

План дипломной работы  
Коррекция многогранников с помощью локальных  
деформаций

Палачев Илья, 510 группа

28 марта 2012 г.

# Содержание

<b>1</b>	<b>Введение</b>	<b>2</b>
1.1	Происхождение вопроса . . . . .	2
1.2	Особенности процесса сканирования алмазов . . . . .	3
1.3	Что понимается под многогранником . . . . .	5
<b>2</b>	<b>Сглаживание смежных граней</b>	<b>6</b>
2.1	Постановка задачи . . . . .	6
2.2	Описание алгоритма . . . . .	7
2.3	Примеры . . . . .	7
2.4	Построение контура . . . . .	8
2.5	Построение средней плоскости . . . . .	9
2.5.1	TODO: Построение средней плоскости с весами (для учета ха- рактерных вершин) . . . . .	10
2.6	Поднятие точек, лежащих ниже плоскости. . . . .	10
2.6.1	Основная функция . . . . .	10
2.6.2	Нахождение поднимаемой точки . . . . .	10
2.6.3	Нахождение перемещения точки . . . . .	11
2.6.4	Перемещение точки . . . . .	13
2.7	Рассечение модифицированного многогранника плоскостью . . . . .	14
2.8	Тестирование алгоритма . . . . .	15
2.8.1	Тестирование на многограннике polyhedron-2010-11-25 . . . . .	15
2.8.2	TODO: Тестирование на других примерах + случай трех-четырёх граней . . . . .	17
<b>3</b>	<b>Сечение многогранника плоскостью</b>	<b>17</b>
3.1	Постановка задачи . . . . .	17
3.2	Описание алгоритма . . . . .	17
3.2.1	Этапы алгоритма . . . . .	17
3.2.2	Предобработка многогранника . . . . .	17
3.2.3	Создание списков текущих ребер для всех граней . . . . .	17
3.2.4	Создание компонент сечения . . . . .	23
3.2.5	Рассечение старых граней плоскостью . . . . .	23
3.2.6	Создание граней и вершин искомого многогранника . . . . .	25
3.3	Тестирование алгоритма . . . . .	25
3.3.1	Тестирование на модельном многограннике cube-cutted . . . . .	26
3.3.2	Тестирование на многограннике poly-big . . . . .	27
3.4	TODO: Модификация алгоритма для задачи слияния смежных граней	30
<b>4</b>	<b>Подвижка вершины и дальнейшая перестройка многогранника</b>	<b>30</b>
4.1	Идея задачи . . . . .	30
4.2	Формализация задачи . . . . .	31

4.2.1	Первый вариант: варьирование вершин . . . . .	31
4.2.2	Второй вариант: варьирование вершин и граней . . . . .	33
4.2.3	Третий вариант: поочередное варьирование вершин и граней . .	35
4.3	Тестирование алгоритма . . . . .	36
4.3.1	Экспоненциальный рост штрафа, локальная минимизация . . .	36
4.3.2	Масштабирование штрафа . . . . .	39
4.3.3	Сбор статистики о выворачивании . . . . .	40
4.3.4	TODO: Регистрирование выворачиваний . . . . .	42
4.3.5	TODO: Устранение выворачиваний путем слияния вершин . . .	42
4.4	TODO: Анализ сходимости и трудоемкости алгоритма . . . . .	42

# 1 Введение

## 1.1 Происхождение вопроса

Данная работа возникла из вопроса автоматизации производственного процесса огранки драгоценных камней.

Огранка подразумевает создание на поверхности камня ряда геометрически правильных плоскостей, граней, от которых будет отражаться попадающий внутрь кристалла свет. После многократного внутреннего отражения и преломления лучи света разделяются на спектральные составляющие и, покидая камень, создают игру оттенков на его поверхности. Огранка различается по сложности выполнения. В наиболее сложной и привлекательной бриллиантовой огранке может быть до 240 граней. Простая огранка создает на поверхности камня 30 плоскостей. Качественно выполненная огранка существенно повышает стоимость драгоценного камня, в то же время неправильная огранка может «убить» камень, внести в него дефекты. [из Wiki]

В конечном счете красота бриллианта определяется оптическими свойствами, как присущими веществу - алмазу, так и проявляющимися в результате огранки и полировки. Совокупность поверхностных и внутренних оптических свойств бриллианта человеческий глаз воспринимает как попавшие в глаз разнообразные лучи — белые и цветные, в целом создающие картину бриллианта. [из <http://3dbook.octonus.com/data/3dbook.html>]

Форма огранки бриллианта зависит от формы исходного кристалла алмаза. Чтобы получить бриллиант максимальной стоимости, огранщики стараются свести к минимуму потери алмаза при обработке. В зависимости от формы кристалла алмаза, при его обработке теряется 55—70 % веса. [из Wiki] Возможно ли автоматизировать этот процесс? Если да, то насколько? Разница всего в 1/10 карата может менять стоимость бриллианта в разы.

## 1.2 Особенности процесса сканирования алмазов

Чтобы автоматизировать процесс огранки, необходимо реализовать построение трехмерных моделей реальных алмазов. Для их создания используются сканеры, которые фотографируют камень. Как это происходит? Камень помещают перед источником света и фотографируют. При этом получается так называемый **тенево́й профиль** камня. Затем камень немного поворачивают и фотографируют еще раз. Так делается много раз и в результате на основе фотографий теневого профиля вращающегося алмаза строится 3D-модель. Опишем этот процесс подробнее.  
[<http://www.octonus.com/oct/support/doc/scanning.html>]

Сначала камень устанавливается на подставке - **держателе**. Эта подставка вращается и периодически производится фотографирование теневого профиля с определенным угловым шагом. Например, в сканере алмазов "Helium Polish" фотографирование может производиться в трех режимах  
[[http://www.octonus.com/oct/download/files/Helium\\_system\\_manual.pdf](http://www.octonus.com/oct/download/files/Helium_system_manual.pdf)]:

Метод	Число контуров	Время сканирования, с	Время сборки, с	Полное время, с
Quick	100	12	3	15
Accuracy	400	47	11	58
Hi accuracy	800	92	30	122

Из этой таблицы видно, что в разных режимах делается от 100 до 800 фотографий одного камня, то есть модель строится по 100, 400 или 800 теневым контурам. Может ли оказаться так, что два различных камня могут быть восприняты системой как одинаковые? Конечно, это возможно, если разница между ними так мала, что все теневые контуры у них будут идентичны. Всякую модель подобного рода можно понимать как пространственное пересечение нескольких бесконечных цилиндров, порожденных теневыми контурами.

Может возникнуть коллизия такого рода, когда сам камень имеет какую-то простую структуру, но система при фотографировании заменяет его на более сложный многогранник. Например, модель может иметь такой вид, как на рис. 1, тогда как в самом деле камень имеет вид, как на рис. 2. Это происходит за счет того, что угол  $\alpha$  между осями проекции очень мал (порядка шага фотографирования).

В таких ситуациях происходит заведомо неоправданное усложнение структуры многогранника. В 3D-моделях камней происходит разделение крупных граней на 2, 3, 4 более мелких. Это происходит за счет добавления в модель несуществующих в камне ребер. Как это может повлиять на дальнейшую работу с моделью? Если идет

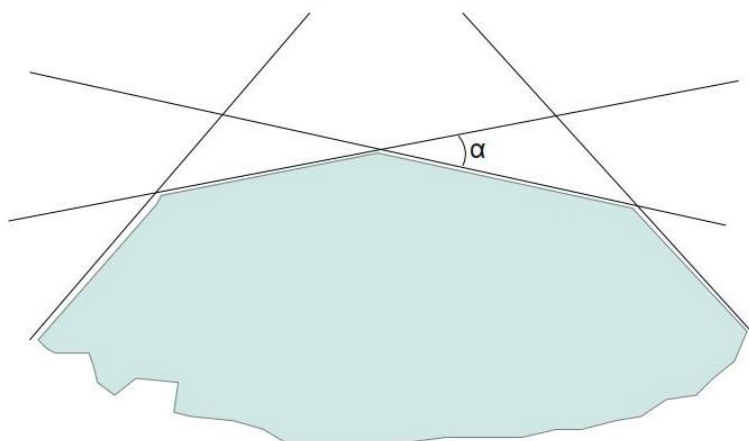


Рис. 1: Смоделированный многогранник

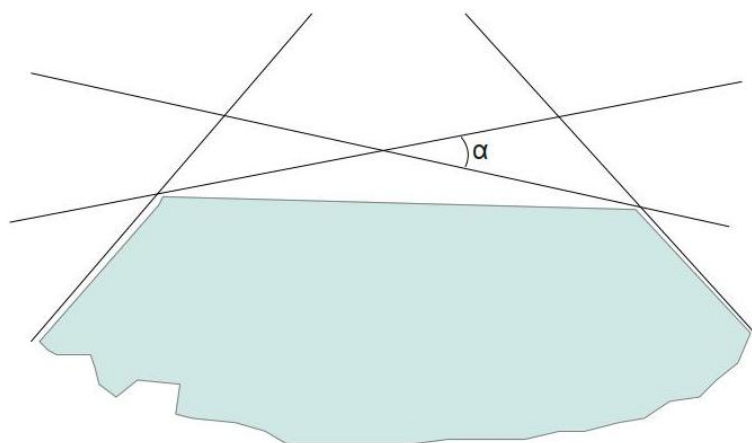


Рис. 2: Реальный камень

работа с одним камнем, то разница незначительна, но при массовом анализе увеличение числа ребер в 2, 3 раза приводит к катастрофическому увеличению времени работы алгоритмов. Кроме того, лишние ребра препятствуют анализу характеристик камня по его модели. Так, если нужно будет найти площадь самой большой грани, то когда в модели эта грань будет разделена на 2, то характеристика будет вычислена неверно.

Эти соображения и послужили поводом написания данной работы. В ней рассмотрены различные способы коррекции многогранников при помощи локальных деформаций, без повторного анализа теневых контуров, то есть по уже построенной 3D-модели камня.

### 1.3 Что понимается под многогранником

Прежде всего следует сказать о том, каким образом задаются многогранники. Во-первых, задается количество вершин и количество граней. Во-вторых, перечисляются координаты всех вершин (это тройки чисел  $x, y, z$ ). В-третьих, перечисляются грани. При этом указывается, сколько вершин находится в грани, в какой плоскости она лежит (четырьмя числами  $a, b, c, d$  из уравнения плоскости  $ax + by + cz + d = 0$ ), и список вершин, которые ее составляют.

Например, куб с центром в начале координат  $(0, 0, 0)$  со сторонами длины 2, параллельными осям координат (см. рис. 3), задается в данной системе следующим образом:

Число вершин	Число граней
8	6

Номер вершины	$x$	$y$	$z$
0	-1	-1	-1
1	1	-1	-1
2	1	1	-1
3	-1	1	-1
4	-1	-1	1
5	1	-1	1
6	1	1	1
7	-1	1	1

Номер грани	Число вершин в грани	$a$	$b$	$c$	$d$	Список вершин
0	4	0	0	-1	-1	1, 0, 3, 2
1	4	0	-1	0	-1	0, 1, 5, 4
2	4	1	0	0	-1	1, 2, 6, 5
3	4	0	1	0	-1	2, 3, 7, 6
4	4	-1	0	0	-1	3, 0, 4, 7
5	4	0	0	1	-1	4, 5, 6, 7

Таким же образом задаются и прочие многогранники. В рассмотренных примерах моделей реальных камней количество вершин варьировалось от 80 до 8000. Таким образом, во всех рассматриваемых алгоритмах встает вопрос о трудоемкости, то есть о времени работы.

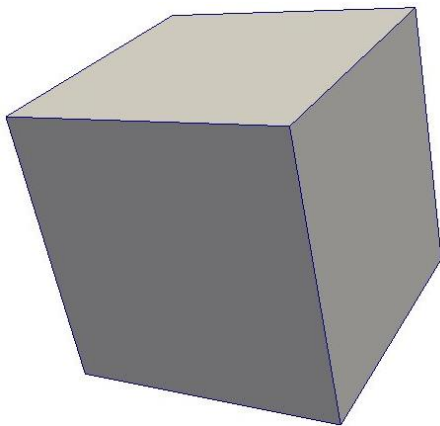


Рис. 3: Куб

## 2 Сглаживание смежных граней

### 2.1 Постановка задачи

Собственно, проблема была сформулирована еще во введении. Имеются две или более граней, которые находятся в многограннике рядом с друг другом, и при этом плоскости, в которых они лежат почти совпадают, то есть угол между их нормальями очень мал. Под словами "грани находятся друг рядом с другом" следует понимать, что на поверхности многогранника они занимают некоторую односвязную область. При этом нужно, чтобы существовал односвязный контур, который бы "охватывал" все рассматриваемые грани.

**Требуется:** вместо данной группы грани построить одну грань, которая бы состояла из точек такого охватывающего контура. Этой гранью нужно заменить исходную группу граней.

Иными словами, требуется удалить лишние ребра из многогранника, которые усложняют его структуру, разделяя одну большую грань на несколько мелких.

Можно подумать, что достаточно всего лишь построить такой охватывающий контур и на этом остановиться. Но грань, которая при этом получится, не будет гранью: не будет гарантии, что все ее точки будут лежать в одной плоскости. Что мы понимаем под тем, что "точка лежит в плоскости некоторой грани"? Из математических соображений ясно, что если  $(x_i, y_i, z_i)$  – координаты  $i$ -й вершины, а  $a_k x + b_k y + c_k z + d_k = 0$  – уравнение плоскости  $k$ -й грани, то утверждение " $i$ -я точка лежит в  $k$ -й грани" можно выразить равенством

$$a_k x_i + b_k y_i + c_k z_i + d_k = 0$$

В нашей же теории подразумевается, что данное равенство выполнено с некоторой точностью  $\varepsilon$ , то есть

$$|a_k x_i + b_k y_i + c_k z_i + d_k| < \varepsilon$$

При этом как правило  $\varepsilon = 10^{-10}$ .

