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

КУРСОВАЯ РАБОТА

НА ТЕМУ:

Визуализация процесса извержения вулкана

Студент ИУ7-64Б (группа) _____ (подпись, дата) И. С. Бугаков (И.О. Фамилия)

Руководитель курсового проекта _____ (подпись, дата) Т. Н. Романова (И.О. Фамилия)

2025 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Вулканическая деятельность оказывает существенное влияние на окружающую среду, экосистемы и ландшафты Земли. Компьютерное модели вулканов активно применяется в научных исследованиях, геологии, мониторинге катастроф и образовательных программах. Современные технологии позволяют не только визуализировать вулканическую активность, но и анализировать её влияние на окружающую среду, что открывает новые возможности для исследований и прогнозирования. [1]

Цель курсовой работы – разработка программного обеспечения для визуализации процесса извержения вулкана. В рамках реализации курсовой работы должны быть решены следующие задачи:

— провести анализ алгоритмов удаления невидимых линий и поверхностей, построения освещения, визуализации и выбрать наиболее подходящие для решения задачи;
— спроектировать программное обеспечение;
— выбрать средства реализации программного обеспечения и разработать его;
— провести исследование характеристик разработанного программного обеспечения.

1 Аналитический раздел

1.1 Объекты сцены

Сцена состоит из следующих объектов:
— вулкан — эффузивное геологическое образование, имеющее кратер, из которого вулканические газы поступают на поверхность из недр планеты. Основной объект сцены;
— источник света на бесконечности — источник света для задания фонового освещения;
— стол тефры — дисперсная масса, выбрасываемая из жерла вулкана при извержении.

1.2 Способы задания модели трехмерных объектов

1.2.1 Каркасная

Модель состоит из вершин и ребер, без данных о внутренних точках поверхностей, ограниченных этими ребрами. [2].

1.2.2 Поверхностная

Поверхность объекта может быть представлена либо аналитическими выражениями, либо полигональной сеткой. Данная модель описывает исключительно внешние геометрические характеристики, такие как форма и границы, но не включает сведения о внутреннем строении и не определяет, где относительно поверхности расположен материал. [2].

1.2.3 Объемная

Поверхность каждого объекта представлена как непрерывная и математически описываемая. Такая модель полностью характеризует трёхмерную форму и включает данные о материале, из которого создан объект. Также содержится информация о направлении внутренней нормали, что позволяет определить расположение материала относительно поверхности. [2].

Для визуализации вулкана была выбрана поверхностная модель, в связи с распространенностью STL моделей, являющихся поверхностными, для представления реально существующего ландшафта земной поверхности.

1.3 Алгоритмы закраски

1.3.1 Простой алгоритм закраски

В простом алгоритме закраски интенсивность цвета определяется для всего полигона на основе внешней нормали к его поверхности с использованием закона Ламберта. При таком алгоритме закраски граница между полигонами получается четко выраженной. Данный метод закраски используется в случаях, когда применение более продвинутых алгоритмов ведет к сильному увеличению вычислительной сложности.

1.3.2 Закраска Гуро

Метод Гуро основывается на интерполяции интенсивности света, при которой разные точки на грани получают различные значения освещенности. Для этого вычисляются нормали в каждой вершине грани, а затем значения интенсивности интерполируются между точками смежных граней. Нормали для граней определяются в первую очередь, затем нормали в вершинах рассчитываются как среднее значение нормалей прилегающих граней. На основе этих нормалей вычисляются значения интенсивности с учетом выбранной модели отражения света, после чего полигоны граней закрашиваются с учетом линейной интерполяции интенсивности в вершинах.

Этот метод эффективно работает для небольших граней, находящихся далеко от источника света. Однако, когда грань становится достаточно большой, расстояние от источника света до её центра будет меньше, чем до вершин, что должно привести к более яркому освещению центра. Из-за линейной интерполяции, применяемой в методе, этого эффекта не удается достичь, что приводит к неестественному распределению освещенности на некоторых участках грани. [2].

1.3.3 Закраска Фонга

Метод закраски по Фонгу использует интерполяцию векторов нормалей вместо простой интерполяции интенсивности вдоль сканирующей строки. Это позволяет точно моделировать кривизну поверхности и создавать изображения с высокой реалистичностью. Такой подход особенно эффективен для симуляции зеркальных отражений, поскольку он обеспечивает плавные переходы освещенности и высокую детализацию. В результате метод Фонга минимизирует эффект полос на поверхностях и создает более гладкие и детализированные переходы света, что способствует получению естественного и детализированного изображения.

Для визуализации вулкана был выбран простой метод закраски, т. к. модель представ-

лена большим числом полигонов, для которых определены только нормали к поверхности. Вычисление нормалей к вершинам полигонов и интерполяция цвета приводят к значительному увеличению вычислений и замедлению визуализации.

1.4 Алгоритмы удаления невидимых линий и поверхностей

1.4.1 Алгоритм Робертса

Алгоритм Робертса является первым известным методом удаления невидимых линий и работает в объектном пространстве. Он предназначен для обработки выпуклых объектов и выполняет удаление рёбер и граней, экранируемых самим телом. Далее оставшиеся видимые рёбра каждого объекта сравниваются со всеми другими объектами сцены, чтобы определить, какие их части оказываются перекрытыми. Метод обладает вычислительной сложностью, возрастающей пропорционально квадрату числа объектов. [2].

1.4.2 Алгоритм Варнока

Алгоритм работает в контексте изображения, выполняя анализ содержимого окна. Его задача — определить, является ли окно пустым или слишком сложным для визуализации. В случае, если содержимое окна слишком детализировано, алгоритм рекурсивно разделяет окно на более мелкие сегменты, пока каждый из них не станет достаточно простым для отображения или не достигнет минимального разрешения. Основным недостатком такого метода является увеличение вычислительных затрат и сложности из-за использования рекурсии, что может привести к замедлению обработки изображений с высоким уровнем детализации. [2].

1.4.3 Алгоритм z – буфера

Этот алгоритм для удаления невидимых поверхностей является одним из базовых. Он функционирует в пространстве изображения, расширяя концепцию буфера кадра. Буфер кадра используется для хранения атрибутов (например, интенсивности) каждого пикселя на изображении. Вместе с буфером кадра используется Z-буфер, представляющий собой специализированную память для хранения координат Z (глубины) каждого видимого пикселя. В процессе работы глубина нового пикселя, который требуется добавить в буфер кадра, сравнивается с глубиной пикселя, уже записанного в Z-буфер. Если новый пиксель находится ближе к наблюдателю, чем тот, который уже находится в буфере, его значения добавляются в буфер кадра, и Z-буфер обновляется соответствующей глубиной. Если же новый пиксель дальше, то ника-

ких изменений не происходит. Таким образом, алгоритм для каждой точки (x, y) вычисляет максимальное значение функции Z(x, y). [2].

1.4.4 Алгоритм трассировки лучей

В этом методе для каждого пикселя на изображении определяется ближайшая грань, к которой этот пиксель относится. Для этого через пиксель направляется луч, который пересекает все грани, и из этих пересечений выбирается ближайшее. Алгоритм трассировки лучей включает несколько шагов. Сначала необходимо найти уравнение луча, который проходит через рассматриваемый пиксель и точку взгляда наблюдателя. Вычисляется точка пересечения этого луча с ближайшим объектом сцены. После этого пиксель закрашивается цветом объекта, с которым луч пересекся, и процесс продолжается для следующего пикселя. В случае, если пересечений с объектами нет, пиксель закрашивается цветом фона, и алгоритм переходит к следующему пикселю. Трассировка лучей учитывает процессы отражения, преломления и прохождения через поверхности объектов, заканчиваясь, когда луч встречает непрозрачный объект, который виден наблюдателю. [2].

Для моделирования вулкана был выбран Z-буфер, т. к. метод позволяет эффективно работать с объектами, представленными большим числом полигонов, например, реальными горными ландшафтами.

1.5 Анализ моделей освещения

1.5.1 Модель Ламберта

Модель Ламберта является одной из простейших моделей освещения, в которой учитывается только идеальное диффузное отражение света от поверхности. В этой модели предполагается, что свет, падающий на точку поверхности, равномерно рассеивается во всех направлениях полупространства, и освещенность в точке зависит лишь от плотности света в этой точке. Освещенность линейно зависит от косинуса угла падения света, при этом положение наблюдателя не имеет значения, так как диффузно отраженный свет равномерно рассеивается во всех направлениях.

Основной закон, описывающий диффузное отражение, — это закон косинусов Ламберта, который утверждает, что интенсивность отраженного света пропорциональна косинусу угла падения света на поверхность. Этот закон выражается следующим уравнением:

$$I = I_l k_d \cos(\theta), \tag{1.1}$$

где:

— I — интенсивность отраженного света,

— I_l — интенсивность падающего света от точечного источника,

— k_d — коэффициент диффузного отражения,

— θ — угол между направлением падающего света L и нормалью к поверхности N.

Закон может быть также представлен в виде, использующем нормированный вектор нормали к поверхности и нормированный вектор направления падающего луча света:

$$I = I_l \cdot (L \cdot N) \tag{1.2}$$

1.5.2 Модель Фонга

Модель освещения Фонга сочетает три компонента: фоновое освещение, диффузное рассеяние и зеркальное отражение. Модель Фонга представляет собой комбинированную модель освещения, которая учитывает как диффузное, так и зеркальное отражение света, а также фоновое освещение. Она особенно полезна для моделирования ярких блестящих объектов, так как зеркальное отражение вызывает появление световых бликов на поверхностях с высокой степенью отражения, таких как металлы или влага.

Коэффициент зеркального отражения в модели Фонга зависит от угла падения света. Даже при перпендикулярном падении света только часть энергии отражается зеркально, а остальная часть либо поглощается, либо рассеивается диффузно. Эти соотношения зависят от материала поверхности и длины волны света. Например, для различных материалов (металлы, стекло, матовые поверхности) коэффициенты отражения будут различаться, что влияет на видимость бликов и отражений.

Формула модели освещения Фонга:

$$I = I_a \cdot K_a + I_d \cdot K_d \cdot (L \cdot N) + I_s \cdot K_s \cdot (R \cdot V)^n. \tag{1.3}$$

Модель Ламберта учитывает только диффузное отражение света, что подходит для визуализации горной поверхности, которая имеет матовую структуру и рассеивает свет в разных направлениях.

