СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Объектом данного исследования является анимация животных. В качестве примера животного, моделирование и построение анимации которого представляет интерес как научная задача, был выбран осьминог, владеющий несколькими способами передвижения, достаточно сложными для имитации. Из существующих подходов к анимации была выбрана процедурная анимация, как одна из методик, позволяющих получить относительно естественную анимацию, при этом достаточно эффективная по вычислительным ресурсам, чтобы ее возможно было генерировать в реальном времени. Как следствие данного выбора, предмет исследования – генерация естественной процедурной анимации для различных способов передвижения осьминога.

Осьминог отличается необычным строением (в первую очередь почти полным отсутствием скелета и подвижными щупальцами), а также большим разнообразием способов передвижения. Также практически не существует животных, близких к осьминогу по строению и поведению. При этом осьминог – предмет большого количества биологических исследований, а также достаточно частый персонаж в видеоиграх и анимационных фильмах. Однако из известных примеров анимации большая часть не стремится к физической реалистичности движений осьминога или моделирует только аспекты его поведения, не связанные с перемещением (такие как захват предметов), в то время как биологические модели оказываются чрезмерно сложными для использования в реальном времени.

Следовательно, актуальность работы состоит в том, что, с одной стороны, создание анимации движения осьминога представляет интерес, а с другой, данная задача еще не была полностью решена. Более того, так как реализаций процедурной анимации осьминога, пригодных для использования в реальном времени, не было найдено вовсе, любое решение этой задачи практически является новым.

Цель работы: разработка алгоритма процедурной анимации для имитации естественного передвижения осьминога. Для достижения этой цели требуется решить следующие задачи:

– Анализ и сравнение существующих методов процедурной анимации

– Исследование анатомии и способов передвижения осьминога

– Поиск и оценка существующих решений, если таковые могут быть найдены

– Формирование требований к алгоритму процедурной анимации

– Выбор подхода к разработке алгоритма для анимации

– Разработка или модификация алгоритма

– Тестирование алгоритма

– Оценка полученного решения и возможных перспектив модернизации

1 Обзор литературы

Для систематизированного начального ознакомления с предметом была взята книга [1], представляющая собой фундаментальную и при этом достаточно новую (2018 года) работу об осьминогах и близких к ним животных, которая сама по себе также базируется на большом числе фундаментальных трудов о морских животных.

Осьминог – один из наиболее известных представителей класса головоногих моллюсков. Этот класс, включающий также таких животных, как кальмар, каракатица и наутилус (рисунок 1), существенно отличается от других классов моллюсков – его представители имеют более развитый мозг, сложные схемы поведения и очень быстро растут, чем оказываются ближе к некоторым рыбам, чем к моллюскам. В целом этот класс по совокупности строения и поведения не похож ни на каких других морских или сухопутных животных. Сам осьминог среди них выделяется почти полным отсутствием скелета (за исключением хрящевого «черепа» у некоторых видов), более развитыми щупальцами и, как следствие, самыми разнообразными способами передвижения.

Рисунок 1 – Головоногие моллюски: 1 — обыкновенный кальмар, 2 — осьминог, 3 — каракатица, 4 — наутилус

Структура щупалец различается у разных головоногих моллюсков, в случае осьминога все 8 щупалец симметричны, имеют продольные ряды присосок и выполняют одинаковые функции, в то время как у некоторых других видов существует дополнительная пара щупалец другой длины и конфигурации присосок (рисунок 2). Присоски используются осьминогом как для захвата предметов, так и для контакта с поверхностью при передвижении [1].

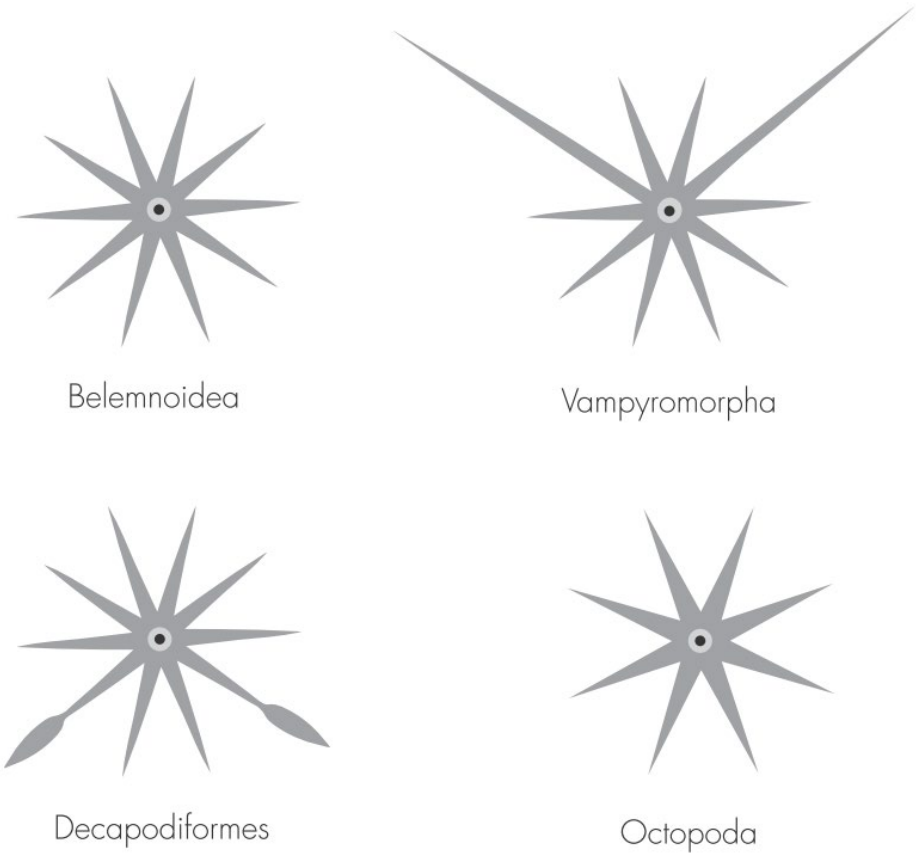

Рисунок 2 – Конфигурация щупалец у различных семейств головоногих (осьминог – справа снизу) [1]

Принципы движения осьминога представляют особый интерес. Из-за отсутствия скелета щупальца двигаются и изгибаются исключительно за счет мышц по принципу сохранения объема – так как мышцы не сжимаются, их сокращение приводит к растяжению в другом направлении. Для сравнения: таким же образом двигаются, например, дождевые черви и человеческий язык [1]. Из-за описанной структуры мышц осьминоги имеют практически неограниченное количество степеней свободы, так как в теории каждое щупальце может изгибаться в любой точке под любым углом [2].

Как следствие, осьминог обладает уникальными способами передвижения (рисунок 3). Широко известен древний и распространенный у всех головоногих реактивный способ плавания, представляющий собой всасывание и выброс потока воды через особый орган – «воронку». Этот способ передвижения позволяет быстро

разгоняться (некоторые виды кальмаров могут даже выпрыгивать из воды с его помощью), однако энергетически очень неэффективен [1]. Из-за особенности расположения «воронки» у осьминога он способен передвигаться таким способом как вперед головой, так и назад [3]. У осьминога существует и другая, более медленная и экономичная техника плавания с синхронным отталкиванием щупальцами, схожая с техникой плавания медузы [1, 3]. Также у осьминогов существуют различные способы передвижения по дну. Обычно осьминог ползет, задействуя попеременно все щупальца, каждый раз выбирая наиболее подходящее, чтобы зацепиться за дно. Несколько новых статей, таких как [5, 6], описывают передвижение осьминога по дну «на двух ногах», первоначально открытое только у одной разновидности, но в последнее время наблюдаемое и у других. В этом случае осьминог приподнимается над поверхностью на двух щупальцах и держит остальные в нетипичных положениях, что существенно меняет его внешний вид, отпугивая хищников. (Одновременно с этим осьминог часто также меняет цвет, что является для него основным способом маскировки [1].)

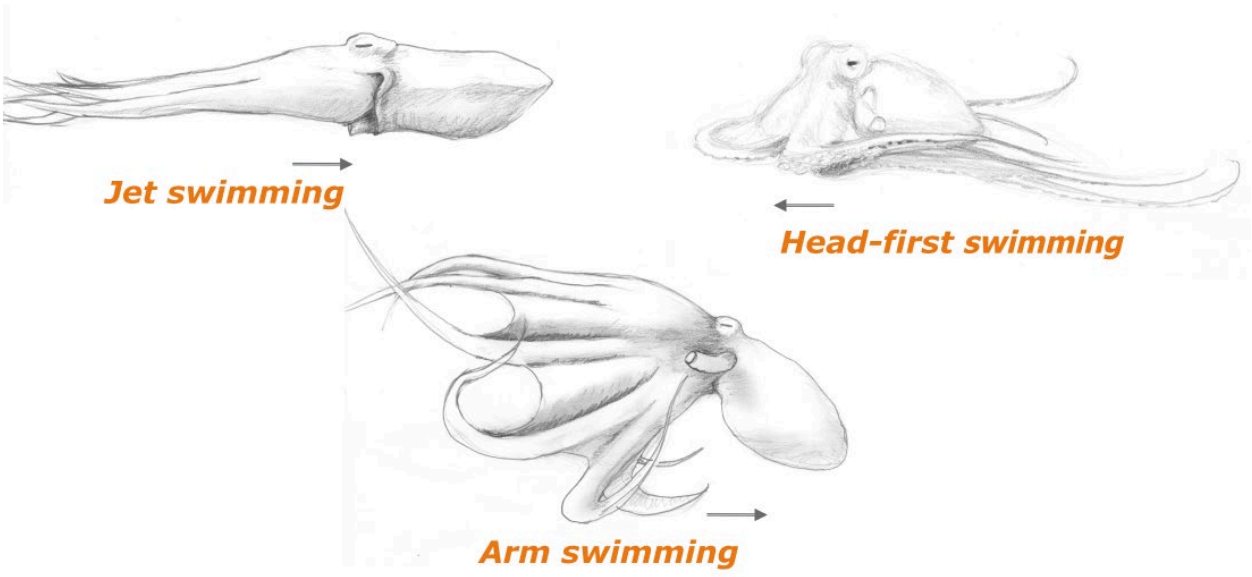

Рисунок 3 – Различные техники плавания осьминога [2]

Осьминог считается «сложным» и «умным» морским животным и имеет относительно крупный мозг, по числу нейронов сопоставимый с мозгом собаки или

кошки. При этом еще большее количество нейронов содержится в его щупальцах и коже, а на конце каждого щупальца даже существует отдельный примитивный «мозг», способный независимо контролировать некоторое количество функций щупальца [7]. Такая сложная и обширная нервная система необходима для координации движений, усложненной большим числом степеней свободы, а также других специфических видов поведения, таких как быстрая (около 200 миллисекунд) смена цвета [1]. Способ координации движений мозгом осьминога продолжает исследоваться до сегодняшнего дня. Позвоночные животные имеют в мозгу «карту» тела, сопоставляющую для каждой части тела полученный сигнал и реакцию. Обеспечивает это ограниченное число суставов и, как следствие, степеней свободы, которые можно явным образом контролировать. Для осьминога подобный способ управления движением потребовал бы недопустимо большой вычислительной мощности.

Уже было доказано, что мозг осьминога не содержит такой структуры, и потому даже самые простые действия (такие как вытягивание щупальца для последующего захвата предмета) выполняются принципиально иначе, в основном с дополнительным искусственным ограничением числа степеней свободы. Если говорить конкретно о перемещении (плавание и передвижение по дну), было замечено, что вне зависимости от положения в пространстве, осьминог держит голову параллельно направлению движения, а глаза – на одной высоте [2]. Все головоногие обладают очень чувствительными органами равновесия, способными также ощущать линейное и угловое ускорение, что необходимо, в частности, для резкой смены направления при быстром реактивном плавании [1]. Расположение данного органа в теле осьминога приводит к тому, что ощущаемое положение в пространстве определяется именно положением головы. Можно сделать вывод, что постоянное удержание головы под одним углом относительно окружающего мира накладывает дополнительное ограничение на объем возможных вариантов

движений и таким образом уменьшает умственные затраты на выбор траектории движения (рисунок 4).

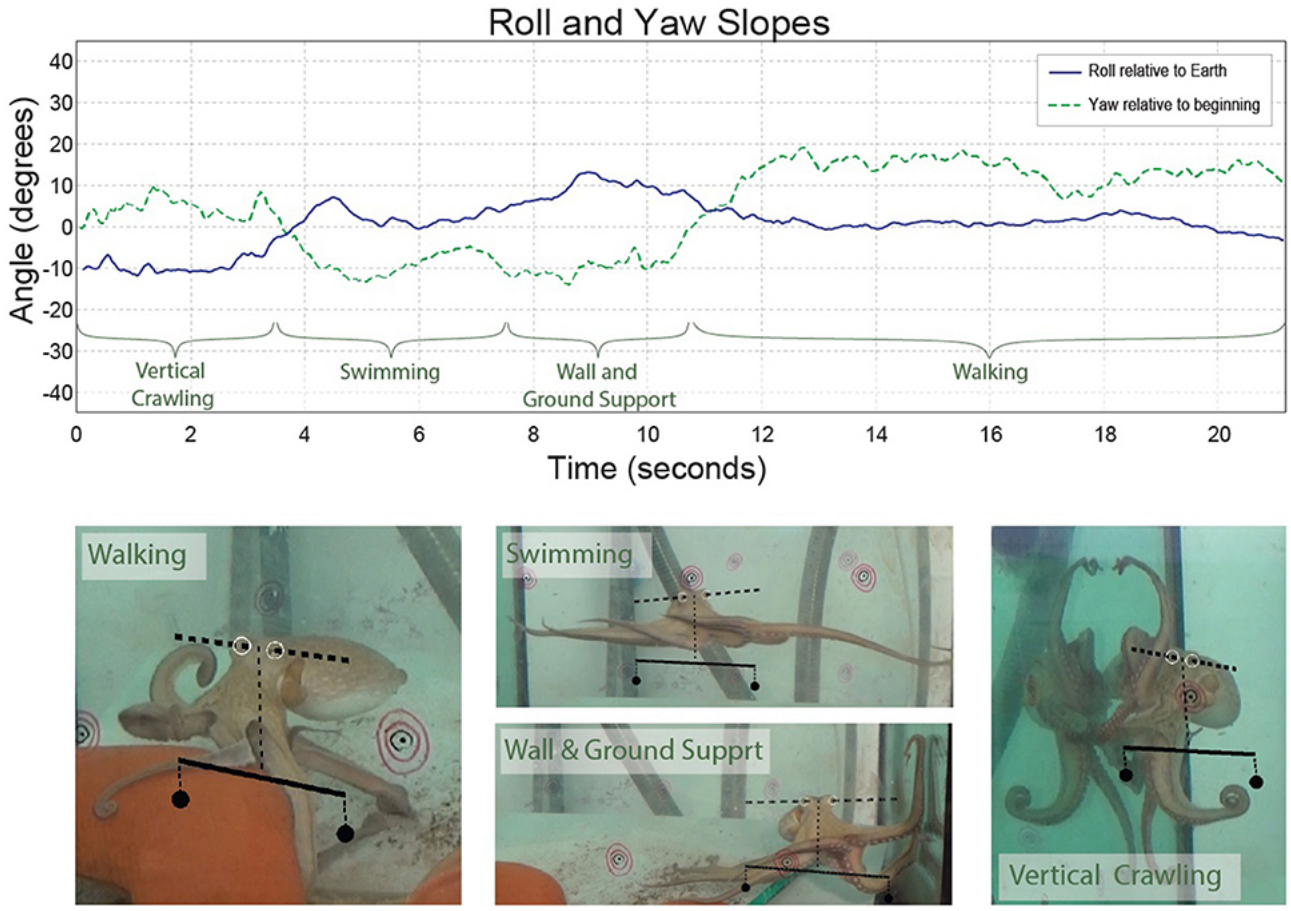

Рисунок 4 – Сверху: синяя линия – вертикальный угол между глазами. Можно заметить, что он довольно близок к 0, по сравнению, например, с горизонтальным отклонением (зеленая линия). Снизу: сравнение оси, на которой расположены глаза осьминога (штриховая линия) с горизонтальным положением относительно земли («весы»). [2]

В случае передвижения по дну у осьминога была обнаружена другая особенность: в отличие, например, от насекомых, переставляющих ноги в определенном порядке при ходьбе, осьминог на ходу принимает решения, каким щупальцем ему «шагнуть» и зацепиться за дно (рисунок 5). Так как все щупальца одинаковой длины и расположены симметрично, его мозг легко вычисляет, какое из них выгоднее использовать в следующую очередь [3].

