

# СОДЕРЖАНИЕ

ВВЕДЕНИЕ	5
1 Аналитическая часть	6
1.1 Постановка задачи . . . . .	6
1.2 Формализация данных . . . . .	8
1.3 Выбор модели данных . . . . .	9
1.3.1 Реляционная модель . . . . .	9
1.3.2 Документно-ориентированная модель . . . . .	10
1.3.3 База данных временных рядов . . . . .	11
1.3.4 Объектно-ориентированная модель . . . . .	11
1.3.5 Графовые базы данных . . . . .	11
1.4 Выбор системы управления базами данных . . . . .	12
1.5 Вывод . . . . .	13
2 Конструкторская часть	14
2.1 Проектирование базы данных . . . . .	14
2.1.1 Таблицы базы данных . . . . .	14
2.1.2 Хранимые процедуры и функции . . . . .	17
2.1.3 Триггеры базы данных . . . . .	18
2.2 Вывод . . . . .	19
3 Технологическая часть	20
3.1 Анализ систем управления базами данных . . . . .	20
3.1.1 Выбор СУБД для решения задачи . . . . .	20
3.2 Выбор средств реализации . . . . .	21
3.3 Детали реализации . . . . .	22
3.3.1 Создание таблиц . . . . .	22
3.3.2 Хранимые процедуры и функции . . . . .	24
3.3.3 Триггеры базы данных . . . . .	27
3.4 Пример работы программы . . . . .	27
3.5 Вывод . . . . .	30

4	Исследовательская часть	31
4.1	Технические характеристики . . . . .	31
4.2	Замеры времени выполнения деформации в зависимости от количества сечений . . . . .	31
4.3	Замеры времени выполнения деформации в зависимости от количества точек . . . . .	32
4.4	Вывод . . . . .	33
	ЗАКЛЮЧЕНИЕ	34
	СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	35

# ВВЕДЕНИЕ

Тонкостенные трубчатые поверхности используются при моделировании объектов в системах автоматизированного проектирования, медицинской визуализации и промышленном дизайне. В процессе их построения возникает задача плавной интерполяции между заданными сечениями после деформации трубки, обеспечивающей непрерывность и гладкость полученной поверхности [1].

Целью работы является разработка базы данных для хранения и управления историей изменения поперечных сечений трубки после деформации.

Для достижения поставленной цели необходимо решить следующие задачи:

- 1 определить информационную модель предметной области и выделить ключевые сущности;
- 2 спроектировать структуру базы данных и установить ограничения целостности данных;
- 3 выбрать СУБД для хранения геометрических данных трубки;
- 4 разработать интерфейс доступа к базе данных;
- 5 исследовать зависимость времени выполнения операции деформации от количества сечений;
- 6 исследовать зависимость времени выполнения операции деформации от количества точек в сечении.

# 1 Аналитическая часть

В данном разделе проводится формализация задачи хранения истории деформаций трубчатых поверхностей и формализация данных, рассматриваются систем управления базами данных и обосновывается выбор решения для поставленной задачи.

## 1.1 Постановка задачи

Дана трубчатая поверхность  $T$ , представляющая собой упорядоченное множество сечений  $S = \{S_1, S_2, \dots, S_m\}$ , соединенных сегментами  $\Gamma$ . Каждое сечение  $S_i$  определяется набором точек  $P_i = \{p_{i1}, p_{i2}, \dots, p_{in_i}\}$  в трехмерном пространстве, где  $p_{ij} = (x_{ij}, y_{ij}, z_i)$ . Сегмент  $\Gamma_k$  между сечениями  $S_k$  и  $S_{k+1}$  содержит множество ребер  $E_k = \{e_{k1}, e_{k2}, \dots, e_{kl}\}$ , соединяющих точки соседних сечений. Конечные точки ребра  $e_{kj}$  не обязательно являются вершинами сечений, а могут лежать на ребрах сечений.

Трубка характеризуется кривой центров  $C = \{c_1, c_2, \dots, c_m\}$ , где центр  $i$ -го сечения определяется как:

$$c_i = \frac{1}{n_i} \sum_{j=1}^{n_i} p_{ij} \quad (1)$$

Требуется реализовать процесс деформации трубки с сохранением истории изменений геометрии. Далее рассмотрены этапы процесса деформации.

### 1. Выбор точки деформации

На кривой центров  $C$  выбирается точка деформации  $p_{def}$  с координатами  $(x_{def}, y_{def}, z_{def})$ .

### 2. Определение целевой точки

На плоскости  $z = z_{def}$  выбирается целевая точка  $p_{target}$  с координатами  $(x_{target}, y_{target}, z_{def})$ , определяющая направление и величину деформации. Вектор смещения вычисляется как:

$$\vec{d} = p_{target} - p_{def} = (x_{target} - x_{def}, y_{target} - y_{def}, 0) \quad (2)$$

### 3. Применение локальной деформации с затуханием

К кривой центров применяется локальная деформация с функцией

затухания. Для каждого центра  $c_i$  вычисляется евклидово расстояние до точки деформации:

$$r_i = \sqrt{(x_i - x_{def})^2 + (y_i - y_{def})^2 + (z_i - z_{def})^2} \quad (3)$$

Вес влияния деформации определяется функцией затухания:

$$w_i(r) = e^{-\frac{r^2}{2\sigma^2}} \quad (4)$$

где  $\sigma$  – радиус влияния деформации.

Новое положение центра  $i$ -го сечения вычисляется по формуле:

$$c'_i = c_i + w_i(r_i) \cdot \alpha \cdot \vec{d} \quad (5)$$

где  $\alpha$  – коэффициент силы деформации,  $0 \leq \alpha \leq 1$ .

Деформированная кривая центров  $C' = \{c'_1, c'_2, \dots, c'_m\}$  образует новую осевую линию трубы.

#### 4. Построение новых сечений

Для каждого деформированного центра  $c'_i$  необходимо построить новое сечение, перпендикулярное касательной к кривой центров в данной точке.

Вектор касательной к кривой в точке  $i$  определяется методом центральных разностей:

$$\vec{t}_i = \begin{cases} \frac{c'_{i+1} - c'_i}{\|c'_{i+1} - c'_i\|}, & i = 1 \\ \frac{c'_i - c'_{i-1}}{\|c'_i - c'_{i-1}\|}, & i = m \\ \frac{c'_{i+1} - c'_{i-1}}{\|c'_{i+1} - c'_{i-1}\|}, & 1 < i < m \end{cases} \quad (6)$$

Плоскость нового сечения определяется уравнением:

$$\vec{t}_i \cdot (P - c'_i) = 0 \quad (7)$$

где  $P = (x, y, z)$  – произвольная точка плоскости.

Точки исходного сечения  $S_i$  проецируются на новую плоскость с сохранением их относительного положения относительно центра.

## 1.2 Формализация данных

Исходя из сформулированных требований, база данных должна содержать информацию о таких объектах, как точка, сечение, ребро, сегмент и трубка.

### 1. Точка

Точка содержит сведения о координатах в трехмерном пространстве  $(x, y, z)$  и порядковом номере внутри сечения.

### 2. Сечение

Сечение состоит из упорядоченного списка точек. Благодаря его упорядоченности, появляется возможность не хранить данные о ребрах сечения, так как ребра образуются посредством соединения точек с соседними индексами. Сечение содержит данные о координатах центра сечения  $(x_{cen}, y_{cen}, z_{cen})$  и порядковом номере сечения в трубке.

### 3. Ребро

Ребро соединяет две точки соседних сечений и содержит сведения о длине ребра и порядковом номере в сегменте.

### 4. Сегмент

Сегмент состоит из упорядоченного списка ребер, соединяющих соседние сечения и содержит сведения о порядковом номере в трубке.

