УДК 004.51-056.24

РЕАЛИЗАЦИЯ БЕСКОНТАКТНОГО ВЗАИМОДЕЙСТВИЯ С ПРОТОТИПОМ КОМПЬЮТЕРНОГО ИНТЕРФЕЙСА В UNITY 3D

В. А. Зенг, О. В. Батенькина
Омский государственный технический университет,
valeriyazeng@mail.ru, oksi-bat@mail.ru

В статье рассмотрен процесс реализации прототипа бесконтактного взаимодействия пользователя с системой с помощью игрового движка Unity 3D. Описано создание объектов системы: пяти рабочих экранов, выдвигающейся панели меню, панели программ, приветственного экрана и окна программ. Для управления прототипом с помощью устройства Kinect произведено подключение ряда скриптов, разработанных компанией Microsoft, которые позволяют программно воспринимать любые жесты. Также описан комплект классов, позволяющих обработчику идентифицировать конкретные жесты пользователя и производить соответствующие действия. Приведены параметры анимации объектов для перехода между различными состояниями, которые могут изменяться посредством скриптов. Вся разработанная система методов и классов была аккумулирована в одном скрипте для удобства управления и вызова функций проверки. Для запуска интерфейса установлен пароль в виде определенного уникального движения, установленного пользователем. Данный прототип реагирует на шесть жестов: активное махание рукой, единовременный взмах руки влево, вправо, вверх, вниз и удержание руки на месте. В качестве альтернативного средства управления прототипом в Unity 3D предусмотрены соответствующие траектории мыши, которые будут восприниматься так же, как и жесты пользователя.

Ключевые слова: человеко-машинное взаимодействие, программное приложение, бесконтактный интерфейс, прототип, захват движения, технологии бесконтактного взаимодействия, пользователи с ограниченными возможностями здоровья.

IMPLEMENTATION OF CONTACTLESS INTERACTION WITH COMPUTER INTERFACE PROTOTYPE USING UNITY 3D

V. A. Zeng, O. V. Batenkina
Omsk State Technical University,
valeriyazeng@mail.ru, oksi-bat@mail.ru

The article presents the prototype implementation process of contactless user-machine interaction using the Unity 3D game engine. It describes the creation of system objects: five desktops, a drop-down menu bar, a taskbar, a welcome screen and a program window. To control the prototype using the Kinect, a number of scripts, which were originally developed by Microsoft Corporation, are used. These scripts allow recognizing and identifying all possible gestures in software. A set of classes that allow identifying pre-set user gesture and performing relevant action are also described. Objects animation parameters for a transition between different states, which can be changed by scripts, are described. The entire system of methods and classes is assembled into one piece to make control and verification functionality easier to manage. A password gesture is set as a certain unique movement by the user to start the user session. This prototype responds to six human gestures: active hand waving, one-time hand wave to the left, to the right, up, down and holding a hand still. Mouse's motions that can be recognized as alternative user gestures are provided.

Keywords: human-machine interaction, software application, contactless interface, program prototype, motion capture, contactless interaction technologies, users with disabilities.

Современные научные работы, посвященные исследованию человеко-машинного взаимодействия, направлены в основном на разработку интерфейсов, ориентированных на опытных пользователей, и почти не затрагивают вопросы человеко-машинной коммуникации для лиц с ограниченными возможностями. Так, глухонемые люди не могут использовать речевые интерфейсы, а люди с проблемами мелкой моторики не способны работать с клавиатурой или жестовыми интерфейсами. Разработка универсального бесконтактного интерфейса, пригодного для всех категорий пользователей, и реализация этого интерфейса, демонстрирующего возможности многомодальной человеко-машинной коммуникации, позволят решить такую проблему. Данный интерфейс будет включать различные естественные для человека способы передачи и восприятия информации: речь, жесты, движения головой и телом, чтение по губам, а также комбинации этих бесконтактных модальностей [1].

Для разработки прототипа бесконтактного компьютерного интерфейса был выбран игровой движок Unity 3D ввиду простоты его использования, возможности написания скриптов на языке C#, а также большого объема обучающей информации, доступной как на официальном сайте, так и на многочисленных тематических форумах [2].

