Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИУ, Информатика и системы управления

КАФЕДРА ИУ7, Программное обеспечение ЭВМ и информационные технологии

Научно-исследовательская работа

НА ТЕМУ:

Исследование звука: Удаление шумов и разделение аудиодорожки на несколько говорящих.

Студент ИУ7-54Б (Группа) _____ (Подпись, дата) **А.А. Андреев** (И.О.Фамилия)

Руководитель НИР _____ (Подпись, дата) **М. С. Шаповалова** (И.О.Фамилия)

Консультант _____ (Подпись, дата) _____ (И.О.Фамилия)

2021 г.

Оглавление

Введение.

Обработка звука - это процесс исследования динамической/статической звуковой дорожки при помощи применения определенного набора линейных и нелинейных алгоритмов с целью получения необходимой информации. Данный процесс происходит с использованием компьютерных программ и зачастую сопровождается трудными техническими вычислениями, которые ложатся на вычислительные мощности компьютера или на отдельные его комплектующие части.

Процесс исследования и обработки звукова так или иначе присутствует в разных сферах профессиональной деятельности, будь то голосовые помощники, встроенные в мобильные устройства или любые другие девайсы, индустрия профессионального бизнес-сообщества для фиксирования необходимой информации или же специальные службы, использующие самые современные технологии для расследования преступлений.

Если мы говорим о задаче обработки звука, то чаще всего имеем в виду применение к звуковой дорожке определенного набора стандартных и собственных алгоритмов, которые позволяют получить определенный срез информации о дорожке или же получить новую трансформированную аудио дорожку.

Цель данной работы – реализовать алгоритмы удаления посторонних шумов из аудио дорожки и разделения дорожки на несколько говорящих.

Чтобы достигнуть поставленной цели, требуется решить следующие задачи:

1) реализовать алгоритм определения основных действующих линий в звуковой дорожке;

2) выбрать стандартные алгоритмы удаления шумов для каждой действующей линии;

3) к выбранному стандартному алгоритму удаления шумов добавить собственную нейронную сеть, изменяющую коэффициенты “очистки” дорожки от шумов в процессе ее жизни;

4) разработать программное обеспечение, которое позволит преобразовать разделенные и очищенные от шумов аудио дорожки в текстовые файлы.

1. Аналитическая часть

В данной части производится анализ и сравнение методов и алгоритмов предметной области, необходимых для выполнения поставленной задачи.

1.1 Постановка задачи

Разработать программу исследования и преобразования звуковой дорожки. Программа должна поддерживать следующие функции: добавление звуковой дорожки; очистка звуковой дорожки от шумов; разделение звуковой дорожки на несколько говорящих; выделение из звуковой дорожки сказанного текста. Можно выделить следующие задачи:

- чтение аудиодорожки в формате .mp3;
- удаление шумов из аудиодорожки;
- разделение одной аудиодорожки на несколько по количеству выявленных говорящих;
- распознавание русского текста в аудио дорожке и его запись в файл;

1.2 Представление звука в компьютере

Записанный звук состоит из множества звуковых волн, одновременно попадающих на датчик микрофона в некоторый промежуток времени, в результате чего мы получаем длинный вектор из чисел - это амплитуды (громкость) сигнала в течение небольшого времени. Частота сигнала проводного телефона 8kHz, это значит что мы за секунду 8000 раз измеряем амплитуду (громкость) суммарного сигнала, звуковые карты как правило используют частоту 44.1 или 48kHz.

Наша последовательность является суммой множества звуковых волн, и мы можем вычислить какие волны приняли участи в нашей сумме. Теоретически, любой сложный звук может быть разложен на последовательность простейших гармонических сигналов разных частот, каждый из которых представляет собой правильную синусоиду и может быть описан числовыми параметрами.

Чтобы делать с аудио сложные вещи, такие как распознавание человека по голосу, перевод речи речи в текст или удаление шума с помощью глубокого обучения, нужно вычислить вклад различных частот в аудиопоследовательность — **спектр**. **Спектр** можно представить в виде спектрограммы — изображения, показывающего зависимость амплитуды сигнала во времени на различных частотах. Один столбец в спектрограмме соответствует спектру короткого участка исходного сигнала, более тёплые тона означают большее значение.

Спектр для спектрограммы можно вычислить с помощью дискретного преобразования Фурье (см. п. 1.6 Алгоритм Фурье), реализованного в библиотеке Numpy. Функция Stft проводит преобразование Фурье. Массив делится на части определённой длины (рассчитанной в calcSpec) и для каждой из частей применяется функция преобразования Фурье, взятая из Numpy возвращает готовую спектрограмму. (пример см. Рисунок 1.2.1)

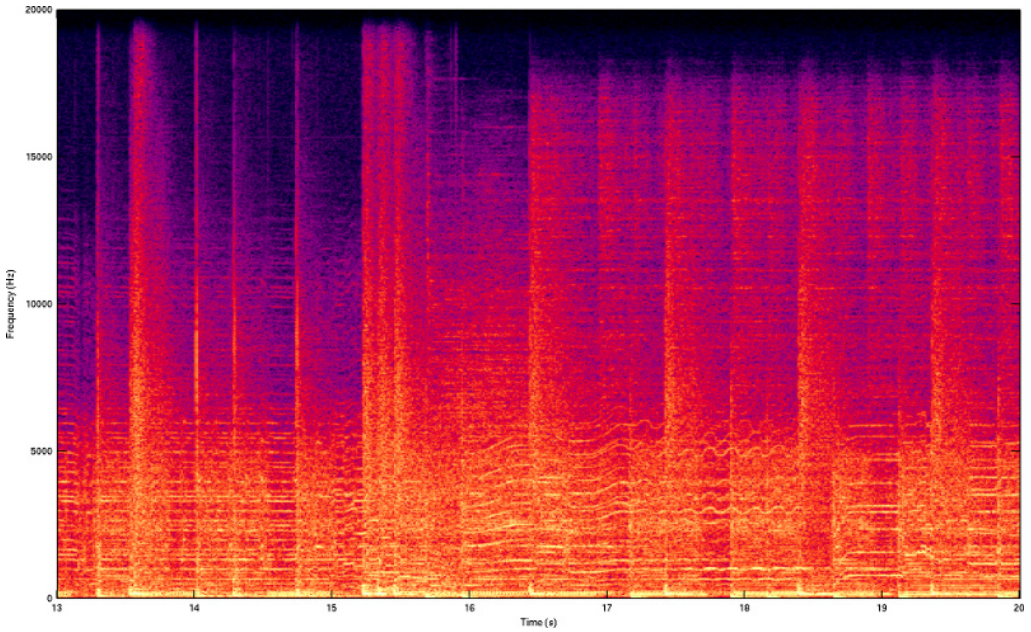


Рисунок 1.2.1: Пример спектрограммы, полученной из звукового файла, при помощи библиотеки Numpy.

1.3 Удаление шума и модель NSNet2

Звуковой сигнал, записываемый в реальных акустических условиях, часто содержит нежелательные шумы, которые могут порождаться окружающей средой или звукозаписывающей аппаратурой. Значит, полученное цифровое описание также будет содержать нежелательные шумы.

Также важной функцией является calcFeat, позволяющая нам прологарифмировать спектрограмму, растягивая нижние частоты и сжимая верхние. Голос человека лежит в диапазоне 85-3000Гц, а диапазон звуковых частот в нашей записи 16кГц — маленький промежуток на всем диапазоне, и помощью логарифмирования мы “растягиваем” нужные нам низкие частоты и “поджимаем” ненужные высокие.

Чтобы “очистить” звук, к цифровому описанию необходимо применить фильтр, который убирает нежелательные шумы. Но возникает другая проблема. Каждый из видов шумов требует свой фильтр, который необходимо подбирать вручную или искать в банках данных фильтров. Отфильтровать шум на частотах, отличающихся от человеческой речи, проблем нет, от них избавлялись еще до этих ваших нейросетей. А вот убрать детский плач или

стучание клавиш без значительного ухудшения качества голоса было проблематично.

В решении данной проблемы могут помочь модели глубокого обучения. Основное преимущество нейросетей перед заранее подготовленными фильтрами заключается в большем охвате различных видов шумов. Нейросеть можно натренировать, постоянно добавляя всё новые виды шума.

В нашем случае мы воспользуемся моделью NSNet2. Эта нейронная сеть использовалась компанией Microsoft. Целью разработки данной сети было создание модели для очистки звука от шума в реальном времени. Данная модель состоит из полносвязного слоя с ReLU, двух рекуррентных GRU (Gated Recurrent Unit) блоков и полносвязных слоев (FF, feed forward) с ReLU и sigmoid активацией.

