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:*

## *«Программа моделирования движения и взаимодействия объектов, составляющих машину Голдберга»*

Студент ИУ7-52Б (Группа) _____ (Подпись, дата) Алахов А. Г. (И. О. Фамилия)

Руководитель курсовой работы _____ (Подпись, дата) Кивва К. А. (И. О. Фамилия)

*2022 г.*

# СОДЕРЖАНИЕ

# ВВЕДЕНИЕ

Одной из основных задач компьютерной графики является генерация реалистичных изображений. Алгоритмы решающие эту задачу, как правило, получают на вход сцену, заданную некими параметрами. Такими могут быть положение источников света, их свойства, угол обзора, точка наблюдения, остальные элементы сцены с заданными свойствами отражения и поглощения падающих на них лучей. Правдоподобность кадра достигается с помощью моделирования законов физики, основными из которых являются преломление и отражение луча света.

Целью данной работы является построение системы трехмерных объектов, образующую «машину Голдберга», состоящей из шара, математического маятника (шара, закреплённого на нити), костей домино и кнопки, с возможностями анимации и изменения положения камеры. Для достижения поставленной цели требуется выполнить следующие задачи:

- изучить различные подходы к построению реалистичных сцен;
- изучить модели движения и взаимодействия твёрдых тел;
- выбрать наиболее подходящие под условие задачи алгоритмы;
- определить структуры данных;
- реализовать алгоритмы;
- провести тестирование;
- создать версию алгоритма, которая может выполняться с использованием нескольких потоков одновременно;
- сравнить временные показатели обычной и распараллеленной версий.

# 1. Аналитический раздел

В данном разделе будут описаны различные методы построения трехмерных сцен и выбран наиболее подходящий для решения поставленной задачи.

## 1.1. Задача преобразования трехмерных объектов в изображение на сцене

Для решения задачи синтеза трехмерного объекта, заданного с помощью формального описания, в визуальное представление на растровом экране требуется решить ряд подзадач. Ниже эти подзадачи рассмотрены подробнее.

### 1.1.1. Удаление невидимых линий

Задача удаления невидимых линий и поверхностей является одной из наиболее сложных в машинной графике. Алгоритмы удаления невидимых линий и поверхностей служат для определения линий ребер, поверхностей или объемов, которые видимы или невидимы для наблюдателя, находящегося в заданной точке пространства [1].

### 1.1.2. Закраска

Если при построении полигональной поверхности каждая грань закрашивается с одной интенсивностью, то модель освещения создает изображение, состоящее из отдельных многоугольников. С помощью методов закраски можно получить более сглаженное изображение.

### 1.1.3. Учет свойств материала и модели освещения

Требуется учитывать различные свойства материалов, из которых состоят объекты сцены. Материалы могут различаться по способу и коэффициенту отражения. Отраженный от объекта свет может быть диффузным или зеркальным.

### 1.1.4. Нахождение теней

На пути луча от источника света к объекту может быть препятствие. В таком случае, для построения реалистичного изображение требуется наложить тень на такой объект.

## 1.2. Методы решения задачи построения трехмерных объектов

Все методы построения трехмерных объектов состоят из трех основных этапов: нахождение теней, удаление невидимых линий и поверхностей и закраска согласно выбранной модели освещения.

Если источник света и наблюдатель находятся в одной точке, то теней видно не будет. В ином случае, задача построения теней сводится к задаче удаления невидимых линий и поверхностей от источника света.

Существует два основных метода закраски. При закраске Гуро сначала определяется интенсивность вершин закрашиваемого многоугольника в соответствии с выбранной моделью освещения, а затем с помощью билинейной интерполяции вычисляется интенсивность каждого пиксела на сканирующей строке. Закраска Фонга отличается от Гуро тем, что вместо интенсивности интерполируется вектор нормали.

Далее будут рассмотрены различные методы удаления невидимых линий и поверхностей.

### 1.2.1. Алгоритм Варнока

Алгоритм анализирует сцену в пространстве изображений. Идея алгоритма заключается в разбиении сцены на области до тех пор, пока не будет пустой или ее содержимое будет просто для визуализации. Конкретная реализация алгоритма зависит от способа разбиения окна. В оригинальной версии окно разбивается на четыре равных подокна [1].

На рисунке 1.1 изображено построение двух наложенных прямоугольников с помощью алгоритма Варнока.

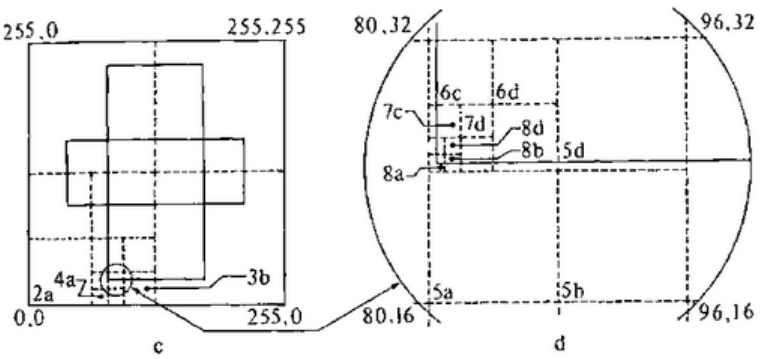



Рисунок 1.1 – Пример работы алгоритма Варнока

### 1.2.2. Алгоритм, использующий z-буфер

Алгоритм, использующий z-буфер, - один из простейших алгоритмов удаления невидимых поверхностей. Работает алгоритм в пространстве изображений. Идея z-буфера является простым обобщением идеи о буфере кадра. Буфер кадра используется для запоминания атрибутов (интенсивности) каждого пиксела в пространстве изображений. В процессе работы глубина или значение z каждого нового писксела, который нужно занести в буфер кадра, сравнивается с глубиной того пиксела, который уже занесен в z-буфер. Если это сравнение показывает, что новый пиксел расположен впереди пиксела, находящегося в буфере кадра, то новый пиксел заносится в этот буфер и, кроме того, производится корректировка z-буфера новым значением z. Если же сравнение дает противоположный результат, то никаких действий не производится [1].

На рисунке 1.2 изображено построение двух наложенных фигур с помощью алгоритма z-буфера.

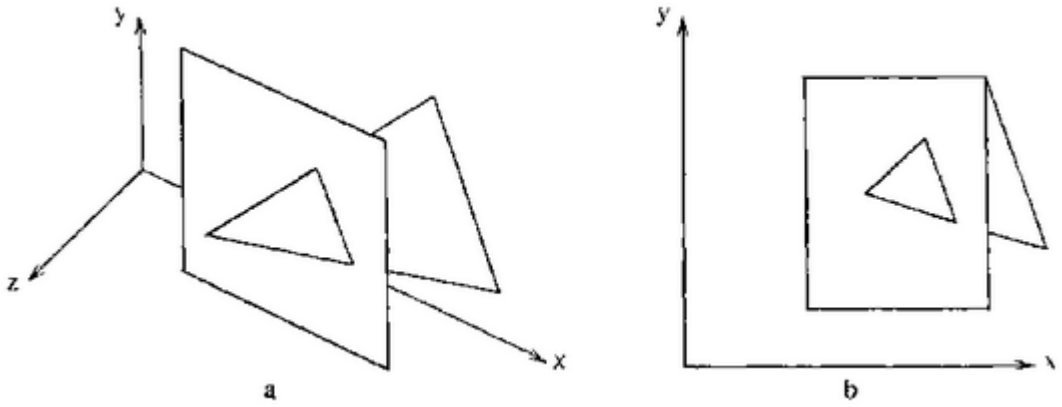



Рисунок 1.2 – Пример работы алгоритма, использующего z-буфер

### 1.2.3. Алгоритм художника

Алгоритм художника является одним из алгоритмов, использующих список приоритетов. Такие алгоритмы предварительно подвергают все объекты сцены сортировке по глубине. Если такой список окончательный, то никакие два элемента не будут взаимно перекрывать друг друга. В таком случае можно последовательно заносить объекты сцены в буфер кадра и получить правильное изображение, так как более близкие к наблюдателю

объекты будут затирать информацию о более дальних в буфере кадра. Такая реализация алгоритма перестает работать в случае циклического перекрытия двух или более объектов сцены, как на рисунке 1.3. В таком случае, нужно разбивать объект на несколько частей [1].

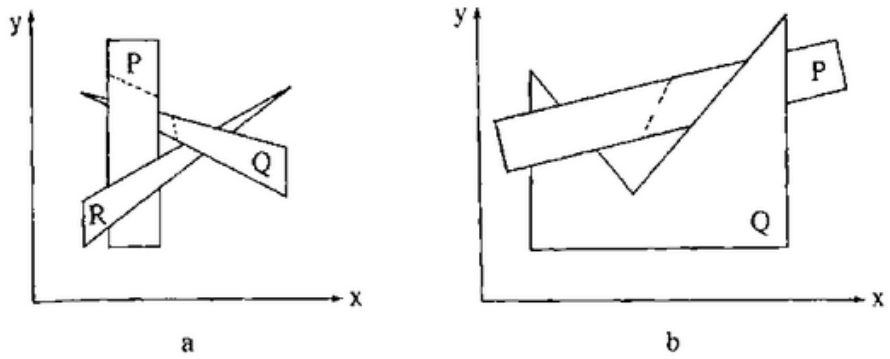



Рисунок 1.3 – Пример цикличного перекрытия объектов

### 1.2.4. Алгоритм трассировки лучей

Алгоритм работает в пространстве изображений. Главная идея алгоритма заключается в том, что наблюдатель видит лучи, которые попали в него от объектов. Так как далеко не все лучи от объектов доходят до наблюдателя было предложено анализировать лучи в обратном направлении, то есть от наблюдателя к объектам сцены. Луч света может отразиться от поверхности, преломиться или пройти через нее. Обычно трассировка заканчивается, когда луч пересекает поверхность видимого непрозрачного объекта [1].

На рисунке 1.4 изображена трассировка луча, используемая в данном алгоритме.

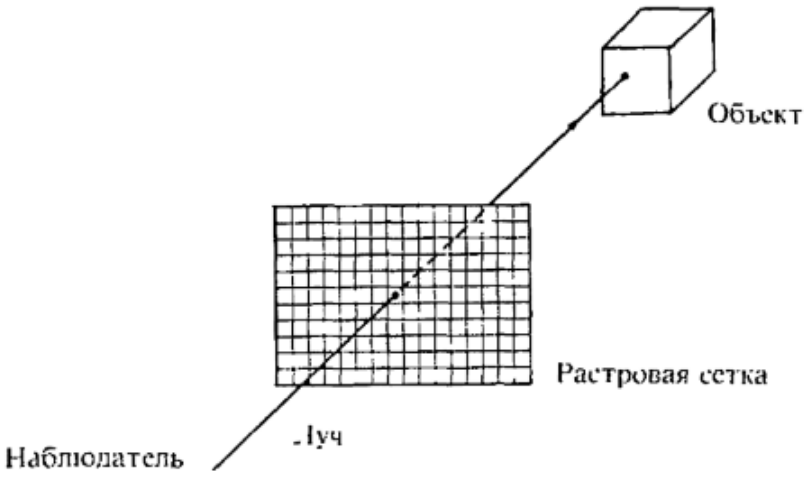



Рисунок 1.4 – Пример трассировки лучей

### 1.2.5. Сравнение методов удаления невидимых линий и поверхностей

Метод трассировки лучей, описанный в пункте 1.2.4, требует множество вычислений, так как требуется искать пересечение каждого луча, число которых пропорционально размеру сетки растра, со всеми объектами сцены. Такой подход сказывается на производительности и быстродействии алгоритма. Для решения поставленной задачи требуется найти более производительное решение, так как при движении объектов сцена каждый раз должна пересчитываться.

Таковым является алгоритм, использующий z-буфер, описанный в пункте 1.2.2. Этот метод обладает рядом преимуществ:

- вычислительная трудоемкость алгоритма не более, чем линейна [1];
- может обрабатывать сцены любой сложности;
- прост в реализации;
- есть возможность распараллелить все вычисления для полигонов до момента обращения к буферу кадра.

