**Федеральное государственное автономное образовательное учреждение высшего образования**

**«Национальный исследовательский университет**

**«Высшая школа экономики»**

**Факультет компьютерных наук**

**Основная образовательная программа Прикладная математика и информатика**

КУРСОВАЯ РАБОТА

**Программный проект на тему**

**Графические JavaScript библиотеки**

Выполнил студент группы 185, 3 курса:

Войтецкий Артем Александрович

Руководитель КР:

Доцент, Никитин Алексей Антонович

**Москва 2021**

Оглавление

[Введение 1](#_Toc73824673)

[Постановка задачи 1](#_Toc73824674)

[Начало работы 1](#_Toc73824675)

[Реализация 2](#_Toc73824676)

[Альтернативный вариант 3](#_Toc73824677)

[Исследование и решение проблемы 4](#_Toc73824678)

[Результаты и выводы 6](#_Toc73824679)

[Ссылки 7](#_Toc73824680)

# Введение

В современном мире люди привыкли получать больше всего информации с помощью визуальных образов. Поэтому использование визуальных материалов может существенно упростить понимание материала школьниками и студентами. Сейчас для этого преподаватели чаще используют доску в классе, на которой каждый раз от руки рисуют схематичные примеры, но даже это происходит не всегда. Именно поэтому внедрение в образовательный процесс современных информационных технологий для создания визуальных материалов очень важная и актуальная задача. В этом может помочь разрабатываемый сервис «VisualMath.ru».

# Постановка задачи

Хотелось бы, чтобы каждый желающий мог открыть учебные материалы на своем компьютере, ноутбуке, смартфоне или планшете. И, несмотря на то, что мощности современных устройств без проблем хватает на работу в браузере, это не значит, что мы не должны стремиться к оптимизации работы наших программ. Замедление работы всей системы влечет за собой низкую продуктивность обучения, ведь пользователь отвлекается от главного. Сокращение времени автономной работы устройств вредит учащимся, которые могли бы использовать сервис прямо на занятиях для лучшего понимания материала.

К сожалению, проблемы не обошли и VisualMath. Множество существующих материалов страдали от утечек памяти (наглядно это будет продемонстрировано позднее), что не может не вредить всему проекту по вышеописанным причинам.

Моей задачей стал поиск и исправление этой проблемы.

# Начало работы

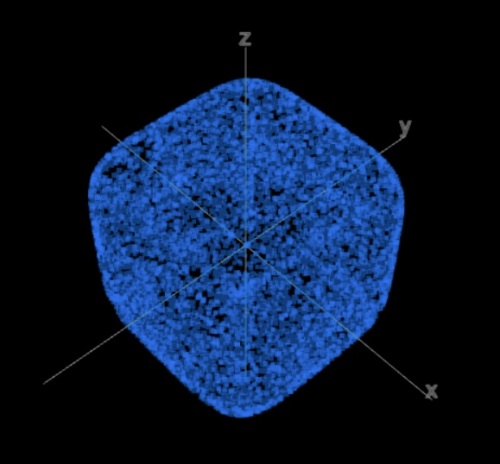
Материалы проекта были написаны на языке программирования JavaScript с использованием библиотеки Grafar. Она разрабатывалась в сотрудничестве с сервисом VisualMath, поэтому эта библиотека является важной частью всего проекта и соответствует его нуждам. Работа осуществляется с помощью высокоуровневого интерфейса, что повышает удобство разработки, а подробная документация позволяет с легкостью разобраться в основах. Также неоспоримым плюсом является прямой контакт с разработчиком, что в некоторых ситуация может помочь с решением возникающих проблем.

