



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
КАФЕДРА

ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ
ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

КУРСОВАЯ РАБОТА

НА ТЕМУ:

*Моделирование изображения трёхмерной
сцены с пузырьковыми кластерами*

Студент

ИУ7-55Б

(группа)

(подпись, дата)

А.К. Федченко

(И.О. Фамилия)

Руководитель курсового
проекта

(подпись, дата)

Л.Л. Волкова

(И.О. Фамилия)

Консультант

(подпись, дата)

(И.О. Фамилия)

2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
1 Аналитический раздел	4
1.1 Поверхностное натяжение	4
1.2 Условия образования кластера, отталкивания и слияния	4
1.3 Геометрическое построение сложных пузырей	6
1.4 Модель взаимодействия двух пузырей при соприкосновении . .	7
1.5 Обзор существующих методов визуализации трёхмерной сцены .	7
1.5.1 Алгоритм определения видимых поверхностей путём трас- сировки лучей	7
1.5.2 Алгоритм, использующий Z-буффер	8
1.5.3 Алгоритм плавающего горизонта	9
1.6 Обоснование выбора метода визуализации	10
2 Конструкторский раздел	11
2.1 Математические основы метода математического моделирования	11
2.1.1 Математические модели	11
2.1.2 Основные математические уравнения, использованные для решения задачи	12
2.2 Разработка алгоритма метода моделирования	12
3 Технологический раздел	21
3.1 Выбор и обоснование языка программирования	21
3.2 Хранение и обмен данными в системе	21
3.3 Интерфейс пользователя	23
3.4 Сообщения об ошибке	24
3.5 Функциональное тестирование	24

4 Исследовательский раздел	27
4.1 Исследование характеристик программы	27
ЗАКЛЮЧЕНИЕ	28
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	29
ПРИЛОЖЕНИЕ А	30
ПРИЛОЖЕНИЕ Б	31

ВВЕДЕНИЕ

Данная работа посвящена моделированию сцены, состоящей из нескольких пузырей, динамически образующих кластеры (в кластере не более двух пузырей), соединяющихся воедино и отталкивающихся. Результат взаимодействия соприкоснувшихся пузырьков рассчитывается с учётом угла соприкосновения.

Цель: моделирование изображения трёхмерной сцены с пузырковыми кластерами.

Задачи:

- 1) провести обзор существующих методов визуализации трёхмерной сцены, обосновать выбор метода;
- 2) описать метод взаимодействия пузырей;
- 3) реализовать возможность создания отдельных пузырей;
- 4) реализовать возможность перемещения отдельных пузырей;
- 5) реализовать взаимодействия пузырей при построении трёхмерного изображения сцены в динамике.

1 Аналитический раздел

1.1 Поверхностное натяжение

Пузырь существует потому, что поверхность любой жидкости имеет некоторое поверхностное натяжение, которое делает поведение поверхности похожим на поведение чего-нибудь эластичного.

Поверхностное натяжение [1] — это явление, при котором вещество (прежде всего, жидкость) стремится приобрести форму с минимально возможной площадью поверхности. Это достигается за счет наличия сил поверхностного натяжения.

Сферическая форма пузыря также получается за счёт поверхностного натяжения. Силы натяжения формируют сферу потому, что сфера имеет наименьшую площадь поверхности при данном объёме.

Важно также упомянуть о мемbrane между пузырями. Это пленка, разделяющая два пузыря. При образовании кластера она принимает форму сферической поверхности, так как её кривизна уравновешивает разницу давлений. В техническом задании сказано, принять мембрану за плоскость.

1.2 Условия образования кластера, отталкивания и слияния

Основные силы и условия, влияющие на формирование сложных пузырей:

1) Силы поверхностного натяжения

- Поверхностное натяжение является ключевым фактором, определяющим форму и стабильность пузырей. Оно стремится минимизировать площадь поверхности пузыря, что приводит к образованию сферических форм.
- При взаимодействии нескольких пузырей, силы поверхностного натяжения могут вызывать как слияние, так и отталкивание пузырей.

2) Избыточное давление

- Избыточное давление внутри пузыря определяется его радиусом.

По формуле Лапласа [2]:

$$\Delta P = \frac{2\sigma}{R}$$

— Разница в избыточном давлении между пузырями может приводить к их слиянию или отталкиванию, в зависимости от их размеров и расстояний между ними.

3) Силы взаимодействия между пузырьками

— Силы взаимодействия между пузырями, такие как силы Ван дер Ваальса, могут способствовать образованию кластеров. При близком расположении пузырей, силы притяжения могут преобладать над силами отталкивания, что приводит к слиянию.

4) Вязкость окружающей среды

— Вязкость воздуха (или жидкости, если пузырьки образуются в жидкости) влияет на динамику движения пузырей. Более вязкие среды замедляют движение пузырей и могут способствовать образованию более стабильных структур.

5) Температура

— Температура влияет на свойства жидкости и газов, а также на поверхностное натяжение. При повышении температуры поверхностное натяжение может уменьшаться, что может способствовать образованию пузырей.

6) Концентрация поверхностно-активных веществ (ПАВ)

— ПАВ снижают поверхностное натяжение и могут стабилизировать пузырьки, предотвращая их слияние. Они также могут влиять на размер и форму пузырей.

7) Геометрия и размеры пузырей

— Размеры и форма пузырей влияют на их взаимодействие. Меньшие пузырьки могут сливаться с большими, что приводит к образованию более крупных пузырей.

8) Скорость потока воздуха

— В условиях течения воздуха пузырьки могут быть вытянуты или деформированы, что влияет на их стабильность и возможность слияния. Высокая скорость потока может привести к образованию более мелких пузырей.

1.3 Геометрическое построение сложных пузырей

В книге «Мыльные пузыри» [3] Бойс Ч.В., опираясь на книгу Жозефа Плато «Статика жидкости», описал геометрическое построение, позволяющее точно вычертить два пузыря и разделяющую их перегородку.

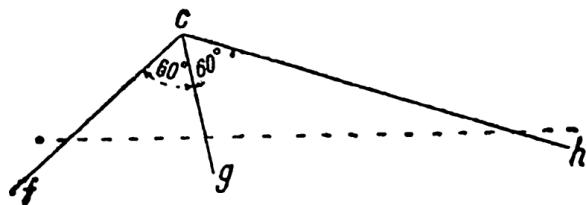


Рисунок 1.1 — Сложные пузыри. Часть 1

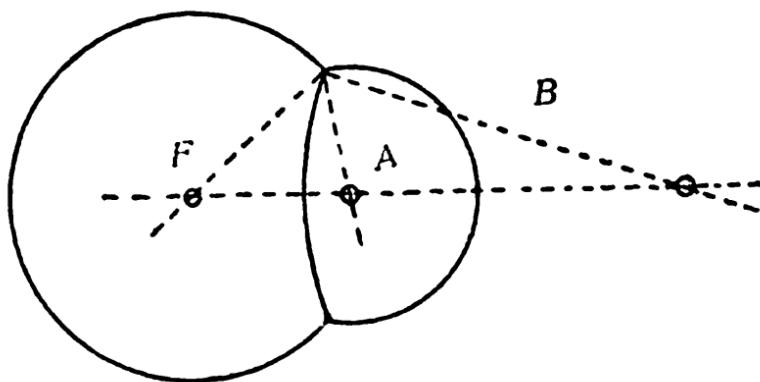


Рисунок 1.2 — Сложные пузыри. Часть 2

Из какой-либо точки С проведены три линии: Cf, Cg, Ch, образующие два угла по 60° , как показано на рис. 1.1. Они пересекаются четвертой прямой линией, проведенной на рисунке пунктиром. Получившиеся три точки пересечения являются центрами трех окружностей, соответствующих трем возможным пузырям. Точка пересечения средней линии является центром окружности малого пузыря, из других же двух точек та, которая ближе к С, представляет собой центр второго пузыря, а та, которая находится дальше от С, — центр перегородки. Теперь, устанавливая одну из ножек циркуля последовательно в каждой из этих точек, проведены отрезки окружностей, проходящих через С, как показано на рис. 1.2, на котором линии рисунка 1.1 воспроизведены пунктиром, дуги же окружностей — сплошными линиями [3].

Такое же рассуждение может быть расширено до трёхмерного построения.

1.4 Модель взаимодействия двух пузырей при соприкосновении

Поскольку на данный процесс влияет такое большое количество разных факторов и нет единой системы уравнений для определения того, что именно произойдёт с двумя соприкоснувшимися пузырями, и техническое задание требует визуализации, а не моделирования, была введена модель для упрощённого определения взаимодействия двух пузырей. Так как в книге «Мыльные пузыри» [3] Бойса Ч.В. пузырьковый кластер строится при угле соприкосновения равном шестидесяти градусам, примем следующую модель взаимодействия:

- образование кластера при $55^\circ \leq \phi \leq 65^\circ$;
- образование одного большого пузыря при $\phi < 55^\circ$;
- отталкивание при $65^\circ < \phi$;

где: ϕ — угол соприкосновения пузырей т.е. наименьший угол между радиусами сфер в любой точке соприкосновения.

1.5 Обзор существующих методов визуализации трёхмерной сцены

Проведён обзор следующих методов визуализации трёхмерных сцен:

- 1) алгоритм определения видимых поверхностей путём трассировки лучей;
- 2) алгоритм, использующий Z-буфер;
- 3) алгоритм плавающего горизонта.

1.5.1 Алгоритм определения видимых поверхностей путём трассировки лучей

Главная идея, лежащая в основе этого метода, заключается в том, что наблюдатель видит любой объект посредством испускаемого неким источником света, который падает на этот объект и затем каким-то путем доходит до наблюдателя. Свет может достичь наблюдателя, отразившись от поверхности, преломившись или пройдя через нее. Если проследить за лучами света, выпущенными источником, то можно убедиться, что весьма немногие из них дойдут до наблюдателя. Следовательно, этот процесс был бы вычислительно неэффек-

тивен. Поэтому было предложено отслеживать (трассировать) лучи в обратном направлении, т. е. от наблюдателя к объекту, как показано на рис. 1.3.

