САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Математико-механический факультет

Кафедра системного программирования

Разработка и реализация алгоритмов деформирования трехмерной геометрии анатомии человека на основе WebGL

Дипломная работа студента 545 группы

Лушникова Андрея Сергеевича

Научный руководитель	.....	старший преподаватель
	/подпись/	Антипов И.Г.
Рецензент	.....	аспирант
	/подпись/	Петров А.Г.
“Допустить к защите”	.....	д.ф.-м.н., проф. Терехов А.Н.
заведующий кафедрой,	/подпись/	

Санкт-Петербург

2012

SAINT PETERSBURG STATE UNIVERSITY

Mathematics & Mechanics Faculty

Software Engineering Chair

Development and implementation of geometry algorithms based on WebGL for 3D human anatomy deformations

by

Andrei Lushnikov

Graduate paper

supervisor	.....	senior lecturer I.G. Antipov
reviewer	.....	graduate student A.G. Petrov
"Approved by" Head of Department	.....	Professor A.N. Terekhov

Saint Petersburg

2012

Содержание

1 Постановка задачи

Каждый программный продукт на протяжении всех стадий жизни сопровождает много разнообразных проблем. Это и проблемы его поддержки, и проблемы переносимости, и проблемы контроля версий и устаревания. Эти проблемы характерны как для рядовых программ, которые знакомы каждому пользователю, так и для специализированных программ, используемых специалистами в их ежедневной практике для оценки, визуализирования и прогнозирования результатов пластических операций. Один из современных подходов к распространению программного обеспечения, носящий название “Software as a Service” - призван решить целый класс таких проблем.

Software-as-a-Service (далее “SAAS”) - модель использования программного обеспечения, при которой программный продукт выполнен в виде Web-приложения, и разработчик самостоятельно управляет его развитием. Пользователи имеют доступ к приложению через сеть интернет, при этом они избавлены от затрат, связанных с установкой, обновлением и поддержкой работоспособности оборудования и работающего на нем программного обеспечения.

Программное обеспечение, разработанное специально для врачей и решающее различные медицинские задачи, зачастую также обладает большим количеством недостатков. Оно не обладает свойством кросс платформенности, а его установка зачастую трудоемка. Оно предъявляет дополнительные требования к рабочим станциям врачей, что сужает область применения.

Основной задачей данной дипломной работы было разработать графический редактор, позволяющий загружать и обозревать трехмерную модель, а так же алгоритм, позволяющий пользователю изменять части геометрии загруженной модели. Редактор должен поддерживать

загрузку трехмерных моделей, представленных в наиболее популярном на данный момент открытом формате *.OBJ. Разработанное приложение должно полностью следовать модели SAAS, что избавило бы его от большого количества недостатков, присущих программному обеспечению в общем и медицинскому программному обеспечению в частности. Программный редактор должен был быть как можно менее зависим от конфигурации клиентской платформы, что упростило бы его использование в больницах. Отдельным достоинством приложения была бы возможность использования на планшетных компьютерах.

2 Архитектура

2.1 Описание архитектуры

Для выполнения поставленных целей было решено использовать клиент-серверную архитектуру. В роли клиента выступает web-приложение, предоставляющее пользователю необходимый интерфейс для взаимодействия, а серверная часть отвечает за конвертацию OBJ-файлов. Обе части взаимодействуют между собой по средствам технологии AJAX и специально разработанного для целей приложения протокола, основанного на сообщениях в формате JSON особого вида. Таким образом взаимодействие между компонентами системы происходит по средствам внешнего API, предоставленного серверной частью, что соответствует принципу инкапсуляции и способствует заменяемости любого из компонентов архитектуры.

2.1.1 Клиент

Клиентская часть реализована в виде web-приложения, написанного на HTML5.0 и Javascript. Для отображения графики используется перспективная технология WebGL.

WebGL[1] - это библиотека для программного обеспечения, которая расширяет возможности языка программирования JavaScript, позволяя ему создавать интерактивную 3D графику внутри любого совместимого с ней web-браузера. Код на WebGL выполняется с помощью вычислительных ресурсов видеокарты клиента. Технология разрабатывается промышленным консорциумом Khronos Group, который специализируется на выработке открытых стандартов интерфейсов программирования в области создания и воспроизведения динамической графики и звука. Активное участие в разработке и внедрении стандарта

так же принимают разработчики браузеров Apple Safari, Google Chrome, Mozilla Firefox, Opera, а также специалисты компаний AMD и NVidia. На данный момент технология поддерживается в последних версиях браузеров Safari, Mozilla, Opera и Chrome, а так же в браузере Internet Explorer при использовании специального дополнения IEWebGL. Среди мобильных устройств данная технология уже поддерживается в браузере телефона Nokia N900, а так же в Safari Mobile начиная с версии операционной системы iOS 4.2.

Выбор технологии WebGL в качестве средства для отображения трехмерной графики преследует цель достижения кросс-платформенной реализации. Перспективы развития этой технологии в отношении рынка мобильных платформ являются самыми многообещающими по сравнению с остальными кросс-платформенными технологиями. [1]

2.1.2 Сервер

Сервеная часть была реализована с использованием технологии с открытым исходным кодом “Node.js”. Эта технология позволяет исполнять JavaScript на стороне сервера, причем в качестве виртуальной машины для исполнения JavaScript-кода используется высокопроизводительная виртуальная машина “V8”, разработанная компанией Google и используемая в браузере Google Chrome.

“Node.js”[2] - платформа с открытым исходным кодом для построения приложений, основанных на асинхронных I/O операциях с использованием языка JavaScript. Несмотря на распространенное мнение, платформа не является первой в своем роде и в некотором роде наследует принципы, заложенные в асинхронном I/O фреймворке “EventMachine” для языка программирования Ruby. Однако большим достоинством этой платформы

[1] стоит отметить, что использование WebGL разрешено в браузере Mobile Safari только в контексте рекламных объявлений iAd

перед “EventMachine” можно назвать язык программирования EcmaScript стандарта 5.0, или JavaScript, применяемый на платформе “Node.js” и исторически ориентированный на асинхронные взаимодействия. Благодаря отсутствию в его спецификации I/O операций (что в некотором смысле удивительно для web-ориентированного языка), появилась возможность создать необходимую библиотеку асинхронных I/O операций, которая и стала стандартной при использовании “Node.js”-приложений.

Использование одного и того же языка на стороне клиента и сервера дает несколько достоинств разработчикам. В первую очередь появляется возможность эффективного переиспользования одного и того же кода как на клиентской, так и на серверной частях приложения. Примером такого переиспользования может служить код проверки данных формы, который из соображений безопасности и целостности обязан присутствовать на стороне сервера, а из соображений удобства пользования интерфейсом и минимизации клиент-серверных взаимодействий должен быть также и на клиентской стороне.

Интересной особенностью технологии “Node.js”, используемой в качестве серверной платформы, можно назвать высокую производительность и чрезвычайно легкую масштабируемость. Эти особенности связаны с асинхронной моделью чтения и записи данных. Таким образом, в противоположность классическому web-серверу, запускающему обработку каждого запроса в отдельном потоке, но выполняющем в этом потоке синхронные операции, “Node.js” использует только один системный поток. В случае необходимости масштабирования “Node.js” серверов, можно запустить несколько серверов приложения, объединив их по средством обратного proxy с балансером. Таким образом приложение, разработанное на этой платформе, фактически не имеет верхней границы по обрабатываемой нагрузке.