Следовательно, нужно переформулировать задачу. **Требуется:** перестроить исходный многогранник таким образом, чтобы исходная группа граней объединилась в одну грань. При этом возможно, что многогранник будет немного изменен по своей структуре: некоторые вершины подвинуты, некоторые новые ребра могут быть добавлены. Желательно, чтобы изменение в структуре было минимальным.

## 2.2 Описание алгоритма

Был реализован следующий алгоритм:

**1. Построение контура.** По данной группе граней строится охватывающий контур, состоящий из точек исходных граней.

**2. Построение средней плоскости.** По точкам полученного контура строится методом наименьших квадратов плоскость, которая лучше всего (в смысле равенства  $a_k x + b_k y + c_k z + d_k = 0$ ) обеспечивает попадание исходных точек в плоскость.

Что можно сказать про построенную плоскость? Часть точек будет лежать выше нее, часть ниже, а часть довольно хорошо (с точностью  $10^{-16}$ ) попадет на нее. Если бы не было точек, которые лежали ниже плоскости, то мы бы просто рассекли многогранник полученной плоскостью и получили решение задачи. Но те точки, которые лежат ниже плоскости, не позволяют нам этого сделать.

**3. Поднятие точек, лежащих ниже плоскости.** Путем перестройки структуры многогранника мы добиваемся, чтобы **все** точки контура оказались выше построенной плоскости. Как это делается, описано в соответствующем пункте.

**4. Рассечение многогранника плоскостью.** После этого ничто не мешает нам просто рассечь многогранник построенной плоскостью наименьших квадратов.

## 2.3 Примеры



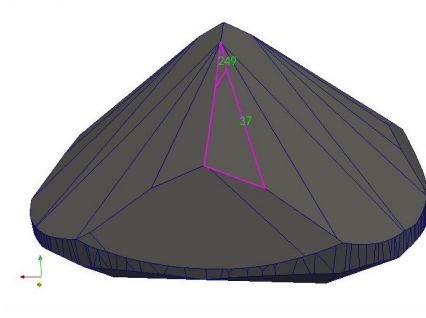


Рис. 4: Пример двух гладко стыкующихся граней

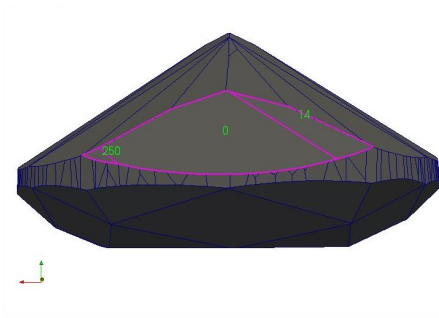


Рис. 5: Пример трех гладко стыкующихся граней

## 2.4 Построение контура

Пусть требуется построить контур по граням с номерами  $f_0, f_1, \dots, f_{n-1}$ . Для построения контура используется следующий алгоритм:

1. Заполняем массив  $NF$  следующим образом:  $NF[i]$  = количеству вхождений  $i$ -й вершины в какие-либо грани группы  $f_0, f_1, \dots, f_{n-1}$ . Для тех вершин, которые вообще не связаны с гранями данной группы,  $NF[i] = 0$ . Нам же интересны в основном те вершины, для которых  $NF[i] = 1$ .
2. Пусть  $v_0$  - такая вершина в грани  $f_0$ , для которой  $NF[v_0] = 1$ .

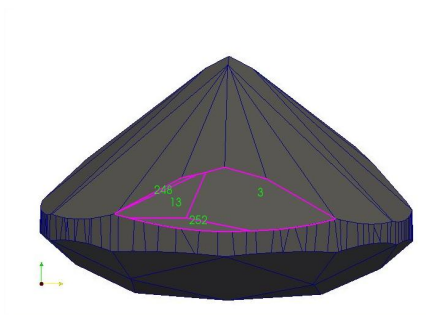


Рис. 6: Пример четырех гладко стыкующихся граней

3.  $i = 0, v = v_0, nv = 0$

4. Производить следующий цикл:

4. 1. Производить следующий цикл пока  $NF[v] = 1$ :

4. 1. 1. Добавить вершину  $v$  в контур.

4. 1. 2.  $nv = nv + 1$

4. 1. 3.  $v$  =следующей за  $v$  вершине (против часовой стрелки) в грани  $f_i$ .

4. 2. Добавить вершину  $v$  в контур.

4. 3.  $nv = nv + 1$

4. 4. Если  $v = v_0$ , то прервать цикл 4.

4. 5. Выполнять цикл для всех  $i = 0, 1, \dots, n - 1$ :

4. 5. 1.  $v_1$  =следующей за  $v$  вершине (против часовой стрелки) в грани  $f_i$

4. 5. 2. Если  $NF[v_1] = 1$ , то прервать цикл 4.5 (т.е. этот цикл позволяет найти номер  $i$  следующей обрабатываемой нами грани).

4. 6.  $v = v_1$

4. 7. Если  $v = v_0$ , то прервать цикл 4.

5. Положить  $facet[0]$  =грани, состоящей из построенного контура.

6. Положить  $facet[1] = facet[2] = \dots = facet[n - 1]$  =пустой грани.

7. Осуществить ПРЕДОБРАБОТКУ многогранника.

8. Удалить все висячие вершины. **Висячей** называется такая вершина, которая лежит всего в двух гранях. Такое происходит, когда две грани имеют два подряд идущих общих ребра.

## 2.5 Построение средней плоскости

Для построения уравнения новой плоскости используется уже готовый алгоритм наименьших квадратов. Построенный контур интерпретируется просто как множество точек и коэффициенты  $a, b, c$  и  $d$  уравнения  $ax + by + cz = 0$  подбираются как решение экстремальной задачи

$$\sum_{i: A_i \in \pi} |ax_i + by_i + cz_i + d|^2 \rightarrow \min$$

Иными словами, минимизируется сумма квадратов расстояний от точек контура до плоскости.

### 2.5.1 TODO: Построение средней плоскости с весами (для учета характерных вершин)

Имеется в виду, что масса грани должна быть равномерно распределена. В обычном подходе этому мешают скопления точек, которые вынуждают плоскость пройти близко к ним, увеличивая погрешность на изолированных точках. Для устранения этого нужно придать изолированным точкам больший вес. делать это можно несколькими способами.

1. Можно считать вес по длинам ребер, исходящих из точки.
2. Можно считать вес по площади (какой?)
3. Может быть, следует обратиться к осям инерции (?).

## 2.6 Поднятие точек, лежащих ниже плоскости.

Этот этап составляет основную содержательную часть алгоритма. Опишем его основные функции.

### 2.6.1 Основная функция

1.  $n_{down}$  = числу вершин, лежащих ниже плоскости
2. Пока ( $n_{down} > 1$ ) производить следующий цикл:
  2. 1.  $i_{min}$  = НАЙТИ\_ПОДНИМАЕМУЮ\_ТОЧКУ
  2. 2. ПОДНЯТЬ\_ТОЧКУ  $i_{min}$
  2. 3.  $n_{down}$  = числу вершин, лежащих ниже плоскости
  2. 4. Произвести ПРЕДОБРАБОТКУ многогранника

### 2.6.2 Нахождение поднимаемой точки

1.  $i_{min} = -1$
2. Цикл (по всем точкам контура  $A_{i_0}, A_{i_1}, \dots, A_{i_{m-1}}$ , то есть  $i = i_0, i_1, \dots, i_{m-1}$ )
  2. 1. Если  $A_i$  лежит выше плоскости или на плоскости, то пропустить шаг цикла.
  2. 2.  $d$  = НАЙТИ\_ПЕРЕМЕЩЕНИЕ\_ТОЧКИ  $A_i$
  2. 3. Если ( $d_{min} > d$  или  $i_{min} = -1$ ) и ( $d_{min} > 10^{-15}$ ), то  $d_{min} := d$ ,  $i_{min} := i$

### 2.6.3 Нахождение перемещения точки

Пусть находится перемещение точки  $A_i$ . Введем сначала некоторые обозначения. Пусть точка расположена так, как показано на рис. 7. Точнее говоря, пусть  $f$  - главная грань, то есть грань построенного контура,  $\pi$  - плоскость главной грани  $f_{-2}, f_{-1}, f_{+1}, f_{+2}$  - соседние с гранью  $f$  грани,  $\pi_{-2}, \pi_{-1}, \pi_{+1}, \pi_{+2}$  - плоскости данных граней. При этом  $\pi \cap \pi_{-1} \cap \pi_{+1} = A_i$ ,  $\pi \cap \pi_{-2} \cap \pi_{-1} = A_{i-1}$ ,  $\pi \cap \pi_{+1} \cap \pi_{+2} = A_{i+1}$ , где  $A_{i-1}$  - предыдущая перед точкой  $A_i$  в контуре,  $A_{i+1}$  - следующая перед точкой  $A_i$  в контуре (контур организован так, что точки хранятся в нем в направлении обхода против часовой стрелки, если смотреть извне многогранника).

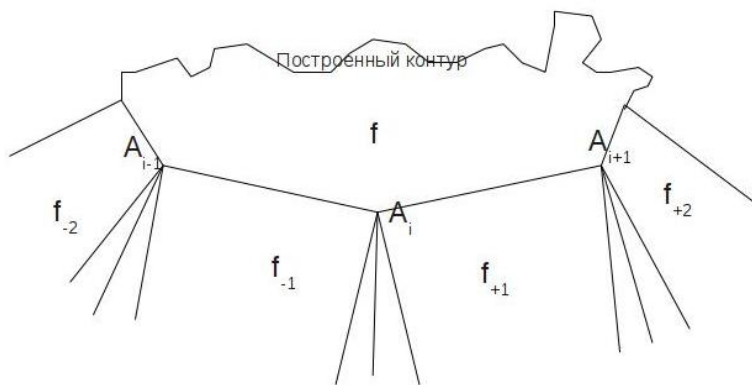


Рис. 7: Расположение точек и граней

Перед тем, как давать описание алгоритма, поясним, что значит **перемещение точки**, чтобы сделать ясной идею данного подхода. Для простоты предположим, что степени точек  $A_i$ ,  $A_{i-1}$  и  $A_{i+1}$  равны 3, то есть эти точки лежат всего в трех гранях многогранника. Это изображено на рис. 8

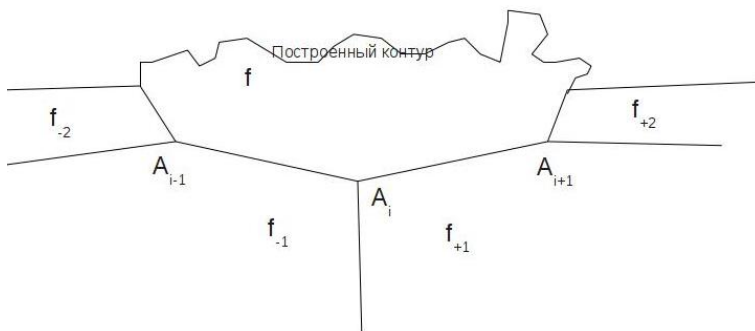


Рис. 8: Расположение точек и граней (упрощенное)

В упрощенном случае грани  $f_{-1}$  и  $f_{+1}$  пересекаются по ребру, одной из вершин которого является точка  $A_i$ . Попробуем *продолжить* в сторону точки  $A_i$ , то есть сдвинем точку  $A_i$  по этому ребру. Такой сдвиг позволяет приближать точку, лежащую ниже плоскости, в более высокое положение. Производя такие сдвиги последовательно, мы надеемся поднять все точки, которые лежат ниже плоскости.

Насколько далеко можно произвести такой сдвиг? Ясно, что точки  $A_{i-1}$  и  $A_{i+1}$  тоже можно сдвигать, поэтому нужно продумать алгоритм так, чтобы соответствующие ребра не пересеклись.

Поступим так: будем сдвигать точку  $A_i$  до тех пор, пока она не попадет на продолжение одного из ребер, по которым двигаются точки  $A_{i-1}$  и  $A_{i+1}$ . Эта идея изображена на рис. 9.

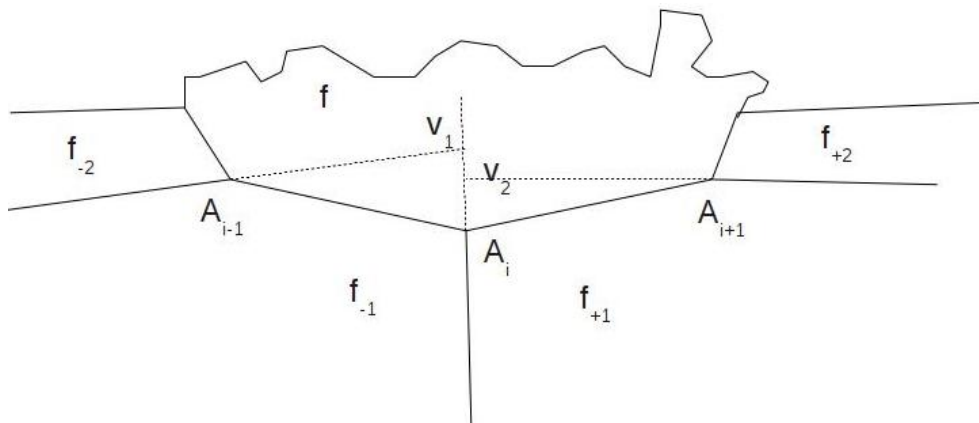


Рис. 9: Перемещения точки  $A_i$

В таком случае мы заменим  $A_i$  на  $v_2$ , а точку  $A_{i+1}$  вовсе выкинем из многогранника, а во всех позициях, где она фигурировала, запишем точку  $A_i = v_2$ . Таким образом в контуре уменьшится количество точек на 1.