Сначала были созданы объекты пяти рабочих экранов посредством двухмерного примитива *Plane* с тем же соотношением сторон, что и разработанные ранее интерфейсы. Подобным образом были созданы такие объекты системы, как выдвигающаяся панель меню, панель программ, приветственный экран и окно программ [3].

Далее следует создать и назначить материалы объектам сцены. Для этого создается пустой материал, на который накладывается текстура и редактируются такие параметры, как отражение, свечение, сглаживание и др. Если материал полупрозрачный, то выбирается отображение не *Standart*, как в случае с обычными текстурами, а *Transperent*, куда потом добавляется текстура типа .png, которая хранит в себе данные о прозрачности. Таким образом создаются и текстурируются все объекты сцены (рис. 1) [4].

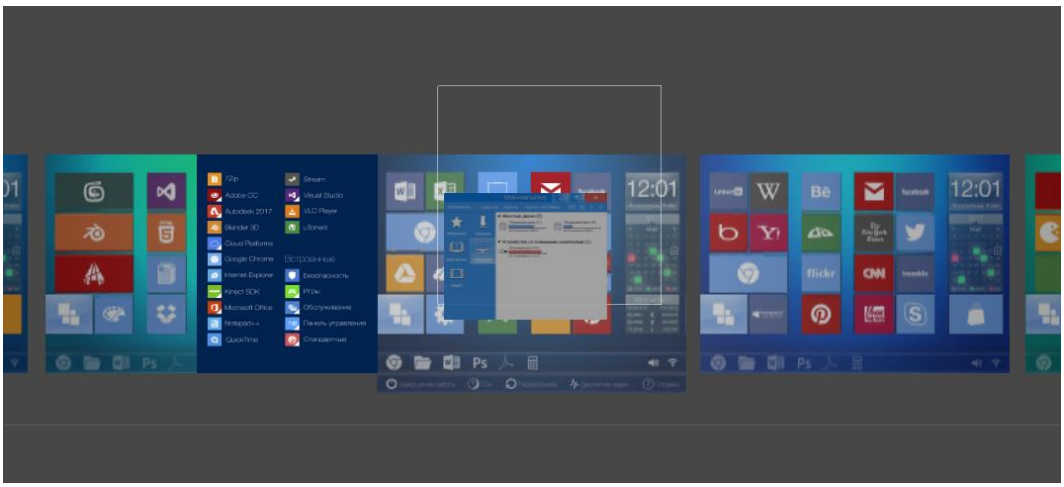

Рис. 1. Отображение рабочих столов в интерфейсе программы Unity 3D

Далее создаются элементы управления системой – кнопки на рабочем столе. Они создаются стандартным UI-примитивом *Button*, который позволяет настраивать анимацию кнопки, не вмешиваясь в программный код. В панели анимации переключаются состояния кнопки и добавляются новые параметры изменений кнопок: размеры, поворот, цвет и др. Также есть возможность с помощью дополнительных настроек функции триггера задать действия, которые будут производить нажатие на кнопку или наведение на нее курсора. Был использован параметр *SetActive()*, который позволяет убирать и восстанавливать видимость объектов сцены. Таким образом настроено появление окна программ и приложений, а также подсказки календаря.

Для реализации взаимодействия пользователя и прототипа посредством устройства Microsoft Kinect требуется подключение ряда скриптов, которые позволяют программно воспринимать жесты.

KinectGestures.cs – скрипт на языке C#, описывающий действия, которые необходимо распознавать, разбит на несколько частей. В первой описываются переменные класса, из которых наибольшее значение имеет переменная *Gestures* типа *enum*. Она используется для объявления, перечисления отдельного типа, состоящего из набора именованных констант, который называется списком перечисления. В данной переменной перечислены названия действий, которые Kinect должен распознавать. Чтобы программный код считывал действия, необходимо снимать информацию о скелете, передаваемую устройством Kinect, и преобразовывать ее в набор переменных, заключенных в переменной *GestureData*, которая описывает идентификатор пользователя, совершаемое действие, время начала действия, сустав скелета и его позицию в 3D-пространстве, и т. д.

Так как прототип интерфейса разрабатывается в качестве системы, адаптированной под людей с ограниченными возможностями здоровья (далее – ОВЗ), было принято решение не проводить идентификацию положения суставов нижней части тела, потому что многие пользователи с ограничениями в передвижении не могут долго стоять, совершая активные движения руками. Поэтому для реализации управления интерфейсов устройству не требуется отслеживать положение головы и ног, а значит, количество суставов ограничено семью элементами: левым/правым запястьем, левым/правым локтем, левым/правым плечом, шеей [5].

