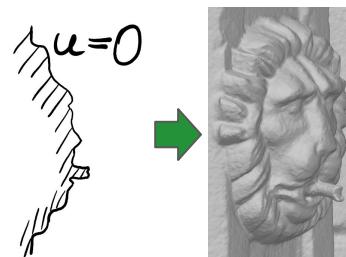


- 2) Множество точек всех карт глубины - порождает **адаптивное октодерево** (дискретизация пространства)
- 3) Балансируем октодерево 2:1 (по каждой стороне макс. два соседа)
- 4) GPU Каждая карта глубины вносит индикаторные f_i голоса за воксели
- 5) Оптимизировали индикаторное поле (**много итераций + Coarse-to-Fine**)

GPU

$$TV-L^1: E = \int_{\Omega} \left\{ |\nabla u| + \lambda \sum_{i \in \mathcal{I}(\vec{x})} w_i(\vec{x}) |u - f_i| \right\} d\vec{x}$$

- 6) Извлекли поверхность маршировкой кубов



Out-of-Core реконструкция 3D модели

Можем ли мы просто
обработать пространство
по кусочкам?

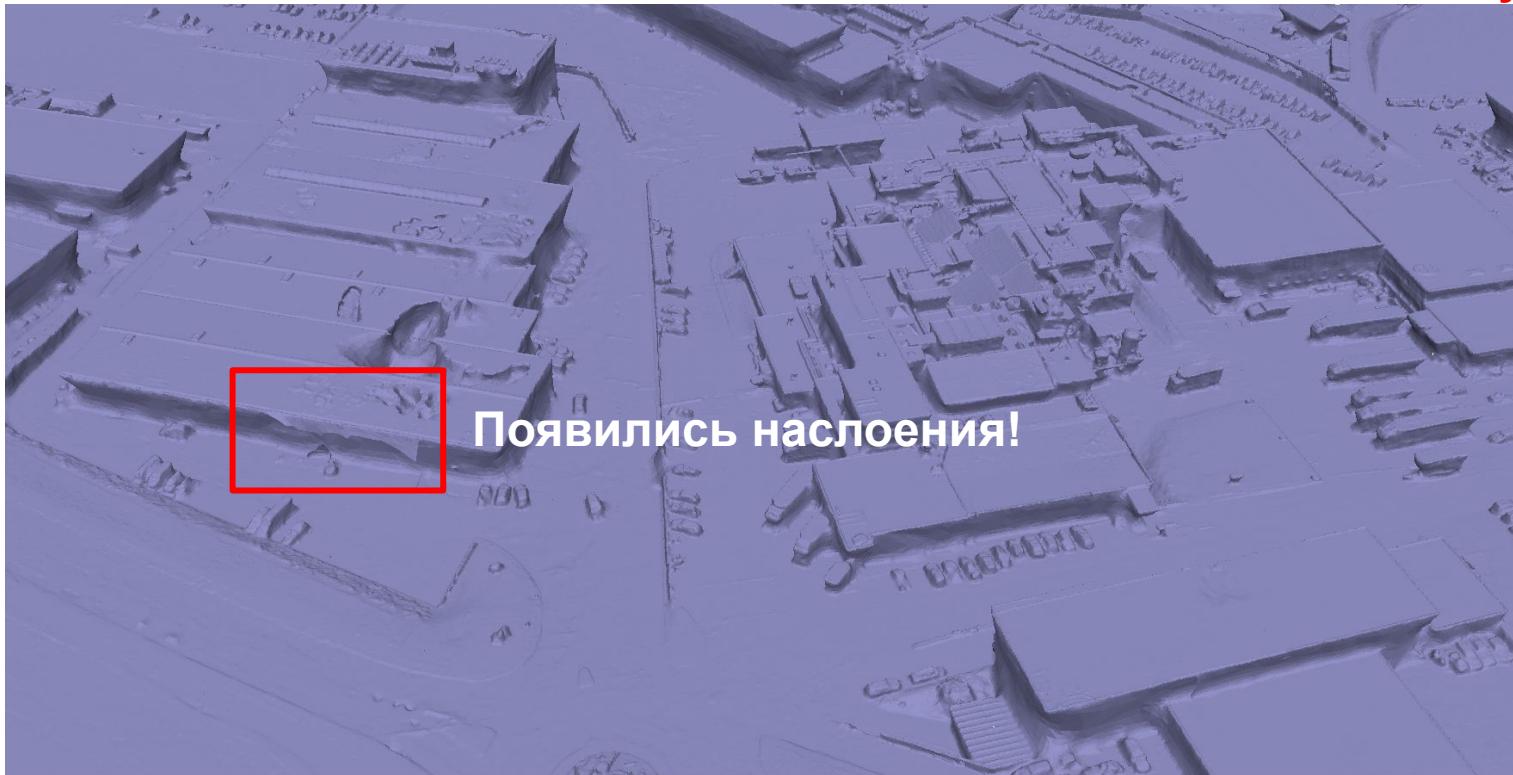
Обработка по кускам:



Out-of-Core реконструкция 3D модели

Обработка с 5% захлестом:

Можем ли мы просто
обработать пространство
по кусочкам?



Out-of-Core реконструкция 3D модели

Обработка in-core или правильный Out-of-Core:

Можем ли мы просто
обработать пространство
по кусочкам?

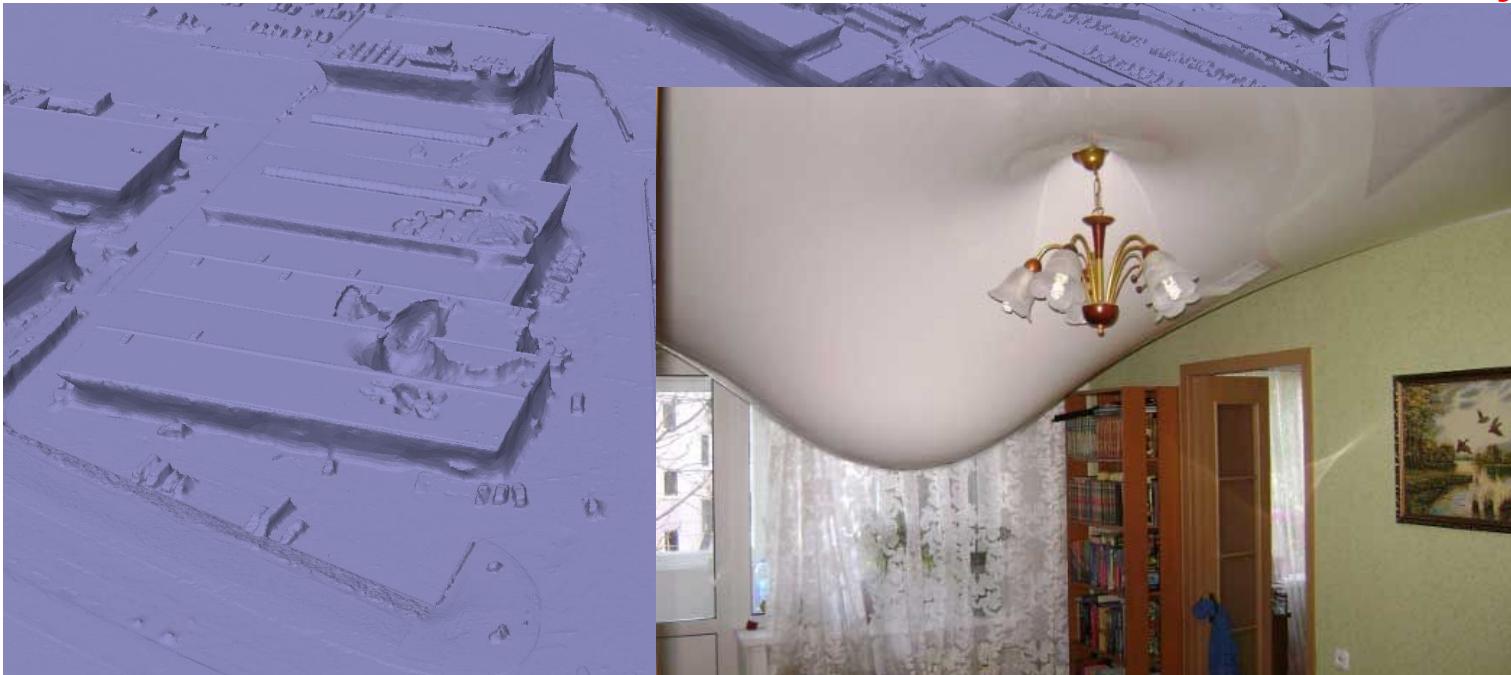


Результат идеален: watertight полигональная модель.

Out-of-Core реконструкция 3D модели

Обработка in-core или Out-of-Core но правильная:

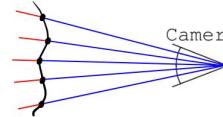
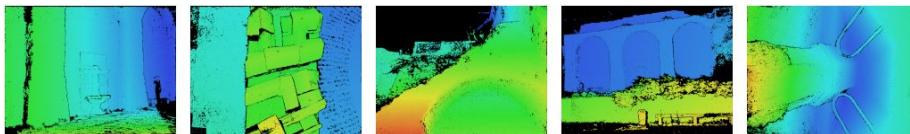
Можем ли мы просто
обработать пространство
по кусочкам?



Результат идеален: watertight полигональная модель.

Out-of-Core реконструкция 3D модели

- 1) Есть множество карт глубины

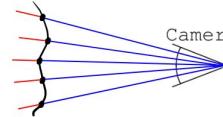
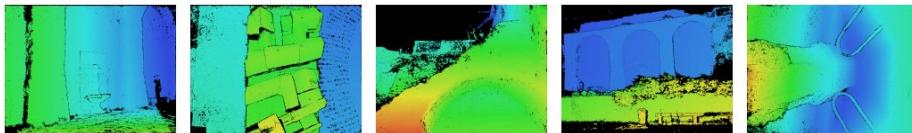


- 2) Множество точек всех карт глубины - порождает **адаптивное** октодерево (дискретизация пространства)

Как построить октодерево не упав по памяти?

Out-of-Core реконструкция 3D модели

- 1) Есть множество карт глубины



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

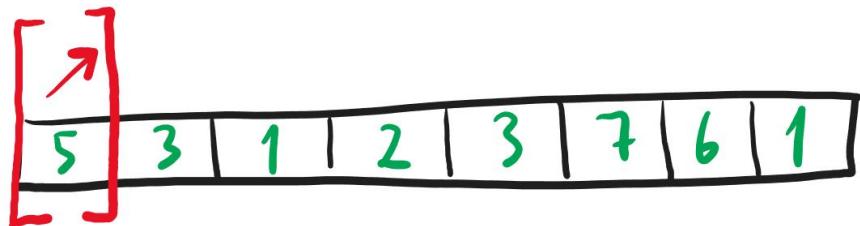
Как построить октодерево не упав по памяти?

Строить его храня на диске!

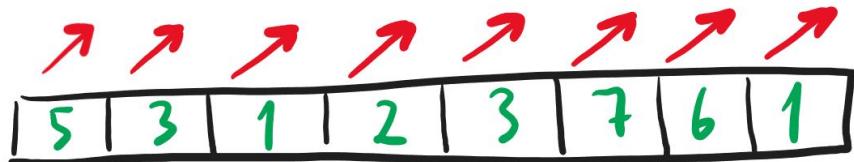
Out-of-Core сортировка (k-way merge sort)



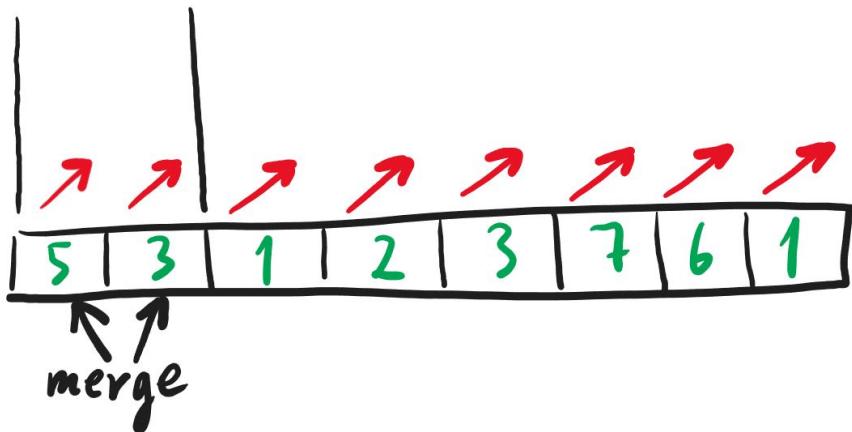
Out-of-Core сортировка (k-way merge sort)



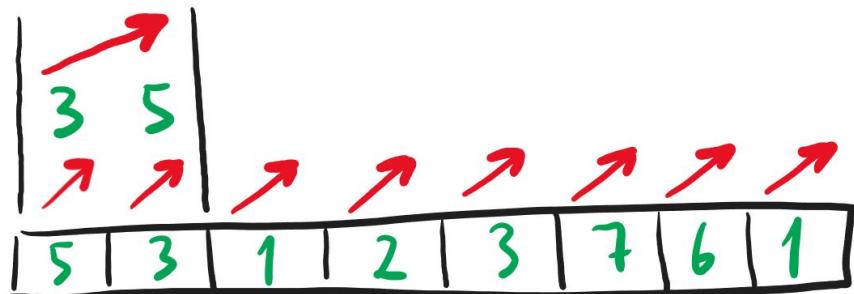
Out-of-Core сортировка (k-way merge sort)



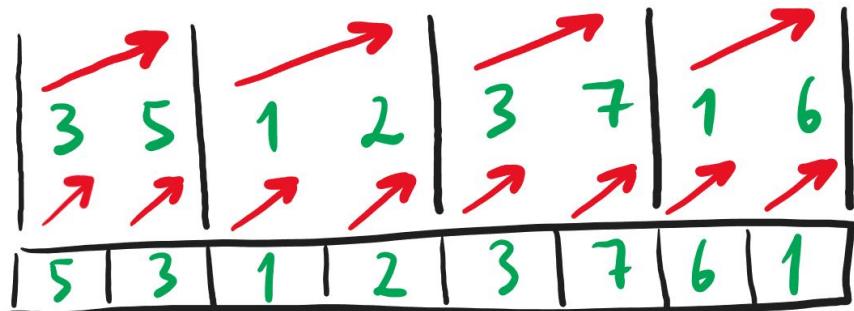
Out-of-Core сортировка (k-way merge sort)



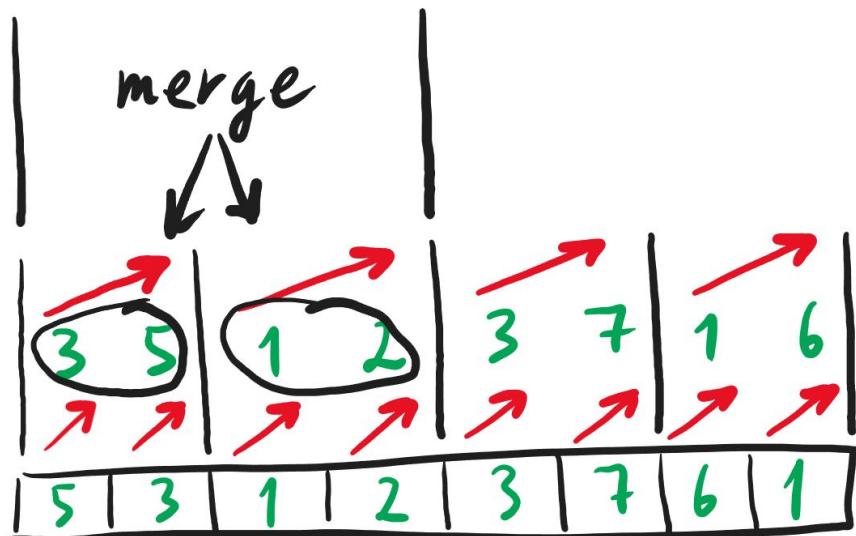
Out-of-Core сортировка (k-way merge sort)



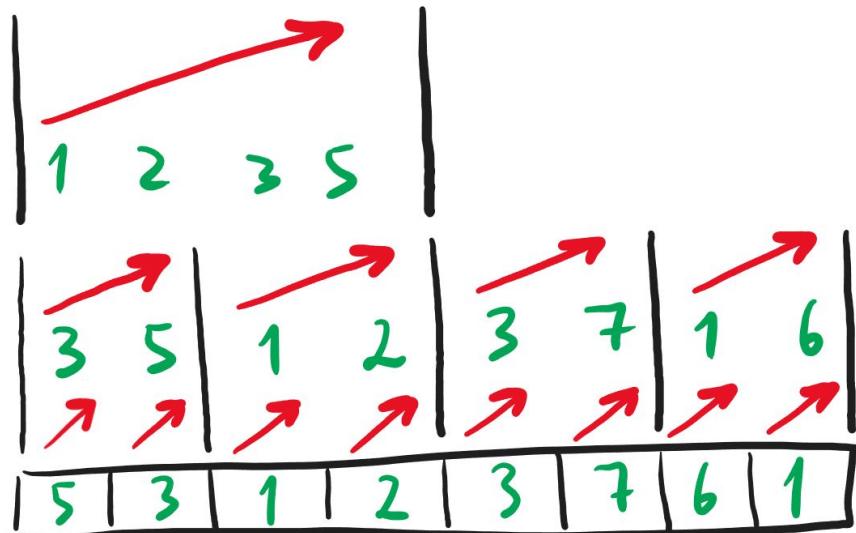
Out-of-Core сортировка (k-way merge sort)



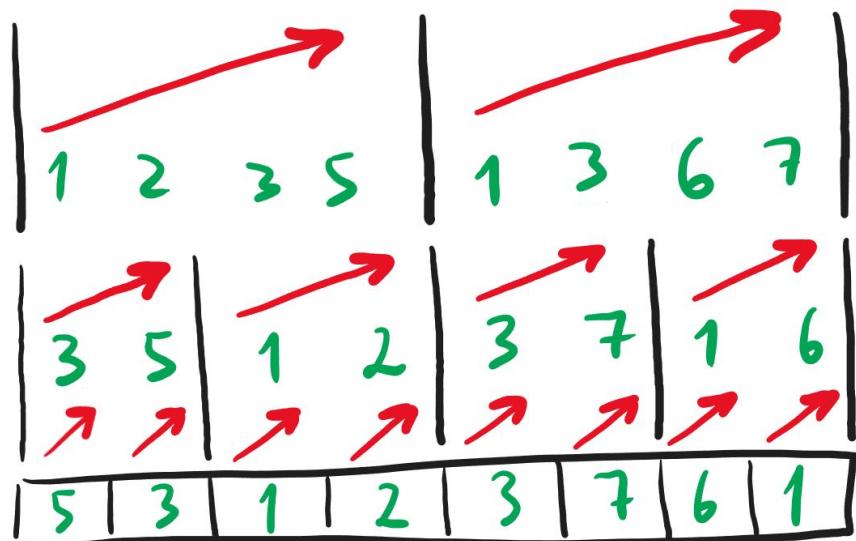
Out-of-Core сортировка (k-way merge sort)



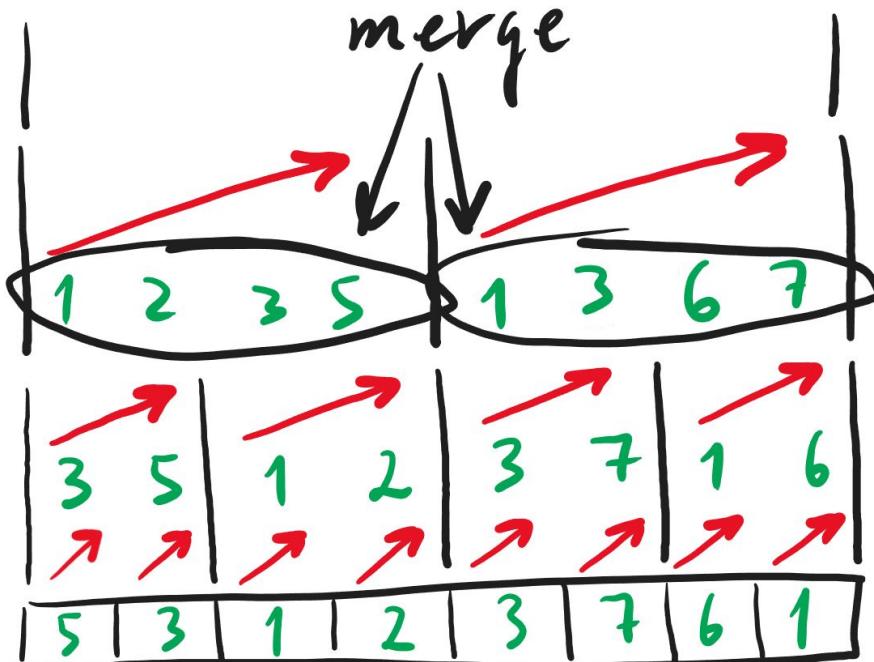
Out-of-Core сортировка (k-way merge sort)



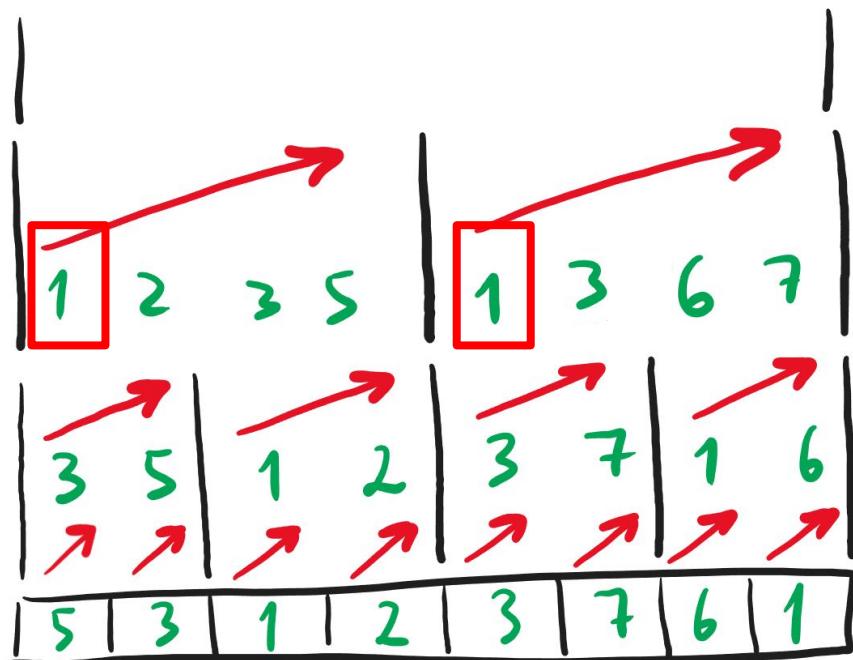
Out-of-Core сортировка (k-way merge sort)



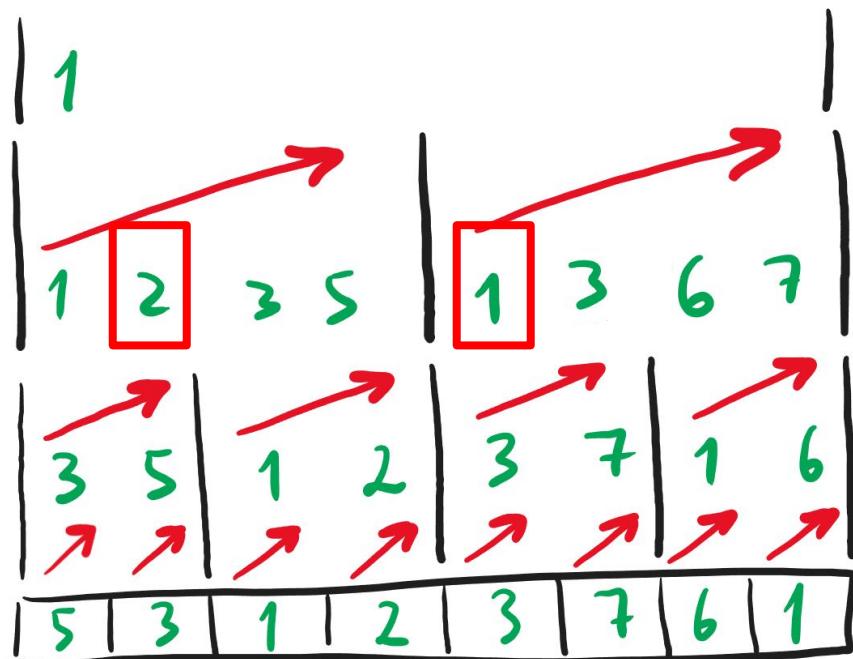
Out-of-Core сортировка (k-way merge sort)