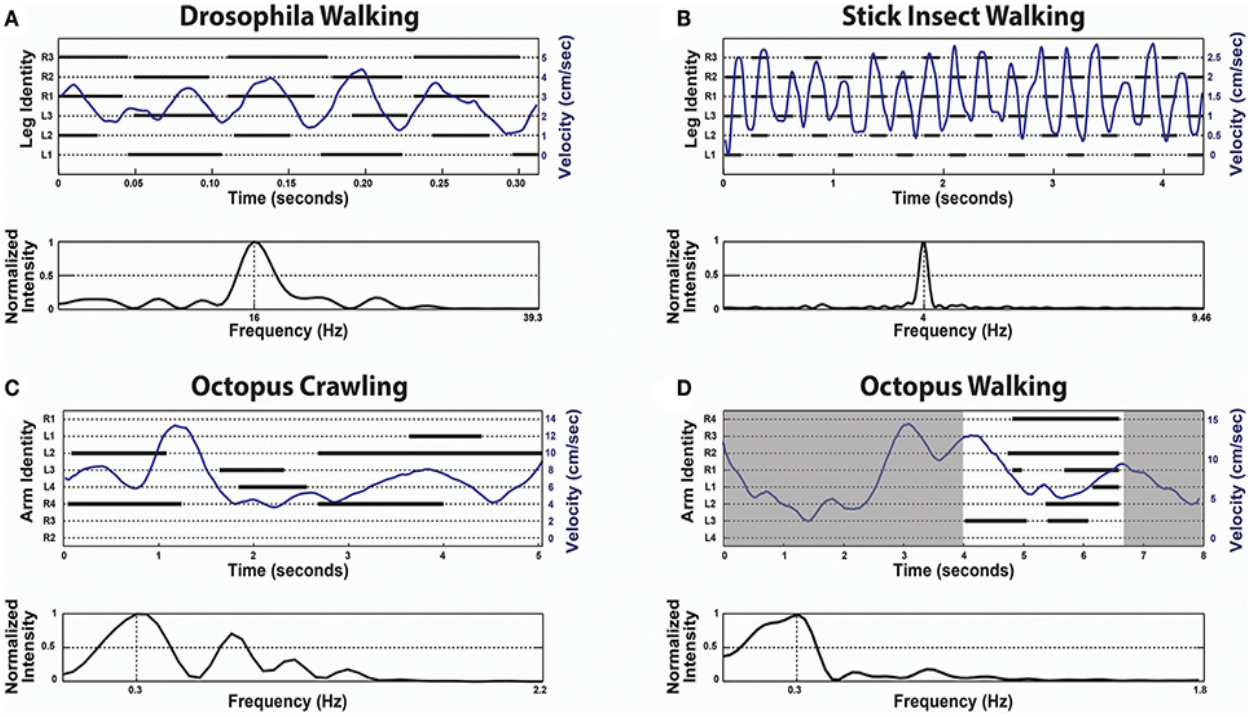

Рисунок 5 – Порядок использования различных щупалец, мгновенная скорость и частота движений осьминога (снизу) по сравнению с дрозофилой (слева сверху) и палочником (справа сверху). У осьминога движения выглядят случайными и не имеют повторяющегося ритма. [2]

Практическая значимость подробного исследования и моделирования движений осьминога проявляется, например, в робототехнике: их результаты используются для создания так называемых «мягких роботов», имитирующих щупальца осьминога. Подобные роботы могут выполнять более сложные и точные действия, чем роботы более жесткой конструкции, а также обеспечивают безопасное взаимодействие с человеком, из-за чего наиболее многообещающим представляется их применение в медицине [1, 6]. Однако из-за типичной конструкции подобных роботов (рука-манипулятор) при их проектировании интерес представляет в основном поведение отдельного щупальца, а не осьминога в целом. Вследствие этого именно хватательные и другие движения, совершаемые независимо одним щупальцем, а не перемещение осьминога в пространстве, на данный момент наиболее подробно исследованы: для них был выделен набор элементарных движений [8], построены сложные математические модели [6],

существуют даже уроки процедурной анимации для начинающих, такие, как [9], в качестве одного из базовых примеров использующие щупальце осьминога. Моделирование целого осьминога, с учетом координации движений всех щупалец, по-прежнему представляется существенно более редкой и сложной задачей.

Однако на текущий момент ситуация начала меняться. Уже в 2012 году была опубликована статья [10], предлагающая не только очередную, более биологически точную модель роботизированного щупальца-манипулятора, но и возможную конструкцию робота, имитирующего осьминога в целом, и алгоритм управления таким роботом. В дальнейшем появились работы, описывающие уже созданные прототипы подобных роботов, например, мягкий робот с гидравлическими активаторами, способный плавать и разворачиваться в воде, довольно точно имитирующий осьминога формой, материалом и способом управления [11], и прототип с использованием волокон Soft Memory Alloy в качестве мышц [12]. Предполагается, что таких роботов можно использовать для подводных наблюдений.

Так как было показано, что строение и поведение осьминога представляет как научный, так и практический интерес, очевидно, что моделирование его перемещения с помощью компьютерной анимации является актуальной задачей. Как показано, например, в статье [13], посвященной использованию вычислительной гидродинамики для моделирования движения человека и животных в воде, существует множество целей, для которых может применяться анимация: точное физическое моделирование для биологических исследований, имитация и анализ отдельных свойств животного (например, обтекаемости или реактивного движения) для создания упомянутых уже роботов, а также других инженерных разработок, например, кораблей и автомобилей, или визуальная имитация для использования в анимационных фильмах [14, 15] и в компьютерных играх [16].

Использование животных (и в частности, осьминогов) в играх может иметь различные цели. Они могут быть как просто элементами окружения, так и полноценными игровыми персонажами – противниками игрока (OctoRaid VR [17]), так и компаньонами (Octopus Bar [18]). Или даже главным персонажем, управляемым игроком (Octogeddon [19], Octodad: Dadliest Catch [20]). Тем не менее, не находится примеров в играх, где движения осьминогов были бы с высокой степенью биологически точными. Зачастую сама форма существ изменяется в угоду стилистике игры, а движения упрощены или скорректированы под поведение персонажа, принципиально не похожее на поведение реального осьминога.

В настоящий момент существует несколько известных методик моделирования движений существ:

Data-driven технология [21] основана на обработке данных, полученных от наблюдения движения реальных животных. Эта технология позволяет получить наиболее реалистичное изображение, однако ее трудно расширить за рамки контекста имеющихся данных или модифицировать для использования с другими формами или организмами.

Physics-based технология является наиболее общим подходом, который предполагает построение биомеханической модели существа и полного рассчета движений на базе этой детальной модели. Эта технология хорошо разработана для человекоподобных существ [22]. Однако расчет движений на основе модели требует больших вычислительных ресурсов. Поскольку осьминоги не имеют костей, то построение полной биомеханической модели затруднено, а если и возможно, то такая модель будет иметь слишком много деталей и расчет движений будет слишком сложным и долгим с точки зрения вычислительных ресурсов.

Процедурная техника использует набор математических формул, которые позволяют приближенно сгенерировать внешний вид движений тела без детальной модели внутреннего устройства. Эта техника является симбиозом эмпирического описания и упрощенной биомеханической модели. Такой подход оказывается

выгоднее с точки зрения нагрузки на вычислительные ресурсы и позволяет в реальном времени вычислять движение нескольких персонажей. Процедурная техника отстает от Physics-based с точки зрения соответствия движений модели натуральному поведению существа. Однако, для моделирования осьминога такой подход может оказаться оправданным, поскольку осьминоги редко попадаются на глаза в реальном мире и определенная потеря натуральности скорее всего не будет воспринята зрителем как недочет анимации. Для анимации в компьютерных играх, требующей выполнения вычислений и генерации изображения на ходу, данный подход кажется наиболее подходящим, так как производительность для современных компьютерных игр очень критична.

Моделирование движений осьминога можно отнести к классу анимации многоногих существ, таких как пауки. Поскольку осьминог не имеет скелета и движение его щупалец не описывается регулярными схемами, то анимация осьминога имеет свои особенности и в данный момент не сильно изучена с точки зрения построения математической модели. Таким образом, методика анимации многоногих существ может служить определенной базой и для осьминога.

Интересный подход к анимации многоногих существ изложен в [21]. Эта работа предлагает полностью процедурный метод расчета движений в реальном времени, основанный в первую очередь на целеуказании перемещения ног и анимации остальных частей тела на базе положения конечностей. Эта работа описывает контроллер движения конечностей, который принимает во внимание две основные фазы движения конечностей: опорная, когда конечность связана с поверхностью, и фаза переноса. Описанный контроллер движения предполагает достаточно ритмичное движение конечности и точную точку опоры. Особенности структуры осьминога требуют изменения такого подхода, поскольку движения щупалец не описываются определенной схемой и могут опираться на поверхность (или прикрепляться к какому-либо объекту) разными частями, то есть любой группой присосок.

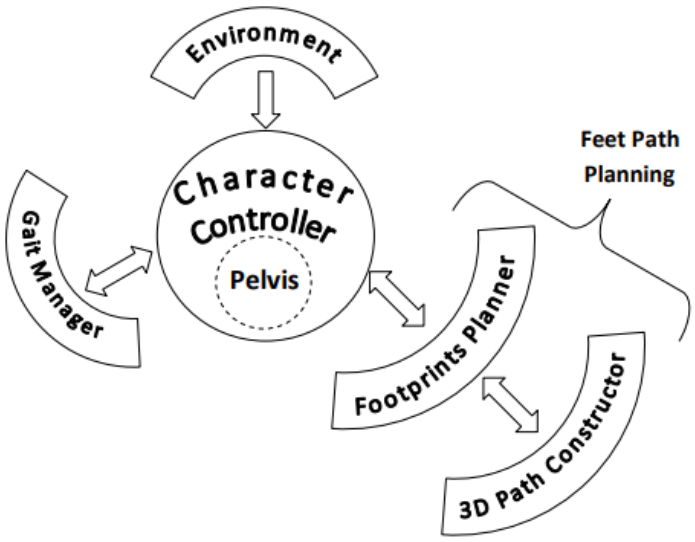

Рисунок 6 – Функциональные блоки контроллера движения многоногих существ, описанного в [21]

В дополнение к контроллеру движения конечностей [21] предлагает использовать контроллер персонажа для общей координации движения существа. Этот контроллер выполняет две существенный функции: управление движением конечностей и рассчет перемещения центра тяжести на основании заданного пользователем движения модели и текущего расположения конечностей на поверхности опоры.

Как и в случае контроллера движений конечностей, предлагаемый контроллер персонажа может плохо подходить для анимации осьминога, потому что форма осьминога сильно варьируется в зависимости от ситуации. Однако, центр тяжести системы тем не менее должен подчиняться законам физики. Таким образом контроллер персонажа может быть использован для вычисления положения центра тяжести осьминога, тогда как для вычисления поверхности его тела потребуются дополнительные механизмы.

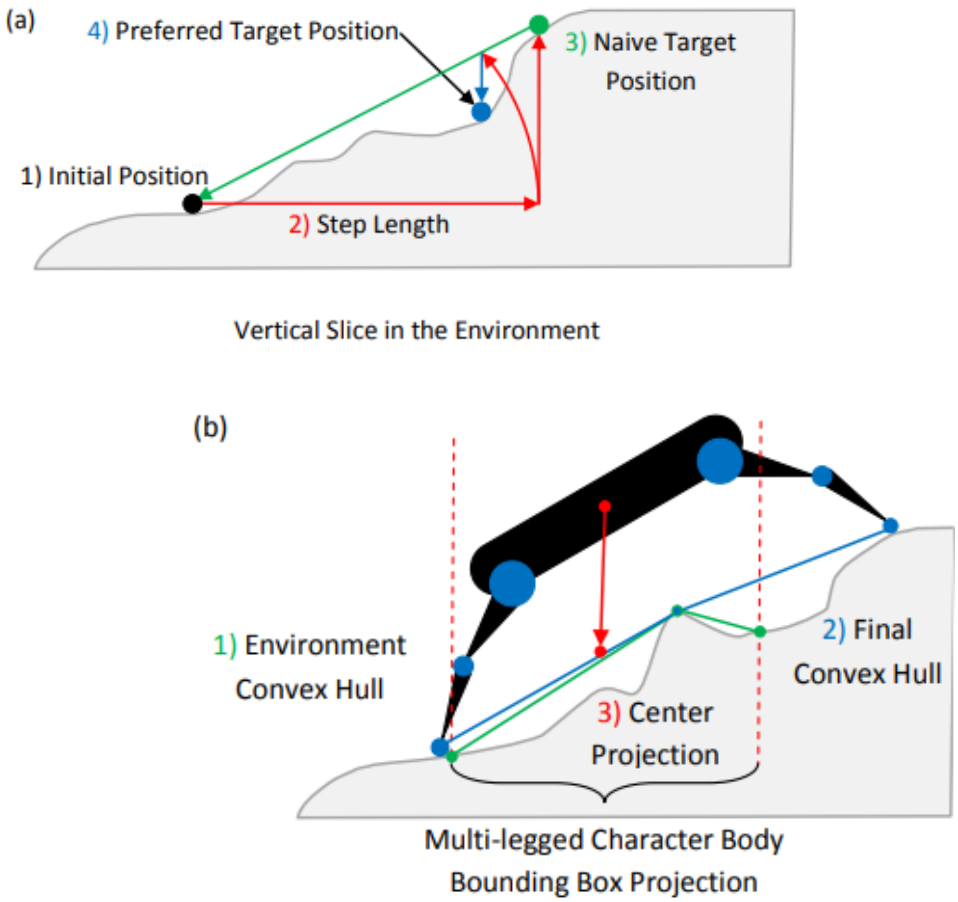

Рисунок 7 – Вычисление целевого положения конечности (a) и положения центра тяжести (b) персонажа в работе [21]

Другие подходы к анимации бесскелетных существ используются в анимационных фильмах, где нет необходимости генерировать анимацию в реальном времени, но при этом требуется большая точность и детализация изображения, чем в играх. Один из возможных подходов описан в коротком сообщении [23]. Этот вариант был реализован в анимационном фильме «Finding Dory» для персонажа Hank. Этот вариант анимации был построен путем создания моделей поверхности отдельно для тела со щупальцами, мантии и присосок осьминога. Модели были созданы с разными свойствами – так, мантия была разбита на более крупные ячейки для упрощения процесса модификации, так как предполагала меньше деформаций и сложных движений, а для присосок применялся предварительный алгоритм для вычисления моментов, в которые они

присасываются к поверхности и отпускают ее, а затем производилась симуляция действующих на них физических сил. Затем были определены возможные степени деформации поверхности и добавлены правила избегания коллизий (наложения одних элементов поверхности на другие).

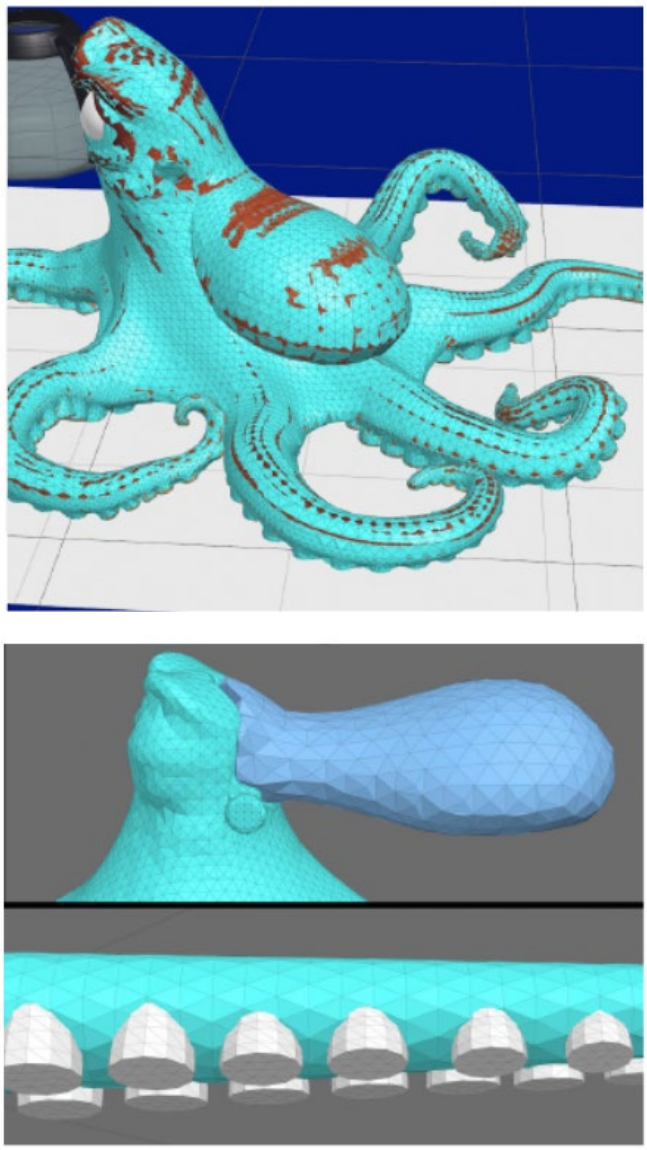

Рисунок 8 – Первая версия (сверху) и окончательный вариант (снизу) модели поверхности для анимированного персонажа Hank [23]

Другое решение данной задачи описано в заметке [24], где оно применялось для анимации персонажей-осьминогов из серии мультфильмов «Penguins of Madagascar». В заметке перечислены несколько проблем, специфичных для

анимации существ, лишенных скелета, и их решения. Так, например, для генерации движений щупалец был переработан алгоритм прямой и инверсной кинематики, ранее использовавшийся для движения хвостов животных, с добавлением большого объема функциональности для обеспечения реалистичной гибкости, например, возможности смещать опорную точку изгиба вдоль щупальца. Также для создания выразительного стилизованного силуэта и облегчения работы аниматоров число щупалец было уменьшено до шести, а в сценах, где неверное их количество оказывалось критичным или слишком заметным, добавлялись вспомогательные щупальца, не присоединенные к осьминогу и управляемые отдельно от него. Остальные описанные проблемы в основном связаны с комбинацией естественной и реалистичной структуры и движений осьминога в целом и более антропоморфным и стилизованным лицом и поведением, требуемым для персонажа. В заметке утверждалось, что полученная методика анимации позволяет добавлять вспомогательных персонажей-осьминогов даже в процессе производства фильма.

Единственная найденная на данный момент работа об анимации плавающего осьминога со ссылкой на код – статья [25]. В этой статье предложен алгоритм анимации разных видов морских животных с мягким телом, в том числе с возможностью задавать структуру животного через пользовательский интерфейс. Алгоритм основан на моделировании принципов распространения нервных импульсов в мышцах, на базе которых с помощью глубокого обучения реализуется перемещение в воде и обход препятствий.

Наибольший интерес представляет уже упомянутая статья [25], как единственный доступный пример решения поставленной задачи. Алгоритм, описанный в этой статье, не специфичен для осьминога, а обобщен на разные виды морских животных с мягким телом. Основан предлагаемый алгоритм на моделировании принципов распространения нервных импульсов в мышцах. Тело животного формируется как структура из треугольной сетки, и в дальнейшем

координаты и скорость каждой дискретной «точки» тела и воздействующие на них силы рассчитываются по методу конечных элементов. При этом для имитации реального мягкотелого животного, состоящего в основном из мышц, структура считается обладающей внутренней эластичностью. Мышечные волокна также аппроксимируются треугольными ячейками, через которые пропускаются нити нервов. Каждый нерв при подаче сигнала активирует все мышцы, находящиеся в заданном радиусе от него, после чего мышцы сокращаются, генерируя силу, которая деформирует модель тела. Для активации нервов авторы создали собственную модель с адаптивным распространением сигнала.

