



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 «Информатика и системы управления»

---

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

---

# РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

## *К КУРСОВОЙ РАБОТЕ*

### *НА ТЕМУ:*

*«Моделирование падения капли на поверхность  
жидкости и генерации волн»*

Студент ИУ7и-54Б  
(Группа)

\_\_\_\_\_  
(Подпись, дата)

Шавиш Тарек  
(И. О. Фамилия)

Руководитель курсовой работы

\_\_\_\_\_  
(Подпись, дата)

Волкова .Л.Л  
(И. О. Фамилия)

*2024 г.*

# СОДЕРЖАНИЕ

<b>ВВЕДЕНИЕ</b>	<b>4</b>
<b>ПОСТАНОВКА ЗАДАЧИ</b>	<b>5</b>
<b>1 Аналитический раздел</b>	<b>6</b>
1.1 Формализация объектов сцены . . . . .	6
1.2 Анализ методов описания трехмерных моделей . . . . .	6
1.2.1 Каркасная модель . . . . .	6
1.2.2 Поверхностная модель . . . . .	7
1.2.3 Твердотельная модель . . . . .	7
1.2.4 Выбор метода описания модели . . . . .	7
1.3 Анализ алгоритмов удаления невидимых поверхностей . . . . .	8
1.3.1 Алгоритм Робертса . . . . .	8
1.3.2 Алгоритм Варнока . . . . .	8
1.3.3 Алгоритм, использующий Z-буфер . . . . .	9
1.3.4 Алгоритм обратной трассировки лучей . . . . .	9
1.3.5 Выбор алгоритма удаления невидимых поверхностей . . . . .	10
1.4 Анализ алгоритмов закрашки . . . . .	11
1.4.1 Простая закрашка . . . . .	11
1.4.2 Закрашка по Гуро . . . . .	11
1.4.3 Закрашка по Фонгу . . . . .	12
1.4.4 Выбор алгоритма закрашки . . . . .	12
1.5 Анализ моделей освещения . . . . .	13
1.5.1 Модель освещения Ламберта . . . . .	13
1.5.2 Модель освещения Фонга . . . . .	14
1.5.3 Выбор модели освещения . . . . .	15
<b>2 Конструкторская часть</b>	<b>16</b>
2.1 Требования к программному обеспечению . . . . .	16
2.2 Описание структур данных . . . . .	16
2.3 Алгоритм построения изображения . . . . .	17
2.4 Приведение к пространству камеры . . . . .	21
2.5 Невидимые грани . . . . .	22

2.6	Алгоритм, использующий Z-буфер . . . . .	22
2.7	Описание уравнения бегущей волны . . . . .	23
2.8	Описание круговых волн . . . . .	25
2.9	Описание вычисления нормалей . . . . .	25
<b>3</b>	<b>Технологическая часть</b>	<b>27</b>
3.1	Технологическая часть . . . . .	27
3.2	Структура программы . . . . .	27
3.3	Реализация алгоритмов . . . . .	28
3.4	Интерфейс программного обеспечения . . . . .	34
<b>4</b>	<b>Исследовательская часть</b>	<b>37</b>
4.1	Постановка исследования . . . . .	37
4.2	Технические характеристики . . . . .	37
4.3	Время выполнения алгоритмов . . . . .	37
	<b>ЗАКЛЮЧЕНИЕ</b>	<b>41</b>
	<b>СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ</b>	<b>42</b>

## Введение

Компьютерная графика — это совокупность методов и способов преобразования информации в графическое представление при помощи ЭВМ. Ее применение находит широкий спектр приложений, от создания удивительных визуальных эффектов в компьютерных играх до моделирования сложных трехмерных объектов. Одной из интересных задач в области компьютерной графики является визуализация и моделирование волн на поверхности жидкости, особенно круговых волн, вызванных падением капли.

Генерация круговых волн на поверхности жидкости при падении капли имеет большое практическое значение в различных областях, включая компьютерные игры, анимацию, визуальные эффекты и моделирование. Точное воссоздание реалистичного движения волн на поверхности жидкости является сложной задачей, требующей разработки эффективных алгоритмов и методов моделирования.

Целью данного курсового проекта является разработка программного обеспечения, позволяющего моделировать генерацию круговых волн на поверхности жидкости.

## Постановка задачи

Чтобы достигнуть поставленной в курсовом проекте цели, требуется решить следующие задачи.

- произвести анализ существующих алгоритмов компьютерной графики;
- выбрать наиболее подходящие алгоритмы для достижения поставленной цели;
- выбрать средства реализации программного обеспечения;
- разработать программное обеспечение и реализовать выбранные алгоритмы и структуры данных;
- провести замеры временных характеристик разработанного программного обеспечения.

# 1 Аналитический раздел

В этом разделе рассматриваются существующие алгоритмы создания изображений, а также производится выбор наиболее подходящих методов для решения поставленной задачи.

## 1.1 Формализация объектов сцены

Сцена включает в себя следующие элементы:

- поверхность жидкости — трёхмерная модель, которая представлена полигональной сеткой, состоящей из соединённых плоских треугольников;
- раковина — трёхмерная модель, внутри которой находится поверхность жидкости;
- капля — трёхмерная модель, вызывающая волновые колебания на поверхности жидкости;
- источник света — задаётся вектором, определяющим направление освещения;
- камера — описывается своим местоположением и направлением обзора.

## 1.2 Анализ методов описания трехмерных моделей

В компьютерной графике выделяют три основных подхода к описанию трехмерных объектов: каркасные, поверхностные и твердотельные модели. Эти методы обеспечивают различные способы представления объектов, что позволяет корректно отображать их форму и размеры на сцене.

### 1.2.1 Каркасная модель

Каркасная модель в трёхмерной графике представляет собой набор вершин и рёбер, которые определяют форму многогранного объекта. Этот тип моделирования является самым простым и обладает существенными ограничениями, основными из которых являются недостаток данных о гранях между линиями и невозможность различить внутреннюю и внешнюю области твердого тела. Несмотря на это, каркасные модели экономичны с точки

зрения памяти и подходят для решения задач, не требующих высокой детализации. Ключевым недостатком каркасных моделей является неоднозначное представление формы объекта [1].

### **1.2.2 Поверхностная модель**

Поверхностная модель в трёхмерной графике описывается через точки, линии и поверхности. Она основывается на предположении, что объекты ограничены поверхностями, которые отделяют их от окружающей среды. Основным недостатком такого подхода является отсутствие информации о материале по обе стороны от поверхности [1].

### **1.2.3 Твёрдотельная модель**

Твёрдотельная модель включает данные о распределении материала по обе стороны объекта. Для корректного описания формы необходимо задавать направление внутренней нормали поверхности.

### **1.2.4 Выбор метода описания модели**

Для решения поставленной задачи наиболее подходящим методом является использование поверхностной модели.

Поверхностная модель описывается при помощи полигональной сетки. Такая сетка включает вершины, рёбра и грани, которые формируют форму объекта в трёхмерном пространстве.

Для представления поверхности жидкости используется сеточная модель, состоящая из массива точек (вершин), соединённых в треугольники. На каждом временном шаге для вычисления координат точек поверхности применяется волновое уравнение. Это позволяет быстро обновлять координаты сетки и их значения, обеспечивая динамическую визуализацию.

## 1.3 Анализ алгоритмов удаления невидимых поверхностей

Удаление невидимых линий и поверхностей является одной из самых сложных задач в области компьютерной графики. Алгоритмы, используемые для удаления невидимых элементов, определяют, какие линии, поверхности или объёмы остаются видимыми для наблюдателя, находящегося в определённой точке пространства.

### 1.3.1 Алгоритм Робертса

Алгоритм Робертса применяется в пространстве объектов и работает исключительно с выпуклыми телами. Если объект не является выпуклым, его предварительно необходимо разбить на составные выпуклые части [2].

Процесс выполнения алгоритма включает четыре этапа:

- Подготовка исходных данных, которая включает составление матриц тел для каждого объекта сцены;
- Удаление рёбер, скрытых самим телом;
- Удаление рёбер, скрытых другими объектами сцены;
- Удаление линий пересечения объектов, скрытых самими телами или другими телами, связанными отношениями пересечения.

Основное преимущество алгоритма заключается в высокой точности вычислений. Однако он работает только с выпуклыми телами и характеризуется ограничением по вычислительной сложности, которая увеличивается пропорционально квадрату количества объектов.

### 1.3.2 Алгоритм Варнока

Алгоритм Варнока действует в пространстве изображения и предназначен для определения, какие грани или их части видимы на сцене. Его принцип основан на разбиении области изображения на более мелкие окна, для которых анализируются связанные многоугольники. Полигоны, чья видимость может быть однозначно определена, отображаются на сцене.

Алгоритм работает рекурсивно, это является его главным недостатком.



