

СОДЕРЖАНИЕ

Введение	4
1 Аналитический раздел	5
1.1 Предметная область	5
1.2 Способы моделирования мышечных сокращений	7
1.2.1 Использование деформируемых эллипсоидов для аппроксимации мышц	8
1.2.2 Обобщённая цилиндрическая модель	8
1.2.3 Mass-spring System	9
1.2.4 Метод конечных элементов	10
1.2.5 Метод конечных объёмов	11
1.3 Выбор метода представления объекта в пространстве	12
1.4 Выбор метода морфинга	14
1.5 Анализ методов удаления невидимых граней	16
1.6 Анализ моделей освещения	19
1.7 Анализ методов закраски	19
1.8 Вывод	20
2 Конструкторский раздел	22
2.1 Общий алгоритм работы программы	22
2.2 Реализуемые алгоритмы алгоритмы	25
2.3 Вывод	29
3 Технологический раздел	30
3.1 Средства реализации	30
3.2 Описание структуры программы	30
3.3 Листинг кода	33
3.4 Интерфейс программы	38

3.5 Вывод.....	40
4 Исследовательский раздел.....	41
Заключение	43
Список использованных источников.....	44

ВВЕДЕНИЕ

Целью данной производственной практики является разработка программы, моделирующей мышечные сокращения. Для достижения поставленной цели необходимо решить следующие задачи:

- выбрать алгоритмы машинной графики, с помощью которых будет визуализирована трёхмерная сцена и построена анимация;
- выбрать метод для визуализации мышц;
- спроектировать архитектуру программы и структуры данных для хранения модели;
- реализовать выбранные алгоритмы.

1 Аналитический раздел

В этом разделе будут рассмотрены предметная область и основные алгоритмы, необходимые для создания реалистичного изображения, произведён и обоснован выбор алгоритмов для реализации в проекте.

1.1 Предметная область

Мышцы – органы, которые состоят из мышечной ткани, способные сокращаться под влиянием нервных импульсов. Они составляют примерно 40% веса тела человека [1]. По типу строения тканей их можно классифицировать по трём типам: скелетные, гладкие и сердечную [2].

- Скелетные мышцы – образуют опорно-двигательный аппарат, способный по желанию человека сокращаться.

- Гладкие мышцы – образуют внутренние органы, кожу и кровеносные сосуды. Играют важную роль в процессах, не контролируемых человеком, например при сокращении зрачка.

- Сердечная мышца – не подконтрольна сознанию человека, ее сокращения стимулируются вегетативной нервной системой.

В данной работе будут рассмотрены в частности скелетные мышцы. Каждая мышца имеет среднюю часть, способную сокращаться и называемую брюшком, и сухожильные концы, не обладающие сократимостью и служащие для прикрепления мышц.

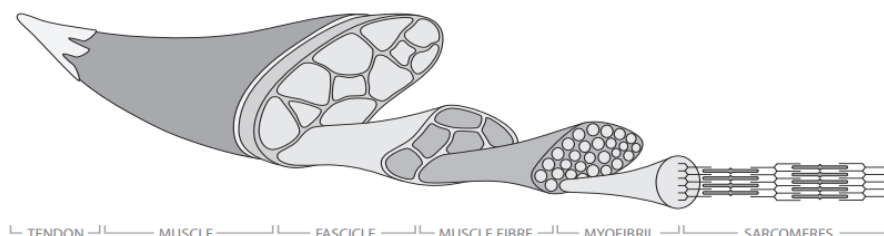


Рисунок 1.1 — Строение мышцы

Мышечное брюшко окружено плотным и прочным покровом, который называется фасцией. Внутри же содержатся различной толщины пучки

мышечных волокон, каждое из которых образованно параллельными миофибриллами. Миофибрилы являются нитевидными образованиями, состоящими из саркомеров.

Все соединительные образования мышцы с мышечного брюшка переходят на сухожильные концы. Они состоят из плотной волокнистой соединительной ткани, коллагеновые волокна которой лежат между мышечными волокнами, плотно соединяясь с их сарколеммой, оболочкой мышечных волокон.

Значительное влияние на работу мышц оказывает направление их волокон. По этому признаку выделяют: мышцы с параллельными, поперечными и косыми волокнами. Косые в свою очередь делятся на одноперистые, если присоединяются к сухожилию с одной стороны, и двуперистые, если с двух сторон. Особенности в строении определяют то, насколько подвижными являются мышцы и какую силу они могут производить.

При сокращении центральная нервная система подаёт сигнал в мышцы, в результате которого нити саркомеров скользят друг вокруг друга, что приводит к укорачиванию саркомера [3]. Скольжение нитей вызывает мышечное напряжение, что является основным вкладом саркомера, так как это действие даёт мышцам их физическую силу. В следствие сокращения форма мышцы меняется – при изотоническом сокращении, когда длина уменьшается, а сама брюшко расширяется, что необходимо корректно отобразить в программе.

Задача программы и аналоги

Задачей программы является представление визуальной модели мышц, которая позволила бы наблюдать за процессом их сокращения, благодаря чему было бы проще изучить их строение и основные принципы работы.

Следует отметить, что существуют приложения, которые позволяют изучать строение мышц. Некоторые из них также моделируют внутренние органы, однако в них возможно наблюдать мышцы только в статическом положении. Это отличает их от разрабатываемого приложения, которое позволит изучать процесс сокращения.

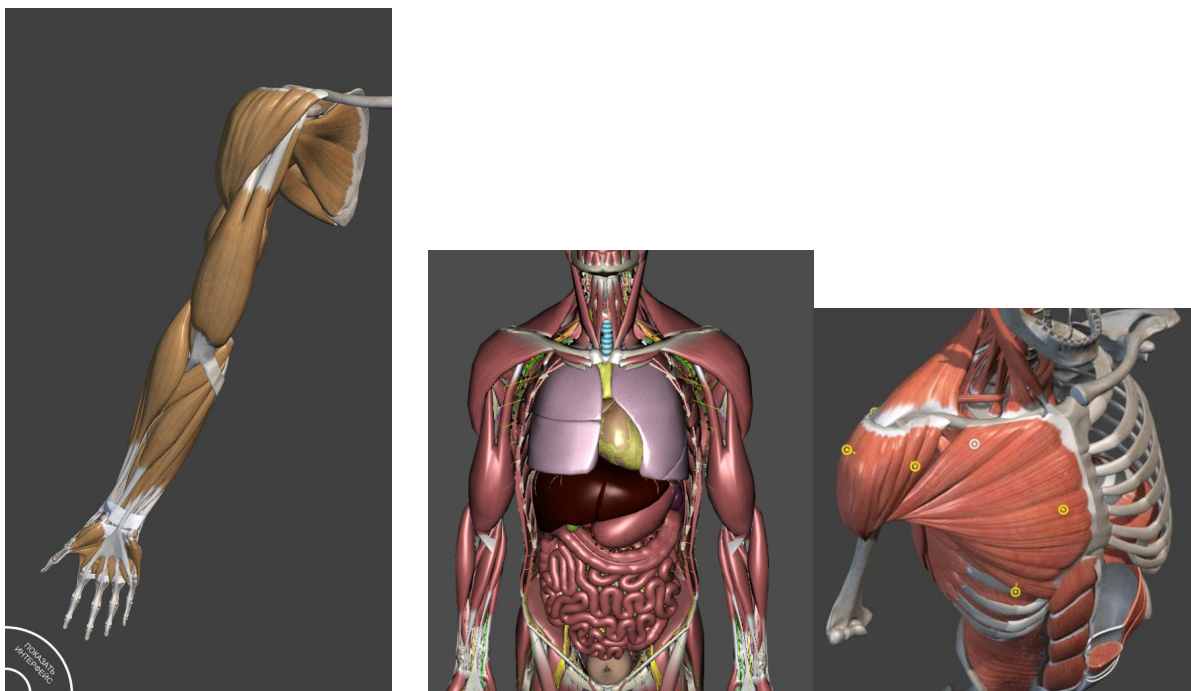


Рисунок 1.2 — Примеры анатомических атласов

1.2 Способы моделирования мышечных сокращений

Существуют различные способы представления мышечных сокращений. В зависимости от принципов, лежащих в основе каждого из методов, их можно разделить на: геометрически-основанные и физически-основанные [4]. В геометрически-основанных подходах большее внимание уделяется анимации расширения и укорачивания, а физический аспект сокращения не учитывается. Они успешно моделируют простые мышцы, но им недостаёт реализма в физиологическом или биомеханическом аспектах. В противовес к таким подходам ставятся физически-обоснованные подходы, в которых рассматриваются сложные проблемы, включающие в себя динамику мышц и свойства тканей. При моделировании физически-корректной модели мышц решают две проблемы: определение силы сокращения и представление изменения мышечной геометрии.

