

# День Науки: Фотограмметрия и GPU

- Как из фотографий сделать 3D модель?
- Какие алгоритмы за этим стоят?
- Градиентный спуск
- Алгоритм Дейкстры
- Merge-sort на видеокарте
- Как жить?

 **Agisoft**  
 **Metashape**

 [@PolarNick239](https://t.me/PolarNick239)  
 [polarnick@agisoft.com](mailto:polarnick@agisoft.com)

Полярный Николай Вадимович



# Глава 1

## Фотограмметрия

2429-16-Г



Данные предоставил: Геоскан

2429-16-Г



Данные предоставил: Геоскан

2429-16-Г



Данные предоставил: Геоскан



STAR WARS

# BATTLEFRONT



Star Wars: Battlefront and the Art of Photogrammetry

GDC<sup>16</sup>  
GDC



# STAR WARS BATTLEFRONT



Star Wars: Battlefront and the Art of Photogrammetry

GDC 16  
GDC

# Использование фотограмметрии в играх и кино

Тизер Cyberpunk 2077



[Behance: Cyberpunk 2077 teaser trailer](#)

# Использование фотограмметрии в играх и кино

Тизер Cyberpunk 2077



[Behance: Cyberpunk 2077 teaser trailer](#)

# Использование фотограмметрии в играх и кино

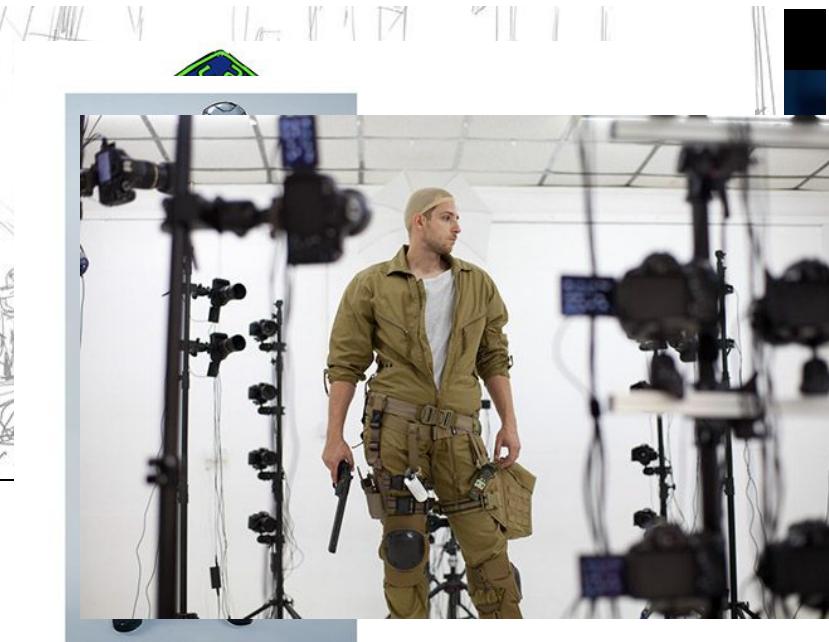
Тизер Cyberpunk 2077



[Behance: Cyberpunk 2077 teaser trailer](#)

# Использование фотограмметрии в играх и кино

Тизер Cyberpunk 2077



[Behance: Cyberpunk 2077 teaser trailer](#)

# Использование фотограмметрии в играх и кино

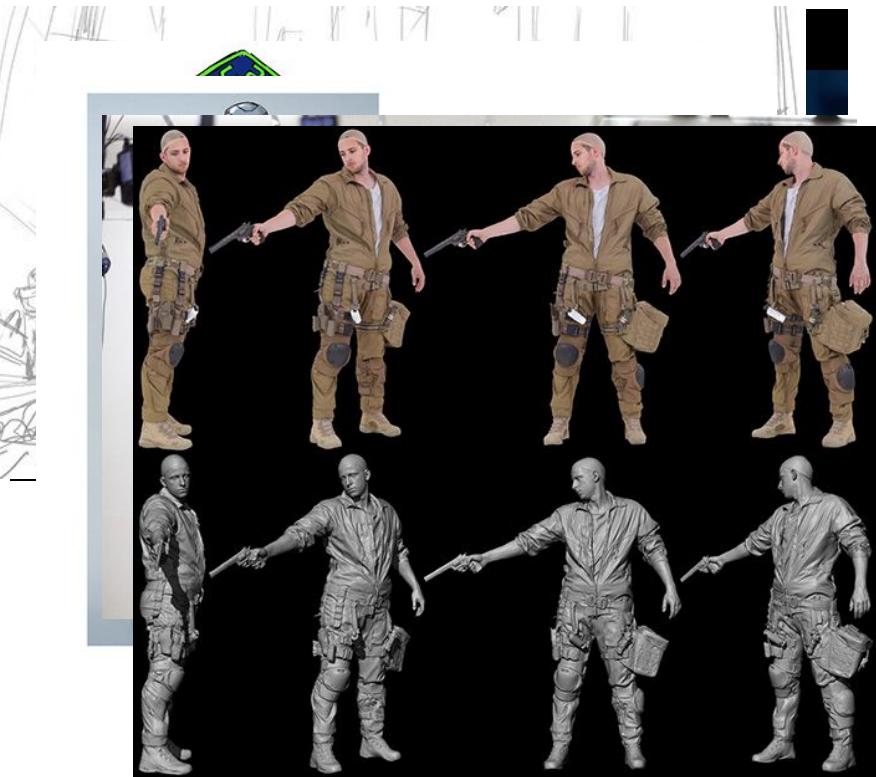
Тизер Cyberpunk 2077



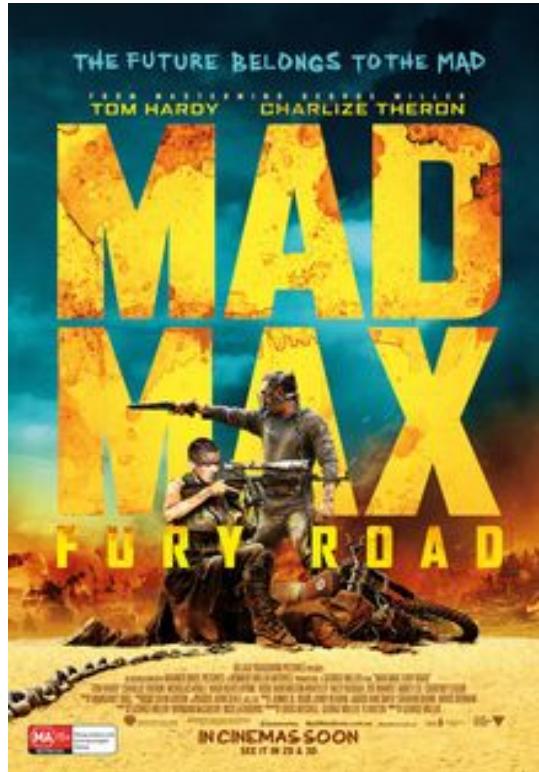
[Behance: Cyberpunk 2077 teaser trailer](#)

# Использование фотограмметрии в играх и кино

Тизер Cyberpunk 2077



Mad Max: Fury Road

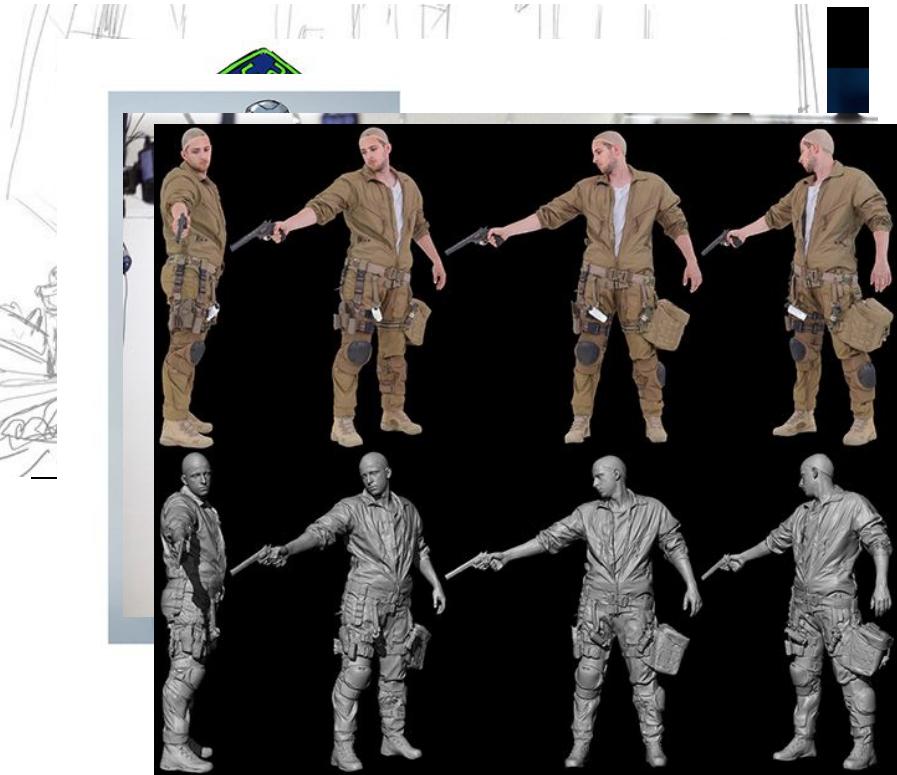


[Behance: Cyberpunk 2077 teaser trailer](#)

[Visual effects of Mad Max: Fury Road](#)

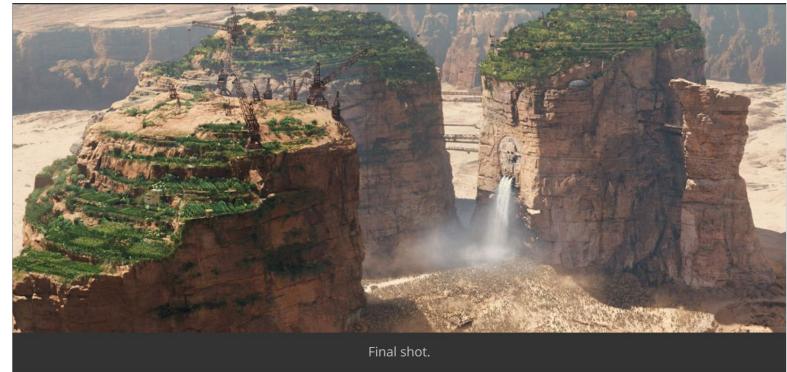
# Использование фотограмметрии в играх и кино

Тизер Cyberpunk 2077



[Behance: Cyberpunk 2077 teaser trailer](#)

Mad Max: Fury Road



[Visual effects of Mad Max: Fury Road](#)



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



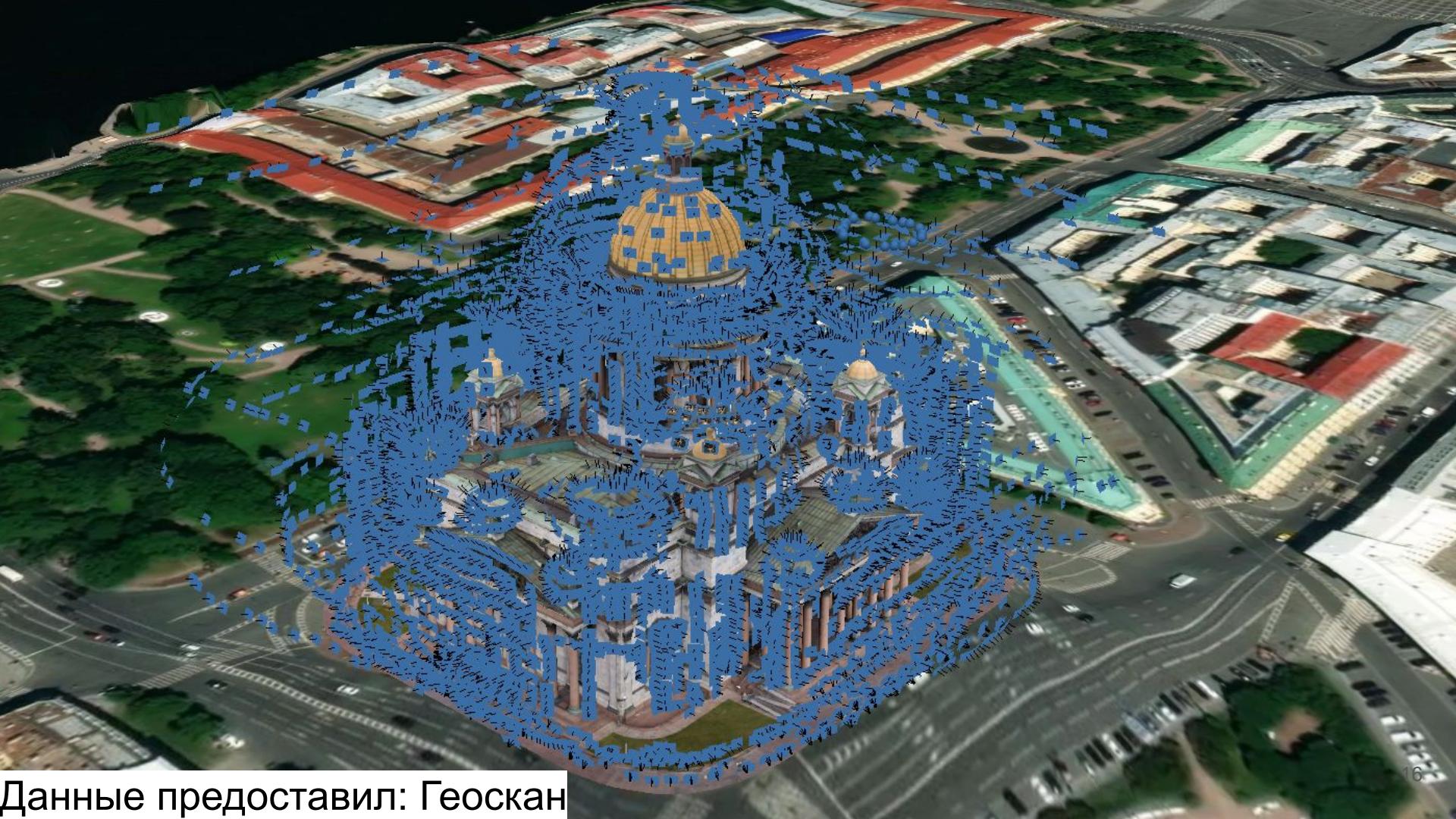
is\_copter-20180820-1-0884    is\_copter-20180820-1-0885    is\_copter-20180820-1-0886    is\_copter-20180820-1-0887    is\_copter-20180820-1-0888    is\_copter-20180820-1-0889    is\_copter-20180820-1-0890



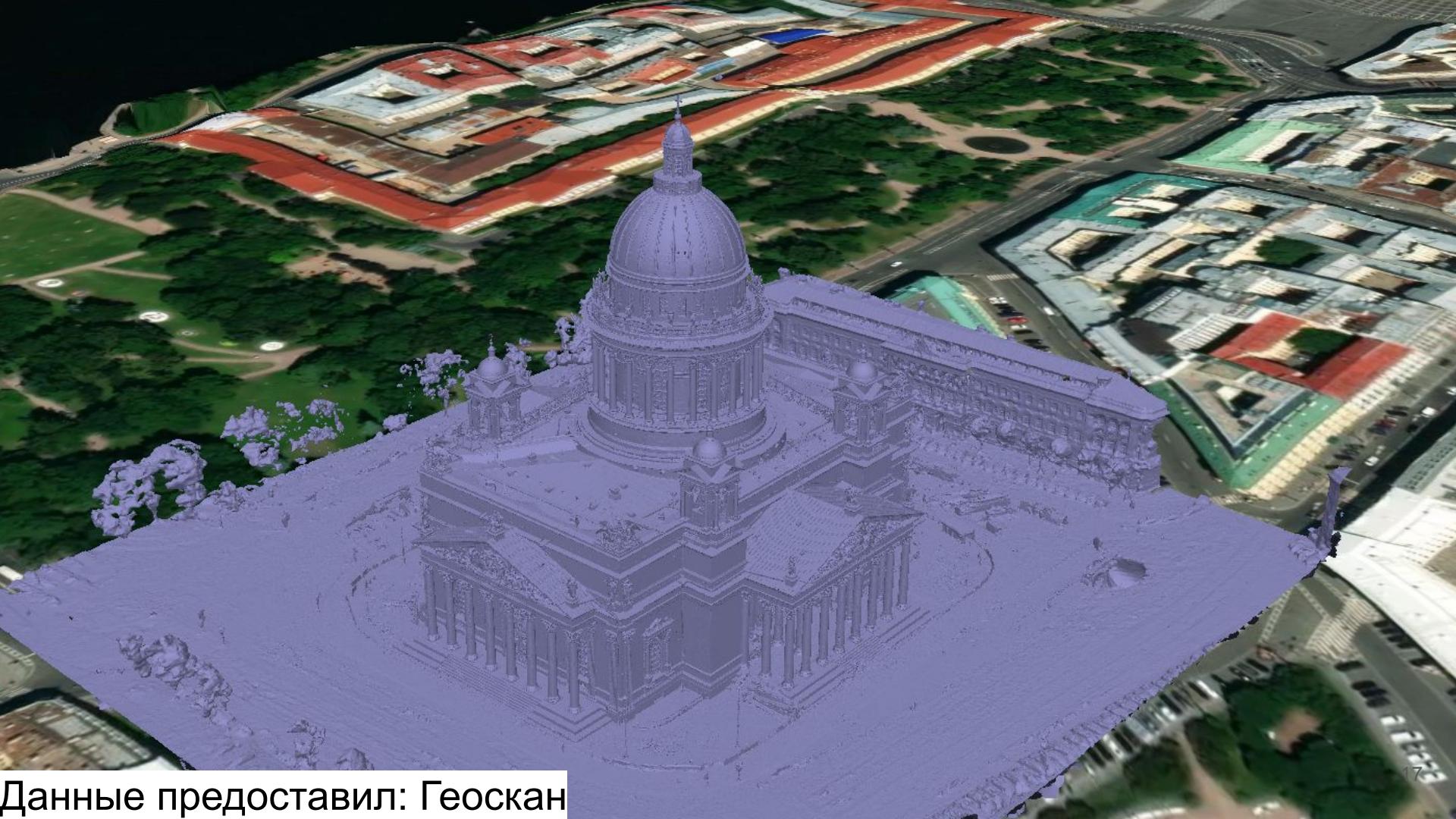
is\_copter-20180821-0-4041    is\_copter-20180821-0-4042    is\_copter-20180821-0-4043    is\_copter-20180821-0-4044    is\_copter-20180821-0-4045    is\_copter-20180821-0-4046    is\_copter-20180821-0-4047



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



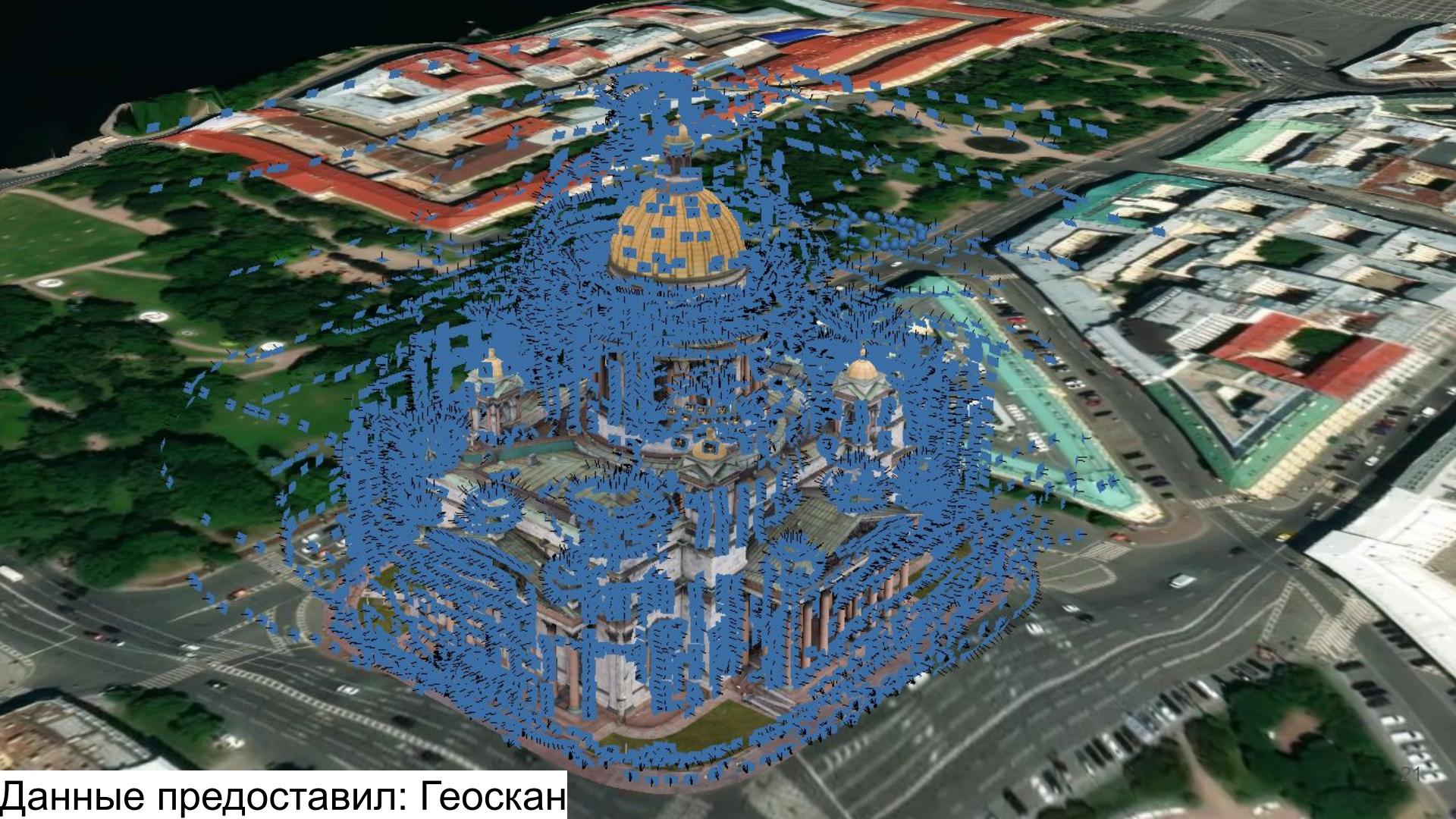
Данные предоставил: Геоскан



ИА ТА ГОСПОДИ УПОВАХОМЪ ДЛ НЕ ПОСТАДИСА ВО ЕЗЫК.