1.6 Алгоритм для визуализации поведения газов

Уравнения Навье—Стокса описывают движение газа и жидкости, и могут быть использованы для моделирования различных физических процессов, включая генерацию дыма. Для симуляции дыма в рамках уравнений Навье-Стокса необходимо учитывать динамику потока воздуха и взаимодействие частиц дыма с окружающей средой.

Математически состояние дыма в данный момент времени моделируется как векторное

поле скорости, которое назначает вектор скорости каждой точке в пространстве. Для визуализации дыма, в том числе в контексте извержения вулкана, можно применить математическую модель, основанную на уравнениях Навье—Стокса, которая описывает изменение векторного поля скорости во времени. Уравнение 1.4 этой модели дает точное описание того, как скорость жидкости или газа будет изменяться в зависимости от действующих сил, что важно для создания визуальных эффектов дыма. При этом, поскольку общих решений уравнений Навье-Стокса не существует, подход предложенный Дж. Стэмом [4] предполагает упрощенное восприятие этих уравнений и использование метода Гаусса—Зейделя для решения СЛАУ [3, 7].

$$\frac{\partial \vec{u}}{\partial t} = -(\vec{u} \cdot \nabla)\vec{u} + \nu\nabla^2\vec{u} + f \tag{1.4}$$

где:

— u — вектор скорости жидкости или газа;

— $\frac{\partial \vec{u}}{\partial t}$ — изменение скорости по времени;

— $(\vec{u} \cdot \nabla)\vec{u}$ — нелинейный член, описывающий изменение скорости из-за движения жидкости;

— $\nu\nabla^2\vec{u}$ — вязкость (диффузия);

— $\vec{f}$ – внешние силы (например, гравитация, теплоотдача).

В контексте визуализации дыма, особенно при моделировании извержений вулканов, необходимо моделировать движение частиц, связанных с потоком горячих газов и пепла. Поскольку моделирование каждой частицы дыма требует значительных вычислительных ресурсов, эффективным решением является использование плотности дыма — непрерывной функции, которая определяет количество частиц в каждой точке пространства. Изменение плотности через поле скорости жидкости или газа может быть описано с использованием уравнения 1.5, аналогичного уравнению для скорости, но с упрощением, поскольку модель плотности является линейной, в отличие от нелинейности модели скорости. Для визуализации дымовых эффектов, в том числе при моделировании вулканических извержений, модель плотности дыма используется для расчета того, как частицы будут распространяться в пространстве, учитывая изменения в потоке горячих газов и пепла. Визуализация достигается путем отображения плотности дыма в различных точках, что позволяет создать реалистичные эффекты подъема и распространения дыма, характерные для извержений вулканов.

$$\frac{\partial \rho}{\partial t} = -(\vec{u} \cdot \nabla)\rho + \kappa\nabla^2\rho + S \tag{1.5}$$

где:

— ρ — плотность жидкости или газ;

— $(\vec{u} \cdot \nabla)\rho$ — изменение плотности дыма из-за потока жидкости или газа;

— $\kappa\nabla^2\rho$ — диффузия дыма (влияние турбулентности или молекулярной диффузии);

— S — источник дыма (например, извержение вулкана, где горячие газы и пепел выбрасываются в атмосферу).

Таким образом необходимо сначала вычислить изменение поля скорости за заданный промежуток времени. Вычисления производятся в порядке, обратно порядку слагаемых, в правой части уравнения 1.4: в первую очередь добавляются источники поля, т. е. внешние силы; затем вычисляется рассеяние поля с учетом вязкости среды; после чего - самоадвекция поля скорости, т. е. перенос поля под действием себя же. При этом в расчете самоадвекции необходимо учесть закон сохранения массы [4].

После обновления поля скорости, вычисляется изменение распределения плотности дыма в пространстве, аналогично вычислению изменения поля скорости. Обработка слагаемых в уравнении 1.5 вновь производится в обратном порядке: добавляются источники дыма с высокой начальной плотностью, которая в дальнейшем будет перенесена полем скорости; после этого вычисляется диффузия дыма; наконец вычисляется адвекция плотности дыма полем скорости. Так как к этому моменту для поля скорости выполняется закон сохранения массы, использовать его при вычислении изменения распределения плотности нет необходимости.

Вывод

Была выбрана поверхностная модель для визуализации вулкана. Для удаления невидимых линий и поверхностей выбран алгоритм использующий z-буфер. Освещение рассчитывается по модели Ламберта, для закраски выбран простой алгоритм. Был рассмотрен алгоритм для визуализации поведения газов.

2 Конструкторский раздел

В конструкторской части разработано программное обеспечение, а также формально описаны применяемые алгоритмы.

2.1 Схемы алгоритмов визуализации трехмерной сцены

2.1.1 Алгоритм, использующий z-буфер

На рисунках 2.1 представлена схема алгоритма с использованием z-буфера. В алгоритме выполняется разложение треугольника в растр, после чего вычисляется глубина полученного пикселя. Вычисленная глубина сравнивается с записанной в z-буфере, в зависимости от результата может быть выполнено обновление z-буфера.

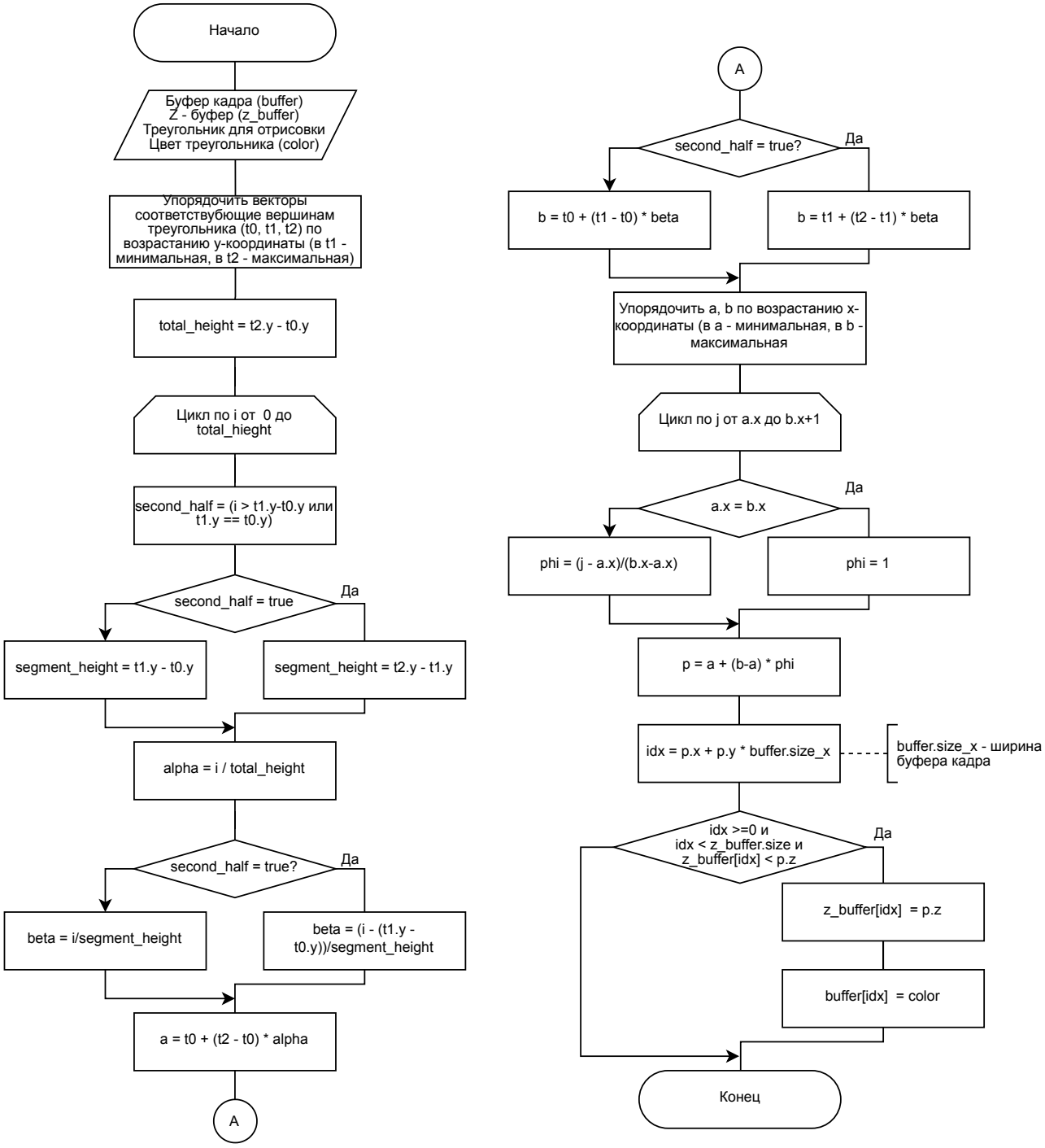

Рисунок 2.1 — Схема алгоритма с использованием z-буфера.

2.1.2 Алгоритм поведения газов

На рисунке 2.2 представлена схема алгоритма поведения газа на основе уравнений Навье—Стокса.