1.2.1 Использование деформируемых эллипсоидов для аппроксимации мышц

Идея данного метода заключается в использовании параметрических эллипсоидов в качестве базовых примитивов для моделирования мышц. Три главные оси отрегулированы так, чтобы представлять выпуклость мышечного брюшка, в то время как объём сохраняется за счёт использования определённых соотношений между этими тремя осями [5]. Данный метод относится к геометрически-основанным. Чтобы симулировать изометрические сокращения мышц, вводится параметр напряжённости t , в связи с чем соотношение между шириной и высотой эллипсоида рассчитывается по формуле:

$$r = (1 - t)r_n + ktr_n = (1 - t + kt)r_n \quad (1.1)$$

где k - параметр контроля напряжённости, который регулирует, насколько выпукла мышца, $r_n = \frac{a_n}{b_n}$ - соотношение ширины к высоте в расслабленном состоянии.

Для представления более сложных форм используется группа эллипсоидов, точки начала и конца которых расположены на сплайне, или же прямой путь между концами мышцы заменяется кривой Безье, представляющей направление силы, и эллипсоидами разных размеров, расположенных вдоль этой кривой [5]

Преимуществами данного метода является возможность моделирования изометрического и изотонического сокращения, а также представления различных форм мышц. Недостатком – то, что он не моделирует физическую силу мышц, что, однако, не влияет на визуальную реалистичность представления

1.2.2 Обобщённая цилиндрическая модель

Аналогичным подходом является представление мышц, основанное на деформируемых цилиндрах, он также является геометрически-основанным. Ось цилиндра - кривая, вдоль которой расположены эллиптические срезы, на основе которых строится полигональный меш [6]. Подразумевается, что

каждый радиус, позиция и наклон каждого эллипса могут быть изменены, что позволяет формировать мышцы сложной формы.

При сокращении ширина и толщина внутренних срезов увеличивается в соотношении $\sqrt{l_n/l}$, где l_n - длина в расслабленном состоянии, а l - текущая длина. За счёт этого поперечная область расширяется, если мышцы становятся короче, и сужается, если удлиняются.

Другой способ – задание мышцы, используя модифицированный обобщенный цилиндр, определяемый с помощью кривой толщины $C_T(t)$ заданной вдоль кривой наклона $C_S(t)$ [7]. $C_S(t)$ это трёхмерный сплайн, представляющий профиль мышцы. Он определяется двумя точками - источником О и целью I. $C_T(t)$ - одномерная функция, представляющая толщину мышцы от t . Срез мышцы в t определяется эллипсом, расположенным вдоль $C_S(t)$, толщина которого получена из $C_T(t)$. Чтобы избежать пересечения секций и упростить сохранение объёма, поперечное сечение всегда расположено перпендикулярно линии действия, лежащей на главной оси мышечной формы.

Его преимущества – возможность моделирования изотонических и изометрических сокращений, а также изогнутых мышц, а недостатком – то, что он как и предыдущий не учитывает моделирует силу сокращения.

1.2.3 Mass-spring System

Данный метод относится к физически-основанным методам. Объект моделируется набором точечных масс, связанных между собой невесомыми пружинами. Такая модель может быть расширена при добавлении различных типов сил, применяемых к пружине, например угловой или силы сжатия. Линии, называемые линиями действия, используются для вычисления силы, прилагаемой к костям при сокращении [8]. В зависимости от формы и сложности их строения, мышцы могут быть представлены одной или несколькими линиями действия, кроме того каждая линия может быть полилинией.

Такой подход разделяет модель на 2 уровня: линию действия, задаваемую точками начала и конца, и форму мышцы. Важной частью формирования данной модели является установление соответствия между точками массы и

формой мышц. Для этого каждая точка должна быть расположена между двумя соседними в горизонтальном и в вертикальном смыслах. Чтобы модель мышц удовлетворяла этим требованиям, необходимо изменять форму ее меша, из-за чего происходят потери в качестве. Кроме того, такой подход применим только к мышцам, обладающим вытянутой формой [8].

Преимуществом модели, использующей точки масс является возможность рассчитать силу сокращения для каждой точки, используя формулу, в которой сила упругости определяется суммой сил, с которыми пружины, связывающие ее с 4 соседями, притягиваются с силой:

$$f_{result}(x_i) = f_{elasticity}(x_i) + f_{curvature}(x_i) + f_{constraints}(x_i) \quad (1.2)$$

$$f_{elasticity} = \sum_{j=0}^3 f_{spring_j}(x_i) = \sum_{j=0}^3 -ks(x_i - x_{rest}) \quad (1.3)$$

где ks - коэффициент упругости пружины, x_i - экстремум её колебания, x_{rest} - расслабленное положение этого экстремума. Для формирования искривлённости мышцы вводятся дополнительные пружины, соединяющие не соседние точки.

Преимущества метода – моделирование физической силы мышц, однако он моделирует только изотоническое сокращения вытянутых мышц.

1.2.4 Метод конечных элементов

Является физически-основанным методом. В его основе лежит разбиение тела на набор конечных элементов, например гексаэдров или тетраэдров. Перемещения и положения в элементе аппроксимированы с помощью интерполяционных функций:

$$\Phi(x) \approx \sum_i h_i(x) \Phi_i \quad (1.4)$$

где h_i - интерполяционная функция для элемента, содержащего x и Φ_i - скалярный вес, сопоставленный с h_i . Такой подход часто применяется для представления твёрдых упругих тел.

Одним из способов применения метода конечных элементов является модель, в которой кроме полигонального представления мышц, применяются

двадцати-вершинные блоки, являющиеся "двигателем" модели [9]. После чего для сетки каждого блока формируются динамические уравнения равновесия. Механическая модель мышц используется для приложения сил к вершинам блоков, после чего их форма динамически меняется, что в свою очередь оказывает влияние на полигоны. Для двадцати-вершинного блока уравнение равновесия имеет вид:

$$M\ddot{u} + C\dot{u} + Ku = R(t) \quad (1.5)$$

Для тела, имеющего n узловых точек, u - это вектор размерности $3n$, отображающий расположение вершин, M , C и K матрицы размерности $3n \times 3n$, описывающие массу, затухание и жёсткость между точками внутри тела, и R - это вектор сил, прилагаемых к каждой вершине, размерностью $3n$.

Чтобы симулировать мышечное сокращение, в вершинах добавляются генераторы силы, которые действуют вдоль продольного направления мышц. В двадцати-вершинном блоке расположено 8 таких генераторов.

Преимущества данной модели – то, что он учитывает физическую природу мышц, может моделировать изометрическое и изотоническое сокращение мышц произвольной формы, а к недостатки – необходимость рассчитывать воздействие "двигателей" и устанавливать соответствие между вершинами блоков и геометрическим представлением мышцы.

1.2.5 Метод конечных объёмов

Аналогично предыдущему методу метод конечных объёмов – физически-основанный метод, который разбивает модель на множество элементов. Однако для расчёта прилагаемых сил объёмные интегралы трансформируют в интегралы по площади, используя формулу Гаусса-Остроградского. Эти условия затем расцениваются как потоки на поверхности каждого конечного объёма. Эта модель используется для представления деформации скелетных мышц, и считается, что она более производительна и обладает меньшим потреблением памяти, чем метод конечных элементов [10].

Вывод было решено использовать модель, использующую деформируемые эллипсоиды для аппроксимации мышцы, так как она позволяет визуализировать мышцы различных форм и размеров, учитывая изометрическое сокращение мышц.

1.3 Выбор метода представления объекта в пространстве

Выделяют три основных типа моделей: каркасные, граничные и сплошные [11]. Каркасная конструкция в виде проволоочной сетки является простейшим способом передачи формы объёмного тела. Проволоочные модели дают возможность быстро визуализировать низко детализированные объекты.

Граничная модель представляет объект системой элементов, создающих его границы. Для задания границ могут использоваться аналитическая и векторная полигональная модели [12]. Первая - это такая модель, в которой для описания поверхности используются математические формулы. Например в виде функций двух аргументов $z = f(x, y)$ или уравнением $F(x, y, z) = 0$.

Преимущества параметрического описания - легко описывать поверхности, которые отвечают неоднозначным функциям, лёгкость процедуры расчёта координат каждой точки поверхности, нормали; небольшой объём информации для описания достаточно сложных форм.

К недостаткам относятся следующие: невозможность в большинстве случаев применения данной формы непосредственно для построения изображения; некоторые сложные формулы описания могут медленно вычисляться на компьютере [12].

Другим способом задания пространственных объектов является векторная полигональная модель. Ее основными элементами являются: вершины, отрезки прямых, полилинии, полигоны, полигональные поверхности. Векторная полигональная модель считается наиболее распространённой в современных системах трёхмерной графики. Положительными чертами векторной полигональной модели являются:

- удобство масштабирования объектов, диапазон которого определяется точностью аппроксимации;

- небольшой объём данных для описания простых поверхностей, аппроксимируемых плоскими гранями;

- необходимость вычислять только координаты вершин при преобразованиях систем координат или перемещении объектов.

Недостатки:

- аппроксимация плоскими гранями приводит к погрешности моделирования;

- сложные алгоритмы выполнения топологических операций, таких, например, как разрезы.