Данные предоставил: Геоскан



Данные предоставил: Геоскан

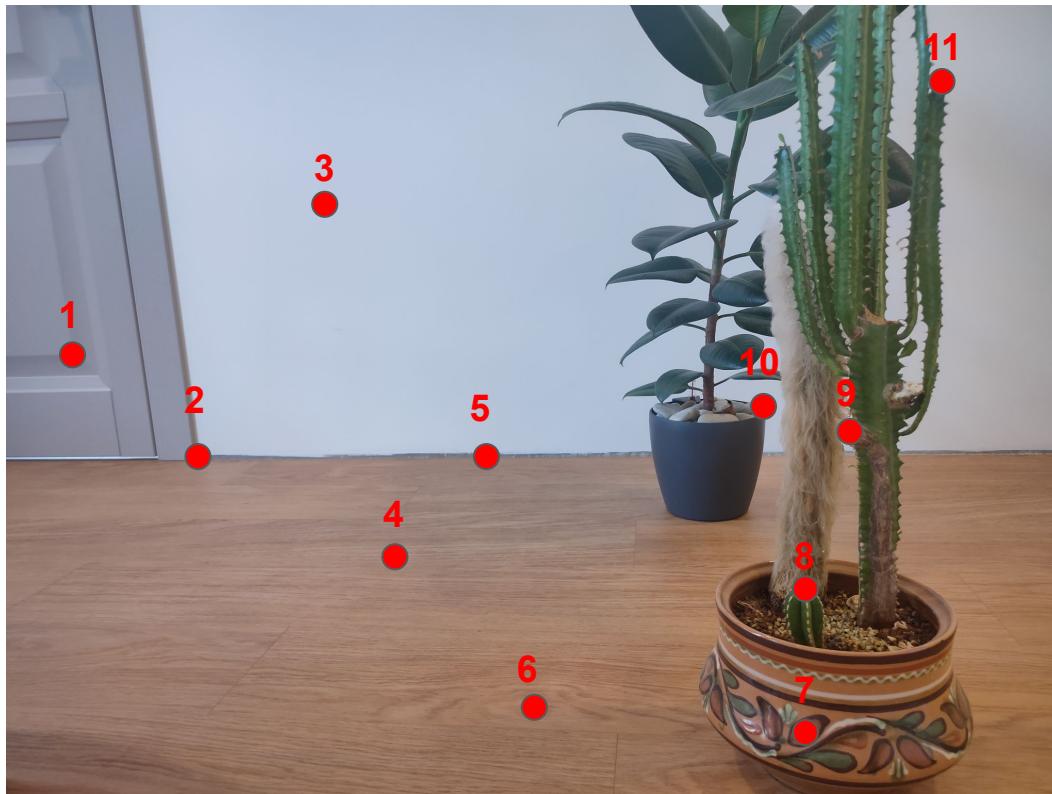
# Глава 2

Рассчет взаимного положения фотографий  
и градиентный спуск

# Фотограмметрия

1) Сопоставляем фотографии

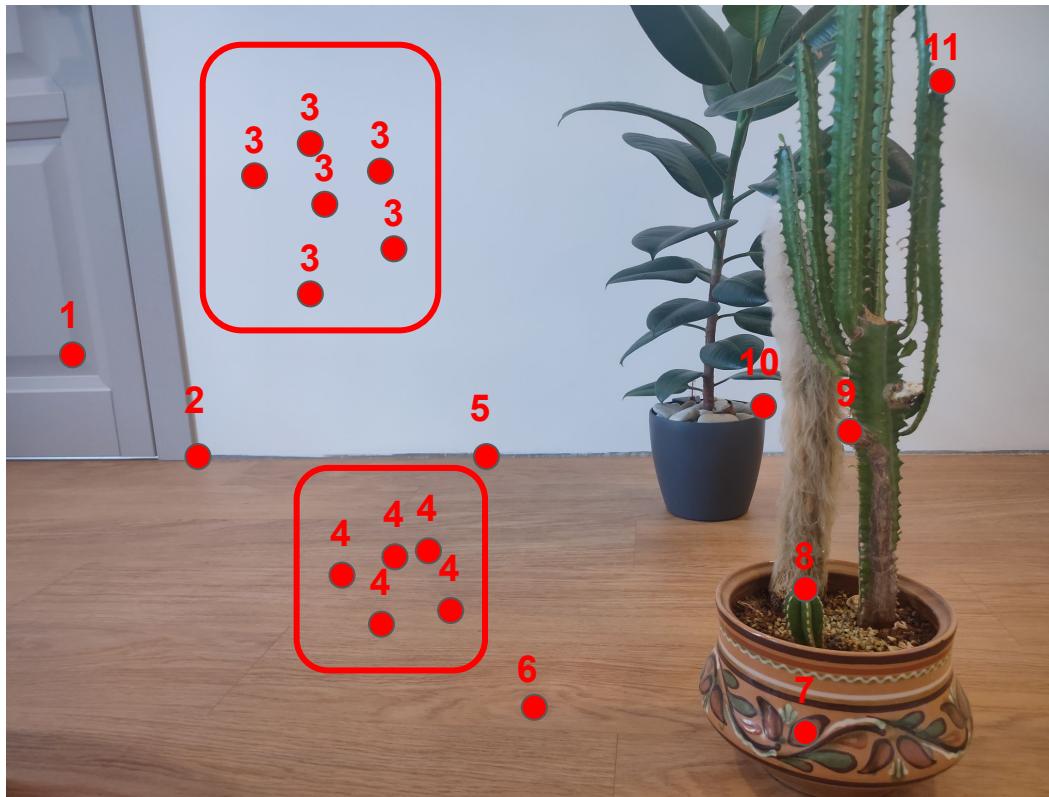
# 1) Сопоставляем фотографии: ключевые точки



Какие точки не получится использовать для сопоставления с другой фотографией?



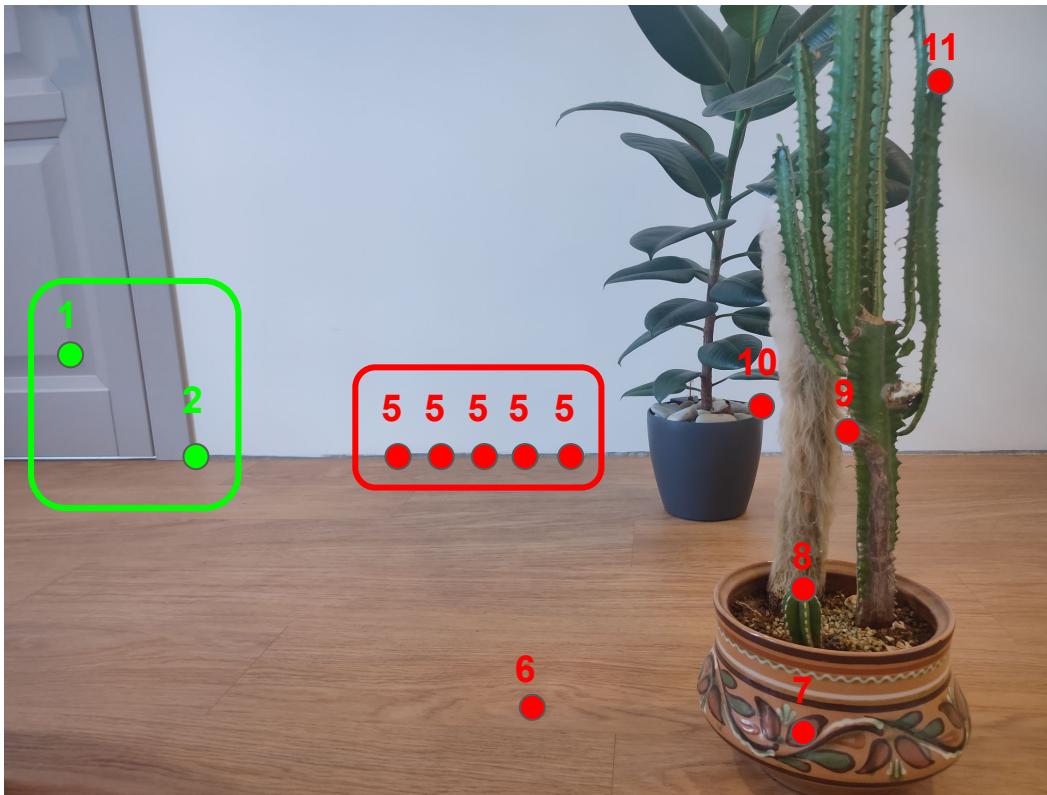
# 1) Сопоставляем фотографии: ключевые точки



Какие точки не получится использовать для сопоставления с другой фотографией?



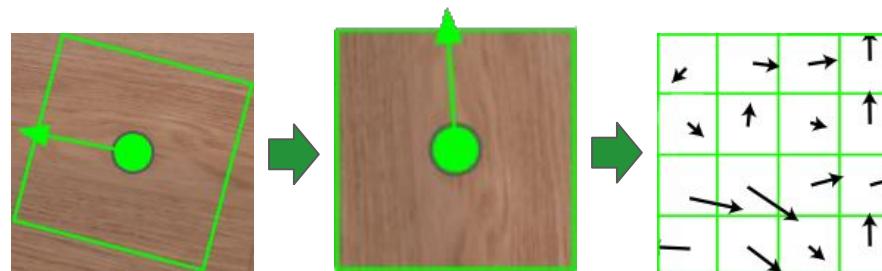
# 1) Сопоставляем фотографии: ключевые точки



Какие точки не получится использовать для сопоставления с другой фотографией?

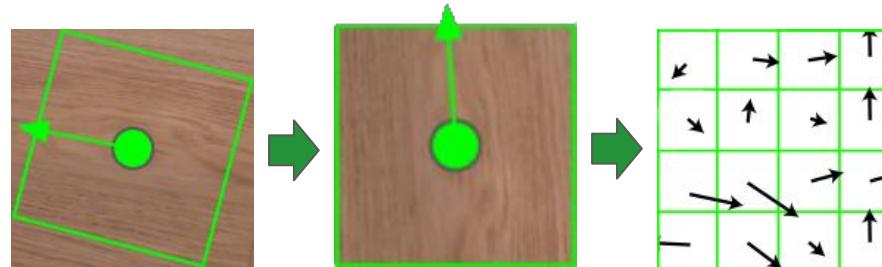


# 1) Сопоставляем фотографии: дескрипторы точек



дескриптор из  
128 вещественных чисел

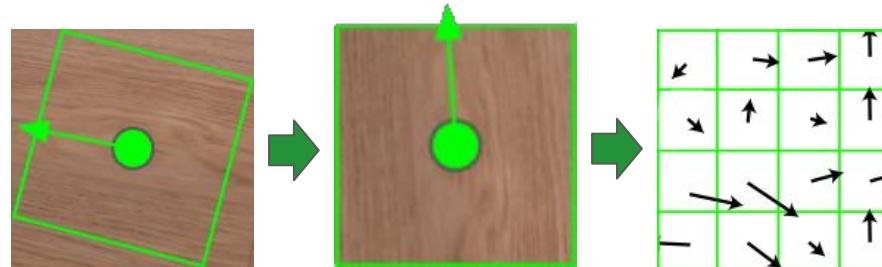
# 1) Сопоставляем фотографии: дескрипторы точек



дескриптор из  
128 вещественных чисел

Пусть есть две точки, точка А и точка В. Как оценить их похожесть?

# 1) Сопоставляем фотографии: дескрипторы точек



дескриптор из

128 вещественных чисел

Пусть есть две точки, точка А и точка В.  
Их похожесть - **Евклидово расстояние**  
между их дескрипторами:

$$dist(A, B) = \sqrt{\sum_{i=1}^{128} (A_i - B_i)^2}$$

# 1) Сопоставляем фотографии: дескрипторы точек

- 1.1) Выбрали ключевые точки
- 1.2) Построили дескрипторы для точек
- 1.3) Сопоставили точки на базе похожести дескрипторов (Евклидово расстояние)



# Фотограмметрия

- 1) Сопоставляем фотографии (ключевые точки + дескрипторы)

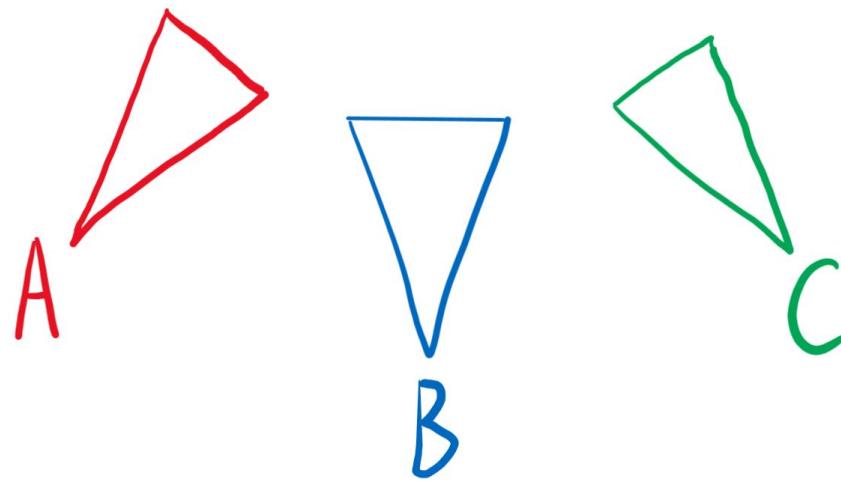
# Фотограмметрия

- 1) Сопоставляем фотографии (ключевые точки + дескрипторы)
- 2) Ищем взаимное положение камер