### 5. Трубка

Трубка объединяет сечения и сегменты и хранит общую длину трубки.

При применении операции деформации к трубчатой поверхности изменяются координаты точек сечений, что приводит к необходимости сохранения новой версии геометрии. Для отслеживания истории таких изменений каждый геометрический объект также сохраняет номер версии и ссылки на предыдущую и следующую версии объекта.

На рисунке 1 представлена ER-диаграмма сущностей.

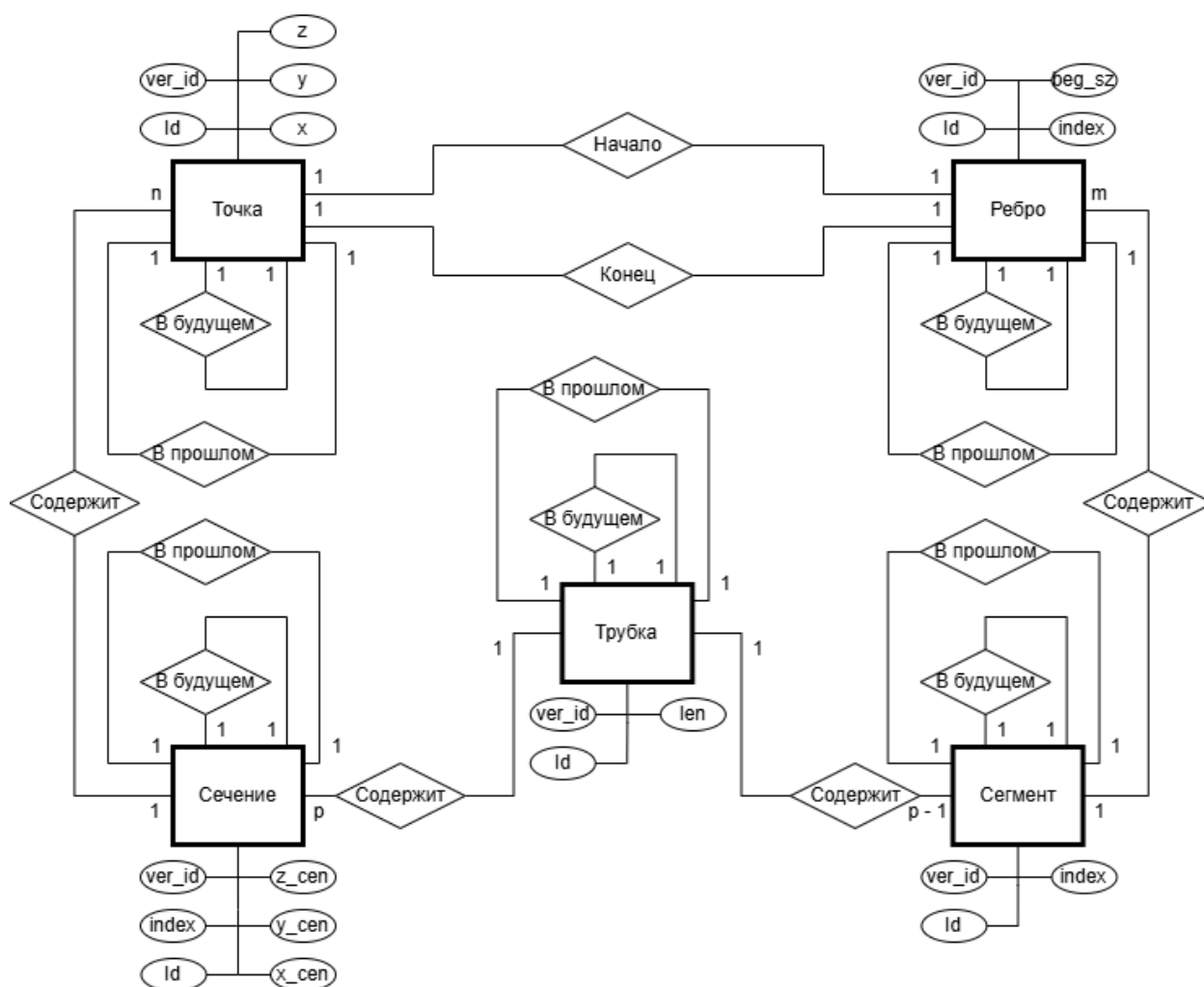


Рисунок 1 – ER-диаграмма сущностей

### 1.3 Выбор модели данных

Модель данных — это совокупность абстракций и методов, с помощью которых мы стремимся имитировать понятия реального мира [10].

#### 1.3.1 Реляционная модель

Реляционные СУБД (PostgreSQL, MySQL, Oracle) основаны на реляционной модели данных, где информация организуется в таблицы, состоящие из строк и столбцов. Каждая таблица имеет определенную структуру, а связи между таблицами устанавливаются с помощью внешних ключей [11].

Реляционные СУБД обеспечивают:

- строгую типизацию данных с контролем на уровне схемы;

- поддержку принципов ACID (атомарность, согласованность, изолированность, долговечность);
- механизмы обеспечения ссылочной целостности через внешние ключи;
- возможность определения сложных ограничений целостности (CHECK, UNIQUE);
- транзакционную модель с поддержкой уровней изоляции [11].

По способу обработки запросов реляционные СУБД делятся на:

- OLTP (строковые) — подходят для систем с частыми изменениями данных;
- OLAP (колоночные) — подходят для частых запросов с объединениями таблиц [11].

### 1.3.2 Документо-ориентированная модель

Документо-ориентированные СУБД (MongoDB, CouchDB) хранят данные в виде документов, обычно в формате JSON или BSON. Каждый документ содержит пары ключ-значение и может иметь вложенную структуру [12].

Документо-ориентированная модель обеспечивает, что:

- документы в одной коллекции могут иметь различную структуру, что позволяет хранить разнородные данные без необходимости изменения схемы базы данных;
- иерархические данные могут быть естественным образом представлены внутри одного документа, что упрощает извлечение связанной информации без операций объединения таблиц;
- высокую скорость чтения относительно других видов СУБД за счет минимизации количества обращений к базе данных;
- значения полей не имеют жесткой привязки к типам данных, что может привести к ошибкам на уровне приложения.



### 1.3.3 База данных временных рядов

БД временных рядов (InfluxDB, TimescaleDB, Prometheus) специализированы для хранения последовательных событий или измерений с временными метками [10].

Баз данных временных рядов обеспечивают, что:

- данные организованы по времени создания;
- устаревшие данные автоматически удаляются на основе правил времени жизни (TTL).

### 1.3.4 Объектно-ориентированная модель

Объектная модель БД (db4o, ObjectDB, Versant) строится на принципах объектно-ориентированного программирования, где данные хранятся в виде объектов с методами [13].

Объектно-ориентированных СУБД обеспечивают:

- естественное отображение объектов — объекты приложения сохраняются непосредственно в базу данных без необходимости преобразования в другую структуру;
- инкапсуляцию — данные и методы объединены в единую сущность, что обеспечивает согласованность бизнес-логики;
- наследование и полиморфизм;
- навигацию по ссылкам — связи между объектами реализуются через прямые ссылки.

### 1.3.5 Графовые базы данных

Графовые СУБД (Neo4j, OrientDB, ArangoDB) специализируются на хранении связей между объектами. Данные представляются в виде узлов (вершин) и связей между ними (ребер), что обеспечивает естественное представление сетевых структур [14].

Графовые базы данных обеспечивают:

- переход между связанными узлами выполняется за константное время независимо от размера графа;
- узлы и ребра могут иметь произвольные свойства, а структура графа может динамически изменяться без модификации схемы;
- Отсутствие JOIN операций.