Сплошная модель включает в объект как граничные, так и внутренние точки. Это позволяет представлять сложный объект в виде композиции простых составляющих - кубов, сфер, конус и т.д. Простейшая декомпозиция заключается в разбиении пространства на кубические или сферические ячейки, называемые вокселями, и установления состояния каждой ячейки - свободна она или занята объёмом тела. Преимущества такой модели:

- позволяет достаточно просто описывать сложные объекты и сцены;
- простое выполнение топологических операций над отдельными объектами и сценой в целом.

Недостатки:

- большое количество информации, необходимой для представления объемных данных;

- значительные затраты памяти ограничивают разрешающую способность, точность моделирования;

- при увеличении или уменьшении изображения возникают проблемы, например при увеличении ухудшается разрешающая способность.

Вывод: выбирая между методами моделирования, было решено использовать полигональный, так как он является более экономичным по памяти, чем воксели, кроме того для анимации достаточно только внешнего строения

мышц. Также такое представление позволяет легко изменять форму модели, что положительно влияет на производительность, при анимации сокращения в процессе которого форма брюшка меняется, сужаясь или расширяясь.

1.4 Выбор метода морфинга

Морфинг - это интерполяционная техника, используемая для создания плавной трансформации одного изображения в другое [13].

Существуют двумерные техники морфинга, их преимущество – отсутствие необходимости повторной генерации изображения, однако они не учитывают изменение видимости, освещения и точки обзора [13], в связи с чем не подходят для визуализации мышечных сокращений, при которых положение и форма мышц изменяется.

К изображениям, сгенерированным на основе трёхмерных моделей, существует альтернатива морфингу самого изображения: трёхмерный морфинг, который формирует промежуточные модели на основе заданных, они затем используются, чтобы произвести последовательность изображений. Преимуществами такого подхода являются:

- промежуточные формы не зависят от точки обзора и свойств освещения;
- учитывает изменение видимости.

Трёхмерные техники могут применяться к моделям, заданным объёмами или гранично описанными – мешами. Меш M можно описать парой (K, V) , где K – это набор, отображающий связь вершин, рёбер и граней и $V = (v_1, \dots, v_n)$ – описывает геометрическое положение вершин в R^d [14]. Обычно, морфинг применяются к двум заданным мешам $M_0 = (K_0, V_0)$ и $M_1 = (K_1, V_1)$. Целью является построение группы мешей $M(t) = (K, V(t)), t \in [0, 1]$, генерация ее обычно выполняется за три последовательных шага [14]:

1. Поиск соответствия между мешами. А именно сопоставление вершин, лежащих в исходной фигуре, с вершинами целевой фигуры.

2. Формирование новой, последовательной связности K для каждой вершины на основе геометрических позиций $V(0), V(1)$. Традиционный подход

к этой проблеме заключается в создании надмножества K_0 и K_1 . Однако, в некоторых случаях рещинг может быть полезен в случае, когда исходные меши обладают разными разрешениями.

3. Создание путей $V(t), t \in]0,1[$ для вершин. Данный шаг обладает некоторыми ограничениям: в большинстве случаев ожидается, что пути будут гладкими, а форма будет сохранять целостность. Самый простой способ, получить такие пути - это линейная интерполяция координат вершин. При заданном параметре t координаты интерполированной формы вычисляются как:

$$V(t) = (1 - t)V(0) + tV(1) \quad (1.6)$$

Такая интерполяция даст хорошие результаты, если формы одинаково ориентированны и в некоторой степени похожи.

Для определения, в какое положение должен перейти тот или иной полигон, можно использовать, например, методы ближайшего соседа и сортировки рёбер [15].

Метод ближайшего соседа Его идея заключается в поиске наименьшего расстояния между треугольниками, но вместо трёхмерного пути, который прошли бы центры этих рёбер при перемещении, рассматриваются возможные перестановки вершин и из них выбирается наименьшее. Так, для двух треугольников возможно 6 перестановок, но 3 из них инвертируют нормаль к полигону, в связи с чем рассматриваются 3 оставшиеся, из них выбирается та, которая минимизирует путь, необходимый для перемещения из одного набора вершин в другой. Однако такой метод не гарантирует, что все сопоставления будут уникальными.

Метод сортировки рёбер – каждый треугольник рассматривается как узел а каждое сопоставление между двумя треугольниками - ребро. И поскольку определение соответствия выполняется между двумя множествами источников и целей, проблема сводится к формированию двудольного графа, который соединяет все вершины, и каждую только один раз. В таком случае алгоритм состоит из следующих шагов:

- Сформировать полный взвешенный биграф, в котором вес - расстояние между двумя полигонами, как в методе ближайшего соседа.
- Упорядочить все ребра по весу.
- Для каждого ребра, в сортированном списке, если обе вершины свободны, то установить соответствие, иначе выбрать следующее.

В результате для каждого полигона будет однозначно установлен полигон, в который он должен перейти.

Вывод: было решено использовать трёхмерный морфинг на основе мешей, с использованием метода сортировки рёбер, который устанавливает однозначное соответствие, однако требует дополнительных вычислений для сортировки рёбер.

1.5 Анализ методов удаления невидимых граней

Одним из этапов построения реалистичного изображения является удаление невидимых граней, для этого существуют различные алгоритмы.

Алгоритм Роберста работает в объектном пространстве с выпуклыми телами. Если имеются невыпуклые тела, то их сначала необходимо разбить на выпуклые [16]. В алгоритме в первую очередь удаляются те ребра и грани, которые экранируются самим телом, затем каждое из видимых рёбер каждого тела сравнивается с каждым из оставшихся тел для определения того, какая его часть или части экранируются другими телами.

Недостатком алгоритма Роберста является то, что теоретически его вычислительная ёмкость растёт как квадрат числа объектов. Однако существуют модификации алгоритма с использованием приоритетной сортировки вдоль оси z и простых габаритных или минимаксных тестов [16], которые позволяют свести вычислительную сложность к почти линейной зависимости от числа объектов.

Алгоритм Варнока работает в пространстве изображений. Идея алгоритма заключается в том, что рассматривается окно и решается вопрос о том, пусто оно или нет его содержимое достаточно просто для визуализации [16]. В противном случае окно разбивается на более маленькие области, для которых снова решается этот вопрос. Предполагается, что если размер окна в результате его разбиения стал совпадать с одним пикселем, но при этом оно содержит не один многоугольник, то необходимо визуализировать тот из них, у которого максимальное значение координаты Z .

К преимуществам алгоритма можно отнести низкие затраты на обработку областей, содержащих малое количество информации. Недостатком алгоритма может являться необходимость проведения большого количества разбиений, что негативно скажется на временной эффективности.

Алгоритм, использующий Z-буфер является одним из простейших алгоритмов удаления невидимых поверхностей [16]. Он работает в пространстве изображения.

Суть алгоритма заключается в использовании Z-буфера, в котором запоминается координата z каждого видимого пикселя, в то время как буфер кадра используется для запоминания атрибутов каждого пикселя. В процессе работы глубина очередного пикселя сравнивается с глубиной, находящейся в Z-буфере, и в результате сравнения либо заносится в этот буфер, или никаких действий не производится.

Преимуществом данного алгоритма является его простота. Кроме того, он позволяет работать со сценами любой сложности, делая тривиальной визуализацию пересечений сложных поверхностей. Алгоритм обладает не более чем линейной вычислительной трудоёмкостью.

Недостатком же алгоритма является необходимость хранить Z-буфер, что увеличивает затраты по памяти, однако для современных компьютеров они являются незначительными [16].

Алгоритм трассировки лучей - метод грубой силы [16]. Главной идеей данного метода является запуск луча из источника освещения, для каждого луча происходит проверка на пересечение с каждым из объектов сцены, после чего пересечения упорядочиваются по глубине. Пересечение с максимальным значением z представляет видимую поверхность для данного пикселя.

К достоинствам алгоритма можно отнести то, что вычислительная сложность метода линейно зависит от сложности сцены. В результате выполнения алгоритма получается очень реалистичное изображение. А также метод даёт возможность создания гладких объектов без аппроксимации их примитивами.

Однако для получения такого изображения необходимо создать огромное количество лучей, проходящих через сцену, и необходимость нахождения пересечения с каждым из объектов, что негативно сказывается на скорости работы программы.

Чтобы уменьшить количество искомых пересечений производится поиск пересечения луча с объёмной оболочкой рассматриваемого объекта. Если оно не найдено, то нет необходимости в поиске пересечения с объектом. В качестве оболочки можно применять параллелепипед или сферу. Другим упрощением алгоритма является запуск луча из точки наблюдателя в каждый пиксель экранной плоскости, такой метод называется обратной трассировкой.

Вывод: было решено использовать алгоритм, использующий Z-буфер, поскольку его реализация проще, чем у других алгоритмов, в то время как его вычислительная трудоёмкость остаётся линейной. И, как уже было отмечено раньше, затраты памяти, связанные с хранением Z-буфера, больше, чем у других алгоритмов, но они всё же незначительны для современных вычислительных машин.

1.6 Анализ моделей освещения

При создании реалистичного изображения важно учитывать освещённость поверхностей, ниже рассмотрены модели освещения, используемые с этой целью.