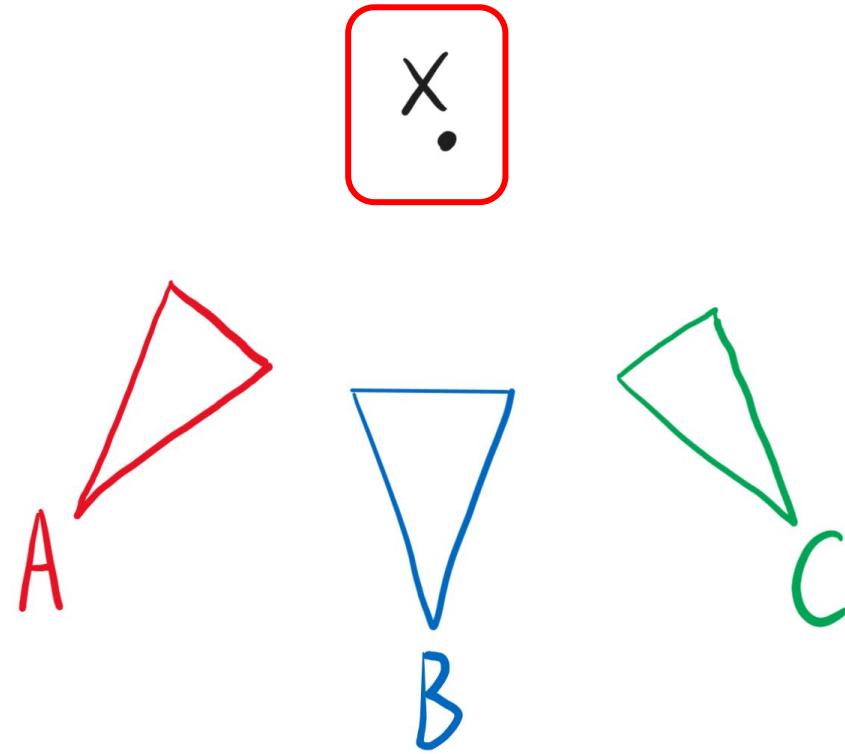


Данные предоставил Stéphane Prodent

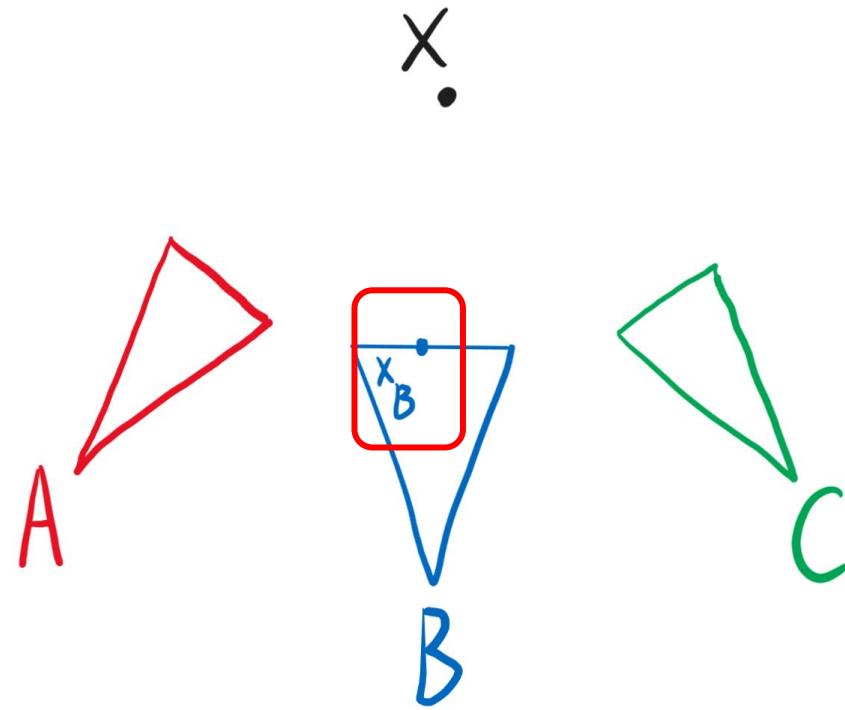
## 2) Ищем взаимное положение камер



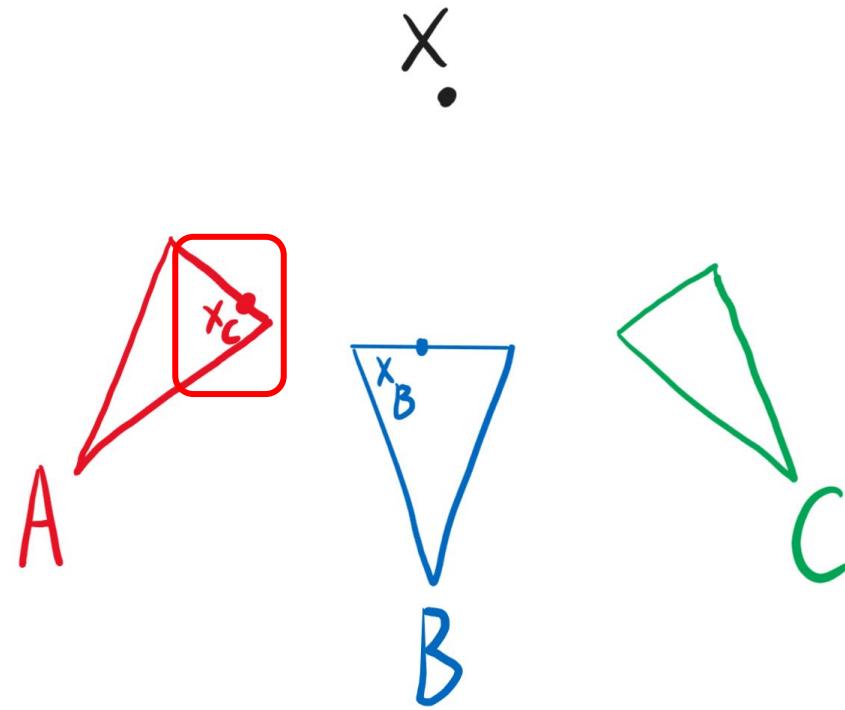
## 2) Ищем взаимное положение камер



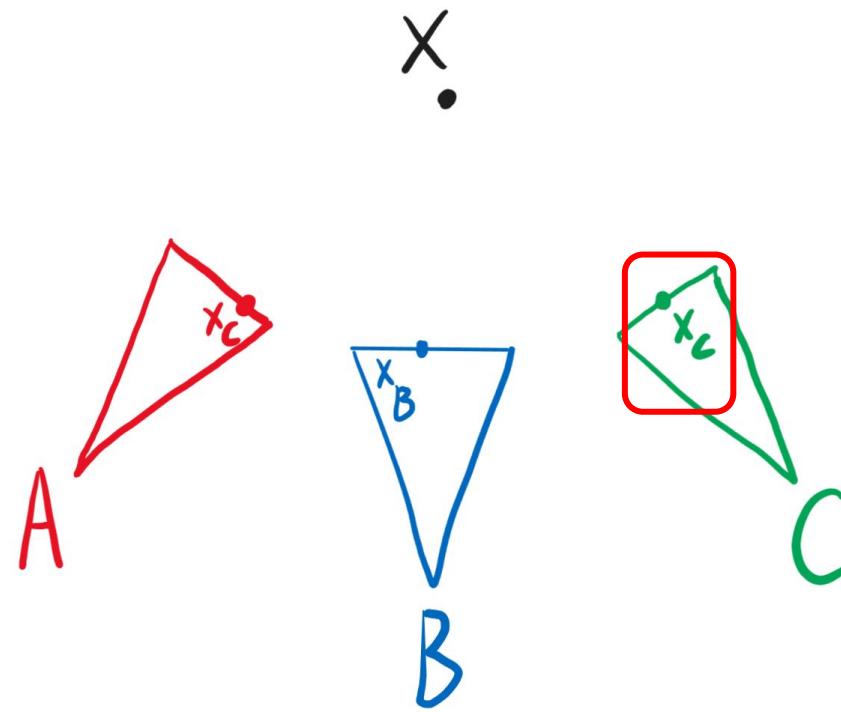
## 2) Ищем взаимное положение камер



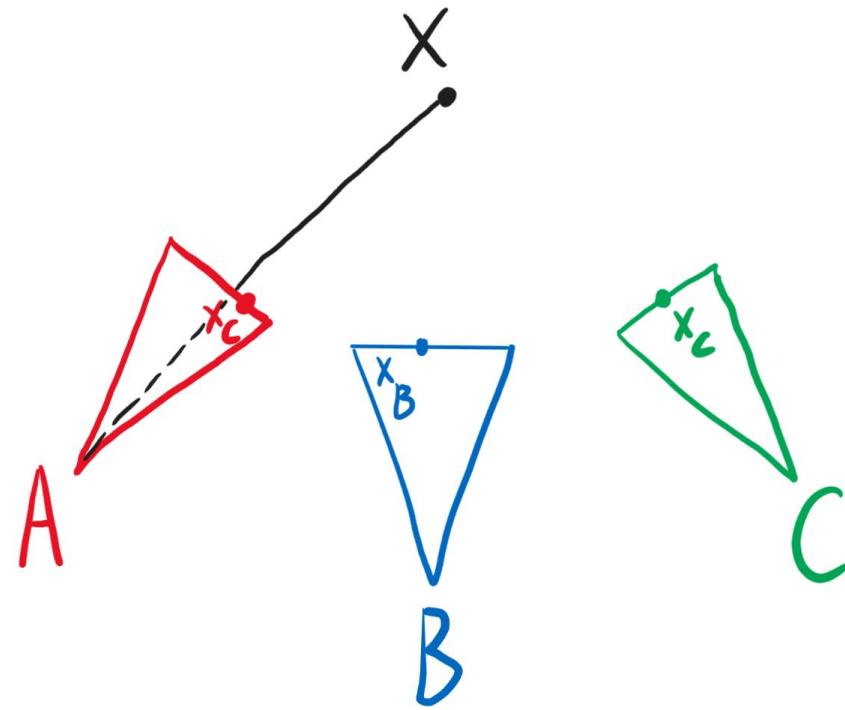
## 2) Ищем взаимное положение камер



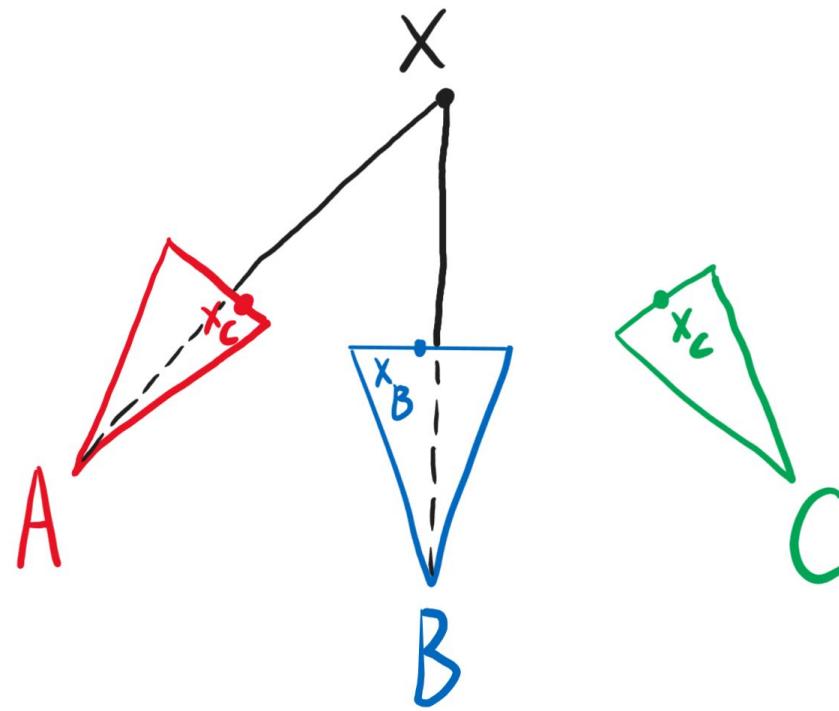
## 2) Ищем взаимное положение камер



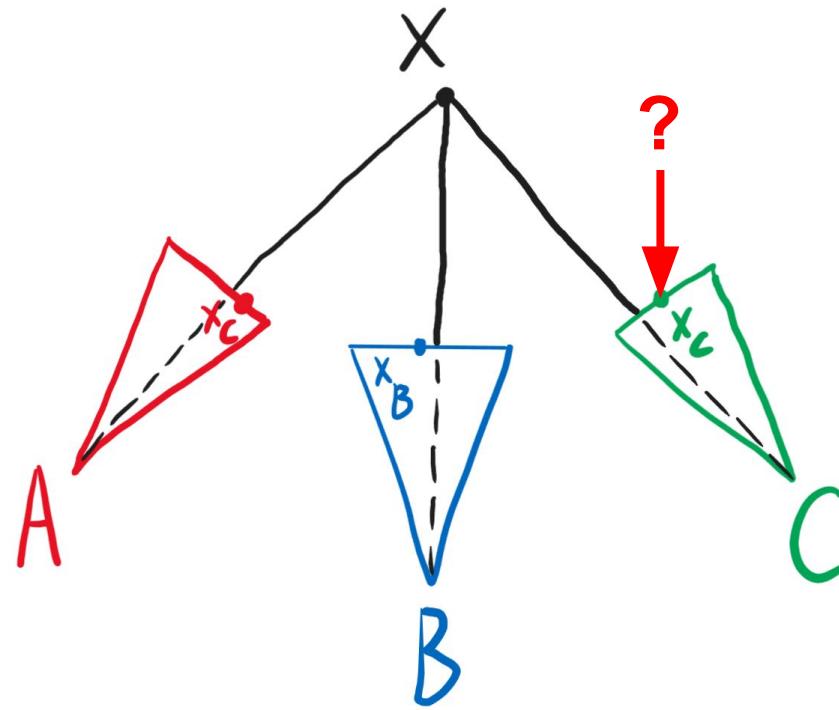
## 2) Ищем взаимное положение камер



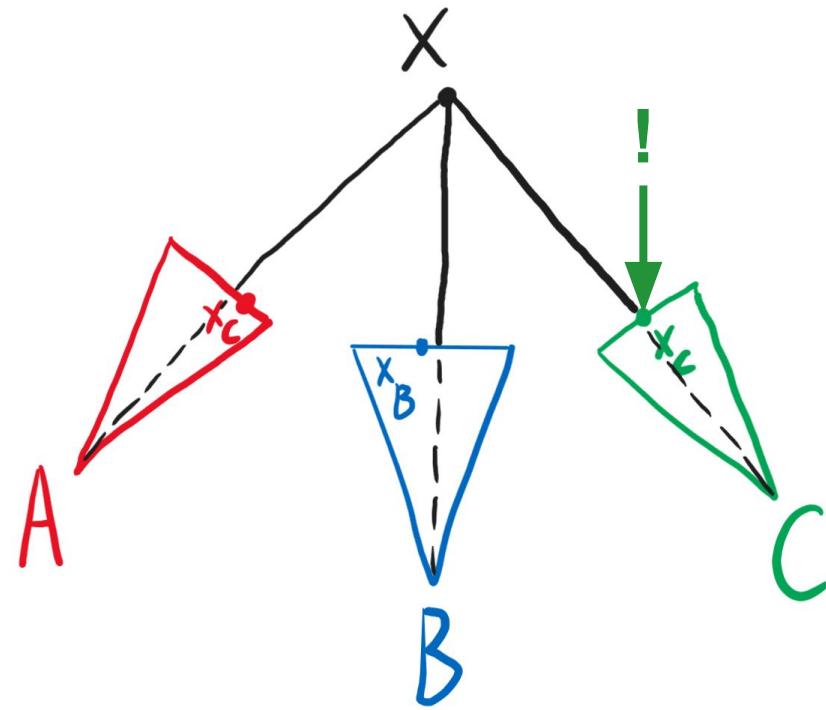
## 2) Ищем взаимное положение камер



## 2) Ищем взаимное положение камер

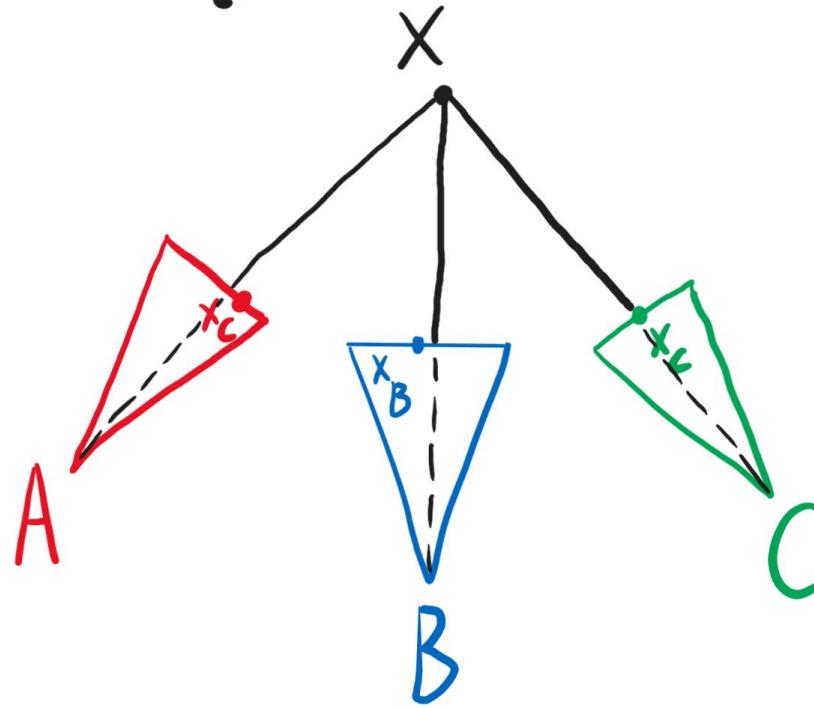


## 2) Ищем взаимное положение камер



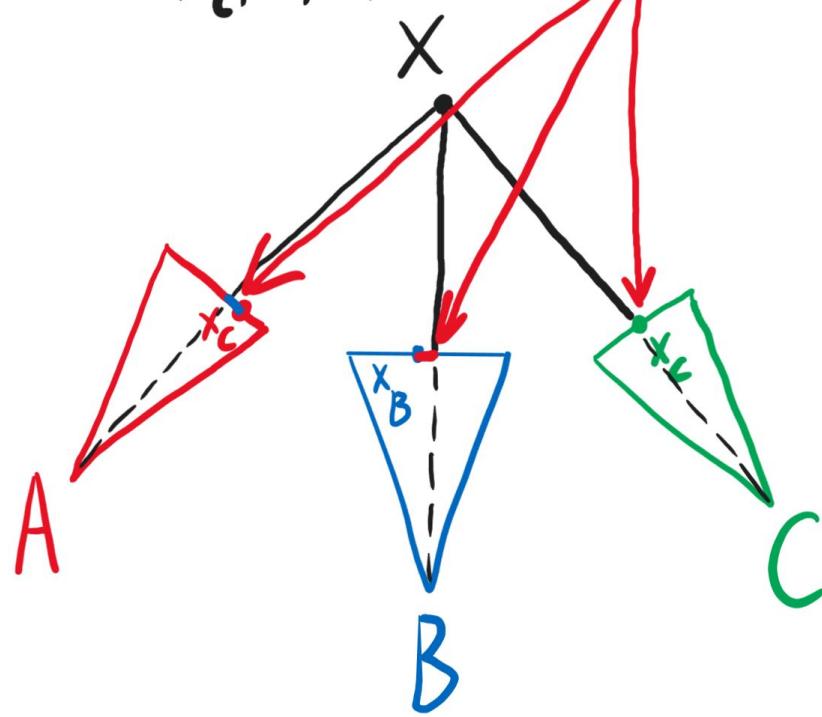
2) Ищем взаимное положение камер

$$\text{loss}(x_i, A, B, C) = ???$$



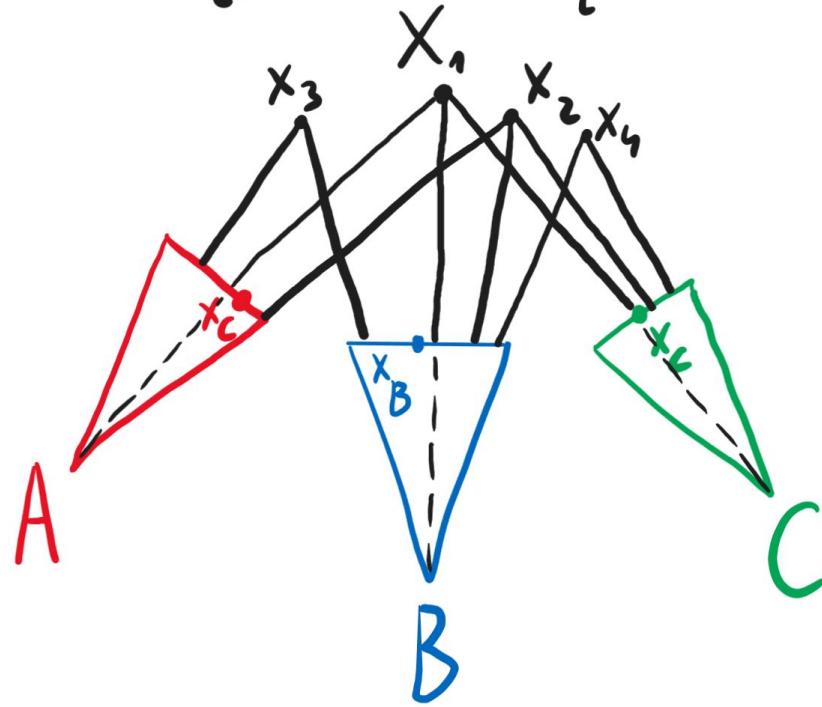
## 2) Ищем взаимное положение камер

$$\text{loss}(x_i, A, B, C) = \sum_{\text{ПРОЕЦИРОВАНИЯ}}^{\text{ошибок}}$$

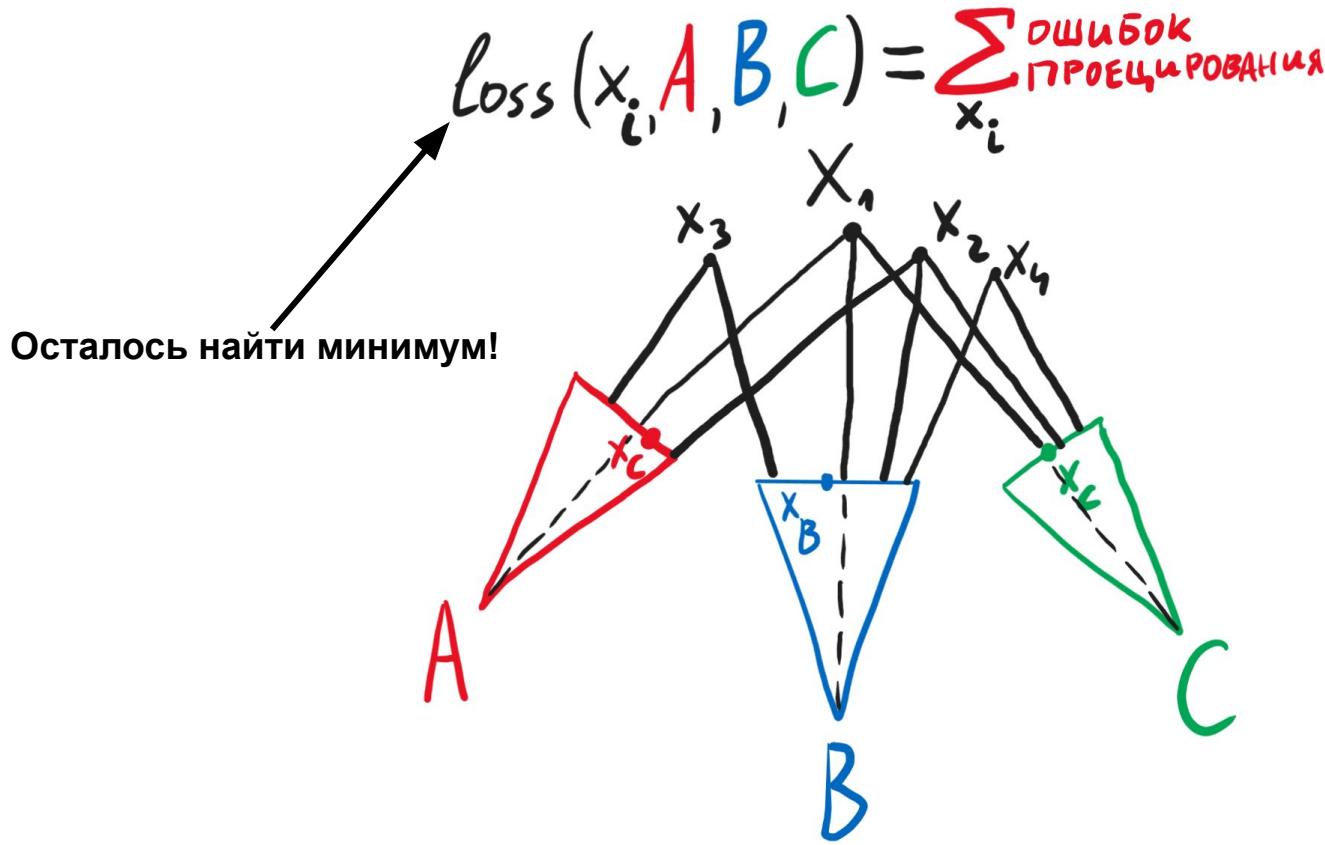


## 2) Ищем взаимное положение камер

$$\text{loss}(x_i, A, B, C) = \sum_{x_i}^{\text{ошибок проектирования}}$$



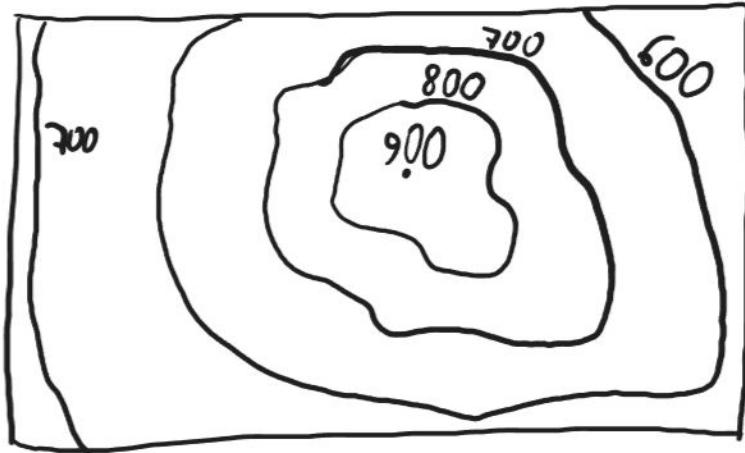
## 2) Ищем взаимное положение камер



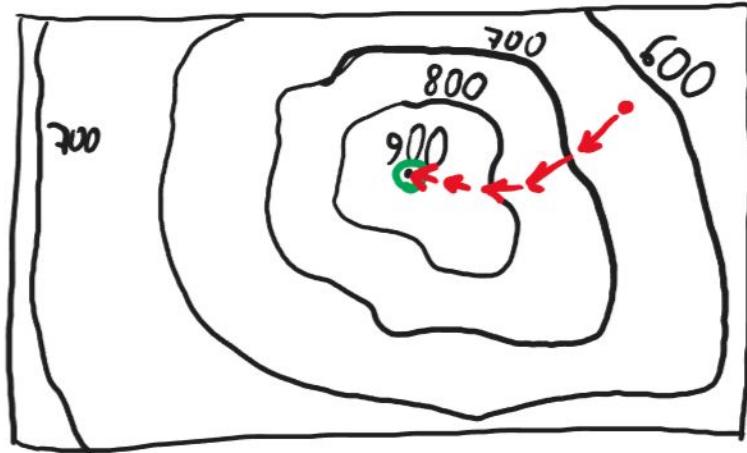
# Идем к вершине горы в сильный туман



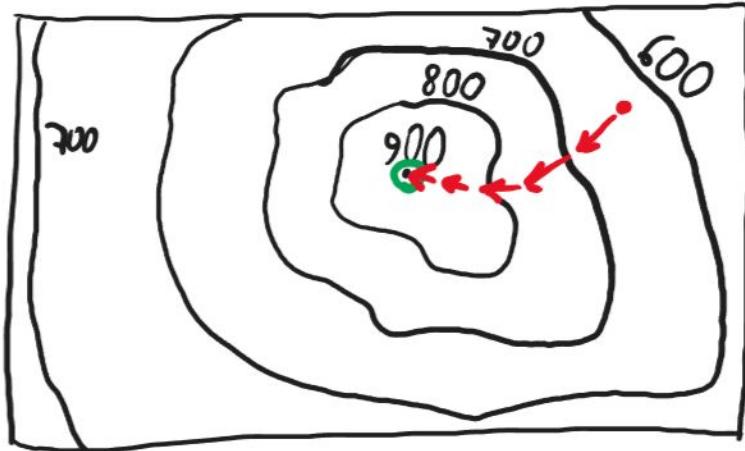
Идем к вершине горы в сильный туман



Идем к вершине горы в сильный туман

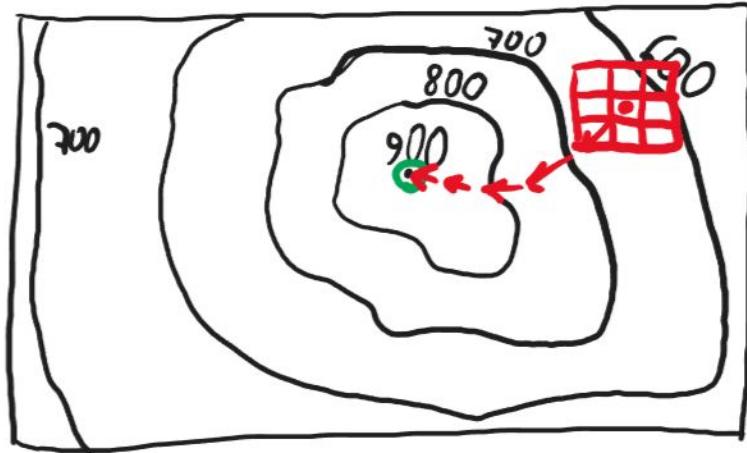


# Идем к вершине горы в сильный туман

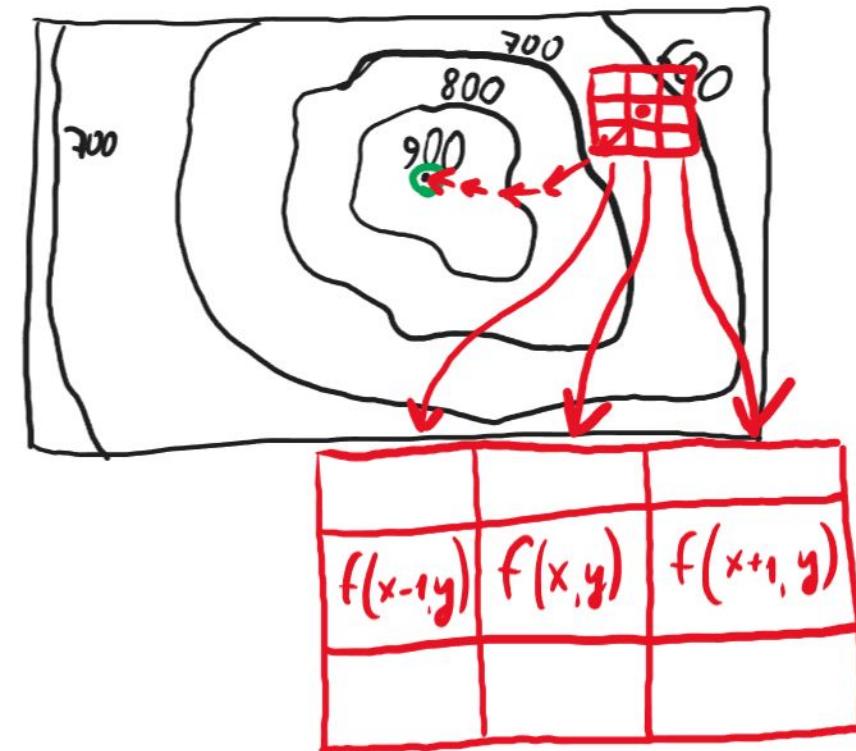


Это называется **градиентный спуск** (хоть мы и поднимаемся).

