Исследование и разработка жестового интерфейса для работы с инструментом визуализации цифровых голограмм в виртуальном окружении*

***В.А. Киселев*[1,2], *С.В. Клименко*[1,2], *М.В. Михайлюк*[3]**
kiselev.va@phystech.edu|stanislav.klimenko@gmail.com|mix@niisi.ras.ru
[1]Московский физико-технический институт
Протвино, [2]Институт физико-технической информатики
Москва, [3]НИИ системных исследований РАН

В различных системах визуализации существует потребность в жестовых интерфейсах взаимодействия. В работе исследуется возможность внедрения жестового интерфейса на основе камеры leap motion в инструмент для визуализации цифровых голограмм в виртуальной реальности. При постановке задачи предложена модель описания жеста и формат его хранения. Описан процесс реализации программного модуля для распознавания жестов. Предложен интерфейс взаимодействия на основе данного жестового интерфейса с системой визуализации. Обозначены недостатки и предложен подход для их устранения.

Ключевые слова: *трекинг, распознование жестов, leap motion, виртуальная реальность*

Research and Development of Gestural Interface for Working with Digital Hologram Visualization Tool in Virtual Environment*

***V.A. Kiselev*[1,2], *S.V. Klimenko*[1,2], *M.V. Mihayljuk*[3]**
[1]Moscow Institute of Physics and Technology
Protvino, [2]Institute of Computing for Physics and Technology
Moscow, [3]Scientific Research Institute for System Analysis of RAS

In some visualization systems, there is a neednes for gestural interaction interfaces. The paper explores the possibility of introducing a gestural interface based on the Leap Motion camera into a tool for visualizing digital holograms in a virtual reality. In the formulation of the problem, a model for describing the gesture and the format of its storage are proposed. The process of implementing a software module for recognizing gestures is described. An interaction interface based on this gestural interface with the visualization system is proposed. Deficiencies are indicated and an approach is proposed for their elimination.

Key words: *tracking, gesture recognition, leap motion, virtual reality*

Введение

В работе [1] предложена техника стерео-визуализации цифровых голограмм биообъектов. В очках виртуальной реальности Oculus Rift DK2 [2] оператор может рассматривать отдельные трёхмерные модели биообъектов и производить различные манипуляции со сценой и данными. Для таких манипуляций контроллер Leap Motion [3] был закреплён на очках, а поставляемое с контроллером SDK было задействовано для рендеринга моделей рук в режиме копирования. В виртуальной сцене были размещены объекты, и оператор мог взаимодействовать с ними подобно объектам из реального мира. Такого рода взаимодействия влияли на параметры визуализации. Например, для вращения модели изучаемого образца, оператор поворачивал рукой куб на соответствующий угол (демонстрируется на Рис. 1).

Однако, визуальное присутствие виртуальных объектов, размещённых для взаимодействий, мешает наблюдать изучаемый образец. Требуется реализация механизма, который бы позволил скрывать и вызывать подобные виртуальные объекты. При этом использование клавиатуры и прочих внешних устройств ввода не является удобным. Так как человек, находящийся в очках виртуальной реальности не видит внешнего мира то, и взаимодействовать с внешними объектами может только на ощупь. Реализация жестового интерфейса управления в этом случае является подходящим решением. Преследуя общую цель по исследованию и разработке человеко-машинного взаимодействия для виртуального окружения на основе датчика глубины, цель данной работы в исследовании возможностей внедрения жестового интерфейса на основе такой системы в инструмент для визуализации цифровых голограмм [1, 4]. В работах [5, 6] был проведён обзор по таким устройствам. Камера Leap Motion по-прежнему является эффективным средством для задач трекинга мелкой моторики рук. Поэтому разработка интерфейса взаимодействия с

Работа выполнена и опубликована при финансовой поддержке РФФИ, гранты 15-29-01135 офи_м, 17-07-20366, 17-57-52011 МНТ_а

