Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего профессионального образования
**«Уральский федеральный университет
имени первого Президента России Б.Н. Ельцина»**

Институт математики и компьютерных наук

Кафедра высокопроизводительных компьютерных технологий

Специализированные жестовые интерфейсы для системы научной визуализации

«Допущен к защите» Зав.кафедрой Созыкин А.В.	Квалификационная работа на соискание степени магистра наук по направлению Компьютерные науки Магистерская программа «Системное программирование» студента гр. МКМ-240902 Пестовой Марины Сергеевны
_____ «___»_____2016 г.	Научный руководитель Авербух Владимир Лазаревич, кандидат технических наук

Екатеринбург
2016

Оглавление

Введение

В современном мире компьютеры становятся все мощнее, и одно из основных их использований – математические вычисления. В связи с тем, что сейчас можно решать более сложные задачи, их результаты не всегда получается обработать аналитически, поэтому разрабатываются все новые и новые представления абстрактных данных.

Одним из самых популярных способов представления является визуализация данных, которая отвечает за обеспечение анализа и интерпретацию получившихся результатов. Основным недостатком большинства современных сред визуализации является их ограниченность пространством монитора и невозможностью изучить данные более подробно.
С другой стороны, сейчас появляется все больше средств, которые позволяют просматривать трехмерные модели (очки дополненной реальности, очки виртуальной реальности), но в большинстве случаев их используют для развлекательных целей.

Мы решили исследовать вопрос визуализации математических данных, в частности сеток, в среде виртуальной реальности.
Целью этой диссертации является описание проблем, возникающих при работе с виртуальной реальностью, а так же, создание набора жестов для навигации в такой среде и их реализация.

В первой главе будут рассмотрены понятия «девайсных» и «естественных» интерфейсов, разница между ними и достоинства и недостатки каждого. Так же будет рассмотрено понятие «персонализированного» интерфейса.

Затем, во второй главе, мы опишем, какие есть проблемы, при взаимодействии с виртуальной реальностью, и предложим некоторые решения для их устранения.
После этого, перейдем к более практической части работы.

В третьей главе подробно описывается технология Leap Motion как с программной, так и с аппаратной стороны, на примере устройства The Leap.

В последней, четвертой главе, рассматривается разработанная нами система жестов, включая описание программной реализации.

Глава 1. «Девайсные» и «естественные» интерфейсы

1. «Девайсные» и «естественные» интерфейсы

Аппаратные средства современного человеко-компьютерного взаимодействия появились более 50 лет назад вместе с первыми дисплеями. Уже к середине 60-ых годов XX века сложился определенный набор устройств ввода информации - алфавитно-цифровая и функциональная клавиатура дисплеев, наборный диск, световое перо, осуществлявшее ввод в различных режимах, мышь (первоначальное название – bug - жук). Чуть позднее появились джойстики (joystick), трэкболлы (trackball), тач-скрины (touch-screen – сенсорные экраны). Такие интерфейсы, реализуемые за счет операций с какими-либо устройствами, можно назвать девайсными интерфейсами (Device Interfaces) в отличие от естественных пользовательских интерфейсов (Natural User Interfaces - NUI), которые и станут предметом наших дальнейших рассуждений.

Используется несколько определений естественных интерфейсов. В одних из них упор делается на то, что в рамках естественных интерфейсов пользовательские операции интуитивно понятны и основаны на естественном бытовом поведении. В других говорится о базирующемся на естественных элементах фактически незаметном интерфейсе (или становящимся таковым после его освоения пользователем).

Мы, говоря о естественных интерфейсах, будем иметь в виду интерфейсы, построенные на фиксации и распознавании какой-либо комбинации движений человека или активности его органов.

Выпишем те известные из современной практики разработок интерфейсы, которые могут рассматриваться как естественные. Упорядочим их сверху вниз, учитывая возможность комбинации в рамках одной реализации фиксации нескольких естественных активностей человека.

Brain-Computer Interfaces;

Интерфейсы на основе распознавания нервных импульсов;

Интерфейсы, основанные на распознавании речи или движения губ

Интерфейсы, основанные на распознавании мимики или перемещения взгляда (Eye Gaze или Eye Tracking);

Интерфейсы, основанные на фиксации движений (motion capture) всего тела человека или отдельных органов (головы, всей руки, кистей рук, пальцев, ног).

Сейчас, одним из самых популярных примеров «естественных» интерфейсов, пожалуй, являются разнообразные распознаватели речевых команд, которые повсеместно встраиваются в современные девайсы (смартфоны, планшеты…).

Обратим внимание на противоречие, содержащееся в описании класса интерфейсов NUI. Подразумевается естественный (natural) интерфейс с компьютером, то есть с заведомо искусственным объектом. В природе (nature) нет компьютеров, а у человека нет органов, которые можно непосредственно без какой-либо аппаратуры (хотя бы простой видеокамеры) связать с каким-нибудь входом вычислительной системы.

В каком-то смысле можно говорить, что устройствами (devices) в случае естественных интерфейсов становятся органы тела самого оператора. Опыт показывает, что пользоваться такими “устройствами” зачастую сложнее, чем обычными манипуляторами или кнопками. То есть, “естественные” интерфейсы могут стать более сложными и неудобными для пользователя по сравнению с “девайсными”, так как требуют напряжения человека для повторения и четкой фиксации движений (или их мысленных образов в случае Brain-Computer Interfaces).

В принципе интуитивно понятным интерфейс становится в случае, если его использование опирается на предыдущий опыт пользователя. В этом отношении интуитивно понятными являются, как раз, “девайсные” интерфейсы, которые уже на самом раннем этапе опирались на опыт работы пользователей с

радиоприемниками, проигрывателями и телевизорами. Но в нашем случае, при взаимодействии с виртуальной реальностью, «естественные» интерфейсы становятся более удобными. Когда человек надевает очки виртуальной реальности и погружается в тот мир, какие-либо девайсы ему начинают только мешать. Даже если взять простую игру, реализованную с помощью 3D очков, в которой управление идет, например, с клавиатуры, то сразу возникает проблема нахождения клавиатуры и необходимых кнопок после надевания очков. Дело в том, что наш мозг пытается обрабатывать то, что мы видим, и если мы видим вокруг, например, лес, то все наши движения будут корректироваться в соответствии с этим. При этом клавиатуру мы не видим, а только знаем, что она есть где-то перед нами, и происходит несоответствие того, что мы видим, и того, что должно быть ли есть на самом деле. Поэтому, при работе с виртуальной реальностью лучше будут «естественные» интерфейсы, особенно если они мы можем отслеживать их действия (отображение моделей рук в соответствии с тем, что мы делаем нашими руками).

Обратим внимание на важное различие между “девайсными” и “естественными” интерфейсами. Распознавание команды в первом случае происходит чрезвычайно просто – от устройства приходит его код и данные определенного формата. Аппаратное обеспечение естественных интерфейсов, как правило, резко сложнее кнопок, джойстиков, etc. Также естественные интерфейсы требуют серьезных усилий по распознаванию образов, основанному на различных, достаточно сложных алгоритмах. Правда, в последнее время стали доступны для широкого использования современные программно-аппаратные средства захвата движений и качественные видеокамеры или целые комплексы стереокамер.

В тоже время естественные интерфейсы в целом ряде случаев оказываются жизненно важными. В частности, они используются для обеспечения связью людей, потерявших способность двигаться и даже говорить.