За время своего существования серверная платформа “Node.js”

приобрела большую инфраструктуру. Благодаря наличию фреймворка Connect, ставшего стандартом де-факто для приложений и задавшего стандарт для разработки сторонних модулей, разработчикам доступно большое количество модулей. Менеджер пакетов обеспечивает централизованный доступ и быструю установку сторонних модулей, становится возможным быстрое прототипирование как самого приложения, так и его отдельных возможностей.

Несмотря на то, что преимущества от использования динамического слабо-типизированного прототипно-ориентированного языка на стороне сервера заметны далеко не сразу, ввиду выше указанных причин является обоснованным и оправданным.

2.2 Обоснование выбора технологий

В процессе выбора технологий для решения поставленных задач было рассмотрено множество альтернативных решений для каждого из участков архитектуры. Каждое из таких решений было оценено в соответствии с поставленной задачей и предъявляемыми к ней требованиями. Результаты этого исследования определили выбор в пользу описанных технологий. Ниже приведены решения и подходы, рассмотренные в качестве альтернатив для реализации поставленных задач, и дан их краткий анализ.

2.2.1 Использование Flash на стороне пользователя

При разработке интерактивных web-приложений большой популярностью обладает технология Flash, позволяющая создавать графически богатые интерфейсы. Для функционирования этой технологии в браузере необходимо специальное дополнение, устанавливающееся пользователем отдельно. За счет своей распространенности технология

обладает обширным сообществом, большим количеством учебных материалов и многочисленными движками для обработки и отображения трехмерной графики.

Несмотря на вышеперечисленные достоинства, технология Flash на данных момент не поддерживается ни на одной из популярных мобильных платформ. Компания Adobe, владеющая всеми правами на технологию Flash и являющаяся единственным ее разработчиком, официально отказалась от развития мобильного направления развития платформы. В задачи же данной работы входило создание кросс-платформенного приложения, в том числе способного выполняться на планшетных компьютерах. Ввиду отсутствия такой теоретической возможности, от этой технологии было решено отказаться.

2.2.2 Использование 3D-фреймворка Unity

Платформа Unity зарекомендовала себя в качестве мощного средства создания двумерных и трехмерных приложений под консоли и настольные компьютеры под управлением OS X и Windows. Благодаря кросс-платформенной сущности движка, существует возможность перекомпилировать приложения, написанные с использованием этой технологии, под целевые платформы. Такое решение позволяет покрыть мобильный сегмент, однако не соответствует модели распространения SAAS, а значит не избавляет разработчиков от проблем фрагментирования версий приложения.

Платформа также занимает свою нишу в области web-приложений, предоставляя разработчикам разрабатывать браузерные игры на Unity и использованием одной из следующих опций

- Использование специального дополнения для браузера, исполняющего код Unity-приложений. Это дополнение должно

быть установлено пользователем.

- Использование экспериментального движка на основе Flash. У пользователя должно быть установлено дополнение, позволяющее исполнять код технологии Flash.

К сожалению, ни одна из предложенных опций не поддерживается на мобильных системах. Ввиду того, что технология не удовлетворяет поставленным условиям, от ее использования было решено отказаться.

2.2.3 Использование Ruby on Rails

Ввиду поставленных задач возникла необходимость использования серверной части в архитектуре приложения. При разработке любого серверного решения акценты ставятся на вопросы масштабирования, а так же возможности быстрого прототипирования приложения. Используемый фреймворк должен поддерживать роутинг запросов соответствующим обработчикам и поддерживать шаблоны для генерации html-страниц. Этим параметрам удовлетворяют несколько популярных серверных решений: фреймворк “Ruby on Rails”, фреймворк “Django” и платформа “Node.js” с серверным фреймворком “Express.js”[3]. Ввиду концептуальной схожести фреймворков “Ruby on Rails” и “Django” ограничимся сравнением Ruby on Rails и “Node.js”.

“Ruby on Rails” является более солидным и развитым проектом, нежели “Express.js”. Обе платформы предлагают менеджеры пакетов и большую библиотеку общедоступных open-source решений для самых разных задач, что делает прототипирование одинаково легким на обоих платформах. Однако использование “Ruby on Rails” налагает на разработчика необходимость конвертации объектов из одного языка в другой при клиент-серверных взаимодействиях, а так же заставляет отказаться от потенциальной возможности переиспользовать некоторые

участки кода на стороне сервера, написанные для исполнения на стороне клиента. Кроме того, “Node.js” приложения легче масштабируются.

Ввиду всего вышесказанного было решено использовать на стороне сервера платформу “Node.js”.

3 Детали реализации

3.1 Клиентские технологии

3.1.1 Использование “Three.js”

WebGL является достаточно низкоуровневой технологией, в которой большинство операций выполняется путем определения и исполнения пиксельных и вершинных шейдеров. Шейдеры - это небольшие функции, написанные на специальном языке программирования шейдеров, которые выполняются графической картой и которые осуществляют преобразования вершин или пикселей[11]. Несмотря на то, что это открывает большие возможности в области создания и обработки трехмерной графики, это достаточно трудоемкий процесс, часто чреватый ошибками. Для повышения скорости разработки было решено использовать один из нескольких фреймворков-оберток вокруг WebGL. Выбор пал на активно развивающийся проект с открытым исходным кодом “Three.js”.

“Three.js”[4] содержит набор классов, написанных на языке JavaScript, которые предоставляют многоуровневую абстракцию над процессами создания и отображения сцены. Пользователи “Three.js” избавлены от необходимости самостоятельно писать шейдерные процедуры [2] и могут мыслить объектами и материалами. Все модели хранятся в виде JavaScript-объектов, которые на каждом этапе рендеринга при необходимости синхронизируются с буферами WebGL. Важно отметить, что все расчеты геометрии (например, пересечение луча с объектами сцены) программируются на языке JavaScript и выполняются на уровне виртуальной JavaScript машины, а значит на прямую зависят от ее

[2] однако такая возможность в рамках движка им предоставляется

быстродействия.

3.1.2 Шаблон Strategy для обеспечения слабой связанности

В созданном приложении есть два инструмента, которыми пользователь может взаимодействовать с объектом.

1. Инструмент "Рука". Этот инструмент используется для свободного вращения объекта вдоль осей
2. Инструмент "Деформация". Этот инструмент используется для изменения геометрии объекта

В процессе написания приложения встала задача разработать такую архитектуру, чтобы добавление новых инструментов в последствии было как можно более простым, а количество логических связей между разными модулями приложения при этом почти не увеличивалось. Для этого было решено использовать шаблон проектирования Strategy[13], который определяет семейство алгоритмов и обеспечивает их взаимозаменяемость.