Для *получения значений положения суставов* и занесения их в соответствующие переменные используется метод класса *KinectWrapper*, который также не является стандартной библиотекой – его необходимо подключать вручную. В этом классе каждому из 20 доступных устройству Kinect суставов присваивается свой номер и проводится ряд операций с целью получения информации о положении каждого сустава, а также получения RGB-карты с камеры Kinect и карты глубины с инфракрасного сенсора. После выполнения всех операций в классе *KinectWrapper* продолжается просчет в классе *KinectGestures*.

Следующим шагом является *отслеживание движения*, и при условии, что жест пользователя завершен, вызывается функция *CheckPoseComplete*, в которой по завершении жеста в переменную *gestureData* записывается информация о времени завершения жеста и позиции отлеживаемого сустава в пространстве. Если пользователь по какой-либо причине прервал жест, то вызывается функция *SetGestureCancelled*, которая обнуляет переменную *gestureData*.

При совершении пользователем какого-либо действия вызывается функция *CheckForGesture*, в которой описаны условия каждого жеста из списка *Gestures*, объявленного в начале класса. Для создания прототипа интерфейса необходимо отслеживание только четырех действий: SwipeLeft, SwipeRight, SwipeDown, SwipUp, Click, Push, Pull, Wave.

Вся комплексная система методов и классов собирается в одном месте для управления и вызова функций проверки. В классе *KinectManager.cs* хранятся переменные, отвечающие за выполненное действие, положение всех отслеживаемых суставов во времени, а также информация об этих суставах и пользователях. С помощью скрипта *KinectManager* возможно распознавать два отдельных скелета, разделив их по ID пользователя, и препятствовать потере отслеживаемого скелета при появлении в кадре второго [6].

Для подключения системы распознавания жестов в игровой движок Unity необходимо просто добавить класс *KinectManager* как компонент к активной камере.

Важным элементом работы распознавания жестов является скрипт *GestureListener.cs*, который также добавляется к активной камере в качестве компонента. В самом начале этого скрипта создаются переменные типа *bool*. Эти переменные в дальнейшем используются в функциях, которые возвращают сигнал о том, что действие было завершено.

Функции *UserDetected* и *UserLost* активируют отслеживание выбранных жестов при появлении пользователя перед камерой устройства Kinect и очищают кэш при исчезновении пользователя из кадра соответственно, а функция *GestureCompleted* возвращает информацию о сделанном движении после серии проверок, выполняемых в методе *Gestures* класса *KinectGestures*.

В данном скрипте созданы глобальные переменные, необходимые для связи этого кода и объектов, находящихся на виртуальной сцене прототипа интерфейса. Реализация связи между переменными в коде и объектами на сцене представлена следующим набором строк:

```
public Texture2D cursorTexture;
public CursorMode cursorMode = CursorMode.Auto;
public Vector2 hotSpot = Vector2.zero;
private bool swipeLeft;
private bool swipeRight;
private bool swipeWave;
private bool isMoving;
public GameObject MenuP;
public GameObject All_programs;
public GameObject hello;
private string gestureName;
private MouseSwipe mouseSwipe = new MouseSwipe();
public Transform target;
public float smoothTime = 0.3F;
public Vector3 pos;
private bool t = false;
```

В данном фрагменте кода *public* является определителем режима доступа (public – глобальная переменная, private – локальная переменная). *GameObject* является типом объекта. В данном случае *GameObject* – игровой объект. Словосочетания *MenuP*, *All_programs* и др. являются названиями переменных.

Уже существующий внешний класс добавляется посредством глобальной переменной, где на месте типа переменной записывается название необходимого класса:

```
Private GestureListener gestureListener.
```

Следующие строки кода обращаются к типам переменных *Animator*. Новые версии Unity включают в себя обновленную систему анимации Mecanim (рис. 2). Ее суть состоит в том, что вся анимация объектов разделяется на различные состояния, переход между ними осуществляется согласно параметрам, которые могут изменяться посредством скриптов.