Наши исследования связаны с жестовыми интерфейсами. Использование трехмерных жестовых интерфейсов в настоящее время весьма актуально для медицинских приложений. Такие интерфейсы могут использоваться в системах визуализации на базе больших экранов или в средах виртуальной реальности типа CAVE для манипуляции виртуальными объектами и для навигации в виртуальном пространстве.

1.2. Персонализированный интерфейс

В случае профессиональных интерфейсов цель деятельности пользователя предопределена заранее. Постановка задачи в целом диктует требования к интерфейсу. “Профессионал” также не может отказаться от использования интерфейса, так как его деятельность строго регламентирована. Проектировщик интерфейса должен изучить цели и особенности данной деятельности с тем, чтобы не исказить ее и не вносить в нее дополнительные сложности. Профессиональный интерфейс не должен предполагать деятельности, противоречащей или отвлекающей пользователя от основной задачи.

В “профессиональные” интерфейсы, по нашему мнению, не следует включать сложные настройки, и вообще всего того, что может в каком-либо смысле рассматриваться как программирование, так как программирование является самостоятельной деятельностью, дополнительной к основным обязанностям профессионала-исследователя. В этом плане необходимы лаконичные интерфейсы с минимальными требованиями к памяти и вниманию пользователя.

Как следствие, разрабатываемый интерфейс должен быть удобен для нашего конкретного пользователя. Такие интерфейсы называются «персонализированными».

Перед нами стояла задача создать систему для просмотра математических сеток в виртуальной реальности. Это позволит более подробно визуализировать и изучить их изнутри. В частности, нам было необходимо создать систему, для управления и взаимодействия с сетками. Для этой цели мы выбрали «естественные» интерфейсы, точнее жесты рук.

Эта задача появилась из – за того, что сетка состоит из множества объектов, и не всегда можно аналитически обнаружить все ее особенности и интересующие нас данные. Причем в последние годы, количество ячеек в сетках значительно возросло. При визуализации мы можем получить наглядное представление того, как это выглядит. Например, можно выделить цветами различные области, или каким-либо еще образом выделить ячейки, которые могут заинтересовать пользователя. Но в связи с тем, что сетки, чаще всего, являются объемными данными со множеством ячеек, то надо уметь в них передвигаться. Мы предлагаем систему навигации, связанную с жестами. Это позволит пользователю передвигаться внутри сетки, с возможностью более близкого рассмотрения интересующих его мест. Так же, есть возможность, добавления задания некоторых ключевых параметров пользователем, перед началом или во время работы.

Мы разрабатываем систему под вполне конкретного пользователя – математика занимающегося сетками. Это значит, что у нас есть конкретный объект, который надо визуализировать – сетка. Мы представляем сетку в виде множества точек, объединенных между собой в ячейки. При этом, каждая точка имеет свои свойства, которые можно узнать, нажав на нее.

Но при работе в виртуальном пространстве, выполнить эти действия становится не так просто. В следующих главах описываются основные проблемы и предлагаются способы их решения.

Глава 2. Общие проблемы при работе с виртуальным пространством.

2.1. Описание общих проблем взаимодействия в виртуальном пространстве

Одной из основных проблем взаимодействия с виртуальной реальностью является невозможность (или сложность) одновременного отслеживанием мозгом и виртуального пространства и реального. В связи с этим, возникает множество проблем как физического, так и психологического характера. Например, часто, людей, в первый раз надевающих очки виртуальной реальности, начинает укачивает или у них болит голова.

Так же, могут возникать проблемы, при взаимодействии с реальными объектами, во время нахождения в виртуальном мире. Дело в том, что виртуальная реальность конструирует полностью новый мир, не всегда похожий на наш. Например, там могут быть изменены элементарные физические законы, и как следствие, после нахождения в таком состоянии, наш мозг может не сразу адаптироваться обратно, к реальности.

Допустим, пользователь управлял своими действиями в виртуальном мире с помощью специальных девайсов, например, джойстиков. Тогда, после возвращения обратно, первое время у него будет возникать желание такого же управления реальным миром.

2.2. Описание общих проблем взаимодействия с виртуальными объектами

При взаимодействии с виртуальными объектами так же возникает множество проблем.

Одна из них – соответствие виртуального объекта – реальному. Например, у нас есть виртуальная комната с мебелью, по которой мы можем ходить. Но в то же время, у нас есть реальная комната, с реальной мебелью. И в результате,

обходя мебель из виртуальной комнаты (которую мы видим), можно врезаться в реальную мебель.

Так же, если у нас, например, есть какой-либо виртуальный объект, то мы хотим с ним как-нибудь взаимодействовать. Обычные, девайсные, интерфейсы здесь не очень удобны, т.к. они находятся как раз таки в реальном мире. Поэтому, надо использовать что-нибудь, что может относиться и будет видно в виртуальном пространстве.

На данный момент есть разнообразные специальные девайсы, например, перчатки с датчиками, которые надеваются на руки и фиксируют их положение. Или специальные джойстики, с помощью которых можно управлять виртуальной реальностью. Но это все требует дополнительных затрат и навыков, со стороны конечного пользователя. Поэтому, самыми естественными, для управления, в данном случае, будут как раз жесты. Мы все привыкли получать большую часть информации именно через жесты, трогая какой-либо предмет. В виртуальной реальности все примерно так же. Мы можем потрогать предмет, взять его, приблизить или отдалить… Но в связи с этим, часто может возникать проблема определения, какой же именно предмет мы хотим изучить.

Современные средства распознавания жестов не всегда обладают достаточной точностью и четкостью, что бы определить это наверняка. Именно поэтому, необходимо создать достаточно простую и понятную систему жестов, которая бы могла помочь в изучении объектов виртуальной реальности.

Глава 3. Leap Motion

3.1. Общие сведения

Рисунок

Leap Motion – это технология, основанная на захвате движений для человеко-компьютерного взаимодействия. Она разрабатывается специально для управления компьютером с помощью движений рук. Отличается хорошей точность (по заявкам производителей 0.01 мм). Основными разработчиками являются Michael Buckwald, Daviv Holz, Keith Mertens и Nick Nam.

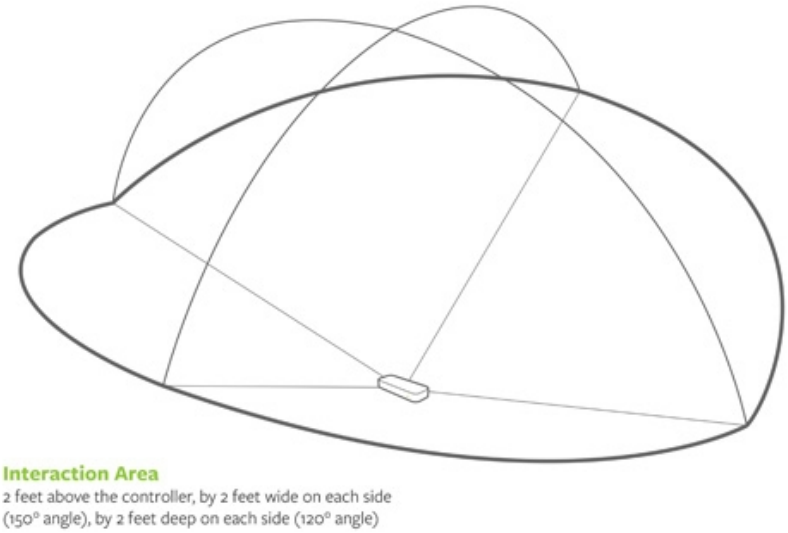
Минимальными требованиями, для корректной работы с Leap Motion являются: система Windows 7 – 10, Mac OS X 10.7, Linux, процессор AMD Phenom II или Intel Core i3, 2 ГБ RAM и порт USB 2.0. Так же в первый запуск необходимо подсоединение к интернету для скачивания необходимых драйверов и библиотек. В дальнейшем можно работать и без интернета.