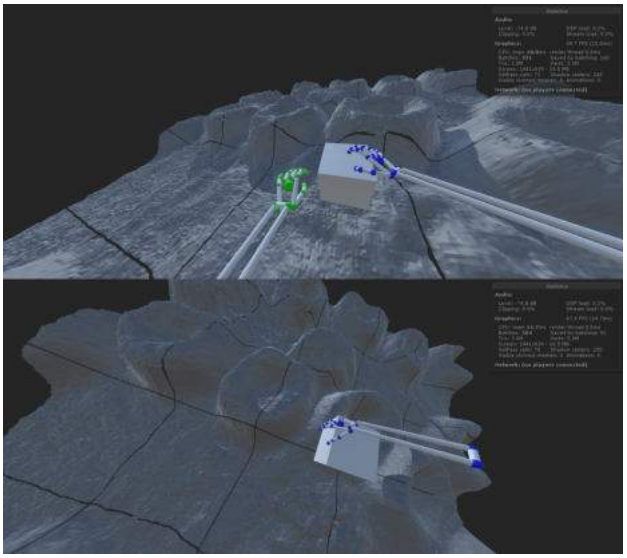

Рис. 1: Оператор вращает исследуемый образец, взаимодействуя виртуальными руками с кубом. Угол поворота куба – соответствуют углу поворота образца. В качестве образца - модель красных кровяных тел человеческой крови

помощью жестов велась с использованием данной камеры.

Постановка задачи

Жесты мелкой моторики можно разделить на две категории: статичные и динамичные. Первое подразумевает определённую неподвижную позу, а второе выполнение жеста в движении. Для однозначного определения статичного жеста одной руки можно обрабатывать вектор нормали ладони в совокупности с расстояниями для каждого пальца от конца дистальной фаланги до центра запястья. Данная модель описания жеста довольно проста, и должна позволять проводить проверку в каждом кадре. Но, например, следующая пара поз может вызвать неоднозначность для алгоритма распознавания, на основе такой модели: поза, когда пястно-фаланговые суставы согнуты на 90 градусов, а все остальные суставы разогнуты, и поза когда все суставы согнуты, кроме пястно-фаланговых. Возможно, существуют и другие неоднозначности. Поэтому, предлагается реализовать алгоритм в виде программного модуля и проверить его работоспособность. Тогда постановку задачи можно разделить на два этапа. Первый этап заключается в создании приложения, которое позволяет:

— распознавать жесты из базы данных, используя вышеописанную модель;
— записывать новые жесты в базу данных;
— управлять с помощью распознанных жестов виртуальным объектом.

Второй этап заключается в исследовании применимости данного метода, для работы с инструментом визуализации цифровых голограмм в виртуальной реальности [1], и проектировании жестового интерфейса управления для данного инструмента. На первом этапе в качестве виртуального объекта будем использовать примитив куб. Для наглядности предлагается визуализировать в сцене виртуальную кисть руки в режиме копирования с реальной руки. Таким образом распознавать жесты для правой и левой рук по отдельности, а при нажатии на определённую клавишу - записывать параметры руки (соответствующей нажатой клавише) в базу данных. Для оперативного доступа к данным будем использовать SDK Leap Motion для среды unity в которой и будем вести разработку на языке c#. Таким образом, на c# можно представить модель описания жеста следующим классом (Рис. 2).

```
public class CheckHandsGestures
{
    public int id;
    public string name;
    public Vector CheckPalmNormValue;
    public float CheckThumb;
    public float CheckIndex;
    public float CheckMiddle;
    public float CheckRing;
    public float CheckPinky;
}
```

Рис. 2: Модель описания жеста на c#

Здесь id – индекс жеста, name – его имя, CheckPalmNormValue – вектор нормали ладони руки, а CheckThumb, CheckIndex, CheckMiddle, CheckRing и CheckPinky – переменные для хранения значений расстояния для каждого пальца от конца дистальной фаланги до центра запястья.