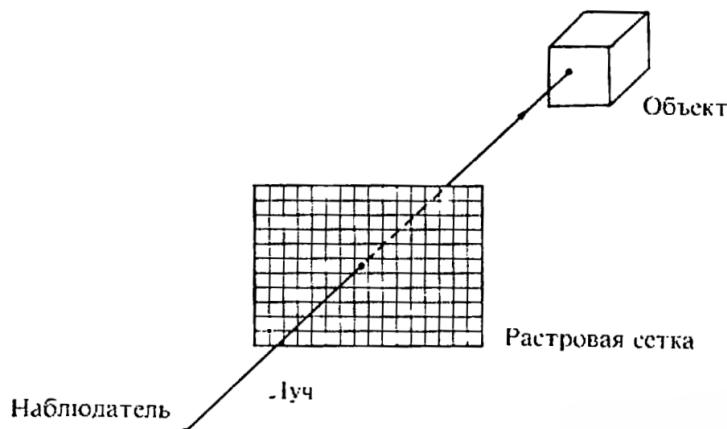


Рисунок 1.3 — Простая трассировка луча

Эти алгоритмы учитывают эффекты отражения одного объекта от поверхности другого, преломления, прозрачности и затенения. Производится также устранение ступенчатости.

Преимущества:

- учитывает эффект отражения одного объекта от поверхности другого;
- учитывает эффект преломления;
- учитывает эффект прозрачности;
- учитывает эффект затенения;
- производит устранение ступенчатости;
- позволяет создавать изображения с высокой степенью реалистичности.

Недостатки:

- требует значительных вычислительных ресурсов;
- реализация может быть сложной для динамических сцен.

1.5.2 Алгоритм, использующий Z-буффер

Этот алгоритм работает в пространстве изображения. Идея z-буфера является обобщением идеи о буфере кадра. Буфер кадра используется для запоминания атрибутов (интенсивности) каждого пикселя в пространстве изображения.

Z-буффер это отдельный буфер глубины, используемый для запоминания координаты z или глубины каждого видимого пикселя в пространстве изображения. В процессе работы глубина или значение z каждого нового пикселя,

который нужно занести в буфер кадра, сравнивается с глубиной того пикселя, который уже занесен в z-буфер. Если это сравнение показывает, что новый пиксель расположен впереди пикселя, находящегося в буфере кадра, то новый пиксель 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. По сути, алгоритм является поиском по x и y наибольшего значения функции z (x, y).

Преимущества:

- простота [4];
- решает задачу об удалении невидимых поверхностей;
- делает тривиальной визуализацию пересечений сложных поверхностей;
- оценка вычислительной трудоемкости алгоритма не более чем линейна и зависит от габаритов пространства изображения.

Недостатки:

- большой объем требуемой памяти.

1.5.3 Алгоритм плавающего горизонта

Алгоритм плавающего горизонта чаще всего используется для удаления невидимых линий трехмерного представления функций, описывающих поверхность в виде:

$$F(x, y, z) = 0$$

Главная идея данного метода заключается в сведении трехмерной задачи к двумерной путем пересечения исходной поверхности последовательностью параллельных секущих плоскостей, имеющих постоянные значения координат x, y или z. Для хранения максимальных и минимальных значений у при каждом значении x используются два массива чисел: массив верхнего горизонта и массив нижнего горизонта.

Преимущества:

- простота;
- метод плавающего горизонта может обрабатывать большие объемы данных в реальном времени.

Недостатки:

- метод плавающего горизонта может не подходить для визуализации данных, которые требуют высокой степени детализации или точности;
- не подходит для реалистичной визуализации с тенями и бликами.

1.6 Обоснование выбора метода визуализации

Для реализации был использован алгоритм определения видимых поверхностей путём трассировки лучей. Это было сделано по следующим причинам:

- из всех проанализированных ранее алгоритмов с помощью данного можно создать наиболее реалистичное изображение;
- можно визуализировать блики;
- можно визуализировать отражение объектов друг в друге.

2 Конструкторский раздел

2.1 Математические основы метода математического моделирования

2.1.1 Математические модели

Математическая модель, выбранная для пузырьков, – это сфера, которая в свою очередь состоит из двух полусфер и круговой поверхности. Данный выбор обусловлен упрощением геометрической формы пузырьков, что позволяет использовать более простые математические уравнения и алгоритмы для описания их поведения и взаимодействия.

Круговая поверхность – это упрощённая до плоскости мембрана между пузырьками в кластере.

Параметрами математической модели пузырька являются:

- координаты центра в трёхмерном пространстве;
- радиус.

Для каждой полусфера составляющей модели, характерны следующие параметры:

- координаты центра в трёхмерном пространстве (такие же как и у общей сферы);
- высота сегмента;
- направление (от центра сферы в две противоположные стороны).

Причиной разделения сферы на две полусферы является первый пункт раздела «Модель взаимодействия двух пузырей при соприкосновении» 1.4. При образовании кластера обе сферы должны быть обрезаны плоскостью соприкосновения их поверхностей, что при модели состоящей из двух полусфер требует лишь уменьшения высоты соприкасающихся полусфер и редактирования направлений.

Математическая модель пузырькового кластера в свою очередь состоит из двух уже описанных выше сфер.

2.1.2 Основные математические уравнения, использованные для решения задачи

Формулы 2.1, 2.2 заслуживают упоминания, хоть и являются достаточно простыми, поскольку лежат в основе любого взаимодействия описанных выше математических моделей.

Формула нахождения расстояния между центрами сфер:

$$d = \sqrt{(X_1 - X_2)^2 + (Y_1 - Y_2)^2 + (Z_1 - Z_2)^2} \quad (2.1)$$

где:

- d – расстояние между центрами сфер;
- (X_1, Y_1, Z_1) – координаты центра первой сферы;
- (X_2, Y_2, Z_2) – координаты центра второй сферы.

Формула нахождения угла соприкосновения:

$$\theta = \arccos \left(\frac{\mathbf{AB} \cdot \mathbf{AC}}{|\mathbf{AB}| \cdot |\mathbf{AC}|} \right) \cdot \frac{180}{\pi} \quad (2.2)$$

где:

- \mathbf{AB} – вектор от центра первой сферы до точки пересечения;
- \mathbf{AC} – вектор от центра второй сферы до точки пересечения;
- P – точка пересечения;
- $|\mathbf{AB}|$ – длина вектора \mathbf{AB} ;
- $|\mathbf{AC}|$ – длина вектора \mathbf{AC} .

Формула нахождения радиуса сфера, чей объём равен сумме объёмов двух других сфер:

$$r_{\text{new}} = \left(\frac{3}{4\pi} \left(\frac{4}{3}\pi r_1^3 + \frac{4}{3}\pi r_2^3 \right) \right)^{\frac{1}{3}} \quad (2.3)$$

где r_1 и r_2 — радиусы первой и второй сферы соответственно.

2.2 Разработка алгоритма метода моделирования

Алгоритм анализа объектов сцены

- 1) Глубина рекурсии равна входному значению функции

- 2) Создать квадратную матрицу взаимодействия объектов на сцене, линейный размер равен количеству объектов
 - 3) Рассчитать значения ячеек:
 - 0 - нет пересечений
 - 1 - одно корректное пересечение
 - 2 - более одного пересечения для кластера (ошибка)
 - 4) При наличии ошибок не финализировать изменение сцены, зафиксировать ошибку, иначе:
 - Для каждой пары объектов, имеющих одно пересечение:
 - В зависимости от сочетания типов объектов определить тип взаимодействия:
 - Объекты отталкиваются
 - Объекты объединяются в кластер *
 - Объекты сливаются
 - * если детектирована попытка создать кластер из 3 пузырей, то досрочно выйти из цикла, не финализировать изменение сцены, зафиксировать ошибку
 - Если ошибок не было и если глубина рекурсии больше нуля, то
 - Вызвать функцию анализа объектов сцены с уменьшенным на единицу значением глубины рекурсии
 - Ошибка была вывести сообщение об ошибке
- 5) Завершить

Схема алгоритма расстановки пузырьков представлена на рисунке 2.1, алгоритмы, который он в себя включает, а именно алгоритм слияния в один большой пузырь, отталкивание пузырьков друг от друга и создания пузырькового кластера, представлены на рисунках 2.2, 2.3, 2.4 соответственно.

