СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Компьютерная графика—совокупность методов и средств преобразования информации в графическую форму и из графической формы с помощью ЭВМ.

Компьютерные техники, создающие анимации преобразования формы искусственного объекта, называются морфингом или метаморфозами. Методы морфинга сегодня широко используются в компьютерной графике для моделирования преобразование между двумя совершенно разными объектами или для создания новых форм с помощью сочетание других существующих форм. Они имеют множество применений, начиная от специальных эффектов в киноиндустрии и других видах изобразительного искусства до медицинской визуализации и научных целей.

Цель работы — разработка программного обеспечения для визуализации процесса морфинга двух фруктов, представленных низкополигональными моделями без отверстий. Считать, что источник света — точечный, и его положение совпадает с положением наблюдателя. Для достижения поставленной цели необходимо решить следующие задачи:

— перечислить основные принципы морфинга трехмерных объектов;

— описать использованные техники морфинга трехмерных объектов;

— проанализировать и выбрать алгоритмы решения основных задач компьютерной графики: удаления невидимых линий и поверхностей, учёта освещения;

— спроектировать программное обеспечение для визуализации процесса морфинга двух фруктов;

— выбрать средства реализации, описать формат входных и выходных данных, реализовать спроектированное программное обеспечение;

— исследовать вклад времени выполнения этапов морфинга в общее время процесса.

1 Аналитическая часть

1.1 Определение и основные принципы морфинга

Термин «морфинг» (от англ. *morphing*) происходит от слова «метаморфоза» и описывает процесс преобразования одного объекта в другой. В контексте компьютерной графики под морфингом трехмерных объектов понимают построение последовательности кадров, которая отображает плавный и непрерывный переход от исходной (начальной) модели к целевой (конечной).

Пример последовательности кадров, получаемой в результате морфинга, представлен на рисунке 1.1.

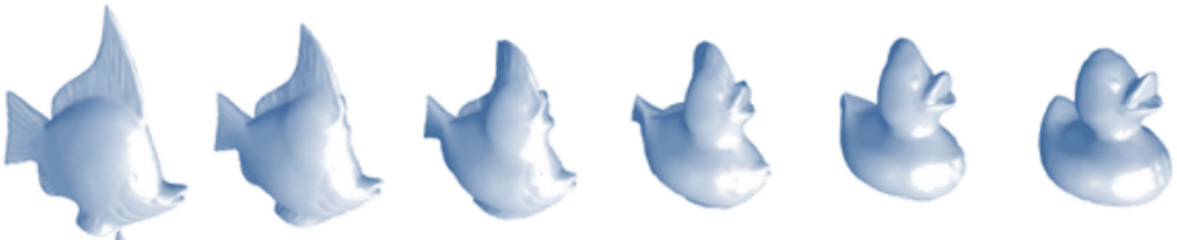

Рисунок 1.1 — Пример последовательности кадров морфинга

Основная задача морфинга заключается в вычислении такого преобразования, которое обеспечивает визуально правдоподобный и плавный переход между формами. Для достижения высокого качества результата необходимо соблюдение следующих ключевых принципов [1]:

— *сохранение топологии* – в процессе трансформации не должны появляться или исчезать отверстия, а сама модель не должна разрываться на части;

— *гладкость* – трансформация должна выполняться плавно, без резких скачков и визуальных артефактов;

— *монотонность* – объемы или площади частей модели должны изменяться монотонно (только увеличиваться или только уменьшаться) на протяжении всего перехода.

1.2 Модель описания объекта

Согласно техническому заданию, фрукты, представлены низкополигональными трехмерных объектами без отверстий. Данное требование однозначно

определяет использование полигональной модели в качестве способа представления трехмерных объектов.

1.2.1 Полигональная модель

В полигональной модели информация об объекте сосотоит из следущих компонентов [2]:

— вершина — точка ((x, y, z) в декартовой системе координат);

— отрезок прямой — задается двумя вершинами;

— полилиня — задается несколькими отрезками прямой;

— полигон — описывает плоскую грань объемного объекта в виде замкнутой линии;

Несколько граней (полигонов) составляют объемный объект в виде полигональной поверхности, также назваемой «полигональной сеткой».

Обычно в качестве граней используются треугольники, поскольку это упрощает процесс отрисовки [1].

1.3 Основные этапы морфинга

1.3.1 Установление соответствия между объектами

3D-сетки часто различаются по топологии (число вершин/граней и связность), поэтому прямое сопоставление поверхностей затруднено. Соответствие получают косвенно через *параметризацию* — биективное отображение поверхности сетки в общую параметрическую область D, обычно единичный диск для незамкнутых сеток (*плоская парамеризация*) или единичная сфера для замкнутых (*сферическая параметризация*) [1].

Сферическая параметризация применяется к замкнутым поверхностям нулевого рода [1] — т. е. не имеющим отверстий, каковыми являются фрукты в рамках данной работы.

1.3.2 Параметризация

Выделяют 3 основных подхода к параметризации сеток: *плоская*, *сферическая*, *разбиение на участи с плоской параметризацией* [1, 3].

Плоская параметризация

Плоская праметризация применима только к незамкнутым сеткам [1, 3], поэтому она непригодна для морфинга замкнутых объектов, таких как фрукты.

Разбиение на участки с плоской параметризацией

Если исходный и целевой объекты имеют существенно различающиеся формы или представляют собой поверхности разного рода, т. е. имеют разное число отверстий, применяют предварительное разбиение на плоские участки [1]. Пользователь вручную разбивает исходную и целевую сетки на участки и задаёт их соответствие; затем для каждого участка выполняется плоская параметризация. Данный метод требует значительного участия пользователя, от которого зависит качество результата, поэтому в настоящей работе он не рассматривается.

Сферическая параметризация

Для сферической параметризации произвольных поверхностей нулевого рода, применяется метод релаксации, основаный на итеративном уточнении положений вершин [3]. Процесс релаксации сетки представлен на рисунке 1.2.

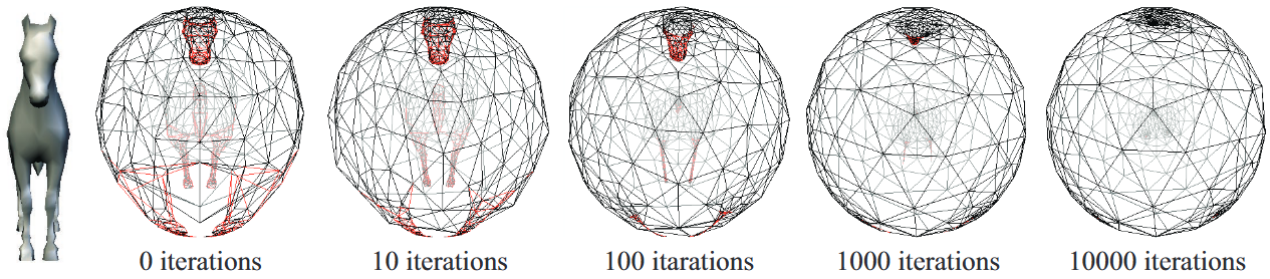

Рисунок 1.2 — Процесс релаксации сетки, красными отмечены грани с непрвильной ориентацией [3]

В качестве начального состояния строят грубую проекцию сетки на единичную сферу: выбирают любую внутреннюю точку модели в качестве центра сферы и проецируют все вершины на её поверхность (нормализуют радиус-векторы). Полученная начальная конфигурация обычно содержит значительные искажения и грани с неправильной ориентацией. Процесс релаксации продолжают до тех пор, пока все грани не приобретут корректную ориентацию — внешняя

сторона каждой грани должна быть обращённой наружу сферы [3].

На каждом раунде релаксации вершины сдвигаются к центру масс своих соседей:

$$v_i^{k+1} = \frac{\sum_{j \in N(i)} v_j^k}{\| \sum_{j \in N(i)} v_j^k \|}, \tag{1.1}$$

где

— v_i^{k+1} — положение i-ой вершины после k-го раунда релаксации;

— v_i^k — положение i-ой вершины на момент k-го раунда релаксации;

— $N(i)$ — множество вершин, смежных с i-ой.

Для предотвращения коллапса всех вершин в одну точку после каждой итерации выполняется ре-центрирование всей сетки относительно начала координат [3]:

$$v_i' = v_i - \frac{\sum_{j=0}^{n} v_j}{n}, \tag{1.2}$$

где v_i' — новое положение i-ой вершины; n — количество вершин.

1.3.3 Создание общего представления

После того как установлено соответствие, необходимо создать единую структуру, которая будет использоваться для всех промежуточных форм. Обычно для этого сроят *суперсетку* — сетку, содержащую вершины исходной и целевой сетки, а также их точки пересечения [1, 3] (пример на рисунке 1.3).

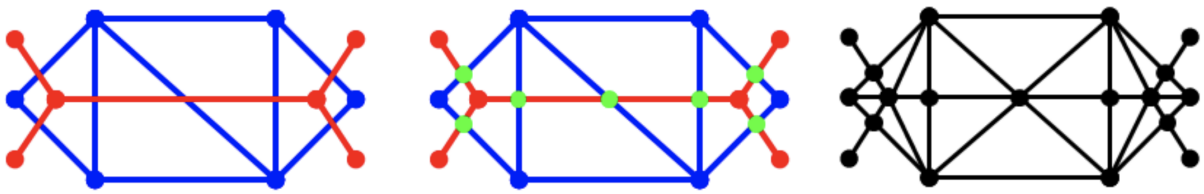

Рисунок 1.3 — Создание суперсетки [1]

На параметрической сфере сетки накладываются друг на друга; в местах пересечения ребер создаются новые вершины, что формирует объединенную сетку, которая может принимать форму, как исходной так и целевой модели; поскольку полученная сетка обычно не является треугольной, выполняют её

треангуляцию. Для этого используется треангуляция Делоне, поскольку она минимизирует количество узких треугольников, что улучшает качество сетки [1, 3].

1.3.4 Построение преобразования и интерполяция атрибутов

Завершающим этапом морфинга является вычисление траекторий движения вершин общего представления (суперсетки) из их начальных положений в конечные. Для построения полноценного преобразования необходимо определить не только положения вершин, но и атрибуты граней, такие как нормали и оптические свойства материала, для исходного и целевого состояний. Процесс генерации промежуточных кадров сводится к интерполяции всех этих атрибутов [1, 3].

Определение положений вершин

Положение каждой вершины суперсетки на исходной и целевой моделях определяется с помощью барицентрических координат в общем параметрическом пространстве [1, 3]. Алгоритм состоит из следующих шагов:

1) Для каждой вершины суперсетки определяется треугольник исходной (или целевой) сетки, в который она попадает в параметрическом пространстве.
2) Вычисление барицентрических координат данной вершины относительно вершин найденного треугольника.
3) Полученные барицентрические координаты применяются для интерполяции мировых координат вершин соответствующего треугольника.

После определения начального (P_0) и конечного (P_1) положений для каждой вершины, ее траектория в процессе морфинга генерируется посредством линейной интерполяции [1, 3]. Положение вершины в момент времени $t \in [0, 1]$ вычисляется по формуле (1.3):

$$P(t) = (1 - t) \cdot P_0 + t \cdot P_1 \tag{1.3}$$

Интерполяция нормалей

Аналогичный подход, основанный на параметрическом пространстве, используется для определения и интерполяции нормалей к граням суперсетки.

1) Для каждого треугольника суперсетки вычисляется его центр масс в параметрическом пространстве.
2) Для этой точки находится содержащий ее треугольник на параметризованной исходной (или целевой) сетке.
3) Нормаль, ассоциированная с найденным треугольником на 3D-модели, присваивается соответствующему треугольнику суперсетки в качестве его начальной ($\mathbf{n_0}$) (или конечной ($\mathbf{n_1}$))нормали.

В процессе морфинга нормаль грани в момент времени t также вычисляется путем линейной интерполяции:

$$\mathbf{n(t)} = (1 - t) \cdot \mathbf{n_0} + t \cdot \mathbf{n_1} \tag{1.4}$$

Интерполяция свойств материала

Оптические свойства поверхности, такие как цвет или коэффициенты отражения, также интерполируются линейно. Если M_0 и M_1 представляют собой наборы свойств материала для исходной и целевой моделей, то промежуточное свойство $M(t)$ вычисляется как:

$$M(t) = (1 - t) \cdot M_0 + t \cdot M_1 \tag{1.5}$$

1.4 Анализ алгоритмов удаления невидимых линий и поверхностей

Алгоритмы удаления невидимых линий и поверхностей служат для удаления ребер, поверхностей или объемов, которые видимы или невидимы для наблюдателя, находящегося в заданной точке пространства [4].

