Санкт-Петербургский государственный университет
Кафедра компьютерного моделирования и многопроцессорных систем

Выпускная квалификационная работа

Визуализация качки судна в реальном времени на основе GPGPU вычислений

Гавриков Антон Александрович

Направление 02.03.02
«Фундаментальные информатика и информационные технологии»
Бакалаврская программа «Программирование и информационные технологии»

Научный руководитель
Богданов А.В.

Санкт-Петербург
2019

Содержание

1. Введение

Одним из ключевых этапов процесса проектирования судна является моделирование его поведения на взволнованной поверхности моря, выполняемое с учетом ожидаемых эксплуатационных характеристик. Подобный процесс моделирования может быть выполнен в реальных условиях на виртуальном полигоне, что позволяет в реальном времени отслеживать влияние внешних воздействий на ходовые характеристики судна с помощью датчиков, установленных на борту. Визуализация результатов такого процесса моделирования позволяет исследователю правильно и целостно воспринимать происходящие события, а также прогнозировать и своевременно реагировать на возникающие опасные ситуации. Такой подход позволяет сэкономить огромное количество средств, так как моделирование качки в опытовых бассейнах стоит больших денег и проводится не с полноразмерной версией судна, а с его уменьшенной моделью. Компьютерная программа позволяет проводить симуляцию в реальном времени, что сэкономит большие деньги при проектировании.

1.1. Виртуальный полигон

Данная исследовательская работа проводится в рамках проекта виртуального полигона. Целью этого проекта является создание системы поддержки принятия решений для моделирования, прогнозирования и предотвращения опасных физических явлений, которые могут возникнуть на судне в море. Ситуации включают

- затопление отсеков,
- пожар в отсеках,
- потеря остойчивости судна,
- волны большой амплитуды
- и т.д.

Наша задача в рамках этого проекта — разработать программу, которая имитирует океанские волны и движение судов по морю. Поскольку в рамках проекта разрабатывается система принятия решений, одним из требований является то, что мы должны делать все в режиме реального времени и предоставлять оператору визуализацию и графический интерфейс пользователя. Это основная задача для нашей команды, потому что подобные программы обычно выполняют вычисления в неинтерактивном пакетном режиме. Данный подход очень удобен для исследователя, так как предоставляет ему удобный и интуитивно понятный интерфейс для симуляции различных явлений.

Стоит отметить, что для работы подобных программ зачастую требуется мощный вычислительный кластер или суперкомпьютер, так как эти программы содержат ресурсоемкие вычисления. В этом случае возникают проблемы, так как для расчетов необходимо получить доступ к вычислительным ресурсам, что зачастую вызывает множество трудностей при оформлении необходимых бумаг. Чтобы избежать всего этого и сделать программу более доступной, в виртуальном полигоне для вычислений используется GPU, если это возможно. Так как GPU используется многими исследователями для визуализации и есть во многих современных компьютерах, это облегчает работу и доступность использования данной программы.

Другая ключевая особенность — реалистичные морские объекты. Они предоставляются в формате IGES, который является независимым от производителя стандартом для обмена объектными моделями между системами CAD.

Это позволяет подгружать множество различных судов без перевода их в специальный формат. Данный формат описывает суда аналитически, что позволяет получать точные модели. В качестве альтернативы мы также используем собственный дискретный формат VSL [1], который имеет несколько преимуществ. Во-первых, для него намного быстрее проводить триангуляцию, необходимую для загрузки судна, во-вторых, данный формат намного проще редактировать. Но у дискретного формата есть и недостаток, который выражается в том, что им невозможно точно описать наше судно.

1.2. Структура

Исходный код разделен на несколько модулей (исполняемые файлы и общие библиотеки) для оптимизации времени сборки. Структура проекта изображена на рисунке 1. Мы отказались от использования полнофункционального игрового движка. Он бы мог значительно упростить процесс разработки, но также он бы заметно уменьшил производительность программу и её гибкость. Вместо этого мы использовали библиотеку Magnum [2], которая предоставляет легкий объектно-ориентированный интерфейс для OpenGL [3] и библиотеку ImGUI для создания графического интерфейса пользователя. Для вычислений на GPU используется OpenCL [4].

Для работы используются многочисленные варианты корпусов судов, которые собраны в формате VSL в базе данных Vessel Санкт-Петербургского государственного университета. Эта база поддерживается с помощью пакета Hull [5], которая позволяет редактировать корпуса корабля и вычисляет определенные гидростатические характеристики. Кроме того, мы использовали OpenCASCADE [6] для импорта файлов IGES и их преобразования в вершины

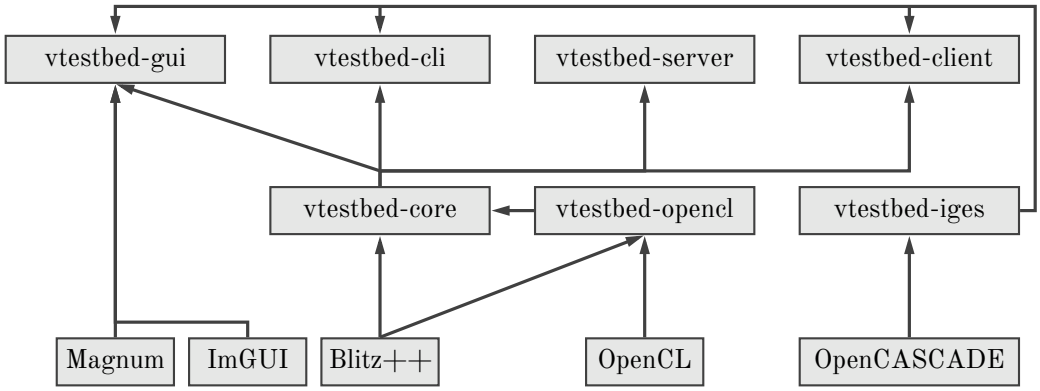

Рис. 1: Структура проекта.