Мы выбрали для разработки именно Leap Motion по нескольким причинам. Во первых, в отличие от Microsoft Kinect, у него есть открытая библиотека, которую можно бесплатно скачать с сайта и сразу же приступить к разработке. Во вторых, устройство Leap Motion обладает достаточно малыми размерами, что позволяет прикреплять его к очкам виртуальной реальности, благодаря чему, можно не беспокоиться о том, что пользователь выйдет из зоны действия устройства. В третьих, заявленные производителем характеристики значительно лучше остальных аналогов. Так же, свою роль сыграла небольшая, по сравнению с остальными аналогичными устройствами, цена. Ну и одной из главных причин выбора этого устройства является его совместимость (как в физическом, так и в программном смысле) с очками виртуальной реальность Oculus Rift, что позволяет не беспокоиться о дополнительных устройствах или знаниях для пользователя. Пример физического соединения см. Рисунок 2.

Рисунок

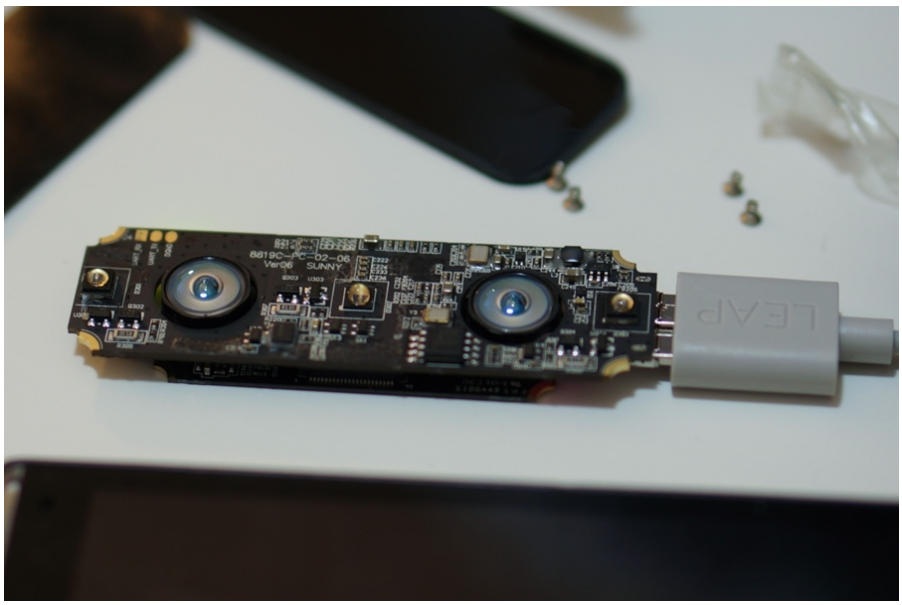
3.2. Аппаратное описание

The Leap - это маленькое устройство (см Рисунок 1), размером примерно 7.5 * 3 см, подключающееся по USB (в комплект входит провод USB 3.0 – USB, в самом устройстве порт USB 3.0) и обладающее 2мя инфракрасными камерами и 3мя инфракрасными светодиодами, которые позволяют отслеживать движения рук в пространственном кубе со стороной размером около 61 см. Примерная область видимости устройства составляет 150 градусов в ширину (по оси X) и 120 градусов в глубину (по оси Z) с каждой стороны (см. Рисунок 3). Максимальная высота отслеживания может изменяться в программных настройках до 25 см.



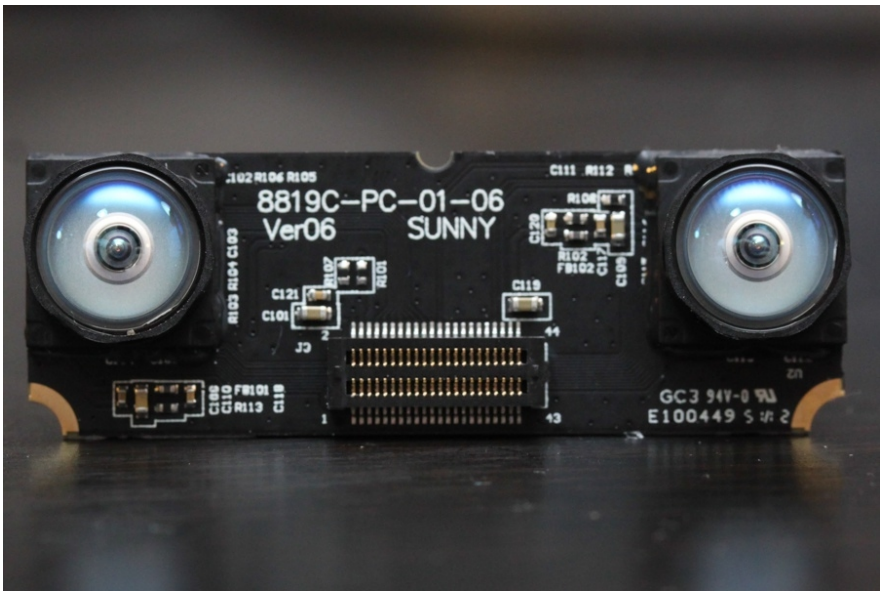
Рисунок

В разобранном виде это устройство выглядит как 2 камеры, прикрепленные на плату (см. Рисунок 4).



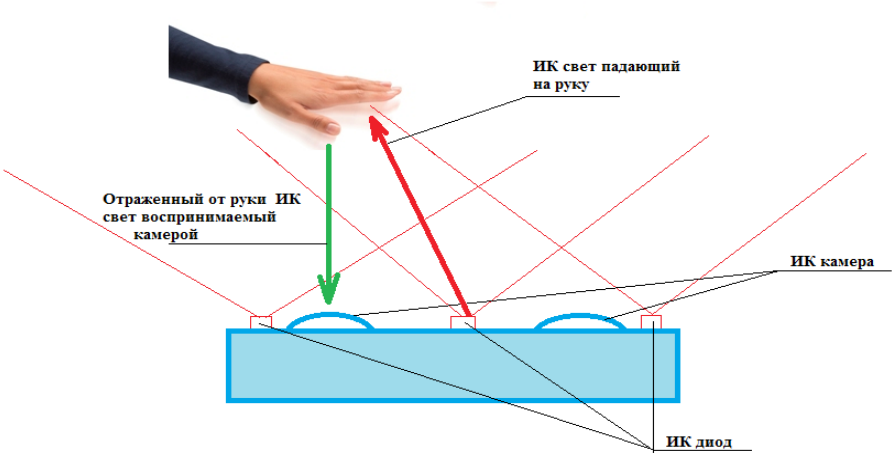
Рисунок

С одной стороны платы: разъем для соединения; 32-битный ARM9 контроллер от Cypress CYUSB301X, 200МГц, USB 3.0 и 2.0, OTG; MOSFET (МОП-транзистор) и два больших конденсатора. С другой стороны, на этой же плате находятся: флеш-память MXIC 25L3206E, SPI-интерфейс, 32 мегабита, в которой хранится прошивка контроллера; разведенные UART (Universal Asynchronous Receiver-Transmitter, универсальны асинхронный приемопередатчик); 3 инфракрасных светодиода, в обвязках, для управления ими; две камеры с объективами «рыбий глаз» и разъем (см. Рисунок 5). Камеры черно-белые с разрешением 640*480, но могут выдавать до 60ти кадров в секунду, это поток информации около 35Мбайт/сек. Именно поэтому поставили USB 3.0, так как для USB 2.0 такая скорость не достижима. В режиме USB 2.0 устройство тоже работает, но с замедленной реакцией.



