

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ  
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
имени М. В. ЛОМОНОСОВА»

МЕХАНИКО-МАТЕМАТИЧЕСКИЙ ФАКУЛЬТЕТ  
КАФЕДРА ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА  
(ДИПЛОМНАЯ РАБОТА) СПЕЦИАЛИСТА

**ПРИМЕНЕНИЕ И ИССЛЕДОВАНИЕ МЕТОДОВ ОПТИМИЗАЦИИ В  
ОДНОЙ ЗАДАЧЕ ВЫЧИСЛИТЕЛЬНОЙ ГЕОМЕТРИИ**

Выполнил студент 604 группы  
Ковальков Максим Николаевич

---

подпись студента

Научный руководитель:  
к. ф.-м. н., доцент  
Валединский Владимир Дмитриевич

---

подпись научного руководителя

Москва  
2021

# **Содержание**

<b>1. Введение</b>	<b>4</b>
1.1. Обзор различных методов реконструкции трехмерного тела . . . . .	4
1.2. Реконструкция кристаллов драгоценных камней . . . . .	10
1.3. Комментарии к структуре работы . . . . .	12
<b>2. Алгоритм построения целевых ребер</b>	<b>13</b>
2.1. Постановка задачи построения целевых ребер . . . . .	13
2.2. Формула проекции на вектор . . . . .	13
2.3. Прямая, наименее уклоняющаяся от набора точек . . . . .	14
2.4. Построение ребра по отрезкам проекций . . . . .	18
<b>3. Построение многогранника по набору полупространств</b>	<b>19</b>
<b>4. Задача условной оптимизации</b>	<b>20</b>
4.1. Математическая формализация задачи. Построение минимизируемого функционала . . . . .	20
4.2. Доказательство выпуклости функционала. . . . .	21
4.3. Условия, налагаемые на функционал. . . . .	22
4.4. Восстановление многогранника . . . . .	24
<b>5. Результаты работы программной реализации алгоритма для задачи условной оптимизации</b>	<b>24</b>
5.1. Тестовая модель куба . . . . .	24
5.2. Первый тестовый многогранник. . . . .	27
5.3. Второй тестовый многогранник. . . . .	29
5.4. Ускорение работы на больших моделях - локальная оптимизация. . . . .	29
<b>6. Сведение задачи к задаче безусловной оптимизации</b>	<b>30</b>
6.1. Математическая формализация задачи. Построение минимизируемого функционала. . . . .	30
6.2. Второй вариант минимизируемого функционала . . . . .	32
6.3. Выбор градиентного метода . . . . .	32
6.4. Выбор стартовой точки оптимизации . . . . .	33
6.5. Восстановление многогранника . . . . .	35
<b>7. Результаты работы программной реализации алгоритма для задачи безусловной оптимизации</b>	<b>35</b>
7.1. Тестовая модель куба . . . . .	35
7.2. Первый тестовый многогранник . . . . .	40
7.3. Второй тестовый многогранник . . . . .	41
<b>8. Заключение</b>	<b>42</b>

<b>Appendices</b>	<b>45</b>
<b>A. Приложение 1: Установка IPOPT</b>	<b>45</b>
<b>B. Приложение 2: C++ Интерфейс IPOPT</b>	<b>47</b>

# 1. Введение

## 1.1. Обзор различных методов реконструкции трехмерного тела

Построение объемной модели реального физического объекта является довольно распространенной задачей математического моделирования. Существует достаточно много способов решения такого класса задач.

Приведем некоторые из них:

- 1) Построение модели по большому числу разноракурсных фотографий, имеющих сильное перекрытие.

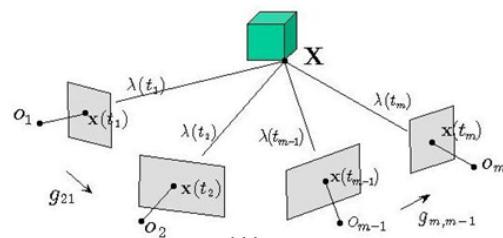


Рис. 1. Общая модель реконструкции объемного тела по набору фотографий

Данный метод нашел широкое распространение в ряде индустрий, так как не требует специального оборудования: все, что необходимо для его реализации найдется практически в любом фото-ателье. Он применяется в отраслях машиностроения и строительства, активно используется при сохранении и реставрации объектов культурного и исторического наследия. Помимо этого значительная доля его практического использования приходится на сферу развлечений: с помощью данного алгоритма дизайнер может создать правдоподобную детализированную модель за значительно более короткий срок.

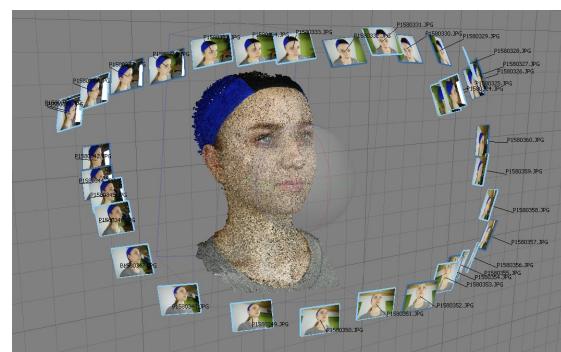


Рис. 2. Применение алгоритма для дизайна персонажа

Метод находит свое применение и в медицинских исследованиях - с его помощью по нескольким рентгеновским снимкам можно построить трехмерную модель кости, или органа(при использовании контрастного вещества), что увеличивает успешность операции. Такой способ позволяет избежать чрезмерного радиационного излучения, который получает пациент при прохождении компьютерной томографии.

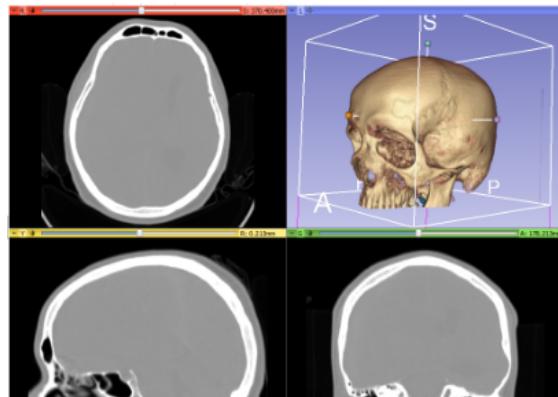


Рис. 3. Применение алгоритма для медицинских исследований

Впрочем, метод имеет множественные недостатки, наиболее значимым из которых является невысокая абсолютная точность получаемой объемной модели, что приводит к трудностям работы с объектами малых размеров. С крупногабаритными предметами также возникают проблемы, ведь алгоритм требует большого количества фотографий, имеющих сильное перекрытие(два соседних кадра должны совпадать минимум на треть) и полного обхода сканируемого объекта со всех возможных ракурсов. Модель по большому количеству кадров строится долго, а в случае отсутствия части информации или небрежности при съемке строится лишь частично - в ней могут появиться "дыры"или сильные искажения.



Рис. 4. Пример совмещения двух снимков. Последняя картинка - векторное поле совмещения таргетных точек.

Алгоритм реконструкции основывается на наличии на фотографиях определенных особенностях объекта, с помощью которых можно сопоставить сосед-

ние снимки. Однако если тело симметрично и не имеет таргетных точек, то совмещение кадров в единый объект становится затруднительным.

Понять суть проблемы можно если представить себе одноцветную сферу. Два снимка сферы сделанные с разных ракурсов будут совершенно идентичны, их можно совместить бесконечным числом способов, однако реконструкция должна быть однозначна.

## 2) Использование 3D-сканеров.

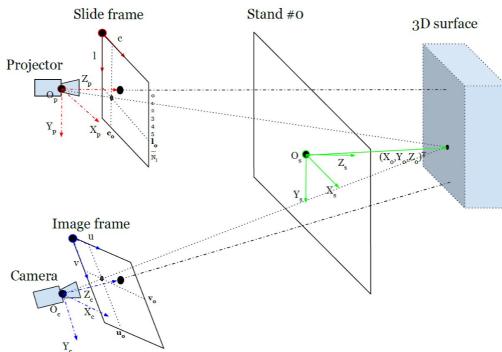


Рис. 5. Модель 3D-сканера

Будем называть 3D-сканером оптическую систему, состоящую из двух устройств: камеры и излучателя, проектирующего некоторый известный паттерн на объект. Сегодня подобные системы обретают все большую популярность и распространённость в ряде индустрий.

Существует достаточно широкое разнообразие 3D-сканеров: от способных обработать только небольшие объекты, до промышленных сканеров, работающих с машинами или крупногабаритными деталями сложных устройств; от аппаратов с ручным способом сканирования, до полностью автоматизированных систем. Также различается точность построения моделей и способность улавливать текстуру материала. Соответственно сканеры с разными характеристиками находят применение к разным задачам. Одной из таких задач является контроль качества выпускаемой продукции. Применение промышленного сканера к такого рода задачам позволяет значительно сократить время прохождения технического контроля. Также использование 3D-сканеров упрощает и ускоряет реверс-инженеринг и создании 3D-документации, что находит применение практически в любой промышленной сфере. Помимо этого сканеры находят свое применение и в области медицины, а именно: с их помощью производится протезирование, используются в пластической хирургии и ортопедии. Еще одно применение технологии - археология: очень часто скелеты вымерших животных, найденные при раскопках являются неполными, однако из нескольких таких скелетов можно собрать полный образец. Не всегда это

можно сделать физически, однако благодаря 3D-сканированию такую процедуру можно выполнить виртуально.

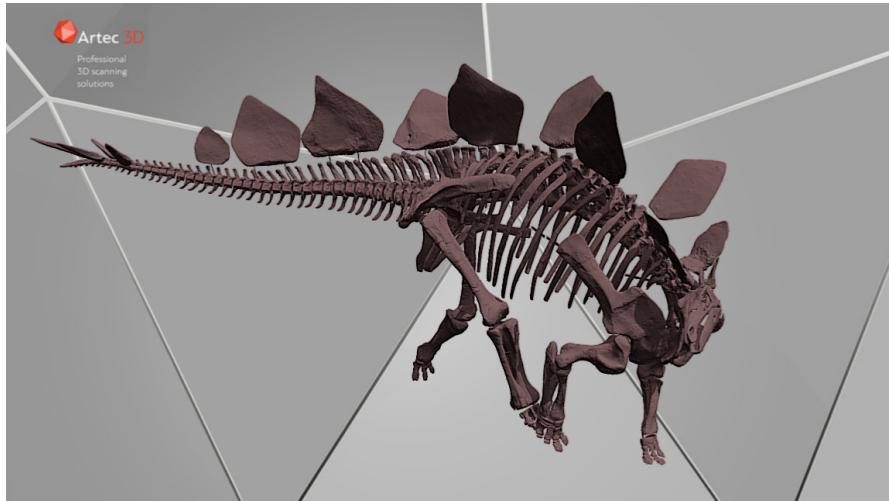


Рис. 6. 3D-модель скелета стегозавра

Существенным недостатком данного способа восстановления объемной модели является ее высокая стоимость, в сравнении со стоимостью проведения реконструкции предыдущим способом. Однако помимо этого метод имеет также ряд других проблем. Его точность также не достаточно высока, хотя и на порядок выше, чем точность реконструкции из набора фотографий. Проблема обработки симметричных объектов тоже актуальна для работы со сканерами. Помимо этого появляются дополнительные сложности в работе с прозрачными или блестящими объектами. Причиной этих проблем является наличие проектора в оптической системе сканера. Для того, чтобы процесс сканирования таких объектов происходил корректно их необходимо предварительно напылить.

3) Использование лидаров или оптических дальномеров.



Рис. 7. Модель Лидара

Принцип работы лидаров в некотором роде сходен принципу работы радаров. Он также основан на измерении времени отражения волны от изучаемого объекта и вычислении расстояния на основании полученного времени. Однако

вместо СВЧ-радиоволн в лидарах используются волны оптического диапазона. На сегодняшний день лидары являются ключевой технологией, используемой при разработке автопилотируемых автомобилей.

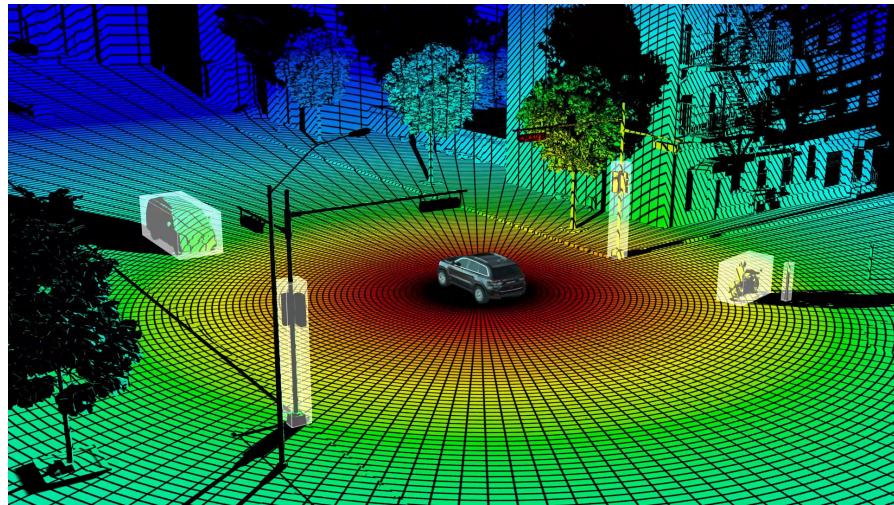


Рис. 8. Как автомобиль видит окружающую среду

Кроме того лидары активно используются при проведении аэрофотосъемки и составлении геодезических карт

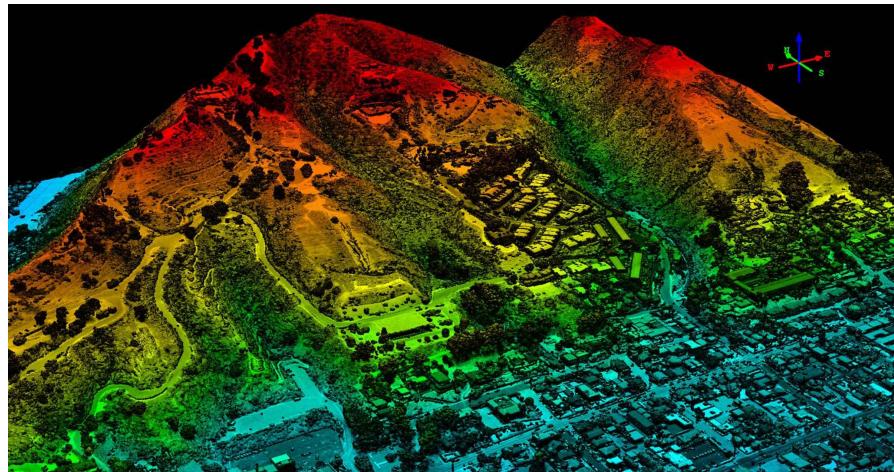


Рис. 9. Аэрофотосъемка

Минусом лидаров является в первую очередь их высокая стоимость. Помимо этого возникают проблемы с обработкой прозрачных объектов и тел малого размера.

- 4) Использование искусственных нейронных сетей для создания объемной модели

по одной или нескольким фотографиям.

Наилучших результатов удалось добиться при реконструкции двух классов объектов: человеческих лиц и положения людей на фотографиях. Для решения задачи реконструкции произвольных объектов нейросети не дают хороших результатов. В конечном итоге даже лучшая из нейронных сетей, существующих на данный момент - человеческий мозг, не позволяет достичь даже миллиметровой точности.

