Оглавление

Введение

В современном мире компьютерная графика применяется повсюду. С помощью синтеза изображения создаются спецэффекты в кинофильмах, графика в компьютерных играх, виртуальная реальность. Компьютерная графика используется в науке и промышленности для моделирования и визуализации различных физических процессов.

Целью данной работы является получение реалистичного изображения полого кусочка льда с жидкостью внутри. Чтобы достигнуть поставленной цели, требуется решить следующие задачи:

- провести анализ алгоритмов построения реалистичных изображений;
- разработать метод построения реалистичного изображения полого кусочка льда с жидкостью внутри;
- реализовать разработанный метод;
- провести оптимизацию реализации с использованием многопоточности;
- сравнить однопоточную и многопоточную реализации.

1 Аналитическая часть

В данном разделе проводится анализ задачи построения трёхмерного изображения сцены из объектов, рассматриваются различные методы, решающие данную задачу.

1.1 Описание модели трёхмерного объекта на сцене

Использование моделей позволяет точно отображать форму и размеры объекта на сцене.

Модели могут задаваться следующими способами:

- Каркасная (проволочная) модель. В этой модели задается информация о вершинах и рёбрах объектов. Это простейший вид моделей. Этим видам модели присущ один, но весьма существенный недостаток: не всегда модель правильно передает представление об объекте.

- Поверхностная модель. Поверхность может описываться аналитически, либо может задаваться другим способом. Недостаток: отсутствует информация о том, с какой стороны поверхности находится материал.

- Объёмная модель. Эта форма модели отличается от поверхностной тем, что в объёмных моделях к информации о поверхности добавляется информация о том, с какой стороны расположен материал. Проще всего это можно сделать путём указания направления внутренней нормали.

Для решения поставленной задачи не подойдёт проволочная модель, так как в этом случае модель будет неправильно передавать представление об объекте. Поверхностная модель также не подойдёт, потому что нам необходимо знать, с какой стороны поверхности расположен материал. Таким образом, выбираем объёмную модель.

1.2 Описание способа задания трёхмерного объекта на сцене

Существует несколько способов задания объёмной модели:

- Аналитический способ. Этот способ характеризуется описанием объекта в неявной форме, то есть для получения визуального представления нужно вычислять значение некоторой функции в различных точках пространства.

- Полигональная сетка. Это совокупность вершин, рёбер и граней, которые определяют форму многогранного объекта в трёхмерной компьютерной графике и объёмном моделировании. Гранями обычно являются треугольники, четырёхугольники или другие простые выпуклые многоугольники (полигоны). В данной работе грани имеют форму треугольников, так как любой полигон можно представить в виде треугольника. В свою очередь существуют различные способы хранения информации о полигональной сетке:

  - Список граней. Характеризуется множеством граней и множеством вершин. В каждую грань входят как минимум три вершины.
  - «Крылатое» представление. Представляет вершины, грани и ребра сетки. Это представление широко используется в программах для моделирования для предоставления высочайшей гибкости в динамическом изменении геометрии сетки, потому что могут быть быстро выполнены операции разрыва и объединения. Их основной недостаток – высокие требования памяти и увеличенная сложность из-за содержания множества индексов.
  - Вершинное представление. Описывает объект как множество вершин, соединенных с другими вершинами. Это простейшее представление, но оно не широко используемое, так как информация о гранях и ребрах не выражена явно. Поэтому нужно обойти все данные чтобы сгенерировать список граней для рендеринга. Кроме того, не легко выполняются операции на ребрах и гранях.

При выборе способа задания объекта в курсовой работе определяющим фактором стала скорость выполнения геометрических преобразований.

Оптимальное представление – полигональная сетка. Такая модель позволит легко описывать сложные объекты сцены.

Способом хранения информации о полигональной сетке был выбран список граней, потому что этот способ даёт явное описание граней, что позволяет эффективно выполнять геометрические преобразования над объектами сцены.

1.3 Формализация объектов синтезируемой сцены

Синтезируемая сцена состоит из следующих объектов:

1) Геометрический объект – представляется в виде полигональной сетки. Для описания геометрического объекта требуется указать координаты вершин, связи вершин (рёбра), фоновое освещение (цвет), диффузное освещение (цвет), зеркальное освещение (цвет), коэффициент фонового освещения, коэффициент диффузного освещения, коэффициент зеркального освещения, степень, аппроксимирующая пространственное распределение зеркально отражённого света, коэффициент отражения, коэффициент преломления, показатель преломления.

2) Источник света – представляется в виде вектора. Также к характеристикам источника света относятся расположение, цвет и интенсивность излучения.

3) Камера – характеризуется пространственным положением и направлением взгляда.

1.4 Анализ задачи построения трёхмерного изображения сцены из объектов

Построение трёхмерного изображения сцены заключается в преобразовании объектов этой сцены в изображение на растровом дисплее. Для создания реалистичного изображения необходимо учитывать несколько факторов:

- Удаление невидимых линий и поверхностей.
- Учёт теней.
- Учёт освещения.
- Учёт свойств материалов объекта: способность отражать свет, способность преломлять свет.

1.4.1 Удаление невидимых линий и поверхностей

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

Алгоритмы удаления невидимых линий и поверхностей делятся на:

- Алгоритмы, работающие в объектном пространстве (мировая система координат, высокая точность).
- Алгоритмы, работающие в пространстве изображений (система координат связана с дисплеем, точность ограничена разрешающей способностью дисплея).

Рассмотрим алгоритмы удаления невидимых линий и поверхностей.

Алгоритм Робертса

Данный алгоритм работает в объектном пространстве, решая задачу только с выпуклыми телами.

Алгоритм выполняется в три этапа.

1) Этап подготовки исходных данных. На данном этапе должна быть задана информация о телах. Для каждого тела сцены должна быть сформирована матрица тела V. Размерность матрицы – $4*n$, где n – количество граней тела.

   Каждый столбец матрицы представляет собой четыре коэффициента уравнения плоскости $ax + by + cz + d = 0$, проходящей через очередную грань.

   Таким образом, матрица тела будет представлена в следующем виде:

$$V = \begin{pmatrix} a_1 & a_2 & \dots & a_n \\ b_1 & b_2 & \dots & b_n \\ c_1 & c_2 & \dots & c_n \\ d_1 & d_2 & \dots & d_n \end{pmatrix} \tag{1.1}$$

   Матрица тела должна быть сформирована корректно, то есть любая точка, расположенная внутри тела, должна располагаться по положительную сторону от каждой грани тела. В случае, если для очередной грани условие не выполняется, соответствующий столбец матрицы надо умножить на -1. Для проведения проверки следует взять точку, расположенную внутри тела. Координаты такой точки можно получить путем усреднения координат всех вершин тела.

2) Этап удаления ребер, экранируемых самим телом. На данном этапе рассматривается вектор взгляда $E = \{0, 0, -1, 0\}$. Для определения невидимых граней достаточно умножить вектор E на матрицу тела V. Отрицательные компоненты полученного вектора будут соответствовать невидимым граням.

3) Этап удаления невидимых ребер, экранируемых другими телами сцены. На данном этапе для определения невидимых точек ребра требуется построить луч, соединяющий точку наблюдения с точкой на ребре. Точка

будет невидимой, если луч на своем пути встречает в качестве преграды рассматриваемое тело. Если тело является преградой, то луч должен пройти через тело. Если луч проходит через тело, то он находится по положительную сторону от каждой грани тела.

Преимущества алгоритма Робертса:

- алгоритм работает в объектном пространстве, точность вычислений высокая.

Недостатки алгоритма Робертса:

- теоретический рост сложности алгоритма – квадрат числа объектов. Для решения данной проблемы достаточно воспользоваться модифицированными реализациями, например, с использованием габаритных тестов или сортировки по оси z;
- все тела сцены должны быть выпуклыми. Данная проблема также приводит к усложнению алгоритма, так как потребуется прибегнуть к проверке объектов на выпуклость и их разбиению на выпуклые многоугольники.

Таким образом, алгоритм Робертса не подходит для решения поставленной задачи по следующим причинам.

- Возникновение проблем при наличии множества невыпуклых тел на сцене.
- Невозможность визуализации зеркальных поверхностей.

Алгоритм, использующий z-буфер

Данный алгоритм работает в пространстве изображения. Используется два буфера:

- буфер кадра, в котором хранятся атрибуты каждого пикселя в пространстве изображения;
- z-буфер, куда помещается информация о координате z для каждого пикселя.

Первоначально в z-буфере находятся минимально возможные значения z, а в буфере кадра располагаются пиксели, описывающие фон. Каждый многоугольник преобразуется в растровую форму и записывается в буфер кадра.

В процессе подсчета глубины нового пикселя, он сравнивается с тем значением, которое уже лежит в z-буфере. Если новый пиксель расположен ближе к наблюдателю, чем предыдущий, то он заносится в буфер кадра и происходит корректировка z-буфера.

Для решения задачи вычисления глубины z каждый многоугольник описывается уравнением $ax + by + cz + d = 0$. При $c = 0$ многоугольник для наблюдателя вырождается в линию.

Для некоторой сканирующей строки $y = const$, поэтому имеется возможность рекуррентно высчитывать z' для каждого $x' = x + dx$:

$$z' - z = -\frac{ax' + d}{c} + \frac{ax + d}{c} = \frac{a(x - x')}{c} \tag{1.2}$$

Получим: $z' = z - \frac{a}{c}$, так как $x - x' = dx = 1$.

При этом стоит отметить, что для невыпуклых многогранников предварительно потребуется удалить нелицевые грани.

Преимущества алгоритма, использующего z-буфер:

- простота реализации;
- алгоритм имеет линейную сложность.