#### 1.4 Выбор системы управления базами данных

Для хранения истории деформаций трубчатых поверхностей база данных должна обладать следующими свойствами:

- строгая типизация данных — наличие встроенных механизмов контроля типов данных на уровне СУБД;
- ссылочная целостность — поддержка внешних ключей и автоматический контроль связей между таблицами;
- поддержка ACID — гарантии атомарности, согласованности, изолированности и долговечности транзакций;
- ограничения целостности — возможность определения ограничений на уровне схемы;
- каскадные операции — автоматическое распространение операций удаления и обновления на связанные записи.

Таблица 1 – Сравнение моделей данных

Критерий	Реляц.	Докум.	Врем. р.	Объект.	Граф.
Строгая типизация	+	—	+	+	—
Ссылочная целостность	+	—	—	+	—
Поддержка ACID	+	—	—	+	—
Ограничения целостности	+	—	—	+	—
Каскадные операции	+	—	—	—	—

На основании проведенного сравнения для решения поставленной задачи необходимо выбрать реляционную модель СУБД.

## 1.5 Вывод

В данном разделе была проведена формализация задачи хранения истории деформаций трубчатых поверхностей и формализация данных, были рассмотрены системы управления базами данных и был обоснован выбор решения для поставленной задачи.

## 2 Конструкторская часть

В данном разделе проектируется базы данных для хранения истории деформаций трубчатых поверхностей: описывается структура таблиц с типами данных и ограничениями, рассматриваются хранимые процедуры и триггеры.

### 2.1 Проектирование базы данных

#### 2.1.1 Таблицы базы данных

Реализуемая база данных включает пять таблиц, соответствующих основным сущностям предметной области: трубка, сечение, точка, сегмент и ребро. Диаграмма базы данных представлена на рисунке 2.

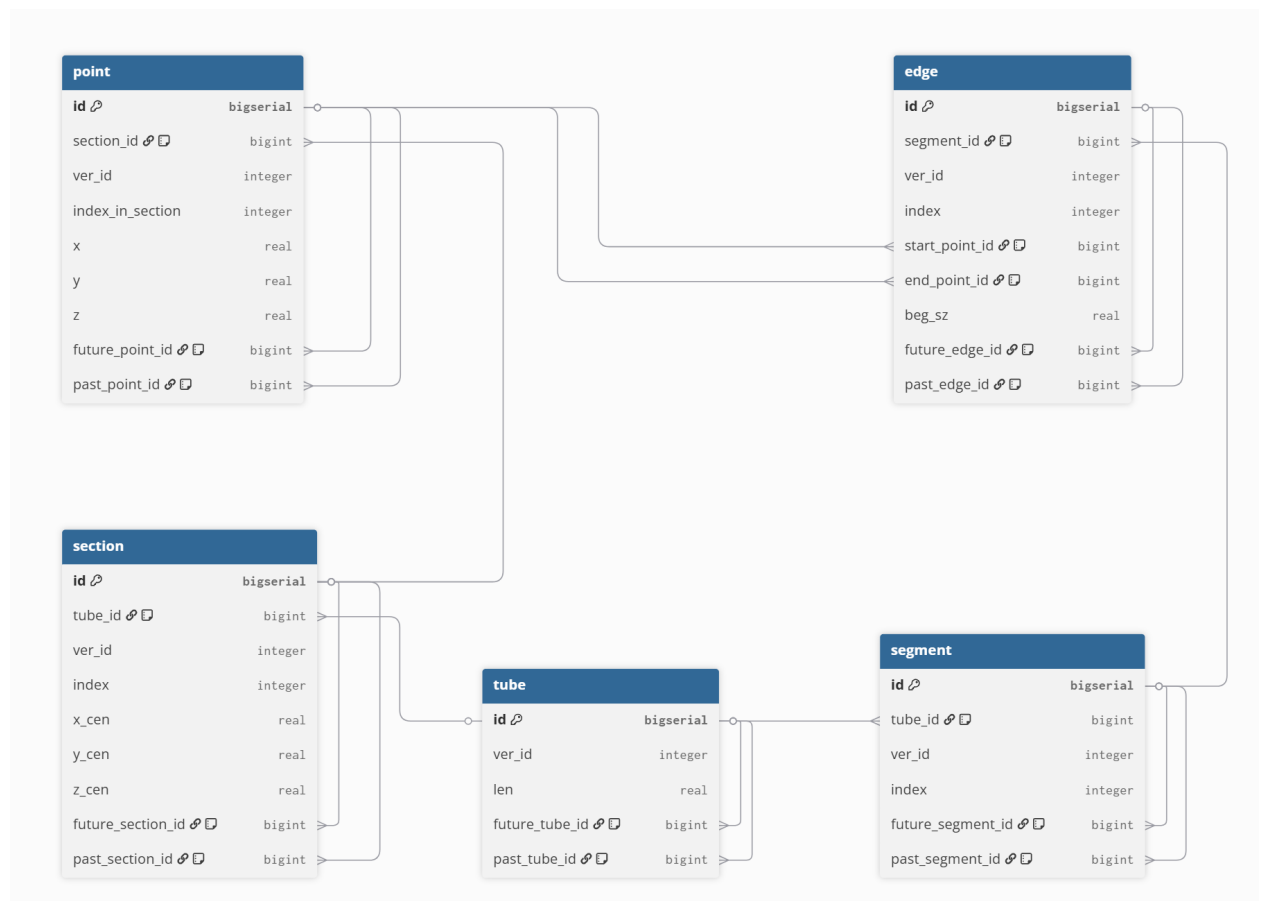


Рисунок 2 – Диаграмма базы данных

1 Таблица **point** содержит информацию о точках сечений и включает следующие поля:

— **id**: bigserial — первичный ключ;

- `section_id`: bigint — ссылка на сечение, внешний ключ на таблицу `section` с каскадным удалением, допускает NULL;
- `ver_id`: integer — номер версии точки;
- `index_in_section`: integer — порядковый номер точки в сечении, допускает NULL;
- `x`: real — координата X точки;
- `y`: real — координата Y точки;
- `z`: real — координата Z точки;
- `future_point_id`: bigint — ссылка на следующую версию точки, внешний ключ на таблицу `point`, допускает NULL;
- `past_point_id`: bigint — ссылка на предыдущую версию точки, внешний ключ на таблицу `point`, допускает NULL.

2 Таблица `section` содержит информацию о сечениях трубки и включает следующие поля:

- `id`: bigserial — первичный ключ;
- `tube_id`: bigint — ссылка на трубку, внешний ключ на таблицу `tube` с каскадным удалением;
- `ver_id`: integer — номер версии сечения;
- `index`: integer — порядковый номер сечения в трубке;
- `x_cen`: real — координата X центра сечения;
- `y_cen`: real — координата Y центра сечения;
- `z_cen`: real — координата Z центра сечения;
- `future_section_id`: bigint — ссылка на следующую версию сечения, внешний ключ на таблицу `section`, допускает NULL;
- `past_section_id`: bigint — ссылка на предыдущую версию сечения, внешний ключ на таблицу `section`, допускает NULL.

3 Таблица `edge` содержит информацию о ребрах, соединяющих точки соседних сечений, и включает следующие поля:

- `id`: bigserial — первичный ключ;

- `segment_id`: bigint — ссылка на сегмент, внешний ключ на таблицу `segment` с каскадным удалением;
- `ver_id`: integer — номер версии ребра;
- `index`: integer — порядковый номер ребра в сегменте;
- `start_point_id`: bigint — ссылка на начальную точку ребра, внешний ключ на таблицу `point` с каскадным удалением;
- `end_point_id`: bigint — ссылка на конечную точку ребра, внешний ключ на таблицу `point` с каскадным удалением;
- `beg_sz`: real — длина ребра;
- `future_edge_id`: bigint — ссылка на следующую версию ребра, внешний ключ на таблицу `edge`, допускает NULL;
- `past_edge_id`: bigint — ссылка на предыдущую версию ребра, внешний ключ на таблицу `edge`, допускает NULL.