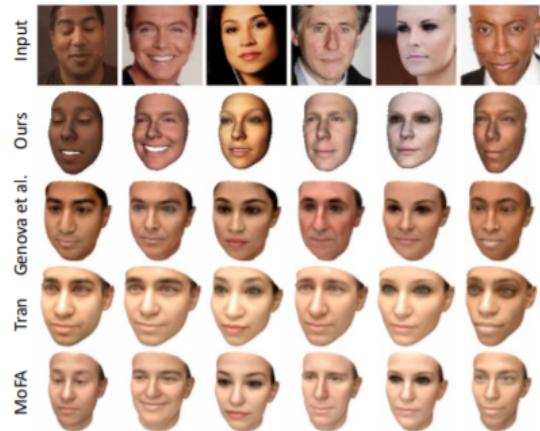


Рис. 10. Реконструкция лиц с использованием нейросетевых алгоритмов



Рис. 11. Определение позы человека

Стоит заметить, что комбинация нейросетевых алгоритмов с другими методами часто позволяет значительно повысить итоговое качество получаемой мо-

дели. Поэтому нейронные сети можно рассматривать, как этап постобработки модели, полученной одним из методов, описанных выше.

Отметим, что практически во всех рассмотренных выше случаях искомая модель трехмерного объекта представляет собой некоторый, возможно, весьма сложный многогранник, который в рамках требуемой точности задается наборами вершин, ребер и плоских граней. Исходные данные, используемые в данном случае представляют собой либо проекции отдельных зон реального тела на заданные плоскости (фотографии, сканы), либо расстояния до отдельных точек тела, измеренные от некоторой фиксированной точки в заданном направлении (лидары и т.п.). Таким образом, задача реконструкции тела опирается на простейшие соотношения аналитической геометрии, но в условиях присутствия шумов, ошибок и погрешностей измерений, что как раз и оказывает критическое влияние на точность получаемых моделей. В следующем разделе рассматривается подобная задача реконструкции трехмерной модели тела в одном частном случае технологической обработки драгоценных камней. В данной задаче возникают свои особенности, вызванные как специфическими требованиями к точности построения, так и требованиями эффективности при включении алгоритма в реальный производственный процесс.

## 1.2. Реконструкция кристаллов драгоценных камней

Суть этого метода заключается в том, что реальный ограненный камень устанавливается на врачающуюся платформу и фотографируется на фоне светлого экрана при разных углах поворота площадки. В результате получается серия из 800–1600 снимков "теневого контура" объекта, из которых потом и восстанавливается искомая модель.

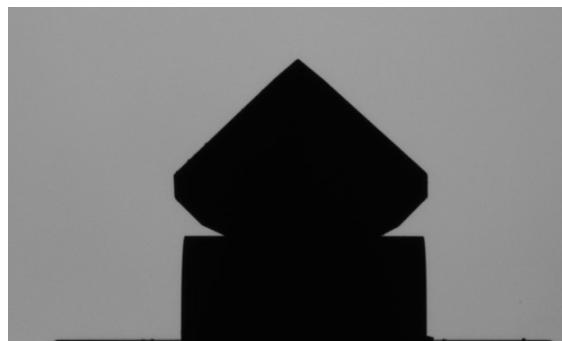


Рис. 12. Тень кристалла

Точность построения в этом случае получается порядка 5–10 микрон, что допустимо для некоторых задач, что все же не хватает для точной оценки качества огранки. Для уточнения применяется дополнительная фотосъемка "отраженных" граней по аналогичным принципам — при повороте камня под разными углами.

Огранка драгоценного камня — ключевой этап производства, который, как правило, оказывает решающее влияние на его конечную стоимость. Вследствие этого,

возникает необходимость автоматической оценки качества огранки. Что, в свою очередь, требует построения точной объемной модели кристалла.

Общим недостатком описанных выше подходов является относительно невысокая точность получаемой модели: ошибка в миллиметр допустима для сканирования практически любого объекта, но для работы с кристаллами драгоценных камней она неприемлемо высока. Помимо проблем с точностью возникает еще ряд особенностей, происходящих из природы кристалла. А именно драгоценный камень прозрачен и симметричен. Эти его свойства значительно осложняют работу с ним: перед сканированием его нужно напылить или заклеить неотражаемой пленкой.

Существует метод построения модели по теневым контурам, разработанный и применяемый специально для задач ювелирной промышленности. Суть этого метода заключается в том, что камень, закрепленный на специальную подставку и освещаемый лампой, поворачивается на заданный угол, и в каждом таком положении фиксируется тень, отбрасываемая им на экран.

Однако этот метод тоже имеет ряд недостатков. В частности, при его применении в модели могут возникнуть дополнительные ребра, одна грань может быть разбита на несколько и т.д. Точность такой модели также сравнительно невысока, однако полученный многогранник можно рассматривать в качестве первого приближения для более точной модели.

Каким образом можно добиться уточнения модели? Одним из способов достижения этой цели является предварительное построение точных ребер кристалла и последующее уточнение многогранника по этим ребрам. Для этого грань, в которой лежит интересующее нас ребро подсвечивается и делается несколько ее разноракурсных фотографий. По набору таких фотографий восстанавливается искомое ребро. Уточненные таким образом ребра будем далее называть целевыми.

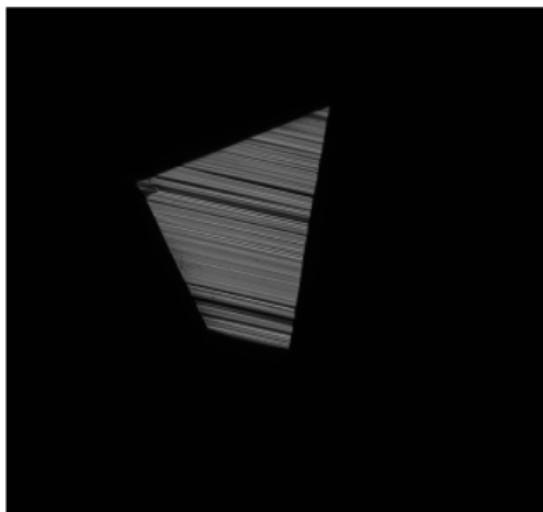
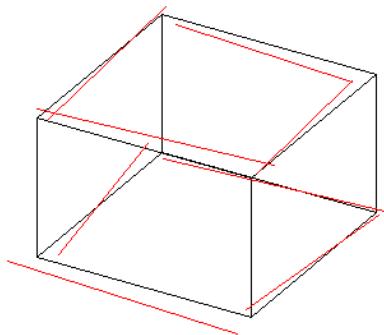


Рис. 13. Подсвеченная грань кристалла

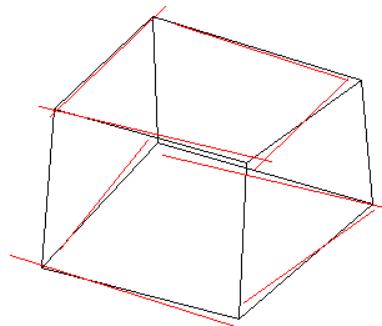
После построения целевых ребер мы имеем набор пространственных отрезков,

вычисленных со значительно более высокой точностью, чем ребра имеющейся модели кристалла, полученной методом теневых контуров (или каким-либо другим методом). Теперь перед нами стоит задача перестроить многогранник так, чтобы он наилучшим образом соответствовал набору целевых ребер.

Приведем графическую иллюстрацию задачи с помощью одного из наиболее просто устроенных многогранников - куба. На рисунке ниже красным изображены целевые ребра, а черным ребра многогранника.



Исходный многогранник



Измененный многогранник

Формально определить степень близости многогранника к набору пространственных отрезков можно многими способами. Однако рисунок выше наглядно демонстрирует неформальное определение этой близости. Здесь показан уточненный многогранник при использовании некоторой метрики расстояния между отрезками. Видно, что в перестроенном варианте ребра модели находятся визуально "ближе" к целевым ребрам, чем это было в исходной конфигурации

Изучению задачи построение многогранника наименее уклоняющегося от набора целевых ребер и сравнению различных методов решения этой задачи посвящена данная дипломная работа.

### 1.3. Комментарии к структуре работы

Кратко опишем структуру работы для лучшей ориентации читателя в ней. Во второй главе описан алгоритм построения целевых ребер. В третьей главе приводится алгоритм построения многогранника по набору полупространств - заключительный этап для обработки результатов оптимизации. Основная часть работы состоит из глав 4-7. В главе 4 математически формулируется задача условной минимизации функционала удаленности модели от набора целевых ребер. В пятой главе описаны результаты численных экспериментов с этой моделью. Глава 6 - способ сведения задачи условной оптимизации к задаче безусловной оптимизации путем выбора иных

переменных. В седьмой главе проведены эксперименты, аналогичные таковым в главе 5, но для задачи безусловной оптимизации.

## 2. Алгоритм построения целевых ребер

### 2.1. Постановка задачи построения целевых ребер

Пусть имеется несколько фотографий одного ребра многогранника, сделанных с разных ракурсов. Ребро на каждой из фотографий - суть набор двумерных точек на фокальной плоскости. Требуется восстановить ребро: предъявить пространственные координаты его начала и конца, опираясь на имеющиеся данные. Делается это в два этапа: сначала по набору точек в фокальных плоскостях строится наименее уклоняющийся отрезок проекции, а затем по набору таких отрезков строится пространственное ребро.

### 2.2. Формула проекции на вектор

Приведем формулу проекции вектора  $\bar{a}$  на вектор  $\bar{b}$ .

Для вектора  $\bar{a}$  справедливо разложение по трем линейно независимым векторам:

$$\bar{a} = k_1 \bar{b} + \underbrace{k_2 \bar{n}_1 + k_3 \bar{n}_2}_{d_1 \bar{n}}$$

где векторы  $\bar{n}_1$  и  $\bar{n}_2$  – взаимно ортогональные векторы из плоскости, перпендикулярной вектору  $\bar{b}$ . Линейная комбинация этих векторов определяет  $d_1 \bar{n}$  – нормальную компоненту вектора  $\bar{a}$ , где  $\bar{n}$  – единичный вектор. А  $k_1 \bar{b}$ , в свою очередь, является проекцией на вектор  $\bar{b}$ . Таким образом, для того, чтобы найти проекцию вектора  $\bar{a}$  на вектор  $\bar{b}$  необходимо вычислить коэффициент  $k_1$ .

Скалярно умножим разложение вектора  $\bar{a}$  на  $\bar{b}$ :

$$(\bar{a}, \bar{b}) = k_1 (\bar{b}, \bar{b}) + \underbrace{d_1 (\bar{n}, \bar{b})}_0 \Rightarrow k_1 = \frac{(\bar{a}, \bar{b})}{(\bar{b}, \bar{b})}$$

Таким образом формула проекции вектора  $\bar{a}$  на вектор  $\bar{b}$  принимает вид:

$$pr_{\bar{b}}^{\bar{a}} = \frac{(\bar{a}, \bar{b})}{(\bar{b}, \bar{b})} \bar{b}$$

Нормальную компоненту можно вычислить с помощью теоремы Пифагора:

$$d_1^2 = (\bar{a}, \bar{a}) - \left( \frac{(\bar{a}, \bar{b})}{(\bar{b}, \bar{b})} \right)^2 (\bar{b}, \bar{b}) = (\bar{a}, \bar{a}) - \frac{(\bar{a}, \bar{b})^2}{(\bar{b}, \bar{b})}$$

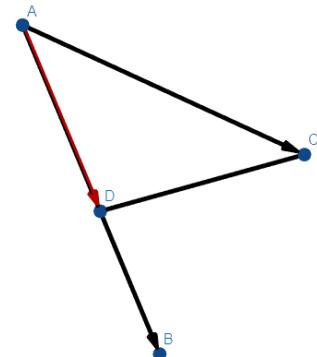


Рис. 14. Проекция на вектор

### 2.3. Прямая, наименее уклоняющаяся от набора точек

Заметим, что по набору точек проекции(пикселей на фотографиях ребра) можно построить наименее уклоняющийся отрезок и свести задачу построения целевого ребра к случаю, когда на каждой плоскости проекции (т.е. на каждом снимке) задан отрезок проекции.

Для построения такого отрезка по облаку точек необходимо сначала построить наименее уклоняющуюся прямую от этого облака.

**Определение:** Пусть имеется множество точек  $p_i = (x_i, y_i, z_i)_{i=1}^n$ , назовем прямую  $l$  наименее уклоняющейся от данного набора, если она доставляет минимум функционалу:

$$f(l) = \sum_{i=1}^n dist(l, p_i)^2 \rightarrow \min$$

, где через  $dist(l, p_i)$  обозначено расстояние между искомой прямой и соответствующей точкой, определенное стандартным образом в евклидовой метрике.

**Лемма 1.** Наименее уклоняющаяся прямая проходит через центр масс набора точек.

*Доказательство.* Будем доказывать от противного. Предположим, что центр масс  $M_1$  не лежит на наименее уклоняющейся прямой  $l$ :

$$M_1 \notin l$$

Проведём два дополнительных построения:

- 1) Проведем через  $M_1$  прямую  $l'$ , параллельную  $l$ .
- 2) Обозначим через  $M_2$  проекцию  $M_1$  на  $l$ .

Теперь для каждой точки множества  $P_i$  обозначим через  $C_i$  проекцию этой точки на прямую  $l'$ , а через  $D_i$  - проекцию на  $l$ , соответственно.

Заметим, что:  $\begin{cases} P_i C_i \perp C_i M_1 \\ P_i D_i \perp D_i M_2 \end{cases} \Rightarrow$  (по теореме о трех перпендикулярах)  $\Rightarrow C_i D_i \perp C_i M_1$

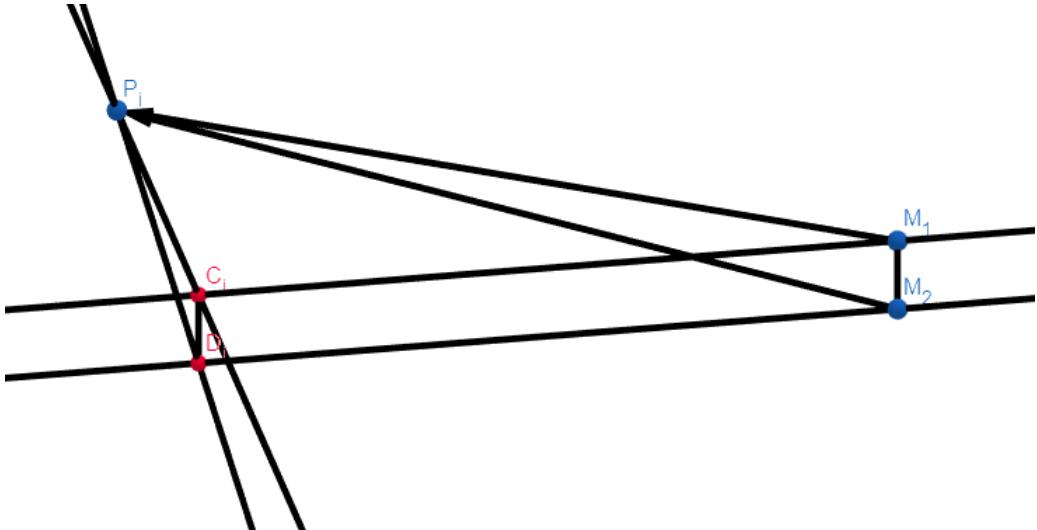
А т.к. помимо этого  $M_1 M_2 \perp C_i M_1$  и  $C_i M_1 \parallel D_i M_2$ , то  $M_1 M_2 D_i C_i$  – прямоугольник  $\Rightarrow M_1 C_i = M_2 D_i$

Прямая  $l$  – наименее уклоняющаяся прямая, следовательно, т.к. минимум – единственен, то

$$f(l) < f(l') \Rightarrow \sum_{i=1}^n dist(l, p_i)^2 < \sum_{i=1}^n dist(l', p_i)^2$$

Согласно теореме Пифагора:

$$M_2 P_i^2 = dist(l, p_i)^2 + D_i M_2^2$$



$$M_1 P_i^2 = \text{dist}(l', p_i)^2 + C_i M_1^2$$

Суммируя по индексу получаем:

$$\begin{aligned} \sum_{i=1}^n M_2 P_i^2 &= f(l) + \sum_{i=1}^n D_i M_2^2 \underset{M_1 C_i = M_2 D_i}{=} f(l) + \sum_{i=1}^n C_i M_1^2 < f(l') + \sum_{i=1}^n C_i M_1^2 = \sum_{i=1}^n M_1 P_i^2 \Rightarrow \\ &\Rightarrow \sum_{i=1}^n M_2 P_i^2 < \sum_{i=1}^n M_1 P_i^2 \end{aligned}$$

Однако  $M_1$  – центр масс, а значит наименее уклоняющаяся от множества точка.

$$\Rightarrow \sum_{i=1}^n M_2 P_i^2 > \sum_{i=1}^n M_1 P_i^2$$

Таким образом мы пришли к противоречию. Значит, наше предположение было ошибочным, и  $M_1 \in l$ .  $\square$

Для того, чтобы задать прямую в параметрическом виде нужна точка прямой и направление. Из леммы следует, что центр масс на прямой лежит, осталось найти направление  $(\alpha, \beta, \gamma)$ .