Недостатки алгоритма, использующего z-буфер:

- большой объем требуемой памяти;
- реализация эффектов прозрачности сложна;
- дополнительные вычислительные операции в случае невыпуклых тел.

Таким образом, алгоритм, использующий z-буфер не подходит для решения поставленной задачи по следующим причинам.

- Сложность визуализации прозрачных поверхностей.
- Невозможность визуализации зеркальных поверхностей.

Алгоритм обратной трассировки лучей

Метод прямой и обратной трассировки [1] заключается в том, что от момента испускания лучей источником света до момента попадания в камеру, траектории лучей отслеживаются, и рассчитываются взаимодействия лучей с лежащими на траекториях объектами. Луч может быть поглощен, диффузно или зеркально отражен или, в случае прозрачности некоторых объектов, преломлен. Ray tracing – метод расчета глобального освещения, рассматривающий освещение, затенение (расчет тени), многократные отражения и преломления. Для расчета теней применяют методы прямой и обратной трассировки лучей. Метод прямой трассировки предполагает построение траекторий лучей от всех источников освещения ко всем точкам всех объектов сцены. Это, так называемые, первичные лучи. Точки, лежащие на противоположной от источника света стороне, исключаются из расчета. Для всех остальных точек вычисляется освещенность с помощью локальной модели освещения. Если объект не является отражающим или прозрачным, то траектория луча на этой точке обрывается. Если же поверхность объекта обладает свойством отражения (reflection) или преломления (refraction), то из точки строятся новые лучи, направления которых совершенно точно определяются законами отражения и преломления. Траектории новых лучей также отслеживаются. Построение новых траекторий и расчеты ведутся до тех пор, пока все лучи либо попадут в камеру, либо выйдут за пределы видимой области. Очевидно, что при прямой трассировке лучей мы вынуждены выполнять расчеты для лучей, которые не попадут в камеру, то есть проделывать бесполезную работу. По некоторым оценочным данным, доля таких «слепых» лучей довольно велика. Эта главная, хотя и далеко не единственная причина того, что метод прямой трассировки лучей считается неэффективным и на практике не используется. Алгоритм обратной трассировки лучей - основной способ расчета освещенности методом трассировки лучей. Метод трассировки лучей – первый метод расчета глобальной освещенности, учитывающий взаимное влияние объектов сцены друг на друга.

Алгоритм трассировки лучей требует большого количества вычислений, поскольку он предполагает поиск пересечений всех объектов сцены со всеми лучами, количество которых равно размеру растра. Поэтому время синтеза изображения оказывается очень большим. Однако из описания алгоритма

видно, что вычисление цвета каждого пиксела может быть выполнено параллельно. Такая оптимизация позволит значительно ускорить синтез изображения.

Преимущества алгоритма обратной трассировки лучей:

- высокая реалистичность синтезируемого изображения;
- реализация алгоритма предполагает учёт теней;
- простота визуализации зеркальных и прозрачных поверхностей.

Недостатки алгоритма обратной трассировки лучей:

- производительность.

Вывод

Таким образом, в качестве алгоритма удаления невидимых рёбер и поверхностей был выбран алгоритм обратной трассировки лучей из-за высокой реалистичного синтезируемого изображения и возможности визуализации зеркальных и прозрачных поверхностей.

1.4.2 Учёт теней

При использовании алгоритма обратной трассировки лучей, выбранного в качестве алгоритма удаления невидимых рёбер и поверхностей, построение теней происходит по ходу выполнения алгоритма: пиксель будет затемнён, если луч испускаемый из точки попадания первичного луча, испускаемого из камеры, попадает на другой объект.

1.4.3 Учёт освещения

Модель освещения предназначена для того, чтобы рассчитать интенсивность отражённого к наблюдателю света в каждой точке (пикселе) изображения. Глобальная модель учитывает не только свет и ориентацию поверхностей, но также и свет, отражённый от других объектов (или пропущен-

ный через них) Благодаря этому глобальная модель освещённости способна воспроизводить эффекты зеркального отражения и преломления лучей (прозрачность и полупрозрачность), а также затенение, что является необходимым для решения поставленной в курсовой работе задачи. Глобальная модель является составной частью алгоритма удаления невидимых рёбер и поверхностей с помощью обратной трассировки лучей.

Глобальная модель освещения для каждого пикселя изображения определяет его интенсивность. Сначала определяется непосредственная освещённость источниками без учёта отражений от других поверхностей (вторичная освещённость): отслеживаются лучи, направленные ко всем источникам. Тогда наблюдаемая интенсивность (или отражённая точкой энергия) выражается следующим соотношением:

$$I = k_0 I_0 + k_d \sum_j I_j (n \cdot l_j) + k_r \sum_j I_j (s \cdot r_j)^\beta + k_r I_r + k_t I_t, \qquad (1.3)$$

где

k_0 – коэффициент фонового (рассеянного) освещения,

k_d – коэффициент диффузного отражения,

k_r – коэффициент зеркального отражения,

k_t – коэффициент пропускания,

n – единичный вектор нормали к поверхности в точке,

l_j – единичный вектор, направленный к j-му источнику света,

s – единичный локальный вектор, направленный в точку наблюдения,

r_j – отражённый вектор l_j,

I_0 – интенсивность фонового освещения,

I_j – интенсивность j-го источника света,

I_r – интенсивность, приходящая по зеркально отражённому лучу,

I_t – интенсивность, приходящая по преломлённому лучу.

В алгоритме удаления невидимых рёбер и поверхностей трассировка луча продолжалась до первого пересечения с поверхностью. В глобальной модели освещения этим дело не ограничивается: осуществляется дальнейшая трассировка отражённого и преломлённого лучей. Таким образом, происходит разветвление алгоритма в виде двоичного дерева. Процесс продолжается до тех пор, пока очередные лучи не останутся без пересечений. Отражение и преломление рассчитываются по законам геометрической оптики. Теоретиче-

ски дерево может оказаться бесконечным, поэтому при его построении желательно задать максимальную глубину, чтобы избежать переполнения памяти компьютера.

1.5 Вывод

Были рассмотрены способы задания трёхмерных моделей и выбрана объёмная форма задания моделей.

Также были рассмотрены алгоритмы удаления невидимых рёбер и поверхностей:

- алгоритм Робертса;
- алгоритм, использующий z-буфер;
- алгоритм обратной трассировки лучей.

В качестве реализуемого был выбран алгоритм обратной трассировки лучей с глобальной моделью освещения.

2 Конструкторская часть

В данном разделе представлены математические основы алгоритма обратной трассировки лучей, разработка алгоритма обратной трассировки лучей, разработка и обоснование используемых типов и структур данных и разработка структуры программного комплекса.

2.1 Математические основы алгоритма обратной трассировки лучей

2.1.1 Поиск пересечения луча с полигонами

Для поиска пересечения луча с полигонами используется барицентрический тест. Это самый известный тест на пересечение «луч-треугольник». Имея три точки на плоскости, можно выразить любую другую точку через ее барицентрические координаты.

Пусть луч $R(t)$ с началом в точке O и нормализованным вектором направления D определяется как:

$$R(t) = O + tD \tag{2.1}$$

Пусть вершины полигона обозначаются как V_0, V_1, V_2 Тогда, точка $T(u, v)$ в полигоне задаётся выражением:

$$T(u, v) = (1 - u - v)V_0 + uV_1 + vV_2, \tag{2.2}$$

где (u, v) – барицентрические координаты $(u \geq 0, v \geq 0, u + v \leq 1)$

Вычисление пересечения луча $R(t)$ и треугольника эквивалентно решению уравнения $R(t) = T(u, v)$. В этом случае получим:

$$O + tD = (1 - u - v)V_0 + uV_1 + vV_2 \tag{2.3}$$

В матричном виде:

$$\begin{bmatrix} -D & V_1 - V_0, V_2 - V_0 \end{bmatrix} \begin{bmatrix} t \\ u \\ v \end{bmatrix} = O - V_0 \tag{2.4}$$

Это означает, что барицентрические координаты (u, v) и расстояние t от точки испускания луча до точки пересечения луча с полигоном могут быть найдены путём решения СЛАУ, которая написана выше.

Вышесказанное можно рассматривать геометрически как перевод полигона (треугольника) в начало координат и преобразование его в в треугольник с единичными длинами по y и z с направлением луча по оси x. Это показано на рисунке 2.1.

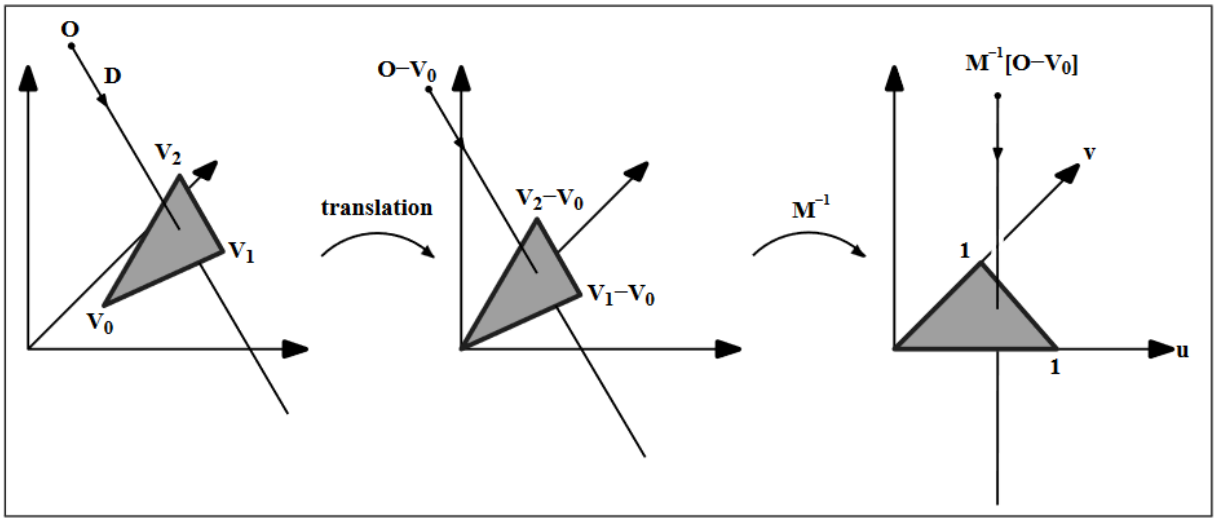

Рисунок 2.1 – Геометрическая интерпретация СЛАУ

Пусть $E_1 = V_1 - V_0, E_2 = V_2 - V_0$ и $T = O - V_0$. Тогда решим 2.4, используя метод Крамера:

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{(D \times E_2) \cdot E_1} \begin{bmatrix} (T \times E_1) \cdot E_2 \\ (D \times E_2) \cdot T \\ (T \times E_1) \cdot D \end{bmatrix} = \frac{1}{P \cdot E_1} \begin{bmatrix} Q \cdot E_2 \\ P \cdot T \\ Q \cdot D \end{bmatrix}, \tag{2.5}$$

где $P = D \times E_2$ и $Q = T \times E_1$.