В рамках приложения шаблон Strategy был реализован следующим образов. Каждый инструмент описывается классом такого вида

```
function FooTool(context) {
    this.context = context;
}

FooTool.prototype.setUp = function() {
    // set up event listeners for context
}

FooTool.prototype.tearDown = function() {
    // remove all set event listeners
}
```

Когда пользователь выбирает инструмент Foo, у текущего контекста вызывается метод

applyMouseStrategy(FooTool), который реализован следующим образом

```
1 ManagedObject.prototype.applyMouseStrategy = function(Strategy) {
2     if (this.mouseStrategy != null) {
3         this.mouseStrategy.tearDown();
4     }
5     this.mouseStrategy = new Strategy(this);
6     this.mouseStrategy.setUp();
7 }
```

В этом методе происходит следующая цепочка событий

1. Метод проверяет, есть ли какой-нибудь действующий инструмент, и если есть, то вызывает у него метод tearDown().

2. Создается новый инструмент с помощью переданной в метод функции-конструктора класса нового инструмента, созданный объект присваивается внутренней переменной выбранного инструмента

3. У нового инструмента вызывается метод setUp(), который устанавливает инструмент на контекст

Таким образом инструменты приложения никак не опираются на остальные модули приложения, а создание нового инструмента ограничивается только реализацией его исходного кода и добавлением обработчика события, который бы активировал нужный инструмент.

3.1.3 Шаблон Observer для дуплексного взаимодействия

В процессе разработки приложения встала проблема организации двусторонних коммуникаций, которые включали бы в себя как сообщения от отображения к контроллеру, так и обратные сообщения от контроллера к отображению, инициированные моделью. Потребность в такого рода взаимодействиях возникла в связи с необходимостью адаптировать пользовательский интерфейс исходя из тех действий, которые делает пользователь.

Для решения этой проблемы было решено добавить в архитектуру глобальный объект EventBus, представляющий собой шину событий, и реализующий шаблон Observer[12].

```
1  // jQuery based implementation of event bus
2
3  var EventBus = {
4      subscribe: function(event, fun) {
5          $(this).bind(event, fun);
6      },
7      publish: function(event, arg) {
8          $(this).trigger(event, arg);
9      }
10 }
```

Шина событий - объект, реализующий простой интерфейс из двух методов:

1. subscribe(event, callback) - подписаться на событие event с помощью функции обратного вызова callback

2. publish(event, arg) - вызвать событие event с параметром arg

Реализация этого интерфейса, представленная в рамках проекта, основывается на фреймворке jQuery[8]. События задаются строковыми литералами, составленными по правилу “<Имя класса>:<имя события>”. Правило, составленное таким образом, позволяет исключить нежелательные пересечения в именах событий. Важно отметить, что на данный момент следование этому правилу нигде не проверяется и остается на совести разработчика.

Благодаря шине событий модель приложения получает возможность оповещать о любых своих внутренних изменениях. Таким образом контроллер не только передает события от отображения к модели, но и, обрабатывая соответствующие события модели, получает возможность адаптировать интерфейс так, чтобы он соответствовал актуальному

состоянию модели.

3.1.4 Использование технологии AJAX

В ходе разработки сервиса было установлено, что пользователь в процессе работы совершает многочисленные запросы к серверу, как для загрузки модели в окно редактора, так и для добавления новой obj-модели в инспектор объектов и вызова серверных вычислений. Каждое действие требовало перезагрузки страницы, что значительно ухудшало опыт использования приложения.

В связи с этими причинами было решено полностью перейти к взаимодействию между клиент-серверной частью по средствам технологии AJAX (Asynchronous Javascript And XML). Технология AJAX базируется на двух основных принципах

1. Использование технологии обращения к серверу без перезагрузки страницы

2. Использование DHTML для динамического изменения содержимого страницы

Классические примеры применения технологии AJAX используют JavaScript-объект XMLHttpRequest. Однако этот подход чреват проблемами совместимости с некоторыми старыми браузерами. Например, в браузере Microsoft Internet Explorer объект XMLHttpRequest не определен, однако есть его аналог с другим именем. Поэтому для его создания рекомендуется применять подобный код

```
1 function getXmlHttp(){
2   var xmlhttp;
3   try {
4     xmlhttp = new ActiveXObject("Msxml2.XMLHTTP");
5   } catch (e) {
6     try {
```

```
7         xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
8       } catch (E) {
9         xmlhttp = false;
10      }
11    }
12    if (!xmlhttp && typeof XMLHttpRequest!='undefined') {
13      xmlhttp = new XMLHttpRequest();
14    }
15    return xmlhttp;
16  }
```

Изучение тематических ресурсов, посвященных этой технологии, показало, что объекты XMLHttpRequest, полученные универсальным способом, описанным выше, несколько отличаются по функциональности. В качестве примера можно привести метод abort(), который должен обрывать текущий запрос, однако в случае Internet Explorer этого не делает.

Таким образом, использование XMLHttpRequest порождает целую серию проблем не только по написанию кросс-браузерного кода, но и по его дальнейшему сопровождению. Для их решения было решено использовать фреймворк jQuery, который помимо специальных функций, “оборачивающих” XMLHttpRequest и решающий таким образом многочисленные проблемы с его использованием, предоставляет поддержку языка запросов к элементам страницы “XPath”[?], что значительно облегчает динамическое манипулирование содержимым страницы.

3.2 Серверные технологии

3.2.1 Фреймворк “Express.js”

При разработке web-сервиса были поставлены следующие общие задачи:

1. Требуется некоторый механизм сопоставления действий адресам,

которые были запрошены пользователем

2. Необходимо поддерживать возможность генерации html-страниц на основе информации, доступной серверу в момент запроса

Для решения этих задач было решено использовать фреймворк для разработки REST-сервисов на платформе “Node.js” под названием “Express.js”. Фреймворк состоит из нескольких частей и решает следующие задачи:

1. маршрутизация запросов пользователя
2. разбор тела запроса (в случае передачи форм и загрузки файла)
3. рендеринг представлений[3] на основе набора параметров

Фреймворк позволяет строить таблицу маршрутизации запросов с помощью обработчиков основных глаголов REST-сервисов: REST, POST, PUT, DELETE. Необходимо особо отметить, что протокол связи HTTP 1.0 поддерживает только три метода[14]: HEAD, GET, POST, а язык разметки HTML вплоть до версии 4.0 не содержит методов PUT и DELETE в качестве опций для отправки форм на сервер.

Поэтому для того, чтобы “Express.js” мог осуществлять маршрутизацию запросов по отправке форм с глаголами PUT и DELETE, формы должны содержать дополнительно скрытое поле с именем method и значением, содержащим имя желаемого метода.

3.2.2 Рендеринг web-страниц

Основной задачей рендеринга является формирование страницы, содержащей информацию, специфическую для данного запроса.

[3]подразумевается генерация HTML-кода web-страниц, который будет отправлен в ответ на запрос пользователя

Наиболее естественным подходом для создания таких страниц является использование всевозможных HTML-препроцессоров, которым можно передать дополнительные параметры. Так, например, на платформе “Ruby on Rails” рендеринг представлений осуществляется за счет обработки erb-файлов [4], а на платформе “Java EE” для этих целей используются Java Server Pages. Стандартом де-факто для фреймворка “Express.js” является технология Jade.

Jade[7] - препроцессор HTML-кода, написанный на языке JavaScript. Jade преобразует код, написанный на специально разработанном DSL[5] (далее - просто язык Jade), в HTML-разметку. Одним из несомненных достоинств языка Jade является двумерный синтаксис, хорошо себя зарекомендовавший для описания структур с глубокой вложенностью, а потому хорошо подходящим для описания компонентов и элементов HTML страницы.