и грани (полигоны).

Помимо этого в проекте используется Blitz++ [7] для многомерных массивов. Для FFT мы используем коды от Takuya OOURA [8] на CPU и на GPU от Василия Волкова [9].

2. Постановка задачи

В данной работе будет описано решение нескольких задач, возникших в ходе работы над виртуальным полигоном. Каждая из описанных ниже задач должна соответствовать требованиям к оформлению кода и включена в итоговую версию.

- Первая часть работы это создание платформы для визуализации полученных результатов. Будет рассмотрен как общий подход к построению данной модели визуализации, так и отдельные элементы программы, такие как визуализация судна, поверхности и среза. Также будет сделан краткий обзор используемого инструментария.

- Во второй части необходимо провести исследование по ускорению вычисления функции поиска смоченной поверхности. Будет рассмотрено несколько подходов, основывающихся на использовании OpenCL, OpenMP, а также OpenCL-OpenGL sharing. Необходимо выбрать наилучший метод и сделать соответствующие выводы об использовании всех версий, а также о преимуществах каждой из них.

- На финальной стадии работы будет описан процесс создания движителя морского судна от выбора математической модели для моделирования до непосредственной реализации и интеграции результатов в проект.

3. Обзор литературы

Расчет смоченной поверхности Для ускорения вычисления расчета смоченной поверхности мы планируем использовать OpenCL-OpenGL sharing, так как вершины судна используются как для визуализации, так и для расчетов. Это значит, что мы можем использовать не два буфера, а создать и использовать только один. Тот же подход используется в [10, 11]. В первой статье авторы разрабатывают систему сшивания панорамных изображений в реальном времени с помощью OpenCL. Они используют OpenCL для быстрых вычислений параллельных задач и OpenCL + OpenGL sharing, чтобы избежать копирования данных. Во второй статье Ukidave и другие описывают способы оптимизации взаимодействия CL / GL. Также был представлен механизм рендеринга слотов для лучшей производительности. Различные тесты проводятся с и без совместного использования OpenGL и OpenCL, а также с различными параметрами их алгоритма. В обоих случаях было получено ускорение при использовании совместного использования OpenGL и OpenCL, поэтому мы также решили использовать его.

Движитель В рамках данной задачи была изучена литература, в которой описывается моделирование движителя судна. Первый источник, который использовался при выполнении работы, это книга Матусиака [12], где было найдено несколько подходов к моделированию, которые отличаются сложностью. В результате был выбран простой подход, так как на данном этапе у нас нет необходимости в создании подробной модели движения. Второй источник за авторством Удачина и Соловьева [13] был использован для формирования общего представления о том, как взаимодействуют между собой винт и рули, а также как их расположение влияет на движение судна.

4. Визуализация

Важной частью нашей работы является визуализация. Она позволяет наглядно видеть результаты всех вычислений. С ее помощью можно быстрее и эффективнее находить и исправлять различного рода ошибки, возникающие в процессе разработки. В наше время наличие графического интерфейса является неотъемлемой частью многих программ, так как он позволяет на интуитивном уровне работать с приложением. Для работы с графикой используется библиотека Magnum, которая является надстройкой над OpenGL.

4.1. OpenGL

OpenGL это спецификация, которая определяет программный интерфейс для компьютерной графики. Данная спецификация получила большое распространение, так как обладает большим и удобным функционалом, а также ее реализации являются кроссплатформенными. Magnum практически не изменяет логику работы, присутствующую в оригинальной спецификации и предоставляет весь функционал OpenGL в более удобном виде, что упрощает дальнейшую разработку. Данная библиотека выбрана для разработки, так как она практически не замедляет процесс сборки, в отличие от более удобных и функциональных аналогов, таких как Unreal Engine 4 [14]. Magnum является низкоуровневым инструментом по работе с графикой, что позволяет писать программы, которые работают быстро, что очень важно в нашем случае. Создание любого графического объекта в виртуальном полигоне состоит из нескольких основных шагов.

- Первый шаг это создание объекта. Здесь необходимо задать все вершины нашего объекта и указать для них правильный обход, чтобы в дальнейшем из этих буферов правильно создались треугольники, с которыми работает OpenGL. Важно отметить, что все вершины подаются лишь один раз для оптимальности передачи данных, а далее в отдельном массиве указывается правильный обход для всех вершин.

- Второй шаг не является обязательным, но часто используется на практике. Он заключается в создании и передачи на GPU текстуры для нашего объекта. Текстуру можно удобно интерполировать на нашем объекте, что открывает дополнительные возможности при разработке. Важной особенностью является то, что в текстуре мы можем передавать не только значение цвета, но и любые другие данные, которые нам нужны. Данная особенность будет продемонстрирована в создании поверхности.

- Наконец, финальный шаг это обработка всех имеющихся данных, созданных на предыдущих шагах, с помощью шейдеров. Шейдер это программа, которая исполняется на графическом ускорителе. На вход данной программе подается одна точка и все данные о ней. На выходе она возвращает итоговое положение этой точки и ее цвет. Шейдеры бывают фрагментные и вершинные. Данное деление условно, так как возможно написать один шейдер, который будет выполнять всю работу сразу, но удобно на практике использовать подход с их делением. Вершинный шейдер используется для каких-либо изменений положений точки в пространстве, а фрагментный для работы с цветом.