2.1.2 Поиск нормали к полигонам

Для поиска нормали к полигонам необходимо найти векторное произведение двух векторов, которые лежат на полигоне:

$$N = (V_2 - V_0) \times (V_1 - V_0), \tag{2.6}$$

где V_0, V_1, V_2 – вершины полигона.

2.1.3 Поиск направления преломлённого и отражённого лучей

Для алгоритма обратной трассировки лучей нужно уметь находить отражённый и преломлённый лучи, при этом учитывая модель освещения Уиттеда. Отражённый луч можно найти, зная направление падающего луча и нормаль к поверхности.

Пусть L – направление луча, а n – нормаль к поверхности. Луч можно разбить на две части: L_p которая перпендикулярна нормали, и L_n – параллельна нормали.

Представленная ситуация изображена на рисунке 2.4:

Учитывая свойства скалярного произведения $L_n = n \cdot (n, L)$ и $L_p = L - n \cdot (n, L)$ Так как отражённый луч выражается через разность этих векторов, то отражённый луч выражается по формуле 2.7:

$$R = 2 \cdot n \cdot (n, L) - L \tag{2.7}$$

По закону преломления падающий, преломлённый луч и нормаль к поверхности лежат в одной плоскости. Пусть μ_i – показатели преломления сред, а η_i – углы падения и отражения света соответственно. Применяя закон Снеллиуса, параметры преломлённого луча можно вычислить по формуле 2.8:

$$R = \frac{\mu_1}{\mu_2} L + (\frac{\mu_1}{\mu_2} cos(\eta_1) - cos(\eta_2))n, \tag{2.8}$$

где $cos(\eta_2) = \sqrt{1 - (\frac{\mu_1}{\mu_2})^2 \cdot (1 - cos(\eta_1))^2}$

2.2 Разработка алгоритмов

Для алгоритмов, разработанных автором работы, представлены схемы алгоритмов. Для алгоритма синтеза изображения, использующего в своей основе алгоритмы Z-буфера[2] и Гуро[3] представлена блок-схема.

2.2.1 Алгоритм деформации мышцы

На рисунке 2.2 представлена схема алгоритма деформации мышцы.

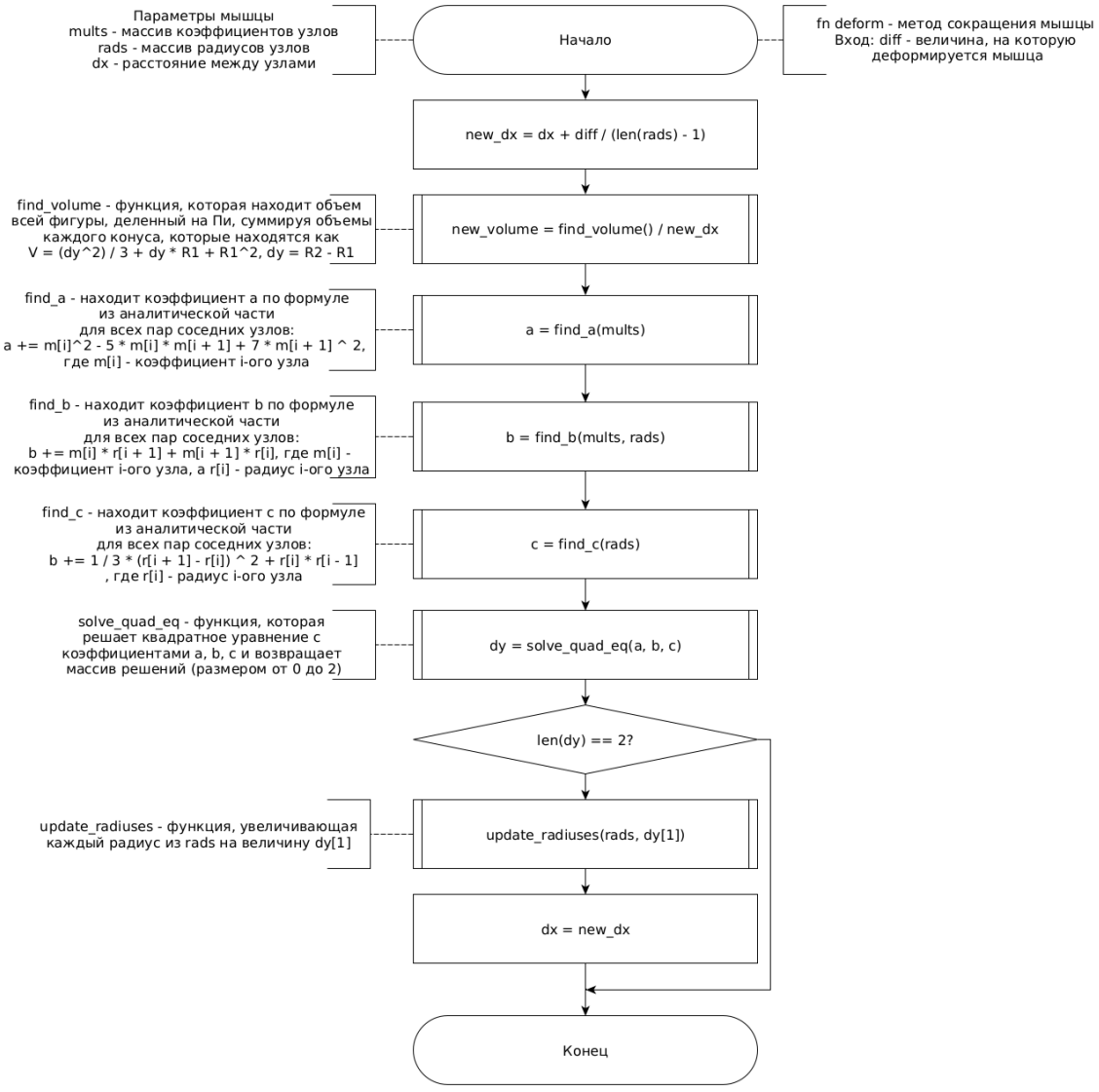

Рисунок 2.2 – Схема алгоритма деформации мышцы

2.2.2 Алгоритм триангуляции мышцы

На рисунке 2.3 представлена схема алгоритма триангуляции мышцы.

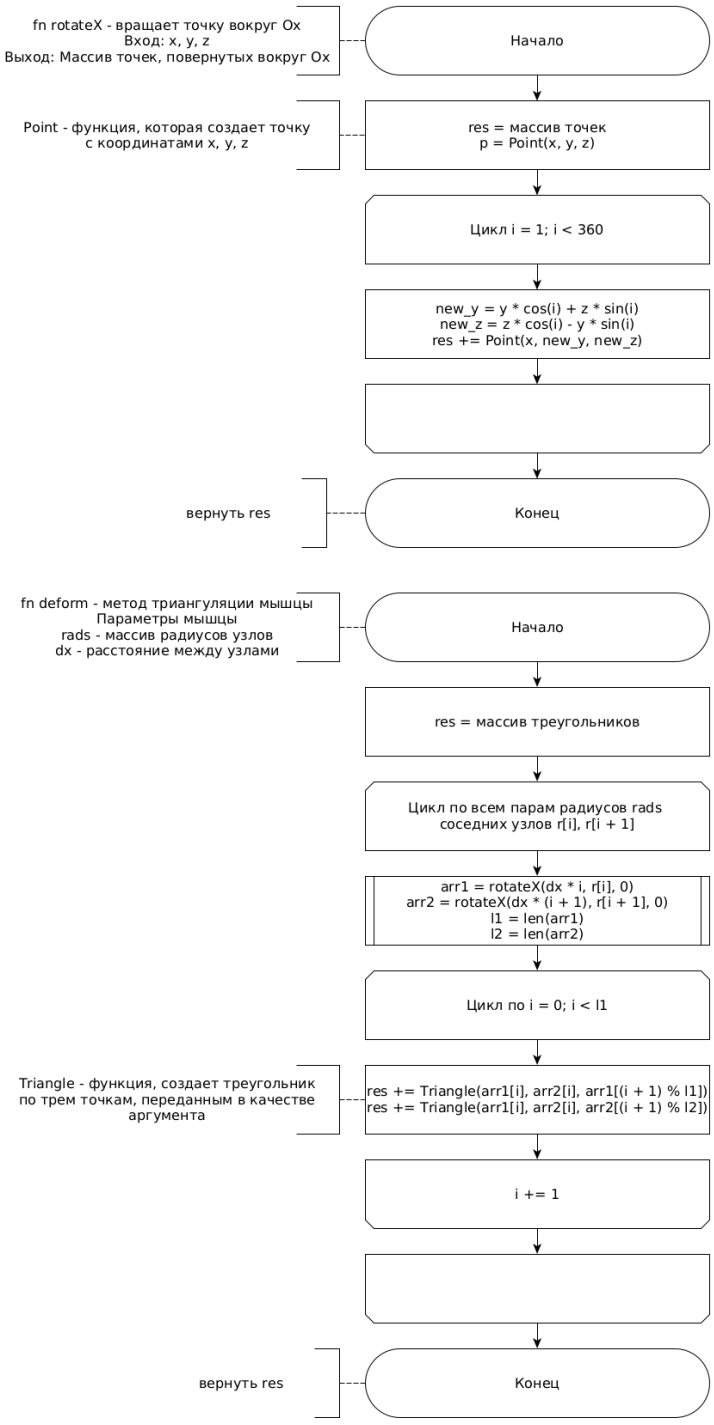

Рисунок 2.3 – Схема алгоритма триангуляции мышцы

2.2.3 Алгоритм синтеза изображения

На рисунке 2.4 представлена блок-схема алгоритма синтеза изображения.

