Ninth Hungarian Conference on Computer Graphics and Geometry, Budapest, 2018

# Using the Kinect body tracking in virtual reality applications

Tamás Umenhoffer[1], Balázs Tóth[1]

[1] Department of Control Engineering and Informatics, Budapest University of Technology and Economics, Budapest, Hungary

**Abstract**

*In this paper we introduce a virtual reality room setup using commonly available, moderately expensive devices. We implement head position tracking with a Microsoft Kinect V2 sensor, and use an Android device with gyroscope to track user head rotation and to display the virtual world. Our workstation which handles the Kinect can also insert the point cloud of the user in the virtual world and can inspect its interaction in real time.*

## 1. Introduction

Virtual reality systems need special devices, which can be rather expensive. Immersive environments need good head position and orientation tracking, while interaction with the environment can also require three dimensional tracking of an input device, or the users hand. Recently virtual reality devices has gone through a great evolution, several solutions exists that makes virtual reality available for everyday users. However the most robust solutions are usually still expensive.

The most popular devices nowadays are HTC Vive [5], Oculus Rift [7], Sony Playstation VR [8], Samsung Gear VR [6], Google Daydream [11] and Google Cardboard [10]. These devices provide different services in different platforms and for different prices. The most expensive device is the HTC Vive, which is a stereo headmounted display equipped with rotational sensors and a camera in the headset. It also has two controllers and two tracking cameras. The active infrared tracking system enables precise positional and rotational tracking of the users head and the two controllers. The tracking area the user can move in is around 5m x 5m. The head mounted LED display has 2160x1200 resolution. The VR application should be developed on a desktop, typically on Windows platform, and the headset is connected directly with wire to the graphics card. The Oculus Rift has similar features, however it only has a single infrared camera for tracking, thus head movement is limited in a roughly 1 meter diameter sphere, and the user should always face the tracking camera. It has a display with the same resolution as the Vive, and the development is also similar: the VR application is developed on desktop platform, and the headset is connected to the graphics card. The two device have their own programing API's. The biggest advantage of the Vive over Oculus is the free navigation it provides: the user can walk and turn around within the tracked area freely. Sony Playstation VR must be connected to a Playstation console. It has a lower resolution 1920x1080 display, it uses the Playstation stereo cameras with active LEDs for headtracking. The Playstation uses the same method to track the position of two controllers in the user's hand. The rotation of the input devices is also tracked with sensors. Just in case of the Oculus Rift the tracked area is limited, and the user should face the tracking cameras. Applications should be developed on the Playstation platform, which is a popular gaming platform, though limits to usage of the device to these consols.

The Google Cardboard is a cheap VR alternative. It is a simple paper frame with two lenses that can project the two half of a smartphone display to the two eyes. It is not limited to any particular device type or platform. In its pure form it is only a stereo display. For tracking a gyroscope should be present in the smartphone. Though this enables rotational tracking only, this gives a great plus to immersion. Head position tracking is not possible, unless the developers add some marker based tracking using the rear camera of the smart phone. Typically the Android platform is used for development, but as it is only a frame, other mobile platforms can also be used. Google Daydream gives a more comfortable, plastic and fabric frame and two remote controllers are also provided. Head position and controller position still not tracked only rotations. The resolution and performance is limited by the handheld device, however it gives a very flexible and widely available solution. Samsung Gear VR uses similar concepts as the Google alternatives, but it targets Samsung Galaxy smartphones only, thus can assume that a

gyroscope and proper hardware is present in the device. It also provides a bluetooth motion sensing controller with a touchpad. Its software API is based on the Oculus platform.

From the above it follows that cheaper solutions have limited tracking capabilities, and performance is also limited by the handheld devices. More advanced solutions can have their own platform (like the Playstation), which can be serious limitation for complex application systems, or they need a high performance desktop computer, and their price is high.

In this paper we would like to extend the capabilities of simpler and cheaper solutions with head tracking using moderate price tracking sensors. We target applications where a desktop PC or notebook is also used as an inspecting machine, thus the user is also visualized in the virtual environment from a free perspective. As a desktop is present, we can use it for tracking head and hand positions and use it as a tracker server, while the handheld device is acting as a tracker client. Head rotation tracking, running the VR application and stereo rendering is performed on the handheld device just as in case of the Google and GearVR solutions.

We use a Kinect V2 connected to the desktop PC for tracking. For head rotation tracking a gyroscope is a must, but we did not have a smartphone with such capabilities. On the other hand we had an Epson Moverio BT-200 [9] smart glasses, which can also be used as a VR headset. When used as a VR headset this device does not provide any additional capabilities over a smart phone with a VR frame and rotation sensor, thus the setup and usability described in this paper also holds for headmounted smartphones with gyroscope.

## 2. Hardware

Our system needs the following main components: a workstation, that is connected to a Kinect V2 device and a rendering device connected to a head mounted stereo display. For the workstation we need USB 3.0 and Windows 10 for the Kinect sensor to operate on, so a PC or notebook should be used. The head mounted device can be a mobile used with a VR viewer like Google cardboard. However the mobile device should have a gyroscope to track head rotation. We used a special head mounted device: the Epson Moverio BT-200 smart glasses.