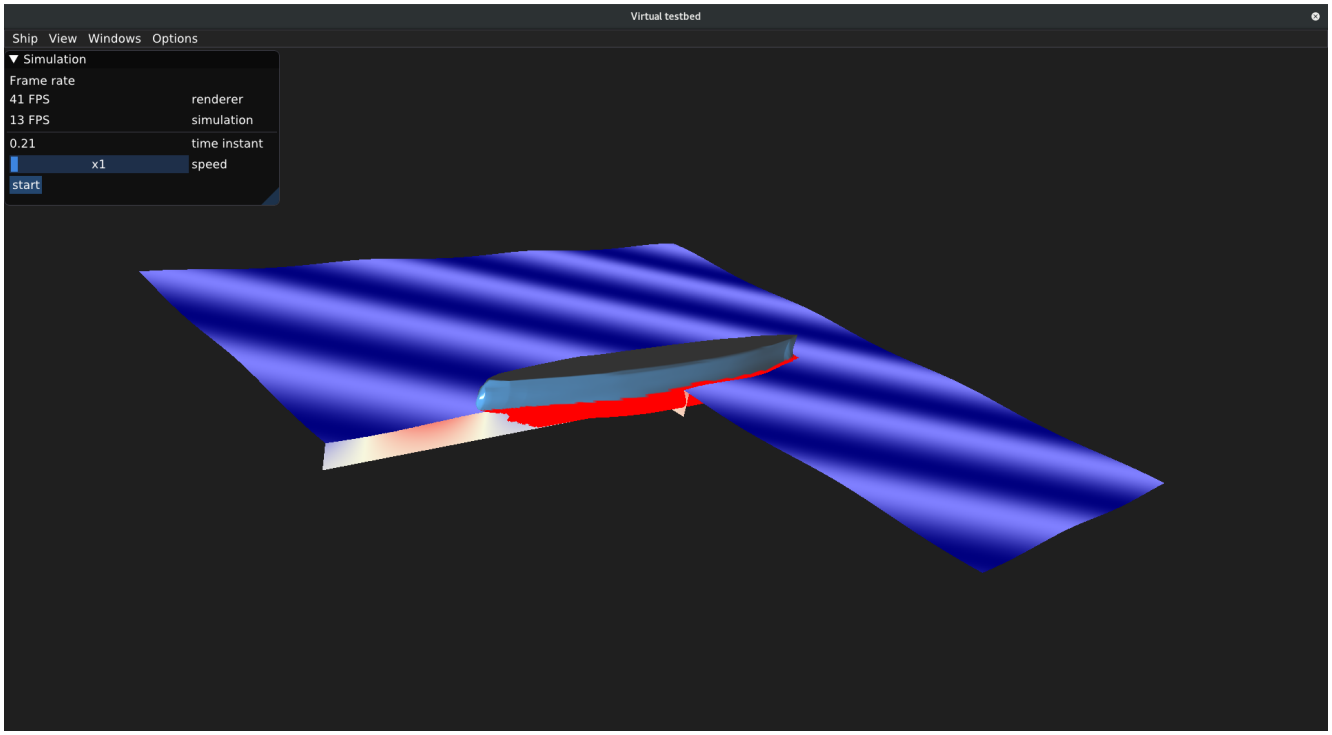

Рис. 2: Рабочее окно.

4.2. Полигон

Визуальная часть состоит из взволнованной морской поверхности ее среза, морского судна и графического интерфейса. Срез создается в выбранной нами точке поверхности. Он отображает данные о поле потенциала скоростей с помощью различных цветов. Также визуализируется морское судно. На рисунке 2 изображена текущая версия графической составляющей. В следующих разделах будет рассмотрены отдельные части визуализации.

Для удобной навигации также добавлена возможность изменения положения камеры в пространстве при помощи ввода с клавиатуры и мыши.

Поверхность На рисунке 3 можно увидеть взволнованную поверхность воды. В частности, это изображение показывает взволнованную поверхность размером сто на сто метров с амплитудой в один метр. Для создание поверхности используется сетка для которой заранее прописан оптимальный обход точек в Magnum. В нашем случае мы не изменяем форму сетки, оставляя

ее прямоугольной, а лишь растягиваем ее таким образом, чтобы ее длина и ширина соответствовали нашим требованиям. Так как размер поверхности в ходе программы меняется только по желанию исследователя, а сама поверхность изменяется после каждого шага симуляции, нет необходимости в том, чтобы заново создавать буфер вершин. Для создания рельефа используется текстура, в которой по каналу красного цвета передается высота волны относительно максимальной высоты. Максимальная высота волны передается как глобальная переменная в шейдер. Далее в вершинном шейдере происходит изменение высоты точек поверхности. Плюсы такого подхода в том, что появляется возможность сэкономить время на создании нового объекта, вместо этого можно просто обновлять текстуру. Также текстуру можно автоматически интерполировать перед использованием в шейдере, так что не придется делать это вручную. Если бы мы создавали заново объект, то пришлось бы брать большую сетку для получения гладкого изображения. Во фрагментном шейдере мы изменяем цвет относительно максимальной высоты волны. Там присутствует линейное изменение цвета от более темного в местах, где уровень воды ниже нуля до более светлого в точках, где вода выше среднего уровня. Это позволяет в итоговой версии лучше ориентироваться в высоте волн.