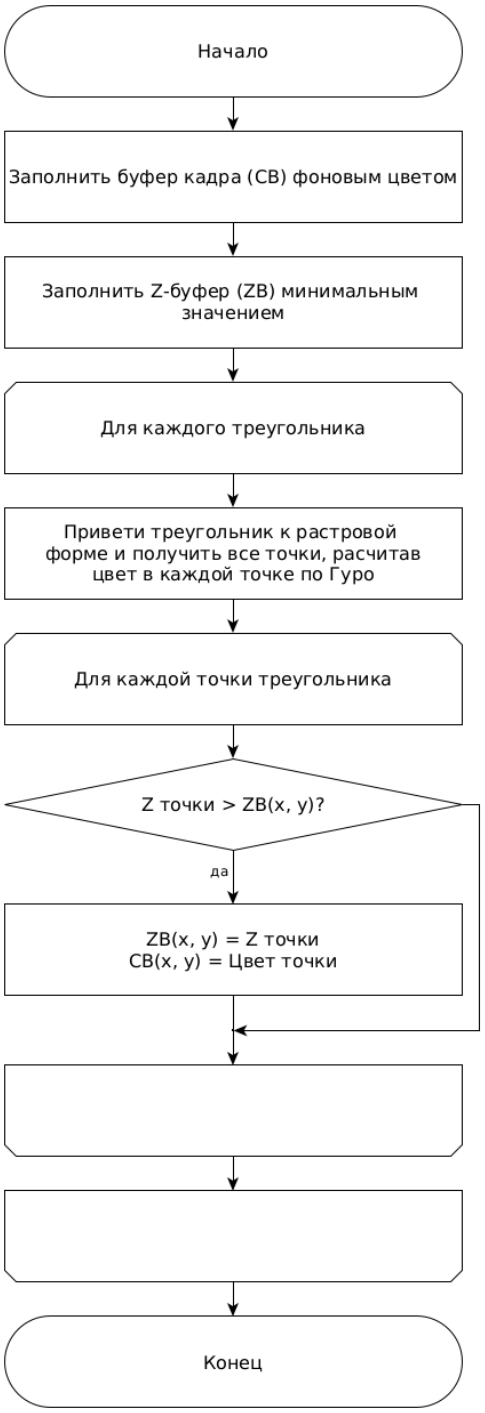

Рисунок 2.4 – Блок-схема алгоритма синтеза изображения

Вывод

В данном разделе были представлены требования к программному обеспечению и разработаны схемы реализуемых алгоритмов.

3 Технологическая часть

В данном разделе представлены средства разработки программного обеспечения, детали реализации и тестирование функций.

3.1 Средства реализации

В качестве языка программирования для разработки программного обеспечения был выбран язык программирования Rust[4]. Данный выбор обусловлен тем, что данный язык предоставляет весь требуемый функционал для решения поставленной задачи, а также обладает связанным с ним пакетным менеджером Cargo[5], который содержит инструменты для тестирования разрабатываемого ПО[6].

Для создания пользовательского интерфейса программного обеспечения была использована библиотека gtk-rs[7]. Данная библиотека содержит в себе объекты, позволяющие напрямую работать с пикселями изображения, а также возможности создания панели управления с кнопками, что позволит в интерактивном режиме управлять изображением.

Для тестирования программного обеспечения были использованы инструменты пакетного менеджера Cargo[5], поставляемого вместе с компилятором языка при стандартном способе установке, описанном на официальном сайте языка[4].

В процессе разработки был использован инструмент RLS[8] (англ. *Rust Language Server*), позволяющий форматировать исходные коды, а также в процессе их написания обнаружить наличие синтаксических ошибок и некоторых логических, таких как, например, нарушение правила владения[9].

В качестве среды разработки был выбран текстовый редактор VIM[10], поддерживающий возможность установки плагинов[11], в том числе для работы с RLS[8].

3.2 Реализация алгоритмов

В листинге ?? представлена структура объекта мышцы, а также реализация методов деформации и триангуляции. В листинге ?? представлена реализация алгоритмов компьютерной графики: z-буфера и Гуро.

Вывод

В данном разделе были рассмотрены средства, с помощью которых было реализовано ПО, а также представлены листинги кода с реализацией объекта мышцы и алгоритмов компьютерной графики.

4 Исследовательская часть

В данном разделе будут приведены примеры работы разработанного программного обеспечения, а также будет поставлен эксперимент, в котором будут сравнены геометрические характеристики разработанной модели с геометрическими характеристиками бицепса реального человека.

4.1 Результаты работы программного обеспечения

В листинге **??** описана конфигурация модели мышцы с рисунков 4.1 и 4.2, на рисунках 4.1 и 4.2 представлен пример модели мышцы в полностью растянутом и сокращённом состояниях соответственно, на рисунке 4.3 представлено окно панели управления.

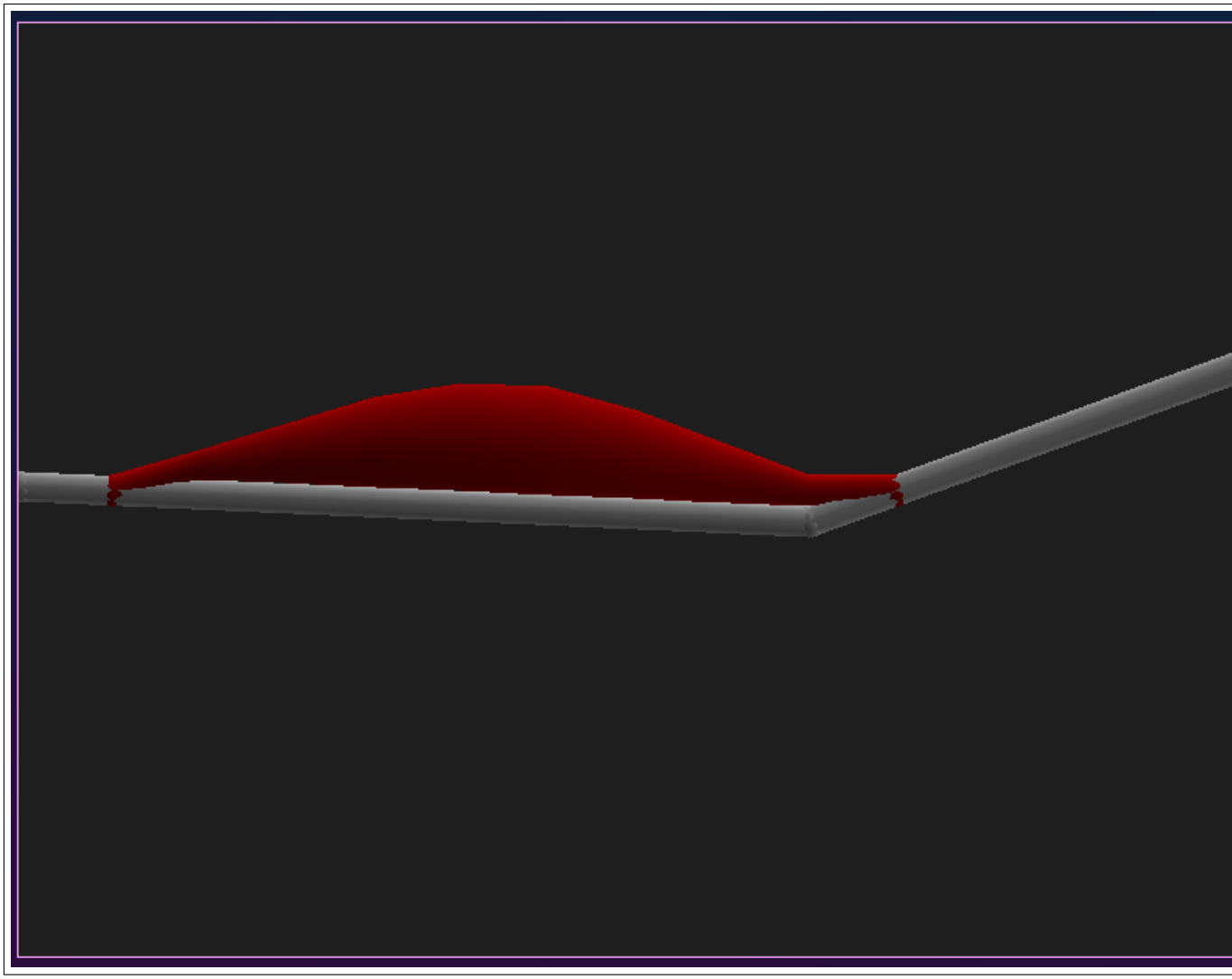

Рисунок 4.1 – Пример модели мышцы в полностью растянутом состоянии.