Простая модель освещения ограничивается отображением света, исходящего от явных точечных источников. В ней не учитываются взаимодействия объектов сцены между собой, но учитывается диффузное отражение света. Интенсивность света, отражённого предметом вычисляется по формуле:

$$I = I_a k_a + I_l k_d \cos \theta \quad 0 \leq \theta \leq \pi/2 \quad (1.7)$$

Глобальная модель описывает освещение, учитывая взаимное влияние объектов сцены. Она рассматривает многократное отражение и преломление света, рассеянное освещение. Для этого отслеживается путь луча до тех пор, пока он не покинет сцену или его энергия не станет достаточно малой, чтобы не учитывать ее. Глобальная модель освещения позволяет получить более реалистичное изображение.

Ещё одна модель – физически-корректный рендеринг [17], его идея заключается в задании материала с определёнными параметрами зеркального и диффузного отражения, а также учитывает закон Френеля, согласно которому блик есть у любого материала.

Вывод: для визуализации мышц можно пренебречь преломлением и бликами, так что было решено использовать простую модель освещения, используя формулу Ламберта. Кроме того использование такой модели позволит выиграть в производительности.

1.7 Анализ методов закраски

Методы закраски регулируют то, каким образом будет рассчитана интенсивность в пределах определённой грани.

Метод простой закраски— его идея заключается в том, что для каждой грани объекта находится вектор нормали, с помощью которого в соответствии с выбранной моделью освещения вычисляется значение интенсивности, с которой закрашивается вся грань.

Метод простой закраски обладает наилучшим быстродействием, однако все пиксели грани обладают одинаковой интенсивностью, в связи с чем сцена может выглядеть нереалистично.

Метод Гуро позволяет получить более сглаженное изображение, чем простая закрашка. Такой результат получается за счёт билинейной интерполяции интенсивностей в пределах одной сканирующей строки.

Недостатками метода Гуро является появления полос Маха в связи с тем, что такой метод интерполяции не обеспечивает непрерывности изменения интенсивности, а также то, что некоторые рёбра могут казаться сглаженными.

Метод Фонга требует больших вычислительных затрат, чем закрашка по Гуро. Идея такой закраски заключается в интерполировании вектора нормали вдоль сканирующей строки, который затем используется в модели освещения для расчёта интенсивности.

Вывод: выбирая между методами закраски было решено использовать метод простой закраски, который является, самым быстрым из представленных, что позволит отображать анимацию более плавно.

1.8 Вывод

В данном разделе была рассмотрена предметная область, аналоги разрабатываемому приложению, основные алгоритмы, необходимые для создания реалистичного изображения визуализации мышечных сокращений. Из них были выбраны:

- использование деформируемых эллипсоидов для аппроксимации мышц;

- полигональный метод моделирования;
- трёхмерный морфинг на основе мешей;
- алгоритм Z-буфера для удаления невидимых граней;
- простая модель освещения.
- метод простой закрашки

2 Конструкторский раздел

В этом разделе будут рассмотрены требования к программе и схемы для реализации алгоритмов, математические расчёты. Программа должна предоставлять возможность:

- загружать модель мышц;
- вращать и перемещать загруженную модель;
- визуализировать мышечные сокращения;
- задавать степень сокращения;
- добавлять и перемещать источники освещения.

2.1 Общий алгоритм работы программы

Алгоритм работы программы можно описать следующими шагами.

1. Загрузить модель из файла.
2. Сформировать модель сокращённой мышцы, учитывая степень сокращения.
3. Сформировать промежуточные меши:
 - 1) для каждого промежуточно меша закрасить объект используя простую модель освещения;
 - 2) заполнить буфер глубины и экранный буфер, применяя алгоритм z-буфер;
 - 3) вывести заполненный экранный буфер.

На рисунках 2.1 и 2.2 приведены IDEF0 диаграммы, которые представляют организацию работы программы.

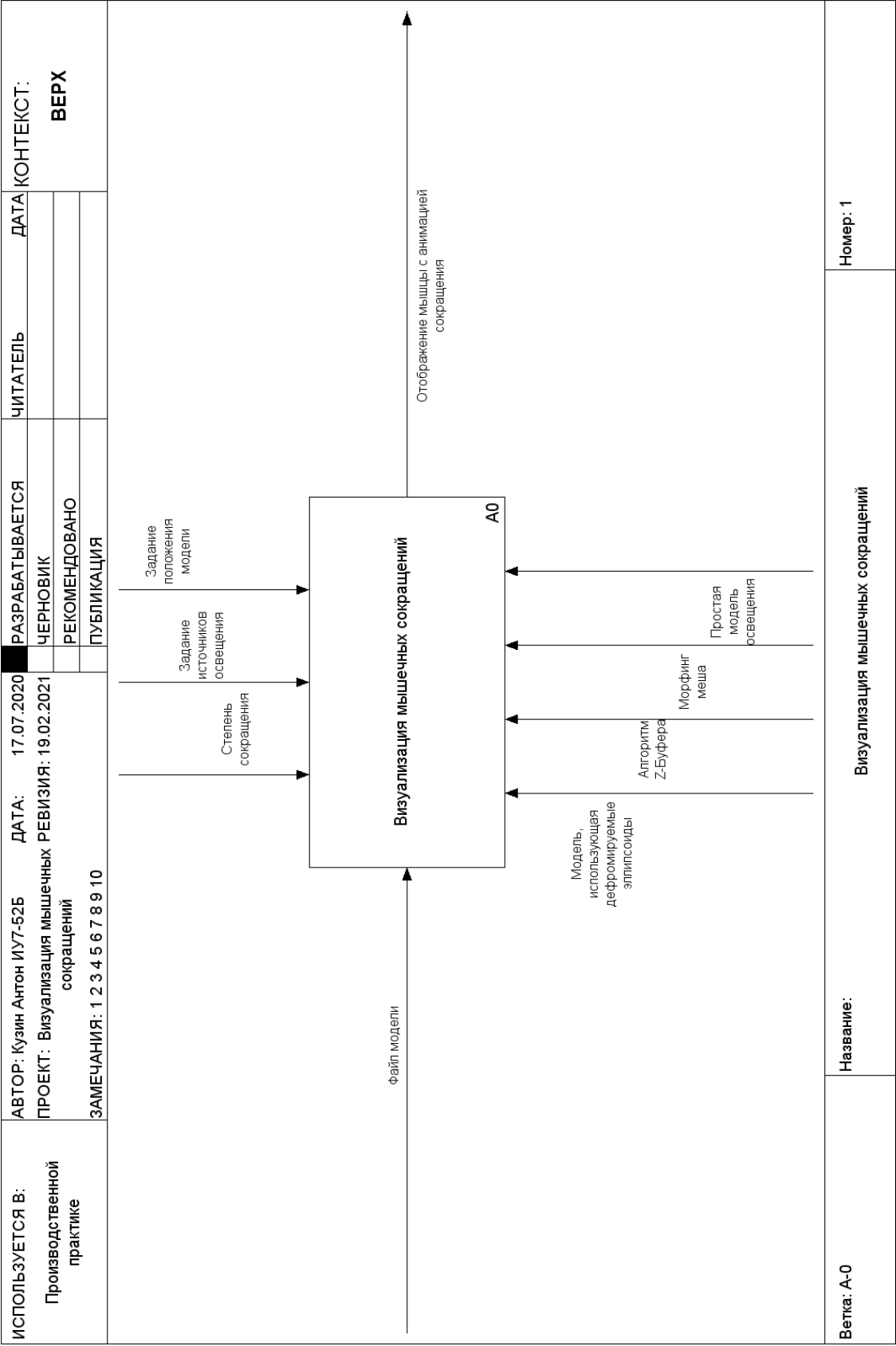
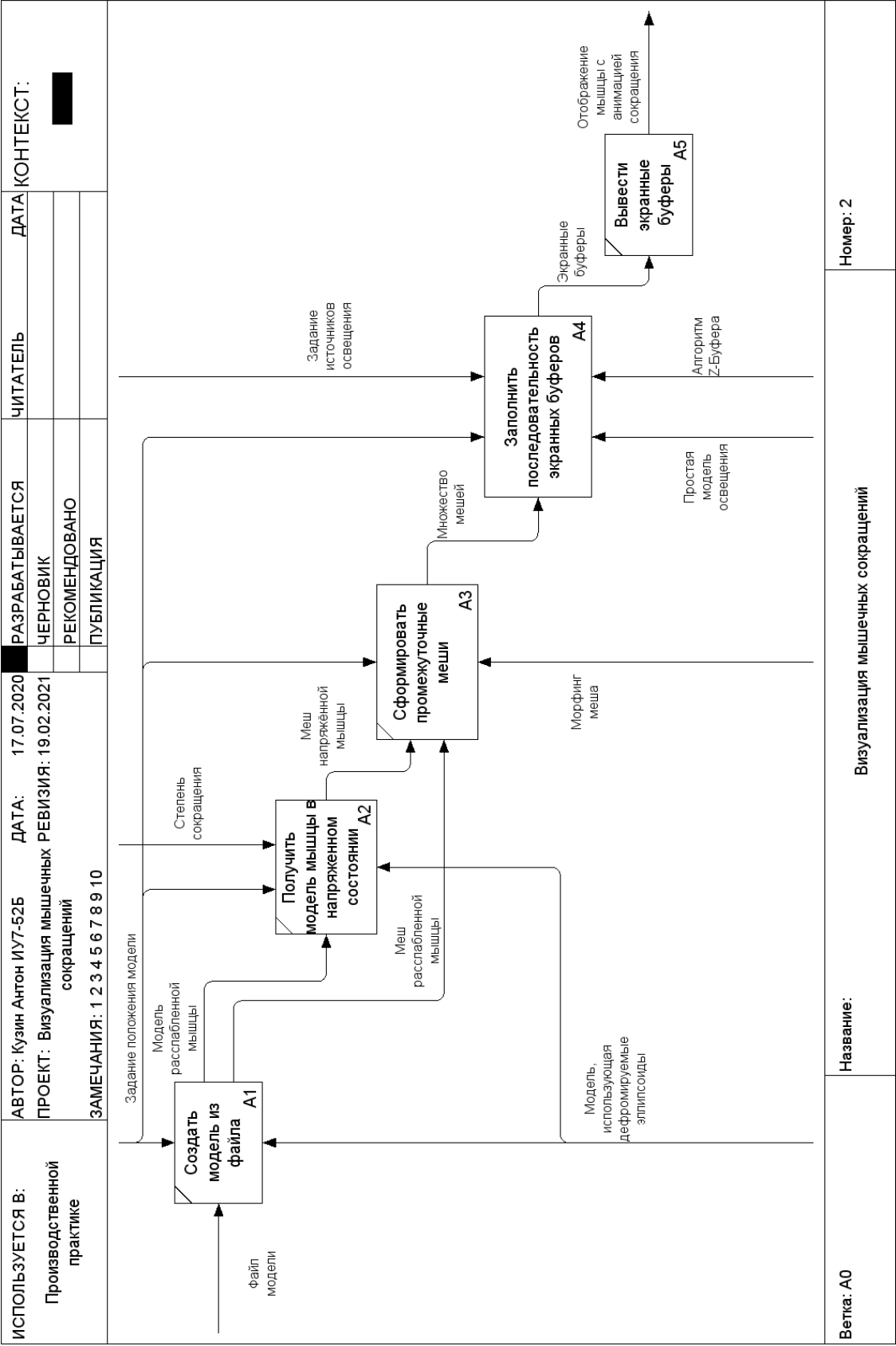