Срез Чтобы продемонстрировать поле потенциала скорости, мы используем срез нашей поверхности в требуемой точке. Поле потенциала скорости считается не везде, а только в области под судном(размер области округляется до ближайшей степени двойки). Для наглядности в визуализации среза используются цвета. Красный цвет предназначен для значений поля потенциала скоростей больше нуля, синий для значений меньше нуля и белый для нуля. Также этот срез хорошо отражает поле давлений, так как оно похоже на поле скоростей. В ходе работы программы срез может менять своё положение и размер каждый шаг симуляции, поэтому буфер вершин обновляется каждый шаг вместе с текстурой, которая отвечает за цвет среза. Пример среза изображен на рисунке 4.

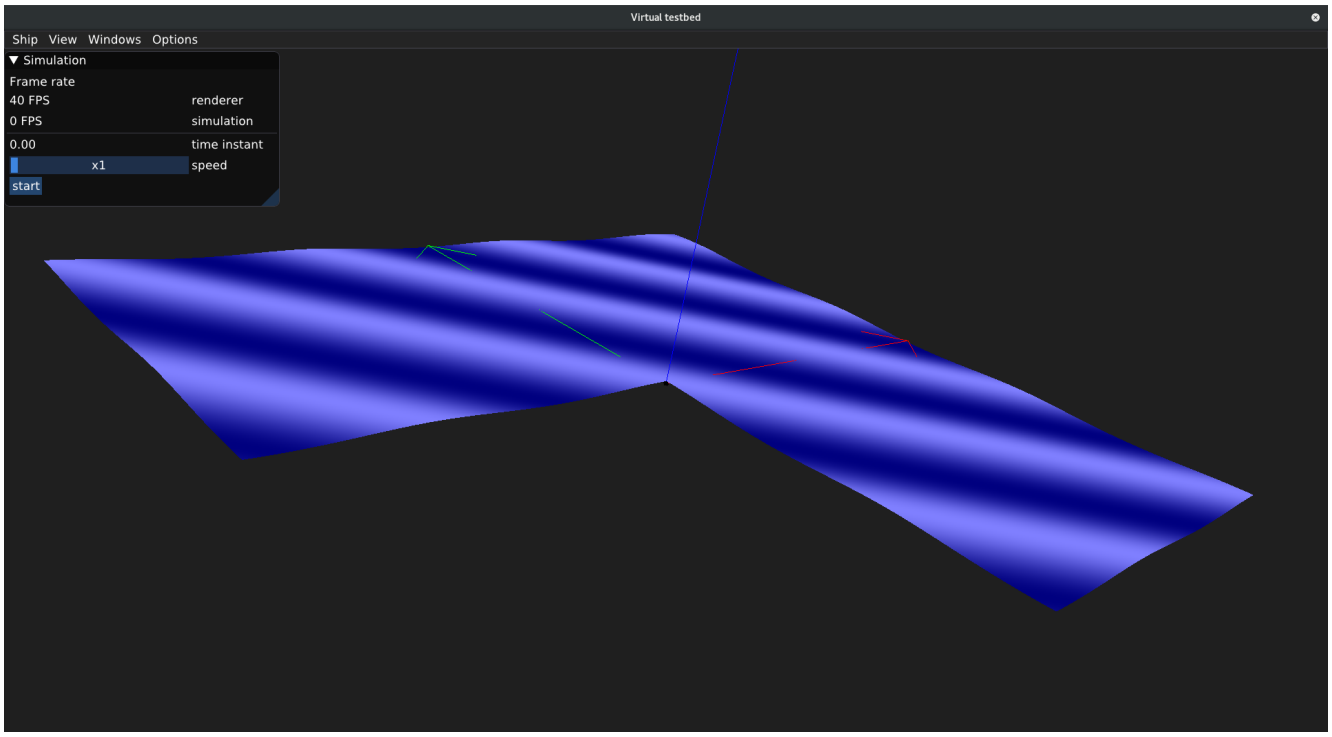

Рис. 3: Взволнованная морская поверхность.