Показательным примером использования технологии Jade будет следующий листинг. Рассмотрим следующий код на языке Jade

```
1  !!!
2  html(lang="en")
3    head
4      title= pageTitle
5      script(type='text/javascript')
6        if (foo) {
7          bar()
8        }
9    body
10     h1 Jade - node template engine
11     #container
12       if youAreUsingJade
13         p You are amazing
14       else
15         p Get on it!
```

[4]embedded ruby file

[5]Domain Specific Language

Препроцессор, запущенный на данном коде с параметром pageTitle = "Hello, Jade", генерирует следующих HTML-код

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Jade</title>
    <script type="text/javascript">
      if (foo) {
        bar()
      }
    </script>
  </head>
  <body>
    <h1>Jade - node template engine</h1>
    <div id="container">
      <p>You are amazing</p>
    </div>
  </body>
</html>
```

Этот пример иллюстрирует основной способ передачи параметров и использования в тексте шаблона.

3.2.3 Загрузка *.obj-объектов

Одной из главных возможностей приложения является динамическая загрузка трехмерных объектов, заданных в открытом формате OBJ. К сожалению, фреймворк “Three.js” не позволяет осуществлять динамическую загрузку непосредственно OBJ-моделей и требует предварительной конвертации файлов в специального вида JSON-объекты. В стандартную поставку библиотеки входит скрипт convert_obj_three.py, написанный на языке Python, созданный специально для этих целей. К сожалению, язык скрипта не позволяет запустить его в браузере на стороне пользователя, а объем кода в 48 килобайт делают задачу портирования достаточно трудоемкой, возникает вероятность многочисленных ошибок.

В качестве решения было преложено ввести в интерфейс приложения инспектор объектов, который бы отображал список уже сконвертированных объектов, доступных для динамической загрузки в редактор. Пользователь имеет возможность удалить любой из объектов инспектора, а так же добавить новые объекты с помощью специальной формы. При добавления нового объекта пользователем происходит следующая цепочка событий

1. Пользователь через специальную форму выбирает расположенные локально на его компьютере OBJ-объекты, которые он хочет увидеть в браузере

2. Выбранные файлы загружаются на сервер, где с помощью скрипта из пакета поставки фреймворка “Three.js” преобразуются в необходимый формат и сохраняются

3. При получении успешного ответа сервера при загрузке файлов клиент автоматически отправляет запрос на получение нового списка объектов для отображения в инспекторе объектов. Пользователь может загрузить любой объект из списка

Ввиду того, что типичный сценарий использования приложения не подразумевает наличия большого количества преобразованных объектов одновременно, было решено организовать их хранилище на основе файловой системы (ФС). Таким образом все преобразованные объекты хранятся на сервере в специальной директории ./objects, а основные операции были организованы следующим образом

- ADD. Эта команда должна добавить новый объект в хранилище. В случае ФС создается новая директория с именем таким же, как имя объекта, и все файлы, относящиеся к объекту, сохраняются в ней

- LIST. Эта команда должна вернуть список имен всех объектов. В случае ФС она возвращает список имен поддиректорий, находящихся в директории ./objects

- REMOVE. Эта команда должна удалить объект по его имени. В случае ФС она удаляет директорию с соответствующим именем.

Для вызовов внешних процедур изнутри “Node.js” использовался метод exec модуля “child_process”. Для каждого из действий, вовлекающих использование внешних вызовов, была разработана соответствующая оболочка на языке JavaScript.

4 Алгоритм деформации

4.1 Описание алгоритма

Базовым средством изменения геометрии фигуры является инструмент “деформация”. С помощью этого инструмента пользователь может скорректировать геометрию загруженного объекта.

Использование инструмента происходит по следующей схеме

1. Пользователь активирует инструмент в меню инструментов
2. После наведения мышки на модель, появляется сфера, демонстрирующая изменения, которые произойдут с геометрией после применения инструмента на этом участке модели.
3. После одиночного щелчка мыши вершины геометрии, которые попали во внутрь сферы, будут спроецированы на ее переднюю поверхность

Инструмент реализован в рамках поведенческого шаблона Strategy. Интересной особенностью этого инструмента является то, что, ввиду необходимости добавлять схематичную сферу на сцену для демонстрации последствий деформации, инструмент вынужден вмешиваться в процесс отображения сцены, при этом не добавляя дополнительных логических связей между собой и другими модулями приложения. Это было достигнуто следующим образом.

При создании инструмента (а инструменты создаются непосредственно перед моментом своей установки) в личном пространстве имен сохраняется функция, выполняющую отображение сцены.

```
1 function ModifyingStrategy(mobject) {
2     this.managedObject = mobject;
3     this.formerRender = mobject.render;
4 }
```

Теперь, в связки с возможностью передавать функциям любой объект, выступающий в роли “this”, у нас есть возможность полностью подменить функцию отображения сцены

```
ModifyingStrategy.prototype.setUp = function() {
    var gthis = this;
    this.managedObject.render = function() {
        // Adding sphere is needed
        ....
        // Calling original rendering method
        gthis.formerRender.call(this);
    }
}
```

В класс-ориентированных языках программирования подобный прием мог быть реализован с помощью наследования и переопределения виртуальных функций, с последующим вызовом супер-реализации этой функции.

Для того, чтобы задать правильные координаты сферы в сцене, необходимо решить задачу нахождения координат точки в трехмерном пространстве сцены, на которую указывает пользователь с помощью мышки. Задача взаимодействия мышкой с трехмерной сценой является типовой, и в фреймворке “Three.js” есть стандартное средство для ее решения. Таким средством является объект Ray, который представляет собой луч, выходящий из точки, и который может быть пересечен с объектами сцены. Результатом пересечения будет массив точек пересечения, каждая из которых, однако, задана в локальной координатной системе пересекаемого объекта. Так как для создания схематичной сферы требуются координаты сцены, то полученную точку пересечения необходимо перевести из локального пространства объекта в пространство сцены.

К сожалению, эта задача не решена на уровне фреймворка. Поэтому для её решения можно воспользоваться матрицей $matrixWorld$, которая есть

у каждого объекта и которая описывает линейный оператор, переводящий точки пространства сцены в точки пространства объекта. Благодаря тому, что оператор является биективным, то для него однозначно определен обратный, который определяется матрицей $matrixWorld^{-1}$. Таким образом координаты точки пересечения в пространстве сцены могут быть получены следующим образом