Недостатком этого алгоритма является большой объем требуемой памяти, который пропорционален размеру сцены. Этот недостаток нивелируется большим объемом доступной оперативной памяти в современных ЭВМ.

С учетом всех преимуществ и недостатков для решения задачи удаления невидимых линий и поверхностей был выбран алгоритм, использующий z-буфер.

## 1.3. Методы решения задачи закраски

Существует два наиболее распространённых метода закраски, которые могут использоваться в зависимости от требований к сглаживанию.

### 1.3.1. Закраска Фонга

В данном методе нормаль в каждой точке многоугольника получается посредством череды линейных интерполяций нормалей в вершинах. Перед

использованием закраски Фонга необходимо усреденение векторов нормалей в вершинах полигонов.

Преимуществом закраски Фонга является лучшая работа с зеркальными отражениями, при этом недостатком закраски Фонга является большое количество вычислений.

### 1.3.2. Закраска Гуро

Закраска Гуро похожа на закраску Фонга за одним отличием - в закраске Гуро для каждой вершины вычисляются интенсивности. Затем с помощью череды линейных интерполяций находятся интенсивности в каждой точке многоугольника.

Преимущетсва закраски Гуро:

- меньшее количество вычислений по сравнению с закраской Фонга;
- лучшая работа с диффузными отражениями.

### 1.3.3. Выбор метода закраски

В данной работе было решено использовать закраску Гуро, так как она дает лучшие результаты при работе с диффузными отражениями и требует меньшего количества вычислений по сравнению с закраской Фонга.

## 1.4. Свойства материала и модель освещения

Для модели освещения будет введён ряд ограничений:

- свет распространяется от точечного источника;
- поверхности моделей обладают диффузным отражением;
- источник испускает свет белого цвета;
- интенсивность отраженного от поверхности света не зависит от ее удаленности от источника.

Диффузное отражение света происходит, когда свет как бы проникает под поверхность объекта, поглощается, а затем вновь испускается. При этом положение наблюдателя не имеет значения, так как диффузно отраженный

свет рассеивается равномерно по всем направлениям. [1] Интенсивность света от точечного источника света можно найти по закону косинусов Ламберта (1.1)

$$I_{dif} = I_l k_d \cos\theta, 0 \leq \theta \leq \pi/2, \tag{1.1}$$

где $I_{dif}$ - интенсивность отраженного света, $I_l$ - интенсивность точечного источника, $k_d$ - коэффициент диффузного отражения, который может принимать значения от 0 до 1, $\theta$ - угол между направлением света и нормалью к поверхности.

Если источников света несколько, то при расчете интенсивности суммируются значения полученные от каждого источника в отдельности (1.2)

$$I = \sum_{1}^{m} I_{difi}, \tag{1.2}$$

где m - число источников света.

На объекты реальных сцен падает еще и рассеянный свет, отраженный от окружающей обстановки. Рассеянному свету соответствует распределенный источник. Поскольку для расчета таких источников требуются большие вычислительный затраты, в машинной графике они заменяются на коэффициент рассеяния - константу, которая входит в формулу линейной комбинации с членом Ламберта (1.3)

$$I = I_\alpha k_\alpha + \sum_{1}^{m} I_{difi}, \tag{1.3}$$

где $I_\alpha$ - интенсивность рассеянного света, $k_\alpha$ - коэффициент диффузного отражения рассеянного света

## 1.5. Методы решения задачи построения теней

При использовании алгоритма трассировки лучей, рассмотренного ранее, построение теней происходит по ходу выполнения алгоритма: пиксел будет затенен, если испускаемый луч попадает на объект, но не попадает в источник света [1].

Данный алгоритм не подходит для решения поставленной задачи, так как при проведении анализа алгоритмов удаления невидимых ребер он не был выбран в качестве реализуемого.

В качестве реализуемого алгоритма была выбрана модификация алгоритма с использованием z-буфера путем добавления вычисления теневого z-буфера из точки наблюдения, совпадающей с источником света [1].

Такой подход позволит не усложнять структуру программы, а также избежать проблем адаптации двух различных методов друг к другу, а, следовательно, уменьшить время отладки программного продукта.

## 1.6. Физические закономерности, использующиеся при моделировании

Выбранная «машина Голдберга» выполняет задачу нажатия на кнопку путём нескольких промежуточных действий, а именно: придания скорости сфере, столкновения сферы и математического маятника, столкновения математического маянтика и кости домино, последовательное столкновение десяти костей домино, нажатие на кнопку последней костью.

Для моделирования движения и взаимодействия объектов использовались перечисленные далее физические закономерности.

### 1.6.1. Равноускоренное движение

Равноускоренное движение — движение тела, при котором его ускорение $\vec{a}$ постоянно по модулю и направлению. Скорость при этом определяется формулой

$$\vec{v}(t) = \vec{v}_0 + \vec{a}t,$$

где $\vec{v}_0$ — начальная скорость тела, $t$ — время. [2]

### 1.6.2. Вращательное движение

Вращательное движение — вид механического движения. При вращательном движении материальная точка описывает окружность. При вращательном движении абсолютно твёрдого тела все его точки описывают окружности, расположенные в параллельных плоскостях. Центры всех окружностей лежат при этом на одной прямой, перпендикулярной к плоскостям окружностей и называемой осью вращения.

Вращение характеризуется углом $\varphi$ , измеряющимся в градусах или ради-

анах, угловой скоростью $\omega = \dfrac{d\varphi}{dt}$ (измеряется в рад/с) и угловым ускорением $\epsilon = \dfrac{d^2\varphi}{dt^2}$ (единица измерения — рад/с$^2$).

Зависимость угловой скорости вращения тела от его линейной скорости [2]:

$$\omega = \frac{v}{R}.$$

### 1.6.3. Математический маятник

Математический маятник — осциллятор, представляющий собой механическую систему, состоящую из материальной точки на конце невесомой нерастяжимой нити или лёгкого стержня и находящуюся в однородном поле сил тяготения. Другой конец нити (стержня) обычно неподвижен.

Математический маятник колеблется только в какой-то одной плоскости (вдоль какого-то выделенного горизонтального направления). При колебаниях в одной плоскости маятник движется по дуге окружности радиуса $L$. [2]

### 1.6.4. Закон сохранения механической энергии

В физике механическая энергия описывает сумму потенциальной и кинетической энергий, имеющихся в компонентах механической системы. Механическая энергия — это энергия, связанная с движением объекта или его положением, способность совершать механическую работу; это энергия движения и сопровождающего его взаимодействия.

Закон сохранения механической энергии утверждает, что в изолированной системе, где действуют только консервативные силы, полная механическая энергия сохраняется, что может быть записано в следующем виде:

$$\frac{mv^2}{2} + mgh = const,$$

где $m$ — масса тела, $g$ — ускорение свободного падения, $h$ — высота положения центра масс тела над произвольно выбранным нулевым уровнем. [2]

### 1.6.5. Абсолютно упругий удар

Абсолютно упругий удар — модель соударения, при которой полная кинетическая энергия системы сохраняется. В классической механике при этом пренебрегают деформациями тел. Соответственно, считается, что энергия на деформации не теряется, а взаимодействие распространяется по всему телу мгновенно.

Для математического описания абсолютно упругих ударов используется закон сохранения механической энергии и закон сохранения импульса.

- $m_1\vec{v}_1 + m_2\vec{v}_2 = m_1\vec{v}_1' + m_2\vec{v}_2'$.
- $\frac{m_1 v_1^2}{2} + \frac{m_2 v_2^2}{2} = \frac{m_1 {v_1'}^2}{2} + \frac{m_2 {v_2'}^2}{2}$.

Здесь $m_1,\ m_2$ — массы первого и второго тел.$\vec{v}_1,\ \vec{v}_1'$ — скорость первого тела до, и после взаимодействия. $\vec{v}_2,\ \vec{v}_2'$ — скорость второго тела до, и после взаимодействия. [2]

## 1.7. Вывод

В данном разделе были описаны различные методы построения трехмерных сцен и выбран наиболее подходящий для решения поставленной задачи. Также были рассмотрены все необходимые для моделирования физические закономерности.

# 2. Конструкторский раздел

В данном разделе представлены схемы алгоритмов z-буфера и его распаралелленной версии, а также структуры данных, которые в них используются.

## 2.1. Структуры данных

Объекты в системе задаются файлами-описателями, которые включают в себя информацию о точках и поверхностях, из которых состоит модель. Для хранения этой информации нужны структуры Dot и Facet.

Dot — структура, которая описывает точки, из которых состоят объекты. Её полями являются 3 вещественных переменных x, y и z, которые содержат значение соответствующей координаты.

Facet — структура, которая описывает поверхности, из которых состоят объекты. Её полями являются 3 целочисленных переменных, которые содержат индекс в массиве точек объекта, и целочисленная переменная, которая отвечает за цвет поверхности.

Для хранения геометрических свойств объектов сцены используется структура PolModel. Она содержит 2 переменные типа Dot, которые отвечают за хранение центра вращения объекта и центра, относительно которого отсчитывается расстояние до других объектов, массив элементов типа Dot для хранения точек, из которых состоит объект, и массив из элементов типа Facet для хранения поверхностей объекта. Также структура держит вещественные переменные, отвечающие за скорость, ускорение, радиус при вращении, радиус при рассёте расстояния до других объектов, масса, текущий угол поворота, ширина.

Для хранения источников освещения используется структура Illuminant. Её полями являются 2 вещественные переменныу xAngle, yAngle, которые содержат углы относительно текущей точки наблюдения.

## 2.2. Распараллеленный алгоритм z-буфера

В распараллеленной версии алгоритма каждая поверхность анализируется в отдельном потоке. Распараллеливание вычислений реализовано благодаря добавлению двух новых переменных `start` и `finish`, которые указывают на диапазон поверхностей, которые необходимо обработать каждому потоку.

Схема алгоритма приведена на рисунке 2.1.

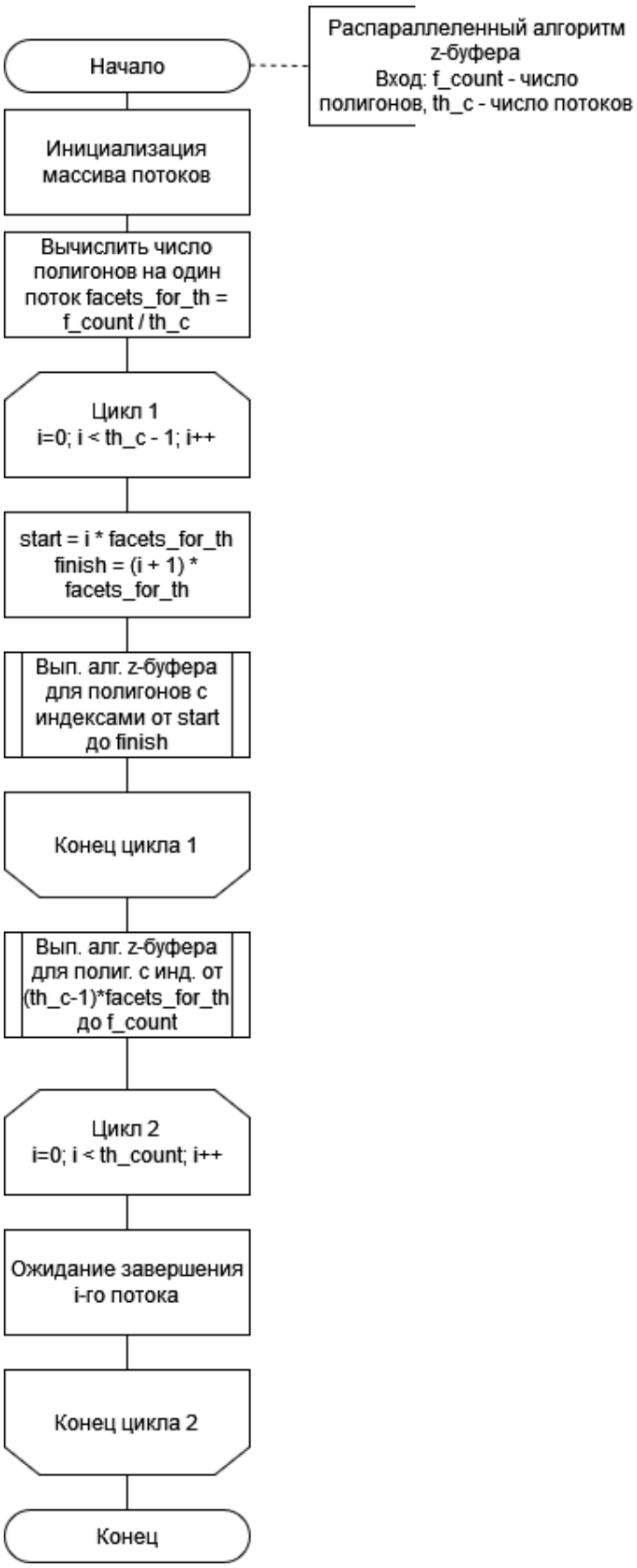



Рисунок 2.1 – Схема распаралеленного алгоритма z-буфера

## 2.3. Алгоритмы движения и взаимодействия объектов

Далее рассмотрены алгоритмы, реализующие физические закономерности для изменения состояния объектов сцены для следующего кадра. Алгоритмы разработаны из условия, что между кадрами проходит единица времени.