**Лемма 2.** *Направляющим вектором прямой, наименее уклоняющейся от облака точек является собственный вектор матрицы*

$$\begin{pmatrix} \sum_{i=1}^n \tilde{x}_i \tilde{x}_i & \sum_{i=1}^n \tilde{x}_i \tilde{y}_i & \sum_{i=1}^n \tilde{x}_i \tilde{z}_i \\ \sum_{i=1}^n \tilde{y}_i \tilde{x}_i & \sum_{i=1}^n \tilde{y}_i \tilde{y}_i & \sum_{i=1}^n \tilde{y}_i \tilde{z}_i \\ \sum_{i=1}^n \tilde{z}_i \tilde{x}_i & \sum_{i=1}^n \tilde{z}_i \tilde{y}_i & \sum_{i=1}^n \tilde{z}_i \tilde{z}_i \end{pmatrix}$$

, где  $(\tilde{x}_i, \tilde{y}_i, \tilde{z}_i) = (x_i - x_M, y_i - y_M, z_i - z_M)$  - центрированные координаты точек из облака.

*Доказательство.* Воспользуемся формулой нахождения расстояния от точки до прямой:

$$d_1^2 = (\bar{a}, \bar{a}) - \frac{(\bar{a}, \bar{b})^2}{(\bar{b}, \bar{b})}$$

Тогда

$$f(\alpha, \beta, \gamma) = \sum_{i=1}^n \underbrace{(x_i - x_M)^2 + (y_i - y_M)^2 + (z_i - z_M)^2}_{=Const_i} - \frac{(\overbrace{\alpha \tilde{x}_i}^{\tilde{x}_i} + \overbrace{\beta \tilde{y}_i}^{\tilde{y}_i} + \overbrace{\gamma \tilde{z}_i}^{\tilde{z}_i})^2}{\alpha^2 + \beta^2 + \gamma^2}$$

, где  $(x_M, y_M, z_M)$  – координаты центра масс.

Наличие квадрата длины направляющего вектора в знаменателе нам мешает, поэтому добавим условие его нормировки  $\alpha^2 + \beta^2 + \gamma^2 - 1 = 0$  и найдем условный минимум.

Лагранжиан, после отбрасывания постоянной части, в этом случае будет равен:

$$\mathcal{L}(\alpha, \beta, \gamma, \lambda) = \sum_{i=1}^n -(\alpha \tilde{x}_i + \beta \tilde{y}_i + \gamma \tilde{z}_i)^2 + \lambda(\alpha^2 + \beta^2 + \gamma^2 - 1)$$

Возьмём частные производные и приравняем их к нулю

$$\begin{aligned} -\frac{1}{2} \frac{\partial \mathcal{L}}{\partial \alpha} &= \sum_{i=1}^n \tilde{x}_i \tilde{x}_i \alpha + \sum_{i=1}^n \tilde{x}_i \tilde{y}_i \beta + \sum_{i=1}^n \tilde{x}_i \tilde{z}_i \gamma - \lambda \alpha = 0 \\ -\frac{1}{2} \frac{\partial \mathcal{L}}{\partial \beta} &= \sum_{i=1}^n \tilde{y}_i \tilde{x}_i \alpha + \sum_{i=1}^n \tilde{y}_i \tilde{y}_i \beta + \sum_{i=1}^n \tilde{y}_i \tilde{z}_i \gamma - \lambda \beta = 0 \\ -\frac{1}{2} \frac{\partial \mathcal{L}}{\partial \gamma} &= \sum_{i=1}^n \tilde{z}_i \tilde{x}_i \alpha + \sum_{i=1}^n \tilde{z}_i \tilde{y}_i \beta + \sum_{i=1}^n \tilde{z}_i \tilde{z}_i \gamma - \lambda \gamma = 0 \end{aligned}$$

Кроме того остается условие нормировки:

$$\alpha^2 + \beta^2 + \gamma^2 - 1 = 0$$

Если записать уравнения в матричном виде

$$\begin{pmatrix} \sum_{i=1}^n \tilde{x}_i \tilde{x}_i - \lambda & \sum_{i=1}^n \tilde{x}_i \tilde{y}_i & \sum_{i=1}^n \tilde{x}_i \tilde{z}_i \\ \sum_{i=1}^n \tilde{y}_i \tilde{x}_i & \sum_{i=1}^n \tilde{y}_i \tilde{y}_i - \lambda & \sum_{i=1}^n \tilde{y}_i \tilde{z}_i \\ \sum_{i=1}^n \tilde{z}_i \tilde{x}_i & \sum_{i=1}^n \tilde{z}_i \tilde{y}_i & \sum_{i=1}^n \tilde{z}_i \tilde{z}_i - \lambda \end{pmatrix} \times \begin{pmatrix} \alpha \\ \beta \\ \gamma \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

□

Таким образом мы пришли к задаче поиска единичных собственных векторов для симметрической матрицы. После решения этой задачи мы отнормируем полученные вектора, а затем выберем из них тот, который доставляет минимум функционала  $f(\alpha, \beta, \gamma)$ .

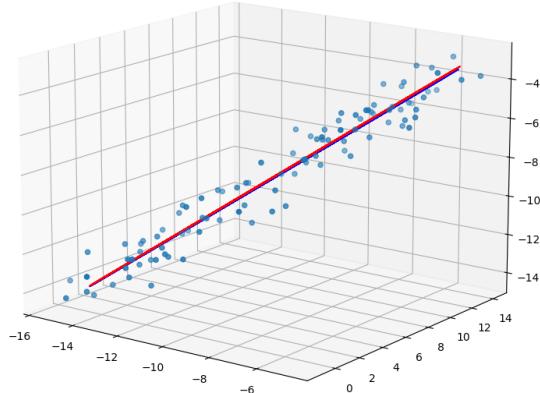


Рис. 15. Построенная наименее уклоняющаяся прямая

Для построения наименее уклоняющегося отрезка осталось только найти минимум и максимум скалярного произведения векторов  $(\tilde{x}_i, \tilde{y}_i, \tilde{z}_i)$  и  $(\alpha, \beta, \gamma)$ .

## 2.4. Построение ребра по отрезкам проекций

Теперь опишем алгоритм построения объемного отрезка по его проекциям на набор плоскостей.

### Предварительные замечания:

- 1) Считаем, что по аппликате проекции точны. Поэтому  $z_1, z_2$  определены.
- 2) Нормой удаленности отрезка и плоскости считаем сумму квадратов расстояний между концами отрезка и плоскостью  $d_1^2 + d_2^2$
- 3) Расстояние от точки  $(x_0, y_0, z_0)$  до плоскости  $Ax + By + Cz + D = 0$  вычисляется с помощью формулы

$$\frac{|Ax_0 + By_0 + Cz_0 + D|}{\sqrt{A^2 + B^2 + C^2}}$$

### **Шаг 1**

Строим плоскости по отрезку проекции и направляющему вектору. Предварительно обозначим  $a_1 = x_2 - x_1, b_1 = y_2 - y_1, c_1 = z_2 - z_1$ , где  $(x_1, y_1, z_1)$  и  $(x_2, y_2, z_2)$ -координаты начала и конца отрезка проекции;  $(a, b, c)$ -направляющий вектор для плоскости. Запишем уравнение плоскости:

$$\begin{vmatrix} x - x_1 & y - y_1 & z - z_1 \\ a & b & c \\ a_1 & b_1 & c_1 \end{vmatrix} = 0 \Rightarrow$$

$$\Rightarrow (x - x_1)bc_1 + (y - y_1)ca_1 + (z - z_1)ab_1 - (z - z_1)ba_1 - (x - x_1)cb_1 - (y - y_1)ac_1 = 0$$

Получаем коэффициенты для уравнения плоскости  $Ax + By + Cz + D = 0$ :

$$A = bc_1 - cb_1$$

$$B = ca_1 - ac_1$$

$$C = ab_1 - ba_1$$

$$D = -x_1(bc_1 - cb_1) - y_1(ca_1 - ac_1) - z_1(ab_1 - ba_1)$$

Осталось нормировать коэффициенты  $A, B, C$  чтобы  $A^2 + B^2 + C^2 = 1$ , для этого поделим их и коэффициент  $D$  на  $\sqrt{A^2 + B^2 + C^2}$ .

### **Шаг 2**

Вычислим сумму расстояний от отрезка с концами  $(x_1, y_1, z_1), (x_2, y_2, z_2)$  и минимизируем ее:

$$S = \sum_i (a_i x_1 + b_i y_1 + c_i z_1 + d_i)^2 + (a_i x_2 + b_i y_2 + c_i z_2 + d_i)^2 \rightarrow \min$$

Для этого дифференцируем  $S$  по  $x_1, x_2, y_1, y_2$  и приравниваем частные производные к нулю.

$$\frac{1}{2} \frac{\partial S}{\partial x_1} = (\sum_i a_i^2)x_1 + (\sum_i a_i b_i)y_1 + \sum_i a_i(c_i z_1 + d_i) = 0$$

$$\frac{1}{2} \frac{\partial S}{\partial y_1} = (\sum_i a_i b_i)x_1 + (\sum_i b_i^2)y_1 + \sum_i b_i(c_i z_1 + d_i) = 0$$

$$\frac{1}{2} \frac{\partial S}{\partial x_2} = \left( \sum_i a_i^2 \right) x_2 + \left( \sum_i a_i b_i \right) y_2 + \sum_i a_i (c_i z_2 + d_i) = 0$$

$$\frac{1}{2} \frac{\partial S}{\partial y_2} = \left( \sum_i a_i b_i \right) x_2 + \left( \sum_i b_i^2 \right) y_2 + \sum_i b_i (c_i z_2 + d_i) = 0$$

Теперь осталось решить 2 системы  $2 \times 2$ , чтобы найти  $x_1, y_1, x_2, y_2$ . Тем самым мы нашли требуемое ребро. Таким образом алгоритм построения целевых ребер полностью описан.

### 3. Построение многогранника по набору полупространств

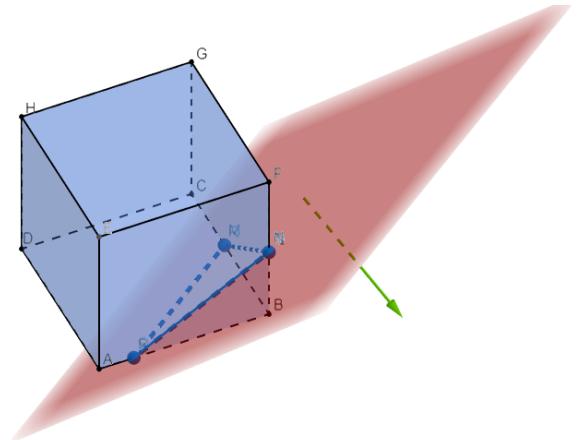
Пусть имеется некоторый набор плоскостей, которые задают грани нового многогранника и стартовый многогранник. Опишем алгоритм построения результирующего многогранника.

Для начала получим из набора плоскостей-полупространства. Для этого вычислим центр масс исходного многогранника и подставим его в уравнения плоскостей, если получаем положительное число, то плоскость задана внутренней нормалью, поэтому умножим коэффициенты плоскости на -1. Теперь мы имеем множество полупространств, пересечением которых является искомый многогранник.

Перейдем к алгоритму построения многогранника по набору полупространств:

Изначально мы создаем объемлющий куб, от которого будем отсекать полупространства. Опишем процедуру отсечения плоскости от куба. Мы проходим по всем граням объемлющего куба и в каждой грани смотрим на пересечение всех ребер этой грани с отсекаемой плоскостью. После получения точек пересечения мы изменяем ребра (берем только ту часть ребра, которая лежит в нужном полупространстве) и добавляем к грани ребро пресечения. Ребра, целиком лежащие в противоположном полупространстве удаляются. Запоминая ребра пересечения мы формируем из них новую грань объемлющего куба (теперь это уже не куб, а объемлющий многогранник) и добавляем ее к его структуре. Проделывая так со всем списком полупространств мы получаем некий многогранник. Однако исходно объемлющий куб мог быть слишком мал, поэтому если после пересечения полупространств в многограннике осталась часть грани исходного объемлющего куба, то мы увеличиваем его размер вдвое и запускаем процесс заново.

Следует отметить, что алгоритм требует рассмотрения большого числа крайних случаев: пересечение по вершине, ребру, плоскости.



```

x=1;
cub=createCub(x);
while True do
    oldFacets=cub.facets;
    for plane  $\in$  planes do
        newFacet= $\emptyset$ ;
        newEdges= $\emptyset$ ;
        for facet  $\in$  cub.facets do
            newPoints= $\emptyset$ ;
            for edge  $\in$  facet.edges do
                if edge  $\cap$  plane  $\neq \emptyset$  then
                    | newPoints=newPoints  $\cup$  (edge  $\cap$  plane);
                end
                reconstructEdge(edge);
            end
            reconstructFacet(facet);
            newEdges=newEdges  $\cup$  edgeConstruct(newPoints)
        end
        newFacet=facetConstruct(newEdges);
        cub.facets=cub.facets  $\cup$  newFacet;
    end
    if cub.facets  $\cap$  oldFacets = $\emptyset$  then
        | break;
    else
        | x=2x;
        | cub=createCub(x);
    end
end

```

## 4. Задача условной оптимизации

### 4.1. Математическая формализация задачи. Построение минимизируемого функционала

Имеется многогранник  $M$ , заданный наборами своих граней, ребер и вершин.

$$\begin{cases} p_i = (x_i, y_i, z_i) - & \text{Координаты вершин многогранника, } i = \overline{1, n_1} \\ e_j = (a_{j,1}, a_{j,2}) - & \text{Номера вершин, которые являются} \\ & \text{концами ребра } e_j, j = \overline{1, n_2} \\ f_k = [b_{k,1}, \dots, b_{k,m_k}] - & \text{Номера ребер, входящих в грань } f_k, k = \overline{1, n_3} \end{cases}$$

Имеется список целевых ребер, которые считаются точными и которые поставлены в соответствие исходным ребрам многогранника.

$$\tilde{e}_i = \begin{cases} (a_{i,x}, a_{i,y}, a_{i,z}), (b_{i,x}, b_{i,y}, b_{i,z}) & \text{координаты двух концов данного целевого ребра.} \\ ind_i & \text{номер исходного ребра, которому соответствует } i\text{-е целевое ребро.} \end{cases}$$

Если в списке целевых ребер нет элемента для текущего ребра, то считаем, что оно является целевым само для себя. Однако вес, с которым оно входит в функционал будет равен  $\omega_i = 0.1$ , в то время, как для полноценного целевого ребра  $\omega_i = 1$ . После расширения списка целевых ребер указанным способом, будем считать, что  $\tilde{e}_i$ -целевое ребро, соответствующее исходному ребру  $e_i$  многогранника  $M$ .