```
1 var v = intersection.point.clone();
2 var m = new THREE.Matrix4().getInverse(mesh.matrixWorld);
3 m.multiplyVector3(v);
```

Теперь рассмотрим непосредственно работу самого алгоритма трансформации. После того, как пользователь совершил нажатие мышью на какой-либо участок геометрии, все точки, попавшие внутрь сферы, проецируются на переднюю поверхность сферы. Определим работу алгоритма более конкретно

А алгоритма есть два параметра: радиус сферы трансформации R, а так же коэффициент заглубления K. Их назначение будет пояснено ниже

1. Определим точку p модели, на которую кликнул пользователь.

2. Определим внешнюю нормированную нормаль $\vec{n}$ к модели в этой точке. Внешняя нормаль определяет направление проецирования точек геометрии

3. Рассмотрим вектор $\vec{v} = -R \cdot \vec{n}$, направленный в противоположную сторону относительно вектора деформации и по модулю равный R. Центр сферы S принимается равным $= p + K \cdot \vec{v}$ а за радиус берется R

4. Для каждой вершины u геометрии, попадающей внутрь сферы S, проверим, лежит ли она в первой половине сферы. Для этого

достаточно посчитать скалярное произведение радиус-векторов

$$(\vec{u} - \vec{c}) \cdot \vec{n} = |\vec{u} - \vec{c}| \cdot |\vec{n}| \cdot cos(\vec{u} \wedge \vec{n})$$

Скалярное произведение будет меньше нуля в том случае, если точка лежит во второй половине сферы, удаленной от зрителя. В таком случае эту точку проецировать не надо.

5. Для каждой точки, подлежащей трансформации, происходит проецирование на поверхность сферы по направлению, заданному вектором деформации $\vec{n}$

Операция проецирования эквивалентна решению следующей задачи.

Рассмотрим прямую, заданную радиус-вектором $\vec{s}$, задающим точку, которую надо спроецировать, и вектором направления проекции $\vec{d}$. Тогда точки этой прямой выражаются следующим уравнением

$$\vec{p}_{line} = \vec{s} + k \cdot \vec{d}$$

В свою очередь, точки сферы с центром в (x_c, y_c, z_c) в общем виде задаются следующим уравнением

$$(x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2 = R^2$$

Подставив первое уравнение во второе, мы найдем один или два корня. В случае нахождения одного корня точка уже является спроецированной на сферу.

В случае нахождения двух корней k_1, k_2 мы получаем две точки пересечения сферы и прямой, и при этом если изначальная точка $\vec{s}$ находилась внутри сферы, то корни будут разных знаков. Так как нас интересует проекция на переднюю половину сферы, то проекцией будет точка, соответствующая положительному корню.

Рис. 1. Работа алгоритма для утолщения губ (результат справа)

4.2 Тесселляция

Алгоритм деформации, передвигая вершины, неизбежно растягивает одни ребра и уменьшает другие. В итоге локальная детализация объекта становится неприемлимо низполигональной. Для устранения этого недостатка было решено использовать алгоритм тесселляции модели.

Тесселляция - прием, с помощью которого можно увеличить количество многоугольников в полигональной трёхмерной модели. В процессе

Рис. 2. Отверстия в модели при тесселляции

тесселляции объектов, составленных целиком из треугольных полигонов [6], необходимо добавлять новые вершины. В этом крылась первая проблема: объекты движка “Three.js”, загруженные в буферы WebGL, не могут изменить количество вершин, содержащихся в их геометрии. Так как объект загружается в буферы WebGL при первом отображении сцены, то для обхода этой проблемы нужно было либо тесселлировать объекты только один раз до необходимого размера ребра непосредственно после загрузки и до отображения сцены, либо динамически тесселлировать объект, удалять его со сцены и пересоздавать. Преимущество первого

[6]В случае использования четырехугольных полигонов добавление новых вершин необязательно, достаточно поделить диагональю каждый полигон пополам

метода над вторым в том, что он не затрагивает производительность приложения и несомненно является более простым в реализации, однако он не позволяет добиться динамической тесселляции, которая становится доступной при использовании второго метода. При разработке планировалось опробовать алгоритм тесселляции с использованием первого метода, а потом перейти на динамическую тесселляцию при достижении хороших результатов на первом этапе.

В качестве реализации алгоритма тесселляции была взята версия алгоритма, входящего в стандартную поставку фреймворка “Three.js”. Однако при предварительном тесселлировании объектов и их дальнейшем отображении была выявлена серия проблем.

- При тесселлировании простейших геометрических фигур с четырехугольными гранями сбиваются цвета граней
- При тесселлировании загруженных *.OBJ-объектов в модели появляются отверстия в местах стыках граней

В то время как четырехугольные грани встречаются довольно редко при задании геометрии трехмерных объектов, то второй недостаток существенен при работе с моделями. Самостоятельное изучение кода тесселляции никаких результатов в исправлении недостатка алгоритма не дало. Отчет об ошибке вместе с примерами кода был отправлен разработчикам, а изменения в коде приложения, связанные с тесселляцией, были вынужденно откачены назад.

5 Тестирование системы

Для того, чтобы установить перспективы использования разработанного web-сервиса, была проведена серия экспериментов. Эксперименты замеряли одну из следующих величин:

1. FPS (Frames per second) - количество итераций отображения трехмерной сцены в секунду. Считается, что значения в 30FPS и более являются достаточно комфортными для человеческого восприятия.

2. Время отклика сервера при использовании серверных вычислений. Вычисляется как временной промежуток между обновленным результатом на клиенте по выполнению серверных вычислений и непосредственной инициацией процесса на клиенте.

Каждый эксперимент имел целью исследование зависимостей между количеством полигонов в модели и тем или иным изучаемым параметром. Исследования проводились на пяти различных моделях

Имя модели	couch1	Jane_solid_obj	ladybird	Woman-head	Mini-cooper
Количество полигонов	3098	12334	23496	114344	254714

Таблица 1. Тестовое множество

Эксперименты проводились на тестовом стенде следующей конфигурации:

- Браузер: Google Chrome v.19.0
- Процессор: Intel Core i5, 1.7ГГц, кэш-память третьего уровня в 3 Мб
- Память: 4 ГБ DDR3 1333МГц
- Графический процессор: Intel HD Graphics 3000 с памятью 384

5.1 Тест: просмотр модели

Эксперимент проводился с целью определения возможностей статического рендеринга трехмерных моделей средствами технологии WebGL и фреймворка “Three.js”. В рамках исследования был проведен ряд тестов, каждый из которых состоит из следующей последовательности шагов:

1. В web-приложение загружается трехмерная модель в формате *.OBJ
2. С загруженной моделью совершают ряд операций поворота и масштабирования
3. Считывание среднего значения FPS, полученного за время работы с моделью

Количество полигонов	3098	12334	23496	114344	254714
FPS	60	60	60	60	59

Таблица 2. Зависимость FPS от количества полигонов модели при просмотре

Сводные результаты тестирования представлены в таблице 2. Из результатов эксперимента видно, что технология WebGL вместе с фреймворком “Three.js” отлично справляется с рендерингом трехмерных моделей.

5.2 Тест: локальное изменение модели

Эксперимент проводился с целью определения возможностей динамического изменения трехмерных моделей на стороне клиента.

Каждый из тестов, проведенный в рамках исследований, состоит из следующей последовательности шагов:

1. В web-приложение загружается трехмерная модель в формате *.OBJ
2. С загруженной моделью с помощью инструмента для локального изменения геометрии совершается ряд преобразований
3. Считывание среднего значения FPS, полученного за время работы с моделью

Количество полигонов	3098	12334	23496	114344	254714
FPS	60	41	22	6	2

Таблица 3. Зависимость FPS от количества полигонов модели при применении алгоритма “Деформация”

Результаты экспериментов приведены в таблице 3.

Эксперимент показала явную регрессию производительности при увеличении количества полигонов в модели. Это связано в первую очередь с тем, что инструмент локального изменения геометрии добавляет в процесс рендеринга дополнительную операцию пересечения луча со сценой (операция необходима для того, чтобы расположить схематичную сферу трансформации на модели).

Операция пересечения луча с объектом, в свою очередь, написана на языке JavaScript и работает за $O(N)$, где N - количество полигонов в объекте. Именно такую зависимость FPS от количества полигонов можно наблюдать в результате эксперимента.

Можно перечислить следующие пути для решения этой проблемы, которые могут быть реализованы в дальнейшем

1. Путем изменения схемы взаимодействия пользователя с объектом можно исключить необходимость в отрисовке схематичной сферы деформации

2. Заметим, что за единицу времени сфера трансформации может сдвинуться на не очень большое расстояние. Такой подход дает возможность для попыток реализации множественных инкрементальных алгоритмов, которые делают первоначальное размещение сферы за время $O(N)$, однако после этого незначительные изменения в положении сферы могут быть пересчитаны за гораздо меньшее количество операций.

5.3 Выводы

Несмотря на хорошие результаты при просмотре моделей, эксперименты демонстрируют неприменимость текущей реализации для высоко-полигональных моделей в связи со значительным падением полигонов при использовании инструмента “Деформация”. Учитывая асимптотически хорошее время алгоритма $O(N)$ и проблемы, которые испытвает этот алгоритм, можно сделать следующие выводы

1. Ввиду того, что многие алгоритмы включают в себя проход вершин модели, подобные проблемы производительности будут встречаться при использовании большинства алгоритмов для динамической обработки моделей на JavaScript

2. Ввиду зависимости времени работы алгоритма от конфигурации клиентской машины, можно ожидать, что на более слабых компьютерах использование приложения будет затруднено

Ввиду вышеперечисленных причин встала задача по организации серверных, или удаленных, вычислений геометрии.

6 Серверные вычисления

6.1 Формулировка задачи

В связи с сильно ограниченными вычислительными возможностями на стороне клиента, было решено отказаться от разработки алгоритмов для обработки геометрии, выполняемые в браузере. Таким образом встала задача использования вычислительно сложных алгоритмов для работы с трехмерной геометрией при условии невозможности реализации этих алгоритмов на языке JavaScript для исполнения на стороне клиента.

Для решения этой проблемы было решено использовать удаленные серверные вычисления. При таком подходе необходимо ввести в архитектуру приложения еще одну сущность - сервер высокопроизводительных вычислений (СВВ). Для начала опишем роль, которую играет этот объект при выполнении серверных вычислений.

При таком подходе для обсчета геометрии с помощью какого-либо алгоритма будут сделаны следующие шаги.

1. Данные о текущем состоянии геометрии фигуры, отображенной в браузере, считываются и передаются на сервер
2. Сервер сериализует эти данные и передает с помощью какого-либо транспорта в СВВ
3. СВВ десериализует эти данные и передает их на вход алгоритму
4. СВВ получает результат работы алгоритма, сериализует их и с помощью того же транспорта передает на сервер
5. Сервер десериализует данные и отправляет их клиенту

Таким образом структурно СВВ является некоторой оболочкой над алгоритмом (или группой алгоритмов), осуществляющий операции

сериализации и десериализации данных, а так же передачи их по назначению с помощью какого-либо транспортного уровня.

Важно отметить, что данная технология подразумевает наличие модуля сериализации и десериализации данных как на стороне сервера, так и на стороне СВВ, причем эти модули должны оперировать одним и тем же форматом данных, но быть написанными на разных языках программирования (в нашем случае, с одной стороны - на JavaScript, с другой стороны на C++).

Задача реализации СВВ, а так же модуля сериализации и десериализации данных на JavaScript была решена с помощью автоматической генерации кода по заданной модели. Для этого существует две широко применяемые технологии.

6.2 Protocol Buffers и Apache Thrift

Для создания приложений с использованием нескольких языков встает проблема взаимодействия модулей. Традиционным решением является абстракция удаленного вызова процедур, когда один модуль по средствам заданного протокола кодирует данные и по средствам транспорта передает их процедуре другого модуля на обработку. Ответ этой процедуры кодируется с использованием того же протокола и отправляется вызывавшему.

В описанном выше взаимодействии важную роль играют процессы сериализации и десериализации объектов, а так же поддержание согласованности программного кода для выполнения этих задач на каждом из языков программирования. Для решения этих задач широко применяется одна из следующих технологий:

- Google Protocol Buffers
- Apache Thrift

Технология Apache Thrift является последователем Google Protocol Buffers и разработана в свете опыта использования последней. Сравнение двух технологий удобно отобразить в виде таблицы (см. таблицу 4).

	Apache Thrift	Google Protocol Buffers
Форматы	Binary, JSON	Binary
Поддержка “Node.js”	Включена в официальную поставку	В виде сторонних библиотек
Транспортный уровень	Включен в официальную поставку	В виде сторонних библиотек
Документация	Посредственная	Очень хорошая

Таблица 4. Сравнение технологий Apache Thrift и Google Protocol Buffers

Стоит отметить, что быстродействие обоих фреймворков примерно одинаково, поэтому по совокупности признаков, описанных в таблице, выбор был сделан в пользу Apache Thrift как наиболее подходящей для поставленных целей и используемых технологий.

6.3 Реализация СВВ средствами Apache Thrift

Apache Thrift[5] - технология, разработанная для создания приложений на разных языках программирования. Для организации такого взаимодействия между компонентами, написанными на двух языках, требуется использовать модули сериализации и десериализации, а так же модуль, ответственный за транспортный уровень передачи данных, для каждого из языков, участвующих во взаимодействии. Такие модули создаются процессом генерации кода, который происходит на основе описанной на специальном языке Thrift модели.

Thrift модель позволяет описывать две сущности: данные, которыми обмениваются стороны взаимодействия, и сигнатуры процедур, которые одна сторона предоставляет для удаленного вызова. Одна из сторон взаимодействия является клиентской и инициирует взаимодействие путем

вызова процедур, предоставленных серверной стороной. Таким образом на основании этих данных Apache Thrift генерирует код описания данных, код, выполняющий сериализацию и десериализацию данных, а так же их отправку и прием с помощью транспортного уровня. Кроме этого генератор создает файл “MainService_server.skeleton.cpp”, создержащий наброски серверной части взаимодействия с заглушками вместо процедур удаленного вызова.