4 Таблица `segment` содержит информацию о сегментах, соединяющих соседние сечения, и включает следующие поля:

- `id`: bigserial — первичный ключ;
- `tube_id`: bigint — ссылка на трубку, внешний ключ на таблицу `tube` с каскадным удалением;
- `ver_id`: integer — номер версии сегмента;
- `index`: integer — порядковый номер сегмента в трубке;
- `future_segment_id`: bigint — ссылка на следующую версию сегмента, внешний ключ на таблицу `segment`, допускает NULL;
- `past_segment_id`: bigint — ссылка на предыдущую версию сегмента, внешний ключ на таблицу `segment`, допускает NULL.

5 Таблица `tube` содержит информацию о трубчатых поверхностях и включает следующие поля:

- `id`: bigserial — первичный ключ;
- `ver_id`: integer — номер версии трубки;
- `len`: real — общая длина трубки;

- `future_tube_id`: `bigint` — ссылка на следующую версию трубки, внешний ключ на таблицу `tube`, допускает `NULL`;
- `past_tube_id`: `bigint` — ссылка на предыдущую версию трубки, внешний ключ на таблицу `tube`, допускает `NULL`.

### 2.1.2 Хранимые процедуры и функции

Функция `calculate_tube_length_on_segment()` вычисляет длину трубки на основе максимальных значений `beg_sz` ребер в каждом сегменте (рисунок 3).

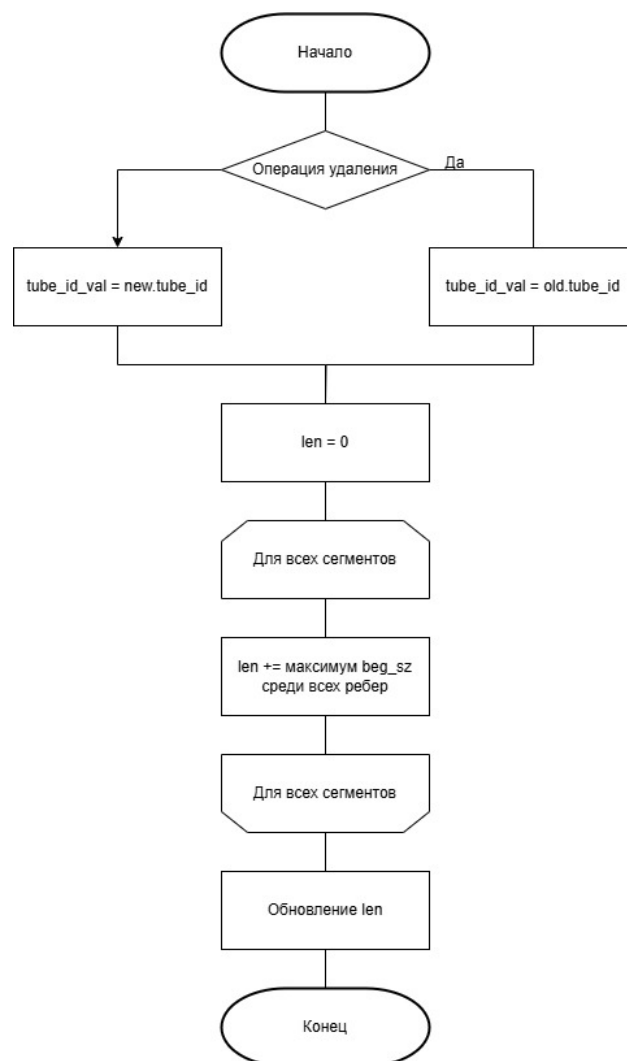


Рисунок 3 – Алгоритм функции `calculate_tube_length_on_segment`

Функция `calculate_tube_length_on_edge()` выполняет аналогичные вычисления при изменении ребер (рисунок 4).

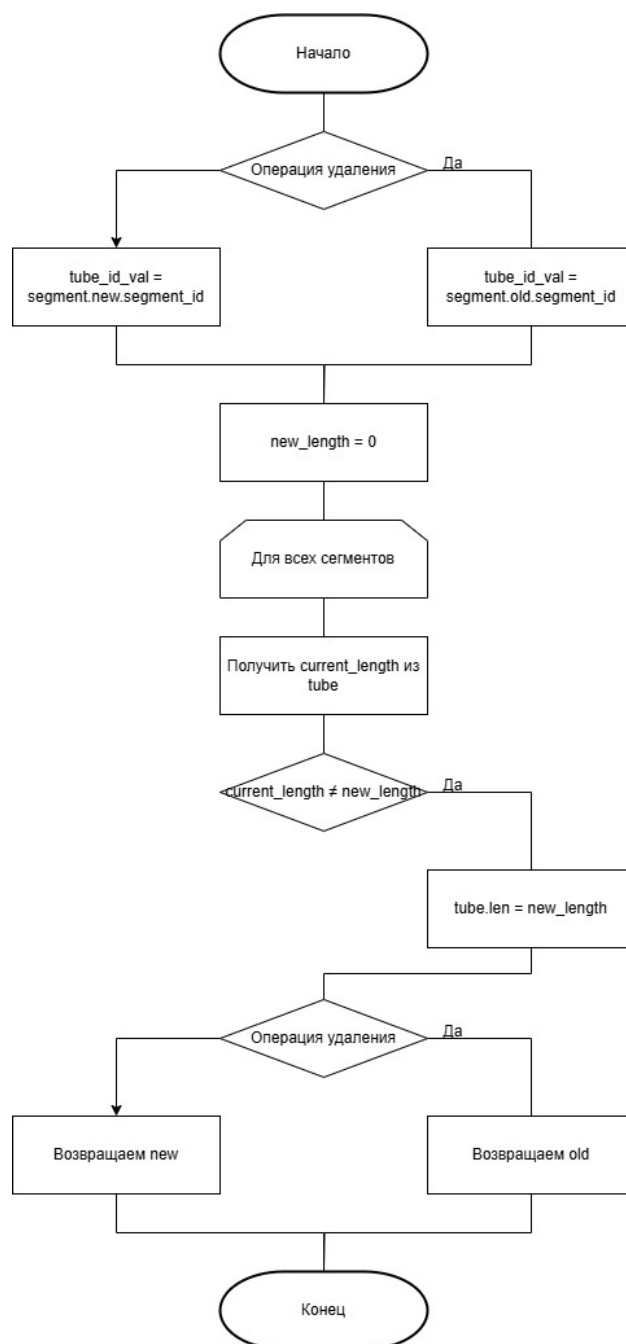


Рисунок 4 – Алгоритм функции `calculate_tube_length_on_edge`

### 2.1.3 Триггеры базы данных

Триггер `update_tube_length_on_segment` срабатывает после операций `INSERT`, `UPDATE` или `DELETE` на таблице `segment` и вызывает функцию `calculate_tube_length_on_segment()` для автоматического пересчёта длины трубки при изменении состава сегментов (рисунок 5).

Триггер `update_tube_length_on_edge` активируется после операций `INSERT`, `UPDATE` поля `beg_sz` или `DELETE` на таблице `edge`, вызывая функцию `calculate_tube_length_on_edge()` для поддержания актуаль-



ности длины трубки при модификации ребер (рисунок 5).