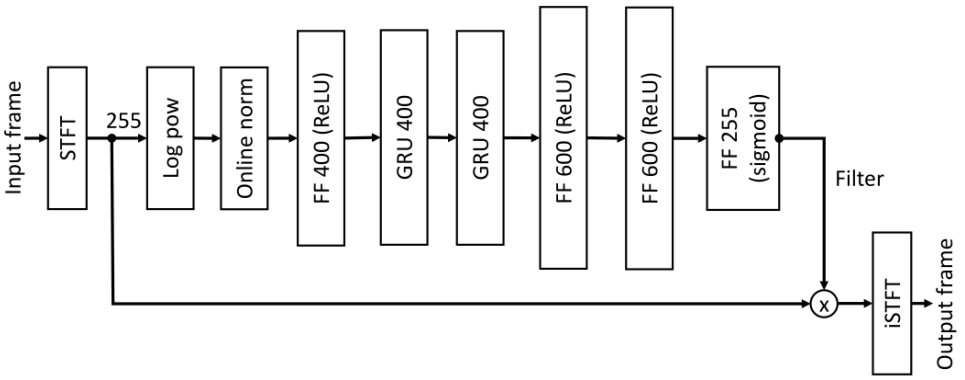


Рисунок 1.2.1: Пример спектрограммы, полученной из звукового файла, при помощи библиотеки Numpy.

На речь влияет большое количество внешних условий. Человек может говорить громко или тихо, быстро или медленно, говорить он может в большой комнате или в маленькой, далеко от микрофона или близко к нему. Для моделирования этих более сложных условий, были применены аугментации. В частности, в данном случае, для модификации звука использовались случайные биквадратные фильтры. Благодаря применения таких аугментаций, зашумление звука наиболее приближено к реальным условиям.

Представленные результаты по качеству работы можно посмотреть в статье Data augmentation and loss normalization for deep noise suppression. Построенная модель имеет хорошие показатели для различных типов шума.

1.4 Конвертация модели в OpenVINO

OpenVINO (Open Visual Inference & Neural Network Optimization) - продукт, разрабатываемый компанией Intel. Как видно из названия, OpenVINO - это набор инструментов для исполнения и оптимизации нейронных сетей.

Существует множество фреймворков для создания и тренировки нейросетей. Для того, чтобы можно было запускать нейросети из различных фреймворков на любом интеловском железе, в составе OpenVINO есть модуль Model Optimizer.

По факту, Model Optimizer - это набор python-скриптов, которые позволяют привести нейронные сети различных форматов к некоторому универсальному представлению, называемому IR (Intermediate Representation). Это позволяет OpenVINO работать с любой нейросетью, независимо от того, из какого фреймворка она взята.

В процессе своей работы Model Optimizer также оптимизирует структуру сверточных нейронных сетей. Например, объединяя результаты сверток, заменяя слои на последовательность линейных операций и т.д.

В последнее время, с появлением API, в Model Optimizer проводится все меньше оптимизаций, и основная его работа сводится к конвертации моделей без каких-либо серьезных изменений.

Конвертация в IR-представление различается для моделей из Open Model Zoo и других моделей. Open Model Zoo – репозиторий глубоких нейросетевых моделей, содержащий большое количество обученных моделей, которые могут исполняться при помощи OpenVINO. Данный репозиторий хранит не только модели, но и параметры для конвертации моделей из разных фреймворков в промежуточный формат OpenVINO.

1.6 Алгоритм Фурье

Быстрое преобразование Фурье (*БПФ*, *FFT*) — алгоритм ускоренного вычисления дискретного преобразования Фурье, позволяющий получить результат за время, меньшее чем $O(N^2)$ (требуемого для прямого, поформульного вычисления).

При применении основного алгоритма дискретное преобразование Фурье может быть выполнено за $O(N(p_1 + \ldots + p_n))$ действий при $N = p_1 p_2 \ldots p_n$.

Дискретное преобразование Фурье преобразует набор чисел $a_1,..,a_{n-1}$ в набор чисел $b_1,..,b_{n-1}$, такой что $b_i = \sum_{j=0}^{n-1} a_j \varepsilon^{ij}$, где ε - первообразный корень из единицы, то есть $\varepsilon^n = 1$ и $\varepsilon^k \neq 1$ при $0 < k < n$ Основной шаг алгоритма состоит в сведении задачи для N чисел к задаче с меньшим числом. Для $N = pq,\ q > 1,\ p > 1$ над полем комплексных чисел вводятся: $\varepsilon_v = \varepsilon^{2\pi/v}$, где $\varepsilon_v^v = 1$, где v – любое число.

Дискретное преобразование Фурье может быть представлено в виде

$$b_i = \sum_{k=0}^{p-1} \sum_{j=0}^{q-1} a_{kq+j} \varepsilon_N^{(kq+j)i}$$

.

(Эти выражения могут быть легко получены, если исходную сумму разбить на меньшее число сумм с меньшим числом слагаемых, а после полученные суммы привести к одинаковому виду путём сдвига индексов и их последующего переобозначения).

Таким образом:

$$b_i = \sum_{k=0}^{p-1} \sum_{j=0}^{q-1} a_{kq+j} \varepsilon_N^{(kq+j)i} = \sum_{j=0}^{q-1} \varepsilon_N^{ij} \left(\sum_{k=0}^{p-1} a_{kq+j} \varepsilon_N^{kiq}\right)$$

.

С учётом того, что $\varepsilon_N^{kiq} = \varepsilon_{N/q}^{ki}$ и $N/q = p$, окончательная запись:

$$b_i = \sum_{j=0}^{q-1} \varepsilon_N^{ij} \left(\sum_{k=0}^{p-1} a_{kq+j} \varepsilon_p^{ki}\right)$$

.

Далее вычисляется каждое b_i, где $i = [0,\ p-1]$, здесь по-прежнему требуется совершить O(N) действий, то есть на этом этапе производится $p * O(N) = O(Np)$ операций.

Далее считается b_{i+mp}, где $i = [0,\ p-1]$, $m = [1,\ q-1]$. При замене $i \to i + mp$ в последней формуле, выражения, стоящие в скобках, остались неизменными, а так как они уже были посчитаны на предыдущем шаге, то на вычисление каждого из них потребуется только

$O(q)$ действий. Всего p(q-1) = N - p чисел. Следовательно, операций на этом шаге

$(N - p) * O(q) = O((N - 1)q) \simeq O(Nq)$. Последнее с хорошей точностью верно при любых N.

Алгоритм быстрого преобразования Фурье логично применять для $N \gg 1$, потому как при малом числе отсчетов он дает небольшой выигрыш в скорости по отношению к прямому расчету дискретного преобразования Фурье. Таким образом, для того чтобы полностью перейти к набору чисел b_0,.., b_{n-1}, необходимо $O(Np) + O(Nq)$ действий. Следовательно, нет разницы, на какие два числа разбивать N — ответ от этого сильно не будет меняться. Уменьшено же число операций может быть только дальнейшим разбиением

N.

Скорость алгоритма $(N = pq)$:

1. $p \gg q \rightarrow O(Np)$
2. $p \sim q \rightarrow O(Np)$
3. $p \ll q \rightarrow O(Np)$

То есть число операций при любом разбиении N на два числа, есть $O(Nc)$, где $c = max(p, q)$.