Для анимации целенаправленного движения животных в воде использовалось обучение с подкреплением. В результате были получены, в том числе для осьминога, достаточно естественно выглядящие анимации плавания, обхода препятствия и прохождения через узкое место. Отдельно был создан интерфейс, позволяющий пользователю вручную задавать структуру животного, на текущий момент только двумерную.

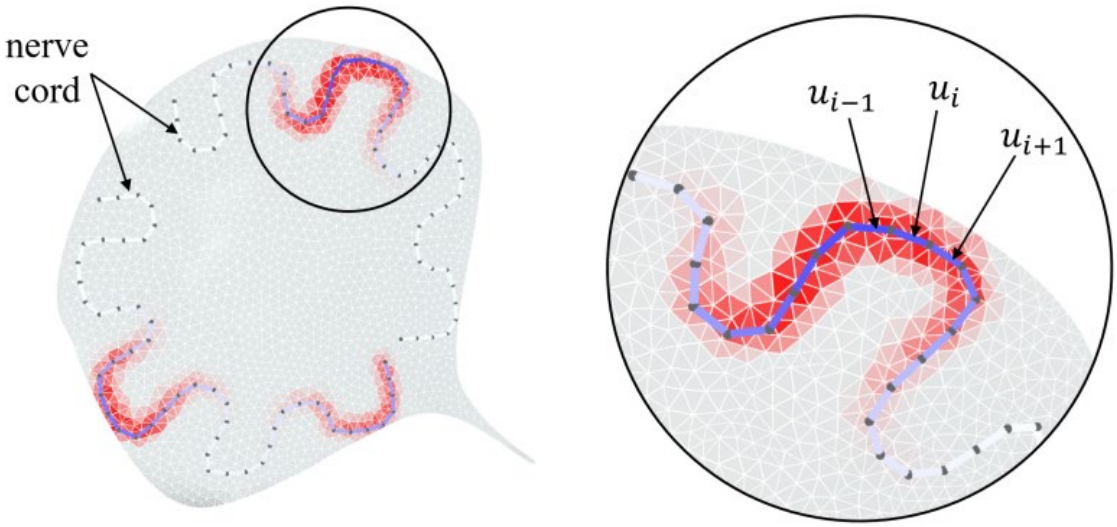

Рисунок 9 – Активация мышечных волокон, аппроксимированных треугольными элементами (синим показан сигнал активации, красным – активированные элементы). [25]

В целом данное решение представляет большой интерес, совмещая биологическую точность и относительную математическую простоту. Однако в самой статье указаны недостатки алгоритма: скорость анимации на персональном компьютере в 3–5 раз меньше реальной из-за обнаружения и обработки столкновений, а также нет возможности воспроизвести некоторые способы передвижения, как, например, реактивное движение осьминога, из-за упрощенной гидродинамической модели. Проблема скорости очень критична для процедурной анимации, к тому же биологическая точность движений не может быть гарантирована, так как принцип построения оптимальной траектории с помощью обучения с подкреплением может существенно отличаться от логики координации осьминога.

Предложенное решение [25] было проверено на ноутбуке (с процессором Intel® Core™ i7-10875H (8 ядер, 16 потоков), объемом оперативной памяти 32 ГБ DDR4-2666 и видеокартой NVIDIA® GeForce RTX™ 2060 с 6 ГБ GDDR6). В результате проверки была подтверждена работоспособность решения. Однако даже в базовом демонстрационном примере (движение одного осьминога при отсутствии окружения) отмечена низкая производительность (не более 10-15 кадров в секунду). Поэтому необходимо либо провести его модернизацию, либо использовать другое решение. Но это является предметом дальнейших исследований.

Несмотря на то, что других реализаций процедурной анимации осьминога найдено не было, имеет смысл провести анализ существующих математических моделей движения осьминога, например, используемых при создании роботов.

Одна из таких моделей представлена в работе [10]. В ее основе трехмерная модель щупальца осьминога, которые затем объединяются в систему, способную реализовывать такие модели передвижения, как ползание и плавание.

В этой архитектуре управление основано на нейрофизиологии осьминога: контроллер планирования более высокого уровня (аналог центральной нервной системы, CNS), расположенный в мозгу осьминога, отправляет желаемую

информацию о движении и времени его завершения на контроллеры, отвечающие за отдельные щупальца (аналог периферической нервной системы, PNS). Они обеспечивает контроль на более низком уровне, преобразуя информацию для избирательной активации мышц в нужное времени. Обратная связь в данной системе при управлении не применяется, но ее планировалось авторами добавить в будущем.

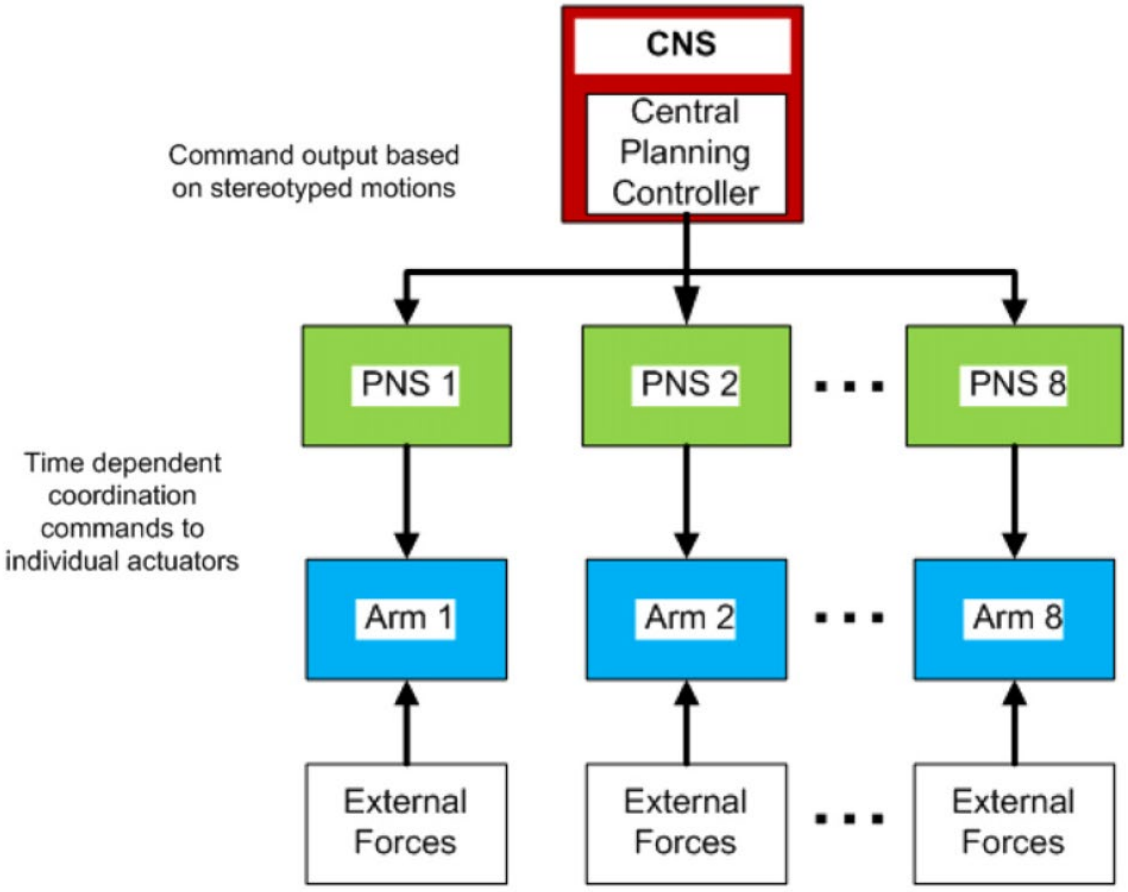

Рисунок 10 – Схема работы системы [10]

Сами щупальца состоят из 20 сегментов с параллельной структурой срабатывания, чтобы имитировать их непрерывную структуру и нужную форму. В модели учитывается наличие радиальных и продольных мышц, сохранение объема при сокращении, гидродинамические силы и силовое взаимодействие между присосками и окружающими предметами.

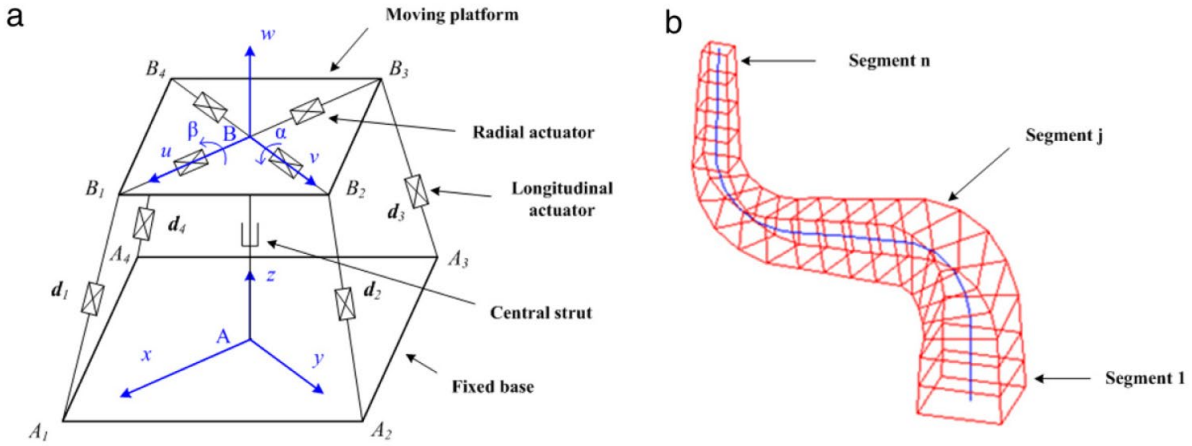

Рисунок 11 – Модель отдельного сегмента щупальца (a) и его целиком (b) [10]

Авторами статьи было проведено моделирование ползания и плавания в MATLAB/SIMULINK. Тем не менее, в статье не приведены измерения производительности, а сама реализация модели, к сожалению, отсутствует в свободном доступе.

Найденные материалы подтверждают актуальность задачи создания процедурной анимации осьминога: существует большое количество исследований поведения осьминога, много попыток его моделирования, однако не все результаты могут быть использованы для генерации анимации. Существуют математические модели для описания движения, часто целью проводимого моделирования было решение различных робототехнических задач, но на данный момент существующие прототипы имеют много недостатков. Если же говорить о «визуальной» анимации, особенно генерируемой в реальном времени, например, для создания персонажей в компьютерных играх, то здесь существующих результатов значительно меньше: на данный момент удалось найти только одну реализацию процедурной анимации осьминога, при этом обладающую недостаточной вычислительной эффективностью для использования в реальном времени. Из этого следует, что эффективный алгоритм для процедурной анимации осьминога будет иметь научную новизну и практическую значимость.

2 Исследование и оптимизация базового решения

В результате поиска источников был найден только один проект, по своим результатам близкий к целям данной работы. В статье [25] представлен обобщенный алгоритм для моделирования движений различных морских животных с мягким телом, в том числе осьминогов, за счет имитации распространения нервных импульсов в мышцах. Для анимации целенаправленного движения животных в воде использовалось обучение с подкреплением. В результате были получены, в том числе для осьминога, достаточно естественно выглядящие анимации плавания, обхода препятствия и прохождения через узкое место. В целом данное решение представляется перспективным для использования, однако оно имеет некоторые недостатки, не позволяющие использовать его на практике без изменений. Во-первых, при запуске на устройстве, даже с учетом отсутствия окружения и всего одного осьминога на сцене, полученная производительность оказалась очень низкой для практического использования – от 5 до 15 кадров в секунду. О низкой производительности предупреждали и авторы исходной статьи. Во-вторых, из движений осьминога в данном проекте было реализовано только плавание, и в демонстрационных примерах было видно, что он пытается плавать, даже находясь вплотную ко дну, в то время как в текущей работе планируется реализовать также по крайней мере один способ перемещения по дну. Следовательно, необходимо также оценить, возможно ли его реализовать на базе существующей реализации плавания.

В основе математической модели, используемой в этом алгоритме, лежит метод конечных элементов. Модель животного (в нашем случае осьминога, но авторы тестировали алгоритм и на моделях других, в основном более простых беспозвоночных животных) строится как сетка, состоящая из конечного числа вершин. Для каждой из точек в каждый дискретный момент времени вычисляется ее положение в пространстве и скорость на следующем шаге, исходя из текущих положения и времени, а также массы и двух обобщенных сил, действующих в этой

точке – внешней и внутренней. Внутренняя сила возникает из деформации эластичных мышц, а внешняя вычисляется из упрощенной модели гидродинамики и складывается, в свою очередь, из двух сил: силы сопротивления и силы тяги. Сила сопротивления сонаправлена со скоростью движения относительно воды, а сила тяги, которая и является источником движения, направлена противоположно нормали к поверхности. Для более эффективного решения системы используется метод проективной динамики.

Для корректной симуляции важен правильный выбор материала. Так как осьминоги и подобные им животные состоят в основном из мышц, их ключевое свойство – эластичность. Сама эластичность не зависит от активации мышц, но к ней добавляется напряжение, генерирующее внутреннюю силу. Для эластичности была выбрана модель, использующая градиент деформации элемента, вычисляемый с помощью SVD-разложения. Мышцы аппроксимируются плоской треугольной или объемной тетраэдральной сеткой, через которую проходят нервные волокна, также аппроксимированные цепочками сегментов. По нервному волокну проходит сигнал, активируя все элементы сетки в заданном радиусе от волокна, причем уровень активации зависит от расстояния до нерва (рисунок 9). Каждый элемент сжимается вдоль ближайшего нерва пропорционально уровню активации. Сжатие (сокращение мышц) генерирует внутреннюю силу, вычисление которой различается для плоской и объемной сетки. В модели осьминога используется только объемная сетка, так как с помощью плоской генерируются только тонкие элементы, например, плавники, которые отсутствуют у осьминогов.

Отдельной задачей был выбор алгоритма для активации мышц. Авторы рассматривали и сравнивали несколько базовых моделей и на их основе создали свою – модель с адаптивным распространением сигнала. Для ее разработки изначально были рассмотрены два крайних случая. Первый – модель, в которой нервные сигналы в каждом сегменте каждого волокна полностью независимы. Эта модель дает максимальное разнообразие движений, но по этой же причине плохо

контролируется и обучается, и, как показали результаты тестирования, движения получаются хаотичными и неестественными. Противоположный случай – все нервные волокна передают один и тот же исходный сигнал. Такая модель получается слишком ограниченной и не позволяет генерировать сложные движения. Компромиссный вариант был уже предложен ранее – в нем сигналы цикличны и все еще исходят из единого центра (мозга), но по разным волокнам могут передаваться разные сигналы. Авторы статьи развили модель еще дальше с учетом особенностей нервной системы осьминога. В различных исследованиях (например, [2]) была описана способность осьминога активно модулировать нервные импульсы, полученные из мозга, а иногда и генерировать их, в любой части щупальца. Созданная на основе этих наблюдений модель с адаптивным распространением сигнала по-прежнему содержит центральный генератор импульсов, однако полученные импульсы в каждом сегменте волокна могут модулироваться по скорости и амплитуде. В качестве базового вида сигнала была взята треугольная волна с добавленной сразу задержкой передачи сигнала для большей реалистичности. В каждой следующей точке вдоль волокна значение сигнала умножается на матрицу, включающую в себя модуляцию от времени и от расстояния, последняя из которых определяется параметрами затухания (определяет, какая часть от текущего уровня сигнала останется в той же точке) и передачи (какая часть будет передана в следующий сегмент). Основное отличие от предыдущей модели состоит в том, что параметры модуляции могут изменяться со временем. Пример модуляции нервных импульсов представлен на рисунке 12.

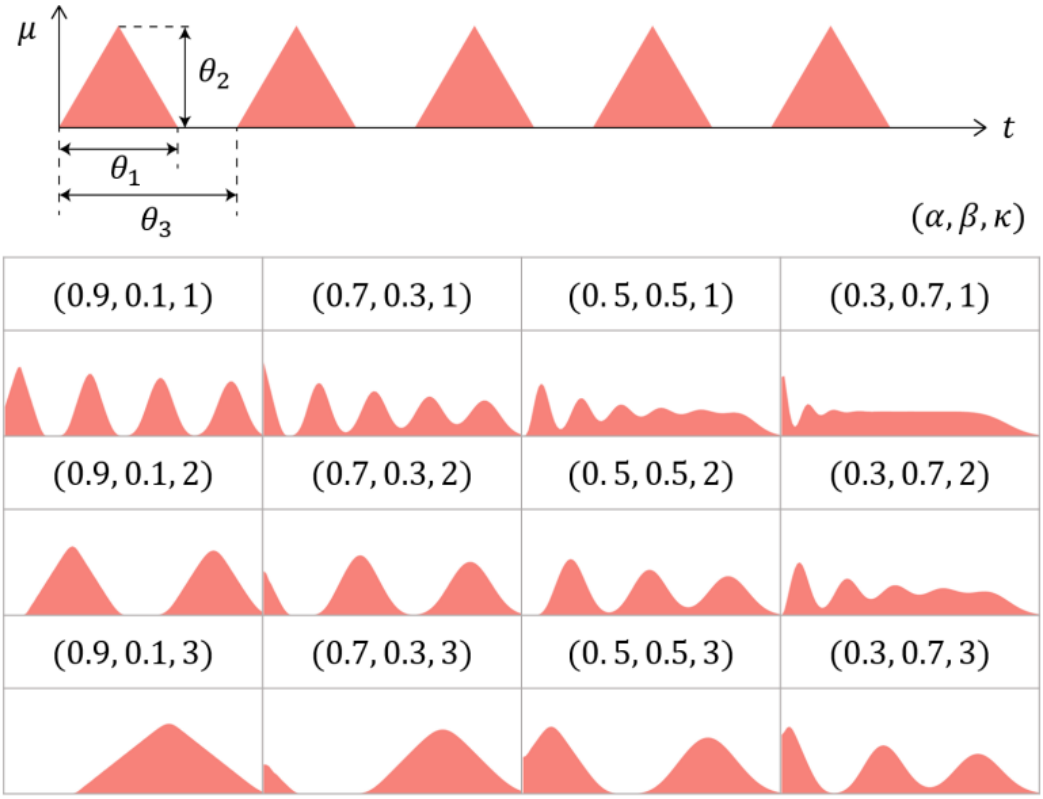

Рисунок 12 – Распространение исходного нервного импульса (сверху) в каждом следующем сегменте при различных наборах заданных и не меняющихся со временем параметров затухания, передачи и модуляции от времени [25]

Для установления связи между нервными импульсами и целенаправленным движением было использовано обучение с подкреплением, где в качестве параметров состояния выступали требуемая скорость, направление движения и вектор, направленный вверх в глобальных координатах. Полученная модель осьминога обучалась на 4-ядерном CPU (i7-7700) порядка 48 часов (для более простых животных время обучения было меньше), после чего могла плыть в заданную точку, прятаться и преодолевать препятствия.