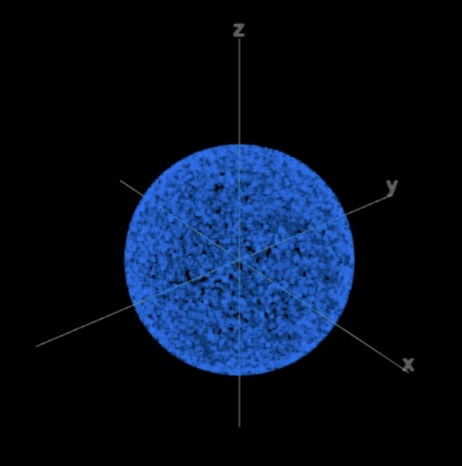
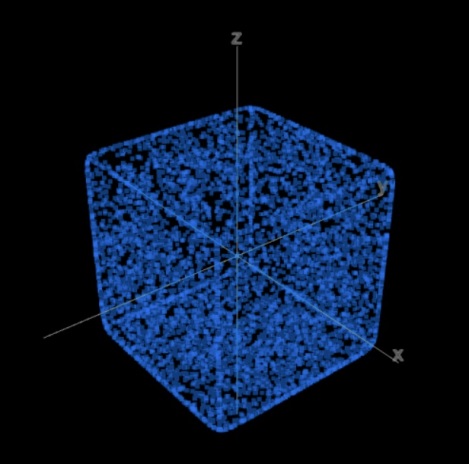
Именно потому что Grafar является такой важной и неотъемлемой частью VisualMath, я решил начать свою работу со знакомства с этой библиотекой. Для этого было реализован тестовый проект, в котором реализована визуализация морфинга куба в сферу.

# Реализация

Для реализации я использовал формулу, где R – это радиус сферы и половина стороны куба:

При n = 2 в результате получается сфера с радиусом R, а с увеличением n результат приближается к кубу. Таким образом реализуется плавное преобразование от одной фигуры к другой. Для этого в библиотеке Grafar я использовал метод grafar.vsolve(), в который передается формула, сколько решений необходимо найти и размерность объекта, а на выходе получаем набор точек, удовлетворяющий условиям. Получается, что фигура отрисовывается множеством точек.





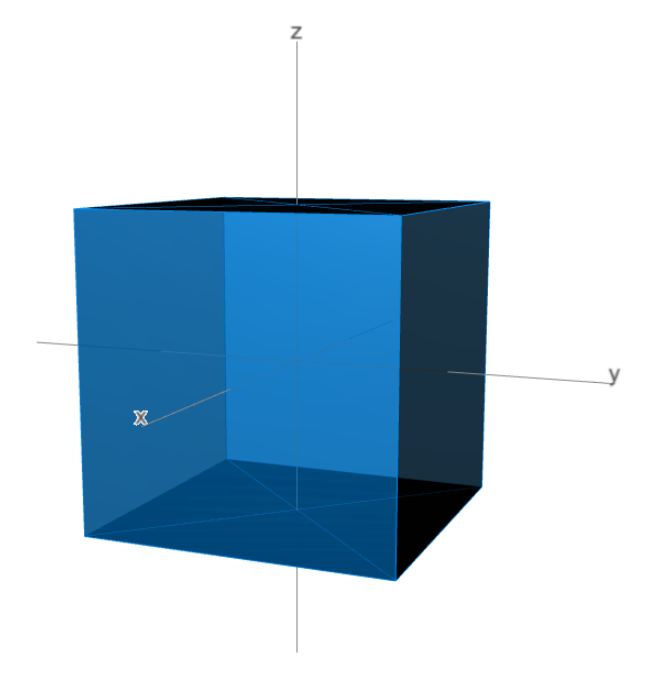
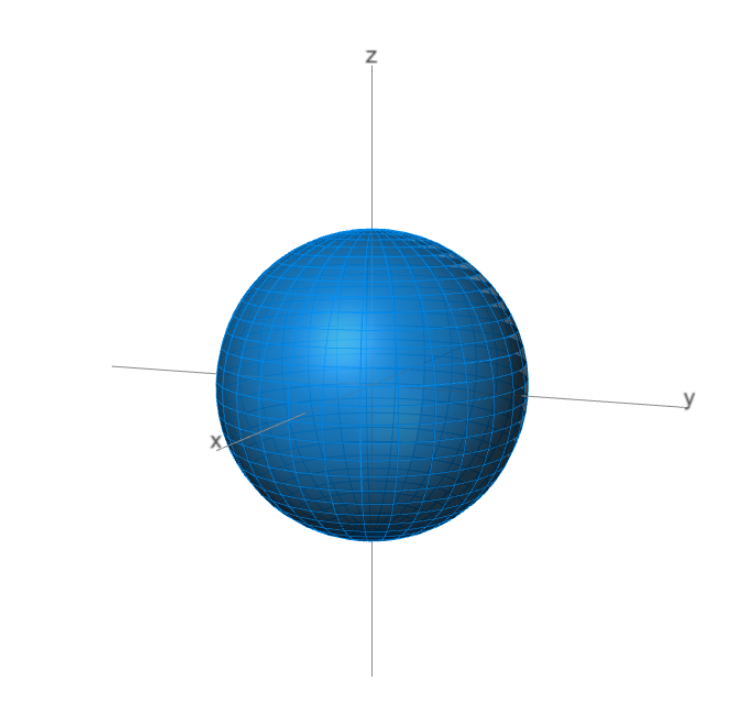
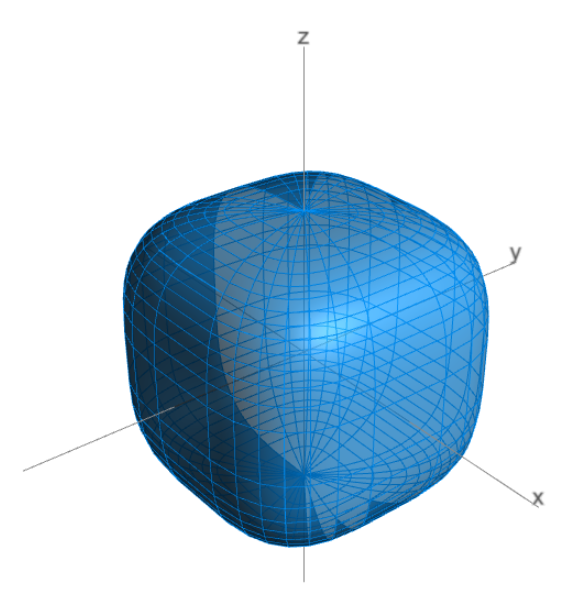
Я реализовал два варианта визуализации. Вариант, в котором пользователь сам выбирает стадию морфинга через специальный инструмент, а также вариант, в котором постепенный морфинг зациклен и происходит автоматически.