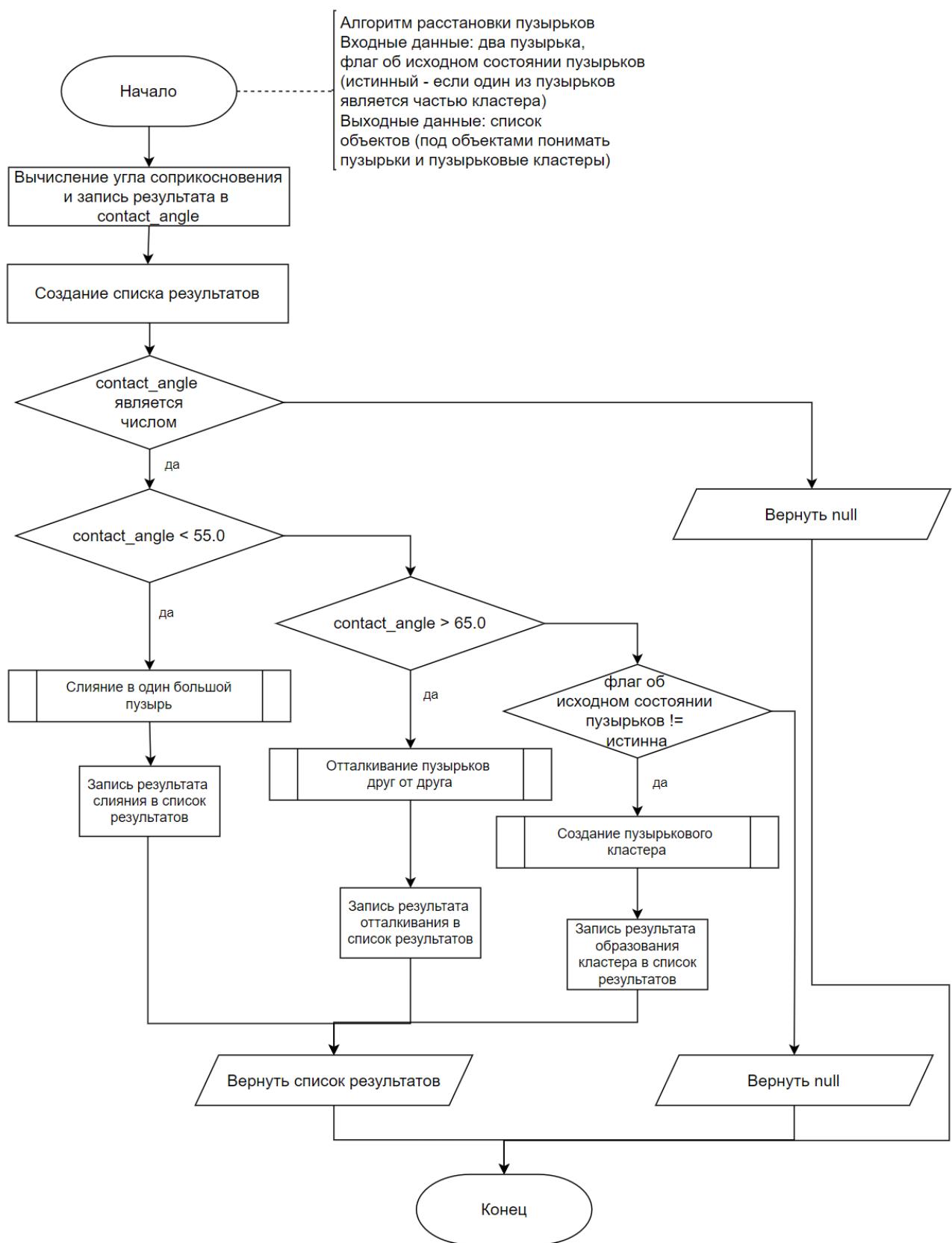


Рисунок 2.1 — Алгоритм расстановки пузырьков

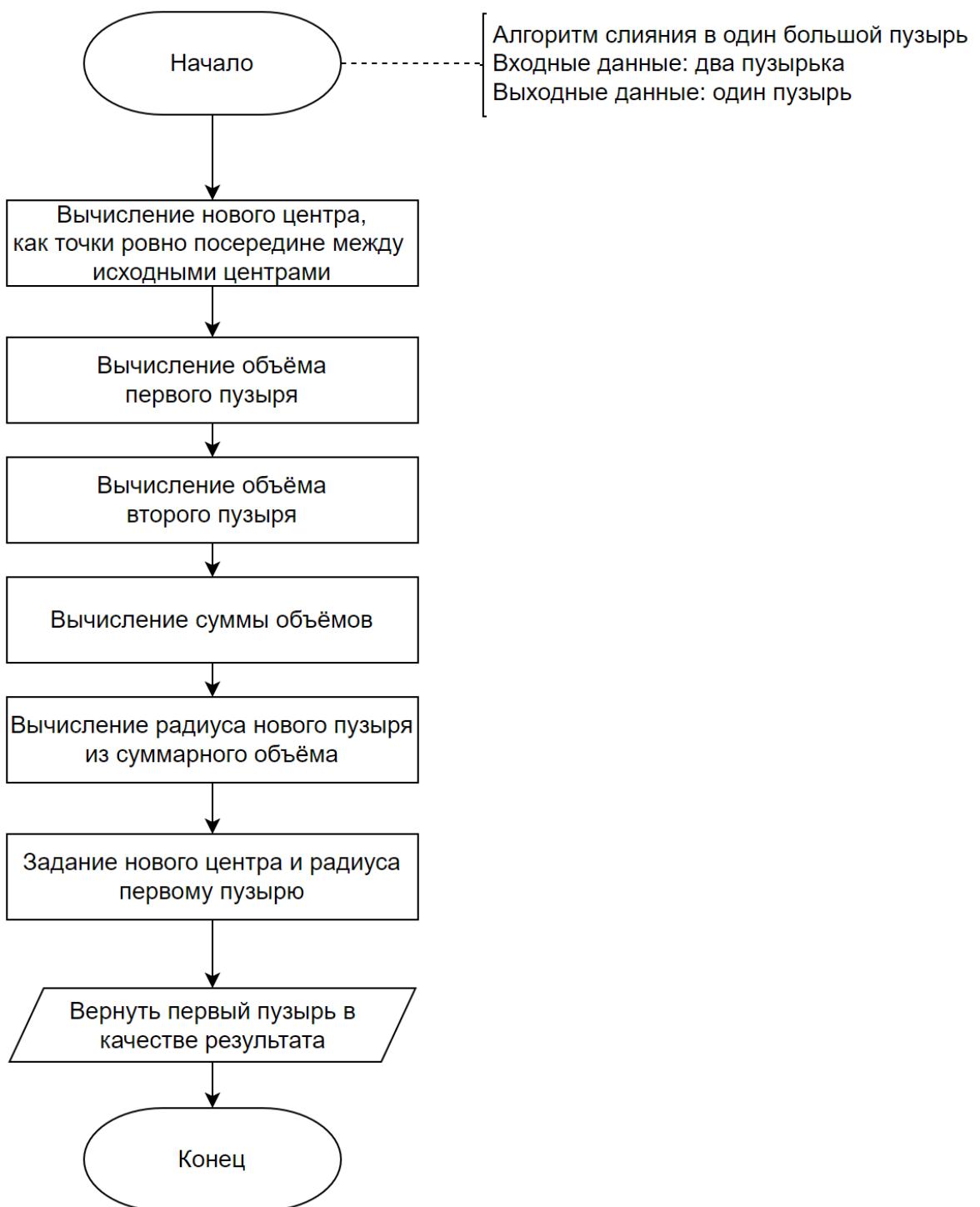


Рисунок 2.2 — Алгоритм расстановки пузырьков, слияние в один большой пузырь

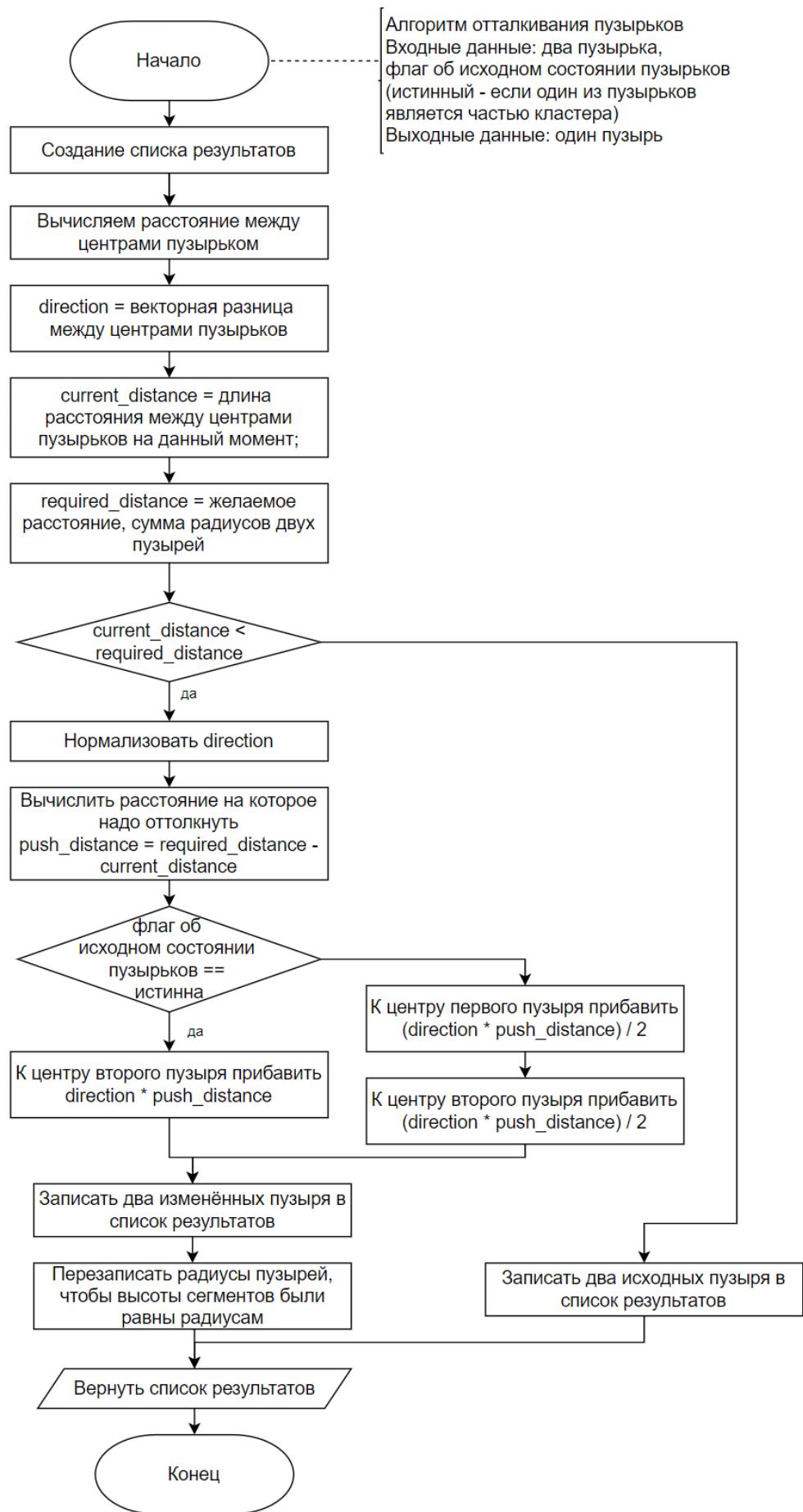


Рисунок 2.3 — Алгоритм расстановки пузырьков, отталкивание пузырьков друг от друга