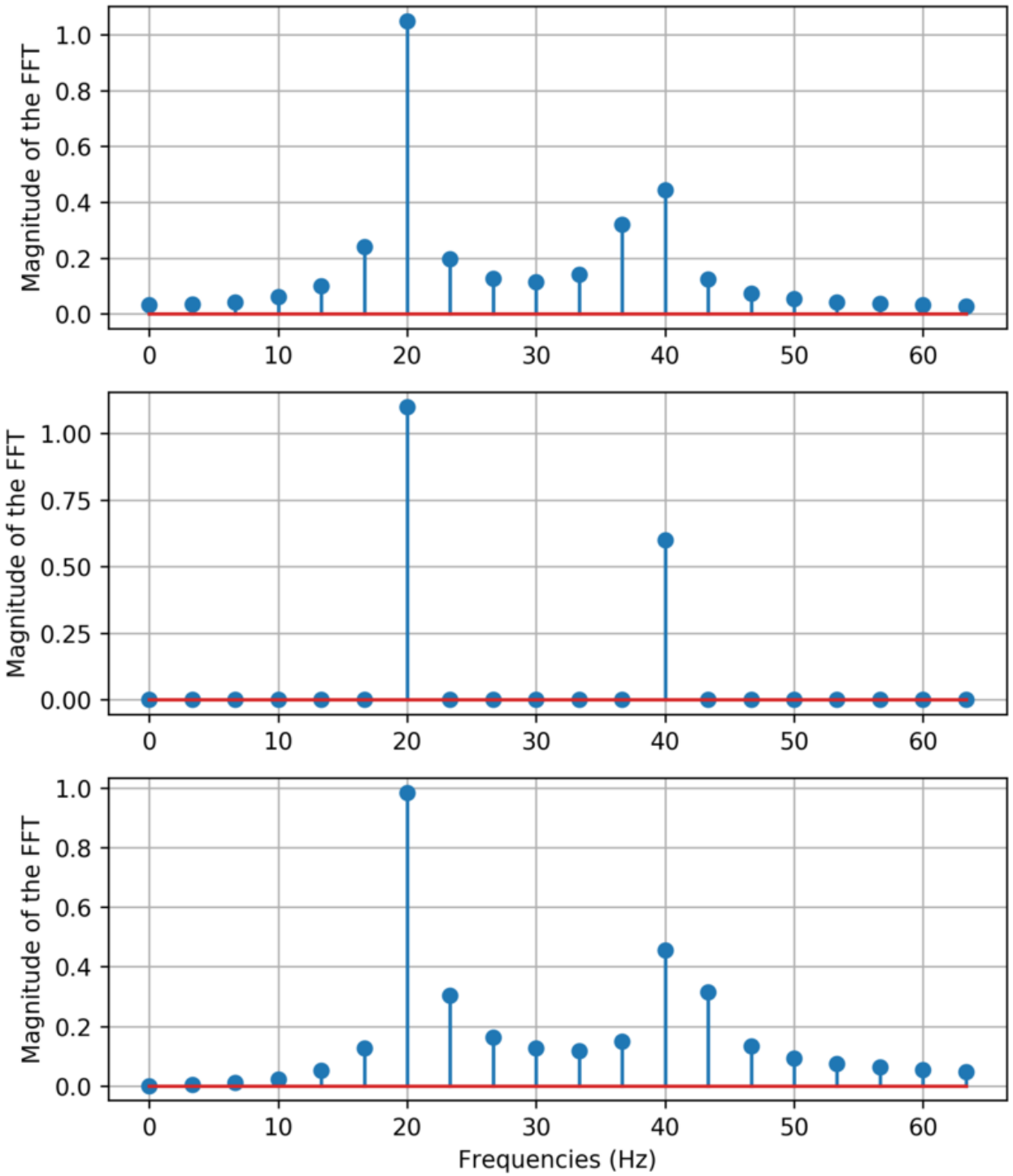


Рисунок 1.7.1: Амплитуды быстрого преобразования Фурье для разного количества компонент разложения N. Первый случай: N=L-10, где L — количество отсчетов сигнала; второй случай: N = L; третий: N = L + 10. В первом и последних случаях спектральные характеристики оцениваются менее точно. Данный эффект связан с явлением Гиббс.

1.7 Преобразование голоса в текст: Распознавание речи

1.7.1 Разбиение слов

Первой задачей, которую приходится решать при распознавании речи, является разбиение этой самой речи на отдельные слова. Для простоты предположим, что в нашем случае речь содержит в себе некоторые паузы (промежутки тишины), которые можно считать “разделителями” слов.

В таком случае нам нужно найти некоторое значение, порог — значения выше которого являются словом, ниже — тишиной. Вариантов тут может быть несколько:

- задать константой (сработает, если исходный сигнал всегда генерируется при одних и тех же условиях, одним и тем же способом);
- кластеризовать значения сигнала, явно выделив множество значений соответствующих тишине (сработает только если тишина занимает значительную часть исходного сигнала);
- проанализировать энтропию;

Как вы уже догадались, речь сейчас пойдёт о последнем пункте :) Начнём с того, что энтропия — это мера беспорядка, “мера неопределённости какого-либо опыта” (с). В нашем случае энтропия означает то, как сильно “колеблется” наш сигнал в рамках заданного фрейма.

Для того, что бы подсчитать энтропию конкретного фрейма выполним следующие действия:

- предположим, что наш сигнал пронормирован и все его значения лежат в диапазоне [-1;1];
- построим гистограмму (плотность распределения) значений сигнала фрейма:

рассчитаем энтропию, как $$E = \sum_{i=0}^{N-1} P[i] * \log_2(P[i])$$;

1.7.2 MFCC

И так, мы у нас есть набор фреймов, соответствующих определенному слову. Мы можем пойти по пути наименьшего сопротивления и в качестве численной характеристики фрейма использовать средний квадрат всех его значений (Root Mean Square). Однако, такая метрика несёт в себе крайне мало пригодной для дальнейшего анализа информации.

MFCC — это своеобразное представление энергии спектра сигнала. Плюсы его использования заключаются в следующем:

- Используется спектр сигнала (то есть разложение по базису ортогональных [ко]синусоидальных функций), что позволяет учитывать волновую “природу” сигнала при дальнейшем анализе;
- Спектр проецируется на специальную mel-шкалу, позволяя выделить наиболее значимые для восприятия человеком частоты;
- Количество вычисляемых коэффициентов может быть ограничено любым значением (например, 12), что позволяет “сжать” фрейм и, как следствие, количество обрабатываемой информации;

1.7.3 Разложение в ряд Фурье

Первым делом рассчитываем спектр сигнала с помощью дискретного преобразования Фурье (желательно его “быстрой” FFT реализацией).

$$X[k] = \sum_{n=0}^{N-1} x[n] * e^{-2*\pi*i*k*n/N}, 0 \leq k < N$$

Так же к полученным значениям рекомендуется применить оконную функцию Хэмминга, чтобы “сгладить” значения на границах фреймов.

$$H[k] = 0.54 - 0.46 * \cos(2 * \pi * k/(N-1))$$

То есть результатом будет вектор следующего вида:

$$X[k] = X[k] * H[k], 0 \leq k < N$$

Важно понимать, что после этого преобразования по оси X мы имеем частоту (hz) сигнала, а по оси Y — магнитуду (как способ уйти от комплексных значений):

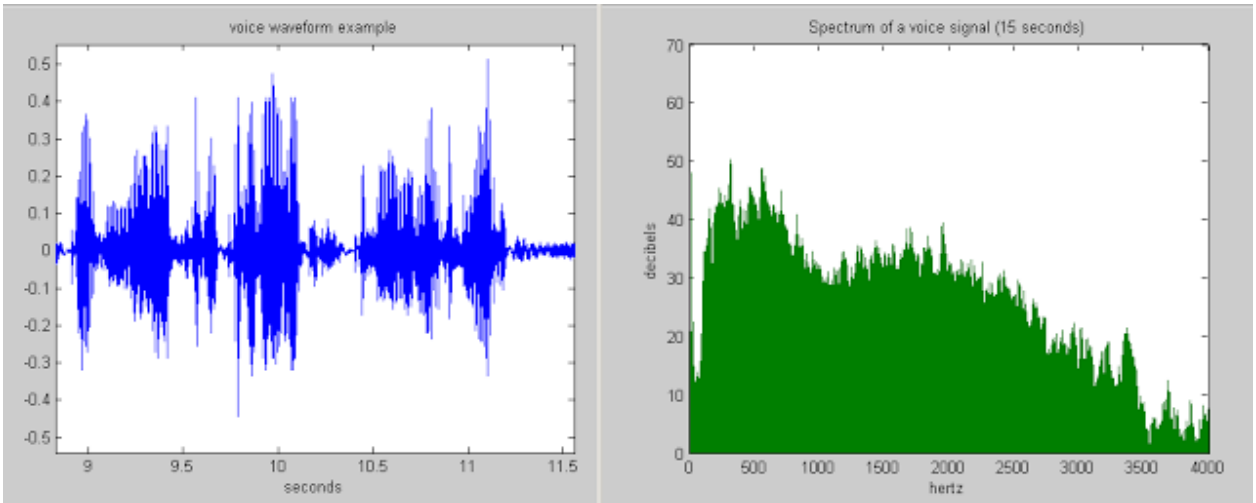


Рисунок 1.7.3.1: Пример голосовых волн и их спектр

1.7.4 Расчет mel-фильтров

Начнём с того, что такое mel. Опять же согласно Википедии, mel — это “психофизическая единица высоты звука”, основанная на субъективном восприятии среднестатистическими людьми. Зависит в первую очередь от частоты звука (а также от громкости и тембра). Другими словами, эта величина, показывающая, на сколько звук определённой частоты “значим” для нас.