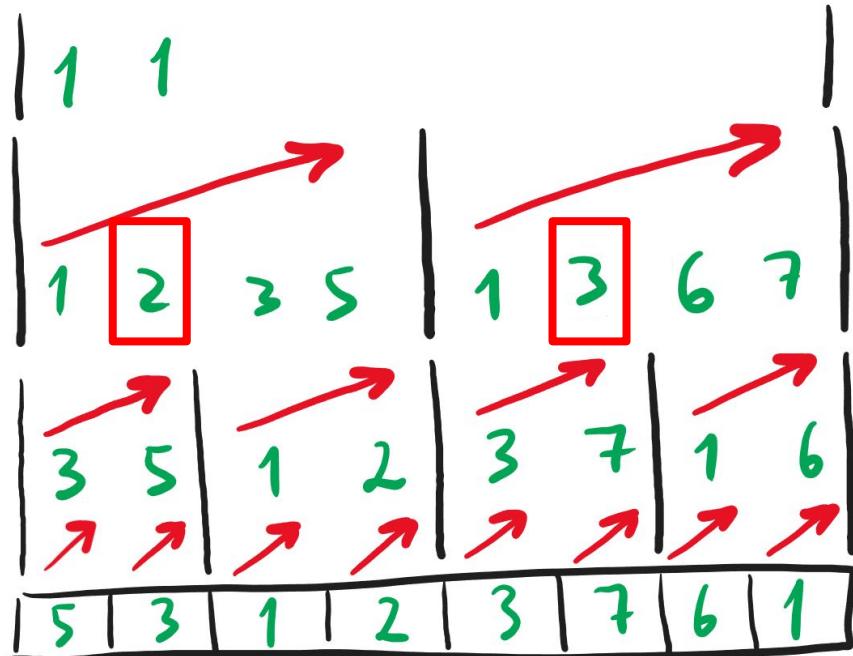
Out-of-Core сортировка (k-way merge sort)



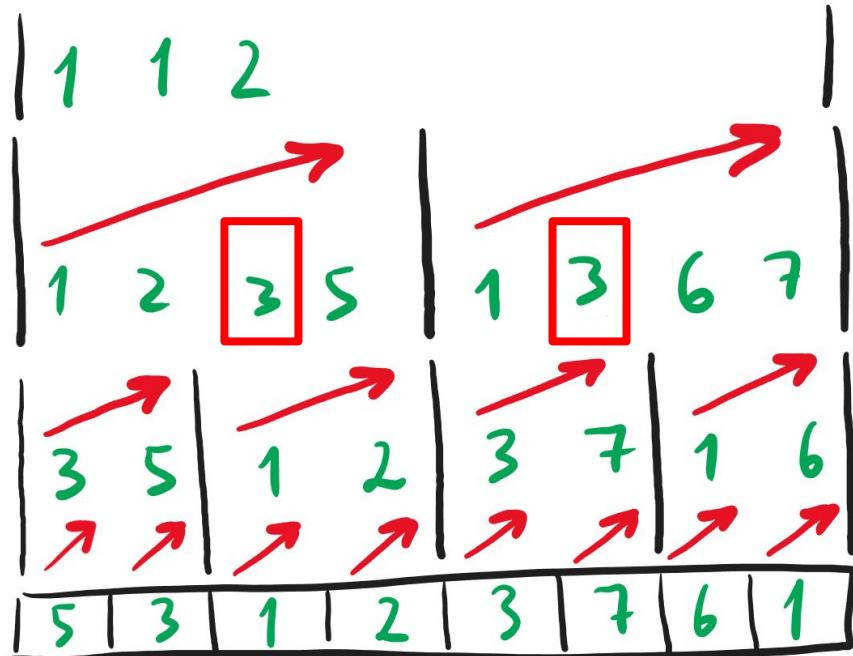
Out-of-Core сортировка (k-way merge sort)



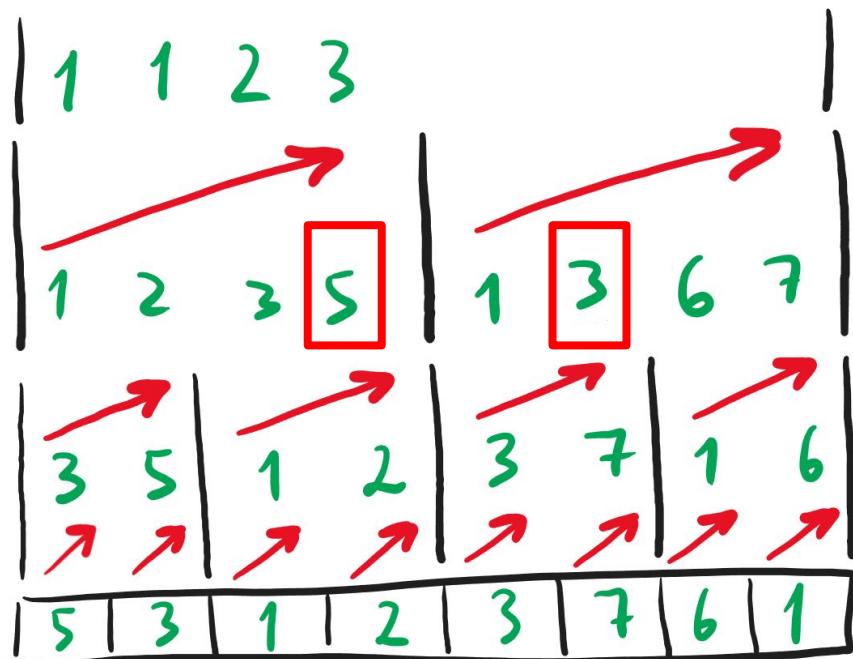
Out-of-Core сортировка (k-way merge sort)



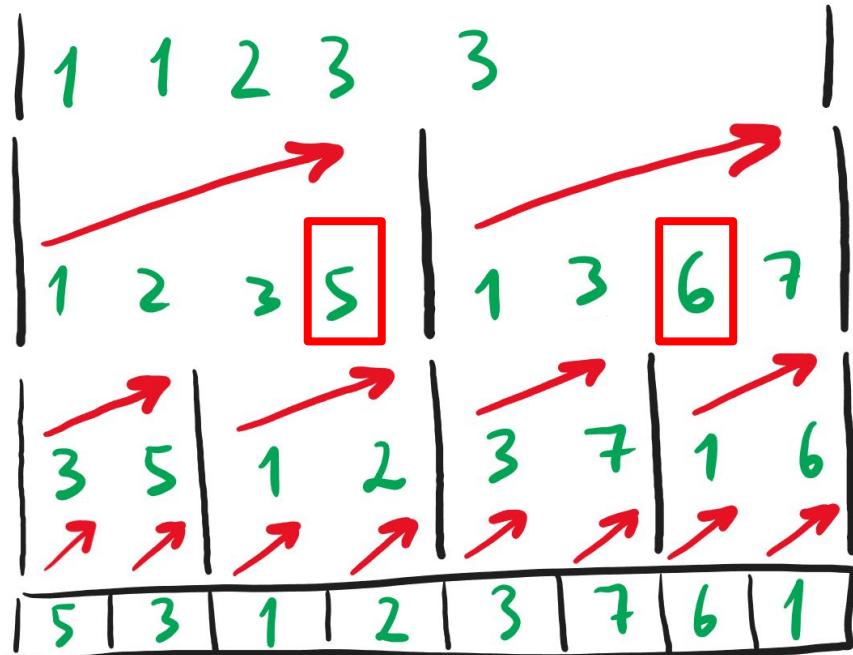
Out-of-Core сортировка (k-way merge sort)



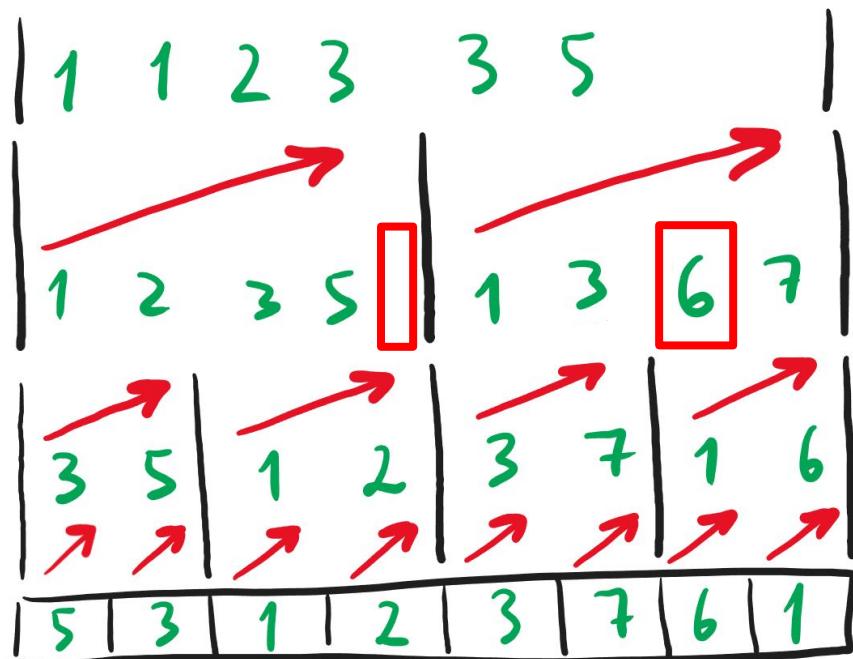
Out-of-Core сортировка (k-way merge sort)



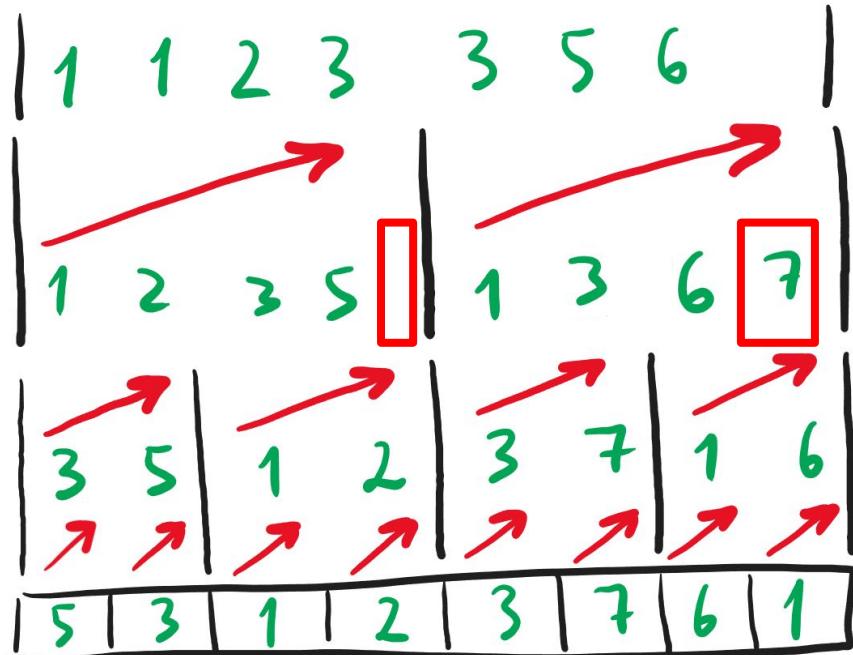
Out-of-Core сортировка (k-way merge sort)



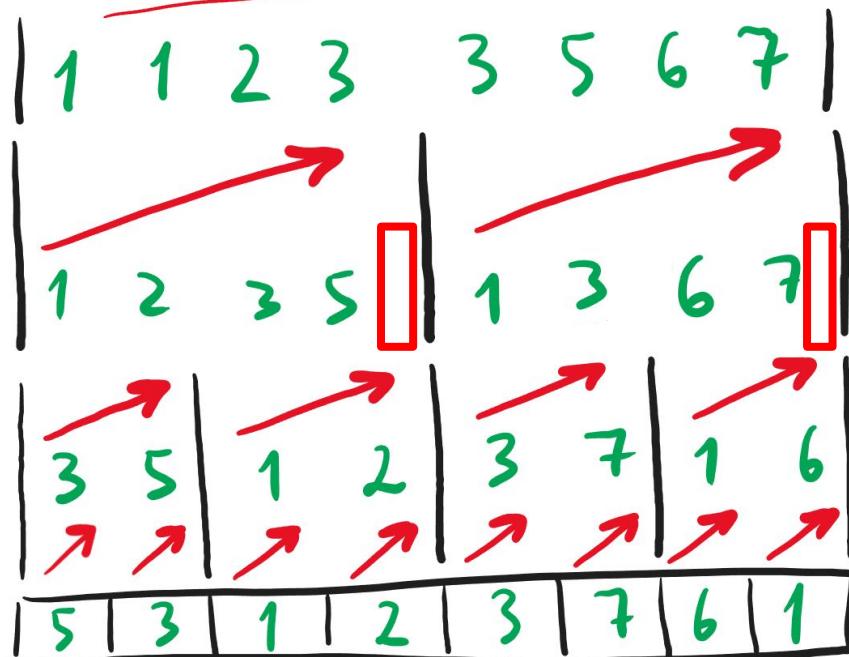
Out-of-Core сортировка (k-way merge sort)



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

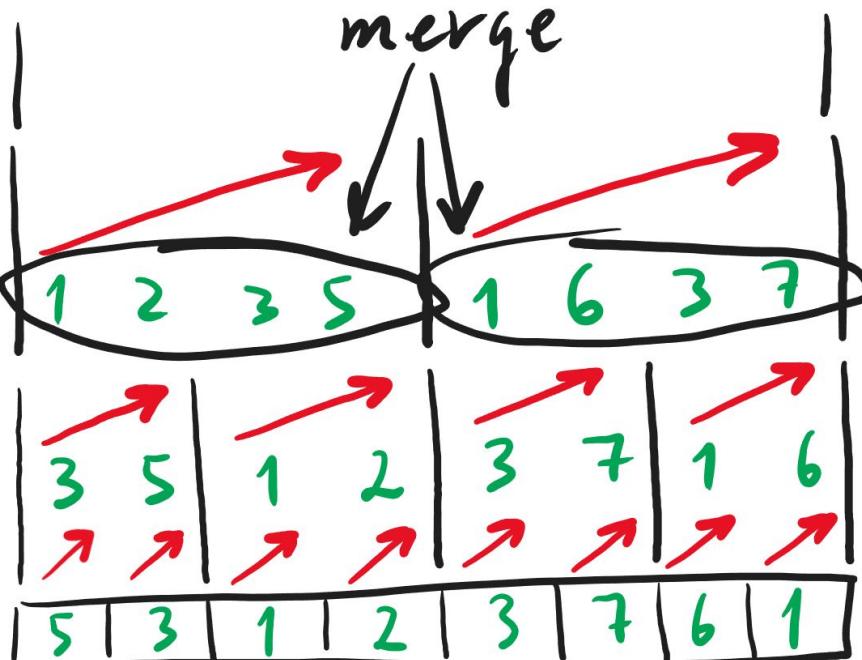


Out-of-Core сортировка (k-way merge sort)



Как отсортировать 10 терабайт чисел?

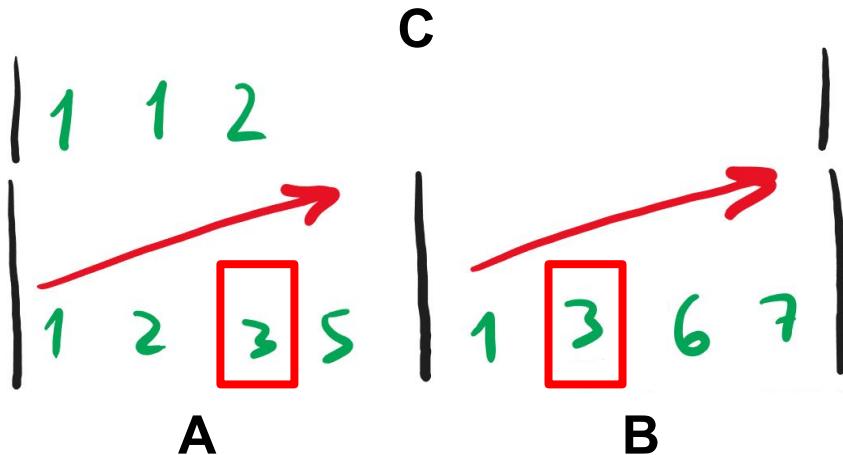
Out-of-Core сортировка (k-way merge sort)



Как отсортировать 10 терабайт чисел?

Достаточно научиться выполнять
merge в **Out-of-Core** манере!

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

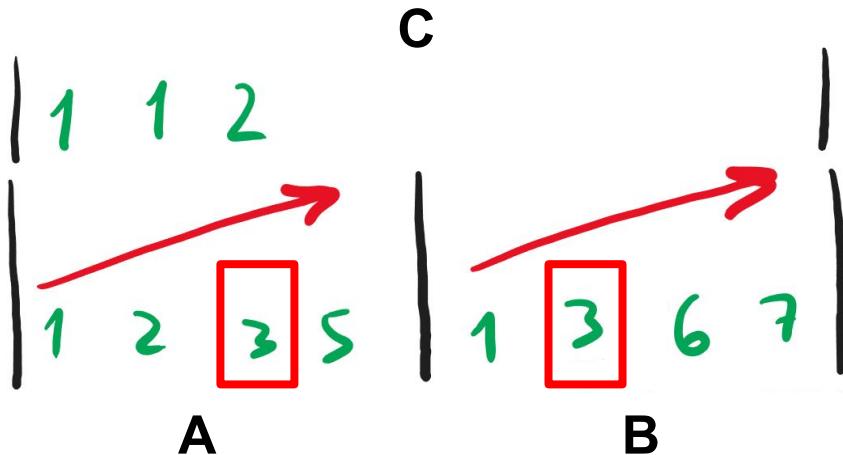


Как отсортировать 10 терабайт чисел?

Достаточно научиться выполнять
 $C = \text{merge}(A, B)$ в **Out-of-Core** манере!

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

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



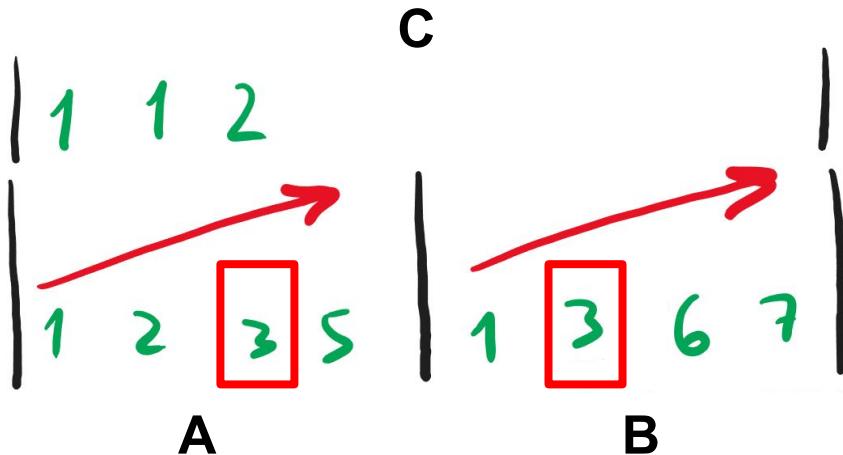
Как отсортировать 10 терабайт чисел?

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)



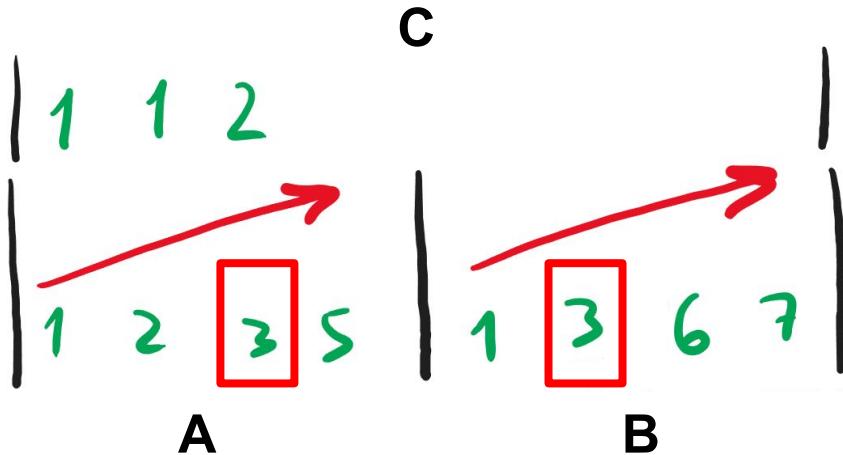
Как отсортировать 10 терабайт чисел?

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)



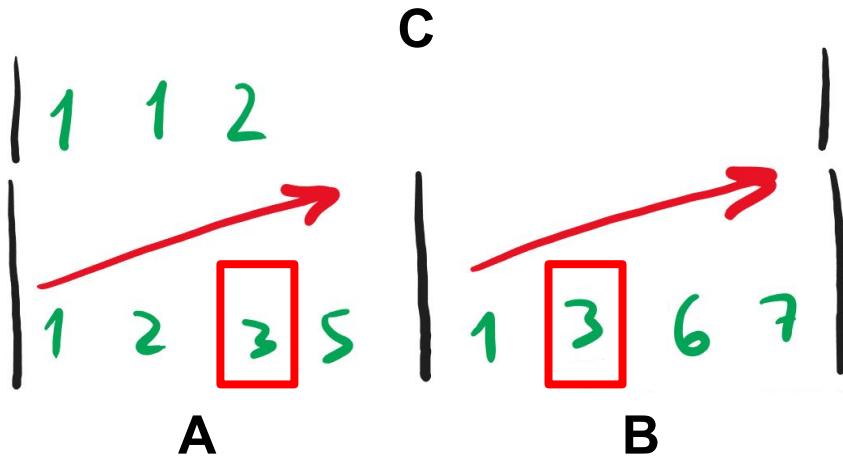
Как отсортировать 10 терабайт чисел?

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)



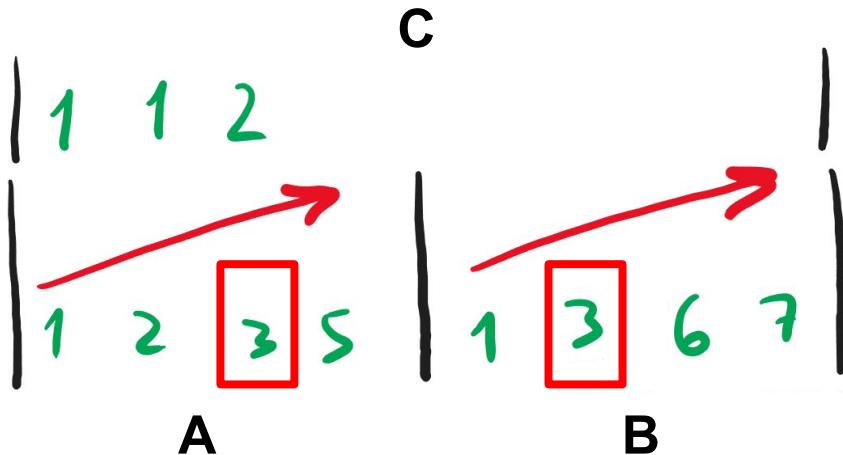
Как отсортировать 10 терабайт чисел?