```
1 struct Point {
2     1: double x,
3     2: double y
4 }
5
6 service ScaleService {
7     Point scalePoint(1:Point point, 2:int scalar)
8 }
```

Listing 1. Пример описания модели и сервиса для технологии Thrift

В нашем случае в качестве сервисов выступают конкретные алгоритмы, выполняющие удаленные вычисления. Так как эти алгоритмы могут добавляться и исчезать в процессе развития проекта, то необходимо изменять соответствующим образом описание сервиса в файле описания thrift, а затем перегенерировать все файлы ввиду внесенных изменений. После генерации файлов необходимо будет обновить файл “server.cpp”, добавив соответствующие директивы “include” и исправив заглушки на вызов соответствующих алгоритмов.

Из всего вышесказанного следует, что для того, чтобы добавить новый алгоритм в систему, потребуется знание формата файлов Thrift, а так же четкое понимание внутренней структуры сервиса, что значительно усложняет процесс расширения функционала всего приложения.

Для того, чтобы облегчить процесс добавления новых алгоритмов, был написан скрипт “generate.sh” для оболочки “bash”, занимающийся автоматизацией этих операций. При использовании этого скрипта, процесс

добавления новых алгоритмов для обработки геометрии сводится к следующей цепочки действий

1. Файл реализации *.cpp и файл заголовка *.h помещаются в директорию ./thrift/algo. Файл заголовка должен содержать только описания функций, обрабатывающих геометрию объекта.

2. Запускается команда “bash generate.sh” из директории ./thrift, которая компилирует серверную часть Thrift

После этого сервер “./thrift/server” готов к использованию и может быть запущен для предоставления сервиса по удаленному использованию процедур. Кроме этого, на стороне серверной части приложения “Node.js” список функций, доступных для удаленного вызова, автоматически обновился.

Скрипт “generate.sh” совершает следующие действия

1. Удаляет директории gen-nodejs и gen-cpp, очищая таким образом данные, полученные при предыдущих запусках скрипта.

2. Сканирует директорию, содержащую все алгоритмы пользователя, и, анализируя все “*.h” файлы, найденные в ней, создает соответствующий файл описания сервиса для технологии Thrift

3. С помощью технологии Thrift и на основе файла описания сервиса и данных, генерирует код на языке JavaScript для платформы “Node.js”

4. Создает файл “remoteComputing.js”, содержащий функции, соответствующие именам найденных функций пользователя. Таким образом на стороне серверной части вызов этих функций ничем не будет отличаться от вызова любой другой функции из фреймворка “Node.js”

5. С помощью технологии Thrift и на основе файла описания сервиса и данных, генерирует код на языке C++.

6. Исправляет файл “server.cpp”, сгенерированный технологией Thrift, убирая в нем заглушки и заменяя их соответствующим вызовом функций пользователя

7. Исправляет использующийся транспорт сервера на соответствующий транспорт, используемый в js-части приложения

8. Компилирует файл “server.cpp”, получая готовое серверное приложение.

Стоит отдельно отметить пункт 2, включающий в себя анализ “*.h” файлов пользователя. Для этой цели было решено использовать утилиту “ctags”. Благодаря ей список функций, определенных среди всех файлов с алгоритмами, можно получить следующим образом