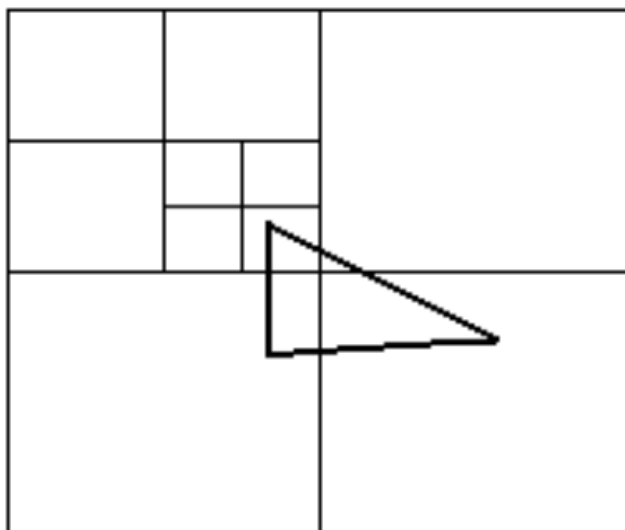


Рисунок 1 – Разбиения в алгоритме Варнока

### 1.3.3 Алгоритм, использующий Z-буфер

Алгоритм, основанный на использовании Z-буфера, представляет собой простой и популярный метод удаления невидимых поверхностей.

В данном подходе используются два буфера: буфер кадра и Z-буфер. Буфер кадра отвечает за хранение атрибутов каждого пикселя в пространстве изображения. Z-буфер представляет собой буфер глубины, который сохраняет значения глубины для каждого видимого пикселя [2].

На начальном этапе Z-буфер инициализируется минимальными значениями, а буфер кадра заполняется атрибутами фона. В ходе работы алгоритма каждый новый пиксель сравнивается со значением глубины в Z-буфере. Если новый пиксель располагается ближе к наблюдателю, его данные записываются в буфер кадра и обновляют Z-буфер.

Преимуществом алгоритма является его простота в реализации. Трудоёмкость работы увеличивается линейно относительно количества объектов на сцене. К недостаткам относится высокая потребность в объёме памяти.

### 1.3.4 Алгоритм обратной трассировки лучей

Алгоритм получил название "обратная трассировка лучей" из-за более эффективного подхода, заключающегося в отслеживании пути лучей от наблюдателя к объекту. Согласно законам оптики, наблюдатель воспринимает объект через лучи света, испускаемые источником, которые попадают на объект и отражаются в направлении глаза наблюдателя.

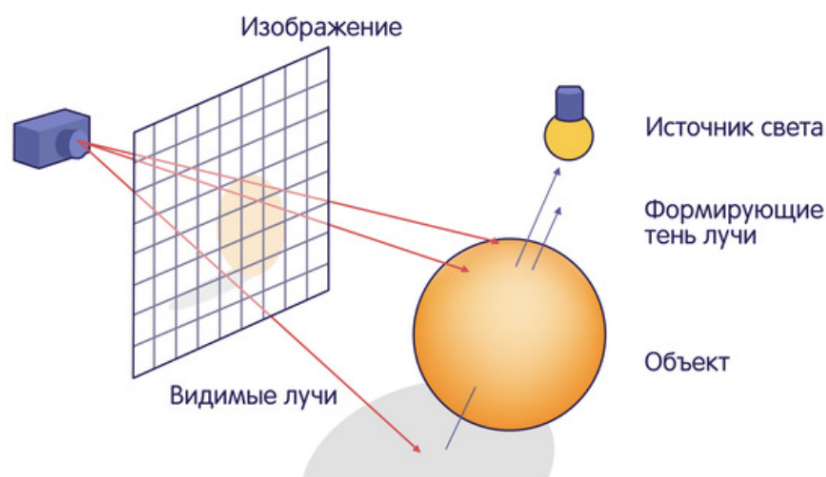


Рисунок 2 – Работы обратной трассировки лучей

Преимуществами являются возможность использования алгоритма в параллельных вычислительных системах и высокая реалистичность получаемого изображения, но необходимо большое количество вычислений.

### 1.3.5 Выбор алгоритма удаления невидимых поверхностей

Поверхность жидкости аппроксимируется треугольниками. Для визуализации поверхности жидкости за основу был взят алгоритм построчного сканирования, использующий Z-буфер.

## 1.4 Анализ алгоритмов закраски

Методы закраски применяются для создания затенения полигонов в трёхмерной модели с учётом освещения сцены. На основе взаимного расположения полигона и источника света определяется уровень освещённости поверхности.

### 1.4.1 Простая закрашка

Простая закрашка подразумевает закрашивание всей грани полигональной модели с одинаковой интенсивностью, рассчитанной по закону Ламберта. Этот метод закраски делает поверхность однотонной.



Рисунок 3 – Пример простой закраски

### 1.4.2 Закраска по Гуро

Метод Гуро устраняет дискретность изменения интенсивности и создает иллюзию гладкой криволинейной поверхности. Он основан на интерполяции интенсивности.



Рисунок 4 – Пример закраски по Гуро

### 1.4.3 Закраска по Фонгу

Закраска Фонга по своей идее похожа на закраску Гуро, но ее отличие состоит в том, что в методе Гуро по всем точкам полигона интерполируется значения интенсивностей, а в методе Фонга — вектора нормалей, и с их помощью для каждой точки находится значение интенсивности.



Рисунок 5 – Пример закраски по Фонгу

### 1.4.4 Выбор алгоритма закраски

Алгоритм закраски Фонга дает наиболее реалистичное изображение, в частности зеркальных бликов. В курсовом проекте будет использоваться метод закраски Фонга и метод закраски Гуро.

## 1.5 Анализ моделей освещения

Существует две основные модели освещения, используемые в компьютерной графике: модель Ламберта и модель Фонга.

### 1.5.1 Модель освещения Ламберта

Модель Ламберта описывает идеальное диффузное освещение. Уровень освещённости в точке поверхности зависит только от плотности света, падающего на неё, и линейно изменяется в зависимости от косинуса угла падения луча. При этом направление, в котором смотрит наблюдатель, не влияет на расчёт, так как диффузно отражённый свет равномерно рассеивается во всех направлениях.

Модель Ламберта является одной из самых простых и базовых моделей освещения, где рассеянная составляющая света вычисляется согласно закону косинусов (закону Ламберта) [3].

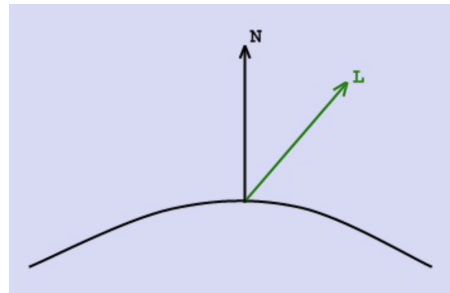


Рисунок 6 – Модель освещения Ламберта

Все векторы берутся единичными. Тогда косинус угла между ними совпадает со скалярным произведением. Формула расчета интенсивности имеет следующий вид:

$$I = I_0 \cdot k_d \cdot \cos(\vec{L}, \vec{N}) \cdot I_d = I_0 \cdot K_d \cdot (\vec{L}, \vec{N}) \cdot I_d \quad (1)$$

Где  $I$  — результирующая интенсивность света в точке;  $I_0$  — интенсивность источника;  $k_d$  — коэффициент диффузного освещения;  $\vec{L}$  — вектор от точки до источника;  $\vec{N}$  — вектор нормали в точке;  $I_d$  — мощность рассеянного освещения.

### 1.5.2 Модель освещения Фонга

Модель представляет собой комбинацию диффузной составляющей и зеркальной составляющей. Работает таким образом, что кроме равномерного освещения на материале может также появиться блик. Отраженная составляющая в точке зависит от того, насколько близки направления от рассматриваемой точки на точку взгляда и отраженного луча. Местонахождение блика на объекте, освещенном по модели Фонга, определяется из закона равенства углов падения и отражения. Если наблюдатель находится вблизи углов отражения, яркость соответствующей точки повышается [3].

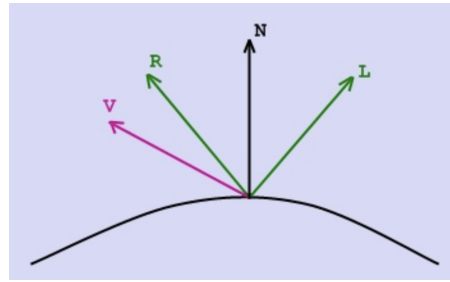


Рисунок 7 – Модель освещения Фонга

Для модели Фонга освещение в точке вычисляется по следующей формуле:

$$I = I_a + I_d + I_s \quad (2)$$

Где  $I$  — результирующая интенсивность света в точке;  $I_a$  — фоновая составляющая;  $I_d$  — рассеянная составляющая;  $I_s$  — зеркальная составляющая;

Падающий и отраженный лучи лежат в одной плоскости с нормалью к отражающей поверхности в точке падения, и эта нормаль делит угол между лучами на две равные части. Таким образом отраженная составляющая освещенности в точке зависит от того, насколько близки направления на наблюдателя и отраженного луча. Это можно выразить следующей формулой [3].

Формула для расчета интенсивности для модели Фонга имеет вид:

$$I = K_a \cdot I_a + I_0 \cdot K_d \cdot (\vec{L}, \vec{N}) \cdot I_d + I_0 \cdot K_s \cdot (\vec{R}, \vec{V})^\alpha \cdot I_s \quad (3)$$

Где  $I$  — результирующая интенсивность света в точке;  $K_a$  — коэффи-

циент фонового освещения;  $I_a$  — интенсивность фонового освещения;  $I_0$  — интенсивность источника;  $K_d$  — коэффициент диффузного освещения;  $\vec{L}$  — вектор от точки до источника;  $\vec{N}$  — вектор нормали в точке;  $I_d$  — интенсивность диффузного освещения;  $K_s$  — коэффициент зеркального освещения;  $\vec{R}$  — вектор отраженного луча;  $\vec{V}$  — вектор от точки до наблюдателя;  $\alpha$  — коэффициент блеска;  $I_s$  — интенсивность зеркального освещения.