Сделаем ребра многогранника "подвижными" для этого введём переменные  $\tilde{p}_{i,1} = (\tilde{x}_{i,1}, \tilde{y}_{i,1}, \tilde{z}_{i,1})$ ,  $\tilde{p}_{i,2} = (\tilde{x}_{i,2}, \tilde{y}_{i,2}, \tilde{z}_{i,2})$ , соответствующие концам "подвижных" ребер. Мы хотим найти такую конфигурацию, чтобы эти ребра были наименее удалены от соответствующим им целевых. Для этого мы должны доставить минимум следующего функционала:

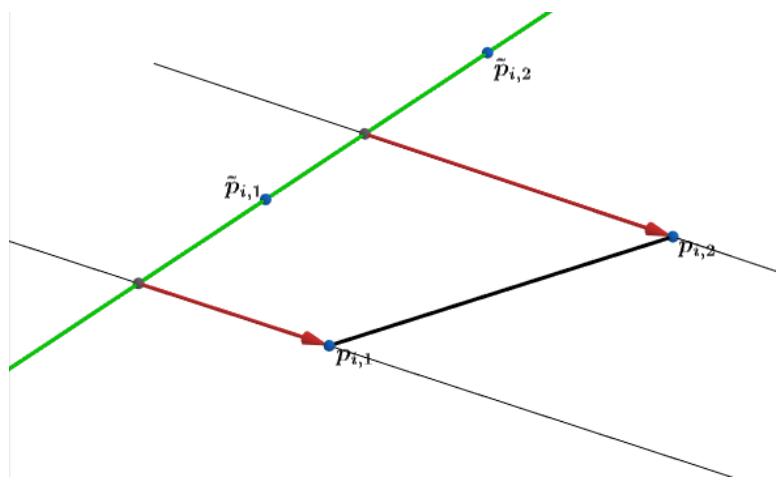


Рис. 16. Расстояние  $\rho$

$$\sum_{i=1}^{n_2} \omega_i (\rho^2(\tilde{p}_{i,1}, \tilde{e}_i) + \rho^2(\tilde{p}_{i,2}, \tilde{e}_i)) \rightarrow \min, \text{ где}$$

$\rho(\tilde{p}_{i,j}, \tilde{e}_i)$ -расстояние между концами "движимого"ребра и прямой, заданной соответствующим целевым ребром.

#### 4.2. Доказательство выпуклости функционала.

Ряд алгоритмов оптимизации требует выпуклости минимизируемого функционала. В частности, соблюдение этого свойства необходимо для корректной работы метода внутренней точки, который в дальнейшем будет активно применяться для построения моделей.

Докажем, что рассматриваемый нами функционал-выпуклый. Рассмотрим одно слагаемое суммы, образующей наш функционал:

$$F = (x - a_x)^2 + (y - a_y)^2 + (z - a_z)^2 - \frac{((x - a_x)(b_x - a_x) + (x - a_y)(b_y - a_y) + (x - a_z)(b_z - a_z))^2}{(b_x - a_x)^2 + (b_y - a_y)^2 + (b_z - a_z)^2}$$

Введем обозначения:  $c_1 := (b_x - a_x)$ ,  $c_2 := (b_y - a_y)$ ,  $c_3 := (b_z - a_z)$ ,  $c := c_1^2 + c_2^2 + c_3^2$ . И запишем матрицу вторых частных производных  $F$ :

$$\begin{pmatrix} \frac{\partial^2 F}{\partial x^2} & \frac{\partial^2 F}{\partial x \partial y} & \frac{\partial^2 F}{\partial x \partial z} \\ \frac{\partial^2 F}{\partial y \partial x} & \frac{\partial^2 F}{\partial y^2} & \frac{\partial^2 F}{\partial y \partial z} \\ \frac{\partial^2 F}{\partial z \partial x} & \frac{\partial^2 F}{\partial z \partial y} & \frac{\partial^2 F}{\partial z^2} \end{pmatrix} = \begin{pmatrix} 2(1 - \frac{c_1^2}{c}) & -\frac{c_1 c_2}{c} & -\frac{c_1 c_3}{c} \\ -\frac{c_1 c_2}{c} & 2(1 - \frac{c_2^2}{c}) & -\frac{c_2 c_3}{c} \\ -\frac{c_1 c_3}{c} & -\frac{c_2 c_3}{c} & 2(1 - \frac{c_3^2}{c}) \end{pmatrix}$$

Проверим матрицу на неотрицательную определенность с помощью Критерия Сильвестра. Для этого вычислим определители главных миноров.

$$1) |\Delta_1| = 2 \frac{c_2^2 + c_3^2}{c} \geq 0$$

$$2) |\Delta_2| = \begin{vmatrix} 2(1 - \frac{c_1^2}{c}) & -\frac{c_1 c_2}{c} \\ -\frac{c_1 c_2}{c} & 2(1 - \frac{c_2^2}{c}) \end{vmatrix} = 4 - 4 \frac{c_1^2 + c_2^2}{c} + 4 \frac{c_1^2 c_2^2}{c^2} - \frac{c_1^2 c_2^2}{c^2} = 4 \frac{c_3^2}{c} + 3 \frac{c_1^2 c_2^2}{c^2} \geq 0$$

$$3) |\Delta_3| = \begin{vmatrix} 2(1 - \frac{c_1^2}{c}) & -\frac{c_1 c_2}{c} & -\frac{c_1 c_3}{c} \\ -\frac{c_1 c_2}{c} & 2(1 - \frac{c_2^2}{c}) & -\frac{c_2 c_3}{c} \\ -\frac{c_1 c_3}{c} & -\frac{c_2 c_3}{c} & 2(1 - \frac{c_3^2}{c}) \end{vmatrix} = 8(1 - \frac{c_1^2}{c})(1 - \frac{c_2^2}{c})(1 - \frac{c_3^2}{c}) - 2 \frac{c_1^2 c_2^2 c_3^2}{c^3} -$$

$$- 2 \frac{c_1^2 c_2^2}{c^2} (1 - \frac{c_2^2}{c}) - 2 \frac{c_2^2 c_3^2}{c^2} (1 - \frac{c_3^2}{c}) - \frac{c_1^2 c_2^2}{c^2} (1 - \frac{c_3^2}{c}) = 8 - 4 \frac{c_1^2 c_2^2 c_3^2}{c^3} + 6(\frac{c_1^2 c_2^2}{c^2} + \frac{c_1^2 c_3^2}{c^2} + \frac{c_2^2 c_3^2}{c^2}) - 8 \frac{c_1^2 + c_2^2 + c_3^2}{c} =$$

$$= 6(\frac{c_1^2 c_2^2}{c^2} + \frac{c_1^2 c_3^2}{c^2}) + 2(\frac{c_2^2 c_3^2}{c^2}) + 4 + \frac{c_2^2 c_3^2}{c^2} (1 - \frac{c_1^2}{c}) = 6(\frac{c_1^2 c_2^2}{c^2} + \frac{c_1^2 c_3^2}{c^2}) + 2(\frac{c_2^2 c_3^2}{c^2}) + 4 + \frac{c_2^2 c_3^2}{c^2} (\frac{c_2^2 + c_3^2}{c}) \geq 0$$

Тем самым мы показали, что матрица вторых частных производных функции- неотрицательна, следовательно эта функция является выпуклой. Осталось вспомнить, что сумма выпуклых функций также является выпуклой (Прямое следствие определения через неравенство Йенсена).

Таким образом, показана выпуклость задачи и, как следствие, корректность применения метода внутренней точки для её решения.

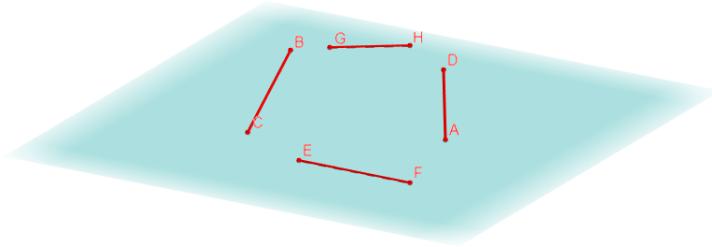
#### 4.3. Условия, налагаемые на функционал.

На функционал нужно наложить ряд ограничений, основным из которых является следующее: если исходные ребра лежали в одной грани, то и соответствующие "подвижные" ребра должны лежать в одной грани. Иными словами:

$$\forall k = \overline{1, n_3} \exists \text{ плоскость } \pi_k : \forall j \in f_k, \overline{\tilde{p}_{j,1}, \tilde{p}_{j,2}} \in \pi_k$$

Математически записать условие принадлежности 4-х точек одной плоскости можно с помощью определителя:

$$\begin{vmatrix} x_2 - x_1 & y_2 - y_1 & z_2 - z_1 \\ x_3 - x_1 & y_3 - y_1 & z_3 - z_1 \\ x_4 - x_1 & y_4 - y_1 & z_4 - z_1 \end{vmatrix} = 0$$



Соответственно для  $n$  точек это условие будет выглядеть следующим образом:

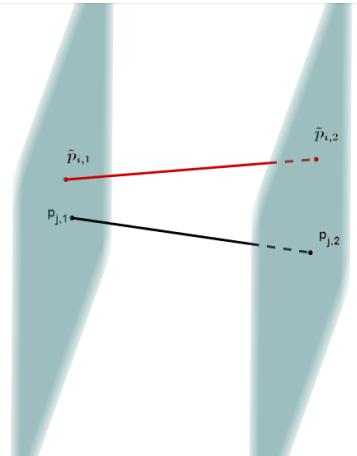
$$\left\{ \begin{array}{l} \begin{vmatrix} x_2 - x_1 & y_2 - y_1 & z_2 - z_1 \\ x_3 - x_1 & y_3 - y_1 & z_3 - z_1 \\ x_4 - x_1 & y_4 - y_1 & z_4 - z_1 \end{vmatrix} = 0 \\ \dots \\ \begin{vmatrix} x_2 - x_1 & y_2 - y_1 & z_2 - z_1 \\ x_3 - x_1 & y_3 - y_1 & z_3 - z_1 \\ x_5 - x_1 & y_5 - y_1 & z_5 - z_1 \end{vmatrix} = 0 \\ \dots \\ \begin{vmatrix} x_2 - x_1 & y_2 - y_1 & z_2 - z_1 \\ x_3 - x_1 & y_3 - y_1 & z_3 - z_1 \\ x_n - x_1 & y_n - y_1 & z_n - z_1 \end{vmatrix} = 0 \end{array} \right.$$

Однако такой способ записи довольно громоздкий и, помимо этого, дает "кубические" условия, налагаемые на задачу. С другой стороны можно ввести новые переменные  $A_k, B_k, C_k, D_k, \forall k = \overline{1, n}$ -коэффициенты из уравнений плоскостей. Они не участвуют в записи минимизируемого функционала, однако с помощью них условия принадлежности одной грани можно записать достаточно просто:

$$\begin{cases} A_k \tilde{x}_{j,1} + B_k \tilde{y}_{j,1} + C_k \tilde{z}_{j,1} + D_k = 0, \forall j \in f_k \\ A_k \tilde{x}_{j,2} + B_k \tilde{y}_{j,2} + C_k \tilde{z}_{j,2} + D_k = 0, \forall j \in f_k \\ A_k^2 + B_k^2 + C_k^2 = 1 - \text{нормировка коэффициентов} \end{cases}$$

Не менее важными являются условия принадлежности концов "подвижных" ребер нормальным плоскостям изначальных ребер многогранника (нечелевых). Их можно записать следующим образом:

$$\begin{cases} \tilde{p}_{j,1} \in N(p_{e_j[1]}, e_j) \\ \tilde{p}_{j,2} \in N(p_{e_j[2]}, e_j) \end{cases}$$



где  $N(p, e)$ -плоскость, нормальная к вектору  $e$  и проходящая через точку  $p$ .

Данные условия нужны для предотвращения разбегания подвижных ребер.

#### **4.4. Восстановление многогранника**

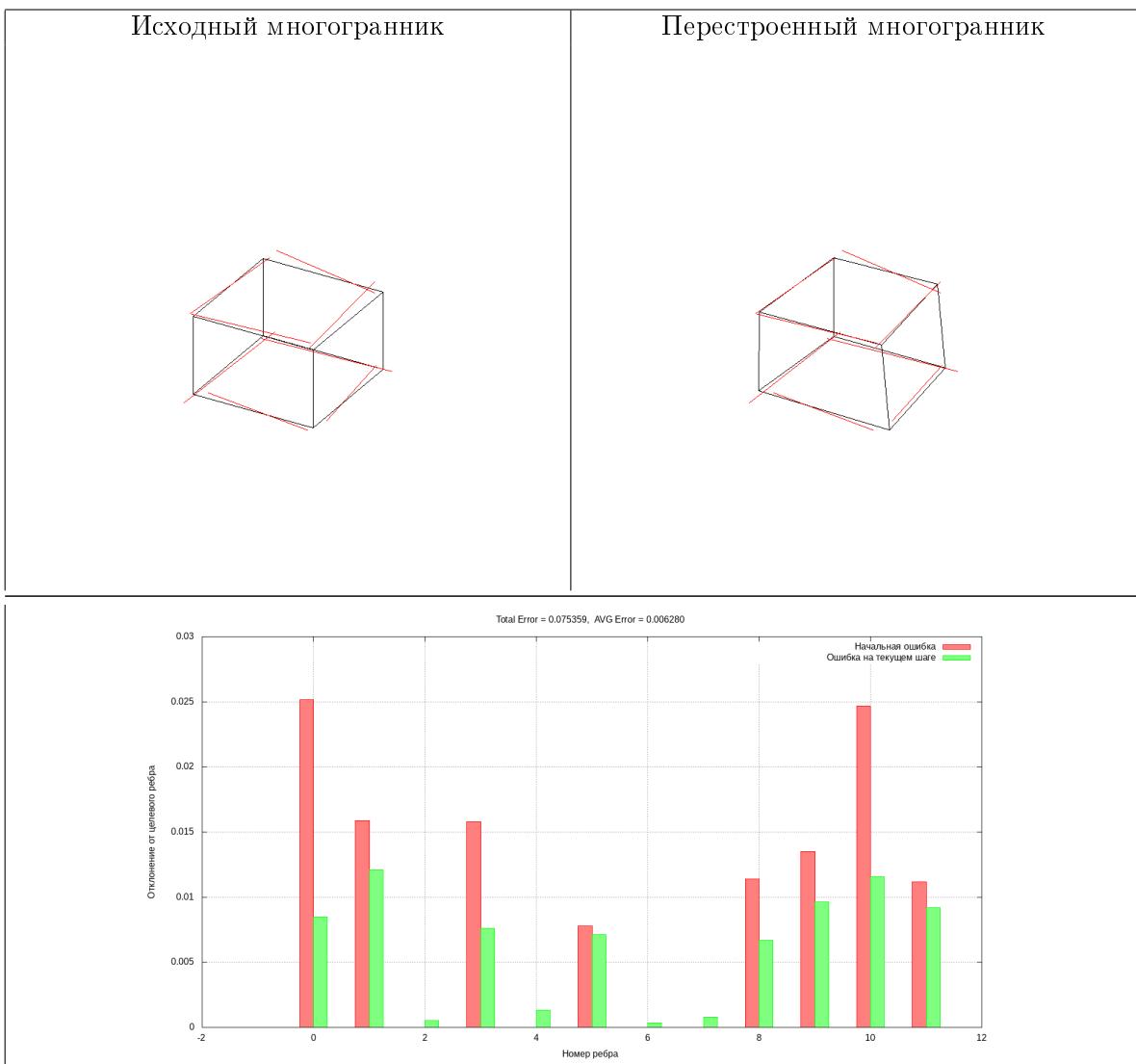
После решения задачи оптимизации мы имеем наборы подвижных ребер и плоскостей, в которых они лежат. Однако нам хотелось бы по этим данным построить многогранник. Решить эту задачу поможет алгоритм построения многогранника по набору полупространств, описанный в 3 главе.