Рисунок 2.1 — IDEF0 диаграмма работы программы



Ветка: A0	Название: Визуализация мышечных сокращений	Номер: 2
-----------	--	----------

Рисунок 2.2 — Последовательность действий для отображения анимации сокращения мышцы

2.2 Реализуемые алгоритмы алгоритмы

Модель скелетных мышц В качестве модели аппроксимации скелетных мышц было решено использовать деформируемые эллипсоиды. Для представления сокращения параметры эллипсоида задаются с помощью математических уравнений [5], что позволяет сохранять объем и соотношение между высотой и шириной мышцы. Объем рассчитывается по формуле:

$$v = \frac{4\pi abc}{3} \quad (2.1)$$

Полагая l' новой длиной мышцы, $r = \frac{a}{b}$, новые параметры вычисляются по формулам:

$$c' = \frac{l'}{2} \quad (2.2)$$

$$b' = \sqrt{\frac{3v}{4\pi rc'}} \quad (2.3)$$

$$a' = b'r \quad (2.4)$$

Для визуализации изометрического сокращения мышцы соотношение r пересчитывается по формуле:

$$r = (1 - t + kt)r_n \quad (2.5)$$

где r_n —соотношение $\frac{a}{b}$ в расслабленном состоянии, t —степень напряжённости $t \in [0,1]$, k —параметр контроля напряжения, который регулирует степень сжатия.

Для изображения моделей мышц, аппроксимируемых эллипсоидами, алгоритмом, использующим Z-буфер, необходимо представить её с помощью полигонов. Алгоритм, триангулирующий эллипсоид, который задан параметрами a, b и c , можно описать следующими шагами:

1. Построить куб в начале координат.
2. Представить его в виде треугольных полигонов.
3. Выполнять рекурсивное разбиение полигонов, путём создания вершин в середине каждого из 3-х рёбер.
4. Нормализовать каждую вершину.

5. К полученной сфере применить преобразования, умножив координаты точек на параметры a , b и c эллипсоида.

Увеличение глубины рекурсии создаст большее число полигонов, что позволит сделать изображение более реалистичным, однако увеличит время необходимое на его обработку.

Алгоритм z-буфера в общем виде может быть описан последовательностью таких шагов:

1. Заполнить буфер кадра фоновым значением интенсивности или цвета.
2. Заполнить z-буфер минимальным значением z .
3. Преобразовать каждый многоугольник в растровую форму в произвольном порядке:
 - 1) Для каждого пикселя в многоугольнике вычислить его глубину z .
 - 2) Сравнить глубину z со значением z в буфере в этой же позиции. Если $z > Z_{\text{буфер}}$, то записать атрибут этого пикселя в буфер кадра и заменить $Z_{\text{буфер}}$ на z .

Простая модель освещения интенсивность рассчитывается по закону Ламберта с учётом рассеянного освещения, представленного константой:

$$I = I_a k_a + I_l k_d \cos \theta \quad 0 \leq \theta \leq \pi/2 \quad (2.6)$$

I - интенсивность отражённого света, I_l - интенсивность точечного источника, k_d - коэффициент диффузного отражения ($0 \leq k_d \leq 1$), θ - угол между направлением света и нормалью к поверхности, I_a - интенсивность рассеянного света, k_a - коэффициент диффузного отражения рассеянного света ($0 \leq k_a \leq 1$).

Морфинг меша можно разделить на два этапа: установление соответствия между полигонами начального и конечного состояний модели и интерполяция положения вершин. На рисунке 2.3 представлен алгоритм морфинга

мешей, следует отметить что для установления связи между полигонами начала и конца не производится никаких вычислений, а каждому i -ому полигону начала в соответствие ставится i -ый полигон конца, это связано с тем, что объект источника и объект цели имеют одинаковую форму и одинаковое количество полигонов, в связи с чем предполагается, что для того, чтобы не нарушить целостность объекта необходимо сопоставить их в том же порядке.

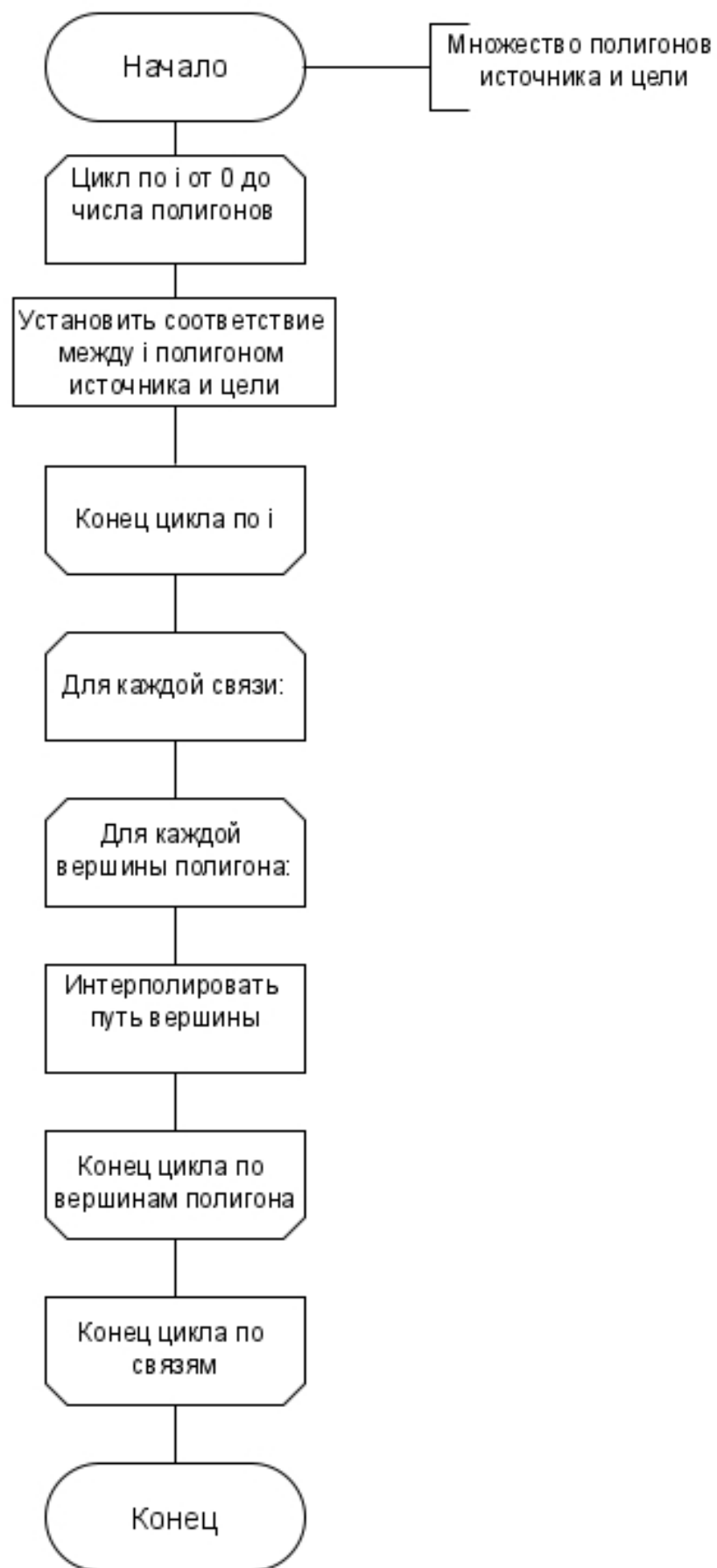


Рисунок 2.3 — Алгоритм морфинга

2.3 Вывод

В этом разделе были рассмотрены требования к программе, общий алгоритм ее работы, представлены схемы для реализации выбранных алгоритмов и математические формулы.