Поднимая вопрос выбора формата данных для хранения жестов, следует учитывать факт разработки приложения для исследования метода обработки, а так же не исключать возможность последующего использования. Поэтому база данных должна легко поддаваться чтению человеком и быть многоплатформенной. Выбирая из текстовых форматов и XML, и JSON имеют библиотеки для создания и обработки данных из многих языков программирования. Но за счёт своей лаконичности по сравнению с XML, формат JSON может быть более подходящим для сериализации сложных структур. Таким образом, пример описания жеста для правой руки на JSON приведён на Рис. 3

Реализация програмного модуля

Корнем модели данных Leap Motion является объект *Frame*, представляющий кадр, который содержит в себе данные в виде свойств в каждый момент времени. Но кадр имеет право изменяться,

```
{
"RightHandChek":[
    {
        "id":0,
        "name":"GoUp",
        "CheckPalmNormValue":{"x":0.0,"y":1.0,"z":-0.2},
        "CheckThumb":0.0799999982118606,
        "CheckIndex":0.09700000286102295,
        "CheckMiddle":0.10700000077486038,
        "CheckRing":0.09700000286102295,
        "CheckPinky":0.08699999749660492
    }
```


Рис. 3: Пример описания жеста в JSON для правой руки

поэтому работать напрямую по указателю на текущий кадр (или какой-то объект, являющийся частью этого кадра) не надёжно. В связи с этим работая с данными Leap Motion приходиться копировать текущий кадр и только потом обрабатывать. Объекты *Frame* можно получать из объектов *Leap.Controller* и *LeapProvider*. Последний преобразует данные кадра в систему координат соответствующую системе координат сцены в unity. Создав таким образом в методе *Update()* экземпляр класса *Frame* можно оперативно получать данные о руках в виде списка «*List<Hand> hands = frame.Hands*» в каждом кадре. Класс *Hand* представляет физическую руку, обнаруженную Leap Motion. Он предоставляет доступ к атрибутам, описывающим положение, ориентацию и движение руки. Руки могут быть идентифицированы левой или правой. Внутри метода *Update()* удобно осуществлять проверку на эту идентификацию - *if (hand.IsRight)*. Данные о пальцах, так же можно получить в виде списка «*List<Finger> fingers = hand.Fingers*». *TipPosition* ялвяется свойством объекта *Finger* и возвращает вектор, содержащий координаты положения конца дистальной фаланги пальца. А вектор представляющий координаты положения запястья можно получить обратившись к методу *PalmPosition* экземпляра класса *Hand*. Тогда искомые значения расстояний считаются, как квадратный корень от суммы квадратов разности для каждой координаты *PalmPosition* и *TipPosition* соответственно. Ну а вектор нормали ладони возвращает метод *PalmNormal*, который так же доступен в свойствах класса *Hand*. Для задач чтения и записи данных в JSON была использована библиотека LitJSON [7], она сравнительно быстрая и написана на c#. Метод «*JsonMapper.ToObject <T>*» используется для указания типа возвращаемого объекта, но когда требуется прочитать данные JSON неопределённого класса можно использовать нетривиальный вариант *ToObject*, который возвращает экземпляр класса *JsonData*. То есть, *JsonData* - это тип общего назначения, который может содержать любые типы данных, поддерживаемые JSON. Работая с ним было реализовано чтение базы данных жестов в список типа данных созданного класса *CheckHandsGestures*. На каждый *id* из всех *itemData["RightHandChek"].Count* считываемая структура распарсивается в объект CheckG1, который затем добавляется в список. На Рис. 4 демонстрируется реализация данного разбора для

```
CheckG1.id=i;
CheckG1.name=(string)itemData["RightHandChek"][i]
["name"];
CheckG1.CheckPalmNormValue = new Vector((float)
(double)itemData["RightHandChek"]
[i]["CheckPalmNormValue"]["x"],
(float)(double)itemData["RightHandChek"]
[i]["CheckPalmNormValue"]["y"],
(float)(double)itemData["RightHandChek"]
[i]["CheckPalmNormValue"]["z"]);
CheckG1.CheckThumb=
(float)(double)itemData["RightHandChek"]
[i]["CheckThumb"];
CheckG1.CheckIndex=
(float)(double)itemData["RightHandChek"][i]["CheckIndex"];
CheckG1.CheckMiddle=
(float)(double)itemData["RightHandChek"][i]["CheckMiddle"];
CheckG1.CheckRing=
(float)(double)itemData["RightHandChek"][i]["CheckRing"];
CheckG1.CheckPinky=
(float)(double)itemData["RightHandChek"][i]["CheckPinky"];
checkRights.Add(CheckG1);
CheckG1 = new CheckHandsGestures();
```

Рис. 4: Реализация разбора данных из JSON в список *checkRights* экземпляров класса *CheckHandsGestures*