### 1.5.3 Выбор модели освещения

Для решения поставленной задачи наиболее оптимальным вариантом является использование модели Фонга в сочетании с закраской по Фонгу. Модель освещения Ламберта будет применяться вместе с методом закраски по Гуро.

### Вывод

В данном разделе был проведён анализ существующих алгоритмов построения изображений, а также выбран наиболее подходящий подход для решения поставленной задачи.

## 2 Конструкторская часть

В данном разделе описаны требования к программному обеспечению, а также рассмотрены структуры данных, алгоритмы и математические уравнения, выбранные для построения сцены.

### 2.1 Требования к программному обеспечению

Программа должна обеспечивать следующий функционал:

- изменение положения камеры и направления её взгляда;
- изменение характеристик волны;
- изменение характеристик капли;
- выбор алгоритма закраски;
- возможность отрисовки полигонов модели.

### 2.2 Описание структур данных

Сцена представляет собой массив моделей, объект камеры и объект источника освещения. Структуры данных для этих объектов следующие:

1) Модель включает в себя следующие данные:

- массив вершин фигуры;
- массив полигонов фигуры;
- массив векторов нормалей к вершинам;
- цвет поверхности;
- матрица аффинных преобразований.

2) Камера содержит:

- положение в пространстве;
- значения углов тангажа и рыскания;
- направление взгляда и ориентацию верха.

3) Источник освещения характеризуется вектором распространения лучей света или его положением в пространстве.



## 2.3 Алгоритм построения изображения

Алгоритм генерации изображения представлен в виде диаграммы, оформленной в соответствии с нотацией IDEF0 и отражающей общую декомпозицию алгоритма [4].

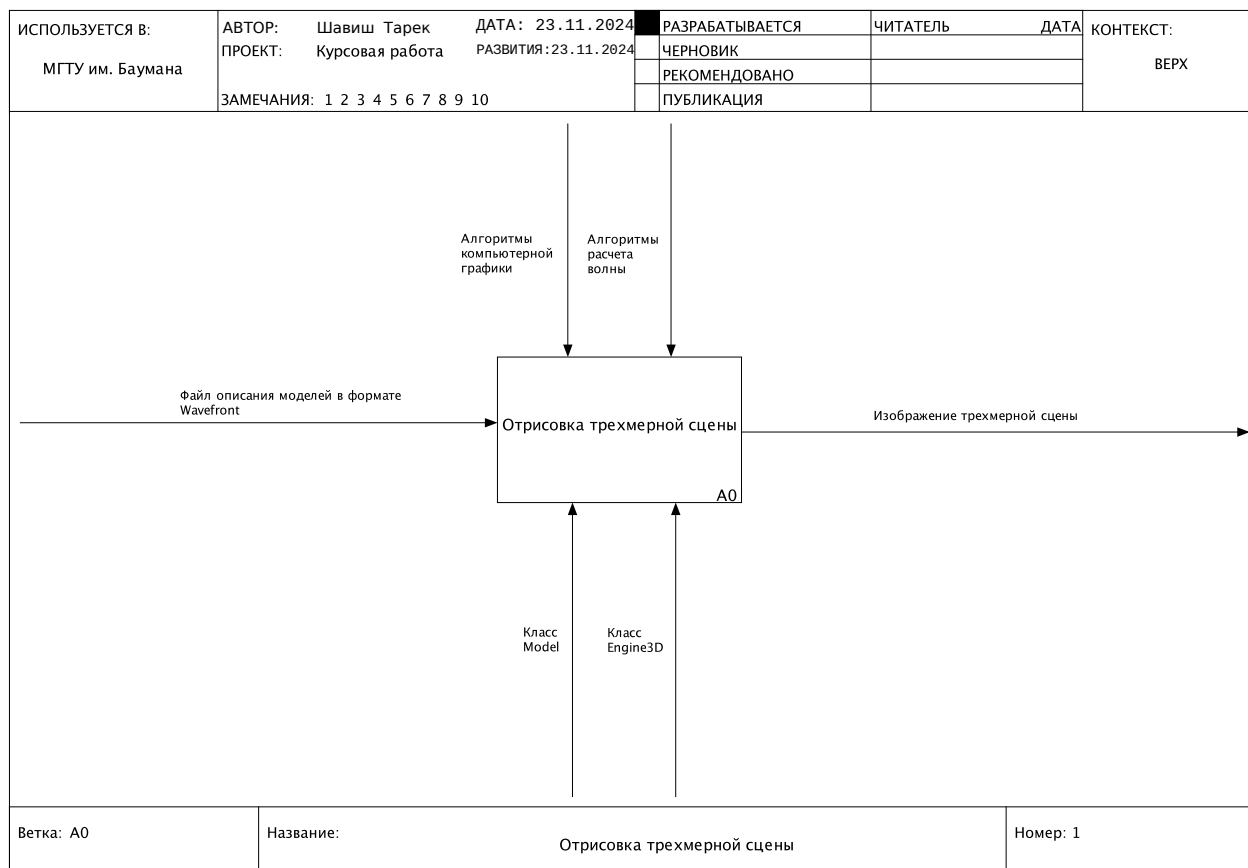


Рисунок 8 – Функциональная схема алгоритма построения изображения, декомпозиция верхнего уровня

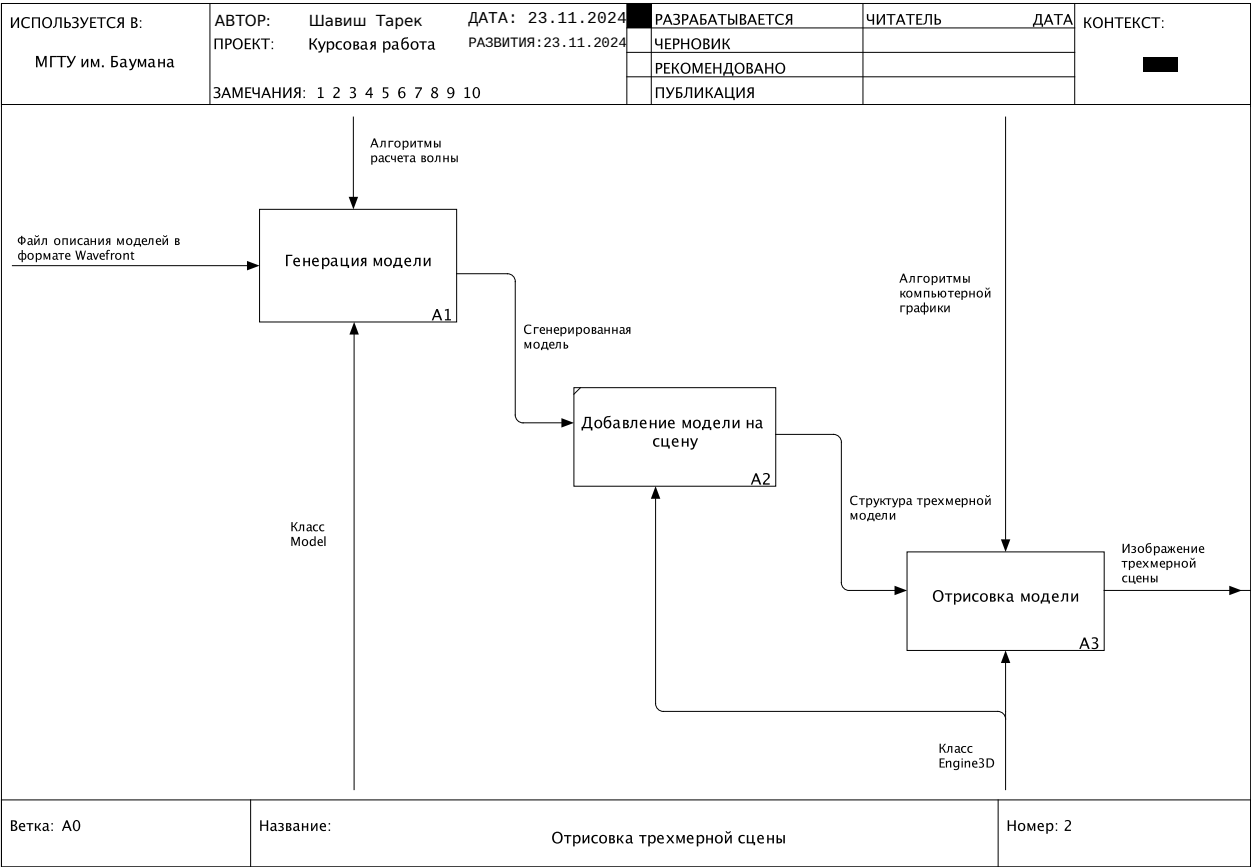


Рисунок 9 – Функциональная схема алгоритма построения изображения, декомпозиция уровня A0