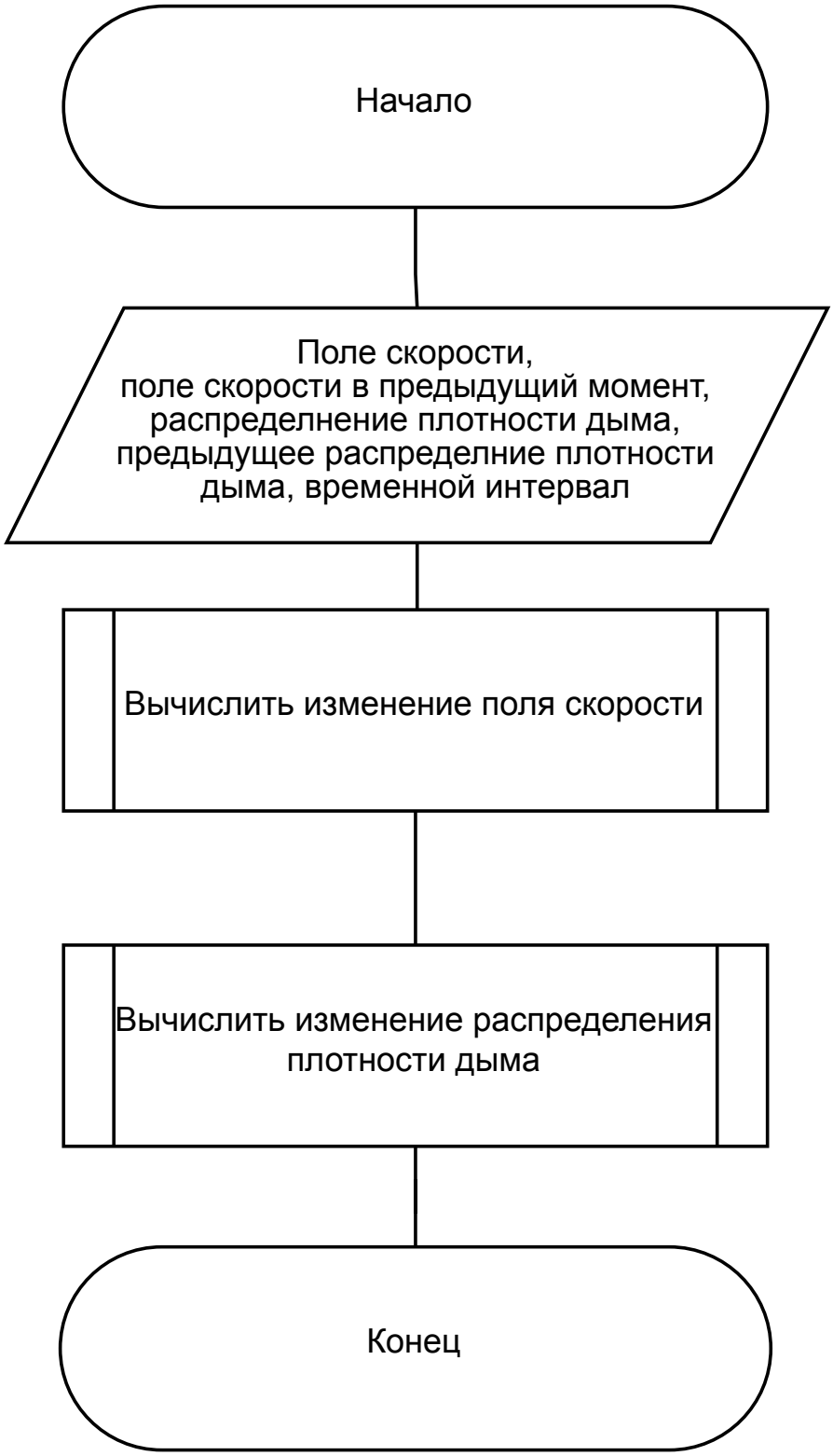

Рисунок 2.2 — Схема алгоритма на основе уравнения Навье—Стокса.

На рисунке 2.3 представлена схема алгоритма обновления распределения плотности дыма.

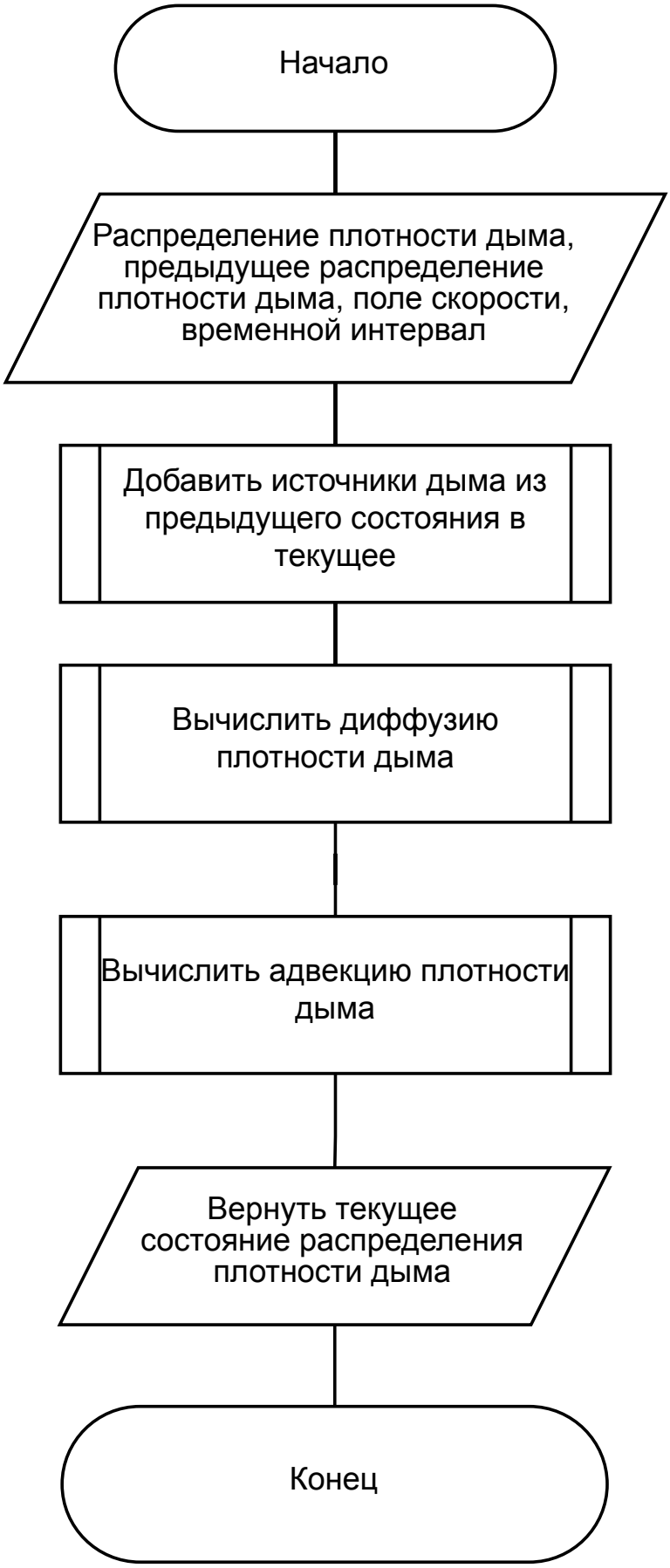

Рисунок 2.3 — Схема алгоритма обновления распределения плотности дыма.

На рисунке 2.4 представлена схема алгоритма обновления поля скорости.

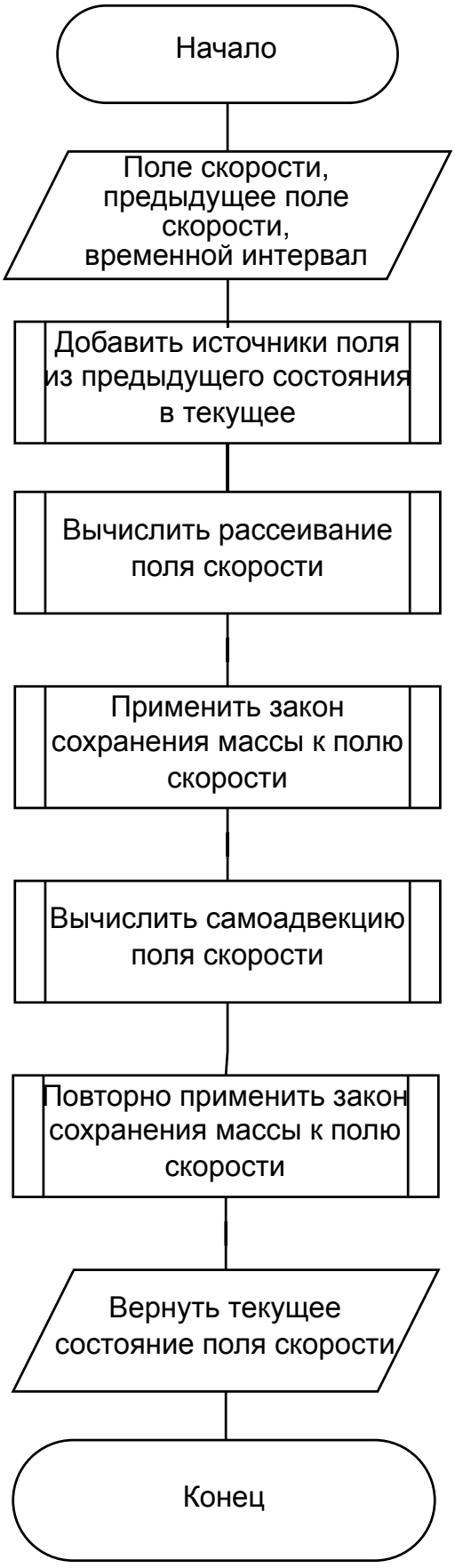

Рисунок 2.4 — Схема алгоритма обновления поля скорости.

На рисунке 2.5 представлена схема алгоритма добавления источников в систему, используемая как в алгоритме обновления поля скорости, так и в алгоритме обновления распределения плотности дыма.

