Проект по моделированию деформированного листа бумаги
Белоковаленко Игорь, Ахметов Кирилл Б05-008

Введение

Данный отчёт является результатом работы по созданию математической модели - алгоритма, моделирующего лист бумаги при различных деформациях. Перед тем как приступить к технической и основной части проекта, я бы хотел посвятить пару слов постановке задачи.

Целью проекта является создание программы, которая должна на вход получать некоторые данные о деформации, а на выходе возвращать пользователю некоторый набор параметров, который будет описывать лист бумаги. Давайте произведём декомпозицию задачи на базовые подзадачи. Проект будет состоять из нескольких частей, подробное описание которых приведено далее:

1. Набор инструментов аналитической геометрии;
2. Создание математической модели листа и серии методов, вносящих в неё изменения.

Набор инструментов аналитической геометрии

Метод `Point::get_mid`

Этот метод находит промежуточную точку на отрезке между текущей точкой и другой точкой `another`, взвешенную коэффициентом `t`.

$$\text{Point}(x + (another.x - x) \cdot t, y + (another.y - y) \cdot t, z + (another.z - z) \cdot t)$$

где t — параметр в диапазоне от 0 до 1, который определяет положение возвращаемой точки на отрезке.

Метод `Point::get_dist`

Этот метод вычисляет Евклидово расстояние между текущей точкой и точкой `another`.

$$\sqrt{(x - another.x)^2 + (y - another.y)^2 + (z - another.z)^2}$$

Метод `Point::get_angle`

Метод вычисляет угол (в градусах) между двумя векторами, один из которых идет от текущей точки к точке `first`, а другой — к точке `second`.

$$\theta = \arccos\left(\frac{(b^2 + c^2 - a^2)}{2bc}\right) \cdot \frac{180}{\pi}$$

где:

- a — расстояние между точками `first` и `second`,
- b — расстояние между текущей точкой и `second`,
- c — расстояние между текущей точкой и `first`.

Функция `crossProduct`

Функция возвращает векторное произведение двух векторов, определяемых точками `first` и `second`.

$\text{Point}(first.y \cdot second.z - first.z \cdot second.y, first.z \cdot second.x - first.x \cdot second.z, first.x \cdot second.y - first.y \cdot second.x)$

Функция `dotProduct`

Функция вычисляет скалярное произведение двух векторов, определяемых точками `first` и `second`.

$$first.x \cdot second.x + first.y \cdot second.y + first.z \cdot second.z$$

Функция `onSegment`

Функция `onSegment` проверяет, лежит ли точка `q` на отрезке, образованном точками `p` и `r`. Для этого метод учитывает координаты всех трех точек в трехмерном пространстве. Проверяется, выполнение следующих условий одновременно для координат x, y, и z:

1. Значение координаты x точки `q` должно находиться между значениями x точек `p` и `r`.
2. Значение координаты y точки `q` должно находиться между значениями y точек `p` и `r`.
3. Значение координаты z точки `q` должно находиться между значениями z точек `p` и `r`.

Формально, условие для координаты x можно записать как:

$$\min(p.x, r.x) \leq q.x \leq \max(p.x, r.x)$$

Аналогично, условия для координат y и z:

$$\min(p.y, r.y) \leq q.y \leq \max(p.y, r.y)$$

$$\min(p.z, r.z) \leq q.z \leq \max(p.z, r.z)$$

Если все три условия выполняются, метод возвращает `true`, указывая на то, что точка `q` действительно лежит на отрезке между точками `p` и `r`. В противном случае возвращается `false`.

Функция `orientation`

Определяет ориентацию трех точек `p`, `q`, и `r`. Результатом является одно из следующих значений:

- 0 — если точки коллинеарны,
- 1 — если точки расположены по часовой стрелке,
- 2 — если точки расположены против часовой стрелки.

Ориентация определяется по значению следующего выражения:

$$val = (q.y - p.y) \times (r.x - q.x) - (q.x - p.x) \times (r.y - q.y)$$

Если $val = 0$, точки коллинеарны. Если $val > 0$, точки расположены по часовой стрелке. В противном случае — против часовой стрелки.