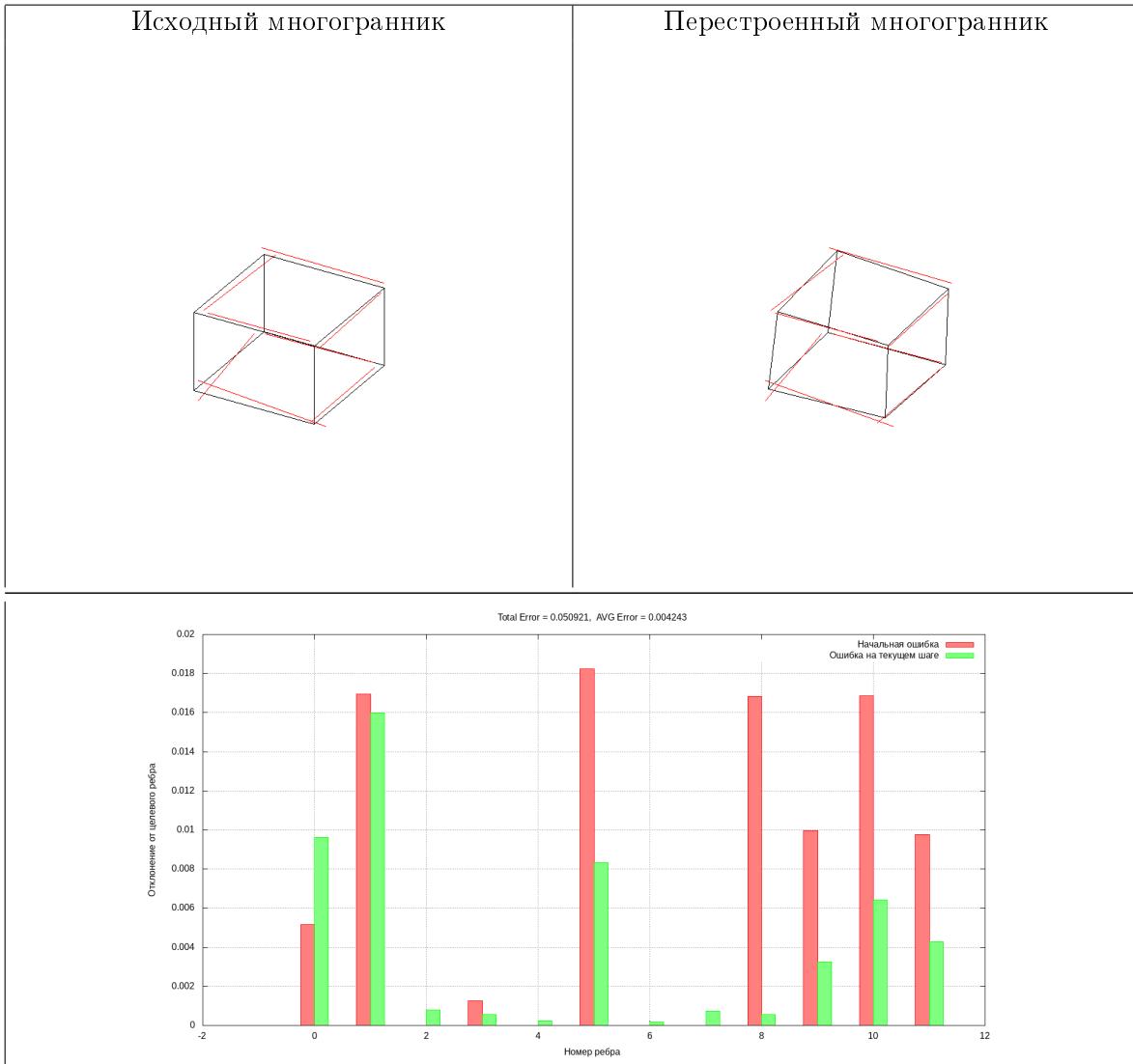
### **5. Результаты работы программной реализации алгоритма для задачи условной оптимизации**

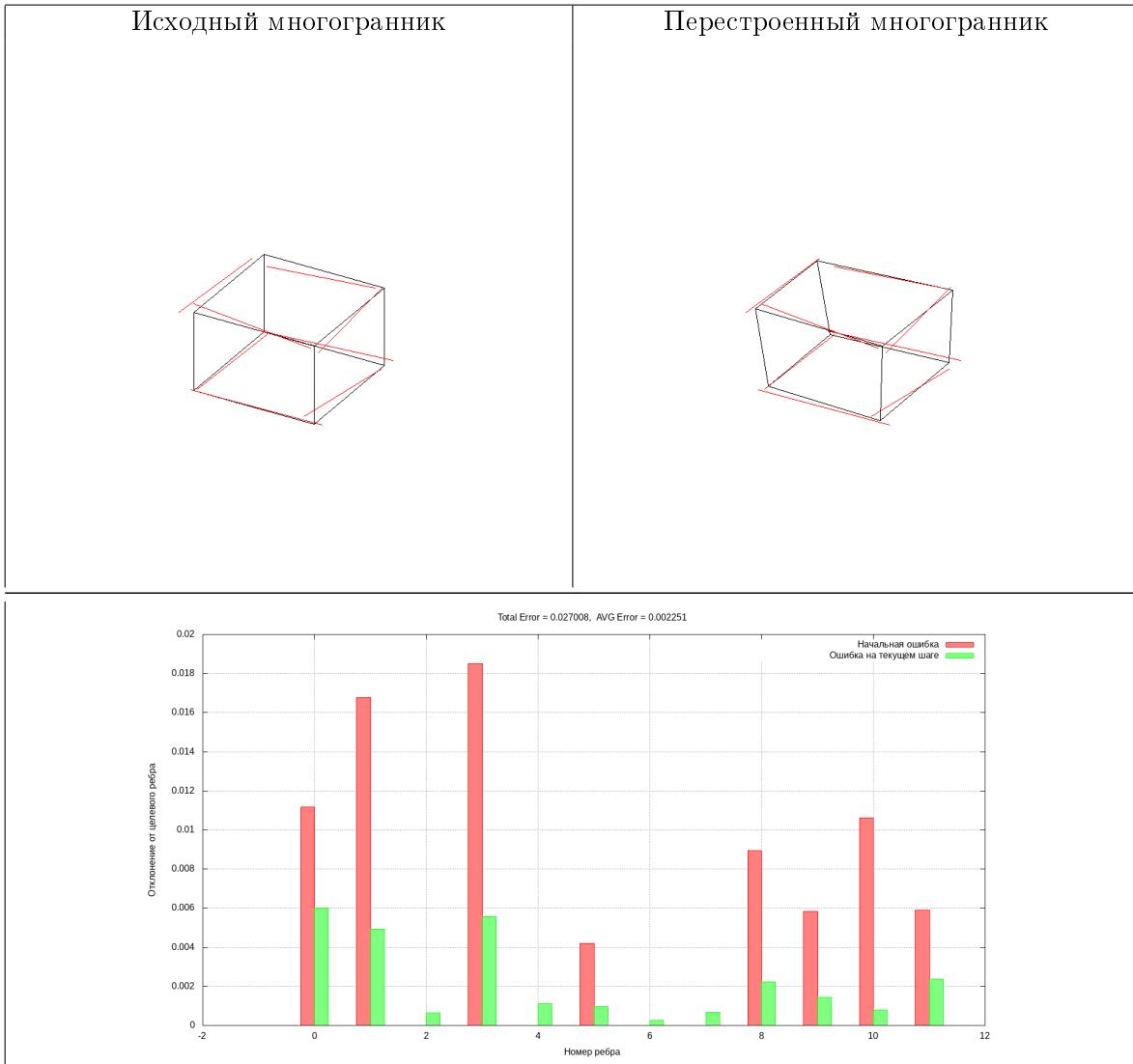
В программной реализации алгоритма использовался пакет оптимизации IPOPT - свободно распространяемая библиотека, включающая алгоритм внутренней точки. Про установку и работу с IPOPT можно прочитать в приложениях.

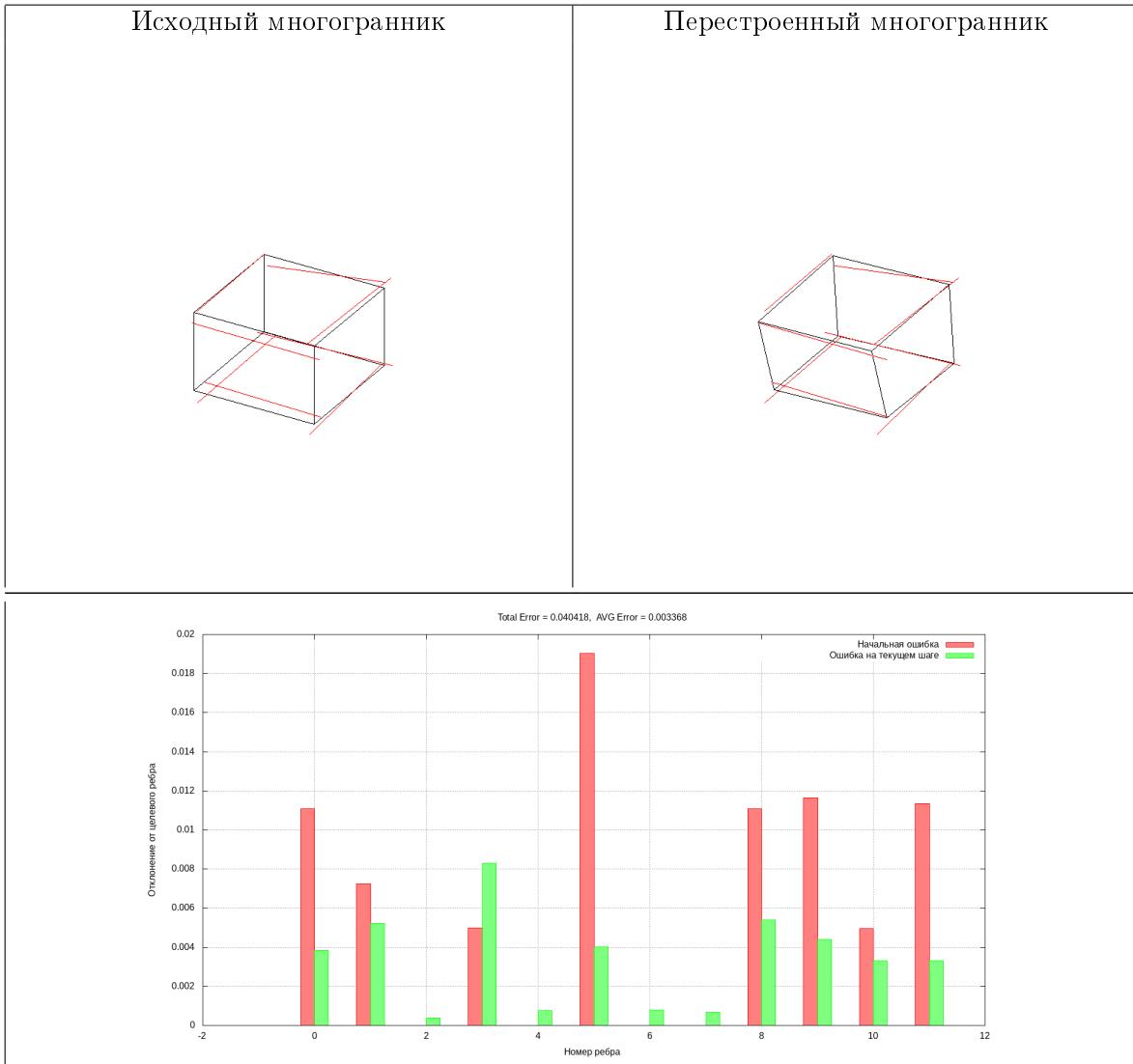
#### **5.1. Тестовая модель куба**

Оценим работоспособность алгоритма с помощью его тестирования на модельном примере – кубе. Зададим целевые ребра для верхней и нижней граней добавив к ним случайную ошибку.







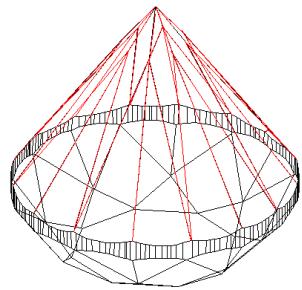


Исследуя полученные результаты можно заметить, что модель имеет право на жизнь, так как перестроенные многогранники не противоречат интуитивному понятию о "приближении" многогранника к набору ребер.

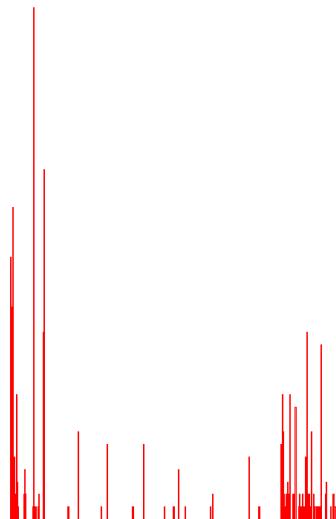
Теперь применим алгоритм к реальным моделям кристаллов.

## 5.2. Первый тестовый многогранник.

Перестроенный многогранник



Ошибка на ребрах



Значение функционала:  $3.9 * 10^{-4}$

Время работы: 405 секунд

Число итераций: 478

Число вершин: 442

Число ребер: 663

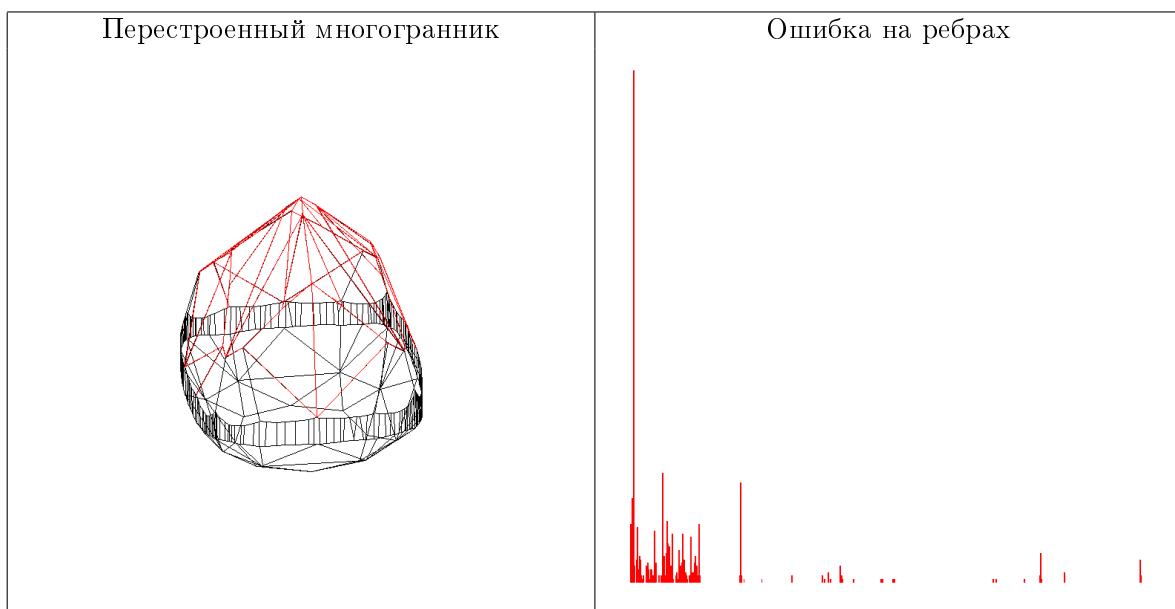
Число граней: 223

Число целевых ребер: 35

Число переменных: 4870

Число ненулевых элементов в Якобиане: 9951

### 5.3. Второй тестовый многогранник.



**Значение функционала:**  $7.6 * 10^{-4}$

**Время работы:** 635 секунд

**Число итераций:** 526

**Число вершин:** 536

**Число ребер:** 804

**Число граней:** 270

**Число целевых ребер:** 68

**Число переменных:** 5904

**Число ненулевых элементов в Якобиане:** 12066

### 5.4. Ускорение работы на больших моделях - локальная оптимизация.

К сожалению, на больших моделях алгоритм работает весьма продолжительное время. Однако из гистограмм ошибок можно заметить, что на более чем половине ребер удаление от целевых ребер-нулевое. Это ребра нижней части кристалла, для которых изначально нет целевых ребер и они притягиваются сами к себе. В результате этого они являются статичными и их движения не происходит, однако обсчет их движения выполняется. Будем теперь включать в функционал только ребра, лежащие в гранях, в которые входит хотя бы одно целевое ребро. Такая модель несколько сложнее в программной реализации, но она дает значительное ускорение. Так первый кристалл приближается за 10 секунд после 37 итераций, а второй-за 20 секунд после 41 итерации. При этом не наблюдается значительных отличий между

результатами работы глобального и локального алгоритмов.