Помимо обнаруженных уже проблем с производительностью, одна из проблем была найдена уже при наблюдении за внешним поведением модели. При запуске упомянутого уже демонстрационного примера было замечено, что осьминог при плавании поворачивается вокруг своей оси при каждом отталкивании щупальцами. Данная проблема не была замечена ранее по причине использования

в примере модели осьминога с полной центральной симметрией. Подобная техника плавания существенно отличается от реального поведения осьминога. Более того, как было указано в статье [2], одна из характерных особенностей передвижения осьминога состоит в том, что при любом способе движения и любой ориентировке в пространстве его голова не меняет угол относительно земли, и линия глаз всегда остается горизонтальной. Поэтому если строить новый алгоритм на естественных для осьминога ограничениях, данное отклонение от реальной техники плавания недопустимо как с точки зрения визуального восприятия – любое животное, постоянно вращающееся вокруг своей оси при движении, даже для неосведомленного зрителя будет выглядеть неестественно – так и с точки зрения фундаментальной структуры алгоритма. В дальнейшем необходимо будет выяснить причину такого поведения и установить явное ограничение вне зависимости от того, будет ли использоваться код данной программы по крайней мере частично и будет ли сохранено машинное обучение и текущий подход к нему.

Далее мы вернулись к проблеме производительности. Для выявления слабых мест в алгоритме было выполнено профилирование с помощью инструмента perf [26], а полученные данные были визуализированы с помощью Flame Graph [27]. Были получены результаты, представленные на рисунке 13. Из них можно увидеть, что первые 9 процессов, занимающие больше всего времени при выполнении, представляют собой системные вызовы и простои из-за ожидания результатов. Было проверено, что в программе реализована многопоточность (с помощью интерфейса OpenMP [28]), однако фактически она используется только на этапе обучения модели. В одной из наиболее тяжелых по производительности частей алгоритма, которая упоминается далее, была закомментирована команда по параллельному выполнению цикла, но из-за того, что в той же части алгоритма была объявлена критическая секция, производительность от этого изменения не повысилась.

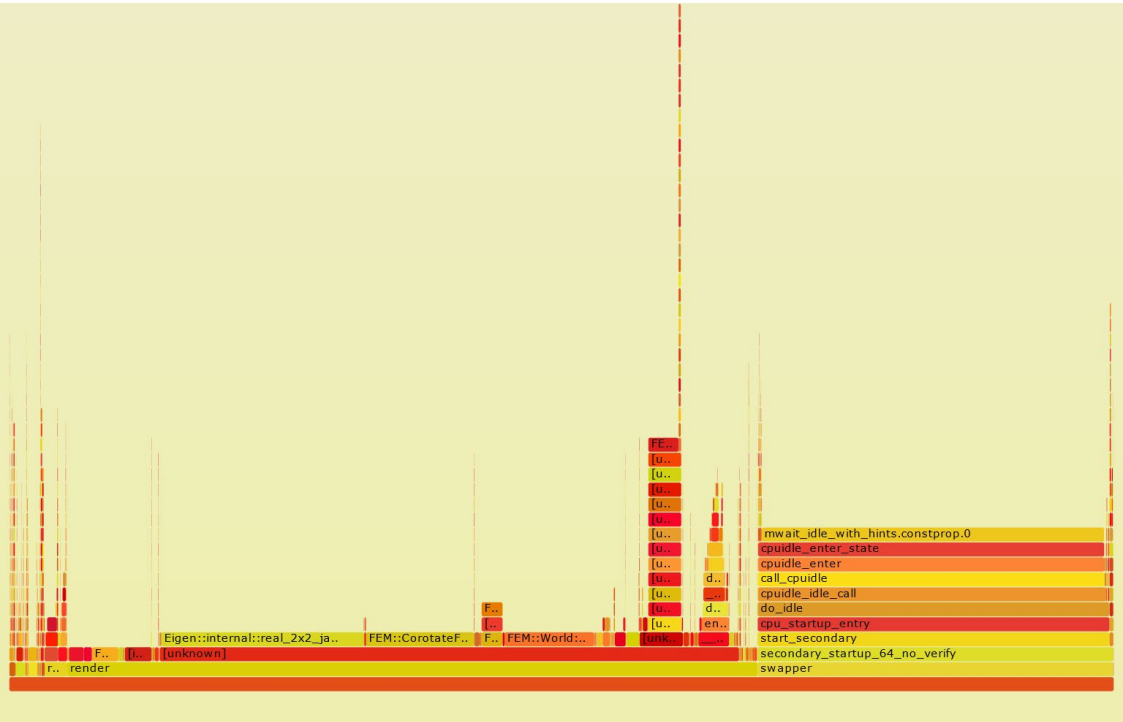

Рисунок 13 – Результат профилирования базовой программы

Оставшиеся процессы, занимающие более 5% от времени выполнения, являются операциями над матрицами, которые используются для реализации SVD-разложения. В результате анализа кода было обнаружено, что разложение выполняется в цикле для большого количества (около 6000) разных матриц. Как было упомянуто ранее, функция SVD-разложения была объявлена как критическая секция, но при ближайшем рассмотрении выяснилось, что все задействованные в разных итерациях матрицы находятся в различных объектах и внутри цикла никак друг на друга не влияют, поэтому было решено, что цикл можно исполнять в несколько потоков. Это стало первой из выполненных оптимизаций, после которой производительность заметно улучшилась. Новое распределение времени выполнения приведено на рисунке 14. Можно заметить, что и функции, связанные с SVD-разложением, и простои занимают существенно меньшую часть от общего времени выполнения.

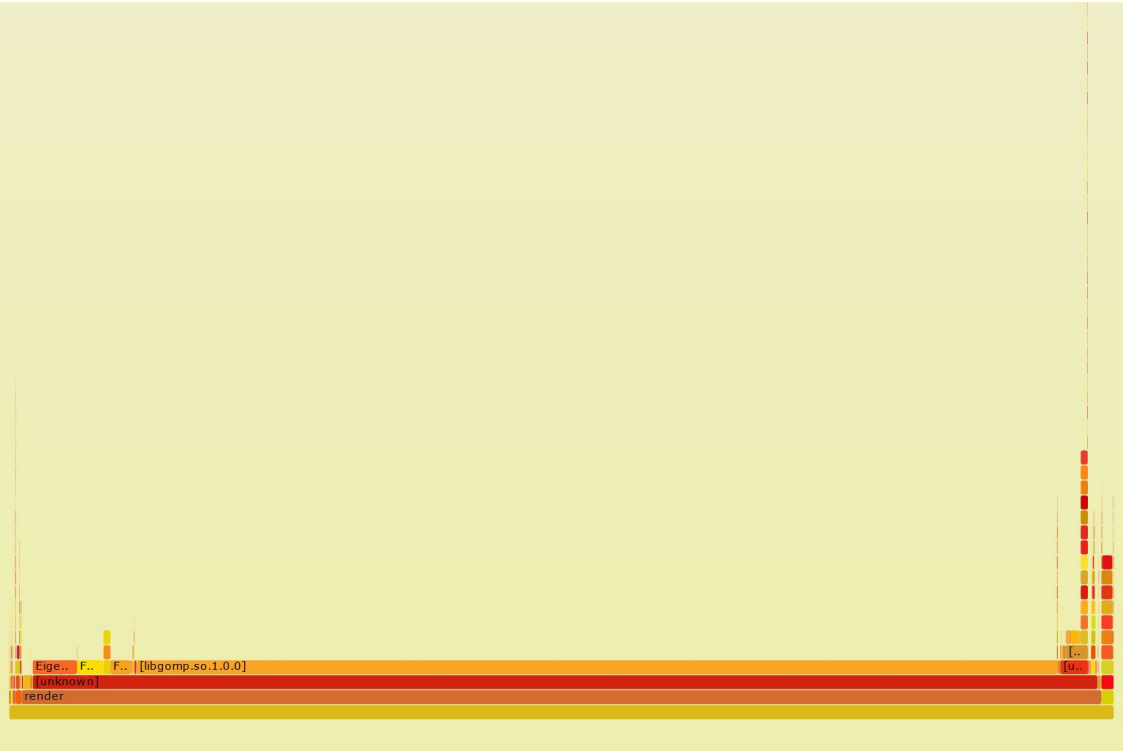

Рисунок 14 – Результат профилирования программы при добавлении многопоточности

Однако по причине того, что производительность после данной оптимизации не достигла минимально допустимого уровня, и по диаграммам можно также заметить, что SVD-разложение все еще выполняется заметно дольше всех остальных вычислений, стала очевидной необходимость следующей оптимизации. Было замечено, что если использовать для вычислений видеокарту, то число доступных потоков должно быть достаточным для того, чтобы вычислять разложение одновременно для всех матриц [29]. Однако для этого необходимо, чтобы во всех вычислениях, которые будут выполняться на видеокарте, использовалась не двойная, а одинарная точность. После перехода к одинарной точности качество анимации визуально не снизилось, однако производительность выросла не намного. Но так как этот этап в основном рассматривается как подготовка для дальнейшего переноса вычислений на видеокарту, имеет значение в основном отсутствие ухудшений качества. Диаграмма для данного этапа оптимизации приведена на рисунке 15.

Рисунок 15 – Результат профилирования программы после перехода к одинарной точности

Кроме построения диаграмм по результатам профилирования, были выполнены измерения частоты кадров для каждого из вариантов реализации, и на их основе была построена таблица 1.

Таблица 1 – Минимальная и средняя частота кадров и загрузка процессора для разных вариантов реализации

	Исходное решение	OpenMP	Одинарная точность
0.1% Min FPS	6.5	10.6	10.8
1% Min FPS	6.7	10.7	11.2
Average FPS	18.3	26.3	35.6
Average Frame Time, ms	83.8	56.6	50.0
CPU Load, %	4.7	58.8	57.9

Из таблицы можно увидеть, что степень загрузки процессора действительно после первой оптимизации. При этом среднюю частоту кадров удалось повысить примерно в два раза, однако она оставалась недостаточно высокой для практического использования.

В дальнейшем, кроме использования вычислений на видокарте, в качестве еще одной возможной оптимизации можно предложить замену самой функции SVD-разложения на более качественную реализацию, если таковая существует.

3 Работа с обучением модели

Для оптимизации и тестирования базового алгоритма на этапе оптимизации использовался способ запуска, описываемый авторами программы как запуск демонстрационного примера. Этот пример не имеет входных параметров, и при данном варианте запуска модель осьминога следует заданному заранее фиксированному маршруту. При попытке внедрить в программу инструменты для управления и запустить данный пример выяснилось, что при таком способе запуска программы это не представляется возможным. Для создания модели с возможностью управления необходимо запускать программу с обученной нейронной сетью в качестве входного параметра. Также выяснилось, что по крайней мере реализация, существующая в открытом доступе, не включает поиск оптимального маршрута. Нейронная сеть используется только для реализации минимальных, «единичных» движений, а результатом обучения должна являться связь между движением в выбранном направлении с выбранной скоростью и последовательностью нервных импульсов, требуемой для этого движения. Так как получить нужную последовательность импульсов аналитически было бы невозможной или очень трудоемкой задачей из-за большого количества настраиваемых параметров, предварительное обучение модели и дальнейшее ее использование для работы с анимацией было признано наиболее целесообразным решением.

В программе используется обучение с подкреплением на основе глубоких сетей с тремя полностью связанными уровнями по 128 нейронов в каждом, с использованием функции гиперболического тангенса в качестве функции активации. Обучение с подкреплением состоит в том, что агент (в данном случае, модель осьминога), совершая некоторое действие, переходит из состояния, в котором находился, в некоторое другое состояние, и за каждый переход между состояниями получает вознаграждение, величина которого зависит от того, насколько желателен данный переход. Когда цепочка переходов между

состояниями закончится по истечении заданного промежутка времени или по другому условию, все вознаграждения суммируются. Задача обучения – получить функцию политики (управления), обеспечивающую максимальное значение вознаграждения. При обучении используется алгоритм Proximal Policy Optimization [30]. Особенность этого метода состоит в том, что в функции потерь, используемой в нем, задается ограничение, не позволяющее резко менять параметры нейронной сети, кроме случаев, когда на предыдущей итерации была допущена ошибка и нежелательное состояние оказалось чересчур вероятным или, напротив, желательное состояние стало практически невозможным – ограничение задано так, чтобы в подобных случаях по возможности отменять изменения, добавленные к нейронной сети при некорректной итерации.

В случае модели осьминога состояние определяется тремя факторами:

– Динамическое состояние – координаты и скорости в вершинах сетки (используются не все вершины, а некоторое случайное подмножество вершин, расположенных близко к нервным волокнам).

– Состояние задачи – целевой вектор скорости, задаваемый пользователем.

– Состояние активации мышц – нервные импульсы и их параметры модуляции от времени и расстояния.

Функция вознаграждения, приведенная в статье [25], построена так, чтобы давать максимальный результат при минимизации трех параметров: разницы между текущим и целевым вектором скорости, разницы между текущим и целевым направлением движения (нормализованным вектором скорости) и отклонением направления «вверх» для осьминога от глобального направления «вверх». Также в функцию может быть добавлен дополнительный пользовательский параметр произвольного вида для постановки какой-либо новой задачи обучения.

Однако при анализе программного кода было обнаружено, что по крайней мере в текущей открытой версии реализованы только первые два условия. Вследствие отсутствия условия о сохранении вертикальной оси и возникает

проблема, обнаруженная на предыдущем этапе работы, связанная с вращением осьминога вокруг своей оси при движении, что противоречит принципу ориентировки реального осьминога в пространстве, состоящему в расположении линии глаз горизонтально относительно земли при любом способе передвижения [2]. Из этого следует необходимость реализовать по крайней мере более частное условие о нахождении глаз на одном уровне, для более биологически точной и аккуратно выглядящей (так как при использовании модели или текстуры, не симметричной относительно центральной оси осьминога, вращение будет заметно) анимации. Также данное условие может оказаться необходимым для реализации передвижения осьминога по дну, рассматриваемого в одном из следующих разделов.

Так как авторы статьи указывали, что обучение модели осьминога занимает около 48 часов на 4-ядерном CPU (i7-7700), было принято решение использовать для тестирования устройство с наиболее мощным процессором из доступных – Intel Core i5 14600K с 14 ядрами. Так как, исходя из данных, предоставленных платформами сравнения характеристик, такими, как [31], производительность процессора Intel Core i5 14600K при использовании всех ядер может быть до 4-5 раз выше, чем у используемого авторами статьи, было сделано предположение, что время обучения с использованием данного процессора должно составить около 12 часов.

При попытке запустить обучение в соответствии с предоставленной авторами инструкцией мы столкнулись с проблемой, не позволяющей запустить скрипт, отвечающий за обучение. Так как проект был закончен в 2019 году, используемые в нем версии языка Python и различных его модулей, в том числе таких базовых, как numpy и scipy, а также библиотеки Boost, существенно отличаются от наиболее современных версий. При попытке использовать новейшие версии, во время запуска программы возникали ошибки из-за отсутствия обратной совместимости. Поэтому на данный момент мы были вынуждены перейти на версию операционной

системы Ubuntu 20.04, позволяющую использовать требуемые версии модулей. В дальнейшем было бы желательно переписать скрипты в соответствии с современной функциональностью языка Python.

После указанных модификаций процесс обучения был успешно запущен. Обучение продлилось около 20 часов, но величина вознаграждения приблизилась к максимальной и перестала существенно расти уже спустя 12 часов после запуска, что соответствует исходным предположениям о времени обучения. На рисунке 16 показаны результаты обучения в виде вознаграждения, полученного на каждом этапе. Можно увидеть, что общая сумма вознаграждения сходится к значению около 800.

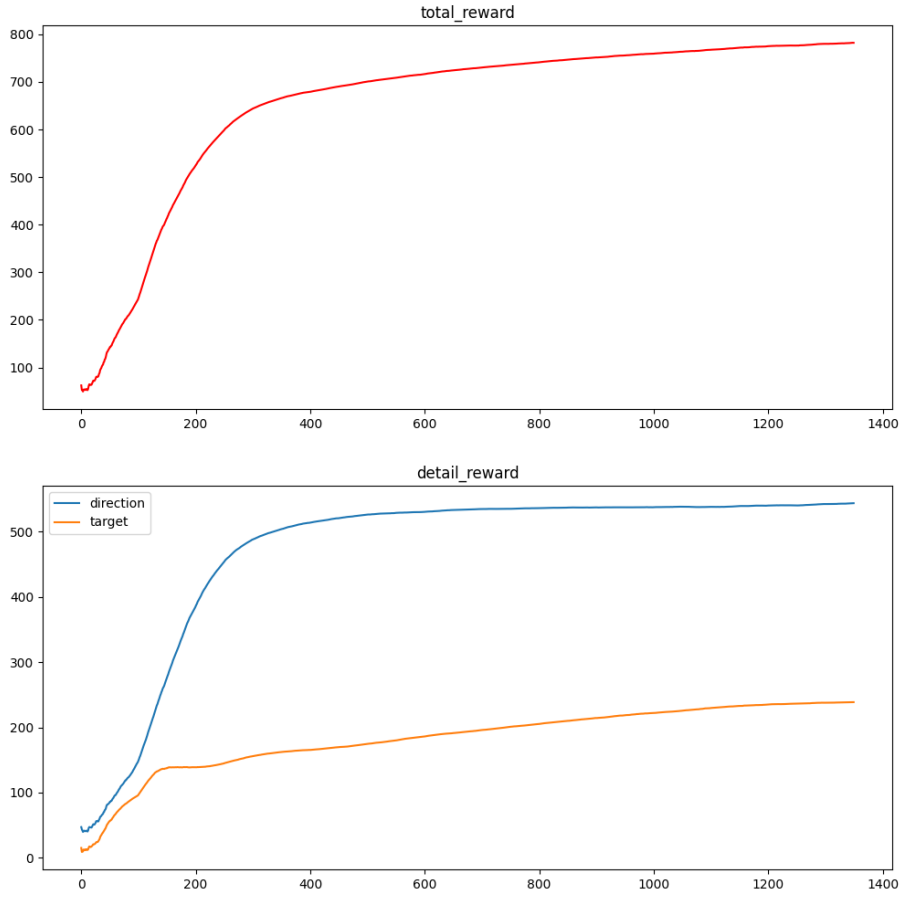

Рисунок 16 – Значение вознаграждения на всех этапах обучения. Сверху – общая сумма вознаграждения, снизу – отдельные составляющие (синий график – вознаграждение, связанное с направлением движения, оранжевый – вознаграждение, связанное с вектором скорости).