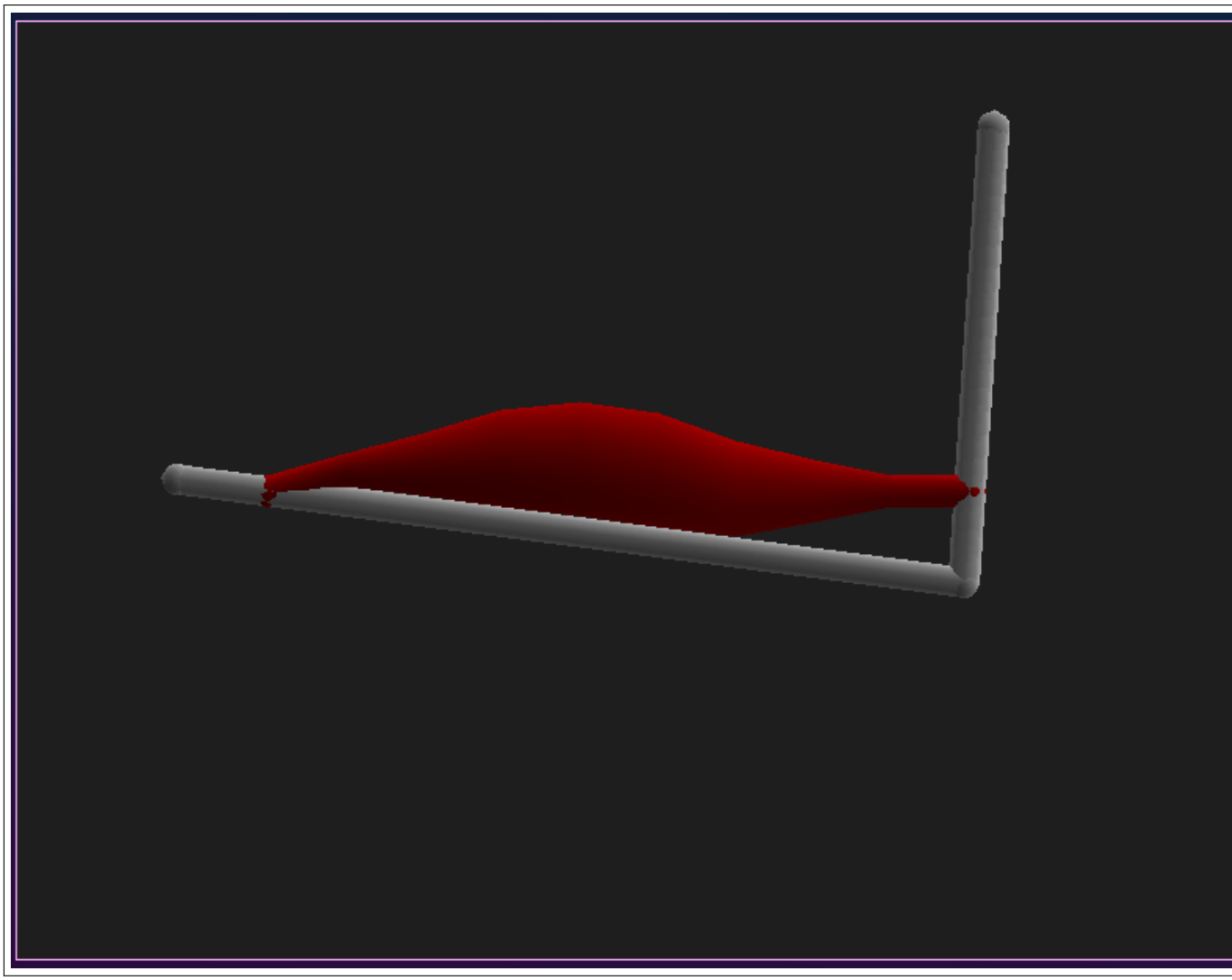

Рисунок 4.2 – Пример модели мышцы в полностью сокращённом состоянии.

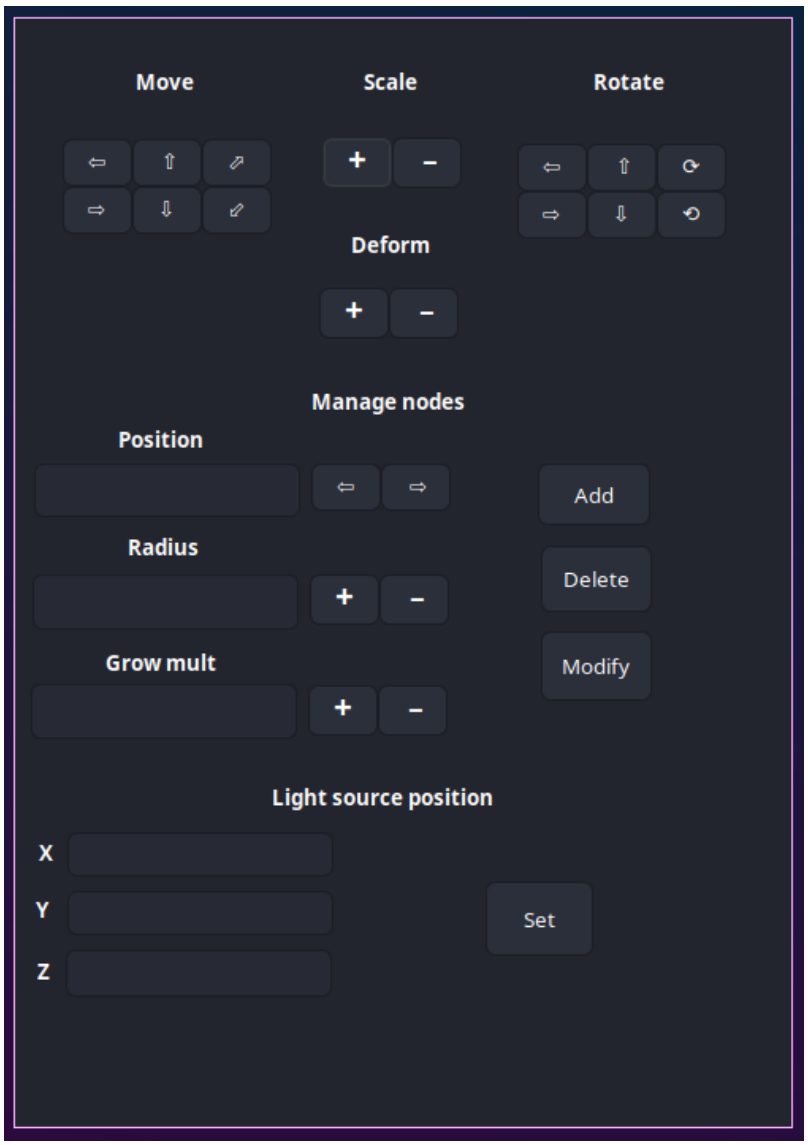

Рисунок 4.3 – Окно панели управления.

4.2 Постановка эксперимента

4.2.1 Цель эксперимента

Цель эксперимента - сравнение геометрических характеристик (радиусов узлов) разработанной модели и реального человеческого бицепса в различных положениях мышцы.

4.2.2 Данные реального бицепса

Данные реального бицепса были взяты из проекта OpenArm 2.0 [12]. Данный проект предоставляет необработанные и сегментированные ультразвуковые сечения плечевой части руки (в том числе бицепса) [13]. Анализ данных может быть произведен с помощью ПО ITK-SNAP [14]. На рисунке 4.4 приведён пример необработанных данных, а на рисунке 4.5 - пример сегментированных данных.