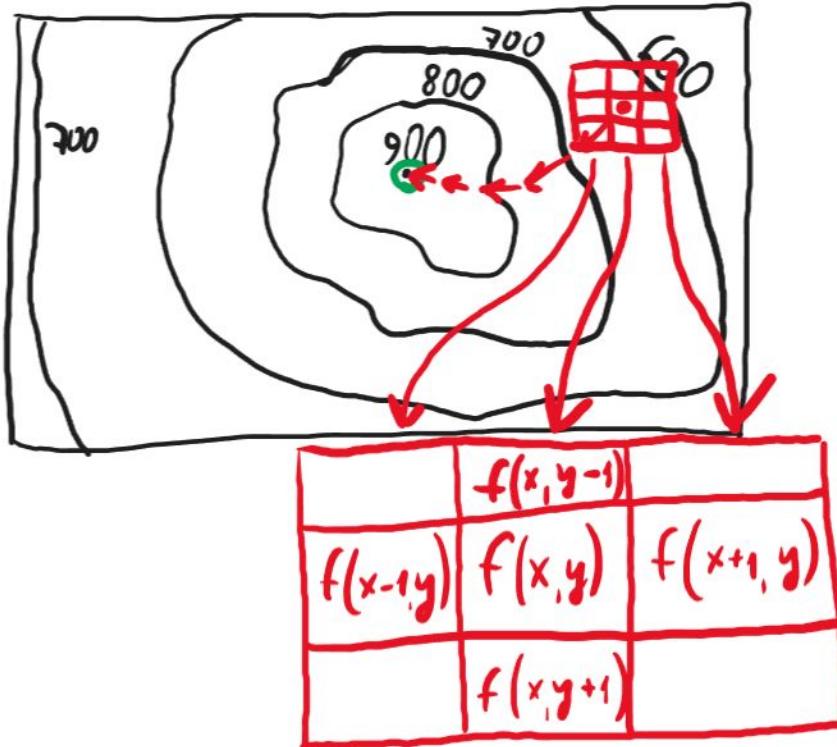
Идем к вершине горы в сильный туман



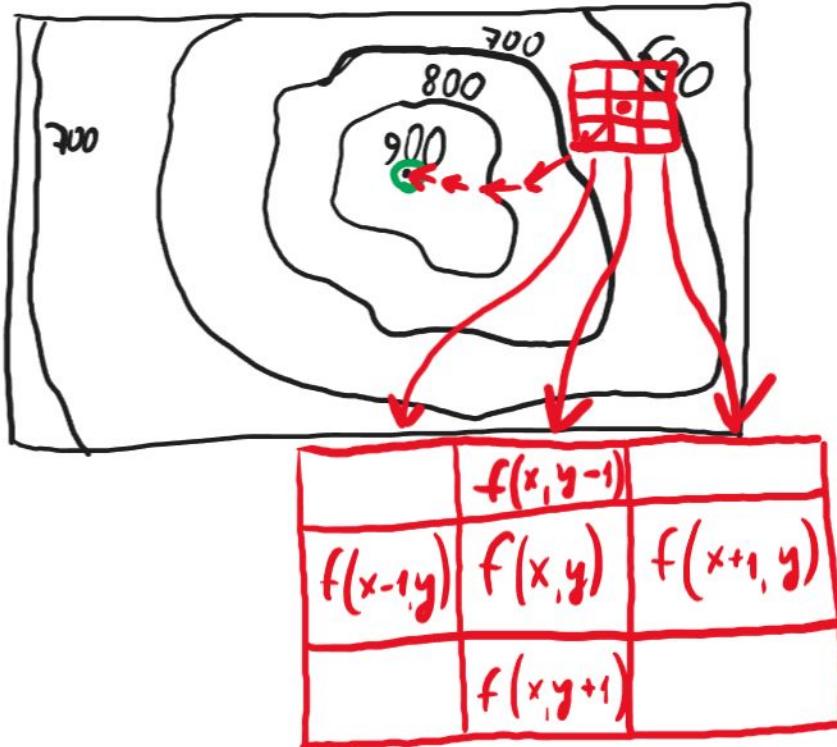
# Идем к вершине горы в сильный туман



# Идем к вершине горы в сильный туман

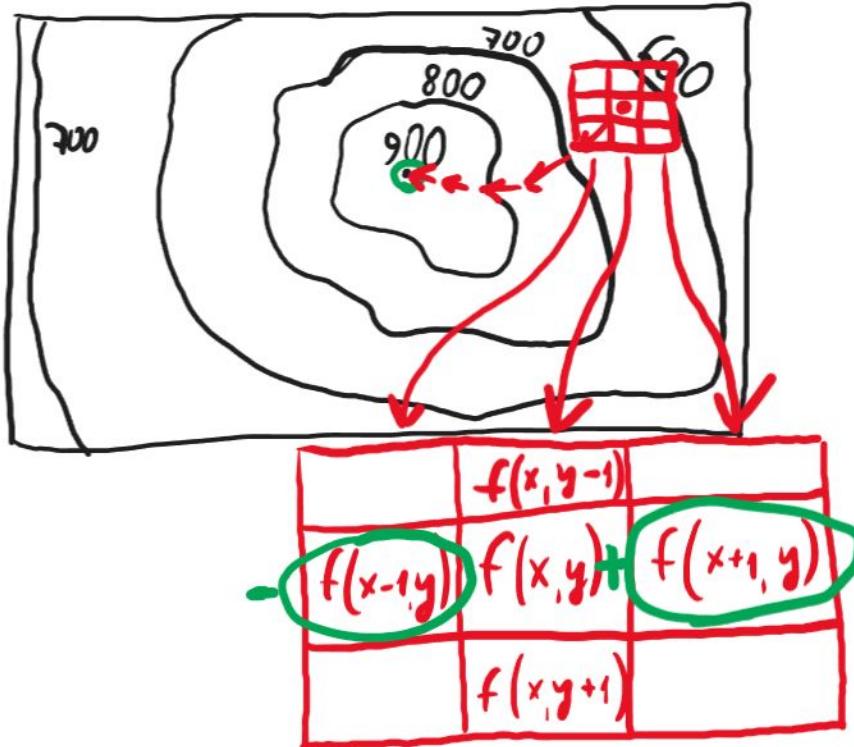


# Идем к вершине горы в сильный туман



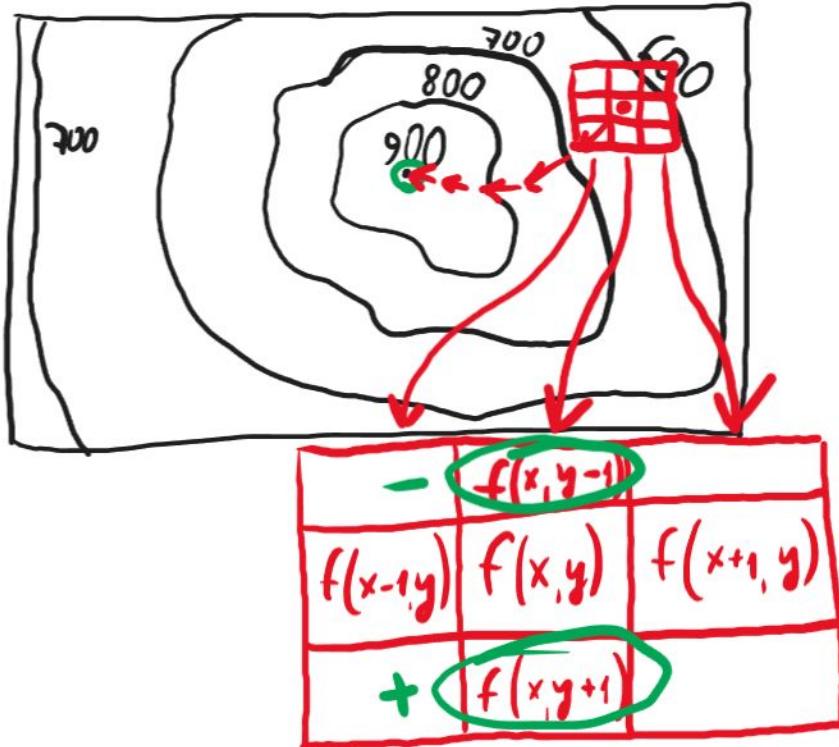
$$f'_x(x, y) = ?$$

# Идем к вершине горы в сильный туман



$$f'_x(x, y) = f(x+1, y) - f(x-1, y)$$

# Идем к вершине горы в сильный туман

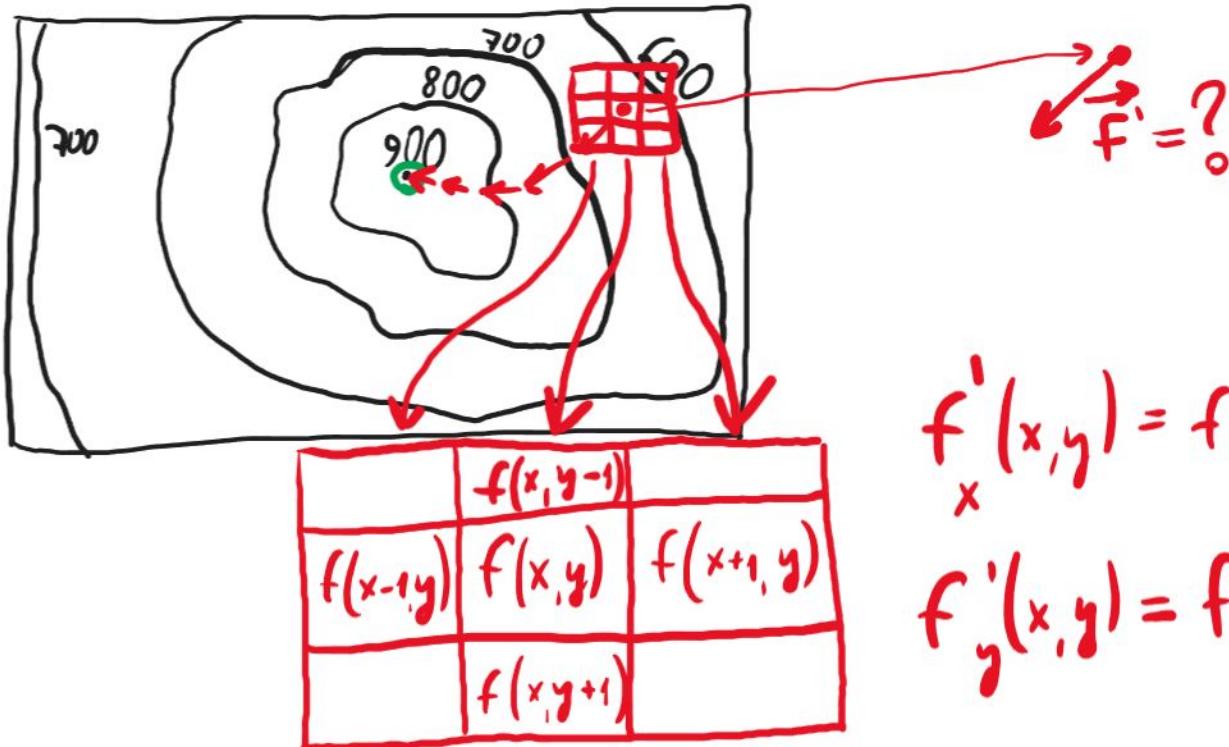


$$f_x'(x, y) = f(x+1, y) - f(x-1, y)$$

$$f_y'(x, y) = f(x, y+1) - f(x, y-1)$$

Это называется оператор Собеля

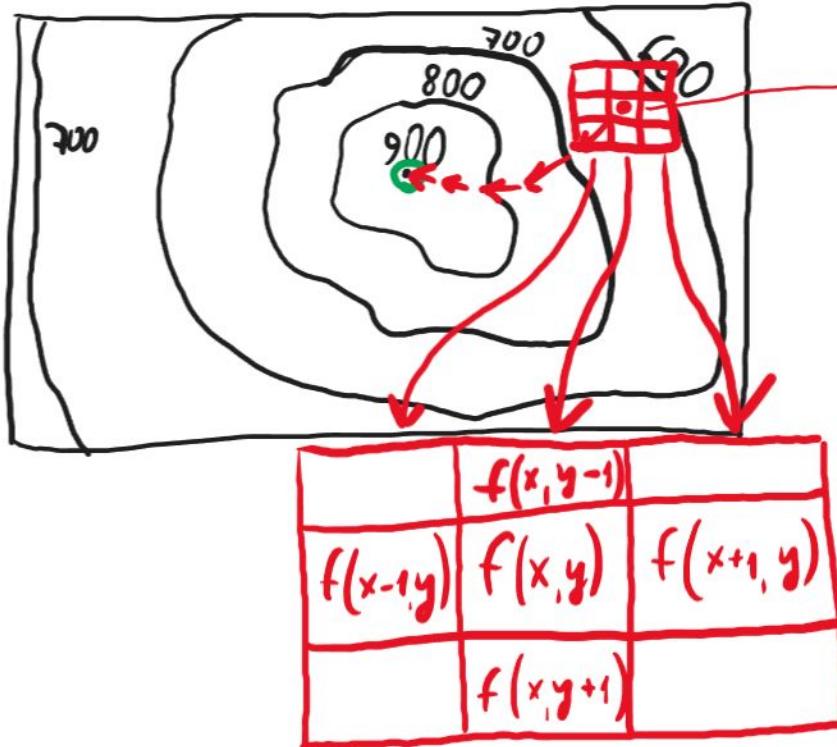
# Идем к вершине горы в сильный туман



$$f'_x(x, y) = f(x+1, y) - f(x-1, y)$$

$$f'_y(x, y) = f(x, y+1) - f(x, y-1)$$

# Идем к вершине горы в сильный туман



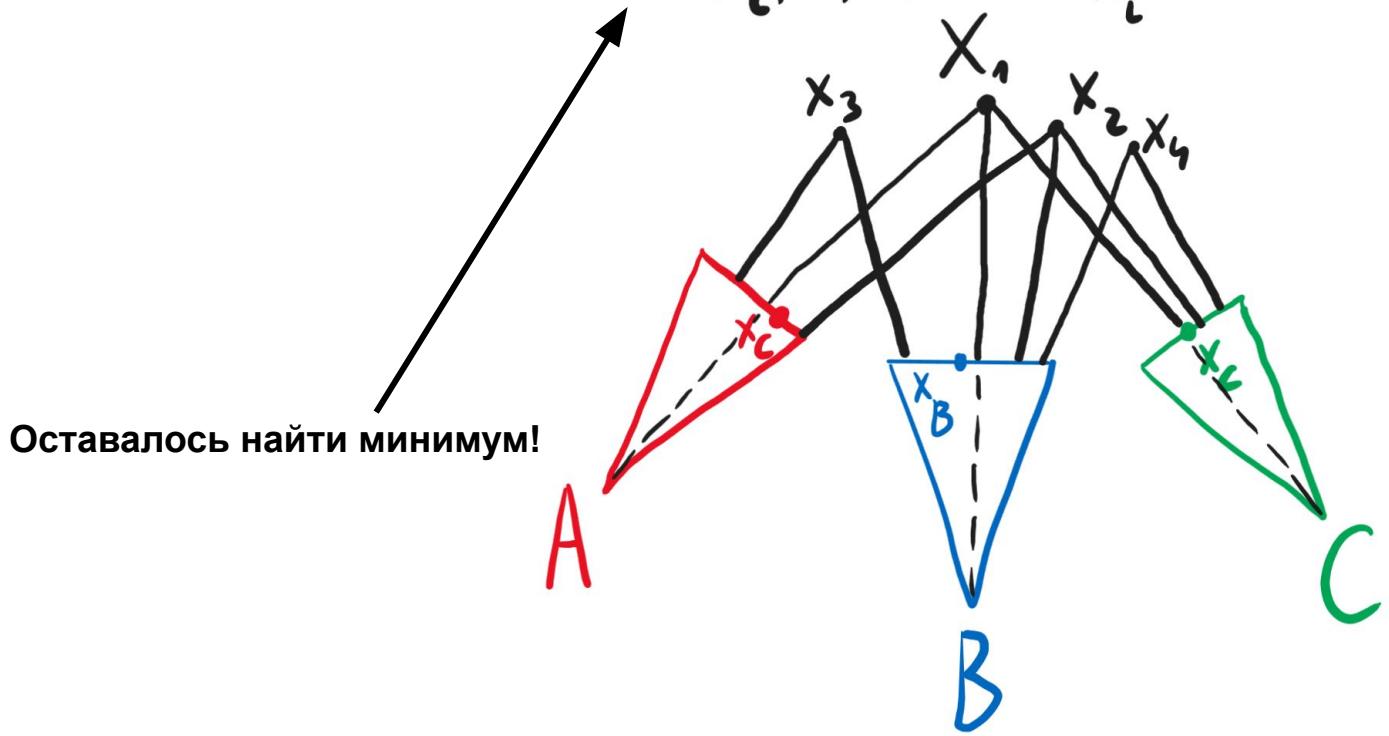
$$\vec{f}' = \underline{\left( f'_x(x, y); f'_y(x, y) \right)}$$

$$f'_x(x, y) = f(x+1, y) - f(x-1, y)$$

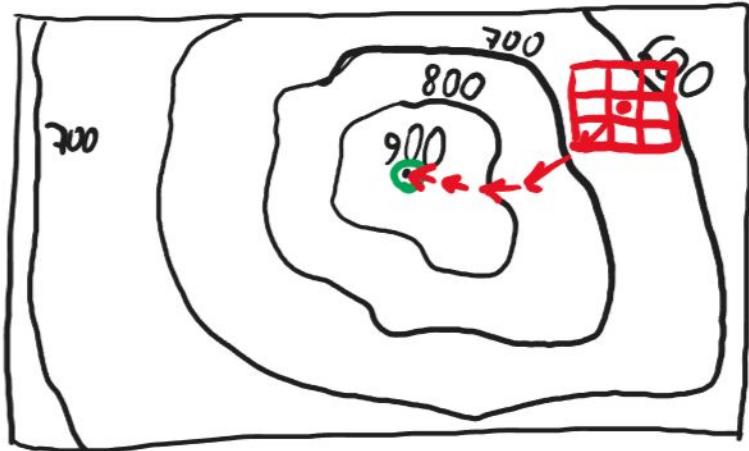
$$f'_y(x, y) = f(x, y+1) - f(x, y-1)$$

2) Ищем взаимное положение камер

$$\text{loss}(x_i, A, B, C) = \sum_{x_i}^{\text{ошибок проецирования}}$$



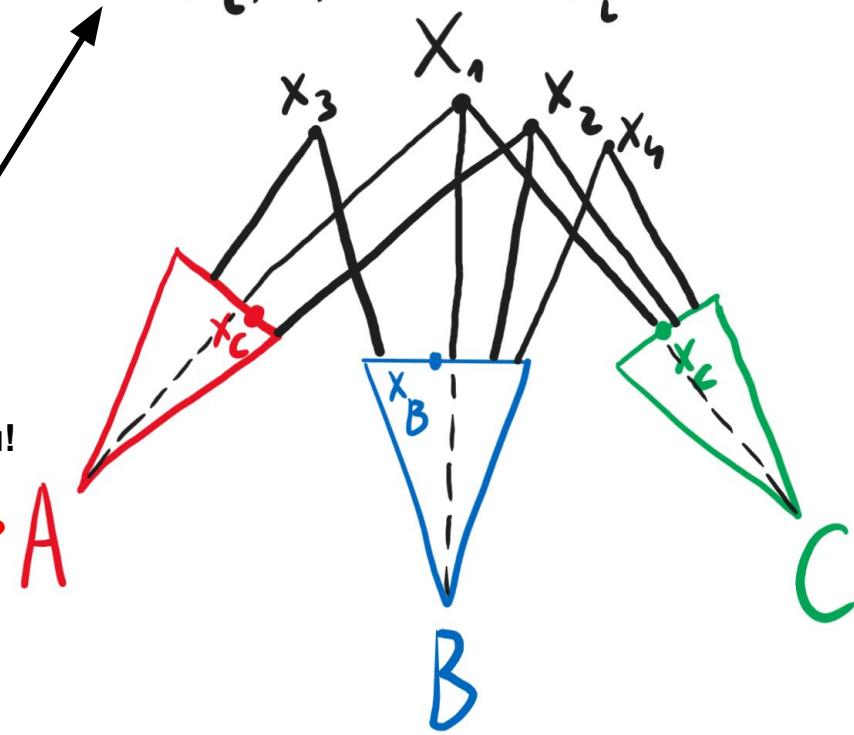
## 2) Ищем взаимное положение камер



Оставалось найти минимум!

Как свести задачу к  
задаче по подъему к вершине?

$$\text{loss}(x_i, A, B, C) = \sum_{x_i}^{\text{ошибок проецирования}}$$



# Глава 3

Карты глубины, 3D модель, текстура

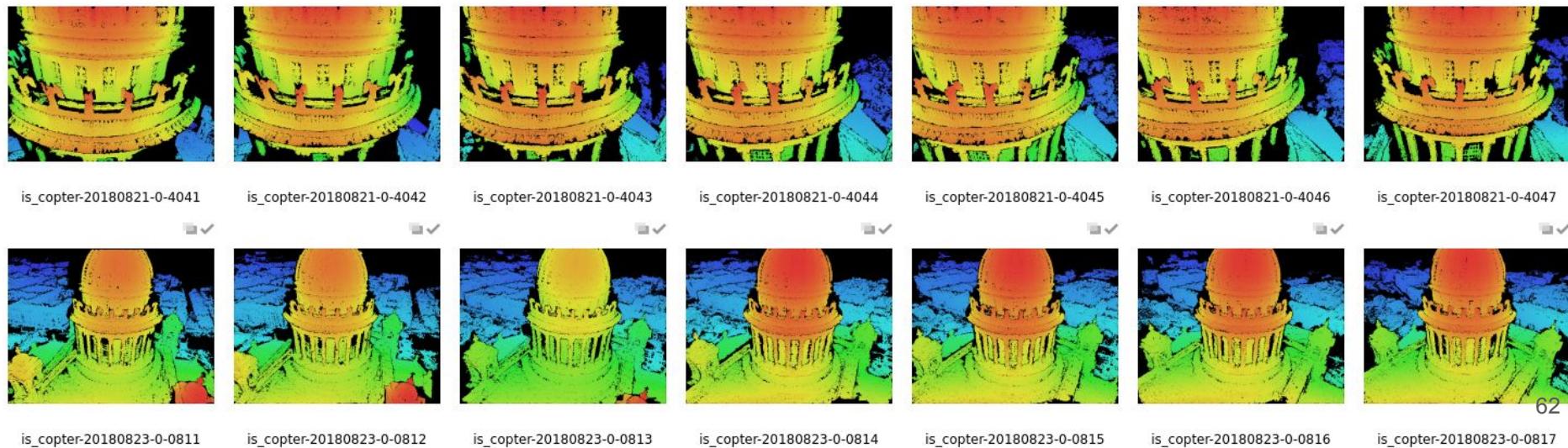
# Фотограмметрия

- 1) Сопоставляем фотографии (ключевые точки + дескрипторы)
- 2) Ищем взаимное положение камер

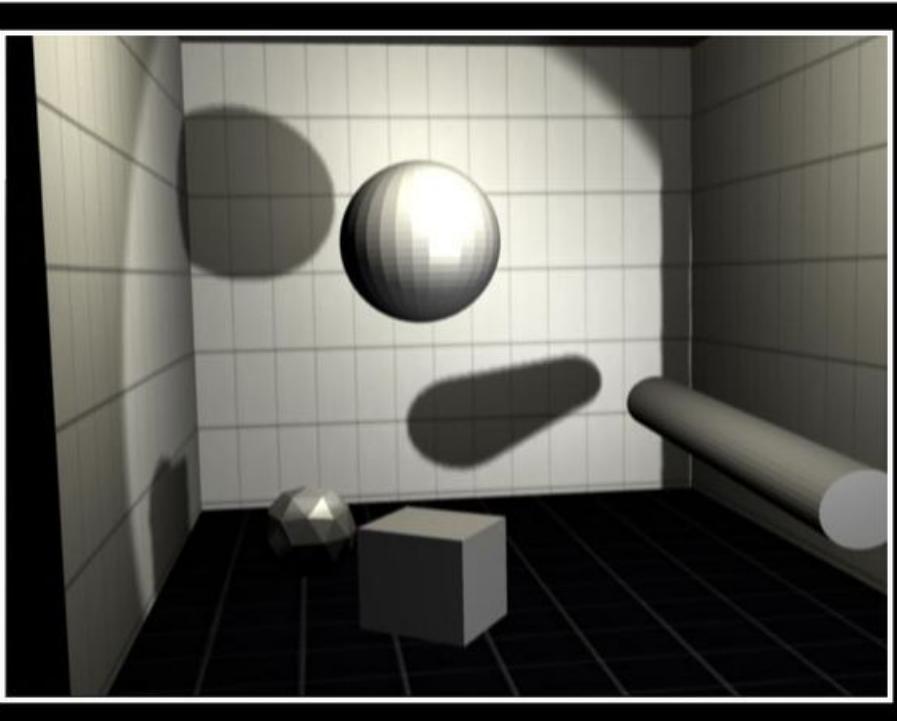


# Фотограмметрия

- 1) Сопоставляем фотографии (ключевые точки + дескрипторы)
- 2) Ищем взаимное положение камер
- 3) Строим карты глубины для каждой фотографии



3) Для каждой фотографии строим карты глубины



возможно благодаря параллаксу

3) Для каждой фотографии строим карты глубины



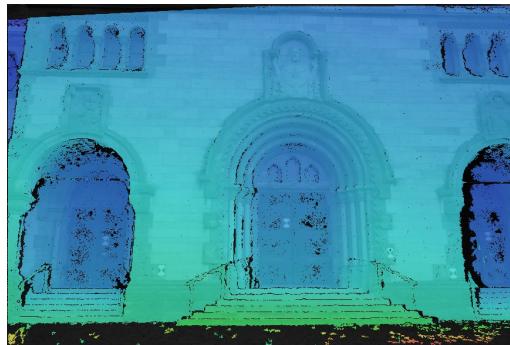
возможно благодаря параллаксу

3) Для каждой фотографии строим карты глубины



возможно благодаря параллаксу

3) Для каждой фотографии строим карты глубины



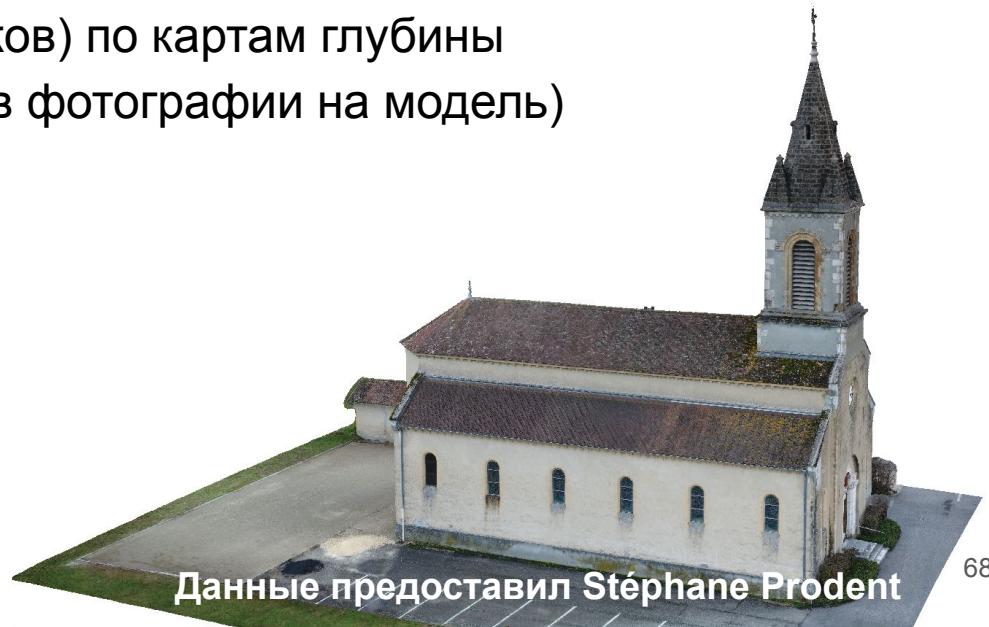
# Фотограмметрия

- 1) Сопоставляем фотографии (ключевые точки + дескрипторы)
- 2) Ищем взаимное положение камер
- 3) Строим карты глубины для каждой фотографии
- 4) Строим модель (из треугольников) по картам глубины



# Фотограмметрия

- 1) Сопоставляем фотографии (ключевые точки + дескрипторы)
- 2) Ищем взаимное положение камер
- 3) Строим карты глубины для каждой фотографии
- 4) Строим модель (из треугольников) по картам глубины
- 5) Строим текстуру (спроектировав фотографии на модель)



Данные предоставил Stéphane Prodent

# Глава 4

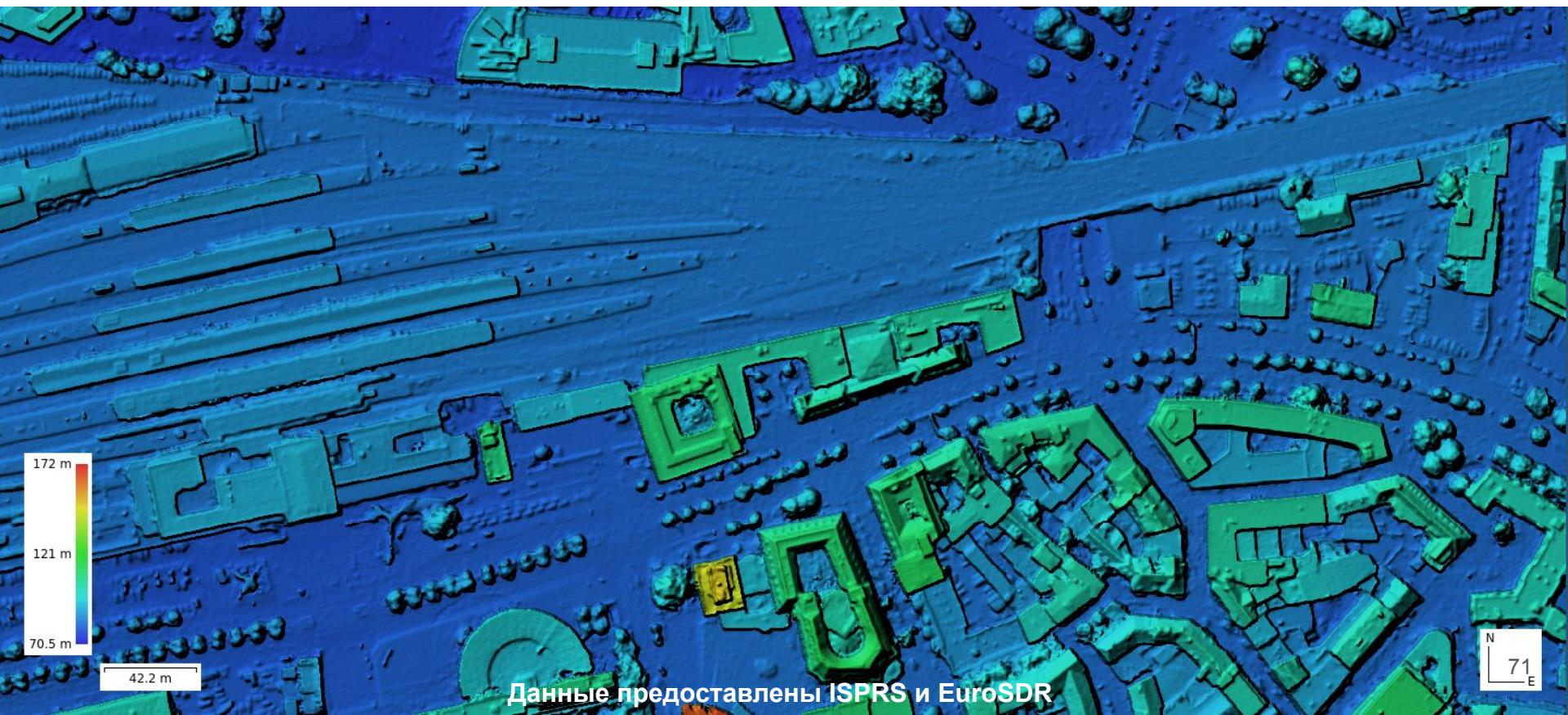
Карта “вид-сверху” и алгоритм Дейкстры

# Фотограмметрия 2.5D

- 1) Сопоставляем фотографии (ключевые точки + дескрипторы)
- 2) Ищем взаимное положение камер
- 3) Строим карты глубины для каждой фотографии
- 4) ~~Строим модель (из треугольников) по картам глубины~~
- 5) ~~Строим текстуру (спроектировав фотографии на модель)~~