3 Технологический раздел

В данном разделе будут представлены средства реализации, разработанные алгоритмы и интерфейс.

3.1 Средства реализации

В качестве языка программирования был выбран Python [18] по скольку это объектно-ориентированный язык, рассматриваемый во время учёбы, что значительно сокращает время написания программы и упрощает разработку.

В качестве среды разработки была выбрана "PyCharm Community Edition", а для представления пользовательского интерфейса библиотека PyQt5 [19].

3.2 Описание структуры программы

Описание основных модулей программы

- main.py – загружает интерфейс и запускает программу;
- window.py – содержит класс интерфейса;
- sceneinterface.py – содержит класс SceneInterface, выступающий в качестве посредника между сценой и интерфейсом;
- scene.py – содержит класс сцены, основной класс, в котором находятся объекты и источники света;
- commands.py – содержит классы команд, которые влияют на сцену;
- controlmanager.py – содержит классы менеджеров, отвечающие за выполнение сложных команд, таких как отрисовка и загрузка моделей;
- builder.py – содержит класс Builder, отвечающий за чтение файла и создание объекта модели;
- bulddirector.py – содержит класс, контролирующий создание объекта;

- `drawer.py` – содержит классы `QDrawer` и `BaseDrawer`, которые предоставляют методы отрисовки на самом низком уровне;
- `drawerfactory.py` – содержит классы `DrawerFactory` и `QDrawerFactory`, целью которых является создание объекта `Drawer`;
- `baseobject.py` – содержит класс базового объекта, точки и вектора;
- `invisibleobject.py` – содержит класс камеры и источника света;
- `visibleobject.py` – содержит класс полигона;
- `visitor.py` – содержит методы, применяемые к разным объектам сцены, а именно перемещение и поворот;
- `visualizer.py` – содержит методы, отвечающие за взаимодействие с `drawer`-ом;
- `musclemodel.py` – содержит класс эллипсоида, прямой мышцы и сложной мышцы;

На рисунке 3.1 представлена диаграмма классов, из неё видно разделение на отдельные модули, что позволяет изменять и дополнять некоторые из них независимо от других, благодаря чему программу будет проще обновлять при необходимости.

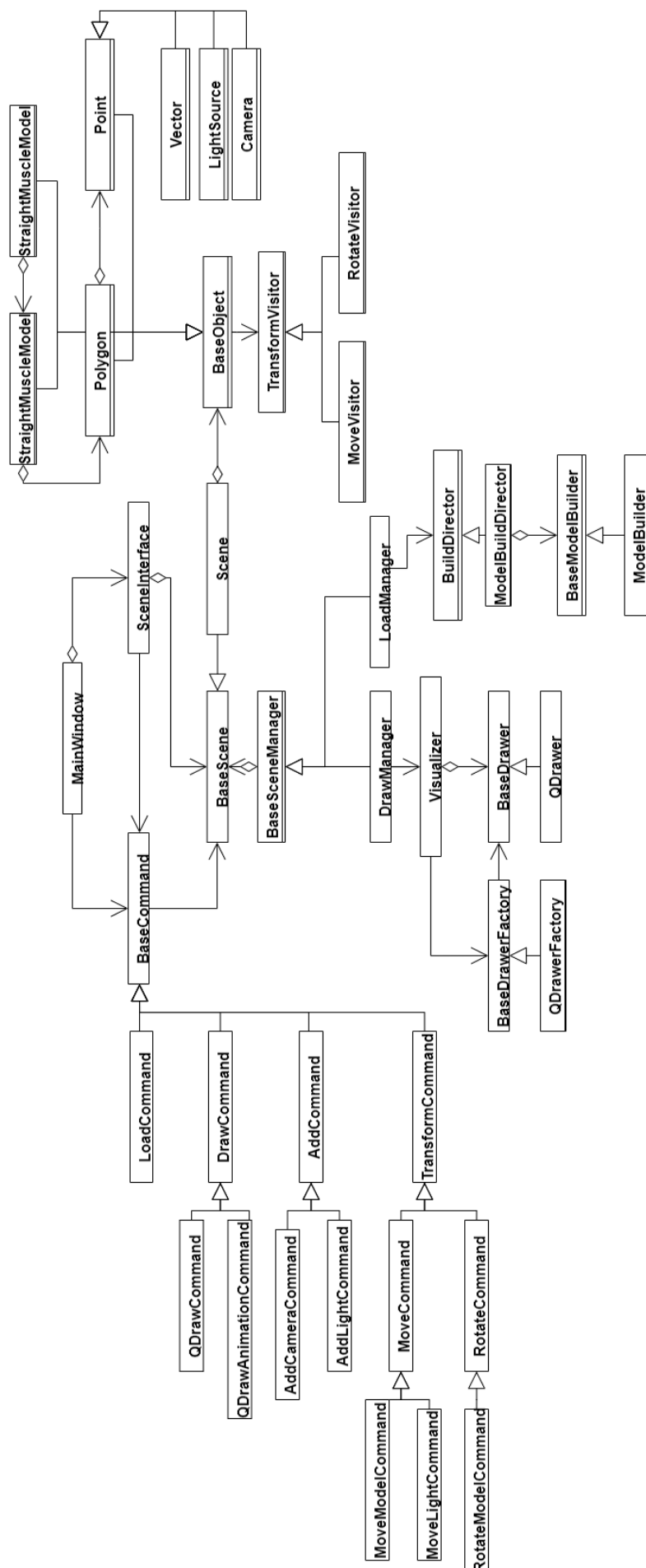


Рисунок 3.1 — Диаграмма классов

3.3 Листинг кода

В листинге 3.1 представлен код класса Z-буфера, который содержит метод **process_polygon(self, polygon, light)** на вход которого подаётся полигон и массив источников света, а он растеризует поданный треугольник, вызывает метод для вычисления интенсивности цвета полигона и заполняет экранный и z-буфер.

Листинг 3.1 — Класс Z-буфера

```
1      class ZBuffer(object):
2          def __init__(self, visualizer, width=900, height=900):
3              self.width = width
4              self.height = height
5              self.visualizer = visualizer
6              self._buf = [[-6000 for _ in range(width)] for __ in
                           range(height)]
7
8          def process_polygon(self, polygon, light):
9              color = polygon.get_color(light)
10
11              points = polygon.get_points()
12              x = [floor(points[i].x) for i in range(3)]
13              y = [floor(points[i].y) for i in range(3)]
14
15              ymax = min(max(y), self.height)
16              ymin = min(min(y), 0)
17
18              x1 = x2 = 0
19              z1 = z2 = 0
20              for y_current in range(ymin, ymax+1):
21                  first_cycle = 1
22                  for n in range(3):
23                      n1 = 0 if n == 2 else n + 1
24                      if y_current >= max(y[n], y[n1]) or y_current <
                           min(y[n], y[n1]):
25                          continue
26
27                      m = float(y[n] - y_current) / (y[n]-y[n1])
28                      if first_cycle == 0:
29                          x2 = x[n] + floor(m * (x[n1] - x[n]))
30                          z2 = points[n].z + m * (points[n1].z - points[n].z)
31                      else:
32                          x1 = x[n] + floor(m * (x[n1] - x[n]))
33                          z1 = points[n].z + m * (points[n1].z - points[n].z)
```

```

34
35         first_cycle = 0
36
37         if x2 < x1:
38             x2, x1 = x1, x2
39             z2, z1 = z1, z2
40
41         x_max = min(x2, self.width)
42         x_min = max(x1, 0)
43         for x_current in range(x_min, x_max):
44             m = float(x1 - x_current) / (x1 - x2)
45             z_current = z1 + m * (z2 - z1)
46             self.process_point(x_current, y_current,
47                               int(z_current), color)
48
49     def process_point(self, x: int, y: int, z: int, color):
50         if z > self._buf[x][y]:
51             self._buf[x][y] = z
52             self.visualizer.drawPoint(x, y, color)

```

В листинге 3.2 представлен метод вычисления интенсивности цвета полигона по формуле Ламберта.