По результатам обучения была получена модель осьминога с возможностью задавать скорость и направление движения. На ее основе был реализован простой контроллер с изменением направления движения по нажатию клавиш.

На рисунке 17 можно увидеть, как обученная модель осьминога реагирует на задание нового направления движения. По нажатию одной из 6 клавиш, отвечающих за управление, осьминогу задается новый целевой вектор скорости. Далее осьминог начинает плавно разворачиваться в заданном направлении, пока его вектор текущей скорости не совпадет с требуемым, после чего продолжает двигаться прямо, пока целевой вектор не поменяется снова. В дальнейшем планируется реализовать контроллер с более плавным управлением, основанным на локальной системе координат осьминога.

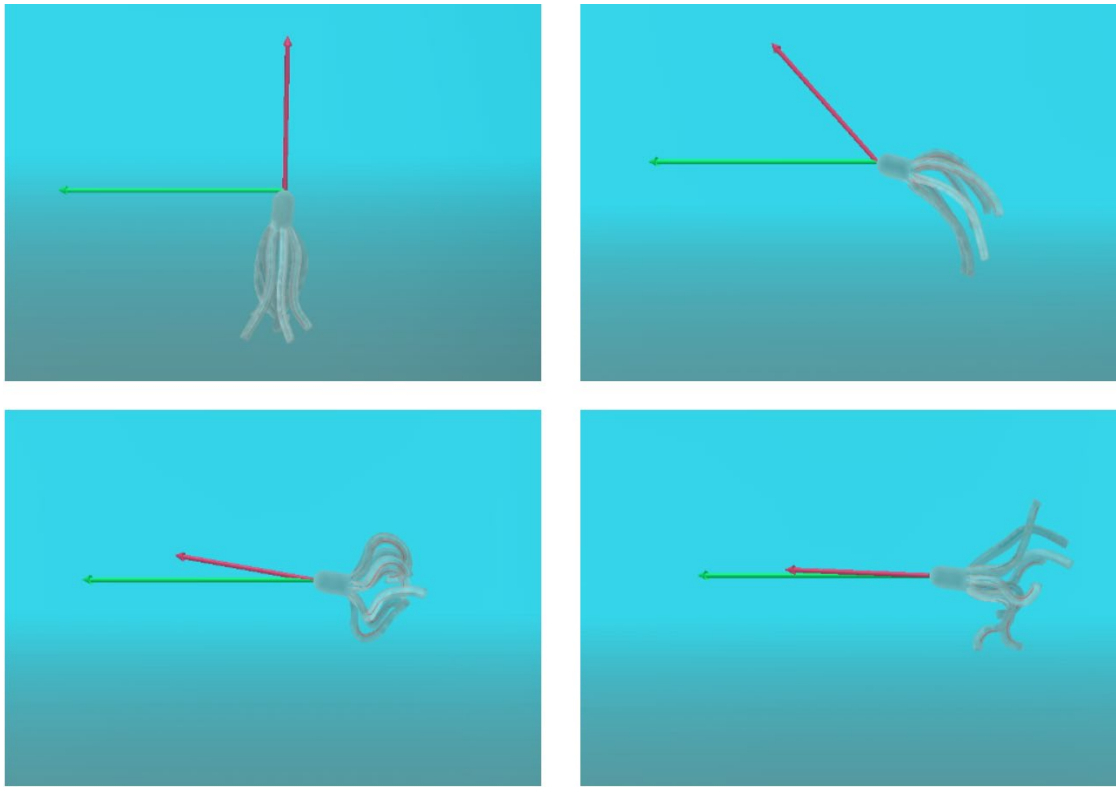

Рисунок 17 – Управление моделью осьминога. Красный вектор – текущая скорость осьминога, зеленый – целевая скорость.

На следующем этапе работы была предпринята попытка решить описанную выше проблему, связанную с вращением осьминога вокруг своей оси при каждом

отталкивании щупальцами. При этом ограничение, связанное с направлением «вверх», предложенное в статье [25], было бы слишком жестким, так как сохранило бы возможность осьминога поворачиваться только в горизонтальной плоскости. Необходимо было задать условие обучения, ограничивающее вращение осьминога только вдоль одной из координатных осей.

Так как принцип, используемый реальными осьминогами для ориентирования в пространстве, был описан в статье [2] как «расположение глаз на одном уровне», на начальном этапе работы аналогичное условие было задано следующим образом. На модели осьминога были выбраны две вершины, расположенные на противоположных сторонах туловища на одинаковом расстоянии от носа, и условно назначены «глазами» (рисунок 18, красный отрезок). В качестве параметра для обучения была взята разность координаты Y (вертикальной) двух выбранных точек в мировой системе координат. По аналогии с другими параметрами, заданными авторами исходной программы, для расчета вознаграждения была выбрана функция вида $r = w \cdot e^{-\sigma|y_1 - y_2|^2}$ (где w – вес параметра), равная 1 при $|y_1 - y_2| = 0$, то есть при наиболее точном выполнении условия поддержания горизонтальности, и монотонно стремящаяся к 0 при увеличении разности. Коэффициент σ определяет скорость уменьшения вознаграждения при значениях разности, отличных от нуля. Исходно, по аналогии с функцией вознаграждения для вектора скорости, было задано $\sigma = 20$. Полученная функция была добавлена в качестве третьей составляющей вознаграждения при обучении.

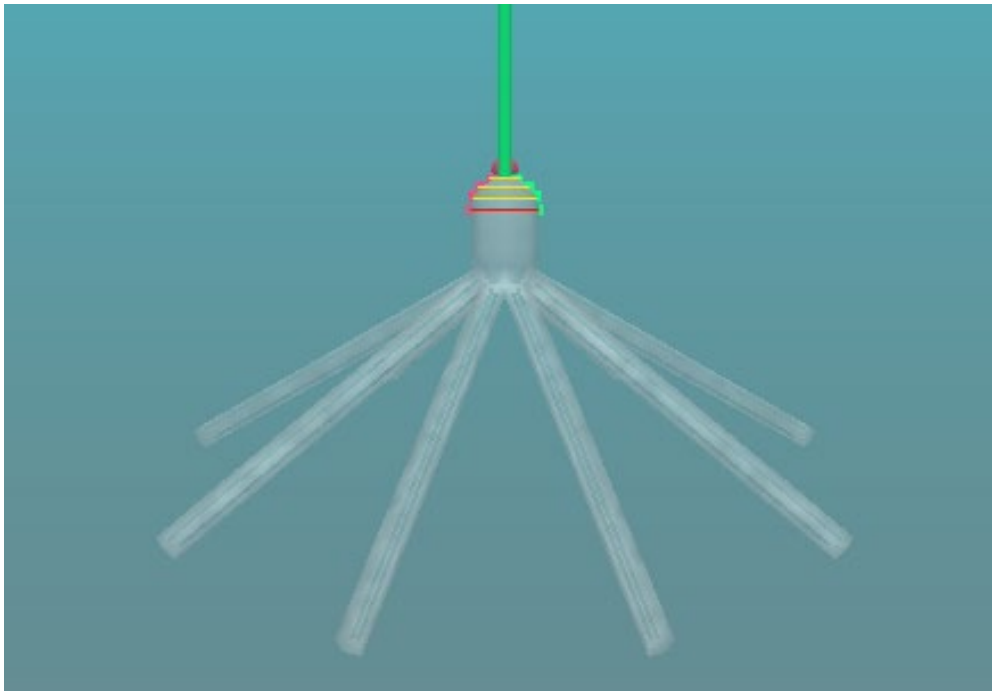

Рисунок 18 – Ключевые точки на модели осьминога. Красные и зеленые метки – расположение самих пар вершин, красным отрезком соединена исходная пара «глаз», желтыми – дополнительные пары, добавленные на следующем этапе.

При попытке обучения модели с новым добавленным слагаемым было обнаружено, что дополнительное ограничение не влияет на результат. Значение вознаграждения с первых итераций обучения оставалось близким к максимально возможному (рисунок 19 (а)), при этом модель осьминога продолжала вращаться вокруг своей оси без изменений. После проверки численных параметров модели осьминога, используемых в исходном коде, было выявлено, что максимальная толщина туловища осьминога составляет всего 0,1 в условных единицах длины. Таким образом стало очевидно, что заданное значение σ, равное 20, настолько мало, что даже при максимально возможной разности высот двух заданных точек (равной 0,1) значение функции вознаграждения превосходит 0,8 из 1 (рисунок 20), и таким образом, вознаграждение растет даже при существенном нарушении условия горизонтальности.

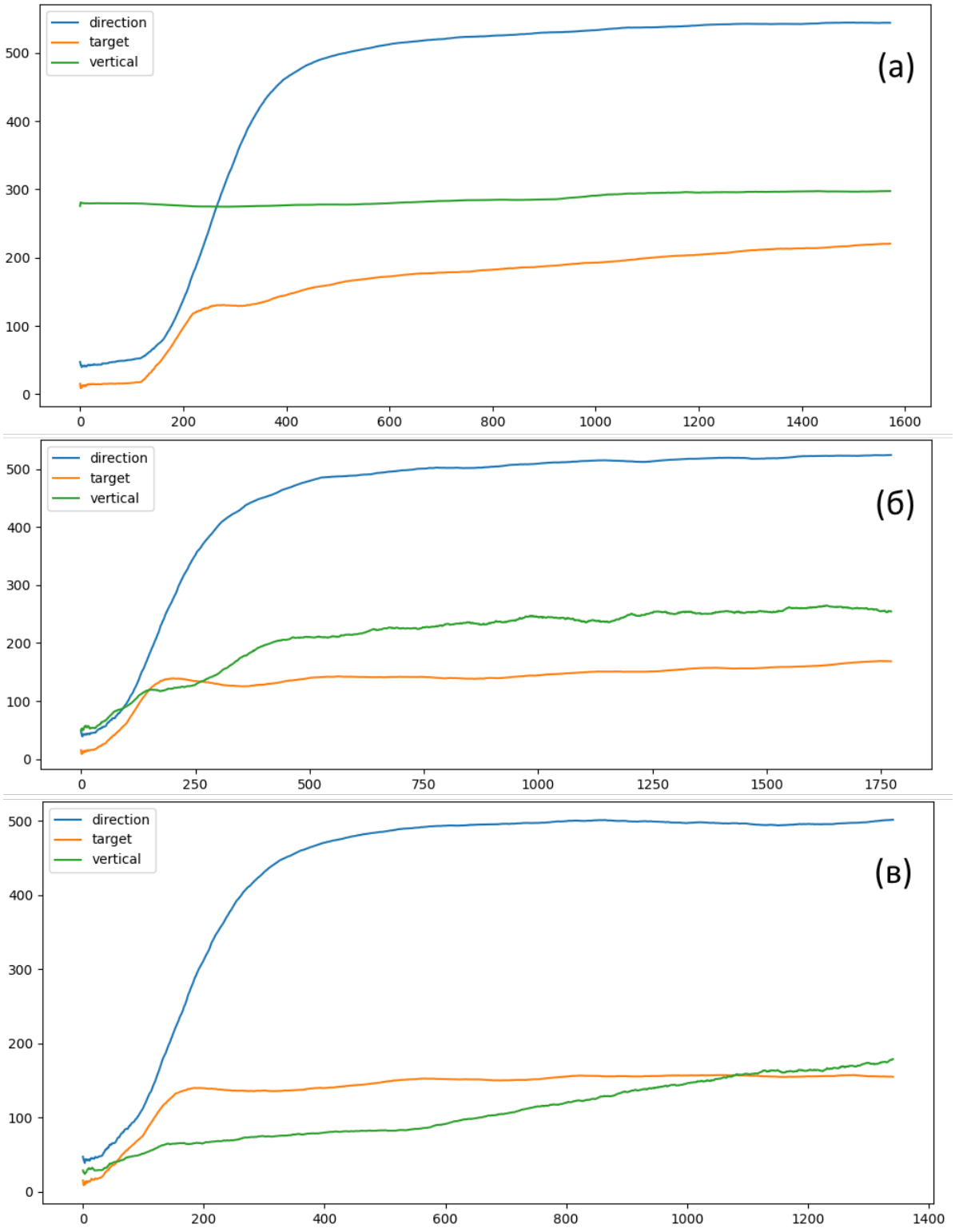

Рисунок 19 – Результаты обучения с тремя параметрами. Зеленый график – параметр разности между глазами. (а) $\sigma = 20$, одна пара точек, (б) $\sigma = 1000$, 4 пары точек, (в) $\sigma = 5000$, 4 пары точек.

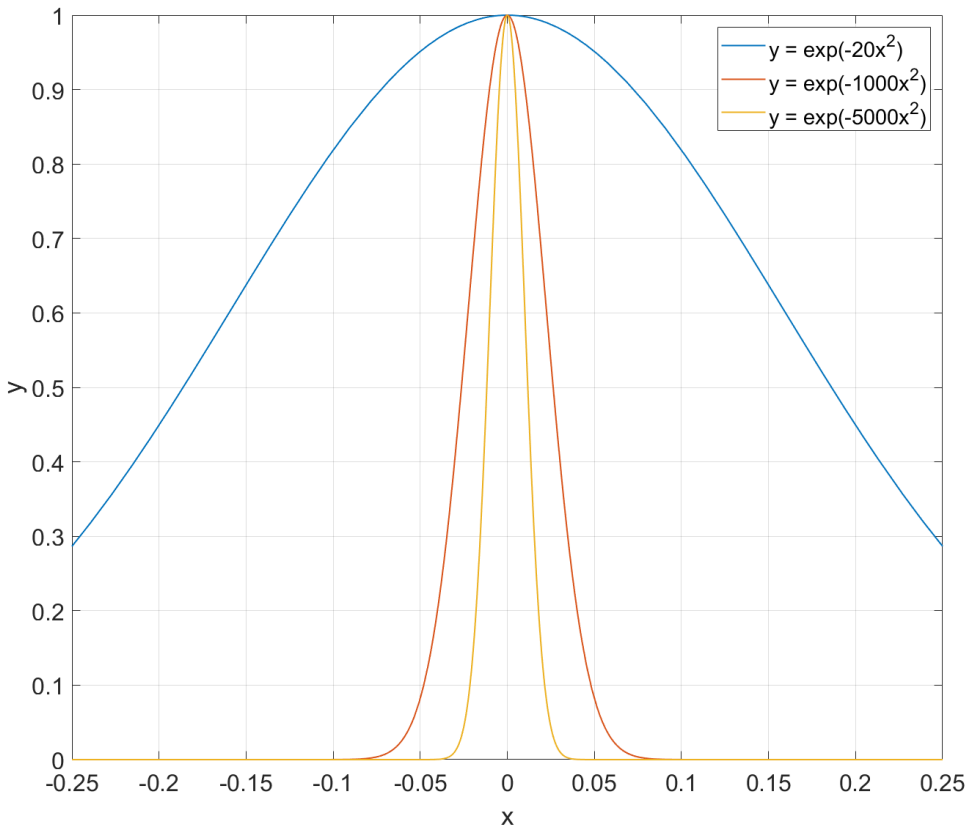

Рисунок 20 – Вид функции $y = e^{-\sigma x^2}$ при $\sigma = 20$, $\sigma = 1000$ и $\sigma = 5000$

Для решения обнаруженной проблемы было сделано два изменения в функции вознаграждения. Во-первых, для увеличения максимальной ошибки при существенных отклонениях осьминога от требуемого положения были выбраны три дополнительных пары точек по такому же принципу, как исходная, и при вычислении вознаграждения суммировались разности Y-координаты для всех четырех пар точек. Во-вторых, экспериментальным путем было подобрано значение $\sigma = 1000$, при котором сумма вознаграждения за новое условие на первых итерациях равна примерно 50, что ненамного превосходит начальные значения других двух составляющих, и затем медленно возрастает до максимального значения (рисунок 19 (б)).

Попытка обучения с измененной функцией вознаграждения привела к более

приемлемому результату: при основной части направлений движения осьминог не делал полных оборотов вокруг своей оси, хотя продолжал в некоторых случаях заметно отклоняться от горизонтального положения. Так как разность между начальной суммой вознаграждения и суммой на последних итерациях для слагаемого, отвечающего за поддержание горизонтальности, все еще оставалась заметно меньше, чем для двух исходных слагаемых, было принято решение протестировать алгоритм с еще более высоким значением σ с расчетом на то, что при дальнейшем его повышении условие горизонтальности будет более строго выполняться. Было выбрано значение $\sigma = 5000$, остальная часть функции вознаграждения была при этом оставлена без изменений.