Рисунок

Принцип работы устройства заключается в том, что инфракрасные диоды подсвечивают руку, а камеры делают захват и передают полученное изображение программному обработчику Leap Motion (см. Рисунок 6).



Рисунок

Инфракрасные камеры построены на принципе восприятия невидимого человеческим глазом инфракрасного излучения. Так же, в Leap Motion стоит специальный черный passing filter, который пропускает только инфракрасное излучение (черная поверхность над камерами). Поэтому камеры воспринимают только узкий инфракрасный спектр, который и получается при отображении света от диодов рукой пользователя. Так, на основе 2х изображений строится

карта глубин и создается массив точек, с которым и работают дальнейшие программы.

Может возникать проблема невозможности распознавания. Например, при прямых солнечных лучах или дополнительном инфракрасном излучении может быть невозможно поймать отраженные лучи.

Скорость работы устройства и распознавания рук во многом зависит от мощности компьютера, на котором обрабатываются данные, полученные с камер.

Одним из главных недостатков является сложность распознавания жестов, когда используются обе руки рядом. Но в последних версиях SDK некоторые подобные жесты уже могут обрабатываться вполне корректно.

3.3. Программное обеспечение

Для работы с Leap Motion необходимо скачать с официального сайта драйверы и специальный модуль SDK, содержащий в себе все основные библиотеки, позволяющие отслеживать данные, приходящие с устройства, и обрабатывать их. В нем хранятся специальные математические алгоритмы, которые позволяют выделить контуры рук, пальцев. В последних версиях появилась возможность выделять отдельные части рук.

На момент написания последней стабильной версией является SDK 3.1.

Есть 15 основных классов-сущностей, которые составляют Leap Motion API.

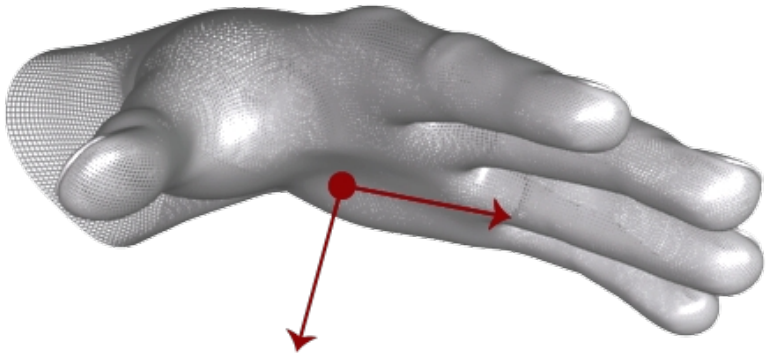
1. Controller – основной интерфейс для работы с Leap Motion. Необходим для отслеживания и получения информации. Имеет свойства: Config Config – возвращает объект Config, с информацией о текущей конфигурации; DeviceList Devices – лист устройств, подключенных в данный момент; SynchronizationContext EventContext – используется для передачи событий; EventHandler<LeapEventArgs> Init – инициализатор соединений; bool

IsConnected – отчет о подключении Leap Motion и его доступности из приложения; bool IsServiceConnected – показывает, есть ли доступ у приложения к сервису Leap Motion. Так же, у интерфейса Controller есть методы: void ClearPolicy(Controller.PolicyFlag policy) – очищает текущую политику; Controller() – конструктор для нового объекта Controller; Controller(int connectionKey) – создает объект Controller используя специальный ключ для соединения; FailedDeviceList FailedDevices() – получает лист подключенных, но не корректно работающих устройств; Frame Frame(int history) – возвращает кадр данных, обнаруженных устройством, с номером «текущий номер кадра - history»; Frame Frame() – возвращает самый последний отслеженный кадр; Frame GetInterpolatedFrame(Int64 time) – возвращает кадр из указанного времени, вставляя данные между существующими кадрами, если необходимо; Frame GetTransformedFrame(LeapTransform trs, int history = 0) – возвращает кадр объекта со всеми данными трансформированными в особую матрицу; Frame GetTransformedInterpolatesFrame(LeapTransform trs, Int64 time) – возвращает кадр из указанного времени, интерполируя данные между существующими кадрами, если необходимо, и трансформирует данные используя специальную матрицу трансформации; bool IsPolicySet(Controller.PolicyFlag policy) – возвращает текущие настройки для данной политики; long Now() – возвращает значение временной метки в микросекундах, которая ближе всего к текущему моменту; Image RequestImages(Int64 frameId, Image.ImageType type) – запрашивает изображение с сервиса, в качестве ImageType используются ImageType.TYPE_DEFAULT – нормальный формат и ImageType.RAW – формат, в котором данные передаются в «сыром», необработанном виде, и это зависит от устройства. Помимо этого, у объекта Controller есть список enum значений PolicyFlag: POLICY_DEFAULT == 0, POLICY_BACKGROUND_FRAMES == (1 << 0) – принимает фон кадров, POLICY_OPTIMIZE_HMD == (1 << 2) – оптимизирует отслеживание для устройства, установленного на очки виртуальной реальности, POLICY_ALLOW_PAUSE_RESUME == (1 << 3) –

разрешает ставить на паузу и снова запускать службы Leap Motion.

2. Frame – класс, представляющий набор обнаруженных рук и пальцев в единичный момент времени. Свойства: float CurrentFramesPerSecond – скорость, с которой Leap Motion обеспечивает кадры данных (возвращает значение количества кадров в секунду); List<Hand> Hands – лист объектов Hand, определенных в этом кадре, в произвольном порядке; long Id – уникальный идентификатор текущего фрейма; InteractionBox InteractionBox – текущий объект InteractionBox в кадре; byte[] Serialize – зашифровывает текущий кадр в строку байтов; long Timestamp – время, за которое осуществлялся захват кадра. Методы: void Deserialize(byte[] arg) – декодирует строку байт, для замены свойства текущего кадра; Frame() – конструктор; Frame(long id, long timestamp, float fps, InteractionBox interactionBox, List<Hand> hands) – конструктор для объекта Frame (например, можно вызвать предыдущий кадр через controller.Frame(1)); Hand Hand(int id) – достает из текущего фрейма объект Hand с конкретным идентификатором; bool Equals(Frame other); string ToString(); Frame TransformedCopy(LeapTransform trs).