Преобразовать частоту в мел можно по следующей формуле (запомним ее как «формула-1»):

$$M = 1127 * \log(1 + F/700)$$

Обратное преобразование выглядит так (запомним ее как «формула-2»):

$$F = 700 * (e^{M/1127} - 1)$$

Получаем график зависимости mel / частота (см. Рисунок 1.7.4.1).

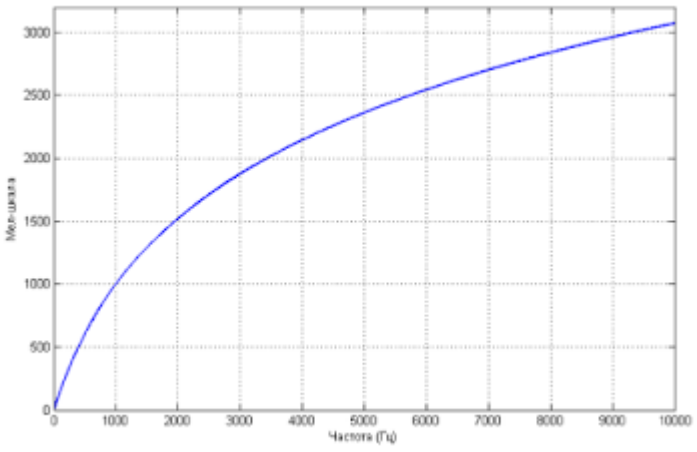


Рисунок 1.7.4.1: График зависимости mel / частота

Допустим у нас есть фрейм размером 256 элементов. Мы знаем (из данных об аудиоформате), что частота звука в данной фрейме 16000hz. Предположим, что человеческая речь лежит в диапазоне от [300; 8000]hz.

Количество искомых мел-коэффициентов положим M = 10 (рекомендуемое значение).

Для того, что бы разложить полученный выше спектр по mel-шкале, нам потребуется создать “гребенку” фильтров. По сути, каждый mel-фильтр это треугольная оконная функция, которая позволяет просуммировать количество энергии на определенном диапазоне частот и тем самым получить mel-коэффициент. Зная количество мел-коэффициентов и анализируемый диапазон частот мы можем построить набор таких вот фильтров (см. Рисунок 1.7.4.2).

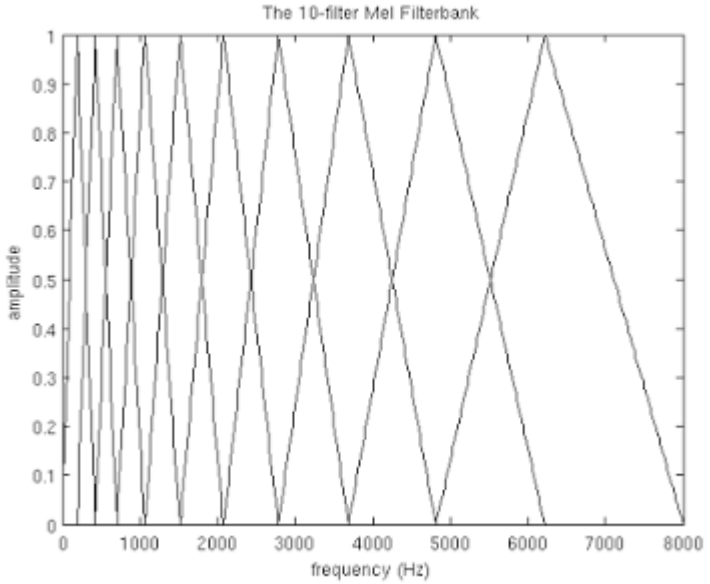


Рисунок 1.7.4.2: Набор фильтров

Чем больше порядковый номер мел-коэффициента, тем шире основание фильтра. Это связано с тем, что разбиение интересующего нас диапазона частот на обрабатываемые фильтрами диапазоны происходит на шкале мелов.

Для нашего случая диапазон интересующих нас частот равен [300, 8000]. Согласно формуле-1 в на мел-шкале этот диапазон превращается в [401.25; 2834.99].

Далее, для того, что бы построить 10 треугольных фильтров нам потребуется 12 опорных точек:

m[i] = [401.25, 622.50, 843.75, 1065.00, 1286.25, 1507.50, 1728.74, 1949.99, 2171.24, 2392.49, 2613.74, 2834.99]

На мел-шкале точки расположены равномерно. Переведем шкалу обратно в герцы с помощью формулы-2:

h[i] = [300, 517.33, 781.90, 1103.97, 1496.04, 1973.32, 2554.33, 3261.62, 4122.63, 5170.76, 6446.70, 8000]

Теперь шкала стала постепенно растягиваться, выравнивая тем самым динамику роста “значимости” на низких и высоких частотах.

Теперь нам нужно наложить полученную шкалу на спектр нашего фрейма. Как мы помним, по оси X у нас находится частота. Длина спектра 256 — элементов, при этом в него умещается 16000hz. Решив нехитрую пропорцию можно получить следующую формулу:

f(i) = floor((frameSize+1) * h(i) / sampleRate)

что в нашем случае эквивалентно

f(i) = 4, 8, 12, 17, 23, 31, 40, 52, 66, 82, 103, 128

Зная опорные точки на оси X нашего спектра, легко построить необходимые нам фильтры по следующей формуле:

$$H_m(k) = \begin{cases} 0 & k < f(m-1) \\ \dfrac{k - f(m-1)}{f(m) - f(m-1)} & f(m-1) \le k \le f(m) \\ \dfrac{f(m+1) - k}{f(m+1) - f(m)} & f(m) \le k \le f(m+1) \\ 0 & k > f(m+1) \end{cases}$$

1.7.5 Логарифмирование энергии спектра

Применение фильтра заключается в попарном перемножении его значений со значениями спектра. Результатом этой операции является mel-коэффициент. Поскольку фильтров у нас M, коэффициентов будет столько же.

$$S[m] = \log(\sum_{k=0}^{N-1} |X[k]|^2 * H_m[k]), 0 \le m < M$$

Однако, нам нужно применить mel-фильтры не к значениям спектра, а к его энергии. После чего прологарифмировать полученные результаты. Считается, что таким образом снижается чувствительность коэффициентов к шумам.

1.7.6 Косинусное преобразование

Дискретное косинусное преобразование (DCT) используется для того, чтобы получить те самые “кепстральные” коэффициенты. Смысл его в том, чтобы “сжать” полученные результаты, повысив значимость первых коэффициентов и уменьшив значимость последних.

В данном случае используется DCTII без каких-либо домножений на $\sqrt{(2/N)}$ (scale factor).

$$C[l] = \sum_{m=0}^{M-1} S[m] * \cos(\pi * l * (m + \frac{1}{2})/M)), 0 \leq l < M$$

Теперь для каждого фрейма мы имеем набор из M mfcc-коэффициентов, которые могут быть использованы для дальнейшего анализа.

1.8 Выводы из аналитического раздела

В данном разделе были описаны: представление звука в компьютере, конвертация модели OpenVINO, модель NSNet2, алгоритм Фурье, Алгоритм распознавания речи.

Надо отметить, что проблема шумоподавления до сих пор не решена полностью. Методике улучшения речи с помощью нейронных сетей в последнее время уделяется огромное внимание как в научных исследованиях, так и в коммерческих приложениях. Одним из важнейших преимуществ использования нейронных сетей для подавления шума является то, что они способны очищать звук от нестационарных шумов. Ранее известные подходы не позволяли этого сделать.

2. Конструкторская часть.

В данном разделе будет приведена блок-схема алгоритма z-буфера. Также будет описан процесс рендера изображения.

1.1 Требования к программе

Программа должна предоставлять следующие возможности:

- визуальное отображение сцены;
- визуальное отображение источника света;
- поворот объекта.

1.2 Выбор алгоритма для удаления невидимых линий

Для задачи визуализации, поворота относительно его центра и перемещения трехмерного объект наиболее подходящим алгоритмом выглядит алгоритм z-буфера, потому что алгоритм a-буфера будет выполнять лишнюю работу, так как объекты сплошные, алгоритм художника и алгоритмы, работающие в трехмерном пространстве, будут затрачивать значительное время при рендеринге объекта ввиду своей трудозатратности.