Другим способом ускорения построения модели является явное вычисление якобиана. Первоначально использовалась симметрическая схема численного дифференцирования

$$\frac{\partial f}{\partial x_i} = \frac{f(\dots, x_i + h, \dots) - f(\dots, x_i - h, \dots)}{2h}$$

Эта схема - точна для полиномов второй степени, а, следовательно, позволяет точно вычислять якобиан для нашей задачи.

## 6. Сведение задачи к задаче безусловной оптимизации

### 6.1. Математическая формализация задачи. Построение минимизируемого функционала.

Будем теперь считать неизвестными уравнения плоскостей, в которых лежат грани кристалла. Для дальнейших рассуждений введем следующие обозначения:

$$\begin{cases} e_i = (p_i^0, p_i^1) - \text{ребра многогранника} \\ \tilde{e}_i = (\tilde{p}_i^0, \tilde{p}_i^1) - \text{соответствующие целевые ребра} \\ \Pi_i^0 : A_i^0 x + B_i^0 y + C_i^0 z + D_i^0 = 0 \\ \Pi_i^1 : A_i^1 x + B_i^1 y + C_i^1 z + D_i^1 = 0 \\ N_i^0 : a_i^0 x + b_i^0 y + c_i^0 z + d_i^0 = 0 : N_i^0 \perp e_i, p_i^0 \in N_i^0 \\ N_i^1 : a_i^1 x + b_i^1 y + c_i^1 z + d_i^1 = 0 : N_i^1 \perp e_i, p_i^1 \in N_i^1 \end{cases} \quad \Pi_i^0 \cap \Pi_i^1 = e_i$$

Из этих обозначений напрямую следует, что вершины ребер

$$p_i^0 = (x_i^0, y_i^0, z_i^0)$$

$$p_i^1 = (x_i^1, y_i^1, z_i^1)$$

являются соответственно решениями систем:

$$\begin{cases} A_i^0 x_i^0 + B_i^0 y_i^0 + C_i^0 z_i^0 + D_i^0 = 0 \\ A_i^1 x_i^0 + B_i^1 y_i^0 + C_i^1 z_i^0 + D_i^1 = 0 \\ a_i^0 x_i^0 + b_i^0 y_i^0 + c_i^0 z_i^0 + d_i^0 = 0 \end{cases} \quad (1)$$

$$\begin{cases} A_i^0 x_i^1 + B_i^0 y_i^1 + C_i^0 z_i^1 + D_i^0 = 0 \\ A_i^1 x_i^1 + B_i^1 y_i^1 + C_i^1 z_i^1 + D_i^1 = 0 \\ a_i^1 x_i^1 + b_i^1 y_i^1 + c_i^1 z_i^1 + d_i^1 = 0 \end{cases} \quad (2)$$

Откуда следуют зависимости:

$$x_i^0 = x_i^0(A_i^0, B_i^0, C_i^0, D_i^0, A_i^1, B_i^1, C_i^1, D_i^1)$$

$$\begin{aligned}
y_i^0 &= y_i^0(A_i^0, B_i^0, C_i^0, D_i^0, A_i^1, B_i^1, C_i^1, D_i^1) \\
z_i^0 &= z_i^0(A_i^0, B_i^0, C_i^0, D_i^0, A_i^1, B_i^1, C_i^1, D_i^1) \\
x_i^1 &= x_i^1(A_i^0, B_i^0, C_i^0, D_i^0, A_i^1, B_i^1, C_i^1, D_i^1) \\
y_i^1 &= y_i^1(A_i^0, B_i^0, C_i^0, D_i^0, A_i^1, B_i^1, C_i^1, D_i^1) \\
z_i^1 &= z_i^1(A_i^0, B_i^0, C_i^0, D_i^0, A_i^1, B_i^1, C_i^1, D_i^1)
\end{aligned}$$

Введем теперь функционал, который будем минимизировать:

$$\begin{aligned}
\Phi = \sum_{i=0}^n \rho^2(\tilde{p}_i^0, p_i^0(A_i^0, B_i^0, C_i^0, D_i^0, A_i^1, B_i^1, C_i^1, D_i^1)) + \\
\rho^2(\tilde{p}_i^1, p_i^1(A_i^0, B_i^0, C_i^0, D_i^0, A_i^1, B_i^1, C_i^1, D_i^1)) \rightarrow \min \quad (3)
\end{aligned}$$

$$\begin{aligned}
\Phi = \sum_{i=0}^n (\tilde{x}_i^0 - x_i^0)^2 + (\tilde{y}_i^0 - y_i^0)^2 + (\tilde{z}_i^0 - z_i^0)^2 + \\
(\tilde{x}_i^1 - x_i^1)^2 + (\tilde{y}_i^1 - y_i^1)^2 + (\tilde{z}_i^1 - z_i^1)^2 \rightarrow \min \quad (4)
\end{aligned}$$

Осуществлять поиск минимума будем методом градиентного спуска. Соответственно необходимо уметь считать градиент:  $\nabla \Phi = (\frac{\partial \Phi}{\partial A_0^0}, \dots, \frac{\partial \Phi}{\partial D_n^1})$

Для примера рассмотрим компоненту градиента, вычисленную по формуле производной сложной функции

$$\begin{aligned}
\frac{\partial \Phi}{\partial A_i^0} = -2(\tilde{x}_i^0 - x_i^0) \frac{\partial x_i^0}{\partial A_i^0} - 2(\tilde{y}_i^0 - y_i^0) \frac{\partial y_i^0}{\partial A_i^0} - 2(\tilde{z}_i^0 - z_i^0) \frac{\partial z_i^0}{\partial A_i^0} \\
- 2(\tilde{x}_i^1 - x_i^1) \frac{\partial x_i^1}{\partial A_i^0} - 2(\tilde{y}_i^1 - y_i^1) \frac{\partial y_i^1}{\partial A_i^0} - 2(\tilde{z}_i^1 - z_i^1) \frac{\partial z_i^1}{\partial A_i^0}
\end{aligned}$$

Остальные компоненты градиента имеют аналогичный вид. Отсюда следует, что для вычисления градиента  $\nabla \Phi$  достаточно знать значения координат вершин  $x_i^0, \dots, z_i^1$  и их градиенты  $\nabla x_i^0, \dots, \nabla z_i^1$ . Если для получения значений координат вершин достаточно решить системы 1 и 2, то для нахождения градиентов эти системы необходимо продифференцировать по  $A_i^0, \dots, D_i^1$ , а затем решить относительно частных производных. Так, если продифференцировать 1 по  $A_i^0$ , то получим:

$$\begin{cases} x_i^0 + A_i^0 \frac{\partial x_i^0}{\partial A_i^0} + B_i^0 \frac{\partial y_i^0}{\partial A_i^0} + C_i^0 \frac{\partial z_i^0}{\partial A_i^0} = 0 \\ A_i^1 \frac{\partial x_i^0}{\partial A_i^0} + B_i^1 \frac{\partial y_i^0}{\partial A_i^0} + C_i^1 \frac{\partial z_i^0}{\partial A_i^0} = 0 \\ a_i^0 \frac{\partial x_i^0}{\partial A_i^0} + b_i^0 \frac{\partial y_i^0}{\partial A_i^0} + c_i^0 \frac{\partial z_i^0}{\partial A_i^0} = 0 \end{cases} \Leftrightarrow \begin{cases} A_i^0 \frac{\partial x_i^0}{\partial A_i^0} + B_i^0 \frac{\partial y_i^0}{\partial A_i^0} + C_i^0 \frac{\partial z_i^0}{\partial A_i^0} = -x_i^0 \\ A_i^1 \frac{\partial x_i^0}{\partial A_i^0} + B_i^1 \frac{\partial y_i^0}{\partial A_i^0} + C_i^1 \frac{\partial z_i^0}{\partial A_i^0} = 0 \\ a_i^0 \frac{\partial x_i^0}{\partial A_i^0} + b_i^0 \frac{\partial y_i^0}{\partial A_i^0} + c_i^0 \frac{\partial z_i^0}{\partial A_i^0} = 0 \end{cases}$$

$$\begin{pmatrix} A_i^0 & B_i^0 & C_i^0 \\ A_i^1 & B_i^1 & C_i^1 \\ a_i^0 & b_i^0 & c_i^0 \end{pmatrix} \begin{pmatrix} \frac{\partial x_i^0}{\partial A_i^0} \\ \frac{\partial y_i^0}{\partial A_i^0} \\ \frac{\partial z_i^0}{\partial A_i^0} \end{pmatrix} = \begin{pmatrix} -x_i^0 \\ 0 \\ 0 \end{pmatrix}$$

Решив систему получим  $(\frac{\partial x_i^0}{\partial A_i^0}, \frac{\partial y_i^0}{\partial A_i^0}, \frac{\partial z_i^0}{\partial A_i^0})$ . Решив остальные системы получим все значения  $\nabla x_i^0, \dots, \nabla z_i^1$ , а следовательно и  $\nabla \Phi$ . Теперь можно применять градиентный спуск.

## 6.2. Второй вариант минимизируемого функционала

Рассмотрим также другой функционал расстояния:

$$\Psi = \sum_{i=0}^n \rho^2(\Pi_i^0, \tilde{p}_i^0) + \rho^2(\Pi_i^0, \tilde{p}_i^1) + \rho^2(\Pi_i^1, \tilde{p}_i^0) + \rho^2(\Pi_i^1, \tilde{p}_i^1)$$

, где  $\rho^2(\Pi, p) = \frac{(Ax+By+Cz+D)^2}{A^2+B^2+C^2}$  -квадрат расстояния от точки  $p$  до плоскости  $\Pi$ .

Такой функционал очевидно не выпуклый, однако если наложить ограничения  $A_i^2 + B_i^2 + C_i^2 = 1$ , то функционал приобретет вид:

$$\begin{aligned} \Psi = \sum_{i=0}^n & (A_i^0 \tilde{x}_i^0 + B_i^0 \tilde{y}_i^0 + C_i^0 \tilde{z}_i^0 + D_i^0)^2 + (A_i^0 \tilde{x}_i^1 + B_i^0 \tilde{y}_i^1 + C_i^0 \tilde{z}_i^1 + D_i^0)^2 + \\ & (A_i^1 \tilde{x}_i^0 + B_i^1 \tilde{y}_i^0 + C_i^1 \tilde{z}_i^0 + D_i^1)^2 + (A_i^1 \tilde{x}_i^1 + B_i^1 \tilde{y}_i^1 + C_i^1 \tilde{z}_i^1 + D_i^1)^2 \quad (5) \end{aligned}$$

Из вида такого функционала явно следует его выпуклость. Таким образом можно как и ранее использовать метод внутренней точки для решения новой задачи в ограничениях. С другой стороны, практика показывает, что для успешного применения градиентных методов для решения такой задачи достаточно на каждом шаге алгоритма предварительно нормировать коэффициенты плоскостей.

## 6.3. Выбор градиентного метода

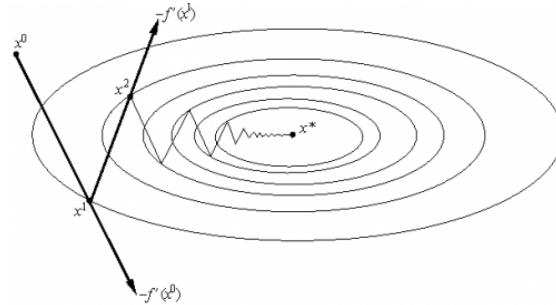


Рис. 17. Метод градиентного спуска с дроблением шага

К сожалению, при такой постановке задачи в обоих случаях(если не переходить во втором случае к задаче в ограничениях) минимизируемый функционал не является выпуклым. Это обуславливает ряд недостатков метода, а именно: результатом работы алгоритма будет локальный, а не глобальный минимум. Помимо этого возникает сложность с применением наискорейшего градиентного спуска, т.к. поиск

оптимального  $\lambda$  на каждом шаге алгоритма не возможен в силу того, что  $f(\lambda)$  - не является унимодальной функцией, а, следовательно, методы поиска минимума одномерной функции найдут только локальный минимум. Тем не менее неплохо себя показал градиентный спуск с измельчением шага. В этом варианте градиентного метода величина шага  $\lambda^k$  на каждой итерации выбирается из условия выполнения неравенства:

$$f(x^{k+1}) = f(x^k - \lambda^k \nabla f(x^k)) \leq f(x^k) - \varepsilon \lambda^k \|\nabla f(x^k)\|^2$$

Пока  $\lambda^k$  не станет удовлетворять этому неравенству выполняется процедура дробления шага:  $\lambda^k = \delta \lambda^k$ .

Оптимальными параметрами для данной задачи являются  $\lambda^0 = 1, \varepsilon = 0.1, \delta = 0.8$

Помимо рукописных градиентных методов я использовал библиотеку оптимизации NLOPT. Очень хорошие результаты были получены при применении квазиньютоновских алгоритмов BFGS и SLSQP. В частности, при использовании BFGS удалось обсчитывать большие модели менее чем за 1 секунду.

#### 6.4. Выбор стартовой точки оптимизации

Кажется логичным выбрать в качестве начальной точки коэффициенты уравнений плоскостей, задающих грани стартового многогранника. Однако существует и другой способ задания начальной точки - строить плоскость, наименее уклоняющуюся от вершин целевых ребер, находящихся в соответствующей грани.

**Определение:** Пусть имеется множество точек  $p_i = (x_i, y_i, z_i)_{i=1}^n$ , назовем плоскость  $\pi : Ax + By + Cz + D = 0$  наименее уклоняющейся от данного набора, если она доставляет минимум функционалу:

$$f(\pi) = \sum_{i=1}^n dist(\pi, p_i)^2 \rightarrow \min$$

Воспользовавшись формулой расстояния до плоскости получаем

$$f(A, B, C, D) = \sum_{i=1}^n \frac{(Ax_i + By_i + Cz_i + D)^2}{A^2 + B^2 + C^2} \rightarrow \min$$

Добавив условие нормировки плоскости  $A^2 + B^2 + C^2 - 1 = 0$  приходим к задаче на поиск условного минимума с функцией Лагранжа:

$$\mathcal{L}(A, B, C, D, \lambda) = \sum_{i=1}^n (Ax_i + By_i + Cz_i + D)^2 - \lambda(A^2 + B^2 + C^2 - 1)$$

Возьмем частные производные функции Лагранжа и приравняем их к нулю:

$$\frac{1}{2} \frac{\partial \mathcal{L}}{\partial A} = \sum_{i=1}^n x_i x_i A + \sum_{i=1}^n x_i y_i B + \sum_{i=1}^n x_i z_i C + \sum_{i=1}^n x_i D - \lambda A = 0$$

$$\begin{aligned}\frac{1}{2} \frac{\partial \mathcal{L}}{\partial B} &= \sum_{i=1}^n y_i x_i A + \sum_{i=1}^n y_i y_i B + \sum_{i=1}^n y_i z_i C + \sum_{i=1}^n y_i D - \lambda B = 0 \\ \frac{1}{2} \frac{\partial \mathcal{L}}{\partial C} &= \sum_{i=1}^n z_i x_i A + \sum_{i=1}^n z_i y_i B + \sum_{i=1}^n z_i z_i C + \sum_{i=1}^n z_i D - \lambda C = 0 \\ \frac{1}{2} \frac{\partial \mathcal{L}}{\partial D} &= \sum_{i=1}^n x_i A + \sum_{i=1}^n y_i B + \sum_{i=1}^n z_i C + \sum_{i=1}^n D = 0\end{aligned}$$

Помимо этого остаётся условие нормировки  $A^2 + B^2 + C^2 - 1 = 0$ .