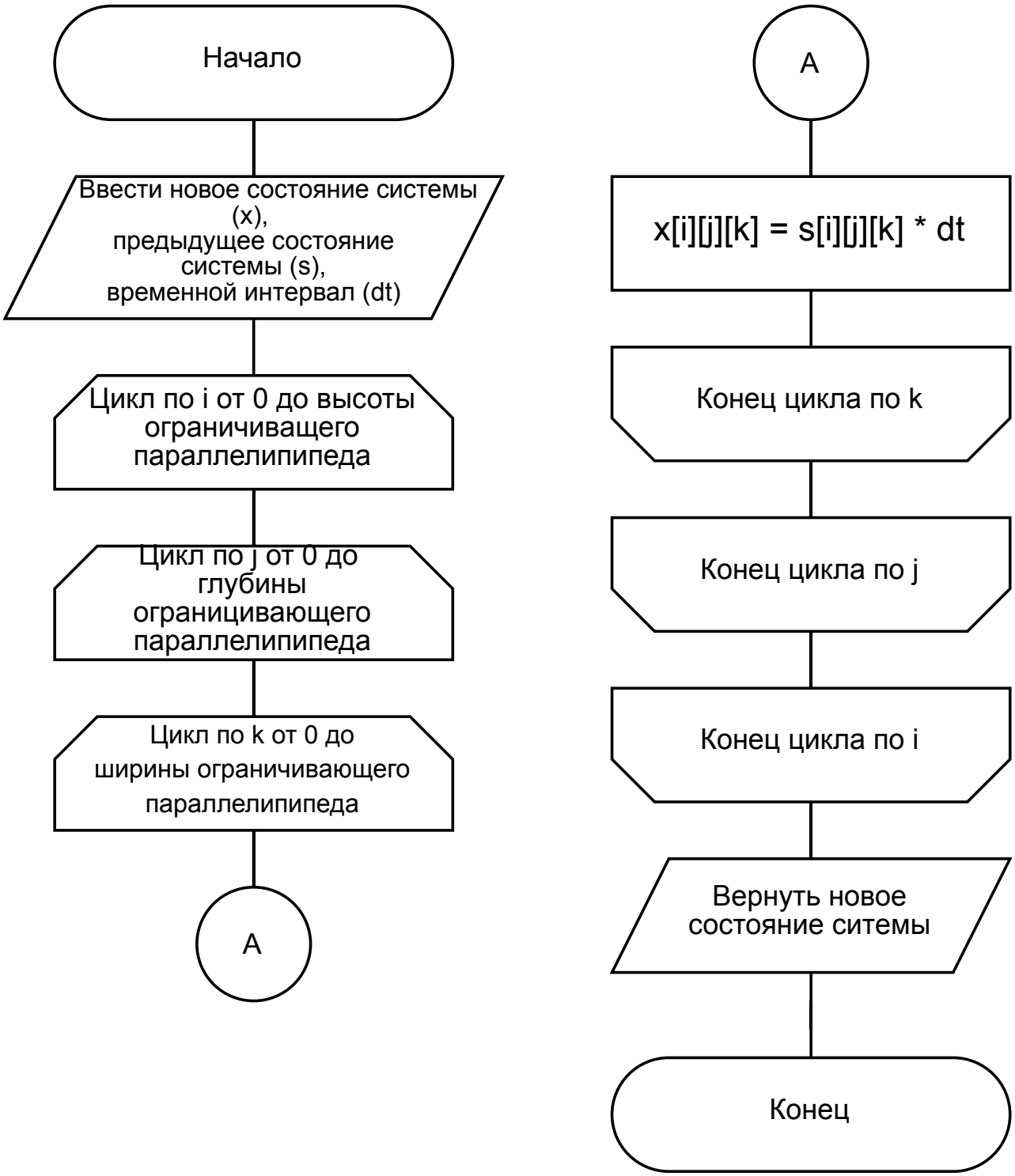

Рисунок 2.5 — Схема алгоритма добавления источников в систему.

На рисунке 2.7 представлена схема алгоритма вычисления диффузии (рассеивания), используемого для обновления поля скорости и распределения плотности газа. В обоих случаях под диффузией предполагается обмен ячейки ограничивающего параллелепипеда плотностью

или скоростью с соседними по граням ячейками. Пример для двумерной симуляции представлен на рисунке 2.6.

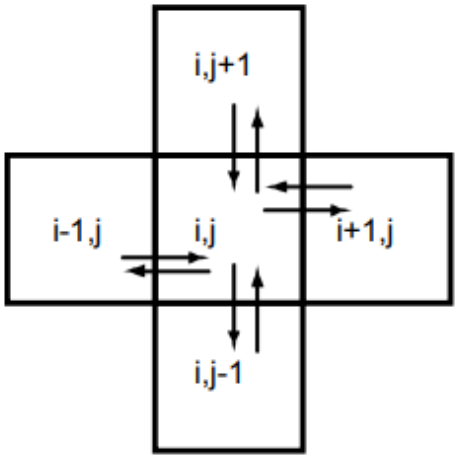

Рисунок 2.6 — Пример диффузии для двумерной симуляции.

Для трехмерного случая вычисление нового распределения плотности сводится к решению системы уравнений вида 2.1, относительно неизвестных x. При этом рассматривается изменение распределения плотности в обратном временном порядке, т. е. для ячейки i, j, k находится плотность, которая при распространении назад во времени дает плотности предыдущего распределения. Для решения СЛАУ применяется итеративный метод Гаусса-Зейделя, алгоритм которого представлен на рисунке 2.8.

$$x_{0_{i,j,k}} = x_{i,j,k} - D \cdot (x_{i-1,j,k} + x_{i+1,j,k} + x_{i,j-1,k} + x_{i,j+1,k} + x_{i,j,k-1} + x_{i,j,k-1} - 6 * x_{i,j,k}) \quad (2.1)$$

где:

— x_0 — исходное распределение плотности;

— x — вычисляемое распределение плотности;

— D — коэффициент диффузии.

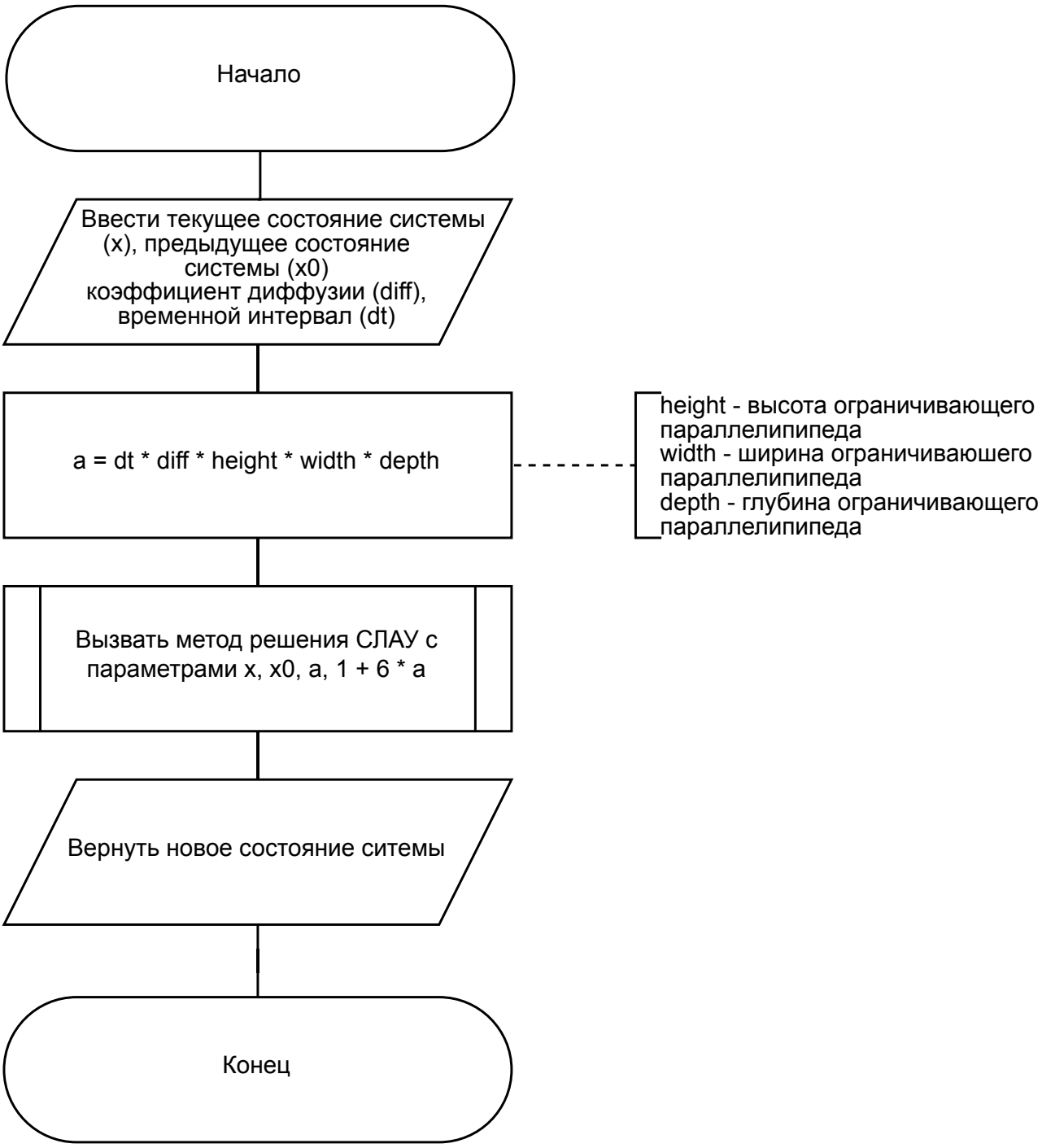

Рисунок 2.7 — Схема алгоритма вычисления диффузии (рассеивания).