### 2.3.1. Алгоритм движения сферы

При движении сферы используются закономерности равноускоренного движения и вращательного движения. В начале объект перемещается на расстояние, равное его скорости. Затем находится угловая скорость объекта и осуществляется его поворот. Последним шагом является изменение скорости на значение ускорения. Схема алгоритма приведена на рисунке 2.2.

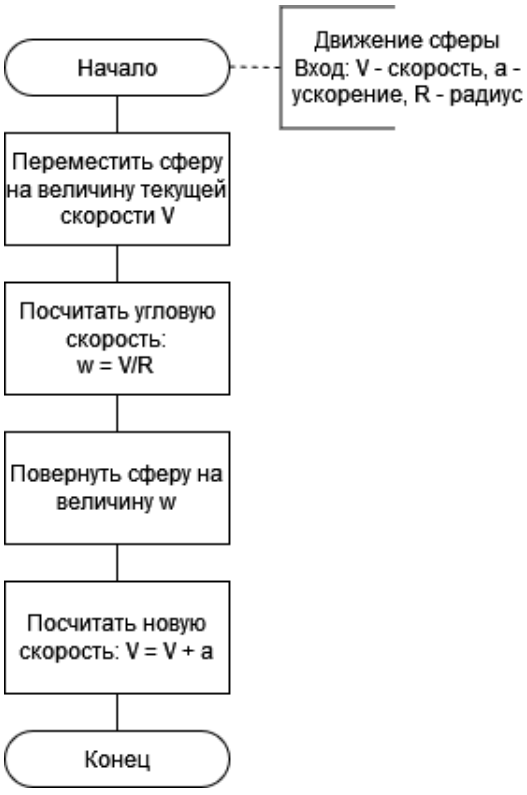



Рисунок 2.2 – Схема алгоритма движения сферы

### 2.3.2. Алгоритм движения маятника

При начале движения маятника известна переданная ему кинетическая энергия, поэтому все последующие скорости маятника находятся исходя из закона сохранения механической энергии. В начале находится угловая скорость объекта и осуществляется его поворот. Затем определяется новая скорость объекта и направление его движения. Схема алгоритма приведена на рисунке 2.3.

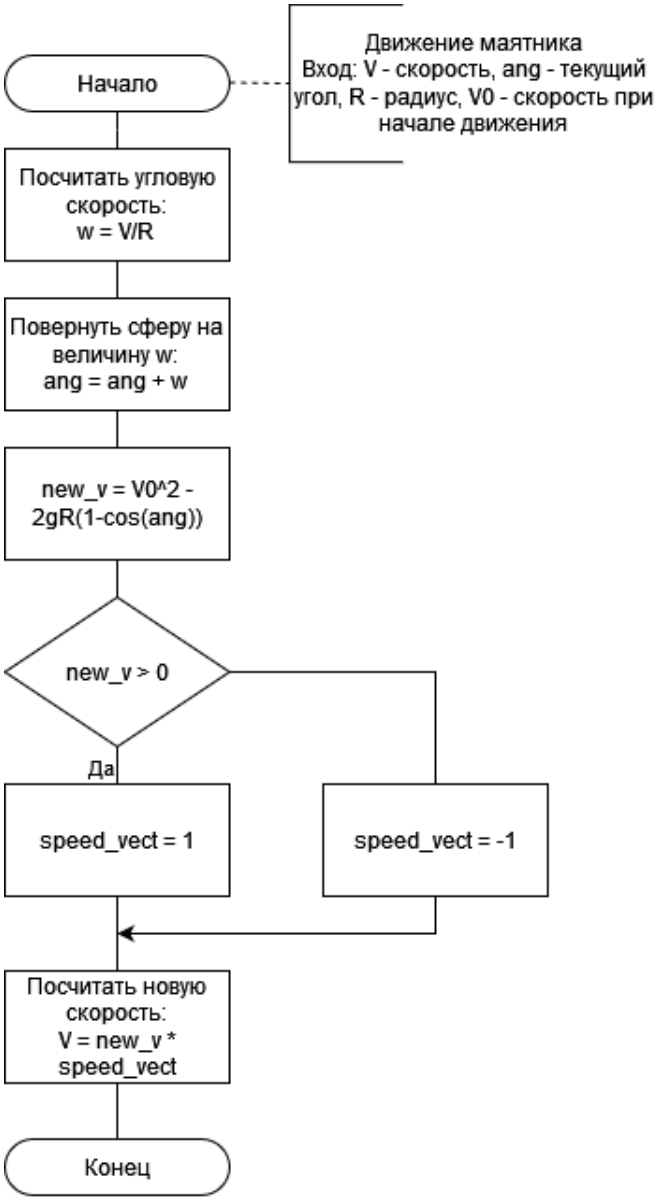



Рисунок 2.3 – Схема алгоритма движения маятника

### 2.3.3. Алгоритм движения кнопки

При движении кнопки используется закономерность равноускоренного движения. В начале скорость объекта изменяется на значение ускорения. Затем объект перемещается на расстояние, равное его скорости. Схема алгоритма приведена на рисунке 2.4.

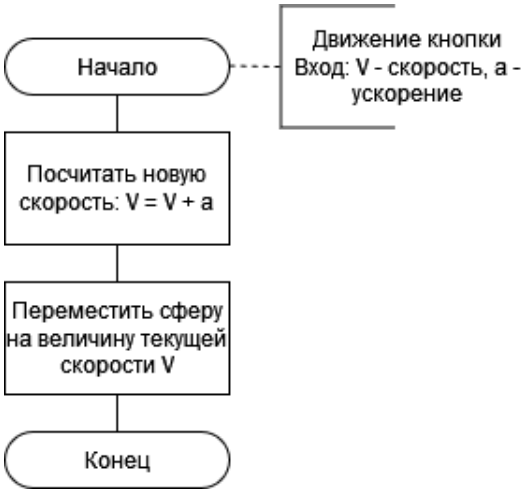



Рисунок 2.4 – Схема алгоритма движения кнопки

### 2.3.4. Алгоритм движения кости домино

При начале движения маятника известна переданная ему кинетическая энергия, поэтому все последующие скорости кости находятся исходя из закона сохранения механической энергии. В начале находится угловая скорость объекта и осуществляется его поворот. Затем определяется новая скорость объекта. При этом выполняется проверка, не меньше ли новый угол, чем минимально допустимый (угол, при котором происходит столкновение со столом или следующей костью). Схема алгоритма приведена на рисунке 2.5.

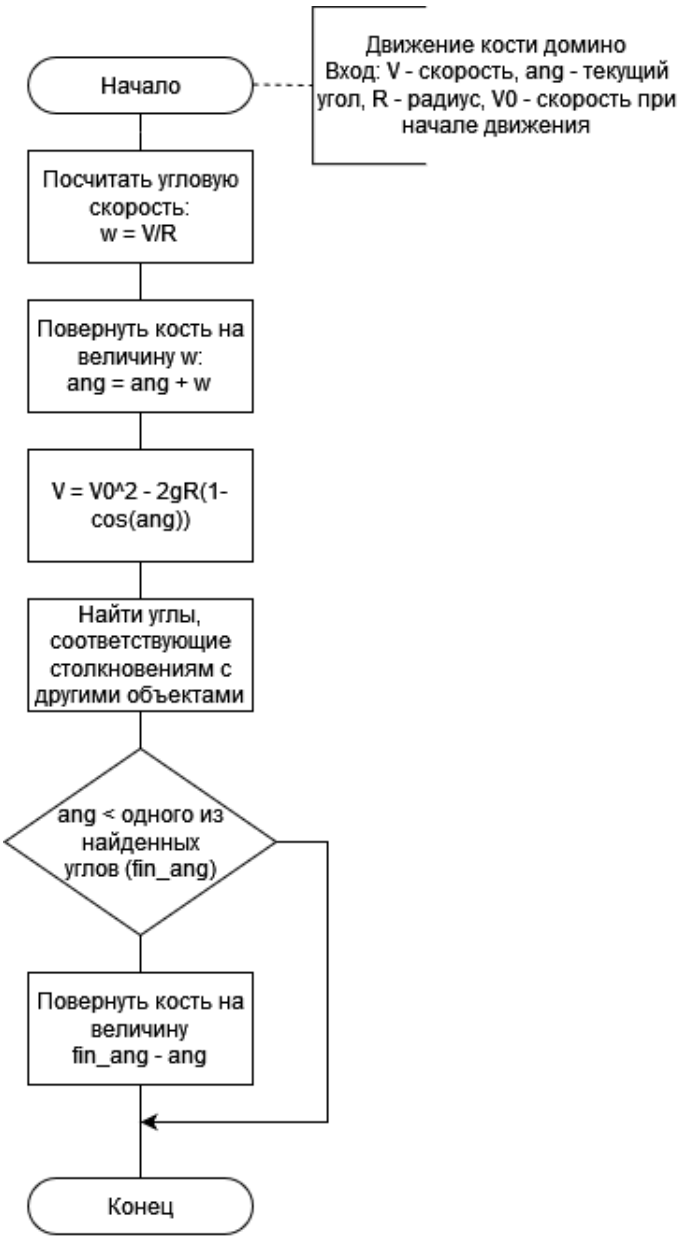



Рисунок 2.5 – Схема алгоритма движения кости домино

### 2.3.5. Алгоритм столкновения сферы и маятника

Факт столкновения объектов определяется из уловия, что расстояние между центрами сферы и сферы, закреплённой на маятнике, меньше либо равно сумме радиусов этих сфер. Находятся новые скорости объектов исходя из закона сохранения механической энергии и закона сохранения импульса. Так как маятник в момент столкновения неподвижен, то его скорость при подсчёте можно не учитывать. Схема алгоритма приведена на рисунке 2.6.