Были рассмотрены следующие алгоритмы: *алгоритм Робертса*, *алгоритм, использующий z-буфер*, *алгоритм трассировки лучей*.

1.4.1 Алгоритм Робертса

Данный алгоритм применим только к выпуклым телам. Если обрабатываемое тало невыпуклое — его необходимо предварительно разбить на выпуклые [4].

Алгоритм состоит из следующих этапов [4]:

1) Удаление граней, экранируемых самим телом.
2) Удаление граней, экранируемых другими телами.
3) Удаление линий пересечения тел, экранируемых самими телами.

Ассимптотическая оценка трудоемкости: $O(N^2)$, где N — количество граней.

1.4.2 Алгоритм, использующий z-буфер

Идея z-буфера является обобщением идеи буфера кадра. Буфер кадра используется для запоминания аттрибутов (интенсивности) каждого пикселя в пространстве изображения. z-буфер — это отдельны буфер глубины, используемый для запоминания координаты z каждого видимого пикселя в пространстве изображения [4].

Этапы работы алгоритма [4]:

1) Заполнить буфер кадра фоновым значением.
2) Заполнить z-буфер минимальным значением глубины.
3) Выполнить переход в пространство изображения.
4) Для каждого пикселя (x, y), пренадлежащего телу вычислить его глубину $z(x, y)$.
5) Если глубина $z(x, y) > z$-*буфер*(x, y), то записать атрибут текущего тела в *буфер-кадра*(x, y), записать глубину $z(x, y)$ в z-*буфер*(x, y).

Асимптотическая оценка трудоекости: $O(N)$, где N — количество граней.

1.4.3 Алгоритм художника

Идея алгоритма состоит в том, чтобы подобно художнику отрисовывать объекты по мере их приближения к наблюдателю.

Основные этапы алгорима [4]:

1) Отсортировать грани по минимальному или максимальному значению

глубины.

2) Отрисовать грани в отсортированном порядке.

Простая сортировка не всегда дает корректный список приоритетов, тогда приходится использовать дополнительные методы разрешения конфликтов [4].

Алгоитм не справляется со случаями циклического перекрытия и пересечения многоугольников.

Ассиптотическая оценка трудоемкости: $O(N)$, где N — количество граней, однако стоит дополнительно учитывать трудоемкость предварительной сортировки. Таким образом итоговая оценка будет $O(N \log N)$.

Алгоритм трассировки лучей

Идея метода заключается в том, что наблюдатель видит любой объект посредством испускаемого неким источником света, который падает на этот объект и некоторомы путем доходит до наблюдателя. Однако обычно используют алгоритм обратной трассировки лучей, в котором лучи пускаются от наблюдателя [4].

Полагая, что объекты сцены уже находятся в простратсве изображения, выполняются следующие этапы для каждого пикселя [4]:

1) Составить уравнение отслеживаемого луча.
2) Найти пересечение луча со всеми гранями.
3) Среди всех пересечений найти ближайшее к наблюдатею и использовать аттрибуты объекта, которому соответвствует пересечение для определения цвета пикселя. Если пересечений нет — закрасить пиксель цветом фона.

Ассимптотическая оценка трудоемкости: $O(WHN)$, где W — ширина экрана в пикселях, H — высота экрана в пикселях, N — количество граней.

1.4.4 Сравнение алгоритмов

В таблице 1.1 представленные результаты сравнения алгоритмов и использованы следующией обозначения:

— Р — алгоритм Робертса;
— ZБ — алгоритм, использующий z-буфер;
— Х — алгоритм Художника;

— ОТЛ — алгоритм обратной трассировки лучей.

Таблица 1.1 — Сравнение алгоритмов удаления невидимых линий и поверхностей

	Р	**ZБ**	**Х**	**ОТЛ**
Совместимость с любыми телами	-	+	+	+
Возможность использования без сортировки	+	+	-	+
Устойчивость к циклическому перекрытию	+	+	-	+
Асиптотическая оценка трудоемкости	$O(N^2)$	$O(N)$	$O(N \log N)$	$O(WHN)$

Среди всех расмотренных алгоритмов, алгоритм, использующий z-буфер подходит больше других, т. к. он имеет лучшую асимптотическую оценку трудоемкости и применим к любым телам, что необходимо при решении задачи визуализации морфинга.

1.5 Модель освещения

В компьютерной графике наиболее распространенными являются две модели освещения: *локальная* и *глобальная* [4].

Локальная модель учитывает только свет, падающий от источника (источников), и ориентацию поверхности [4].

Глобальная модель освещения учитывает также свет, отраженный от других объектов сцены или пропущенный через них [4].

Поскольку на сцене будет находиться только один объект, то будет использована локальная модель освещения.

Интенсивность I в точке P в локальной модели вычисляется по формуле (1.6) [4].

$$I = k_a I_a + \frac{I_l}{d + K}[k_d(\hat{\mathbf{n}} \cdot \hat{\mathbf{L}}) + k_s(\hat{\mathbf{R}} \cdot \hat{\mathbf{S}})^{\alpha}], \tag{1.6}$$

где

— k_a — коэффициент диффузного отражения фонового освещения;

— I_a — интенсивность фонового освещения;

— I_l — интенсивность источника света;

— d — расстояние от источника света до точки P;

— K — добавка уменьшения интенсивности света с расстоянием (выбирается из эстетических предпочтений);

— k_d — коэффициент диффузного отражения поверхности;

— $\mathbf{n}$ — вектор нормали к поверхности в точке P;

— $\mathbf{L}$ — вектор, обратный вектору падения луча;

— k_s — коэффициент зеркального отражения поверхности;

— $\mathbf{R}$ — вектор, отраженного луча;

— $\mathbf{S}$ — вектор, направленный на наблюдателя из точки P;

— α — степень, аппроксимирующая пространственное распределение зеркально отраженного света.

1.6 Анализ алгоритмов закраски

Основными алгоритмами закраски в комьютерной графике являются: *однотонная закраска*, *закраска методом Гуро*, *закраска методом Фонга* [4].

1.6.1 Однотонная закраска

При однотонной закраске для каждой грани (многоугольника) полигональной поверхности вычисляется один уровень интенсивности, с которым закрашивается вся грань. В результате такой закраски изображенный состоит из отдельных иногоугольников и объект выглядит, как многогранник [4].

1.6.2 Закраска методом Гуро

Этот метод предназначен для создания иллюзии гладкой криволинейной поверхности, описанной в виде многогранников или полигональной сетки с плоскиими гранями [2, 4, 5]

Метод Гуро основан на интерполяции интенсивности каждого пикселя при закраске. Закрашивание граней по методу гуро осуществляется в четыре этапа [2]:

1) Определение нормали к каждой грани.
2) Определение нормалей в вершинах путем усреднения нормалей прилежащих граней.
3) На основе нормалей в вершинах вычисляются значения интенсивностей в вершинах согласно выбранной модели освещения.
4) Закрашиваются полигоны граней цветом, соответствующим интерполяции значений интенсивности в вершинах.

1.6.3 Закраска методом Фонга

Аналогичен методу Гуро, но при использовании метода Фонга для определения цвета в каждой точке интерполируются не интенсивности отраженного света, а векторы нормалей. При этом значение интенсивность вычисляется в каждом внутреннем пикселе грани [2, 4].

1.6.4 Сравнение алгоритмов закраски

Визуальное сравнение алгоитмов представленно на рисунке 1.4.

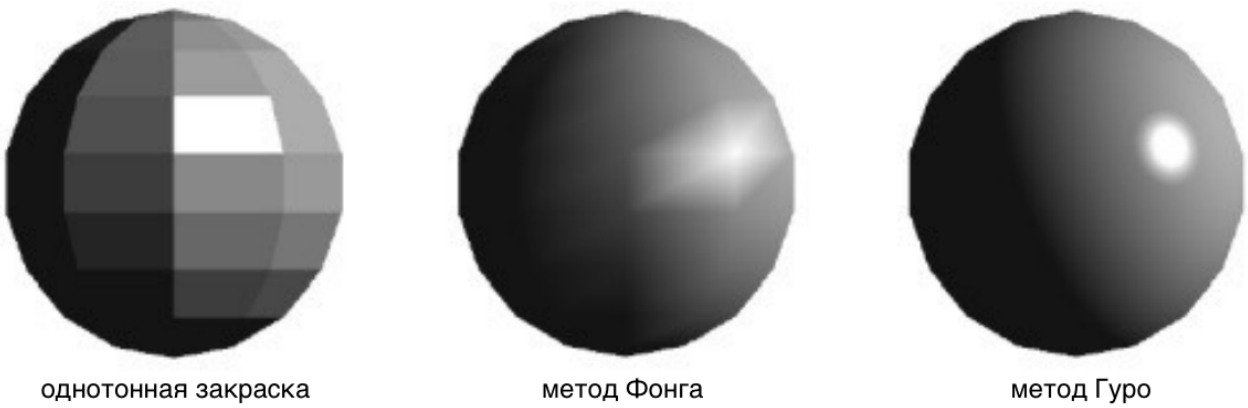

Рисунок 1.4 — Визуальное сравнение методов закрски

С целью лучшего отслеживания изменений полигональной сетки объектов (фруктов) применяется однотонная закраска. Этот метод делает видимой каждую грань, наглядно демонстрируя трансформацию модели. В результате мофринга часто возникает множество полигонов, составляющих одну грань, из-за чего при однотонной закраске возникают световые артефакты, как на рисунке 1.5, а.

Для устранения этой проблемы была использована модифицированная версия закраски Гуро: значение интенсивности вершин в пределах одной грани вычисляется с помощью нормали к этой грани, а не нормалей в вершинах. Это обеспечивает равномерное распределение интенсивности по грани и предотвращает сглаживание рёбер, что продемонстрированно на рисунке 1.5, б.

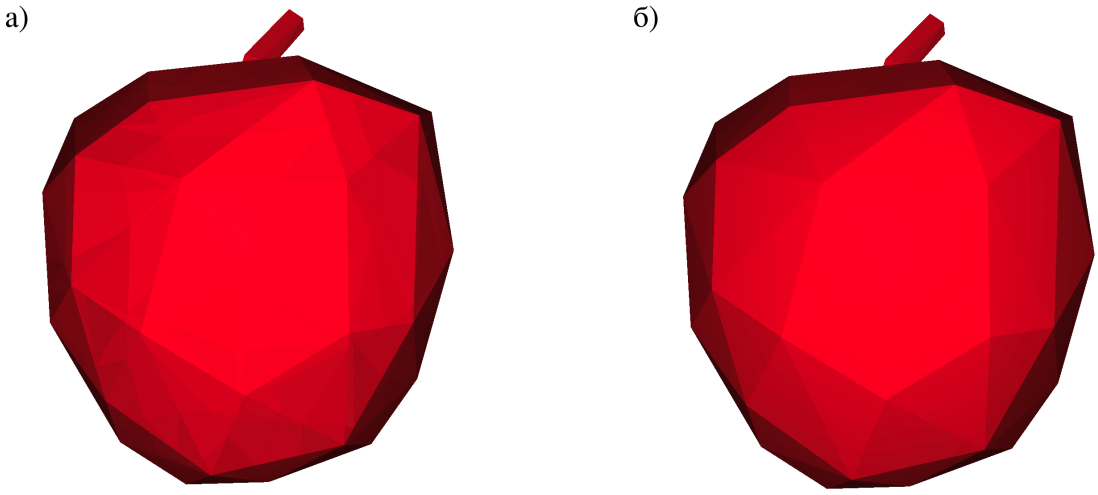

Рисунок 1.5 — Результат морфинга с использованием различных алгоритмов закраски: а) однотонная закраска; б) закраска модифицированным методом Гуро

Вывод

В аналитической части были рассмотрены основные принципы морфинга трехмерных объектов, проанализированы алгоритмы решения основных задач морфинга и компьютерной графики для объектов, представленных полигональной сеткой. По результатам проведенного анализа для решения задач морфинга был сформирован многоэтапный процесс, включающий сферическую параметризацию с последующей релаксацией сетки, создание суперсетки с триангуля-

цией Делоне и последующую линейную интерполяцию атрибутов. Для решения основных задач компьютерной графики были выбраны: полигональное представление модели, алгоритм, использующий z-буфер для удаления невидимых линий и поверхностей, простая модель освещения, модифицированная закраска Гуро.