3. Hand – класс, представляющий физические характеристики распознанной руки. Свойства: Arm Arm – объект Arm (предплечье) прикрепленный к текущей руке; LeapTransform Basis – ориентация руки в базисной матрице; float Confidence – уровень доверия к данным о заданной руке, от 0.0 до 1.0; Vector Direction – направление от ладони к пальцам; List<Finger> Fingers – лист объектов палец (Finger) принадлежащих этой руке; float GrabAngle – угол между пальцами и ладонью; float GrabStrength – сила сжатия руки; int Id – уникальный ID объекта рука (Hand); bool IsLeft, bool IsRight – true, если текущая рука левая, правая; Vector PalmNormal – вектор нормали ладони (см. Рисунок 7);



Рисунок

Vector PalmPosition – центр позиции ладони в миллиметрах; Vector PalmVelocity – скорость изменения позиции ладони в миллиметрах/секунду; float PalmWidth – ширина ладони в миллиметрах; float PinchDistance – расстояние между большим и указательным пальцем в миллиметрах; float PinchStrength – сила сжатия между большим и указательным пальцами; Vector StabilizedPalmPosition – стабилизируемая позиция ладони; float TimeVisible – время, в течении которого, рука отслеживалась контроллером; Vector WristPosition – позиция запястья. Методы: Finger Finger(int id) – возвращает объект палец (Finger) с переданным id; Hand() – конструктор; Hand(long frameID, int id, float confidence, float grabStrength, float grabAngle, float pinchStrength, float pinchDistance, float palmWidth, bool isLeft, float timeVisible, Arm arm, List< Finger > fingers, Vector palmPosition, Vector

stabilizedPalmPosition, Vector palmVelocity, Vector palmNormal, Vector direction, V ector wristPosition) – конструктор для нового объекта Hand, где confidence – уровень доверия к данным (от 0 до 1), stabilizedPalmPosition – отфильтрованная по времени позиция ладони; bool Equals(Hand other); string ToString(); Hand TransformedCope(LeapTransform trs).

4. Arm – класс представляющий предплечье. У него есть свойства: Vector ElbowPosition - положение локтя и Vector WristPosition - положение кисти. Так же есть методы: Arm() – пустой конструктор, который дает значение Arm из объекта Hand; Arm(Vector elbow, Vector wrist, Vector center, Vector direction, float length, float width, LeapQuaternion rotation) - конструктор для нового объекта, в котором задаются позиция локтя, позиция кисти, позиция центральной точки между локтем и кистью, вектор направления от локтя к кисти, длина между локтем и кистью в миллиметрах, средняя ширина руки и матрица базиса, представляющего ориентацию руки; new Arm TransformedCope(LeapTransform trs) – создание копии этого объекта, трансформируемой специальным трансформером; а так же стандартные методы объекта string ToString() и bool Equals(Arm otherArm)

5. Finger – класс, представляющий отслеживаемый палец. Берет значение пальца из объекта руки (Hand) или кадра(Frame). Имеет свойства: Vector Direction – направление, которое указывает этот палец (см. Рисунок 8);

Рисунок

int HandId – идентификатор руки (Hand) ассоциированной с этим пальцем; int Id – уникальный ID, который не изменяется, пока объект (палец) остается в кадре; bool IsExtended – является ли палец выпрямленным или согнутым; float Length – длина пальца в миллиметрах; Vector StabilizedTipPosition – стабилизированное положение кончика пальца; float TimeVisible – промежуток времени, когда этот палец отслеживался; Vector TipPosition – позиция кончика пальца в

миллиметрах от начала координат Leap Motion; Vector TipVelocity – скорость изменения позиции кончика пальца в миллиметрах/секунду; Finger.FingerType Type – название текущего пальца; float Width – ширина пальца в миллиметрах. Методы: Bone Bone(Bone.BoneType boneIx) – возвращает кость этого пальца с индексом boneIx; Finger() – конструктор для объекта палец; Finger(long frameId, int handId, int fingerId, float
timeVisible, Vector tipPosition, Vector tipVelocity, Vector direction,Vector stabilized TipPosition, float width, float length, bool
isExtended, Finger.FingerType type, Bone metacarpal, Bone
proximal, Bone intermediate, Bone distal) – конструктор для создания нового объекта Finger, где frameId – id кадра, в котором этот палец появляется, handId – id руки, которой принадлежит этот палец, fingerId – id этого пальца (handId + 0-4 в зависимости от позиции пальца), timeVisible – как долго этот объект отслеживался, metacarpal, proximal, intermediate, distal – кости, из которых состоит этот палец, начиная с ближайшей к ладони; string ToString(); Finger TransformedCope(LeapTransform trs). Помимо этого, у Finger есть перечислитель – FingerType enum – название пальцев: TYPE_THUMB == 0, TYPE_INDEX == 1, TYPE_MIDDLE == 2, TYPE_RING == 3, TYPE_PINKY == 4, TYPE_UNKNOWN == -1

6. Bone – класс, представляющий отслеживаемые кости. Каждый палец (finger) содержит 4 кости. Каждая кость в пальце имеет свой номер (от 0 до 3) от основания до кончика. Имеет свойства: LeapTransform Basis – ортонормированный базисный вектор, Vector Center – центр кости, Vector Direction – нормализованное направление от основания к кончику, float Length – длина кости в миллиметрах, Vector NextJoint – конец кости, тот который ближе к кончику пальца, Vector PrevJoint – конец кости, тот который ближе к ладони, LeapQuaternion Rotation – ориентация кости представленная в виде кватерниона (Quaternion), Bone.BoneType Type – название текущей кости и float Width – средняя ширина плоти вокруг кости в миллиметрах. Так же есть 5 функций:

Bone() – пустой конструктор;
Bone(Vector prevJoint, Vector nextJoint, Vector center, Vector direction, float length, float width, Bone.BoneType type, LeapQuaternion rotation) – создает новый объект Bone, где prevJoint, nextJoint – позиция предыдущего, следующего сочленения, type – тип кости, а остальные переменные аналогичны переменным с такими же названиями из конструкторов, рассмотренных ранее; так же содержит стандартные методы bool Equals(Bone otherBone), string ToString() и Bone TransformedCope(LeapTransform trs). Помимо свойств и методов, сущность Bone имеет перечислитель BoneType enum, который содержит названия костей пальца и имеет значения: «TYPE_INVALID = = -1; TYPE_METACARPAL = = 0; TYPE_PROXIMAL = = 1; TYPE_INTERMEDIATE = = 2; TYPE_DISTAL = = 3»

7. Device – класс, представляющий физически присоединенной устройство. По сути, представляет собой все данные о подключённом Leap Motion. Имеет свойства: float Baseline – дистанция между центральными точками камер, является базовым значением и вместе с максимальным разрешением влияет на максимальную дальность; float HorizontalViewAngle – угол обзора по оси x, вдоль длинной стороны устройства (см. Рисунок 9), возвращает значение в радианах;

Рисунок

bool IsEmbedded – возвращает true, если устройство непосредственно встроено в компьютер (ноутбук, клавиатуру…) и false, если это автономный контроллер; bool IsLightingBad – проверяет, достаточность инфракрасной подсветки, и выдает false, если ее слишком мало или слишком много, что может помешать корректному отслеживанию; bool IsSmudged – обнаружение пятна на светофильтре контроллера, которое может помешать правильному распознаванию; bool IsStreaming – сообщает, находится ли текущее устройство в потоковой передаче данных в приложение, единовременно мажет быть только одно устройство со значением true в этом поле; float Range – максимально

допустимое расстояние от центра устройства в миллиметрах, на котором гарантируется точное распознавание; string SerialNumber – серийный номер устройства, уникален, состоит из 2х букв и 11 цифр; Device.DeviceType Type – тип устройства; float VerticalViewAngle – угол обзора по оси z, вдоль короткой стороны устройства, возвращает значение в радианах (см. Рисунок 10).