Функция `doIntersect`

Проверяет, пересекаются ли два отрезка, заданные парами точек $(p1, q1)$ и $(p2, q2)$. При помощи следующего алгоритма:

1. Вычислить ориентацию четырех пар точек: $(p1, q1, p2)$, $(p1, q1, q2)$, $(p2, q2, p1)$, и $(p2, q2, q1)$, что дает значения $o1$, $o2$, $o3$, и $o4$.
2. Если ориентации $o1$ и $o2$ различны, а также $o3$ и $o4$ различны, отрезки пересекаются.
3. Если отрезки коллинеарны ($o1 = 0$ или $o2 = 0$ или $o3 = 0$ или $o4 = 0$) и любая из точек отрезка лежит на другом отрезке, считается, что отрезки пересекаются.
4. Во всех остальных случаях отрезки не пересекаются.

Функция `distance`

Вычисляет Евклидово расстояние между двумя точками `first` и `second`.

$$\text{distance} = \sqrt{(first.x - second.x)^2 + (first.y - second.y)^2 + (first.z - second.z)^2}$$

Функция `rotatePointAroundAxis`

Функция `rotatePointAroundAxis` вращает точку вокруг заданной оси на определенный угол. Предполагается, что ось вращения задается двумя точками `lineStart` и `lineEnd`, а угол вращения задается в градусах.

Алгоритм вращения

1. Перевод угла вращения из градусов в радианы:

$$\theta = \frac{\text{angleDegrees} \cdot \pi}{180}$$

2. Вычисление единичного вектора оси вращения $\vec{a}$ как:

$$\vec{a} = \frac{\vec{lineEnd} - \vec{lineStart}}{\|\vec{lineEnd} - \vec{lineStart}\|}$$

3. Перенос начала координат к `lineStart` и представление координат точки `point` в новой системе координат:

$$\vec{p} = \vec{point} - \vec{lineStart}$$

4. Применение матрицы вращения Родрига для получения координат вращенной точки $\vec{p'}$:

$$\vec{p'} = \cos(\theta)\vec{p} + (1 - \cos(\theta))(\vec{a} \cdot \vec{p})\vec{a} + \sin(\theta)(\vec{a} \times \vec{p})$$

   Здесь $\vec{a} \cdot \vec{p}$ обозначает скалярное произведение, а $\vec{a} \times \vec{p}$ — векторное произведение.

5. Возвращение координат точки из новой системы координат в исходную:

$$\vec{newPoint} = \vec{p'} + \vec{lineStart}$$

Математическое представление

Данное вращение можно представить через матрицу вращения Родрига $\mathbf{R}$, которая используется для вращения вектора $\vec{p}$ вокруг оси $\vec{a}$:

$$\mathbf{R} = \cos(\theta)\mathbf{I} + (1 - \cos(\theta))\vec{a}\vec{a}^T + \sin(\theta)\mathbf{A}_{\vec{a}}$$

где $\mathbf{I}$ — единичная матрица, а $\mathbf{A}_{\vec{a}}$ — матрица, полученная из вектора $\vec{a}$ для векторного произведения.

Математическая модель листа бумаги

В качестве переменных, которые будут исследуемую сущность, были выбраны:

- Набор координат, описывающий каждую вершину листа бумаги.
- Набор пар индексов вершин, между которыми проведены рёбра.
- Набор пар индексов вершин, между которыми проведены линии сгиба.

Далее давайте опишем все методы соответствующего класса, которые применяются для того, чтобы вносить изменения в нашу модель.

Метод `Paper::addEdge`

Принимает два целочисленных аргумента: startIdx и endIdx, которые представляют индексы начальной и конечной точек ребра соответственно. Эти индексы относятся к вектору вершин (точек), представляющих геометрическую форму бумаги. Метод добавляет в список edges новый элемент, который представляет ребро, соединяющее указанные точки. Это ребро хранится в виде пары индексов, указывающих на начало и конец ребра в векторе вершин.