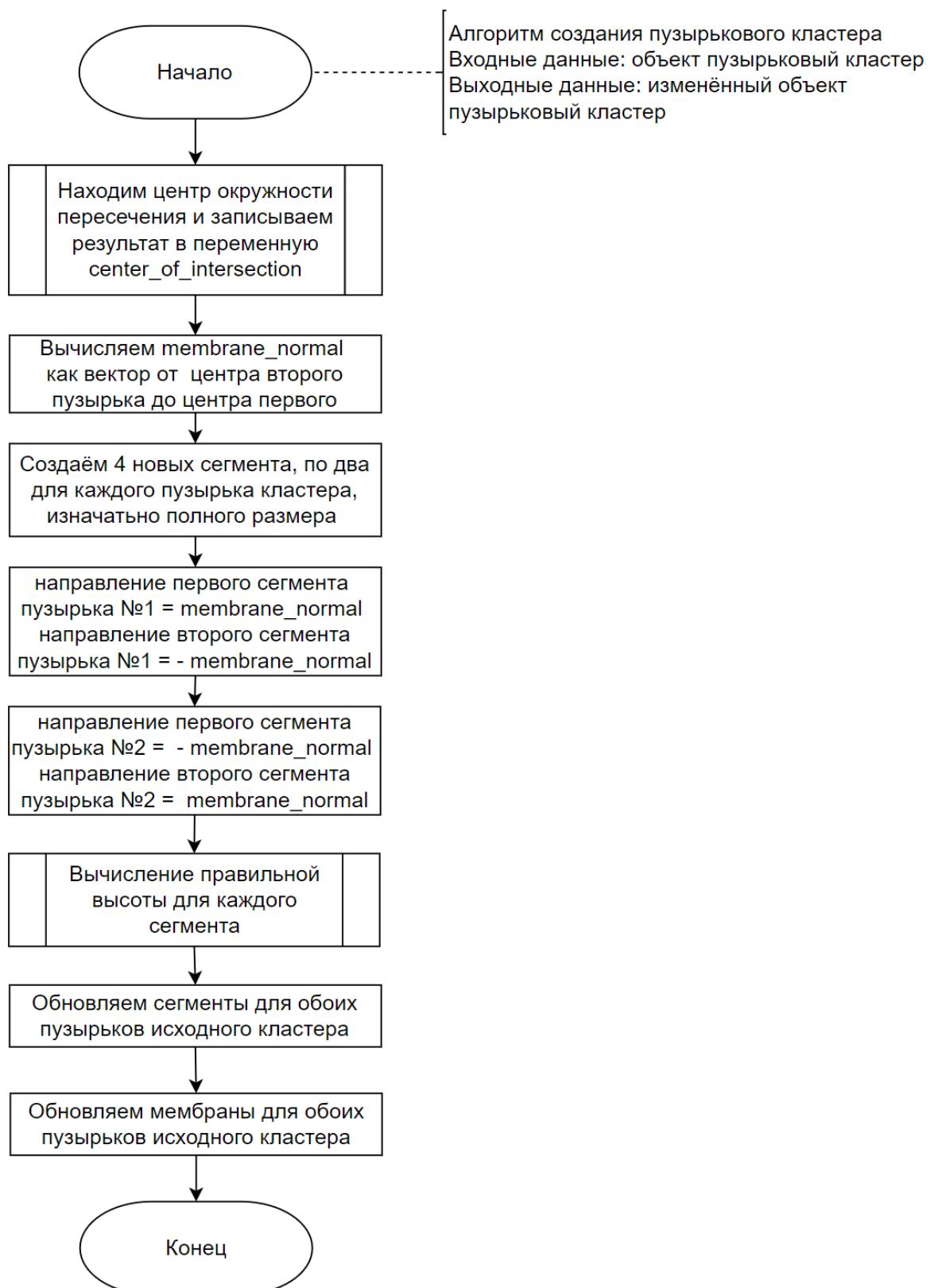


Рисунок 2.4 — Алгоритм расстановки пузырьков, создание пузырькового кластера

Схема алгоритма обратной трассировки лучей представлена на рисунках 2.5 и 2.6. Важно отметить, что алгоритм обратной трассировки лучей для конкретного луча является рекурсивным.

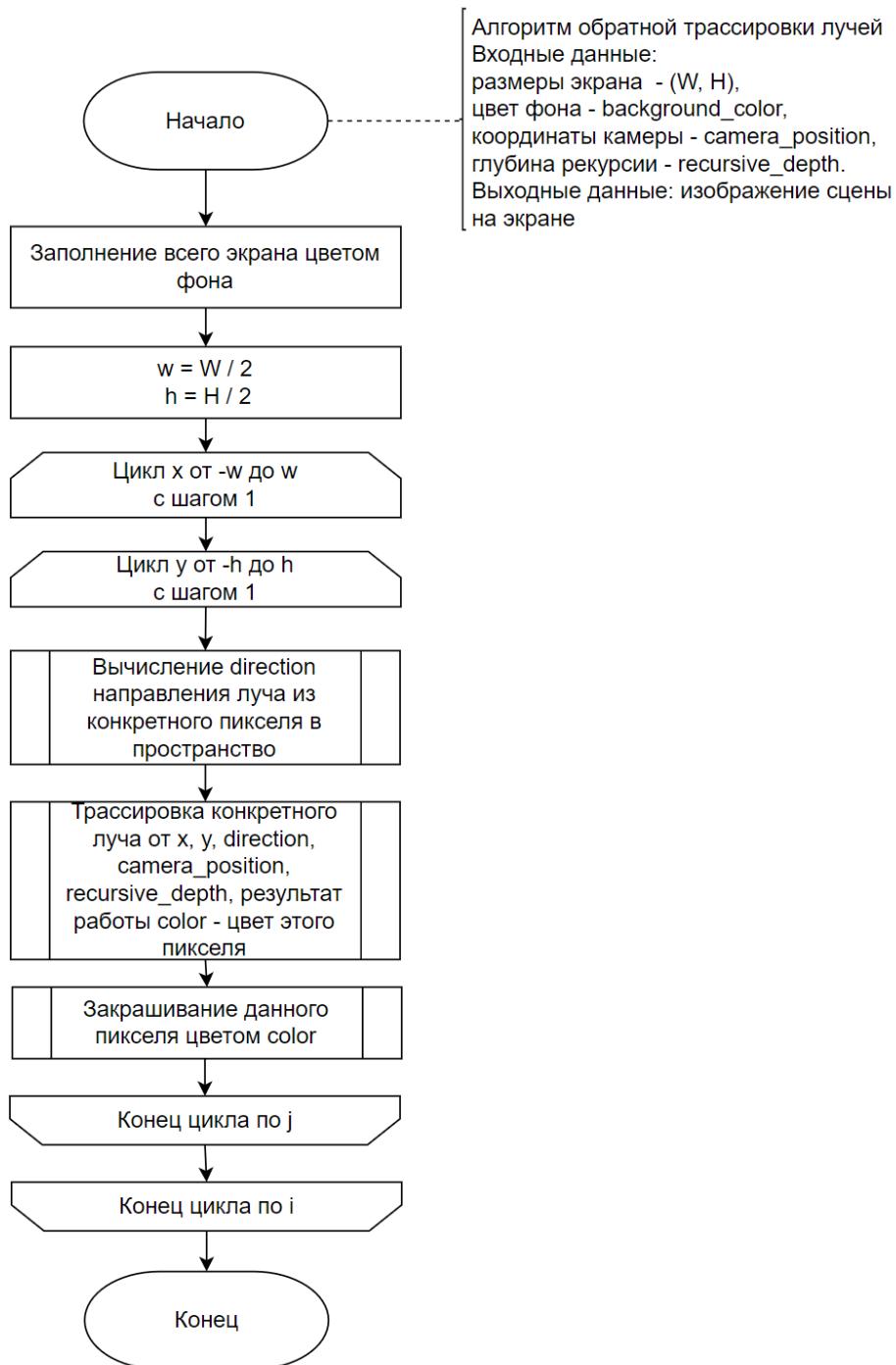


Рисунок 2.5 — Алгоритм обратной трассировки лучей

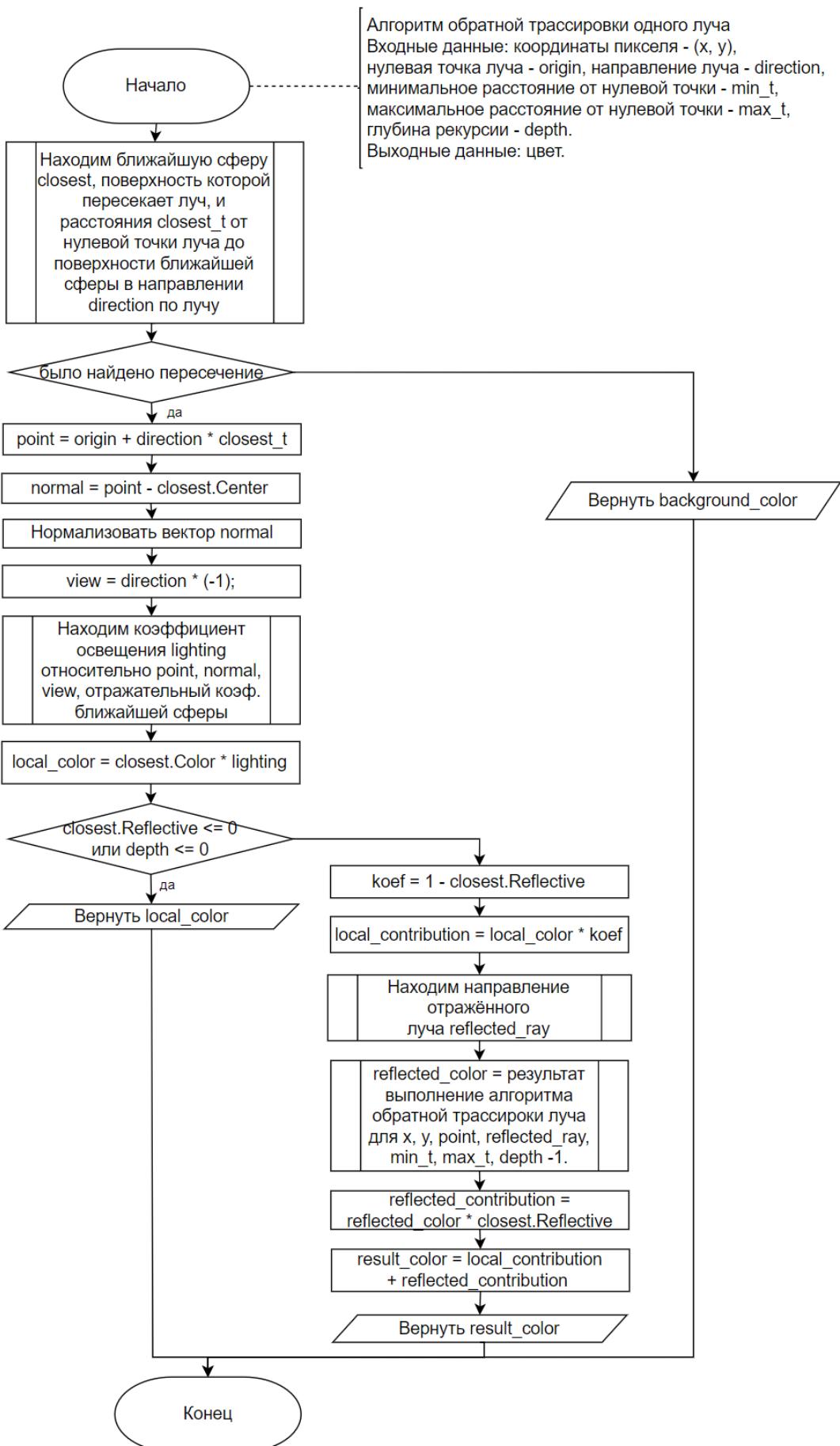


Рисунок 2.6 — Алгоритм обратной трассировки лучей для одного луча

Поскольку в техническом задании есть следующие слова: «...моделированию сцены, состоящей из нескольких пузырей, динамически образующих кластеры...», а алгоритм обратной трассировки лучей, работает слишком долго для динамических преобразований, была введена ещё одна математическая модель – окружность. Этот элемент является двумерным слепком перемещающейся сферы на экране. Преобразование параметров сферы ((X, Y, Z) – координаты центра сферы, R – радиус сферы) в параметры окружности на экране ((x, y) – координаты центра окружности, r – радиус сферы) и обратно, представлено следующими выражениями 2.4 и 2.5.