The Epson Moverio BT-200 smart glasses is basically an augmented reality device, but can also be used for virtual reality applications too if the glasses are shaded. These glasses has their own plastic shades for this purpose. The Moverio glasses has two main components: the glasses wire connected to a handheld device. The device can be seen on Figure 1. This device is similar to a smartphone, but it does not have an LCD screen, as it uses the glasses for display. In the place of the LCD screen a touch pad was built in to provide an easy to use input device. The device itself runs Android operating system, and has similar capabilities as other smart

**Figure 1:** *The Epson Moverio BT-200 smart glasses.*

phones. It does not have cellular network interface, but supports WiFi, bluetooth, has acceleration and magnetic sensors and a gyroscope and a GPS.

The glasses have two see-through display lenses, each have 960x540 resolution. The accelerometer, magnetometer and gyroscope was also built into the glasses too, and also a VGA camera next to the right eye. The lenses are directly connected to the handheld device. Each display lenses have an approximate 23 degree field of view. This angle is sufficient for augmented reality applications where the glasses are used in a see through mode, but can be rather small in case of virtual reality usage. A headphone can also be attached to the device which we did not use.

The Kinect one sensor has a high resolution (1920x1080) color camera and a lower resolution (512x424) depth camera. The device is shown on Figure 2. Both cameras has rather big filed of view: 84.1 degree horizontal for the RGB and 70.6 degree for the depth camera. It also has a microphone array which we did not use in our system. The depth range of the depth sensor is between 0.5 and 8 meters, but accuracy decreases drastically beyond 4 meters.

**Figure 2:** *The Kinect V2 sensor.*

We prepared a VR room in the following way: we covered a wall with green fabric to ease background removal.

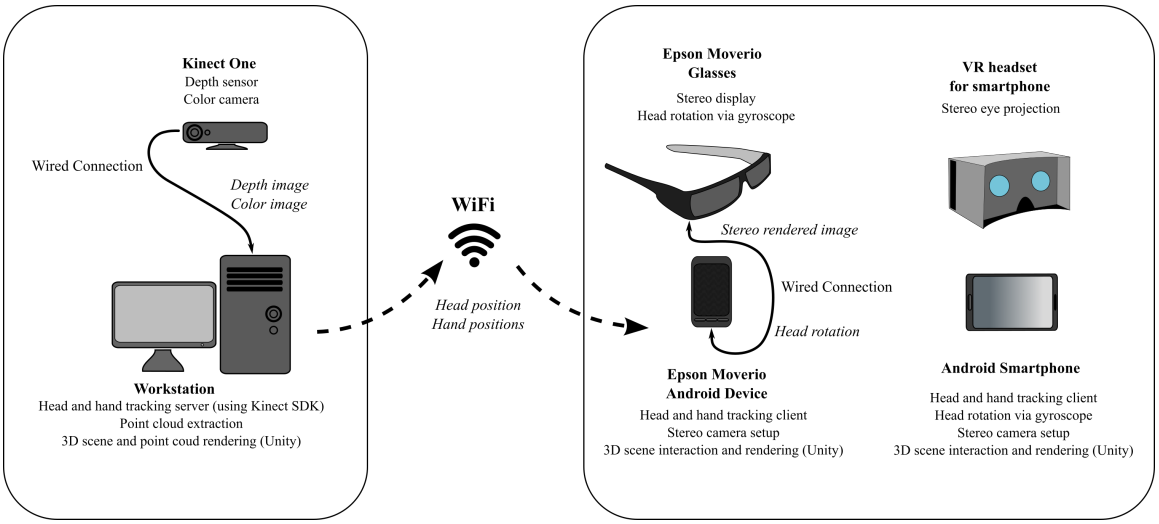


**Figure 3:** *Our VR system.*

We placed the Kinect sensor at 2.5 meters from the wall. The Kinect cameras were rotated to have a horizontal view direction perpendicular to the wall. The user can explore the virtual environment between the sensor and the wall using the smart glasses. The head movement of the user will be tracked by the Kinect, so the user can move in a trapezoid shaped area limited by the Kinect's nearest depth range and field of view. Figure 4 shows our VR room setup.

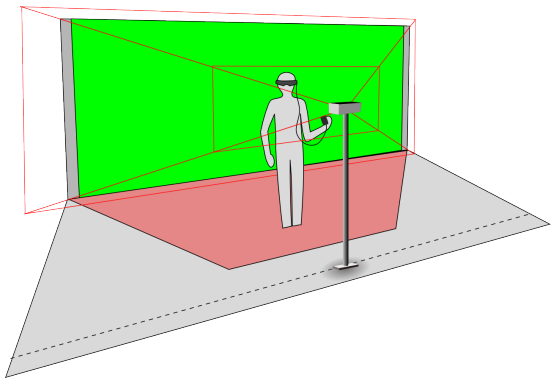

**Figure 4:** *Our VR studio setup.*

## 3. System overview

Our system is shown in Figure 3. A Kinect V2 device is connected to a workstation machine. The purpose of this machine is to use the Kinect SDK to track the user's head and hand position. These position values are sent to the Epson Moverio handheld device over a wireless network. The handheld device acts as a client and receives the tracked data in each frame. It also reads head rotation values from the gyroscope located in the glasses. The glasses are directly connected with the device thus provide high speed rotation value updates.