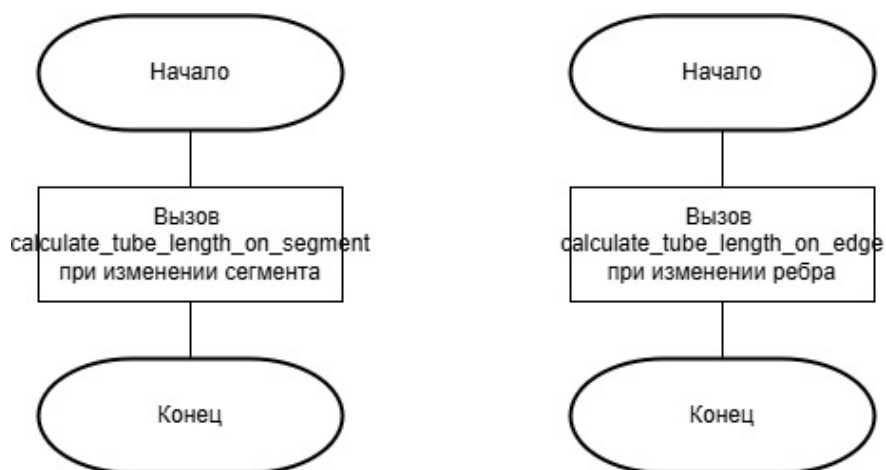


Рисунок 5 – Алгоритм триггера `update_tube_length_on_segment` (слева) и триггера `update_tube_length_on_edge` (справа)

## 2.2 Вывод

В данном разделе была спроектирована база данных для хранения истории деформаций трубчатых поверхностей: описаны пять таблиц со спецификацией типов данных и ограничений целостности, рассмотрены основные хранимые процедуры, функции и триггеры.

## 3 Технологическая часть

В данном разделе проводится анализ систем управления базами данных и выбирается СУБД для решения поставленной задачи, обосновываются средства реализации приложения, приводятся детали реализации базы данных и представляется пример работы программы.

### 3.1 Анализ систем управления базами данных

Для хранения истории деформаций трубчатых поверхностей была выбрана реляционная база данных. Были рассмотрены СУБД для работы с ними [11]:

- PostgreSQL — объектно-реляционная СУБД с открытым исходным кодом, соответствующая стандартам SQL [11]. Обеспечивает строгую типизацию данных, поддержку внешних ключей с каскадными операциями, сложных ограничений целостности (CHECK, UNIQUE), триггеров и хранимых процедур. Поддерживает механизм MVCC (многоверсионное управление конкурентным доступом).
- MySQL — реляционная СУБД, разрабатываемая корпорацией Oracle [8]. Имеет ограниченную поддержку механизмов ограничения целостности. Не поддерживает внешние ключи и транзакции. Не соответствует стандартам SQL [8].
- Oracle Database — коммерческая объектно-реляционная СУБД, обеспечивающая безопасность и масштабируемость [7]. Полная функциональность доступна только в платных версиях.

#### 3.1.1 Выбор СУБД для решения задачи

Для выбора оптимальной СУБД были определены критерии сравнения, основанные на требованиях поставленной задачи:

- K1 — полная поддержка ACID-транзакций для обеспечения согласованности данных при сохранении версий;
- K2 — поддержка внешних ключей с каскадными операциями для автоматического поддержания связей между таблицами;

- К3 — поддержка ограничений целостности (CHECK, UNIQUE) для валидации геометрических данных;
- К4 — бесплатное использование для академических и коммерческих целей.

Результаты сравнения СУБД по выбранным критериям приведены в таблице 2.

Таблица 2 – Сравнение СУБД по выбранным критериям

СУБД	K1	K2	K3	K4
PostgreSQL	+	+	+	+
MySQL	—	—	—	+
Oracle Database	+	+	+	—

По результатам сравнения для решения поставленной задачи выбрана СУБД PostgreSQL.

### 3.2 Выбор средств реализации

Для разработки программного обеспечения взаимодействия с базой данных в качестве основного языка программирования был выбран C++ [2]. Обоснование выбора:

- объектно-ориентированный подход позволяет моделировать иерархию объектов (трубка, сечение, точка);
- наличие библиотек для работы с PostgreSQL;
- поддержка библиотеки шаблонов для работы с коллекциями данных;
- возможность низкоуровневой оптимизации памяти и вычислений.

Для графического интерфейса используется фреймворк Qt [5], обеспечивающий:

- кроссплатформенность разработки;
- интеграцию с OpenGL [9] для визуализации трехмерных объектов;
- набор виджетов для создания пользовательского интерфейса.

Для замеров времени использовался класс `QElapsedTimer` из библиотеки QT [5]. Для визуализации трехмерных графиков был использован Python [3], а именно библиотека `matplotlib` [4].

Для взаимодействия с базой данных PostgreSQL выбрана библиотека Qt SQL [6] — библиотека, предоставляющая:

- интерфейс для работы с соединениями и транзакциями;
- параметризованные запросы для защиты от SQL-инъекций;
- интеграцию с фреймворком Qt;
- поддержку подготовленных выражений для повышения производительности.

### 3.3 Детали реализации

#### 3.3.1 Создание таблиц

Создание таблицы `point` для хранения точек сечений представлено на листинге 1.

```
1 create table point (  
2     id bigint primary key,  
3     section_id bigint references section(id) on delete cascade,  
4     ver_id integer not null,  
5     index_in_section integer,  
6     x real not null,  
7     y real not null,  
8     z real not null,  
9     future_point_id bigint references point(id) on delete set  
10        null,  
11     past_point_id bigint references point(id) on delete set  
12        null,  
13     constraint unique_point_in_section  
14         unique (section_id, index_in_section)  
15 );
```

Листинг 1 – Создание таблицы `point`

Создание таблицы `section` для хранения сечений трубки представлено на листинге 2.

```

1 create table section (
2     id bigserial primary key,
3     tube_id bigint not null references tube(id) on delete
4         cascade,
5     ver_id integer not null,
6     index integer not null,
7     x_cen real not null,
8     y_cen real not null,
9     z_cen real not null,
10    future_section_id bigint references section(id) on delete
11        set null,
12    past_section_id bigint references section(id) on delete set
13        null,
14    constraint unique_section_in_tube unique (tube_id, index)
15 );

```

Листинг 2 – Создание таблицы section

Создание таблицы `edge` для хранения ребер между точками представлено на листинге 3.

```

1 create table edge (
2     id bigserial primary key,
3     segment_id bigint not null references segment(id)
4         on delete cascade,
5     ver_id integer not null,
6     index integer not null,
7     start_point_id bigint not null references point(id)
8         on delete cascade,
9     end_point_id bigint not null references point(id)
10        on delete cascade,
11     beg_sz real not null,
12     future_edge_id bigint references edge(id) on delete set
13        null,
14     past_edge_id bigint references edge(id) on delete set null,
15     constraint unique_edge_in_segment unique (segment_id, index
16         ),
17     constraint different_points
18         check (start_point_id != end_point_id)
19 );

```

Листинг 3 – Создание таблицы edge

Создание таблицы **segment** для хранения сегментов между сечениями представлено на листинге 4.

```
1 create table segment (  
2     id bigserial primary key,  
3     tube_id bigint not null references tube(id) on delete  
4         cascade,  
5     ver_id integer not null,  
6     index integer not null,  
7     future_segment_id bigint references segment(id) on delete  
8         set null,  
9     past_segment_id bigint references segment(id) on delete set  
10        null,  
11    constraint unique_segment_in_tube unique (tube_id, index)  
12 );
```

Листинг 4 – Создание таблицы segment

Создание таблицы **tube** для хранения информации о трубках представлено на листинге 5.

```
1 create table tube (  
2     id bigserial primary key,  
3     ver_id integer not null,  
4     len real not null,  
5     future_tube_id bigint references tube(id) on delete set  
6         null,  
7     past_tube_id bigint references tube(id) on delete set null  
8 );
```