правой руки. Необходимость преобразовывать данные из *JsonData* в *double*, а уже затем в *float*, обусловлена особенностями этого текстового формата. Разбор происходит в аналогичном цикле и для левой руки. Оба списка заполняются до первого обновления кадров в функции события *Start()*. Затем эти элементы списка в цикле проверяются на соответствие заданным. Такая проверка происходит в каждом кадре. Модуль разности хранимого в базе данных параметра и соответствующего параметра в объекте *Frame* должен быть меньше заданного коэффициента. Такие коэффициенты символизируют погрешность распознования и подбираются экспериментально для каждого пальца. Ещё одним условием соответствия является сравнения угла между вышеупомянутыми векторами с аналогичным коэффициентом. Угол определяется с помощью метода *PalmNormal.AngleTo()*. Жест считается распознанным, когда одновременно выполнились все шесть условий соответствия. В случае успеха переменной присваивается *id* распознанного жеста, а в противном «*-1*». Таким образом, возможность манипуляции кубом реализована путём использования оператора управления *switch()* по данной переменной. Изменение его параметров осуществляется с помощью прибавления или вычитания постоянных значений к его различным свойствам. С каким свойством произойдёт изменение зависит от того, в какую метку *case* нас приведёт *switch()*. Как только жест перестаёт выполняться, меняется *id*, и соответственно *case*. На одну из та-

ких меток *case* можно установить и запись нового жеста в базу данных, но на данном этапе запись нового жеста реализована на нажатие клавиши (*Input.GetKey()*). Так же следует уточнить, что разбор данных при записи жеста происходит по аналогии с разбором при чтении, но вместо метода *ToObject()* используется метод *ToJson()*.

Проектирование интерфейса взаимодействия

Прежде чем проектировать интерфейс взаимодействия, следует сделать несколько допущений относительно инструмента визуализации цифровых голограмм, и тем самым пройти этап разработки концепции. Зафиксировав голограмму на матрицу цифровой камеры, и восстановив её численными методами можно получать различные 3D модели, в том числе поверхности. В данной концепции постулируется работа только с 3D моделями которые являются поверхностями. Такую поверхность будем называть изучаемым объектом или образцом. В один момент времени в виртуальной сцене может находиться только один изучаемый объект. Все остальные объекты в виртуальной сцене вспомогательные, а любые манипуляции с данными и со сценой должны быть направлены на изменение параметров визуализации изучаемого образца. Здесь не будут описаны алгоритмы следующие за этими манипуляциями, так как это является другим этапом работы. Управление изучаемым объектом будет происходить «виртуальной рукой». То есть, положения виртуальных кистей рук будут отрисовываться в режиме копирования с реальных рук. А определённые статичные жесты «виртуальной руки» должны вызывать в сцену виртуальные объекты, с которыми можно взаимодействовать посредством виртуальной руки. Такие объекты будут являться виртуальными интерфейсами взаимодействия (далее ВИВ) пользователя с изучаемым объектом. Управляя данными ВИВ (с помощью «виртуальной руки») можно будет влиять на следующие параметры: менять положение виртуальной камеры (точки наблюдения), производить кадрирование (обрезать изучаемый объект), менять точность детализации. То есть ВИВ разделены по смыслу, а внутренняя оболочка каждого из них должна соответствовать совершаемому с образцом действию. Более того, в виртуальном пространстве единовременно может быть размещён только один интерфейс взаимодействия (ВИВ). Для его удаления из виртуальной сцены, требуется повторить статичный жест, которым он (ВИВ) был вызван. Так же стоит задача управлять масштабированием изучаемого объекта. Для её решения предлагается использовать наглядный динамический жест «виртуальной руки». Если пользователь однозначно задаст «виртуальной рукой» геометрическую фигуру, то один из вариантов – это сопоставлять в динамике изменение её пропорции изменению пропорции изучаемого объекта. Например, соединив на каждой руке большой и указательный палец – можно обозначить в пространстве две точки. Интерпретируя эти точки как противоположные углы прямоугольника можно изменять его пропорции, тем самым изменяя пропорции изучаемого объекта – то есть масштабируя его. Предлагаемое описание оболочки для каждого ВИВ приведено ниже, и снабжено иллюстрацией для наглядности.

