

Введение в фотограмметрию

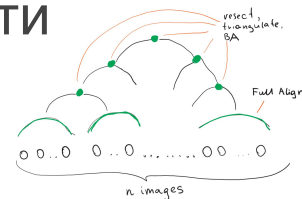
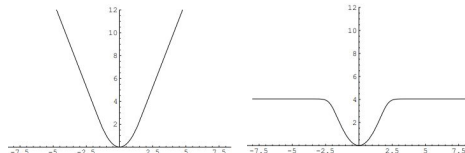
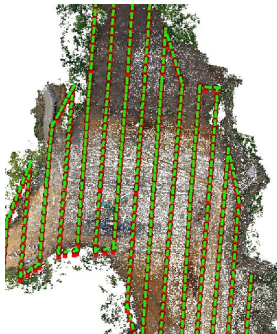
Bundle Adjustment

Ceres Solver

Фотограмметрия. Лекция 10

- Учет Rolling Shutter
- Loss functions
- Фильтрация выбросов
- Ceres Solver и Dual Numbers
- Автоматическое дифференцирование
- Hierarchical Bundle Adjustment
- Контроль точности

```
struct Jet {  
    // The scalar part.  
    T a;  
    // The infinitesimal part.  
    Eigen::Matrix<T, N, 1> v;  
};
```



Полярный Николай

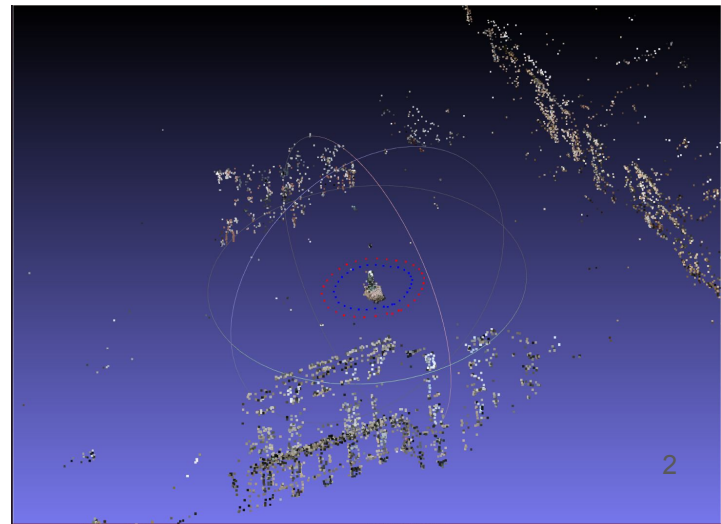
polarnick239@gmail.com

Какие уже есть этапы? Что мы хотим сделать?

- 1) Ключевые точки (SIFT)
- 2) Сопоставление ключевых точек (RANSAC, K-ratio test, left-right, cluster filtering)
- 3) Определение взаимного расположения пар камер
- 4) Уточнение расположения камер и их калибровок (Bundle Adjustment)

Итак какие **параметры** мы ищем
и **что** мы вообще оптимизируем?

Есть функция **project(3D точка) -> 2D пиксель**
она позволяет сформулировать невязку



Учет Rolling Shutter

Иногда матрица камеры сохраняет изображение **построчно** на протяжении некоторого времени, а **не синхронно**.



Учет Rolling Shutter

Иногда матрица камеры сохраняет изображение **построчно** на протяжении некоторого времени, а **не синхронно**.

В профессиональных фотограмметрических камерах используется **глобальный затвор** чтобы избежать этого эффекта.



Учет Rolling Shutter

Иногда матрица камеры сохраняет изображение **построчно** на протяжении некоторого времени, а **не синхронно**.

Что будет на фотографии если объект/камера движутся?



Учет Rolling Shutter

Иногда матрица камеры сохраняет изображение **построчно** на протяжении некоторого времени, а **не синхронно**.

Если камера движется - в разных строках будет наблюдение в разный момент времени **из разного положения в пространстве**.



Учет Rolling Shutter

Иногда матрица камеры сохраняет изображение **построчно** на протяжении некоторого времени, а **не синхронно**.

Если камера движется - в разных строках будет наблюдение в разный момент времени **из разного положения в пространстве**.

При съемке большой площади дрон делает это на ходу чтобы ускорить процесс!



Учет Rolling Shutter

Линейная модель rolling shutter:

- 1) камера движется и вращается равномерно и прямолинейно (пропорционально у-координате)



Учет Rolling Shutter

Линейная модель rolling shutter:

- 1) камера движется и вращается равномерно и прямолинейно (пропорционально у-координате)
- 2) за время съемки кадра камера сдвинулась на ΔT , повернулась на ΔR



Учет Rolling Shutter

Линейная модель rolling shutter:

- 1) камера движется и вращается равномерно и прямолинейно (пропорционально у-координате)
- 2) за время съемки кадра камера сдвинулась на ΔT , повернулась на ΔR
- 3) апдейт параметров модели камеры:

$$T, R, \{k_1-k_3, c_x, c_y, f\}, \underline{\Delta T, \Delta R}$$



Учет Rolling Shutter

Линейная модель rolling shutter:

- 1) камера движется и вращается равномерно и прямолинейно (пропорционально у-координате)
- 2) за время съемки кадра камера сдвинулась на ΔT , повернулась на ΔR
- 3) апдейт параметров модели камеры:

$$T, R, \{k_1-k_3, c_x, c_y, f\}, \underline{\Delta T, \Delta R}$$

- 4) если камера ускорялась то наши полномочия все



Учет Rolling Shutter

Линейная модель rolling shutter:

- 1) камера движется и вращается равномерно и прямолинейно (пропорционально у-координате)
- 2) за время съемки кадра камера сдвинулась на ΔT , повернулась на ΔR
- 3) апдейт параметров модели камеры:

$$T, R, \{k_1-k_3, c_x, c_y, f\}, \underline{\Delta T, \Delta R}$$

- 4) если камера ускорялась то наши полномочия все
(но как и в случае если хитрая линза)



Учет Rolling Shutter

Линейная модель rolling shutter:

- за время съемки кадра камера сдвинулась на ΔT , повернулась на ΔR
- параметры камеры:

$$T, R, \{k_1, k_2, k_3, c_x, c_y, f\}, \Delta T, \Delta R$$



Учет Rolling Shutter

Линейная модель rolling shutter:

- за время съемки кадра камера сдвинулась на ΔT , повернулась на ΔR
- параметры камеры:

$$T, R, \{k_1-k_3, c_x, c_y, f\}, \Delta T, \Delta R$$

Как реализовать в функции `project(3D) -> 2D`?



Учет Rolling Shutter

Линейная модель rolling shutter:

- за время съемки кадра камера сдвинулась на ΔT , повернулась на ΔR
- параметры камеры:

$$T, R, \{k_1, k_2, k_3, c_x, c_y, f\}, \Delta T, \Delta R$$

Если камера движется строго вбок наблюдая точку на горизонте, то ее y не меняется (только x):



Как реализовать в функции `project(3D) -> 2D`?

Учет Rolling Shutter

Линейная модель rolling shutter:

- за время съемки кадра камера сдвинулась на ΔT , повернулась на ΔR
- параметры камеры:

$$T, R, \{k_1, k_2, k_3, c_x, c_y, f\}, \Delta T, \Delta R$$

Если камера движется строго вбок наблюдая точку на горизонте, то ее y не меняется (только x):



Что если есть поворот камеры?

Как реализовать в функции `project(3D) -> 2D`?

Учет Rolling Shutter

Линейная модель rolling shutter:

- за время съемки кадра камера сдвинулась на ΔT , повернулась на ΔR
- параметры камеры:

$$T, R, \{k_1, k_2, k_3, c_x, c_y, f\}, \Delta T, \Delta R$$

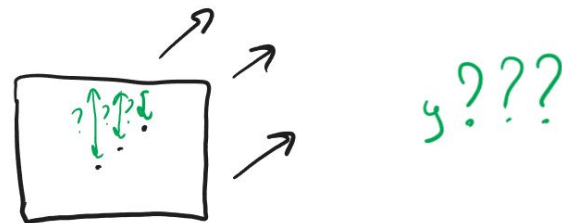
Как реализовать в функции `project(3D) -> 2D`?

Если камера движется строго вбок наблюдая точку на горизонте, то ее y не меняется (только x):



Если камера вращалась, то мы не знаем насколько камера успела повернуться пока не найдем y .

Как найти y ?



Учет Rolling Shutter

Линейная модель rolling shutter:

- за время съемки кадра камера сдвинулась на ΔT , повернулась на ΔR
- параметры камеры:

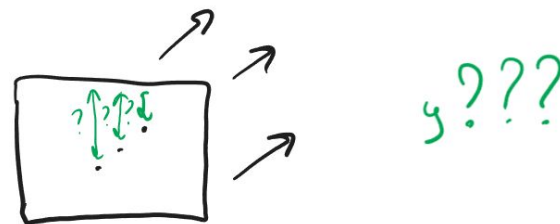
$$T, R, \{k_1, k_2, k_3, c_x, c_y, f\}, \Delta T, \Delta R$$

Как реализовать в функции `project(3D) -> 2D`?

Если камера движется строго вбок наблюдая точку на горизонте, то ее y не меняется (только x):



Если камера вращалась, то мы не знаем насколько камера успела повернуться пока не найдем y . Но мы не знаем y пока не решили насколько камера успела повернуться!