2 Конструкторская часть

2.1 Требования к программному обеспечению

Программа должна предоставлять пользователю графический интерфейс, позволяющий:

— загрузить исходный и целевой объект из *.obj* файла;

— изменить цвет и оптические свойства поверхности загруженных объектов;

— просматривать по отдельности каждый объект и результат морфинга посредством поворота и масштабирования;

— изменять масштабирование и поворот исходного и целевого объектов морфинга;

— выбирать стадию морфинга объектов.

Программа должна корректно реагировать на любые действия пользователя.

2.2 Используемы типы и структуры данных

1) Сцена состоит из:
   — объекта;
   — камеры;
   — источника света.
2) Объект состоит из:
   — массива точек;
   — массива треугольников (индексы вершин);
   — массива единичных внешних нормалей к треугольникам;
   — материала;
   — матрицы преобразований.
3) Результат морфинга, состоит из:
   — массива точек;
   — массива функций, интерполирующих вершины в момент t;
   — массива треугольников (индексы вершин);
   — массива единичных внешних нормалей к треугольникам;
   — массива функций, интерполирующих внешние нормали в момент

t;

— материала;

— функции интерполирующей материал в момент t;

— матрицы преобразований.

4) камера состоит из:

— положения;

— точки, на которую направлен взгляд;

— угол обзора в радианах;

— отношение сторон;

— уаление ближней плоскости пирамиды видимости;

— удаление дальней плоскости пирамды видимости.

5) источник света состоит из:

— положения;

— интесивности;

6) материал состоит из:

— цвета в формате *RGB*;

— коэффициента диффузного отражения;

— коэффициента зеркального отражения;

— степени, аппроксимирующей пространственное распределение зеркально отраженного света.

Для построения суперсетки используется структура данных DCEL (doubly connected edge list) [3]. Визуальное представление структуры данных приведено на рисунке 2.1.

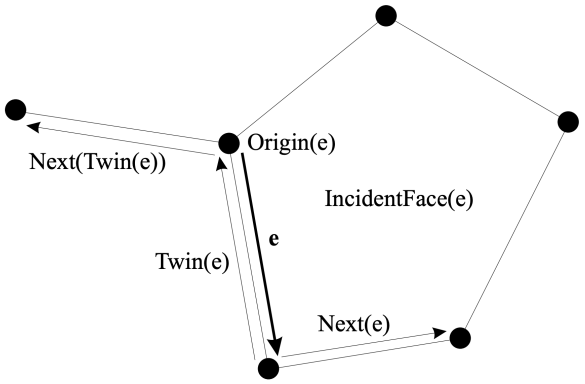

Рисунок 2.1 — DCEL (doubly connected edge list) [3]

DCEL состоит из:

— массива вершин;

— массива полуребер;

— массива граней;

Полуребро состоит из:

— индекса вершины-начала;

— индекса полуребра-двойника;

— индекса смежной грани;

— индекса следующего полуребра;

Грань состоит из индекса полуребра, связанного с ней.

2.3 Описание алгоритмов

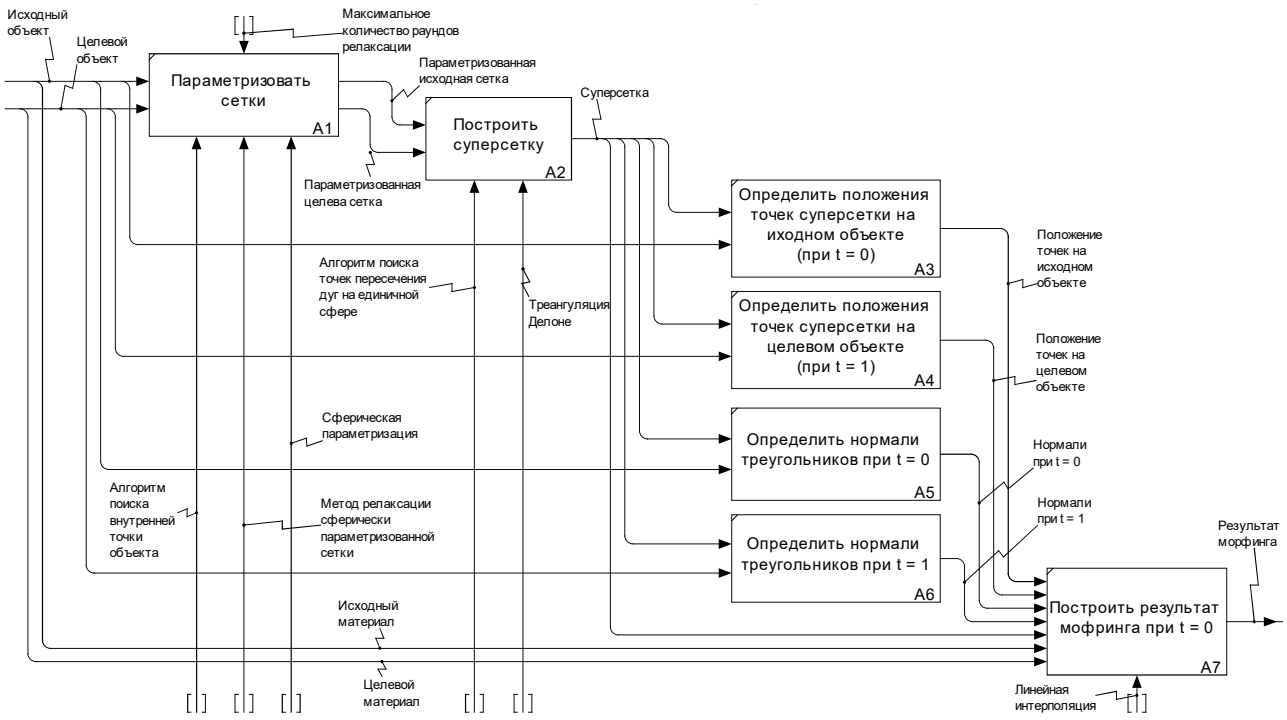

Рисунок 2.2 — Функциональная диаграмма первого уровня, описывающая процесс построения морфинга

Схемы основных алгоритмов, используемых в программном обеспечены представлены на рисунках 2.3–2.8.

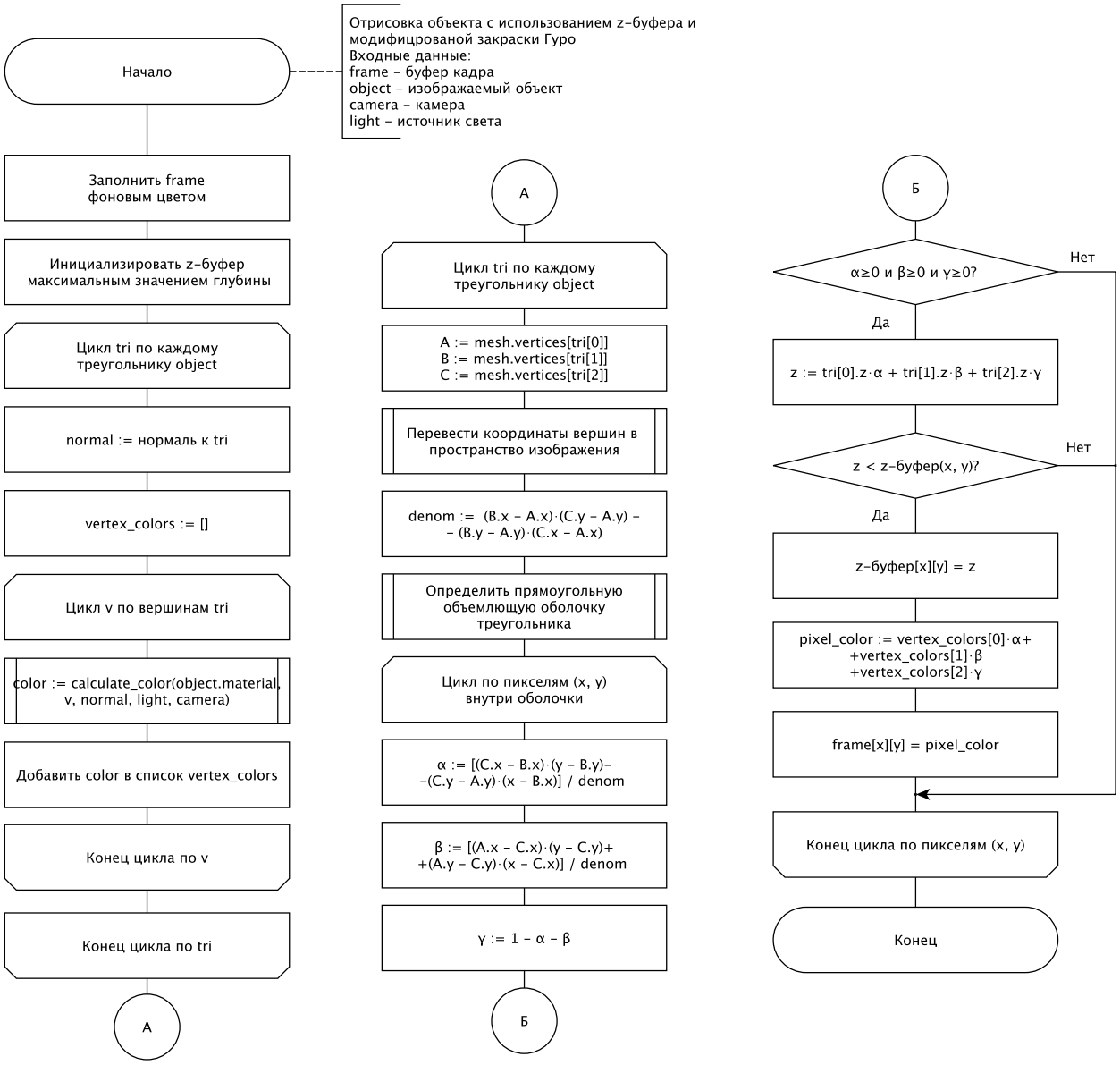

Рисунок 2.3 — Схема алгоритма отрисовки объекта с использованием *z*-буфера и модифицированной закраски Гуро