```
1    ctags -x --c-kinds=p *.h | cut -f1 -d' '
```

6.4 Десериализация вещественного типа в Node-Thrift

В процессе интеграции технологии Apache Thrift в приложение был использован достаточно молодой модуль “Node-Thrift” для платформы “Node.js”, включающий в себя необходимый код для передачи данных с помощью транспорта, а так же сериализацию и десериализацию простейших типов данных.

К сожалению, после первых экспериментов было установлено, что модуль работает некорректно. В качестве подтверждения был написан следующий echo-сервис, который работал ошибочно в условиях взаимодействия компонентов на языках “Node.js” и C++.

```
1 service EchoService {
2     double echo(1:double msg);
```

```
3 }
```

В результате генерации кода на основании этого файла описания данных, клиентское приложение всегда получало 0 в качестве ответа на удаленный вызов процедуры. Однако проблемы не возникало при использовании любых других типов данных, отличных от double. Кроме того, проблемы не возникало при использовании языка Ruby для генерации клиентской части. Из этого следовало, что ошибка кроется в библиотеке “Node-Thrift” для платформы “Node.js”. Соответствующий отчет об ошибке был отправлен разработчику.

Ввиду необходимости продолжать разработку, была предпринята попытка устранить ошибку в библиотеке Node-Thrift самостоятельно. После изучения исходных кодов модуля было установлена, что ошибка возникает на этапе десериализации данных и связана с неправильным декодированием 8-байтного вещественного типа. После обнаружения неисправность была устранена, а соответствующее исправление[6], устраняющее проблему, было выслано разработчику модуля.

7 Тестирование серверных вычислений

Для того, чтобы оценить возможности использования серверных вычислений, были сделаны эксперименты по их выполнению с разными моделями. В качестве алгоритма, исполняемого на сервере, выступал однопроходный алгоритм масштабирования.

```
#include "scale_x2.h"
using namespace std;
using namespace threejs;

void scale_x2(Geometry& _return, const Geometry& geom) {
    vector<Vertex> vv = geom.vertices;
    for(vector<Vertex>::iterator it = vv.begin(); it != vv.end(); ++it) {
        Vertex v = *it;
        v.x *= 2;
        v.y *= 2;
        v.z *= 2;

        _return.vertices.push_back(v);
    }
}
```

Результатом каждого эксперимента была задержка - количество времени, прошедшее с начала отправления зарпоса в браузере, до получения и применения ответа. Каждый эксперимент проводился при условии соединения клиентской машины с сетью Интернет как по локальной сети с помощью WiFi, так и через протокол передачи данных 3G, что обусловленно нацеленностью проекта на мобильные технологии.

Сервер, обрабатывающий запросы, географически располагался в Германии, и имеет следующие параметры

- Процессор: 3073МГц
- Память: 500Мб

- Канал интернет: обеспечивает скорость более 2 Мегабайт/с на загрузку, и столько же на отдачу

В результате проведения экспериментов были получены результаты, сведенные в единую таблицу 5

Количество полигонов	3098	12334	23496	114344	254714
LAN задержка, мс	397	826	2394	4103	17956
3G задержка, мс	3344	8402	34018	-	-
Взаимодействие node-thrift	43.5	148.5	672	1278.5	5448.5

Таблица 5. Задержка серверных вычислений

При анализе результатов эксперимента оказалось, что несмотря на достаточно хорошие результаты работы по LAN, использование приложения по 3G не представляется возможным.

8 Зеркалирование

Для ускорения выполнения СВВ была применена сильная оптимизация, позволившая значительно сократить расходы на передачу данных по сети. Оптимизация основывается на следующих наблюдениях:

- Все деформации геометрии, происходящие на стороне клиента, являются детерменированными и полностью задаются входными данными и парамтрами.
- На стороне сервера изначальная модель уже есть в формате, распазнаваемым фреймворком Thrift.js

Исходя из этого можно применить следующий прием: вместо того, чтобы при необходимости выполнить серверные вычисления пересылать всю геометрию на веб-сервер, можно переслать только историю примененных преобразований и их параметров. После этого сервер эмулирует эти преобразования одно за одним и результат эмуляции передает СВВ в качестве входных аргументов.

Для того, чтобы организовать такое поведение, необходимо решить следующие технические проблемы.

8.1 Начальное чтение модели

Изначально серверу необходимо прочитать и распознать модель, хранящуюся в закрытом формате трехмерных файлов фреймворка Three.js. Ручной разбор этого файла представляется довольно сложным занятием, в первую очередь по причине необходимости постоянной дальнейшей поддержки. Вместо этого было решено интерпретировать код фреймворка Three.js на стороне сервера и воспользоваться той его частью, которая ответственная за разбор и представление собственного формата.