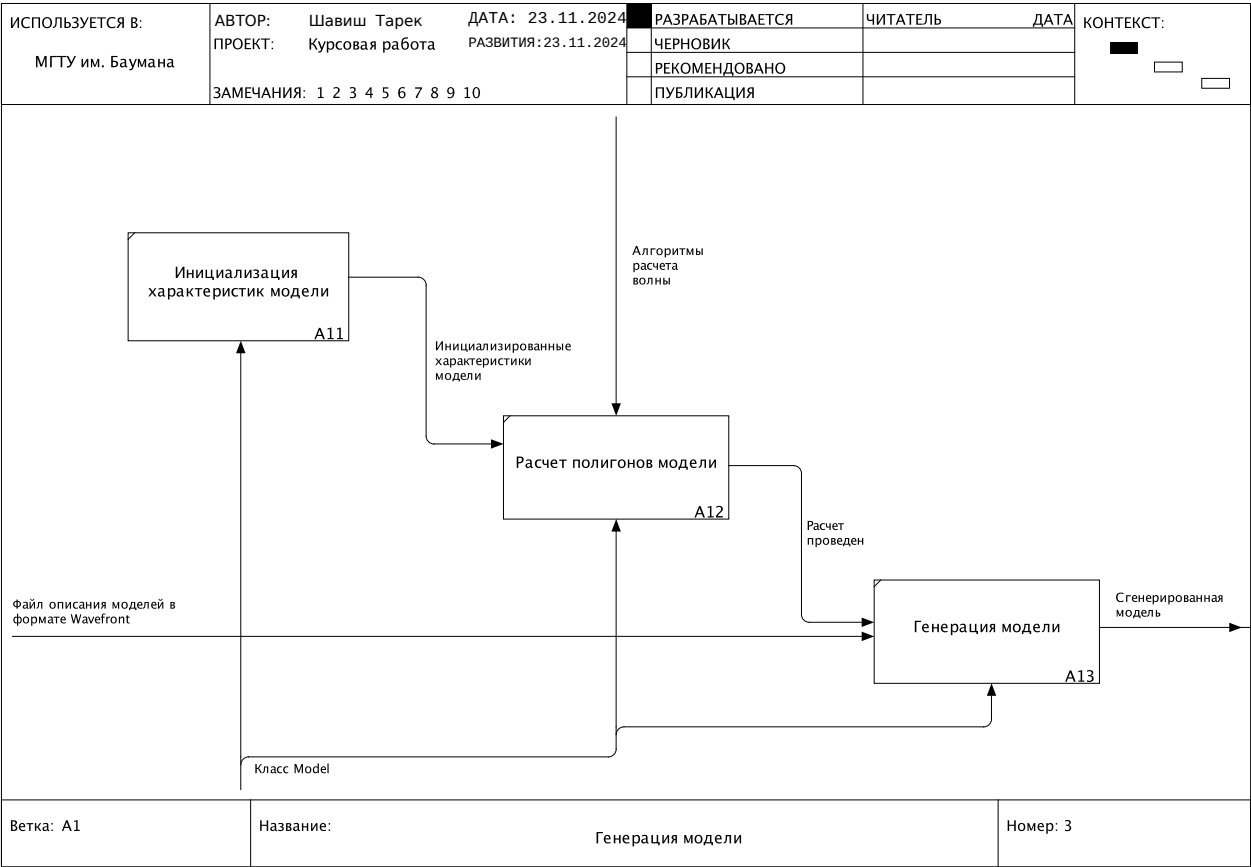


Рисунок 10 – Функциональная схема алгоритма построения изображения, декомпозиция уровня A1



## 2.4 Приведение к пространству камеры

Для перемещения по сцене используется камера, которая определяется точкой положения в пространстве и собственной системой координат, состоящей из трех ортогональных векторов.

Обозначим:

- Point (P) — положение камеры;
- forward (F) — вектор взгляда;
- Up (U) — вектор вверх;
- Right (R) — вектор вправо.

Для перехода в пространство камеры выполняются два этапа, описанных ниже.

- 1) Перенос полигона в отрицательную стороны от камеры на расстояние Point с помощью матрицы переноса [5]:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -Px & -Py & -Pz & 1 \end{pmatrix} \quad (1)$$

- 2) Преобразование полигона к системе координат камеры при помощи матрицы поворота [5]:

$$\begin{pmatrix} Rx & Ux & Fx & 0 \\ Ry & Uy & Fy & 0 \\ Rz & Uz & Fz & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (2)$$

Управление камеры производится с помощью изменения углом Эйлера. Обозначим  $\alpha$  — угол поворота вокруг оси ОУ (тангаж) и  $\beta$  — угол поворота вокруг оси ОХ (рыскание). Тогда координаты вектора направления камеры

можно вычислить по следующим формулам:

$$F_x = \cos(\alpha) \cdot \cos(\beta) \quad (3)$$

$$F_y = \sin(\alpha) \quad (4)$$

$$F_z = \cos(\alpha) \cdot \sin(\beta) \quad (5)$$

## 2.5 Невидимые грани

Для оптимизации процесса построения изображения можно значительно сократить временные затраты, отбрасывая нелицевые грани моделей. Это позволяет избежать растеризации невидимых полигонов.

Для определения видимости грани используется следующая формула:

$$(\vec{N}, \vec{V}) = \begin{cases} \geq 0, & \text{если грань невидима} \\ < 0, & \text{если грань видима} \end{cases}, \quad (6)$$

где  $\vec{N}$  — вектор внешней нормали к грани модели,  $\vec{V}$  — вектор от камеры до любой точки грани.

## 2.6 Алгоритм, использующий Z-буфер

Для растеризации треугольного полигона необходимо вычислить глубину каждой его точки. Затем нужно провести сравнение значения глубины точки с глубиной, хранящейся в Z-буфере. Если глубина пикселя меньше, это означает, что пиксель находится ближе к камере и должен быть растеризован. В этом случае значение глубины пикселя записывается в Z-буфер, а его интенсивность вычисляется и заносится в буфер кадра.

Полная схема алгоритма Z-буфера представлена на рисунке 12.

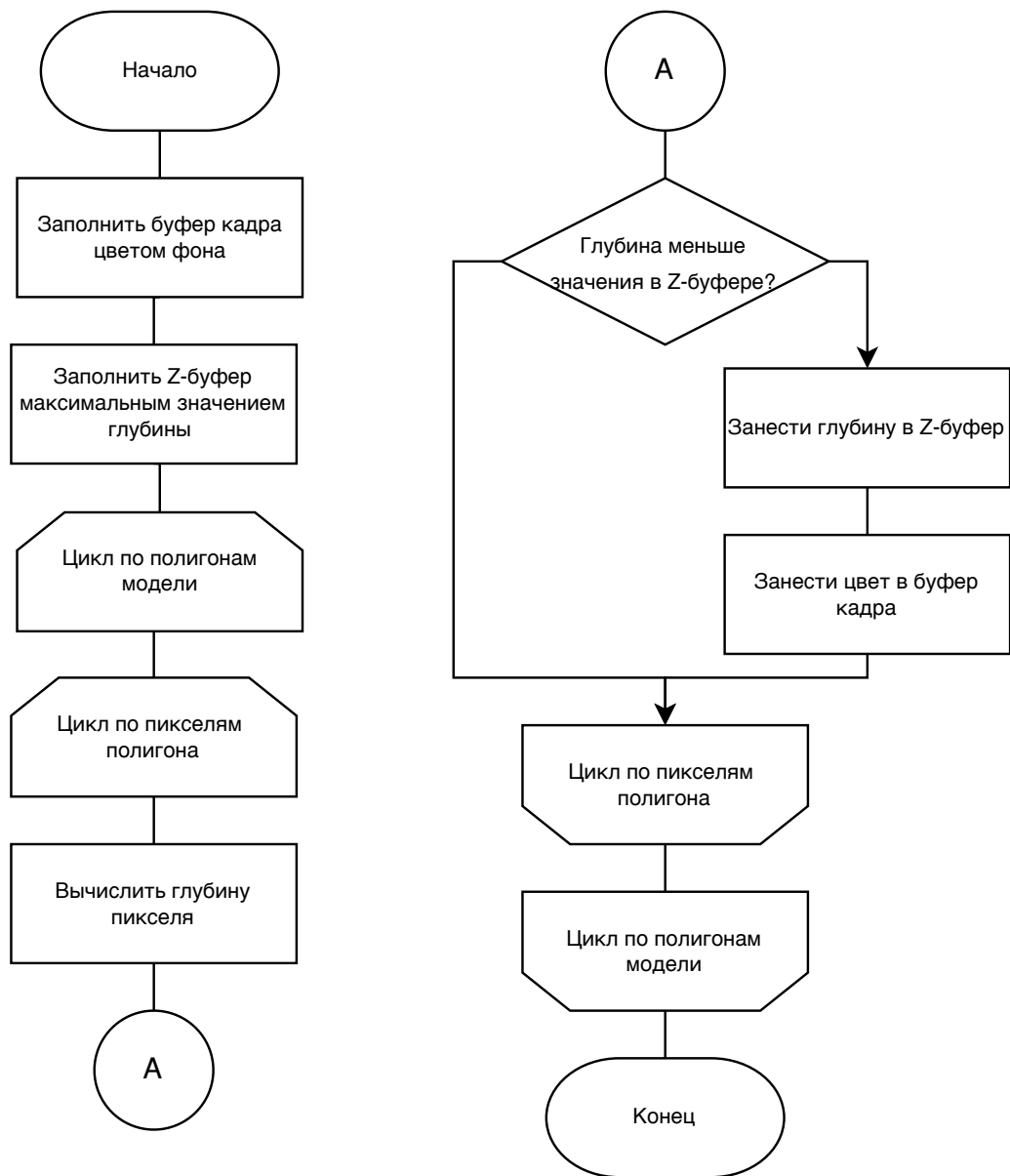


Рисунок 12 – Схема алгоритма, использующего Z-буфер

## 2.7 Описание уравнения бегущей волны

Бегущие волны моделируются свободными гармоническими колебаниями. Для перехода в нормализованное пространство, в котором значение функции лежит в пределах от 0 до 1, выполняется преобразование:

$$f(x) = \frac{\sin x + 1}{2} \quad (7)$$

С целью получения реалистичного изображения волн необходимо принять во внимание тот факт, что волны могут иметь большую крутизну и остроту пиков.