# DEM - Digital Elevation Model



# Ортомозаика - объединение проекций фотографий на DEM



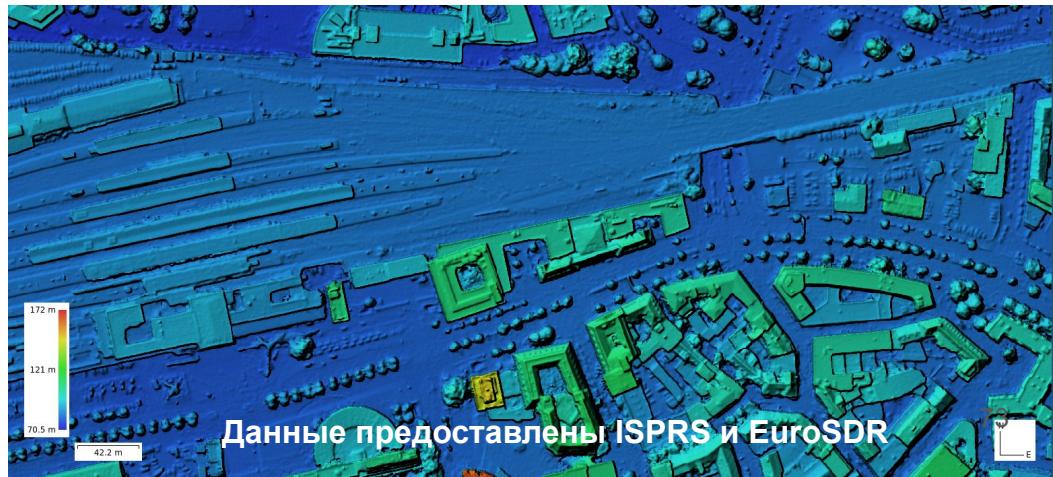
42.2 m

Данные предоставлены ISPRS и EuroSDR

N  
72  
E

# Фотограмметрия 2.5D

- 1) Сопоставляем фотографии (ключевые точки + дескрипторы)
- 2) Ищем взаимное положение камер
- 3) Строим карты глубины для каждой фотографии
- ~~4) Строим модель (из треугольников) по картам глубины~~
- ~~5) Строим текстуру (спроектировав фотографии на модель)~~
- 4) Строим карту высот (DEM)



# Фотограмметрия 2.5D

- 1) Сопоставляем фотографии (ключевые точки + дескрипторы)
- 2) Ищем взаимное положение камер
- 3) Строим карты глубины для каждой фотографии
- ~~4) Строим модель (из треугольников) по картам глубины~~
- ~~5) Строим текстуру (спроектировав фотографии на модель)~~
- 4) Строим карту высот (DEM)
- 5) Строим ортомозаику



Данные предоставлены ISPRS и EuroSDR

Ближайший кадр (с точки зрения nadir-ности)

Фотография 1

Фотография 2

Фотография 3

Фотография 4

Ближайший кадр (с точки зрения nadir-ности)

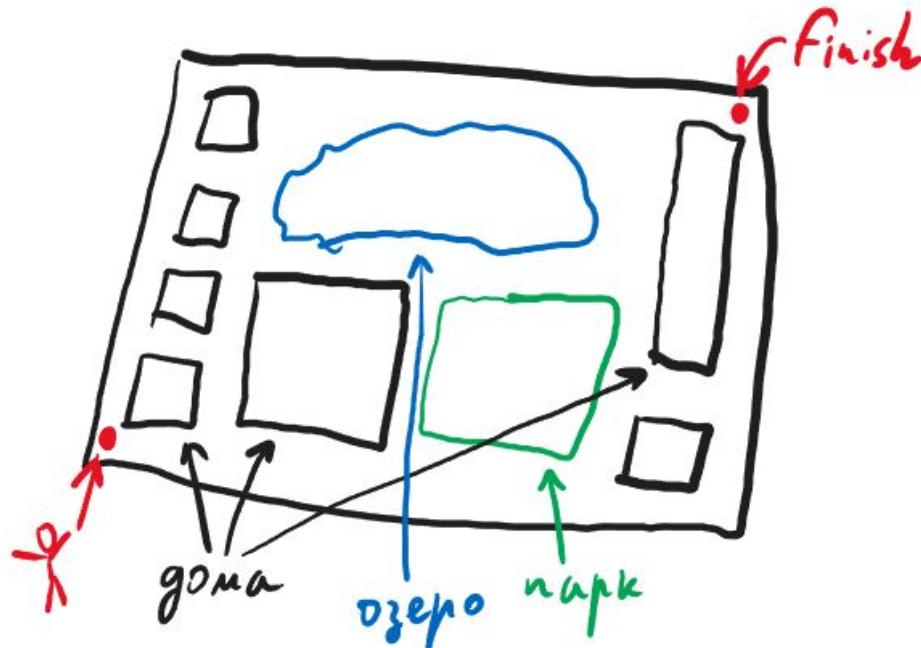
Фотография 1

Фотография 2

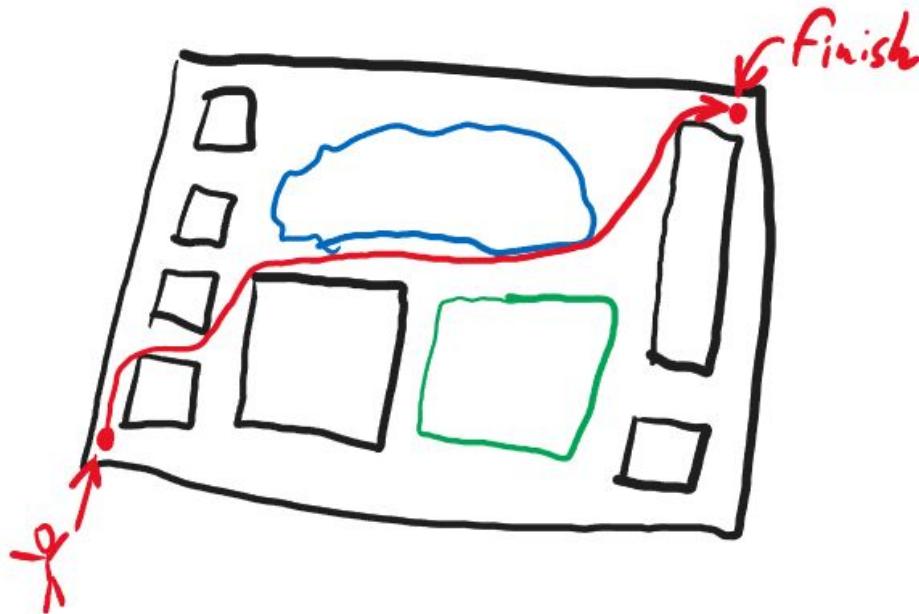
Фотография 3

Фотография 4

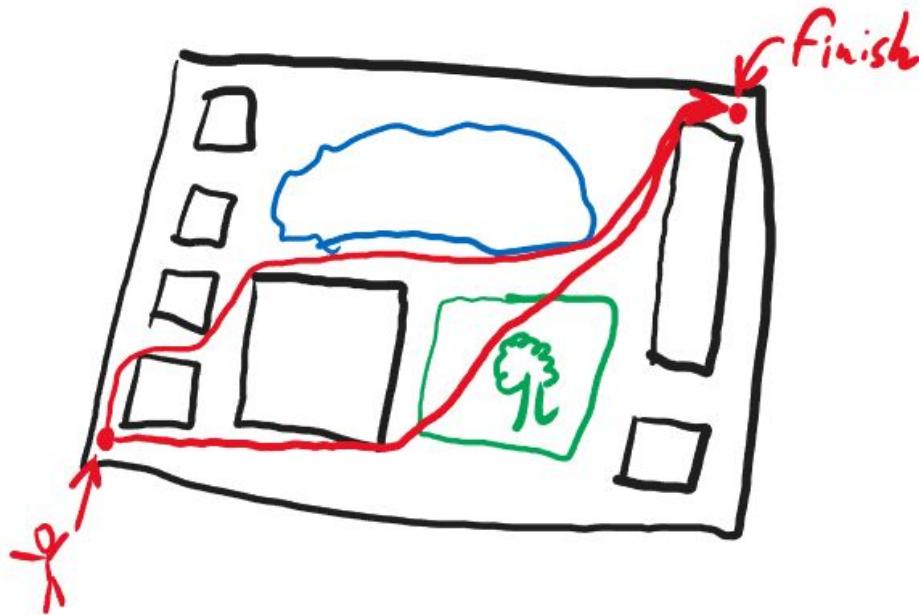
Отвлечемся: найдем кратчайший путь в городе



Отвлечемся: найдем кратчайший путь в городе



Отвлечемся: найдем кратчайший путь в городе



# Отвлечемся: найдем кратчайший путь в городе



Если построим граф, то задача решаема.  
Например **алгоритмом Дейкстры**.

Отвлечемся: найдем кратчайший путь в городе



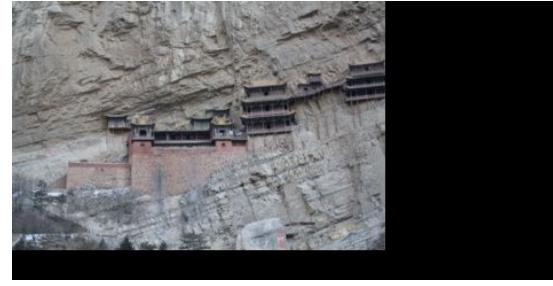
Отвлечемся: найдем кратчайший путь в городе



Отвлечемся: найдем кратчайший путь в городе



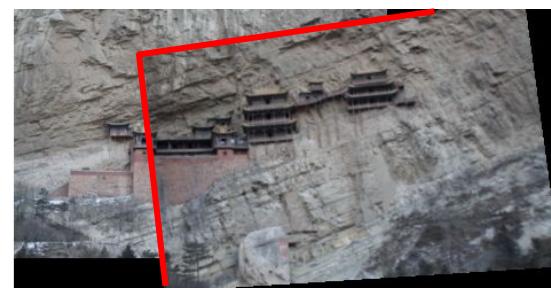
# Прокладка умного шва в панораме



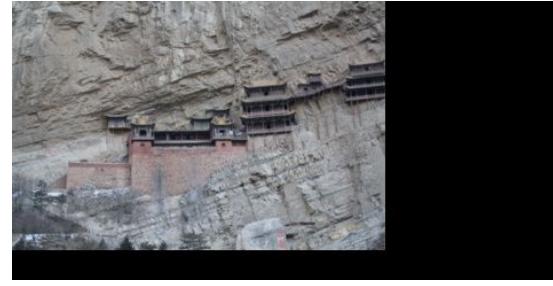
# Прокладка умного шва в панораме



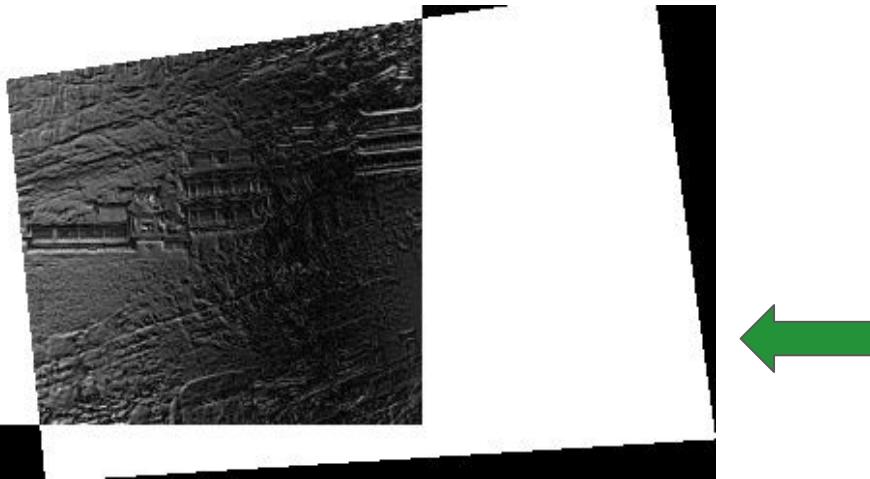
# Прокладка умного шва в панораме



# Прокладка умного шва в панораме



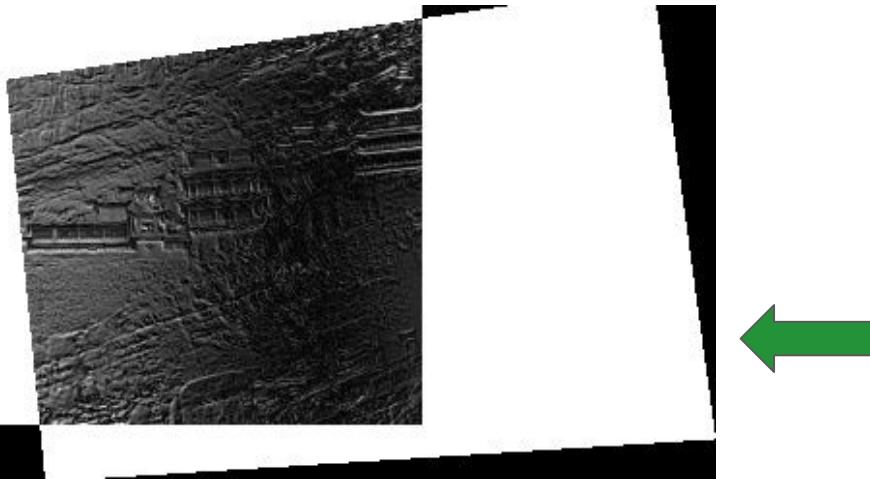
Построим картинку-“панораму” где в каждом пикселе  
написано **насколько отличаются картинки**:



# Прокладка умного шва в панораме

Где тогда нам бы хотелось провести шов?

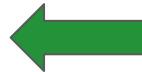
Построим картинку-“панораму” где в каждом пикселе  
написано **насколько отличаются картинки**:



# Прокладка умного шва в панораме



# Прокладка умного шва в панораме

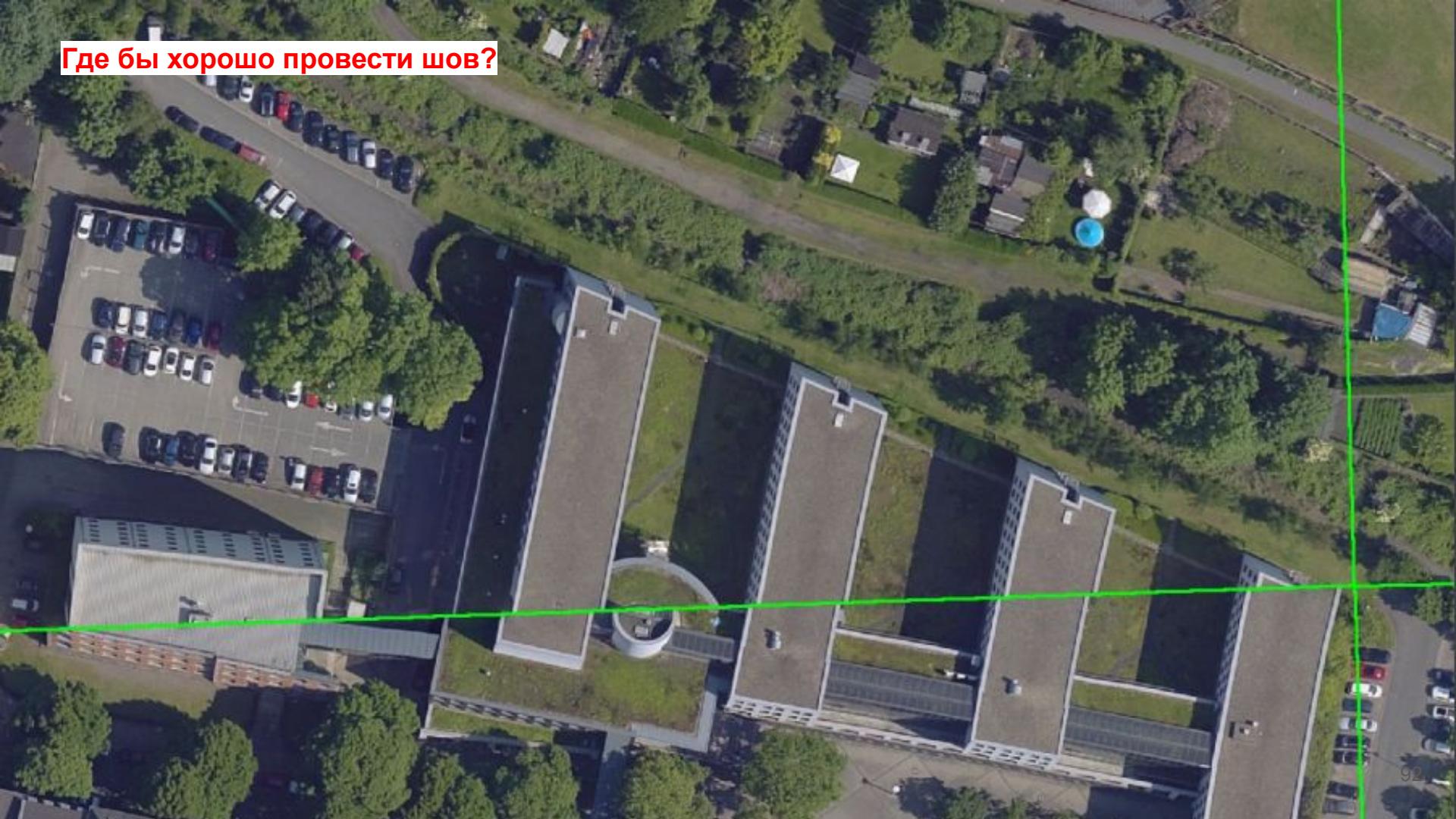


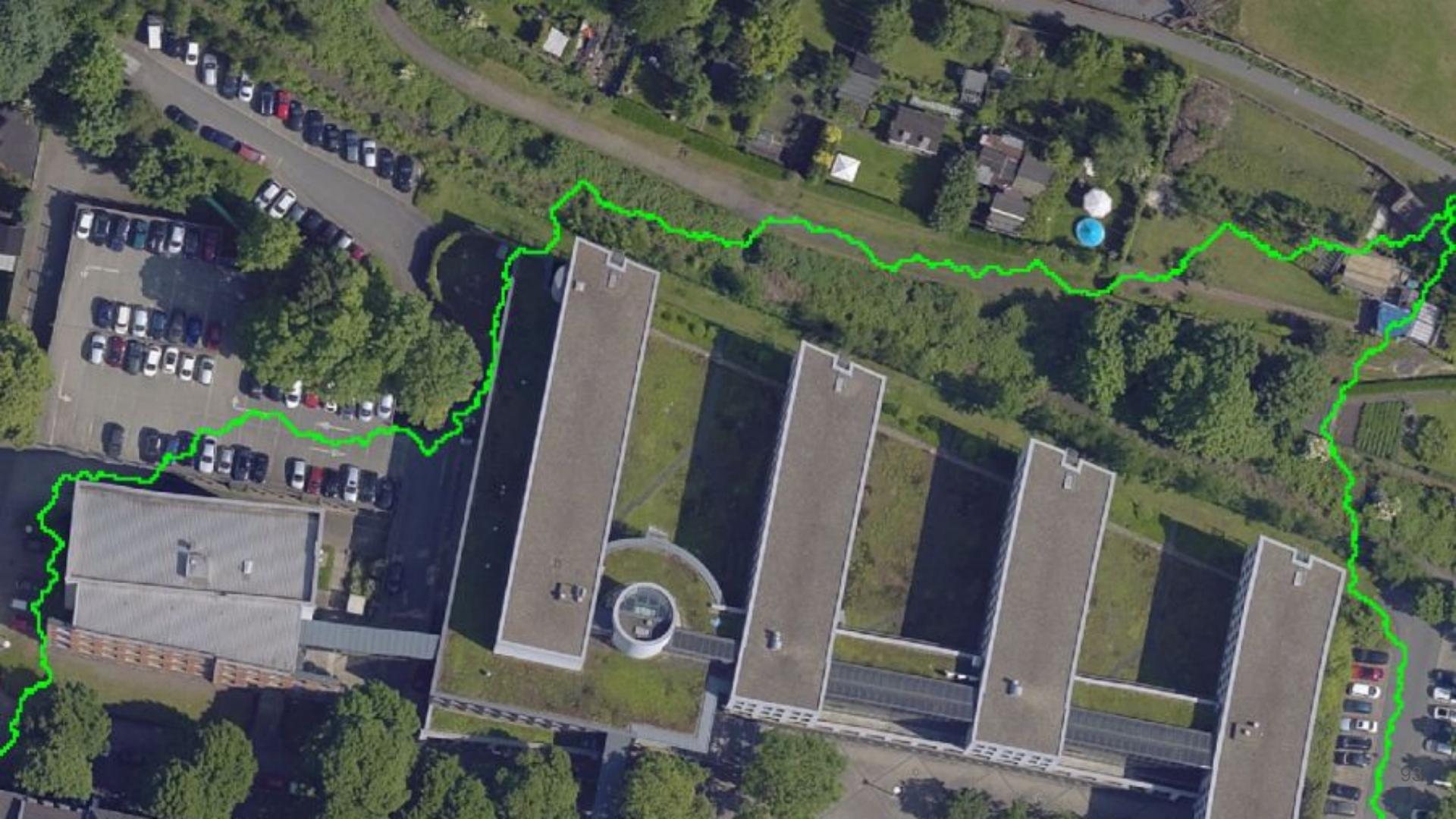
# Прокладка умного шва в панораме

Получили бесшовный результат:



Где бы хорошо провести шов?