Запишем уравнения в матричном виде:

$$\begin{pmatrix} \sum_{i=1}^n x_i x_i - \lambda & \sum_{i=1}^n x_i y_i & \sum_{i=1}^n x_i z_i & \sum_{i=1}^n x_i \\ \sum_{i=1}^n y_i x_i & \sum_{i=1}^n y_i y_i - \lambda & \sum_{i=1}^n y_i z_i & \sum_{i=1}^n y_i \\ \sum_{i=1}^n z_i x_i & \sum_{i=1}^n z_i y_i & \sum_{i=1}^n z_i z_i - \lambda & \sum_{i=1}^n z_i \\ \sum_{i=1}^n x_i & \sum_{i=1}^n y_i & \sum_{i=1}^n z_i & \sum_{i=1}^n 1 \end{pmatrix} \times \begin{pmatrix} A \\ B \\ C \\ D \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Вычтя из первых трех строк матрицы последнюю строку, умноженную на соответствующие коэффициенты мы приведем матрицу к виду:

$$\begin{pmatrix} \sum_{i=1}^n (x_i - x_M) x_i - \lambda & \sum_{i=1}^n (x_i - x_M) y_i & \sum_{i=1}^n (x_i - x_M) z_i & 0 \\ \sum_{i=1}^n (y_i - y_M) x_i & \sum_{i=1}^n (y_i - y_M) y_i - \lambda & \sum_{i=1}^n (y_i - y_M) z_i & 0 \\ \sum_{i=1}^n (z_i - z_M) x_i & \sum_{i=1}^n (z_i - z_M) y_i & \sum_{i=1}^n (z_i - z_M) z_i - \lambda & 0 \\ \sum_{i=1}^n x_i & \sum_{i=1}^n y_i & \sum_{i=1}^n z_i & \sum_{i=1}^n 1 \end{pmatrix} \times \begin{pmatrix} A \\ B \\ C \\ D \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

, где  $(x_M, y_M, z_M)$  – центр масс.

Для решения этой задачи найдём сначала единичные собственные вектора матрицы:

$$\begin{pmatrix} \sum_{i=1}^n (x_i - x_M) x_i - \lambda & \sum_{i=1}^n (x_i - x_M) y_i & \sum_{i=1}^n (x_i - x_M) z_i \\ \sum_{i=1}^n (y_i - y_M) x_i & \sum_{i=1}^n (y_i - y_M) y_i - \lambda & \sum_{i=1}^n (y_i - y_M) z_i \\ \sum_{i=1}^n (z_i - z_M) x_i & \sum_{i=1}^n (z_i - z_M) y_i & \sum_{i=1}^n (z_i - z_M) z_i - \lambda \end{pmatrix} \times \begin{pmatrix} A \\ B \\ C \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

А затем вычислим  $D$  из последнего линейного уравнения:

$$A_j \sum_{i=1}^n x_i + B_j \sum_{i=1}^n y_i + C_j \sum_{i=1}^n z_i + n D_j = 0 \Rightarrow D_j = -A_j x_M - B_j y_M - C_j z_M$$

для каждого нормированного собственного вектора  $(A_j, B_j, C_j)_{j=1}^3$   
 Четверка  $(A_j, B_j, C_j, D_j)_{j=1}^3$ , доставляющая минимум функционалу

$$f(A_j, B_j, C_j, D_j) = \sum_{i=1}^n (A_j x_i + B_j y_i + C_j z_i + D_j)^2 \rightarrow \min$$

и является решением задачи.

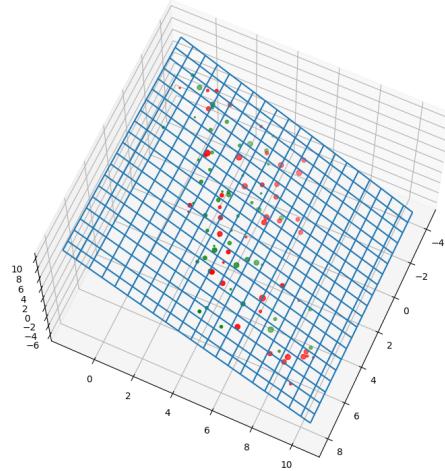


Рис. 18. Построенная наименее уклонающаяся плоскости

## 6.5. Восстановление многогранника

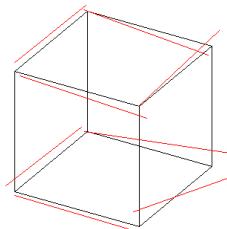
После решения задачи оптимизации мы имеем наборы плоскостей, задающих грани многогранника. Для построения измененного многогранника воспользуемся алгоритмом, описанном в 3 главе.

# 7. Результаты работы программной реализации алгоритма для задачи безусловной оптимизации

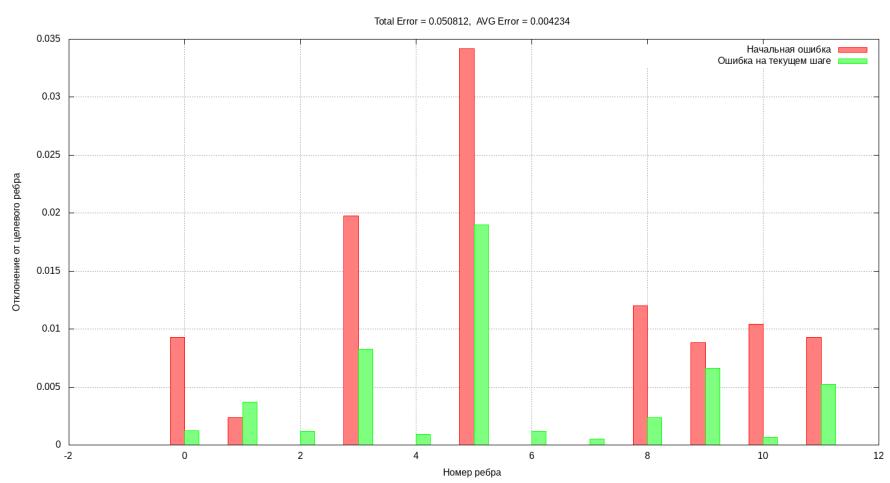
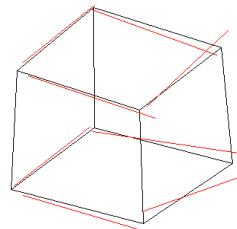
## 7.1. Тестовая модель куба

Как и для задачи условной оптимизации сначала проверим работоспособность алгоритма на тестовой модели куба. Как и ранее к вершинам ребер куба добавляем случайную ошибку и берем полученные таким образом ребра в качестве целевых. При реконструкции модели использовался второй функционал и алгоритм оптимизации BFGS (реализация из библиотеки NLOpt).

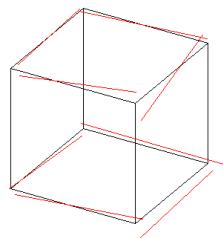
Исходный многогранник



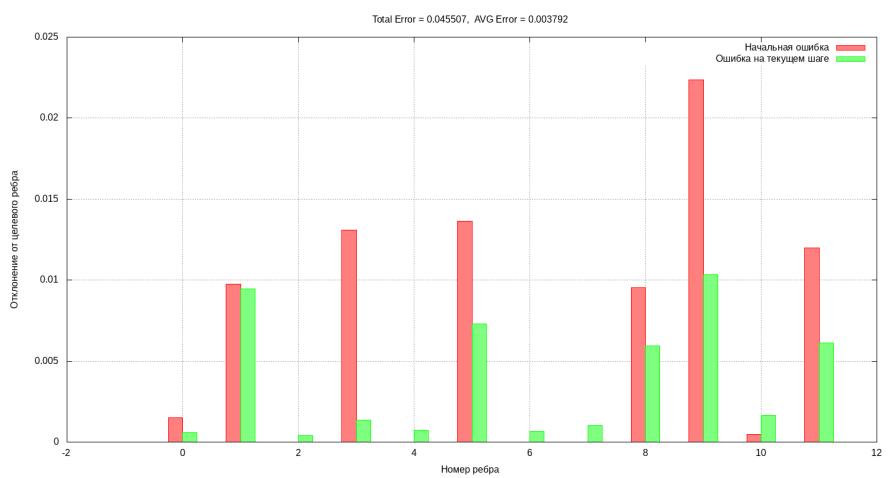
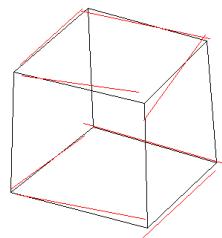
Перестроенный многогранник



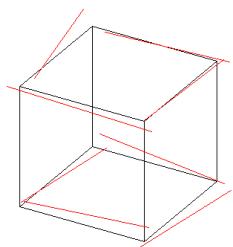
Исходный многогранник



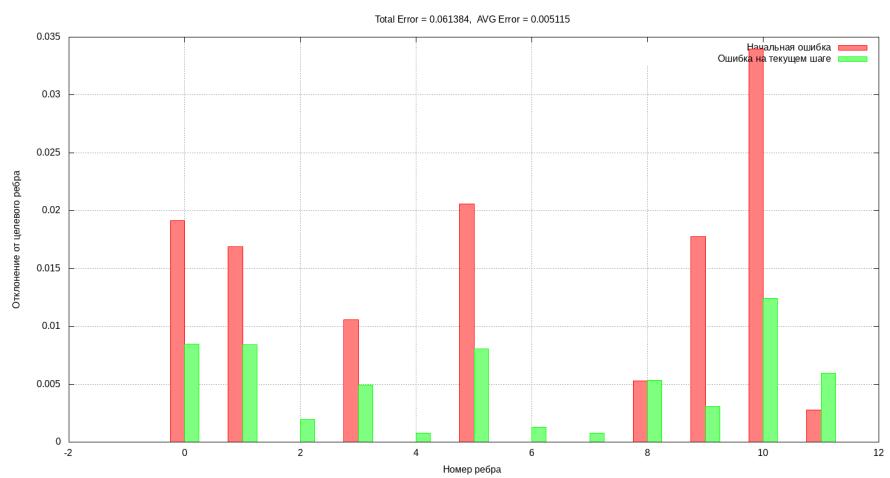
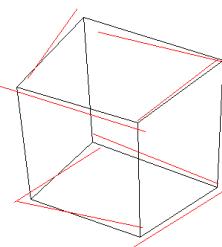
Перестроенный многогранник