$$f(x) = \frac{\sin x + 1}{2} \quad (8)$$

Необходимо также учитывать направление волны. Ее моделирование происходит в двумерном поле высот, поэтому требуется определять движение волн в обоих направлениях. Вектор направления должен быть параллельным плоскости невозмущенной поверхности жидкости, т. е. иметь координату  $z$ , равную нулю. [6].

$$S = Dir(x, y) * Pos(x, y) \quad (9)$$

где  $Dir(x, y)$  — вектор направления волны,  $Pos(x, y)$  — вектор координат точки.

Учет частоты происходит в формуле в соответствии с известным соотношением определения частоты  $f$ :

$$f = \frac{2\pi}{\lambda} \quad (10)$$

где  $\lambda$  — длина волны.

Тогда получим:

$$S = Dir(x, y) * Pos(x, y)f \quad (11)$$

Для учета скорости волны необходимо определить фазовую постоянную в соответствии с выражением [6].

$$\psi = vf = \frac{2v\pi}{\lambda} \quad (12)$$

Окончательное выражение для вычисления координаты точки  $z$  на поверхности жидкости в зависимости от ее координат  $x$ ,  $y$  и времени имеет следующий вид:

$$f(x, y, t) = A \left( \frac{\sin S + 1}{2} \right)^k \quad (13)$$

где  $S = Dir(x, y) * Pos(x, y)f + t\psi$



## 2.8 Описание круговых волн

Учесть этот эффект можно с помощью переопределения вектора направления в каждой точке поверхности и изменения координаты точки смещением начала координат в точку центра круговой волны [6].

$$D(x, y) = \left( \frac{(x, y) - C}{|(x, y) - C|} \right), \quad (14)$$

где  $C$  — точка начала распространения круговой волны;  $X, Y$  — координаты рассматриваемой точки.

Измененные значения подставляются в формулу бегущей волны.

## 2.9 Описание вычисления нормалей

Используя уравнения поверхностей, определим нормали в любой ее точке. Уравнение нормали к поверхности в точке  $(x, y, z)$ :

$$N(x, y, z) = NB(x, y, z) \otimes NT(x, y, z) \quad (15)$$

где  $NB(x, y, z)$  и  $NT(x, y, z)$  — частные производные по  $x$  и  $y$ ;

$$NB(x, y, z) = \frac{\partial f(x, y, t)}{\partial x} \quad (16)$$

$$NT(x, y, z) = \frac{\partial f(x, y, t)}{\partial y} \quad (17)$$

где  $f(x, y, t)$  — итоговое уравнение поверхности;

$$\frac{\partial f(x, y, t)}{\partial x} = 0.5 Dir_x f A \left( \frac{\sin S + 1}{2} \right)^{k-1} \cos S, \quad (18)$$

$$\frac{\partial f(x, y, t)}{\partial y} = 0.5 Dir_y f A \left( \frac{\sin S + 1}{2} \right)^{k-1} \cos S. \quad (19)$$

Задавая векторы нормалей проекциями на координатные оси, получим [6].

$$NB(x, y, z) = \left( \frac{\partial x}{\partial x}, \frac{\partial y}{\partial x}, \frac{\partial f(x, y, t)}{\partial x} \right) = \left( 1, 0, \frac{\partial f(x, y, t)}{\partial x} \right), \quad (20)$$

$$NT(x, y, z) = \left( \frac{\partial x}{\partial y}, \frac{\partial y}{\partial y}, \frac{\partial f(x, y, t)}{\partial y} \right) = \left( 0, 1, \frac{\partial f(x, y, t)}{\partial y} \right). \quad (21)$$

## Вывод

В данном разделе были представлены требования к программному обеспечению, рассмотрены структуры данных, алгоритмы и математические уравнения, выбранные для построения сцены.

## 3 Технологическая часть

В данной части рассматривается выбор средств реализации, описывается структура классов программы и приводится интерфейс программного обеспечения.

### 3.1 Технологическая часть

Для написания данного курсового проекта был выбран язык Python [7].

Для разработки интерфейса и работы с пикселями изображения был выбран фреймворк Qt [8].

Используемые инструменты обладают полным функционалом для разработки, профилирования и отладки необходимой программы, а также создания графического пользовательского интерфейса.

### 3.2 Структура программы

Разработанная программа состоит из следующих классов. Базовые математические классы:

- Mat4x4 – класс матриц;
- Vector3D – класс векторов трехмерного пространства.

Классы для работы с моделями:

- Model – базовый класс моделей;
- Wave – класс модели поверхности жидкости.

Вспомогательные классы:

- Camera – класс камеры с возможностью перемещения по сцене;
- ZBuffer – класс z-буфера;
- RebderWidget – класс интерфейса;
- EngineBase – класс для отрисовки треугольников;
- Engine3D – класс для отрисовки моделей.

На рисунке 13 представлена схема разработанных классов.