Учет Rolling Shutter

Линейная модель rolling shutter:

- за время съемки кадра камера сдвинулась на ΔT , повернулась на ΔR
- параметры камеры:

$$T, R, \{k_1, k_2, k_3, c_x, c_y, f\}, \Delta T, \Delta R$$

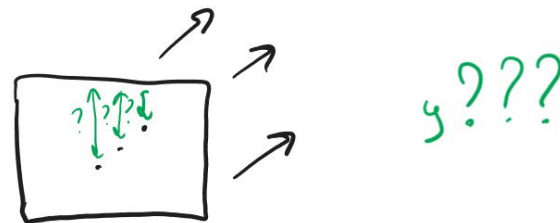
Как реализовать в функции `project(3D) -> 2D`?

Что делать?

Если камера движется строго вбок наблюдая точку на горизонте, то ее y не меняется (только x):



Если камера вращалась, то мы не знаем насколько камера успела повернуться пока не найдем y . Но мы не знаем y пока не решили насколько камера успела повернуться!



Учет Rolling Shutter

Линейная модель rolling shutter:

- за время съемки кадра камера сдвинулась на ΔT , повернулась на ΔR
- параметры камеры:

$$T, R, \{k_1, k_2, k_3, c_x, c_y, f\}, \Delta T, \Delta R$$

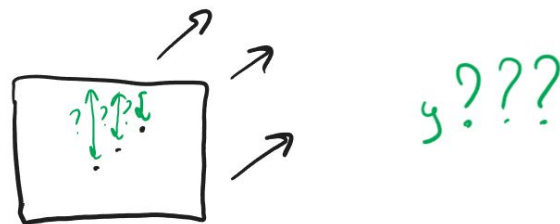
Уточняем проекцию за несколько итераций:

$$\begin{aligned} p_x &= \text{calib.project}(p) \\ \tilde{p} &= \text{shutter.project}(p, p_x \cdot y) \\ \text{n times } \uparrow & \\ p_x &= \text{calib.project}(\tilde{p}) \end{aligned}$$

Если камера движется строго вбок наблюдая точку на горизонте, то ее y не меняется (только x):

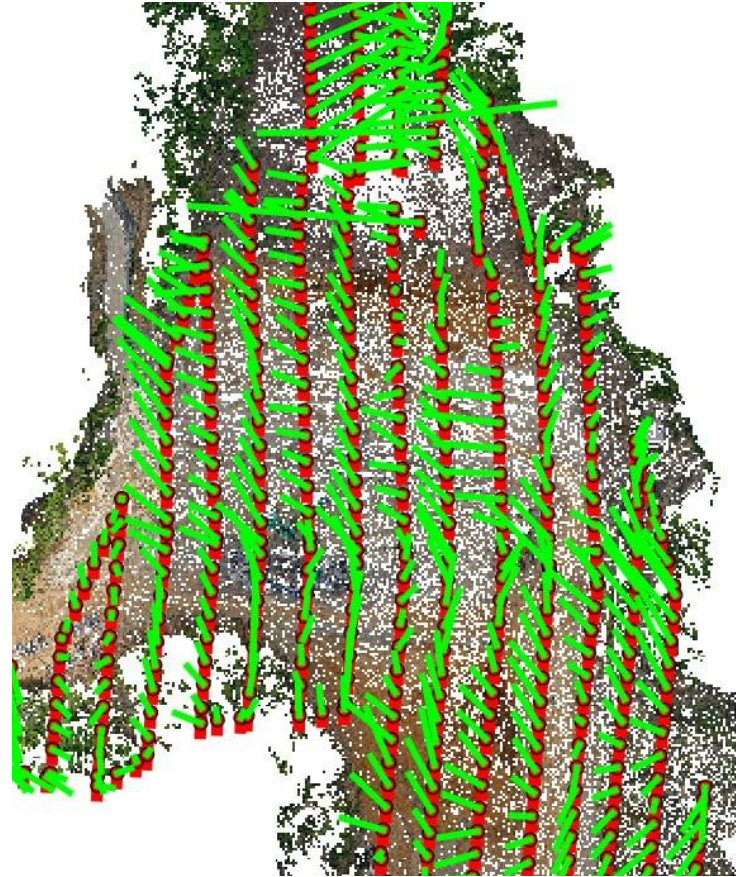


Если камера вращалась, то мы не знаем насколько камера успела повернуться пока не найдем y . Но мы не знаем y пока не решили насколько камера успела повернуться!



Учет Rolling Shutter

- 1) иногда 6 параметров слишком много и калибровочные параметры разъезжаются во время ВА, **почему так?**

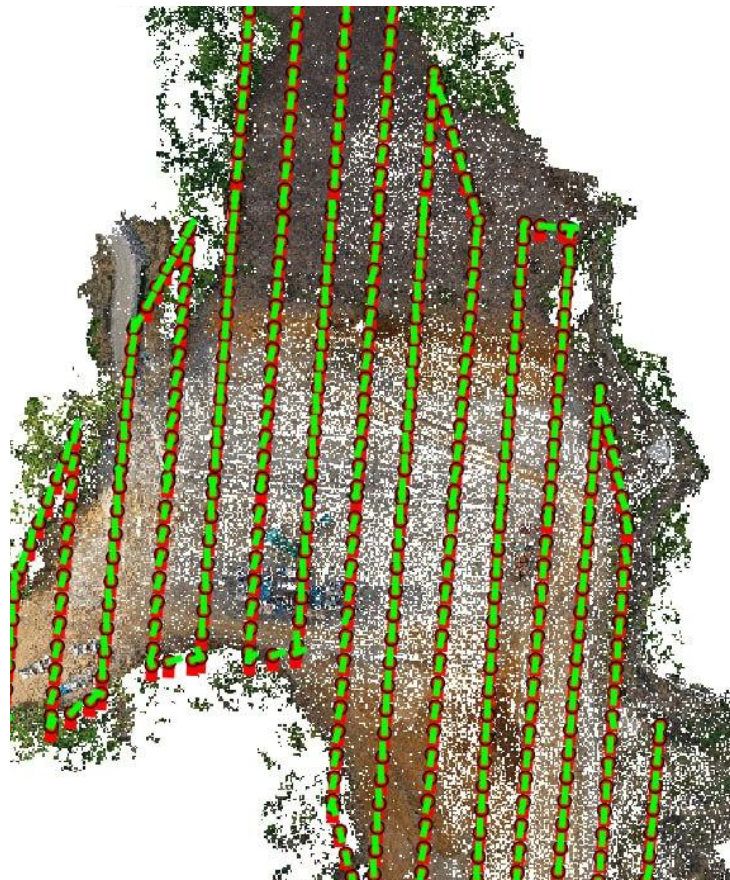


зеленые стрелочки - расчетное движение камеры ΔT_1

Учет Rolling Shutter

- 1) иногда 6 параметров слишком много и калибровочные параметры разъезжаются во время ВА
- 2) может помочь зажать модель: например, для дрона снимающего заметающей прямой можно оставить два параметра T_x , T_y вместо шести

почему картинка стала такой приятной?
почему можно утверждать что теперь
результат точнее чем был?



зеленые стрелочки - расчетное движение камеры ΔT^2

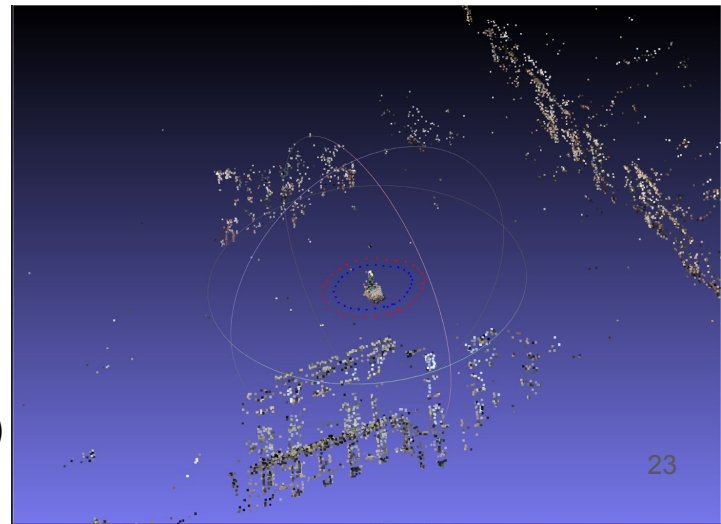
Какие уже есть этапы? Что мы хотим сделать?

- 1) Ключевые точки (SIFT)
- 2) Сопоставление ключевых точек (RANSAC, K-ratio test, left-right, cluster filtering)
- 3) Определение взаимного расположения пар камер
- 4) Уточнение расположения камер и их калибровок (Bundle Adjustment)

Итак какие **параметры** мы ищем
и **что** мы вообще оптимизируем?

Есть функция **project(3D точка) -> 2D пиксель**
она позволяет сформулировать невязку

Осталось лишь вычислить Якобиан! (**производные**)



Какие уже есть этапы? Что мы хотим сделать?