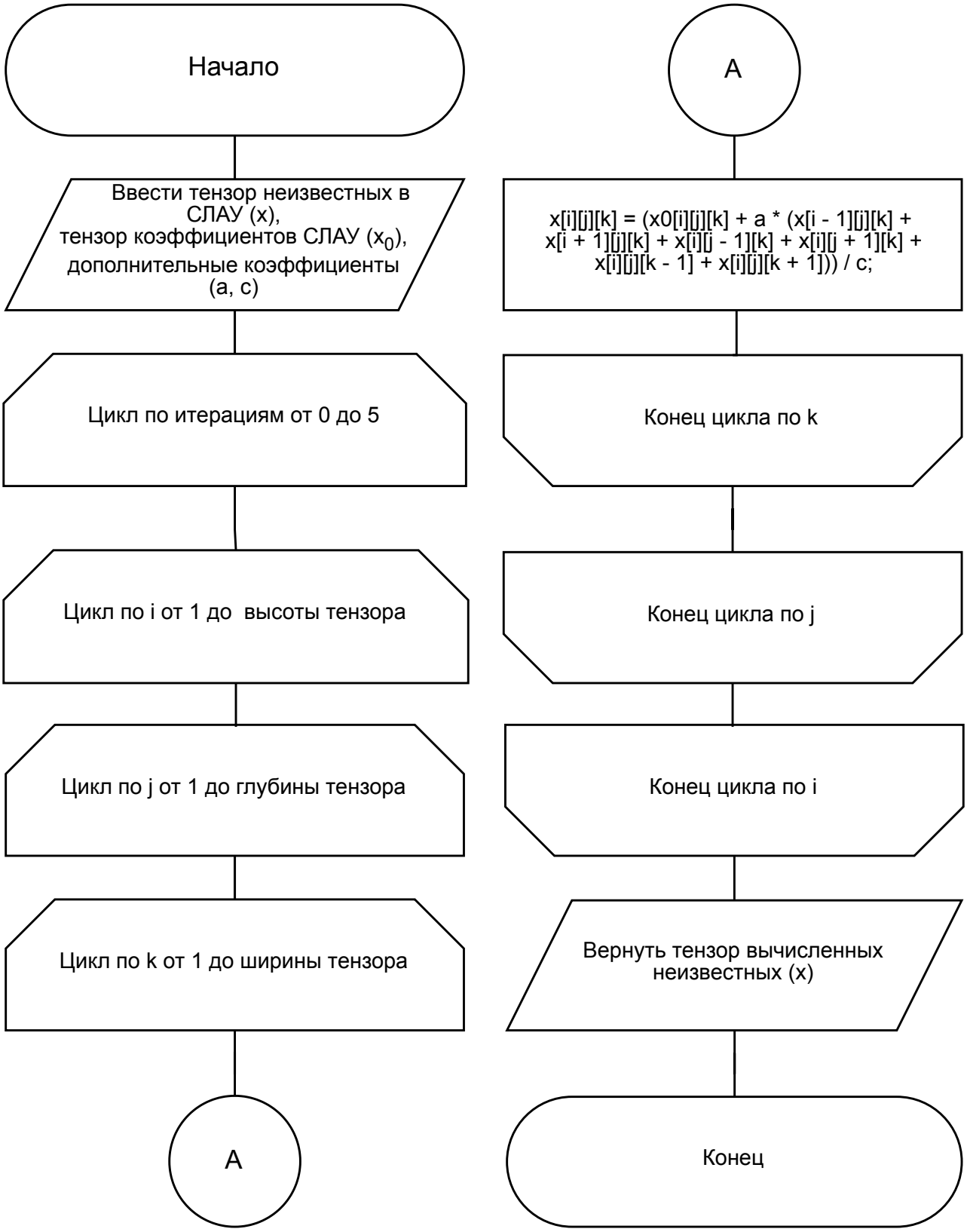

Рисунок 2.8 — Схема алгоритма решения СЛАУ.

На рисунке 2.10 представлена схема алгоритма вычисления адвекции и самоадвекции для распределения плотности и поля скорости соответственно. Для вычисления адвекции также используется подход обратного временного распространения. Для значений, которые в итоге

находятся в центре ячейки ограничивающего параллелепипеда, линейной интерполяцией определяются доли значений соседних ячеек перенесенных полем скорости за единичный временной шаг. На рисунке 2.9 проиллюстрирована идея вычисления адвекции для двумерной симуляции.

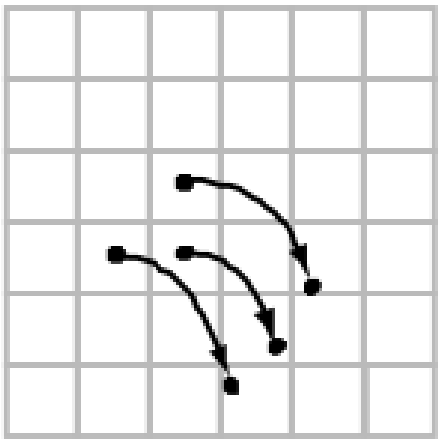

Рисунок 2.9 — Пример вычисления адвекции. Стрелками показано обратное распространение плотности во времени.

Таким образом исходная точка, откуда была перенесена плотность за один временной интервал вычисляются по формуле 2.2.

$$\vec{p_0} = \vec{p} - \vec{v} \cdot \Delta t \tag{2.2}$$

где:

— $\vec{p_0}$ — вычисленная исходная точка;

— $\vec{p}$— точка соответствующая центру ячейки, для которой вычисляется новое значение плотности;

— $\vec{v}$ — скорость в точке, соответствующей центру ячейки, для которой вычисляется новое значение плотности;

— Δt — временной интервал.

Для каждой координаты полученной исходной точки определяется отрезок интерполяции с помощью формул 2.3 (пример приведен для x-координаты).

$$\begin{aligned} x_0 &= \lfloor x \rfloor \\ x_1 &= x_0 + 1 \end{aligned} \tag{2.3}$$

где:

— x — x-координата вычисленной исходной точки;

— x_0, x_1 — нижняя и верхняя границы отрезка интерполяции соответственно.

Далее для каждой координаты определяются расстояния до границ отрезка интерполя-

ции по формулам 2.4 (пример также приведен для x-координаты).

$$\begin{aligned} k_{x1} &= p_{0_x} - x_0 \\ k_{x0} &= 1 - k_{x1} \end{aligned} \tag{2.4}$$

где:

— p_{0_x} — x-координата вычисленной исходной точки;

— x_0 — нижняя граница отрезка интерполяции;

— k_{x1}, k_{x0} — расстояния до нижней и верхней границ интерполяции соответственно.

Наконец, проводится интерполяция плотности в точке p по формуле 2.5.

$$\begin{aligned} d_p = k_{x0}(k_{y0}k_{z0} * d0_{x_0,y_0,z_0} + k_{y1}k_{z0}d0_{x_0,y_1,z_0} + k_{y0} * k_{z1} * d0_{x_0,y_0,z_1} + k_{y1} * k_{z1} * d0_{x_0,y_1,z_1}) + \\ k_{x1} * (k_{y0} * k_{z0} * d0_{x_1,y_0,z_0} + k_{y1} * k_{z0} * d0_{x_1,y_1,z0} + k_{y0} * k_{z1} * d0_{x1,y0,z1} + k_{y1} * k_{z1} * d0_{x1,y1,z1}) \end{aligned} \tag{2.5}$$

где:

— d_p — вычисляемое распределение плотности в точке p;

— $d0$ — распределение плотности в один временной интервал назад;

— x_0, y_0, z_0 — координаты нижней границы интерполяции;

— x_1, y_1, z_1 — координаты верхней границы интерполяции;

— k_{x0}, k_{y0}, k_{z0} — расстояние до нижней границы интерполяции по каждой из x, y, z координат соответственно;

— k_{x1}, k_{y1}, k_{z1} — расстояние до верхней границы интерполяции по каждой из x, y, z координат соответственно.

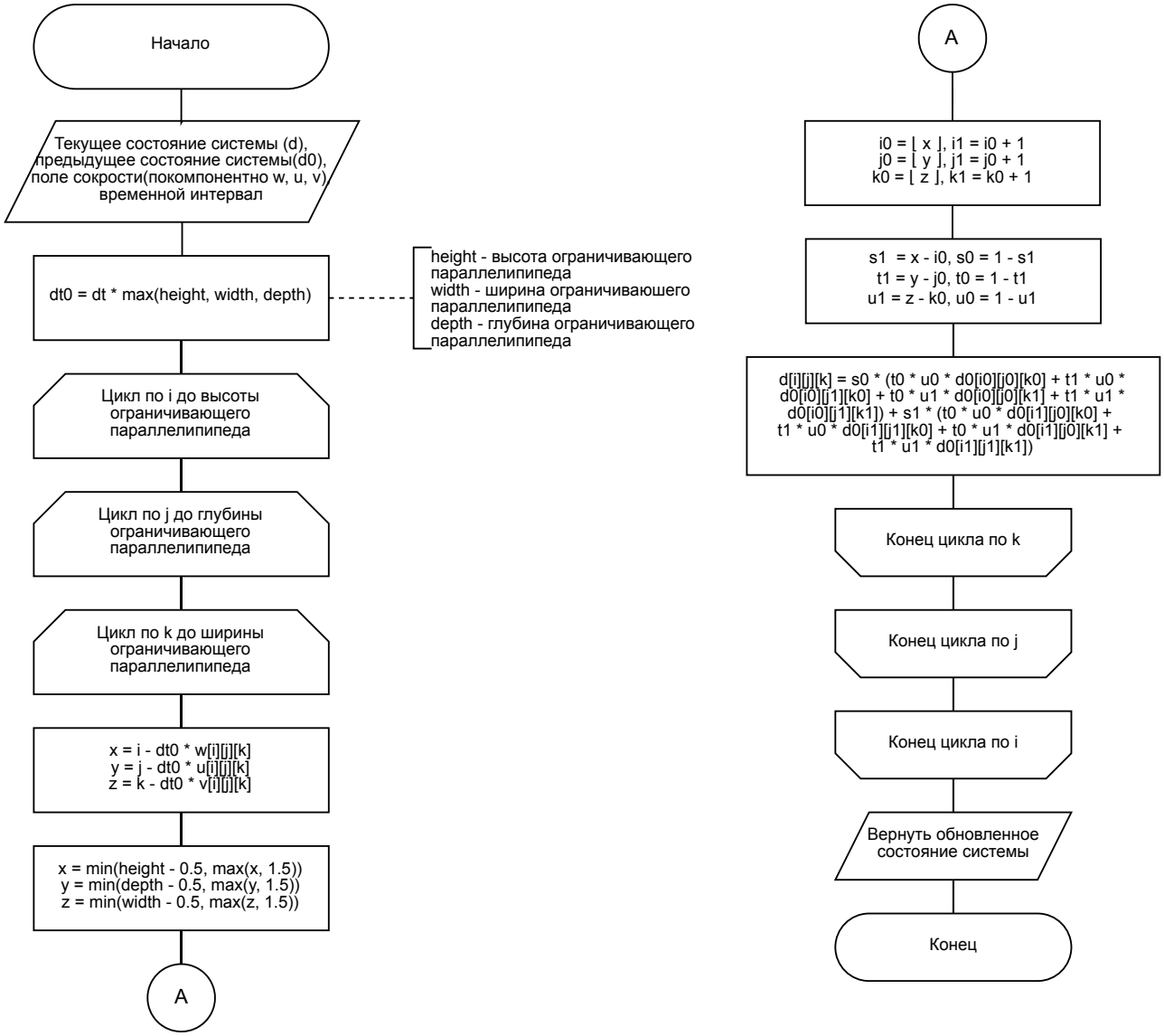

Рисунок 2.10 — Схема алгоритма вычисления адвекции.

На рисунке 2.12 представлена схема алгоритма применения закона сохранения массы к полю скорости. Так как несжимаемость является неотъемлемой частью реальных газов, необходимо, чтобы данный закон выполнялся для итогового поля скорости. В соответствии с декомпозицией Ходжа, каждое поле скорости является суперпозицией несжимаемого поля и градиентной составляющей, как показано на рисунке 2.11. Для получения поля, которое будет сохранять массу необходимо в каждой точке вычесть из существующего поля градиент. Таким образом, задача сводится к нахождения градиента в точке, для чего можно решить систему линейных уравнений методом, использованным ранее. Закон сохранения массы применяется дважды лишь потому, что вычисление адвекции происходит точнее, когда для поля скорости выполняется данный закон [4].