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)



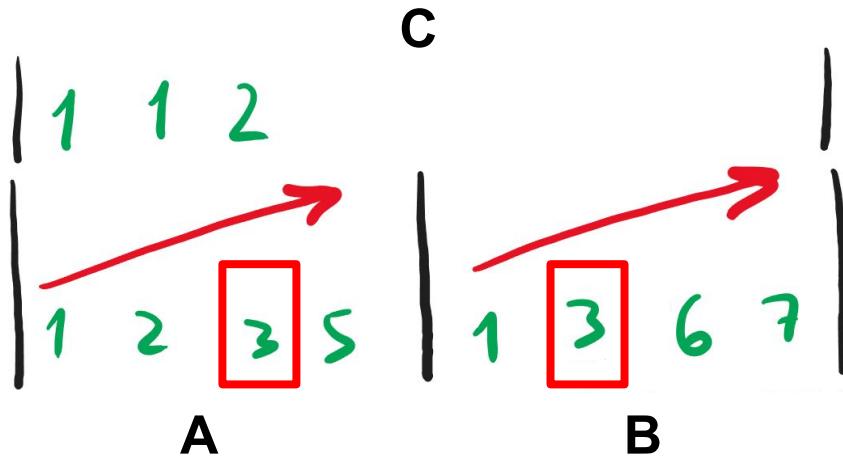
Как отсортировать 10 терабайт чисел?

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)



Как отсортировать 10 терабайт чисел?

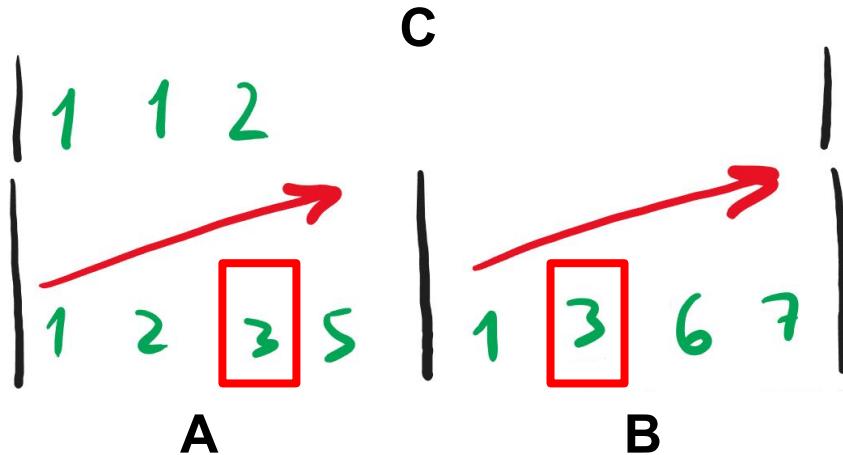
Достаточно научиться выполнять
C = merge(A, B) в **Out-of-Core** манере!

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

Какое суммарное **IO merge**(5 ТВ + 5 ТВ)?

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!

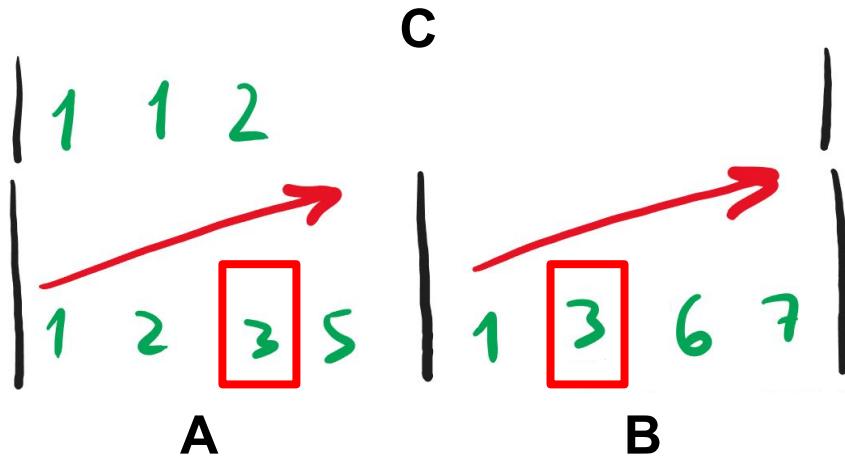
Как выполнить **merge** двух 5 терабайтных
файлов **отсортированных** чисел?

- 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

Какое суммарное **IO merge**(5 ТБ + 5 ТБ)?

Какое суммарное **IO merge-sort**(N ТБ)?

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!

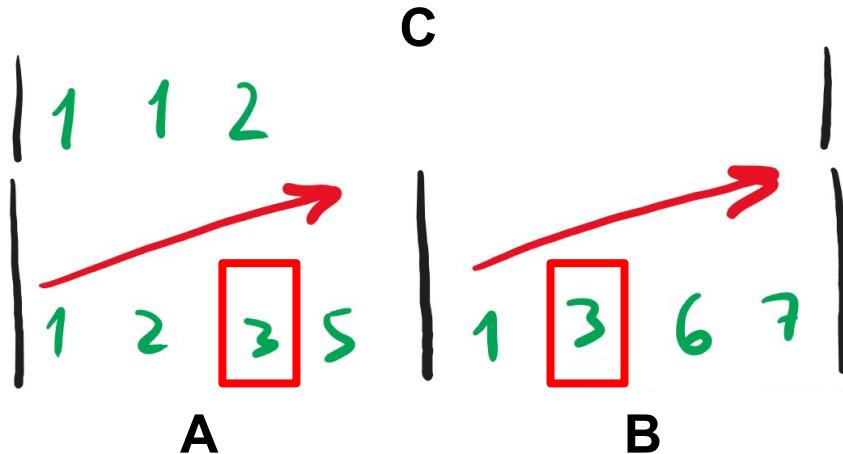
Как выполнить **merge** двух 5 терабайтных
файлов **отсортированных** чисел?

- 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

сколько влезает в память (т.е. можно обработать **in-core**)

$$\log_2 \frac{N}{B} \cdot (2 \cdot N)$$

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



Какое суммарное **IO** $\text{merge}(5 \text{ TB} + 5 \text{ TB})$?

Какое суммарное **IO** $\text{merge-sort}(N \text{ TB})$?

$$\log_2 \frac{N}{B} \cdot (2 \cdot N)$$

Как уменьшить?

сколько влезает в память (т.е. можно обработать **in-core**)

Как отсортировать 10 терабайт чисел?

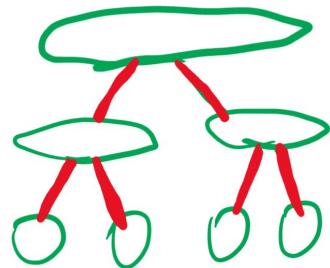
Достаточно научиться выполнять
 $C = \text{merge}(A, B)$ в **Out-of-Core** манере!

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)

$k=2$



Как отсортировать 10 терабайт чисел?

Достаточно научиться выполнять
 $C = \text{merge}(A, B)$ в **Out-of-Core** манере!

Как выполнить **merge** K файлов
отсортированных чисел?

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

$$\log_2 \frac{N}{B} \cdot (2 \cdot N)$$

сколько влезает в память (т.е. можно обработать **in-core**)

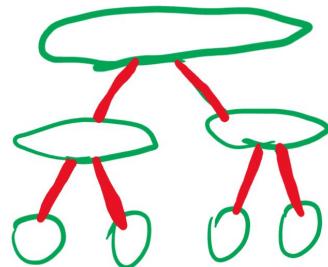
сколько файлов объединяется за проход (число кандидатов)

Как уменьшить?

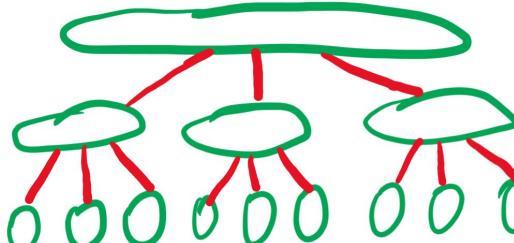
$$\log_{\sqrt{k}} \frac{N}{B} \cdot (2 \cdot N)$$

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

$k=2$



$k=3$



Как отсортировать 10 терабайт чисел?

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

Как выполнить **merge** K файлов
отсортированных чисел?

Какое суммарное **IO merge**(5 TB + 5 TB)?

Какое суммарное **IO merge-sort**(N TB)?

$$\log_2 \frac{N}{B} \cdot (2 \cdot N)$$

Как уменьшить?

$$\log_{\boxed{k}} \frac{N}{B} \cdot (2 \cdot N)$$

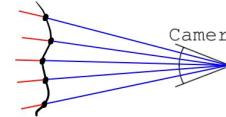
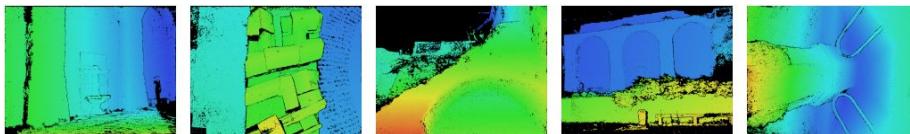
- 1) Создали пустой файл для **C**
- 2) **K** указателей - позиция чтения **A, B**
- 3) В памяти только **K** чисел-кандидатов
- 4) Минимальный из них - **пишем в C**
- 5) **Считываем** новое число-кандидат

сколько влезает в память (т.е. можно обработать **in-core**)

сколько файлов объединяется **за проход** (число кандидатов)

2) Out-of-Core реконструкция: строим октодерево

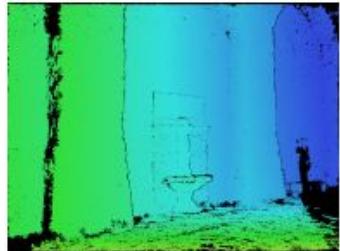
- 1) Есть множество карт глубины



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

Как построить октодерево не упав по памяти?

2) Out-of-Core реконструкция: строим октодерево

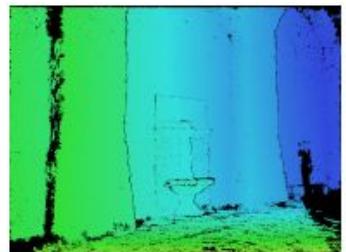


Octree #1



Воксели

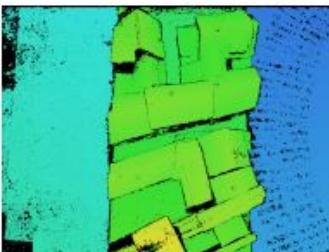
2) Out-of-Core реконструкция: строим октодерево



Octree #1



Воксели

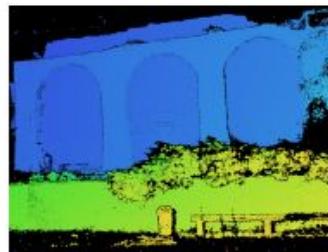


Octree #2



Воксели

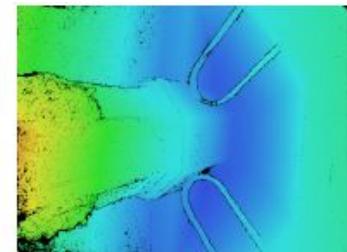
...



Octree #N-1



Воксели

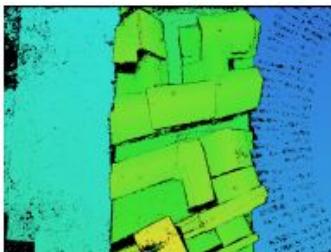
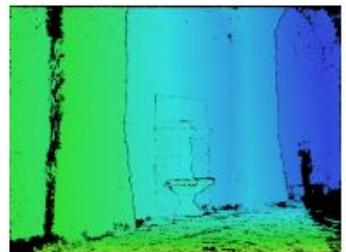


Octree #N



Воксели

2) Out-of-Core реконструкция: строим октодерево



Octree #1



~~Воксели~~

96-bit

Morton Codes

Octree #2



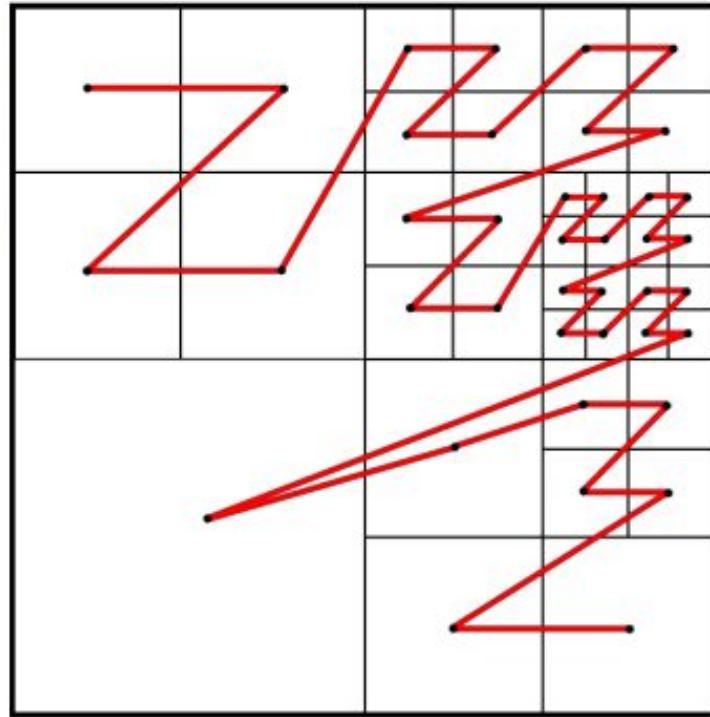
~~Воксели~~

96-bit

Morton Codes

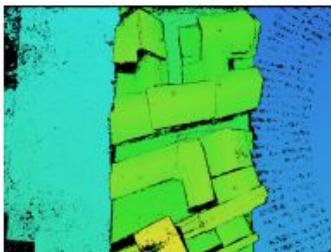
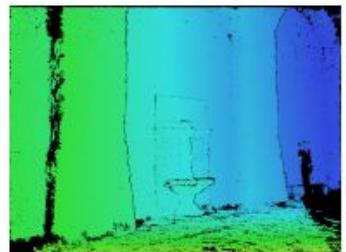
...

...



Z-curve: ввели линейный порядок на вокселях.
Есть пространственная локальность!

2) Out-of-Core реконструкция: строим октодерево



Octree #1



~~Воксели~~

96-bit

Morton Codes

Octree #2



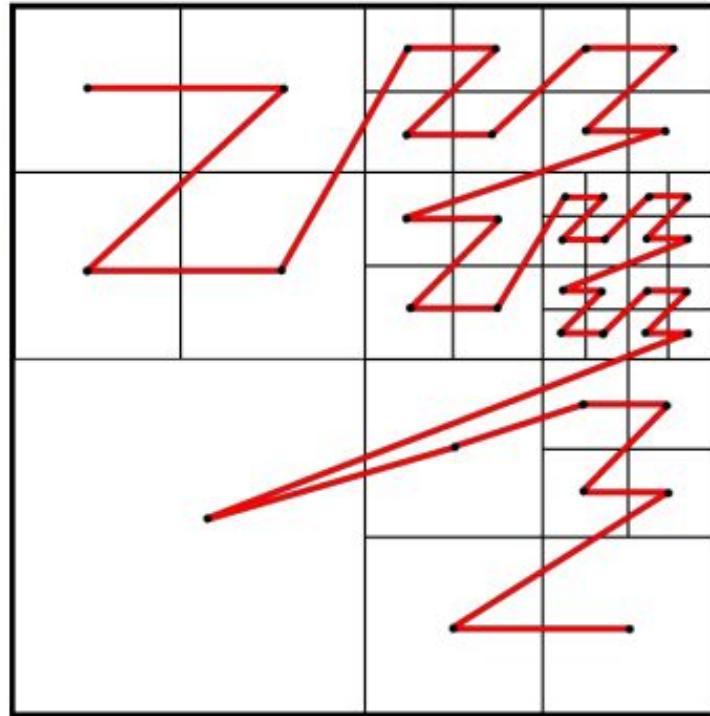
~~Воксели~~

96-bit

Morton Codes

...

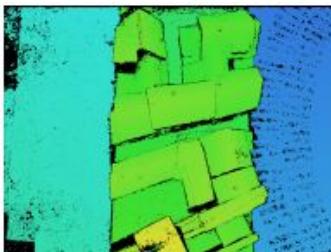
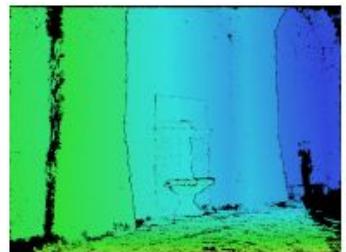
...



Z-curve: ввели линейный порядок на voxелях.
Есть пространственная локальность!

Morton Codes: биекция voxель \rightarrow число,
такая что числа соблюдают порядок Z-curve.

2) Out-of-Core реконструкция: строим октодерево



Octree #1



~~Воксели~~

96-bit

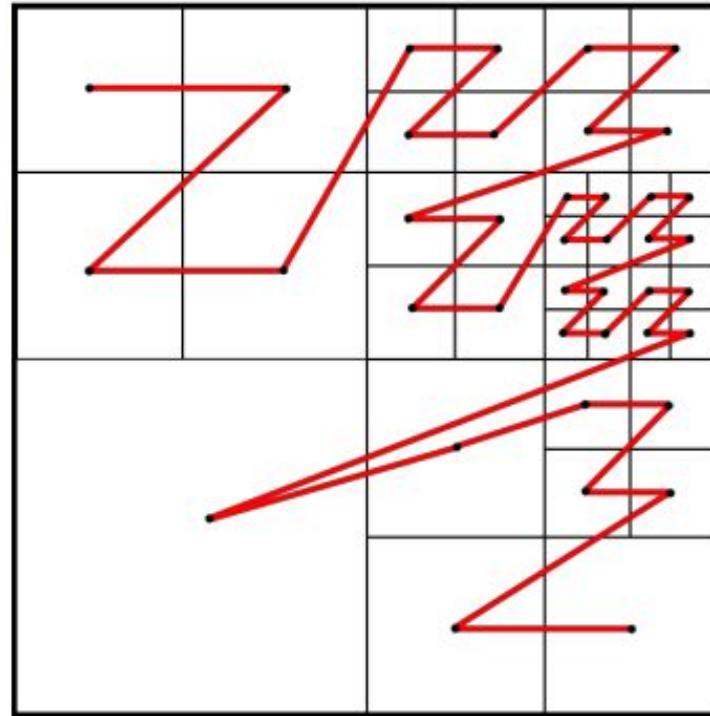
Morton Codes Morton Codes

Shuffled xyz Morton Code:

$x_1y_1z_1x_2y_2z_2 \dots x_Dy_Dz_D$

...

...

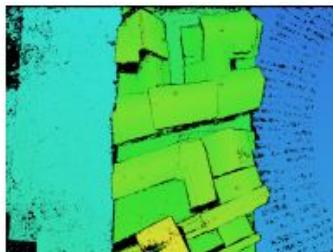
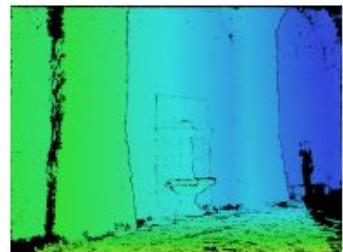


Z-curve: ввели линейный порядок на voxelах.

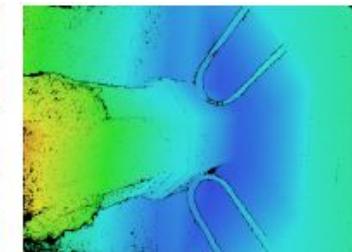
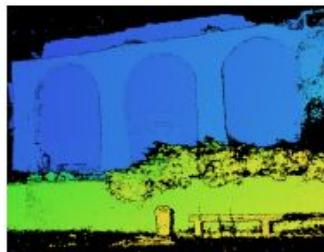
Есть пространственная локальность!

Morton Codes: биекция voxel → число, такая что числа соблюдают порядок Z-curve.

2) Out-of-Core реконструкция: строим октодерево



...



Octree #1



96-bit

Morton Codes



Octree #2



96-bit

Morton Codes

...



Octree #N-1



96-bit

Morton Codes



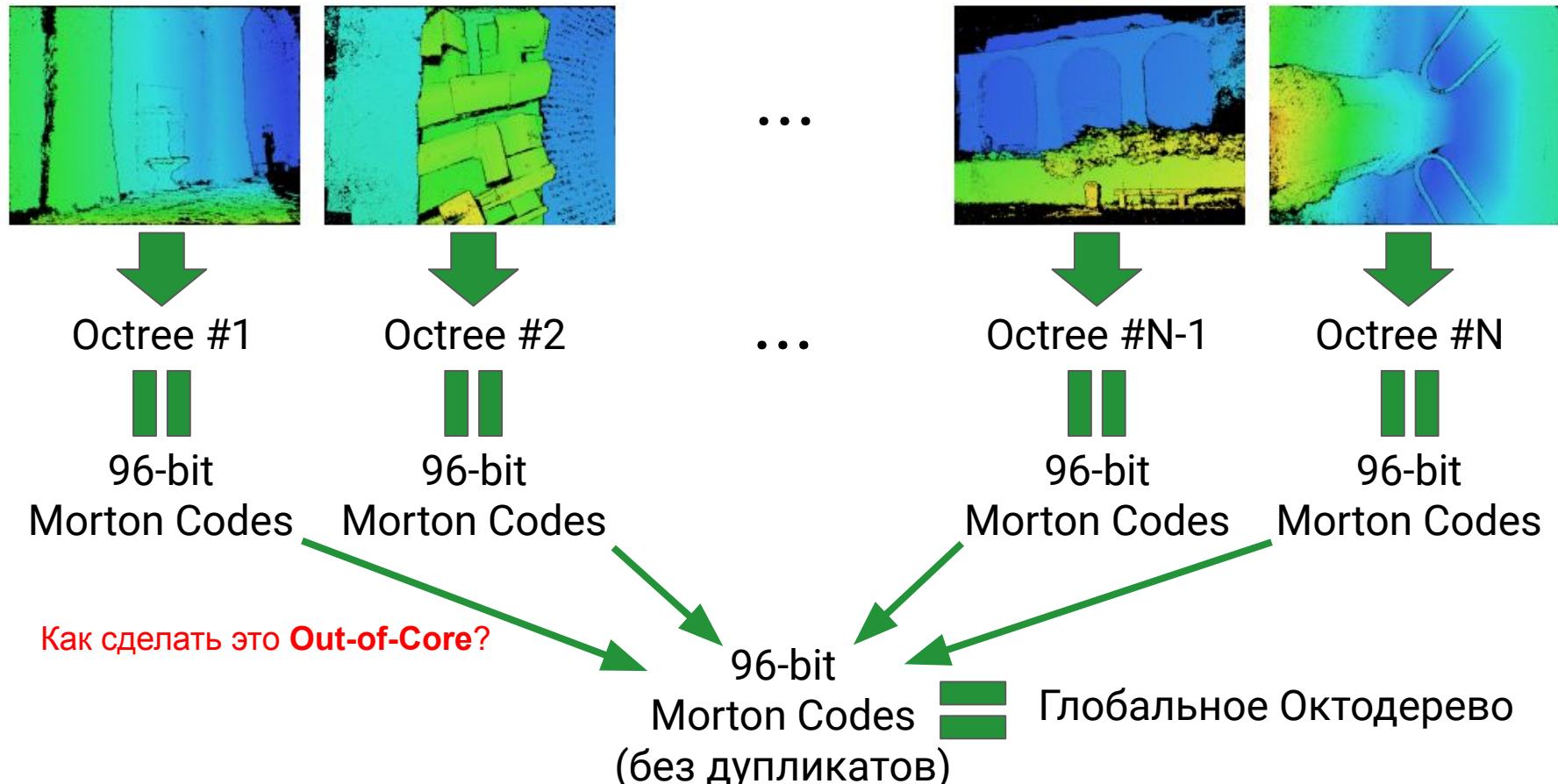
Octree #N



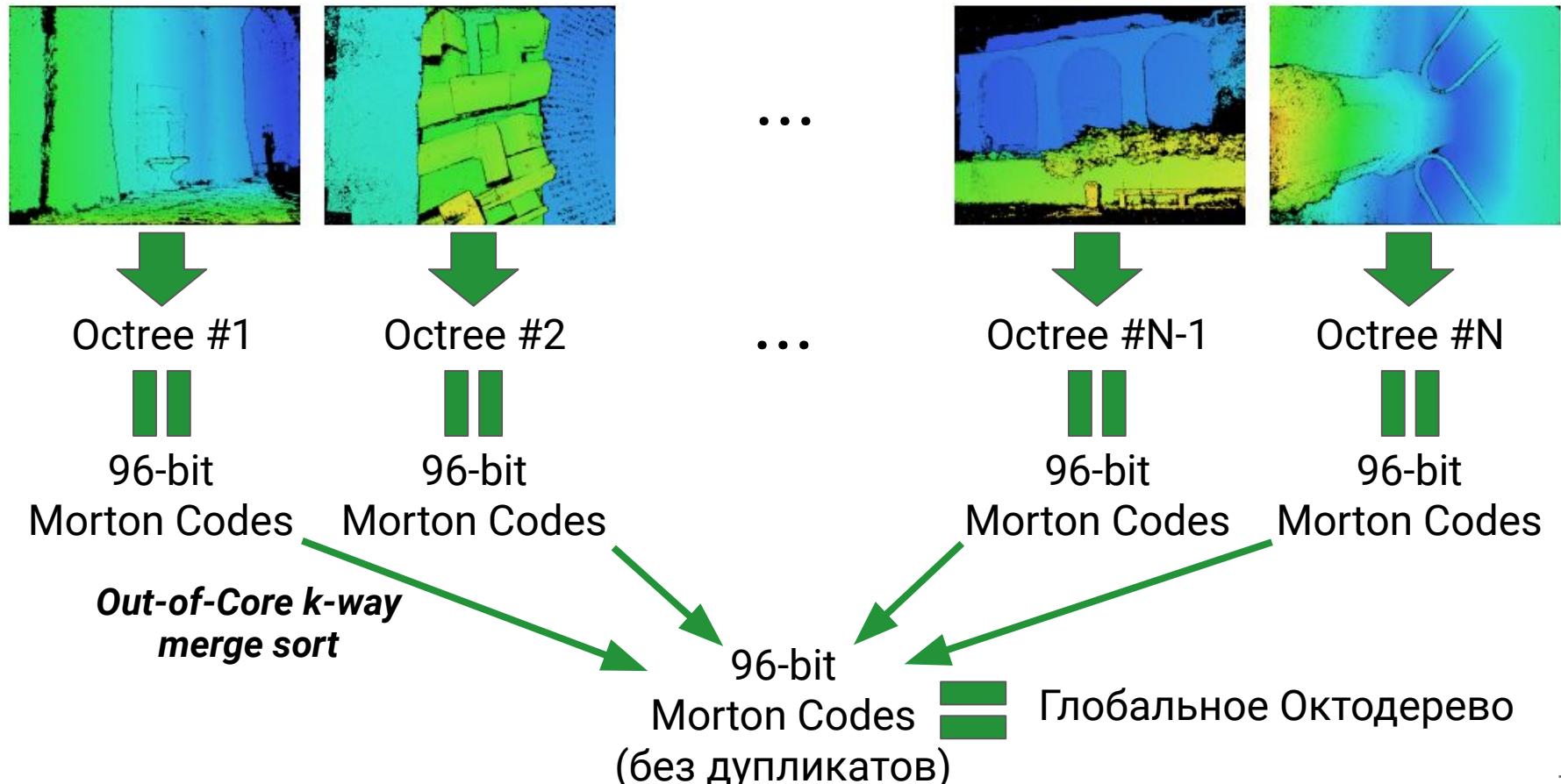
96-bit

Morton Codes

2) Out-of-Core реконструкция: строим октодерево

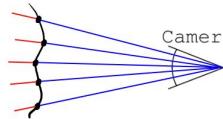
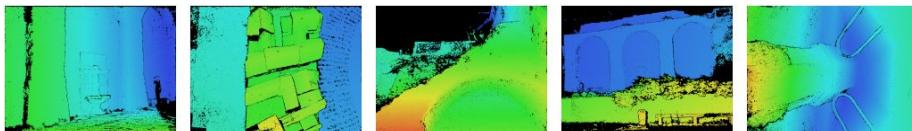