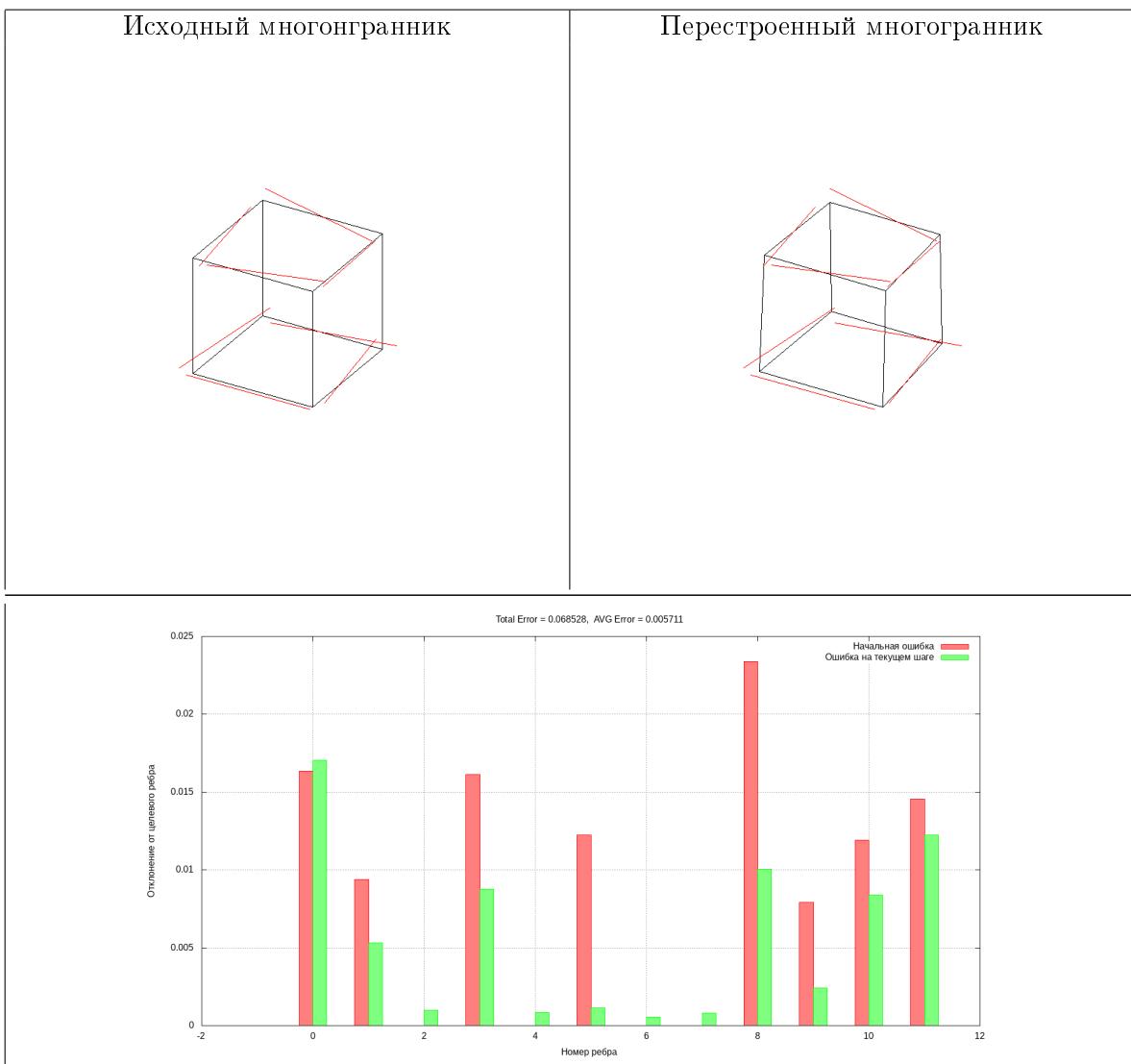


Исходный многогранник



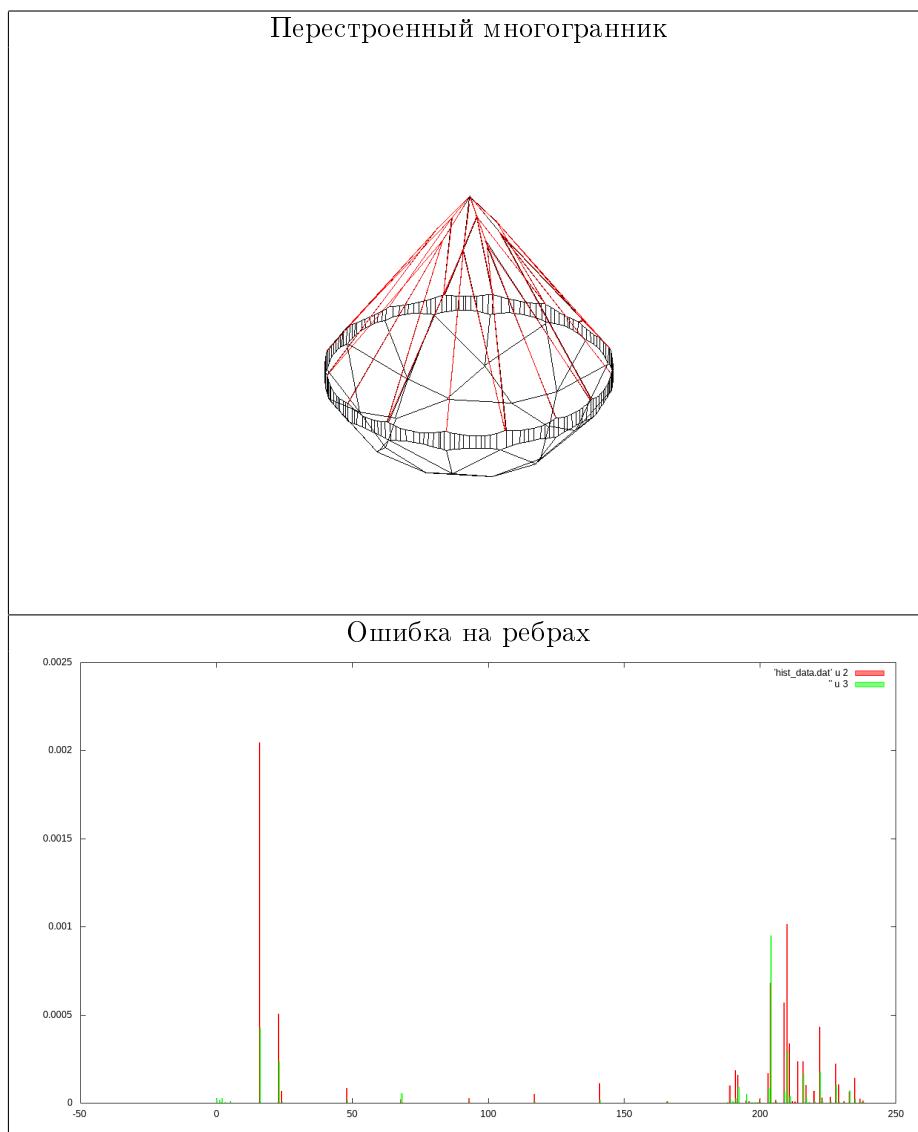
Перестроенный многогранник





Исходя из анализа результатов, полученных на тестовой модели можно сделать вывод о жизнеспособности алгоритма. Во-первых, полученные многогранники соответствуют представлению о том, как куб должен изогнуться. Во-вторых, из гистограмм ошибок видно, что среднее отклонение многогранника от целевых ребер - значительно уменьшилось.

## 7.2. Первый тестовый многогранник



Значение функционала:  $2.084788 \times 10^{-4}$

Время работы: 0.3 секунд

Число итераций: 115

Число вершин: 442

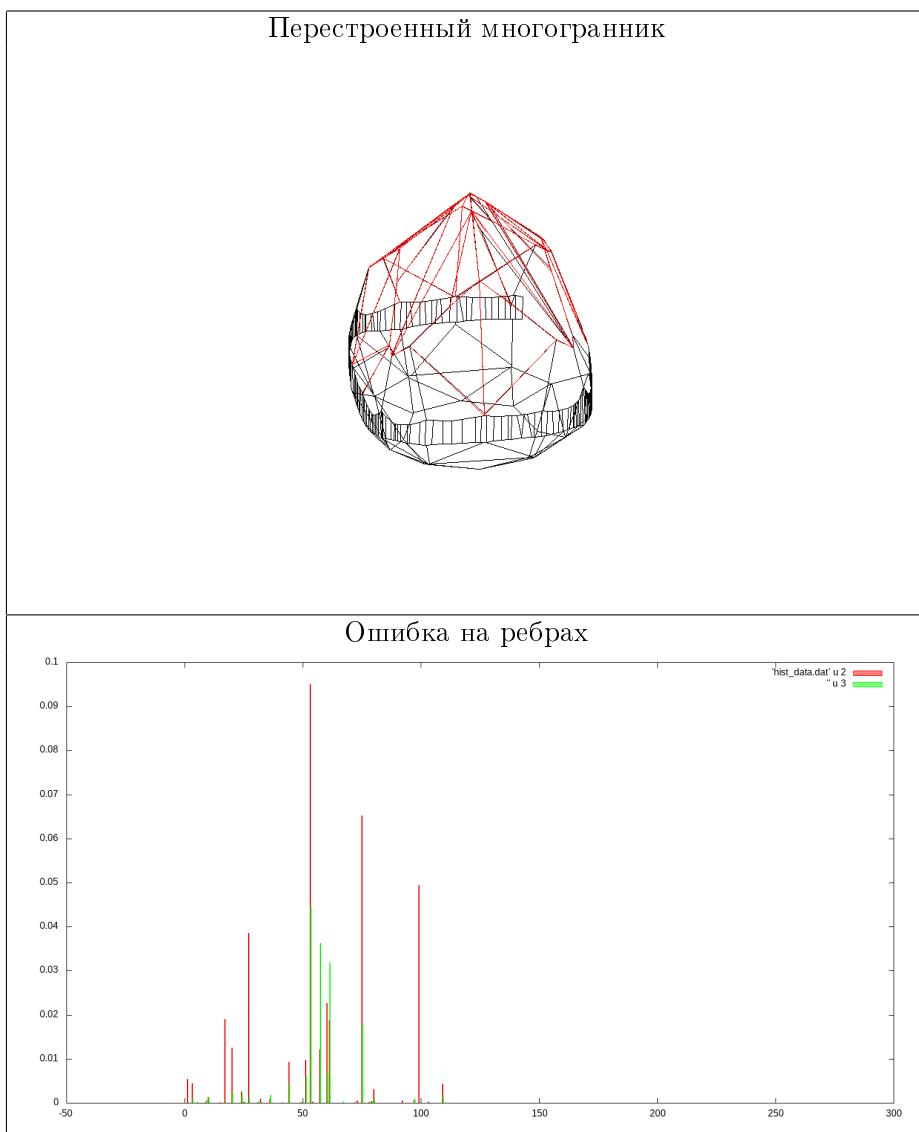
Число ребер: 663

Число граней: 223

Число целевых ребер: 35

Число переменных: 892

### 7.3. Второй тестовый многогранник



Значение функционала:  $2.530139 * 10^{-4}$

Время работы: 0.6 секунд

Число итераций: 76

Число вершин: 536

Число ребер: 804

Число граней: 270

Число целевых ребер: 68

Число переменных: 1080

Таким образом главная проблема задачи условной оптимизации, а именно весьма

значительное время работы алгоритма - решена. При этом нет потери точности по сравнению с первым алгоритмом.

## 8. Заключение

В ходе работы было проведено изучению задачи построение многогранника наименее уклоняющегося от набора целевых ребер и сравнение различных методов решения этой задачи. Основными результатами работы можно считать:

- 1) Разработан алгоритм построения целевого ребра по набору разноракурсных фотографий.
- 2) Описан алгоритм построения многогранника по набору полупространств - важная часть общего алгоритма реконструкции многогранника.
- 3) Были исследованы различные алгоритмы построения многогранника по набору целевых ребер. Даны разные способы математической формализации задачи, выбора целевого функционала, стартовой точки и алгоритма оптимизации.
- 4) Проведено тестирование этих алгоритмов как на модельном примере, так и на нескольких реальных кристаллах.
- 5) Сделан вывод об общей работоспособности описанных в работе методов. Разобраны различия в их быстродействии и точности.
- 6) Создан комплекс программного обеспечения, способный за приемлемое время провести реконструкцию многогранника.

## Список литературы

- [1] Веселов А. П., Троицкий Е. В. Лекции по аналитической геометрии. Учебное пособие. — Изд. новое. — М.: МЦНМО, 2016. — 152 с.
- [2] Алексеев В.М., Галеев Э.М., Тихомиров В.М. Сборник задач по оптимизации. Теория. Примеры. Задачи: Учеб. пособие. — 2-е изд. М.: ФИЗМАТЛИТ, 2005. — 256 с.
- [3] О.А. Щербина КРАТКОЕ ВВЕДЕНИЕ В AMPL - СОВРЕМЕННЫЙ АЛГЕБРАИЧЕСКИЙ ЯЗЫК МОДЕЛИРОВАНИЯ (препринт), 2012. — 29 с.
- [4] Попов А.В. GNUPLOT и его приложения. — М.: Издательство попечительского совета механико-математического факультета МГУ, 2015 , —240 с.
- [5] Introduction to Ipopt: A tutorial for downloading, installing, and using Ipopt. July 20, 2016
- [6] Dewi Retno Sari Saputro, Purnami Widyaningsih. Limited Memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) Method for The Parameter Estimation on Geographically Weighted Ordinal Logistic Regression Model (GWOLR): The 4th International Conference on Research, Implementation, and Education of Mathematics and Science, 04 August 2017
- [7] Пантелейев А.В., Летова Т.А. Методы оптимизации в примерах и задачах.
- [8] NLOpt Documentation: <https://nlopt.readthedocs.io/en/latest/>
- [9] Богачев К.Ю. Практикум на ЭВМ, Методы решения линейных систем и нахождения собственных значений, 1998
- [10] Jialiang Zhang, Lixiang Lin, Jianke Zhu Senior Member, Steven C.H. Hoi Fellow. Weakly-Supervised Multi-Face 3D Reconstruction
- [11] Guoxiang Zhang and YangQuan Chen. A METRIC FOR EVALUATING 3D RECONSTRUCTION AND MAPPING PERFORMANCE WITH NO GROUND TRUTHING
- [12] Генерация 3D-моделей по фотографиям. <https://habr.com/ru/post/64080/>
- [13] Исследование фотограмметрии. <https://habr.com/ru/post/339464/>
- [14] Udit Kumar, Sumit Soman, Jayavdeva. Benchmarking NLOpt and state-of-art algorithms for Continuous Global Optimization via Hybrid IACOR
- [15] Shivam Duggal, Zihao Wang, Wei-Chiu Ma, Sivabalan Manivasagam, Justin Liang, Shenlong Wang and Raquel Urtasun. Secrets of 3D Implicit Object Shape Reconstruction in the Wild

- [16] Xuepeng Shi, Qi Ye, Xiaozhi Chen, Chuangrong Chen, Zhixiang Chen, Tae-Kyun Kim. Geometry-based Distance Decomposition for Monocular 3D Object Detection
- [17] Pablo Palafox Aljaz Bozic Justus Thies Matthias Nießner Angela Dai. NPMs: Neural Parametric Models for 3D Deformable Shapes
- [18] Abdallah Dib, Gaurav Bharaj, Junghyun Ahn, Cédric Thébault, Philippe-Henri Gosselin, Marco Romeo, Louis Chevallier
- [19] Mona Zehni, Zhizhen Zhao. UVTOMO-GAN: AN ADVERSARIAL LEARNING BASED APPROACH FOR UNKNOWN VIEW X-RAY TOMOGRAPHIC RECONSTRUCTION

# Appendices

## A. Приложение 1: Установка IPOPT

Метод внутренней точки реализован в библиотеке IPOPT. Опишем процесс установки этой библиотеки.

- 1) Скачаем библиотеку из репозитория:

```
$ svn co https://projects.coin-or.org/svn/Ipopt/stable/3.12 CoinIpopt
```

- 2) Перейдем в папку с дистрибутивом IPOPT:

```
$ cd CoinIpopt
```

- 3) Скачаем сторонние библиотеки, необходимые для сборки IPOPT:

```
$ cd /ThirdParty/Blas  
$ ./get.Blas  
$ cd ..  
$cd Lapack  
$ ./get.Lapack  
$ cd ..  
$cd ASL  
$ ./get.ASL  
и т.д. для всех папок в ThirdPart
```

- 4) Создаём каталог build:

```
$ mkdir build
```

и переходим в него:

```
$ cd build
```

- 5) Запускаем скрипт configure:

```
$ ../configure
```

- 6) Собираем:

```
$ make
```

- 7) Запускаем короткий тест, чтобы проверить, что компиляция прошла успешно:

```
$ make test
```

8) Устанавливаем IPOPT:

```
$ make install
```

После этого в каталоге bin должен появиться исполняемый файл ipopt.

9) Переходим в корневую папку:

```
$ cd
```

И затем в bin:

```
$ cd bin
```

Сохраняем в эту папку файл ipopt.

## B. Приложение 2: C++ Интерфейс IPOPT

Основным классом, с которым предстоит работать является MyNLP, унаследованный от стандартного класса Ipopt::TNLP. В нем должны быть определены следующие методы класса:

- 1) `bool MyNLP::get_nlp_info(Index& n, Index& m, Index& nnz_jac_g,  
Index& nnz_h_lag, IndexStyleEnum& index_style)`

Задает общую информацию о задаче: число переменных, число ограничений, число ненулевых элементов матрицы Якоби для ограничений, число ненулевых элементов матрицы Гесса для функции Лагранжа.

- 2) `bool MyNLP::get_bounds_info(Index n, Number* x_l, Number* x_u,  
Index m, Number* g_l, Number* g_u)`

Задает ограничения на переменные и на условия. Если ограничений нет, то

```
x_l[i] = -1.0e19;  
x_u[i] = 1.0e19;
```

- 3) `bool MyNLP::get_starting_point(Index n, bool init_x, Number* x,  
bool init_z, Number* z_L, Number* z_U,  
Index m, bool init_lambda,  
Number* lambda)`

Задает стартовую точку алгоритма. Важно, чтобы она удовлетворяла всем условиям g. Алгоритм не просто так называется алгоритмом внутренней точки.

- 4) `bool MyNLP::eval_f(Index n, const Number* x, bool new_x, Number& obj_value)`  
Здесь нужно задать минимализируемый функционал.

- 5) `bool MyNLP::eval_grad_f(Index n, const Number* x, bool new_x, Number* grad_f)`  
Посчитать градиент от функционала.

- 6) `bool MyNLP::eval_g(Index n, const Number* x, bool new_x, Index m, Number* g)`  
В этой функции мы задаем ограничения на функционал.

- 7) `bool MyNLP::eval_jac_g(Index n, const Number* x, bool new_x,  
Index m, Index nele_jac, Index* iRow, Index *jCol,  
Number* values)`

Здесь задается структура ненулевых элементов матрицы Якоби.

```
iRow[i]=rowNumber;  
jCol[i] = colNumber;
```

А затем значение этого элемента

- ```
values[i]=value;
```
- 8) bool MyNLP::eval\_h(Index n, const Number\* x, bool new\_x,  
Number obj\_factor, Index m, const Number\* lambda,  
bool new\_lambda, Index nele\_hess, Index\* iRow,  
Index\* jCol, Number\* values)
- Вычисление Гессиана от Лагранжиана функционала. Можно пропустить, написав специальную команду в опциях компиляции.
- 9) void MyNLP::finalize\_solution(SolverReturn status,  
Index n, const Number\* x, const Number\* z\_L, const Number\* z\_U,  
Index m, const Number\* g, const Number\* lambda,  
Number obj\_value,  
const IpoptData\* ip\_data,  
IpoptCalculatedQuantities\* ip\_cq)
- То, что будет выполнено по завершению поиска оптимума. Здесь я запускаю построение многогранника по набору полупространств и его последующую отрисовку.