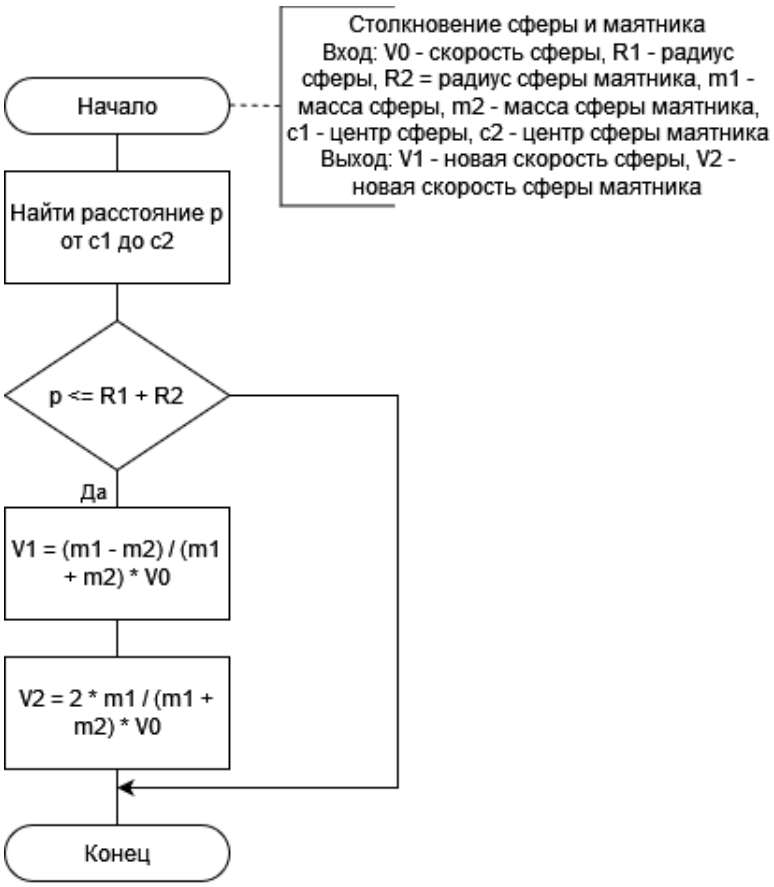



Рисунок 2.6 – Схема алгоритма столкновения сферы и маятника

### 2.3.6. Алгоритм столкновения маятника и первой кости домино

Факт столкновения объектов определяется из уловия, что расстояние между цетром сферы, закреплённой на маятнике, и левой гранью кости меньше либо равно радиусу этой сферы. При этом координата z центра сфера не должна быть больше координаты z верхней грани кости домино. Находятся новые скорости объектов исходя из закона сохранения механической энергии и закона сохранения импульса. Так как кость в момент столкновения неподвижна, то её скорость при подсчёте можно не учитывать. Схема алгоритма приведена на рисунке 2.7.

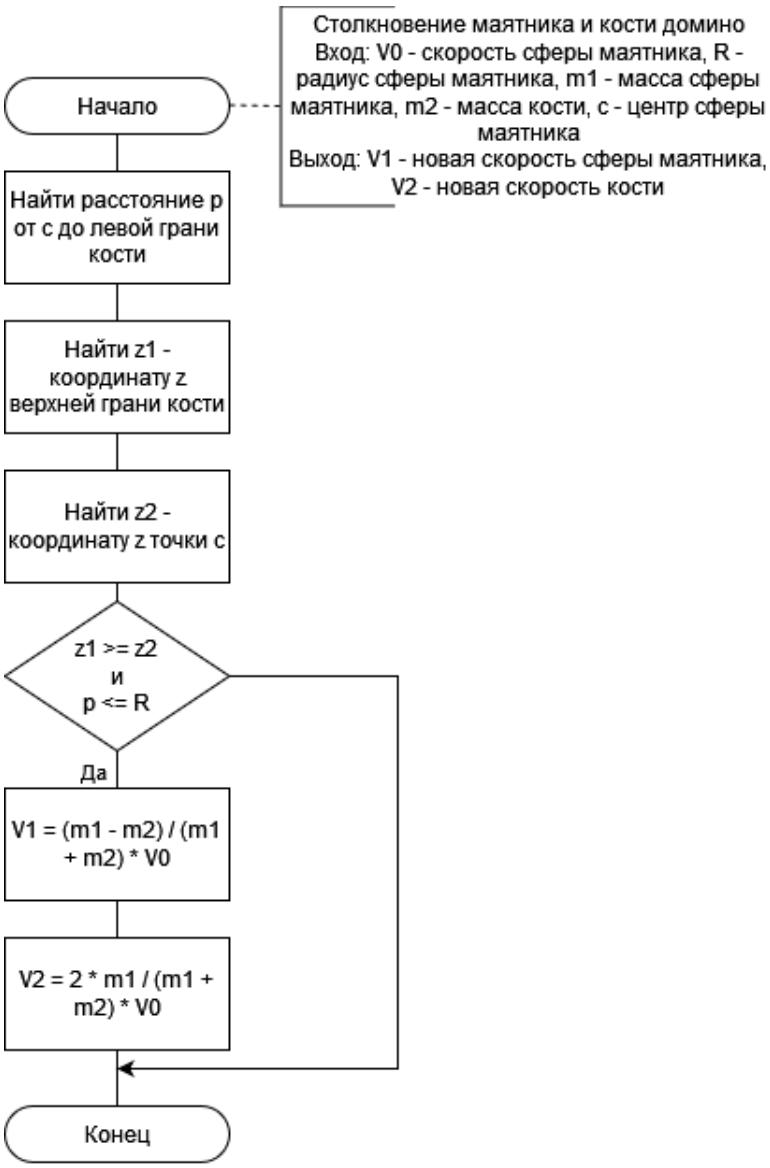



Рисунок 2.7 – Схема алгоритма столкновения маятника и первой кости домино

### 2.3.7. Алгоритм столкновения костей домино

Факт столкновения объектов определяется из уловия, что расстояние между самым правым ребром первой кости и левой гранью правой кости меньше либо равно 0 или погрешности измерения. Находятся новые скорости объектов исходя из закона сохранения механической энергии и закона сохранения импульса. Так как вторая кость в момент столкновения неподвижна, то её скорость при подсчёте можно не учитывать. Схема алгоритма приведена на рисунке 2.8.

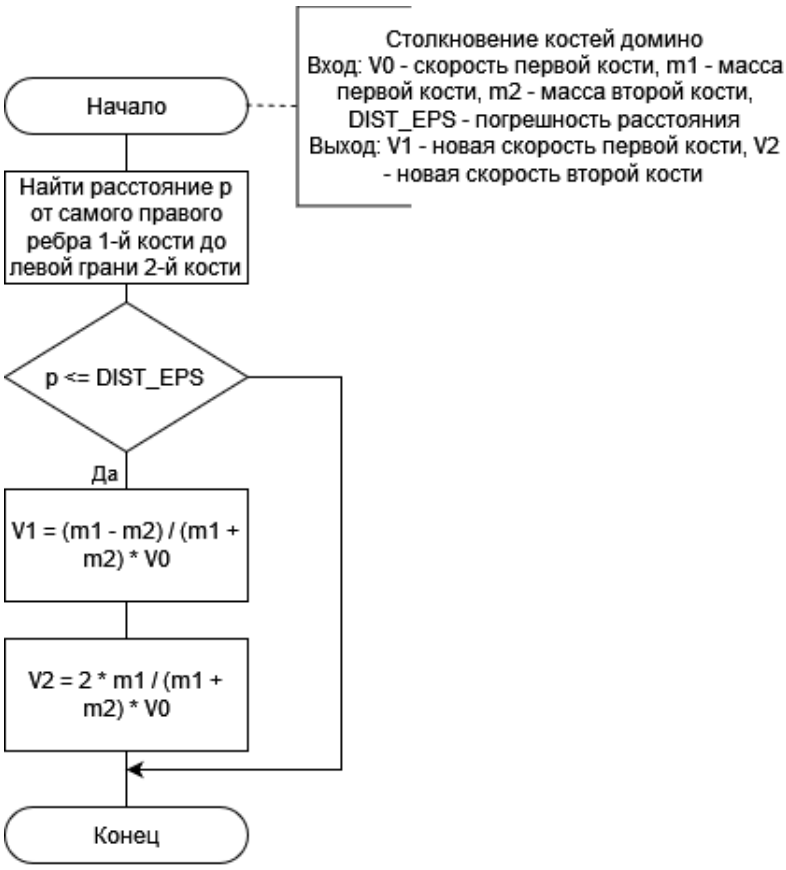



Рисунок 2.8 – Схема алгоритма столкновения костей домино

### 2.3.8. Алгоритм столкновения кости домино и кнопки

Факт столкновения объектов определяется из уловия, что расстояние между самым правым ребром кости и левой гранью правой кости меньше либо равно 0 или погрешности измерения. При этом координата z правого ребра кости не должна быть меньше координаты z самой нижней точки кнопки. Находятся новые скорости объектов исходя из закона сохранения механической энергии и закона сохранения импульса. Так как кнопка в момент столкновения неподвижна, то её скорость при подсчёте можно не учитывать. Схема алгоритма приведена на рисунке 2.9.

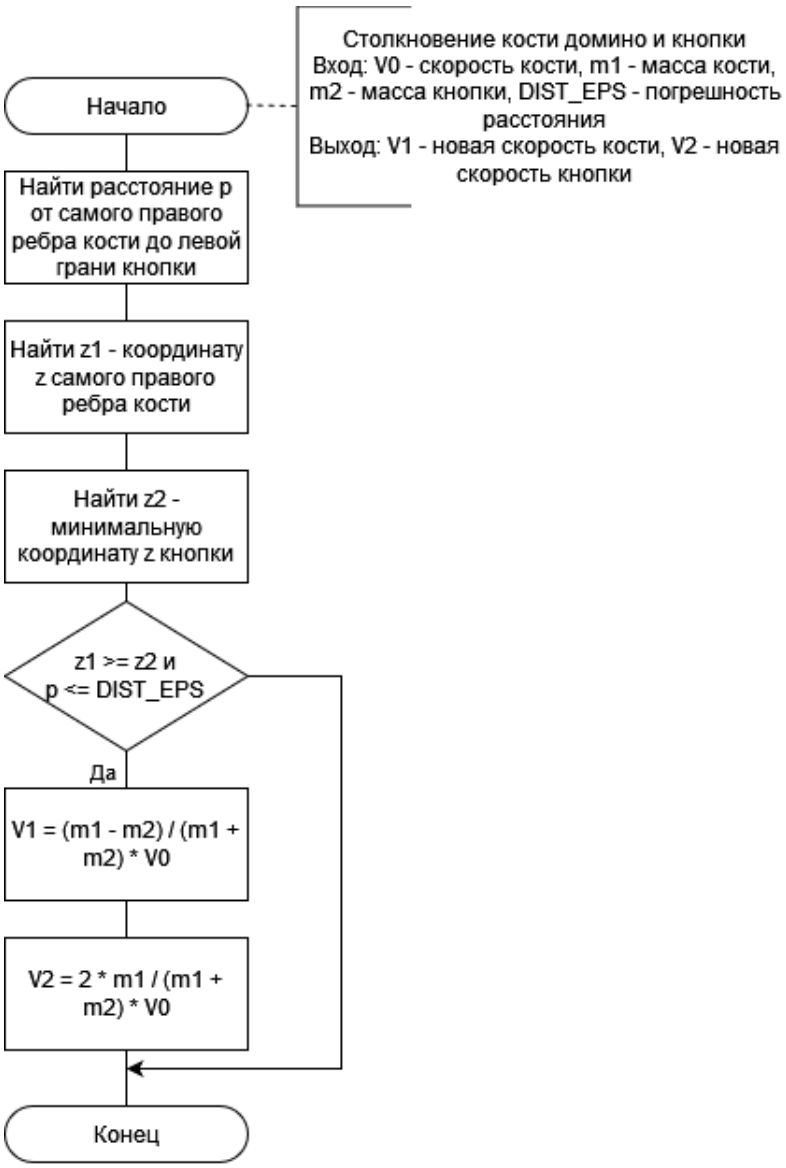



Рисунок 2.9 – Схема алгоритма столкновения кости домино и кнопки

## 2.4. Вывод

В данном разделе были выбраны структуры данных для решения задачи, в также разработаны алгоритмы для движения и взаимодействия объектов.

# 3. Технологический раздел

В этом разделе рассмотрены средства разработки программного продукта и детали его реализации.

## 3.1. Средства реализации

Для реализации программного продукта был выбран язык программирования C++ [3]. Этот язык является компилируемым, что дает преимущество в скорости в сравнении с интерпретируемыми языками программирования. C++ поддерживает различные механизмы объектно-ориентированного программирования. Эта парадигма сочетается с задачей отображения объектов реального мира, так как каждый объект можно задать с помощью отдельного класса с соответствующим набором полей и методов [4].

Для реализации интерфейса приложения был выбран фреймворк Qt [5]. Данный выбор обусловлен тем, что он содержит все базовые классы, которые требуются для интерфейса приложения.