Но, несмотря на то, что получившиеся картинки достаточно информативны, их можно вращать в разных плоскостях, приближать и отдалять, есть и некоторые недочеты. Такой способ реализации требует больше вычислительных ресурсов, ведь сначала нужно найти подходящие точки, а потом нарисовать каждую в своем месте. Также это не самый красивый вариант изображения фигур.

# Альтернативный вариант

Для лучшего результата можно использовать суперформулу:

Также в библиотеке Grafar есть методы grafar.pin() и grafar.map(), комбинацией которых можно отобразить представленную выше суперформулу. В таком случае уже не отрисовывается множество конкретных точек, а строятся поверхности по результатам формулы, что существенно сокращает требуемые вычислительные ресурсы.

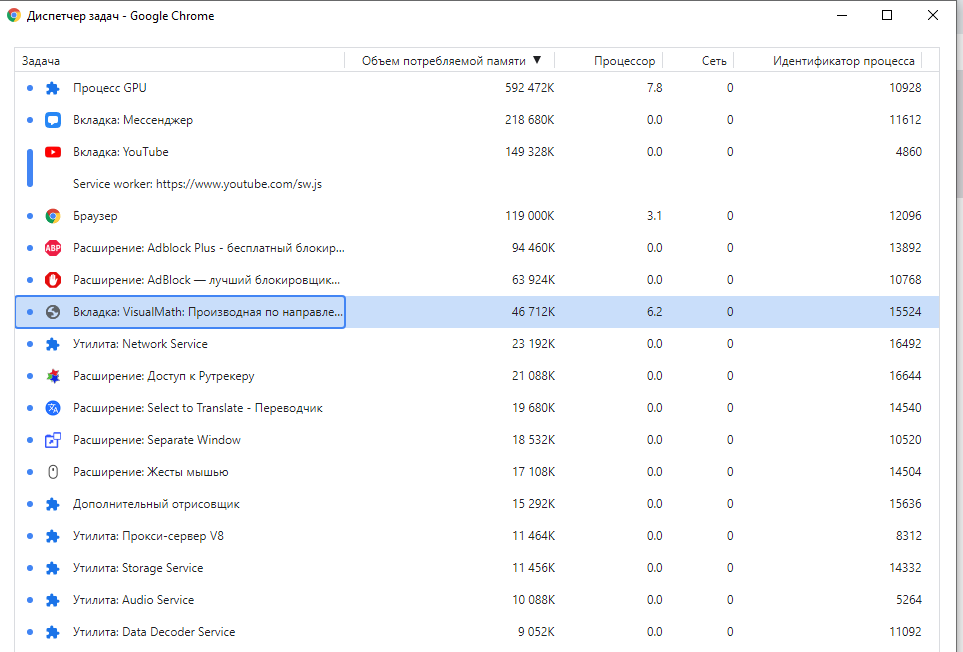
Как видно из скриншотов, результат получился визуально приятнее. Также использование этих методов раскрывает одно важное достоинство библиотеки. Она построена с использованием принципа реактивной анимации. Этот принцип позволяет явно выразить зависимости между элементами, а также реагировать на событие, вычисляя новые значения всех зависимых элементов. Таким образом достигается простота в построении различных анимаций, ведь нет нужды вручную просчитывать все шаги.