# Фотограмметрия 3D и 2.5D

- 1) Сопоставляем фотографии (ключевые точки + дескрипторы)
- 2) Ищем взаимное положение камер
- 3) Строим карты глубины для каждой фотографии

Если нужна **3D** геометрия:

- 4) Строим модель (из треугольников) по картам глубины
- 5) Строим текстуру (спроектировав фотографии на модель)



Данные предоставил Stéphane Prodent

# Фотограмметрия 3D и 2.5D

- 1) Сопоставляем фотографии (ключевые точки + дескрипторы)
- 2) Ищем взаимное положение камер
- 3) Строим карты глубины для каждой фотографии

Если нужна **3D** геометрия:

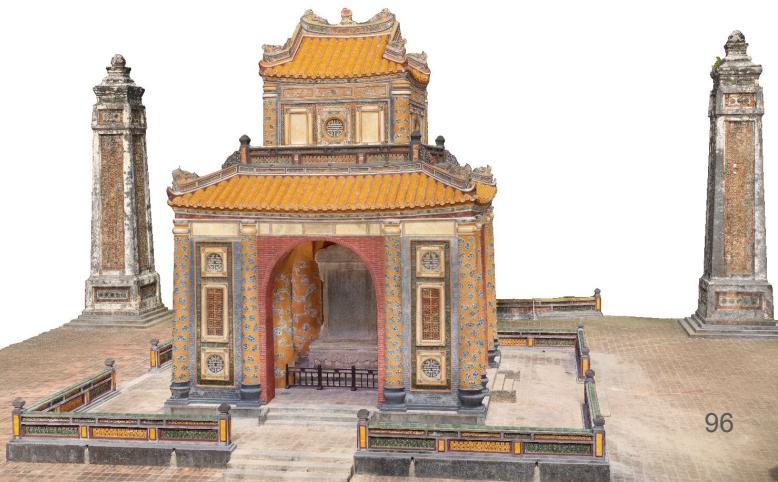
- 4) Строим модель (из треугольников) по картам глубины
- 5) Строим текстуру (спроектировав фотографии на модель)



Если нужна **2.5D** геометрия:

- 4) Строим карту высот (DEM)
- 5) Строим ортомозаику







3D модель Кижского погоста (Agisoft)



3D модель Кижского погоста (Agisoft)



3D модель Кижского погоста (Agisoft)



3D модель Кижского погоста (Agisoft)

# Глава 5

Отличия программирования на видеокартах,  
алгоритм merge-sort

# Видеокарты и массовый параллелизм

A  10    34    12    34    54    113    ...    1

+  
B  32    12    57    12    14    126    ...    5

||  
C  42    46    69    46    68    239    ...    6

# Видеокарты и массовый параллелизм

A

10    34    12    34    54    113 ... 1

+

B

32    12    57    12    14    126 ... 5

||

C

42    46    69    46    68    239 ... 6

```
void solveCPU(int[] a, int[] b, int c[], int n) {  
    for (int i = 0; i < n; ++i) {  
        int sum = a[i] + b[i];  
        c[i] = sum;  
    }  
}
```

# Видеокарты и массовый параллелизм

i=0  
A 

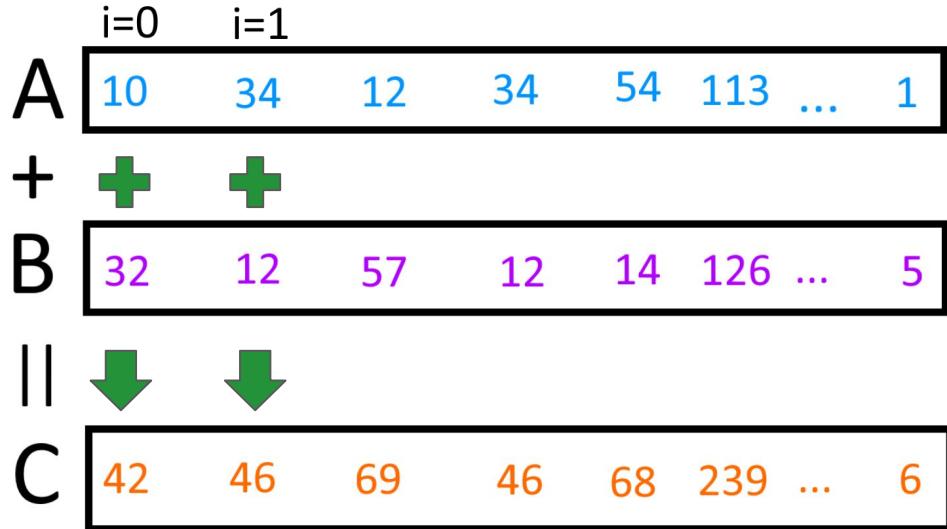
+ 

B 

||   
C 

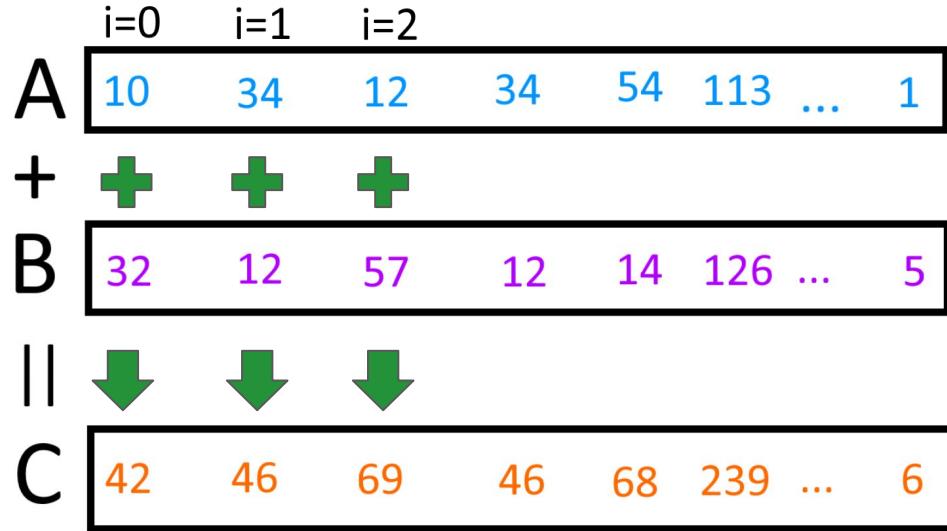
```
void solveCPU(int[] a, int[] b, int c[], int n) {  
    for (int i = 0; i < n; ++i) {  
        int sum = a[i] + b[i];  
        c[i] = sum;  
    }  
}
```

# Видеокарты и массовый параллелизм



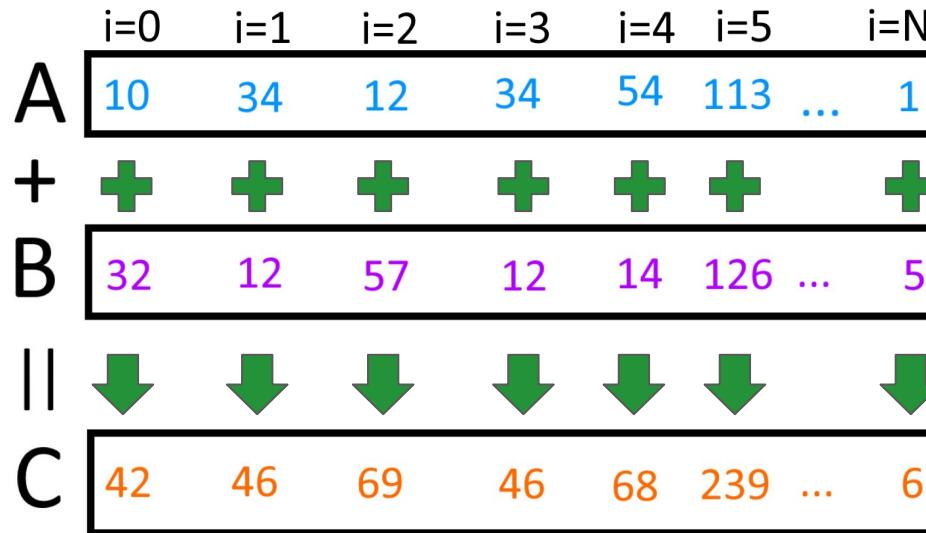
```
void solveCPU(int[] a, int[] b, int c[], int n) {  
    for (int i = 0; i < n; ++i) {  
        int sum = a[i] + b[i];  
        c[i] = sum;  
    }  
}
```

# Видеокарты и массовый параллелизм



```
void solveCPU(int[] a, int[] b, int c[], int n) {  
    for (int i = 0; i < n; ++i) {  
        int sum = a[i] + b[i];  
        c[i] = sum;  
    }  
}
```

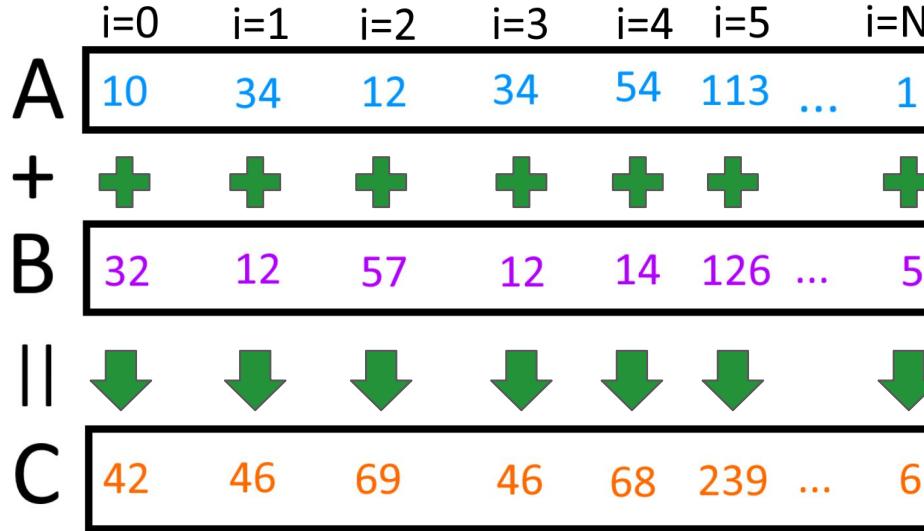
# Видеокарты и массовый параллелизм



```
void solveCPU(int[] a, int[] b, int c[], int n) {  
    for (int i = 0; i < n; ++i) {  
        int sum = a[i] + b[i];  
        c[i] = sum;  
    }  
}
```

Какая  
асимптотика?

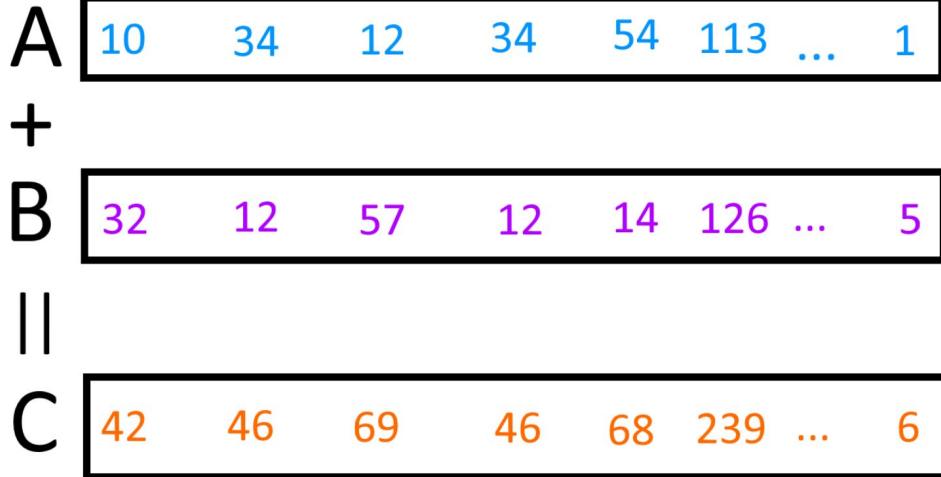
# Видеокарты и массовый параллелизм



```
void solveCPU(int[] a, int[] b, int c[], int n) {  
    for (int i = 0; i < n; ++i) {  
        int sum = a[i] + b[i];  
        c[i] = sum;  
    }  
}
```

**O(N)**

# Видеокарты и массовый параллелизм



**NVIDIA RTX 4090 - 16 тысяч ядер!**



# Видеокарты и массовый параллелизм

A [ 10    34    12    34    54    113 ...    1 ]

+  
B [ 32    12    57    12    14    126 ...    5 ]

||  
C [ 42    46    69    46    68    239 ...    6 ]

~~void solveCPU(int[] a, int[] b, int c[], int n) {  
 for (int i = 0; i < n; ++i) {  
 int sum = a[i] + b[i];  
 c[i] = sum;  
 }  
}~~

~~void solveGPU(int[] a, int[] b, int c[], int n) {  
 const int i = get\_global\_id(0);  
 c[i] = b[i] + a[i];  
}~~ Супер многопоточно!



NVIDIA RTX 4090 - 16 тысяч ядер!



# Видеокарты и массовый параллелизм

A [ 10    34    12    34    54    113 ... 1 ]

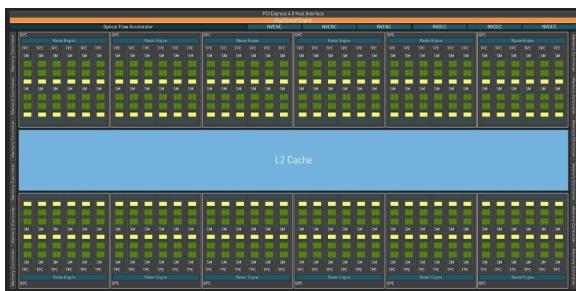
+  
B [ 32    12    57    12    14    126 ... 5 ]

||  
C [ 42    46    69    46    68    239 ... 6 ]

~~void solveCPU(int[] a, int[] b, int c[], int n) {  
 for (int i = 0; i < n; ++i) {  
 int sum = a[i] + b[i];  
 c[i] = sum;  
 }  
}~~

~~void solveGPU(int[] a, int[] b, int c[], int n) {  
 const int i = get\_global\_id(0);  
 c[i] = b[i] + a[i];  
}~~

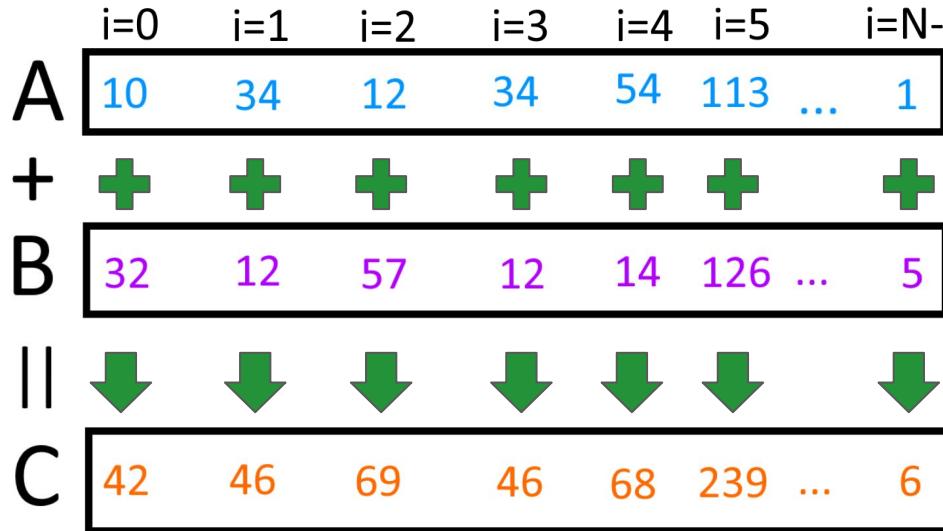
O(N)  
**Какая асимптотика?**



**NVIDIA RTX 4090 - 16 тысяч ядер!**



# Видеокарты и массовый параллелизм



~~void solveCPU(int[] a, int[] b, int c[], int n) {  
 for (int i = 0; i < n; ++i) {  
 int sum = a[i] + b[i];  
 c[i] = sum;  
 }  
}~~

$O(N)$

~~void solveGPU(int[] a, int[] b, int c[], int n) {  
 const int i = get\_global\_id(0);  
 c[i] = b[i] + a[i];  
}~~

$O(N / 16384)$



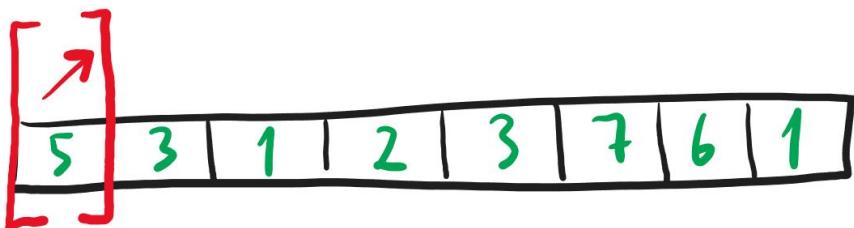
NVIDIA RTX 4090 - 16 тысяч ядер!