Листинг 3.2 — Метод расчёта интенсивности цвета полигона

```

1     def get_color(self, lights):
2         x, y, z = 0, 0, 0
3         for p in self.get_points():
4             xt, yt, zt = p.get_position()
5             x += xt
6             y += yt
7             z += zt
8
9         centroid = Point(x / 3, y / 3, z / 3)
10        normal = self.get_normal().normalize()
11
12        intensity = 0.4
13        for light in lights:
14            v = Vector().from_points(light, centroid).normalize()
15            cos = max(v.scalar_multiplication(normal), 0)
16            intensity += light.intensity * cos
17
18        rgb = list(QColor(self.color).getRgbF())
19
20        rgb[0] *= intensity
21        rgb[1] *= intensity
22        rgb[2] *= intensity

```

23

24 **return** QColor().fromRgbF(rgb[0], rgb[1], rgb[2])

В листинге 3.3 представлен код класса эллипсоида, в котором в методе `__init__(self)` определяются координаты вершин исходного куба, а в методе `rec_subdivide(self, d)` производится рекурсивное разбиение его полигонов.

Листинг 3.3 — Класс эллипсоида

```

1      class Ellipsoid(object):
2          def __init__(self):
3              x = 1
4              z = 1
5              y = 1
6
7              self.vertex_list = [
8                  [-x, -y, -z], [x, -y, -z], [x, -y, z], [-x, -y, z],
9                  [-x, y, -z], [x, y, -z], [x, y, z], [-x, y, z]
10             ]
11
12             self.triangle_list = [
13                 [0,2,1], [0,3,2], [1,5,0], [5,4,0], [0,4,7], [7,3,0],
14                 [6,5,1], [1,2,6], [7,2,3], [7,6,2], [4,5,6], [6,7,4]
15             ]
16
17             self.result_ind = []
18             self.result_vertexes = []
19
20         @staticmethod
21         def normalize(v):
22             d = sqrt(v[0] * v[0] + v[1] * v[1] + v[2] * v[2])
23             if d == 0.0:
24                 print("zero division")
25             v[0] /= d
26             v[1] /= d
27             v[2] /= d
28
29         def start_division(self, d):
30             self.result_ind = []
31             self.result_vertexes = []
32             for i in range(12):
33                 self.rec_subdivide(
34                     self.vertex_list[self.triangle_list[i][0]],
35                     self.vertex_list[self.triangle_list[i][1]],
36                     self.vertex_list[self.triangle_list[i][2]], d)

```

```

37         for vertex in self.result_vertexes:
38             self.normalize(vertex)
39
40     def rec_subdivide(self, v1, v2, v3, d):
41         v12 = [0, 0, 0]
42         v23 = [0, 0, 0]
43         v31 = [0, 0, 0]
44
45         if d == 0:
46             if v1 not in self.result_vertexes:
47                 self.result_vertexes.append(v1)
48             if v2 not in self.result_vertexes:
49                 self.result_vertexes.append(v2)
50             if v3 not in self.result_vertexes:
51                 self.result_vertexes.append(v3)
52             t = [0, 0, 0]
53             t[0] = self.result_vertexes.index(v1)
54             t[1] = self.result_vertexes.index(v2)
55             t[2] = self.result_vertexes.index(v3)
56
57             self.result_ind.append(t)
58             return
59
60         for i in range(3):
61             v12[i] = (v1[i] + v2[i]) / 2
62             v23[i] = (v2[i] + v3[i]) / 2
63             v31[i] = (v3[i] + v1[i]) / 2
64
65         self.rec_subdivide(v1, v12, v31, d - 1)
66         self.rec_subdivide(v2, v23, v12, d - 1)
67         self.rec_subdivide(v3, v31, v23, d - 1)
68         self.rec_subdivide(v12, v23, v31, d - 1)
69
70     def reshape(self, x, y, z):
71         for vertex in self.result_vertexes:
72             vertex[0] *= x
73             vertex[1] *= y
74             vertex[2] *= z
75
76     def polygonise(self, center_x=0, center_y=0, center_z=0):
77         polygons = []
78         for triangle in self.result_ind:
79             p1 = Point(self.result_vertexes[triangle[0]][0] + center_x,
80                       self.result_vertexes[triangle[0]][1] + center_y,
81                       self.result_vertexes[triangle[0]][2] + center_z)

```

```

80         p2 = Point(self.result_vertexes[triangle[1]][0] + center_x,
                    self.result_vertexes[triangle[1]][1] + center_y,
                    self.result_vertexes[triangle[1]][2] + center_z)
81         p3 = Point(self.result_vertexes[triangle[2]][0] + center_x,
                    self.result_vertexes[triangle[2]][1] + center_y,
                    self.result_vertexes[triangle[2]][2] + center_z)
82         polygons.append(Polygon(p1, p2, p3))
83     return polygons

```

В листинге 3.4 представлен код класса Morph, который содержит в себе методы, необходимые для вычисления промежуточных мешей, а именно метод установления между начальными и результирующими полигонами **straight_map(self)** и метод **calculate_paths(self)**, рассчитывающий путь, который необходимо преодолеть каждой вершине каждого полигона.

Листинг 3.4 — Класс морфинга

```

1     class Morph(object):
2         def __init__(self, begin, end, frames=30):
3             self.begin = begin
4             self.end = end
5             self.paths = []
6             self.pairs = []
7
8             self.frames = frames
9
10            self.straight_map()
11            self.calculate_paths()
12
13        def straight_map(self):
14            number = len(self.begin)
15            self.pairs = [[i, i, 0] for i in range(number)]
16            return self.pairs
17
18        def calculate_paths(self):
19            for pair in self.pairs:
20                p1 = self.begin[pair[0]].get_points()
21                p2 = self.end[pair[1]].get_points()
22                v3 = []
23                for i in range(3):
24                    v = Vector()
25                    v.from_points(p1[i], p2[i])
26                    v3.append(v)
27
28            self.paths.append(v3)

```

```

29
30         return self.paths
31
32     def get_frame(self, frame_number):
33         number = len(self.begin)
34         result = []
35
36         for i in range(number):
37             points = self.begin[self.pairs[i][0]].get_points()
38             for p in range(3):
39                 points[p] = points[p] + \
40                     self.paths[self.pairs[i][0]][p]. \
41                         number_multiplication(frame_number) / self.frames
42             result.append(Polygon(points[0], points[1], points[2],
43                                   self.begin[self.pairs[i][0]].color))
44
45         return result

```

3.4 Интерфейс программы

На рисунке 3.2 представлен интерфейс программы.

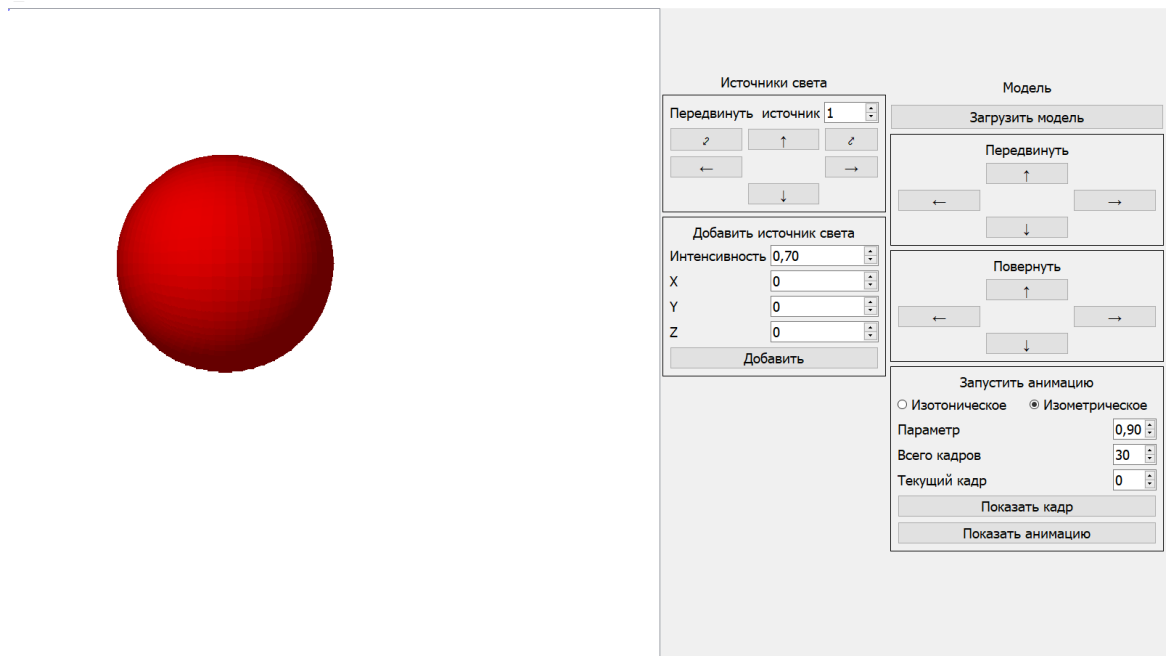


Рисунок 3.2 — Интерфейс программы

Интерфейс включает в себя блок взаимодействия с источниками света и блок взаимодействия с моделью. Первый, представленный на рисунке 3.3, предоставляет возможности:

- передвинуть источник света по одной из 3 осей;
- добавить новый источник света, задав его интенсивность и координаты.