2) Out-of-Core реконструкция: строим октодерево



2) Out-of-Core реконструкция: строим октодерево

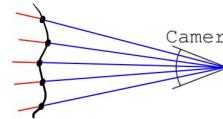
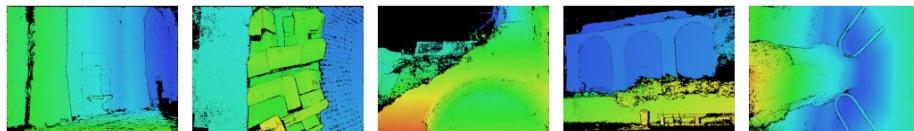
- 1) Есть множество карт глубины



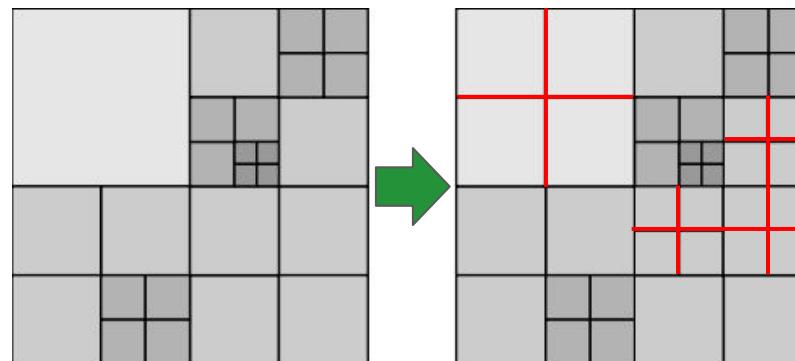
- 2) Множество точек всех карт глубины - порождает **адаптивное** октодерево (дискретизация пространства)
- 3) Балансируем октодерево 2:1 (по каждой стороне макс. два соседа)

3) Out-of-Core реконструкция: балансируем октодерево

- 1) Есть множество карт глубины

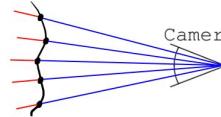
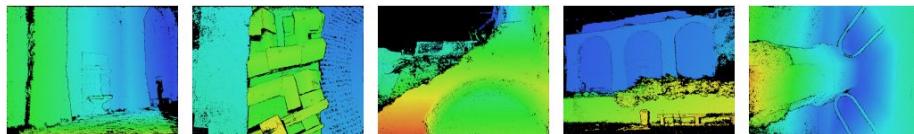


- 2) Множество точек всех карт глубины - порождает **адаптивное** октодерево (дискретизация пространства)
- 3) Балансируем октодерево 2:1 (по каждой стороне макс. два соседа)

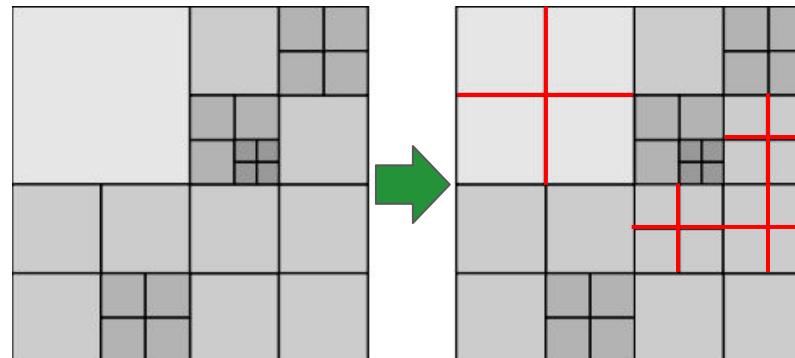


3) Out-of-Core реконструкция: балансируем октодерево

- 1) Есть множество карт глубины

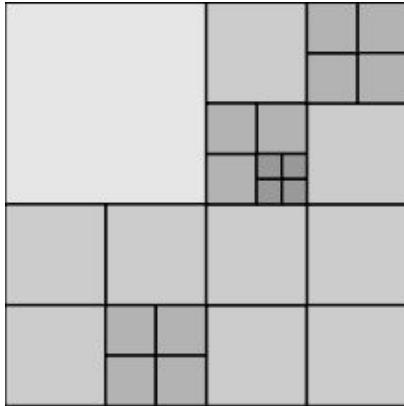


- 2) Множество точек всех карт глубины - порождает **адаптивное** октодерево (дискретизация пространства)
- 3) Балансируем октодерево 2:1 (по каждой стороне макс. два соседа)

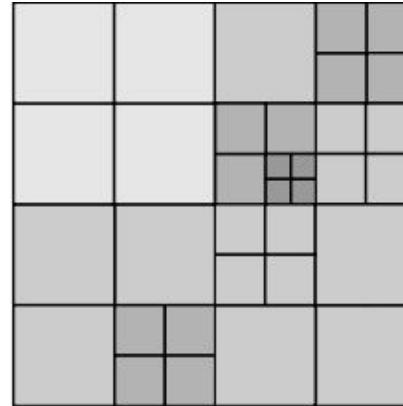


Как выполнить балансировку Out-of-Core?

3) Out-of-Core реконструкция: балансируем октодерево



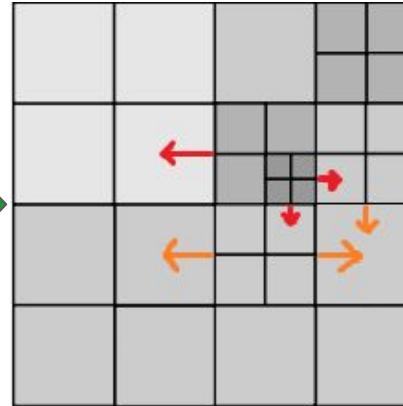
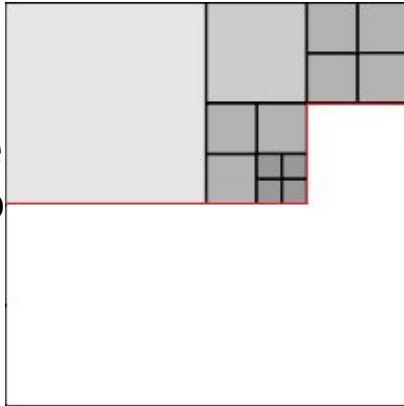
Глобальное
Октодерево:
один огромный файл
с отсортированными
Morton Codes



Балансированное
Глобальное
Октодерево

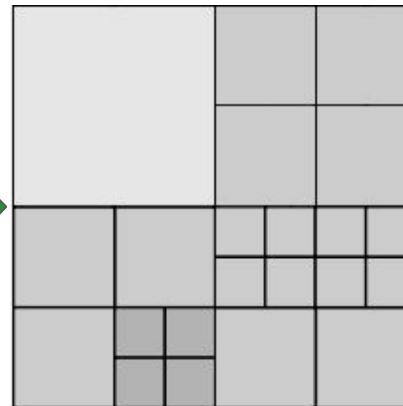
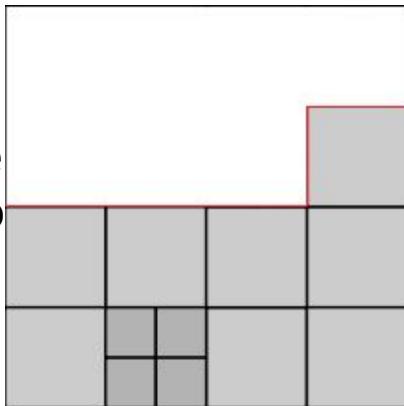
3) Out-of-Core реконструкция: балансируем октодерево

Глобальное
Октодерево
(part #1)



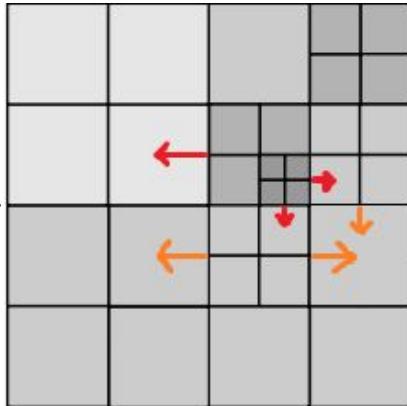
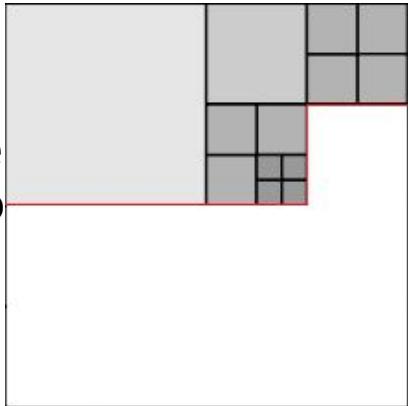
IO-friendly:
Исходные кубы на диске
последовательны
благодаря сортировке
Morton Codes

Глобальное
Октодерево
(part #2)

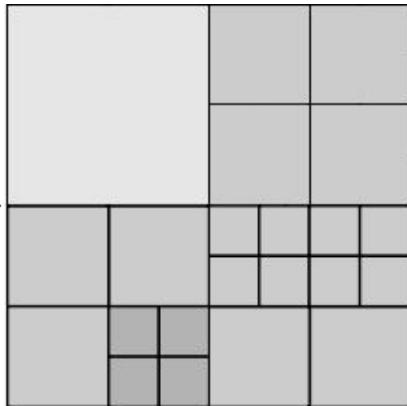
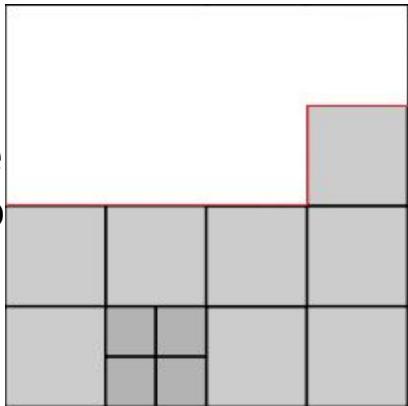


3) Out-of-Core реконструкция: балансируем октодерево

Глобальное
Октодерево
(part #1)

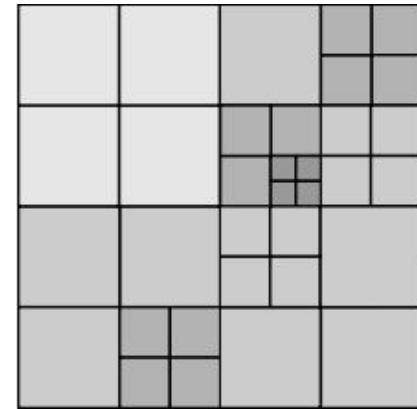


Глобальное
Октодерево
(part #2)



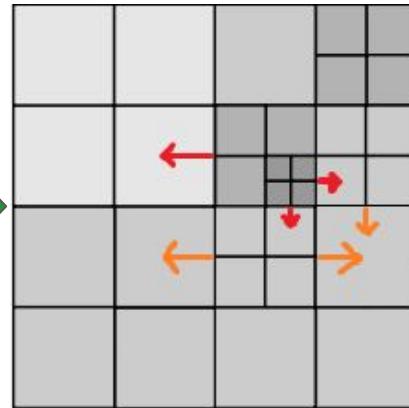
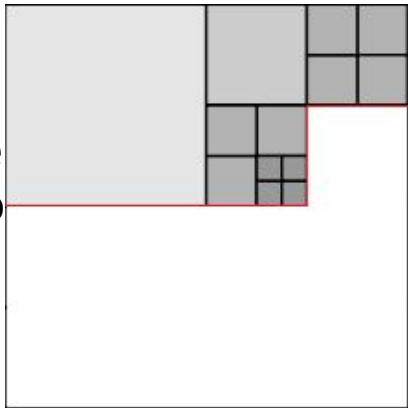
Как объединить
Out-of-Core?

Балансированное
Глобальное
Октодерево

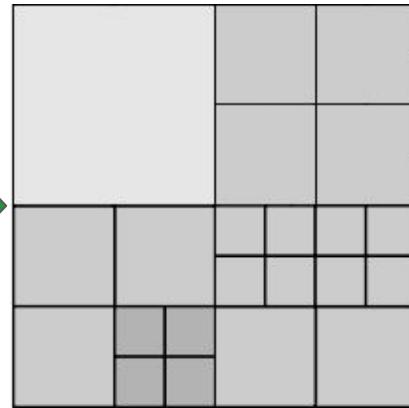
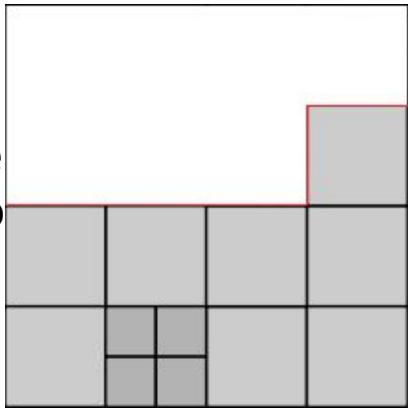


3) Out-of-Core реконструкция: балансируем октодерево

Глобальное
Октодерево
(part #1)

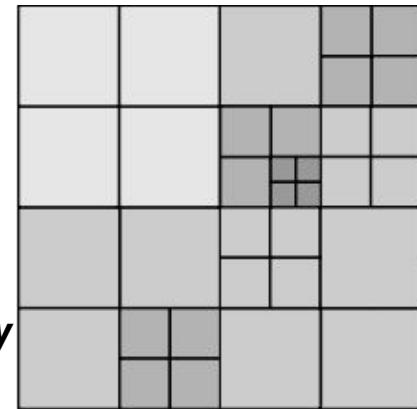


Глобальное
Октодерево
(part #2)



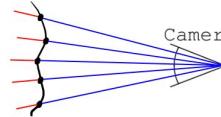
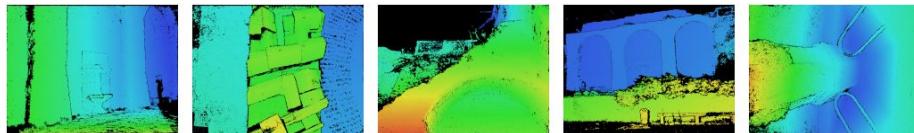
*Out-of-Core k-way
merge sort*

Балансированное
Глобальное
Октодерево



3) Out-of-Core реконструкция: балансируем октодерево

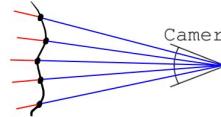
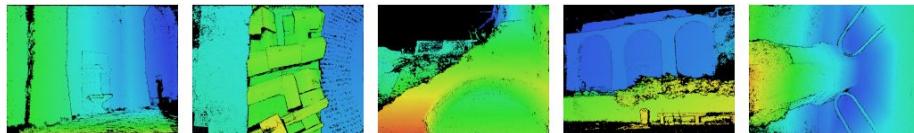
- 1) Есть множество карт глубины



- 2) Множество точек всех карт глубины - порождает **адаптивное** октодерево (дискретизация пространства)
- 3) Балансируем октодерево 2:1 (по каждой стороне макс. два соседа)
- 4)  Каждая карта глубины вносит индикаторные f_i голоса за voxели

4) Out-of-Core реконструкция: карты глубины голосуют

- 1) Есть множество карт глубины

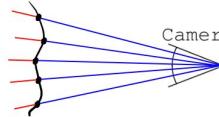
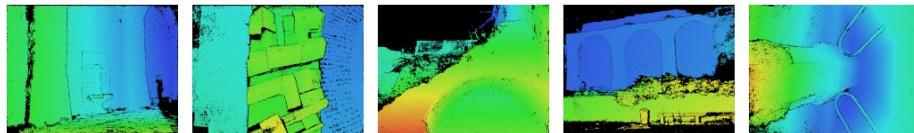


- 2) Множество точек всех карт глубины - порождает **адаптивное** октодерево (дискретизация пространства)
- 3) Балансируем октодерево 2:1 (по каждой стороне макс. два соседа)
- 4)  Каждая карта глубины вносит индикаторные f_i голоса за воксели

Нужно спроектировать каждый воксель, но октодерево не влезает в память!
Что делать?

4) Out-of-Core реконструкция: карты глубины голосуют

- 1) Есть множество карт глубины

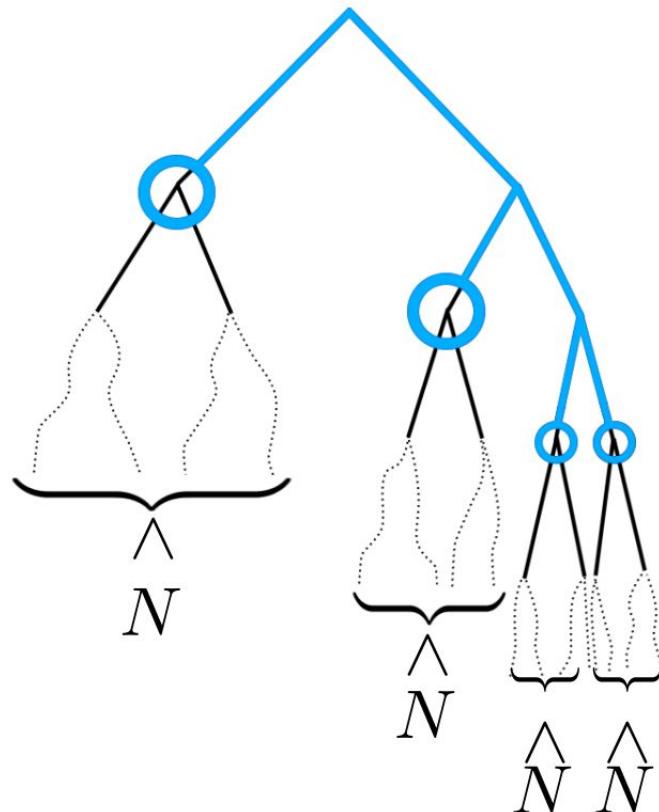


- 2) Множество точек всех карт глубины - порождает **адаптивное** октодерево (дискретизация пространства)
- 3) Балансируем октодерево 2:1 (по каждой стороне макс. два соседа)
- 4)  Каждая карта глубины вносит индикаторные f_i голоса за воксели

Нужно спроектировать каждый воксель, но октодерево не влезает в память!
Значит необходимо разбить октодерево на куски, и обработать по кускам!