$$\begin{aligned} scaleFactor &= W \\ r &= R \cdot \left(\frac{1}{d} \right) \\ x &= \left(\frac{W}{2} + X \cdot \left(\frac{1}{d} \right) \cdot scaleFactor \right) \\ y &= \left(\frac{H}{2} - Y \cdot \left(\frac{1}{d} \right) \cdot scaleFactor \right) \end{aligned} \tag{2.4}$$

где:

- W – ширина экрана;
- H – высота экрана, ($W = H$);
- d – расстояние по оси z от позиции камеры до центра сферы.

$$\begin{aligned} scaleFactor &= W/1.0 \\ X &= \frac{(x + shift_x) - \frac{W}{2}}{scaleFactor} \cdot \left(\frac{1.0}{d} \right) \\ Y &= \frac{\frac{H}{2} - (y + shift_y)}{scaleFactor} \cdot \left(\frac{1.0}{d} \right) \end{aligned} \tag{2.5}$$

где:

- W – ширина экрана;
- H – высота экрана, ($W = H$);
- d – расстояние по оси z от позиции камеры до центра сферы;
- $shift_x$ – перемещение относительно оси ox ;
- $shift_y$ – перемещение относительно оси oy .

3 Технологический раздел

3.1 Выбор и обоснование языка программирования

Для реализации программы был выбран язык *C#* по нескольким причинам, среди которых возможность создания настольных приложений с использованием *WindowsForms*, интеграция с *.NET Framework*, что обеспечивает высокую производительность и безопасность приложений, и поддержка объектно-ориентированного программирования.

3.2 Хранение и обмен данными в системе

Диаграмма классов представлена на рисунке 3.1. Математическая модель выбранная для пузырьков представлена в программе классом *Bubble*, являющимся потомком класса *Obj*. Также от класса *Obj* наследуется класс *CombinedBubble*, то есть пузырьковый кластер.

Объекты класса *Contour* используются для отражения перемещения пузырьков и пузырьковых кластеров, без постоянной перерисовки трёхмерной сцены с использованием алгоритма обратной трассировки лучей.

Класс *Form1* представляет собой основное окно пользовательского интерфейса приложения, именно в нём обрабатываются все события связанные с элементами управления.

Важно упомянуть, тип данных, характерный для *C#* и являющийся частью почти всех классов данной программы, – *Vector3D* [5]. Это структура, имеющая три поля, которые могут соответствовать координатам в пространстве, и все основные методы, нужные для работы с координатами или векторами.

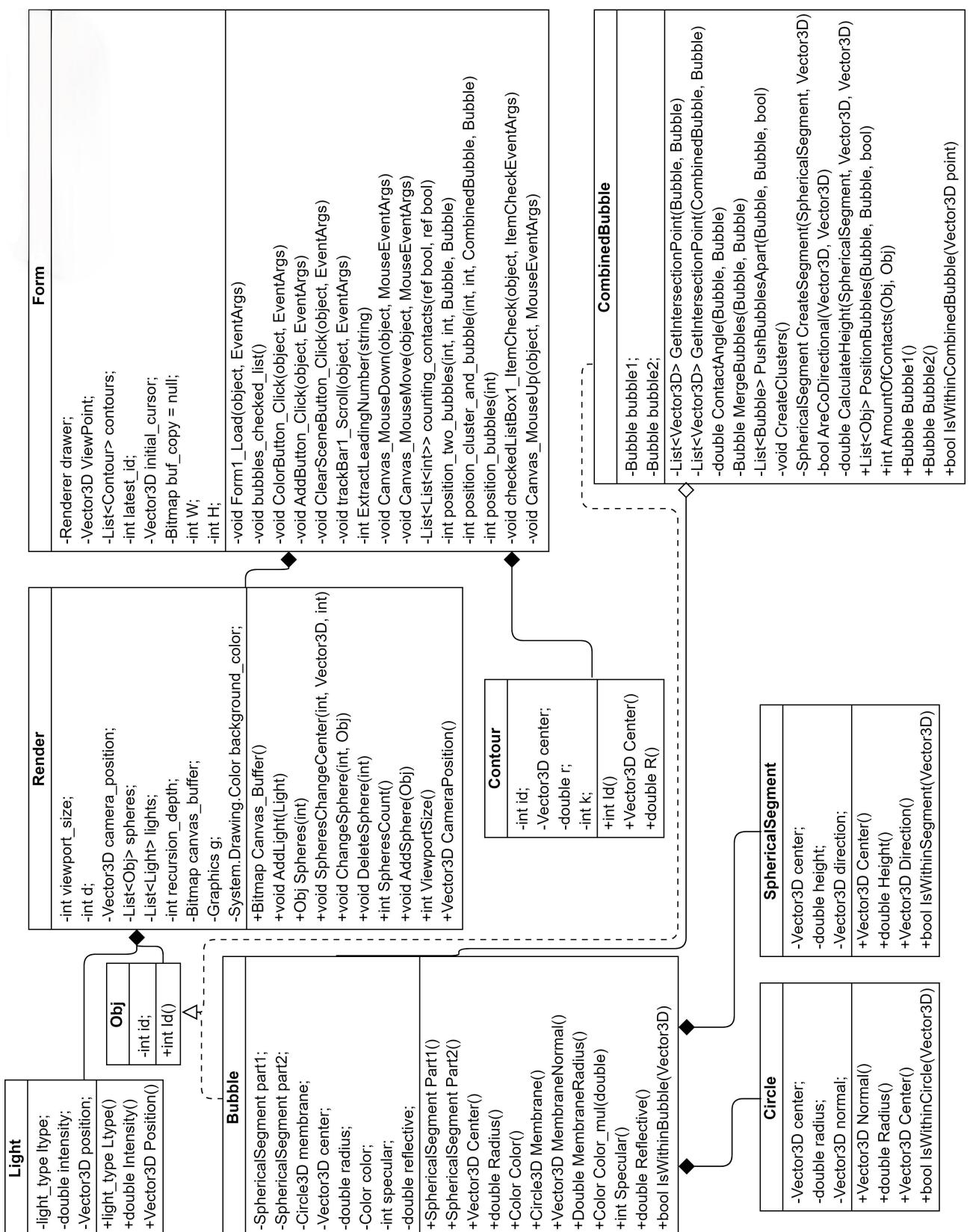


Рисунок 3.1 — Диаграмма классов

3.3 Интерфейс пользователя

На рисунке 3.2 представлен пользовательский интерфейс. Ось ox направлена вправо, oy – вверх, oz – от пользователя. Подробнее:

- 1) блок, позволяющий добавить на сцену ещё один пузырёк (можно указать координаты центра в пространстве, радиус и цвет);
- 2) элемент управления, позволяющий приблизить или отдалить камеру от объектов на сцене, вдоль оси oz ;
- 3) список объектов, присутствующих на сцене (при выборе элемента в этом блоке, его можно будет перемещать по сцене);
- 4) кнопка очистки сцены;
- 5) сцена.