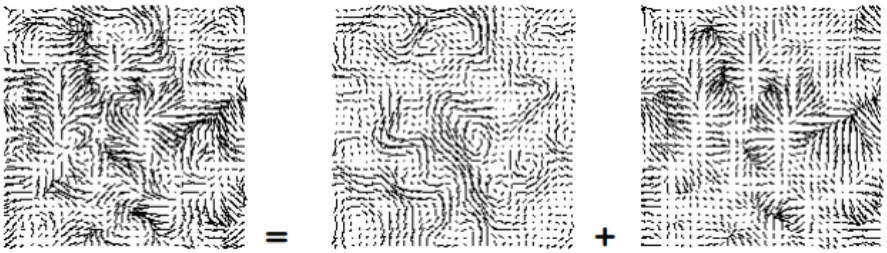

Рисунок 2.11 — Декомпозиция Ходжа. После знака равенства: слева — несжимаемая составляющая, справа — градиентная.

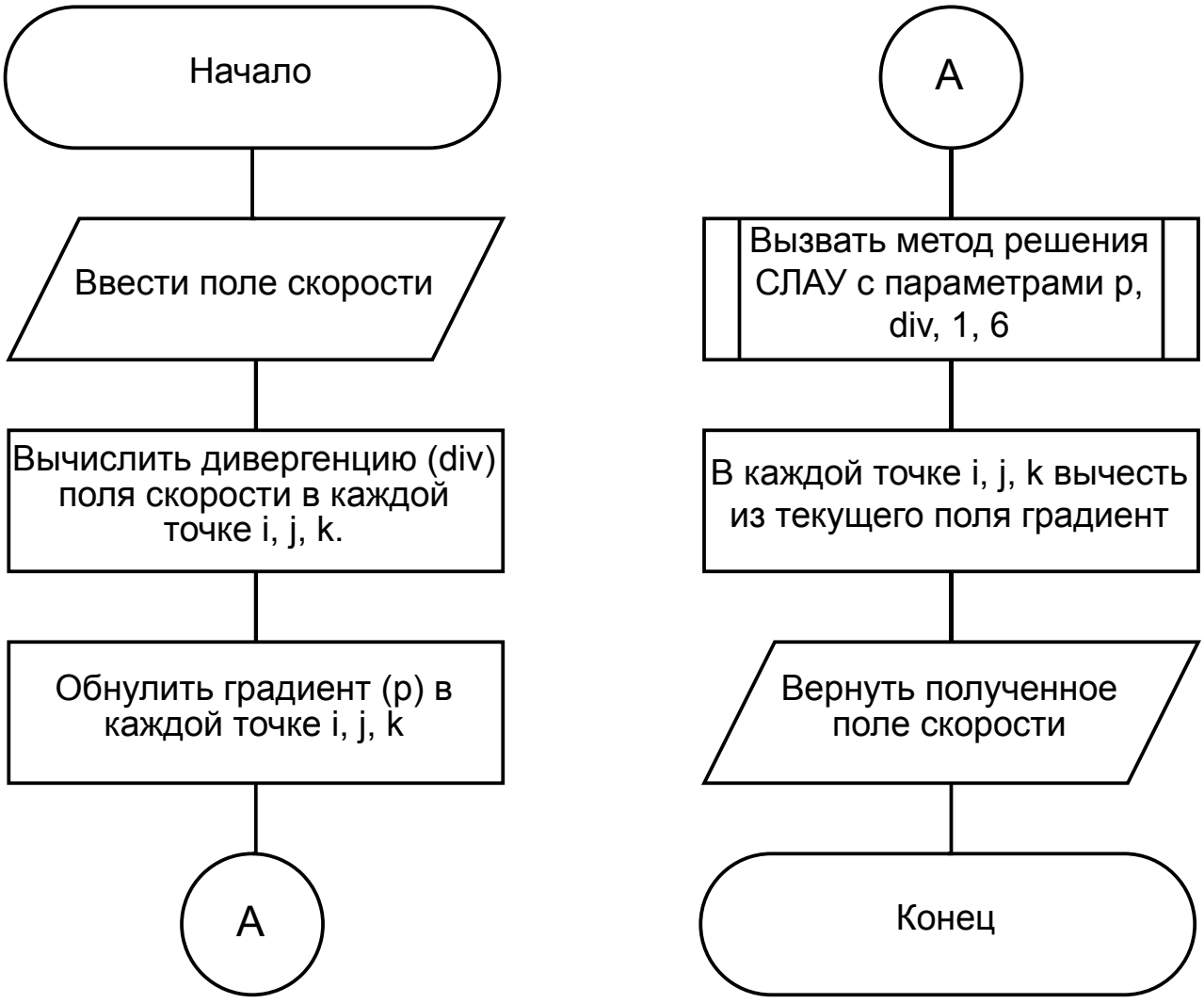

Рисунок 2.12 — Схема алгоритма применения закона сохранения массы.

2.2 Диаграмма классов

На рисунке 2.13 представлена UML-диаграмма классов программного обеспечения.

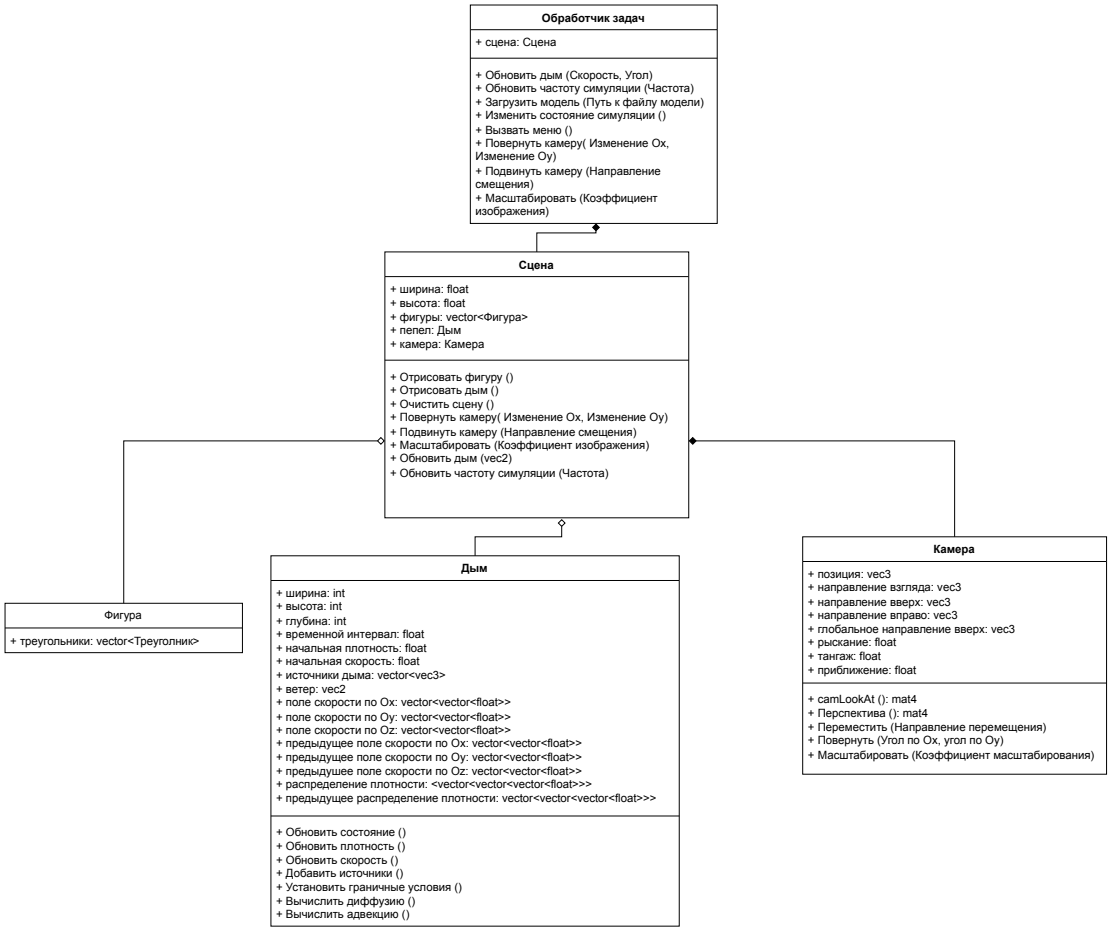

Рисунок 2.13 — UML-диаграмма классов.

Вывод

В разделе спроектировано разрабатываемое программное обеспечение для визуализации извержения вулкана, приведены схемы алгоритмов и UML-диаграмма классов ПО.

3 Технологический раздел

В технологическом разделе выбраны и описаны средства реализации программного обеспечения и представлены детали его реализации. В качестве языка программирования был выбран C++, т. к. он как он позволяет реализовать все алгоритмы, выбранные в результате проектирования, а также поддерживает все требуемые структуры данных. Для создания пользовательского интерфейса использовался фреймворк Qt, т. к. в нем присутствуют необходимые для этого инструменты, для задач визуализации — библиотека SFML.

3.1 Графический интерфейс программы

На рисунке 3.1 представлен интерфейс программы. Интерфейс позволяет загрузить модель вулкана, задать скорость и направление ветра, частоту вывода кадров симуляции и перейти к окну визуализации.