4) Out-of-Core реконструкция: карты глубины голосуют

Разбиение дерева на куски: построение верхушки дерева

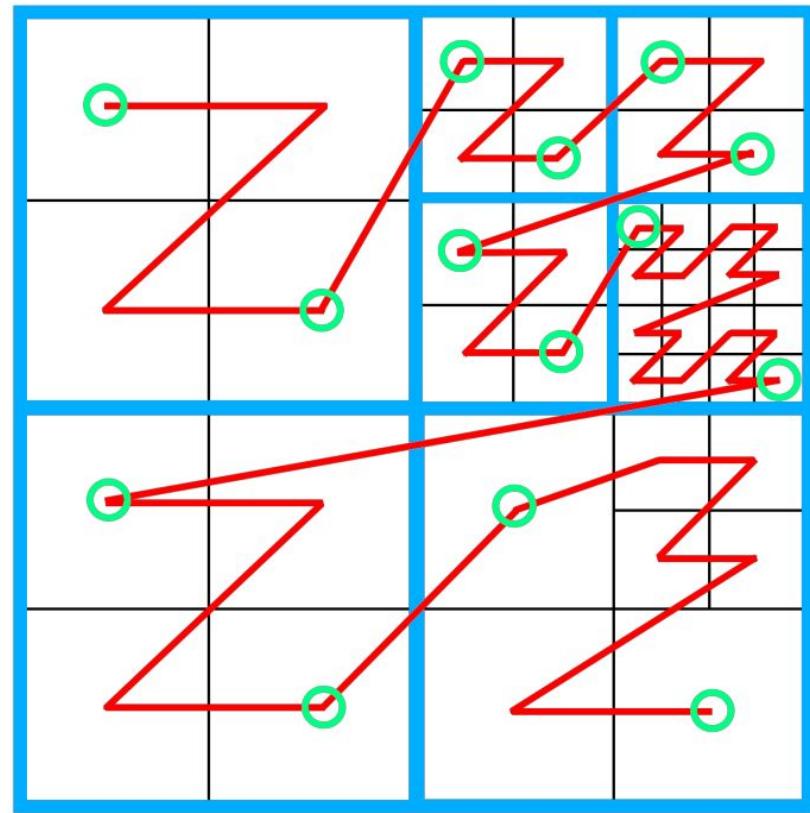
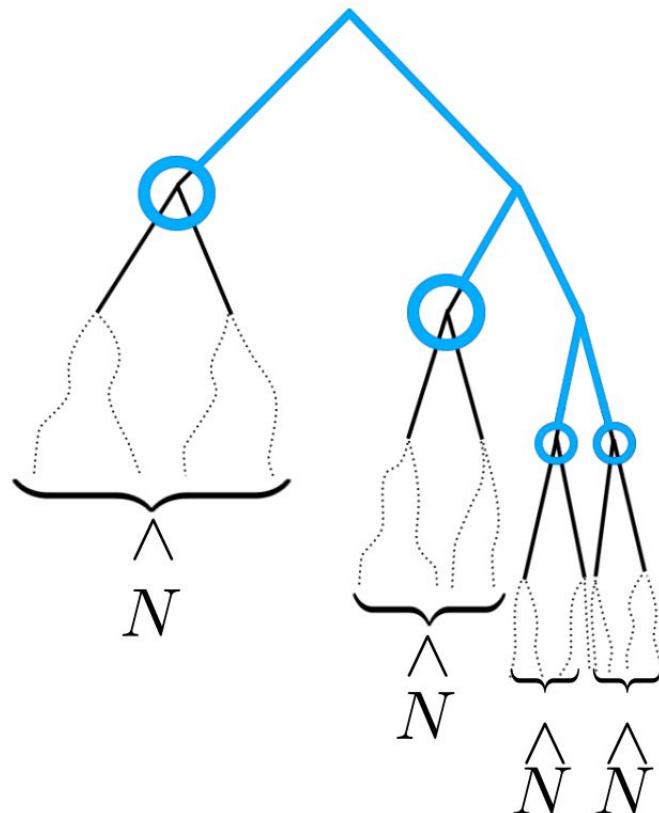


IO-friendly:

Исходные кубы на диске
последовательны
благодаря сортировке
Morton Codes

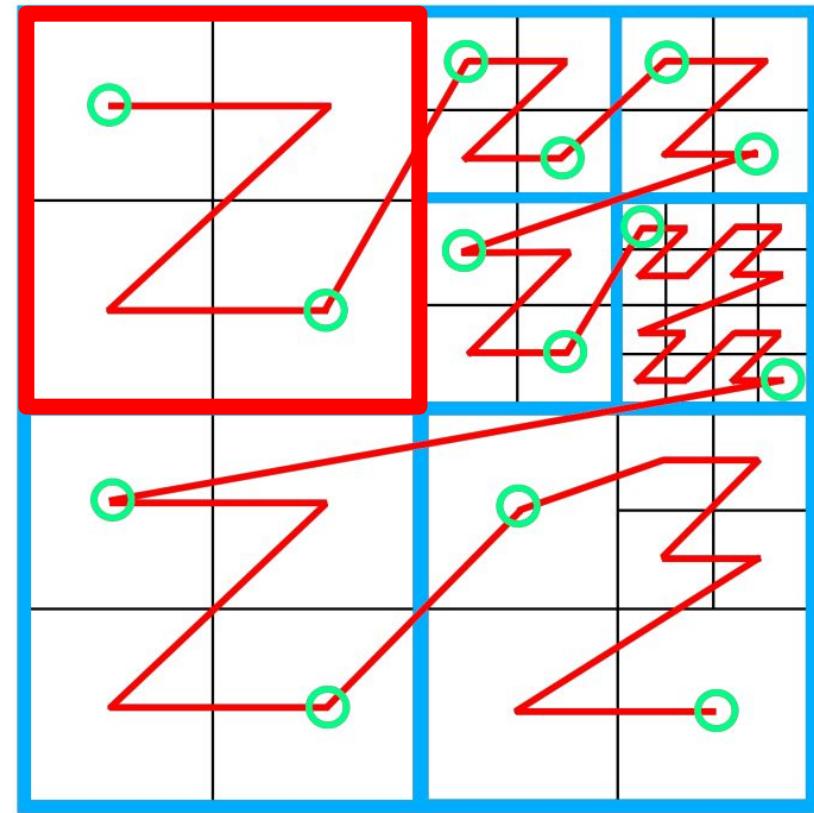
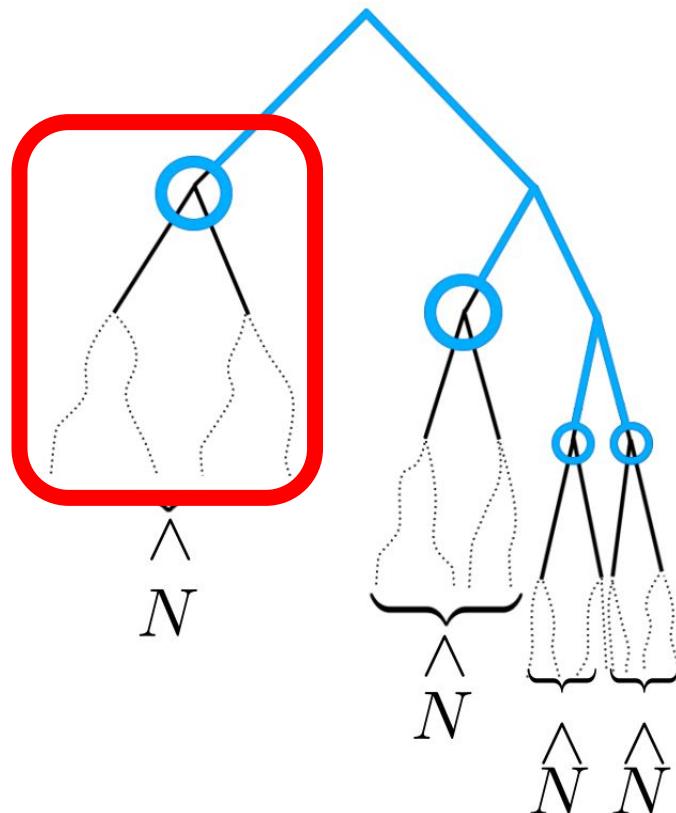
4) Out-of-Core реконструкция: карты глубины голосуют

Разбиение дерева на куски: построение верхушки дерева

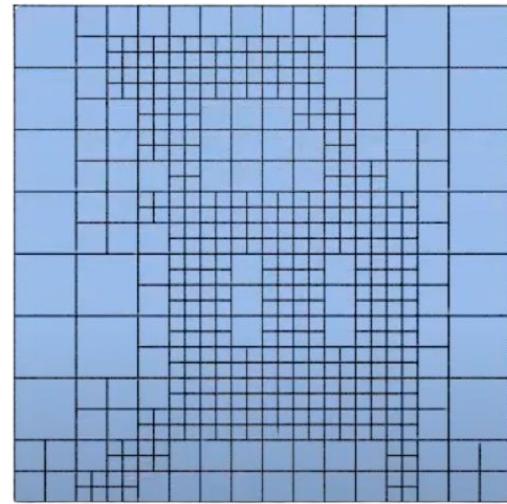
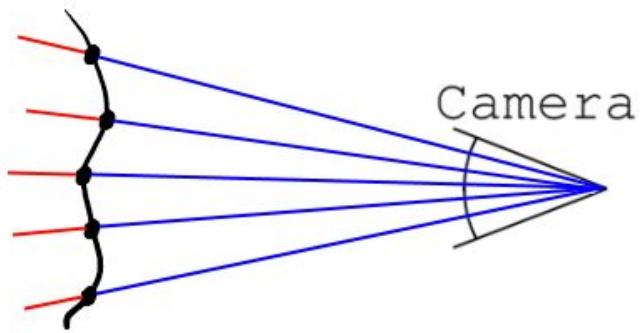


4) Out-of-Core реконструкция: карты глубины голосуют

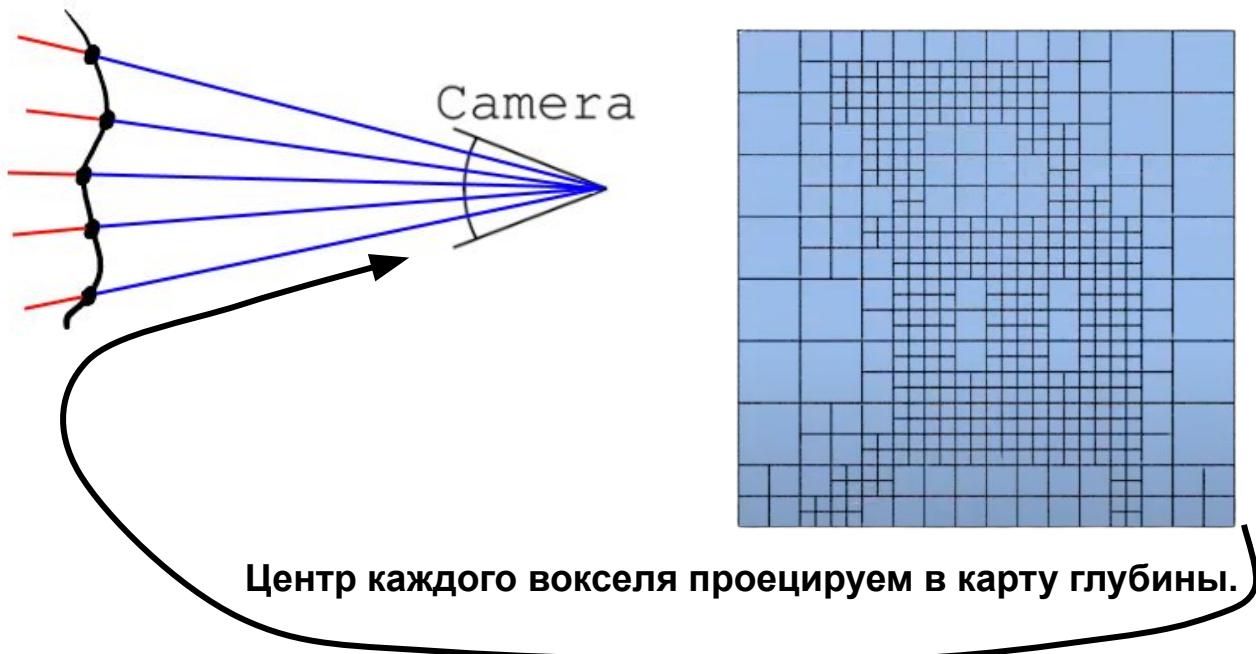
Разбиение дерева на куски: построение верхушки дерева



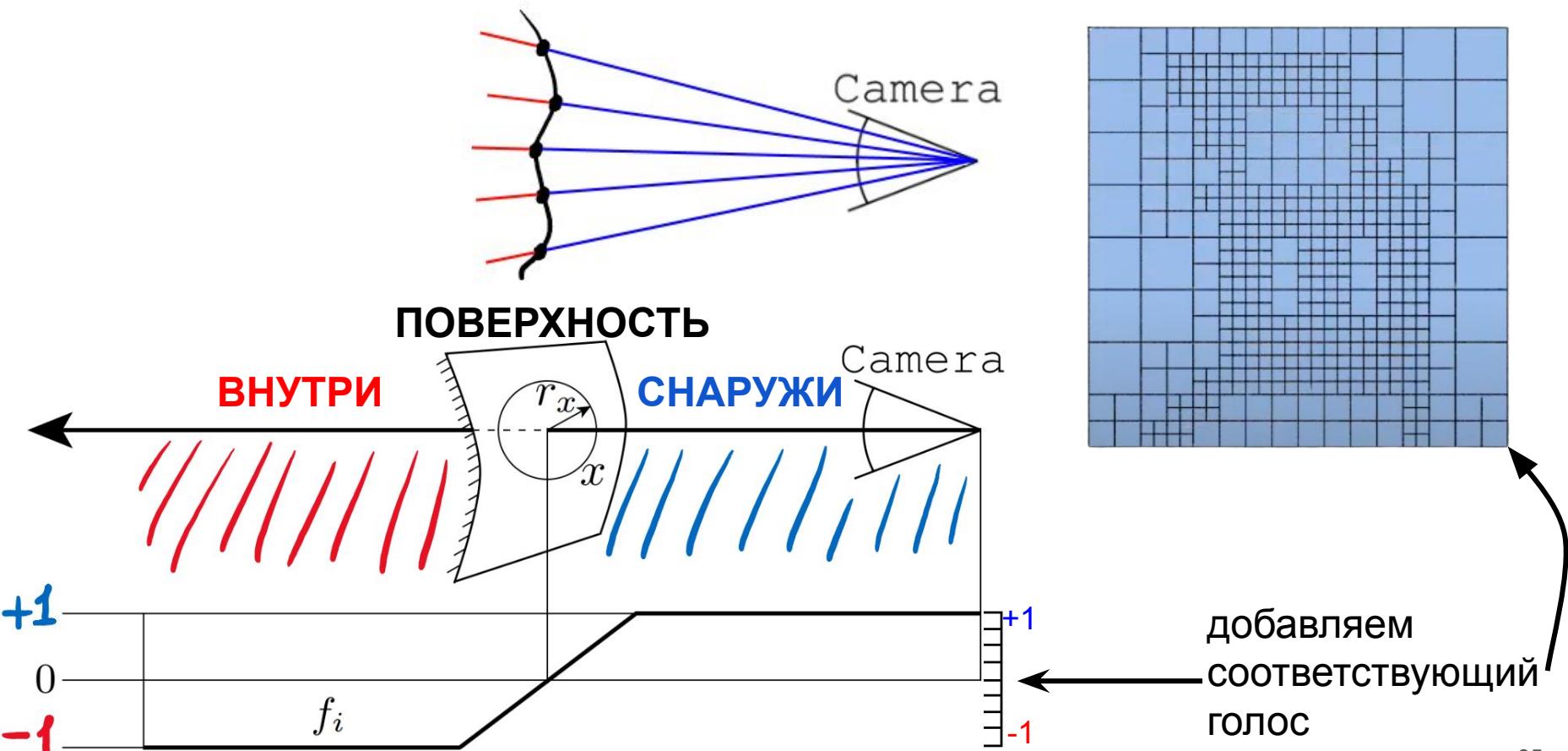
4) Out-of-Core реконструкция: карты глубины голосуют



4) Out-of-Core реконструкция: карты глубины голосуют

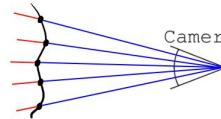
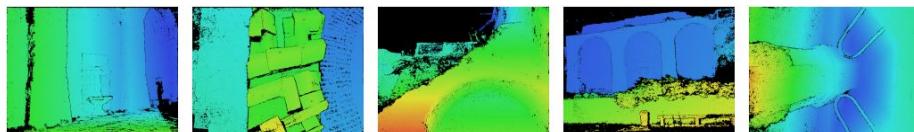


4) Out-of-Core реконструкция: карты глубины голосуют



4) Out-of-Core реконструкция: карты глубины голосуют

- 1) Есть множество карт глубины

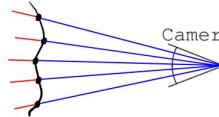
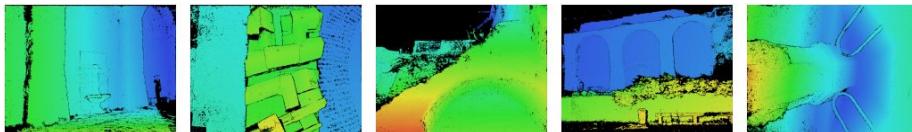


- 2) Множество точек всех карт глубины - порождает **адаптивное** октодерево (дискретизация пространства)
- 3) Балансируем октодерево 2:1 (по каждой стороне макс. два соседа)
- 4) Каждая карта глубины вносит индикаторные f_i голоса за voxели
- 5) Оптимизировали индикаторное поле (**много итераций + Coarse-to-Fine**)

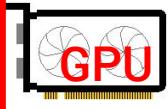

$$\text{TV-}L^1: E = \int_{\Omega} \left\{ |\nabla u| + \lambda \sum_{i \in \mathcal{I}(\vec{x})} w_i(\vec{x}) |u - f_i| \right\} d\vec{x}$$

5) Out-of-Core реконструкция: Coarse-to-Fine итерации

- 1) Есть множество карт глубины

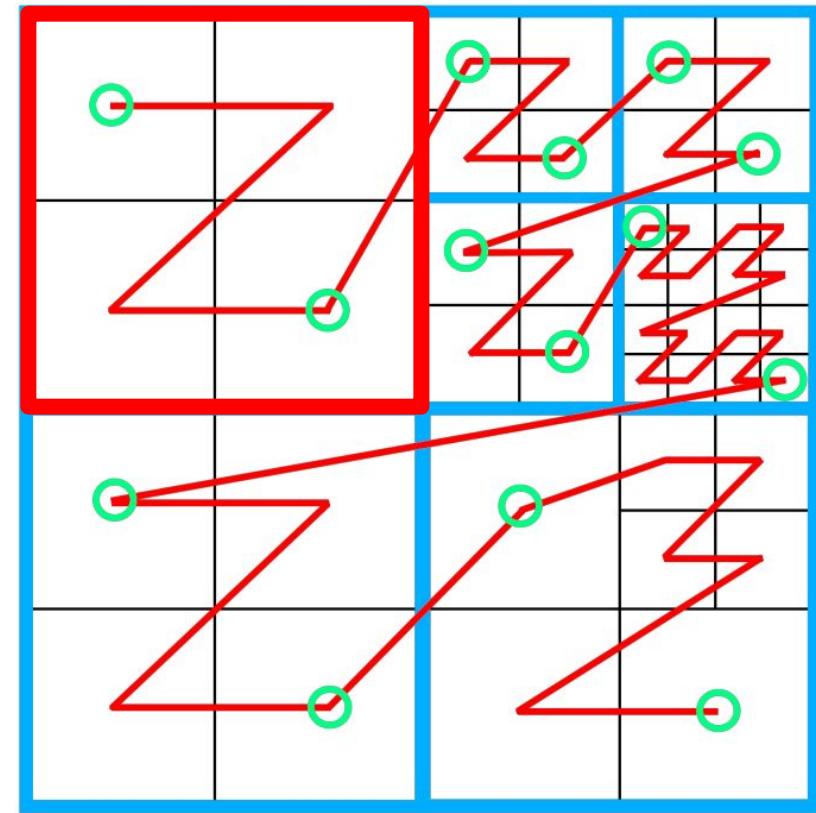
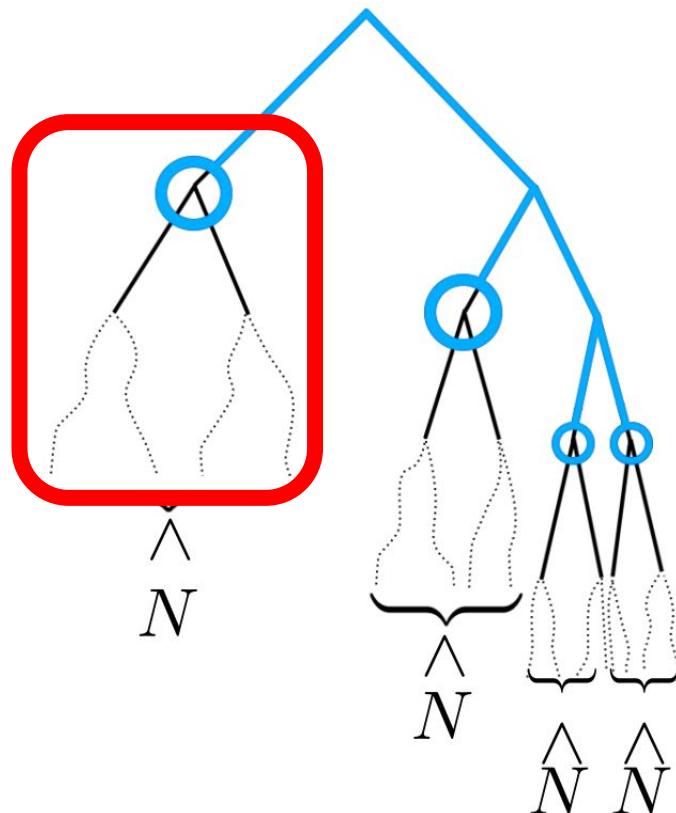


- 2) Множество точек всех карт глубины - порождает **адаптивное** октодерево (дискретизация пространства)
- 3) Балансируем октодерево 2:1 (по каждой стороне макс. два соседа)
- 4) Каждая карта глубины вносит индикаторные f_i голоса за voxели
- 5) Оптимизировали индикаторное поле (**много итераций + Coarse-to-Fine**)


$$TV-L^1: E = \int_{\Omega} \left\{ |\nabla u| + \lambda \sum_{i \in \mathcal{I}(\vec{x})} w_i(\vec{x}) |u - f_i| \right\} d\vec{x}$$

5) Out-of-Core реконструкция: Coarse-to-Fine итерации

Разбиение дерева на куски: переиспользуем верхушку



5) Out-of-Core реконструкция: Coarse-to-Fine итерации

Разбиение дерева на куски: переиспользуем верхушку



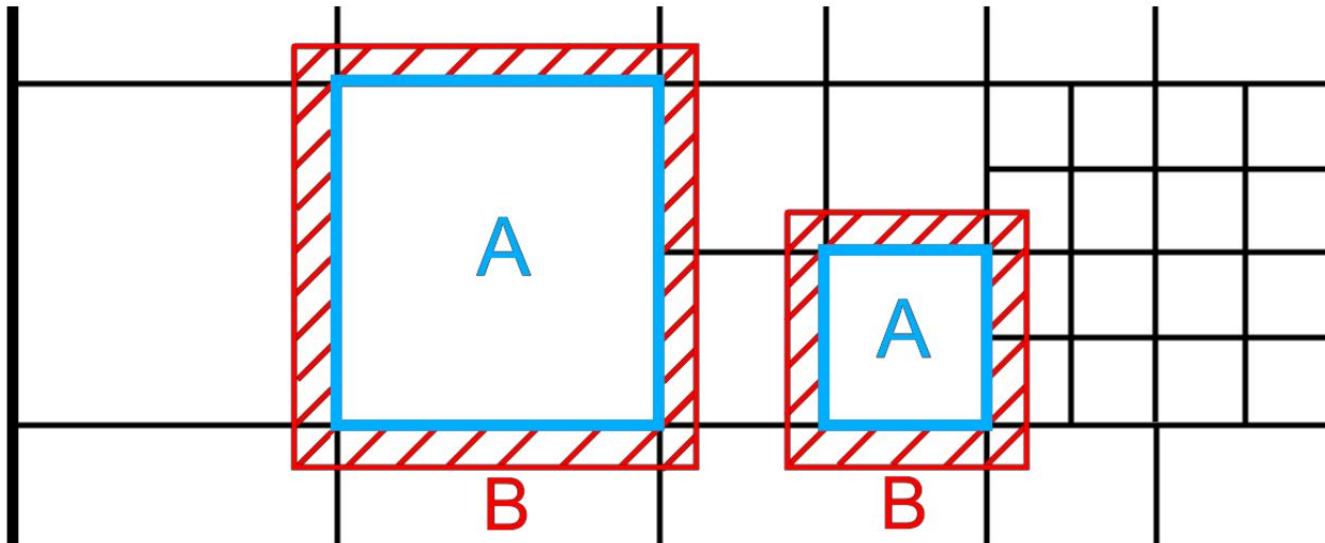
5) Out-of-Core реконструкция: Coarse-to-Fine итерации

Разбиение дерева на куски: переиспользуем верхушку



5) Out-of-Core реконструкция: Coarse-to-Fine итерации

Разбиение дерева на куски: переиспользуем верхушку

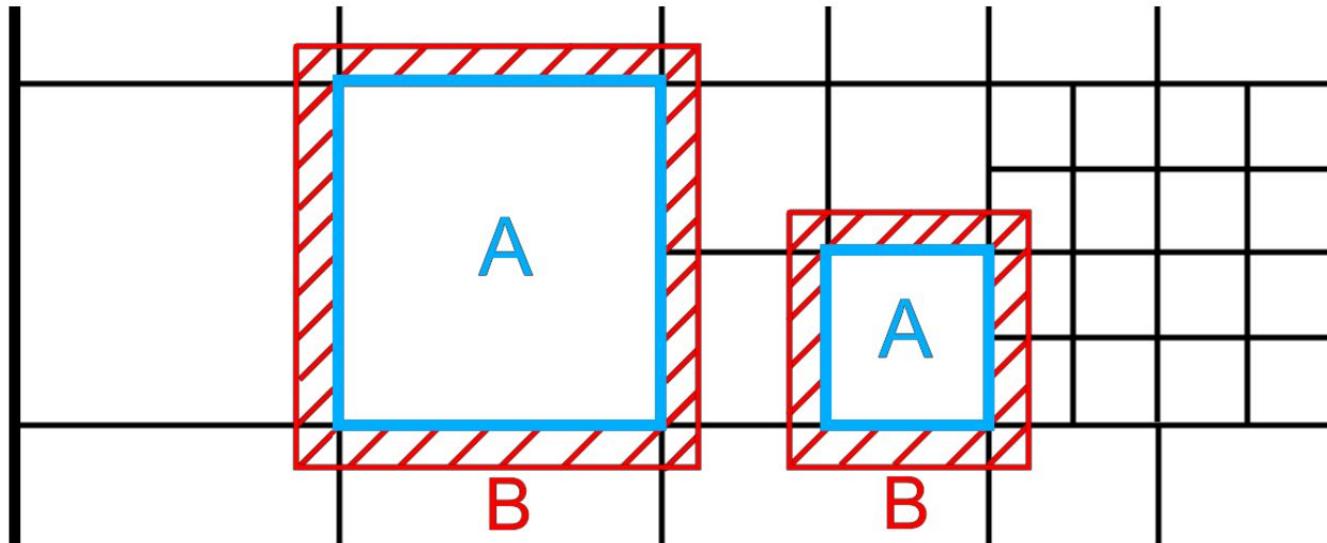


Заморозка границ!