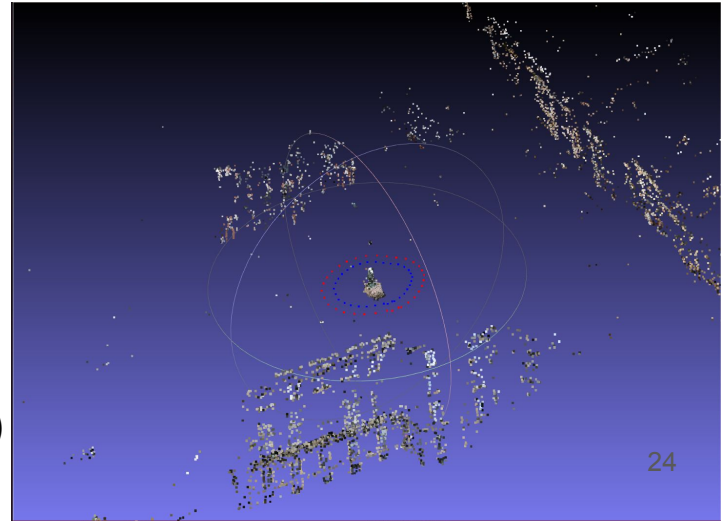
- 1) Ключевые точки (SIFT)
- 2) Сопоставление ключевых точек (RANSAC, K-ratio test, left-right, cluster filtering)
- 3) Определение взаимного расположения пар камер
- 4) Уточнение расположения камер и их калибровок (Bundle Adjustment)

Итак какие **параметры** мы ищем

и **что** мы вообще оптимизируем? **Брать просто ошибку проекции?**
Или квадрат ошибки?

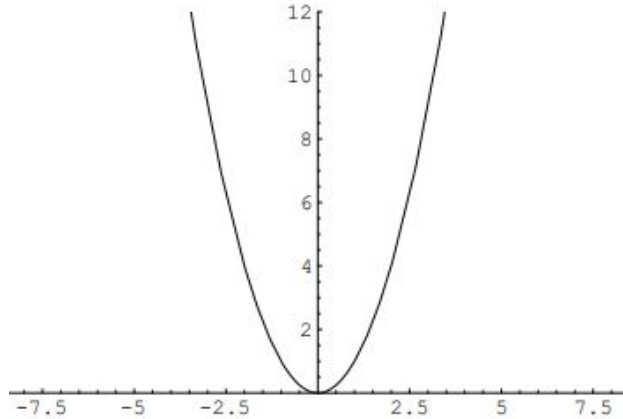
Есть функция **project(3D точка) -> 2D пиксель**
она позволяет сформулировать невязку

Осталось лишь вычислить Якобиан! (**производные**)



Loss functions (функции потерь)

$$C(\delta) = \delta^2$$

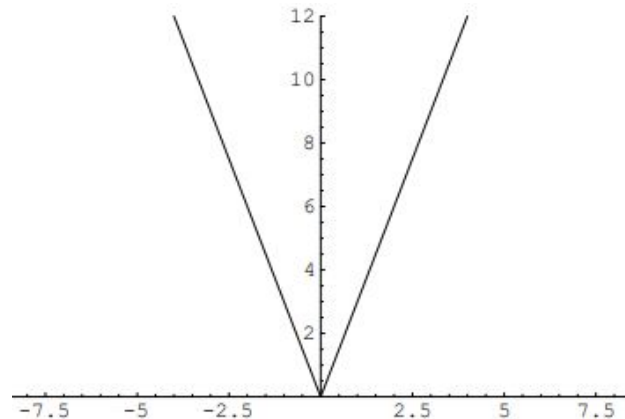


Squared-Error

L2 norm

Trivial Loss

$$C(\delta) = |\delta|$$

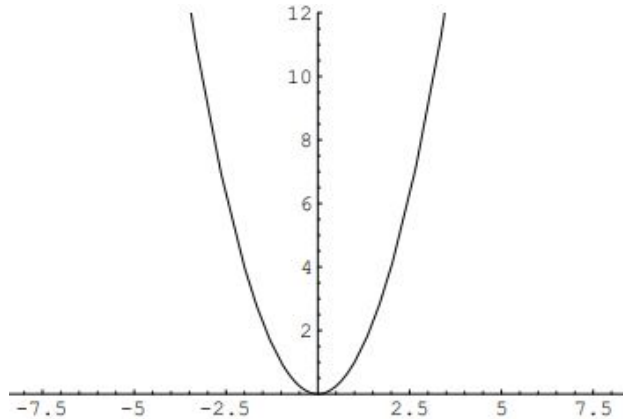


L1 norm

Total Variation Absolute Loss

Loss functions (функции потерь)

$$C(\delta) = \delta^2$$

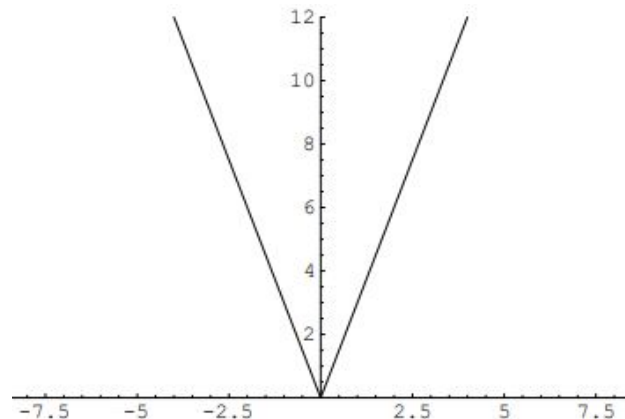


Squared-Error

L2 norm

Trivial Loss

$$C(\delta) = |\delta|$$



L1 norm

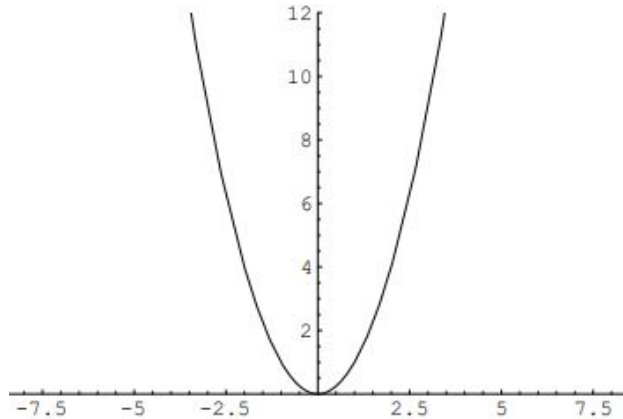
Total Variation Absolute Loss

Чем отличаются? Какие преимущества?

Что можно сказать про учет/игнорирование шумов/выбросов?

Loss functions (функции потерь)

$$C(\delta) = \delta^2$$



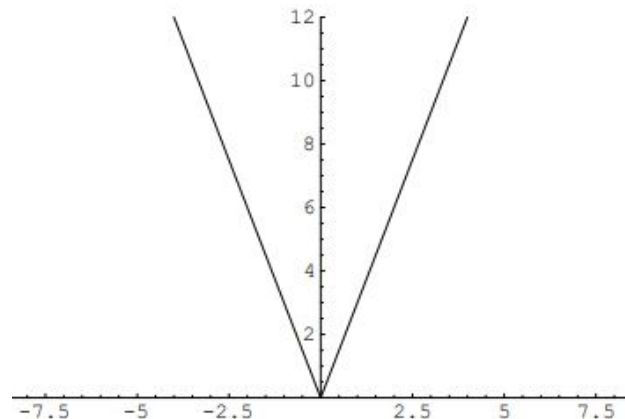
Squared-Error

L2 norm

Trivial Loss

- Наблюдения-выбросы (outliers) имеют сильное влияние.

$$C(\delta) = |\delta|$$



L1 norm

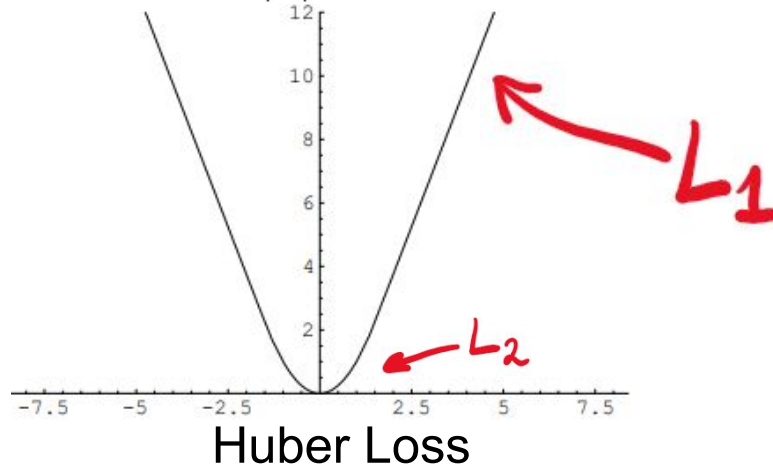
Total Variation Absolute Loss

- Устойчива к выбросам
(robust to outliers).

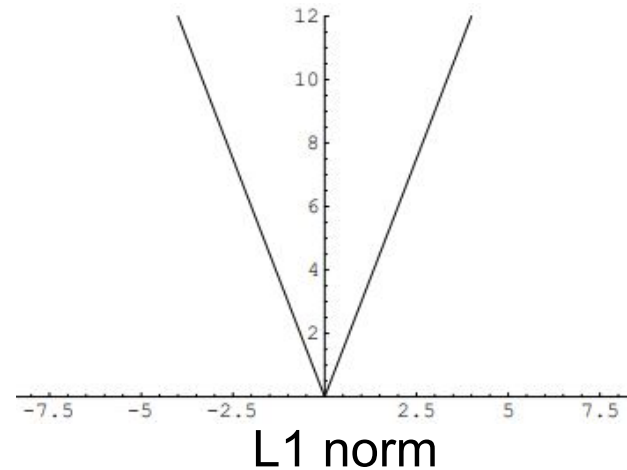
- Ищет медиану значений, игнорирует остальные значения.