Теперь мы можем описать процесс вычисления перемещения точки. Через  $n$  обозначим нормаль к главной грани, то есть грани построенного контура  
НАЙТИ\_ПЕРЕМЕЩЕНИЕ\_ТОЧКИ  $A_i$

```
{
  Если  $(f_{-2} \parallel f_{+1})$ 
     $v1 = \pi \cap f_{-1} \cap f_{+1}$ 
  Иначе
  {
     $v1 = f_{-2} \cap f_{-1} \cap f_{+1}$ 
```

```

Если ( $\pi(v_1) > 0$ )
{
     $v_1 = \pi \cap f_{-1} \cap f_{+1}$ 
}
}
 $d_1 = (v_1 - A_i, n)$ 
Если ( $f_{-1} \parallel f_{+2}$ )
     $v_2 = \pi \cap f_{-1} \cap f_{+1}$ 
Иначе
{
     $v_2 = f_{-1} \cap f_{+1} \cap f_{+2}$ 
    Если ( $\pi(v_2) > 0$ )
    {
         $v_1 = \pi \cap f_{-1} \cap f_{+1}$ 
    }
}
 $d_2 = (v_2 - A_i, n)$ 
 $d = \min\{d_1, d_2\}$ 
ВЕРНУТЬ  $d$ 
}

```

В общем случае кратности точек  $A_{i-1}$ ,  $A_i$  и  $A_{i+1}$  могут быть больше 3. В таких случаях точки движутся не по ребрам, а по пересечениям плоскостей, то есть не старое ребро продожается, но создается новое.

Зачем мы производили минимизацию по перемещению среди точек контура? Нельзя ли было просто поднимать их все подряд? Делается это затем, чтобы избежать самопересечений, так как если производить поднятие не последовательно, то могут возникнуть очень большие сдвиги.

#### 2.6.4 Перемещение точки

Итак, точка, которую нужно перемещать, найдена. Опишем процесс перемещения. Во-первых, повторяется вычисление перемещений  $d_1$  и  $d_2$ . Для определенности будем считать, что  $d_1 < d_2$ , то есть движение определяет левый сосед вершины.

Принципиальным здесь является следующее замечание. Если  $\pi_{-2} \parallel \pi_{+1}$  или плоскости  $\pi_{-2}$  и  $\pi_{+1}$  пересекаются, но  $\pi(v_1) > 0$ , то есть  $v_1$  лежит выше плоскости  $\pi$ , то точки  $A_{i-1}$  и  $A_i$  либо не сольются, либо сольются уже тогда, когда окажутся выше плоскости  $\pi$ . В этом случае не нужно объединять эти точки, но перемещать их до попадания на плоскость и оставить топологическую структуру многогранника как есть.

Содержательную часть алгоритма составляет случай, когда точки  $A_{i-1}$  и  $A_i$  сливаются ниже плоскости  $\pi$ . Его мы и опишем.

```

ПОНЯТЬ_ВЕРШИНУ  $A_i$ 
{
    ...
    Если ( $\text{СТЕПЕНЬ}(A_i) > 3$  и  $\text{СТЕПЕНЬ}(A_{i-1}) > 3$ )
    {
        Создать новую вершину с номером  $N$ 
         $vertex[N] = v_1$ 
        Добавить точку  $v_1$  в грань  $f_{-2}$  после точки  $A_{i-1}$ 
        Добавить точку  $v_1$  в грань  $f_{-1}$  после точки  $A_i$ 
        Добавить точку  $v_1$  в грань  $f_{+1}$  до точки  $A_i$ 
        Удалить точку  $A_i$  из грани  $f$ 
        Заменить точку  $A_{i-1}$  на точку  $v_1$  в грани  $f$ 
    }
    Иначе - Если ( $\text{СТЕПЕНЬ}(A_i) > 3$  и  $\text{СТЕПЕНЬ}(A_{i-1}) = 3$ )
    {
         $A_{i-1} := v_1$ 
        Добавить точку  $A_{i-1}$  в грань  $f_{+1}$  до точки  $A_i$ 
        Удалить точку  $A_i$  из грани  $f$ 
    }
    Иначе - Если ( $\text{СТЕПЕНЬ}(A_i) = 3$  и  $\text{СТЕПЕНЬ}(A_{i-1}) > 3$ )
    {
         $A_i := v_1$ 
        Добавить точку  $A_i$  в грань  $f_{-2}$  после точки  $A_{i-1}$ 
        Удалить точку  $A_{i-1}$  из грани  $f$ 
    }
    Иначе - Если ( $\text{СТЕПЕНЬ}(A_i) = 3$  и  $\text{СТЕПЕНЬ}(A_{i-1}) = 3$ )
    {
         $A_{i-1} := v_1$ 
        Удалить точку  $A_i$  из грани  $f$ 
        Удалить точку  $A_i$  из грани  $f_{-1}$ 
        Заменить точку  $A_i$  на точку  $A_{i-1}$  в грани  $f_{+1}$ 
    }
    ...
}

```

## 2.7 Рассечение модифицированного многогранника плоскостью

После того, как все точки контура оказались лежащими выше плоскости или на плоскости, можно просто применить алгоритм рассечения. Многогранник при этом не является обычным в смысле плоскостности граней, поскольку в главной грани не все точки лежат в одной плоскости. Поэтому сначала рассмотрим алгоритм рассечения обычного многогранника плоскостью, а затем скажем, что в нем нужно модифицировать, чтобы он работал и для нашего случая. Описанию программы

рассечения посвящен следующий раздел.

## 2.8 Тестирование алгоритма

Данный алгоритм был протестирован на нескольких 3D-моделях реальных камней. Далее перечислены достигнутые результаты.

### 2.8.1 Тестирование на многограннике polyhedron-2010-11-25

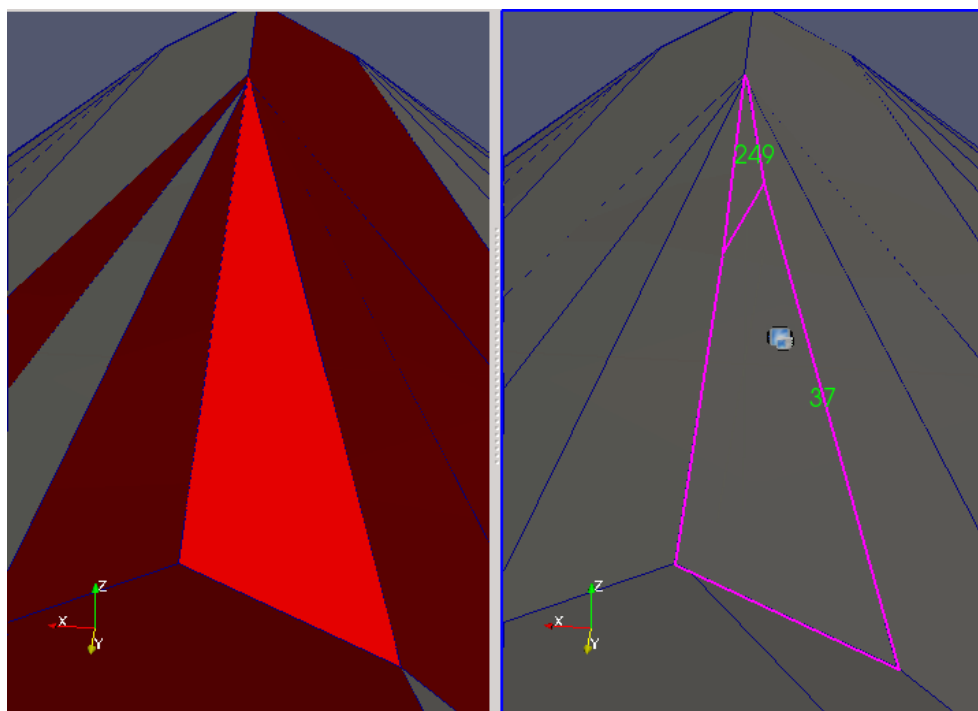


Рис. 10: Слияние граней 37 и 249



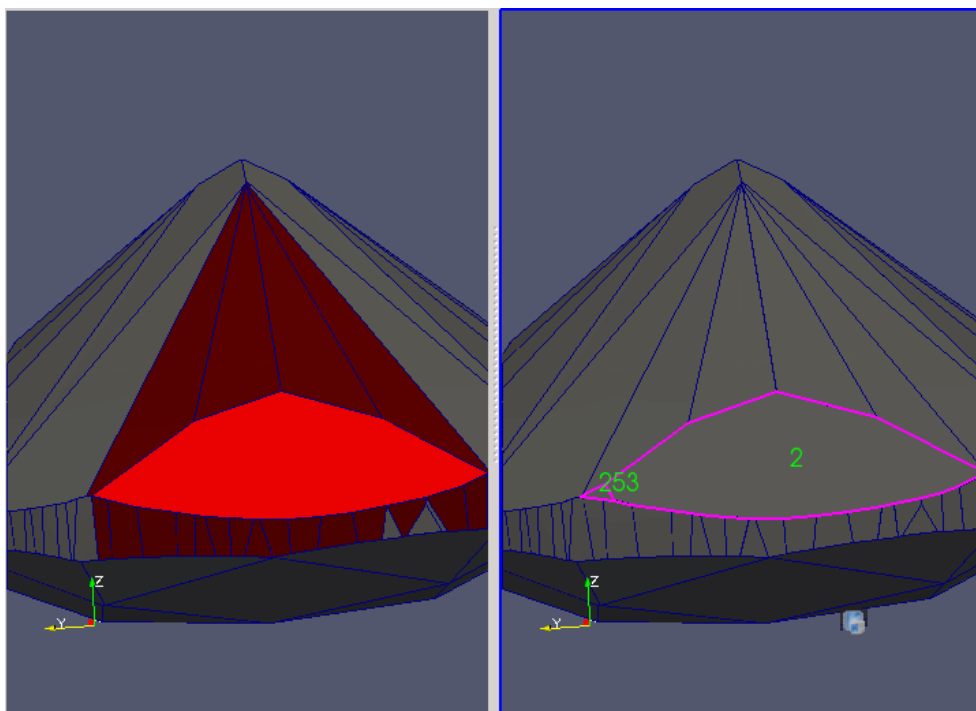


Рис. 11: Слияние граней 2 и 253

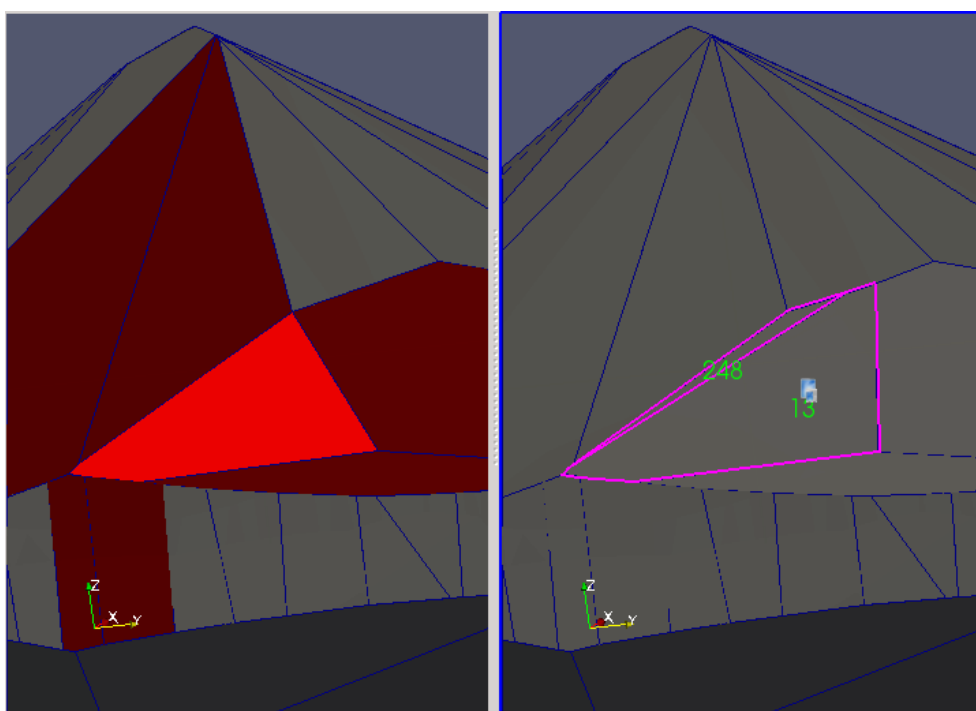


Рис. 12: Слияние граней 13 и 248

### **2.8.2 TODO: Тестирование на других примерах + случай трех-четырех граней**

## **3 Сечение многогранника плоскостью**

### **3.1 Постановка задачи**

Реализовать программу, позволяющую получать сечение произвольного многогранника данной плоскостью. Предусмотреть случай, когда образуется несколько компонент сечения.