На основе именно второго метода в дальнейшем строились более сложные визуализации.

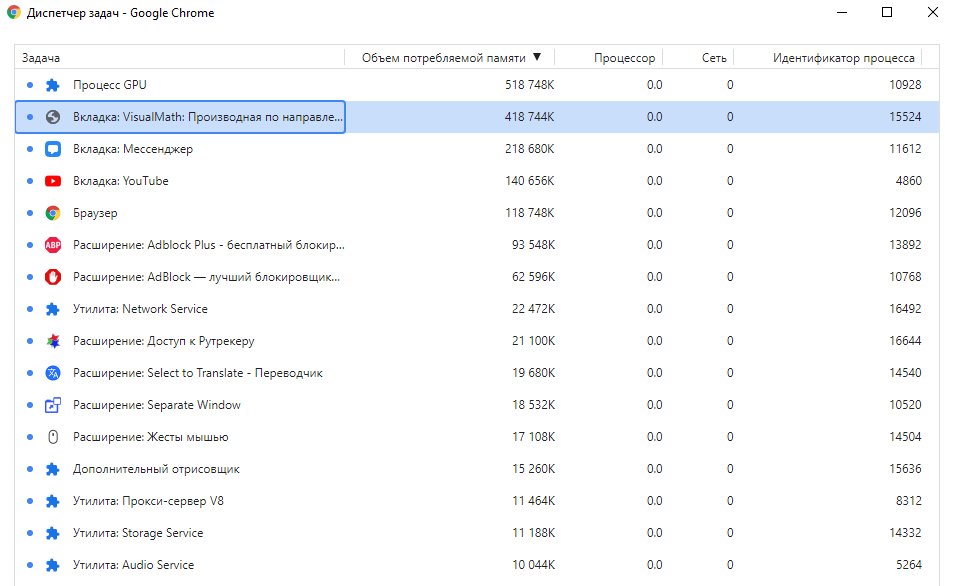
# Исследование и решение проблемы

Далее я приступил к исследованию проблемы, которую предстояло решить.

На скриншотах представляю вам потребление памяти для двух популярных интернет-страниц: vk.com и youtube.com, а также потребление памяти одного из проблемных проектов сразу после его запуска. В начале все достаточно хорошо, но утечка проявляется во время работы.



На втором скриншоте представлено потребление памяти для тех же двух ресурсов и нашего подопытного проекта, но уже спустя непродолжительное время работы в нем.



Проблема утечки памяти становится не просто очевидна, а пугает своим масштабом, ведь это далеко не предел. Разумеется, аналогичное наблюдалось и в других проектах.

После подтверждения проблемы, начались поиски ее причины. Утечка памяти сохранялась во всех проектах в той или иной степени, поэтому поиски я начал с общих элементов - библиотек. Такой подход привел меня к успеху, источник был найден, обнаружился он в библиотеке Grafar. Утечка происходила из-за того, что старое состояние визуализации удалялось не до конца, т.е. потребление памяти накапливалось со временем. Поговорив с разработчиком (наглядная демонстрация преимущества тесного сотрудничества), он подтвердил мои выводы.