Листинг 5 – Создание таблицы tube

### 3.3.2 Хранимые процедуры и функции

Создание функции для пересчета длины трубки при изменениях в **segment** 6.

```
1 create or replace function calculate_tube_length_on_segment()  
2 returns trigger as $$  
3 declare  
4     tube_id_val bigint;  
5     new_length real;
```

```

6      current_length real;
7  begin
8      if tg_op = 'DELETE' then
9          tube_id_val := old.tube_id;
10     else
11         tube_id_val := new.tube_id;
12     end if;
13
14     select coalesce(sum(max_edge_length), 0) into new_length
15     from (
16         select s.id, max(e.beg_sz) as max_edge_length
17         from segment s
18         left join edge e on e.segment_id = s.id
19         where s.tube_id = tube_id_val
20         group by s.id
21     ) as segment_lengths;
22
23     select len into current_length
24     from tube
25     where id = tube_id_val;
26
27     if current_length is distinct from new_length then
28         update tube
29         set len = new_length
30         where id = tube_id_val;
31     end if;
32
33     if tg_op = 'DELETE' then
34         return old;
35     else
36         return new;
37     end if;
38 end;
39 $$ language plpgsql;

```

Листинг 6 – Функция для пересчета длины трубки при изменениях в segment

Создание функции для пересчета длины трубки при изменениях в edge 7.

```

1 create or replace function calculate_tube_length_on_edge()

```

```

2 returns trigger as $$
3 declare
4     tube_id_val bigint;
5     new_length real;
6     current_length real;
7 begin
8     if tg_op = 'DELETE' then
9         select tube_id into tube_id_val
10        from segment
11        where id = old.segment_id;
12    else
13        select tube_id into tube_id_val
14        from segment
15        where id = new.segment_id;
16    end if;
17
18    select coalesce(sum(max_edge_length), 0) into new_length
19    from (
20        select s.id, max(e.beg_sz) as max_edge_length
21        from segment s
22        left join edge e on e.segment_id = s.id
23        where s.tube_id = tube_id_val
24        group by s.id
25    ) as segment_lengths;
26
27    select len into current_length
28    from tube
29    where id = tube_id_val;
30
31    if current_length is distinct from new_length then
32        update tube
33        set len = new_length
34        where id = tube_id_val;
35    end if;
36
37    if tg_op = 'DELETE' then
38        return old;
39    else
40        return new;
41    end if;
42 end;
43 $$ language plpgsql;

```



---

Листинг 7 – Функция для пересчета длины трубки при изменениях в edge

### 3.3.3 Триггеры базы данных

Создание триггера для пересчета длины трубки при изменениях в segment 8.

```
1 create trigger update_tube_length_on_segment
2 after insert or update or delete on segment
3 for each row
4 execute function calculate_tube_length_on_segment();
```

Листинг 8 – Триггер для пересчета длины трубки при изменениях в segment

Создание триггера для пересчета длины трубки при изменениях в edge 9.

```
1 create trigger update_tube_length_on_edge
2 after insert or update of beg_sz or delete on edge
3 for each row
4 execute function calculate_tube_length_on_edge();
```

Листинг 9 – Триггер для пересчета длины трубки при изменениях в edge

## 3.4 Пример работы программы

Интерфейс программы включает следующие основные элементы:

- главное окно с трехмерным просмотром трубки;
- панель управления сечениями;
- панель настройки параметров деформации;
- панель истории версий с навигацией;
- диалоговые окна для сохранения и загрузки данных.

Разработанное программное обеспечение предоставляет следующий функционал:

## 1 Создание и сохранение трубки:

- построение трубчатой поверхности на основе заданных сечений;
- сохранение геометрии трубки в базу данных с присвоением версии;
- автоматическое сохранение всех сечений, точек, сегментов и ребер.

## 2 Применение деформации:

- выбор точки деформации на кривой центров трубки (рисунок 6);

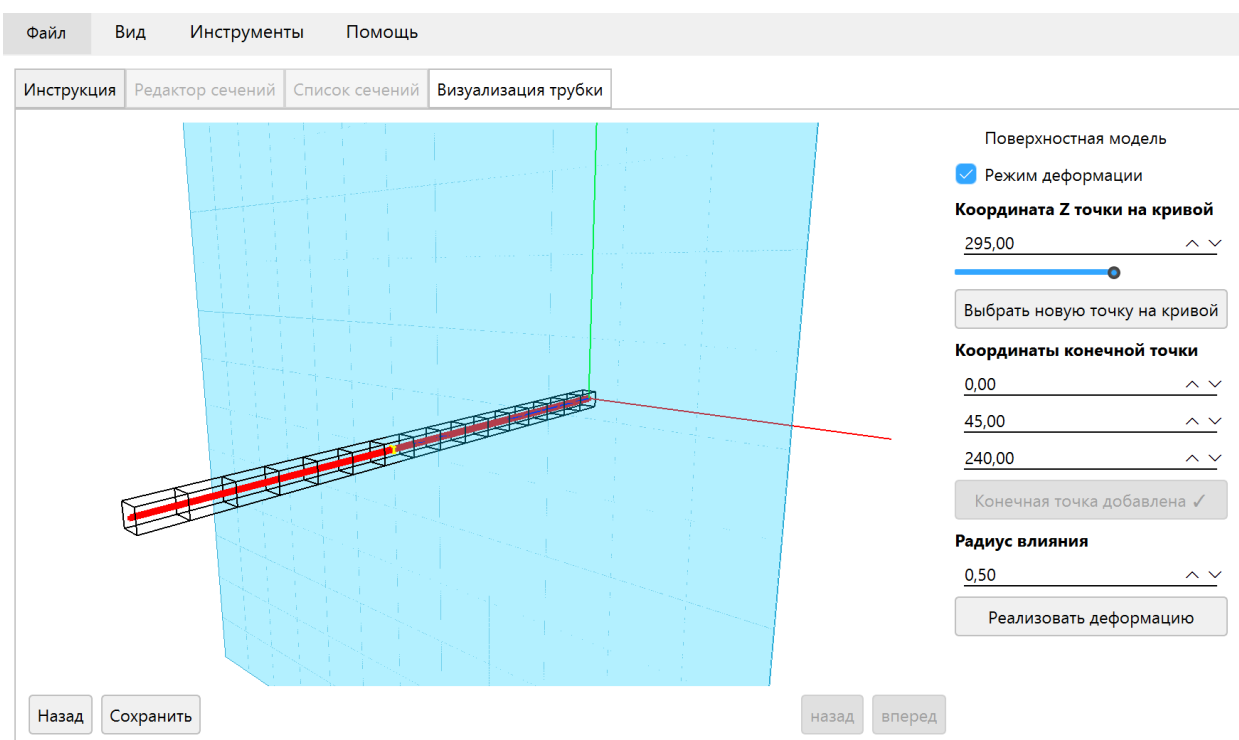


Рисунок 6 – Выбор точки деформации

- указание целевой позиции для перемещения выбранной точки;
- настройка радиуса влияния и функции затухания деформации;
- визуализация деформированной кривой центров в режиме предварительного просмотра (рисунок 7);