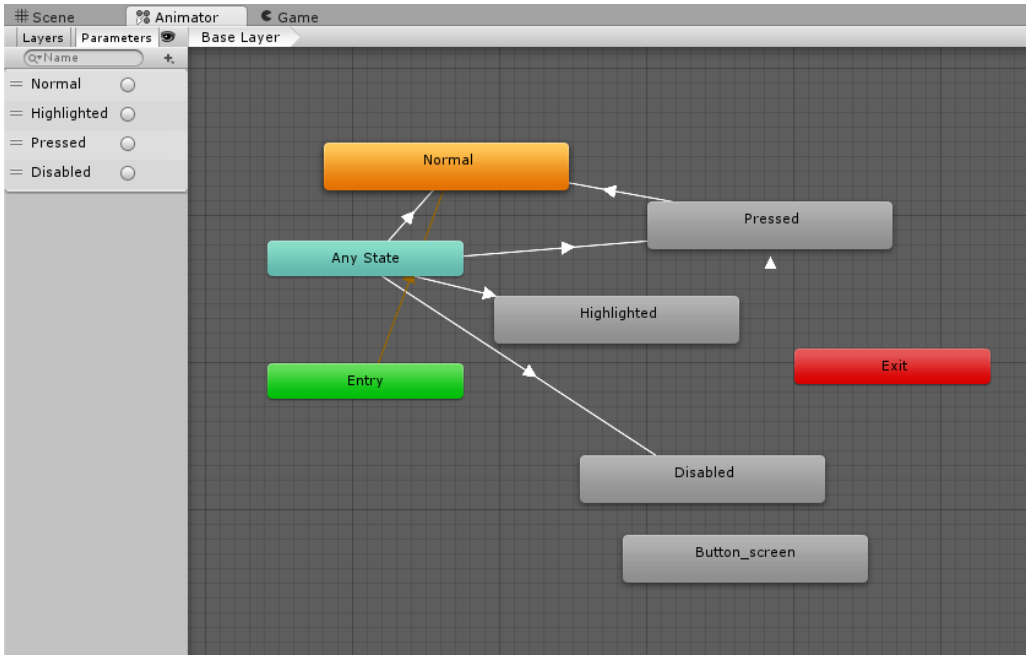

Рис. 2. Окно состояний анимации кнопки в системе Mecanim

В каждом состоянии записана определенная анимация, которая является отдельным объектом типа *Animation*. Все анимационные клипы связаны друг с другом в единый глобальный *Animation Controller* (контроллер анимации). Чтобы изменить его параметры через скрипт, нужно перевести компоненты анимации в переменные. Для доступа к состояниям анимации необходимо создание переменных типа Animator:

```
private Animator up;
private Animator programs;
```

Но сами по себе переменные не несут никакой информации об анимации объекта. Для присвоения им соответствующих контроллеров необходимо обратиться к функции *Start*, вызов которой происходит при появлении игрового объекта. В теле данной функции записываются следующие строки:

```
up = MenuP.GetComponent<Animator>();
programs = All_programs.GetComponent<Animator> ();
```

В данных строках кода переменным типа *Animator* присваиваются значения по шаблону: *Игровой_Объект.Получить_Компонент<Animator>()*. То есть программно вызываются компоненты, присоединенные к игровым объектам, имеющим определенное имя, которое указывается в первой части шаблона. Система анимации Mecanim позволяет переходить из одного состояния в другое при изменении определенных параметров, создаваемых пользователем.

Для запуска интерфейса необходимо ввести пароль в виде определенного уникального движения, установленного пользователем. Для данного прототипа необходимо помахать рукой, тем самым активировав функцию *IsWave()*. При срабатывании функции приветственное окно скрывается, предоставляя возможность дальнейшей работы с системой:

```
if (gestureName == "Wave" || gestureListener.IsWave()|| Input.GetKeyUp(KeyCode.Alpha6))
{
hello.SetActive(false);
t=true;
}
```

Дальнейшее взаимодействие с прототипом осуществляется в основном шестью жестами: взмахом руки влево, вправо, вверх, вниз и кликом. Эти жесты описаны в скрипте *KinectManager.cs* и для получения к ним доступа необходимо подключение дополнительных строк кода, служащих триггерами, по которым обработчик получает доступ к соответствующим функциям. Этим набором триггеров служит компонент активной камеры, носящий название Gesture Listener (Script).