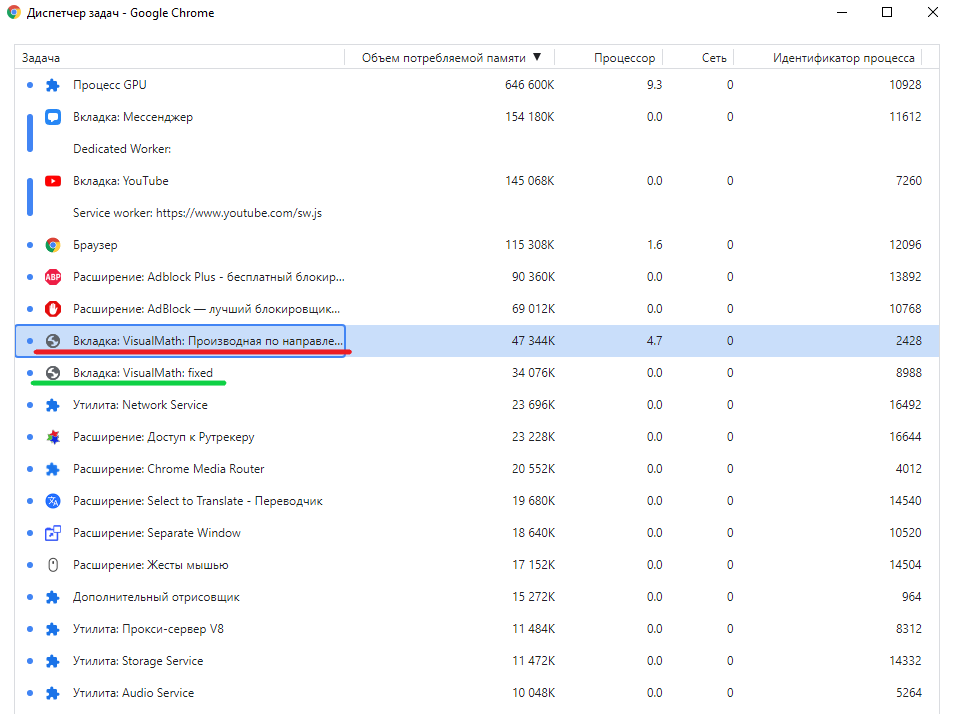
Решение нашлось в обновлении версии Grafar на более свежую. Предоставленные проекты были написаны достаточно давно, чтобы библиотека успела сильно обновиться и избавиться от этой проблемы. Но возникла иная – отсутствие обратной совместимости. Некоторые методы изменились, а некоторые и вовсе перестали существовать. Также дополнительную сложность вызвало отсутствие полной документации. На момент написания отчета появился удобный сайт, в котором многое описано, но изначально учиться можно было лишь на небольшом количестве предоставленных примеров, а о работе некоторых методов из предыдущей версии приходилось догадываться или проверять опытным путем.

Мне пришлось полностью переписать данные проекты, сохраняя задачу каждого. Также удалось придать коду более понятный вид и избавиться от некоторых ненужных действий. Но не везде удалось полностью повторить исходное изображение, так как часть необходимого функционала отсутствует в современной версии библиотеки. Например, управление толщиной линии или размером точки. Но даже в таких случаях удалось найти альтернативное решение, которое сохраняет для пользователя простоту визуального восприятия.