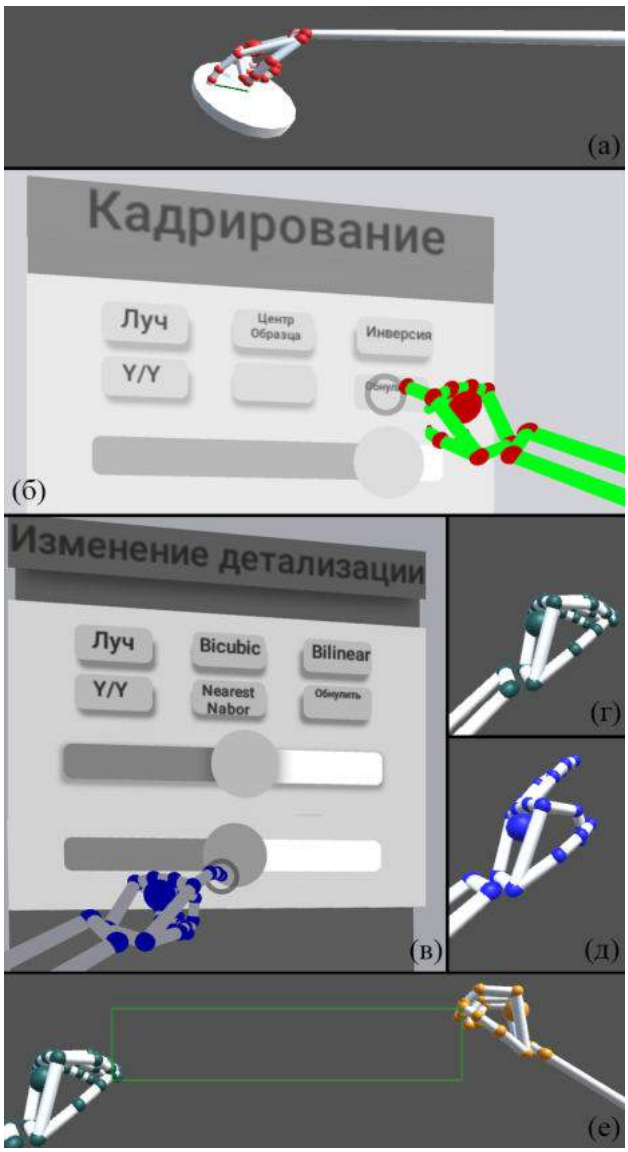

Рис. 5: Демонстрация виртуальных интерфейсов взаимодействия.

— **Интерфейс взаимодействия для перемещения точки наблюдения камеры по виртуальной сцене (перемещение пользователя по виртуальной сцене)** (см Рис. 5.а)

Функции меню выполняют два цилиндра. Один цилиндр размещён справа, в поле досягаемости для рук пользователя, другой слева. Если построить вектор между центром цилиндра и ближайшим элементом виртуальной кисти к нему, то скорость движения зависит от его абсолютного значения, а направление совпадает с направлением вектора. Движение происходит только в момент, когда кисть находится внутри контура цилиндра. Правый цилиндр размещён горизонтально и отвечает за движение в плоскости XY, а левый вертикально и отвечает за движение в плоскости ZX. Положение цилиндров сопряжено с положение пользователя в виртуальной сцене, и закреплено по центру между двух виртуальных камер, образующих стереопару. Угол наблюдения пользователь выбирает сам поворотом головы.

— **Интерфейс взаимодействия для кадрирования объекта** (см Рис. 5.б)
Предполагает форму с полосой прокрутки и кнопками. Управление происходит виртуальной рукой. Изменения к изучаемому объекту будут применяться по нажатию завершающей кнопки. Точное наполнение данного меню зависит от методов реализации кадрирования, модуль которых пока находится в стадии разработки.

— **Интерфейс взаимодействия для изменения детализации образца** (см Рис. 5.в)
Предполагает форму с полосами прокрутки и кнопками. Управление происходит виртуальной рукой. Изменения к изучаемому объекту будут применяться по нажатию завершающей кнопки. Точное наполнение данного меню зависит от методов реализации изменений детализации, модуль которых пока находится в стадии разработки. Планируется обрабатывать поверхность, как карту высот, к которой могут быть применены бикубическое преобразование, билинейное преобразование и преобразование ближайшего соседа для изменения детализации.