Рисунок

Имеет всего 3 стандартных метода: Device() – конструирует объект Device, bool Equals(Device other) и string ToString().

8. DeviceList – представляет лист объектов типа Device. Имеет свойства: Device ActiveDevice, показывает девайс, который в данный момент потокого отслеживает данные, и bool IsEmpty, который true – если DeviceList не пуст. Имеет только один метод – конструктор, DeviceList().

9. FailedDevice – предоставляет информацию об аппаратном обеспечении LeapMotion, который физически подключен к компьютеру, но работает не правильно. Имеет два свойства: FailedDevice.FailureType Failure – показывает причину некорректной работы (почему устройство попало в этот класс) и string PnpId – id девайса plug-and-play. В свойстве Failure может быть одно из следующих значений: FailureType::FAIL_UNKNOWN – причина ошибки неизвестна; FailureType::FAIL_CALIBRATION – устройство имеет плохую калибровочную запись; FailureType::FAIL_FIRMWARE – встроенной программное обеспечение устройства повреждено или не обновленно; FailureType::FAIL_TRANSPORT – устройство не отвечает на запросы; FailureType::FAIL_CONTROL – служба не может установить необходимые интерфейсы управления USB; FailureType::FAIL_COUNT – в настоящее время не используется. Так же этот класс имеет один метод bool Equals(FailedDevice other).

10. FailedDeviceList – список объектов FailedDevice, в котором содержатся

все объекты LeapMotion, которые вызвали ошибки при подключении к компьютеру. Имеет одной свойство bool IsEmpty. Так же имеет два метода: FailedDeviceList() – конструктор, создающий пустой лист и FailedDeviceList Append(FailedDeviceList other) – добавляет список не работающих устройств к текущему.

11. Config – обеспечивает доступ к информации о конфигурации системы Leap Motion. Не имеет свойств, но имеет много методов: Config(int connectionKey) – создает новый объект Config для задания параметров конфигурации во время выполнения; bool Get<T>(string key, Action<T> onResult) – запрашивает значение конфигурации, где key – строковый идентификатор параметра, onResult – действие, выполняемое при возвращении значения; bool GetBool(string Key), bool GetFloat(string key), bool GetInt32(string key), bool GetString(string key) – дают логическое, с плавающей точкой, целочисленное и строковое представление для указанного ключа key; bool Save() – сохраняет текущее состояние конфигурации; bool Set<T>(string key, T value, Action<bool> onResult), bool SetBool(string key, bool value), bool SetFloat(string key, float value), bool SetInt32(string key, int value), bool SetString(string key, string value) – устанавливают значение value в значение ключа key; Config.ValueType Type(string key) – возвращает тип значения ключа. Type может возвращать только одно из следующих значений: TYPE_UNKNOWN == 0, TYPE_BOOLEAN == 1, TYPE_INT32 == 2, TYPE_FLOAT == 6, TYPE_STRING == 8.

12. Vector – класс, описывающий структуру, которая представляет собой трех-компонентный математический вектор или точку, такую как направление или позиция в трехмерном пространстве. По умолчанию оси располагаются как показано на Рисунке 11.

Рисунок

Объект имеет следующие свойства: float Magnitude – величина или длина вектора; float MagnitudeSquared – квадрат величины или длины вектора; Vector Normalized – нормализованная копия этого вектора;

float Pitch – угол наклона в радианах, вращение вокруг оси x (см. Рисунок 12);

Рисунок

float Roll – угол при вращении вокруг оси y (см Рисунок 13);

Рисунок

float Yaw – угол поворота вокруг вертикальной оси z (см Рисунок 14)

Рисунок

Помимо этих, имеет компоненты float x, float y, float z – соответствующие значениям вектора. Методы: float AngleTo(Vector other) – угол между текущим вектором и другим (other) в радианах; Vector Cross(Vector other) – векторное произведение текущего вектора и заданного; float DistanceTo(Vector other) – расстояние между точками, представляющими текущий объект вектора (Vector) и заданный; float Dot(Vector other) – скалярное произведение текущего вектора с заданным; bool IsValid() – проверяет на конечность и определенность все объекты текущего вектора, true – если все конечны и определены; float[] ToFloatArray() – конвертирует текущий вектор в массив из трех чисел с плавающей точкой [x, y, z]; Vector(float x, float y, float z) – создает новый вектор на основе заданных координат; Vector(Vector vector) – копирует вектор vector; bool Equals(Vector other). Так же имеет публичные статические функции, переопределяющие операторы: bool operator!=(Vector v1, Vector v2) – сравнение; Vector operator*(Vector v1, float scalar) – умножает вектор на число; Vector operator*(float scalar, Vector v1) – умножает вектор на скаляр с левой стороны; Vector operator+(Vector v1, Vector v2) – складывает два вектора покомпонентно;

Vector operator-(Vector v1, Vector v2) – покомпонентно вычитает вектор; Vector operator-(Vector v1) – отрицает вектор; Vector operator/(Vector v1, float scalar) – делит вектор на скаляр; bool operator==(Vector v1, Vector v2) – сравнивает два вектора на равенство.

13. Image – класс, представляющий единичное изображение с одной из камер. Так же предоставляет карту искажения для коррекции объектива. Свойства: int BytesPerPixel – количество байт в пикселе; byte[] Data – данные изображения; float[] Distortion – калибровочная карта для этого изображения; int DistortionHeight – искажение карты высот; int DistortionWidth – шаг карты искажений; Image.FormatType Format – формат изображения; int Height – высота изображения; Image Invalid – возвращает неправильный объект изображения, если такой был; bool IsValid – true, если объект Image содержит корректные данные; float RayOffsetX– горизонтальный луч смещения; float RayOffsetY – вертикальный луч смещения; float RayScaleX – горизонтальный луч масштаба; float RayScaleY – вертикальный луч масштаба; long SequenceId – последовательный ID изображения; long Timestamp – возвращает временную метку, которая показывает, когда это изображение было захвачено устройством; int Width – ширина изображения. Методы: Image() – конструктор; Vector Rectify(Vector uv) – обеспечивает исправление искажения, перехватывая заданную точку на изображении; Vector Warp(Vector xy) – используется для коррекции изображения; Vector Warp(Vector xy, float targetWidth, float targetHeight) – убирает искажение яркости для пикселя в изображении;bool Equals(Image other); string ToString().

14. InteractionBox – класс, представляющий максимальный параллелепипед, полностью вписанный в поле зрения контроллера Leap Motion (см Рисунок 15).