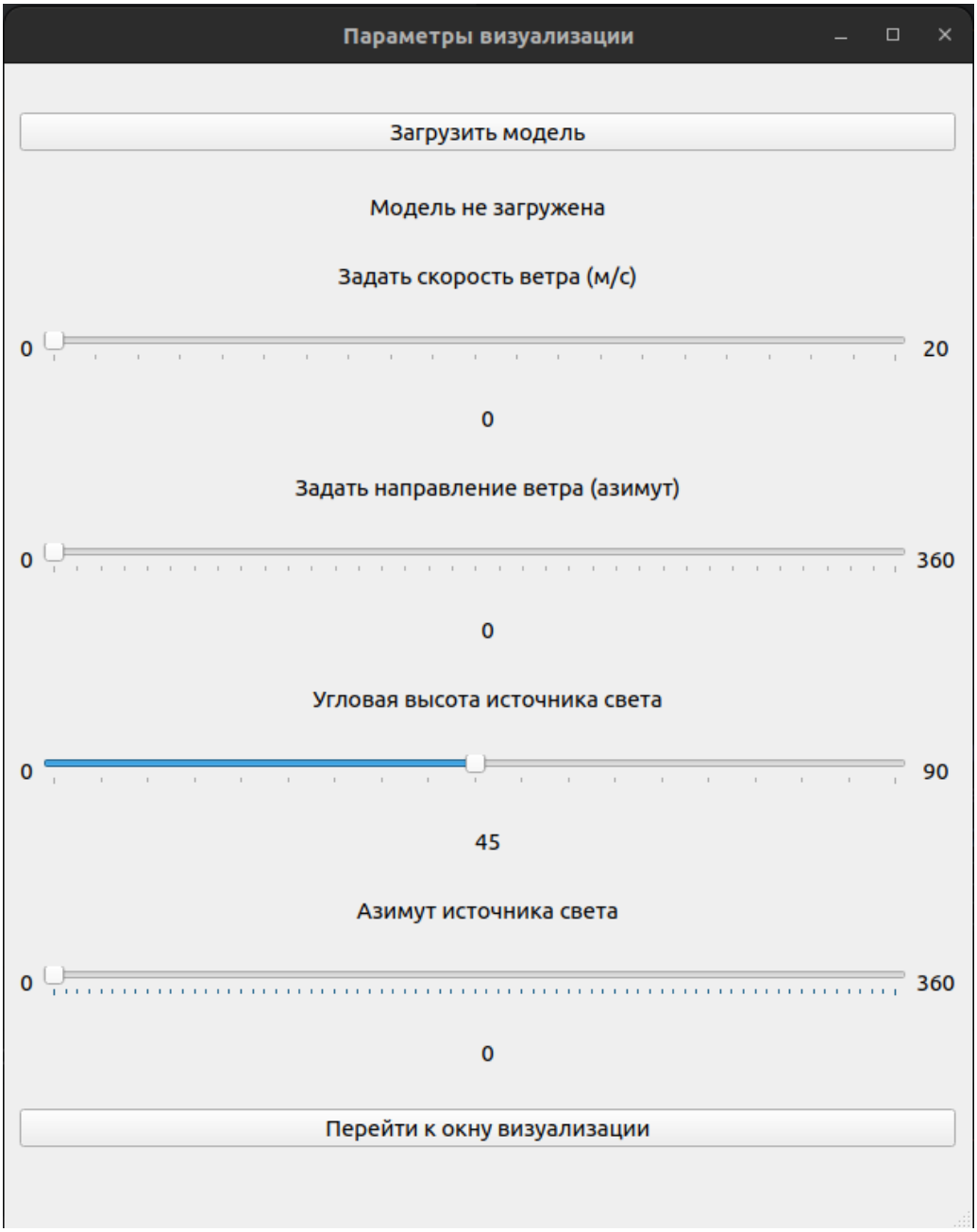

Рисунок 3.1 — Интерфейс программы.

На рисунке 3.2 представлено окно визуализации. Для перемещения камеры используются клавиши W, A, S, D, для масштабирования — колесико мыши, для поворота камеры — перемещение мыши с зажатой левой кнопкой. Визуализацию можно поставить на паузу с помощью клавиши Enter, вызвать окно настроек клавишей Escape.

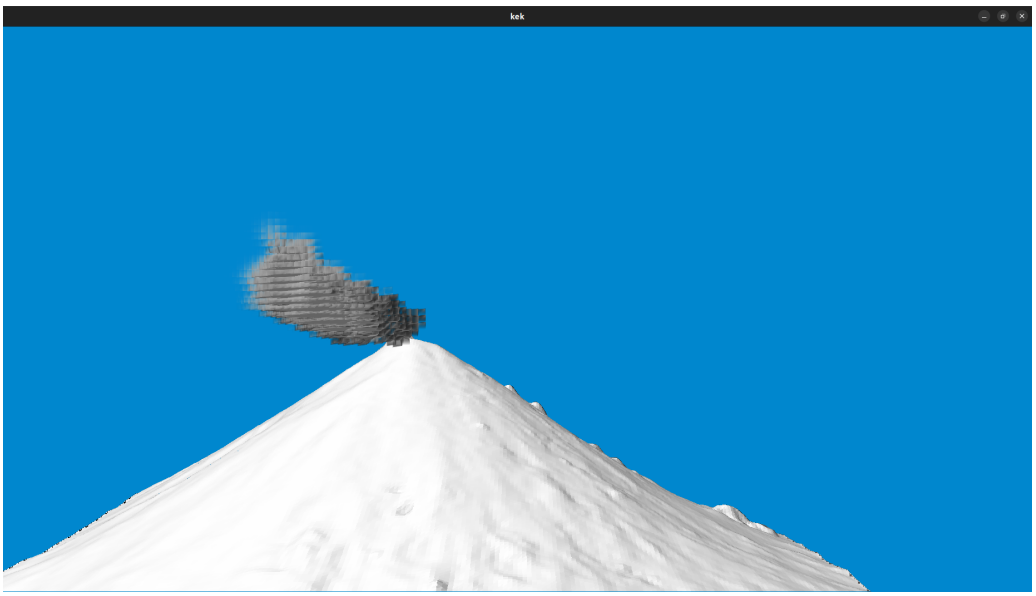

Рисунок 3.2 — Окно визуализации

3.2 Реализация алгоритмов

В листинге 3.1 представлен листинг функции с использованием z-буфера для визуализации модели вулкана.

Листинг 3.1 — Функция использующая z-буфер для визуализации треугольника

```
void z_buffer(array<glm::vec3, 3> points, sf::RenderTarget &image,
   const sf::Color &color,
vector<float> &z_buffer) {
  if (abs(points[0].y - points[1].y) < 1e-5 && abs(points[1].y - points
     [2].y) < 1e-5) return;
  glm::vec<3, int, glm::defaultp> t0(round(points[0].x), round(points
     [0].y), round(points[0].z)),
  t1(round(points[1].x), round(points[1].y), round(points[1].z)),
  t2(round(points[2].x), round(points[2].y), round(points[2].z));
  if (t0.y > t1.y) std::swap(t0, t1);
  if (t0.y > t2.y) std::swap(t0, t2);
  if (t1.y > t2.y) std::swap(t1, t2);
  int total_height = t2.y - t0.y;
  for (int i = 0; i < total_height; i++) {
    bool second_half = i > t1.y - t0.y || t1.y == t0.y;
    int segment_height = second_half ? t2.y - t1.y : t1.y - t0.y;
    float alpha = (float) i / total_height;
    float beta = (float) (i - (second_half ? t1.y - t0.y : 0)) /
    segment_height;
    glm::vec3 A = glm::vec3(t0) + glm::vec3(t2 - t0) * alpha;
    glm::vec3 B = (second_half ? glm::vec3(t1) + glm::vec3(t2 - t1) *
       beta :
    glm::vec3(t0) + glm::vec3(t1 - t0) * beta);
    if (A.x > B.x)
      std::swap(A, B);
    for (int j = round(A.x); j <= round(B.x); j++) {
      float phi = B.x == A.x ? 1. : (float) (j - A.x) / (float) (B.x -
         A.x);
      glm::vec3 P = glm::vec3(A) + glm::vec3(B - A) * phi;
      int idx = round(P.x + P.y * image.getSize().x);
      if (idx >= 0 && idx < z_buffer.size() && z_buffer[idx] < P.z) {
        z_buffer[idx] = P.z;
        image.draw(vector<sf::Vertex>(1, sf::Vertex(sf::Vector2f(P.x, P
           .y), color)).data(), 1, sf::Points);
      }
    }
  }
}
```

В листинге 3.2 приведена функция обновления распределения плотности газа.

Листинг 3.2 — Функция обновления распределения плотности дыма

```
void smoke::dens_step(vector<vector<vector<float>>> &x, vector<vector<
    vector<float>>> &x0, vector<vector<vector<float>>> &u_,
vector<vector<vector<float>>> &v_, vector<vector<vector<float>>> &w_,
    float d, float diff) {
  add_source(x, x0, d);
  x0.swap(x);
  diffuse(0, x, x0, diff, d);
  x0.swap(x);
  advect(0, x, x0, u_, v_, w_, d);
}
```

В листинге 3.3 представлена функция обновления поля скорости.

Листинг 3.3 — Функция обновления поля скорости

```
void smoke::vel_step(vector<vector<vector<float>>> &u_, vector<vector<
    vector<float>>> &v_, vector<vector<vector<float>>> &w_, vector<
    vector<vector<float>>> &v0, vector<vector<vector<float>>> &u0,
    vector<vector<vector<float>>> &w0, float visc, float d) {
  add_source(u_, u0, d);
  add_source(v_, v0, d);
  add_source(w_, w0, d);
  u0.swap(u_);
  diffuse(1, u_, u0, visc, d);
  v0.swap(v_);
  diffuse(2, v_, v0, visc, d);
  w0.swap(w_);
  diffuse(3, w_, w0, visc, d);
  project(u_, v_, w_, u0, v0);
  u0.swap(u_);
  v0.swap(v_);
  w0.swap(w_);
  advect(1, u_, u0, u0, v0, w0, d);
  advect(2, v_, v0, u0, v0, w0, d);
  advect(3, w_, w0, u0, v0, w0, d);
  project(u_, v_, w_, u0, v0);
}
```

В листинге 3.4 представлена функция вычисления адвекции.