### **3.2 Описание алгоритма**

#### **3.2.1 Этапы алгоритма**

1. Предобработка многогранника
2. Создание списков секущихся ребер для всех граней
3. Создание компонент сечения
4. Рассечение старых граней плоскостью
5. Создание граней и вершин искомого многогранника

#### **3.2.2 Предобработка многогранника**

1. В цикле для каждой грани создаются массивы номеров ее соседних граней и индексов, которыми обладают ее вершины в массивах соседей
2. В цикле для каждой вершины создается специальная структура, хранящая номера граней, в которых она лежит, и индексы, которыми она в этих гранях обладает

Оба цикла реализованы при помощи простого перебора

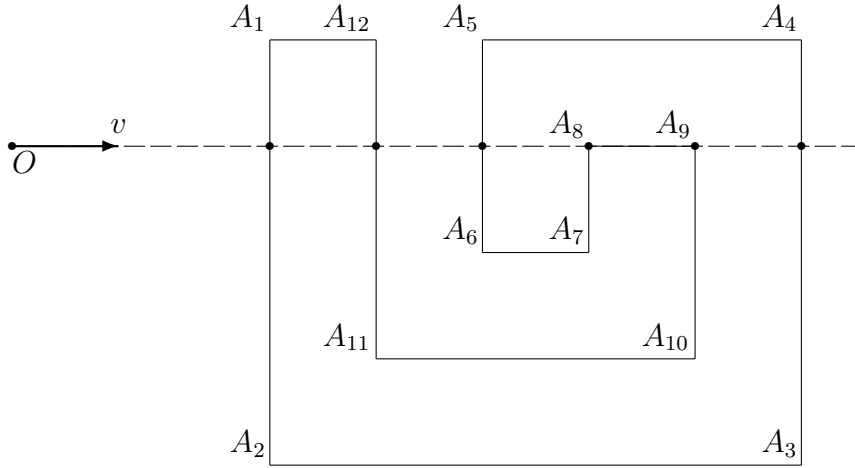
#### **3.2.3 Создание списков секущихся ребер для всех граней**

В результате выполнения этой программы образуется специальная структура, которая хранит в себе следующее:

- а) все ребра, секущиеся плоскостью, и все вершины, лежащие в плоскости, кроме так называемых висячих
- б) массив упорядочивающих координат
- в) массив номеров соседних плоскостей

г) массив направлений перехода

При этом ребра хранятся упорядоченно. А именно, если рассмотреть прямую, являющуюся пересечением плоскости грани  $\Phi$  и секущей плоскости  $\pi$ , т. е.  $l = \Phi \cap \pi$ , то вершины и точки пересечения ребер с плоскостью  $\pi$  расположены на этой прямой упорядоченно (в одном из двух возможных направлений).



$$F(\Phi_1, (A_1, A_2)) = (\Phi_2, (A_{11}, A_{12}))$$

Если в приведенном выше примере функции подать на вход грань  $\Phi_1$  и ребро  $(A_1, A_2)$ , то она обратится к массиву г), определит по величине 1, соответствующей ребру  $(A_1, A_2)$ , что нужно двигаться вправо, и обратится к следующему ребру массива а), т. е.  $(A_{11}, A_{12})$  и, соответственно к плоскости  $\Phi_2$ . На выходе она возвратит грань  $\Phi_2$  и ребро  $(A_{11}, A_{12})$ . Направления в массиве г) будут расставлены так, как показано на рисунке.

Как реализовано построение массива в)? Для ребра берется соседняя по этому ребру грань, которая определяется по структуре, построенной на этапе предобработки 1).

$$(A_1, A_2), (A_{11}, A_{12}), (A_5, A_6), A_8, A_9, (A_3, A_4)$$

Как реализовано это упорядочение? Пусть  $v$  - направляющий вектор прямой  $l = \Phi \cap \pi$ ,  $O \in l$  - некоторая фиксированная точка на этой прямой  $B_i$  - точка пересечения ребра грани с  $\pi$ :  $B_i = \pi \cap [A_i, A_{i+1}]$  (или вершина грани, если сама вершина лежит на  $\pi$ ). Тогда величина

$$\gamma_i = (v, OB_i)$$

есть координата точки  $B_i$  на прямой  $l$  в системе координат с началом в точке  $O$  и единичным вектором  $e = v$ . Поэтому величины  $\gamma_i$  могут быть использованы в качестве упорядочивающих координат.

Списки ребер строятся в два этапа:

1. При помощи простого перебора отыскиваются все секущиеся ребра и лежащие в плоскости вершины. Список строится таким образом, чтобы он сразу был упорядоченным. А именно, при добавлении ребра в структуру используется двоичный поиск по упорядочивающим координатам.
2. Затем заполняются массивы из пунктов б) и в), которые будут активно использоваться на дальнейших этапах алгоритма.

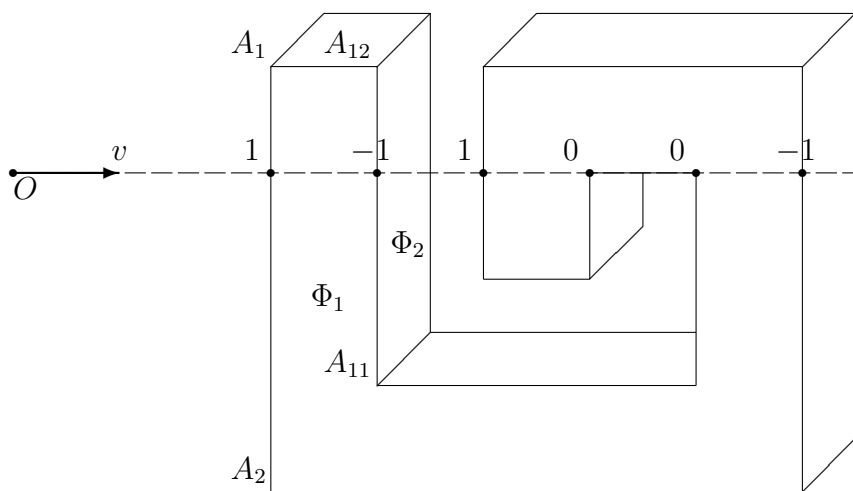
Какой цели служат массивы б) и в)? В основном алгоритме мы будем шагать по граням, собирая секущиеся ребра и вершины компонент сечения. Чтобы определить, в какую грань шагать дальше, заведена специальная функция, скажем  $F$ , использующая массивы б) и в). На вход она получает текущее ребро и грань, а на выходе возвращает следующее ребро и грань.

В массиве г) хранятся числа  $-1, 1, 0$ .

1 - если нужно двигаться вправо по списку

$-1$  - если нужно двигаться влево по списку

0 - если направление движения отдается на усмотрение пользователя

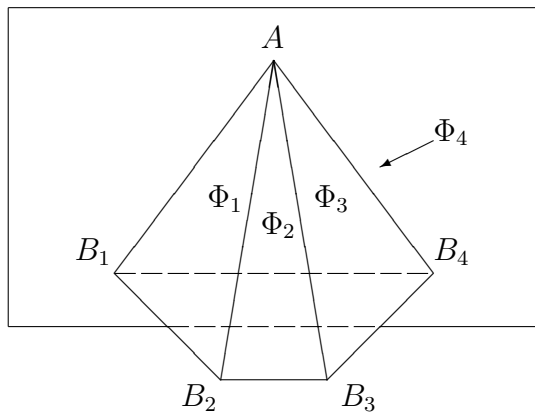


$$F(\Phi_1, (A_1, A_2)) = (\Phi_2, (A_{11}, A_{12}))$$

Если в приведенном выше примере функции подать на вход грань  $\Phi_1$  и ребро  $(A_1, A_2)$ , то она обратится к массиву г), определит по величине 1, соответствующей ребру  $(A_1, A_2)$ , что нужно двигаться вправо, и обратится к следующему ребру массива а), т. е.  $(A_{11}, A_{12})$  и, соответственно к плоскости  $\Phi_2$ . На выходе она возвратит грань  $\Phi_2$  и ребро  $(A_{11}, A_{12})$ . Направления в массиве г) будут расставлены так, как показано на рисунке.

Как реализовано построение массива  $v$ )? Для ребра берется соседняя по этому ребру грань, которая определяется по структуре, построенной на этапе предобработки 1).

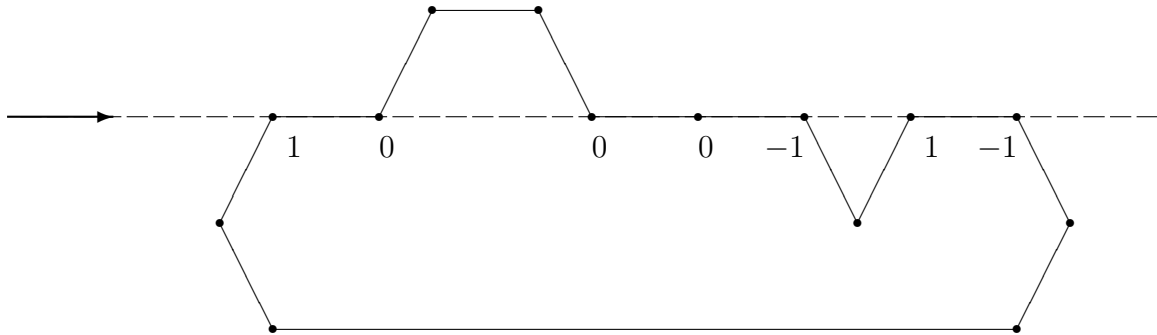
Для вершины также используется структура из этапа предобработки 1). А именно, перебирается массив содержащих эту вершину граней и проверяется, лежат ли вершины по одну или по разные стороны от плоскости. Граней, в которой они лежат по разные стороны, будет всего две. Одна из них - текущая, а вторую возьмем в качестве следующей.



В примере на приведенном выше рисунке для вершины  $A$  искомыми гранями будут  $\Phi_1$  и  $\Phi_2$ , поскольку вершины  $B_1$  и  $B_2$ ,  $B_3$  и  $B_4$  лежат по разные стороны от плоскости  $\pi$ .

Заметим, что для вершины предусмотрена одна исключительная ситуация: если в массиве  $g$ ) на соответствующей позиции стоит 0, то следующая грань не ищется, а считается, что в этой точке следующей гранью является текущая.

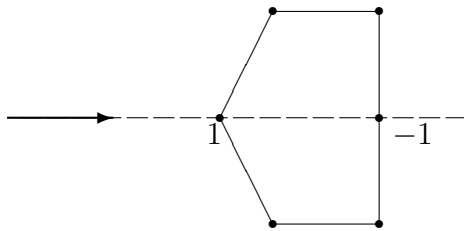
Как реализовано построение массива  $g$ )? При реализации используется следующая идея. Грань - замкнутое плоское множество. Рассмотрим прямую  $l = \pi \cap \Phi$  и будем двигаться непрерывно вдоль нее из бесконечности, положив начальное направление равным  $\nu = -1$ . Когда мы попадаем на грань, то меняем направление:  $\nu = \nu \cdot (-1) = 1$  и приписываем его задействованному ребру или вершине. Продолжаем двигаться дальше. Если мы оказались вне грани, то поменяем направление:  $\nu = \nu \cdot (-1) = -1$  и приписываем его задействованному ребру или вершине. В промежутке между сменами  $\nu$  нам могли попадаться вершины, лежащие на плоскости. Если они висят, то пропустим их, как было сказано выше. Если же не являются, то припишем им направление, равное 0.



Если применить процесс к такой грани, то направления будут приписаны так, как показано на рисунке.

Как реализовать этот процесс при помощи алгоритма?

Если точка пересечения грани с плоскостью одна, то она висячая, и рассмотрение такой грани можно отбросить. Если их две, то построение тривиально: одной приписываем направление 1, а другой -  $(-1)$ .



В общем случае точек пересечения не менее трех, и тогда запускается второй цикл по массиву вершин грани. При этом используются 2 вспомогательные переменные:  $s$  и  $\nu$ .  $\nu$  обозначает, находится ли шаг цикла внутри (1) или вне  $(-1)$  грани.  $s$  обозначает, находится ли шаг цикла выше (1) или ниже  $(-1)$  секущей плоскости. Для простоты алгоритм начинается с самой левой точки пересечения и идет по списку вершин грани в таком направлении, двигаясь по которому, мы получим следующую точку в списке за самой левой точкой (а не самую правую, как если бы выбрать противоположное направление).

$v_0$  = первой вершине

$v_{+1}$  = следующей за  $v_0$  вершине

цикл(по всем вершинам грани)

{

$s_0 = \text{знак}(v_0)$

$s_{+1} = \text{знак}(v_{+1})$

если( $s_0 = 0$  или  $s_0 \cdot s_{+1} = -1$ )

{

если( $s_{+1} = -s$  //следующая вершина уходит на другую сторону

На самом деле данный алгоритм описывает не совсем тот процесс, который был описан выше.



А именно, он перебирает ребра не в порядке их нахождения в списке. Впрочем, очевидно, что параметр  $s$  меняется нужным образом: количество подъемов равно количеству спусков за счет того, что не рассматриваются висячие вершины. А параметр  $\nu$  меняется четное число раз, и как раз в тех точках, где меняется параметр  $s$ . Значит, в результате алгоритма ребрам будут приписаны искомые направления.

Построение массивов  $v$ ) и  $g$ ) выделено в отдельный цикл, т. к. в алгоритме существенно, чтобы обход начинался с самой левой вершины в списке, а это возможно организовать только после того, как все текущие ребра и вершины зарегистрированы.

### 3.2.4 Создание компонент сечения

Организуется простой перебор полученных списков. Выбирается начальное ребро и при помощи функции перехода  $F$  ищутся те ребра, которые за ним следуют, до тех пор, пока цикл не придет к исходному ребру. Если в результате количество задействованных ребер меньше, чем то, которое хранится в списках, то выбирается какое-нибудь еще не задействованное ребро и цикл повторяется. В результате будет проделано ровно столько циклов, сколько компонент в сечении многогранника. Текущие ребра и вершины, образующие вершины этих компонент, запоминаются в специальных структурах.