Т.е. добавили в нашу минимизацию граничное условие.

5) Out-of-Core реконструкция: Coarse-to-Fine итерации

Разбиение дерева на куски: переиспользуем верхушку



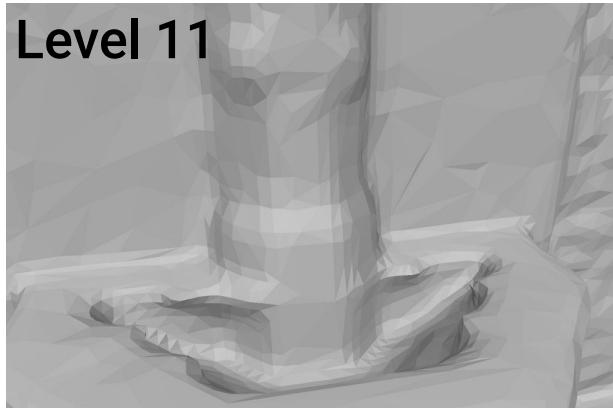
Заморозка границ!

Т.е. добавили в нашу минимизацию граничное условие.

Проблема курицы и яйца - мы не знаем результат соседа!

Откуда взять индикатор по периметру для заморозки?

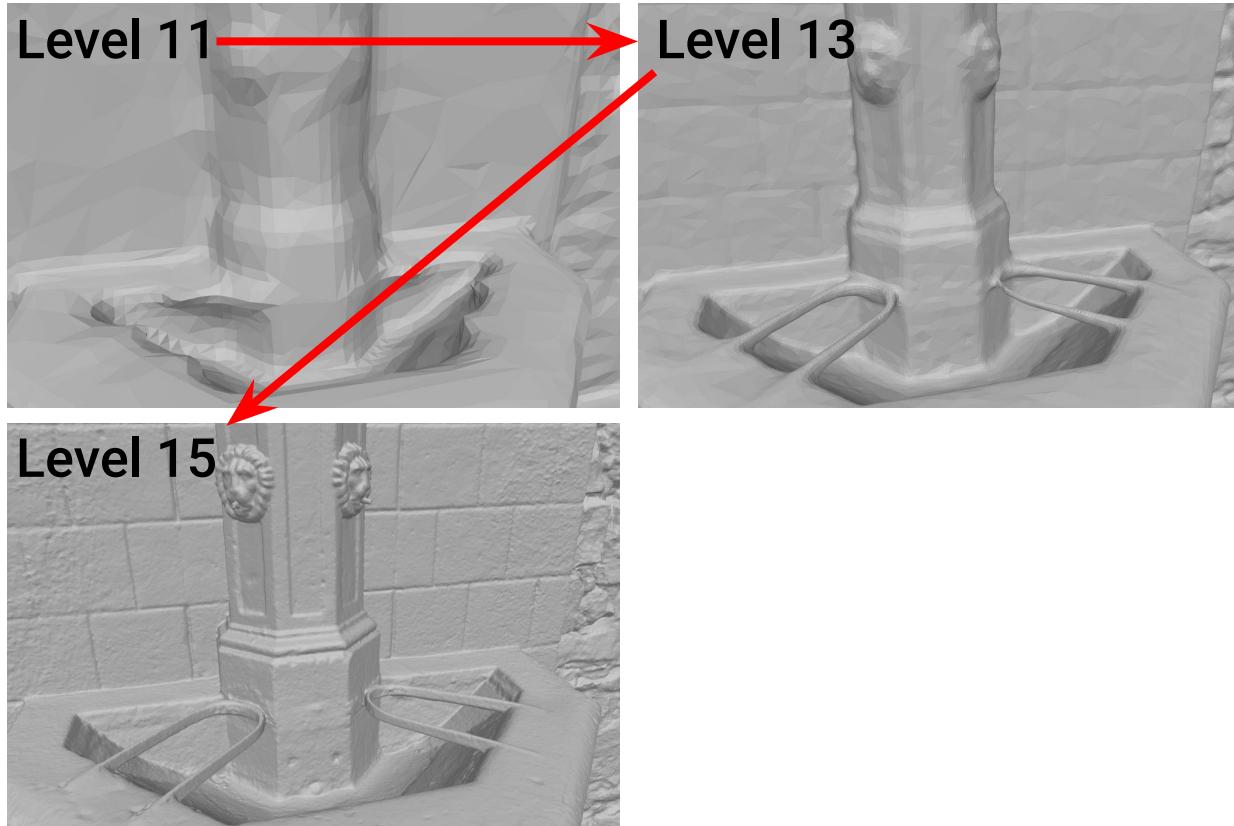
5) Out-of-Core реконструкция: Coarse-to-Fine итерации



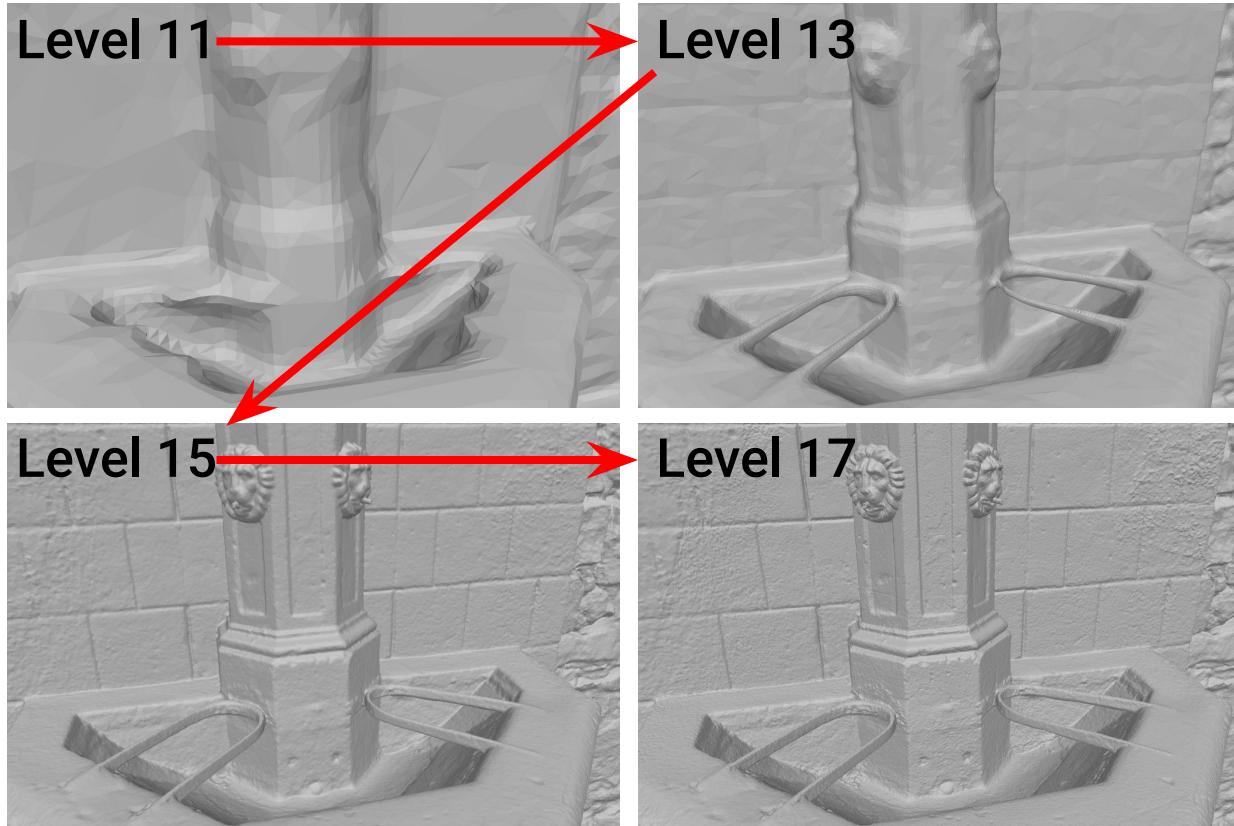
5) Out-of-Core реконструкция: Coarse-to-Fine итерации



5) Out-of-Core реконструкция: Coarse-to-Fine итерации

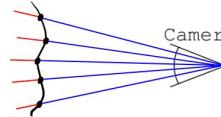
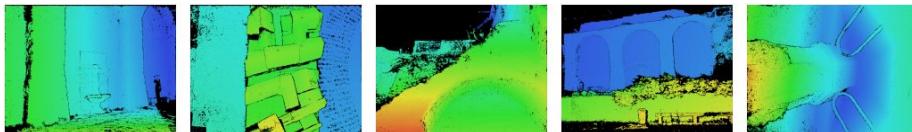


5) Out-of-Core реконструкция: Coarse-to-Fine итерации



5) Out-of-Core реконструкция: Coarse-to-Fine итерации

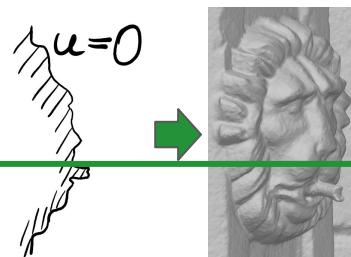
- 1) Есть множество карт глубины



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



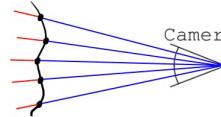
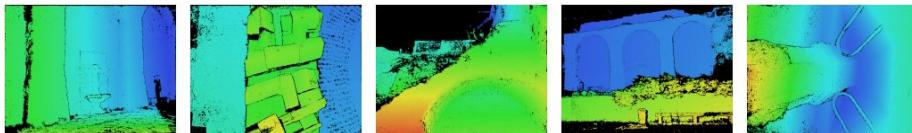
TV- L^1 :
$$E = \int_{\Omega} \left\{ |\nabla u| + \lambda \sum_{i \in \mathcal{I}(\vec{x})} w_i(\vec{x}) |u - f_i| \right\} d\vec{x}$$



- 6) Извлекли поверхность маршировкой кубов

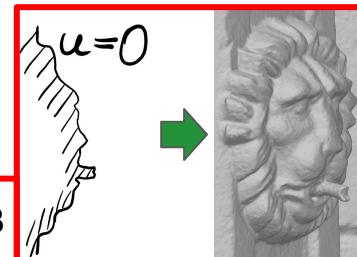
6) Out-of-Core реконструкция: маркировка кубов

- 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) Каждая карта глубины вносит индикаторные f_i голоса за воксели
- 5) Оптимизировали индикаторное поле (**много итераций + Coarse-to-Fine**)


$$TV-L^1: E = \int_{\Omega} \left\{ |\nabla u| + \lambda \sum_{i \in \mathcal{I}(\vec{x})} w_i(\vec{x}) |u - f_i| \right\} d\vec{x}$$

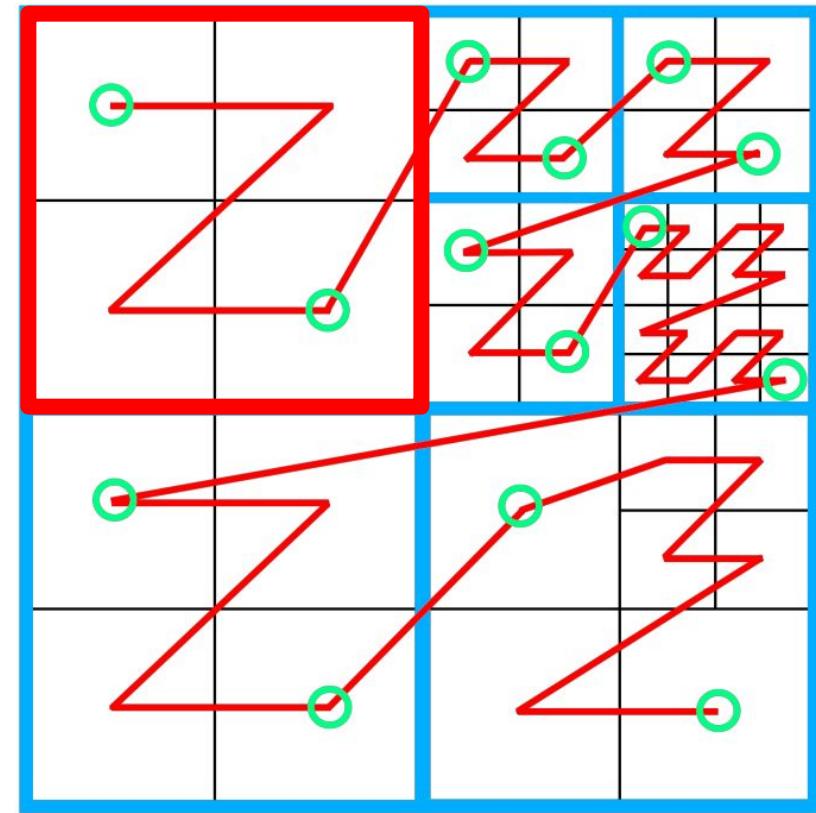
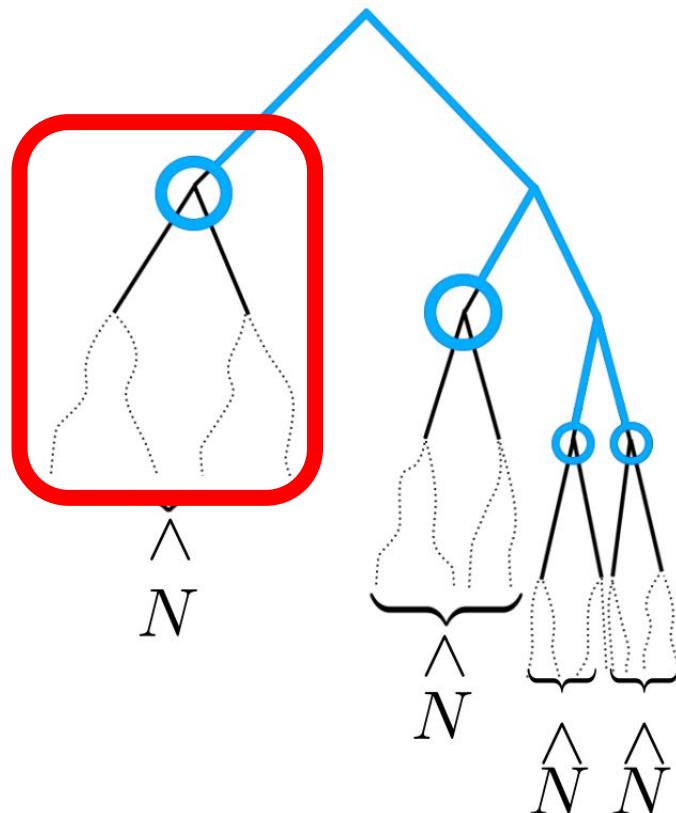


- 6) Извлекли поверхность маркировкой кубов

Как?

6) Out-of-Core реконструкция: маркировка кубов

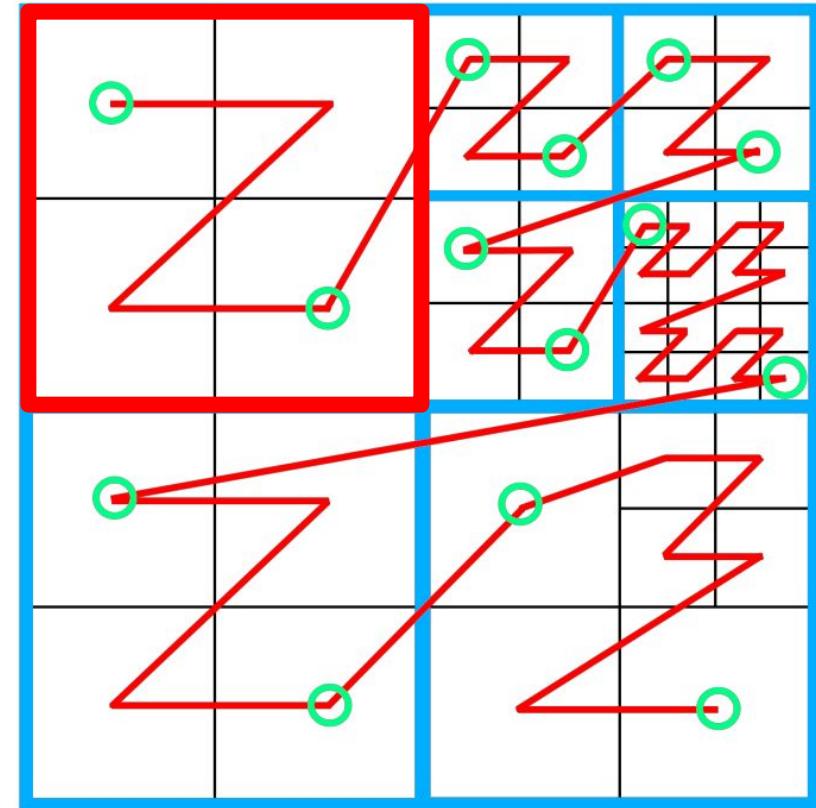
Разбиение дерева на куски: переиспользуем верхушку



6) Out-of-Core реконструкция: маркировка кубов

Разбиение дерева на куски: переиспользуем верхушку

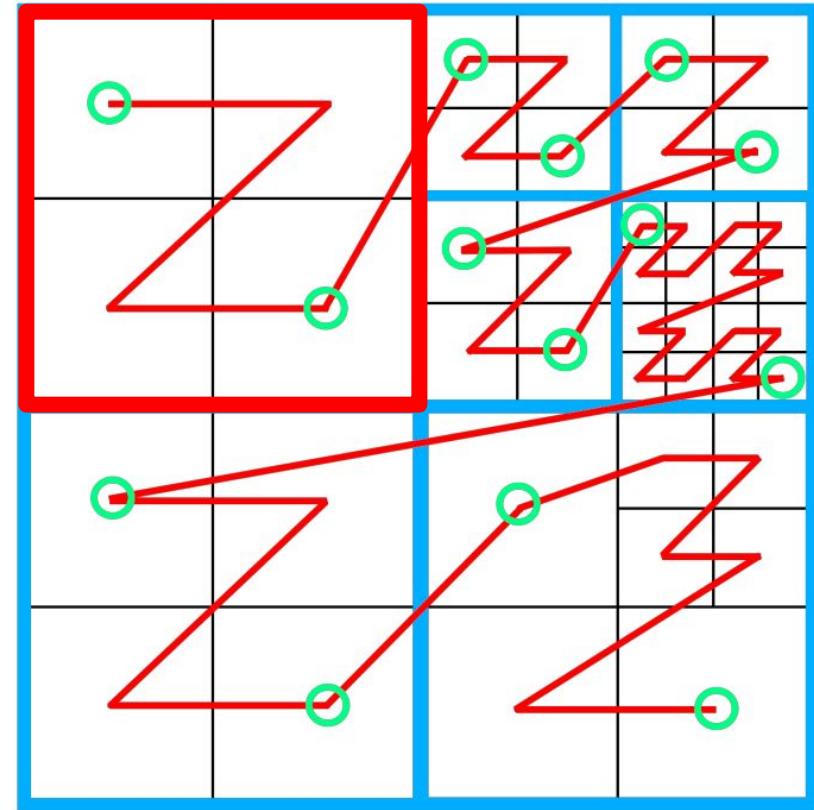
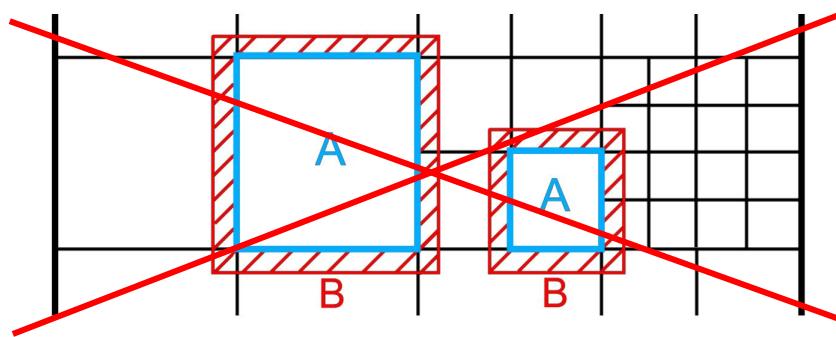
Могут ли быть трещины?



6) Out-of-Core реконструкция: маркировка кубов

Разбиение дерева на куски: переиспользуем верхушку

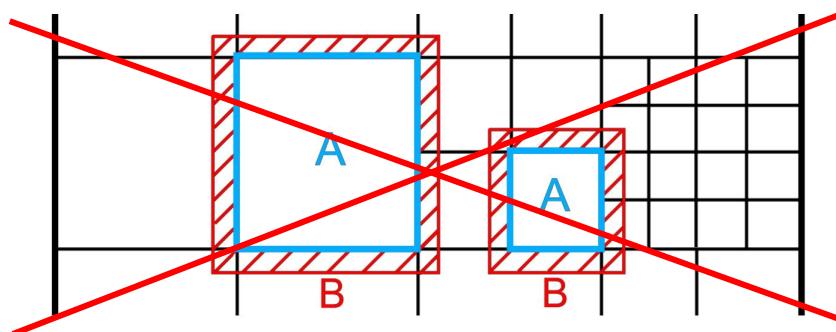
Могут ли быть трещины
если не делать заморозку?



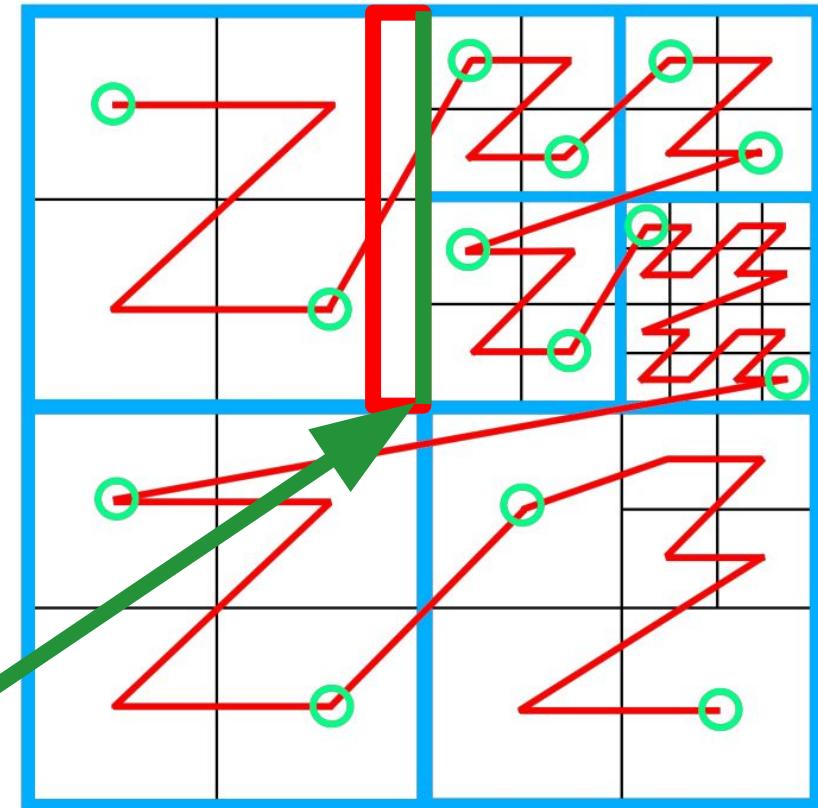
6) Out-of-Core реконструкция: маркировка кубов

Разбиение дерева на куски: переиспользуем верхушку

Может тогда заморозка не нужна?