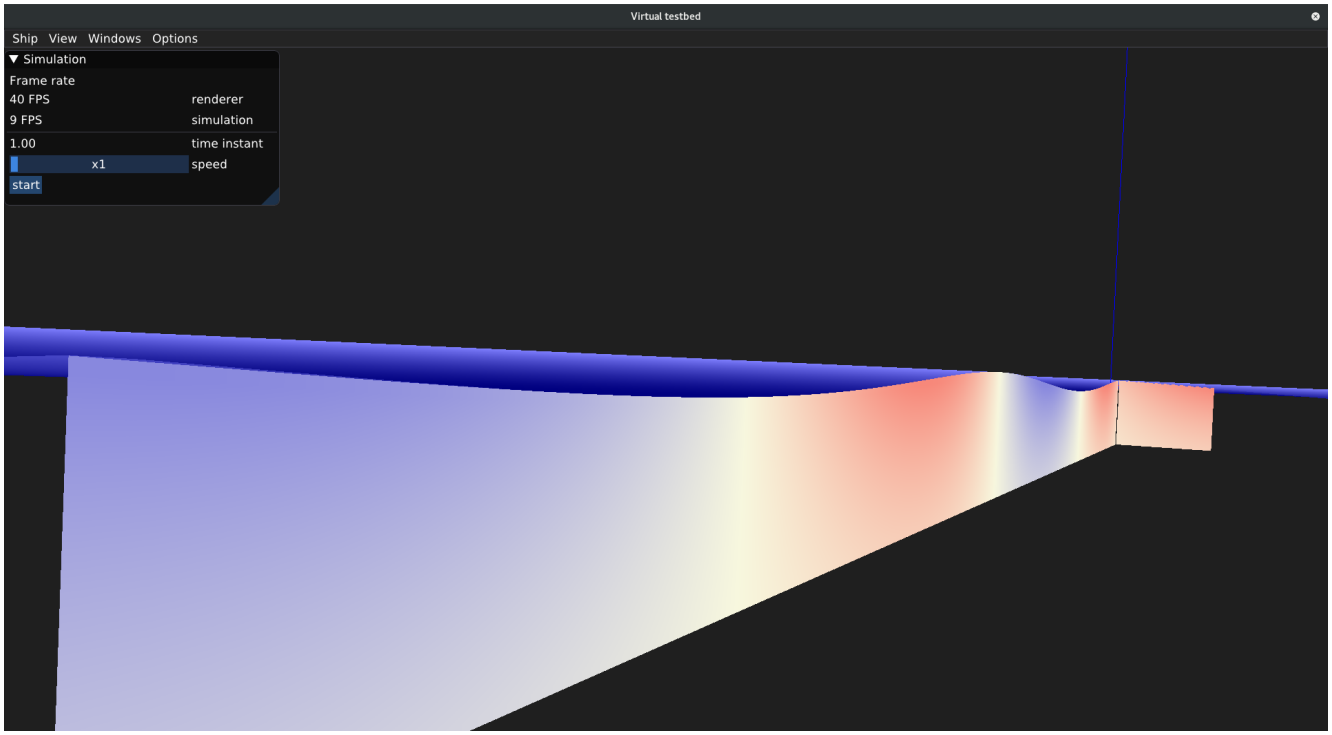

Рис. 4: Срез морской поверхности.

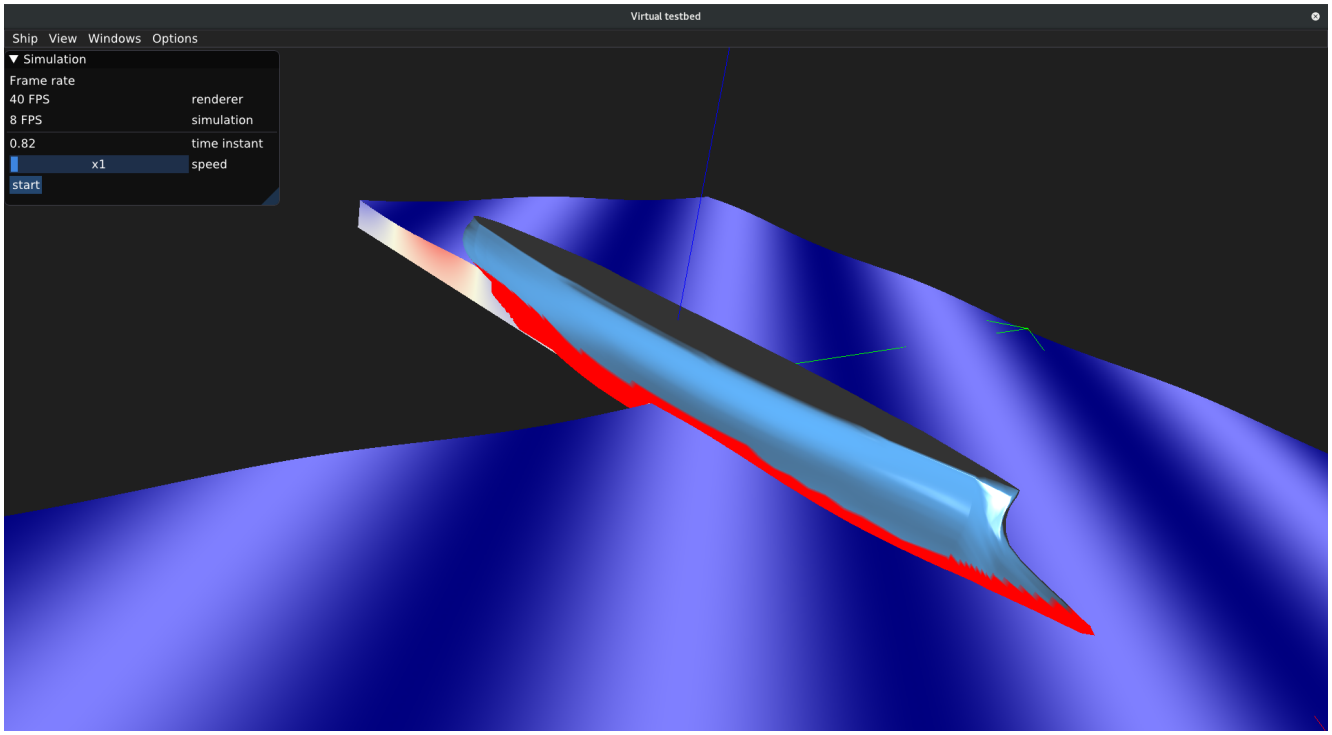

Рис. 5: Судно.

Судно На рисунке 5 вы можете увидеть судно, загруженное из нашей базы данных Vessel в формате VSL. Также мы можем использовать и IGES формат, но в процессе обработки IGES формата у нас возникли трудности с расчетом нормалей судна. Поскольку формат аналитически описывает судно, их необходимо было вычислять дополнительно. Это занимает около 20 секунд, поэтому после первого расчета мы записываем их в кэш, чтобы не повторять вычисления каждый раз при запуске. Красным цветом изображаются панели, которые при расчете считаются смоченными, т.е. располагаются в воде. Для освещения оставшейся части корабля используется шейдер, основанный на затенении Фонга [15].