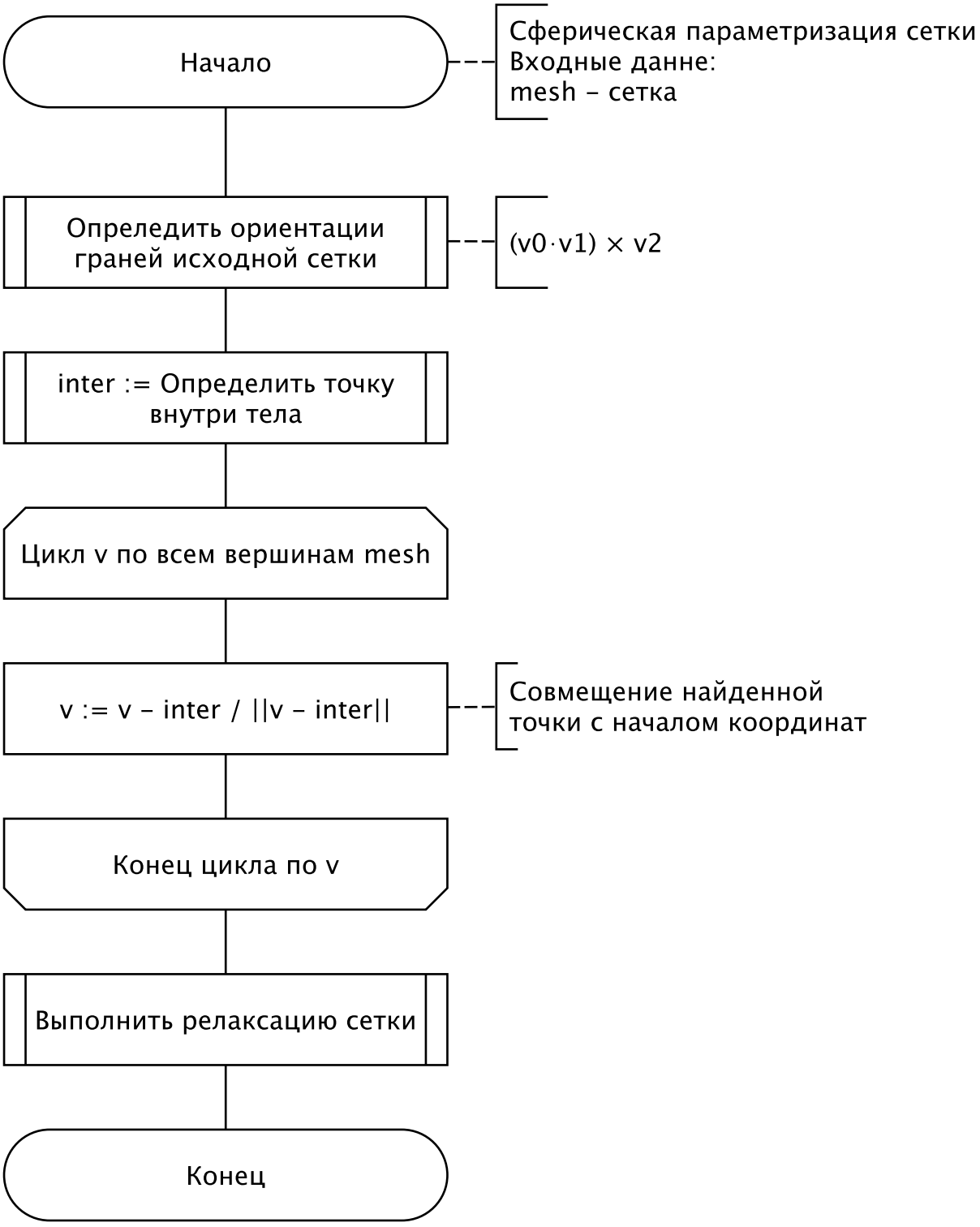

Рисунок 2.4 — Схема алгоритма сферической параметризации сетки

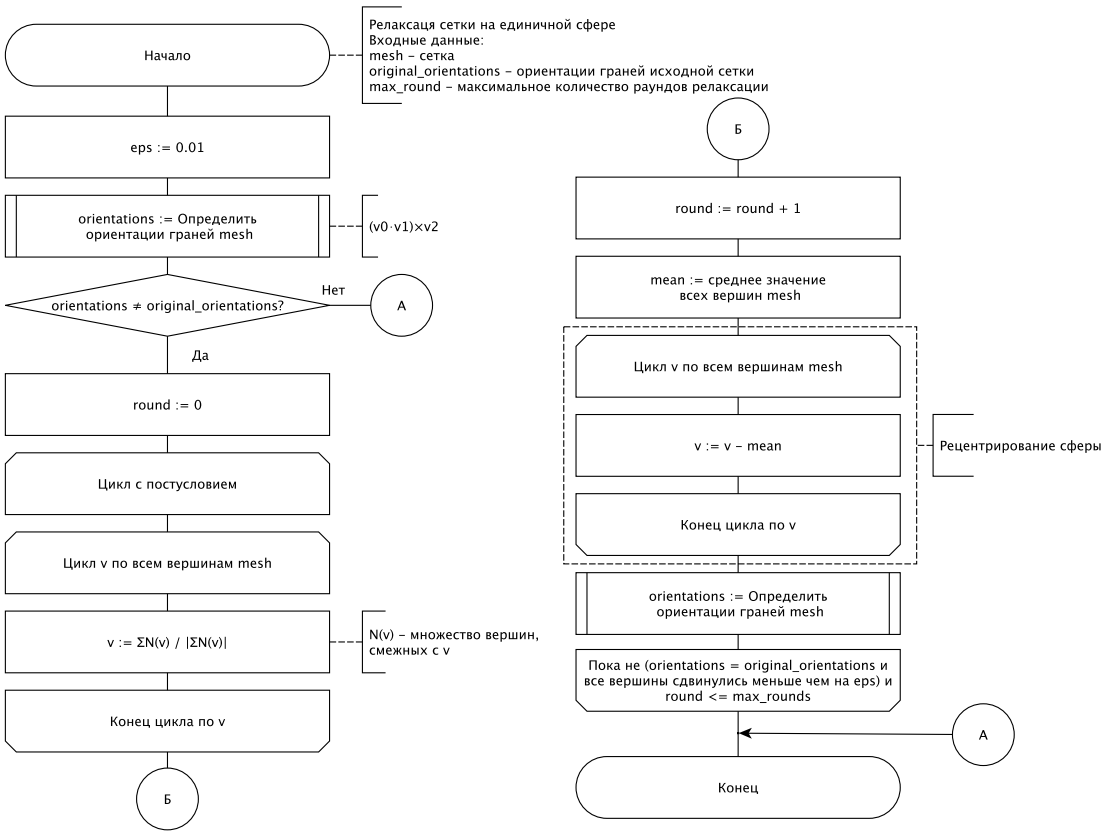

Рисунок 2.5 — Схема алгоритма релаксации сетки

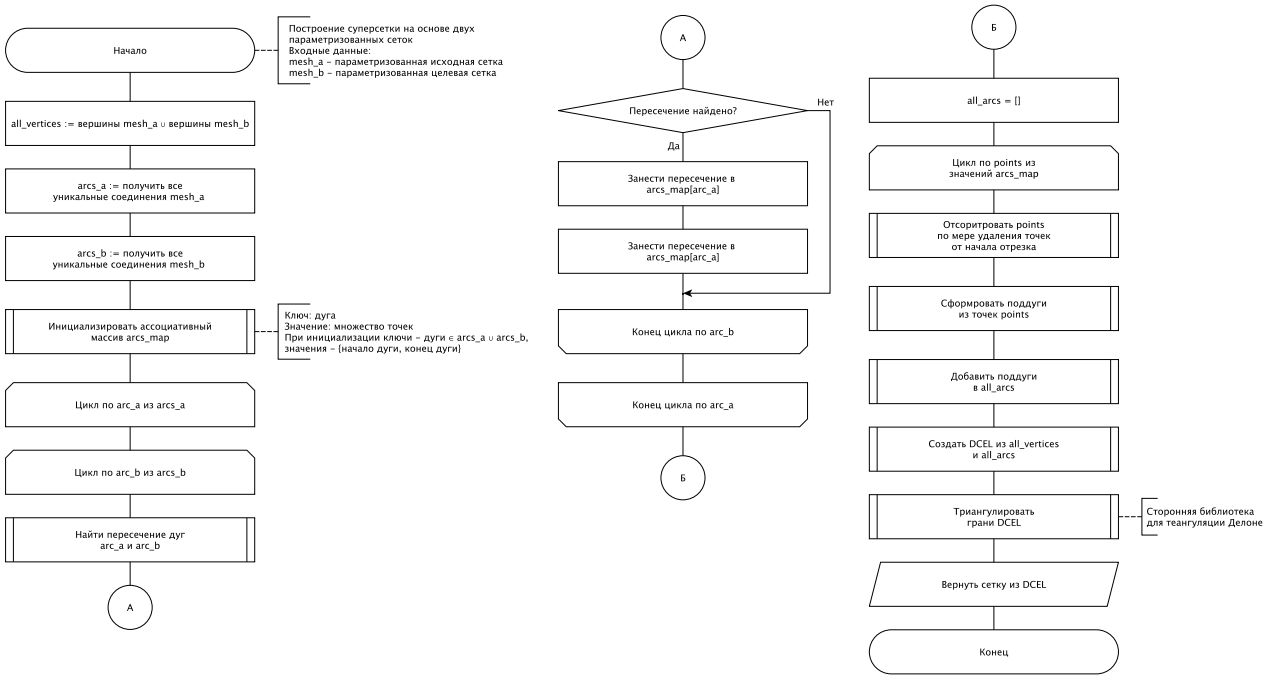

Рисунок 2.6 — Схема алгоритма построения суперсетки на единичной сфере

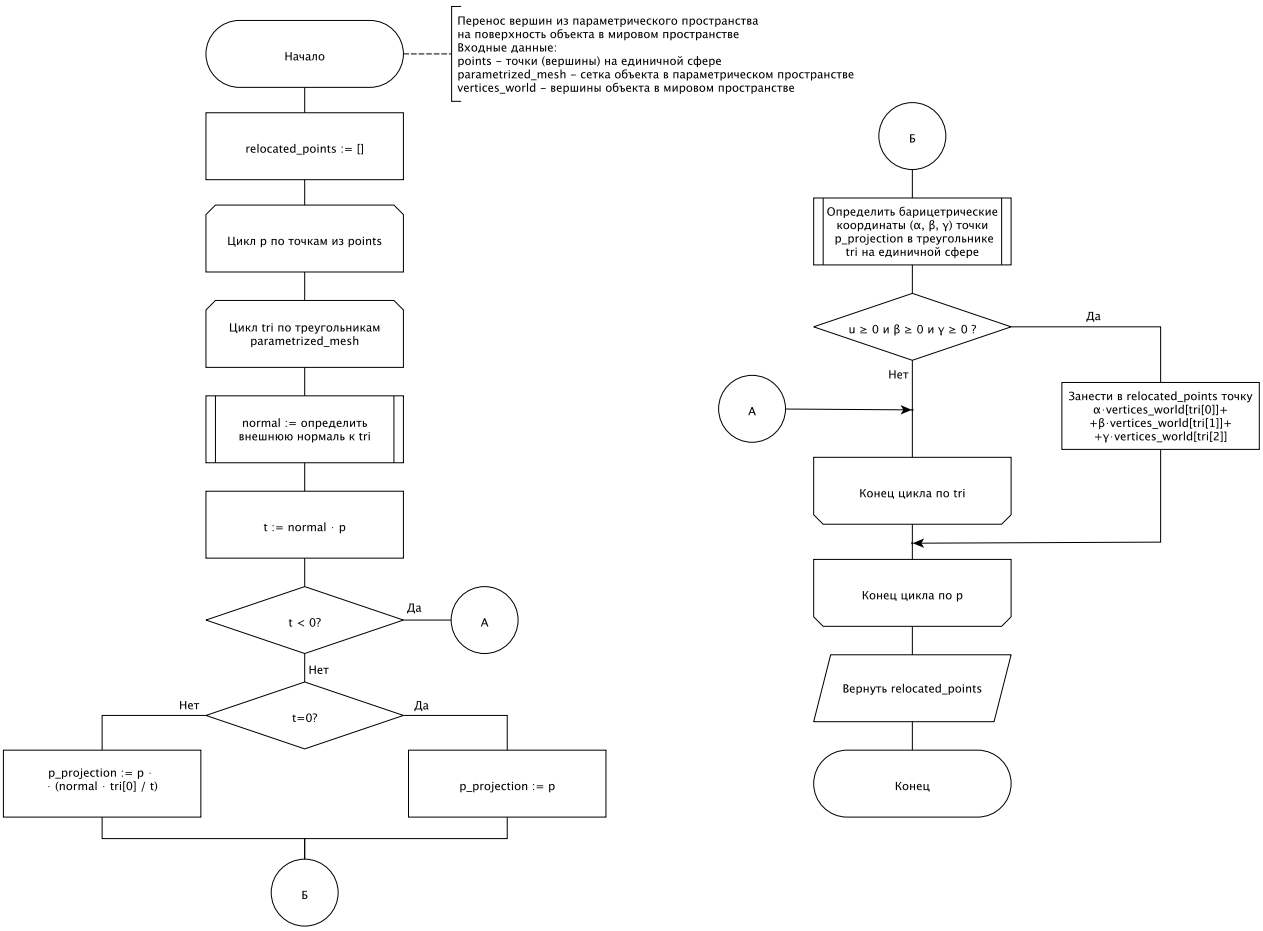

Рисунок 2.7 — Схема алгоритма вычисления положения вершин суперсетки на поверхности объекта по их параметрическим координатам