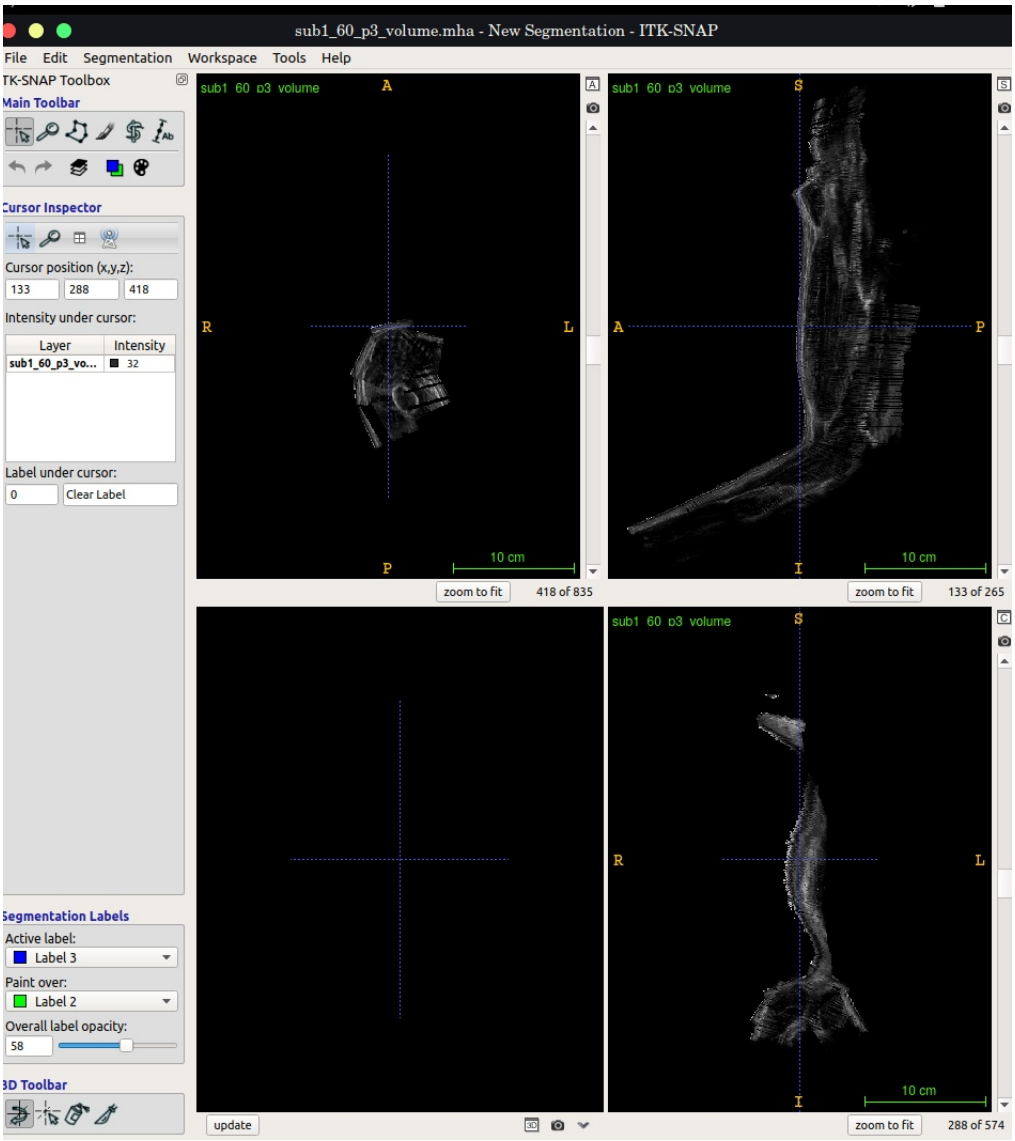

Рисунок 4.4 – Необработанные данные.

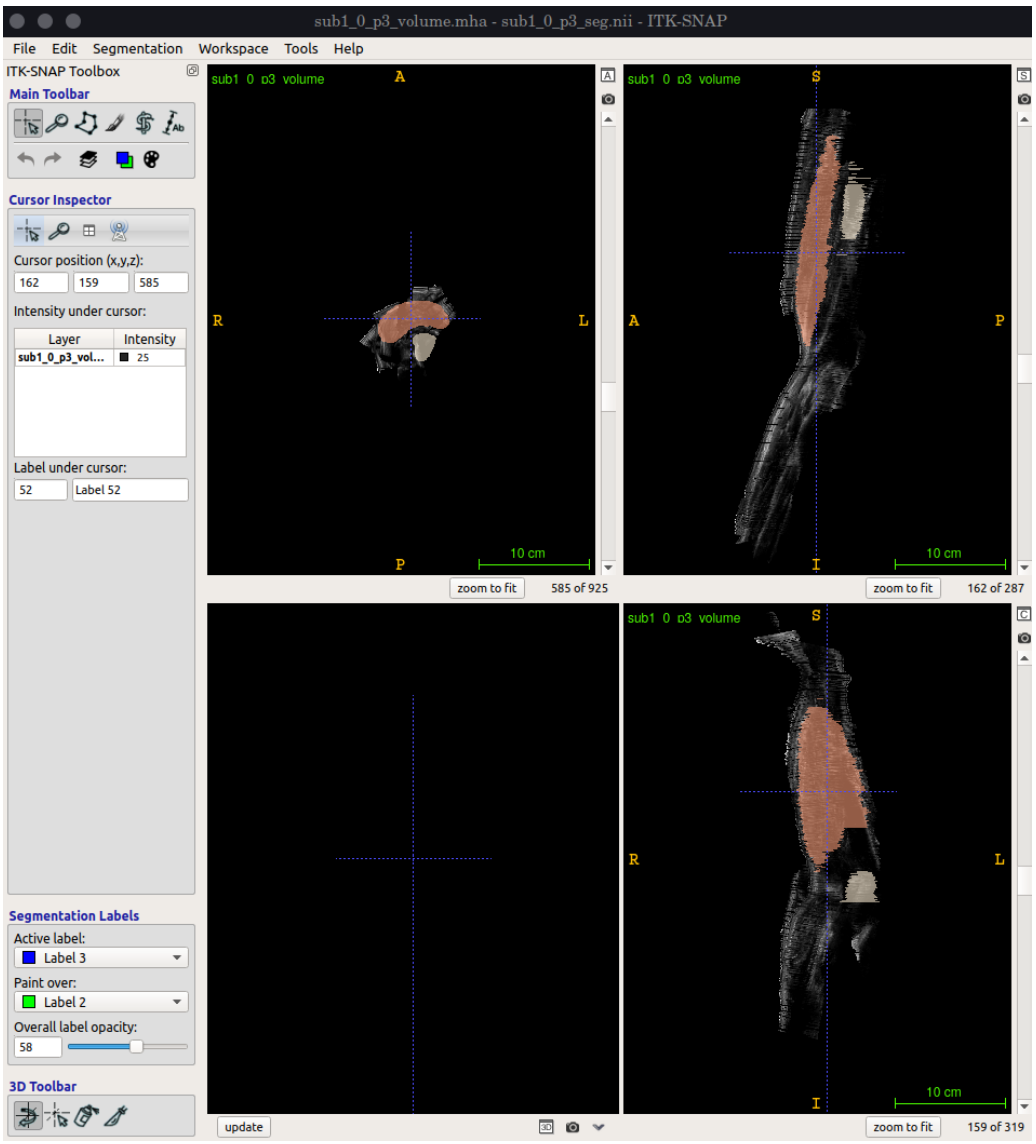

Рисунок 4.5 – Сегментированные данные.

4.2.3 Замеры реального бицепса

На рисунках 4.6 – 4.9 приведены замеры для 9 узлов мышцы в 4 положениях: под углом 0°, 30°, 60° и 90°. Данные замеров сведены в таблицу 4.1.

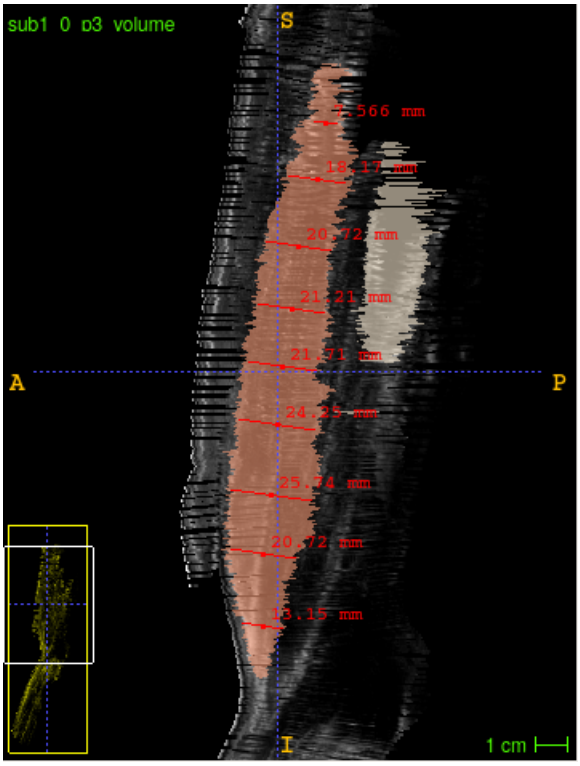

Рисунок 4.6 – Замеры для мышцы в положении 0°.

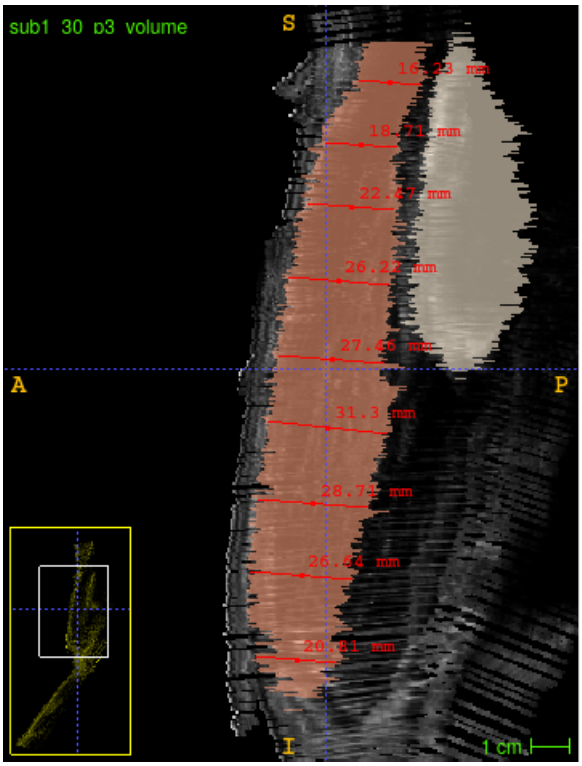

Рисунок 4.7 – Замеры для мышцы в положении 30°.