Скрипт *GestureListener.cs* является дочерним объектом класса *KinectGestures* и берет свои основные параметры из функции *GestureListenerInterface*. Наследование параметров становится возможным благодаря строке кода, которая является строкой объявления глобального класса *GestureListener*:

```
public class GestureListener : MonoBehaviour, KinectGestures.GestureListenerInterface
```

В данной строке кода параметр *MonoBehaviour* означает, что для работы класса необходим материнский объект. То есть описанный класс не может работать, не будучи привязанным к какому-либо объекту [7].

Тело класса описывает ряд функций, возвращающих значение булевого типа. Одна из таких функций – *IsSwipeLeft():*

```
public bool IsSwipeLeft()
{
if(swipeLeft)
{
Debug.Log(«Swipe Left!»);
swipeLeft = false;
return true;
```

```
}
 return false;
}
```

Данная функция глобального типа может возвращать значения либо «истина», либо «ложь». Для выбора одного из этих значений происходит проверка переменной *SwipeLeft* булевого типа. При значении true (истина) сперва передается сообщение «Swipe Left!» в консоль отладки. Затем переменной *SwipeLeft* присваивается значение false (ложь) и в итоге возвращается значение true (истина) для всей функции. Если при инициализации функции значение переменной *SwipeLeft* было равно *false* (ложь), то сразу присваивается значение *false* (ложь) для всей функции.

После описания функций обработки жестов необходим ряд функций для мониторинга информации, передаваемой устройством Kinect. И первой является функция *UserDetected*, которая инициализируется при появлении пользователя перед устройством. Работа данной функции заключается в создании списка движений специально для распознанного пользователя, который сохраняется в переменной manager, принадлежащей классу *KinectManager*. Дальнейшим действием является выбор необходимых жестов, подлежащих распознаванию устройством Kinect.

После завершения жеста инициализируется функция *GestureCompleted*, которая описывает действия, принимаемые обработчиком при распознавании одного из жестов, описанных в переменной *manager*. Если жест совпадает с одним из описанных, то соответствующей переменной булевого типа присваивается значение *true* (истина) и управление передается обработчикам, описанным в начале класса.

Для прямого взаимодействия с основным скриптом, описывающим поведение программы при распознавании жестов, необходимо записывать информацию о срабатывании триггеров в специальную глобальную переменную, носящую название *GestureListener*. Присвоение данной переменной необходимого значения осуществляется с помощью пользовательского интерфейса программы Unity 5. Достаточно просто перенести мышью необходимый класс в соответствующий раздел окна *Inspector*.

После подключения всех необходимых классов стало возможным простое описание необходимых движений, заключающееся в вызове соответствующей функции обработчика класса *GestureListener.cs*, но ввиду того, что взаимодействие с интерактивным прототипом может осуществляться не только жестами, но и мышью (при отсутствии возможности подключения устройства Kinect), возникла необходимость создания отдельного класса, описывающего жесты, совершаемые с помощью мыши, которые соответствуют жестам, распознаваемым в классе *KinectManager* [8].

Класс *MouseSwipe.cs* состоит из двух блоков. Первый – описание переменных, необходимых для работы класса:

```
Vector2 firstPressPos;
Vector2 secondPressPos;
Vector2 currentSwipe;
private enum gestState{None, Up, Down, Left, Right, Click};
private int gestureId;
```

Все пять переменных – локальные, так как значения, записанные в них, необходимы только внутри класса.

Три из пяти переменных являются переменными типа *Vector 2*, который хранит в себе координаты по двум осям. В случае скрипта *MouseSwipe.cs* необходимыми являются переменные, содержащие в себе значение положения мыши при нажатии на левую кнопку мыши (*firstPressPos*), значение положения мыши при завершении жеста (*seconsPressPos*) и нормализованный вектор (*currentSwipe*). Нормализованным вектором является проекция на оси X и Y, построенная между двумя точками прямой.

Нормализованный вектор вычисляется путем вычитания начальной позиции курсора мыши из конечной:

currentSwipe = new Vector2(secondPressPos.x-firstPressPos.x, secondPress-Pos.y – firstPressPos.y);

После нормализации вектора проверяется ряд условий, описывающих все возможные направления вектора, а именно: вверх, вниз, влево и вправо. При совпадении направления вектора с одним из условий функция возвращает значения типа *string* (строка), содержащее в себе название жеста. В ином случае возвращается значение *null* (нуль) [9].