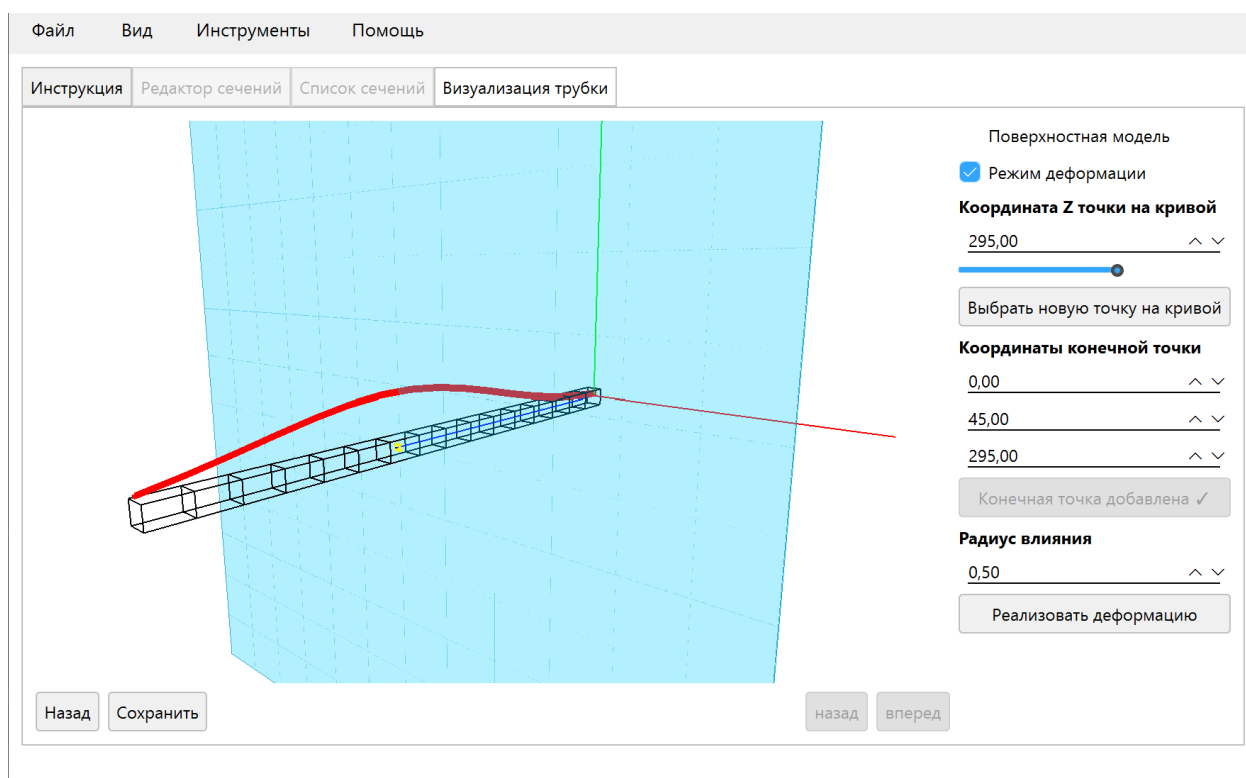


Рисунок 7 – Визуализация деформированной кривой центров

— применение деформации с автоматическим пересчетом геометрии (рисунок 8).

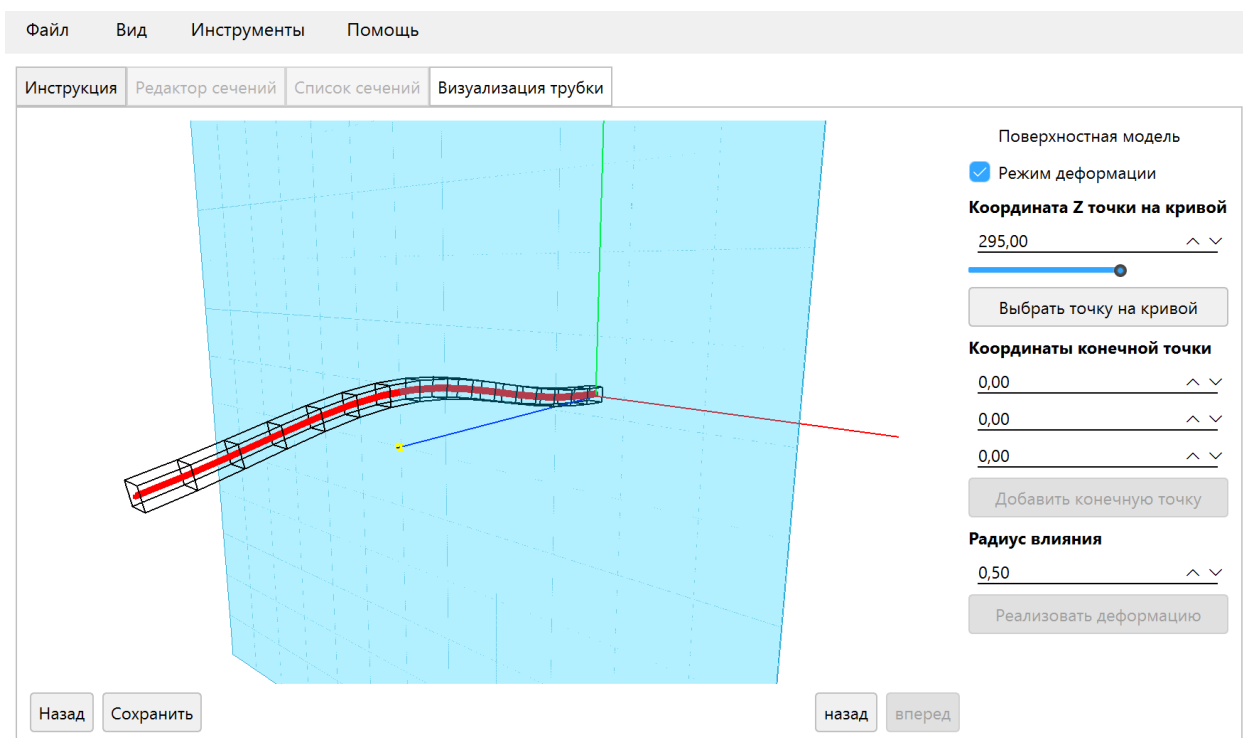


Рисунок 8 – Применение деформации

### 3 Просмотр истории изменений.

#### 3.5 Вывод

В данном разделе проведен анализ систем управления базами данных, в результате которого для решения поставленной задачи выбрана СУБД PostgreSQL, обоснован выбор языка программирования C++ и библиотеки Qt SQL для взаимодействия с базой данных, приведены детали реализации и представлен пример работы программы.

## 4 Исследовательская часть

В данном разделе сравниваются временные характеристики построения деформированной трубки в зависимости от количества сечений и количества точек в каждом сечении.

Замеры для нахождения зависимости времени от количества сечений проводились для трубок с количеством сечений от 10 до 50. В каждом сечении было от 25 точек. Замеры для нахождения зависимости времени от количества точек проводились для трубок с количеством точек от 10 до 100. В трубке было 20 сечений. Деформация происходила по центру трубки с одинаковым радиусом влияния. Все замеры проводились 100 раз и в таблицу заносилось среднее арифметическое значение времени.

### 4.1 Технические характеристики

Тестирование программы проводилось на устройстве с операционной системой Windows 11, оснащенном процессором Intel Core i7-1260p, оперативной памятью объемом 16 Гб и видеокартой Intel Iris Xe Graphics. Во время тестирования, компьютер был подключен к сети питания и не использовался для других задач.