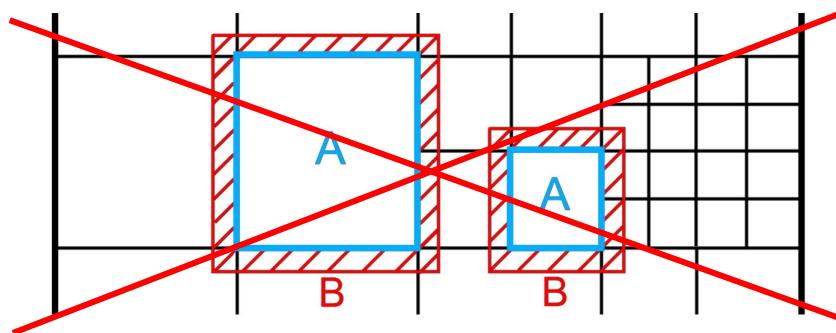
Трещин быть не может т.к.
маркировка кубов по одним и тем
же значениям индикатора **на стыке**.
Значит вершины одни и те же!



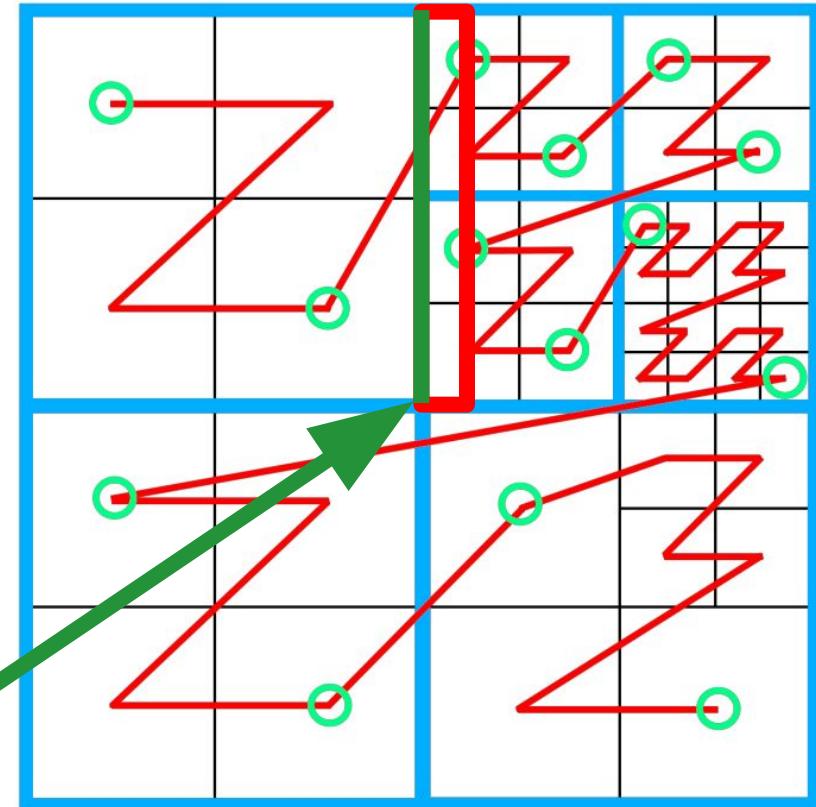
6) Out-of-Core реконструкция: маркировка кубов

Разбиение дерева на куски: переиспользуем верхушку

Может тогда заморозка не нужна?



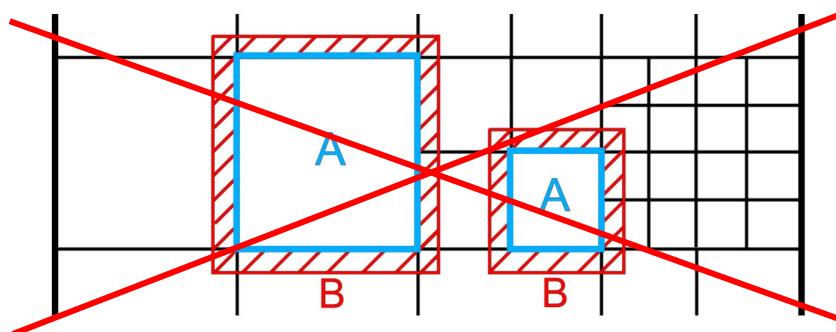
Трещин быть не может т.к.
маркировка кубов по одним и тем
же значениям индикатора **на стыке**.
Значит вершины одни и те же!



6) Out-of-Core реконструкция: маркировка кубов

Разбиение дерева на куски: переиспользуем верхушку

Может тогда заморозка не нужна?
Без нее могут быть резкие скачки



Трещин быть не может т.к.
маркировка кубов по одним и тем
же значениям индикатора **на стыке**.
Значит вершины одни и те же!

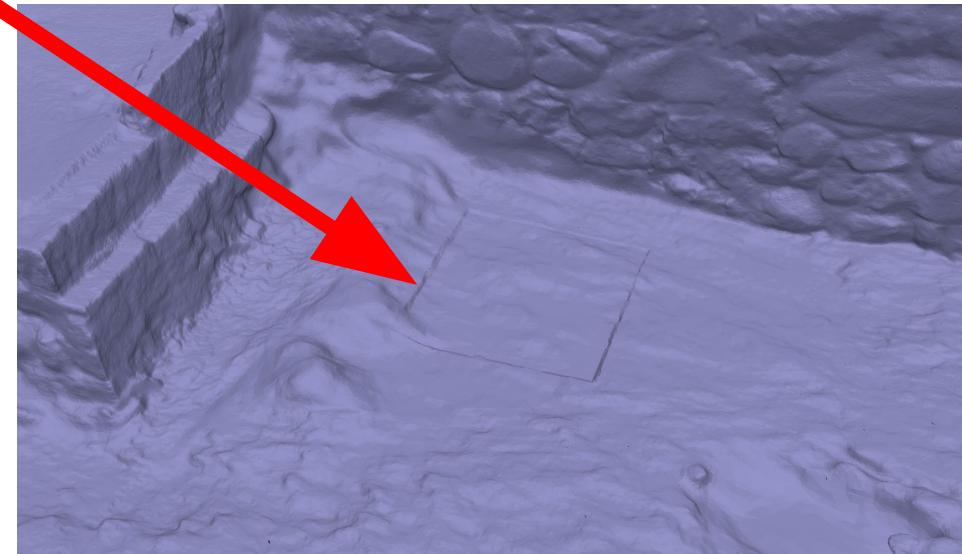
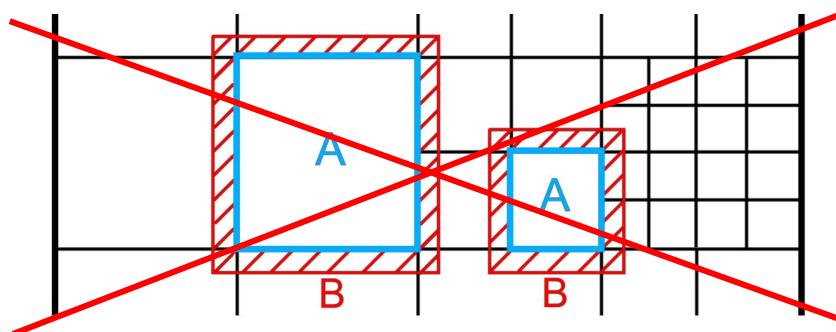


6) Out-of-Core реконструкция: маркировка кубов

Разбиение дерева на куски: переиспользуем верхушку

Может тогда заморозка не нужна?

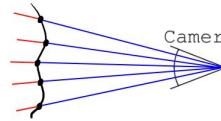
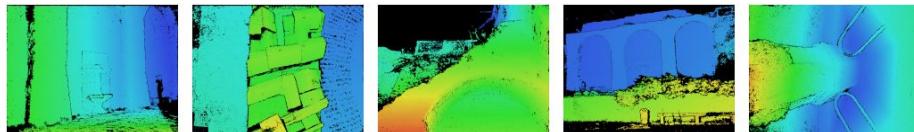
Без нее могут быть резкие скачки



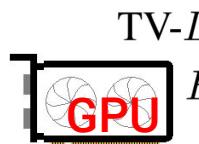
Трещин быть не может т.к.
маркировка кубов по одним и тем
же значениям индикатора **на стыке**.
Значит вершины одни и те же!

6) Out-of-Core реконструкция: маркировка кубов

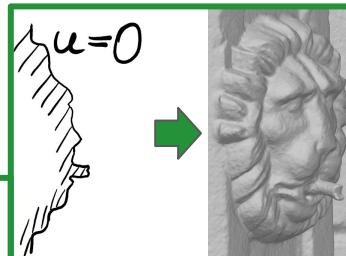
- 1) Есть множество карт глубины



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

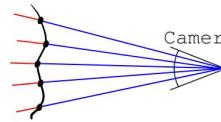
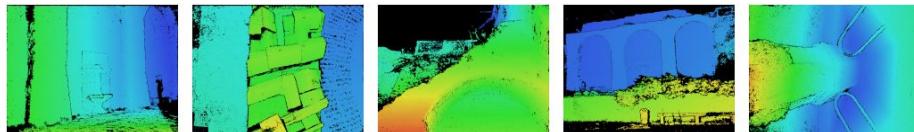

$$\text{TV-}L^1: E = \int_{\Omega} \left\{ |\nabla u| + \lambda \sum_{i \in \mathcal{I}(\vec{x})} w_i(\vec{x}) |u - f_i| \right\} d\vec{x}$$

- 6) Извлекли поверхность маркировкой кубов

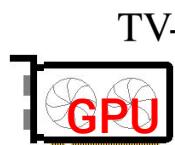


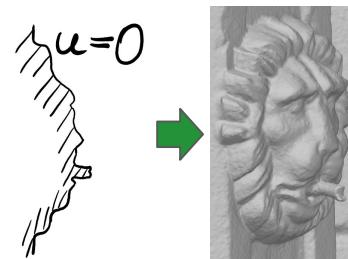
6) Out-of-Core реконструкция: маркировка кубов

- 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) Каждая карта глубины вносит индикаторные f_i голоса за воксели
- 5) Оптимизировали индикаторное поле (**много итераций + Coarse-to-Fine**)


$$\text{TV-}L^1: E = \int_{\Omega} \left\{ |\nabla u| + \lambda \sum_{i \in \mathcal{I}(\vec{x})} w_i(\vec{x}) |u - f_i| \right\} d\vec{x}$$

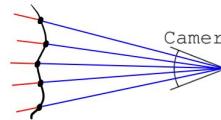
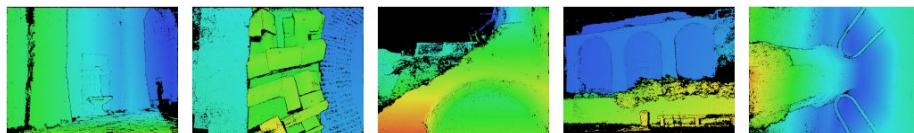


- 6) Извлекли поверхность маркировкой кубов

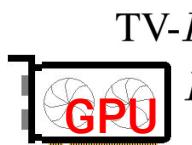
Получили миллиарды треугольников!

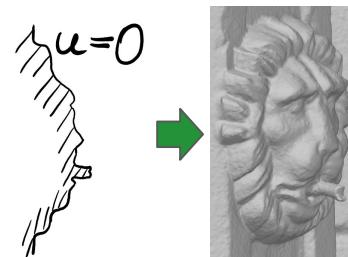
7) Out-of-Core реконструкция: упрощение геометрии

- 1) Есть множество карт глубины



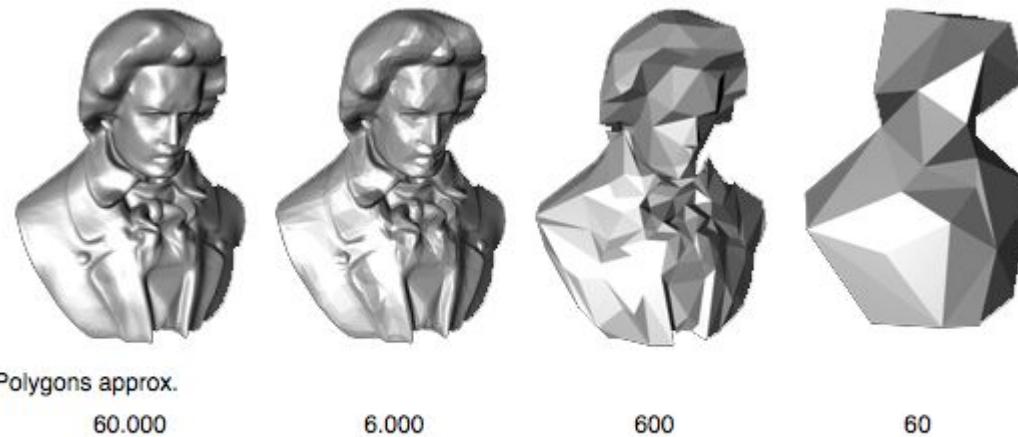
- 2) Множество точек всех карт глубины - порождает **адаптивное октодерево** (дискретизация пространства)
- 3) Балансируем октодерево 2:1 (по каждой стороне макс. два соседа)
- 4) Каждая карта глубины вносит индикаторные f_i голоса за воксели
- 5) Оптимизировали индикаторное поле (**много итераций +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


$$\text{TV-}L^1: E = \int_{\Omega} \left\{ |\nabla u| + \lambda \sum_{i \in \mathcal{I}(\vec{x})} w_i(\vec{x}) |u - f_i| \right\} d\vec{x}$$



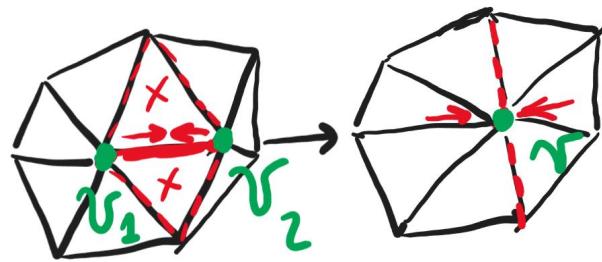
- 6) Извлекли поверхность маршировкой кубов
- 7) Упрощение полигональной геометрии

7) Out-of-Core реконструкция: упрощение геометрии



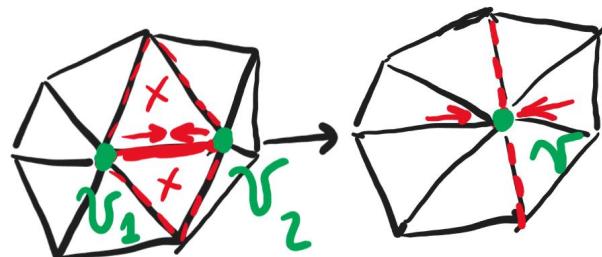
7) Out-of-Core реконструкция: упрощение геометрии

QSLim decimation: edge collapse (схлопывание ребер)



7) Out-of-Core реконструкция: упрощение геометрии

QSLim decimation: edge collapse (схлопывание ребер)



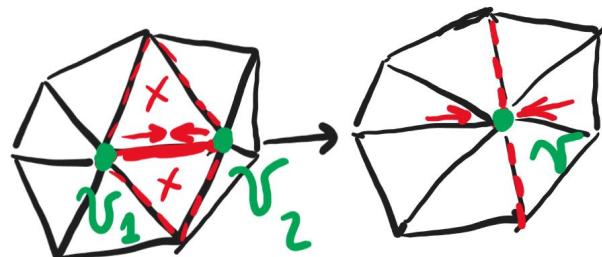
Quadric Error Metric (**QEM**)

На каждую вершину: 4×4 матрица Q

Ошибка каждой вершины: $\mathbf{v}^T Q \mathbf{v}$

7) Out-of-Core реконструкция: упрощение геометрии

QSLIM decimation: edge collapse (схлопывание ребер)

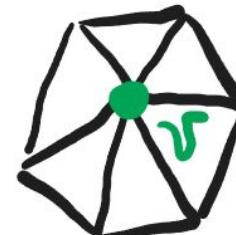


Quadric Error Metric (**QEM**)

На каждую вершину: 4×4 матрица Q

Ошибка каждой вершины: $\mathbf{v}^T Q \mathbf{v}$

$$Q = ?$$

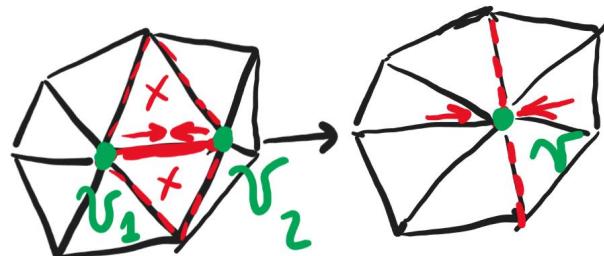


σ = пересечение плоскостей

$$\text{error} = \sum \text{plane dist}^2$$

7) Out-of-Core реконструкция: упрощение геометрии

QSLIM decimation: edge collapse (схлопывание ребер)



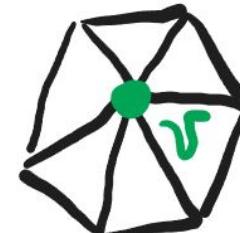
$(v_1, v_2) \xrightarrow{\text{edge collapse}} v = ???$

Quadric Error Metric (**QEM**)

На каждую вершину: 4×4 матрица Q

Ошибка каждой вершины: $v^T Q v$

$$Q = ?$$

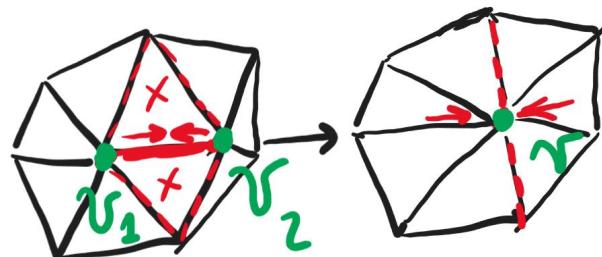


\cap = пересечение плоскостей

$$\text{error} = \sum \text{plane dist}^2$$

7) Out-of-Core реконструкция: упрощение геометрии

QSLIM decimation: edge collapse (схлопывание ребер)



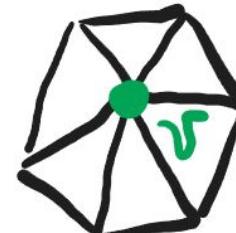
$$(\mathbf{v}_1, \mathbf{v}_2) \xrightarrow{\text{edge collapse}} \mathbf{v} = \underset{\mathbf{v}}{\operatorname{argmin}} \mathbf{v}^T \bar{Q} \mathbf{v}$$
$$\bar{Q} = ???$$

Quadric Error Metric (**QEM**)

На каждую вершину: 4x4 матрица \mathbf{Q}

Ошибка каждой вершины: $\mathbf{v}^T \bar{Q} \mathbf{v}$

$$\bar{Q} = ?$$

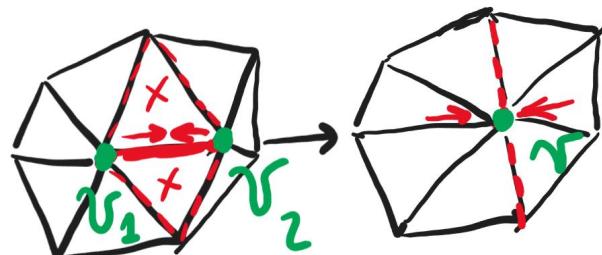


\mathbf{v} = пересечение плоскостей

$$\text{error} = \sum \text{plane dist}^2$$

7) Out-of-Core реконструкция: упрощение геометрии

QSLIM decimation: edge collapse (схлопывание ребер)



$$\begin{pmatrix} \mathbf{v}_1 & \mathbf{v}_2 \end{pmatrix} \xrightarrow{\text{edge collapse}} \mathbf{v} = \underset{\mathbf{v}}{\operatorname{argmin}} \mathbf{v}^T \bar{Q} \mathbf{v}$$
$$\bar{Q} = Q_1 + Q_2$$

Quadric Error Metric (**QEM**)

На каждую вершину: 4x4 матрица \mathbf{Q}

Ошибка каждой вершины: $\mathbf{v}^T \bar{Q} \mathbf{v}$

$$Q = ?$$

\mathbf{v} = пересечение плоскостей

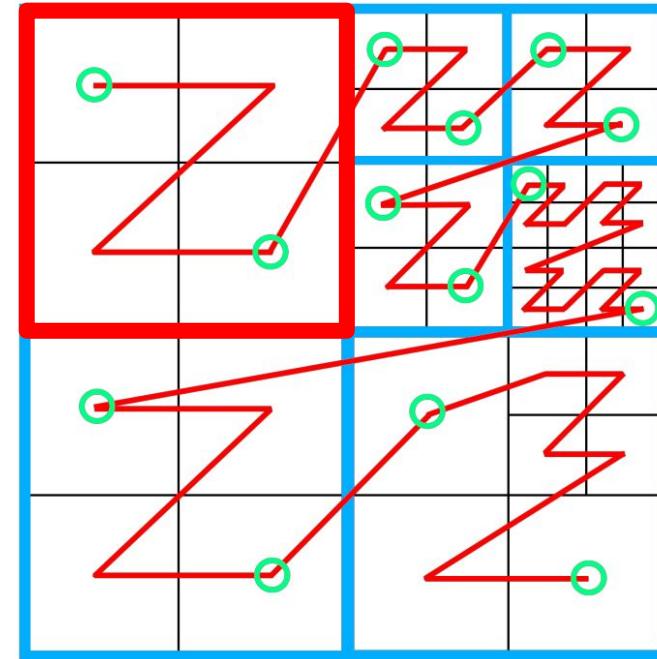
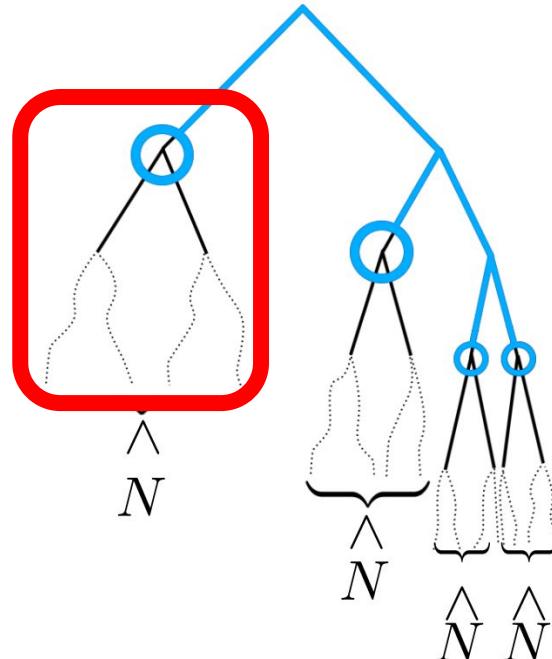
$$\text{error} = \sum \text{plane dist}^2$$

7) Out-of-Core реконструкция: упрощение геометрии

QSLim decimation: edge collapse (схлопывание ребер)

Не будет ли проблем на границах?

Не появятся ли трещины?



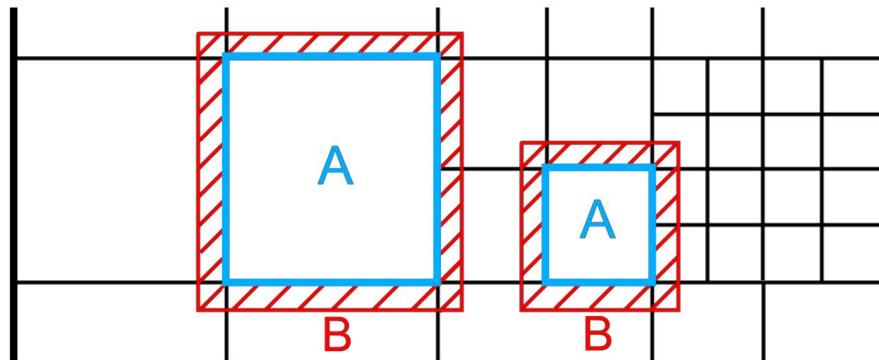
7) Out-of-Core реконструкция: упрощение геометрии

QSLim decimation: edge collapse (схлопывание ребер)

Не будет ли проблем на границах?

Не появятся ли трещины?