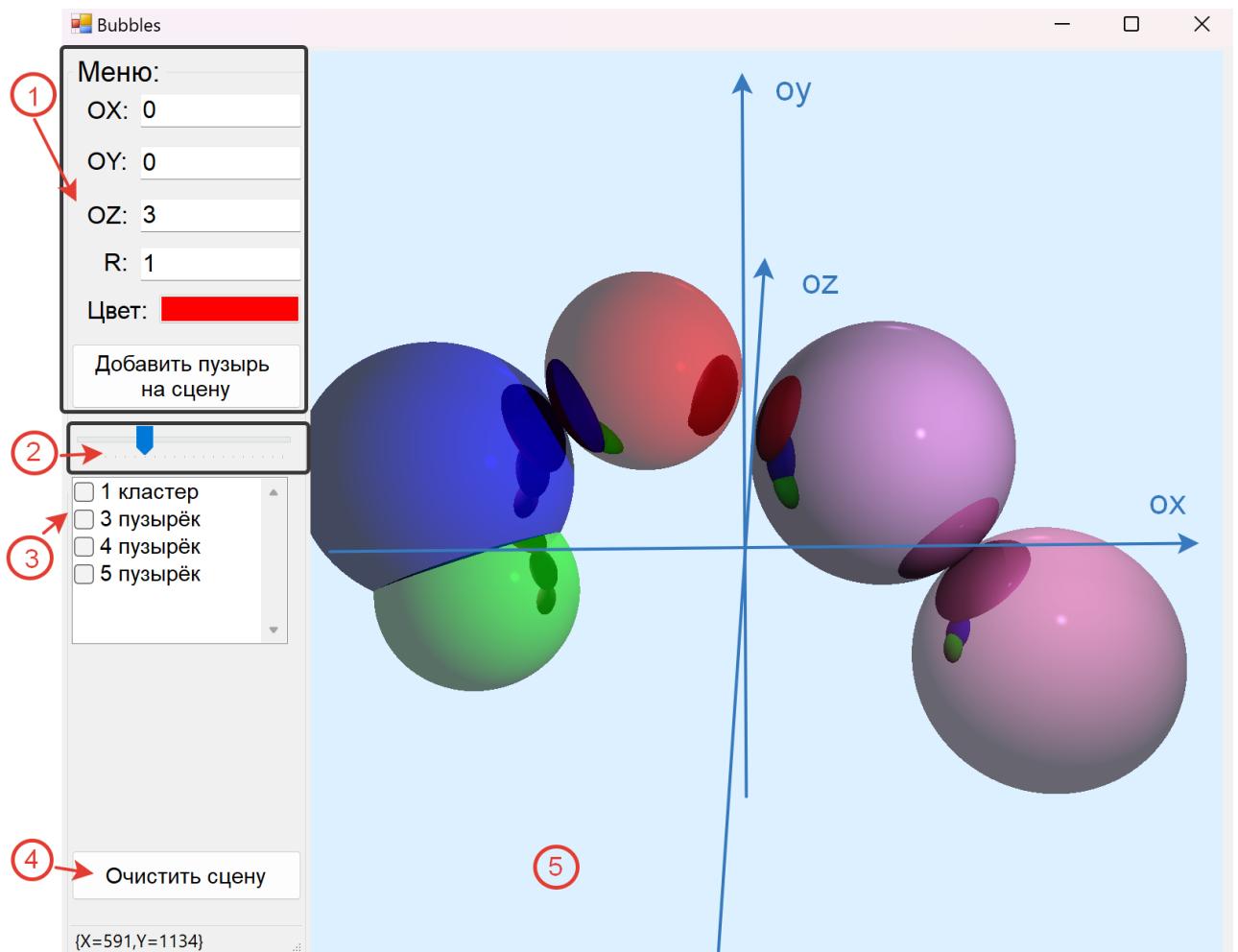


Рисунок 3.2 — Интерфейс пользователя

3.4 Сообщения об ошибке

Если после перемещения или создания пузырь будет иметь более чем одно пересечение с другими пузырями на сцене, будет выведено сообщение об ошибке «Больше чем одно пересечение».

Если после перемещения или создания пузырь с одним из пузырей какого-либо кластера будет иметь предполагающий образование кластера, будет выведено сообщение об ошибке «Угол соприкосновения принадлежит [55, 65]. Невозможно образование кластера из трёх пузырей».

3.5 Функциональное тестирование

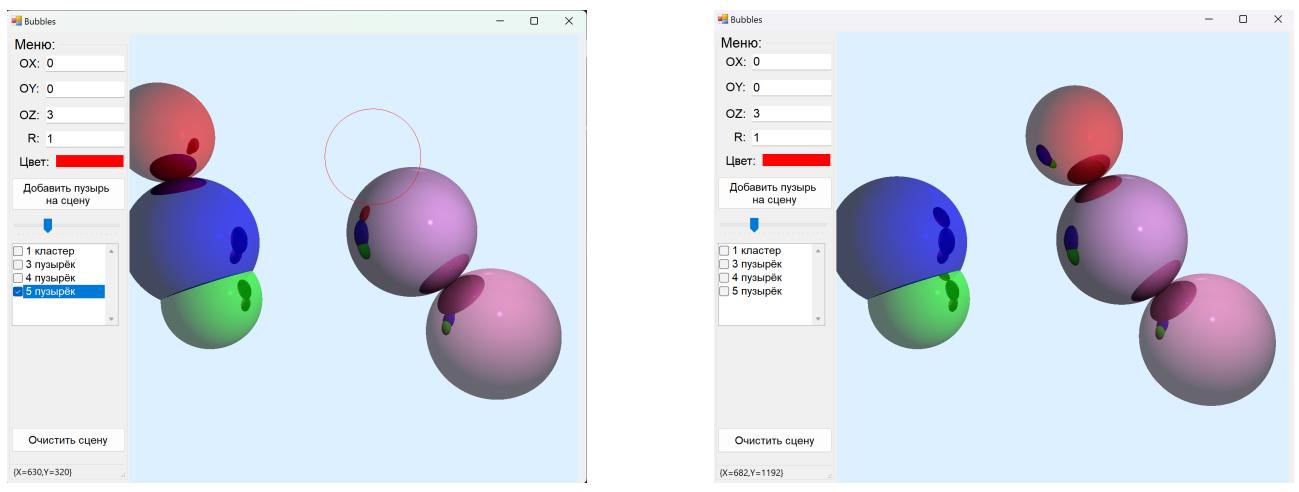
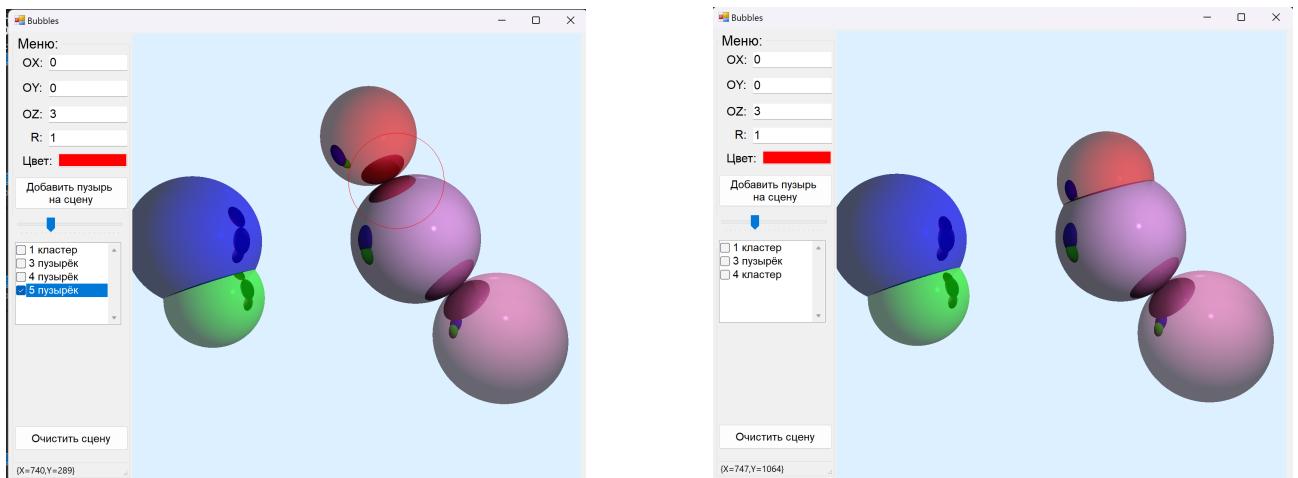


Рисунок 3.3 — Тест №1: отталкивание

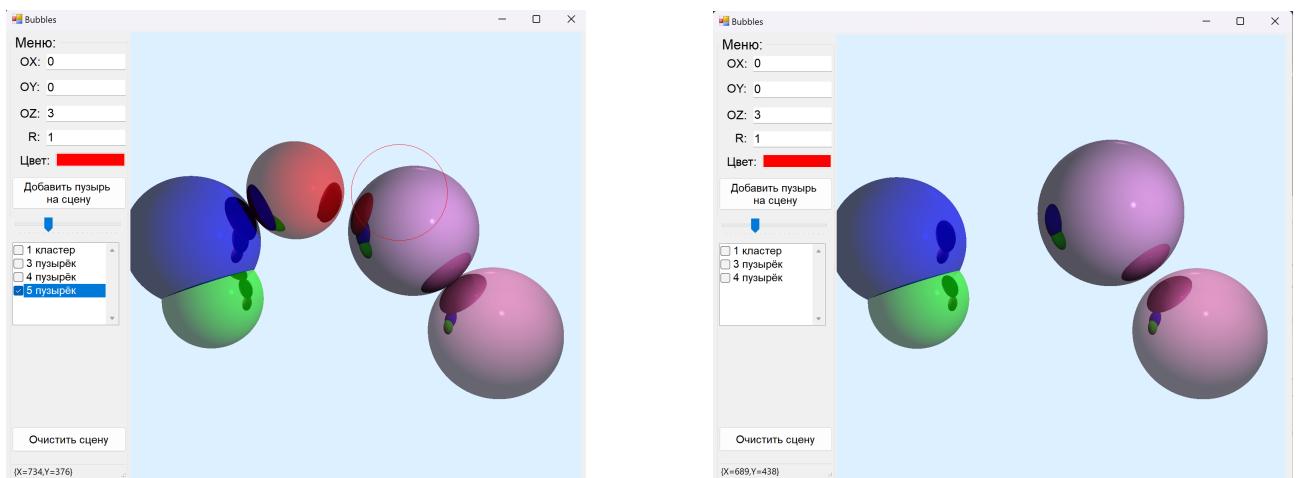
Все функциональные тесты пройдены успешно.



(a) До преобразования

(b) После преобразования

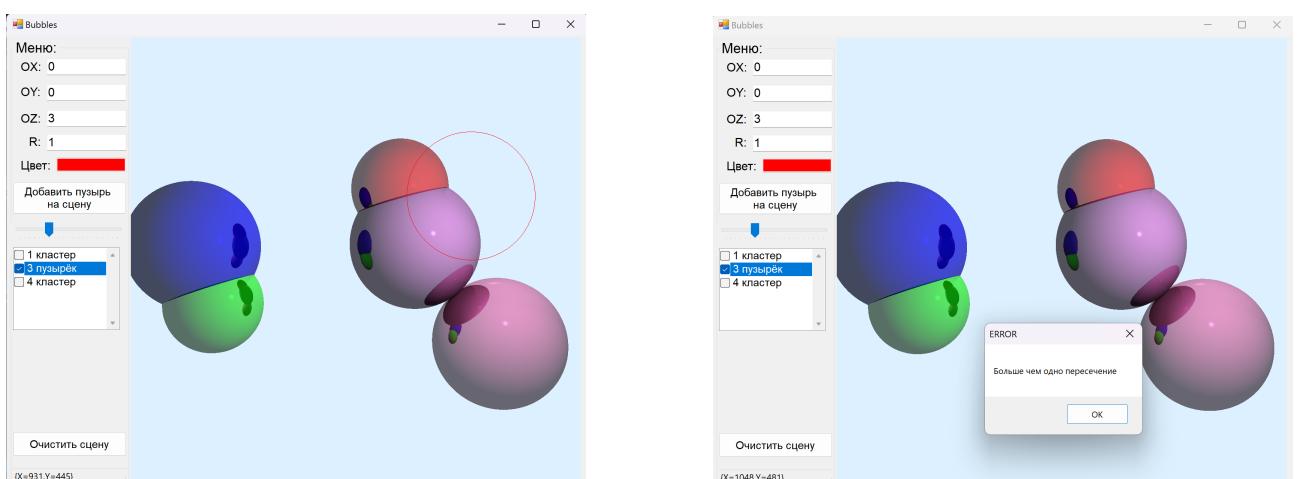
Рисунок 3.4 — Тест №2: образование кластера