### 4.2 Замеры времени выполнения деформации в зависимости от количества сечений

Таблица 3 – Зависимость времени построения деформированной трубки от количества сечений (в миллисекундах)

Количество сечений	Время построения (мс)
10	981.263
20	1196.770
30	1342.660
40	1587.540
50	1845.920

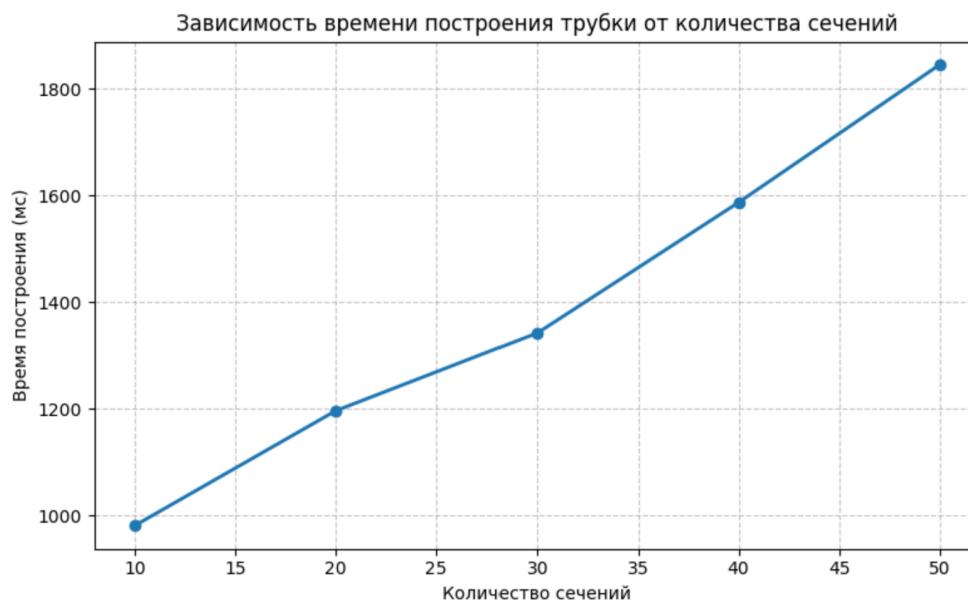


Рисунок 9 – График зависимости времени построения деформированной трубки от количества сечений

#### 4.3 Замеры времени выполнения деформации в зависимости от количества точек

Таблица 4 – Зависимость времени построения деформированной трубки от количества точек в сечении (в миллисекундах)

Количество точек	Время построения (мс)
10	992.191
25	1196.770
50	1345.230
75	1462.880
100	1578.440

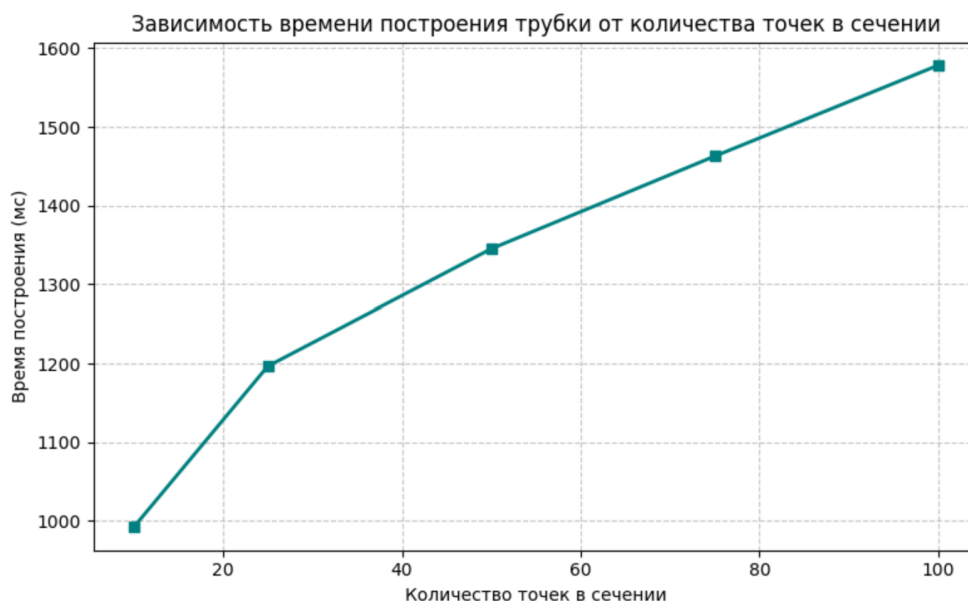


Рисунок 10 – График зависимости времени построения деформированной трубки от количества точек

#### 4.4 Вывод

В данном разделе были сравнены временные характеристики построения деформированной трубки в зависимости от количества сечений и количества точек в каждом сечении.

Полученные результаты показывают, что время построения деформированной трубки линейно возрастает как с увеличением количества сечений, так и с ростом числа точек в каждом сечении.

## ЗАКЛЮЧЕНИЕ

В рамках данной работы была:

- 1 определена информационная модель предметной области;
- 2 спроектирована структура базы данных и установлены ограничения целостности данных, обеспечивающие корректность хранения геометрической информации;
- 3 выбрана система управления базами данных;
- 4 разработан интерфейс доступа к базе данных, позволяющий выполнять основные операции с геометрическими объектами;
- 5 исследована зависимость времени выполнения операции деформации от количества сечений;
- 6 исследована зависимость времени выполнения операции деформации от количества точек в сечении.

Проведенные исследования показали, что время построения деформированной трубки возрастает линейно как с увеличением количества сечений, так и с ростом числа точек в каждом сечении. Это свидетельствует о линейной зависимости вычислительной сложности от объема входных данных.



## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] Иванов В.Н. Геометрия и конструирование трубчатых оболочек — Вестник Российского университета дружбы народов, 2005. — С.109-114.
- [2] Standart C++ — Электронный ресурс. — Режим доступа:<https://isocpp.org> (дата обращения: 14.06.2025)
- [3] Python — Электронный ресурс. — Режим доступа:<https://www.python.org> (дата обращения: 14.06.2025)
- [4] Matplotlib: Visualization with Python — Электронный ресурс. — Режим доступа:<https://matplotlib.org> (дата обращения: 14.06.2025)
- [5] QT|Cross-platform software development for embedded&desktop — Электронный ресурс. — Режим доступа:<https://www.qt.io> (дата обращения: 14.06.2025)
- [6] QT SQL documentation&desktop — Электронный ресурс. — Режим доступа:<https://doc.qt.io/qt-6/sql-programming.html> (дата обращения: 14.06.2025)
- [7] Oracle help center&desktop — Электронный ресурс. — Режим доступа:<https://docs.oracle.com/en> (дата обращения: 14.06.2025)
- [8] MySQL help center&desktop — Электронный ресурс. — Режим доступа:<https://dev.mysql.com/doc> (дата обращения: 14.06.2025)
- [9] OpenGL — Электронный ресурс. — Режим доступа:<https://www.opengl.org> (дата обращения: 14.06.2025)
- [10] Комаров В. И. Путеводитель по базам данных — ДМК-Пресс, 2024, с. 21-58.
- [11] Васильева К. Н., Хусаинова Г. Я. Реляционные базы данных — СФ БашГУ, 2019, с. 22-23.
- [12] Лучинина З. С., Сидоркина И. Г. МАТЕМАТИЧЕСКАЯ МОДЕЛЬ ДОКУМЕНТНО-ОРИЕНТИРОВАННОЙ БАЗЫ ДАННЫХ — Вестник Чувашского университета, 2015, с. 174-179.

- [13] Эльдарханов А. М. Обзор моделей данных объектно-ориентированных СУБД — Труды Института системного программирования РАН, 2011, с. 205-224.
- [14] Отраднов К. К., Алёшкин А. С., Калинин В. Н. ПРЕИМУЩЕСТВА ИСПОЛЬЗОВАНИЯ ГРАФОВЫХ БАЗ ДАННЫХ ПРИ РАЗРАБОТКЕ ПРИКЛАДНЫХ ПРИЛОЖЕНИЙ — Вестник РГРТУ, 2023, с. 73-83.