GUI Наконец, в приложении существует графический интерфейс, который разрабатывался отдельно и не зависит от данной работы. Интерфейс предоставляет удобный инструментарий для работы с приложением. Есть такие возможности задания различным параметров моделирования, не выходя из программы. Также можно смотреть различные графики в реальном времени, что упрощает отладку.

5. Расчет смоченной поверхности

5.1. Смоченная поверхность

Для работы нам необходимо знать, какие из панелей судна находятся под водой. Поскольку панелей достаточно много, мы используем OpenMP для ускорения вычислений при помощи CPU. Также для всех панелей под водой мы должны найти их нормаль, площадь, объем и центр. Все эти операции легко могут быть перенесены на графический процессор. В версии OpenMP у нас есть точки останова для панелей над водой и для панелей с нулевой площадью. Мы были удивлены, когда обнаружили панели с нулевой площадью, это было вызвано форматом данных, и мы игнорируем эти панели. Наконец, модели настоящих судов состоят из десятков тысяч панелей, поэтому мы ожидаем значительного ускорения на GPU, так как GPU хорошо подходит для подобных задач. Результаты этой работы также описаны в статье [16].

Описание метода Для всех полигонов из которых состоит наше судно мы выполняем следующие действия:

- Считываем новый треугольник в системе координат судна, применяем к нему матрицу поворота и сдвигаем его, чтобы перевести в общую систему координат.
- Вычисляем центр, нормаль и площадь треугольника, а также объем тетраэдра, который получается при добавлении вершины, соответствующей началу координат.
- Для каждой вершины находим соответствующий ее координатам уровень

воды, вычисляем, какая из вершин находится выше уровня воды.

- Если все вершины выше уровня воды, то мы помечаем эту панель, как не смоченную.
- Если под водой находятся одна или две панели, то нам необходимо скорректировать значение площади, центра и объема для фигуры, которая получается путем отсечения части, не находящейся под водой. При поиске точки пересечения грани тетраэдра с водой используется метод бисекции.
- Сохраняем значение площади, объема, нормали, центра.

5.2. OpenCL и OpenGL sharing

Для решения этой задачи мы использовали технологию OpenGL + OpenCL sharing. Данная технология позволяет нам для визуализации и для расчетов использовать одни и те же данные с видеокарты, а не обновлять их с процессора. Такой подход позволяет нам избежать лишнего копирования данных с процессора. В дальнейшем при обновлении корпуса судна на видеокарте мы собираемся получить еще больше ускорения от применения такого подхода. Одна проблема, которая возникает при использовании OpenGL для визуализации и OpenCL для вычислений, заключается в том, как синхронизировать данные между двумя технологиями, работающими на одном и том же графическом процессоре? Для этого мы используем расширение совместимости CL / GL. Оно доступно на большинстве платформ GPU OpenCL. Алгоритм прост. Сначала мы связываем буфер OpenCL и буфер OpenGL, чтобы они использовали одно и то же место в памяти. Когда нам необходимо обновить буфер из

Версия	Количество панелей			
	1280	5120	20480	81920
OpenMP	1.987	3.587	10.287	38.498
OpenCL	0.560	0.646	1.721	5.379
OpenMP+OpenCL	0.540	0.615	1.530	4.639

Таблица 1: Результаты тестов. Время выполнения одного шага программы в мс. для разных версий программы.

OpenCL, мы используем функции блокировки и разблокировки, чтобы предотвратить состояние гонки. Наконец, мы вызываем `glFinish`, чтобы убедиться, что OpenGL завершил все операции с буфером.

5.3. Тесты

Для тестирования результатов использования OpenCL с совместным использованием OpenGL мы измерили время вычисления трех версий нашей функции: OpenMP, OpenCL и OpenCL с совместным использованием OpenGL. В качестве параметра мы использовали количество панелей: 1280, 5120, 20480 и 81920. Для наших тестов мы использовали компьютер с процессором AMD FX-8370 и графическим процессором GeForce GTX 1060 6GB. За время наблюдения около 90 секунд мы получили среднее время выполнения нашей функции, которое показано в таблице 1.

Версия с OpenCL + OpenGL sharing показывает лучшие результаты. Версия OpenMP выполняется примерно в шесть раз дольше по сравнению с использованием OpenCL + OpenGL sharing. Каждая панель может быть обработана отдельно, поэтому эта задача подходит для графического процессора и показывает высокое ускорение. Наконец, версия OpenCL была немного мед-

леннее, чем совместное использование OpenCL + OpenGL, от 3,7% на 1280 панелях до 15,9% на 81920. На самом деле, сейчас эта функция использует одни и те же данные каждый раз, поэтому мы копируем наш буфер с хоста на устройство только один раз, поэтому мы не видим никаких причин для ускорения обмена. Мы исходили из той точки зрения, что эта технология поможет нам в будущем, но получили интересные результаты.

Также есть график, который показывает зависимость времени выполнения основного цикла программы от количества панелей под водой (рисунок 6). У нашей функции есть цикл с некоторыми точками останова продолжения на OpenMP, поэтому можно заметить, что время выполнения на OpenMP зависит от количества панелей под водой линейно, тогда как OpenCL от этого не зависит. Кроме того, 30% времени выполнения OpenCL + OpenGL версия копирует результат с устройства на хост, но вскоре мы перепишем код, которому нужен этот результат, чтобы избежать этого копирования, поэтому мы можем получить дополнительно 30% ускорения.

5.4. Проблемы при реализации

При дальнейшем внедрении OpenCL + OpenGL sharing возникла проблема в зависимостях и синхронизации. Нам необходимо подключать OpenGL везде, где используется sharing, потому что заново создаются GL буферы, которые необходимы для работы. Так как для массивов память выделяется динамически, буферы придется изменять часто. Все это привело к усложнению в логике работы программы и подключению библиотеки OpenGL в основную вычислительную часть программы. Это вынудило нас отказаться от использования данной технологии и вернуться к обычной OpenCL версии программы,

которая практически не уступает в скорости работы, а также намного проще в плане разработки.