## 3.2. Формат файла описателя модели

Пример файла описателя модели представлен на листинге 3.1

Листинг 3.1 – Файла описателя модели на примере параллелепипеда

```
0 0 0
8
0 -0.76 0
0 -0.76 2
0 0.74 0
0 0.74 2
1.5 -0.76 0
1.5 -0.76 2
1.5 0.74 0
1.5 0.74 2
12
0 1 2
1 2 3
2 3 6
3 6 7
4 6 7
4 5 7
0 4 5
0 1 5
1 3 7
```

```
1 5 7
0 2 6
0 4 6
```

Первые три значения в файле - координаты центра объекта, относительно которого заданы все остальные точки. После идет число точек, из которых состоит объект и их координаты. Дальше указано число поверхностей и их описания в виде трёх чисел - индексов точек, из которых поверхность состоит.

## 3.3. Реализация основных модулей системы

В приложении А на листингах А.1 и А.2 представлены алгоритмы движения и столкновения оъектов соответственно.

## 3.4. Интерфейс программного обеспечения

Интерфейс главного окна приложения, изображенного на рисунке 3.1, включает в себя:

- группу работы с источником света. Позволяет добавлять и удалять источник света;
- группу работы со сценой. Позволяет создавать новую сцену, а также запускать процесс моделирования.

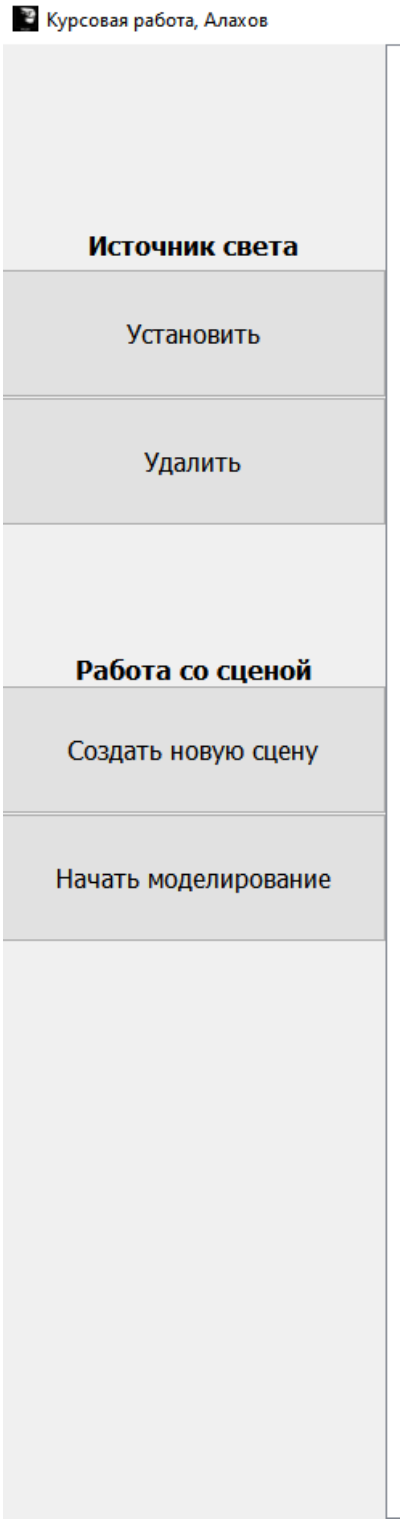



Рисунок 3.1 – Интерфейс главного окна приложения

Первым действием пользователя должно быть создание новой сцены нажатием соответствующей кнопки. Затем пользователь может добавить новый источник света (интерфейс создания источника света будет описан далее), или начать моделирование. Во время моделирования пользователю

будут недоступны все кнопки приложения. После окончания моделирования пользователь может создать новую сцену или поменять источник света в текущей.

После создания сцены пользователю становится доступно перемещение камеры. Вращение осуществляется с помощью клавиш q, w, e, a, s, d, перемещения с помощью стрелок, а масштабирование с помощью z и x.

Интерфейс окна выбора параметров добавляемого источника света, изображенного на рисунке 3.2, включает в себя поля ввода углов поворота по осям X и Y относительно точки наблюдения.

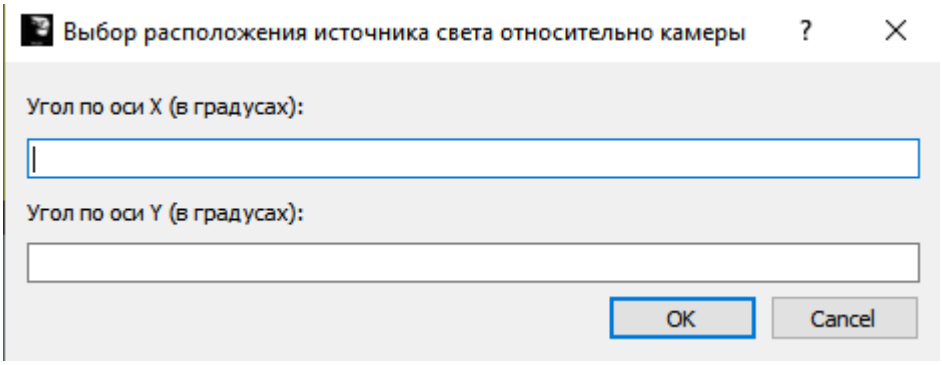



Рисунок 3.2 – Интерфейс окна выбора параметров добавляемого источника света

## 3.5. Тестирование системы

В рамках тестирования программы были выделены следующие случаи:

- Все объекты находятся на одной линии и перекрывают друг друга;
- Один объект находится в тени другого;
- Повороты камеры;
- Изменение масштаба;
- Перемещение камеры.