Рисунок

Свойства: Vector Center – центр объекта в координатах устройства (в миллиметрах), равноудаленная точка от противоположных сторон; float Depth –

глубина объекта в миллиметрах, измеренная вдоль оси z; float Height – высота объекта в миллиметрах, измеренная вдоль оси y; bool IsValid – возвращает true, если объект подходящий; Vector Size – измерение параллелепипеда вдоль каждой из осей; float Width – ширина объекта в миллиметрах, измеренная вдоль оси x. Методы: Vector DenormalizePoint(Vector normalizedPosition) – преобразует позицию, определенную нормированным взаимодействием в координаты контроллера в миллиметрах; InteractionBox(Vector center, Vector size) – конструктор для нового объекта с заданным центром и размером; Vector NormalizePoint(Vector position, bool clamp = true) – нормализация координат точки используя текущий объект ;bool Equals(InteractionBox other); string ToString().

15. Matrix – класс, представляющий структуру матрицы преобразования. Свойства: Vector origin – коэффициент перевода для всех трех осей; Vector xBasis, Vector yBasis, Vector zBasis – базисные векторы осей x, y, z. Методы: Matrix(Matrix other) – создает копию матрицы; Matrix(Vector xBasis, Vector yBasis, Vector zBasis) – конструктор нового объекта Matrix на основе заданных базисов; Matrix(Vector xBasis, Vector yBasis, Vector zBasis, Vector origin) – конструктор нового объекта Matrix на основе заданных базисов и коэффициента перевода; Matrix(Vector axis, float angleRadians) – конструктор для нового объекта Matrix на основе угла вращения вокруг заданного вектора; Matrix(Vector axis, float angleRadians, Vector translation) – конструктор матрицы преобразований, задающий вращение вокруг указанного вектора и последующее его перемещение; Matrix RigidInverse() – представляет матрицу, обратную к текущей; void SetRotation(Vector axis, float angleRadians) – устанавливает текущую матрицу преобразования для представления вращения вокруг указанного вектора; float[] ToArray3x3(float[] output), double[]

ToArray3x3(double[] output), float[] ToArray3x3(), float[] ToArray4x4(float[] output), double[] ToArray4x4(double[] output), float[] ToArray4x4() – копирует текущую матрицу в массив из 9, 16 значений в главных столбцах; Vector TransformDirection(Vector direction) – трансформирует вектор заданной матрицей используя только повороты и масштабирования; Vector TransformPoint(Vector point) – преобразовывает вектор текущей матрицей, используя поворот, масштабирование и перемещение;bool Equals(Matrix other); string ToString(). Так же имеет одну статичную функцию Matrix operator*(Matrix ma1, Matrix m2) – перемножает 2 матрицы.

Глава 4. Реализации

4.1. Выбор и обоснование жестов

Одной из главных задач нашего исследования стоял выбор и обоснование удобной системы жестов, для навигации в виртуальном пространстве.

Первым, самым необходимым, был жест для определения объекта, с которым пользователь хочет взаимодействовать. В связи с тем, что сейчас очень распространены сенсорные экраны, то вполне естественным жестом для выполнения этой задачи стал аналог жеста «ScreenTap» (см Рисунок 16). Он заключается в нажатии пальцем на желаемом объекте, и удержание пальца на несколько секунд. После этого, в зависимости от объекта, происходят какие-либо действия (открывается меню, выбирается его пункт…).

Рисунок

Следующее, что необходимо для удобной навигации – движение. Так как мы разрабатываем систему, для взаимодействия с 3D очками, то поворот в ней осуществляется поворотом головы (или всего тела) в нужную сторону. В связи с этим, нам необходимо обработать только движение вперед или назад (иначе говоря приблизить – отдалить объект). Для этого мы выбрали жест называемый «Pinch» (см Рисунок 17). Он заключается в соединении большого и указательного пальцев одной руки, и удержании их в таком положении. Для движения вперед удерживается правой рукой, для движения назад – левой. Мы решили использовать обе руки, так как это в некоторой степени упрощает использование нашей системы. Если использовать только одну руку, то появляется много лишних действий: сначала надо выбрать направление движения вперед или назад, затем начать движение и после этого остановить движение. Это сразу добавляет несколько движений, которые пользователю надо запоминать и делать. Использование обеих рук упрощает эту задачу. Выбор направления происходит автоматически, в зависимости от вида руки. И пользователю остается только задать начало движения и скомандовать, когда остановиться. Для этого и используется жест «Pinch». Когда пользователь зажимает пальцы необходимым образом, то спустя несколько секунд, необходимых для опознавания того, что это был не случайный жест, он начинает двигаться. Как только жест меняется – движение останавливается. Иными словами, движение идет только пока пользователь удерживает жест «Pinch».

Рисунок

После выбора предыдущих жестов, мы решили, что должен быть некий жест, который устанавливает так называемую «паузу», то есть пользователь использует этот жест, и все остальные жесты игнорируются до повторного введения жеста «паузы». Это необходимо, если, например, пользователю надо ненадолго отвлечься. Для выполнения этих функций мы выбрали жест «Stop»: рука ладонью от себя, все пальцы вместе (см Рисунок 18). Его так же

необходимо удерживать несколько секунд, для предотвращения ложных срабатываний. Сопровождается надписью о паузе. Для возобновления отслеживания жестов необходимо сделать его еще раз.

Рисунок

Следующим жестом, который мы решили включить в систему, стал жест выхода. Для этого мы выбрали движение «QuickSwitch». Оно заключается в необходимости провести рукой перед лицом снизу-вверх. Такой выбор обосновывается тем, что это не возвратный жест, то есть в отличие от остальных, его не получится отменить. В отношении остальных жестов, мы можем каким-либо образом отменить нежелательные действия (например, вернуться назад, если слишком далеко продвинулись вперед), в случае же выхода, с этим уже сложнее. Поэтому решено было сделать не просто какой-нибудь жест, который можно воспроизвести случайно, а целое движение.

После совещания со специалистами, для которых разрабатывается эта система, мы решили дополнить ее еще двумя жестами – действиями.

Первый из них: вращение объекта вокруг вертикальной оси будет осуществляться с помощью жеста «Пролистывание» (см Рисунок 19) в том направлении, куда мы хотим повернуть, то есть либо слева на право, либо наоборот. Это удобно использовать, например, когда мы хотим рассмотреть изучаемый объект издалека и со всех сторон.

Рисунок

Аналогично второй жест: вращение объекта вокруг горизонтальной оси. Так же используем жест пролистывания, но сверху вниз или снизу-вверх, для поворота.

4.2. Реализация

Для реализации были использованы специальный контроллер, отслеживающий движения Leap Motion с очками виртуальной реальности Oculus Rift и программное обеспечение Unity 3D с языком программирования c#. В качестве основы взяты библиотеки из LeapMotionCoreAssets.

Oculus Rift это специализированные очки виртуальной реальности, состоящие из 2х линз-дисплеев, на которые транслируется изображение. Их особенность заключается в наличии различных датчиков, благодаря которым можно определять положение головы человека в текущий момент, что позволяет создавать полноценную 3D модель. Так же их можно соединить непосредственно с Leap Motion, что позволяет удобно использовать их вместе.

В качестве основной среды разработки мы выбрали Unity. Unity – это инструмент для разработки двух- и трехмерных приложения для систем Windows, OS X, Android и другими. Есть возможность создавать приложения для запуска в браузерах, с помощью специального модуля Unity Web Player или через реализацию WebGL. Мы выбрали именно Unity для разработки, т.к. в этой программе можно достаточно удобно отлаживать код, отслеживать все действия, и у Leap Motion есть специальный модуль упрощающий работу через Unity.