### 3.2.5 Рассечение старых граней плоскостью

Используются построенные в пункте 2 списки.

```

пока(в списке есть неиспользованные ребра / вершины)
{
   $(v_0^0, v_1^0)$  = неиспользованное ребро или вершина
   $(v_0, v_1) = (v_0^0, v_1^0)$ 
  делать //цикл А
  {
    добавить ребро  $(v_0, v_1)$ 
     $(v_0, v_1)$  = следующее за  $(v_0, v_1)$  ребро
     $s_{+1}$  = знак следующей за  $v_0$ 
     $s_{-1}$  = знак предыдущей для  $v_0$ 
  }пока( $s_{-1} = 0$  и  $s_{+1} = 0$ )
  добавить ребро  $(v_0, v_1)$ 
  выбрать положительное направление
   $I$  = следующая вершина в положительном направлении
  делать //цикл Б
  {
    добавить вершину  $I$ 
  }
}

```

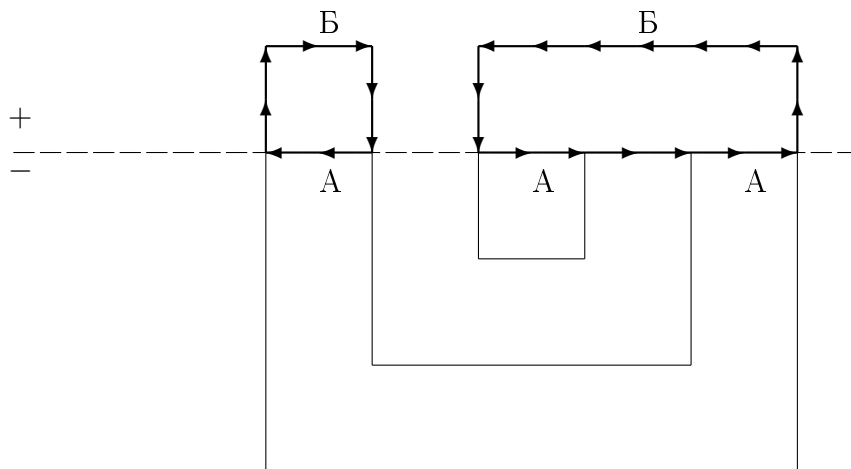


```

 $I$  = следующая за  $I$  в положительном направлении
 $s_0 = \text{знак}(I)$ 
}пока( $s_0 = 1$ )
если( $s_0 = 0$ )
{
 $v_0 = v_1 = I$ 
}
если( $s_0 = -1$ )
{
 $v_0 = I$ 
 $v_1 =$  предыдущей вершине для  $I$ 
}
}пока( $v_0 \neq v_0^0$  или  $v_1 \neq v_1^0$ )

```

Если оказалось, что в самом начале список пуст, то нужно попробовать найти в грани положительные вершины. Если таковых нет, то грань удаляется. Если такие есть, то все вершины являюся либо положительными, либо висячими, и грань целиком лежит выше плоскости, и тогда ее нужно оставить без изменения.



Алгоритм найдет 2 компоненты для такой грани. Стрелки на рисунке показывают порядок, в котором будут находиться вершины. А соответствует циклу А, а Б - циклу Б из алгоритма выше.

В цикле А движение происходит по плоскости, пока мы находимся внутри грани. Как только мы выходим за ее пределы, шагаем вверх и двигаемся до тех пор, пока снова не опустимся на плоскость (это соответствует циклу Б).

### 3.2.6 Создание граней и вершин искомого многогранника

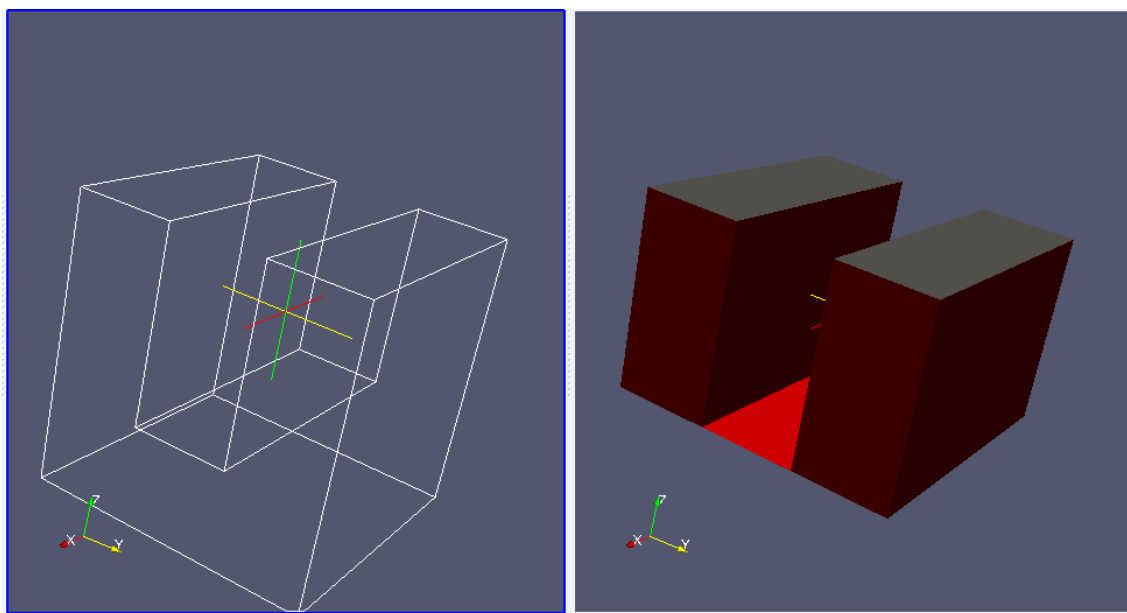
1. По мере создания списков секущихся ребер и вершин возникающие ребра запоминаются, т. к. они в пересечении с плоскостью создают новые вершины. Для запоминания используется структура типа множество.
2. Когда выполнены этапы 2 - 4, начинается создание новых граней по секущимся ребрам, которые были сохранены во время этапов 3 и 4. При создании вершинам присваиваются номера  $N + i$ , где  $N$  - количество вершин в старом многограннике,  $i$  - номер вершины в структуре типа множество.
3. После этого удаляются все отрицательные вершины и те грани, в которых нет положительных вершин.
4. Затем вершины и грани перенумеровываются, чтобы в массивах не было пустых элементов.

### 3.3 Тестирование алгоритма

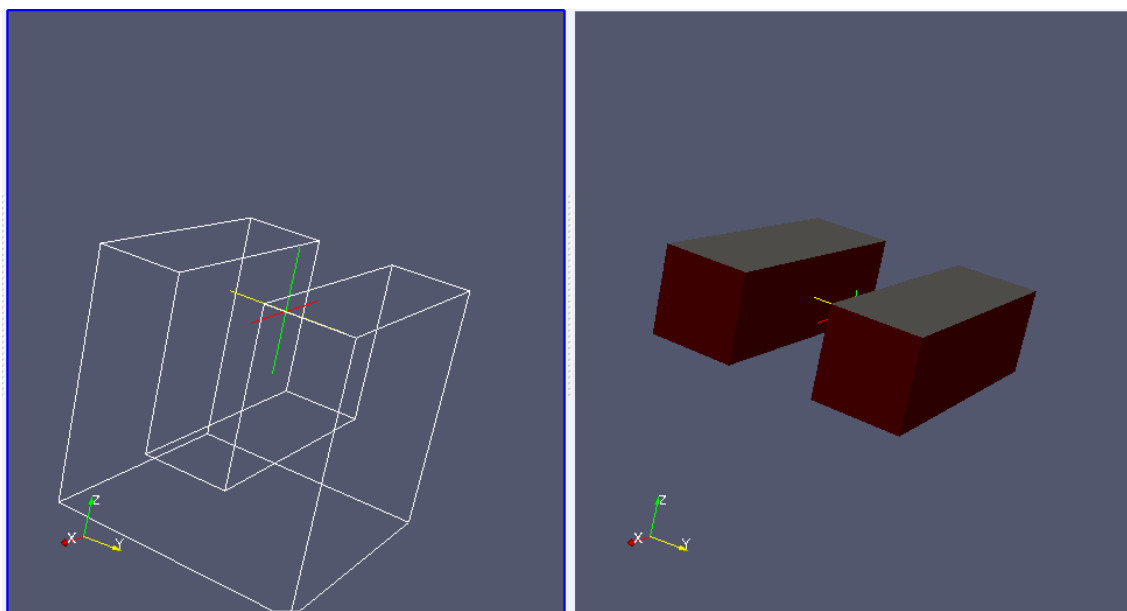
Алгоритм был протестирован на модельных и реальных примерах, в том числе таких, в которых возникают многокомпонентные сечения и грани, которые при рассечении распадаются на несколько компонент-граней.

Во всех приведенных ниже иллюстрациях слева изображен исходный многогранник, справа - его сечение. Компоненты сечения раскрашены ярко-красным цветом, а те грани, которые образовались в результате рассечения старых граней плоскостью, раскрашены темно-красным цветом.

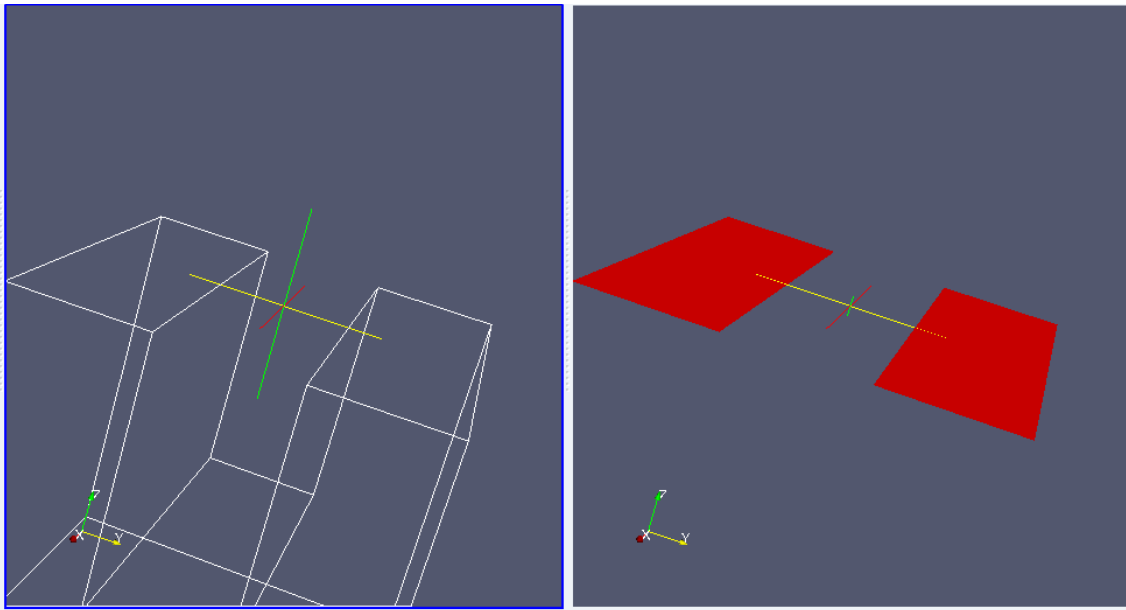
### 3.3.1 Тестирование на модельном многограннике cube-cutted



Сечение многогранника cube-cutted плоскостью  $z = 1$ . В данном случае образуется 1 компонента сечения. Заметим, что она состоит из всего одной грани, несмотря на то, что существует грань исходного многогранника, которая целиком содержится в компоненте.



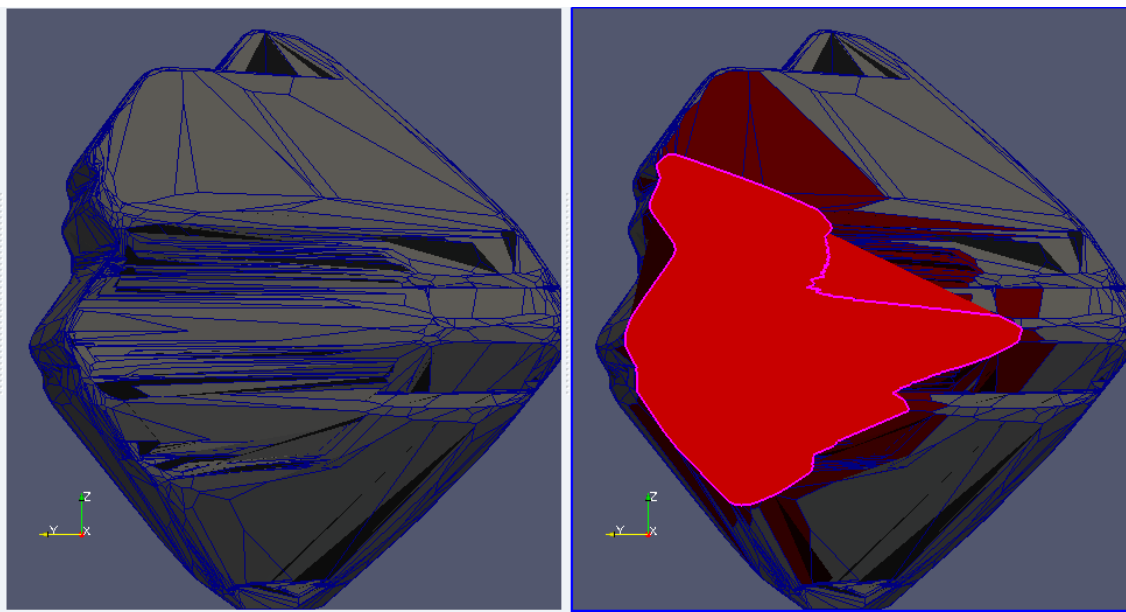
Сечение многогранника cube-cutted плоскостью  $z = 2$ . В данном случае образуется 2 компоненты сечения.