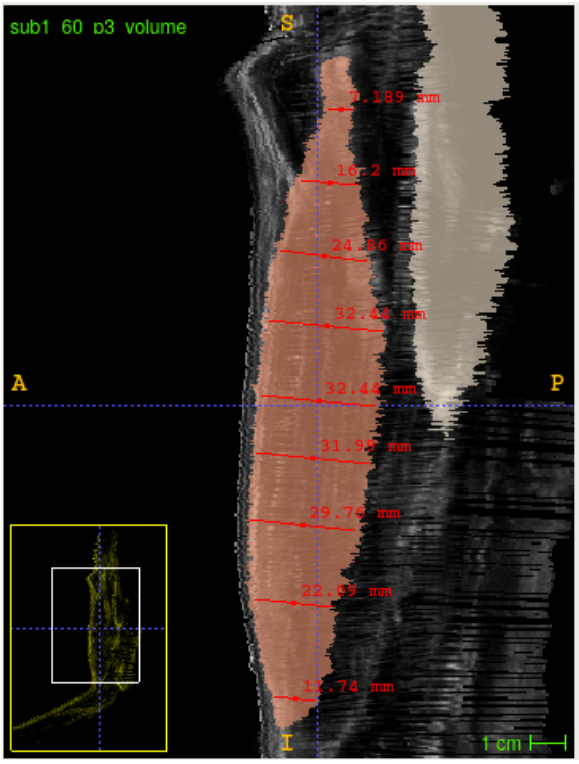

Рисунок 4.8 – Замеры для мышцы в положении 60°.

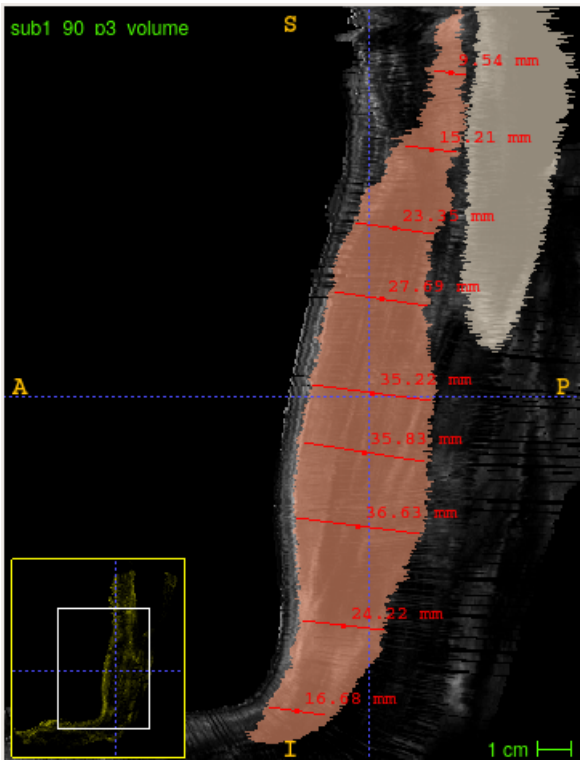

Рисунок 4.9 – Замеры для мышцы в положении 90°.

Номер узла	0°	30°	60°	90°
1	7.6	16.2	7.8	9.5
2	18.2	18.7	16.2	15.2
3	20.7	22.5	24.9	23.4
4	21.2	26.2	32.4	27.7
5	21.7	27.5	32.4	35.2
6	24.3	31.3	32.0	35.8
7	25.7	28.7	29.8	36.6
8	20.7	26.6	22.0	24.2
9	13.2	20.8	12.7	16.7

Таблица 4.1 – Радиусы узлов при различном угле наклона в локте

4.2.4 Замеры разработанной модели

Для измерения геометрических характеристик модели необходимо сперва ее сконфигурировать. Конфигурация будет выполняться исходя из данных реальной мышцы: начальные значения узлов модели пропорциональны начальным значениям узлов реальной мышцы, коэффициенты роста пропорциональны приросту реальной мышцы при переходе из начального в конечное состояние.

Таким образом конфигурация будет иметь вид, описанный в листинге **??**.

Вид модели представлен на рисунках 4.10 – 4.11. Результаты представлены в таблице 4.2.

4.2.5 Сравнение замеров

В таблице 4.3 приведены приросты для всех узлов для реального бицепса (РБ) и модели относительно при переходе из состояния 0° в состояние 90°.

Из таблицы видно, что реализованная модель может сохранять задаваемые пропорции, но ввиду того, что модель сохраняет объем, все приращения получились в 5-6 раз меньше.

Из таблиц 4.1 и 4.2 можно заметить, что приращения модели при сокращении мышцы строго положительные, в то время как с реальным бицепсом

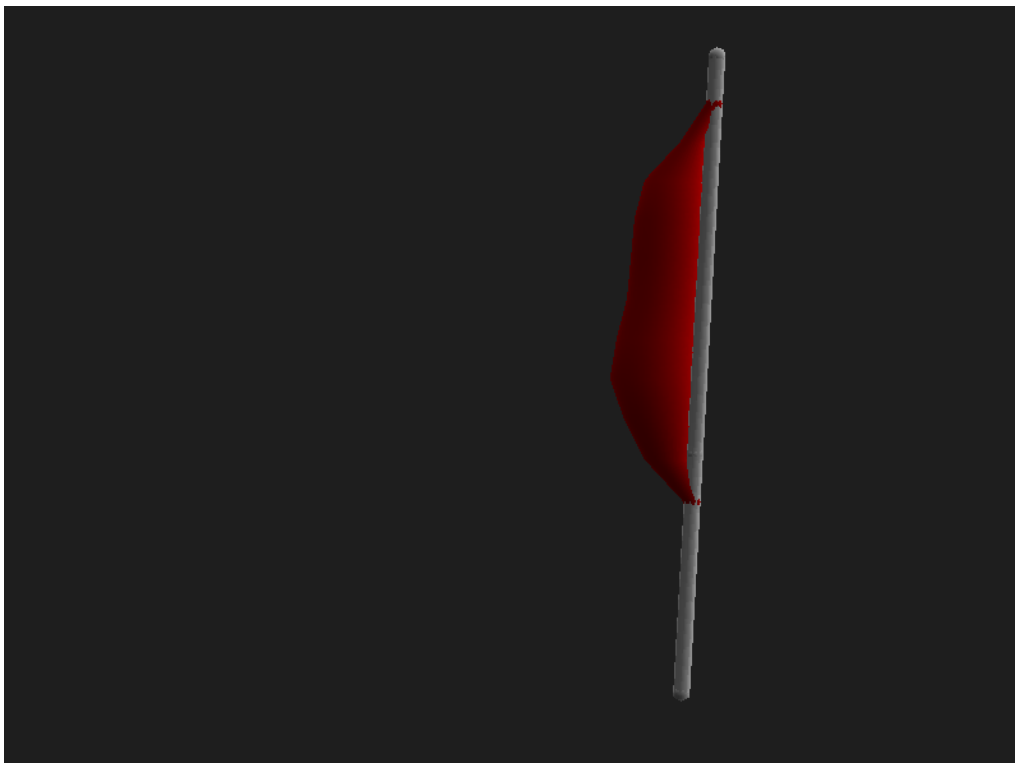

Рисунок 4.10 – Модель в положении 0°.

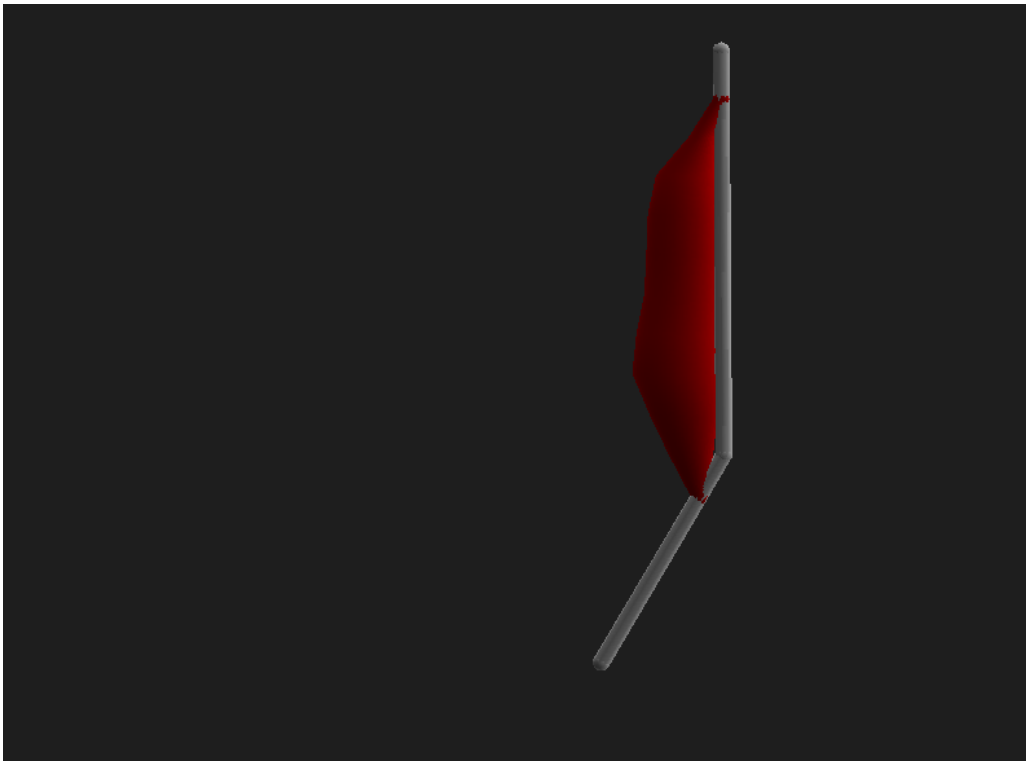

Рисунок 4.11 – Модель в положении 30°.

могут быть как положительные, так и отрицательные.