Loss functions (функции потерь)

$$C(\delta) = \delta^2 \text{ for } |\delta| < b$$
$$= 2b|\delta| - b^2 \text{ otherwise}$$



$$C(\delta) = |\delta|$$

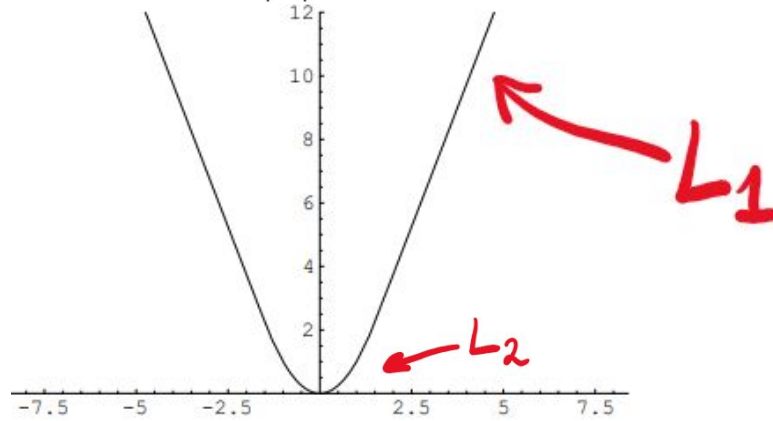


Total Variation Absolute Loss

- Устойчива к выбросам (robust to outliers).
- Ищет медиану значений, игнорирует остальные значения.

Loss functions (функции потерь)

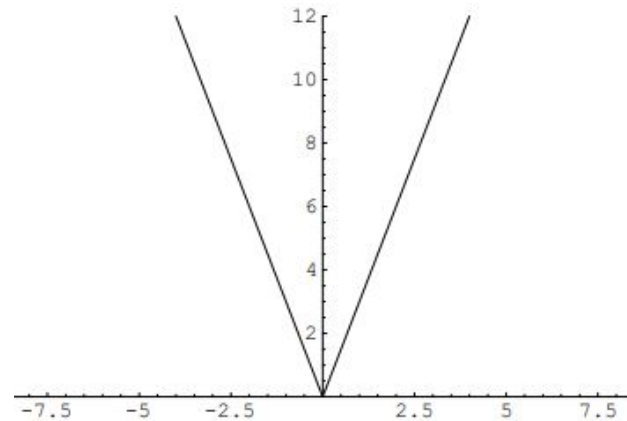
$$C(\delta) = \delta^2 \text{ for } |\delta| < b$$
$$= 2b|\delta| - b^2 \text{ otherwise}$$



Huber Loss

- Устойчива к выбросам (robust to outliers).
- Нет проблемы про медиану (т.е. хорошо приближает норм. распределение).

$$C(\delta) = |\delta|$$



L1 norm

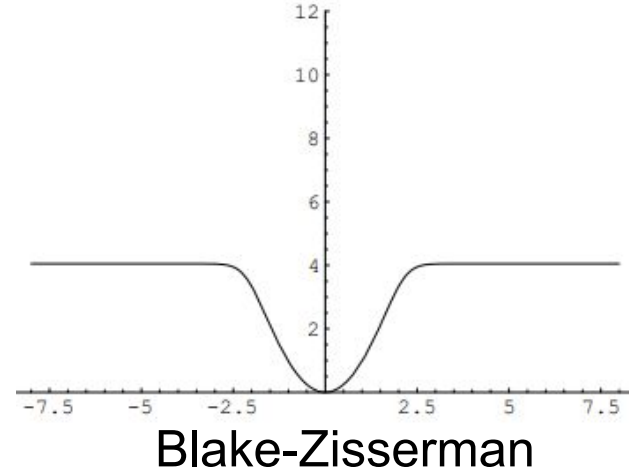
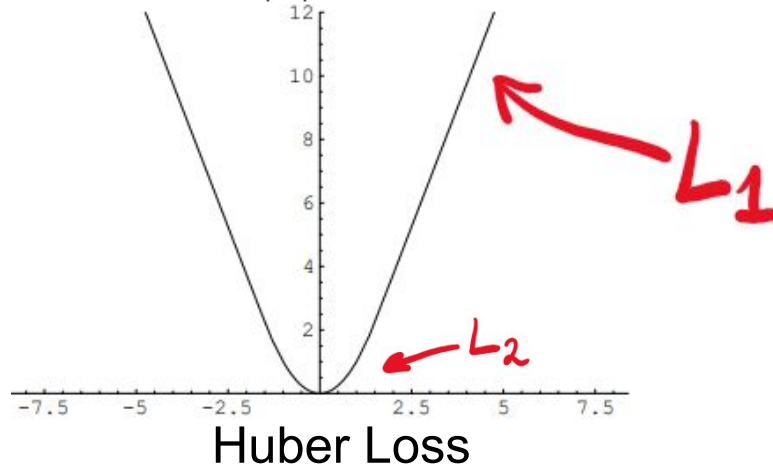
Total Variation Absolute Loss

- Устойчива к выбросам (robust to outliers).
- Ищет медиану значений, игнорирует остальные значения.

Loss functions (функции потерь)

$$\begin{aligned} C(\delta) &= \delta^2 \text{ for } |\delta| < b \\ &= 2b|\delta| - b^2 \text{ otherwise} \end{aligned}$$

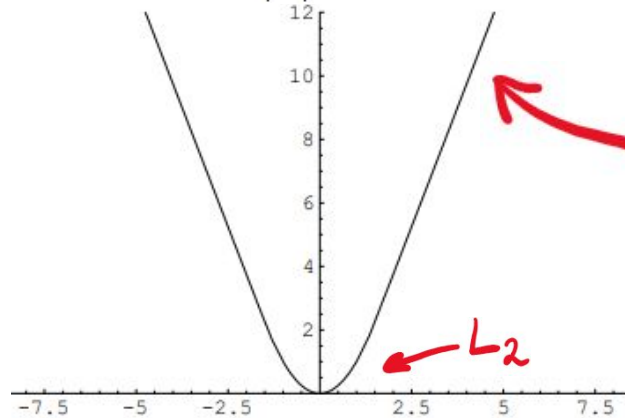
$$C(\delta) = -\log(\exp(-\delta^2) + \epsilon) \text{ and } \epsilon = \exp(-\alpha^2)$$



- Устойчива к выбросам (robust to outliers).
- Нет проблемы про медиану (т.е. хорошо приближает норм. распределение).

Loss functions (функции потерь)

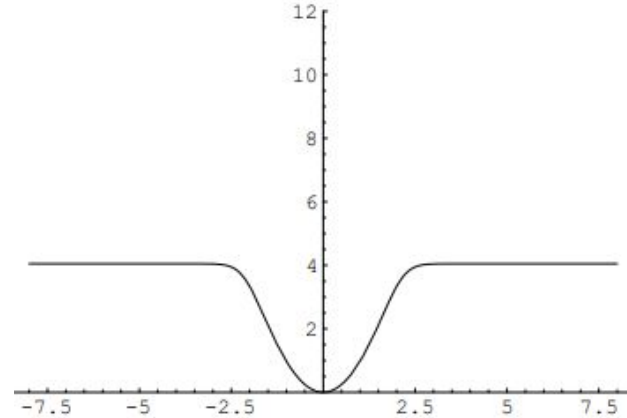
$$C(\delta) = \delta^2 \text{ for } |\delta| < b$$
$$= 2b|\delta| - b^2 \text{ otherwise}$$



Huber Loss

- Устойчива к выбросам (robust to outliers).
- Нет проблемы про медиану (т.е. хорошо приближает норм. распределение).

$$C(\delta) = -\log(\exp(-\delta^2) + \epsilon) \text{ and } \epsilon = \exp(-\alpha^2)$$



Blake-Zisserman

- Супер устойчива к выбросам.
- Если изначальное приближение - плохое, то в нем и застрянем.

Фильтрация выбросов среди 3D ключевых точек

А как без хитрых Loss functions? Как обойтись L2 trivial loss?

Фильтрация выбросов среди 3D ключевых точек

А как без хитрых Loss functions? Как обойтись L2 trivial loss?

Давайте выбрасывать точки у которых ошибка проекции **большая**.

Фильтрация выбросов среди 3D ключевых точек

А как без хитрых Loss functions? Как обойтись L2 trivial loss?

Давайте выбрасывать точки у которых ошибка проекции **большая**.

Что значит большая? Какой критерий?

Фильтрация выбросов среди 3D ключевых точек

А как без хитрых Loss functions? Как обойтись L2 trivial loss?

Давайте выбрасывать точки у которых ошибка проекции **большая**.

Критерий - **три-сигма** фильтрация (где сигма - стандартное отклонение).

Фильтрация выбросов среди 3D ключевых точек

А как без хитрых Loss functions? Как обойтись L2 trivial loss?

Давайте выбрасывать точки у которых ошибка проекции **большая**.

Критерий - **три-сигма** фильтрация (где сигма - стандартное отклонение).

А что делать если точка **стала** выбросом лишь на очередном шаге ВА?

Фильтрация выбросов среди 3D ключевых точек

А как без хитрых Loss functions? Как обойтись L2 trivial loss?

Давайте выбрасывать точки у которых ошибка проекции **большая**.

Критерий - **три-сигма** фильтрация (где сигма - стандартное отклонение).

Делаем такую фильтрацию после каждого шага оптимизации.

Фильтрация выбросов среди 3D ключевых точек

А как без хитрых Loss functions? Как обойтись L2 trivial loss?

Давайте выбрасывать точки у которых ошибка проекции **большая**.

Критерий - **три-сигма** фильтрация (где сигма - стандартное отклонение).