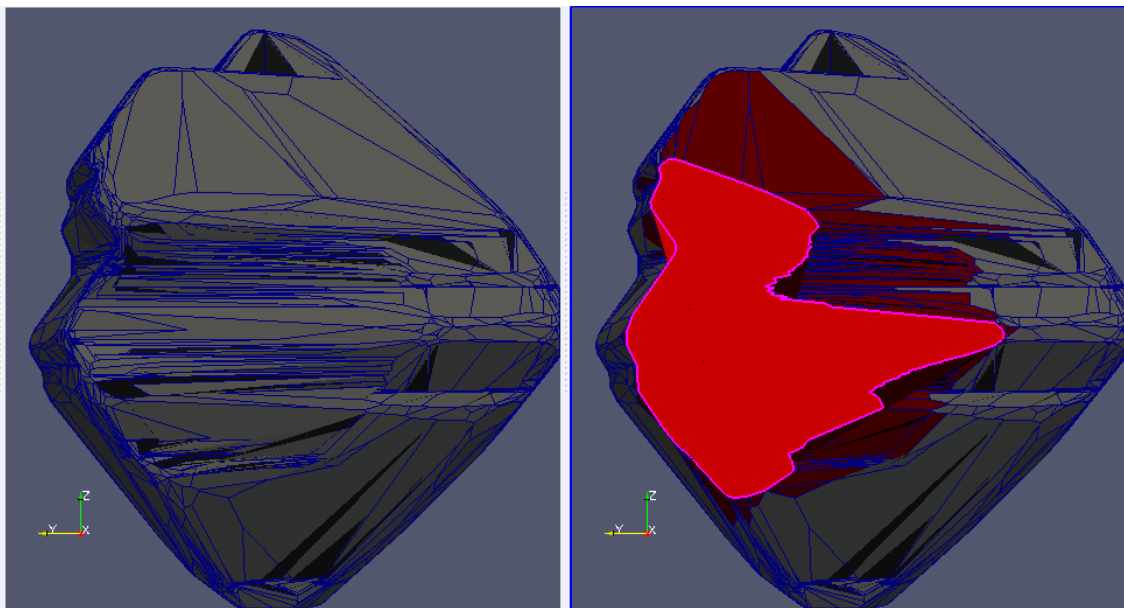
Сечение многогранника cube-cutted плоскостью  $z = 3$ . Многогранник понимается как выпуклое замкнутое множество, поэтому считается, что грань, целиком лежащая в секущей плоскости, должна либо сама войти в новый многогранник, либо содержаться внутри какой-то новой грани. В приведенном выше примере две грани многогранника лежат в плоскости, а все остальные лежат ниже.

### 3.3.2 Тестирование на многограннике poly-big

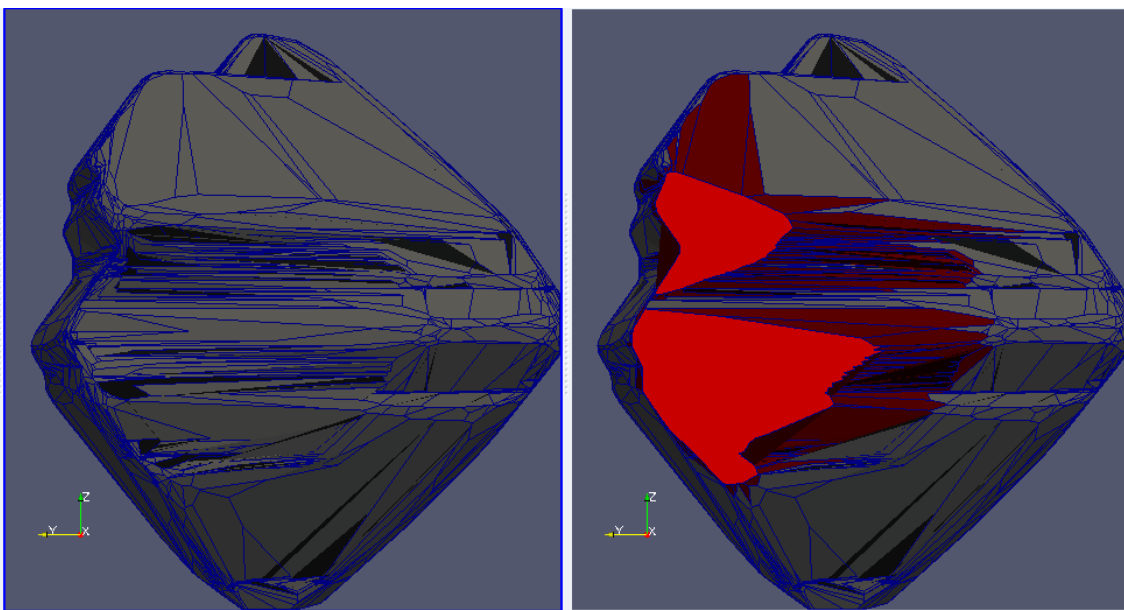
При тестировании возникают случаи, когда сечение имеет 2 или 3 компоненты



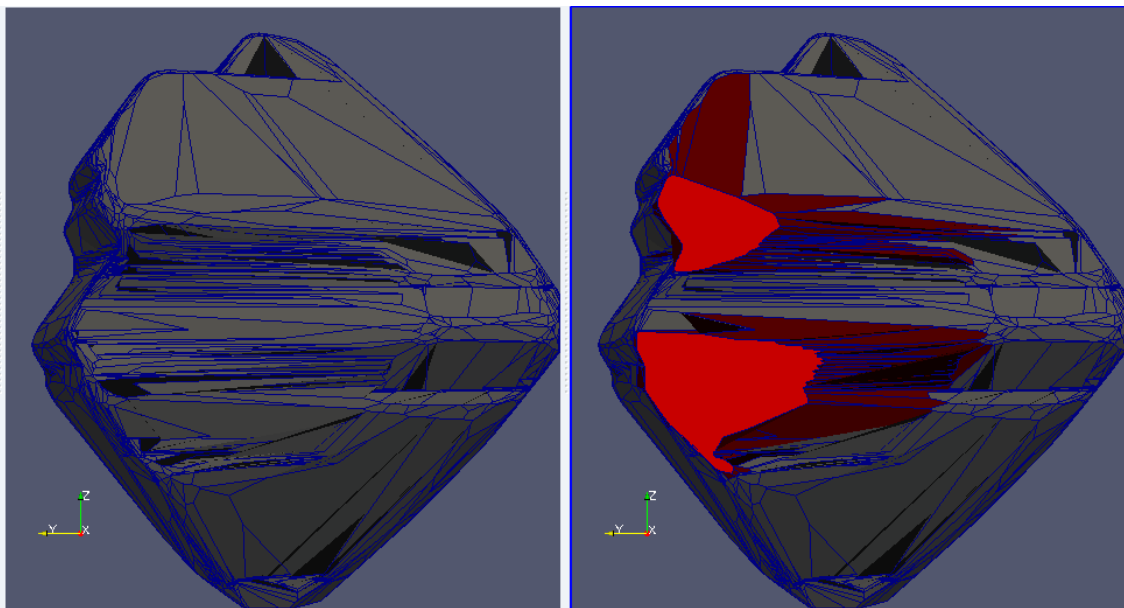
Сечение многогранника poly-big плоскостью  $x = -3$ .



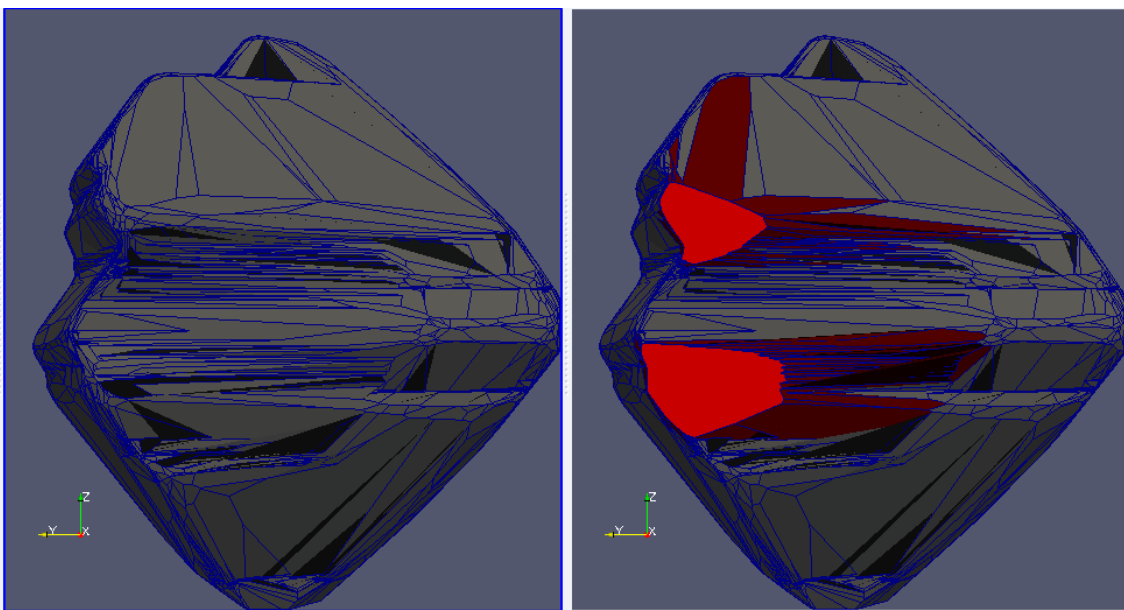
Сечение многогранника poly-big плоскостью  $x = -3.1$ .



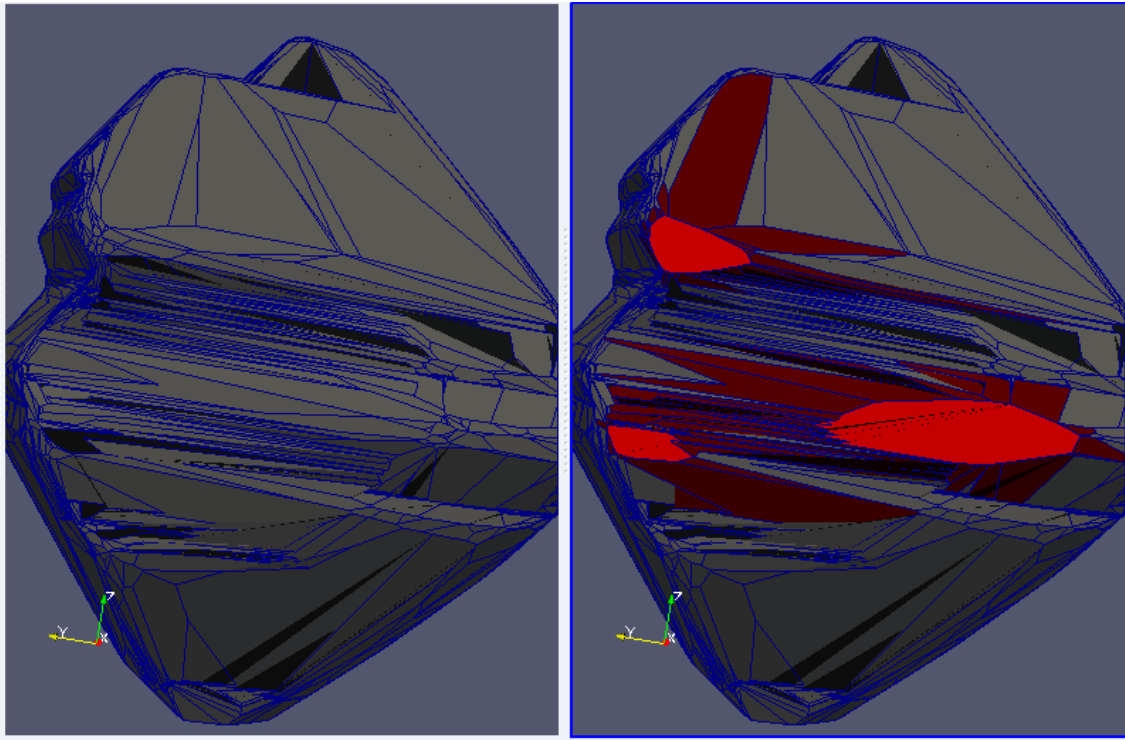
Сечение многогранника poly-big плоскостью  $x = -3.3$ .



Сечение многогранника poly-big плоскостью  $x = -3.4$ .



Сечение многогранника poly-big плоскостью  $x = -3.5$ .



Сечение многогранника poly-big особой плоскостью, при котором образуется 3 компоненты в сечении.

### 3.4 TODO: Модификация алгоритма для задачи слияния смежных граней

## 4 Подвижка вершины и дальнейшая перестройка многогранника

### 4.1 Идея задачи

В разделе "Введение" была сформулирована идея коррекции многогранника путем сглаживания граней. Она возникла в результате анализа дефектов в процессе создания 3D-моделей реальных камней. Есть и другая проблема, которая возникает в этом процессе. Дело в том, что как правило в многогранниках не бывает вершин со степенью выше 3, а если в реальном камне таковые имеются, то они разбиваются на несколько вершин. Возникает ситуация, когда заведомо неоправданно увеличивается число точек и усложняется модель.

Как бороться с этой проблемой? Можно попытаться зарегистрировать такие "кучки" точек, заменять их на одну вершину и затем перестраивать многогранник так, чтобы восстановилась плоскостность. Для начала, чтобы двигаться в этом

направлении, нужно научиться менять местоположение хотя бы *одной* точки. Итак, возникает следующая задача.

Имеется многогранник, состоящий из вершин и граней. Все его точки с некоторой погрешностью лежат в определенных гранях. Затем берется одна его точка, меняется ее местоположение и фиксируется. **Требуется** скорректировать положение точек и граней многогранника, чтобы точки вновь стали попадать в соответствующие грани.

У этой задачи имеется несколько вариантов формализации.

## 4.2 Формализация задачи

### 4.2.1 Первый вариант: варьирование вершин

Введем обозначения. Пусть вершины многогранника представлены следующим образом:

$$A_1^0 = \begin{pmatrix} x_1^0 \\ y_1^0 \\ z_1^0 \end{pmatrix}, A_2^0 = \begin{pmatrix} x_2^0 \\ y_2^0 \\ z_2^0 \end{pmatrix}, \dots, A_N^0 = \begin{pmatrix} x_N^0 \\ y_N^0 \\ z_N^0 \end{pmatrix}.$$

Пусть грани многогранника заданы следующим образом:

$$\begin{aligned} \pi_1 &= A_{k(1,1)}^0 A_{k(1,2)}^0 \dots A_{k(1,n_1)}^0, \\ \pi_2 &= A_{k(2,1)}^0 A_{k(2,2)}^0 \dots A_{k(2,n_2)}^0, \\ &\dots \\ \pi_M &= A_{k(M,1)}^0 A_{k(M,2)}^0 \dots A_{k(M,n_M)}^0 \end{aligned}$$

где  $k(i, j)$  -  $j$ -я вершина в  $i$ -й грани,  $n_i$  - количество вершин в  $i$ -й грани.  $N$  - количество вершин многогранника,  $M$  - количество граней многогранника

Деформируем  $N$ -ю точку многогранника:

$$A_N = A_N^0 + \Delta A_N = \begin{pmatrix} x_N^0 + \Delta x_N \\ y_N^0 + \Delta y_N \\ z_N^0 + \Delta z_N \end{pmatrix},$$

причем приращение  $\Delta A_N$  фиксировано.