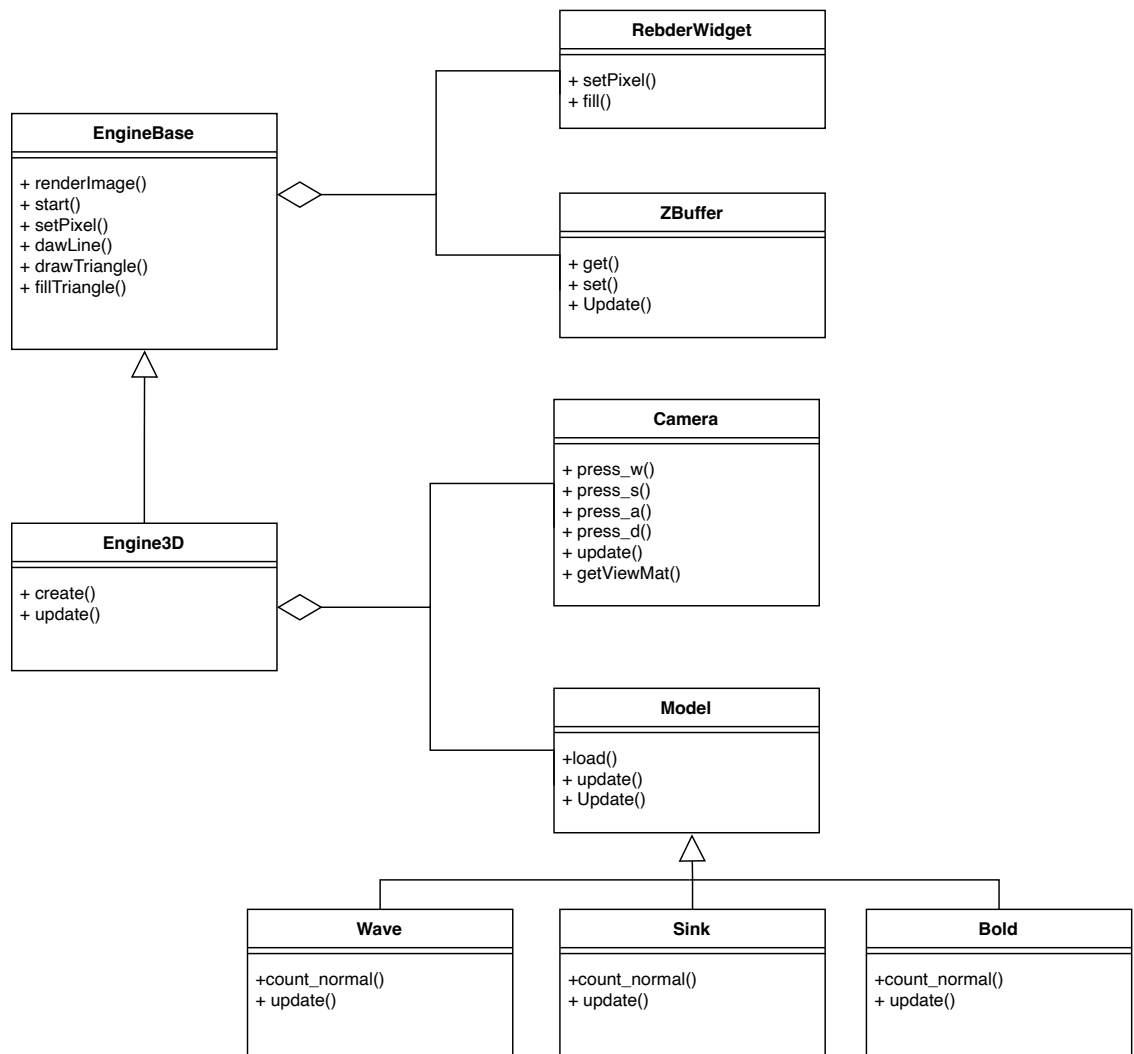


Рисунок 13 – Схема классов программы

### 3.3 Реализация алгоритмов

На листингах 1 – 4 приведены реализации алгоритмов

```
def gen_poligon(self)
    start = -0.85
    stop = 0.85
    num = 11
    step = (stop - start) / (num - 1)
    x = [start + i * step for i in range(num)]

    z_def = 0

    for i in range((len(x) - 1)):
        for j in range((len(x) - 1)):
            mid = num // 2
            if i < mid and j < mid or i >= mid and j >= mid:
                t1 = Triangle(Vec3D(x[i], x[j], z_def),
                               Vec3D(x[i], x[j + 1], z_def),
                               Vec3D(x[i + 1], x[j], z_def))
                self.append(t1)

                t2 = Triangle(Vec3D(x[i + 1], x[j], z_def),
                               Vec3D(x[i], x[j + 1], z_def),
                               Vec3D(x[i + 1], x[j + 1],
                                     z_def))
                self.append(t2)
            else:
                t1 = Triangle(Vec3D(x[i], x[j], z_def),
                               Vec3D(x[i + 1], x[j + 1],
                                     z_def),
                               Vec3D(x[i + 1], x[j], z_def),
                               )
                self.append(t1)

                t2 = Triangle(Vec3D(x[i], x[j], z_def),
                               Vec3D(x[i], x[j + 1], z_def),
                               Vec3D(x[i + 1], x[j + 1],
                                     z_def),
                               )
                self.append(t2)
```

```
def update(self):
    self.time += 1 / self.fps

    for tri in self.tris:
        for p in tri.p:
            p.z = 0
            direction = Vec3D.sub(p, self.c)
            direction.normalize()
            s = Vec3D.dp(direction, p) * self.frequency -
                self.time * self.fi
            p.z = self.amplitude * ((math.sin(s) + 1) * 0.5)
                ** self.k

    self.count_normal()

def count_normal(self):
    for tri in self.tris:
        for i in range(3):
            direction = Vec3D.sub(tri.p[i], self.c)
            direction.normalize()
            s = Vec3D.dp(direction, tri.p[i]) *
                self.frequency - self.time * self.fi

            df_dx = 0.5 * self.k * direction.x *
                self.frequency * self.amplitude * \
                    ((math.sin(s) + 1) * 0.5) ** (self.k -
                        1) * math.cos(s)

            df_dy = 0.5 * self.k * direction.y *
                self.frequency * self.amplitude * \
                    ((math.sin(s) + 1) * 0.5) ** (self.k -
                        1) * math.cos(s)

            tri.n[i] = Vec3D(df_dx, -df_dy, -1).normalize()
```

### Листинг 3 – Реализация Z-буфера

```
class ZBuffer:
    def __init__(self, width, height):
        self.width = width
        self.height = height

        self.array = [float('+inf')] * width * height

    def get(self, x, y):
        if 0 <= x < self.width and 0 <= y < self.height:
            return self.array[x + y * self.width]

        return float('+inf')

    def set(self, x, y, value):
        if 0 <= x < self.width and 0 <= y < self.height:
            self.array[x + y * self.width] = value
```

### Листинг 4 – Модель освещения Фонга

```
light_dir = Vec3D.sub(light_pos, p)
light_dir.normalize()

eye_dir = Vec3D.sub(self.camera.point, p)
eye_dir.normalize()

reflect_dir = Vec3D.sub(Vec3D.mul_value(n, 2 * Vec3D.dp(n,
    light_dir)), light_dir).mul(-1)
reflect_dir.normalize()

kd = 0.75
ks = 0.25
spect_alpha = 8

ambient = 0.25
diffuse = Vec3D.dp(n, light_dir)
spect = Vec3D.dp(reflect_dir, eye_dir) ** spect_alpha
intensity = ks * spect + kd * diffuse + kd * ambient
```

Листинг 5 – Алгоритм закраски по Фонгу (интерполяция нормалей)

```
def fill_triangle_fong(self, rp0, rp1, rp2, rn0, rn1, rn2, red,
    green, blue):
    total_height = p2.y - p0.y
    for i in range(total_height):
        second_half = i > p1.y - p0.y or p1.y == p0.y
        segment_height = p2.y - p1.y if second_half else p1.y -
            p0.y
        alpha = i / total_height
        beta = (i - (p1.y - p0.y)) / segment_height if
            second_half else i / segment_height
        a = Vec3D.add(p0, Vec3D.sub(p2, p0).mul(alpha))
        u = Vec3D.dist(p0, a) / Vec3D.dist(p2, p0)
        n_a = Vec3D.add(Vec3D.mul_value(n0, 1 - u),
            Vec3D.mul_value(n2, u)).normalize()
        b = Vec3D.add(p1, Vec3D.sub(p2, p1).mul(beta)) if
            second_half else Vec3D.add(p0, Vec3D.sub(p1,
                p0).mul(beta))
        w = Vec3D.dist(p1, b) / Vec3D.dist(p2, p1) if
            second_half else Vec3D.dist(p0, b) / Vec3D.dist(p1,
                p0)
        n_b = Vec3D.add(Vec3D.mul_value(n1, 1 - w),
            Vec3D.mul_value(n2, w)).normalize() if second_half
            else Vec3D.add(Vec3D.mul_value(n0, 1 - w),
                Vec3D.mul_value(n1, w)).normalize()
        if a.x > b.x:
            a, b = b, a
            n_a, n_b = n_b, n_a
        for j in range(int(a.x), int(b.x) + 1):
            phi = 1 if b.x == a.x else (j - a.x) / (b.x - a.x)
            p = Vec3D.add(a, Vec3D.sub(b, a).mul(phi))
            p.int()
            n = n_a.copy()
            if not Vec3D.equal(a, b):
                t = Vec3D.dist(a, p) / Vec3D.dist(b, a)
                n = Vec3D.add(Vec3D.mul_value(n_a, 1 - t),
                    Vec3D.mul_value(n_b, t)).normalize()
```



Листинг 6 – Алгоритм закрашки по Гуро (интерполяция интенсивностей)

```
def fill_triangle_guro(self, rp0, rp1, rp2, rn0, rn1, rn2, red,
    green, blue):
    total_height = p2.y - p0.y

    for i in range(total_height):
        second_half = i > p1.y - p0.y or p1.y == p0.y
        segment_height = p2.y - p1.y if second_half else p1.y -
            p0.y
        alpha = i / total_height
        beta = (i - (p1.y - p0.y)) / segment_height if
            second_half else i / segment_height

        a = Vec3D.add(p0, Vec3D.sub(p2, p0).mul(alpha))
        a.int()

        ia = i0 + (i2 - i0) * alpha

        b = Vec3D.add(p1, Vec3D.sub(p2, p1).mul(beta)) if
            second_half else \
            Vec3D.add(p0, Vec3D.sub(p1, p0).mul(beta))
        b.int()

        ib = i1 + (i2 - i1) * beta if second_half else \
            i0 + (i1 - i0) * beta

        if a.x > b.x:
            a, b = b, a
            ia, ib = ib, ia

        for j in range(int(a.x), int(b.x) + 1):
            phi = 1 if b.x == a.x else (j - a.x) / (b.x - a.x)

            p = Vec3D.add(a, Vec3D.sub(b, a).mul(phi))
            p.int()

            ip = ia + (ib - ia) * phi
```

### 3.4 Интерфейс программного обеспечения

При запуске программы в левой части определен виджет сцены, в правой части интерфейса определены разделы для настройки сцены (рисунок 14).