По результатам обучения был построен график вознаграждения (рисунок 19 (в)), на основе которого можно было предположить, что требуемое условие станет выполняться еще точнее: начальная сумма вознаграждения стала еще меньше, при этом через 1400 итераций сумма была уже максимальной, и на протяжении всего обучения соответствующее слагаемое функции вознаграждения более плавно возрастало и меньше колебалось, чем при $\sigma = 1000$. Для получения качественного результата в этом случае необходимо было обучать модель еще дольше, более 1900 итераций, так как при 1400, несмотря на кажущийся выход функции на насыщение, при запуске обученного алгоритма получается очень нестабильная модель, в которой регулярно возникают ситуации, где при движении без каких-либо изменений направления осьминог, до того сохранявший горизонтальную ось между глазами, в случайный момент начинает вращаться так же непрерывно, как это происходило до внедрения новой составляющей вознаграждения. Можно предположить, что при дальнейшем увеличении значения σ до некоторого предела будет повышаться точность соблюдения горизонтальности, но при этом будет расти время обучения, требуемого для получения приемлемого результата.

В процессе решения задачи выбора оптимального значения σ была обнаружена проблема, связанная с исходной формулировкой нового слагаемого в

функции вознаграждения. Вне зависимости от значения коэффициента и от точности соблюдения соответствующего условия при движении осьминога в горизонтальном или наклонном направлении, осьминог все еще продолжает вращаться при движении вертикально вверх или вниз. Это обосновано тем, что условие сохранения горизонтальной оси между глазами сформулировано через вертикальные координаты рассматриваемых вершин. При движении вдоль вертикальной оси, вне зависимости от поворота в плоскости других координат, для точек на туловище осьминога, симметричных относительно его центральной оси, условие будет выполняться автоматически и, таким образом, станет бесполезным.

На данный момент был выработан один возможный альтернативный подход к формулировке требования «глаз на одном уровне» или исключения поворотов вокруг своей оси, потенциально устраняющий проблему с вертикальным движением. Предполагается основывать функцию вознаграждения не на координатах, а на скоростях вершин, составляющих туловище осьминога. Если снова рассматривать две точки, расположенные на противоположных сторонах туловища, то при вращении осьминога вокруг своей оси вектора скорости двух точек хотя бы по одной из координат окажутся разнонаправленными, то есть если сравнивать проекции двух векторов скорости на каждую из координатных осей, то при вращении какая-то из пар проекций окажется разных знаков.

В качестве начального способа записи данного условия было решено использовать функцию всего с двумя возможными значениями, соответствующими наличию или отсутствию компонент с разными знаками. Так как другие слагаемые функции вознаграждения могут принимать значения в промежутке от 0 до 1, сначала было выбрано значение 0 для «неправильного» состояния системы – наличия хотя бы одной пары компонент с разными знаками – и 1 для «правильного». Однако при попытке запуска обучения с этими значениями был получен результат, схожий с первой версией функции с условием горизонтальности – с первой итерации получается практически максимально возможная сумма

вознаграждений, после чего обучение по данному параметру перестает выполняться.

Так как в обучении с подкреплением могут использоваться как вознаграждения, так и штрафы [32], далее было решено при наличии вращения добавлять к функции отрицательное слагаемое, исходно заданное равным -1. Затем было предпринято несколько попыток обучения модели с разными значениями вознаграждения за отсутствие вращения. При значении вознаграждения, равном 1, даже при наличии штрафа сумма вознаграждения также выходила на насыщение уже на первых итерациях. В противоположном случае, когда значение было равно 0, то есть составляющая от данного слагаемого могла быть только нулевой или отрицательной, в процессе обучения рассматриваемый параметр вообще переставал учитываться и продолжал уменьшаться в процессе обучения, в то время как остальные слагаемые росли. В результате многократного тестирования на параметрах со значениями вознаграждения между 0 и 1 и последовательного увеличения значения штрафа было найдено состояние относительного равновесия при значениях вознаграждения, равного 0,7, и штрафа, равного -3. При этом наборе значений все слагаемые, включая новое, выходили на насыщение в пределах первых нескольких сотен итераций.

При длительном обучении (около 2500 итераций) с данным набором параметров была получена модель, более приемлемо работающая при движении по вертикали, чем предыдущая, однако дающая существенно худшие результаты во всех остальных положениях. При использовании этой модели осьминог периодически спонтанно начинает вращаться вне зависимости от направления движения и зачастую не возвращается в исходное положение. Однако в целом можно предположить, что при более точной настройке параметров и еще более длительном обучении подобная модель позволит решить поставленную задачу.

На данный момент был протестирован еще один варант модели, включающий сразу и слагаемое, отвечающее за расположение глаз на одном уровне, и слагаемое,

связанное со скоростью. Можно было предположить, что при учете обоих условий осьминог будет не только реже переворачиваться, но и стабилизироваться снова, если уже перевернулся. Однако в результате, даже после почти 4000 итераций обучения, полученная модель осьминога оказалась менее стабильной, чем обе предыдущих – существенно чаще спонтанно поворачивалась и зачастую не стабилизировалась вовсе, продолжая вращаться так же, как при исходном виде функции. В целом вероятно, что для любой из моделей при более длительном обучении результат станет более стабильным, как в случае с первым ограничением при $\sigma = 5000$, однако при наличии ограничения, связанного со скоростью, минимальное число необходимых итераций оказывается значительно больше.

На более стабильных из полученных моделей была также протестирована более интуитивная и применимая в видеоиграх версия контроллера, в которой изменение направления реализуется через углы поворота по двум осям, вычисляемые относительно текущего положения. На данный момент даже при наиболее стабильных вариантах данный контроллер использовать затруднительно, так как при внезапных поворотах вокруг своей оси направление поворота при управлении будет существенно отличаться от требуемого.

В целом можно сделать вывод, что модификации, связанные с алгоритмом обучения, не полностью решили обнаруженные проблемы в исходной программе, однако некоторые положительные результаты были получены. На основе данных результатов можно в дальнейшем построить модель, способную стабилизировать положение осьминога по аналогии с его реальным поведением.

4 Обработка столкновений

Несмотря на то, что в статье [25] описана возможность создания среды с препятствиями, метод избежания столкновений и даже обучение модели осьминога прохождению через узкие места, в доступной реализации программы отсутствует какая-либо функциональность для обработки столкновений. Поэтому возникла необходимость реализовать обнаружение и обработку столкновений самостоятельно.

Предложенный в статье алгоритм состоит в обнаружении всех треугольников в модели осьминога, пересекающихся с препятствиями, на каждом шаге симуляции, при этом, судя по описанию, наличие пересечения определяется с помощью попарного сравнения всех треугольников в модели осьминога и в препятствиях. Затем части осьминога, оказавшиеся внутри пересечения, проецируются в область, свободную от препятствий. Сами авторы указывают, что обнаружение и обработка столкновений является одним из наиболее слабых мест алгоритма по производительности. Поэтому более рациональным решением, чем написание аналогичного алгоритма по имеющемуся описанию, был бы поиск более эффективного способа работы со столкновениями.

В соответствии с теоретическими представлениями о возможных методах обработки столкновений были выделены следующие необходимые этапы решения, каждый из которых должен быть реализован в соответствии с особенностями данной задачи:

1) Приближенное вычисление столкновений

2) Точное вычисление столкновений

3) Обработка столкновений (обход или взаимодействие с препятствием)

В зависимости от выбора конкретного метода расчета столкновений первая и вторая задача могут совместно решаться одним методом, уже включающим в себя различные степени точности вычисления (к примеру, с помощью иерархии ограничивающих объемов [33], которая будет подробнее описана далее).

В работе [33] проведен обзор различных подходов к отслеживанию столкновений в видеоиграх. Так как для видеоигр производительность является ключевым фактором, используемые в них методы направлены на максимальное ускорение вычисления коллизий. Описанные в обзоре варианты разделены на две категории. Первая включает различные способы разделения пространства сцены на части для избежания проверки на наличие коллизии между объектами, находящимися в отдаленных друг от друга точках пространства. Вторая включает способы построения вокруг каждого объекта сложной формы иерархии объемов более простой формы, опять же для последовательного отбрасывания объектов, заведомо не пересекающихся с более внешними объемами в иерархии. При этом в обзоре приведены такие примеры моделей персонажей и жанры игр, для которых при достаточном количестве разбиений или глубине дерева можно считать коллизию с наиболее близким к персонажу объемом эквивалентной коллизии с персонажем.

Однако для осьминога подобный подход представляется не в полной мере приемлемым. Предварительное отбрасывание невозможных коллизий с помощью ограничивающего объема, скорее всего, может существенно повысить производительность программы при наличии обширной сцены с большим числом объектов, но уже при найденном пересечении объекта с ограничивающим объемом следующим этапом будет поиск пересечения непосредственно с моделью осьминога. Такое предположение связано с деформируемостью и практически неограниченной подвижностью осьминога, довольно точно имитируемой его моделью, используемой в программе. Так как каждое щупальце осьминога может изгибаться в произвольных направлениях, растягиваться, сжиматься и скручиваться, построение ограничивающего объема вокруг него приведет либо к объему, близкому к размеру осьминога в целом и потому бесполезному для использования, либо к постоянному трудоемкому пересчету объема, который может оказаться хуже для производительности, чем вычисление коллизий

непосредственно со щупальцем. При этом, как уже было предположено, данный подход возможно использовать для предварительной, грубой проверки на наличие коллизии, позволяющей быстро исключать из рассмотрения большое количество объектов, заведомо находящихся на большом расстоянии от осьминога. Кроме того, существуют алгоритмы, такие как [34], в которых минимальным элементом иерархии является не объем заданной формы, а точка, ребро или грань исходной модели. При использовании подобных алгоритмов коллизия в любом случае будет находиться путем последовательных приближений, после любого из которых вычисления могут быть прекращены в случае, если коллизия на этом уровне приближения не обнаружится. Таким образом, точность и скорость вычисления в каждом случае будет зависеть от расстояния между осьминогом и объектом с подозрением на коллизию, и за счет этого количество лишних вычислений будет минимизировано.

Если рассматривать отдельно вопрос приблизительного вычисления коллизий, то наиболее известным способом является построение иерархии объемов. В работе [33] приведено сравнение разных видов ограничивающих объемов, их достоинства и недостатки. На рисунке 21 представлен результат сравнения трех наиболее простых и распространенных видов: объема, ограниченного сферой, AABB (параллельного осям ограничивающего параллелепипеда) и OBB (ориентированного ограничивающего параллелепипеда). При использовании сферических объемов быстрее всего вычисляются коллизии и задействуется меньше памяти, так как для описания сферы достаточно ее радиуса и координат центра. При этом сфера в среднем оставляет больше пустого места вокруг объекта, из-за чего будет возникать больше обнаруженных коллизий, которые при более точном вычислении окажутся несуществующими. Ориентированный параллелепипед, напротив, замедляет вычисления, из-за необходимости пересчитывать его углы поворота даже в том случае, если размеры объекта не меняются, но при этом более точно настраивается под все измерения объекта.

Промежуточное положение по производительности и точности занимает ограничивающий параллелепипед, параллельный осям. Именно по причине усредненных показателей на данный момент было принято решение использовать в программе этот вид ограничивающих объемов. Скорость вычисления в данном проекте чрезвычайно важна, поэтому, с одной стороны, использовать ориентированный параллелепипед, требующий постоянных разворотов, было бы нецелесообразно, а с другой, выбор сферы также привел бы к излишним вычислительным затратам – осьминог не обладает центральной симметрией и в процессе движения часто существенно растягивается или сжимается по одной или двум осям, поэтому со сферическим объемом будет возникать большое количество ложных коллизий, для которых вынужденно будет применяться трудоемкий алгоритм более точного вычисления.

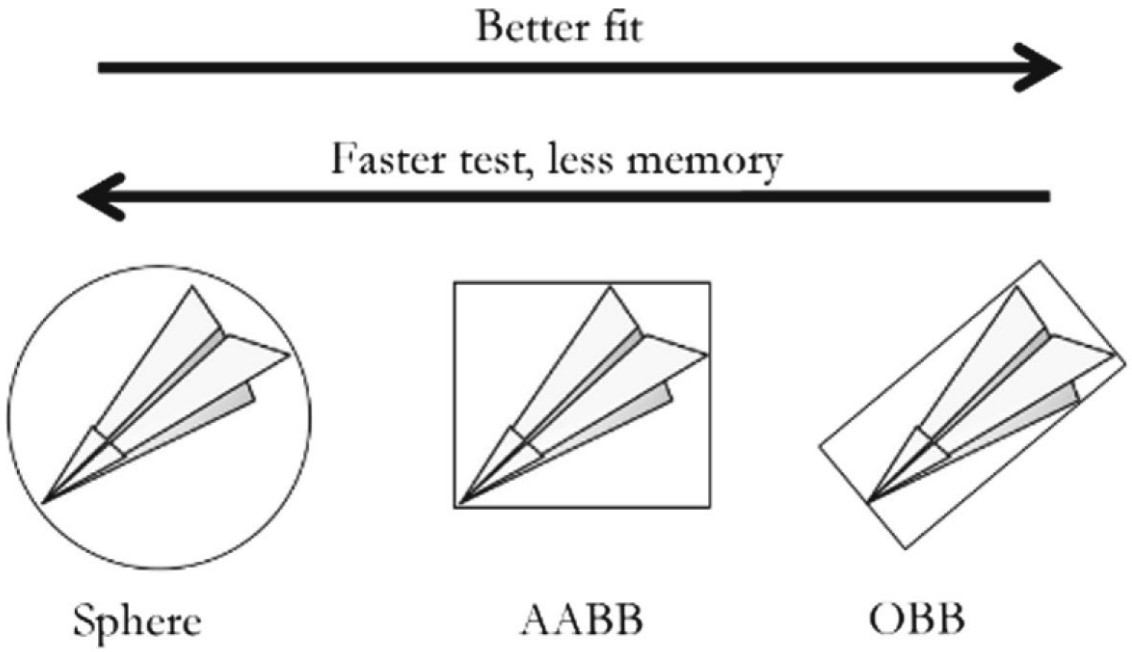

Рисунок 21 – Сравнение производительности и точности наиболее простых видов ограничивающих объемов [33]

Для решения второй задачи – точного вычисления коллизии – помимо обзора [33], сосредоточенного в основном на небольшом выборе похожих между собой методов, чаще всего используемых в видеоиграх, был также рассмотрен обзор [35],

посвященный методам поиска и обработки пересечений и самопересечений в деформируемых объектах. Так как осьминог, как уже было указано выше, определенно является деформируемым объектом, рассмотрение подходов, учитывающих данную специфику, представляется наиболее разумным решением. Кроме того, в работе [35] представлены несколько различных классов алгоритмов, многие из которых не столь широко известны и практически не используются в видеоиграх. Первые главы обзора посвящены уже упомянутым в работе [33] иерархиям ограничивающих объемов и разбиению пространства, однако включают и некоторые менее известные разновидности данных методов, в том числе адаптированные для определенных узкоспециализированных задач. Также расматривается метод определения коллизий через двумерные проекции трехмерной сцены, не позволяющий, однако, вычислять глубину коллизии и другую информацию о ней. Кроме того, описываются и несколько других алгоритмов, в том числе комбинирующих оба этих подхода.

Наиболее интересной на данный момент представляется группа методов, основанных на вычислении пересечений между индивидуальными треугольными гранями моделей. Как уже было указано, щупальца осьминога могут деформироваться в любой точке, и следовательно, именно подобный метод мог бы позволить максимально точно вычислить и обработать его коллизию с другим объектом. В обзоре рассматривается достаточно много вариаций такого подхода, начиная с давней и наиболее прямолинейной реализации, описанной в статье [36], где представлен только алгоритм поиска пересечения двух треугольников, однако не предлагается способа отсеивания невозможных коллизий, кроме ссылок на упомянутые уже иерархии объемов. Предположительно, данная реализация могла исходно использоваться в базовой программе, на которой основывается наше решение. Судя по информации в статье [25], этот метод или подобный ему мог использоваться для вычисления коллизий, при этом, так как иерархии объемов или аналогичные алгоритмы не были упомянуты, возможно, что коллизии находились

перебором всех пар треугольников во всех препятствиях, результатом чего и стала низкая производительность.

В то же время существуют более современные методы той же группы, повышающие производительность вычислений: начиная от работы [37], где предлагается алгоритм всего с 6 проверками на каждую пару треугольников вместо 15, и заканчивая методами, использующими видеокарту для ускорения или исключающими из рассмотрения большую часть треугольников, которые не могут участвовать в коллизии. Последняя категория представляется наиболее интересной, так как близка к предложенному уже подходу, при котором точное вычисление коллизии производится только после отсеивания неактуальных препятствий и частей модели. Так, например, в работе [34], уже упомянутой выше, используется иерархия ограничивающих объемов для локализации возможных коллизий, в то же время для исключения повторных проверок одних и тех же треугольников реализована концепция «репрезентативных треугольников», в которой треугольникам также присваивается информация о некоторых из их вершин или ребер, чтобы возможная коллизия ребра или вершины с другими элементами не проверялась многократно для всех соседних треугольников. В статье утверждается, что использование данного алгоритма приводит к повышению производительности вычислений в 5 раз. В работе [38] предложен другой подход, основанный на обнаружении смещений между соседними кадрами, с использованием вычислений на видеокарте для обработки столкновений. Здесь также заявлено повышение производительности в 7-10 раз. Стоит обратить внимание на то, что данные методы, как и еще некоторые из той же категории, предлагаются авторами для симуляции ткани, которая, как и осьминог, является очень подвижным и легко деформируемым объектом. При этом в рассмотренных статьях и в целом в большинстве случаев для симуляции ткани используются модели с десятками и сотнями тысяч вершин, в то время как в используемой здесь модели осьминога вершин всего несколько сотен. С одной стороны, это вероятнее всего значит, что для более простой модели

производительность данных алгоритмов будет существенно выше. С другой стороны, может оказаться, что и преимущество этих методов по сравнению с более простыми будет существенным только при большом числе вершин. Однако в целом данный класс алгоритмов представляется многообещающим. Основная проблема, обнаруженная на данный момент, состоит в том, что реализации данных методов не доступны для использования, поскольку отсутствуют в открытом доступе. Таким образом, для того, чтобы протестировать их применимость для анимации осьминога, необходимо реализовать их самостоятельно по описаниям, приведенным в статьях, что является достаточно трудоемкой задачей.