Делаем такую фильтрацию после каждого шага оптимизации.

Доп. фильтрация:

- точка оказалась “за спиной” камеры
- точка с проекциями в две камеры такова что угол между лучами близок к 0

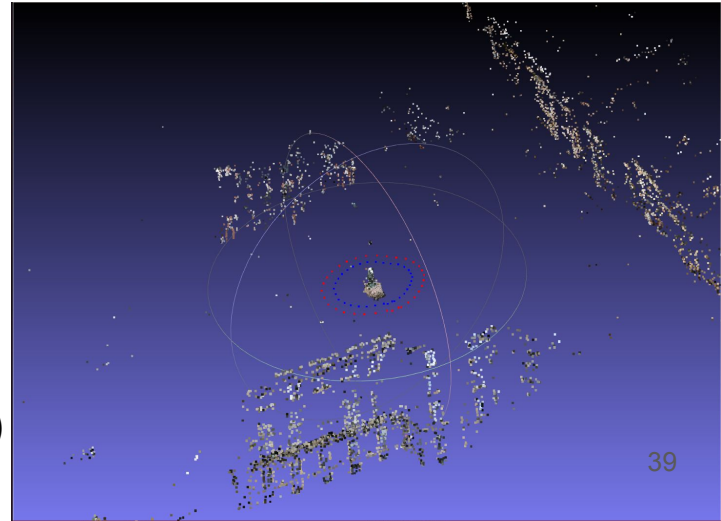
Какие уже есть этапы? Что мы хотим сделать?

- 1) Ключевые точки (SIFT)
- 2) Сопоставление ключевых точек (RANSAC, K-ratio test, left-right, cluster filtering)
- 3) Определение взаимного расположения пар камер
- 4) Уточнение расположения камер и их калибровок (Bundle Adjustment)

Итак какие **параметры** мы ищем
и **что** мы вообще оптимизируем?

Есть функция **project(3D точка) -> 2D пиксель**
она позволяет сформулировать невязку

Осталось лишь вычислить Якобиан! (**производные**)



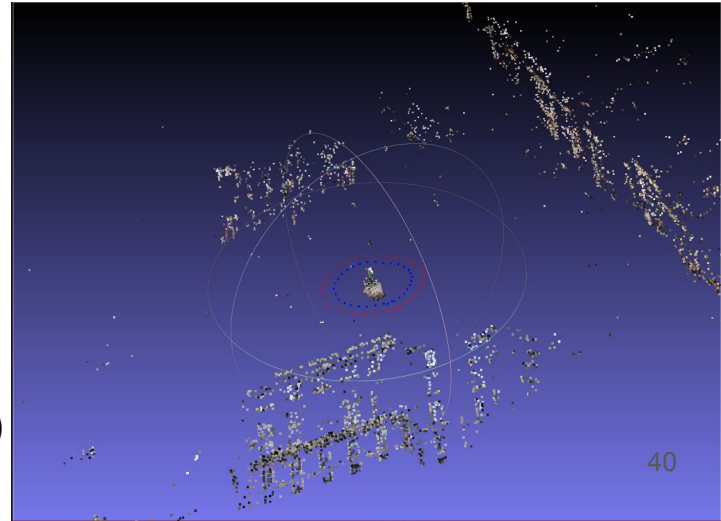
Какие уже есть этапы? Что мы хотим сделать?

- 1) Ключевые точки (SIFT)
- 2) Сопоставление ключевых точек (RANSAC, K-ratio test, left-right, cluster filtering)
- 3) Определение взаимного расположения пар камер
- 4) Уточнение расположения камер и их калибровок (Bundle Adjustment)

Итак какие **параметры** мы ищем
и **что** мы вообще оптимизируем?

Есть функция **project(3D точка) -> 2D пиксель**
она позволяет сформулировать невязку **Как их
посчитать?**

Осталось лишь вычислить Якобиан! (**производные**)



Non-linear Least Squares

Introduction

Ceres can solve bounds constrained robustified non-linear least squares problems of the form

$$\min_{\mathbf{x}} \quad \frac{1}{2} \sum_i \left(\|f_i(x_{i_1}, \dots, x_{i_k})\|^2 \right)$$

Non-linear Least Squares

Introduction

Ceres can solve bounds constrained robustified non-linear least squares problems of the form

$$\begin{aligned} \min_{\mathbf{x}} \quad & \frac{1}{2} \sum_i \rho_i \left(\|f_i(x_{i_1}, \dots, x_{i_k})\|^2 \right) \\ \text{s.t.} \quad & l_j \leq x_j \leq u_j \end{aligned}$$

Non-linear Least Squares

Introduction

Ceres can solve bounds constrained robustified non-linear least squares problems of the form

$$\min_{\mathbf{x}} \quad \frac{1}{2} \sum_i \rho_i \left(\|f_i(x_{i_1}, \dots, x_{i_k})\|^2 \right)$$

искомый параметр → \mathbf{x}

s.t. $l_j \leq x_j \leq u_j$

ограничения (constraints) на искомый параметр

Loss Function - функция потерь

Residual func. ме. невязки

(Parameter Block) блок параметров от которых зависит Cost Function $f_i(\dots)$

Residual Block #i блок невязки

Автоматическое дифференцирование

$$f(x+\varepsilon) = f(x) + f'(x)\varepsilon + o(\varepsilon^2)$$

ряд Тейлора

Автоматическое дифференцирование

$$f(x+\varepsilon) = f(x) + f'(x)\varepsilon + \cancel{O(\varepsilon^2)}$$

ряд Тейлора

Автоматическое дифференцирование

$$f(x+\varepsilon) = f(x) + f'(x)\varepsilon + \cancel{o(\varepsilon^2)}$$

$$\begin{aligned} f(x+\varepsilon)g(x+\varepsilon) &= (f + f'\varepsilon)(g + g'\varepsilon) = \\ &= fg + (f'g + fg')\varepsilon + f'g'\varepsilon^2 \end{aligned}$$

Автоматическое дифференцирование

$$f(x+\varepsilon) = f(x) + f'(x)\varepsilon + \cancel{o(\varepsilon^2)}$$

$$\begin{aligned} f(x+\varepsilon)g(x+\varepsilon) &= (f + f'\varepsilon)(g + g'\varepsilon) = \\ &= fg + \boxed{(f'g + fg')}\varepsilon + f'g'\varepsilon^2 \end{aligned}$$

что это?

Автоматическое дифференцирование

$$f(x+\varepsilon) = f(x) + f'(x)\varepsilon + \cancel{o(\varepsilon^2)}$$

$$\begin{aligned} f(x+\varepsilon)g(x+\varepsilon) &= (f + f'\varepsilon)(g + g'\varepsilon) = \\ &= fg + \boxed{(f'g + fg')}\varepsilon + f'g'\varepsilon^2 \end{aligned}$$

производная произведения функций

Автоматическое дифференцирование

$$f(x+\varepsilon) = f(x) + f'(x)\varepsilon + \cancel{o(\varepsilon^2)}$$

$$f(x+\varepsilon)g(x+\varepsilon) = (f + f'\varepsilon)(g + g'\varepsilon) =$$

перенесем
влево

$$= \boxed{fg} + (f'g + fg')\boxed{\varepsilon} + f'g'\boxed{\varepsilon^2}$$

поделим на эпсилон

$$\frac{f(x+\varepsilon)g(x+\varepsilon) - \boxed{f(x)g(x)}}{\boxed{\varepsilon}} = f'g + fg' + f'g' \cdot \boxed{\varepsilon}$$

Автоматическое дифференцирование

$$f(x+\varepsilon) = f(x) + f'(x)\varepsilon + \cancel{o(\varepsilon^2)}$$

$$\begin{aligned} f(x+\varepsilon)g(x+\varepsilon) &= (f + f'\varepsilon)(g + g'\varepsilon) = \\ &= fg + (f'g + fg')\varepsilon + f'g'\varepsilon^2 \end{aligned}$$

$$\frac{f(x+\varepsilon)g(x+\varepsilon) - f(x)g(x)}{\varepsilon} = f'g + fg' + f'g' \cdot \varepsilon$$

$$(fg)' \equiv \lim_{\varepsilon \rightarrow 0} \text{---} // \text{---} = f'g + fg'$$

Автоматическое дифференцирование

$$f(x+\varepsilon) = f(x) + f'(x)\varepsilon + \cancel{o(\varepsilon^2)}$$

Итого:

1) берем функции **в виде**

$$f(x+\varepsilon)g(x+\varepsilon) = (f + f'\varepsilon)(g + g'\varepsilon) =$$
$$= fg + (f'g + fg')\varepsilon + f'g'\varepsilon^2$$

$$\frac{f(x+\varepsilon)g(x+\varepsilon) - f(x)g(x)}{\varepsilon} = f'g + fg' + f'g' \cdot \varepsilon$$

$$(fg)' \equiv \lim_{\varepsilon \rightarrow 0} \frac{f(x+\varepsilon)g(x+\varepsilon) - f(x)g(x)}{\varepsilon} = f'g + fg'$$

Автоматическое дифференцирование

$$f(x+\varepsilon) = f(x) + f'(x)\varepsilon + \cancel{o(\varepsilon^2)}$$

Итого:

- 1) берем функции **в виде**
- 2) получаем **производную**

$$\begin{aligned} f(x+\varepsilon)g(x+\varepsilon) &= (f + f'\varepsilon)(g + g'\varepsilon) = \\ &= fg + (f'g + fg')\varepsilon + f'g'\varepsilon^2 \end{aligned}$$

$$\frac{f(x+\varepsilon)g(x+\varepsilon) - f(x)g(x)}{\varepsilon} = f'g + fg' + f'g' \cdot \varepsilon$$

$$(fg)' \equiv \lim_{\varepsilon \rightarrow 0} \frac{f(x+\varepsilon)g(x+\varepsilon) - f(x)g(x)}{\varepsilon} = f'g + fg'$$