Не будем упрощать ребра на границе!



7) Out-of-Core реконструкция: упрощение геометрии

QSLim decimation: edge collapse (схлопывание ребер)

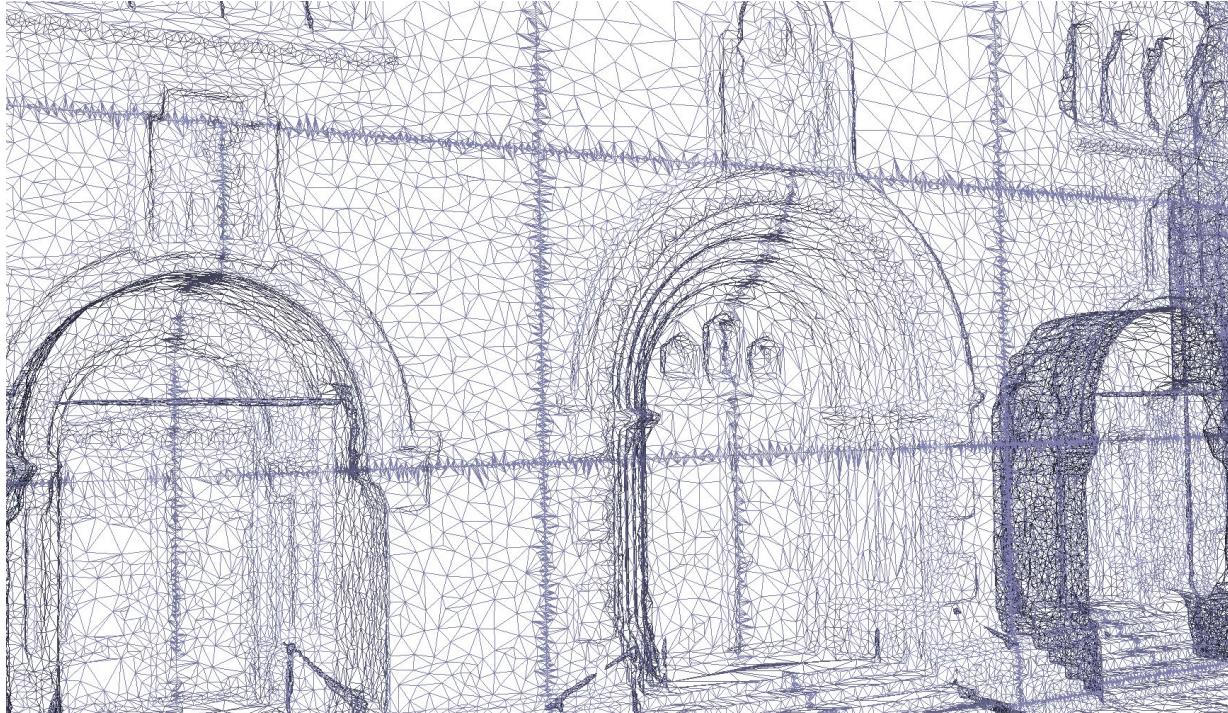
Не будем упрощать ребра на границе!



7) Out-of-Core реконструкция: упрощение геометрии

QSLim decimation: edge collapse (схлопывание ребер)

Не будем упрощать ребра на границе!

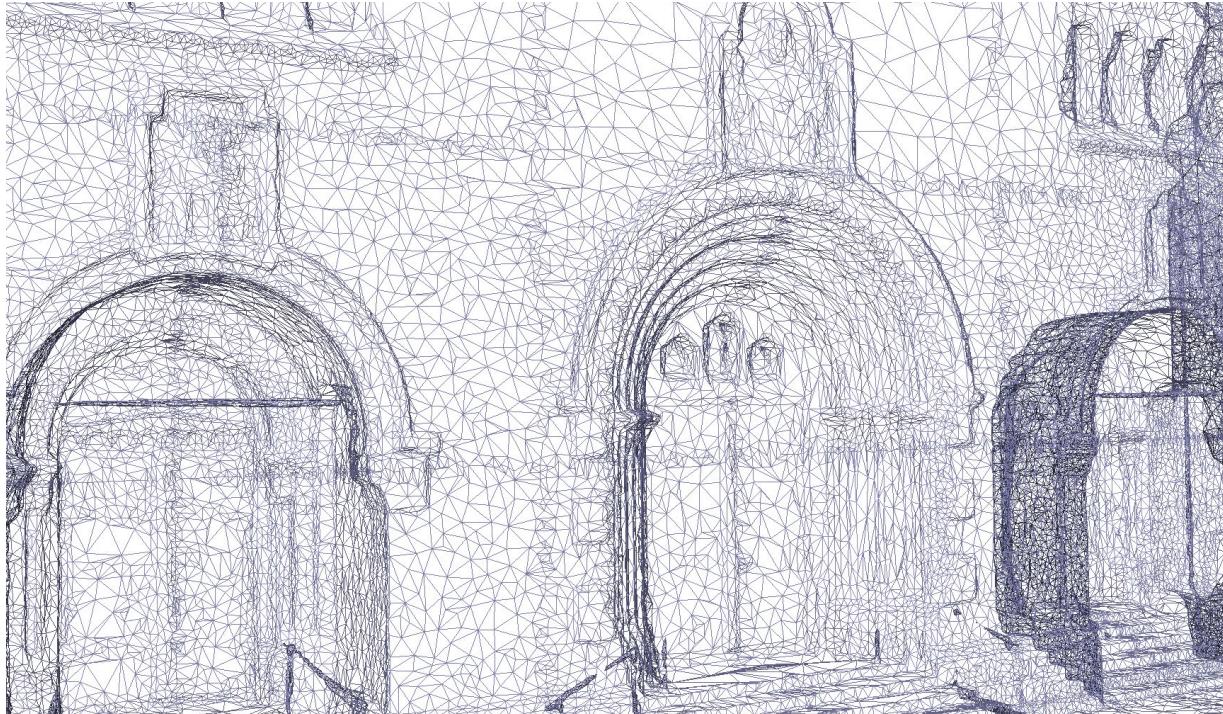


Simplifying surfaces with color and texture using quadric error metrics, Garland et. al., 1998 129

7) Out-of-Core реконструкция: упрощение геометрии

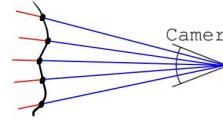
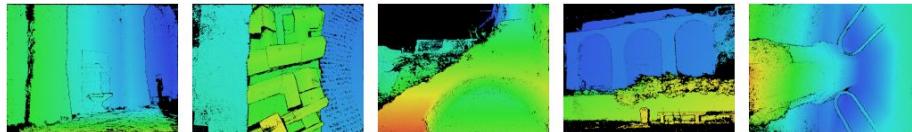
QSLim decimation: edge collapse (схлопывание ребер)

Не будем упрощать ребра на границе! И в конце упростим границу!

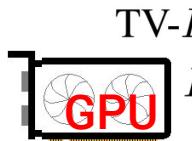


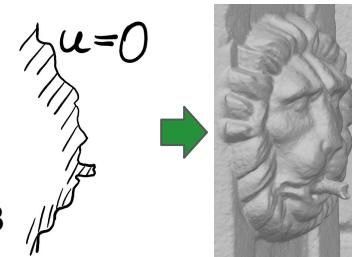
Out-of-Core реконструкция 3D модели

- 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) Каждая карта глубины вносит индикаторные f_i голоса за воксели
- 5) Оптимизировали индикаторное поле (**много итераций + Coarse-to-Fine**)


$$\text{TV-}L^1: E = \int_{\Omega} \left\{ |\nabla u| + \lambda \sum_{i \in \mathcal{I}(\vec{x})} w_i(\vec{x}) |u - f_i| \right\} d\vec{x}$$



- 6) Извлекли поверхность маршировкой кубов
- 7) Упрощение полигональной геометрии

Out-of-Core реконструкция 3D модели

Свойства алгоритма

Качественные свойства:

- **Scale-diverse** (адаптивное разрешение)
- Сильные свойства фильтрации выбросов (благодаря лучам видимости)
- Бесшовная поверхность

Out-of-Core реконструкция 3D модели

Свойства алгоритма

Качественные свойства:

- **Scale-diverse** (адаптивное разрешение)
- Сильные свойства фильтрации выбросов (благодаря лучам видимости)
- Бесшовная поверхность

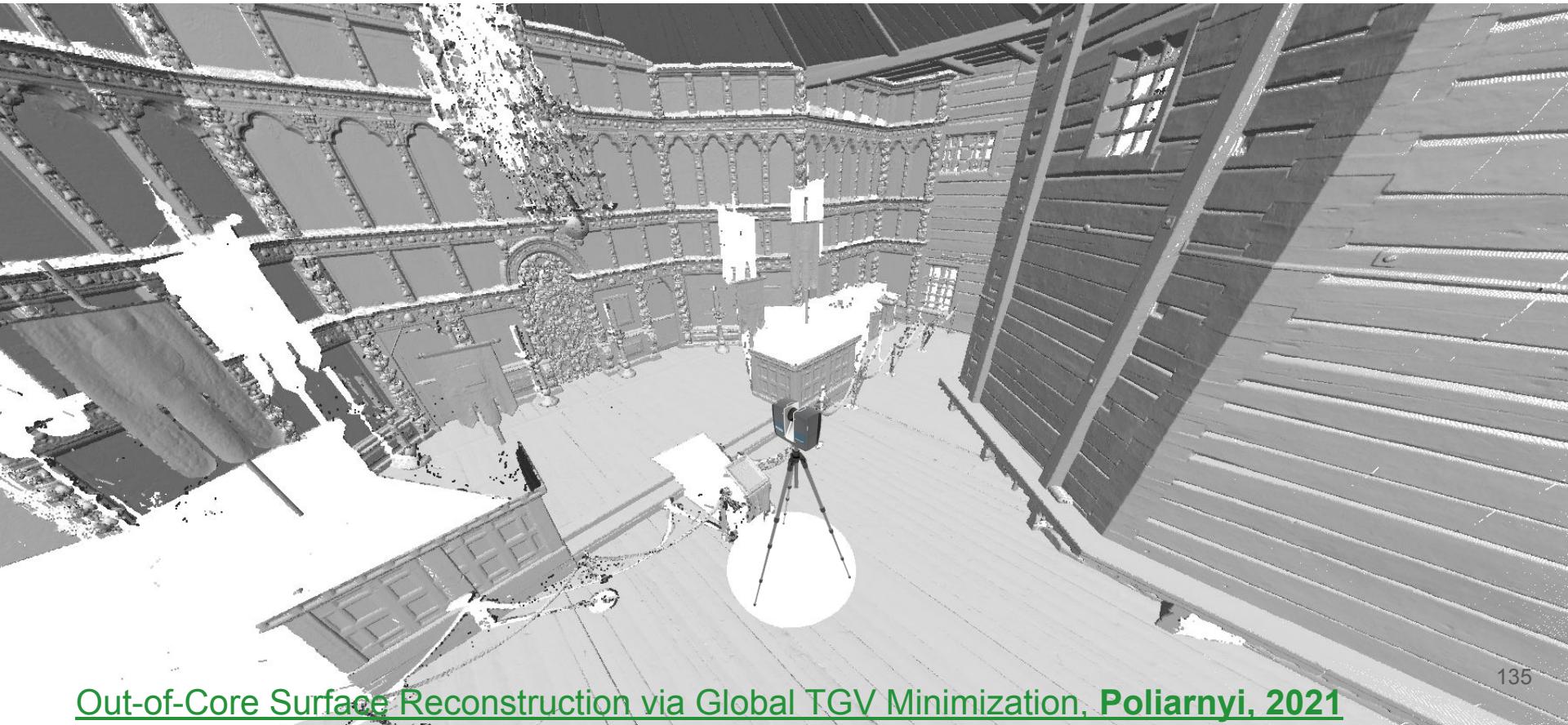
Практические свойства:

- **Out-of-Core** (ограниченные требования к RAM)
- Быстрый (ускорение на **GPU** & **IO-friendly**)
- Можно запускать на **клUSTERе**
- Поддерживает **LIDAR** сканы и спутниковые снимки

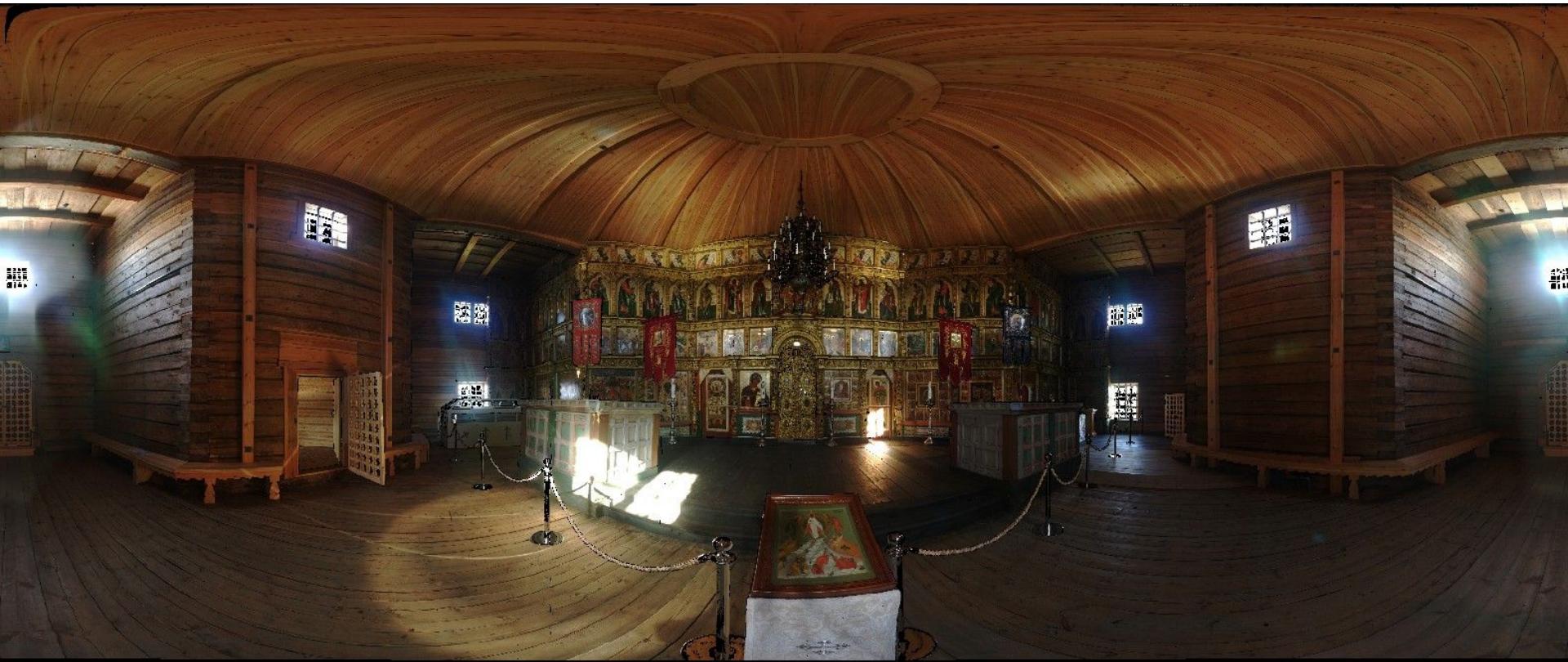
Входные данные: Terrestrial LIDAR



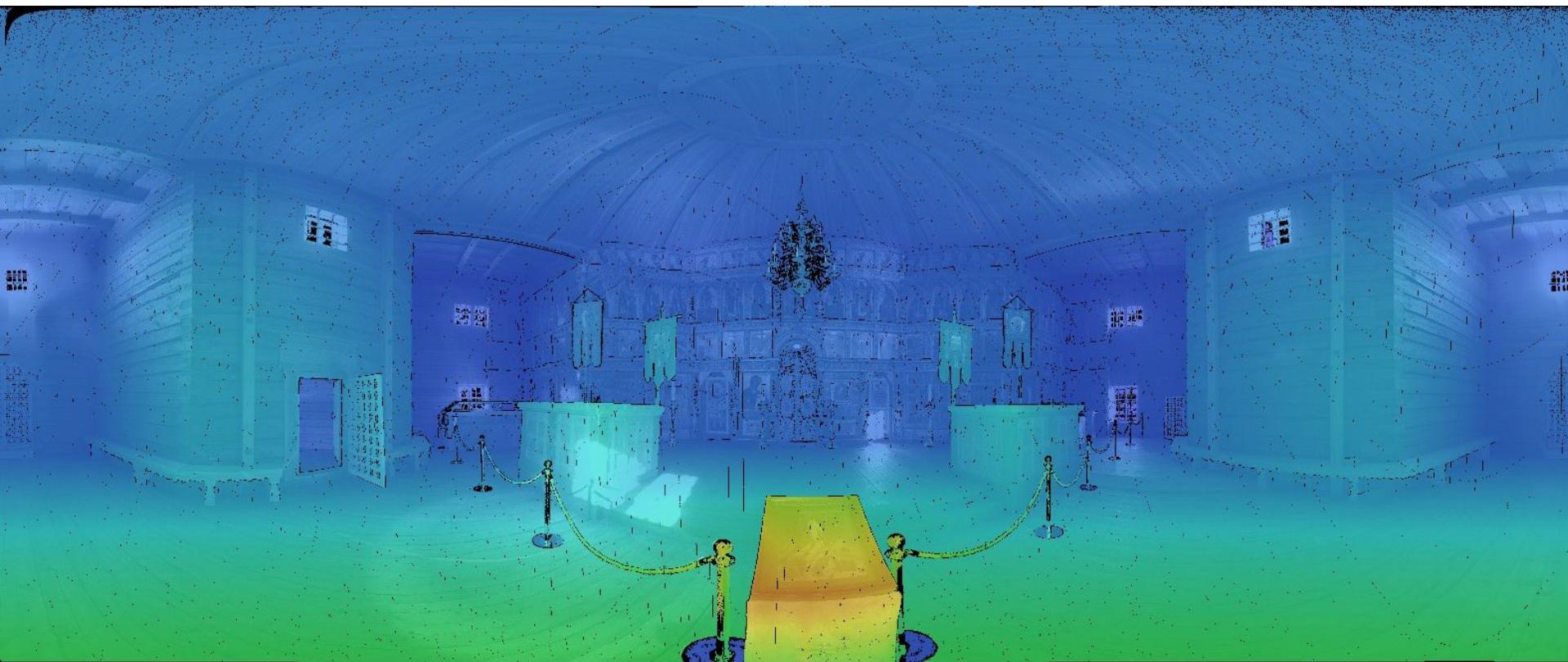
Входные данные: Terrestrial LIDAR

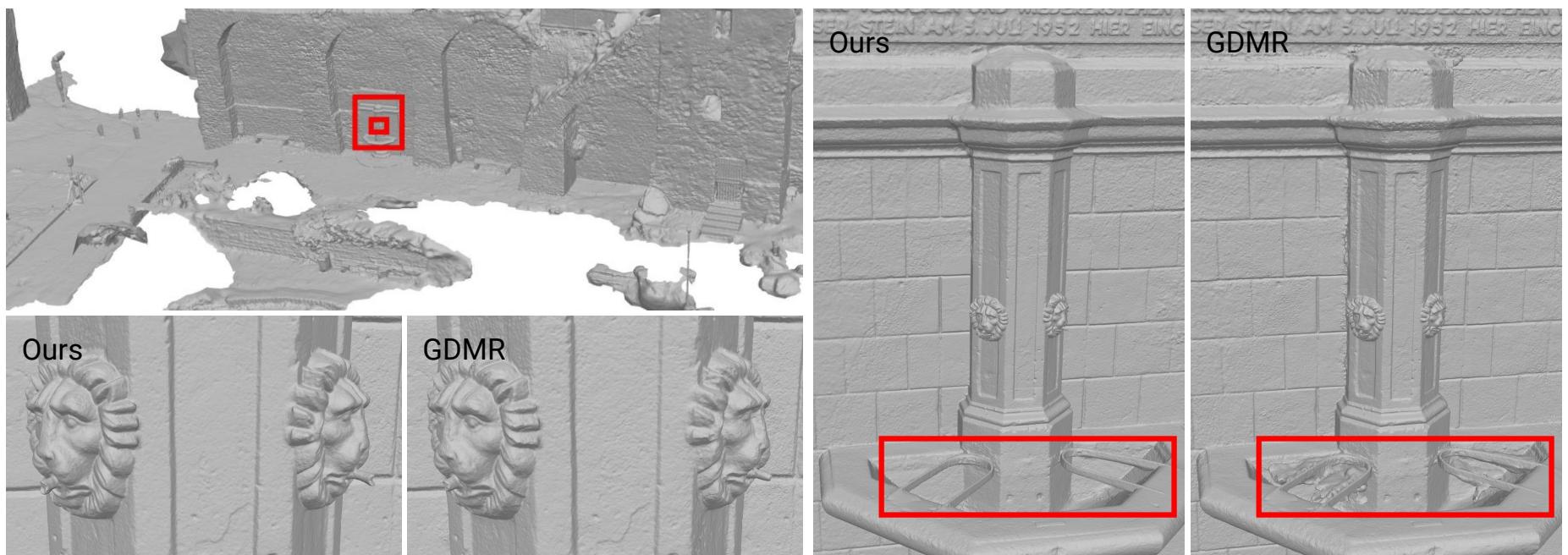


Terrestrial LIDAR = карта глубины сферической камеры



Terrestrial LIDAR = карта глубины сферической камеры

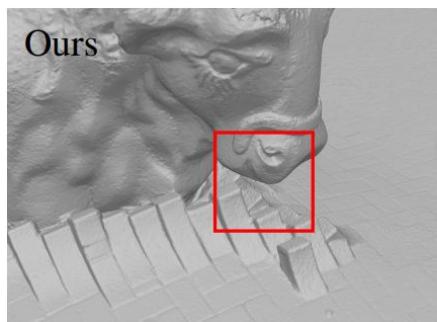




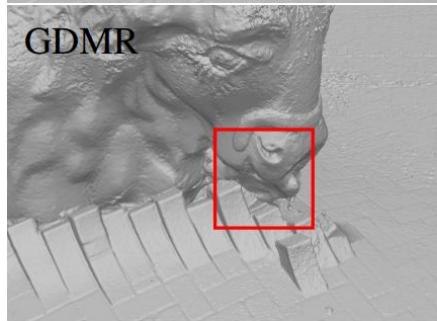
Dataset name	Input data	GDMR Peak RAM	GDMR time	Our Peak RAM	Our time	SSR Peak RAM	SSR time
Citywall	564 depth maps	75 GB	19 h	13.17 GB	63 min	32*8.9 GB	58 h

x19 faster

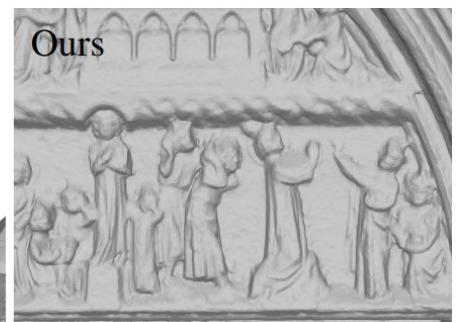
Ours



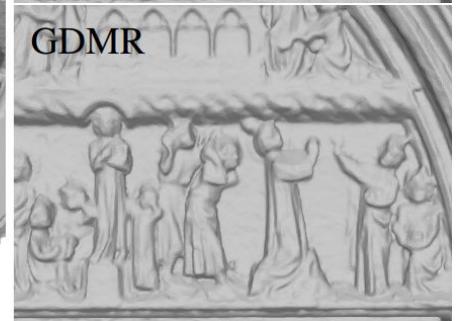
GDMR



Ours



GDMR



Dataset name	Input data	GDMR Peak RAM	GDMR time	Our Peak RAM	Our time	SSR Peak RAM	SSR time
Citywall	564 depth maps	75 GB	19 h	13.17 GB	63 min	32*8.9 GB	58 h
Breisach	2111 depth maps	64 GB	76 h	10.07 GB	260 min	N/A	N/A

x17 faster
Out-of-Core Surface Reconstruction via Global TGV Minimization, Poliarnyi, 2021

Ссылки

- 3D модель Кижского погоста (Agisoft)
- 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

Вопросы?



Полярный Николай
polarnick239@gmail.com