```
var fs = require('fs');
var vm = require('vm');

var t = fs.readFileSync('./public/js-libs/Three.js', 'utf8');

self = {};
window = {};
vm.runInThisContext(t);
```

После этого необходимый файл модели можно легко прочитать с использованием следующей функции

```
function loadModel(path, callback) {

    var l = new THREE.JSONLoader();

    fs.readFile(path, 'utf8', function(err, j) {
        var jj = JSON.parse(j);

        l.createModel(jj, callback);
    });
}
```

8.2 Сессии пользователя

Ввиду того, что один сервер теперь хранит у себя состояние большого количества пользователей, возникла необходимость различать различных клиентов. Для этого был использовн механизм сессий, который для каждого пользователя сервиса присваивает персональный номер: SessionID. Благодаря этому SessionID, который передается с каждым запросом пользователя на веб-сервис, у сервера появляется возможность поддерживать словарь, сопоставляющий для каждого пользователя его состояние модели.

Поддержка сессий в приложении была реализована с помощью средств фреймворка Connect.js

9 Тестирование

Экспериментальная ветка проекта, реализующая указанную выше асимптотику, была протестирована на тех же тестах, на которых было проведено изначальное тестирование.

В результате проведения экспериментов были получены результаты, сведенные в единую таблицу 6

Количество полигонов	3098	12334	23496	114344	254714
LAN задержка, мс	213	513.5	1941.5	3718.5	12572
3G задержка, мс	603	1571	3871	6003	-
Взаимодействие node-thrift	33.5	143	654	1230	-

Таблица 6. Задержка оптимизированных серверных вычислений

Таким образом эта оптимизация, почти не изменяя расклад сил на тестах с использованием LAN, приносит десятикратный прирост в скорости на тесте большого размера.

10 Результат

Результатом дипломной работы стал графический редактор трехмерных объектов. Сервис предоставляет функционал по обработке трехмерных моделей, а именно:

- Загрузку трехмерных моделей в формате *.OBJ
- Просмотр трехмерных моделей с функциями поворота, масштабирования и режимом автоматического вращения
- Локальное изменение трехмерных моделей с помощью специального инструмента
- Изменение трехмерных моделей за счет алгоритмов, реализованных на стороне сервера на языке C++

Разработанный сервис предоставляет следующие возможности модульного расширения:

1. Возможность добавления инструментов для взаимодействия с трехмерным объектом
2. Возможность добавления алгоритмов на языке C++ для серверной обработки геометрии

На рисунке 3 представлен внешний вид клиентской части разработанного клиент-серверного приложения

1 Информация Здесь предоставляется информация о количестве полигонов и радиусе описанной окружности модели.

2 Инструменты Набор инструментов для взаимодействия с моделью. Для выбора возможно использование быстрых клавиш

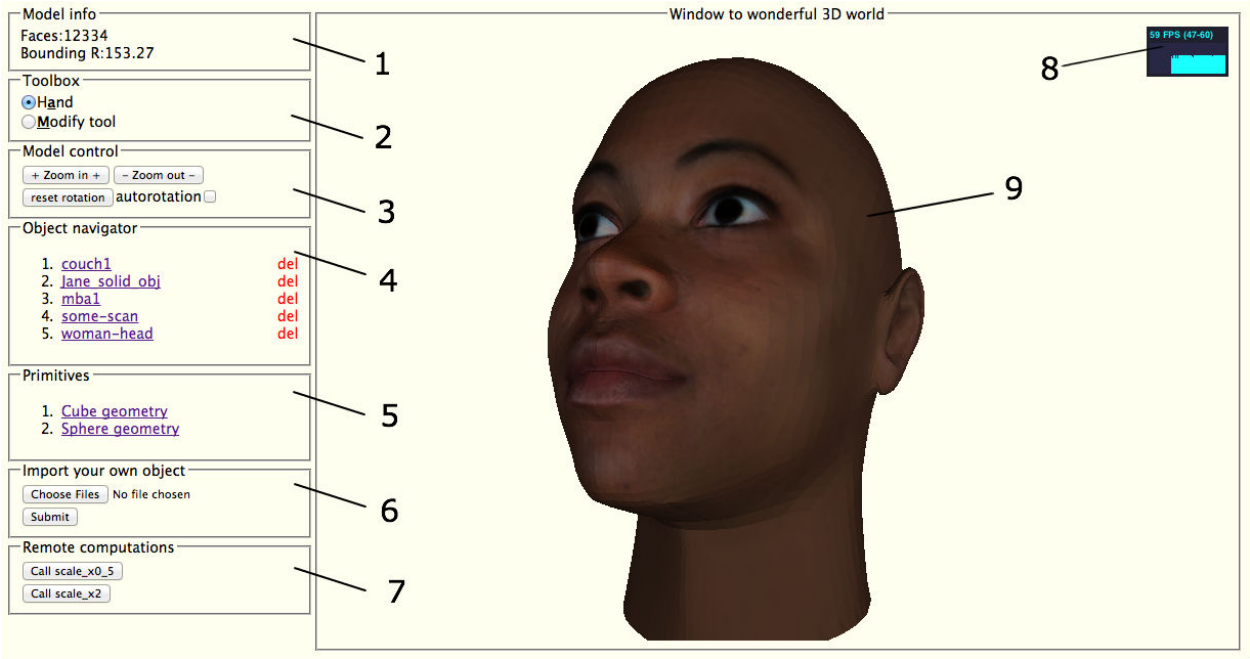

Рис. 3. Интерфейс приложения

3 Управление моделью Управление объектом: приближение, отдаление, автоматическое вращение объекта

4 Инспектор объектов База преобразованных OBJ-файлов, доступных для динамической загрузки в редактор. Нажатие ссылки “del” рядом с каждым объектом приведет к его удалению из базы.

5 Графические примитивы Загрузка графических примитивов в поле редактора

6 Импорт OBJ-объектов Форма для добавления OBJ-объектов в инспектор объектов.

7 Серверные вычисления Список доступных функций преобразования геометрии, заданных и выполняемых на стороне сервера

8 FPS гистограмма Статистика по количеству FPS

9 3D сцена Сцена с трехмерным объектом

Список литературы

[1] Спецификация WebGL
http://www.khronos.org/webgl.

[2] Официальный сайт Node.js
http://nodejs.org/

[3] Официальный сайт Express.js
http://expressjs.com/

[4] Официальный репозиторий Three.js
https://github.com/mrdoob/three.js/

[5] Официальный сайт Apache Thrift
http://thrift.apache.org/

[6] Ветка проекта node-thrift с исправленной ошибкой
https://github.com/aslushnikov/node-thrift

[7] Официальный сайт Jade
http://jade-lang.com/

[8] Официальный сайт JQuery
http://jquery.com/

[9] Описание проекта JQuery Form
http://jquery.malsup.com/form/

[10] Официальный сайт Underscore
http://documentcloud.github.com/underscore/

[11] Введение в WebGL. Шейдеры.
http://learningwebgl.com/blog/?p=28

[12] Шаблон проектирования Observer

http://en.wikipedia.org/wiki/Observer_pattern

[13] Шаблон проектирования Strategy

http://en.wikipedia.org/wiki/Strategy_pattern

[14] Методы HTTP 1.0

http://www.w3.org/Protocols/HTTP/1.0/draft-ietf-http-spec.html#Methods