(a) До преобразования

(b) После преобразования

Рисунок 3.5 — Тест №3: слияние



(a) До преобразования

(b) После преобразования

Рисунок 3.6 — Тест №4: больше чем одно пересечение

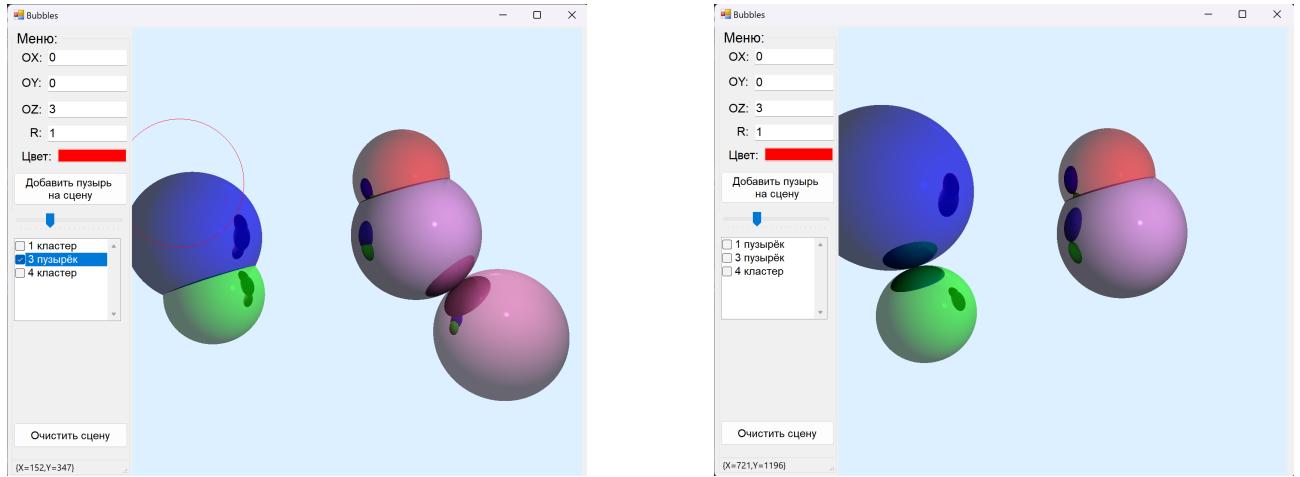


Рисунок 3.7 — Тест №5: слияние с частью кластера

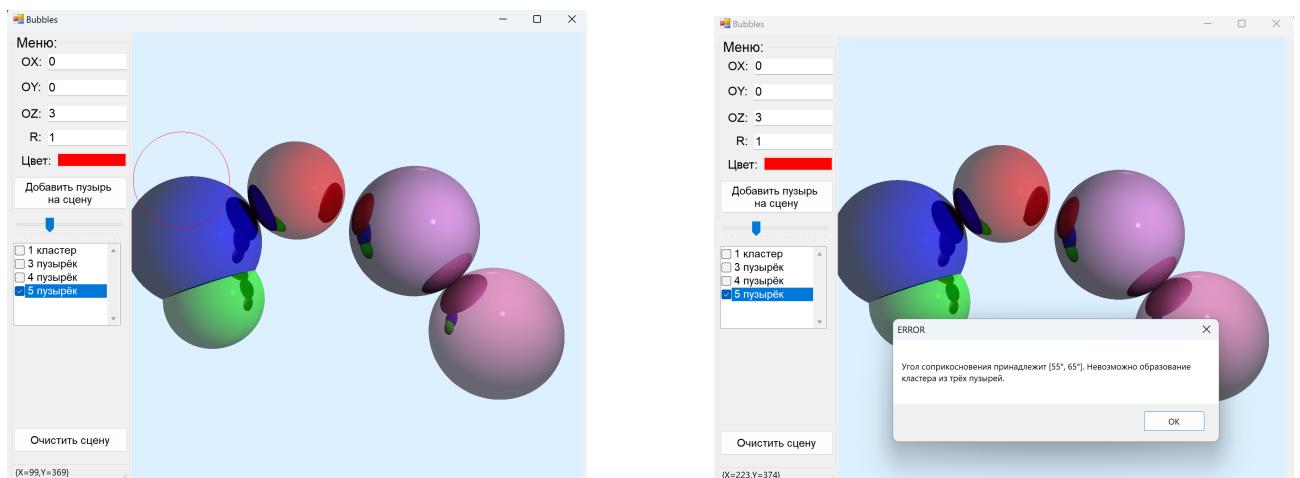


Рисунок 3.8 — Тест №6: кластер из трёх пузырьков

4 Исследовательский раздел

4.1 Исследование характеристик программы

Технические характеристики устройства, на котором выполнялись замеры по времени, представлены далее:

- процессор 12th Gen Intel(R) Core(TM) i7-1260P 2.10 ГГц;
- оперативная память: 16,0 ГБ;
- операционная система: Windows 11.

На рисунке 4.1 продемонстрировано сравнение времени работы алгоритма визуализации сцены от числа объектов для глубины рекурсии от 1 до 3. Можно заметить, что в среднем алгоритм обратной трассировки лучей для трёхмерной сцены с глубиной рекурсии 3 работает дольше остальных, но разница, незначительна, примерно 0.25 секунды. Также стоит отметить, что с возрастанием числа объектов на сцене также возрастает время работы алгоритма.

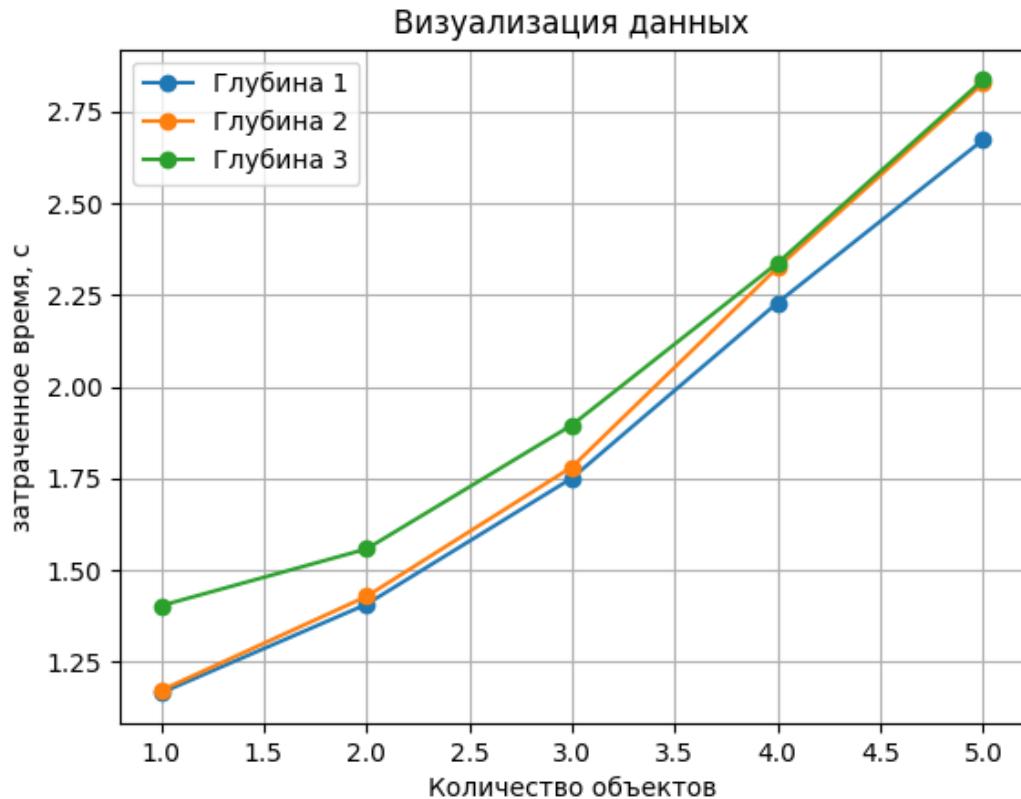


Рисунок 4.1 — Сравнение времени работы алгоритма визуализации сцены от числа объектов для глубины рекурсии от 1 до 3

ЗАКЛЮЧЕНИЕ

Цель достигнута: было произведено моделирование изображения трёхмерной сцены с пузырьковыми кластерами.

Для достижения поставленной цели были выполнены следующие задачи.

- 1) проведён обзор существующих методов визуализации трёхмерной сцены, обоснован выбор метода;
- 2) описан метод взаимодействия пузырей;
- 3) реализована возможность создания отдельных пузырей;
- 4) реализована возможность перемещения отдельных пузырей;
- 5) реализовано взаимодействие пузырей при построении трёхмерного изображения сцены в динамике.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. КРЫЛОВ А. Б. Поверхностное натяжение и связанные с ним явления, Учебно-методическое пособие. 2008. — С. 4.
2. "7.3. Явления на границе раздела газа, жидкости и твердого тела | Физическая термодинамика | МГТУ им. Н.Э. Баумана. Кафедра физики [Электронный ресурс]". "Режим доступа: http://fn.bmstu.ru/data-physics/library/physbook/tom2/ch7/texthtml/ch7_3_text.htm(дата обращения: 21.10.2024)".
3. Бойс Г. Мыльные пузыри. Ленинград: Центральный комитет Всесоюзного Ленинского Коммунистического Союза Молодежи, Издательство детской литературы, 1937.
4. Роджерс Д. Алгоритмические основы машинной графики. 1989. — С. 339 – 360.
5. "Vector3D Структура (System.Windows.Media.Media3D) | Microsoft Learn [Электронный ресурс]". "Режим доступа: <https://learn.microsoft.com/ru-ru/dotnet/api/system.windows.media.media3d.vector3d?view=windowsdesktop-7.0>(дата обращения: 15.12.2024)".