Решение третьей задачи, заключенной в построении алгоритма для обработки коллизий, существенно зависит от выбранного подхода к решению двух предыдущих задач. Однако, если говорить о физических принципах обработки коллизии, их можно рассмотреть независимо от конкретной технической реализации. По сути, выбор метода состоит в том, каким образом, если обнаружено пересечение между двумя объектами, следует их разделить так, чтобы пересечения больше не было. В материалах [39] представлены несколько различных возможных подходов к разрешению коллизии после ее обнаружения.

Первый, наиболее простой подход – метод проекции, состоящий в нахождении вектора нормали коллизии и смещении столкнувшихся объектов в противоположных направлениях вдоль этого вектора. Судя по информации в статье [25], исходно в базовой программе использовался именно такой способ обработки коллизий, с поправкой на то, что смещался, скорее всего, только осьминог, так как остальные объекты в сцене предполагались статичными, а также в основном более твердыми и тяжелыми, чем осьминог. В зависимости от желаемого результата данный подход можно продолжать использовать: если реализуемое поведение визуально и интуитивно соответствует представлению пользователя о возможном поведении осьминога, вполне допустимым представляется использование упрощенной модели, не предполагающей точного физического моделирования.

Однако необходимо также учитывать метод обнаружения коллизий и тот факт, что осьминог является деформируемым, но упругим. Таким образом, при коллизии будет смещаться только часть точек, составляющих осьминога, и необходимо учесть влияние этого смещения на существующую уже модель движения таким образом, чтобы поведение осьминога оставалось внешне реалистичным.

Следующий предложенный метод – метод импульсов, состоящий в задании пересекающимся объектам мгновенного изменения скорости. Такой подход намного больше соответствует реальному взаимодействию предметов при столкновении. В реальном мире, когда два предмета сталкиваются, за счет закона сохранения импульса составляющие их скоростей, направленные вдоль нормали коллизии, изменятся на противоположные (скорее всего, с некоторым уменьшением, коэффициент которого зависит от упругости предметов). Таким образом, описанная модель в целом соответствует реальной ситуации. При этом с точки зрения процедурной анимации и удобства использования она также имеет свои преимущества. Так, существенная проблема метода проекции состоит в том, что даже при успешном разрешении коллизии на текущем шаге симуляции коллизия с большой вероятностью возникнет и на следующем, и в зависимости от положений и скоростей объектов может вообще отсутствовать возможность выйти из коллизии [39]. При этом в случае задания объектам, или одному из них, скорости они продолжат двигаться и после выхода из коллизии, и таким образом коллизия не повторится на следующем шаге. В случае управляемого пользователем осьминога и твердого статичного препятствия, осьминог оттолкнется от препятствия, давая пользователю время изменить направление его движения.

Аналогичные аргументы можно привести и в пользу третьего метода, основанного на задании объектам ускорения или, иначе говоря, приложения сил, отталкивающих объекты друг от друга. Данное описание является несколько менее корректным с точки зрения физики, чем метод импульсов, так как при столкновении происходит скорее мгновенное изменение импульса, чем длительное воздействие

какой-либо силы. Однако для процедурной анимации данный подход может иметь определенные преимущества, в том числе для текущей задачи. Так, если рассматривать существующую модель осьминога, в ней уже реализованы уравнения для вычисления суммы сил, воздействующих на осьминога в каждой точке, поэтому решение задачи добавления новой составляющей представляется более простым, чем построение полностью нового алгоритма в первом случае или учет в уравнениях скорости, возникающей не через пользовательское управление и не за счет действия сил, во втором.

Для практической реализации системы обработки коллизий в первую очередь в программу была добавлена возможность загрузки препятствий, также заявленная, но фактически не реализованная в исходной версии. Загрузка препятствий осуществляется с использованием текстового файла, в котором указываются пути до файлов с моделями для препятствий и параметры геометрических преобразований, применяемых к моделям для размещения их в нужном месте сцены.

Исходно было запланировано начать решение с задачи точного вычисления коллизий, с расчетом на выбор алгоритма, автоматически включающего в себя и отсеивание лишних элементов сцены до начала поиска пересечения. Однако, как было уже упомянуто, представляющие наибольший интерес алгоритмы, такие как [34] и [38], не предлагают свободного доступа к реализации. Так как самостоятельная реализация подобного алгоритма потребовала бы значительного объема времени, было решено в первую очередь решать две других задачи. Приблизительного вычисления коллизий достаточно для тестирования корректности загрузки и расположения препятствий, а также управляемого целенаправленного передвижения осьминога по сцене. Реализация обработки коллизий при невозможности их точного вычисления не позволит получать реалистичные локальные деформации при столкновении, однако упростит тестирование вышеупомянутых результатов обнаружения коллизий, а также

позволит сравнить различные варианты реакции на коллизию по крайней мере на уровне осьминога в целом и сравнить их влияние на управление.

Для решения первой задачи был реализован алгоритм создания ограничивающих объемов как для осьминога, так и для препятствий. В случае препятствия, после загрузки модели и применении к ней преобразований среди ее вершин находится максимальное и минимальное значение каждой из трех координат. Полученные 6 значений сохраняются, и на их основе получаются и, при необходимости в целях отладки, изображаются на сцене, границы параллелепипеда, ограничивающего препятствие. Аналогичные действия производятся и для осьминога, но в его случае границы параллелепипеда пересчитываются каждый кадр, так как осьминог постоянно перемещается и существенно деформируется при движении.

Реализация была протестирована на предмет корректного расположения и отображения препятствий, корректного вычисления координат ограничивающих параллелепипедов и отслеживания пересечений между ними (рисунок 22). Так как задача обработки найденных коллизий пока не была решена, на первом этапе проверялся только факт начала и окончания пересечения между ограничивающими параллелепипедами осьминога и препятствий. В процессе тестирования не было обнаружено явных проблем с коллизиями, и таким образом на основе данной реализации можно было решать следующие две поставленных задачи.

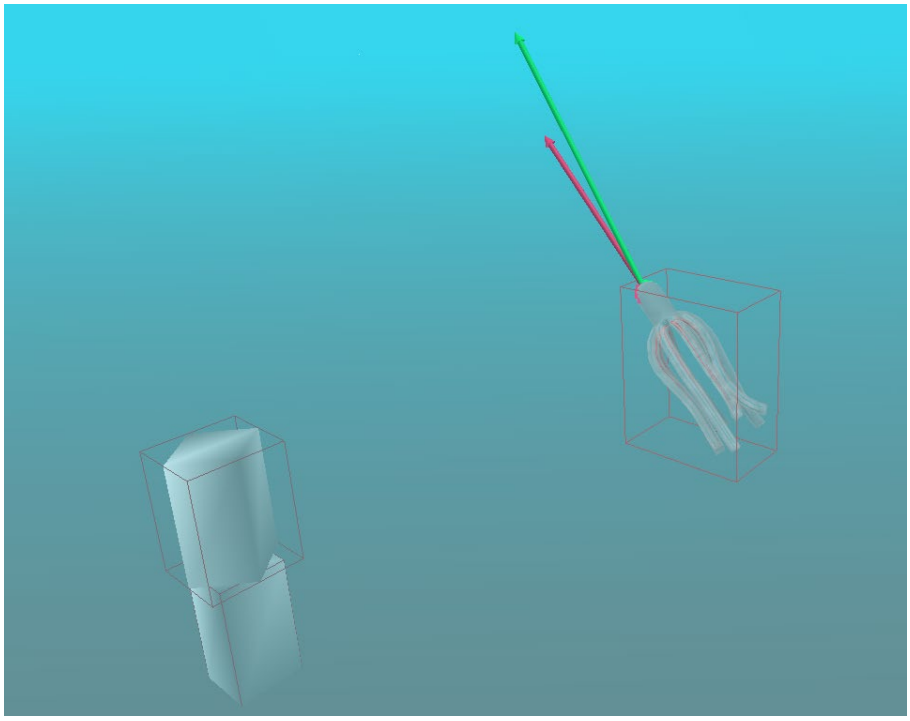

Рисунок 22 – Сцена с препятствиями. Тонкими красными линиями обозначены ограничивающие параллелепипеды.

Далее для обработки коллизий было решено использовать комбинированный подход: методом проекции обеспечивался выход из коллизии, а для моделирования дальнейшего поведения осьминога после столкновения предполагалось использовать метод импульсов или ускорений. Обосновано данное решение тем, что при симуляции вычисление положения и скорости осьминога, а также действующих на него сил, происходит в дискретные моменты времени. Поэтому очевидно, что и обнаружение коллизий тоже происходит в эти же дискретные моменты. Таким образом, если коллизия произошла между двумя шагами симуляции, к моменту ее обнаружения осьминог уже будет пересекаться с предметом, находясь частично внутри него. Такое поведение, очевидно, не соответствует реальному, и, следовательно, компенсировать его также можно с помощью процессов, не происходящих при реальной коллизии, если при этом конечный результат будет достаточно близок к реальному.

При этом также предполагается, что все препятствия в сцене не только

статичны, но и обладают намного большей массой, чем осьминог, и таким образом, при столкновении не смещаются и не деформируются видимым образом. Следовательно, любое смещение и изменение скорости будет применяться только к осьминогу. Таким образом, для выхода из коллизии было выбрано простое решение, основанное на смещении осьминога вдоль вектора нормали к поверхности препятствия в обратном направлении, до момента касания с препятствием. Так как на данный момент все коллизии вычисляются для параллелепипедов, ориентированных вдоль осей, нормали к граням препятствия также будут совпадать с координатными осями. Поэтому достаточно найти, по какой из координатных осей глубина погружения в препятствие будет наименьшей, и тогда именно эта ось станет нормалью коллизии [39]. Реализация данного этапа была относительно успешной, однако привела к тому, что при коллизии в том случае, если осьминог сталкивается с препятствием под углом, близким к прямому, он упирается в препятствие практически без возможности развернуться и выйти из коллизии. При более острых углах осьминог начинает скользить вдоль препятствия, однако по-прежнему не может развернуться, и коллизия заканчивается только тогда, когда он доходит до края препятствия.

Следующим этапом работы над алгоритмом была попытка реализации отталкивания от препятствия одним из рассмотренных методов – через импульс или ускорение. Вопреки ожиданиям, более очевидный способ реализации был найден для метода импульсов, а так как данный метод уже был предпочтительным с точки зрения физической точности, в дальнейшем по большей части предпринимались попытки реализовать именно его. Так как при упрощенной модели коллизии определить точку приложения импульса было бы невозможно, было предложено также упрощенное решение. Для всех точек с ненулевой составляющей скорости, направленной в сторону препятствия, значение этой составляющей заменяется на противоположное с умножением на коэффициент восстановления, добавленный для моделирования не полностью упругого удара (рассматривались значения

коэффициента от 0.25 до 0.5). Данное решение показывало результаты, близкие к ожидаемым, для поведения осьминога непосредственно после коллизии: осьминог отталкивается от препятствия достаточно сильно, но с заметно меньшей скоростью, чем до столкновения. Однако в процессе тестирования была обнаружена существенная неустойчивость модели: после многократных столкновений скорость осьминога резко возрастала, затем он начинал хаотично двигаться и деформироваться. Причина данного явления не была полностью выяснена, но основная гипотеза состоит в том, что из-за резких изменений скорости в момент коллизии начинают неконтролируемо возрастать воздействующие на осьминога внешние силы. Так как значения сил сопротивления и тяги зависят от скорости движения осьминога относительно воды, при большом модуле и различных направлениях скорости в разных точках осьминога эти силы будут тянуть разные части осьминога в разные стороны, и на каждом следующем шаге разница в скоростях будет только возрастать.

На данный момент нет уверенности, каким образом следует решать данную проблему, однако возможно, что единственным рациональным решением будет возвращение к задаче точного вычисления коллизий и построение корректной физической модели уже на ее основе. Причина, в первую очередь, в том, что в предлагаемой модели существует противоречие между рассмотрением осьминога как единого, почти однородного объекта при обработке столкновений и рассмотрением его как сложной системы подвижных частей в исходной системе разностных уравнений и при моделировании сокращений мышц. К тому же задачу точного вычисления коллизий в любом случае следует решить, так как в противном случае большая часть специфики и достоверности модели будет утеряна. Следовательно, даже если существующий алгоритм обработки столкновений возможно настроить таким образом, чтобы избежать неустойчивости, более логичным было бы реализовать новый алгоритм на основе более точного метода обнаружения коллизий.

5 Результаты и дальнейшие перспективы

На начальных этапах работы базовый проект запускался на ноутбуке с процессором Intel Core i7-10875H CPU (8 ядер, 16 потоков), с операционной системой Ubuntu 22.04.2 LTS. В случае решения с использованием OpenMP задействовались все доступные 16 потоков. Стоит заметить, что на этом этапе был протестирован только демонстрационный пример, непригодный для самостоятельного использования. За счет добавления многопоточности удалось повысить частоту кадров в 1.5 - 2 раза, однако протестировать модификации и сравнить результаты для управляемой модлеи с использованием обучения на тот момент не было возможности.

В дальнейшем из-за технических проблем, а также по причине значительно лучших характеристик доступной аппаратуры, для последующей работы был использован компьютер с процессором Intel Core i5 14600K с 14 ядрами и 20 потоками, с операционнной системой Ubuntu 20.04. При запуске на данном устройстве как демонстрационный пример, так и полноценный алгоритм с использованием нейронной сети, в том числе с последующими модификациями имеют примерно одинаковую, заметно более низкую производительность, в основном составляющую примерно 65-70 миллисекунд на кадр.

Очевидно, что на данный момент производительность остается недопустимо низкой для практического использования, однако можно выделить несколько возможных путей ее повышения. Наиболее перспективным представляется подключение вычислений на видеокарте, которое позволит вместо существующих 20 потоков для вычисления однотипных трудоемких преобразований матриц использовать несколько тысяч. С другой стороны, возможен поиск более эффективных реализаций используемых в вычислениях операций с матрицами. При этом вычисления на видеокарте имеют и другое, независимое применение – ускорение процесса обучения нейронной сети. В процессе работы параметры обучения многократно модифицировались, и в дальнейшем, вероятно, потребуется

модифицировать их снова. Поэтому, несмотря на то, что обучение не происходит в реальном времени при запуске анимации, ускорение данного процесса существенно ускорило бы работу над алгоритмом в целом.

Кроме того, часть проблем с производительностью может быть связана с вынужденным использованием устаревших библиотек и операционной системы. На данный момент базовая программа несовместима с современными версиями языка Python и библиотек, используемых для обучения, следовательно, в дальнейшем ее следует адаптировать под их современные версии.

Также существенное влияние на производительность оказывает выбор метода обнаружения коллизий. Временно используемый сейчас приближенный метод не влияет видимым образом на производительность, однако более точный метод определенно будет требовать намного больше вычислений. Необходимо внимательно подойти к вопросу его выбора, так как большинство методов для точного вычисления коллизий чрезвычайно дороги в плане производительности [35]. Очевидно, что повысить эффективность базовой программы данный метод неспособен, однако чрезвычайно важно, чтобы производительность не ухудшилась существенно из-за его использования.

После оптимизации и модернизации программы закономерной перспективой представляется внедрение анимации в какой-либо игровой движок для возможности использовать ее в видеоиграх. На текущий момент анимация существует лишь в виде самостоятельного приложения, что достаточно для ее тестирования и демонстрации, но недостаточно для практического применения. Так как при постановке данной задачи была заявлена цель создать анимацию, пригодную для использования именно в видеоиграх, адаптация ее для игровых движков становится практически обязательным планом дальнейшего развития.

Несмотря на это, наиболее значимыми все же остаются задачи, связанные с качественным улучшением алгоритма. На начальных этапах работы с базовой программой, помимо проблем с производительностью, были обнаружены

недостатки в некоторых элементах анимации или даже полное их отсутствие. Кроме того, были предложены возможные улучшения, не реализованные в базовой программе и не упоминаемые в статье [25], но позволяющие сделать анимацию более реалистичной.

Еще во время запуска демонстрационного примера было обнаружено, что осьминог при движении неконтролируемо вращается, что противоречит принципам реального движения осьминогов, описанным, к примеру, в статье [2], а также препятствует построению удобной системы управления. Так как в самой статье [25] была описана модель, в которой с помощью дополнительного параметра при обучении нейронной сети обеспечивается горизонтальное положение осьминога в пространстве, было решено подойти к проблеме аналогичным образом. Было проведено обучение с двумя вариантами дополнительного параметра, с различными коэффициентами вознаграждения и штрафа, в результате чего был получен приемлемый результат – модель, при использовании которой осьминог практически перестает вращаться вокруг своей оси. Результат все еще не идеален, так как даже с самым стабильным вариантом модели осьминог часто заметно отклоняется от горизонтального положения и периодически все же начинает вращаться, что существенно усложняет управление. Однако так как результаты симуляции заметно улучшились, можно сделать вывод, что при более точном подборе параметров и, возможно, более длительном обучении задача будет полностью решена.

Еще одной ключевой задачей является обработка столкновений с препятствиями. Данная задача была рассмотрена в статье [25], однако в доступной реализации программы ее решение отсутствовало. Более того, судя по информации в статье, это решение было чрезвычайно неэффективным. При этом задача обхода препятствий для модели осьминога весьма нетривиальна, так как в отличие от животных, обладающих внутренним или внешним скелетом и, как следствие, конечным числом суставов и подвижных частей, осьминог представляет собой

скорее деформируемое тело сложной формы, по ожидаемой реакции на коллизию напоминающее скорее ткань, чем модель со скелетом.

Прежде чем приступить к решению как таковой задачи вычисления коллизий, необходимо было построить сцену, на которой можно было бы тестировать полученный алгоритм. С использованием уже существующих в базовой программе средств работы с трехмерными моделями была реализована функция создания препятствий из моделей на основе текстового файла, в котором, кроме файла модели, указываются преобразования, применяемые к геометрическому объекту, для расположения его в требуемом месте на сцене.

Затем решаемая задача была поделена на три составляющих: приблизительное вычисление коллизии, точное вычисление и реакция. Наибольший интерес представляют вторая и третья задачи – в особенности вторая, так как эффективных способов точно вычислять коллизии с деформируемыми объектами чрезвычайно мало. Было рассмотрено несколько подходящих методов, в основном из класса методов, созданных для моделирования ткани. Однако на данный момент решению данной задачи препятствует отсутствие какого-либо из упомянутых методов в открытом доступе. В дальнейшем возможна попытка их воспроизведения на основе имеющейся информации (в том числе с возможными целенаправленными модификациями в соответствии со спецификой задачи), однако на данный момент такое решение потребовало бы больших временных затрат, поэтому было решено в первую очередь переключить внимание на задачу обработки столкновений и физической реакции на них.