Автоматическое дифференцирование: **Dual Numbers**

В исходниках **Ceres Solver** есть очень хорошее описание:

[github/ceres-solver/include/ceres/jet.h](https://github.com/ceres-solver/include/ceres/jet.h)

По аналогии с комплексными числами $a + b \cdot i$ дополним число бесконечно малой компонентой. И введем ϵ по аналогии с мнимой единицей i ($i^2 = -1$)

Автоматическое дифференцирование: **Dual Numbers**

В исходниках **Ceres Solver** есть очень хорошее описание:

[github/ceres-solver/include/ceres/jet.h](https://github.com/ceres-solver/include/ceres/jet.h)

По аналогии с комплексными числами $a + b \cdot i$ дополним число бесконечно малой компонентой. И введем ϵ по аналогии с мнимой единицей i ($i^2 = -1$)

$x + x \cdot \epsilon$
↑
бесконечно
малое:
 $\epsilon^2 = 0$

Автоматическое дифференцирование: **Dual Numbers**

$x + x' \cdot \epsilon$
 \uparrow
бесконечно
малое:
 $\epsilon^2 = 0$

Автоматическое дифференцирование: **Dual Numbers**

\mathbb{R} $x + x' \cdot \epsilon$

↑
бесконечно
малое:
 $\epsilon^2 = 0$

$$f(x) = x^2$$

Автоматическое дифференцирование: **Dual Numbers**

\mathbb{R} $x + x' \cdot \epsilon$
↑
бесконечно
малое:
 $\epsilon^2 = 0$

$$f(x) = x^2$$

$$f(10) = 100 \quad f'_x(10) = 20$$

Автоматическое дифференцирование: **Dual Numbers**

$x + x' \cdot \epsilon$
 $\downarrow \quad \uparrow$
 $\mathbb{R} \quad \text{бесконечно}$
 малое:
 $\epsilon^2 = 0$

$$f(x) = x^2$$

$$f(10) = 100 \quad f'_x(10) = 20$$

$$f(10 + \epsilon) = (10 + \epsilon)^2 = 100 + 20 \cdot \epsilon + \epsilon^2$$

Автоматическое дифференцирование: **Dual Numbers**

\mathbb{R} $x + x' \cdot \epsilon$
↑
дифференциально
малое:
 $\epsilon^2 = 0$

$$f(x) = x^2$$

$$f(10) = 100 \quad f'_x(10) = 20$$

$$f(10 + \epsilon) = (10 + \epsilon)^2 = 100 + 20 \cdot \epsilon + \cancel{\epsilon^2}$$

0

Автоматическое дифференцирование: **Dual Numbers**

\mathbb{R} $x + x' \cdot \epsilon$
↑
бесконечно
малое:
 $\epsilon^2 = 0$

$$f(x) = x^2$$

$$f(10) = 100 \quad f'_x(10) = 20$$

$$f(10 + \epsilon) = (10 + \epsilon)^2 = 100 + 20 \cdot \epsilon + \cancel{\epsilon^2}$$

"0"

dual number

Автоматическое дифференцирование: **Dual Numbers**

\mathbb{R} $x + x' \cdot \epsilon$
↑
бесконечно
малое:
 $\epsilon^2 = 0$

$$f(x) = x^2$$
$$f(10) = 100 \quad f'_x(10) = 20$$
$$f(10 + \epsilon) = (10 + \epsilon)^2 = \boxed{100 + 20 \cdot \epsilon} + \cancel{\epsilon^2}$$

$\begin{array}{ccc} \parallel & \parallel & \parallel \\ f(10) & f'_x(10) & 0 \end{array}$

dual number

Автоматическое дифференцирование: Dual Numbers

\mathbb{R} $x + x' \cdot \epsilon$
↑
дифференциально
малое:
 $\epsilon^2 = 0$

$$f(x) = x^2$$

$$f(10) = 100 \quad f'_x(10) = 20$$

$$f(10 + \epsilon) = (10 + \epsilon)^2 = \boxed{100 + 20 \cdot \epsilon} + \cancel{\epsilon^2}$$

\parallel \parallel \parallel
 $f(10)$ $f'_x(10)$ 0

dual
number

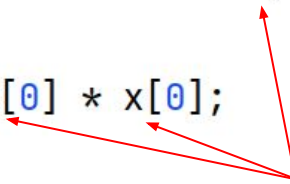
$$f(x_0 + x' \cdot \epsilon) = f(x_0) + f'_x(x_0) \cdot x' \cdot \epsilon$$

Автоматическое дифференцирование: **Dual Numbers**

```
1 struct CostFunctor {  
2     template <typename T>  
3     bool operator()(const T* const x, T* residual) const {  
4         //  $f(x) = x^2$   
5         residual[0] = x[0] * x[0];  
6         return true;  
7     }  
8 };
```

Автоматическое дифференцирование: Dual Numbers

```
1 struct CostFunctor {  
2     template <typename T>  
3     bool operator()(const T* const x, T* residual) const {  
4         //  $f(x) = x^2$   
5         residual[0] = x[0] * x[0];  
6         return true;  
7     }  
8 };
```



искомое
(оптимизируемые параметры)

Автоматическое дифференцирование: Dual Numbers

```
1 struct CostFunctor {  
2     template <typename T>  
3     bool operator()(const T* const x, T* residual) const {  
4         //  $f(x) = x^2$   
5         residual[0] = x[0] * x[0];  
6         return true;  
7     }  
8 };
```

T - подставляемый тип
T=double или **Jet**

искмое
(оптимизируемые параметры)

Автоматическое дифференцирование: Dual Numbers

```
1 struct CostFunctor {  
2     template <typename T>  
3     bool operator()(const T* const x, T* residual) const {  
4         //  $f(x) = x^2$   
5         residual[0] = x[0] * x[0];  
6         return true;  
7     }  
8 };
```

T - подставляемый тип
T=double или **Jet**

искмое
(оптимизируемые параметры)

```
211 struct Jet {  
285     // The scalar part.  
286     T a;  
288     // The infinitesimal part.  
289     Eigen::Matrix<T, N, 1> v;  
294 };
```

Автоматическое дифференцирование: Dual Numbers

```
1 struct CostFunctor {  
2     template <typename T>  
3     bool operator()(const T* const x, T* residual) const {  
4         //  $f(x) = x^2$   
5         residual[0] = x[0] * x[0];  
6         return true;  
7     }  
8 };
```

T - подставляемый тип

T=double или Jet

искомое
(оптимизируемые параметры)

```
211 struct Jet {  
285     // The scalar part.  
286     T a;  
288     // The infinitesimal part.  
289     Eigen::Matrix<T, N, 1> v;  
294 };
```

Dual Number

Автоматическое дифференцирование: Dual Numbers

```
1 struct CostFunctor {  
2     template <typename T>  
3     bool operator()(const T* const x, T* residual) const {  
4         //  $f(x) = x^2$   
5         residual[0] = x[0] * x[0];  
6         return true;  
7     }  
8 };
```

T - подставляемый тип
T=double или **Jet**

искомое
(оптимизируемые параметры)

```
211 struct Jet {  
285     // The scalar part.  
286     T a;  
288     // The infinitesimal part.  
289     Eigen::Matrix<T, N, 1> v;  
294 };
```

Dual Number

**Почему не
просто число?**

Автоматическое дифференцирование: Dual Numbers

```
1 struct CostFunctor {
2     template <typename T>
3     bool operator()(const T* const x, T* residual) const {
4         // f(x) = x^2
5         residual[0] = x[0] * x[0];
6         return true;
7     }
8 };
```

T - подставляемый тип
T=double или **Jet**

искомое
(оптимизируемые параметры)

```
211 struct Jet {
285     // The scalar part.
286     T a;
288     // The infinitesimal part.
289     Eigen::Matrix<T, N, 1> v;
294 };
```

Dual Number

хранит производные
по всем параметрам
функции

Автоматическое дифференцирование: **Dual Numbers**

$$f(x_0 + \epsilon) = f(x_0) + f'_x(x_0) \cdot \epsilon$$

Проверим правило взятия производной сложной функции (**Chain Rule**).