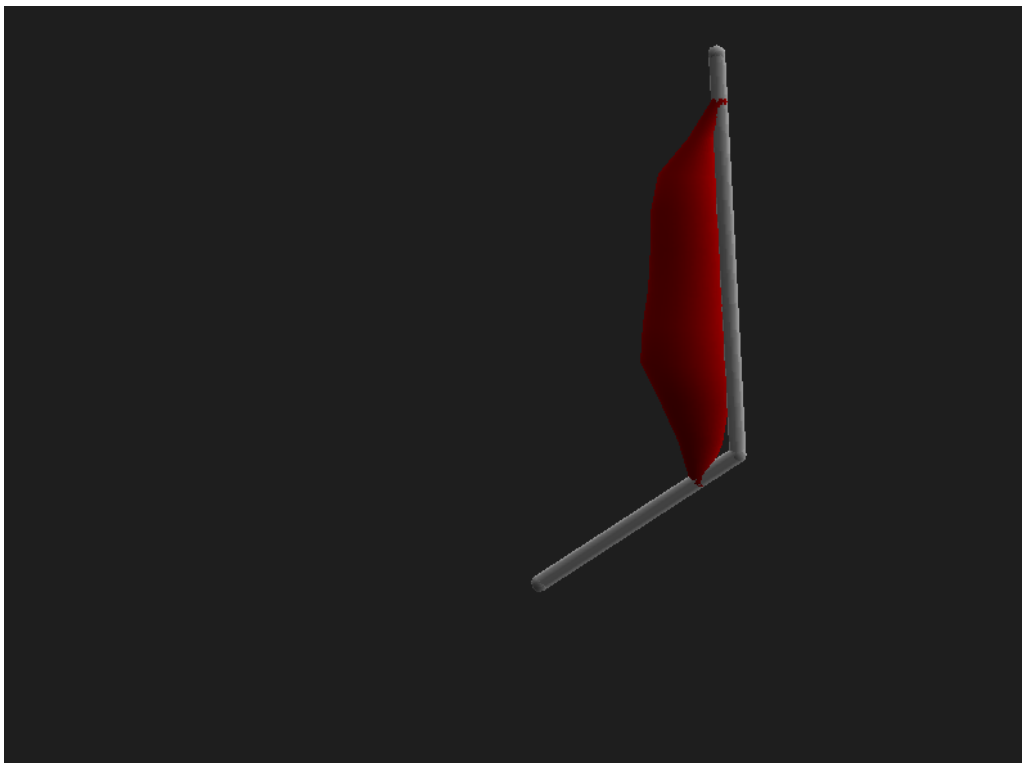

Рисунок 4.12 – Модель в положении 60°.

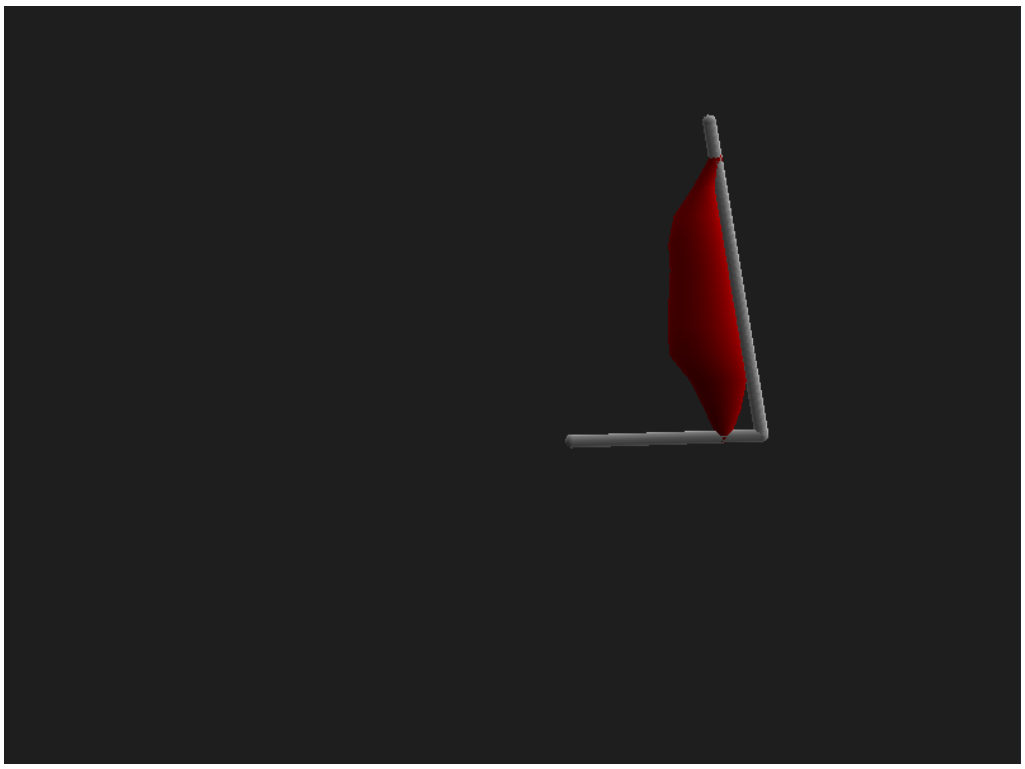

Рисунок 4.13 – Модель в положении 90°.

Вывод

В данном разделе были рассмотрены примеры работы программного обеспечения, а также были вычислены и сравнены радиусы узлов реального би-

Номер узла	0°	30°	60°	90°
1	7.6	7.7	7.8	8.0
2	18.2	18.1	17.9	17.6
3	20.7	20.8	21.0	21.2
4	21.2	21.3	21.7	22.3
5	21.7	22.1	23.0	24.4
6	24.3	24.6	25.4	26.6
7	25.7	26.0	26.7	27.9
8	20.7	20.8	21.0	21.4
9	13.2	13.3	13.5	13.9

Таблица 4.2 – Радиусы узлов модели при различном угле наклона в локте

Номер узла	РБ	модель
1	1.9	0.4
2	-3.0	-0.6
3	2.7	0.5
4	6.5	1.1
5	13.5	2.7
6	11.5	2.3
7	10.9	2.2
8	3.5	0.7
9	3.5	0.7

Таблица 4.3 – Радиусы узлов при различном угле наклона в локте

цепса и сделанной модели. В результате сравнения были получены следующие результаты:

- Модель способна сохранять пропорции приращения, заданные реальной мыщцой.
- Модель ведёт себя монотонно, то есть, например, при сокращении радиусы не перестают расти (или убывать, если коэффициент роста меньше 0), в то время как узел в реальном бицепсе при сокращении может как увеличиться, так и уменьшиться.
- Ввиду ограничения, связанного с постоянством объема, радиусы узлов модели получают прирост в 5-6 раз меньший, нежели узлы реальной мышцы.

Заключение

В ходе курсового проекта было разработано программное обеспечение, которое предоставляет возможности загрузки параметров геометрической модели бицепса на узлах из конфигурационного файла, изменения этих параметров в интерактивном режиме, управления состоянием модели (сокращение и растяжение), а также положением (вращение, перемещение и масштабирование). В процессе выполения данной работы были выполнены следующие задачи:

- формально описана структура моделей мышцы и каркаса;
- рассчитаны формулы деформации геометрической модели с сохранением объема;
- выбраны алгоритмы трехмёрной графики, визуализирующие модель;
- реализованы алгоритмы для визуализации описанных выше объектов.

В процессе исследовательской работы было выяснено, что полученная геометрическая модель имеет ограничения, не позволяющие полностью воспроизвести поведение реального бицепса, однако реализованная модель способна получить схожее поведение: сохранить пропорции с уменьшенной в 5-6 раз амплитудой роста/уменьшения (из-за ограничения, связанного с постоянством объема), а также с требуемой степенью детализации получить некоторое статическое (изначальное) состояние.

Литература

[1] Метод прямой и обратной трассировки [Электронный ресурс]. — Режим доступа: https://cyberleninka.ru/article/n/metod-pryamoy-i-obratnoy-trassirovki (дата обращения: 17.10.2022).

[2] Алгоритм Z-буфера. — Режим доступа: http://compgraph.tpu.ru/zbuffer.htm (дата обращения: 21.08.2020).

[3] Модели затенения. Плоская модель. Затенение по Гуро и Фонгу. — Режим доступа: https://compgraphics.info/3D/lighting/shading_model.php (дата обращения: 18.06.2020).

[4] Rust Programming Language [Электронный ресурс]. — Режим доступа: https://doc.rust-lang.org/std/index.html (дата обращения: 20.07.2020). — 2017.

[5] The Cargo Book. — Режим доступа: https://doc.rust-lang.org/cargo/ (дата обращения: 21.07.2020).

[6] Документация по ЯП Rust: бенчмарки [Электронный ресурс]. — Режим доступа: https://doc.rust-lang.org/1.7.0/book/benchmark-tests.html (дата обращения: 21.09.2020).

[7] Gtk-rs. — Режим доступа: https://gtk-rs.org/ (дата обращения: 21.07.2020).

[8] Rust Language Server (RLS). — Режим доступа: https://github.com/rust-lang/rls (дата обращения: 21.07.2020).

[9] Rustbook. What is Ownership? — Режим доступа: https://doc.rust-lang.org/book/ch04-01-what-is-ownership.html (дата обращения: 20.07.2020).

[10] VIM the editor. — Режим доступа: https://www.vim.org/ (дата обращения: 20.07.2020).

[11] VimAwesome. — Режим доступа: https://vimawesome.com/ (дата обращения: 20.07.2020).

[12] OpenArm 2.0. — Режим доступа: https://berkeley.app.box.com/v/openarm-2p0-data/file/445757315449 (дата обращения: 11.12.2020).

[13] OpenArm 1.0. — Режим доступа: https://berkeley.app.box.com/s/mj9yano1umbtbi1aj0l3b7y2pm864wnb/file/692415066210 (дата обращения: 11.12.2020).

[14] ITK-SNAP. — Режим доступа: http://www.itksnap.org/pmwiki/pmwiki.php (дата обращения: 11.12.2020).