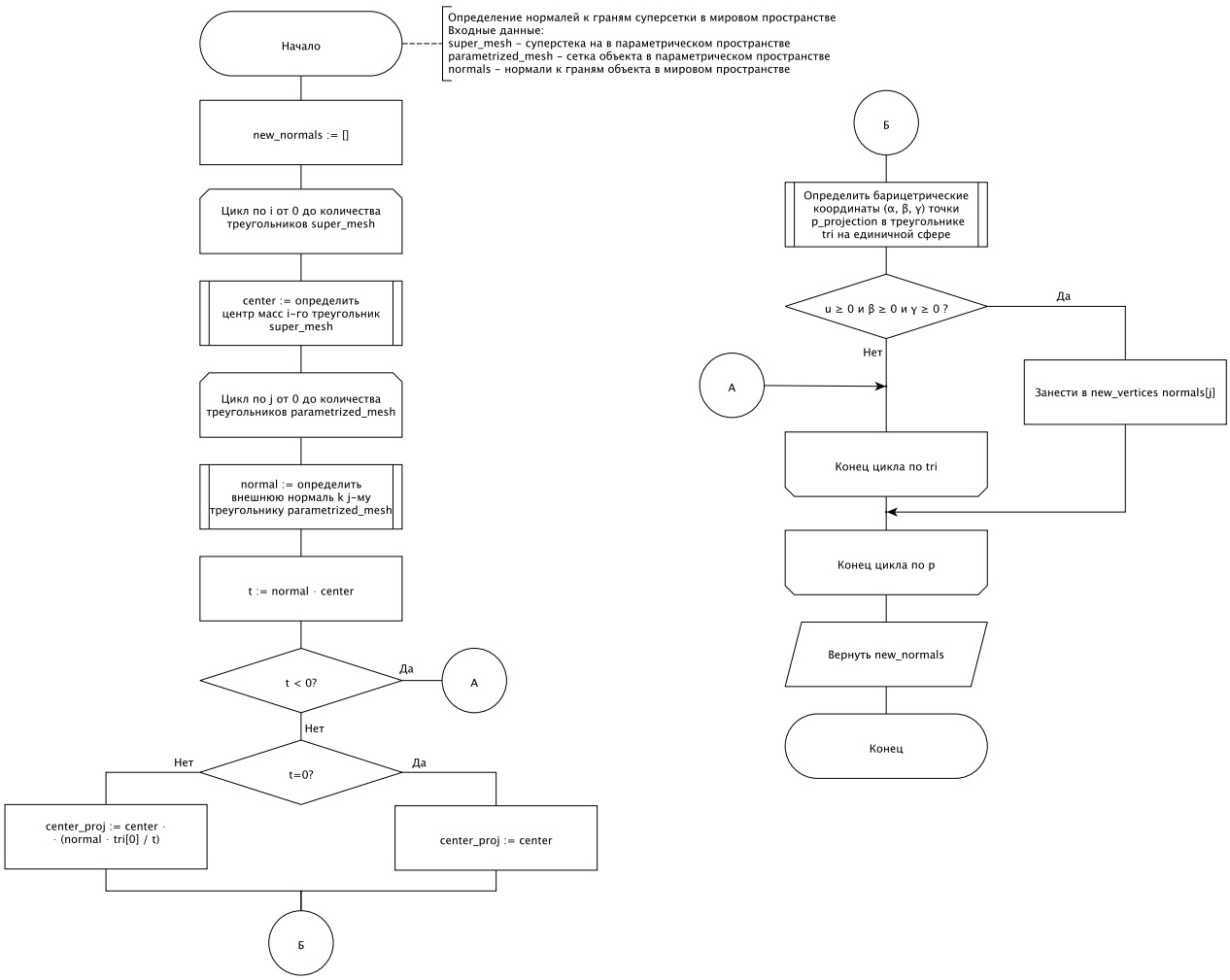

Рисунок 2.8 — Схема алгоритма определения нормалей к граням суперсетки на основе соответствия с сетокой объекта в параметрическом пространстве

2.4 Математические основы алгоритмов

2.4.1 Барицентрические координаты

Барицентрические координаты α, β и γ определяют положение точки P относительно вершин треугольника A, B и C [5].

Точка P находится внутри или на границах треугольника ABC, если она может быть представлена в виде аффинной комбинации вершин [5]:

$$P = \alpha A + \beta B + \gamma C, \text{ где } \alpha + \beta + \gamma = 1, \text{ и } \alpha, \beta, \gamma \geq 0 \text{ [5]} \tag{2.1}$$

Если хотя бы одна из координат α, β, γ отрицательна, то точка P лежит вне треугольника [5].

Барицентрическая координата α точки P пропорциональна площади тре-

угольника PBC (см. рисунок 2.9) (треугольника, противолежащего вершине A) [5].

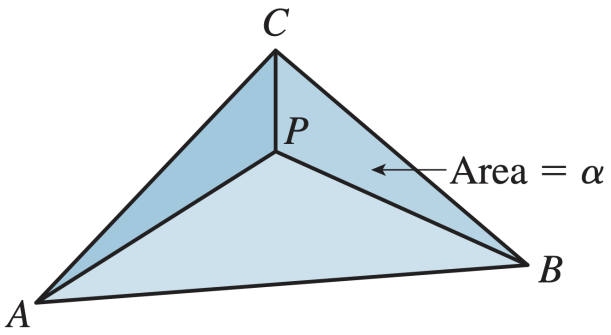

Рисунок 2.9 — Точка P разделяет треугольник ABC на три меньших треугольника, площади которых соотносятся как α, β и γ; Барицентрические координаты точки P равны (α, β, γ) [5]

Координата α точки P определяется как отношение площади треугольника PBC к площади всего треугольника ABC [5]:

$$\alpha = \frac{S_{\triangle PBC}}{S_{\triangle ABC}} \tag{2.2}$$

Аналогичные соотношения используются для β (пропорционально $S_{\triangle PAC}$) и γ (пропорционально $S_{\triangle PAB}$).

На плоскости барицентрические координаты точки P в треугольнике ABC могут быть вычислены при помощи косого произведения:

$$\begin{cases} \alpha = \frac{|BC \times BP|}{|AB \times AC|} = \frac{(C-B)_x \cdot (P-B)_y - (C-B)_y \cdot (P-B)_x}{(B-A)_x \cdot (C-A)_y - (B-A)_y \cdot (C-A)_x} \\ \beta = \frac{|CA \times CP|}{|AB \times AC|} = \frac{(A-C)_x \cdot (P-C)_y - (A-C)_y \cdot (P-C)_x}{(B-A)_x \cdot (C-A)_y - (B-A)_y \cdot (C-A)_x} \\ \gamma = 1 - \alpha - \beta \end{cases} \tag{2.3}$$

2.4.2 Поиск пересечения дуг на единичной сфере

Для нахождения пересечения дуг необходимо:

1) найти прямую по которой пересекаются плоскости, содержащие дуги;
2) найти точки пересечение прямой с единичной сферой;
3) проверить принадлежность точек обеим дугам.

Пусть дуга A задана радиус-векторами $\mathbf{A_0}$ и $\mathbf{A_1}$, а дуга B — векторами $\mathbf{B_0}$

и $\mathbf{B_1}$. Нормали к плоскостям, содержащим дуги, вычисляются согласно (2.4):

$$\begin{cases} \mathbf{n_A} = \mathbf{A_0} \times \mathbf{A_1} \\ \mathbf{n_B} = \mathbf{B_0} \times \mathbf{B_1} \end{cases} \tag{2.4}$$

Направляющий вектор $\mathbf{d}$ прямой их пересечения равен векторному произведению нормалей, как показано в (2.5):

$$\mathbf{d} = \mathbf{n_A} \times \mathbf{n_B} = (\mathbf{A_0} \times \mathbf{A_1}) \times (\mathbf{B_0} \times \mathbf{B_1}) \tag{2.5}$$

Радиус-векторы точек-кандидатов, полученные пересечением этой прямой с единичной сферой, определяются по формуле (2.6):

$$\mathbf{P}_{1,2} = \pm \frac{\mathbf{d}}{\|\mathbf{d}\|} \tag{2.6}$$

Точка $\mathbf{P}$ лежит на дуге $\mathbf{A_0A_1}$ тогда и только тогда, когда сумма сферических расстояний от концов дуги до точки равна длине самой дуги. На единичной сфере, где радиус-векторы точек единичны, сферическое расстояние равно углу между векторами и вычисляется через арккосинус их скалярного произведения.

Следовательно, одна из точек $\mathbf{P}$, найденных в (2.6), является пересечением, если для нее выполняются условия (2.7):

$$\begin{cases} \arccos(\mathbf{A_0} \cdot \mathbf{P}) + \arccos(\mathbf{P} \cdot \mathbf{A_1}) \leqslant \arccos(\mathbf{A_0} \cdot \mathbf{A_1}) \\ \arccos(\mathbf{B_0} \cdot \mathbf{P}) + \arccos(\mathbf{P} \cdot \mathbf{B_1}) \leqslant \arccos(\mathbf{B_0} \cdot \mathbf{B_1}) \end{cases} \tag{2.7}$$

2.4.3 Поиск точки внутри объекта

Основные шаги:

1) Испустить луч из центра масс O любой грани в направлении внутренней нормали.
2) Найти все точки пересечения с другими гранями.
3) Определить ближайшую к началу луча точку пересечения I.
4) Вернуть точку по середине отрезка OI.

Пересечение луча с полигоном

В параметрическом представлении луча задается уравнением (2.8).

$$P(t) = P_0 + \mathbf{d} \cdot t, \tag{2.8}$$

где P_0 — начальная точка луча; $\mathbf{d}$ — направляющий вектор луча.

Произвольная точка M лежит на плоскости A, если для нее выполняется равенство (2.9).

$$(M - M_A) \cdot \mathbf{n_A} = 0, \tag{2.9}$$

где M — произвольная точка
M_A — известная точка на плоскости A
$\mathbf{n_A}$ — вектор нормали к A.

После подстановки (2.8) в (2.9), получилось уравнение (2.10) для нахождения параметра t точки пересечения луча с плоскостью полигона.

$$t = \frac{(M_A - P_0) \cdot \mathbf{n_A}}{\mathbf{d} \cdot \mathbf{n_A}} \tag{2.10}$$

Если знаменатель $\mathbf{d} \cdot \mathbf{n_A}$ равен нулю, то луч параллелен плоскости полигона. Если $t < 0$, то пересечение находится позади начала луча, и такая точка отбрасывается, так как поиск ведется внутри объекта.

Найдя точку пересечения луча с плоскостью, необходимо определить, лежит ли она внутри полигона. Для этого можно использовать барицентрические координаты.

2.4.4 Матрицы преобразований

Матрицы преобразований в мировое пространство

Для перевода объектов из их локального пространства в мировое пространство используются матрицы масштабирования и поворота [2, 4, 5]. Матрицы поворота вокруг осей X, Y и Z представлены в формулах (2.11), (2.12) и (2.13) соответственно.

$$R_X(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.11}$$

$$R_Y(\theta) = \begin{pmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.12}$$

$$R_Z(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.13}$$

Матрица масштабирования представлена в формуле (2.14).

$$S = \begin{pmatrix} S_X & 0 & 0 & 0 \\ 0 & S_Y & 0 & 0 \\ 0 & 0 & S_Z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.14}$$

Вывод

В данном разделе была приведена функциональная схема морфинга, разработаны и представлены схемы основных алгоритмов и их математические основы.

3 Технологическая часть

3.1 Средства реализации

В качестве языка программирования для реализации алгоритмов выбран язык *Rust* [6], так как он удовлетворяет всем требования.

Для реализации графического интерфейса использована библиотека *egui* [7].

В качестве среды разработки была выбрана IDE *RustRover* [8].

3.2 Формат входных и выходных данных

Исходный и целевой объекты загружаются из файлов формата *.obj*. Файлы этого формата представляют собой текстовые файлы, в которых каждая строка описывает элемент геометрии. На листинге 3.1 приведен пример содержимого *.obj* файла.

Листинг 3.1 — Пример .obj файла

```
v 1.0 1.0 0.0
v -1.0 1.0 0.0
v -1.0 -1.0 0.0
v 1.0 -1.0 0.0

vn 0.0 0.0 1.0

f 1//1 2//1 3//1 4//1
```

Структура файла состоит из директив, каждая из которых начинается с ключевого слова. В рамках данной работы используются следующие:

— `v` — определяет геометрическую вершину и ее координаты (x, y, z).

— `vn` — определяет нормаль к вершине и ее компоненты (x, y, z).

— `f` — определяет грань (полигон). После ключевого слова перечисляются индексы вершин и нормалей, составляющих грань, в формате `v//vn`. Индексация начинается с 1.

Выходными данными является *RGB* изображение.

3.3 Реализация алгоритмов

В данном разделе приведены листинги кода, реализующего основные алгоритмы.