Автоматическое дифференцирование: **Dual Numbers**

$$f(x_0 + \epsilon) = f(x_0) + f'_x(x_0) \cdot \epsilon$$

Проверим правило взятия производной сложной функции (**Chain Rule**).

$$\left[g(f(x_0)) \right]_x' = g'_x(f(x_0)) \cdot f'_x(x_0)$$

Автоматическое дифференцирование: **Dual Numbers**

$$f(x_0 + \epsilon) = f(x_0) + f'_x(x_0) \cdot \epsilon$$

Проверим правило взятия производной сложной функции (**Chain Rule**).

$$\left[g(f(x_0)) \right]'_x = g'_x(f(x_0)) \cdot f'_x(x_0)$$

$$g(f(x_0 + \epsilon)) = g(f(x_0) + f'_x(x_0) \cdot \epsilon)$$

Автоматическое дифференцирование: **Dual Numbers**

$$f(x_0 + \epsilon) = f(x_0) + f'_x(x_0) \cdot \epsilon$$

Проверим правило взятия производной сложной функции (**Chain Rule**).

$$\left[g(f(x_0)) \right]'_x = g'_x(f(x_0)) \cdot f'_x(x_0)$$

$$\begin{aligned} g(f(x_0 + \epsilon)) &= g(f(x_0) + f'_x(x_0) \cdot \epsilon) \\ &= g(f(x_0)) + g'_x(f(x_0)) \cdot f'_x(x_0) \cdot \epsilon \end{aligned}$$

Автоматическое дифференцирование: **Dual Numbers**

$$f(x_0 + \epsilon) = f(x_0) + f'_x(x_0) \cdot \epsilon$$

Проверим правило взятия производной сложной функции (**Chain Rule**).

$$\left[g(f(x_0)) \right]'_x = g'_x(f(x_0)) \cdot f'_x(x_0)$$

$$\begin{aligned} g(f(x_0 + \epsilon)) &= g(f(x_0) + f'_x(x_0) \cdot \epsilon) \\ &= g(f(x_0)) + g'_x(f(x_0)) \cdot f'_x(x_0) \cdot \epsilon \end{aligned}$$

||

$$\left[g(f(x_0)) \right]'_x = g'_x(f(x_0)) \cdot f'_x(x_0)$$

Автоматическое дифференцирование: **Dual Numbers**

Осталось **распространить** все базовые операции с вещественных чисел на **dual numbers**:

- Сложение, умножение, вычитание, деление
- Возведение в степень
- Тригонометрические функции
- ...

Автоматическое дифференцирование: **Dual Numbers**

Осталось **распространить** все базовые операции с вещественных чисел на **dual numbers**:

- Сложение, **умножение**, вычитание, деление
- Возведение в степень
- Тригонометрические функции
- ...

Автоматическое дифференцирование: Dual Numbers

Умножение: [github/ceres-solver/include/ceres/jet.h](https://github.com/ceres-solver/include/ceres/jet.h) `operator*`

```
347 // Binary *
348 template <typename T, int N>
349 inline Jet<T, N> operator*(const Jet<T, N>& f, const Jet<T, N>& g) {
350     return Jet<T, N>(f.a * g.a, f.a * g.v + f.v * g.a);
351 }
```

```
211 struct Jet {
285     // The scalar part.
286     T a;
288     // The infinitesimal part.
289     Eigen::Matrix<T, N, 1> v;
294 };
```

Автоматическое дифференцирование: Dual Numbers

Умножение: [github/ceres-solver/include/ceres/jet.h](https://github.com/ceres-solver/include/ceres/jet.h) operator*

```
347 // Binary *
348 template <typename T, int N>
349 inline Jet<T, N> operator*(const Jet<T, N>& f, const Jet<T, N>& g) {
350     return Jet<T, N>(f.a * g.a, f.a * g.v + f.v * g.a);
351 }
```

211 struct Jet {
285 // The scalar part.
286 T a;
288 // The infinitesimal part.
289 Eigen::Matrix<T, N, 1> v;
294 };

real

ϵ

Автоматическое дифференцирование: Dual Numbers

Умножение: [github/ceres-solver/include/ceres/jet.h](https://github.com/ceres-solver/include/ceres/jet.h) operator*

```
347 // Binary *
348 template <typename T, int N>
349 inline Jet<T, N> operator*(const Jet<T, N>& f, const Jet<T, N>& g) {
350     return Jet<T, N>(f.a * g.a, f.a * g.v + f.v * g.a);
351 }
```

```
211 struct Jet {
285     // The scalar part.
286     T a;
288     // The infinitesimal part.
289     Eigen::Matrix<T, N, 1> v;
294 };
```

$$(a + b\varepsilon) \cdot (c + d\varepsilon) = a \cdot c + (b \cdot c + a \cdot d) \cdot \varepsilon + \cancel{b \cdot d \cdot \varepsilon^2}$$

$\varepsilon^2 = 0$

Автоматическое дифференцирование: Dual Numbers

Умножение: [github/ceres-solver/include/ceres/jet.h](https://github.com/ceres-solver/include/ceres/jet.h) `operator*`

```
347 // Binary *
348 template <typename T, int N>
349 inline Jet<T, N> operator*(const Jet<T, N>& f, const Jet<T, N>& g) {
350     return Jet<T, N>(f.a * g.a, f.a * g.v + f.v * g.a);
351 }
```

```
211 struct Jet {
285     // The scalar part.
286     T a;
288     // The infinitesimal part.
289     Eigen::Matrix<T, N, 1> v;
294 }
```

$$\overbrace{(a + b\varepsilon)}^f \cdot \overbrace{(c + d\varepsilon)}^g = a \cdot c + (b \cdot c + a \cdot d) \cdot \varepsilon + \cancel{b \cdot d \cdot \varepsilon^2}$$

$\varepsilon^2 = 0$

Автоматическое дифференцирование: Dual Numbers

Умножение: [github/ceres-solver/include/ceres/jet.h](https://github.com/ceres-solver/include/ceres/jet.h) `operator*`

```
347 // Binary *
348 template <typename T, int N>
349 inline Jet<T, N> operator*(const Jet<T, N>& f, const Jet<T, N>& g) {
350     return Jet<T, N>(f.a * g.a, f.a * g.v + f.v * g.a);
351 }
```

```
211 struct Jet {
285     // The scalar part.
286     T a;
288     // The infinitesimal part.
289     Eigen::Matrix<T, N, 1> v;
294 }
```

$$\overbrace{(a + b\varepsilon)}^f \cdot \overbrace{(c + d\varepsilon)}^g = a \cdot c + (b \cdot c + a \cdot d) \cdot \varepsilon + \cancel{b \cdot d \cdot \varepsilon^2}$$

$\varepsilon^2 = 0$

Автоматическое дифференцирование: Dual Numbers

Умножение: [github/ceres-solver/include/ceres/jet.h](https://github.com/ceres-solver/include/ceres/jet.h) `operator*`

```
347 // Binary *
348 template <typename T, int N>
349 inline Jet<T, N> operator*(const Jet<T, N>& f, const Jet<T, N>& g) {
350     return Jet<T, N>(f.a * g.a, f.a * g.v + f.v * g.a);
351 }
```

```
211 struct Jet {
285     // The scalar part.
286     T a;
288     // The infinitesimal part.
289     Eigen::Matrix<T, N, 1> v;
294 };
```

$$\overbrace{(a + b\varepsilon)}^f \cdot \overbrace{(c + d\varepsilon)}^g = a \cdot c + (b \cdot c + a \cdot d) \cdot \varepsilon + \cancel{b \cdot d \cdot \varepsilon^2}$$

$\varepsilon^2 = 0$

Автоматическое дифференцирование: Dual Numbers

Умножение: [github/ceres-solver/include/ceres/jet.h](https://github.com/ceres-solver/include/ceres/jet.h) operator*

```

347 // Binary *
348 template <typename T, int N>
349 inline Jet<T, N> operator*(const Jet<T, N>& f, const Jet<T, N>& g) {
350     return Jet<T, N>(f.a * g.a, f.a * g.v + f.v * g.a);
351 }

```

```
211 struct Jet {
285     // The scalar part.
286     T a;
288     // The infinitesimal part.
289     Eigen::Matrix<T, N, 1> v;
294 };
```

$$\overbrace{(a + b\varepsilon)}^f \cdot \overbrace{(c + d\varepsilon)}^g = \overbrace{a \cdot c}^{||} + (b \cdot c + a \cdot d) \cdot \varepsilon + \cancel{b \cdot d \cdot \varepsilon^2}$$

Автоматическое дифференцирование: Dual Numbers

Умножение: [github/ceres-solver/include/ceres/jet.h](https://github.com/ceres-solver/include/ceres/jet.h) operator*

```

347 // Binary *
348 template <typename T, int N>
349 inline Jet<T, N> operator*(const Jet<T, N>& f, const Jet<T, N>& g) {
350     return Jet<T, N>(f.a * g.a, f.a * g.v + f.v * g.a);
351 }

```

```
211 struct Jet {
285     // The scalar part.
286     T a;
288     // The infinitesimal part.
289     Eigen::Matrix<T, N, 1> v;
294 };
```

$$\overbrace{(a + b\varepsilon)}^f \cdot \overbrace{(c + d\varepsilon)}^g = \overbrace{a \cdot c}^{||} + (b \cdot c + a \cdot d) \cdot \varepsilon + \cancel{b \cdot d \cdot \varepsilon^2}$$

Автоматическое дифференцирование: Dual Numbers

Умножение: [github/ceres-solver/include/ceres/jet.h](https://github.com/ceres-solver/include/ceres/jet.h) `operator*`

```
347 // Binary *
348 template <typename T, int N>
349 inline Jet<T, N> operator*(const Jet<T, N>& f, const Jet<T, N>& g) {
350     return Jet<T, N>(f.a * g.a, f.a * g.v + f.v * g.a);
351 }
```

```
211 struct Jet {
285     // The scalar part.
286     T a;
288     // The infinitesimal part.
289     Eigen::Matrix<T, N, 1> v;
294 };
```

$$\overbrace{(a + b\varepsilon)}^f \cdot \overbrace{(c + d\varepsilon)}^g = \overbrace{a \cdot c} + \overbrace{(b \cdot c + a \cdot d)} \cdot \varepsilon + \cancel{b \cdot d \cdot \varepsilon^2} = 0$$