1). Вычислим для граней  $\pi_1, \pi_2, \dots, \pi_M$  те плоскости, в которых они реально расположены, - например, методом наименьших квадратов:

$$\begin{aligned} \pi_1 : a_1^0 x + b_1^0 y + c_1^0 z + d_1^0 &= 0, \\ \pi_2 : a_2^0 x + b_2^0 y + c_2^0 z + d_2^0 &= 0, \end{aligned}$$



...

$$\pi_M : a_M^0 x + b_M^0 y + c_M^0 z + d_M^0 = 0$$

При расчете тех граней, к которым принадлежит подвинутая вершина  $A_N$ , нужно помнить, что точка должна лежать в этих гранях точно. Пусть таких граней всего  $m_N$  штук.

2). Решим следующую экстремальную задачу:

$$\sum_{i=1}^{N-1} |\Delta A_i|^2 = \sum_{i=1}^{N-1} (|\Delta x_i|^2 + |\Delta y_i|^2 + |\Delta z_i|^2) \rightarrow \min$$

при следующих условиях связи:

$$\left\{ \begin{array}{l} a_1^0 x_{k(1,1)} + b_1^0 y_{k(1,1)} + c_1^0 z_{k(1,1)} + d_1^0 = 0 \\ a_1^0 x_{k(1,2)} + b_1^0 y_{k(1,2)} + c_1^0 z_{k(1,2)} + d_1^0 = 0 \\ \dots \\ a_1^0 x_{k(1,n_1)} + b_1^0 y_{k(1,n_1)} + c_1^0 z_{k(1,n_1)} + d_1^0 = 0 \\ \dots \\ \dots \\ a_m^0 x_{k(m,1)} + b_m^0 y_{k(m,1)} + c_m^0 z_{k(m,1)} + d_m^0 = 0 \\ a_m^0 x_{k(m,2)} + b_m^0 y_{k(m,2)} + c_m^0 z_{k(m,2)} + d_m^0 = 0 \\ \dots \\ a_m^0 x_{k(m,n_m)} + b_m^0 y_{k(m,n_m)} + c_m^0 z_{k(m,n_m)} + d_m^0 = 0 \end{array} \right.$$

Эти условия описывают тот факт, что вершины точным образом лежат на соответствующих им гранях. Число таких условий равно

$$\sum_{i=1}^M n_i - m_N = \sum_{j=1}^{N-1} m_j \geq 3(N-1),$$

поскольку каждая вершина лежит как минимум в трех гранях ( $m_j$  - количество граней, содержащих  $j$ -ю вершину). Таким образом, мы имеем не менее  $3(N-1)$  условий относительно  $3(N-1)$  переменных  $\Delta x_i, \Delta y_i, \Delta z_i, i = 1, \dots, N-1$ . Если каждая вершина лежит ровно в трех гранях, то для решения задачи нужно решить систему линейных уравнений методом Гаусса. Однако, ничего нельзя будет сказать о значении функционала на этом решении. В общем же случае система переопределена и решение найти нельзя. Есть несколько вариантов решить эту проблему.

#### 4.2.2 Второй вариант: варьирование вершин и граней

Не будем фиксировать полученные уравнения плоскостей граней, т. е. введем в систему дополнительные переменные

$$\begin{aligned} a_1 &= a_1^0 + \Delta a_1, b_1 = b_1^0 + \Delta b_1, c_1 = c_1^0 + \Delta c_1, d_1 = d_1^0 + \Delta d_1, \\ &\dots \\ a_M &= a_M^0 + \Delta a_M, b_M = b_M^0 + \Delta b_M, c_M = c_M^0 + \Delta c_M, d_M = d_M^0 + \Delta d_M \end{aligned}$$

Рассмотрим тогда следующую экстремальную задачу

$$\sum_{i=1}^{N-1} (|\Delta x_i|^2 + |\Delta y_i|^2 + |\Delta z_i|^2) + \sum_{j=1}^M (|\Delta a_j|^2 + |\Delta b_j|^2 + |\Delta c_j|^2 + |\Delta d_j|^2) \rightarrow \min$$

при следующих условиях связи:

$$\left\{ \begin{array}{l} a_1 x_{k(1,1)} + b_1 y_{k(1,1)} + c_1 z_{k(1,1)} + d_1 = 0 \\ a_1 x_{k(1,2)} + b_1 y_{k(1,2)} + c_1 z_{k(1,2)} + d_1 = 0 \\ \dots \\ a_1 x_{k(1,n_1)} + b_1 y_{k(1,n_1)} + c_1 z_{k(1,n_1)} + d_1 = 0 \\ \dots \\ \dots \\ a_m x_{k(m,1)} + b_m y_{k(m,1)} + c_m z_{k(m,1)} + d_m = 0 \\ a_m x_{k(m,2)} + b_m y_{k(m,2)} + c_m z_{k(m,2)} + d_m = 0 \\ \dots \\ a_m x_{k(m,n_m)} + b_m y_{k(m,n_m)} + c_m z_{k(m,n_m)} + d_m = 0 \end{array} \right.$$

Поступим стандартным образом. Составим для этой задачи функцию Лагранжа:

$$\begin{aligned} L &= \sum_{i=1}^{N-1} (|\Delta x_i|^2 + |\Delta y_i|^2 + |\Delta z_i|^2) + \\ &+ \sum_{j=1}^M (|\Delta a_j|^2 + |\Delta b_j|^2 + |\Delta c_j|^2 + |\Delta d_j|^2) - \\ &- \sum_{i=1}^M \sum_{j=1}^{n_i} \lambda_{k(i,j)} (a_i x_{k(i,j)} + b_i y_{k(i,j)} + c_i z_{k(i,j)} + d_i) \end{aligned}$$

Экстремум этой функции достигается в тех же точках, что и у исходного функционала. Продифференцируем эту функцию и приравняем частные производные к нулю:

$$\frac{\partial L}{\partial \Delta x_1} = 2(x_1^0 + \Delta x_1) - \sum_{\{i: A_1 \in \pi_i\}} \lambda_{k(i,j_i)} (a_i^0 + \Delta a_i) = 0$$

$$\begin{aligned}
\frac{\partial L}{\partial \Delta y_1} &= 2(y_1^0 + \Delta y_1) - \sum_{\{i: A_1 \in \pi_i\}} \lambda_{k(i, j_i)}(b_i^0 + \Delta b_i) = 0 \\
\frac{\partial L}{\partial \Delta z_1} &= 2(z_1^0 + \Delta z_1) - \sum_{\{i: A_1 \in \pi_i\}} \lambda_{k(i, j_i)}(c_i^0 + \Delta c_i) = 0 \\
&\dots \\
\frac{\partial L}{\partial \Delta x_{N-1}} &= 2(x_{N-1}^0 + \Delta x_{N-1}) - \sum_{\{i: A_{N-1} \in \pi_i\}} \lambda_{k(i, j_i)}(a_i^0 + \Delta a_i) = 0 \\
\frac{\partial L}{\partial \Delta y_{N-1}} &= 2(y_{N-1}^0 + \Delta y_{N-1}) - \sum_{\{i: A_{N-1} \in \pi_i\}} \lambda_{k(i, j_i)}(b_i^0 + \Delta b_i) = 0 \\
\frac{\partial L}{\partial \Delta z_{N-1}} &= 2(z_{N-1}^0 + \Delta z_{N-1}) - \sum_{\{i: A_{N-1} \in \pi_i\}} \lambda_{k(i, j_i)}(c_i^0 + \Delta c_i) = 0 \\
\frac{\partial L}{\partial \Delta a_1} &= 2(a_1^0 + \Delta a_1) - \sum_{j=1}^{n_1} \lambda_{k(1, j)}(x_{k(1, j)}^0 + \Delta x_{k(1, j)}) = 0 \\
\frac{\partial L}{\partial \Delta b_1} &= 2(b_1^0 + \Delta b_1) - \sum_{j=1}^{n_1} \lambda_{k(1, j)}(y_{k(1, j)}^0 + \Delta y_{k(1, j)}) = 0 \\
\frac{\partial L}{\partial \Delta c_1} &= 2(c_1^0 + \Delta c_1) - \sum_{j=1}^{n_1} \lambda_{k(1, j)}(z_{k(1, j)}^0 + \Delta z_{k(1, j)}) = 0 \\
\frac{\partial L}{\partial \Delta d_1} &= 2(d_1^0 + \Delta d_1) - \sum_{j=1}^{n_1} \lambda_{k(1, j)} = 0 \\
&\dots \\
\frac{\partial L}{\partial \Delta a_M} &= 2(a_M^0 + \Delta a_M) - \sum_{j=1}^{n_M} \lambda_{k(M, j)}(x_{k(M, j)}^0 + \Delta x_{k(M, j)}) = 0 \\
\frac{\partial L}{\partial \Delta b_M} &= 2(b_M^0 + \Delta b_M) - \sum_{j=1}^{n_M} \lambda_{k(M, j)}(y_{k(M, j)}^0 + \Delta y_{k(M, j)}) = 0 \\
\frac{\partial L}{\partial \Delta c_M} &= 2(c_M^0 + \Delta c_M) - \sum_{j=1}^{n_M} \lambda_{k(M, j)}(z_{k(M, j)}^0 + \Delta z_{k(M, j)}) = 0 \\
\frac{\partial L}{\partial \Delta d_M} &= 2(d_M^0 + \Delta d_M) - \sum_{j=1}^{n_M} \lambda_{k(M, j)} = 0
\end{aligned}$$

Вместе с условиями связи эти равенства образуют систему из

$3(N-1) + 4M + \sum_{i=1}^M n_i - m_N$  уравнений относительно  $3(N-1) + 4M + \sum_{i=1}^M n_i - m_N$  неизвестных. Решение этой системы нелинейных уравнений представляет собой открытый вопрос.

### 4.2.3 Третий вариант: поочередное варьирование вершин и граней

Здесь же вариация происходит поочередно:

1. Вычисляем плоскости методом наименьших квадратов
2. Изменяем координаты вершин таким образом, чтобы минимизировать функционал

$$J = \sum_{i=1}^{N-1} (|x_i - x_i^0|^2 + |y_i - y_i^0|^2 + |z_i - z_i^0|^2) + K \sum_{j=1}^M \sum_{i: A_i \in \pi_j} |a_j x_i + b_j y_i + c_j z_i + d_j|^2 \rightarrow \min$$

При этом нужно помнить, что коэффициенты уравнений плоскостей на этом шаге уже фиксированы, и менять их не нужно. Переменная  $K$  называется штрафом

3. Возвращаемся к пункту 1. и т. д.

Решим задачу минимизации функционала  $J$ . Запишем частные производные функционала по переменным  $x_i, y_i, z_i$  и приравняем их к нулю.

$$\begin{aligned} \frac{\partial J}{\partial x_i} &= 2x_i - 2x_i^0 + 2K \sum_{j: A_i \in \pi_j} a_j (a_j x_i + b_j y_i + c_j z_i + d_j) = 0 \\ \frac{\partial J}{\partial y_i} &= 2y_i - 2y_i^0 + 2K \sum_{j: A_i \in \pi_j} b_j (a_j x_i + b_j y_i + c_j z_i + d_j) = 0 \\ \frac{\partial J}{\partial z_i} &= 2z_i - 2z_i^0 + 2K \sum_{j: A_i \in \pi_j} c_j (a_j x_i + b_j y_i + c_j z_i + d_j) = 0 \end{aligned}$$

Заметим, что система расщепляется на системы размерности  $3 \times 3$ . В матричной форме они будут иметь вид:

$$\begin{pmatrix} 1 + K(a, a) & (a, b) & (a, c) \\ (a, b) & 1 + K(b, b) & (b, c) \\ (a, c) & (b, c) & 1 + K(c, c) \end{pmatrix} \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} = \begin{pmatrix} -K(a, d) + x_i^0 \\ -K(b, d) + y_i^0 \\ -K(c, d) + z_i^0 \end{pmatrix}$$

$$\begin{aligned} \text{Здесь введены такие обозначения: } (a, a) &= \sum_{j: A_i \in \pi_j} a_j^2, (a, a) = \sum_{j: A_i \in \pi_j} a_j^2, (b, b) = \sum_{j: A_i \in \pi_j} b_j^2, \\ (c, c) &= \sum_{j: A_i \in \pi_j} c_j^2, (a, b) = \sum_{j: A_i \in \pi_j} a_j b_j, (a, c) = \sum_{j: A_i \in \pi_j} a_j c_j, (b, c) = \sum_{j: A_i \in \pi_j} b_j c_j, \\ (a, d) &= \sum_{j: A_i \in \pi_j} a_j d_j, (b, d) = \sum_{j: A_i \in \pi_j} b_j d_j, (c, d) = \sum_{j: A_i \in \pi_j} c_j d_j. \end{aligned}$$

Данные обозначения можно интерпретировать еще и как скалярные произведения векторов  $a, b, c, d$  - записанных в строку коэффициентов тех граней, в которых содержится точка  $A_i$ .