3.3.1 Реализация алгоритма с использованием z-буфера и модифицированной закраски Гуро

На листингах А.1, А.2 приведена реализация алгоритма с использованием z-буфера и модифицированной закраски Гуро.

3.3.2 Реализация алгоритмов морфинга

На листинге А.3 представлена реализация алгоритма построения объекта морфинга из исходной и целевой сеток.

На листингах 3.2–3.5 и А.4–А.5 представлены реализации алгоритмов основных этапов построения морфинга.

Листинг 3.2 — Реализация алгоритма сферической параметризации сетки

```
    }

    // 2. Проверка, находится ли P между Start и End.
    // Угол(start, p) + Угол(p, end) должен быть равен Углу(start
       , end).
    // Используем acos для получения углов.
    let angle_sp = start_vec.dot(&p_vec).clamp(-1.0, 1.0).acos();
    let angle_pe = p_vec.dot(&end_vec).clamp(-1.0, 1.0).acos();
    let angle_se = start_vec.dot(&end_vec).clamp(-1.0, 1.0).acos
       ();

    // clamp(-1.0, 1.0) защищает от ошибок f64, когда dot-продукт
        чуть-чуть > 1.0.

    // Проверяем, что сумма углов равна общему углу с некоторой т
       очностью.
    if (angle_sp + angle_pe - angle_se).abs() < ON_ARC_EPSILON {
        return true;
    }

    false
```

```
}

/// Finds the intersection point of two great-circle arcs on a
   unit sphere.
/// Returns `Some(Point3<f64>)` if a unique intersection is found
   , otherwise `None`.
fn intersect_arcs(arc_1: [&Point3<f64>; 2], arc_2: [&Point3<f64>;
    2]) -> Option<Point3<f64>> {
```

Листинг 3.3 — Реализация алгоритма построения суперсетки (часть 1)

```
        if t < 0. {
            continue;
        }

        let projected_point = if t < f64::EPSILON {
            *p
        } else {
            p * (normal.dot(&v0.coords) / t)
        };

        if t < f64::EPSILON {
            println!("{}", projected_point);
        }

        // 3. Определяем принадлежность точки треугольнику по бар
           ицентрическим координатам
        let bary = barycentric(&projected_point, v0, v1, v2);
```

Листинг 3.4 — Реализация алгоритма вычисления положения вершин суперсетки на поверхности объекта по их параметрическим координатам

```
    }

    Ok(result_normals)
}

/// Диагностирует DCEL и выводит подробную информацию о его струк
   туре
fn diagnose_dcel(dcel: &DCEL) {
    eprintln!("\n=== ДИАГНОСТИКА DCEL ===");
    eprintln!("Всего вершин: {}", dcel.vertices.len());
```

```
    eprintln!("Всего полурэбер: {}", dcel.half_edges.len());
    eprintln!("Всего граней: {}", dcel.faces.len());

    // Анализ граней
    let mut face_sizes: Vec<(usize, usize)> = Vec::new();
    for face_idx in 0..dcel.faces.len() {
        let vertices = dcel.get_face_vertices(face_idx);
        face_sizes.push((face_idx, vertices.len()));
    }

    // Сортируем грани по количеству вершин
```

Листинг 3.5 — Реализация алгоритма переноса нормалей граней суперсетки на объект на основе их соответствия в параметрическом пространстве

```
    for (i, &(face_idx, size)) in face_sizes.iter().take(10).
       enumerate() {
        let vertices = dcel.get_face_vertices(face_idx);
        eprintln!("{}. Грань {}: {} вершин", i + 1, face_idx,
           size);

        // ДЕТАЛЬНАЯ ДИАГНОСТИКА для граней с 10+ вершинами
        if size >= 10 {
            eprintln!("\n   === ДЕТАЛЬНЫЙ АНАЛИЗ ГРАНИ {} ===",
               face_idx);

            // Выводим ВСЕ вершины с координатами
            eprintln!("   Все {} вершин:", vertices.len());
            for (j, &v_idx) in vertices.iter().enumerate() {
                let v = &dcel.vertices[v_idx];
                eprintln!(
                    "      {}: v{} = ({:.6}, {:.6}, {:.6})",
                    j, v_idx, v.x, v.y, v.z
                );
            }

            // Выводим все рэбра грани
            eprintln!("\n   Рэбра грани (дуги на сфере):");
            let mut edges = Vec::new();
```

3.4 Интерфейс программы

На рисунках 3.1, 3.2 представлен пример работы программы.

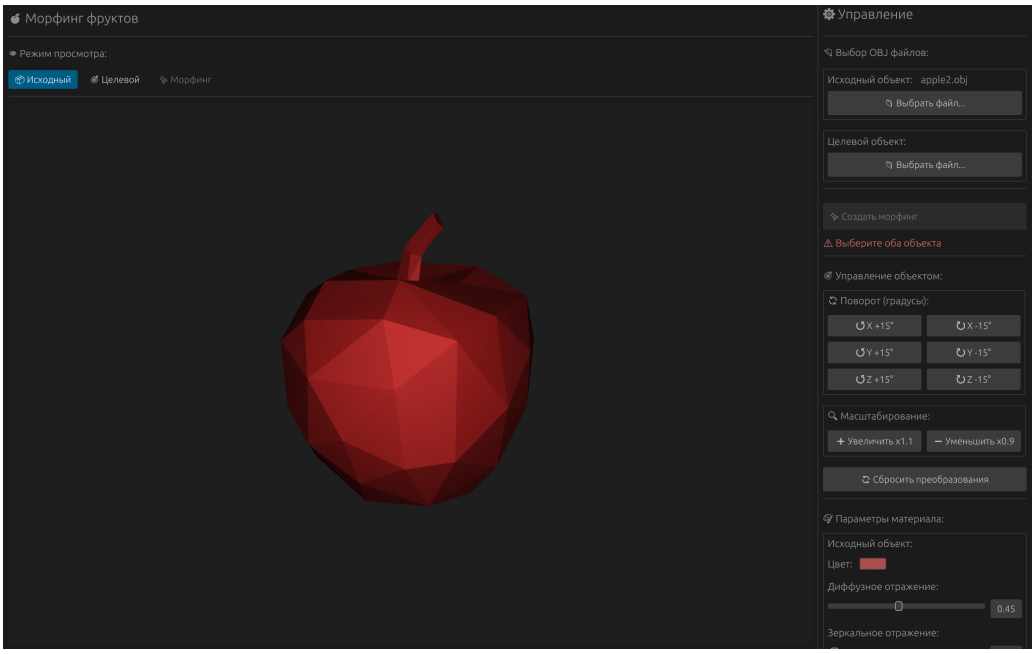

Рисунок 3.1 — Пример графического интерфейса программы — управление исходным объеком

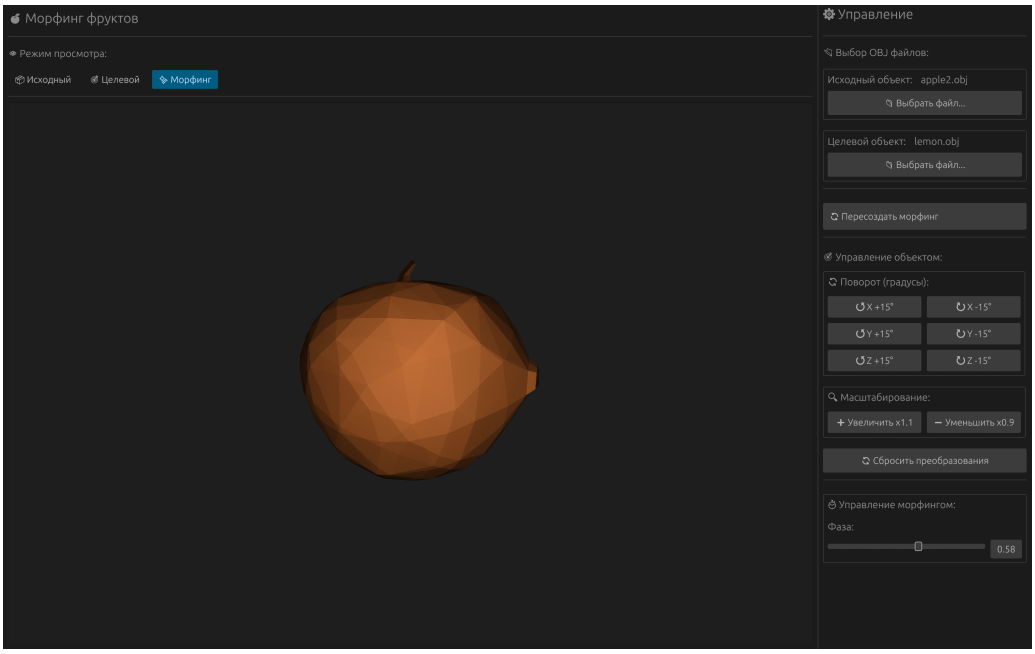

Рисунок 3.2 — Пример графического интерфейса программы — управления морфингом

Для загрузки объектов используются кнопки «Загрузить исходный объект» и «Загрузить целевой объект».

После того как объекты загружены, становятся доступна кнопка «Создать морфинг», по нажатию на которую создастся объекта морфинга. После того как объект морфинга будет создан, на верхней панели станет доступна кнопка «Морфинг».

Кнопки на верхней панели позволяют переключаться между просмотром исходного, целевого объекта и результата морфинга. Пример приведен на рисунке 3.1.

На вкладке «Морфинг» можно управлять стадией морфинга, путем изменения параметра «Фаза морфинга» и просматривать результат.

Поворот и масштабирование объектов выполняется с помощью мыши: зажатие левой кнопки мыши и движение мыши выполняет поворот объекта, а прокрутка колесика мыши выполняет масштабирование объекта. Либо при помощи соответствующих кнопок на боковой панели.

Изменение оптических свойств поверхности исходного или целевого объекта осуществляется с соответствующих ползунков на боковой панели.

3.5 Функциональное тестирование

Функциональное тестирование осуществлялось на основе визуального сравнения полученных изображений с ожидаемыми результатами. В таблице 3.1 приведены результаты функционального тестирования.

Таблица 3.1 — Результаты функционального тестирования

№	Входные данные	Ожидаемый результат	Фактический результат
1	Файл с моделью яблока, файл с моделью груши	Плавная анимация превращения яблока в грушу при увеличении параметра t	Плавная анимация превращения яблока в грушу при увеличении параметра t
2	Файл с моделью лимона, файл с моделью яблока	Плавная анимация превращения лимона в яблоко при увеличении параметра t	Плавная анимация превращения лимона в яблоко при увеличении параметра t
3	Файл с моделью банана, файл с моделью лимона	Плавная анимация превращения банана в лимон при увеличении параметра t	Плавная анимация превращения банана в лимон при увеличении параметра t
4	Файл с моделью груши, файл с моделью банана	Плавная анимация превращения груши в банан при увеличении параметра t	Плавная анимация превращения яблока в грушу при увеличении параметра t

Все тесты прошли успешно.

Вывод

В этом разделе были описаны средства реализации, описан формат входных и выходных данных, представлены реализации основных алгоритмов, продемонстрирован интерфейс программы.

4 Исследовательская часть

4.1 Технические характеристики

Технические характеристики устройства, на котором производились замеры

— процессор: Intel i9-9880H 8-core 2.30–4.80 ГГц [9];

— оперативное запоминающее устройство: 2667 MHz DDR4 2×8 ГБ [10];

— операционная система: macOS Sequoia 15.7.1 [11].

4.2 Порядок проведения исследования

Максимальное количество раундов релаксации — 10000. Для каждой пары моделей было выполнено не менее 100 независимых измерений процессорного времени каждого этапа, после чего полученные значения были усреднены. Во время проведения измерений исследуемое устройство было подключено к электросети, из сторонних приложений был запущен только эмулятор терминала *iTerm2* [12], а также были отключены *Wi-Fi* и *Bluetooth*.

При измерении времени были использованы следующие модели:

— Яблоко — 78 вершин, 152 полигона;

— Груша — 199 вершин, 394 полигона;

— Банан — 53 вершин, 102 полигона;

— Лимон — 72 вершины, 140 полигона.

4.3 Результаты исследования

В таблице 4.1 представлены результаты измерений времени построения морфинга и выполнения каждого отдельного этапа для даной реализации.