Для решения данной задачи была сначала реализована простая система приближенного вычисления коллизий с использованием ограничивающих параллелепипедов, ориентированных вдоль осей. Затем на ее основе осуществлялся поиск и тестирование возможных алгоритмов обработки коллизий. Был успешно реализован метод проекции для вывода осьминога из коллизии, однако для более физически корректного и интуитивно понятного поведения модели требовалось

также передавать осьминогу при столкновении импульс или, иначе говоря, изменять его скорость, направляя его от препятствия. Было предпринято несколько попыток реализовать это свойство, однако все реализации оказывались неустойчивыми. При любой реализации, где на каждом столкновении осьминогу сообщался импульс (или, в альтернативной, менее физически точной версии – ускорение), после нескольких коллизий силы, действующие на осьминога, начинают неконтролируемо расти. На данный момент задача осталась нерешенной, однако можно предположить, что наиболее логичным подходом будет ее решать уже на основе алгоритма точного вычисления коллизии. Физическая модель, в которой импульс от коллизии распространяется на всего осьминога, в целом чрезвычайно неточная и с трудом совмещается с уже существующей моделью, реализованной в базовой программе. В случае же точного вычисления точек коллизии будет проще корректно рассчитывать скорости и набор сил в этих точках, и общая физическая модель будет более логичной.

Уже после применения описанных модификаций получившаяся программа была адаптирована для запуска на новейшей версии операционной системы Ubuntu (24.04). По причине использования в исходной базовой программе многочисленных библиотек и модулей и их достаточно частого обновления, существующая до того момента реализация на современной системе запускаться не могла. Также можно было предположить, что за счет использования более новых версий библиотек производительность программы может улучшиться. На текущий момент был реализован переход на новую систему с минимальными изменениями, с заменой устаревших модулей на новейшие или совместимые с ними. В результате адаптация программы прошла успешно и привела к повышению производительности примерно до 50 миллисекунд на кадр и упрощению процесса установки.

В процессе исследования поведения осьминога была обнаружена еще одна потенциальная задача, представляющая интерес. В базовой программе для осьминога реализован только механизм плавания, однако в природе осьминогу

доступны и другие способы передвижения, в частности, ползание по дну, являющееся наименее энергозатратным способом и практически единственным, используемым в неволе. Реализаций процедурной анимации для данного способа передвижения не было найдено. В предыдущем отчете уже обсуждалась возможность создания такой анимации и указывались возможные сложности при ее разработке, связанные с кажущейся случайностью выбора щупальца для следующего «шага» при таком способе передвижения, а также с общей нехваткой информации об этом способе. Однако после более подробного исследования существующей программы можно сделать новое предположение о возможном подходе к решению задачи.

Текущая модель движения осьминога реализована с использованием нейронной сети, обучение которой позволяет переводить симуляцию мышечных сокращений в осмысленные, управляемые движения. Скорее всего, аналогичный принцип можно использовать и для моделирования перемещения по дну.

Исследования осьминогов показывают, что движения отдельных щупалец при передвижении осьминога по дну выглядят случайными, и обнаружить отчетливой закономерности их выбора и перемещения не удалось. Однако было замечено, что осьминог выбирает такое движение, которое перемещает его как можно дальше в требуемом направлении. Поэтому можно предположить, что подобный принцип движения можно также реализовать с помощью нейронной сети, так как ее способ принятия решений с большой вероятностью окажется аналогичным реально используемому осьминогом, по крайней мере, по конечным результатам.

Для реализации перемещения по дну необходимо будет установить другой набор физических сил и ограничений, актуальных для данной модели передвижения. Предположительно, нужно будет учесть силу трения о дно, за счет которой и будет выполняться движение. Вероятно, силы сопротивления и тяги также будут вычисляться иначе, так как будет учитываться гравитация.

Не менее важно установить корректные ограничения на модель при обучении. Основным ограничением, скорее всего, будет сохранение ориентировки тела при отсутствии необходимости его менять [40]. Другие возможные ограничения еще нужно исследовать, и некоторые из них, возможно, следовало бы задать не как ограничения, а как предпочтения: так, в работе [40] указано, что в то время как выбор щупалец не имеет однозначной закономерности, он не является полностью случайным (рисунок 23). В ней также указано, что у осьминога есть и предпочтения по выбору направления ползания относительно положения тела, отчасти связанные с его зрением: при движении ровно вперед или назад цель оказывается на границе поля зрения двух глаз, поэтому эти направления осьминог обычно не выбирает.

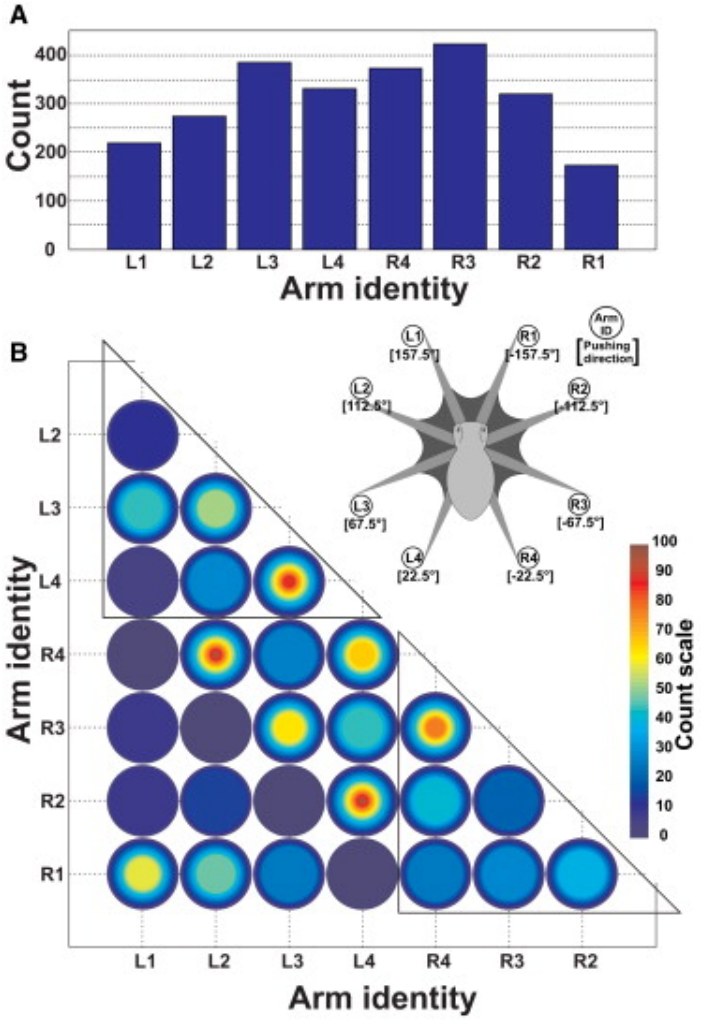

Рисунок 23 – Распределение частоты одновременного использования осьминогом разных пар щупалец.

На данный момент еще требуется собрать все условия, ограничения и предпочтения осьминога при его движении по дну, которые затем нужно будет объединить в общую модель движения для дальнейшего обучения. Также в дальнейшем, после реализации обоих способов движения, следует найти способ их совместного использования и автоматического переключения между использованием одного и другого способа в зависимости от положения осьминога в пространстве. Уже описанная проблема обнаружения коллизий имеет отношение и к данной задаче: передвижение по дну не только невозможно само по себе без точного и быстрого вычисления коллизий, но и для переключения между разными режимами передвижения будет требоваться возможность различать некоторые определенные ситуации, связанные с коллизиями, например, отличать ситуацию «осьминог лег на дно» от касания дна несколькими щупальцами при плавании. Задача удержания осьминогом горизонтального положения для этой задачи также оказывается не менее важной, чем для плавания, а нарушение данного условия наверняка стало бы еще более заметным для наблюдателя или игрока.

ЗАКЛЮЧЕНИЕ

В результате проведенного обзора источников, связанных с анатомией осьминога, были выделены уникальные особенности его строения и поведения, в первую очередь – отсутствие костей в щупальцах, приводящее к практически бесконечному числу степеней свободы, а также большое разнообразие способов движения. Были исследованы классические подходы к процедурной анимации различных животных, и по итогам исследования сделан вывод о невозможности их непосредственного применения для моделирования осьминога из-за специфики его строения.

Проведен поиск и анализ существующих разработок по созданию анимации осьминога. По большей части обнаруженные решения были либо чересчур сложны для использования в реальном времени, либо моделировали только отдельные щупальца, а не осьминога в целом. Было найдено одно базовое решение [25], ставшее основой для дальнейшей разработки, несмотря на то, что решение в большей степени относилось не к процедурной анимации, а к физическому моделированию.

Помимо таких очевидных требований, как внешнее сходство анимации с реальными движениями осьминога и производительность, достаточная для генерации анимации в реальном времени, на основе материалов по анатомии возникло такое дополнительное условие, как удержание горизонтальной оси между глазами при движении, являющееся в природе важным принципом координации осьминога.

В базовой реализации с самого начала присутствовала модель распространения нервных импульсов и сокращения мышц осьминога, а также упрощенная модель гидродинамики. При этом не было реализовано пользовательское управление и взаимодействие с препятствиями, производительность была недопустимо низкой, а также не выполнялось вышеупомянутое дополнительное условие.

Было проведено дополнение и модернизация базового решения: модифицирован алгоритм обучения с учетом наложенных ограничений, добавлено пользовательское управление, загрузка объектов в сцене, программа адаптирована для современной версии операционной системы, решена часть задач, связанных с обнаружением препятствий, а также исследованы возможности реализации другого вида движения на основе того же решения. В дальнейшем предстоит решить проблему производительности, выбрать оптимальный метод обработки коллизий и добавить возможность передвижения по дну.

Актуальная на текущий момент версия реализации алгоритма доступна по ссылке: https://github.com/elenastotskaya/octopus-animation. К ней прилагается более подробное описание структуры программы и инструкция по установке.

В результате тестирования полученного решения можно сделать вывод, что по сравнению с исходной программой были реализованы существенные улучшения. Однако с учетом оставшихся проблем, в особенности связанных с производительностью, на данный момент существующая реализация анимации еще не пригодна для встраивания в игровой движок и использования в видеоиграх.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Octopus, squid, and cuttlefish: a visual, scientific guide to the oceans' most advanced invertebrates / Hanlon, Roger T., Vecchione, Michael, Allcock, A. Louise – The University of Chicago Press, 2018. – 225 p.

2. Levy G., Hochner B. Embodied Organization of Octopus vulgaris Morphology, Vision, and Locomotion // Front. Physiol. – 2017. – Vol. 8. – P. 164.

3. Kazakidi A., Botvinnik A., Kuba M., Sfakiotakis M. Swimming patterns of the Octopus vulgaris // 22nd Annual Meeting of Neural Control of Movement Society. – 2012.

4. Hernández-Urcera J., Garcí M., Cabanellas-Reboredo M. Bipedal locomotion in Octopus vulgaris // Marine Biodiversity – 2020. – Vol. 50 – Article number 87.

5. Amodio P., Josef N., Shashar N., Fiorito G. Bipedal locomotion in Octopus vulgaris: A complementary observation and some preliminary considerations // Ecology and Evolution – 2021. – Vol. 11 (9) – P. 3679-3684.

6. Chang H.S., Halder U., Shih C.H. et al. Energy Shaping Control of a Muscular Octopus Arm Moving in Three Dimensions // Proceedings of the Royal Society A – 2022. – Preprint.

7. Carls-Diamante S. Where Is It Like to Be an Octopus? // Front. Syst. Neurosci. – 2022. – Vol. 16 – Article number 840022.

8. Zelman I., Titon M., Yekutieli Y. et al. Kinematic decomposition and classification of octopus arm movements // Front. Comput. Neurosci. – 2013. – Vol. 7 – P. 60.

9. Inverse Kinematics for Tentacles [Электронный ресурс]. – URL: https://www.alanzucconi.com/2017/04/12/tentacles/. – (Дата обращения: 09.01.2023).

10. Kang R., Branson D., Guglielmino E., Caldwell D. Dynamic modeling and control of an octopus inspired multiple continuum arm robot. // Computers & Mathematics with Applications – 2012. – Vol. 64 (5) – P. 1004-1016.

11. Fras J., Noh Y., Macias M., et al. Bio-inspired octopus robot based on novel soft fluidic actuator // Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA) – 2018. – № 8460629.

12. Ahmed F., Waqas M., Shaikh B. et al. Multi-material Bio-inspired Soft Octopus Robot for Underwater Synchronous Swimming // Journal of Bionic Engineering – 2022. – Vol. 19 – P. 1229–1241.

13. Matko T., Chang J., Xiao Z. Recent Progress of Computational Fluid Dynamics Modeling of Animal and Human Swimming for Computer Animation // Lecture Notes in Computer Science – 2017. – Vol. 10582 – P. 3–17.

14. Nakajima R., Shigeno S., Zullo L. et al. Cephalopods Between Science, Art, and Engineering: A Contemporary Synthesis // Front. Commun. – 2018. – Vol. 3 – Article number 20.

15. The Squid Cinema from Hell: Kinoteuthis Infernalis and the Emergence of Chthulumedia / Fleming D.H., Brown W. – The Edinburgh University Press, 2020. – 328 p.

16. Game: Animals, Video Games, and Humanity / Tyler T. – University of Minnesota Press, 2022. – 152 p.

17. OctoRaid VR (2023) [Электронный ресурс]. – URL: https://store.steampowered.com/app/1795820/OctoRaid_VR/. – (Дата обращения: 09.01.2023).

18. Octopus Bar (2017) [Электронный ресурс]. – URL: https://store.steampowered.com/app/708680/Octopus_Bar/. – (Дата обращения: 09.01.2023).

19. Octogeddon (2018) [Электронный ресурс]. – URL: https://store.steampowered.com/app/525620/Octogeddon/. – (Дата обращения: 09.01.2023).

20. Octodad: Dadliest Catch (2014) [Электронный ресурс]. – URL: https://store.steampowered.com/app/224480/Octodad_Dadliest_Catch/. – (Дата обращения: 09.01.2023).

21. Abdul Karim A., Gaudin T., Meyer A. et al. Procedural Locomotion of Multi-Legged Characters in Dynamic Environments // Computer Animations & Virtual Worlds – 2012. – Vol. 24 (1) – P. 3-15.

22. Van Welbergen H., Van Basten B. J. H., Egges A. et al. Real Time Animation of Virtual Humans: A Trade-off Between Naturalness and Control // Computer Graphics Forum – 2010. – Vol. 29 (8) – P. 2530-2554.

23. Kautzman R., Wise B., Yu M. et al. Finding Hank: Or How to Sim An Octopus // ACM SIGGRAPH 2016 Talks – 2016. – Article number 61.

24. Boucher E., Dirksen N. Rigging octopuses in Penguins of Madagascar // Proceedings of the 2015 Symposium on Digital Production – 2015. – P. 55–56.

25. Min S., Won J., Lee S. et al. Simulation and Control of Soft-Bodied Animals with Biomimetic Actuators // ACM Trans. Graph. – 2019. – Vol. 38 (6) – Article number 208.

26. Perf Wiki (2023) [Электронный ресурс]. – URL: https://perf.wiki.kernel.org/index.php/Main_Page. – (Дата обращения: 11.06.2023).

27. Flame Graphs visualize profiled code (2023) [Электронный ресурс]. – URL: https://github.com/brendangregg/FlameGraph. – (Дата обращения: 12.06.2023).

28. OpenMP (2023) [Электронный ресурс]. – URL: https://www.openmp.org/. – (Дата обращения: 11.06.2023).

29. Using Eigen in CUDA kernels (2023) [Электронный ресурс]. – URL: https://eigen.tuxfamily.org/dox/TopicCUDA.html – (Дата обращения: 11.06.2023).

30. Schulman J., Wolski F., Dhariwal P. et al. Proximal Policy Optimization Algorithms // arXiv:1707.06347 – 2017.

31. Intel Core i7 7700 vs Intel Core i5 14600K [Электронный ресурс]. – URL: https://www.topcpu.net/en/cpu-c/intel-core-i7-7700-vs-intel-core-i5-14600k#. – (Дата обращения: 23.01.2024).

32. Reinforcement Learning: An Introduction [Электронный ресурс]. – URL: https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf. – (Дата обращения: 26.04.2024).

33. Lazaridis L., Papatsimouli M., Kollias K.-F. et al. Hitboxes: A Survey About Collision Detection in Video Games // Fang, X. (eds) HCI in Games: Experience Design and Game Mechanics. HCII 2021. Lecture Notes in Computer Science(). – 2021. – Vol. 12789 – P. 314-326.

34. Curtis S., Tamstorf R., Manocha D. Fast Collision Detection for Deformable Models using Representative-Triangles // I3D '08: Proceedings of the 2008 symposium on Interactive 3D graphics and games. – 2008. – P. 61-69.

35. Wang M., Cao J. A review of collision detection for deformable objects // Comput Anim Virtual Worlds – 2021. – e1987.

36. Möller T. A fast triangle-triangle intersection test. // Graph Tools. – 1997. – Vol. 2(25) – P. 30.

37. Tropp O., Tal A., Shimshoni I. A fast triangle to triangle intersection test for collision detection // Comput Animat Virtual Worlds – 2006. – Vol. 17(527) –P. 35.

38. Tang M., Wang T., Liu Z. et al. I-Cloth: incremental collision handling for GPU-based interactive cloth simulation. // ACM Trans. Graph. – 2018. – Vol. 37(6) – P. 1–10.

39. Game Technologies [Электронный ресурс]. – URL: https://research.ncl.ac.uk/game/mastersdegree/gametechnologies. – (Дата обращения: 10.05.2024).

40. Arm Coordination in Octopus Crawling Involves Unique Motor Control Strategies / Levy G. et al. // Current Biology. – 2015. – Vol. 25 (9) – P. 1195-1200.