Автоматическое дифференцирование: Dual Numbers

Умножение: [github/ceres-solver/include/ceres/jet.h](https://github.com/ceres-solver/include/ceres/jet.h) operator*

```
347 // Binary *
348 template <typename T, int N>
349 inline Jet<T, N> operator*(const Jet<T, N>& f, const Jet<T, N>& g) {
350     return Jet<T, N>(f.a * g.a, f.a * g.v + f.v * g.a);
351 }
```

```
211 struct Jet {
285     // The scalar part.
286     T a;
288     // The infinitesimal part.
289     Eigen::Matrix<T, N, 1> v;
294 };
```

$$\overbrace{(a+b\varepsilon)}^f \cdot \overbrace{(c+d\varepsilon)}^g = \overbrace{a \cdot c}^{=} + \overbrace{(b \cdot c + a \cdot d)}^{=} \cdot \varepsilon + \cancel{b \cdot d \cdot \varepsilon^2} = 0$$

Автоматическое дифференцирование: Dual Numbers

Умножение: [github/ceres-solver/include/ceres/jet.h](https://github.com/ceres-solver/include/ceres/jet.h) operator*

```
347 // Binary *
348 template <typename T, int N>
349 inline Jet<T, N> operator*(const Jet<T, N>& f, const Jet<T, N>& g) {
350     return Jet<T, N>(f.a * g.a, f.a * g.v + f.v * g.a);
351 }
```

```
211 struct Jet {
285     // The scalar part.
286     T a;
288     // The infinitesimal part.
289     Eigen::Matrix<T, N, 1> v;
294 };
```

$$(a + b\varepsilon) \cdot (c + d\varepsilon) = a \cdot c + (b \cdot c + a \cdot d) \cdot \varepsilon + \cancel{b \cdot d \cdot \varepsilon^2} = 0$$

Автоматическое дифференцирование: Dual Numbers

Умножение: [github/ceres-solver/include/ceres/jet.h](https://github.com/ceres-solver/include/ceres/jet.h) operator*

```
347 // Binary *
348 template <typename T, int N>
349 inline Jet<T, N> operator*(const Jet<T, N>& f, const Jet<T, N>& g) {
350     return Jet<T, N>(f.a * g.a, f.a * g.v + f.v * g.a);
351 }
```

```
211 struct Jet {
285     // The scalar part.
286     T a;
288     // The infinitesimal part.
289     Eigen::Matrix<T, N, 1> v;
294 };
```

$$\overbrace{(a + b\varepsilon)}^f \cdot \overbrace{(c + d\varepsilon)}^g = \overbrace{a \cdot c} + \overbrace{(b \cdot c + a \cdot d)} \cdot \varepsilon + \cancel{b \cdot d \cdot \varepsilon}$$

$\varepsilon^2 = 0$

Автоматическое дифференцирование: Dual Numbers

Синус: [github/ceres-solver/include/ceres/jet.h](https://github.com/ceres-solver/include/ceres/jet.h) `sin()`

```
// sin(a + h) ~= sin(a) + cos(a) h
template <typename T, int N>
inline Jet<T, N> sin(const Jet<T, N>& f) {
    return Jet<T, N>(sin(f.a), cos(f.a) * f.v);
}
```

```
211 struct Jet {
285     // The scalar part.
286     T a;
288     // The infinitesimal part.
289     Eigen::Matrix<T, N, 1> v;
};
```

Примеры задач

1) Минимизировать $\frac{1}{2}(10 - x)^2$

Какие блоки параметров? Какая функция невязки (residual)?

Примеры задач

- 1) Минимизировать $\frac{1}{2}(10 - x)^2$
- 2) Есть фиксированная прямая и фиксированный параболоид. Найти точку пересечения.

Какие блоки параметров? Какая функция невязки (residual)?

Примеры задач

- 1) Минимизировать $\frac{1}{2}(10 - x)^2$
- 2) Есть фиксированная прямая и фиксированный параболоид. Найти точку пересечения.
- 3) Есть сколько-то шумных замеров (с выбросами), нужно зафитить прямой.

Какие блоки параметров? Какая функция невязки (residual)? Какая функция потерь (Loss function)?

Примеры задач

- 1) Минимизировать $\frac{1}{2}(10 - x)^2$
- 2) Есть фиксированная прямая и фиксированный параболоид. Найти точку пересечения.
- 3) Есть сколько-то шумных замеров (с выбросами), нужно зафитить прямой.
- 4) Bundle Adjustment.

Какие блоки параметров? Какая функция невязки (residual)? Какая функция потерь (Loss function)?

Примеры задач

- 1) Минимизировать $\frac{1}{2}(10 - x)^2$
- 2) Есть фиксированная прямая и фиксированный параболоид. Найти точку пересечения.
- 3) Есть сколько-то шумных замеров (с выбросами), нужно зафитить прямой.
- 4) Bundle Adjustment.

Какие блоки параметров? Какая функция невязки (residual)? Какая функция потерь (Loss function)?

Как фильтровать шумные ключевые точки из разреженного облака точек для более надежного последующего добавления очередной камеры?

Hierarchical SFM

- 1) Что делать если в датасете много кадров (например, 10K)
 - а) даже самый мощный компьютер будет обрабатывать долго

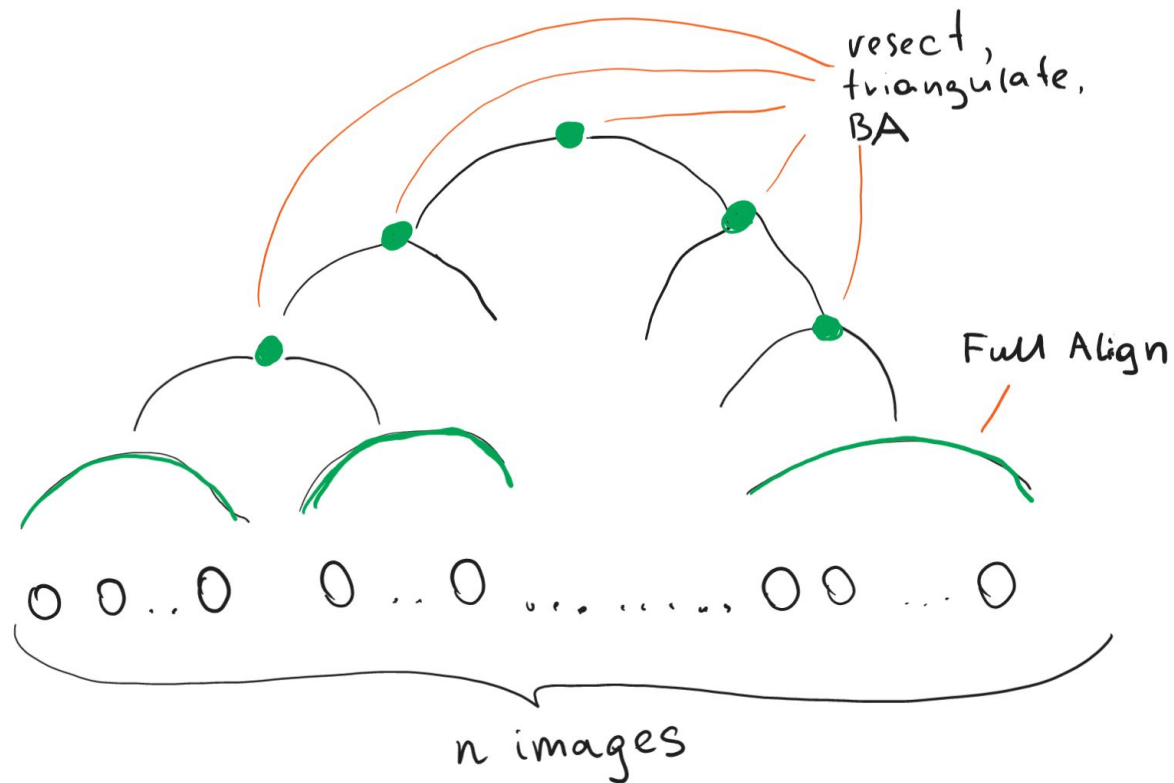
Hierarchical SFM

- 1) Что делать если в датасете много кадров (например, 10K)
 - a) даже самый мощный компьютер будет обрабатывать долго
 - b) а если кадров еще в 10 раз больше?

Hierarchical SFM

- 1) Что делать если в датасете много кадров (например, 10K)
 - a) даже самый мощный компьютер будет обрабатывать долго
 - b) а если кадров еще в 10 раз больше?
- 2) **Hierarchical SFM**: используем сразу много компьютеров (узлов) соединенных по локальной сети
 - a) разобьем датасет на небольшие группы (например по 100 кадров)
 - b) подзадача для каждого узла: выровнять группу из 100 кадров, либо взять две выровненные группы, объединить облака точек и выполнить BA

Hierarchical SFM



Геопривязка

- 1) Что если мы хотим по построенной геометрии делать измерения?
Например, построение планов местности, кадастровый учет, измерения объемов руды на карьерах

Геопривязка

- 1) Что если мы хотим по построенной геометрии делать измерения?
Например, построение планов местности, кадастровый учет, измерения объемов руды на карьерах
- 2) Можно с помощью точного GPS датчика измерить несколько характерных хорошо заметных точек на поверхности земли (Контрольные точки, Ground Control Points), и добавить их в ВА как дополнительные измерения с большим весом

Геопривязка

- 1) Что если мы хотим по построенной геометрии делать измерения?
Например, построение планов местности, кадастровый учет, измерения объемов руды на карьерах
- 2) Можно с помощью точного GPS датчика измерить несколько характерных хорошо заметных точек на поверхности земли (Контрольные точки, Ground Control Points), и добавить их в ВА как дополнительные измерения с большим весом
- 3) Как оценить качество выравнивания?

Вопросы?