Метод `Paper::creaseIntersection`

Определяет, пересекаются ли два сгиба (линии) на бумаге. Каждый сгиб представлен парой индексов его начальной и конечной точек. Метод использует функцию doIntersect, которая реализует алгоритм определения пересечения двух отрезков. Если отрезки (сгибы) пересекаются, функция возвращает true, в противном случае — false. Этот метод важен для определения взаимного расположения сгибов на листе бумаги и используется для проверки корректности сложения бумаги.

Метод `Paper::isPointOnEdge`

Проверяет, лежит ли заданная точка на определённом ребре. Для этого метод сначала вычисляет вектора от начальной и конечной точек ребра к заданной точке. Затем вычисляется векторное произведение этих двух векторов. Если результат векторного произведения не равен нулевому вектору, это означает, что точка не лежит на линии, образованной ребром, так как векторы не коллинеарны.

Дополнительно, метод проверяет, находится ли точка между начальной и конечной точками ребра посредством скалярных произведений векторов от начальной и конечной точек ребра к заданной точке на вектор ребра. Если произведения имеют одинаковые знаки, точка не лежит между начальной и конечной точками ребра, и метод возвращает false. В противном случае, если все проверки пройдены, метод возвращает true, указывая, что точка действительно лежит на ребре.

Метод `Paper::pointsNeedToRotate`

Предназначен для определения вершин, которые нужно вращать вокруг определённой оси. Ось вращения задаётся через индексы начальной и конечной точек (`indexStart` и `indexEnd`), а также через индекс вершины, относительно которой будет проводиться проверка на необходимость вращения (`index`).

Алгоритм

1. Инициализация множества `pointsSet` для хранения индексов вершин, которые будут вращаться.
2. Использование обхода в ширину (BFS) для поиска всех вершин на пути между `indexStart` и `indexEnd`, исключая сами эти точки.
3. Если вершина с индексом `index` не находится в полученном множестве, значит, выбранная вершина не находится на пути между `indexStart` и `indexEnd`, и возвращается список всех вершин, не участвующих в вращении.
4. В противном случае, возвращается список вершин, которые должны быть повёрнуты.

Метод `Paper::pointOnEdges`

Выполняет проверку на принадлежность заданной точки к любому из рёбер объекта `Paper`. Для этого метод последовательно анализирует каждое ребро, опираясь на функционал метода isPointOnEdge, чтобы установить, лежит ли точка на рассматриваемом ребре. В случае положительного результата проверки для хотя бы одного ребра, метод возвращает `true`, подтверждая факт принадлежности точки к рёбрам. В противном случае, если ни одно ребро не содержит данную точку, метод возвращает `false`, указывая на её внешнее расположение по отношению к текущим рёбрам.

Метод `Paper::checkPointsOnEdge`

Определяет, принадлежат ли обе заданные точки (`lineStart` и `lineEnd`) к одному и тому же ребру.

Алгоритм

1. Итерация по всем рёбрам.
2. Для каждого ребра проверяется, лежат ли обе заданные точки на этом ребре.
3. Если найдено ребро, на котором лежат обе точки, возвращается `true`.
4. Если такого ребра не найдено, возвращается `false`.

Метод `Paper::addCrease`

Добавляет сгиб на бумагу, который определяется начальной и конечной точками. Этот процесс включает в себя несколько проверок и обновлений структур данных, отражающих состояние бумаги.

Проверки перед добавлением сгиба

1. **Принадлежность точек рёбрам:** Сначала проверяется, что обе точки (`start` и `end`) принадлежат хотя бы одному из рёбер. Если одна из точек не принадлежит ни одному ребру, генерируется исключение `logic_error`.
2. **Проверка нахождения точек на одном ребре:** Затем проверяется, что точки не лежат на одном и том же ребре. Если они лежат на одном ребре, генерируется исключение `invalid_argument`.

Логика обновления вершин и рёбер

После проверок, если точки удовлетворяют условиям, начинается процесс обновления вершин и рёбер:

1. **Обновление вершин:** Если точки `start` и `end` уже присутствуют среди вершин, используются их текущие индексы. В противном случае точки добавляются в список вершин, и используются их новые индексы.
2. **Удаление и добавление рёбер:** Для каждой из точек `start` и `end` проверяется её принадлежность к существующим рёбрам. Если точка принадлежит ребру, это ребро удаляется, а вместо него добавляются два новых ребра, отражающие новые сгибы, образованные добавлением сгиба.