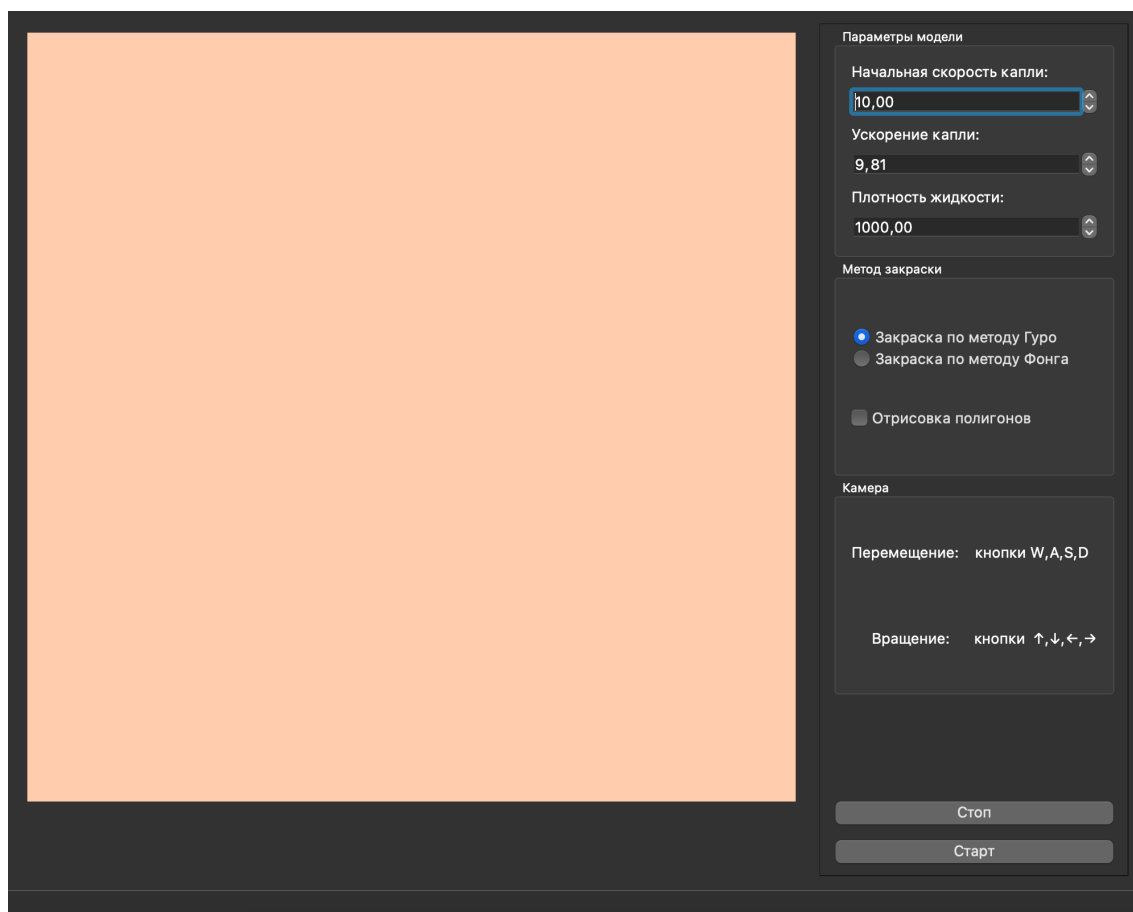


Рисунок 14 – Графический интерфейс программы

Для создания модели пользователю необходимо выбрать параметры начальной скорости капли, её ускорения и плотности жидкости в разделе «Параметры модели» (рисунок 15).

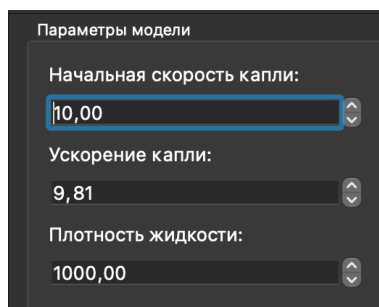


Рисунок 15 – Параметры модели

Далее пользователю необходимо выбрать метод закраски модели (Фонг, Гуро) и указать необходимость вывода полигонов в разделе «Метод закраски» (рисунок 16).

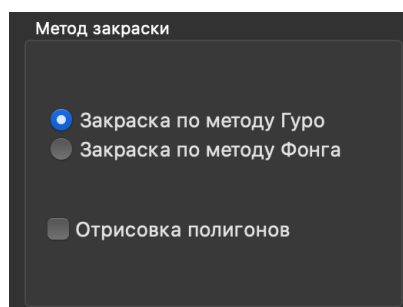


Рисунок 16 – Метод закраски

Управление камерой описано в разделе «Камера» (рисунок 17).

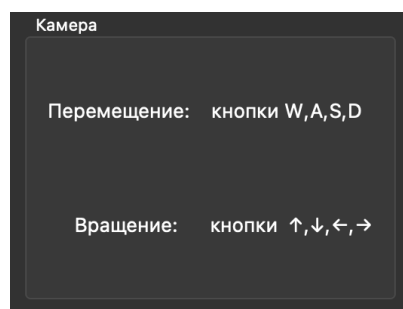


Рисунок 17 – Камера

Для создания модели на сцене необходимо нажать кнопку «Старт». Меню с настройками модели будет заблокировано, камера станет доступна для изменения своего расположения. Для изменения параметров модели или метода закрашки необходимо нажать кнопку «Стоп».

На рисунке 18 приведен пример работы программы.

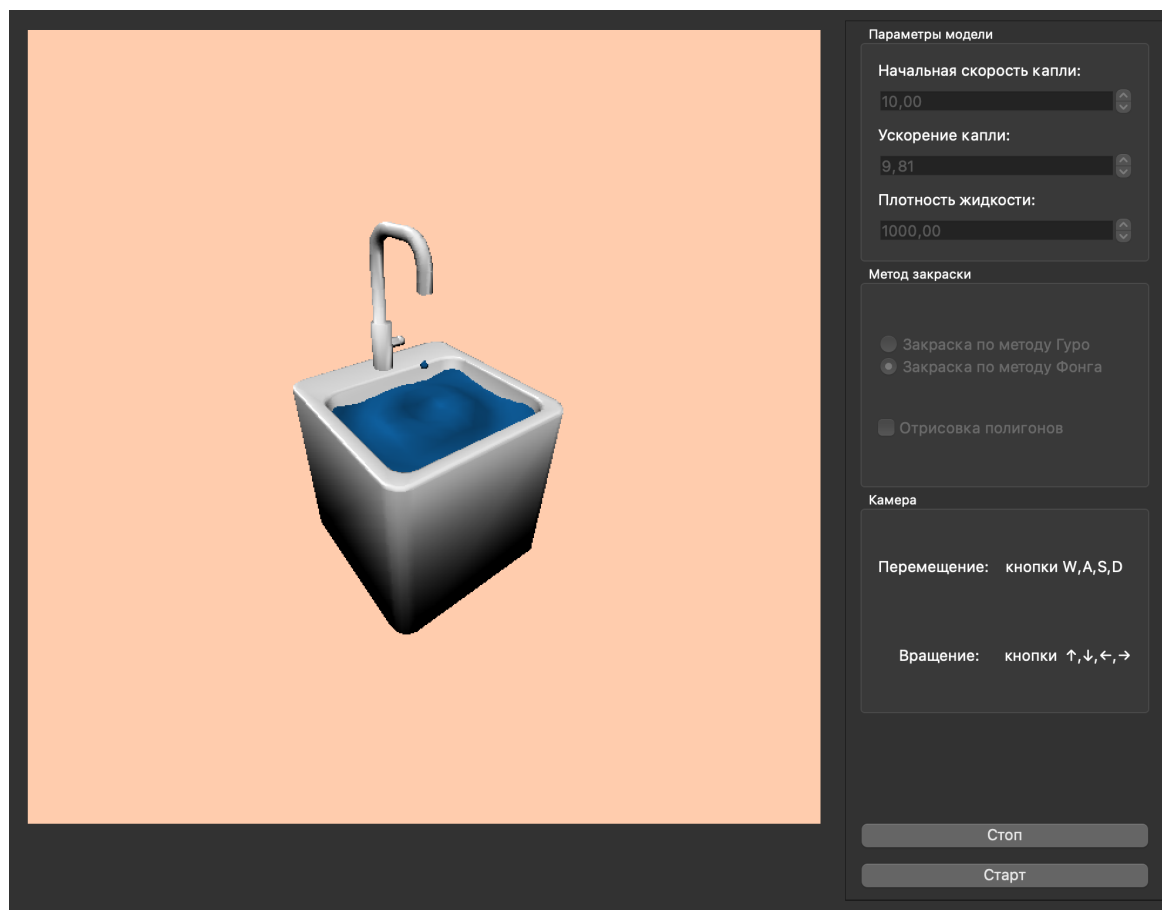


Рисунок 18 – Тестовая сцена

## Вывод

В данном разделе были выбраны средства реализации, описаны структуры классов программы, описаны модули, а также рассмотрен интерфейс программы.

## 4 Исследовательская часть

### 4.1 Постановка исследования

Целью исследования является определение времени пересчета полигонов поверхности жидкости и времени закраски поверхности жидкости по методам Гуро и Фонга от количества полигонов.

### 4.2 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование:

- операционная система MacOS Ventura 13.5.2;
- 16 ГБ оперативной памяти;
- процессор 2,6 ГГц 6-ядерный Intel Core i7 [9].

Во время тестирования устройство было подключено к сети электропитания, нагружено приложениями окружения и самой системой тестирования.

### 4.3 Время выполнения алгоритмов

Алгоритмы тестировались при помощи функции *process\_time()* из библиотеки *time* [10] языка *Python* [7]. Данная функция возвращает текущее значение системного процессорного времени в секундах.

Замеры времени для каждого количества полигонов проводились 20 раз. В качестве результата взято среднее время работы алгоритма.

В таблице 1 приведены результаты замера времени пересчета полигонов поверхности жидкости (в микросекундах).