Таблица 4.1 — Время выполнения этапов морфинга (в наносекундах)

Пара моделей	Общее время	Параметризация	Построение суперсетки	Перенос вершин	Перенос нормалей
Лимон - Груша	36208501	639596	18366430	4040569	13161906
Яблоко - Лимон	13532119	75514	7521913	1475417	4459275
Лимон - Банан	9370467	81964	5339853	1077027	2871624
Яблоко - Груша	36569032	669114	3894843	12528669	19476405
Яблоко - Банан	9228840	107190	960505	2555064	5606079
Банан - Груша	26814225	669574	3091515	9457313	13595823

На рисунке 4.1 представлена диаграмма, иллюстрирующая вклад каждого этапа в общее время выполнения морфинга.

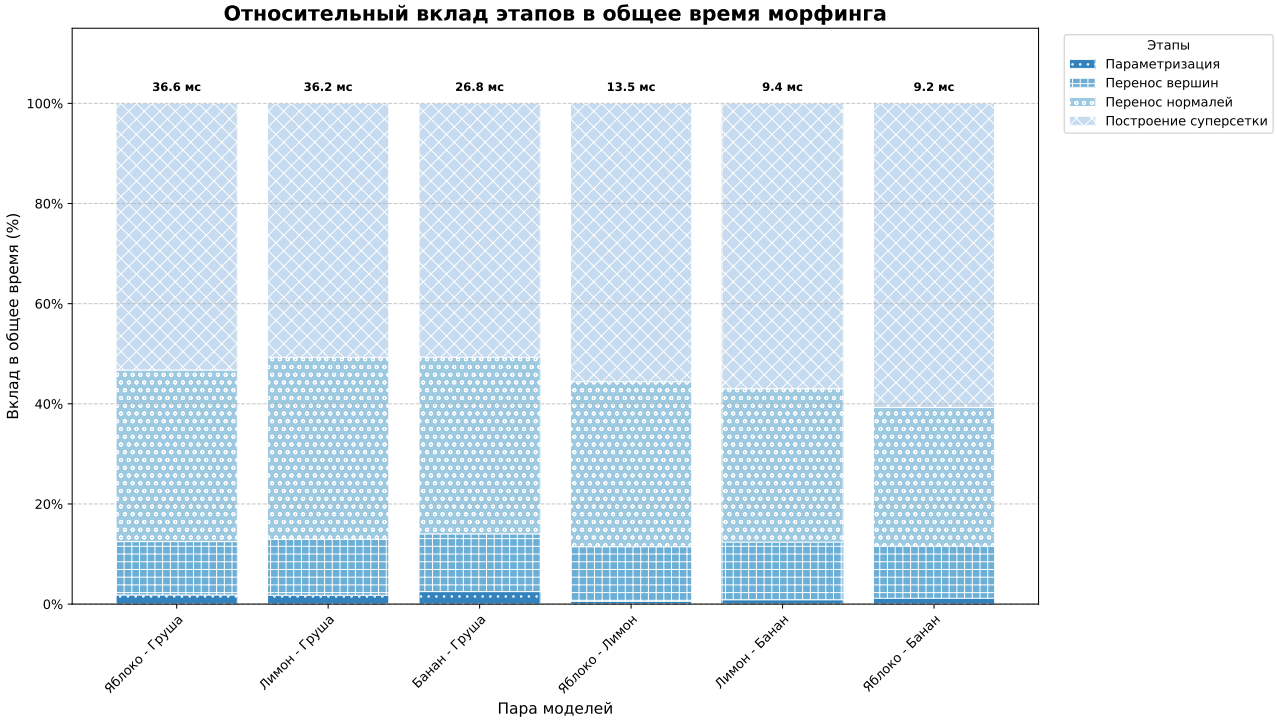

Рисунок 4.1 — Относительный вклад каждого этапа в общее выполнения морфинга

В таблице 4.2 представлены минимальная, максимальная и средняя доля времени выполнения каждого отдельного этапа в процентах, на основе результатов измерений в таблице 4.1.

Таблица 4.2 — Процентное соотношению времени выполнения этапов

Этап	Мин. %	Макс. %	Сред. %
Параметризация	0.56	2.50	1.45
Построение суперсетки	50.70	60.75	54.67
Перенос вершин	10.41	11.53	11.02
Перенос нормалей	27.69	36.35	32.86

На рисунке 4.2 представлена круговая диаграмма, иллюстрирующая среднее отношение времени выполнения этапов к общему времени выполнения морфинга.

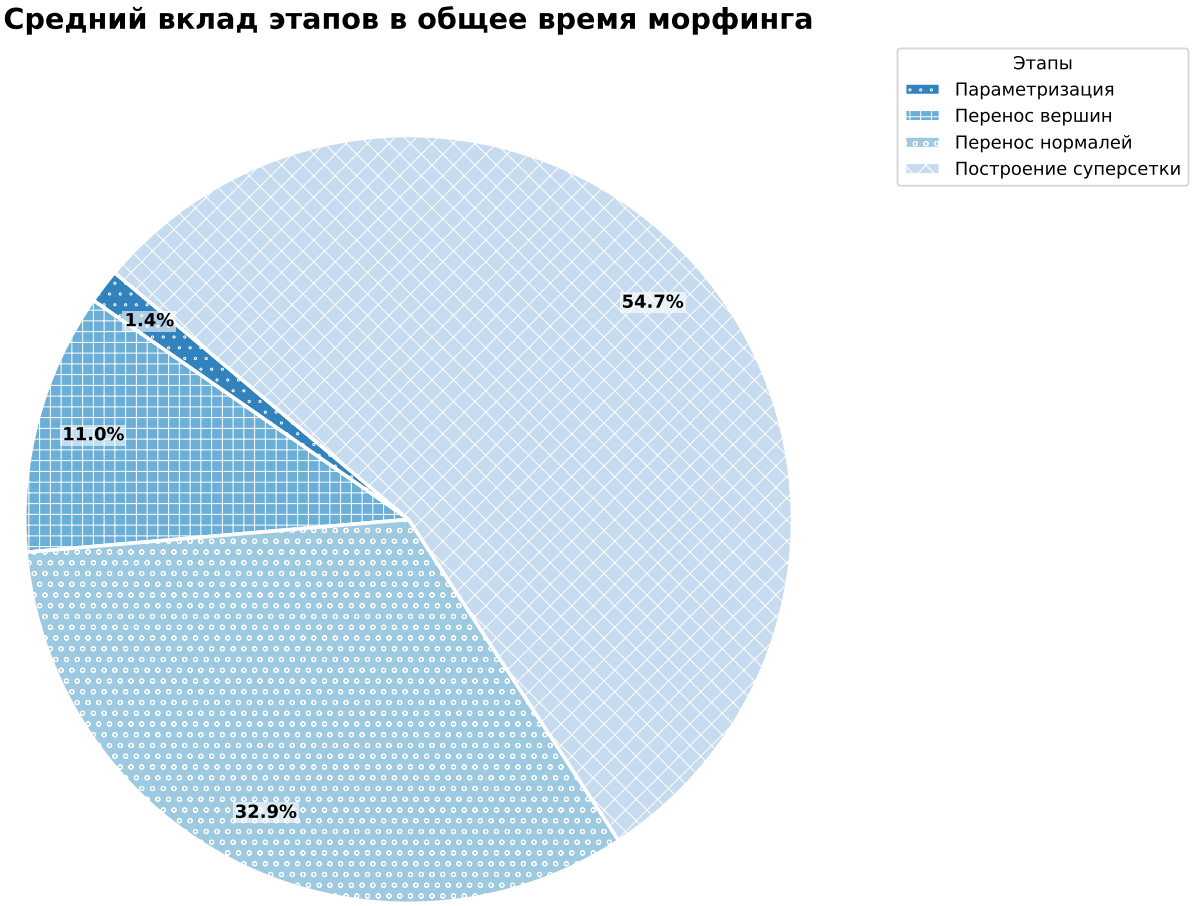

Рисунок 4.2 — Средний временной вклад каждого этапа в общее выполнение морфинга

4.4 Вывод

В рамках представленной реализации наиболее затратными по времени этапами являются построение суперсетки (в среднем 55% времени) и пере-

нос нормалей (около 33%). Вместе они занимают почти 90% общего времени выполнения. Наименее затратным является этап параметризации (менее 2%). Следует отметить, что в зависимости от сложности формы этап может занимать значительно больше времени, однако в рамках настоящего исследования для выбранных входных моделей условия остановки итерационного процесса релаксации достигались быстро.

ЗАКЛЮЧЕНИЕ

Поставленная цель достигнута, было разработано программное обеспечение для визуализации морфинга двух фруктов, представленных низкополигональными моделями без отверстий. Выполнены все поставленные задачи:

— перечислены основные принципы морфинга трехмерных объектов;

— описаны использованные техники морфинга трехмерных объектов;

— проанализированы и выбраны алгоритмы решения основных задач компьютерной графики;

— спроектировано программное обеспечение для визуализации процесса морфинга двух фруктов;

— выбраны средства реализации, описан формат входных и выходных данных, реализовано спроектированное программное обеспечение;

— исследован вклад времени выполнения этапов морфинга в общее время процесса.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Mocanu Bogdan-Costel. 3D Mesh Morphing. Thèse de doctorat / Doctoral thesis: TELECOM SudParis; Universitatea Politehnica Bucuresti. France / Roumanie, 2013. Numéro national de thèse : 2013TELE0010; HAL Id: tel-00836048.

2. Порев В. Компьютерная графика. СПб.: БХВ-Петербург., 2002.

3. Alexa M. Mesh Morphing. 2001.

4. Роджерс Д. Алгоритмические основы машинной графики: Пер. с англ. — СПб: БХВ-Петербург, 1989. — С. 512.

5. Foley J. Computer Graphics: Principles and Practice. 3 edition. Addison Wesley, 2013.

6. The Rust Programming Language. URL: `https://doc.rust-lang.org/book/` (дата обращения: 26.10.2025).

7. egui - Rust. URL: `https://docs.rs/egui/latest/egui/` (дата обращения: 26.10.2025).

8. RustRover: IDE для Rust. URL: `https://www.jetbrains.com/ru-ru/rust/` (дата обращения: 27.10.2025).

9. Intel® Core™ i9-9880H Processor. URL: `https://www.intel.com/content/www/us/en/products/sku/192987/intel-core-i99880h-processor-16m-cache-up-to-4-80-ghz/specifications.html` (дата обращения: 7.11.2025).

10. Micron MTA8ATF1G64HZ-2G6E1 part detail. URL: `https://www.micron.com/products/memory/dram-modules/sodimm/part-catalog/part-detail/mta8atf1g64hz-2g6e1?srsltid=AfmBOoqVXmH61q2uCshgOSvDHvCb_vyM5K4Oela_niedC5dCKjg3Tqee` (дата обращения: 7.11.2025).

11. macOS Sequoya 15.1. URL: https://developer.apple.com/documentation/macos-release-notes/macos-15_1-release-notes (дата обращения: 7.11.2025).

12. iTerm2. URL: https://iterm2.com (дата обращения: 7.11.2025).