Источники света

Передвинуть источник 1

↑ ↓ ← →

Добавить источник света

Интенсивность 0,70

X 0

Y 0

Z 0

Добавить

Рисунок 3.3 — Блок взаимодействия с источниками света

Второй блок, представленный на рисунке 3.4, содержит поля, отвечающие за:

- загрузку модели;
- перемещение и поворот модели;
- выбор типа сокращения – изометрическое или изотоническое сокращение;
- выбор количества кадров анимации;
- отображение анимации и отдельных промежуточных кадров.

Модель

Загрузить модель

Передвинуть

↑

←

→

↓

Повернуть

↑

←

→

↓

Запустить анимацию

☐ Изотоническое

☒ Изометрическое

Параметр

0,90

Всего кадров

30

Текущий кадр

0

Показать кадр

Показать анимацию

Рисунок 3.4 — Блок взаимодействия с моделями

3.5 Вывод

В данном разделе были представлены средства, используемые для реализации выбранных алгоритмов, приведены листинги основных классов и функций. Также был рассмотрен реализованный интерфейс программы, модули и диаграмма классов.

4 Исследовательский раздел

В данном разделе будет проведено исследование реализованной программы.

Цель эксперимента – оценка времени выполнения алгоритмов в зависимости от количества полигонов.

Постановка эксперимента – эксперимент проводился на компьютере со следующими параметрами Intel(R) Core(TM) i5-8250, 8гб оперативной памяти, операционная система Windows 10.

Количество полигонов изменялось за счёт увеличения числа рекурсивных повторений функции разбиения, так что их число можно посчитать по формуле

$$p = 12 * 4^n \quad (4.1)$$

где p – число полигонов, n – глубина рекурсии.

Для каждого n эксперимент проводился 5 раз, измерялось время для отрисовки 1 кадра, после чего результат усреднялся:

- $n=1$, 48 полигонов, затраченное время 0,2с;
- $n=2$, 192 полигонов, затраченное время 0,24с;
- $n=3$, 768 полигонов, затраченное время 0,29с;
- $n=4$, 3072 полигонов, затраченное время 0,42с;
- $n=5$, 12288 полигонов, затраченное время 0,94с.

На рисунке 4.1 представлен график с результатами проведённого исследования.

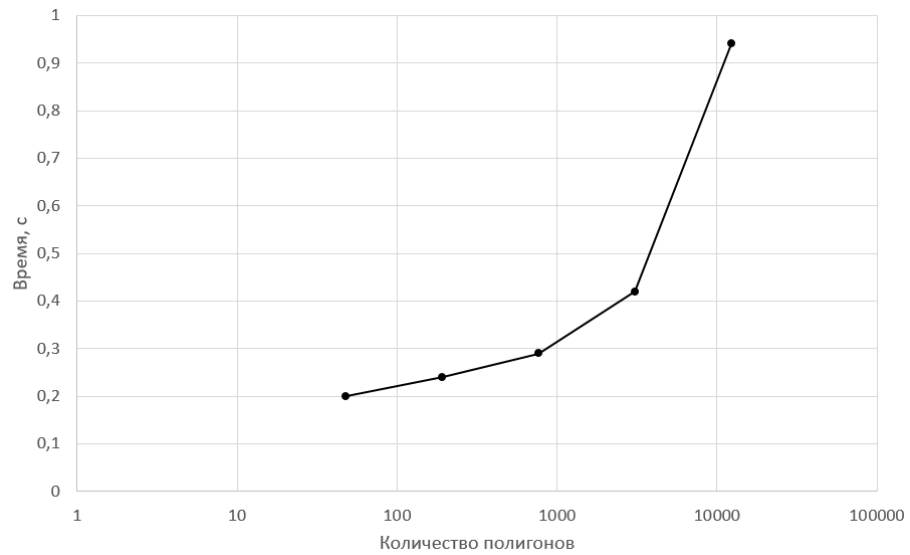


Рисунок 4.1 — График зависимости времени отрисовки одного кадра от количества полигонов

Вывод из представленных результатов видно, что рост числа полигонов в 4 раза незначительно увеличивает время обработки модели, однако при увеличении глубины рекурсии с 4 до 5 затрачиваемое время растёт более чем в 2 раза, в то время как реалистичность изображения изменяется не сильно.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы были проанализированы существующие алгоритмы удаления невидимых линий, морфинга, освещения и закраски, рассмотрены методы аппроксимации мышц с целью анимации их сокращения и моделирования. Из них были выбраны и реализованы наиболее подходящие для решения поставленной задачи. Было разработано программное обеспечение для визуализации мышечных сокращений.

Реализованная программа предоставляет пользователю возможность наблюдать как изотоническое, так и изометрическое сокращение мышц, задавая степень изменения длины или напряжённости, загружать модель из файла, вращать и перемещать её, добавлять и перемещать источники света.

В качестве перспектив дальнейшего развития можно предположить добавление новых моделей мышц, возможности редактирования параметров загруженной модели, улучшение реалистичности изображения путём изменения метода закраски на метод Гуро или Фонга, или путём наложения текстур на мышцы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Гладышева А. А. Анатомия человека. Учебник для техникумов физической культуры. — М. : Физкультура и спорт, 1977. — 343 с.
2. Строение мышц человека [Электронный ресурс]. — 2015. — Режим доступа: <https://fit-baza.com/stroenie-myshc-cheloveka/> (дата обращения: 08.07.2020).
3. Структура и части саркомера, функции и гистология [Электронный ресурс]. — 2020. — Режим доступа: <https://ru.thpanorama.com/articles/anatoma-y-fisiologa/sarcmero-estructura-y-partes-funciones-e-histologa.html> (дата обращения: 08.07.2020).
4. Dongwoon Lee Michael Glueck Azam Khan Eugene Fiume Ken Jackson. Modeling and Simulation of Skeletal Muscle for Computer Graphics: A Survey // Foundations and Trends in Computer Graphics and Vision. — 2012. — Vol. 7, no. 4. — P. 229–276.
5. Ferdi Scheepers Richard E. Parent Wayne E. Carlson Stephen F. May. Anatomy-Based Modeling of the Human Musculature // SIGGRAPH Computer graphics. — 1997. — P. 163–172.
6. Jane Wilhelms Allen van Gelder. Anatomically Based Modeling // SIGGRAPH Computer graphics. — 1997. — P. 173–180.
7. Ramos Juan, Larboulette Caroline. A Muscle Model for Enhanced Character Skinning // Journal of WSCG.
8. Nedel Luciana, Thalmann Daniel. Real Time Muscle Deformations Using Mass-Spring Systems // Proceedings of Computer Graphics International. IEEE.
9. Chen David T., Zeltzer David. Pump It up: Computer Animation of a Biomechanically Based Model of Muscle Using the Finite Element Method // SIGGRAPH Comput. Graph. — 1992. — Vol. 26, no. 2. — P. 89–98.

10. Teran J. Blemker S. Hing V. N. T., Fedkiw R. Finite volume methods for the simulation of skeletal muscle // SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation.
11. Никулин Е.А. Компьютерная геометрия и алгоритмы машинной графики. — СПб. : БХВ-Петербург, 2003. — 560 с.
12. Порев В.Н. Компьютерная графика. — СПб. : БХВ-Петербург, 2002. — 432 с.
13. Apostolos Lerios Chase D. Garfinkle, Levoy. Marc. Feature-Based Volume Metamorphosis // Proceedings of SIGGRAPH '95. — 1995. — Vol. 26, no. 2. — P. 35–42.
14. Alexa Marc. Recent Advances in Mesh Morphing // Computer Graphics Forum. — 2002. — Vol. 21.
15. 3D Morphing [Электронный ресурс]. — 2001. — Режим доступа: <http://web.mit.edu/manoli/morph/www/morph.html#morphing> (дата обращения: 17.07.2020).
16. Роджерс Д. Адамс Дж. Алгоритмические основы машинной графики: Пер. с англ. — М. : Мир, 1989. — 512 с.
17. Описание принципов PBR: какие правила оптики учитывает физически-корректный рендеринг [Электронный ресурс]. — 2019. — Режим доступа: <https://dtf.ru/gamedev/81762-opisanie-principov-pbr-kakie-pravila-optiki-uchityvaet-fizich> (дата обращения: 19.02.2021).
18. Python 3.7.9 documentation [Электронный ресурс]. — 2020. — Режим доступа: <https://docs.python.org/3.7/index.html> (дата обращения: 14.02.2021).
19. Qt for Python Documentation [Электронный ресурс]. — 2021. — Режим доступа: <https://doc.qt.io/qtforpython/api.html#basic-modules> (дата обращения: 14.02.2021).