Результаты тестов приведены на рисунках 3.3 - 3.7

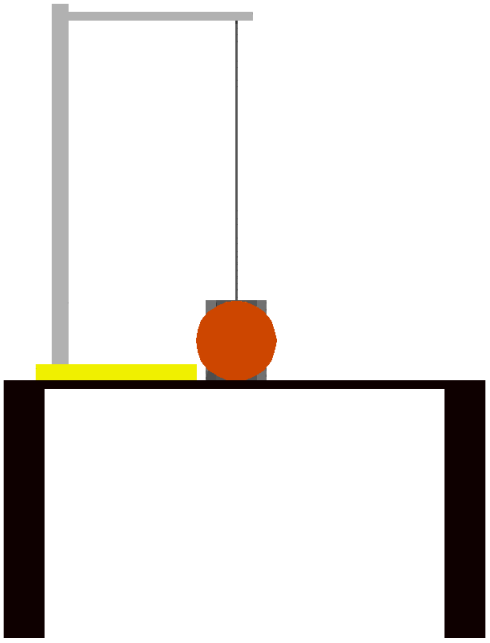


Рисунок 3.3 – Объекты находятся на одной линии

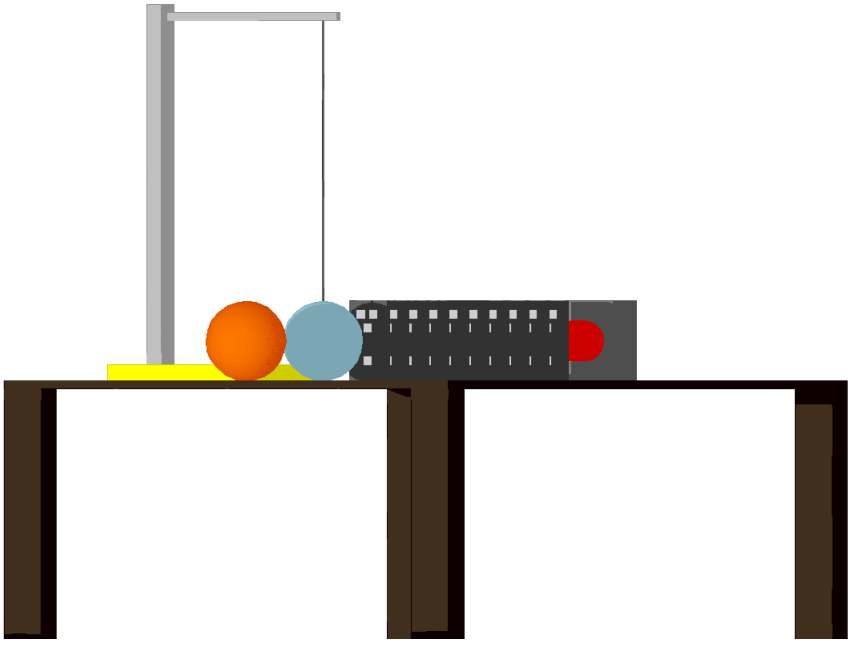


Рисунок 3.4 – Один объект находится в тени другого

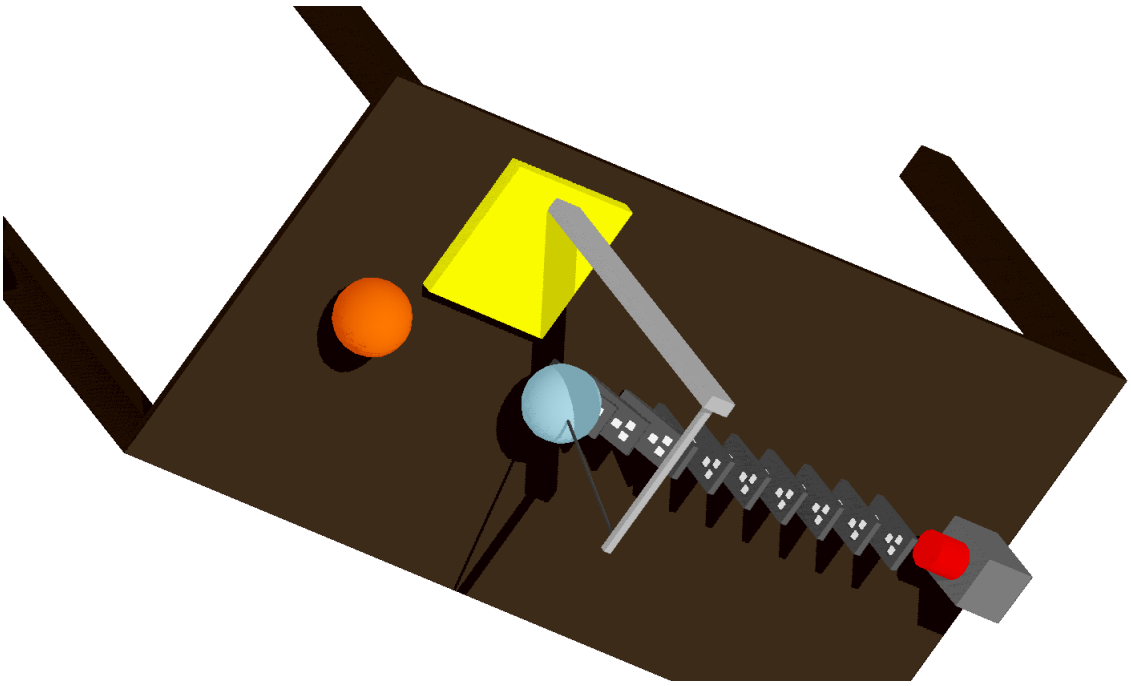


Рисунок 3.5 – Поворот камеры по всем осям

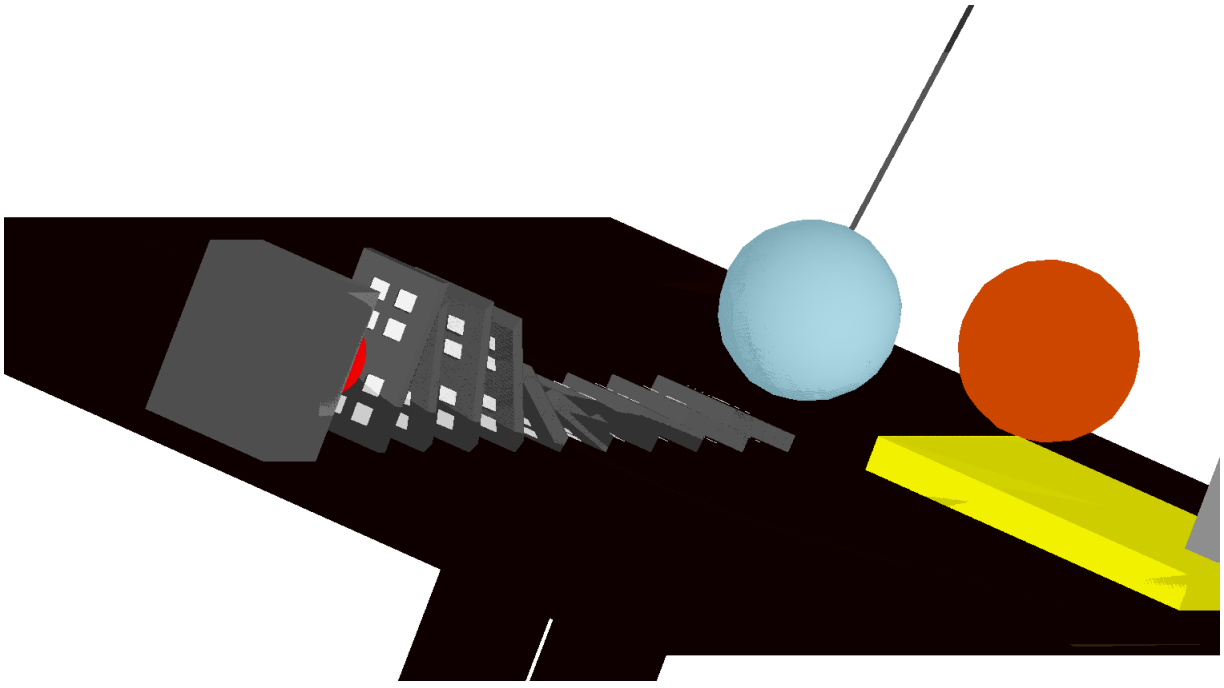


Рисунок 3.6 – Изменение масштаба

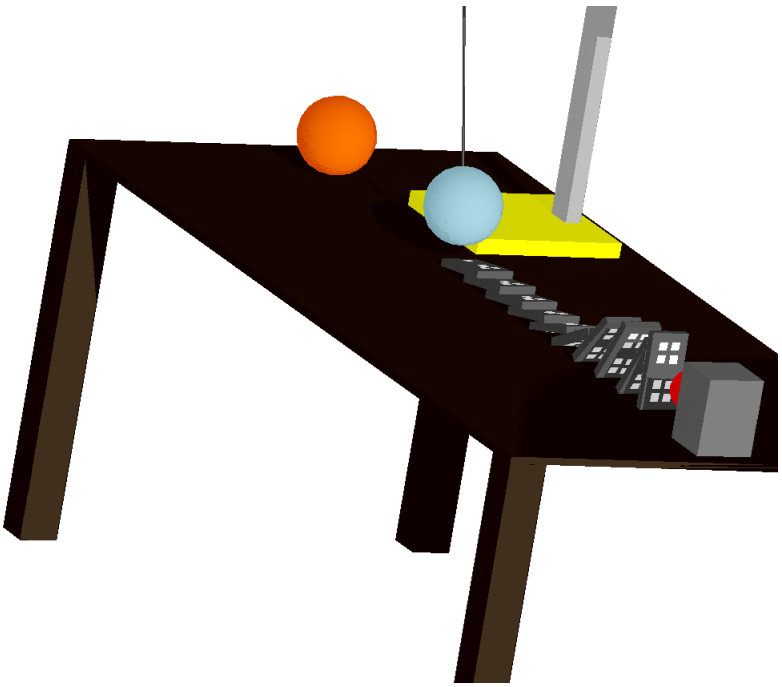


Рисунок 3.7 – Перемещение камеры

Все тесты успешно пройдены.

## 3.6. Вывод

В данном разделе были выбраны средства реализации программного продукта, определен формат описателей моделей, были реализованы и протестированы модифицированный алгоритм z-буфера, алгоритмы закраски и поиска теней, а также алгоритмы, реализующие физические закономерности.

# 4. Исследовательский раздел

В данном разделе проведено сравнение временных характеристик обычного алгоритма z-буфера и его распаралелленной версии.

## 4.1. Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование и исследование:

- операционная система – Windows 10 Pro [6] 64-bit;
- память: 8 GB;
- процессор: Intel(R) Core(TM) i5-4460 [**Intel**] CPU @ 3.20GHz;
- количество логических процессоров: 4.

Эксперименты проводились на компьютере, включенном в сеть электропитания. Во время тестирования компьютер был нагружен только встроенными приложениями окружения рабочего стола, окружением рабочего стола, а также непосредственно системой тестирования. Во время тестирования оптимизации компилятора были отключены.

## 4.2. Замеры времени

В связи с работой алгоритма в нескольких потоках, измерялось реальное время работы, а не процессорное.

Результаты замеров времени (в мс) приведены в таблице 4.1. На рисунке 4.1 приведены зависимости времени работы алгоритмов от количества полигонов всех моделей.

Таблица 4.1 – Замер времени работы алгоритмов для разного числа полигонов

| **Число полигонов** | **Время в мс для алгоритма** | | | | | |
|---|---|---|---|---|---|---|
| | **Стандартного** | **Распараллеленного на n потоков** | | | | |
| | | **1** | **2** | **4** | **8** | **16** |
| 660 | 547.18 | 548.87 | 429.00 | 259.27 | 257.82 | 256.73 |
| 1160 | 550.18 | 551.67 | 432.55 | 263.43 | 260.88 | 259.35 |
| 1660 | 553.20 | 554.02 | 436.81 | 264.08 | 263.34 | 262.51 |
| 2160 | 557.18 | 557.58 | 439.37 | 269.09 | 266.99 | 265.09 |
| 2660 | 560.18 | 560.36 | 443.24 | 275.68 | 274.86 | 271.91 |

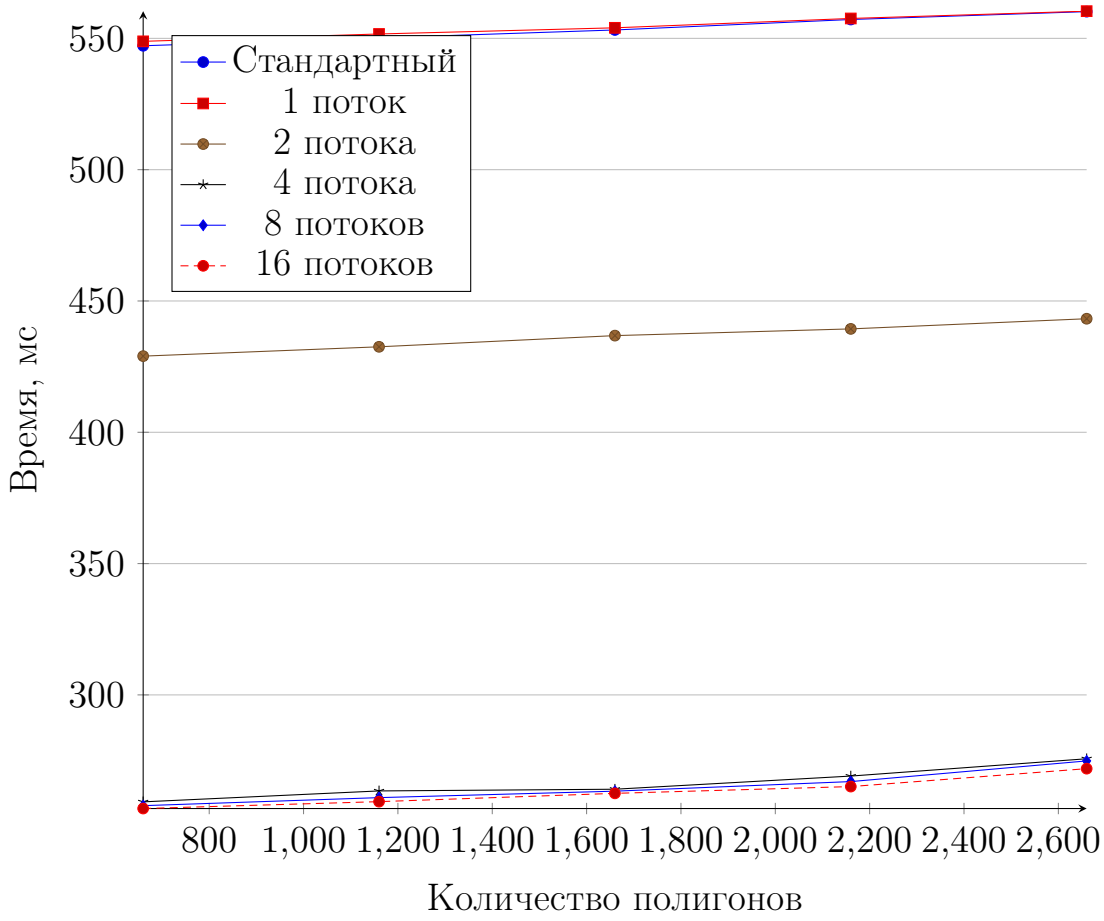



Рисунок 4.1 – Зависимость времени работы алгоритма от количества полигонов.

## Вывод

Из графиков отношения количества полигонов ко времени работы видно, что алгоритм на одном потоке работает примерно столько же времени, как и стандартный алгоритм (разница не более 0,3%). При использовании двух и более потоков, время работы алгоритма уменьшается в разы. В случае, когда количество потоков равно числу логических ядер процессора, алгоритм работает в 2 раза быстрее. При использовании количества потоков, большего, чем число логических ядер процессора, время работы изменяется в пределах 2%, относительно предыдущего рассмотренного случая.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта были получены знания в области компьютерной графики, закреплены навыки проектирования программного обеспечения. Были проанализированы различные подходы к построению реалистичных сцен и выбран наиболее подходящий под условие задачи подход. Были спроектированы схемы выбранных алгоритмов. Выбранные алгоритмы были реализованы в соответствии с выбранными схемами. Разработанное ПО было протестировано. Было проведено исследование времени работы обычной и распараллеленной версий алгоритма z-буфера.

В результате была создана программа, способная моделировать движение и взаимодействие системы объектов с заданным набором законов для перемещения между кадрами.

В результате исследования было выяснено, что версия алгоритма с использованием параллельных вычислений может дать выигрыш во времени в 2 раза.

В качестве дальнейшего развития ПО может быть предложено добавление новых объектов в цепочку взаимодействия, или добавление возможности задавать начальное положение существующих объектов.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ


1. *Роджерс Д.* Алгоритмические основы машинной графики //. — Москва 'Мир', 1989.
2. *Канторович С. С.* ОБЩАЯ ФИЗИКА Механика //. — Издательство Уральского университета, 2012.
3. Документация по Microsoft C++, C и ассемблеру [Электронный ресурс]. — Режим доступа: `https://docs.microsoft.com/ru-ru/cpp/` (дата обращения: 11.09.2021).
4. *Гради Буч Роберт А. Максимчук М. У. Э.* Объектно-ориентированный анализ и проектирование с примерами приложений //. — Издательский дом Вильямс, 2008.
5. Документация по фреймворку qt [Электронный ресурс]. — Режим доступа: `https://doc.qt.io` (дата обращения: 16.10.2021).
6. Клиентские ресурсы и документация Windows для ИТ-специалистов [Электронный ресурс]. — Режим доступа: `https://docs.microsoft.com/ru-ru/windows/resources/` (дата обращения: 14.09.2021).