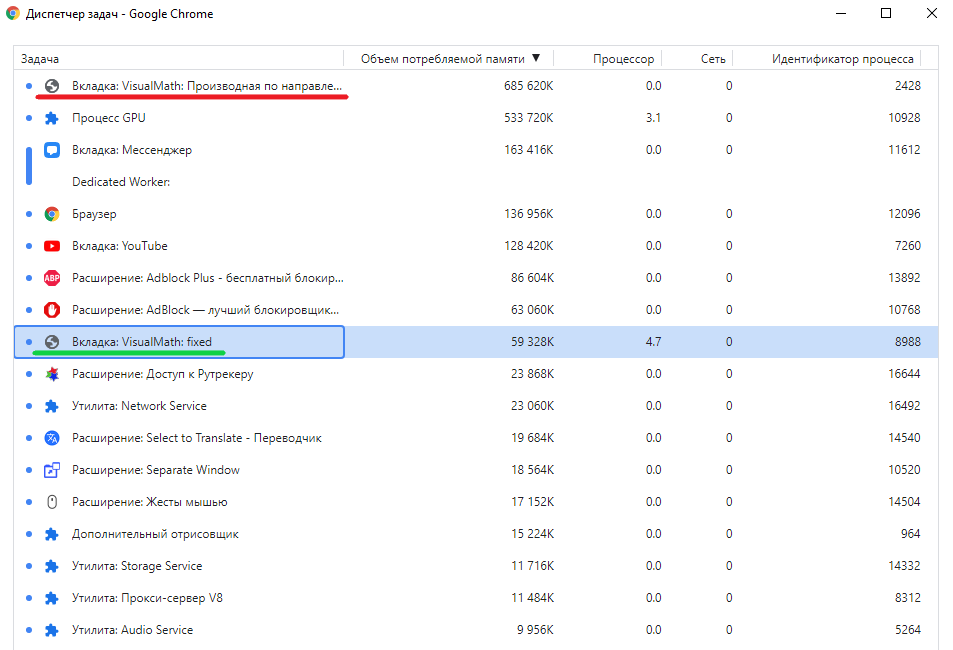
# Результаты и выводы

Начну подведение итогов со сравнения изначального проекта и исправленного.

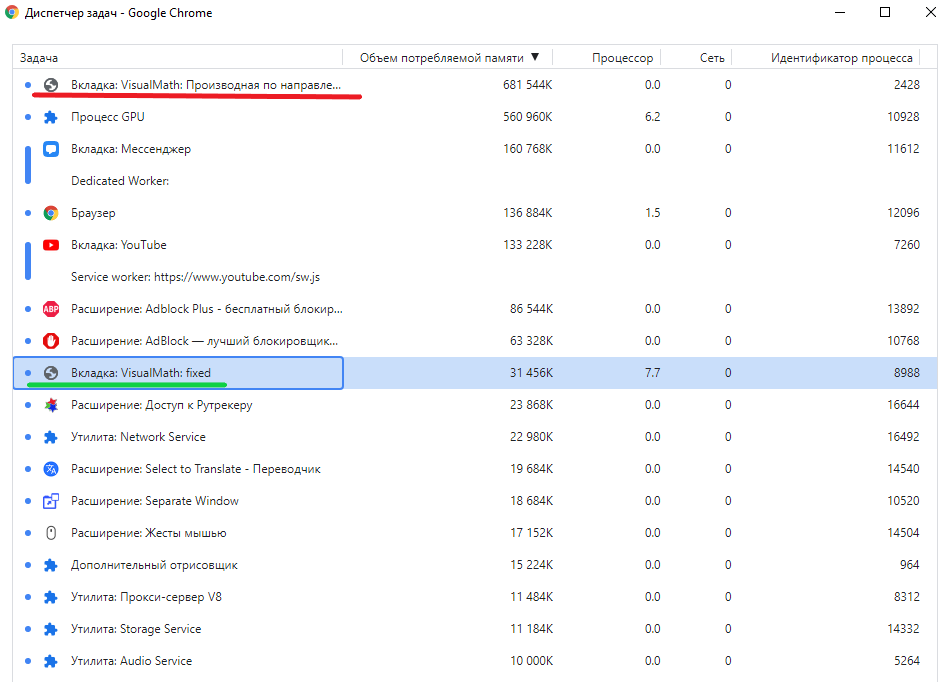
Как и в прошлый раз, начну с только что открытого проекта. Можно заметить экономию по памяти уже сейчас, но не будем останавливаться.



На следующем скриншоте представлено потребление памяти после 2 минут активного взаимодействия с каждым из проектов.



Здесь может показаться, что проблема решена не до конца, ведь потребление памяти увеличилось все равно, пусть и не так значительно, но стоит немного подождать и память вернется обратно к изначальному потреблению.



Таким образом, можно сделать вывод, что проблема с утечкой памяти полностью решена. Также, согласно субъективным ощущениям, визуализации стали более отзывчивы на действия пользователя.

# Ссылки

Ссылка на проект: <https://github.com/MrARVO/JavaProj>

Grafar: <https://github.com/thoughtspile/Grafar>

Документация Grafar: [https://thoughtspile.github.io/grafar/?new#/README](https://thoughtspile.github.io/grafar/?new%23/README)

Superformula wiki: <https://en.wikipedia.org/wiki/Superformula>

Superformula: <http://paulbourke.net/geometry/supershape/>