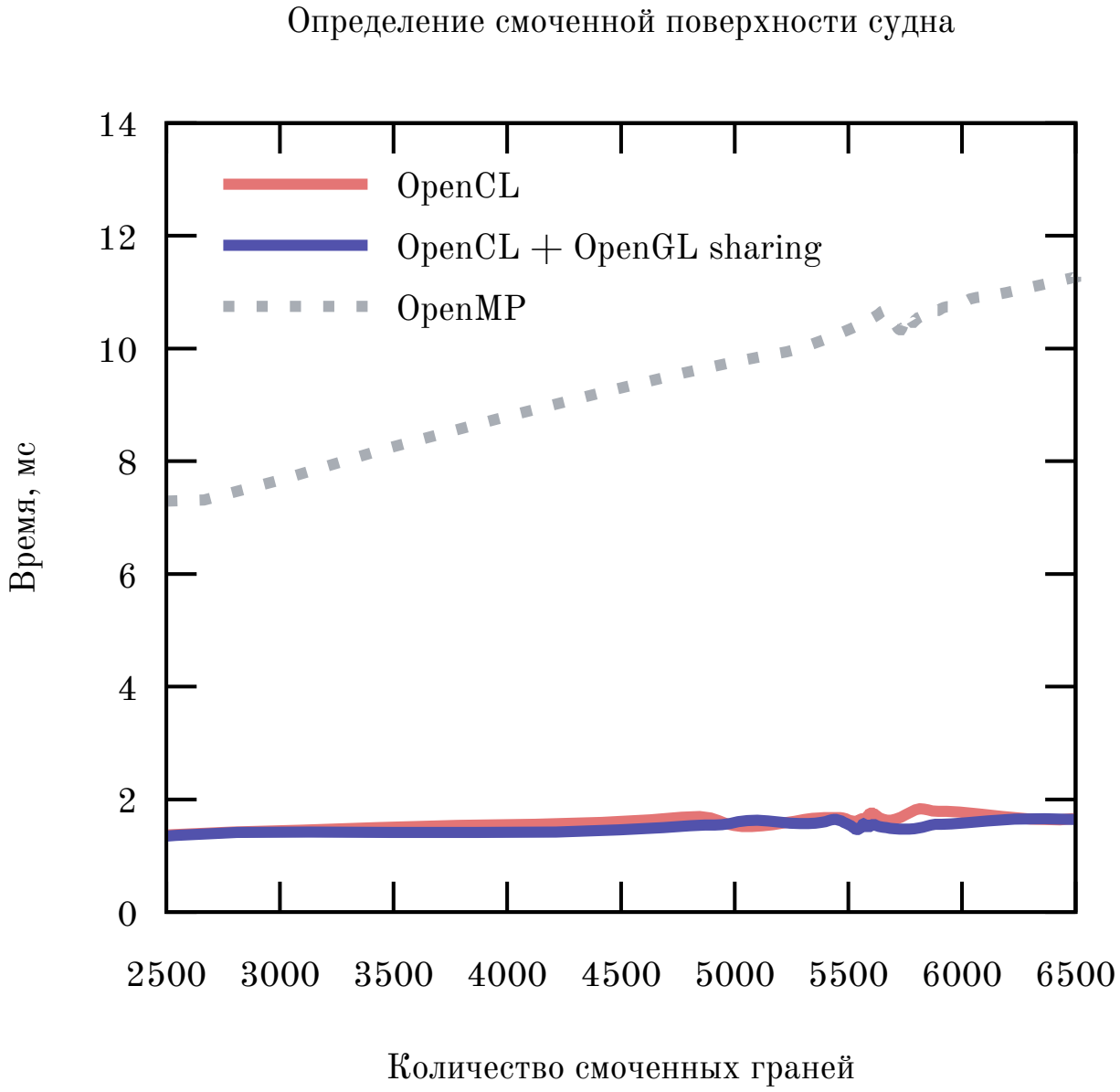

Рис. 6: Время выполнения, мс. различных версий функции в зависимости от количества панелей.

6. Движитель

Движитель судна это механизм, который состоит из двух основных частей: винт и руль. Винт формирует тягу судна, а руль влияет на направление движения судна и создает момент скорости, за счет которого поворачивается судно. Винты и рули бывают разных типов, что соответственно влияет на ходовые характеристики судна. Но в нашей работе нам нужна простая модель, которую можно быстро реализовать и которая не будет требовать больших вычислительных мощностей.

В качестве одного из допущений, мы не учитываем направление вращения винтов. Изначально судно с одним винтом, руль которого находится в начальном положении (т.е. параллельно корпусу самого судна), будет откланяться в сторону за счет сил, создаваемых винтом при вращении. Мы не учитываем это при разработке нашей модели и считаем, что в этом случае судна будет двигаться прямо. Важно упомянуть, что если на судне установлена два одинаковых винта, которые вращаются в разных направлениях, то судно должно двигаться прямо, так как силы будут уравновешивать друг друга.

Еще одно допущение это тип и количество рулей. Так как на реальных судах количество рулей может быть более одного, каждый из таких вариантов по-своему влияет на движение судна. Также винты бывают нескольких типов. В нашем подходе мы считаем, что у судна только один руль, который характеризуется углом своего поворота.

В итоге были выбраны следующие формулы.