1.3 Процесс отрисовки изображения

В качестве алгоритма визуализации объекта был выбран алгоритм z-буфер. Данный выбор был аргументирован в аналитической части.

Основные этапы работы данного алгоритма:

1. инициализация и заполнение буфера кадра фоновым значением интенсивности цвета;
2. инициализация и заполнение z-буфера максимальным значением z;
3. преобразование объекта в растровую форму (порядок преобразования не имеет значения);
4. для каждого пиксела в многоугольнике вычислить его глубину z(x, y) или расстояние по оси z. Сравнить вычисленную глубину z(x, y) со значением в zbuffer(x, y);
5. Если значение z(x, y) будет меньше значения zbuffer(x, y), тогда заменяем значение zbuffer(x, y) на z(x, y) и вычисляем новое значение интенсивности цвета для точки с координатами (x, y, z(x, y));
6. в противном случае, никаких действий не выполнять.

Процесс рендера программном модуле проходит через 4 этапа:

1. преобразование мировых координат к координатам сцены;
2. удаление невидимых линий и поверхностей при помощи алгоритма z-буфер;
3. вычисление интенсивности цвета для граней методом тонирования Гуро;
4. вывод изображения на экран.

Пункты 2 и 3 выполняются совместно.

Схема работы алгоритма представлена на рисунках 2.2.1 - 2.2.2.

Рис. 2.2.1: Схема работы алгоритма z-буфер часть 1

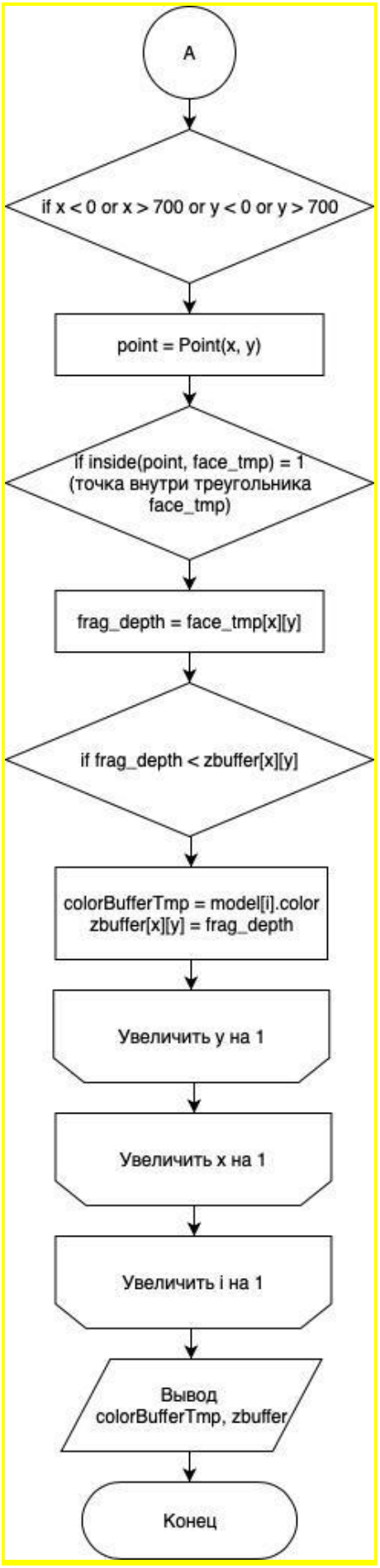

A
if x < 0 or x > 700 or y < 0 or y > 700
point = Point(x, y)
if inside(point, face_tmp) = 1
(точка внутри треугольника
face_tmp)
frag_depth = face_tmp[x][y]
if frag_depth < zbuffer[x][y]
colorBufferTmp = model[i].color
zbuffer[x][y] = frag_depth
Увеличить y на 1
Увеличить x на 1
Увеличить i на 1
Вывод
colorBufferTmp, zbuffer
Конец

Рис. 2.2.2: Схема работы алгоритма z-буфер часть 2

1.4 Предоставляемый функционал

Интерфейс представляет собой некое пространство, в котором пользователь может выполнять различные преобразования с моделью. Можно выделить следующие возможности:

• выбор детали детского конструктора для рендеринга;
• перемещение деталей детского конструктора внутри пространства;
• вращение деталей детского конструктора внутри пространства;

1.5 Вывод

В данном разделе была приведена схема работы алгоритма z-буфер, была описана работа рендера изображения в целом. Также были описаны возможности пользователя.

2 Технологическая часть.

В данном разделе будут рассмотрены требования к разрабатываемому программному обеспечению, средства, использованные в процессе разработки для реализации поставленных задач.

3.1 Требования к программному обеспечению

Программное обеспечение должно реализовывать поставленную на курсовой проект задачу. В программном модуле должен присутствовать интерфейс для взаимодействия с объектом. Программа не должна аварийно завершаться, сильно нагружать процессор, а также не должна задействовать большой объем оперативной памяти.

3.2 Выбор и обоснование языка и среды программирования.

Для разработки данной программы применён язык Python 3, интерфейс Tkinter по следующим причинам:

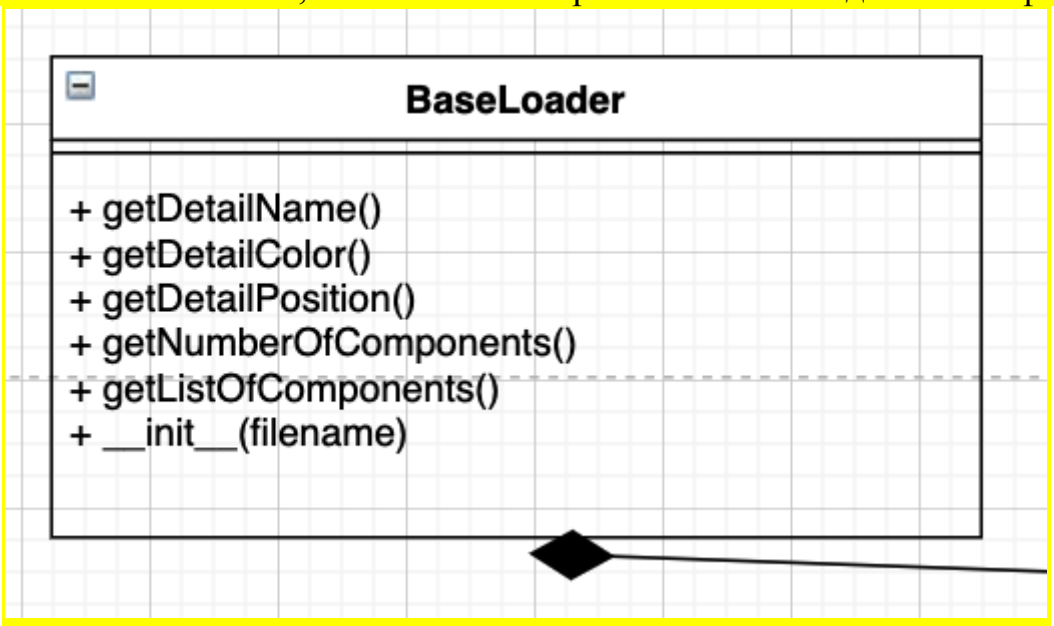
- широкие возможности реализации алгоритмов;
- большое количество виджетов и их гибкость;
- удобство отладки и редактирования программы;

3.3 Использованные типы и структуры данных

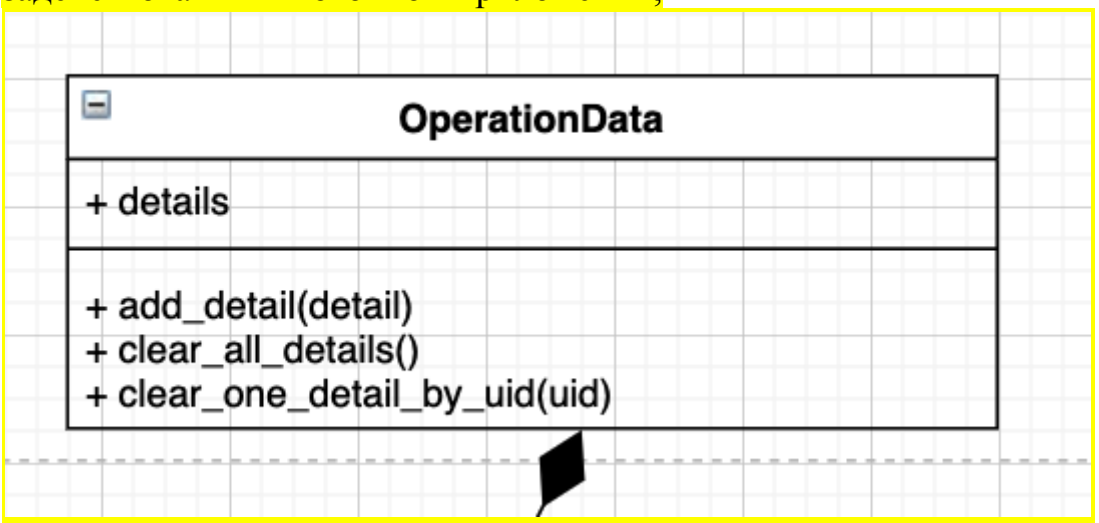
В реализованной программе использовались некоторые базовые типы и структуры данных языка Python 3, такие как integer, float, string, list, map.