Таблица 1 – Результаты измерений времени пересчета полигонов поверхности жидкости

Количество полигонов, штук	Время, микросекунды
8	213
32	785
72	1 753
128	3 112
200	4 843
288	6 988
392	9 506
512	12 387
648	15 489
800	18 840
968	22 818
1 152	26 544
1 352	30 861
1 568	35 217

По таблице 1 был построен график на рисунке 19

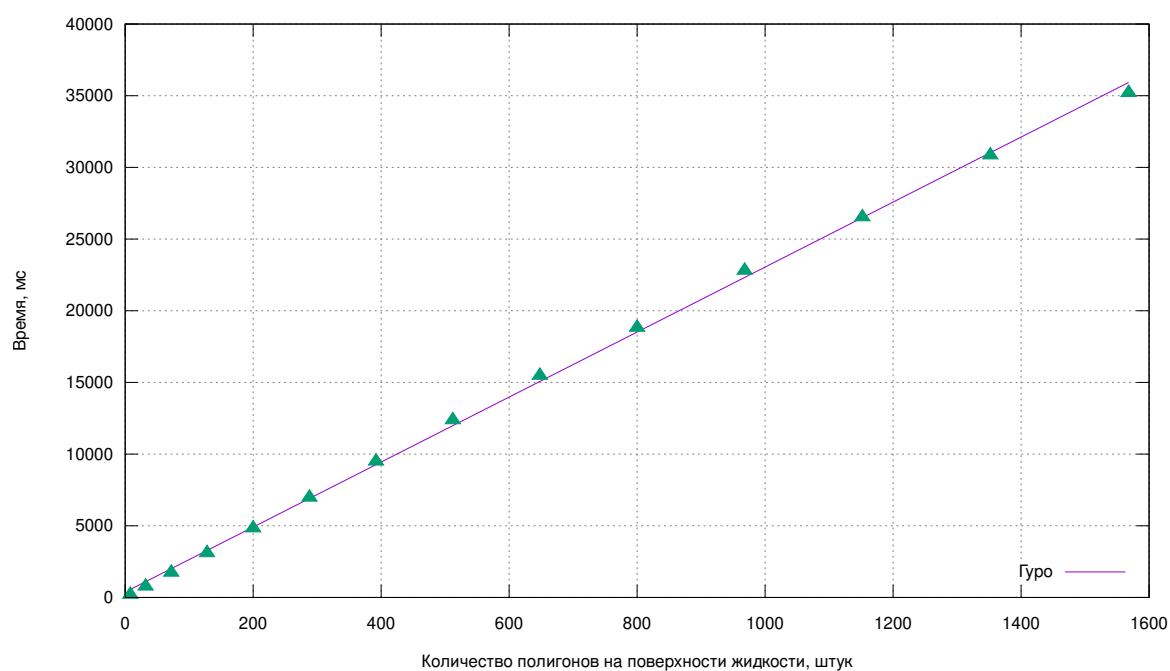


Рисунок 19 – График зависимости измерений времени пересчета полигонов поверхности жидкости от количества полигонов

Как видно из графика, время пересчета полигонов поверхности жидкости линейно зависит от количества полигонов.

В таблице 2 приведены результаты замера времени закрашки поверхности жидкости по методам Гуро и Фонга (в микросекундах).

Таблица 2 – Результаты измерений времени закрашки поверхности жидкости по методам Гуро и Фонга

Количество полигонов, штук	Время, микросекунды	
	Гуро	Фонг
8	134 711	920 098
32	158 717	963 961
72	184 150	1 011 580
128	206 442	1 072 216
200	250 857	1 135 146
288	300 670	1 245 661
392	369 820	1 356 393
512	440 931	1 477 853
648	523 032	1 574 341
800	604 238	1 722 877
968	704 506	1 871 794
1 152	812 945	2 009 417
1 352	935 350	2 162 179
1 568	1 054 775	2 308 075

По таблице 2 был построен график на рисунке 20

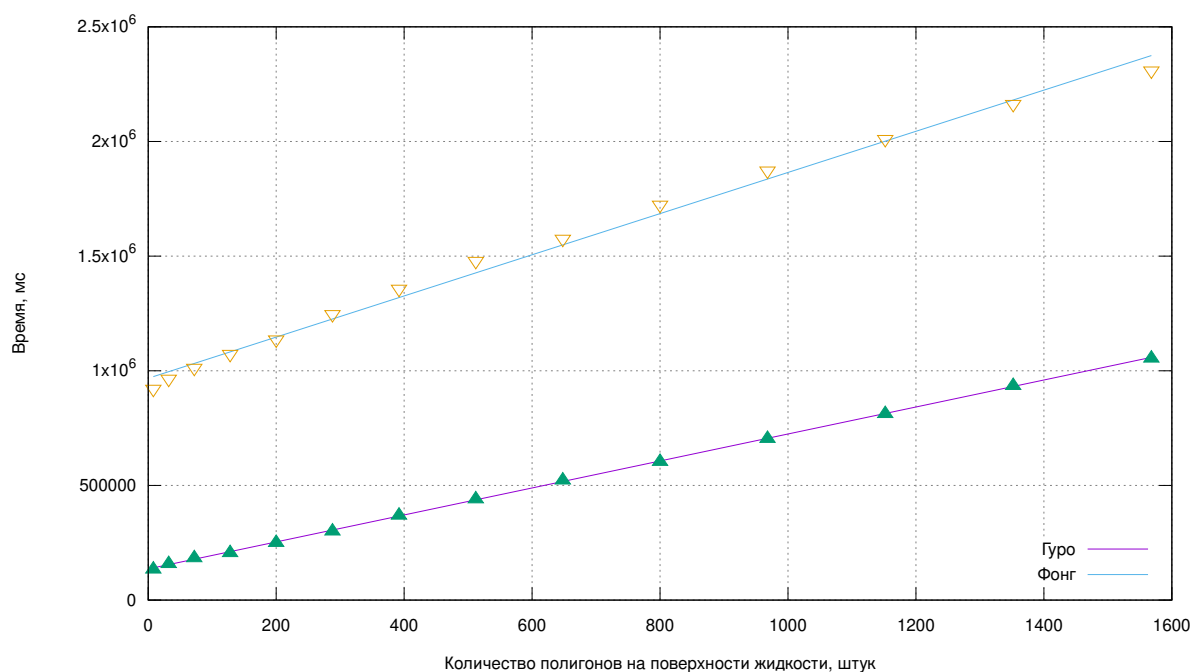


Рисунок 20 – График зависимости измерений времени закрашки поверхности жидкости по методам Гуро и Фонга от количества полигонов

Как видно из графика, время закрашки поверхности жидкости по методам Гуро и Фонга линейно зависит от количества полигонов, при этом метод закрашки по Фонгу более чем в 2 раза медленнее закрашки по методу Гуро. Это обусловлено тем, что интенсивность каждой точки полигона в методе Фонга рассчитывается отдельно по вектору нормали и вектору направления источника освещения в данной точке полигона. В закрашке по методу Гуро данная операция производится только в вершинах полгона, далее полученные значения интенсивности интерполируются.

## Вывод

В данном разделе были исследованы результаты измерения времени пересчета полигонов поверхности жидкости и времени закрашки поверхности жидкости по методам Гуро и Фонга от количества полигонов.



## Заключение

В ходе выполнения курсовой работы была успешно достигнута поставленная цель: разработано программное обеспечение для моделирования генерации круговых волн на поверхности жидкости.

Для достижения цели были выполнены следующие задачи:

- проведён анализ существующих алгоритмов в области компьютерной графики;
- выбраны наиболее подходящие алгоритмы для решения поставленной задачи;
- определены средства реализации программного обеспечения;
- разработано программное обеспечение с реализацией выбранных алгоритмов и структур данных;
- проведены замеры временных характеристик разработанного программного обеспечения.

Разработанное программное обеспечение можно дополнительно улучшить, добавив возможность наложения нескольких волн, что позволит наблюдать явление интерференции.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Методы трехмерного моделирования [Электронный ресурс]. — 2022. — Режим доступа: [https://studopedia.ru/19\\_307536\\_metodi-trehmernogo-modelirovaniya-karkasnoe-modelirovanie-poverhnostnoe-tverdotelnoe-modelirovanie-tipi-poverhnostey-chto-predstavlyayut-s-soboy-trehmernie-ob-ekti.html](https://studopedia.ru/19_307536_metodi-trehmernogo-modelirovaniya-karkasnoe-modelirovanie-poverhnostnoe-tverdotelnoe-modelirovanie-tipi-poverhnostey-chto-predstavlyayut-s-soboy-trehmernie-ob-ekti.html) (дата обращения: 29.11.2024).
2. *Роджерс Д.* Алгоритмические основы машинной графики. — 1989.
3. Простые модели освещения [Электронный ресурс]. — 2022. — Режим доступа: <https://cgraph.ru/node/435> (дата обращения: 29.11.2024).
4. Методология IDEF0 [Электронный ресурс]. — 2022. — Режим доступа: [https://studme.org/87184/ekonomika/metodologiya\\_idef0](https://studme.org/87184/ekonomika/metodologiya_idef0) (дата обращения: 29.11.2024).
5. Placing a Camera: the LookAt Function [Электронный ресурс]. — Режим доступа: <https://www.scratchapixel.com/lessons/mathematics-physics-for-computer-graphics/lookat-function/framing-lookat-function.html> (дата обращения: 29.11.2024).
6. *Куров А. В.* Моделирование волн на поверхности жидкости. — МГТУ им. Н.Э. Баумана, 2013.
7. Welcome to Python [Электронный ресурс]. — Режим доступа: <https://www.python.org> (дата обращения: 29.11.2024).
8. All Qt Documentation [Электронный ресурс]. — Режим доступа: <https://doc.qt.io/all-topics.html> (дата обращения: 29.11.2024).
9. Intel [Электронный ресурс]. — Режим доступа: <https://www.intel.com/content/www/us/en/products/details/processors/core/i7.html> (дата обращения: 29.11.2024).
10. time — Time access and conversions [Электронный ресурс]. — Режим доступа: <https://docs.python.org/3/library/time.html#functions> (дата обращения: 29.11.2024).