Листинг 3.4 — Функция вычисления адвекции

```
void smoke::advect(int b, vector<vector<vector<float>>> &d, vector<
   vector<vector<float>>> &d0, vector<vector<vector<float>>> &u_,
   vector<vector<vector<float>>> &v_, vector<vector<vector<float>>> &w_
   , float dt_) {
  int i, j, i0, j0, k0, i1, j1, k1;
  float x, y, z, s0, t0, s1, t1, dt0, u1, u0;
  dt0 = dt_ * max(height, width);
  #pragma omp parallel for
  for (i = 1; i <= height; i++)
    for (j = 1; j <= height; j++)
      for (int k = 1; k <= width; ++k) {
        x = i - dt0 * w_[i][j][k];
        y = j - dt0 * u_[i][j][k];
        z = k - dt0 * v_[i][j][k];
        if (x < 1.5f) x = 1.5f;
        if (x > height - 0.5f) x = height - 0.5f;
        i0 = (int) x;
        i1 = i0 + 1;
        if (y < 1.5f) y = 1.5f;
        if (y > height - 0.5f) y = height - 0.5f;
        j0 = (int) y;
        j1 = j0 + 1;
        if (z < 1.5f) z = 1.5f;
        if (z > width - 0.5f) z = width - 0.5f;
        k0 = (int) z;
        k1 = k0 + 1;
        s1 = x - i0;
        s0 = 1 - s1;
        t1 = y - j0;
        t0 = 1 - t1;
        u1 = z - k0;
        u0 = 1 - u1;
        d[i][j][k] = s0 * (t0 * u0 * d0[i0][j0][k0] + t1 * u0 * d0[i0][
           j1][k0] + t0 * u1 * d0[i0][j0][k1] + t1 * u1 * d0[i0][j1][k1
           ]) + s1 * (t0 * u0 * d0[i1][j0][k0] + t1 * u0 * d0[i1][j1][
           k0] + t0 * u1 * d0[i1][j0][k1] + t1 * u1 * d0[i1][j1][k1]);
      }
  set_bnd(b, d);
}
```

В листинге 3.5 представлена функция решения СЛАУ алгоритмом на основе метода Гаусса—Зейделя.

Листинг 3.5 — Функция решения СЛАУ

```
void smoke::lin_solve(int b, vector<vector<vector<float>>> &x, vector<
   vector<vector<float>>> &x0, float a, float c) {
  int i, j, iter;
  for (iter = 0; iter < 5; iter++) {
    for (i = 1; i <= height; i++) {
      for (j = 1; j <= height; j++) {
        for (int k = 1; k <= width; ++k) {
          x[i][j][k] = (x0[i][j][k] + a * (x[i - 1][j][k] + x[i + 1][j
             ][k] + x[i][j - 1][k] + x[i][j + 1][k] + x[i][j][k - 1] +
             x[i][j][k + 1])) / c;


        }
      }
    }
    set_bnd(b, x);
  }
}
```

Вывод

В данном разделе были выбраны средства и проведена реализация программного обеспечения.

4 Исследовательский раздел

В данном разделе проведено исследование разработанной программы.

Исследования проводились на машине со следующими характеристиками:

— процессор Intel(R) Core(TM) i5-10210U, тактовая частота 1.60 ГГц;

— оперативная память: 16 ГБ;

— операционная система: Ubuntu 22.04.4 LTS.

4.1 Цель исследования

Размер ограничивающего столб пепла параллелепипеда и, как следствие, число вокселей напрямую влияет на производительность симуляции. Большинство методов расчета движения газа имеют кубическую сложность. Метод решения СЛАУ также дополнительно требует провести несколько итераций вычисления каждого неизвестного. Для увеличения производительности предлагается [5] использование параллельных вычислений.

Цель исследования — определить какую долю от общего времени выполнения алгоритма, занимают основные его части, для дальнейшего распараллеливания вычислений.

4.2 Исследование

Для исследования выделены основные части в алгоритме основанном на уравнениях Навье—Стокса:

— добавление источника дыма;

— решение СЛАУ для вычисления диффузии дыма;

— адвекция дыма;

— добавление источников скорости;

— решение СЛАУ для вычисления рассеивания скорости;

— самоадвекция скорости.

Для выполнения исследования, были проведены замеры времени основных этапов алгоритма для генерации 1000 кадров. Полученные результаты представлены в таблице 4.1.

Таблица 4.1 — Распределение времени выполнения этапов алгоритма генерации дыма

Этап	Доля от общего времени выполнения (%)	Среднее время выполнения (сек)
Добавление источников дыма	0.98	0.003
Решение СЛАУ для диффузии	70.26	0.214
Адвекция дыма	19.94	0.061
Добавление источников скорости	0.28	0.001
Решение СЛАУ для рассеивания скорости	22.49	0.068
Самоадвекция скорости	8.55	0.026
Сумма	100	0.372

На рисунке 4.1 приведены результаты исследования в виде круговой диаграммы.

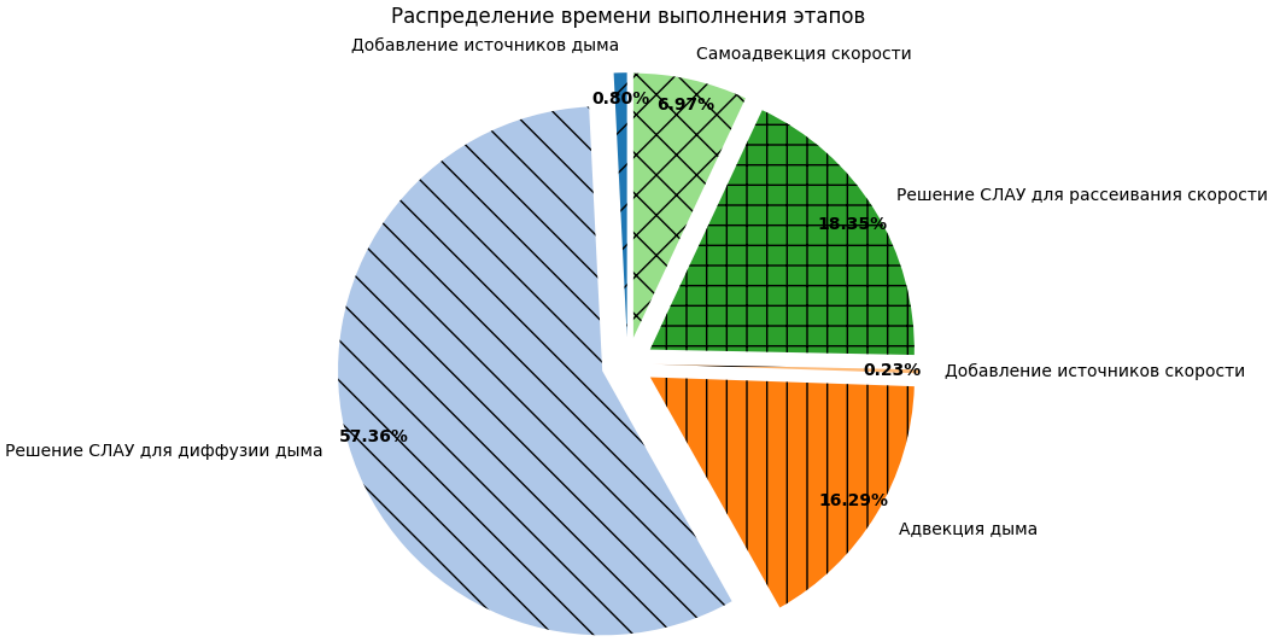

Рисунок 4.1 — Диаграмма распределения времени выполнения этапов алгоритма генерации дыма

Для проверки полученных результатов, были проведены аналогичные замеры для тех же этапов с использованием параллельных вычислений. Для обеспечения параллельности была использована библиотеки OpenMP [8]. Полученные результаты представлены в таблице 4.2 и на рисунке 4.2. Распределение времен выполнения при параллельном вычислении всех этапов осталось прежним, однако время выполнения наиболее долгих этапов снизилось. Как следствие, уменьшилось и общее время работы алгоритма.

Таблица 4.2 — Распределение времени выполнения этапов алгоритма генерации дыма при использовании параллельных вычислений

Этап	Доля от общего времени выполнения (%)	Среднее время выполнения (сек)
Добавление источников дыма	1.99	0.004
Решение СЛАУ для диффузии	57.14	0.106
Адвекция дыма	11.68	0.022
Добавление источников скорости	0.56	0.001
Решение СЛАУ для рассеивания скорости	22.49	0.041
Самоадвекция скорости	6.14	0.011
Сумма	100	0.185

Рисунок 4.2 — Диаграмма распределения времени выполнения этапов алгоритма генерации дыма с использованием параллельных вычислений

Вывод

В данной части была описана цель исследования, технические характеристики машины на которой выполнялись замеры, приведены результаты исследования.

В результате исследования было определено, что наиболее трудоемкими частями алгоритма являются решение СЛАУ и вычисление адвекции. Применение к ним параллельных вычислений дало ускорение в приблизительно 2 раза, что значительно сказывается на частоте кадров визуализации.

ЗАКЛЮЧЕНИЕ

В ходе работы были решены задачи:

— проведен анализ алгоритмов удаления невидимых линий и поверхностей, построения освещения, визуализации и выбраны наиболее подходящие для решения задачи;

— спроектировано программное обеспечение;

— выбраны средства реализации программного обеспечения и проведена его разработка;

— проведено исследование характеристик разработанного программного обеспечения.

Все задачи были выполнены, цель работы была достигнута.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Starodubtsev I.A., Vasev P.P. Modeling and Visualization of Lava Flows // 2022.

2. Д., Роджерс. Алгоритмические основы машинной графики. / Роджерс Д. — М.Мир, 1989. — С. 512.

3. Foster N., Metaxas D. Realistic Animation of Liquids // Graphical Models and Image Processing. — 1996. — №5. — С. 471-483.

4. R. Fedkiw, J. Stam, and H. W. Jensen. Visual Simulation of Smoke. In SIGGRAPH 2001 Conference Proceedings, Annual Conference Series , pages 15-22, August 2001. графики. – С.Пб: БХВ–Петербург, 2003. – 560 с.

5. T. Vik, A.C. Elster, and T. Hallgren. Real-time visualization of smoke through parallelizations. In G.R. Joubert, W.E. Nagel, F.J. Peters, and W.V. Walter, editors, Parallel Computing, volume 13 of Advances in Parallel Computing, pages 371–378. North-Holland, 2004.

6. Lars Kjelldahl. Real-time large scale fluids for games. 2008.

7. Russell M. Cummings, William H. Mason, Scott A. Morton, David R. McDaniel. Birmingham Applied Computational Aerodynamics. — Cambridge University Press, 2015. — 888 с.

8. OpenMP documentation [Электронный ресурс] // URL: `https://www.openmp.org/` (дата обращения 20.02.2025).

9. Qt documentation [Электронный ресурс] // URL: `https://doc.qt.io/qt-6/reference-overview.html` (дата обращения 12.10.2024).

10. SFML documentation [Электронный ресурс] // URL: `https://www.sfml-dev.org/documentation/2.6.2/` (дата обращения 15.11.2024)

Приложение А

Презентация к курсовой работе на 15 слайдах.