Имеет ли эта система решение? Представим ее матрицу в следующем виде:

$$\begin{pmatrix} 1 + K(a, a) & (a, b) & (a, c) \\ (a, b) & 1 + K(b, b) & (b, c) \\ (a, c) & (b, c) & 1 + K(c, c) \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} + K \begin{pmatrix} (a, a) & (a, b) & (a, c) \\ (a, b) & (b, b) & (b, c) \\ (a, c) & (b, c) & (c, c) \end{pmatrix}$$

Таким образом, матрица представляется в виде суммы единичной матрицы и матрицы Грама векторов  $a, b, c$ , которая является положительно определенной. Значит и исходная матрица положительно определена, более того, она симметрична, значит система невырождена и обязательно имеет решение.

### 4.3 Тестирование алгоритма

Выше представлен всего лишь примерный план реализации алгоритма. В действительности есть много его модификаций: от тех, которые дают довольно адекватный результат, до тех, которые полностью "выворачивают" многогранник наизнанку. При этом происходят следующие процессы: в гранях (как в многоугольниках) возникают самопересечения, некоторые ребра начинают пересекать те грани, в которых они не лежат.

Множественность возникает из-за того, что задаче есть два изменяемых параметра:

1. Штраф  $K$ . Каким его сделать? Если слишком большим, то алгоритм перестанет "чувствовать" смещения вершин, а если слишком малым, то наоборот - алгоритм будет плохо решать задачу восстановления плоскостности многогранника.
2. Как чередовать шаги 1 и 2? Менять ли при этом штраф  $K$ ?

#### 4.3.1 Экспоненциальный рост штрафа, локальная минимизация

В результате тестирования одним из приемлемых оказался следующий подход. Минимизируется не функционал

$$J = \sum_{i=1}^{N-1} (|x_i - x_i^0|^2 + |y_i - y_i^0|^2 + |z_i - z_i^0|^2) + K \sum_{j=1}^M \sum_{i: A_i \in \pi_j} |a_j x_i + b_j y_i + c_j z_i + d_j|^2,$$

но функционал

$$J = \sum_{i=1}^{N-1} (|x_i^{(n+1)} - x_i^{(n)}|^2 + |y_i^{(n+1)} - y_i^{(n)}|^2 + |z_i^{(n+1)} - z_i^{(n)}|^2) + K \sum_{j=1}^M \sum_{i: A_i \in \pi_j} |a_j x_i^{(n+1)} + b_j y_i^{(n+1)} + c_j z_i^{(n+1)} + d_j|^2,$$

то есть минимизируется не сдвиг в целом, а лишь сдвиг на каждом шаге. При этом не обязательно находится минимум исходного функционала, а лишь некоторая конфигурация многогранника, которая удовлетворяет условиям плоскостности, не являясь, вообще говоря, оптимальной.

Следующий алгоритм позволил достичь удовлетворительных результатов:

1.  $K = 100$
2.  $\varepsilon = 10^{-7}$
3. Цикл 1. Делать 10 раз:
  - 3.1. Делать цикл 2:
    - 3.1.1. Вычислить новые плоскости методом наименьших квадратов
    - 3.1.2. Вычислить координаты вершин, минимизирующих функционал со штрафом, равным  $K$
    - 3.2.3. Вычислить погрешность попадания вершин в плоскости
  - 3.2 ...пока эта погрешность больше, чем  $\varepsilon$
  - 3.3 Конец цикла 2
  - 3.4 Увеличиваем  $K$  в два раза
  - 3.5 Уменьшаем  $\varepsilon$  в два раза
4. Конец цикла 1.

Как этот алгоритм работает? На тестируемых многогранниках он выдал следующее поведение. Программа заходит в цикл 1, затем в нем - в цикл 2, делает примерно 30 шагов, выходит из 2 цикла, снова начинает 2 цикл (с увеличенным в 2 раза штрафом и с уменьшенной в 2 раза погрешностью), но теперь уже делает только 1 шаг, снова выходит из 2 цикла и т. д. - всего примерно 10 раз. Вот таблица результатов алгоритма ( $\Delta$  - сумарный сдвиг вершин,  $\varepsilon$  - погрешность попадания вершин в плоскости,  $\varepsilon_0$  - граница погрешности, пересечение которой обеспечивает конец цикла 2,  $K$  - штраф):

цикл 1	цикл 2	$\Delta$	$\varepsilon$	$\varepsilon_0$	$K$
0	0	0.114942	4.220043e-02	1.000000e-07	100.000000
0	1	0.048647	2.640706e-02	1.000000e-07	100.000000
0	2	0.018739	1.637491e-02	1.000000e-07	100.000000
0	3	0.007239	1.015491e-02	1.000000e-07	100.000000
0	4	0.002856	6.344031e-03	1.000000e-07	100.000000
0	5	0.001151	3.991388e-03	1.000000e-07	100.000000
0	6	0.000472	2.528785e-03	1.000000e-07	100.000000
0	7	0.000196	1.634950e-03	1.000000e-07	100.000000
0	8	0.000082	1.064094e-03	1.000000e-07	100.000000
0	9	0.000035	6.940496e-04	1.000000e-07	100.000000
0	10	0.000015	4.535194e-04	1.000000e-07	100.000000
0	11	0.000006	2.967911e-04	1.000000e-07	100.000000
0	12	0.000003	1.944614e-04	1.000000e-07	100.000000
0	13	0.000001	1.275375e-04	1.000000e-07	100.000000
0	14	0.000001	8.370982e-05	1.000000e-07	100.000000
0	15	0.000000	5.497753e-05	1.000000e-07	100.000000
0	16	0.000000	3.612497e-05	1.000000e-07	100.000000
0	17	0.000000	2.374594e-05	1.000000e-07	100.000000
0	18	0.000000	1.561315e-05	1.000000e-07	100.000000
0	19	0.000000	1.026795e-05	1.000000e-07	100.000000
0	20	0.000000	6.753804e-06	1.000000e-07	100.000000
0	21	0.000000	4.442928e-06	1.000000e-07	100.000000
0	22	0.000000	2.923018e-06	1.000000e-07	100.000000
0	23	0.000000	1.923204e-06	1.000000e-07	100.000000
0	24	0.000000	1.265446e-06	1.000000e-07	100.000000
0	25	0.000000	8.326840e-07	1.000000e-07	100.000000
0	26	0.000000	5.479375e-07	1.000000e-07	100.000000
0	27	0.000000	3.605724e-07	1.000000e-07	100.000000
0	28	0.000000	2.372805e-07	1.000000e-07	100.000000
0	29	0.000000	1.561485e-07	1.000000e-07	100.000000
0	30	0.000000	1.027587e-07	1.000000e-07	100.000000
0	31	0.000000	6.762423e-08	1.000000e-07	100.000000
1	32	0.000000	2.585086e-08	5.000000e-08	200.000000
2	33	0.000000	9.229164e-09	2.500000e-08	400.000000
3	34	0.000000	3.159339e-09	1.250000e-08	800.000000
4	35	0.000000	1.053913e-09	6.250000e-09	1600.000000
5	36	0.000000	3.462331e-10	3.125000e-09	3200.000000
6	37	0.000000	1.131674e-10	1.562500e-09	6400.000000
7	38	0.000000	3.720410e-11	7.812500e-10	12800.000000
8	39	0.000000	1.253741e-11	3.906250e-10	25600.000000
9	40	0.000000	4.448228e-12	1.953125e-10	51200.000000

### 4.3.2 Масштабирование штрафа

Можно попробовать реализовать следующую идею.

$$J(x) = \sum_{i=1}^{N-1} (|\Delta x_i|^2 + |\Delta y_i|^2 + |\Delta z_i|^2) + K \sum_{j=1}^M \sum_{i:A_i \in \pi_j} |a_j x_i + b_j y_i + c_j z_i + d_j|^2 = S_1 + K S_2$$

Пусть теперь  $S_1 = \varepsilon_1$ ,  $S_2 = \varepsilon_2$ .

Тогда выберем  $K = \frac{\varepsilon_1}{\varepsilon_2}$ .

Это дает преимущество в том, что теперь слагаемые будут одного порядка.

Прежний алгоритм выдавал вот такие результаты:

цикл 1	цикл 2	$S_2$ , погрешности по- паданий в плоскости	$S_1$ , сдвиги вершин	штраф $K$
0	0	4.509368e-03	1.224731e-03	100.000000
0	1	2.818460e-03	1.055994e-03	100.000000
...	...	...	...	...
0	10	5.619919e-05	1.414728e-03	100.000000
...	...	...	...	...
0	27	7.088412e-08	1.435461e-03	100.000000
1	28	2.768007e-08	1.435470e-03	200.000000
2	29	1.009637e-08	1.435476e-03	400.000000
3	30	3.540058e-09	1.435479e-03	800.000000
4	31	1.210947e-09	1.435482e-03	1600.000000
5	32	4.084458e-10	1.435484e-03	3200.000000
6	33	1.370517e-10	1.435485e-03	6400.000000
7	34	4.614675e-11	1.435486e-03	12800.000000
8	35	1.591248e-11	1.435486e-03	25600.000000
9	36	5.662650e-12	<b>1.435487e-03</b>	51200.000000

Новый алгоритм выдал такие результаты



цикл	$S_2$ , погрешности попадания в плоскости	$S_1$ , сдвиги вершин	штраф $K$
0	3.517835e-02	1.615489e-05	0.392949
100	4.049939e-02	8.219693e-06	0.000203
...	...	...	...
2000	5.218465e-03	6.583846e-04	0.123768
2059	3.083940e-04	7.281009e-04	1.975172
2060	2.520618e-04	7.286694e-04	2.360944
2061	1.999617e-04	7.291923e-04	2.890836
2062	1.527278e-04	7.296639e-04	3.646660
2063	1.112085e-04	7.300783e-04	4.777544
2064	7.646352e-05	7.304289e-04	6.564948
2065	4.911293e-05	7.307096e-04	9.552645
2066	2.932750e-05	7.309185e-04	14.878152
2067	1.642868e-05	7.310611e-04	24.922635
2068	8.918421e-06	7.311513e-04	44.499065
2069	4.825920e-06	7.312062e-04	81.982142
2070	2.578748e-06	7.312397e-04	151.516422
2071	1.301561e-06	7.312606e-04	283.563904
2072	5.751982e-07	7.312737e-04	561.833444
2073	1.913991e-07	7.312819e-04	1271.342200
2074	3.378585e-08	7.312870e-04	3820.717979
2075	1.477041e-09	7.312901e-04	21644.771524
2076	7.527428e-12	<b>7.312920e-04</b>	495104.846438

Вывод: ценой увеличения числа шагов в 50 раз получили выигрыш: функционал

$$S_1(x) = \sum_{i=1}^{N-1} (|\Delta x_i|^2 + |\Delta y_i|^2 + |\Delta z_i|^2) \rightarrow \min$$

уменьшается в 2 раза, то есть от 1.435487e-03 до 7.312920e-04

Недостатки у данного алгоритма следующие:

1. Нет гарантий сходимости алгоритма
2. Алгоритм работает слишком долго (слишком много шагов)
3. Часто алгоритм вообще расходится

#### 4.3.3 Сбор статистики о выворачивании

Статистика собиралась следующим образом. Для каждого многогранника 100 раз случайным образом выбиралась вершина. Затем для этой вершины вычислялось расстояние  $\delta$  от нее до ближайшей вершины. После этого случайным образом выбиралось направление (три случайных числа) и оно нормировалось так, чтобы

длина этого вектора была равна  $\delta$ ,  $0.5\delta$ ,  $0.1\delta$ ,  $0.05\delta$ ,  $0.01\delta$ ,  $0.005\delta$ ,  $0.001\delta$ ,  $0.0005\delta$ ,  $0.0001\delta$ , или  $0.00005\delta$ .

Производилась деформация. Если после этого многогранник выворачивался или если алгоритм не давал результата за нужное число шагов, то этот факт регистрировался.

Так делалось для каждой из 100 случайно выбранных вершин многогранника.

Имя многогранника: poly-small

Порядок сдвига	Вероятность "выворачивания"
1.000000	0.770000
0.500000	0.530000
0.100000	0.360000
0.050000	0.230000
0.010000	0.090000
0.005000	0.060000
0.001000	0.000000
0.000500	0.000000
0.000100	0.000000
0.000050	0.000000

Имя многогранника: poly-med

Порядок сдвига	Вероятность "выворачивания"
1.000000	0.720000
0.500000	0.600000
0.100000	0.210000
0.050000	0.150000
0.010000	0.040000
0.005000	0.020000
0.001000	0.010000
0.000500	0.010000
0.000100	0.000000
0.000050	0.010000

Имя многогранника: polyhedron-2010-11-25

Порядок сдвига	Вероятность "выворачивания"
1.000000	0.810000
0.500000	0.740000
0.100000	0.280000
0.050000	0.240000
0.010000	0.090000
0.005000	0.000000
0.001000	0.010000
0.000500	0.000000
0.000100	0.000000
0.000050	0.000000

Имя многогранника: polyhedron-2010-12-19

Порядок сдвига	Вероятность "выворачивания"
1.000000	0.790000
0.500000	0.660000
0.100000	0.190000
0.050000	0.120000
0.010000	0.000000
0.005000	0.000000
0.001000	0.000000
0.000500	0.000000
0.000100	0.000000
0.000050	0.000000

**Вывод.** Чтобы гарантировать корректность многогранника, нужно брать деформацию на 2-3 порядка меньше, чем расстояние до ближайшего соседа.

#### 4.3.4 TODO: Регистрирование выворачиваний

Представляют интерес следующие вопросы:

1. В какой момент, то есть на каком шаге происходит выворачивание?
2. Что при этом происходит, то есть почему нарушается корректность многогранника?
3. Как бороться с выворачиванием? Возможно поменять топологическую структуру многогранника так, чтобы можно было ликвидировать выворачивания?

#### 4.3.5 TODO: Устранение выворачиваний путем слияния вершин

### 4.4 TODO: Анализ сходимости и трудоемкости алгоритма

Про сходимость алгоритма до сих пор ничего не известно, доказать не удалось. (TODO)