Обработка сгибов

Наконец, добавляется сам сгиб (`creaseCandidate`) в список сгибов (`creases`), при условии, что он не пересекается с существующими сгибами. В случае пересечения генерируется исключение.

Метод `Paper::rotateVertices`

Отвечает за вращение вершин бумаги вокруг оси, определенной сгибом. Процесс начинается с выбора набора вершин для вращения, что осуществляется на основе их расположения относительно указанного сгиба. Ключевым моментом является идентификация тех вершин, которые фактически находятся на «пути» вращения, исключая начальные и конечные точки самого сгиба. После определения этих вершин происходит их непосредственное вращение. Для каждой выбранной вершины вычисляется ее новое положение, что достигается применением преобразования, основанного на матрице вращения. Это преобразование изменяет координаты вершин таким образом, что они описывают дугу вокруг оси сгиба на заданный угол.

Завершающим этапом является обновление координат вершин в общем списке вершин объекта `Paper`, гарантируя тем самым, что модификации будут отражены в общей структуре бумаги.

Метод `Paper::writePaperToFil`

Метод `writePaperToFile` занимается сохранением текущего состояния объекта `Paper` в файл, что позволяет фиксировать и воспроизводить конфигурацию бумаги. Процедура начинается с открытия файла для записи. В случае успеха в файл последовательно выводятся данные о каждой вершине, каждом ребре и каждом сгибе бумаги. Эти данные оформляются в четко определенном формате, обеспечивая таким образом читаемость и стандартизацию записи.

Для вершин указываются их координаты, для рёбер — индексы связанных вершин, а для сгибов — пары индексов, описывающих начальные и конечные точки каждого сгиба. После завершения записи всех элементов файл закрывается, тем самым сохраняя внесенные в него данные.

Метод `Paper::processPaperFromJson`

Задействуется для создания и обработки объекта `Paper` на основе данных из файла в формате JSON. Процесс начинается с открытия файла. В случае неудачи выводится сообщение об ошибке. После успешного открытия файла содержимое парсится с целью извлечения данных о бумаге, включая её размеры, расположение сгибов и углы вращения.

При отсутствии ключевых полей в JSON файле, метод сообщает об ошибке и завершает выполнение. Если же все необходимые данные присутствуют, производится чтение и преобразование информации о сгибах и углах из относительных координат в абсолютные, учитывая размеры бумаги.

На основе считанных данных формируется объект `Paper`, для которого последовательно добавляются сгибы. Далее, для каждого сгиба применяется вращение на соответствующий угол. В конце процесса текущее состояние объекта `Paper` сохраняется в файл, после чего файл закрывается. В завершение, на основе полученных данных генерируется 3D модель в формате OBJ.

Примеры работы программы

Приведём несколько разнообразных результатов моделирования листа бумаги при помощи нашей программы. Входные данные будут прикреплены вместе с отчётом.

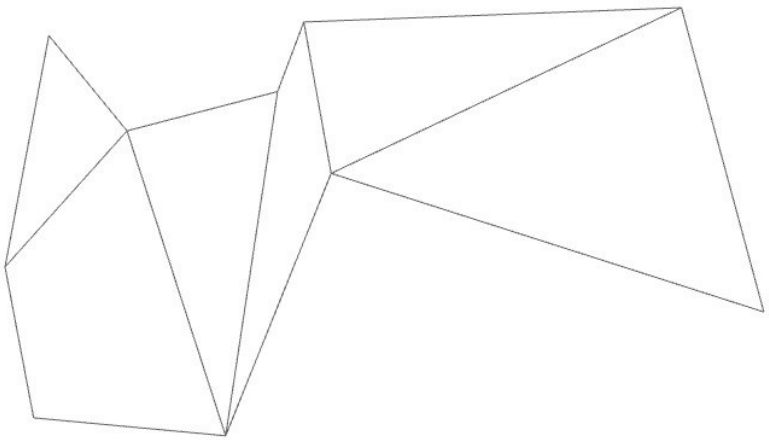

Рисунок 1. Несколько хаотичных сгибов (example1.json).