ПРИЛОЖЕНИЕ А

Презентация к курсовой работе

Презентация содержит 17 слайдов.

ПРИЛОЖЕНИЕ Б

Реализации алгоритмов

```
1 public static List<Obj> PositionBubbles(Bubble b1, Bubble b2, bool
2   from_combined)
3 {
4   double contactAngle = ContactAngle(b1, b2);
5   List<Obj> res = new List<Obj>();
6   if (contactAngle.Equals(double.NaN))
7     return null;
8   if (contactAngle < 55.0)
9   {
10     Bubble res_b = MergeBubbles(b1, b2);
11     res.Add(res_b);
12   }
13   else if (contactAngle > 65.0)
14   {
15     List<Bubble> bubbles = PushBubblesApart(b1, b2, from_combined);
16     res.Add(bubbles[0]);
17     res.Add(bubbles[1]);
18   }
19   else
20   {
21     if (from_combined)
22       return null;
23     CombinedBubble cluster = new CombinedBubble(b1.Id, b1, b2);
24     cluster.CreateClusters();
25     res.Add(cluster);
26   }
27   return res;
}
```

Листинг 4.1 — Реализация алгоритма расстановки пузырьков

```
1 private static Bubble MergeBubbles(Bubble b1, Bubble b2)
2 {
3   if (b1 == null || b2 == null)
4     return null;
5
6   Vector3D newCenter = (b1.Center + b2.Center) / 2;
7
8   double volume1 = (4.0 / 3.0) * Math.PI * Math.Pow(b1.Radius, 3);
9   double volume2 = (4.0 / 3.0) * Math.PI * Math.Pow(b2.Radius, 3);
10
11   double totalVolume = volume1 + volume2;
```

```

12
13     double newRadius = Math.Pow((totalVolume * 3.0) / (4.0 * Math.PI), 1.0 / 3.0);
14
15     b1.Center = newCenter;
16     b1.Radius = newRadius;
17
18     b2 = null;
19
20     return b1;
21 }
```

Листинг 4.2 — Реализация слияния в один большой пузырь

```

1 private static List<Bubble> PushBubblesApart(Bubble b1, Bubble b2, bool
2 from_combined)
3 {
4     List<Bubble> res = new List<Bubble>();
5     if (b1 == null || b2 == null)
6     {
7         res.Add(b1);
8         res.Add(b2);
9         return res;
10    }
11
12    Vector3D direction = b2.Center - b1.Center;
13    double currentDistance = direction.Length;
14    double requiredDistance = b1.Radius + b2.Radius;
15
16    if (currentDistance < requiredDistance)
17    {
18        direction.Normalize();
19
20        double pushDistance = requiredDistance - currentDistance;
21        if (from_combined)
22            b2.Center += direction * (pushDistance + 0.0000002);
23        else
24        {
25            b1.Center -= (direction * (pushDistance / 2 + 0.0000001));
26            b2.Center += direction * (pushDistance / 2 + 0.0000001);
27        }
28        b1.Radius = b1.Radius;
29        b2.Radius = b2.Radius;
30    }
31    res.Add(b1);
32    res.Add(b2);
33    return res;
```

33 }

Листинг 4.3 — Реализация отталкивания пузырьков друг от друга

```
1 private void CreateClusters()
2 {
3     if (bubble1 == null || bubble2 == null)
4         return;
5
6     List<Vector3D> Points = GetIntersectionPoint(bubble1, bubble2);
7     Vector3D centerOfIntersect = Points[0];
8
9     Vector3D membraneNormal = bubble2.Center - bubble1.Center;
10    membraneNormal.Normalize();
11
12    bubble1.MembraneNormal = membraneNormal;
13    bubble2.MembraneNormal = membraneNormal;
14
15    SphericalSegment newPart1A = CreateSegment(bubble1.Part1, bubble1.
16        MembraneNormal);
16    SphericalSegment newPart1B = CreateSegment(bubble1.Part2, bubble1.
17        MembraneNormal);
18
18    SphericalSegment newPart2A = CreateSegment(bubble2.Part1, bubble2.
19        MembraneNormal);
19    SphericalSegment newPart2B = CreateSegment(bubble2.Part2, bubble2.
20        MembraneNormal);
21
21    newPart1A.Direction = membraneNormal;
22    newPart1B.Direction = -newPart1A.Direction;
23
24    newPart2A.Direction = -newPart1A.Direction;
25    newPart2B.Direction = -newPart1B.Direction;
26
27    newPart1A.Height = CalculateHeight(newPart1A, centerOfIntersect,
28        newPart1A.Direction);
28    newPart1B.Height = CalculateHeight(newPart1B, centerOfIntersect,
29        newPart1B.Direction);
29    newPart2A.Height = CalculateHeight(newPart2A, centerOfIntersect,
30        newPart2A.Direction);
30    newPart2B.Height = CalculateHeight(newPart2B, centerOfIntersect,
31        newPart2B.Direction);
31
32    bubble1.Part1 = newPart1A;
33    bubble1.Part2 = newPart1B;
34    bubble2.Part1 = newPart2A;
35    bubble2.Part2 = newPart2B;
```

```

36
37     double membraneRadius = bubble1.MembraneRadius;
38     bubble1.Membrane = new Circle3D(centerOfIntersect, membraneRadius,
39     bubble1.MembraneNormal);
40     bubble2.Membrane = new Circle3D(centerOfIntersect, membraneRadius,
41     bubble2.MembraneNormal);
42 }
```

Листинг 4.4 — Реализация создания пузырькового кластера

```

1 private int position_bubbles(int n)
2 {
3     if (n <= 0)
4         return -1;
5     bool any_intersection = false;
6     bool any_more_than_2 = false;
7     List<List<int>> a = counting_contacts(ref any_intersection, ref
any_more_than_2);
8
9     if (any_more_than_2)
10    {
11        MessageBox.Show(
12            "More than two contacts",
13            "ERROR");
14        return -1;
15    }
16    else if (any_intersection == false)
17    {
18        Console.WriteLine("no intersections");
19        return 0;
20    }
21    else
22    {
23        for (int i = 0; i < a.Count; i++)
24            for (int j = 0; j < a[i].Count; j++)
25            {
26                if (a[i][j] == 1)
27                {
28                    int rc = 0;
29                    if (drawer.Spheres(i) is Bubble && drawer.Spheres(j) is Bubble)
30                        rc = position_two_bubbles(i, j, (Bubble)drawer.Spheres(i),
(Bubble)drawer.Spheres(j));
31                    else if (drawer.Spheres(i) is CombinedBubble && drawer.Spheres(j)
is Bubble)
32                        rc = position_cluster_and_bubble(i, j, (CombinedBubble)drawer.
Spheres(i), (Bubble)drawer.Spheres(j));
33                    else if (drawer.Spheres(i) is Bubble b4 && drawer.Spheres(j) is
Bubble)
34                        rc = position_bubble_and_bubble(i, j, (Bubble)b4, (Bubble)drawer.
Spheres(j));
35                }
36            }
37        }
38    }
39 }
```

```

        CombinedBubble cb2)
34            rc = position_cluster_and_bubble(j, i, (CombinedBubble)drawer.
Spheres(j), (Bubble)drawer.Spheres(i));
35            if (rc == -1)
36                return -1;
37            a[i][j] = 0;
38            a[j][i] = 0;
39            position_bubbles(n);
40        }
41    }
42}
43
44    return position_bubbles(n - 1);
45}

```

Листинг 4.5 — Реализация алгоритма анализа объектов сцены

```

1 public void Render()
2 {
3     g.FillRectangle(new SolidBrush(Form1.DefaultBackColor),
4         0, 0, canvas_buffer.Width, canvas_buffer.Height);
5
6     int w = canvas_buffer.Width / 2;
7     int h = canvas_buffer.Height / 2;
8
9     for (double x = -w; x < w; x++)
10    {
11        for (double y = -h; y < h; y++)
12        {
13            Vector3D direction = CanvasToViewport(w, h, x, y);
14            System.Drawing.Color color = TraceRay(new TraceRayArgs((int)x, (int)
y, camera_position, direction, 1, double.PositiveInfinity,
recursion_depth));
15            PutPixel((int)x, (int)y, color, false);
16        }
17    }
18}
19 System.Drawing.Color TraceRay(Object obj)
20{
21    if (obj is TraceRayArgs args)
22    {
23        KeyValuePair<Bubble, double> intersection = ClosestIntersection(args.
origin, args.direction, args.min_t, args.max_t);
24        if (intersection.Key == null)
25            return background_color;
26
27        Bubble closest = intersection.Key;

```

```

28     double closest_t = intersection.Value;
29
30     Vector3D point = args.origin + args.direction * closest_t;
31     Vector3D normal = point - closest.Center;
32     normal.Normalize();
33
34     Vector3D view = args.direction * (-1);
35     double lighting = ComputeLighting(point, normal, view, closest.
36 Specular);
37     System.Drawing.Color local_color = closest.Color_mul(lighting);
38
39     if (closest.Reflective <= 0 || args.depth <= 0)
40         return local_color;
41
42     double koef = 1 - closest.Reflective;
43     System.Drawing.Color local_contribution = System.Drawing.Color.
44 FromArgb(
45         (int)(local_color.R * koef),
46         (int)(local_color.G * koef),
47         (int)(local_color.B * koef)
48     );
49
50     Vector3D reflected_ray = ReflectRay(view, normal);
51     System.Drawing.Color reflected_color = TraceRay(new TraceRayArgs(args.
52 x, args.y, point, reflected_ray, EPSILON,
53         double.PositiveInfinity, args.depth - 1));
54     System.Drawing.Color reflected_contribution = System.Drawing.Color.
55 FromArgb(
56         (int)(reflected_color.R * closest.Reflective),
57         (int)(reflected_color.G * closest.Reflective),
58         (int)(reflected_color.B * closest.Reflective)
59     );
60
61     System.Drawing.Color result_color = System.Drawing.Color.FromArgb(
62         Math.Min(255, local_contribution.R + reflected_contribution.R),
63         Math.Min(255, local_contribution.G + reflected_contribution.G),
64         Math.Min(255, local_contribution.B + reflected_contribution.B)
65     );
66
67     return result_color;
68 }
69
70 return background_color;
71 }
```

Листинг 4.6 — Реализация алгоритма обратной трассировки лучей