Для устранения ложных срабатываний, все жесты необходимо удерживать несколько секунд.

Каждый из выбранных нами жестов можно отличить по нескольким критериям.

1. Жест «ScreenTap». Сила сжатия руки близка к 1(сжата в кулак), распрямлен только указательный палец (угол между костьми пальца близок к 180), остальные согнуты.

2. Жест «Pinch». Расстояние между большим и указательным пальцами близко к 0, сила сжатия ладони не большая.

3. Жест «Stop». Сила сжатия руки близка к 0 (рука полностью разжата),

направлена ладонью от себя, угол между пальцами близок к 0.

4. Жест «QuickSwitch». Рука полностью разжата и ее координаты изменяются только по одной оси z(по остальным изменения не сильные)

5. Жест «Пролистывание». Кисть (предплечье) не меняет свое положение в пространстве, двигаются только пальцы либо по горизонтали либо по вертикали.

Для работы с Leap Motion мы подключили библиотеку “Leap”, в которой хранятся все классы-сущности для контроллера.

Для начала, опишем логику работы. Сначала мы проверяем, сделан ли в данный момент жест Stop (точнее удерживался ли он необходимое время), если да, то меняем специальную переменную bool _stopTracking на противоположное значение. После этого смотрим, если в данный момент переменная _stopTracking находится в значении true, значит включен режим игнорирования жестов, и мы ничего не делаем, пока снова не отследим жест Stop. Если эта переменная false, значит идем дальше. Дальнейшие жесты можно проверять в любом порядке. Мы начинаем с жеста QuickSwitch. Если он распознался, значит выходим, завершая программу. Затем проверяем жест Pinch и если он распознался, то начинаем двигаться, пока его не отпустят. И последним, мы проверяем жест ScreenTap.

Проверка каждого жеста и выполнение действий, связанных с ним, описываются в отдельных функциях.

Для реализации жеста ScreenTap была создана специальная сфера малого диаметра, на конце указательного пальца. Касанием объекта считается попадание этого объекта, или его части, в сферу. Если таких объектов несколько, то выбираем тот из них, расстояние до центра которого меньше. Это описывается в специальной функции «UpdateScreenTap»

Для реализации жеста Pinch была написана функция «UpdatePinch». В этой функции достается компонент «Рука» (HandModel).

HandModel hand_model = GetComponent<HandModel>();

Hand leap_hand = hand_model.GetLeapHand();

После этого, проверяется степень ее сжатости. Если она близка к 1, то выходим из функции, это не жест Pinch. Иначе, берется «Большой палец» руки

Vector leap_thumb_tip = leap_hand.Fingers[0].TipPosition;

и сравнивается расстояние от него до конца кости, ближайшей к кончику указательного пальца этой же руки

if

(leap_joint_position.DistanceTo(leap_hand.Fingers[1].Bone((Bone.BoneType)

3).NextJoint) < trigger_distance)

trigger_pinch = true;

Если это расстояние меньше константы (trigger_distance), то мы считаем, что это жест Pinch. Если эти условия непрерывно выполняются в течении нескольких секунд (проверка ложного движения), значит мы действительно сделали этот жест и можно начинать движение. Само движение описывает функция void Zoom(bool isLeftHand). Движение зависит от руки: левая – в функцию передается true, и движение идет назад, правая – в функцию передается false, и движение идет вперед. Направление определяется поворотом головы. При повороте головы в очках виртуальной реальности, они так же меняют и поворот камеры. При воспроизведении жеста Pinch мы фиксируем это направление и двигаемся только вдоль него. При изменении положения головы во время движения мы продолжаем движение в новом направлении. Но не рекомендуется слишком резко вращать головой, во избежание неприятных побочных эффектов от работы в виртуальной реальности.

Для реализации жеста Stop была написана функция «UpdateStopMove». Сначала, мы так же получаем объект рука.

HandModel hand_model = GetComponent<HandModel>();

Hand leap_hand = hand_model.GetLeapHand();

Затем считаем угол между кистью и предплечьем.

```
var angle = leap_hand.PalmPosition.AngleTo(leap_hand.WristPosition) * 180 /
Mathf.PI;
```

После этого, считаем угол между костьми в пальцах (он должен быть близок к 180, т.е. пальцы прямые)

```
var angleBetweenFinger = 0.0;
for (int i = 1; i < HandModel.NUM_FINGERS; ++i)
{
  Finger finger = leap_hand.Fingers[i];
  Finger prevFinger = leap_hand.Fingers[i - 1];
  for (int j = 0; j < FingerModel.NUM_BONES; ++j)
  {
        Vector leap_joint_position=finger.Bone((Bone.BoneType)j).Direction;
angleBetweenFinger+=leap_joint_position.AngleTo(prevFinger.Bone((Bone.BoneTy
pe)j).Direction);
  }
}
```

Вычислив эти данные, мы проверяем общие условия: если угол между ладонью и предплечьем < 15, угол между костьми в пальце < 2.5, сила сжатия руки меньше 0.1 и рука повернута ладонью от нас, то считаем, что это жест Stop.

```
if ((angle < 15) && (angleBetweenFinger < 2.5) && (leap_hand.GrabStrength
< 0.1) && (leap_hand.PalmNormal.x > 0) && (leap_hand.PalmNormal.y > 0)
&& (leap_hand.PalmNormal.z > 0))
```

Затем смотрим, если этот жест удерживался необходимое время, то, в зависимости от текущего значения специальной переменной, либо включаем либо отключаем паузу.

Для реализации жеста QuickSwitch была написана функция «UpdateQuichSwitch». В ней мы проверяем: если сила сжатия ладони меньше 0.1 и положение руки изменяется только по оси z с предыдущим положением (изменения по остальным осям не превышают 0.1), то считаем что это

необходимый нам жест. Затем смотрим на специальную сигнальную переменную: если она true, значит в данный момент мы в режиме просмотра и хотим выйти. Значит меняем значение этой переменной на false и отключаемся. После этого программа считается завершенной и трансляция на очки виртуальной реальности заканчивается.

Заключение

В ходе выполнения магистерской диссертации были изучены проблемы, возникающие при взаимодействии с виртуальной реальностью и предложены некоторые способы их устранения. Так же, был разработан специальный набор жестов для работы в виртуальной реальности. Для реализации этого набора был изучен специальный контроллер Leap Motion, с использованием возможностей которого, мы в дальнейшем написали необходимые функции.

Список литературы

1. Авербух В.Л., Авербух Н.В., Стародубцев И.С., Тоболин. Д.Ю. Использование жестовых интерфейсов при взаимодействии с объектами // Научная перспектива. № 10 (56) / 2014. Стр. 57-66.

2. Раскин Джеф, «The Human Interface. New Directions for Designing Interactive Systems»

3. Официальный сайт LeapMotion https://www.leapmotion.com/

4. Статья «Принцип работы устройства Leap Motion. Краткий обзор.» http://dvornichenko.com/2014/07/princip-raboty-ustrojstva-leap-motion-kratkij-obzor/

5. Статья «Leap Motion. Что внутри?» https://habrahabr.ru/company/avi/blog/199230/

6. Официальный сайт для разработчиков Leap Motion https://developer.leapmotion.com/documentation/

7. "Gesture-based Interaction Design: Communication and Cognition", 2014 CHI Workshop Toronto, Canada