Кроме того, в программе используются специально разработанные для поставленной задачи классы:

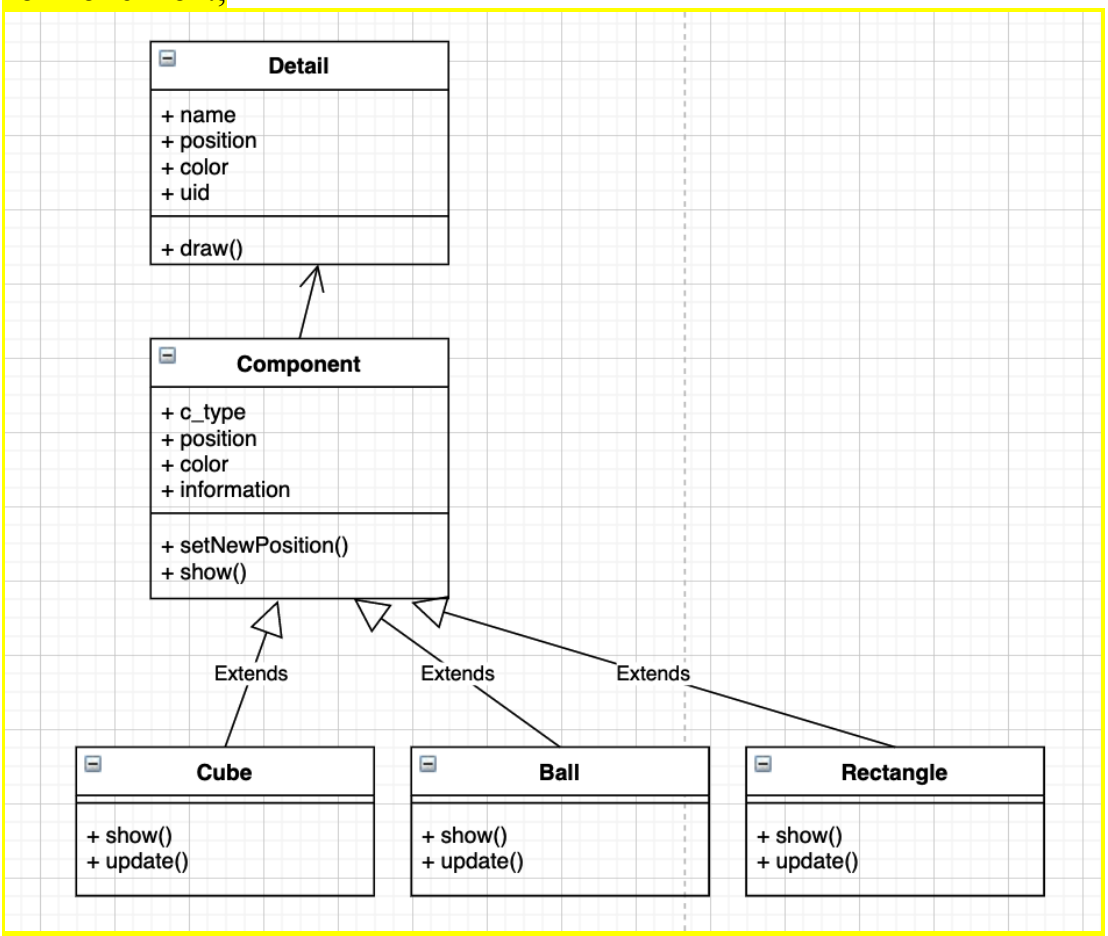
- BaseLoader - класс, назначение которого - считывать данные из файла.



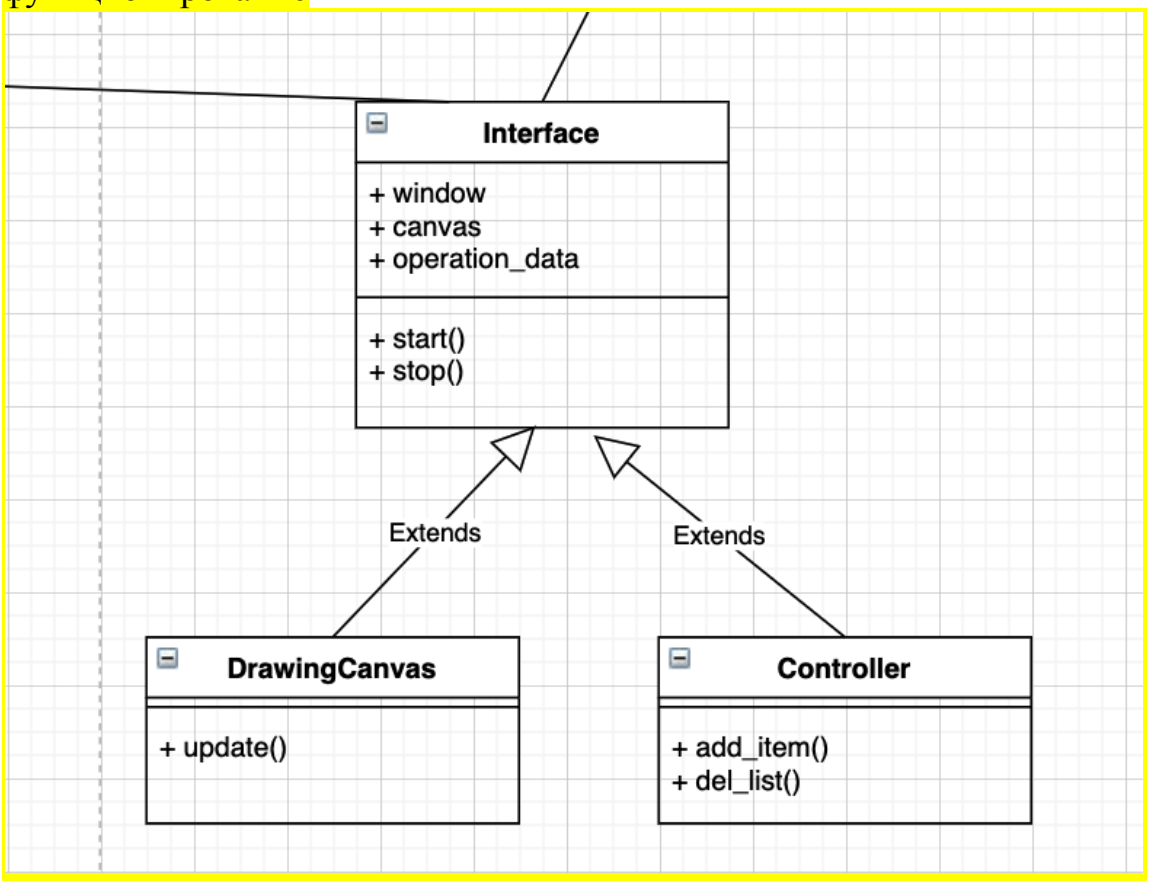
- OperationData - структура, предназначенная для хранения данных, задействованных в оконном приложении;



- Detail - структура данных, используемая для работы с деталью (объектом) детского конструктора. Является составной из компонентов.;



- Interface - класс, отвечающий за отображение блоков интерфейса и их функционирование



3.4 Формат хранения файла

Был выбран формат файла описания геометрии OBJ (Wavefront object) — это разработка Wavefront Technologies для анимационного пакета Advanced Visualizer, потому что он является открытым и был принят другими разработчиками приложений 3D графики. Он может быть экспортирован/импортирован во многих программах, связанных с многомерным моделированием. Основная причина выбора - общепринятый формат.