Классы *KinectManager.cs, GestureListener.cs* и *MouseSwipe.cs* позволяют описать необходимые условия для взаимодействия с прототипом.

Обращение происходит в теле функции *Update(),* которая инициализирует каждый кадр при запуске приложения. Прописанное условие отслеживания взмаха правой руки для дальнейшего описания действий выглядит следующим образом:

```
if(gestureName == «Left» || gestureListener.IsSwipeLeft())
{
...
}
```

Данные строки кода описывают условие: если жест мыши является жестом влево либо жест, распознанный устройством Kinect, является взмахом влево, – выполнять строки кода, заключенные в теле этого условия. Описание остальных трех жестов реализовано по тому же принципу.

Данные строки прописаны внутри тела условий, описывающих отслеживание жестов взмаха рукой влево, вправо, вверх, вниз, а также жесты удержания руки на месте (клик), направления кисти на себя, от себя и перемещения ладони в приветственном жесте.

Проверка параметров и дополнительная настройка отслеживания движений необходимы по причине того, что данный жест должен быть доступен для распознавания не на всех разделах интерфейса. Для его ограничения в данном случае достаточно перечислить сцены, в которых инициализация функции *isPush()* и *IsPull()* была бы ошибочной. В данном случае это параметры появления новых элементов системы – окна папки изображений.

Последним необходимым компонентом управления прототипом интерфейса является контроль над курсором мыши. Ввиду того, что пользователь будет находиться на расстоянии 2–3 метров от экрана, у него не будет возможности воспользоваться мышью или клавиатурой. Поэтому следует привязать курсор мыши к суставу правой или левой руки виртуального скелета.

Скрипт *KinectManager.cs* содержит условия, позволяющие управлять курсором мыши без создания дополнительных скриптов. Для активации режима контроля курсора необходимо изменить значение параметра активной камеры *ControlMouseCursor* в окне *Inspector*. Для отображения курсора на экране необходимо создать игровой объект, который также прикрепляется к соответствующему параметру в окне настроек скрипта *KinectManager.cs*.

После активации всех необходимых параметров в прототипе интерфейса появляется возможность управления курсором без использования компьютерной мыши.

Активация четырех жестов и возможность контроля курсора в скрипте *GestureListener.cs* позволили создать все необходимые условия для полноценного контроля над интерфейсом. Этот факт позволяет утверждать, что бесконтактный интерфейс настолько адаптивен, что может использоваться в качестве основного ПО как людьми с ОВЗ, так и пользователями без ограничений моторики.

Работоспособность проекта можно проверить в ходе разработки в самом игровом движке Unity3D, так как он полностью интегрирован в среду разработки. Для этого можно использовать кнопки Run и Pause, что позволяет, во-первых, увидеть критические ошибки, которые не допускают запуск проекта, а во-вторых, проверить распределение производи-

тельности через встроенный Profiler (профайлер). Кроме всего прочего, профайлер позволяет измерять время, затраченное на выполнение того или иного процесса, отдельных процедур, функций, модулей и т. д. (рис. 3) [10].