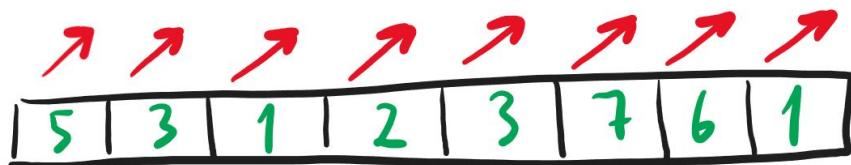
# Merge-sort



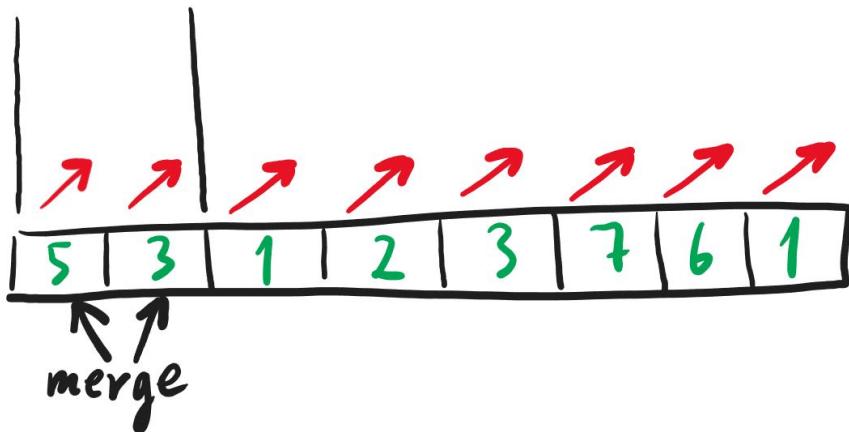
# Merge-sort



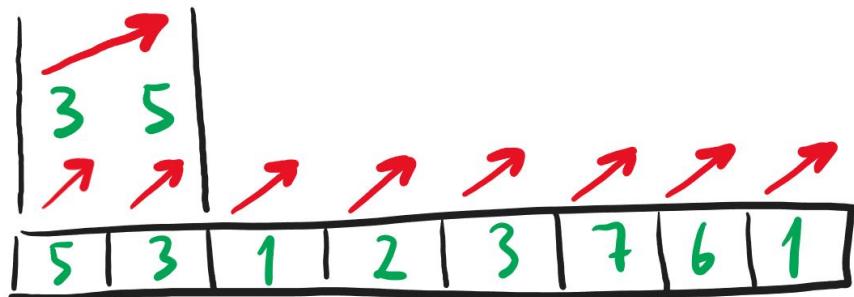
# Merge-sort



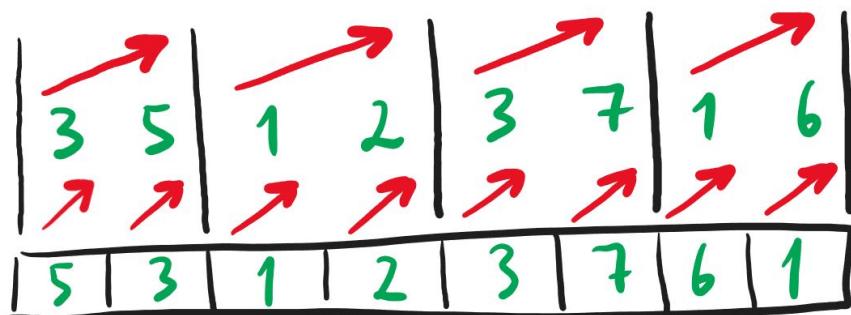
# Merge-sort



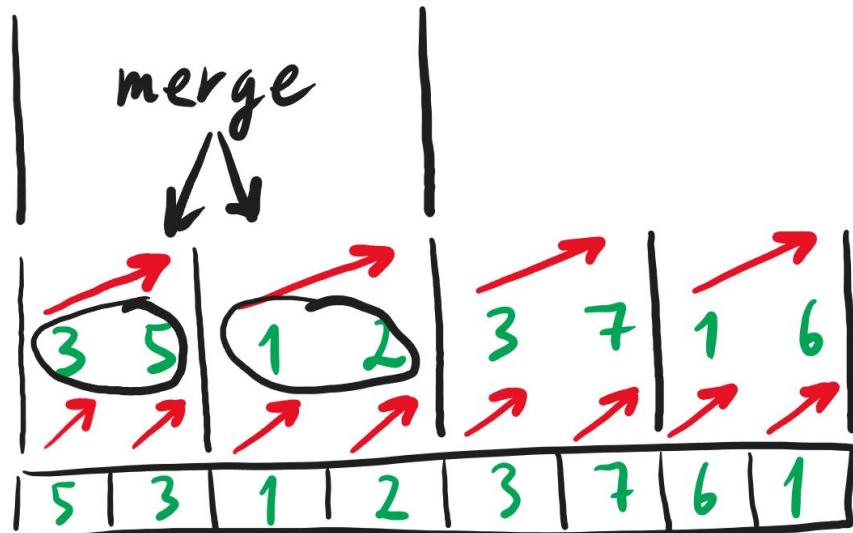
# Merge-sort



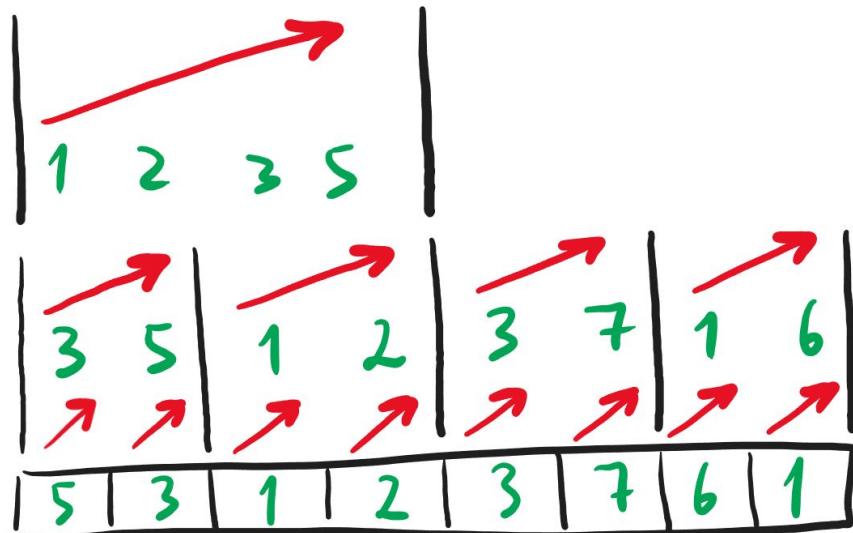
# Merge-sort



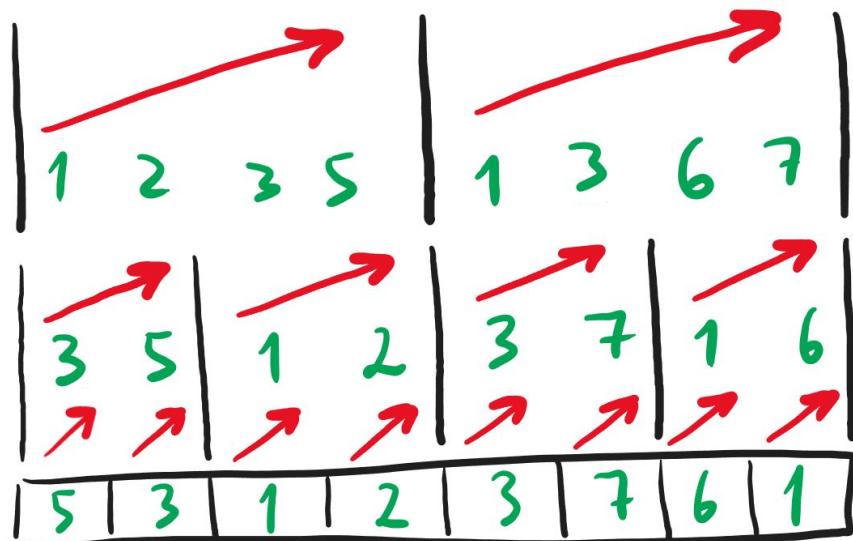
# Merge-sort



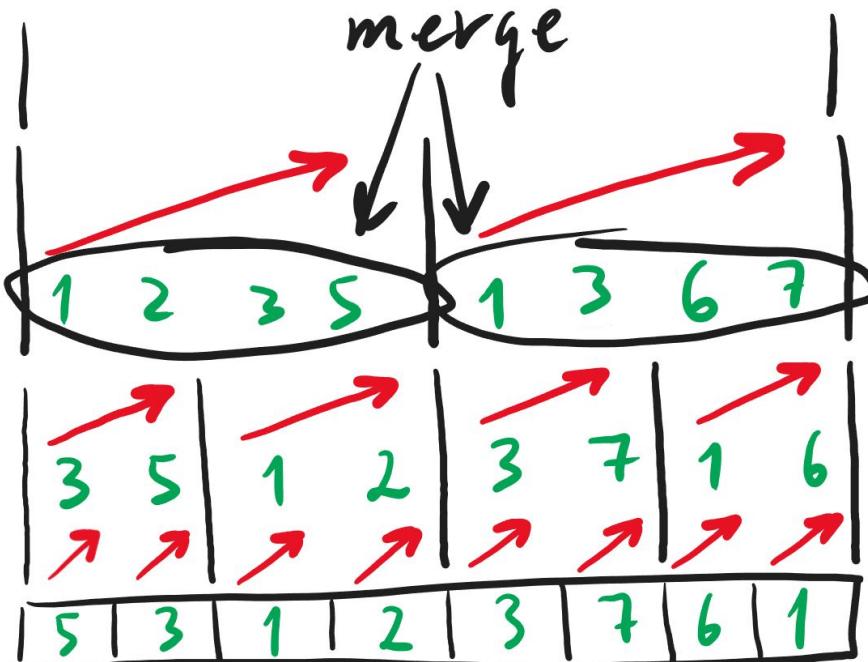
# Merge-sort



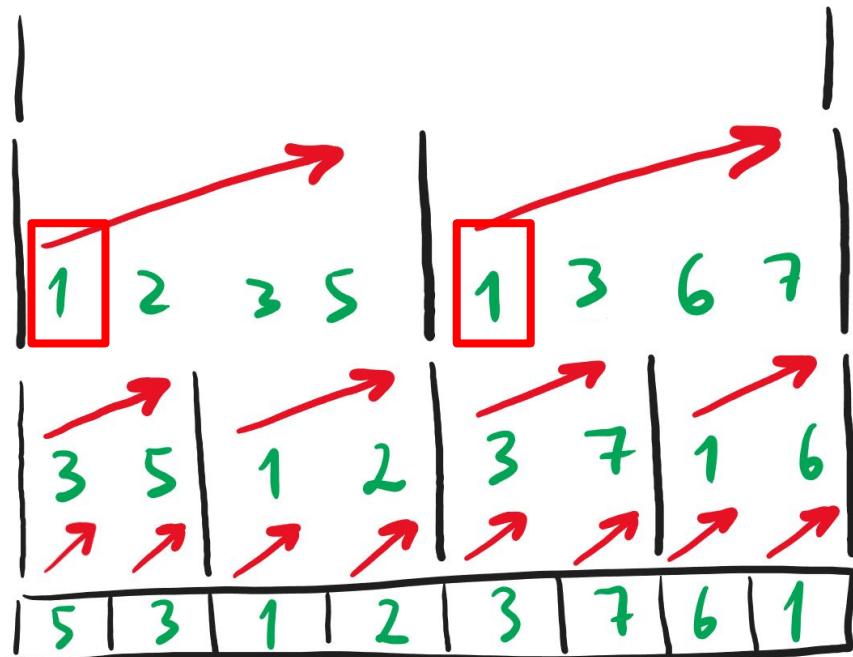
# Merge-sort



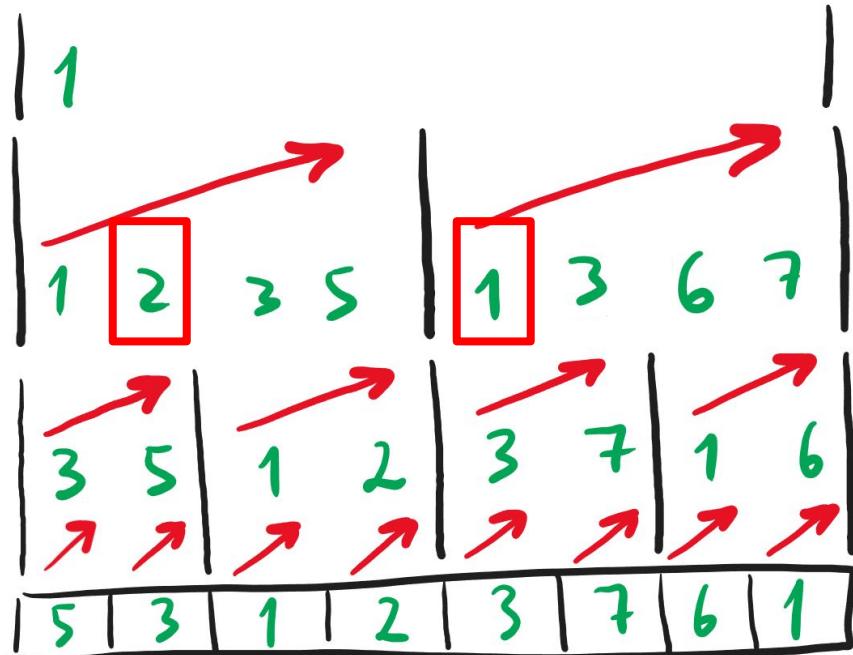
# Merge-sort



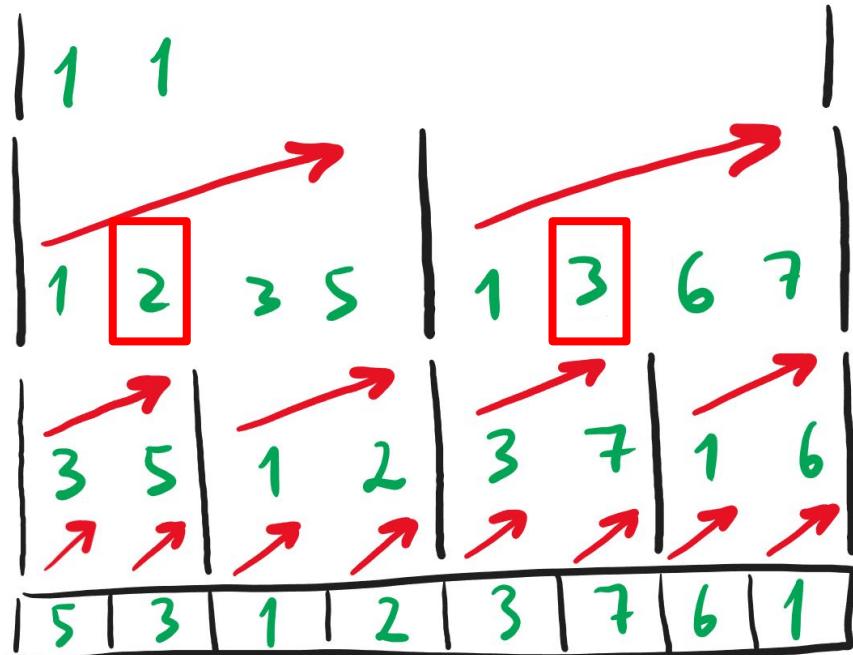
# Merge-sort



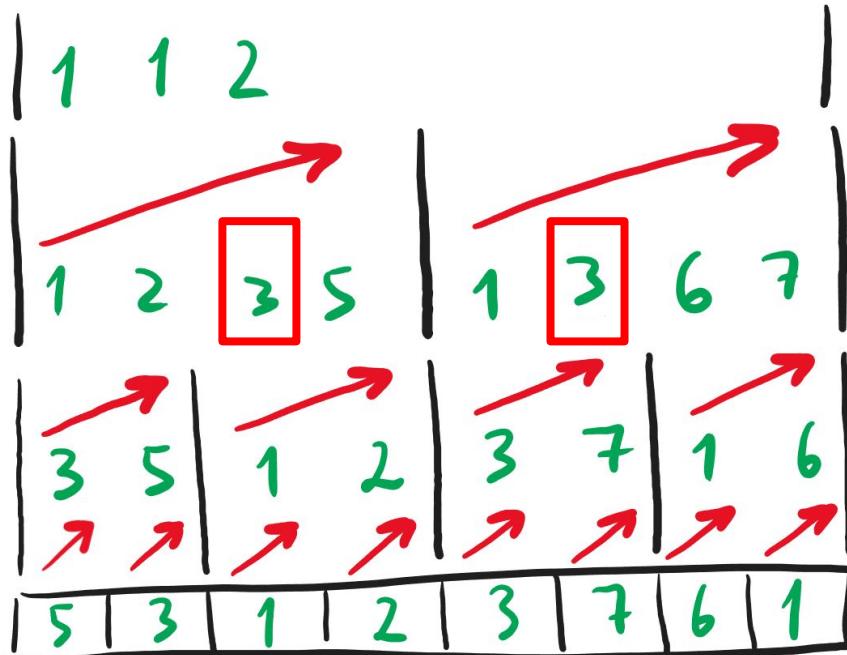
# Merge-sort



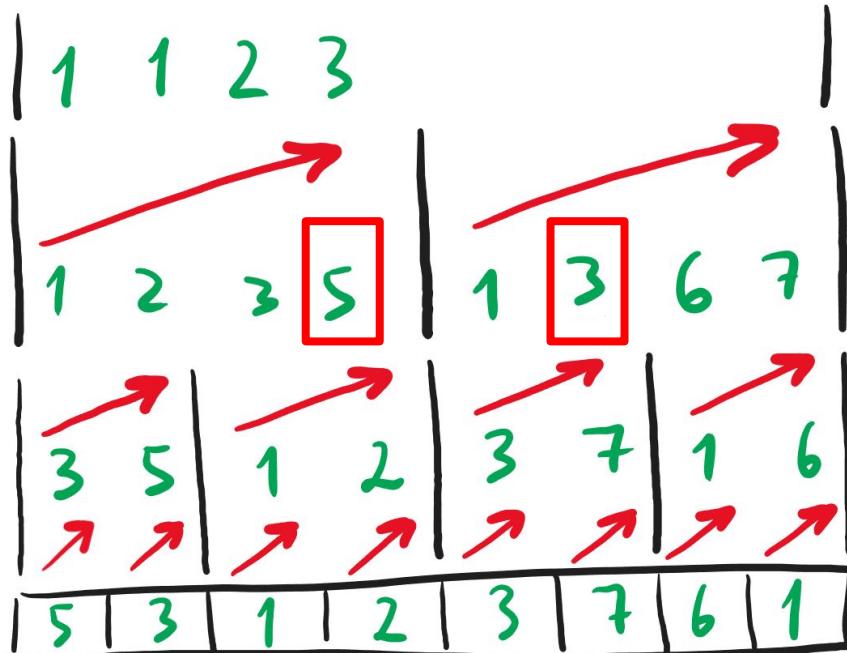
# Merge-sort



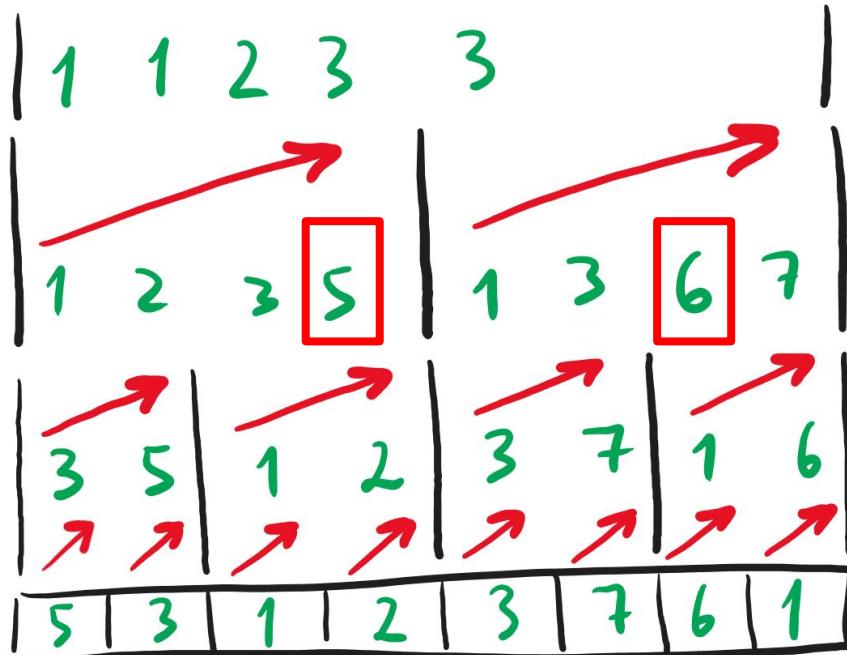
# Merge-sort



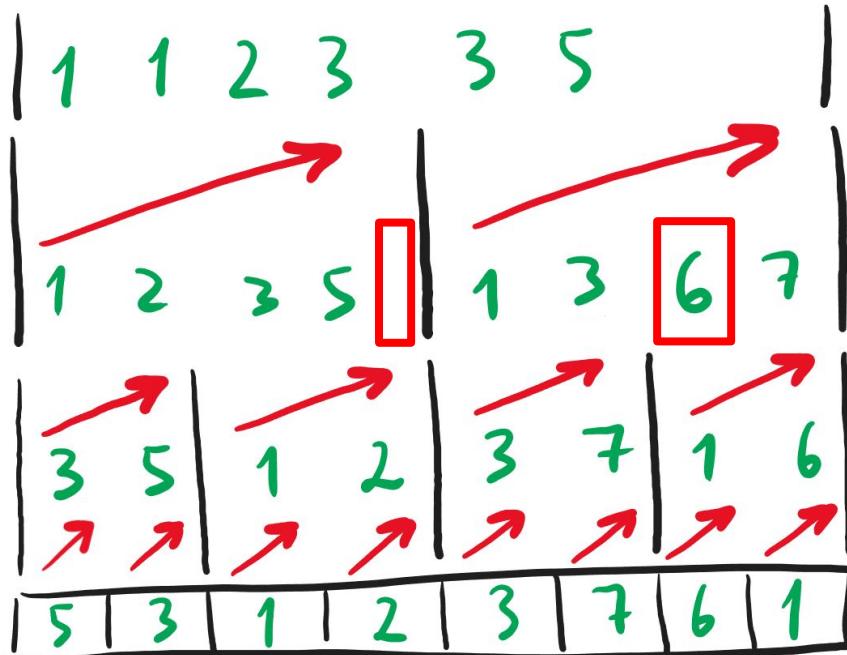
# Merge-sort



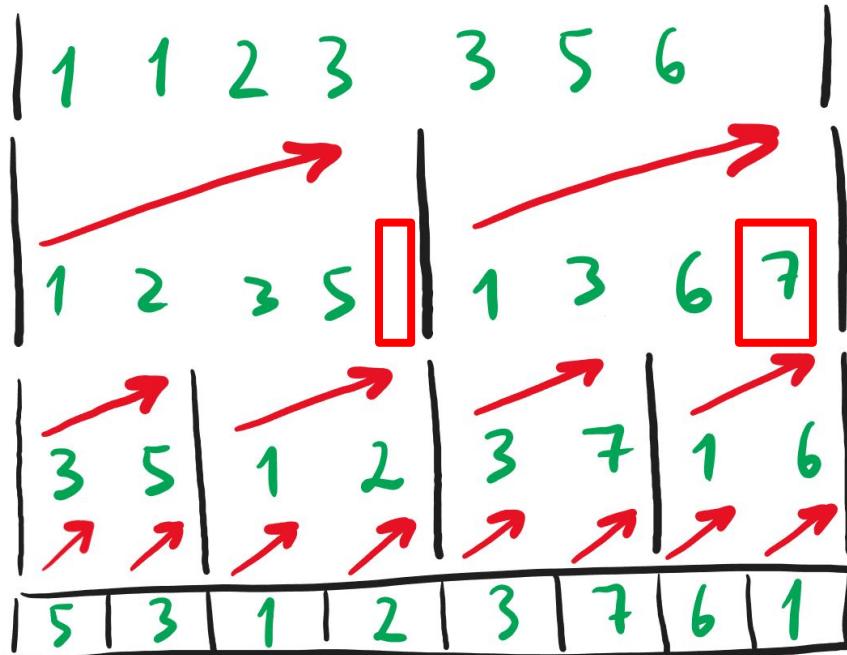
# Merge-sort



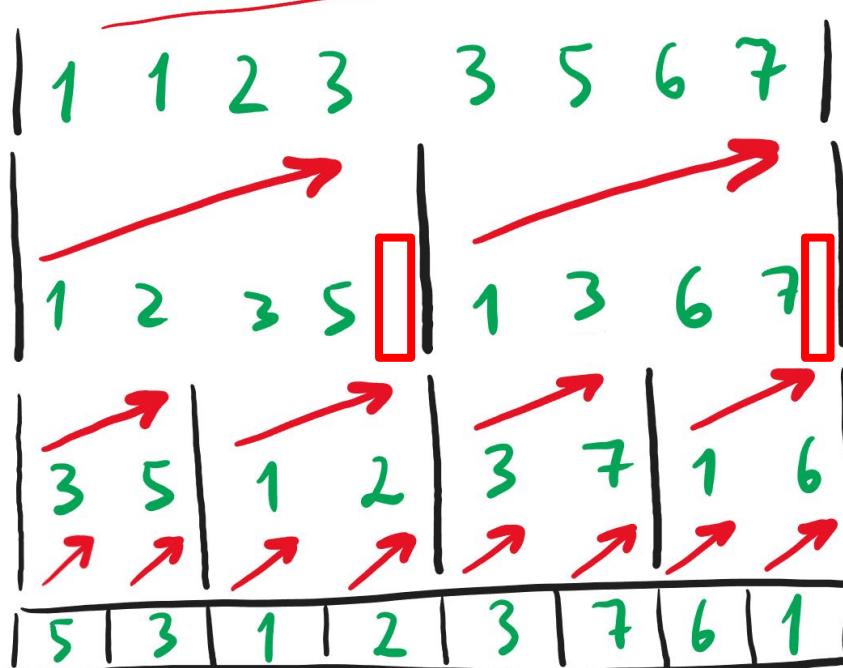
# Merge-sort



# Merge-sort

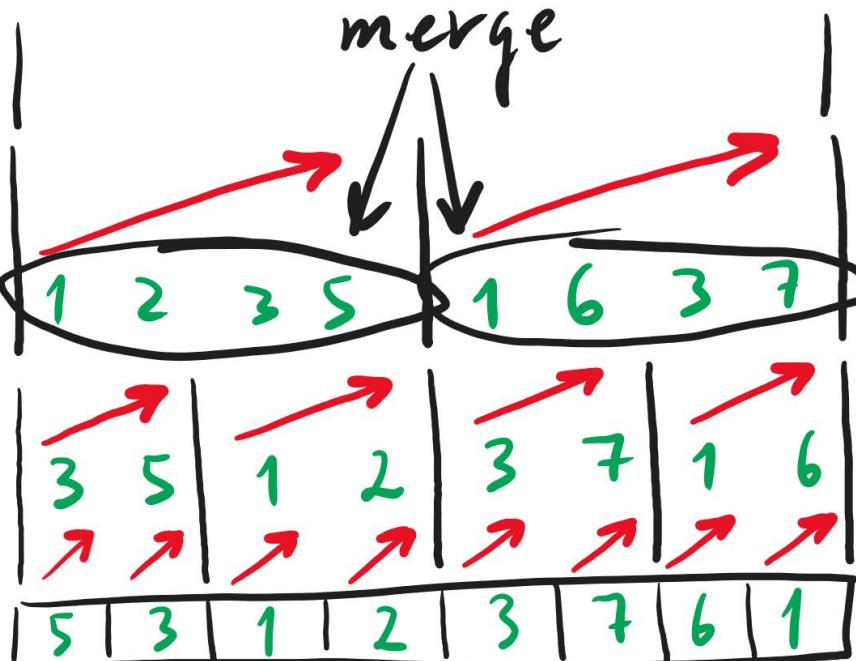


## Merge-sort



Как отсортировать на видеокарте?  
Т.е. как отсортировать супер многопоточно?

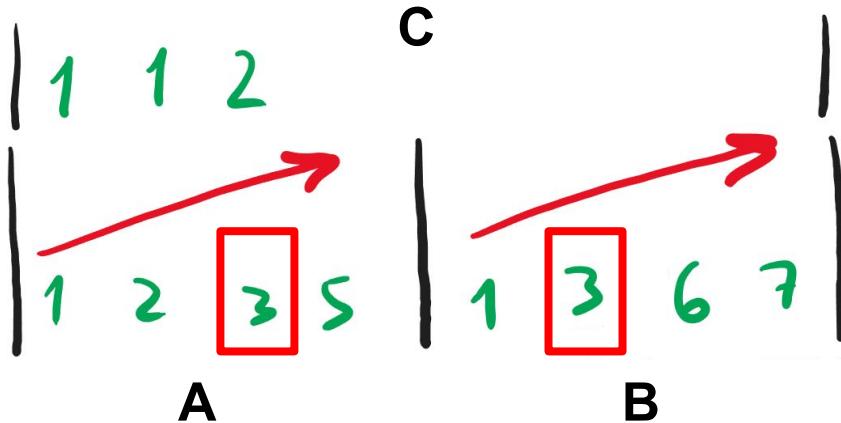
# Merge-sort



Как отсортировать на видеокарте?  
Т.е. как отсортировать супер многопоточно?

Достаточно научиться выполнять  
операцию merge супер многопоточно!

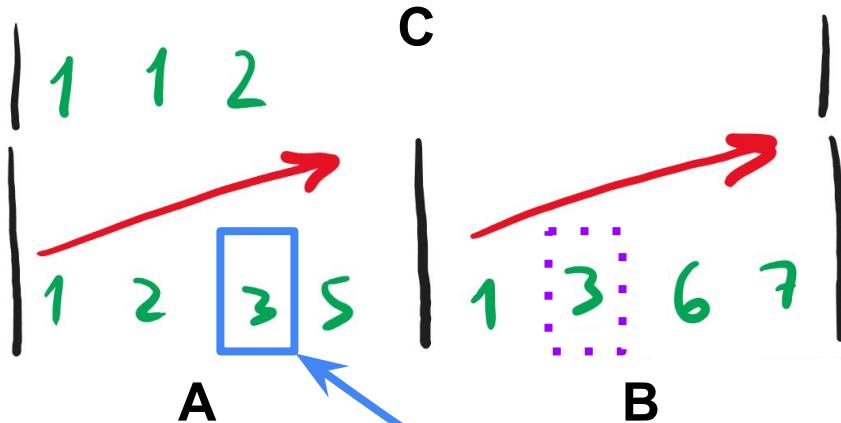
# Merge-sort



Как отсортировать на видеокарте?  
Т.е. как отсортировать супер многопоточно?

Достаточно научиться выполнять  
операцию *merge* супер многопоточно!

# Merge-sort

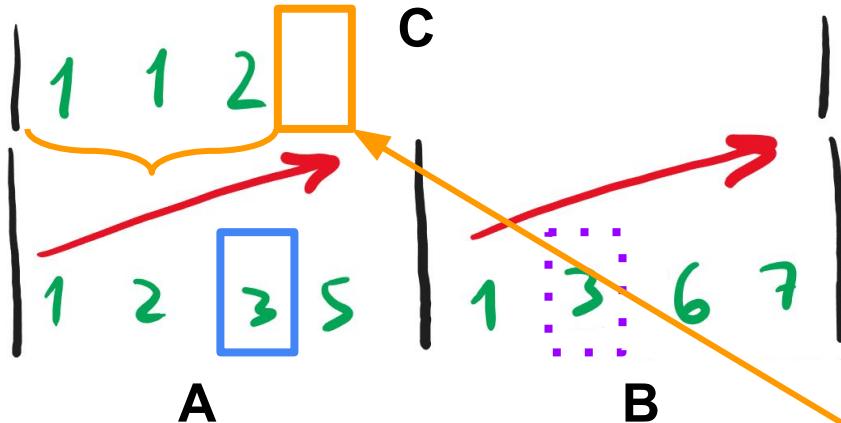


Как отсортировать на видеокарте?  
Т.е. как отсортировать супер многопоточно?

Достаточно научиться выполнять  
операцию *merge* супер многопоточно!

Например один поток - **одно число из А**

## Merge-sort



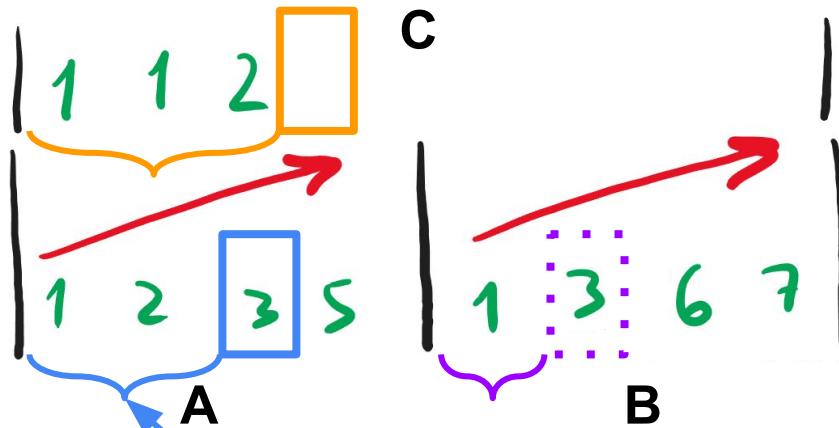
Как отсортировать на видеокарте?  
Т.е. как отсортировать супер многопоточно?

Достаточно научиться выполнять  
операцию *merge* супер многопоточно!

Например один поток - **одно число из А**

На какую **позицию в С** записать его?

## Merge-sort



Знаем ли мы сколько чисел до нас в А?

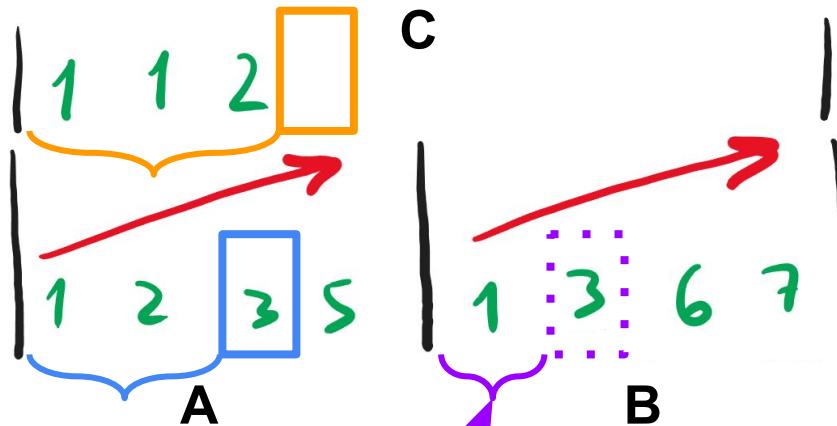
Как отсортировать на видеокарте?  
Т.е. как отсортировать супер многопоточно?

Достаточно научиться выполнять  
операцию *merge* супер многопоточно!

Например один поток - **одно число из А**

На какую **позицию в С** записать его?

# Merge-sort



Знаем ли мы сколько чисел до нас в А?

А как найти сколько элементов в В  
меньше чем наше число из А?