# ПРИЛОЖЕНИЕ А
## Листинги кода

Листинг А.1 – Движения

```
// Sphere
if (model1->getVelocity()) {
    v = model1->getVelocity();
    model1->move(v, 0, 0);
    model1->rotateZX(-v / model1->getRadius());
    new_v = v + model1->getAcceleration();
    model1->setVelocity(new_v * v < 0 ? 0 : new_v);
}

// Mayatnik
v = model2->getVelocity();
if (v)
    model2->rotateZX(v / model2->getRadius());
if (m2_v0) {
    new_v = pow(m2_v0, 2) - 2 * G * model2->getRadius() * (1 - cos(model2->getAngle()));
    if (new_v < 0)
        m2_speed_vect *= -1;
    new_v = sqrt(abs(new_v));
    model2->setVelocity(new_v * m2_speed_vect);
}

// Button
if (model3->getVelocity()) {
    v = model3->getVelocity();
    new_v = v + model3->getAcceleration();
    model3->setVelocity(new_v * v < 0 ? 0 : new_v, MU_BUTTON);
    v = model3->getVelocity();
    model3->move(v, 0, 0);
}

// Domino
v = dominos[9]->getVelocity();
if (v)
    dominos[9]->rotateZX(-v / dominos[9]->getRadius());
if (dominoV0[9]) {
    new_v = sqrt(abs(pow(dominoV0[9], 2) + 2 * G * dominos[9]->getRadius() * (1 -
        sin(dominos[9]->getAngle()))));
    if (dominos[9]->getAngle() - new_v / dominos[9]->getRadius() < 0) {
        dominoV0[9] = 0;
        dominoMoving[9] = 0;
        dominoEnded[9] = 1;
        dominos[9]->rotateZX(-dominos[9]->getAngle());
        new_v = 0;
```

```
    }
    dominos[9]->setVelocity(new_v);
}
for (int i = 8; i > -1; i--) {
    v = dominos[i]->getVelocity();
    if (v > 0)
        dominos[i]->rotateZX(-v / dominos[i]->getRadius());
    if (dominoMoving[i]) {
        new_v = sqrt(abs(pow(dominoV0[i], 2) + 2 * G * dominos[i]->getRadius() * (1 -
            sin(dominos[i]->getAngle()))));
        fin_angle = atan(dominos[i + 1]->getWidth() / SPACE_BETWEEN_DOMINO);
        collision_angle = acos((SPACE_BETWEEN_DOMINO - dominos[i + 1]->getWidth()) /
            dominos[i]->getRadius());
        new_angle = dominos[i]->getAngle() - new_v / dominos[i]->getRadius();
        collAngle = dominos[i]->findCollisionAngle(dominos[i + 1]);

        Dot rotatedRightUp = dominos[i]->getRightUp();
        rotatedRightUp.rotateY(new_angle, dominos[i]->getCenter());

        if (!dominoMoving[i + 1] && !dominoEnded[i + 1] && new_angle < collision_angle) {
            dominos[i]->rotateZX(-dominos[i]->getAngle() + collision_angle);
        }
        else if (!dominoEnded[i + 1] && new_angle < collAngle) {
            dominos[i]->rotateZX(-dominos[i]->getAngle() + collAngle);
        }
        else if (dominoEnded[i + 1] && new_angle < fin_angle) {
            dominoV0[i] = 0;
            dominoMoving[i] = 0;
            dominoEnded[i] = 1;
            dominos[i]->rotateZX(-dominos[i]->getAngle() + fin_angle);
            new_v = 0;
        }
        dominos[i]->setVelocity(new_v);
    }
}
```

Листинг А.2 – Столкновения

```
// Collision sphere and mayatnik
if (model2->getCollisionCenter().findDistance(model1->getCollisionCenter()) <=
    model2->getCollisionRadius() + model1->getCollisionRadius()) {
    double v0 = model1->getVelocity();
    double m1 = model1->getMass();
    double m2 = model2->getMass();
    m2_v0 = 2 * m1 / (m1 + m2) * v0;
    model1->setVelocity((m1 - m2) / (m1 + m2) * v0);
    model2->setVelocity(m2_v0);
}

```

```
// Collision mayatnik and domino
if (dominos.front()->getLeftUp().getZ() >= model2->getCollisionCenter().getZ() &&
    model2->getCollisionCenter().findDistanceToLine(dominos[0]->getLeftUp(),
    dominos[0]->getLeftDown()) <= model2->getCollisionRadius()) {
    double v0 = model2->getVelocity();
    double m1 = model2->getMass();
    double m2 = dominos[0]->getMass();
    dominoMoving[0] = 1;
    dominoV0[0] = 2 * m1 / (m1 + m2) * v0;
    model2->setVelocity((m1 - m2) / (m1 + m2) * v0);
    dominos[0]->setVelocity(dominoV0[0]);
}

// Collision domino and domino
for (int i = 8; i > -1; i--) {
    if (!dominoEnded[i + 1] && dominos[i]->getRightUp().findDistanceToLine(dominos[i +
        1]->getLeftUp(), dominos[i + 1]->getLeftDown()) <= DIST_EPS) {
        double v0 = dominos[i]->getVelocity();
        double m1 = dominos[i]->getMass();
        double m2 = dominos[i + 1]->getMass();
        dominoV0[i] = (m1 - m2) / (m1 + m2) * v0;
        dominoV0[i + 1] = 2 * m1 / (m1 + m2) * v0;
        dominoMoving[i + 1] = 1;
        dominos[i]->setVelocity(dominoV0[i]);
        dominos[i + 1]->setVelocity(dominoV0[i + 1]);
    }
}

// Collision domino and button
if (dominos.back()->getRightUp().getZ() >= model3->getLeftDown().getZ() &&
    dominos.back()->getRightUp().findDistanceToLine(model3->getLeftUp(),
    model3->getLeftDown()) <= DIST_EPS) {
    double v0 = dominos.back()->getVelocity();
    double m1 = dominos.back()->getMass();
    double m2 = model3->getMass();
    dominoV0.back() = (m1 - m2) / (m1 + m2) * v0;
    dominos.back()->setVelocity(dominoV0.back());
    model3->setVelocity(2 * m1 / (m1 + m2) * v0, MU_BUTTON);
}
```

# ПРИЛОЖЕНИЕ Б
## Презентация

# Моделирование движения и взаимодействия объектов, составляющих машину Голдберга

Студент: А. Г. Алахов ИУ7-52Б
Руководитель: К. А. Кивва

## Постановка задачи

Цель: построение системы трехмерных объектов, образующую «машину Голдберга», состоящей из шара, математического маятника, костей домино и кнопки, с возможностями анимации и изменения положения камеры.
Задачи:

- изучить различные подходы к построению реалистичных сцен;
- изучить модели движения и взаимодействия твёрдых тел;
- выбрать наиболее подходящие под условие задачи алгоритмы;
- определить структуры данных;
- реализовать алгоритмы;
- провести тестирование;
- создать версию алгоритма, которая может выполняться с использованием нескольких потоков одновременно;
- сравнить временные показатели обычной и распараллеленной версий.

2

## Методы решения задачи удаления невидимых линий

Для решения задачи удаления невидимых линий был выбран

алгоритм z-буфера.

Его преимущества:

- Вычислительная трудоемкость алгоритма не более, чем линейна
- Может обрабатывать сцены любой сложности
- Прост в реализации
- Есть возможность использования распараллеливания

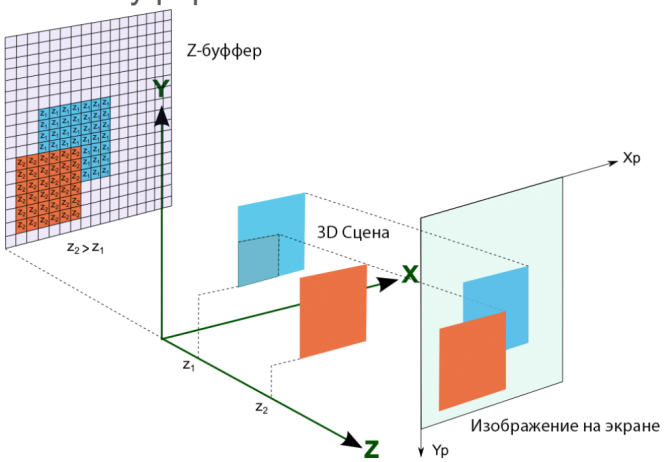
Недостатки:

- Большой объем требуемой оперативной памяти

3

## Методы решения задачи построения теней

Для проверки нахождения объекты в тени используется алгоритм, использующий теневой z-буфер.




Суть данного алгоритма заключается в добавлении вычисления теневого z-буфера из точки наблюдения, совпадающей с источником света.

4

## Методы решения задачи получения сглаженного изображения

В качестве закраски был выбран метод Гуро.

Его преимущества:

- Меньшее количество вычислений по сравнению с закраской Фонга
- Лучшая работа с диффузными отражениями

Недостатки:

- Плохая работа с зеркальными отражениями

5

## Расчетные соотношения для вычисления интенсивности

- Поиск нормали к поверхности

$$n = [a,b] = (a \times b) = \begin{vmatrix} i & j & k \\ a_x & a_y & a_z \\ b_x & b_y & b_z \end{vmatrix}$$

- Нахождение интенсивности света в точке поверхности

$$I_{dif} = I_l k_d \cos\theta, 0 \leq \theta \leq \pi/2$$

Обозначения:

Idif - интенсивность отраженного света

Il - интенсивность точечного источника

kd - коэффициент диффузного отражения

θ - угол между направлением света и нормалью к поверхности

6

## Расчетные соотношения для использовавшихся физических законов

- Скорость при равноускоренном движении

$$v_x = v_{0x} + a_x t.$$

- Зависимость угловой скорости вращения тела от его линейной скорости

$$\omega = \frac{v}{R}.$$

- Закон сохранения механической энергии

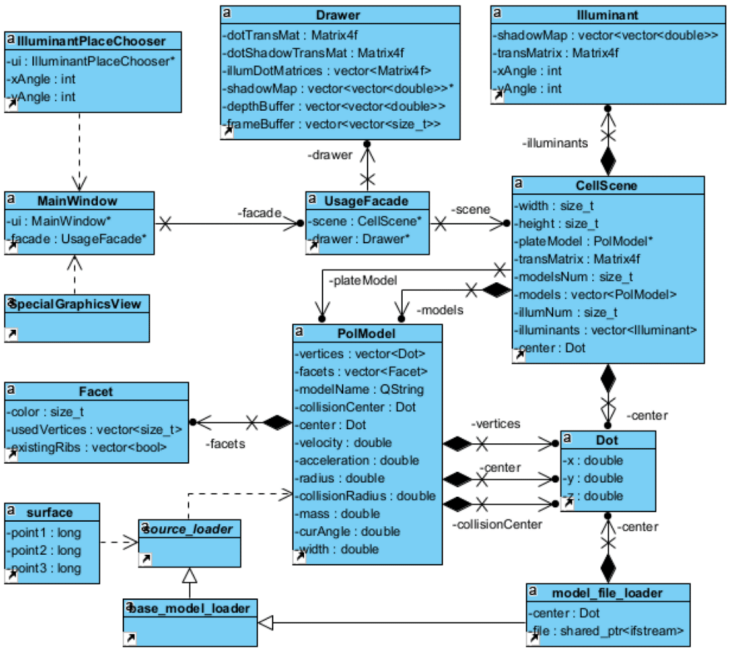
$$\frac{mv^2}{2} + mgh = const$$

- Закон сохранения импульса

$$m_1\vec{v}_1 + m_2\vec{v}_2 = m_1\vec{v}_1' + m_2\vec{v}_2'.$$

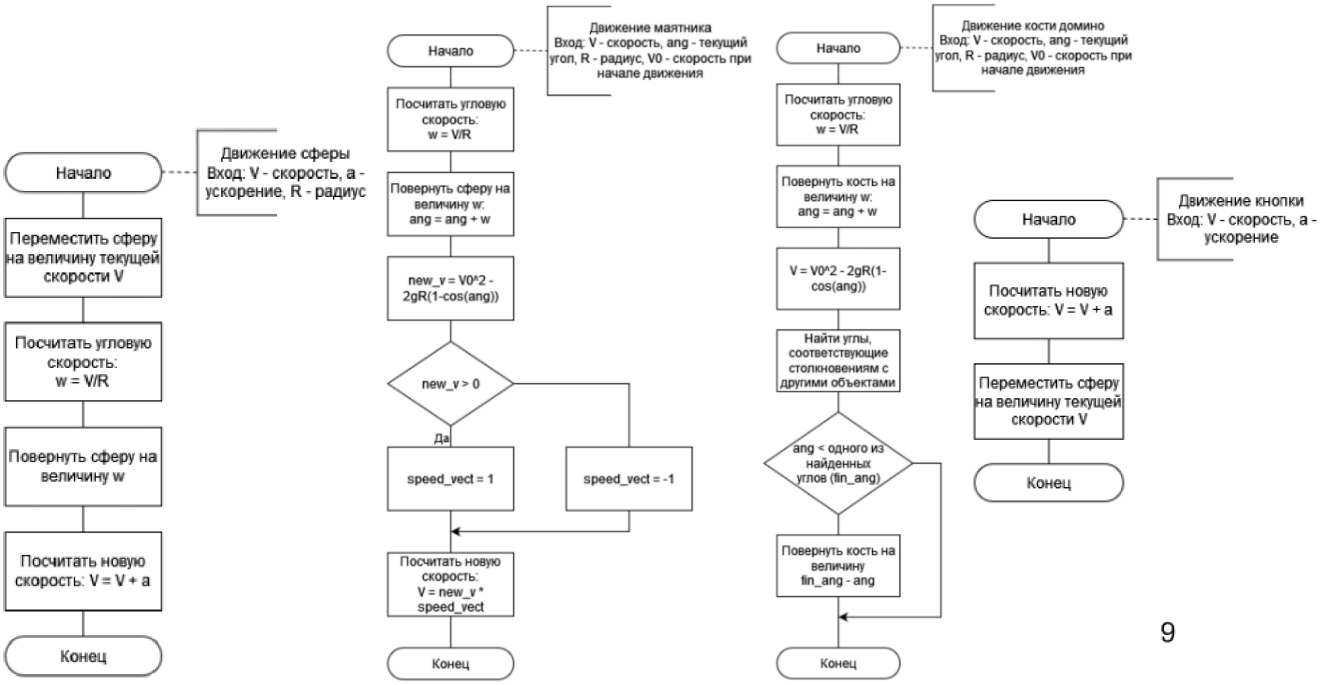
7

## Диаграмма классов



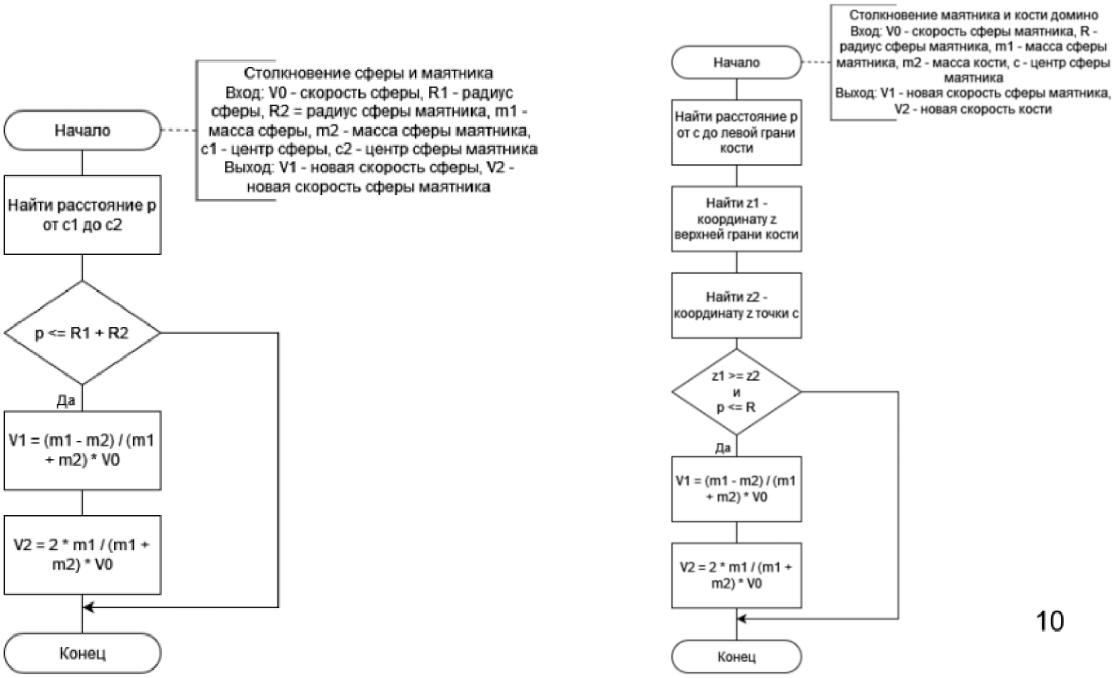

8

## Схемы алгоритмов движения объектов



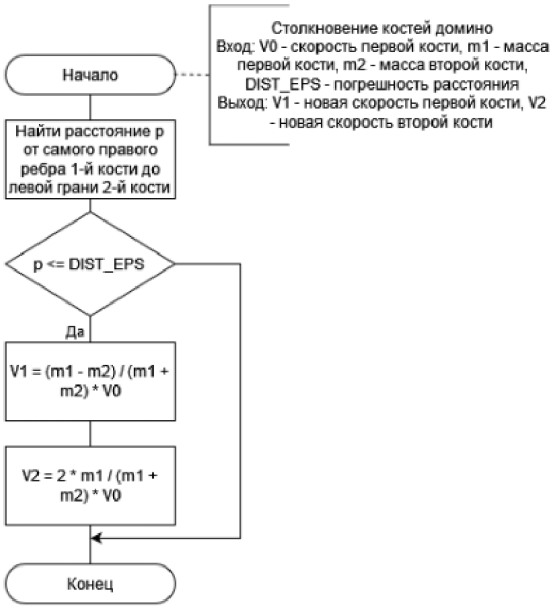

9

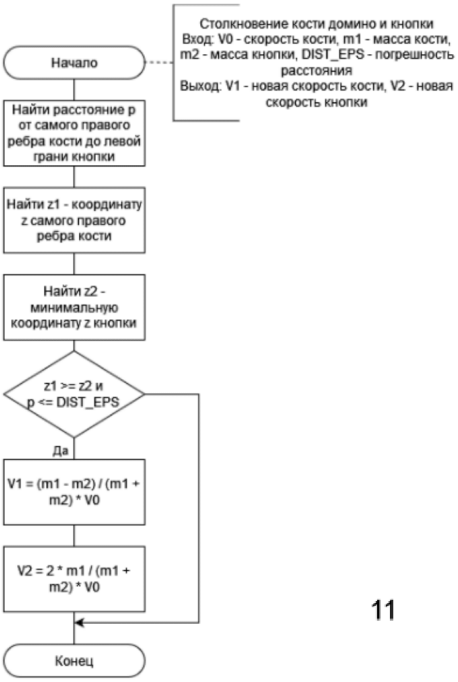
## Схемы алгоритмов столкновения объектов (1)




10

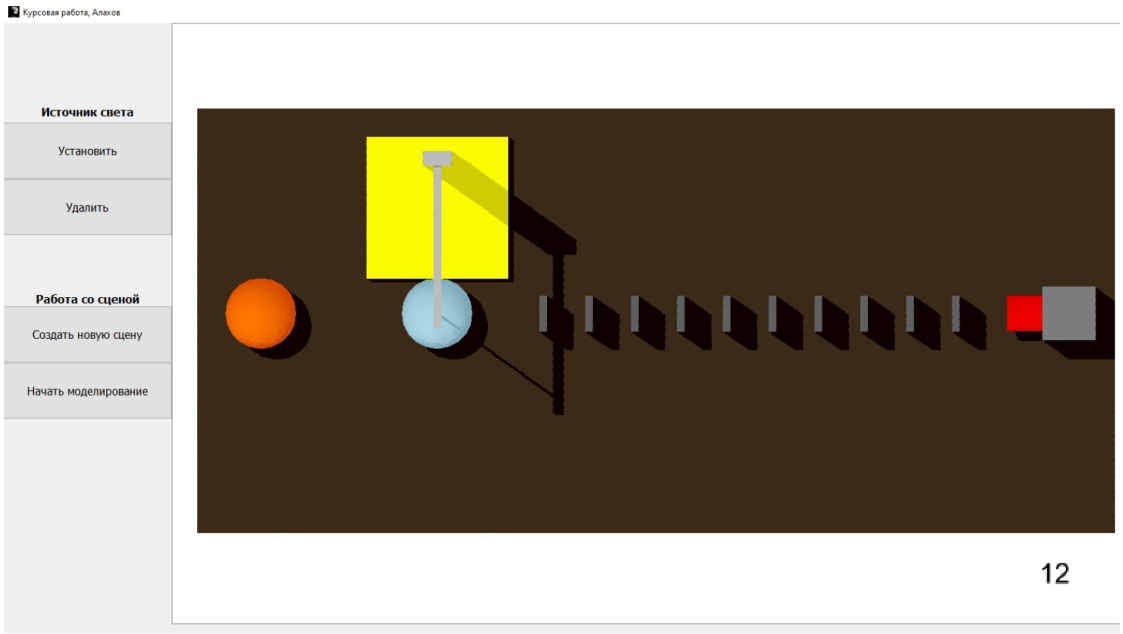
## Схемы алгоритмов столкновения объектов (2)



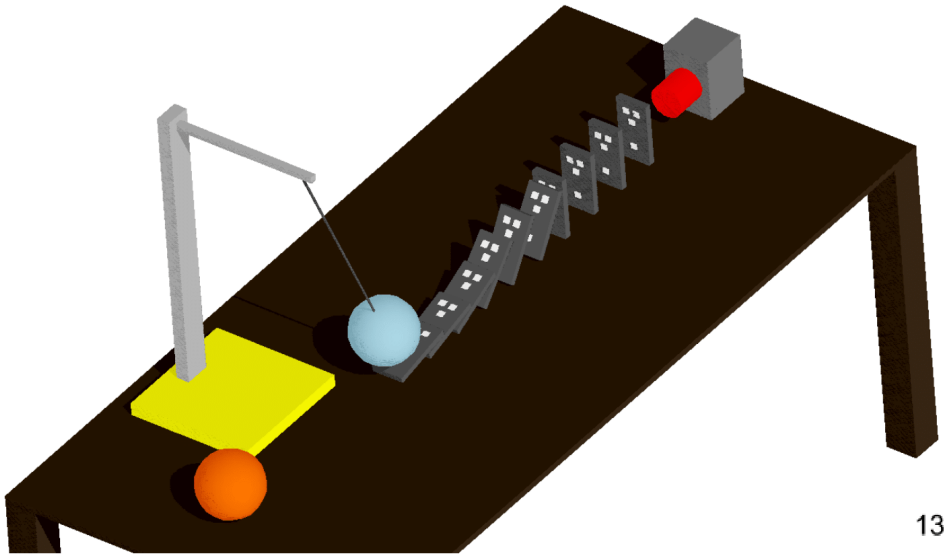

11

## Интерфейс программы




12

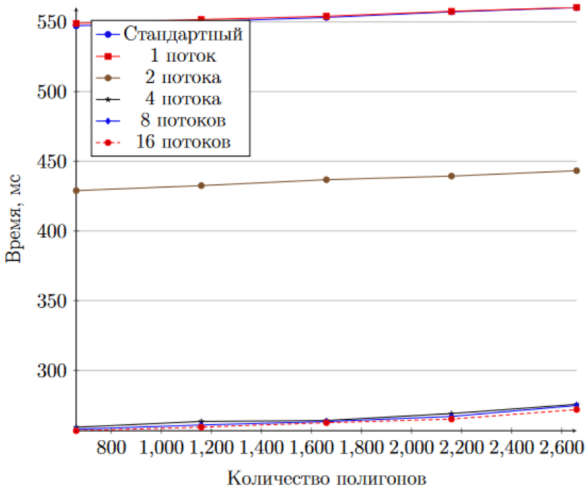
## Примеры кадров в процессе моделирования



13

## Результаты исследований

Наилучшие временные показатели при 4 потоках, что соответствует числу логических ядер.




14

## Заключение

В ходе выполнения курсовой работы были:

- изучены различные подходы к построению реалистичных сцен;
- изучены модели движения и взаимодействия твёрдых тел;
- выбраны наиболее подходящие под условие задачи алгоритмы;
- определены структуры данных;
- реализованы алгоритмы;
- проведено тестирование;
- создана версия алгоритма, которая может выполняться с использованием нескольких потоков одновременно;
- проведено сравнение временных показателей обычной и распараллеленной версий.

15

## Дальнейшее развитие ПО

В качестве дальнейшего развития ПО может быть предложено:

- добавление новых объектов в цепочку взаимодействия;
- добавление возможности задавать начальное положение существующих объектов.

16