Приложение А

Листинг А.1 — Реализация алгоритма с использованием z-буфера и модифицированной закраски Гуро (часть 1)

```
    fn draw_object(
        &mut self,
        image: &mut RgbImage,
        model: &dyn Model3D,
        camera: &Camera,
        light_source: &LightSource,
    ) {
        let (width, height) = image.dimensions();
        let mvp_matrix = camera.camera_matrix * model.
           model_matrix();
        let viewport_matrix = Self::calculate_viewport_matrix(
           width, height);
        let mvpv_matrix = viewport_matrix * mvp_matrix;

        let screen_vertices: Vec<Point3<f64>> =
            Self::transform_vertices_to_screen(model.vertices(),
               &mvpv_matrix);

        for (i, tri) in model.triangles().iter().enumerate() {
            let tri_colors = [tri.0, tri.1, tri.2].map(|v_idx| {
                calculate_color(
                    model.material(),
                    &model.normals()[i].xyz(),
                    &model.vertices_world()[v_idx],
                    light_source,
                    &camera.pos,
                )
            });

            self.draw_triangle(
                image,
                &[
                    screen_vertices[tri.0],
                    screen_vertices[tri.1],
                    screen_vertices[tri.2],
```

```
                ],
                &tri_colors,
            );
        }
    }
```

Листинг А.2 — Реализация алгоритма с использованием z-буфера и модифицированной закраски Гуро (часть 2)

```
    fn draw_triangle(
        &mut self,
        image: &mut RgbImage,
        tri: &[Point3<f64>; 3],
        tri_colors: &[Rgb<u8>; 3],
    ) {
        let [p1, p2, p3] = *tri;

        // Находим ограничивающий прямоугольник, ограничивая разм
           ерами изображения.
        let min_x = p1.x.min(p2.x).min(p3.x).round() as u32;
        let max_x = (p1.x.max(p2.x).max(p3.x).round() as u32).min
           (self.width - 1);
        let min_y = p1.y.min(p2.y).min(p3.y).round() as u32;
        let max_y = (p1.y.max(p2.y).max(p3.y).round() as u32).min
           (self.height - 1);

        // Предварительно вычисляем общие компоненты, чтобы избеж
           ать избыточных вычислений в цикле.
        let denom = (p2.x - p1.x) * (p3.y - p1.y) - (p2.y - p1.y)
            * (p3.x - p1.x);

        for y in min_y..=max_y {
            for x in min_x..=max_x {
                // Вычисляем барицентрические координаты.
                let u =
                    ((p3.x - p2.x) * (y as f64 - p2.y) - (p3.y -
                       p2.y) * (x as f64 - p2.x)) / denom;
                let v =
                    ((p1.x - p3.x) * (y as f64 - p3.y) - (p1.y -
                       p3.y) * (x as f64 - p3.x)) / denom;
```

```
                let bary = Point3::new(u, v, 1.0 - u - v);

                // Проверяем, находится ли пиксель внутри треугол
                   ьника.
                if bary.x > -f64::EPSILON && bary.y > -f64::
                   EPSILON && bary.z > -f64::EPSILON {
                    let z = p1.z * bary.x + p2.z * bary.y + p3.z
                       * bary.z;

                    // Выполняем проверку по Z-буферу.
                    if z < self.get_depth(x, y) {
                        self.set_depth(x, y, z);

                        // Интерполируем цвета корректно для кажд
                           ого канала.
                        let r = (bary.x * tri_colors[0].0[0] as
                           f64
                            + bary.y * tri_colors[1].0[0] as f64
                            + bary.z * tri_colors[2].0[0] as f64)
                            .clamp(0.0, 255.0) as u8;
                        let g = (bary.x * tri_colors[0].0[1] as
                           f64
                            + bary.y * tri_colors[1].0[1] as f64
                            + bary.z * tri_colors[2].0[1] as f64)
                            .clamp(0.0, 255.0) as u8;
                        let b = (bary.x * tri_colors[0].0[2] as
                           f64
                            + bary.y * tri_colors[1].0[2] as f64
                            + bary.z * tri_colors[2].0[2] as f64)
                            .clamp(0.0, 255.0) as u8;

                        image.put_pixel(x, y, Rgb([r, g, b]));
                    }
                }
            }
        }
```

Листинг А.3 — Реализация алгоритма построения мофринга из исходной и целевой сеток

```
// 1. Параметризация исходных сеток
let mut parametrized_source_mesh = source_object.clone();
parametrize_mesh(&mut parametrized_source_mesh);

let mut parametrized_target_mesh = target_object.clone();
parametrize_mesh(&mut parametrized_target_mesh);

// 2. Построение суперсетки
let (vertices, triangles) =
    create_supermesh(&parametrized_source_mesh, &
       parametrized_target_mesh)?;

// 3. Находим положения точек на исходной и целевой сетка
   x
let src_vertices = relocate_vertices_on_mesh(
    &vertices,
    &parametrized_source_mesh,
    source_object.vertices_world(),
)?;
let dst_vertices = relocate_vertices_on_mesh(
    &vertices,
    &parametrized_target_mesh,
    target_object.vertices_world(),
)?;

let src_normals = find_normals(
    &vertices,
    &triangles,
    &parametrized_source_mesh,
    source_object.normals(),
)?;
let dst_normals = find_normals(
    &vertices,
    &triangles,
    &parametrized_target_mesh,
    target_object.normals(),
)?;

// 4. Строим интерполяции
let vertex_interpolations: Vec<VertexInterpolation> =
```

```
        src_vertices
            .into_iter()
            .zip(dst_vertices.into_iter())
            .map(|(src_v, dst_v)| -> VertexInterpolation {
                Box::new(move |t: f64| Point::from((1. - t) *
                    src_v.coords + t * dst_v.coords))
            })
            .collect();

        let normals_interpolations: Vec<NormalInterpolation> =
            src_normals
            .into_iter()
            .zip(dst_normals.into_iter())
            .map(|(src_n, dst_n)| -> NormalInterpolation {
                Box::new(move |t: f64| lerp(src_n, dst_n, t))
            })
            .collect();

        let src_material = source_object.material().clone();
        let dst_material = target_object.material().clone();
        let material_interpolation: MaterialInterpolation =
            Box::new(move |t: f64| Material::lerp(&src_material,
                &dst_material, t));

        // 5. Строим интерполяции при t=0
        // 5.1 Строим вершины
        let vertices: Vec<Point> = vertex_interpolations.iter().
            map(|lerp| lerp(0.)).collect();
        let vertices_world = vertices.clone();

        // 5.2 Строим нормали
        let normals: Vec<Vector4<f64>> =
            normals_interpolations.iter().map(|lerp| lerp(0.)).
                collect();
        let normals_world = normals.clone();

        // 5.3 Строим материал
        let material = material_interpolation(0.);

        Ok(Morph {
```

```
            vertices,
            vertices_world,
            triangles,
            normals,
            normals_world,
            material,
            vertex_interpolations,
            normals_interpolations,
            material_interpolation,
            model_matrix: Matrix4::identity(),
        })
    }
}
```

Листинг А.4 — Реализация алгоритма релаксации сетки на единичной сфере

```
            .iter()
            .zip(vertices.iter())
            .all(|(prev, curr)| (prev - curr).norm() <
               epsilon_threshold);

        // Центрирование сферы для избежания коллапса вершин
        let mean: Vector3<f64> =
            vertices.iter().map(|v| v.coords).sum::<Vector3<f64
               >>() / vertices.len() as f64;
        vertices.iter_mut().for_each(|v| *v -= mean);

        // Главное условие остановки: Ориентации граней совпадают
            с оригинальными (нет вывернутых граней)
        orientations = get_orientations(
            parametrized_mesh.vertices_world(),
            parametrized_mesh.triangles(),
        );
        orientations_established = original_orientations.iter().
           eq(orientations.iter());

        round_no += 1;
    }

    // println!("{}", round_no);
}
```

```
fn find_inner_point(mesh: &TriangleMesh) -> Option<Vertex> {
    let vertices = mesh.vertices_world();
    let normals = mesh.normals();
    let triangles = mesh.triangles();

    // 1. Испускаем луч из середины первого попавшегося полигона
       в направлении внутренней нормали
    let tri = triangles[0];
    let ray_direction = normals[0].xyz().scale(-1.0);
    let ray_origin =
        (vertices[tri.0].coords + vertices[tri.1].coords +
           vertices[tri.2].coords) / 3.0;

    // 2. Находим все пересечения луча с другими полигонами
    let intersections = izip!(triangles, normals)
        .skip(1)
        .filter_map(|(tri, normal)| {
            let normal = normal.xyz();
            let t = (vertices[tri.0].coords - ray_origin).dot(&
               normal) / ray_direction.dot(&normal);
            if t < f64::EPSILON || t.is_infinite() || t.is_nan()
               {
                return None;
            }

            let intersection = Vertex::from(ray_origin +
               ray_direction.scale(t));

            let bary = barycentric(
                &intersection,
                &vertices[tri.0],
                &vertices[tri.1],
                &vertices[tri.2],
            );
            if bary.x > -f64::EPSILON && bary.y > -f64::EPSILON
               && bary.z > -f64::EPSILON {
                Some(intersection)
            } else {
                None
```

Листинг А.5 — Реализация алгоритма построения суперсетки (часть 2)

```
// Проверяем вершины сетки A на рэбрах сетки B
find_vertices_on_edges(&mapping_a, &segments_b, &all_vertices
   , &mut segment_map);

// Проверяем вершины сетки B на рэбрах сетки A
find_vertices_on_edges(&mapping_b, &segments_a, &all_vertices
   , &mut segment_map);

// 5. Находим точки пересечения между дугами
for &seg_a in &segments_a {
    for &seg_b in &segments_b {
        // Пропускаем, если сегменты имеют общие вершины
        if seg_a[0] == seg_b[0]
            || seg_a[0] == seg_b[1]
            || seg_a[1] == seg_b[0]
            || seg_a[1] == seg_b[1]
        {
            continue;
        }

        let arc_1 = [&all_vertices[seg_a[0]], &all_vertices[
           seg_a[1]]];
        let arc_2 = [&all_vertices[seg_b[0]], &all_vertices[
           seg_b[1]]];

        if let Some(intersection_point) = intersect_arcs(
           arc_1, arc_2) {
            let inter_idx = find_or_add_vertex(&mut
               all_vertices, &intersection_point);
            segment_map.get_mut(&seg_a).unwrap().insert(
               inter_idx);
            segment_map.get_mut(&seg_b).unwrap().insert(
               inter_idx);
        }
    }
}

// 6. Генерируем финальный список подотрезков
let mut all_segments: HashSet<Segment> = HashSet::new();
```

```
for ([start_idx, end_idx], points_idx_set) in segment_map.
   into_iter() {
    let mut points_indices: Vec<usize> = points_idx_set.
       into_iter().collect();

    // Сортируем точки вдоль дуги на основе их угла поворота
       от начальной точки
    let start_coords = all_vertices[start_idx].coords;

    points_indices.sort_unstable_by(|&a_idx, &b_idx| {
        let a_coords = all_vertices[a_idx].coords;
        let b_coords = all_vertices[b_idx].coords;

        let dot_a = start_coords.dot(&a_coords);
        let dot_b = start_coords.dot(&b_coords);

        dot_b.partial_cmp(&dot_a).unwrap()
    });

    // Добавляем начальную и конечную вершины
    points_indices.insert(0, start_idx);
    points_indices.push(end_idx);

    // Создаем новые подсегменты
    for i in 0..points_indices.len() - 1 {
        let mut seg = [points_indices[i], points_indices[i +
           1]];
        if seg[0] == seg[1] {
            continue; // Пропускаем вырожденные сегменты
        }

        // Делаем сегмент каноническим
        if seg[0] > seg[1] {
            seg.swap(0, 1);
        }

        all_segments.insert(seg);
    }
}
```

```
    DCEL::new(all_vertices, all_segments)
}

/// Треангулирует плоскую грань многогранника с использованием тр
   иангуляции Делоне.
fn triangulate_face(face_vertices: &Vec<&Vertex>) -> Result<Vec<
   usize>, Box<dyn Error>> {
    // Проверка минимального количества вершин
    if face_vertices.len() < 3 {
        eprintln!(
            "DEBUG: triangulate_face - недостаточно вершин: {} (т
               ребуется >= 3)",
            face_vertices.len()
        );
        return Err(format!(
            "Грань должна содержать минимум 3 вершины, получено:
               {}",
            face_vertices.len()
        )
        .into());
    }

    // 1. Находим нормаль к грани многогранника
    // Поскольку грань может содержать отрезки, лежащие на одной
       прямой,
    // подбираем вектор, не параллельный первому
    let v1 = face_vertices[1] - face_vertices[0];
    let mut normal = Vector3::default();
    for i in 2..face_vertices.len() {
        normal = v1.cross(&(face_vertices[i] - face_vertices[0]))
           ;
        if normal.norm() > 0. {
            break;
        }
    }

    // Проверка валидности нормали
    if normal.norm() < f64::EPSILON {
        eprintln!("DEBUG: triangulate_face - не удалось вычислить
```

```
            нормаль к грани");
        eprintln!("  v1 = {}", v1);
        if face_vertices.len() > 2 {
            eprintln!("  v2 = {}", face_vertices[2] -
               face_vertices[0]);
        }
        eprintln!("  normal.norm() = {}", normal.norm());
        eprintln!("  количество вершин: {}", face_vertices.len())
           ;
        return Err("Не удалось вычислить нормаль к грани: все вер
           шины коллинеарны".into());
    }

    normal.normalize_mut();
```

Приложение Б

Презентация на N слайдах.