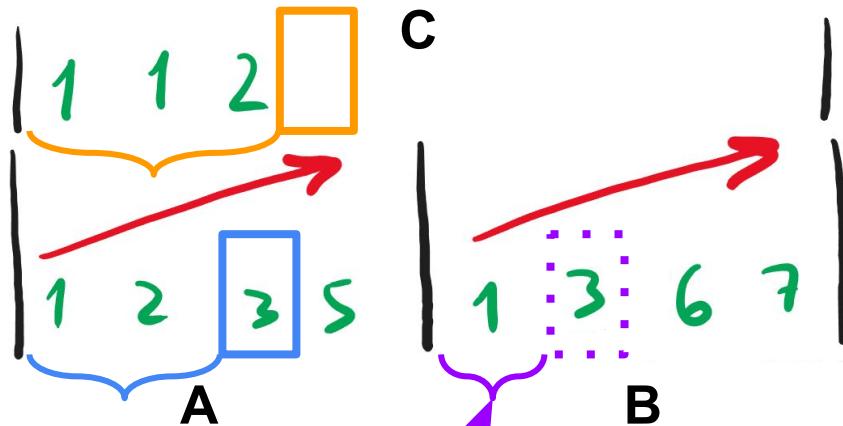
Как отсортировать на видеокарте?  
Т.е. как отсортировать супер многопоточно?

Достаточно научиться выполнять  
операцию merge супер многопоточно!

Например один поток - **одно число из А**

На какую позицию в С записать его?

# Merge-sort



Знаем ли мы сколько чисел до нас в А?

А как найти сколько элементов в В  
меньше чем наше число из А?

Бинарный поиск!  $O(\log N)$

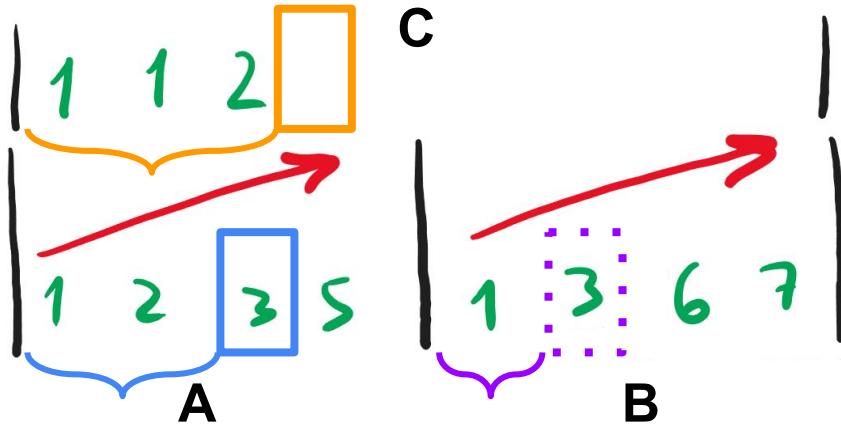
Как отсортировать на видеокарте?  
Т.е. как отсортировать супер многопоточно?

Достаточно научиться выполнять  
операцию merge супер многопоточно!

Например один поток - **одно число из А**

На какую позицию в С записать его?

## Merge-sort



Пусть размер **A** и **B** -  $N$  элементов.

Пусть у нас **K** ядер в видеокарте.

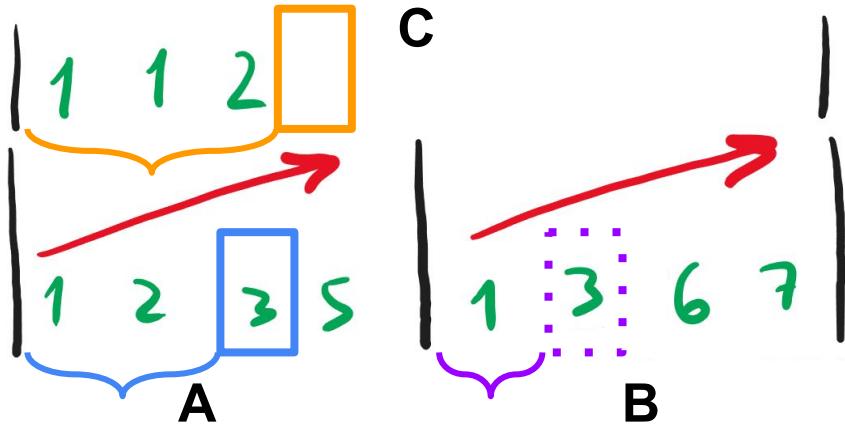
Какая асимптотика **merge** операции?

Знаем ли мы сколько чисел до нас в **A**?

А как найти сколько элементов в **B** меньше чем наше число из **A**?

Бинарный поиск!  $O(\log N)$

## Merge-sort



Пусть размер **A** и **B** -  $N$  элементов.  
Пусть у нас **K** ядер в видеокарте.

Какая асимптотика **merge** операции?

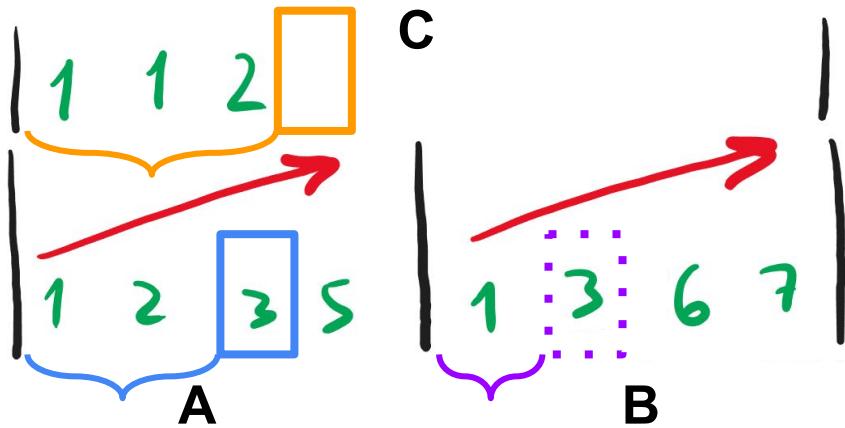
$O(?) * O(\log N) / ?)$

Знаем ли мы сколько чисел до нас в **A**?

А как найти сколько элементов в **B**  
меньше чем наше число из **A**?

Бинарный поиск!  $O(\log N)$

## Merge-sort



Пусть размер **A** и **B** -  $N$  элементов.  
Пусть у нас **K** ядер в видеокарте.

Какая асимптотика **merge** операции?

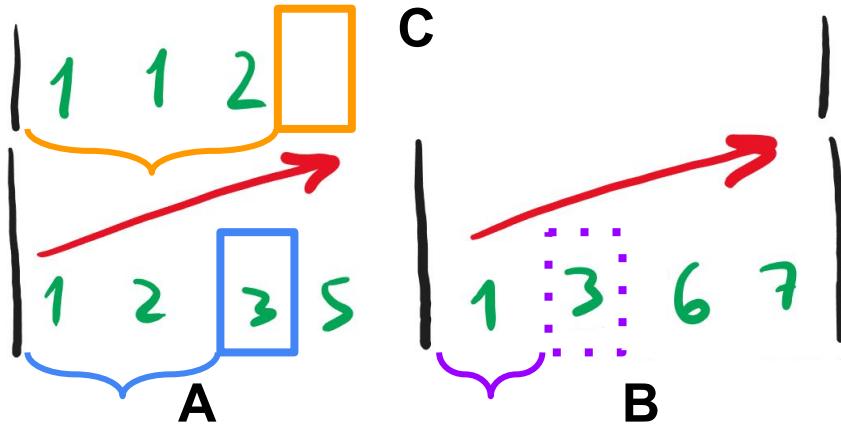
$O(N * O(\log N) / ?)$

Знаем ли мы сколько чисел до нас в **A**?

А как найти сколько элементов в **B**  
меньше чем наше число из **A**?

Бинарный поиск!  $O(\log N)$

## Merge-sort



Пусть размер **A** и **B** -  $N$  элементов.  
Пусть у нас **K** ядер в видеокарте.

Какая асимптотика **merge** операции?

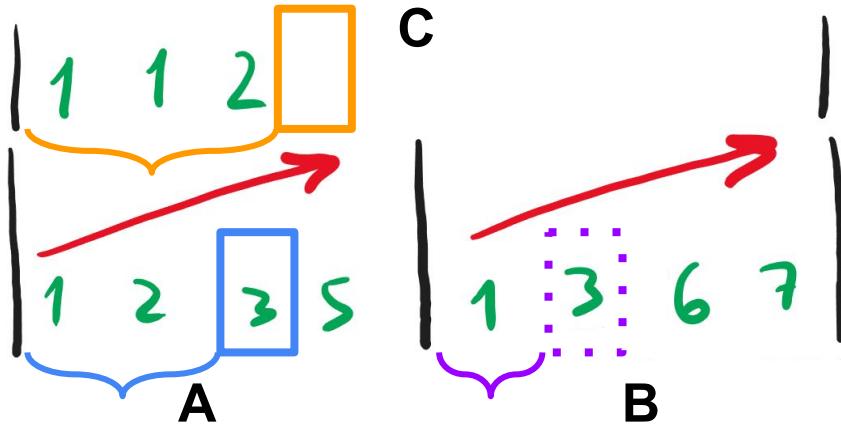
$O(N * O(\log N) / K)$

Знаем ли мы сколько чисел до нас в **A**?

А как найти сколько элементов в **B**  
меньше чем наше число из **A**?

Бинарный поиск!  $O(\log N)$

# Merge-sort



Пусть размер **A** и **B** -  $N$  элементов.

Пусть у нас **K** ядер в видеокарте.

Какая асимптотика **merge** операции?

$O(N * O(\log N) / K)$

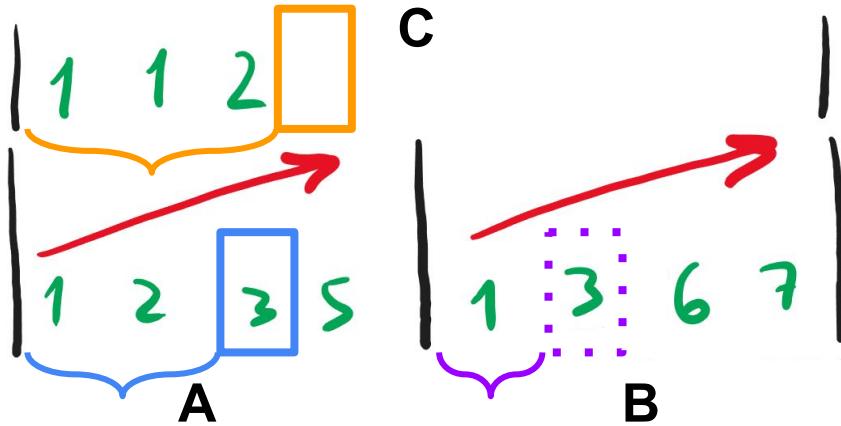
Всего у нас  $O(\log N)$  операций слияния

Знаем ли мы сколько чисел до нас в **A**?

А как найти сколько элементов в **B** меньше чем наше число из **A**?

Бинарный поиск!  $O(\log N)$

# Merge-sort



Знаем ли мы сколько чисел до нас в A?

А как найти сколько элементов в B  
меньше чем наше число из A?

Бинарный поиск!  $O(\log N)$

Пусть размер A и B - N элементов.

Пусть у нас K ядер в видеокарте.

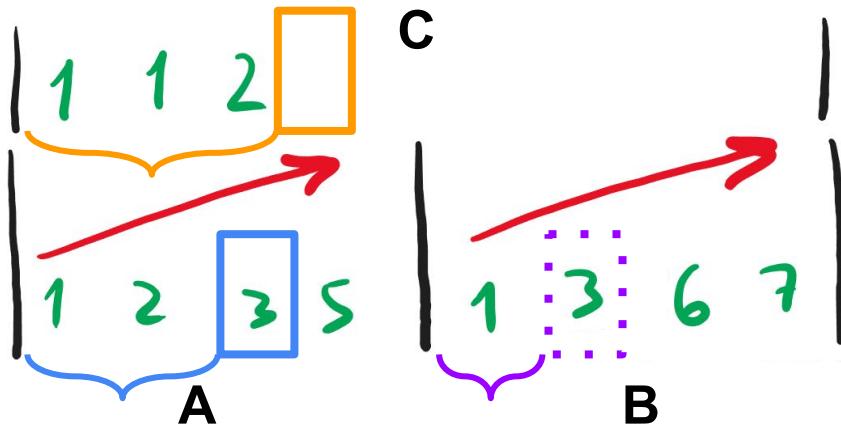
Какая асимптотика merge операции?

$O(N * O(\log N) / K)$

Всего у нас  $O(\log N)$  операций слияния

Какая асимптотика на CPU?

# Merge-sort



Знаем ли мы сколько чисел до нас в A?

А как найти сколько элементов в B  
меньше чем наше число из A?

Бинарный поиск!  $O(\log N)$

Пусть размер A и B - N элементов.

Пусть у нас K ядер в видеокарте.

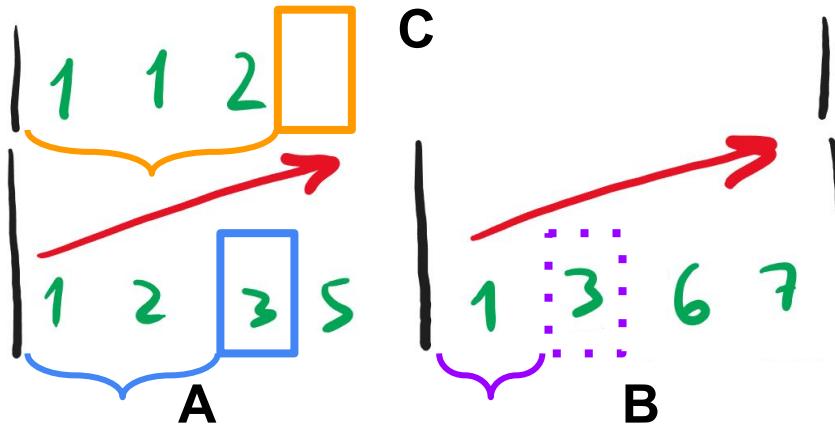
Какая асимптотика merge операции?

$O(N * O(\log N) / K)$

Всего у нас  $O(\log N)$  операций слияния

Какая асимптотика на CPU?  $O(\log N * N)$

# Merge-sort



Знаем ли мы сколько чисел до нас в A?

А как найти сколько элементов в B  
меньше чем наше число из A?

Бинарный поиск!  $O(\log N)$

Пусть размер A и B - N элементов.

Пусть у нас K ядер в видеокарте.

Какая асимптотика merge операции?

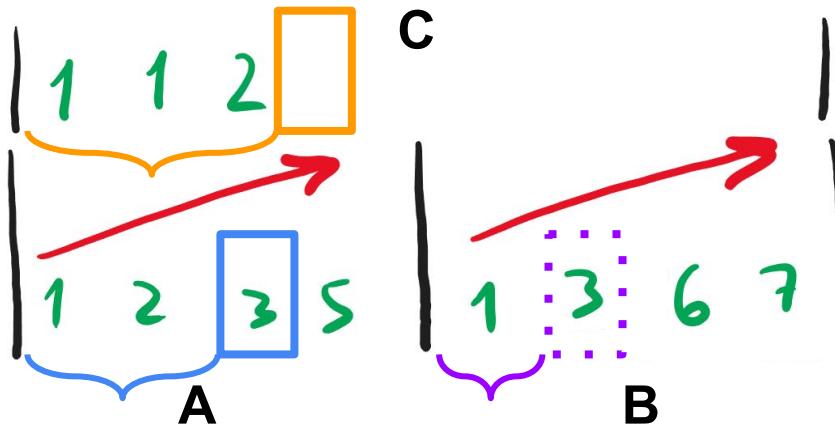
$O(N * O(\log N) / K)$

Всего у нас  $O(\log N)$  операций слияния

Какая асимптотика на CPU?  $O(\log N * N)$

A на GPU?  $O(\log N * ?)$

# Merge-sort



Знаем ли мы сколько чисел до нас в A?

А как найти сколько элементов в B  
меньше чем наше число из A?

Бинарный поиск!  $O(\log N)$

Пусть размер A и B - N элементов.

Пусть у нас K ядер в видеокарте.

Какая асимптотика merge операции?

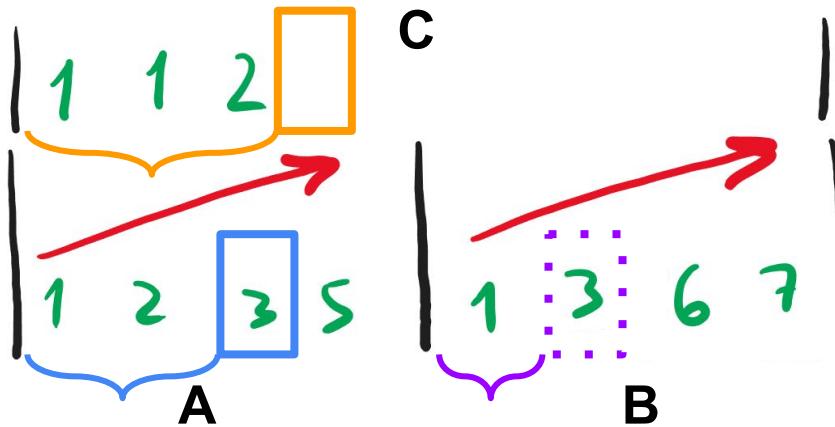
$O(N * O(\log N) / K)$

Всего у нас  $O(\log N)$  операций слияния

Какая асимптотика на CPU?  $O(\log N * N)$

А на GPU?  $O(\log N * ?)$

# Merge-sort



Знаем ли мы сколько чисел до нас в A?

А как найти сколько элементов в B  
меньше чем наше число из A?

Бинарный поиск!  $O(\log N)$

Пусть размер A и B - N элементов.

Пусть у нас K ядер в видеокарте.

Какая асимптотика merge операции?

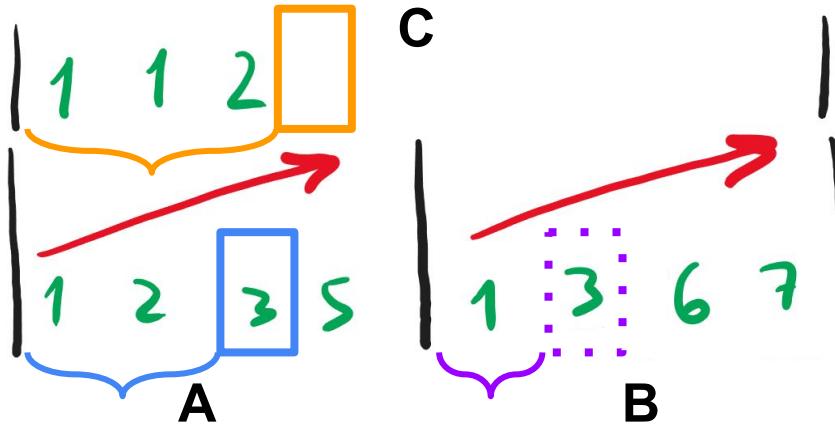
$O(N * O(\log N) / K)$

Всего у нас  $O(\log N)$  операций слияния

Какая асимптотика на CPU?  $O(\log N * N)$

А на GPU?  $O(\log N * N * \log N / K)$

# Merge-sort



Знаем ли мы сколько чисел до нас в A?

А как найти сколько элементов в B  
меньше чем наше число из A?

Бинарный поиск!  $O(\log N)$

Пусть размер A и B - N элементов.

Пусть у нас K ядер в видеокарте.

Какая асимптотика merge операции?

$O(N * O(\log N) / K)$

Всего у нас  $O(\log N)$  операций слияния

Какая асимптотика на CPU?  $O(\log N * N)$

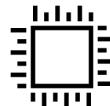
А на GPU?  $O(\log N * N * \log N / K)$

Отличается в  $(\log N / K)$  раз, K - большое!

## Merge-sort

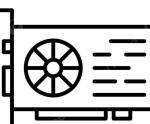
$$N = 33.554.432 \sim 3 \cdot 10^7$$

CPU - Intel 13700K  
 $O(\log N * N)$



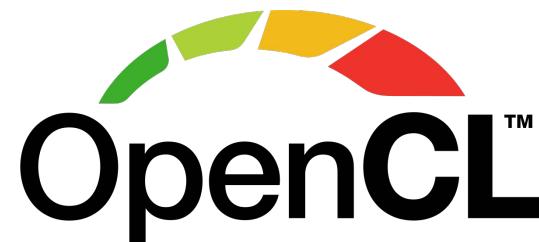
6.462 секунд

GPU - NVIDIA RTX 4090  
 $O(\log N * N * \log N / K)$



0.011 секунд  
**Быстрее в 582 раза!**

Отличается в  $(\log N / K)$  раз,  
где  $K = 16$  тысяч ядер



# Глава 6

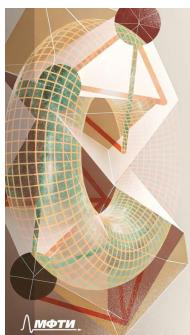
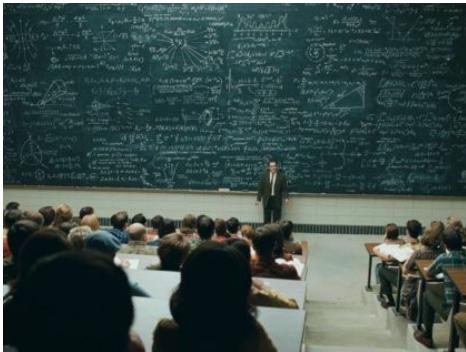
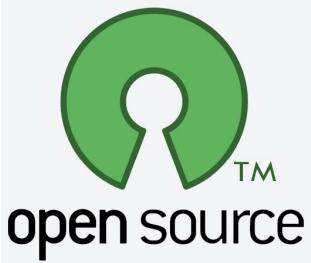
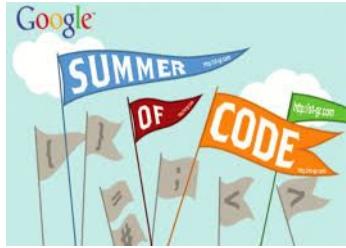
Как жить?

# Как жить?



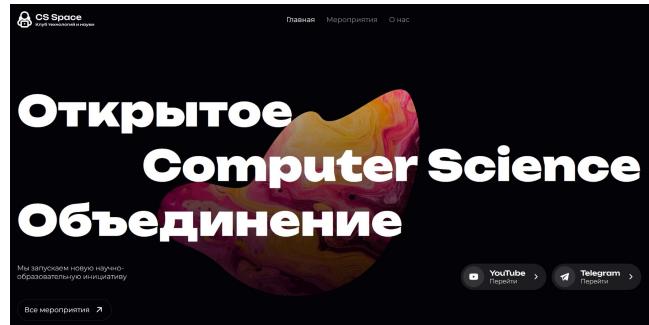
EUROPEAN CONFERENCE ON COMPUTER VISION

- Искать интересное (области)



Летняя  
исследова-  
тельская  
программа  
студентов

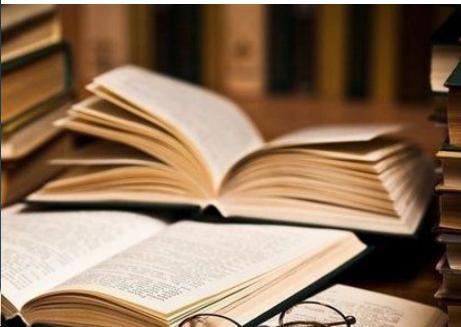
7 июля -  
8 августа  
2025



ШКОЛА АНАЛИЗА ДАННЫХ 152

# Как жить?

- Искать интересное (области)
- Искать интересное (увлечения)



# Как жить?

- Искать интересное (области)
- Искать интересное (увлечения)
- Спорт



# Как жить?

- Искать интересное (области)
- Искать интересное (увлечения)
- Спорт
- Друзья, семья

**НИКТО:  
ПАРНИ НАХОДЯТ З-Х ДРУЗЕЙ  
В 15 ЛЕТ И РЕШАЮТ, ЧТО  
БОЛЬШЕ СОЦИАЛИЗИРОВАТЬСЯ  
ДО КОНЦА ЖИЗНИ ИМ НЕ НУЖНО**



# Как жить?

- Искать интересное (области)
- Искать интересное (увлечения)
- Спорт
- Друзья, семья
- Не недооценивайте себя



# Курс по Фотограмметрии

Курс читается в:

- ИТМО: **КТ, ИПКН**
- СПбГУ: **МКН** (в т.ч. направление Современного Программирования)

Если вы в другом вузе но вам интересно слушать эти лекции - просто напишите что хотите посещать как внешний студент.

Обычно фотограмметрия читается весной, курс по GPU - осенью.

Лекции доступны на youtube - можно найти по “[Курс Фотограмметрии 2025](#)”

Домашние задание - <https://github.com/PhotogrammetryCourse>

# Вопросы?

Видеозапись будет тут:



Agisoft

Metashape

@PolarNick239

[polarnick@agisoft.com](mailto:polarnick@agisoft.com)

Полярный Николай Вадимович