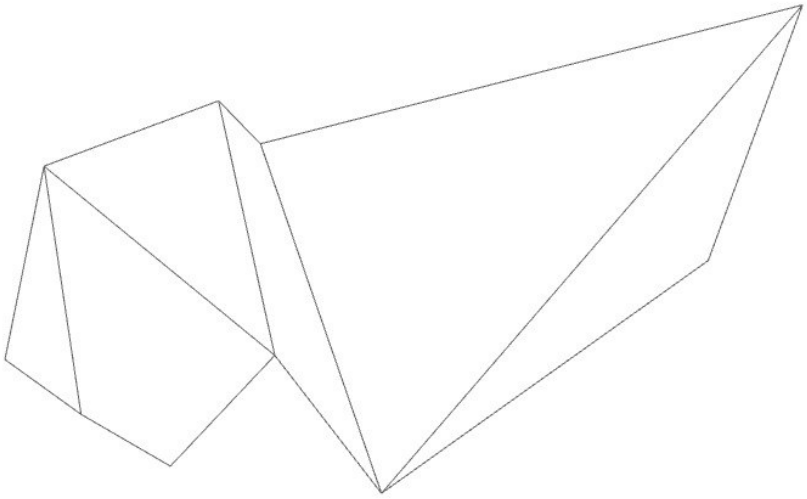

Рисунок 2. Несколько хаотичных сгибов под другой перспективой (example1.json).

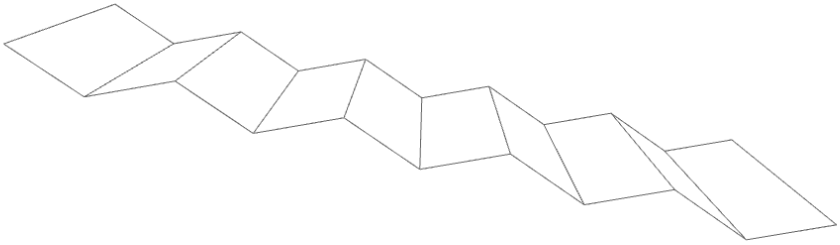

Рисунок 3. Гармошка (example2.json)

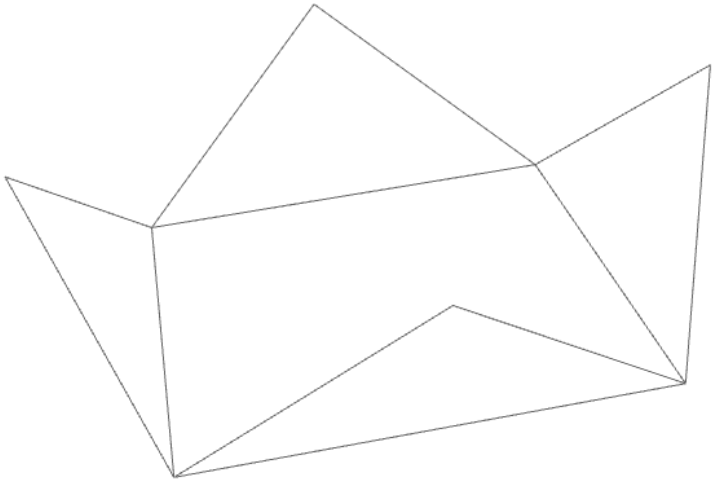

Рисунок 4. Конверт (example3.json).

Инструкция по установке и сипользованию

Проект можно найти по следующей ссылке:

`https://github.com/lilcosinass18/paper_modeling`

Сборка проекта

git clone https://github.com/lilcosinass18/paper_modeling
В склонированном репозитории нужно создать папку lib, и положить туда скаченную библиотеку **nlohmann** для работы с json файлами. Далее собрать проект:
mkdir build
cd build
cmake ./path_to_prj
make
cd build
cd bin
./paper_modeling

Как визуализировать?

После того, как программа выведет в командной строке путь до файла, в котором сохранено описание листа, пройдите по следующей ссылке:

`https://3dviewer.net/index.html`

И загрузите соответствующий файл. Перед Вами появится картинка деформированного листа бумаги.