В файле формата .obj хранятся данные вершин, элементы, кривые произвольной формы, связь между поверхностями свободной формы. В рамках курсового проекта были использованы следующие данные:

- геометрические вершины (v);
- вершины нормалей (vn);
- грани (f).

В очень сложных по детализации файлах формата obj типов данных намного больше, их насчитывается больше 30 разных видов.

Формат записи данных в файл можно увидеть на рисунке 1.3.1.

```
g objMesh
v −0.631852 0.857320 −0.007216
v −0.631852 0.834007 −0.011006
v −0.631852 0.857320 0.007222
vt 0.000000 0.000000
vt 0.030303 0.000000
vt 0.000000 0.045455
vn −1.000000 0.000000 0.000000
vn −1.000000 0.000000 0.000000
vn −1.000000 0.000000 0.000000
f 1/1/1 2/2/2 3/3/3
f 4/4/4 3/3/3 2/2/2
f 5/5/5 6/6/6 7/7/7
```

Рис. 1.3.1. Хранение данных в файлах формата OBJ.

3.5 Оформление итогового изображения

Требование к итоговому изображению:

- Хорошая детализация изображения: отсутствие «рваных пикселей»;
- Правильное отображение световых столбов.

В связи с установленными требованиями к изображению было принято решение установить разрешение полотна **600 x 600 пикселей**. Данное разрешение было установлено с целью снижения затрат по времени на рендер изображения и отрисовку световых столбов.

Итоговое изображение продемонстрировано на рисунке 3.4.1.

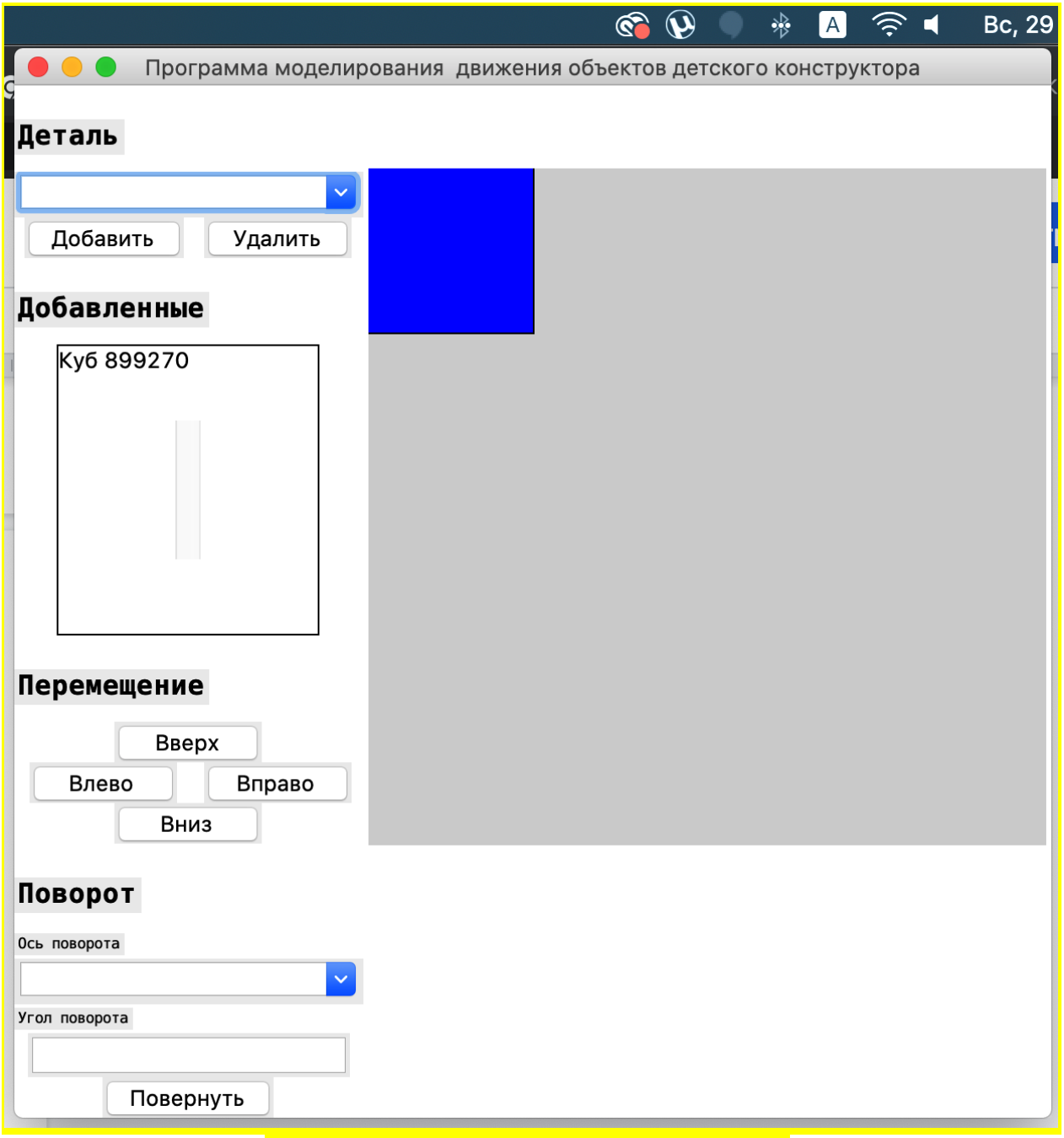


Рис. 3.4.1: Пример работы программы

3.6 Пользовательский интерфейс.

Программа имеет русскоязычный интерфейс, организованный в виде набора виджетов, позволяющих пользователю после запуска программы выбрать объекты (детали детского компьютера) и выполнять действия с ними:

- Добавлять / удалять на холсте
- Перемещать в пространстве
- Поворачивать и поворачивает

После запуска программы перед пользователем открывается пустой холст и интерфейс для добавления / удаления / изменения деталей детского конструктора, см. на рисунке 3.5.1.