Для скорости:

$$X_{\text{prop}} = Zpn^2D^4K_T \tag{1}$$

Здесь Z — количество винтов, n — оборотов винта в секунду, D — диаметр пропеллера, p — плотность воды, K_T — специальный параметр, описывающий

технические характеристики винта.

Для момента скорости:

$$Y_R = r \cos \delta \tag{2}$$

Здесь r — это радиус-вектор от центра масс судна до одного из его рулей, δ — угол поворота руля.

Прототип движителя позволяет добавлять необходимое количество винтов и один руль, изменять их параметры, а также получать на выходе общую скорость и момент скорости, которые прилагаются к судну. Данный прототип был добавлен в основой проект. Для выявления недочетов было проведено тестирование, в ходе которого было найдено несколько ошибок в имеющемся программном коде. Первая ошибка заключалась в неправильном вычислении итогового момента силы. Проблема была в неправильном вычислении момента сил гравитации. Вторая ошибка заключалась в отсутствии перевода из глобальной системы координат в локальную при расчете скорости и положения судна в следующий момент времени методом Рунге—Кутта [17, 18].

7. Вывод

Была реализована система для визуализации процессов виртуального полигона, состоящая из нескольких элементов. Процесс визуализации практически не занимает времени работы, так как полностью отрисовывает картинку за 2 миллисекунды. Также было получено ускорение в расчете смоченной поверхности. В результате была выбрана версия только с OpenCL, так как OpenGL sharing требует подключения большого количества лишних зависимостей в вычислительную часть программы, а также усложняет процесс разработки. Наконец, была создана модель движителя, которая способна простейшим способом моделировать работу винтов и руля, а также задавать их параметры.

8. Заключение

В результате была разработана версия программы, которая работает в асинхронном режиме отрисовки в 60 кадров в секунду. Реализован весь необходимый на данный момент инструментарий для визуализации, который включает в себя отрисовку взволнованной поверхности, судна и среза поверхности. При помощи GPU было получено существенное ускорение вычисления смоченной поверхности, что снижает требования к вычислительным способностям конечной машины и расширяет возможности для исследования с большим количеством параметров. И также был добавлен движитель, с помощью которого исследователь может задавать движение судна.

Список литературы

[1] Bogdanov Alexander, Khramushin Vasily. Vessel: Blueprints for the analysis of hydrostatic characteristics, stability and propulsion of the ship (in Russian). — 2015. — Access mode: http://www1.fips.ru/fips_servl/fips_servlet?DB=EVM&DocNumber=2015621368&TypeFile=html.

[2] Magnum Engine. — Access mode: https://magnum.graphics/.

[3] OpenGL - The Industry Standard for High Performance Graphics. — Access mode: https://www.opengl.org/.

[4] OpenCL - The Open Standard for Parallel Programming of Heterogeneous Systems. — Access mode: https://www.khronos.org/opencl/.

[5] Khramushin Vasily. Analytic ship hull shape construction, wave resistance calculations, theoretical blueprint feature curve calculations, and ship stability diagrams (in Russian). — 2010. — Access mode: http://www1.fips.ru/fips_servl/fips_servlet?DB=EVM&DocNumber=2010615849&TypeFile=html.

[6] OpenCASCADE. — Access mode: https://www.opencascade.com/.

[7] Veldhuizen Todd L. Blitz++: The Library that Thinks it is a Compiler // Advances in Software Tools for Scientific Computing / Ed. by Hans Petter Langtangen, Are Magnus Bruaset, Ewald Quak. — Berlin, Heidelberg : Springer Berlin Heidelberg, 2000. — P. 57–87.

[8] Ooura Takuya. General Purpose FFT (Fast Fourier/Cosine/Sine Transform) Package. — 2006. — Access mode: http://www.kurims.kyoto-u.ac.jp/~ooura/fft.html.

[9] Volkov Vasily, Kazian Brian. Fitting FFT onto the G80 architecture // University of California, Berkeley. — 2008. — Vol. 40.

[10] Real-time spherical panorama image stitching using opencl / Wei-Sheng Liao, Tung-Ju Hsieh, Wen-Yew Liang et al. // Proceedings of the International Conference on Computer Graphics and Virtual Reality (CGVR). — 2011. — P. 1.

[11] Ukidave Yash, Gong Xiang, Kaeli David. Performance evaluation and optimization mechanisms for inter-operable graphics and computation on GPUs // Proceedings of Workshop on General Purpose Processing Using GPUs / ACM. — 2014. — P. 37.

[12] Matusiak J. Dynamics of a Rigid Ship, Aalto University Publication Series Science+ Technology // Unigrafia Oy, Helsinki, Finland. — 2013.

[13] Удачин ВС, Соловьев ВБ. Судовождение на внутренних водных путях // М.: Транспорт. — 1990.

[14] What is Unreal Engine 4. — Access mode: https://www.unrealengine.com/en-US/.

[15] Phong Bui Tuong. Illumination for computer generated pictures // Communications of the ACM. — 1975. — Vol. 18, no. 6. — P. 311–317.

[16] Real-time visualization of ship and wavy surface motions based on GPGPU computations / Anton Gavrikov, Andrei Ivashchenko, Ivan Gankevich et al. — 2018.

[17] Mathews John H., Fink Kurtis D. Numerical methods using MATLAB. — 4th edition. — London : Pearson Prentice Hall, 2004.

[18] Mathews John H., Fink Kurtis D. Runge—Kutta–Fehlberg method

(RKF45). — 2004. — Access mode: http://maths.cnam.fr/IMG/pdf/RungeKuttaFehlbergProof.pdf.