— **Интерфейс взаимодействия для масштабирования образца** (см Рис. 5.е)
В момент, когда пользователь соединяет на обеих руках большой и указательный пальцы – между точками «зажатия» строиться прямоугольник (левая рука – нижний левый угол, правая – верхний правый угол), размер этого прямоугольника в момент времени зажатия соответствует масштабу объекта 100%, а нулевой размер масштабу 0% . Отсюда высчитывается линейное соответствие между пропорцией размера прямоугольника и образца. В момент разжатия палецов – объект меняет свой масштаб. Горизонтальное ребро – соответствует масштабированию объекта в плоскости, вертикальное – масштабированию по оси Z. В ходе работы предполагается исследование вопроса удобства, то есть при необходимости добавить амортизирующий коэффициент в пропорцию соответствия масштаба прямоугольника, масштабу объекта (как вариант сопоставлять нулевой размер не нулевому масштабу).

Выводы

В реализованном программном модуле три жеста, были задействованы для вращения примитива куб. Каждый жест поворачивал виртуальный куб вокруг соответствующей оси. Суммарно было апробировано 36 жестов для анализа применимости данного метода и подбора коэффициентов погрешностей для распознавания. Поочерёдно тестируя каждую тройку жестов, формировалось абстрактное представление, как о распознаваемости жеста, так и об удобстве его использования. Подобные эксперименты не являются сильно показательными, тем не менее, они прояснили, что для разных жестов следует использовать различные погрешности, а отслеживание направления нормали ладони ни всегда актуально. Поэтому модель описания жеста стоит усовершенствовать, добавив в её иерархию, кроме погрешностей справочную переменную, которая бы управляла количеством проверяемых параметров в алгоритме распознавания (который тоже стоит дополнить с учётом этого). Например, выполняя жест, предполагающийся для масштабирования образца сложно запомнить какими должны быть положения не соединённых пальцев (см Рис. 5.г и Рис. 5.д). Кроме того вышеописанный процесс масштабирования, предполагает, что направление может и должно меняться в ходе выполнения этого процесса. Было сформировано впечатление, что используя интуитивно понятные жесты управлять виртуальным объектом удобно. Метод распознавания продемонстрировал себя как быстрый, не ресурсозатратный и пригодный для использования в инструменте визуализации цифровых голограмм.

Литература

[1] Каленков Г.С., Каленков С.Г., Киселев А.В., Клименко С.В., Сысоев Н.А., Хайден В.И. Визуализация цифровых голограмм биообъектов в виртуальном окружении // Труды Международной конференции «Ситуационные центры и ИАС4i для задач мониторинга и безопасности» SCVRT1516. – Изд-во ИФТИ, Протвино-Москва, 2016, ISBN 978-5-88835-043-0, С.291-294

[2] Официальная документация очков Oculus Rift DK2 Web: `https://developer.oculus.com/`

[3] Официальная документация камеры leap motion Web: `https://developer.leapmotion.com/`

[4] Каленков Г.С., Каленков С.Г., Киселев В.А., Клименко С.В. Виртуальное окружение как техника

визуализации гиперспектральных голограмм // VI Международная конференция по фотонике и информационной оптике. Сборник научных трудов. Москва. Изд-во НИЯУ МИФИ, 2017, ISBN 978-5-7262-2333-9, С.266-267

[5] Хворов Н.С., Чувилин К.В. Трекинг мелкой моторики в реальном времени// Труды Международной научной конференции по физико-технической информатике (СРТ2015). Москва-Протвино 2016

[6] Киселев В.А., Клименко А.С., Клименко С.В., Михайлюк М.В., Пестриков В.И., Хламов М.А., Чувилин К.В., Фурса М.В., Хаким Н.Л., Ши Т.К. Современные устройства трекинга для систем виртуального окружения// Труды Международной научной конференции по физико-технической информатике (СРТ1617). Москва-Протвино, 2016-2017, ISBN 978-5-88835-049-2

[7] LitJSON - Библиотека для обработки данных в формате JSON. Проект на GitHub: `https://lbv.github.io/litjson/`