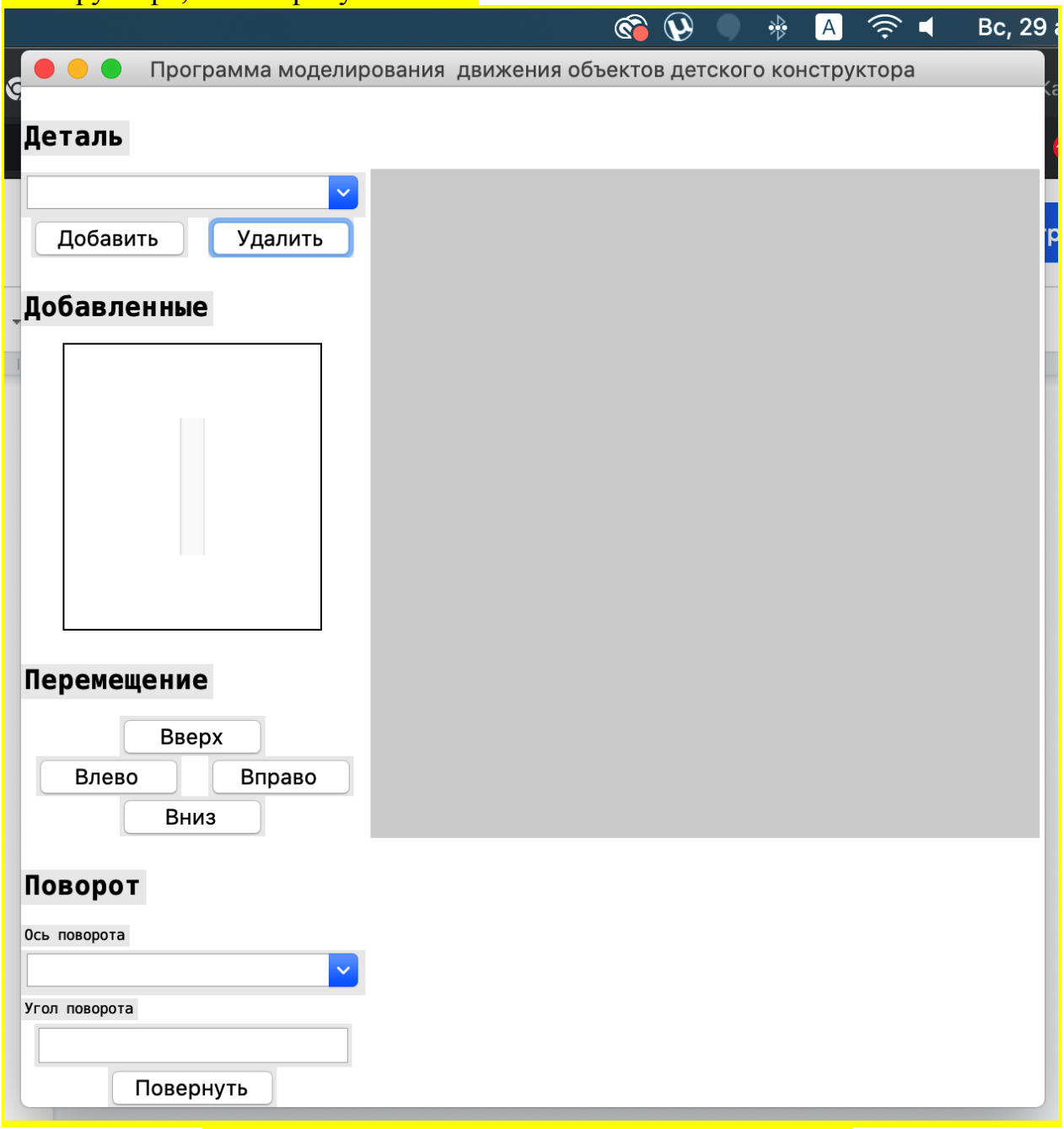


Рис. 3.5.1: Стартовое изображение программы

Интерфейс разбит на 4 блока по типу взаимодействия с полотном, см. рисунок 3.5.1:

1) Блок «Деталь» отвечает за выбор из представленных объектов для отрисовки.
   a) Функциональная клавиша “Добавить” -> Добавить элемент из выбранных в 1.
   b) Функциональная клавиша “Удалить” -> Удалить элемент из выбранных в 2.

2) Блок «Добавленные» отвечает за выбор из отрисованных объектов для операций над ними.
   a) Находится список с возможностью выбора элемента по клику на него
3) Блок «Перемещение» отвечает за перемещение выбранной детали в блоке 2.
   a) Функциональная клавиша “Вверх” -> Переместить элемент из выбранных в 1 вверх.
   b) Функциональная клавиша “Влево” -> Переместить элемент из выбранных в 1 влево.
   c) Функциональная клавиша “Вправо” -> Переместить элемент из выбранных в 1 вправо.
   d) Функциональная клавиша “Вниз” -> Переместить элемент из выбранных в 1 вниз.
4) Блок «Поворот» за поворот выбранной детали в блоке 2.
   a) Выпадающий списка -> Выбор оси поворота (x, y, z). Деталь при вращении относительно оси воспринимается, как точка - ее геометрический центр.
   b) Поле ввода -> Ввод угла поворота вокруг оси с клавиатуры

Доступ ко всем функциям и настройкам осуществляется непосредственно через компоненты основного окна программы.

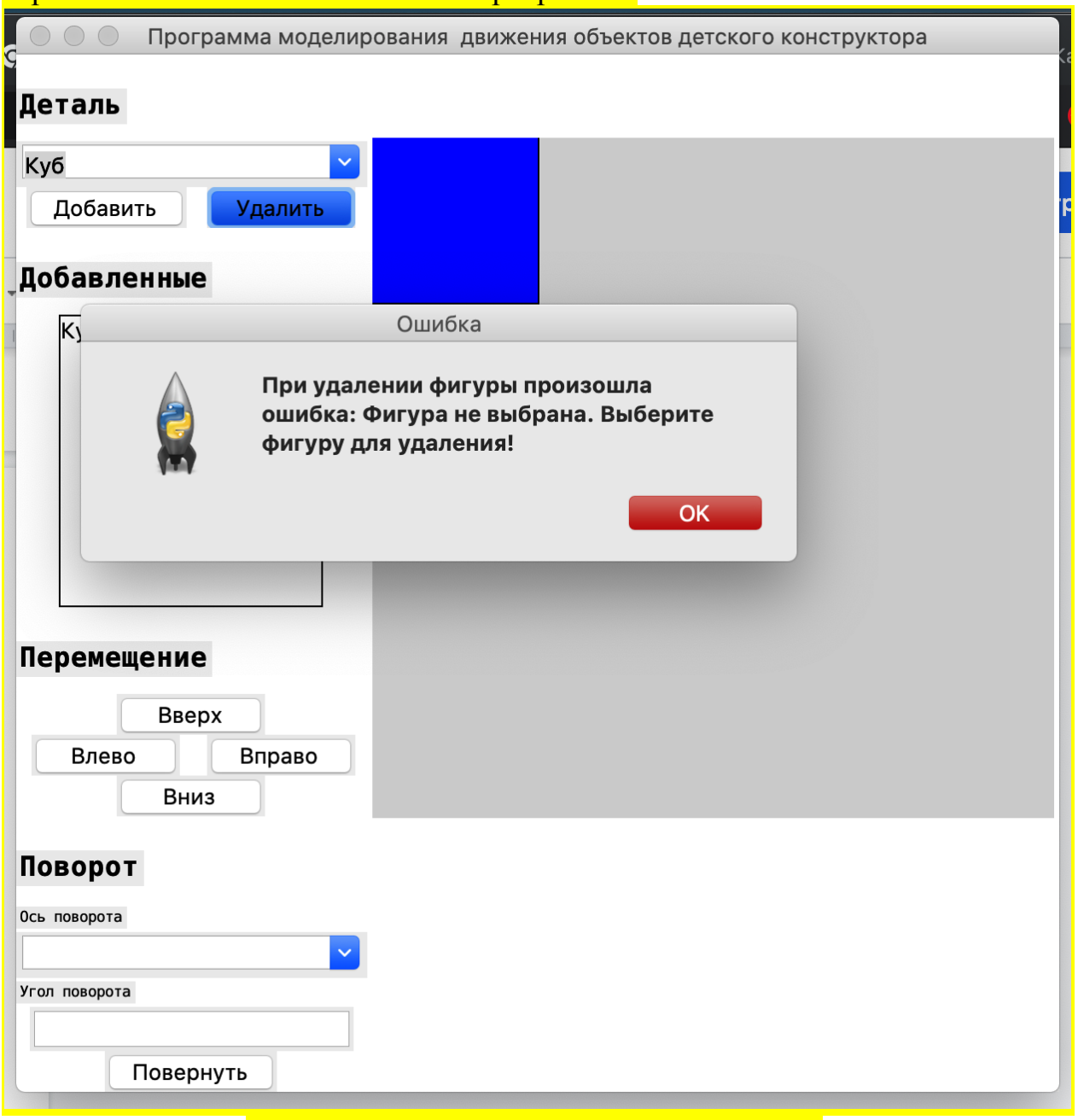


Рис. 3.5.1: Пример уведомления об ошибке

Пользователь получает оповещения при невозможности выполнить запрашиваемое действие, см. Рис 3.7.1

Заключение.

Разработанный программный комплекс отвечает всем предъявляемым к нему требованиям и обеспечивает возможность моделирования световых столбов. Для получения наглядных характеристик алгоритмов необходимо провести исследования на различных наборах данных. Из понимания работы алгоритмов мы можем предположить, что алгоритм z-буфера покажет себя с лучшей стороны скоростью отрисовки сцены, но он не характеризуется реалистичностью синтезированного изображения из-за отсутствия таких эффектов, как освещённость, отражение, преломление и т.д. Сымитировать подобные эффекты возможно, накладывая на объекты заранее просчитанные текстуры освещённости, что в совокупности с эффективными алгоритмами оптимизации по быстродействию приближает изображение к реальности.

Список использованной литературы

1. Роджерс Д., Алгоритмические основы машинной графики: пер. с англ.— М.: Мир, 1989.— 512 с.: ил.
2. Марк Пилгрим, Dive into Python, 2017.
3. Steven F. Lott , Dusty Phillips, Python Object-Oriented Programming - Fourth Edition
4. Проблемы трассировки лучей – из будущего в реальное время. [Электронный ресурс]. – Режим доступа: https://nvworld.ru/articles/ray-tracing/3/
5. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж., Приёмы объектно-ориентированного проектирования. Паттерны проектирования. – СПб: Питер, 2001. – 368 с.: ил. (Серия «Библиотека программиста»)
6. Particle Systems. [Электронный ресурс]. – Режим доступа: http://homepages.inf.ed.ac.uk/tkomura
7. А.Н.Романюк, М.В.Куринный, АЛГОРИТМЫ ПОСТРОЕНИЯ ТЕНЕЙ