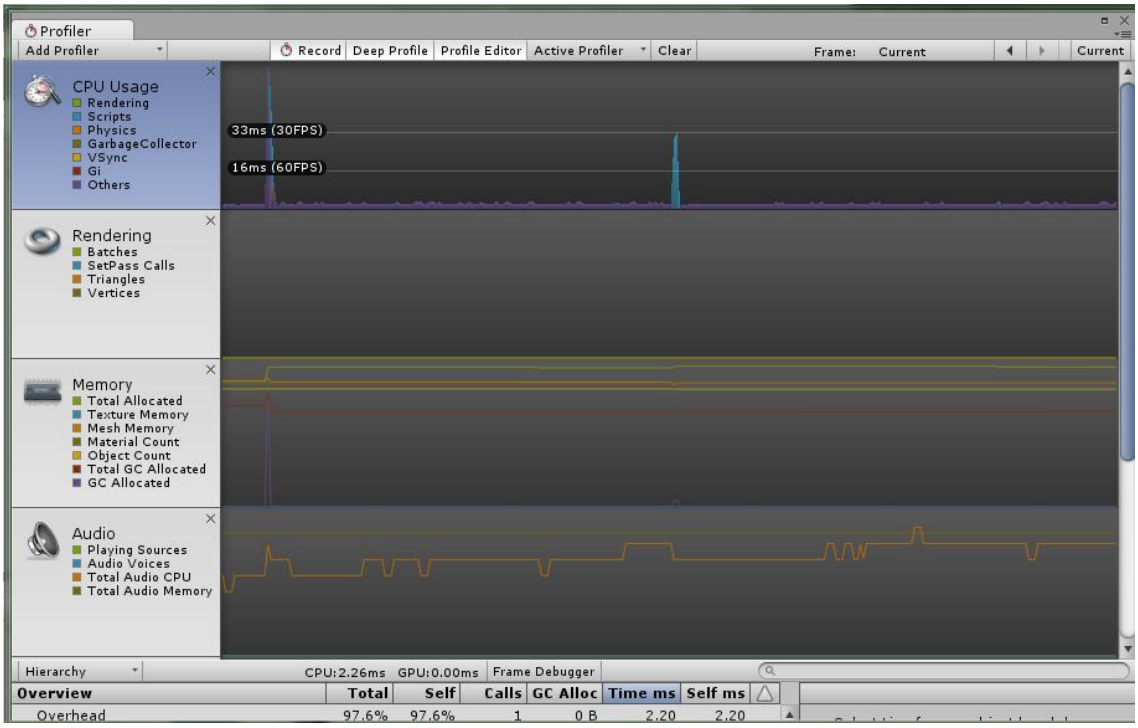

Рис. 3. Окно Profiler

Прототип бесконтактного интерфейса является полноценным приложением и, благодаря реализованному управлению с использованием мыши наравне с устройством Kinect, может быть установлен и запущен как пользователями без ограничений здоровья, так и людьми с ОВЗ.

Разработанный продукт пригоден для эксплуатации как электронный ресурс-приложение с расширением .exe, который может использоваться как людьми с ОВЗ, так и пользователями без ограничений моторики. Он может быть запущен в одной из трех операционных систем: Windows, MacOS или Linux. Таким образом, прототип интерфейса с бесконтактным взаимодействием является кроссплатформенным.

Литература

1. Зенг В. А. Формирование базового словаря жестов для естественного компьютерного бесконтактного интерфейса // Вестн. НГУ. Сер. Информ. технологии. 2018. Т. 16, № 3. С. 105–112.

2. Horton W. K. The icon book: visual symbols for computer systems and documentation. New York: J. Wiley, 2016. 417 p.

3. Кораблев Д. А. Выбор и обоснование показателя эффективности элементов экранных интерфейсов систем электронного документооборота // Сб. тезисов VII конф. молодых ученых. Вып. № 1. Информ. технологии. СПб. : СПб ГУ ИТМО, 2014. С. 112–119.

4. Zhang Z. Microsoft Kinect Sensor and its Effect // IEEE Computer Society. 2012. Vol. 19, No. 2. P. 4–12.

5. Smisek J., Jancosek M., Pajdla T. 3D with Kinect // Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on, 2011. P. 1154–1160.

6. El-laithy R., Jidong H., Yeh M. Study on the use of Microsoft Kinect for Robotics Applications // Position Location and Navigation Symposium (PLANS), 2017 IEEE/ION, 2017. P. 1280–1288.

7. Hodges S., Freeman D., Hilliges O., Molyneaux D., Newcombe R., Shotton J. KinectFusion: Real-time 3D Reconstruction and Interaction Using a Moving Depth Camera // UIST'16 Proceedings of the 29th Annual ACM Symposium on User Interface Software and Technology, 2016. P. 559–568.

8. Xia L., Chen C., Agarwal J. Human Detection Using Depth Information by Kinect // Computer Vision and Pattern Recognition Workshops (CVPRW), 2015 IEEE Computer Society Conference on, 2015. P. 15–22.

9. Варфел. Т. З. Прототипирование. М. : Манн, Иванов и Фербер, 2013. 240 с.

10. Быковский В. П. Моделирование прототипа интерфейса // Модели систем распределения информации и их анализ. М., 2012. С. 101–112.