Knowing the head rotation and position, an application using 3D accelerated graphics can render the virtual world in realtime form the current viewpoint of the user. This rendering uses a stereo camera setup to enable proper depth perception. Attaching interactive game elements to the hand positions also enables the user to interact with the virtual world. The Epson Moverio can be replaced with any Android smart phone that has a gyroscope and WiFi connection, and can be placed in a VR headset. The simplest and cheapest solution is using Google Cardboard.

To inspect the user interacting with the virtual world, the workstation machine can display the same virtual virtual world from a free perspective. This desktop application also receives head and hand position data, and can visualize these locations in space. Using the Kinect sensor's depth and color data, a three dimensional point cloud can also be built and placed into the virtual world. The following sections describe these steps in more details.

## 4. Tracking using the Kinect V2

The Kinect V2 device can stream an RGB, an infrared and a depth image to the PC it is connected to via a high speed USB 3 port. These streams are accessible with the help of the Kinect API available for multiple programming languages. This API not only provides these streams to the programmer but has several additional advanced features. V2 feature is the tracking of the user's body.

The body tracking in the Kinect is an image processing

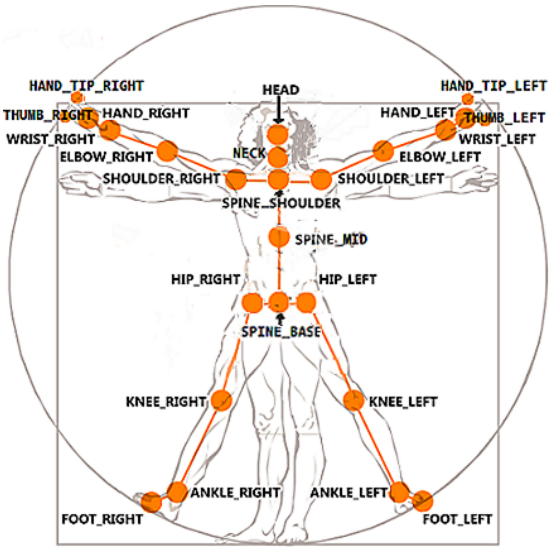


**Figure 5:** *The tracked joints and their names in the Kinect V2 platform.*

based approach, where the algorithm searches for a human-like shape and identifies its main body parts. The API can track multiple bodies at once and stores a skeleton for each of them. A skeleton is a set of connected joint positions. The Kinect V2 uses a skeleton with 25 joints and can track at most 6 bodies. Figure 5 shows the joints tracked by the Kinect V2 sensor. In our case we are only interested in the position of three joints: head, left hand and right hand. These positions are given in the coordinate system of the sensor, where the origo is in the nodal point of the infrared camera, the $y$ axis points upward and the $z$ axis points in the view direction of the infrared camera.

As the tracking is image processing based, in a first step the user's body should be identified. The tracker can easily do this if the the user turns toward the sensor, its hands are lifted aside from its body, and the ellipse of the head is clearly seen. If the user turns sideways, or the hands are occluded, tracking can become unstable. The torso of the user is usually well tracked, however the limbs can be uncertain in many situations. The tracker also stores a reliability value to each of the joints, which describes the estimated accuracy of the tracking of the given joint in the current frame. If a low reliability value is given we can neglect the tracked position for the joint as it can contain invalid values. The tracking does not searches for facial features so the user can turn away from the camera, head position tracking will probably not fail. However hand occlusion is more likely to happen in this case.

We implemented the tracking in a standalone application written in C++. This application will be responsible for point cloud extraction too, which will be described in section 6.

We constantly read tracked joint positions from the Kinect API and send it to the render devices over network. We used a connectionless UDP stream to transfer data, as tracking data loss is not a great problem unlike slow connection, which results serious lags in the system. We should also make sure that all packets are transferred immediately and not buffered by the network driver of the operating system, as buffering would result in periodical stalls on the client side. As handheld devices usually provide wireless networking capabilities we set up a local wireless network between the workstation and the mobile device.

### 4.1. Improving the tracking accuracy

As the tracking of the user is a key part of our system we applied postprocessing both on the tracker and the renderer side to improve the user experience. In the tracker component we apply two type of filtering. We identify when the head and hand joint positions are unreliable based on the tracking states reported by the sensor. The unreliable measurements are excluded from further processing. The joint positions returned by the sensor are accurate in the sense, that when the user stands still the average of the measured joints over time is close to the real position of the tracked body parts. However, the distinct position samples are scattered in a centimeter range. To eliminate this uncertainty in the tracking position, which cause a small but noticeable shaking in our VR environment, we apply a jitter removal filter. This filter attempts to smooth out the sudden changes of the joint positions by limiting the changes allowed in each frame as

$$\hat{X}_n = \begin{cases} X_n, & if |X_n - \hat{X}_{n-1}| < d \\ \alpha X_n + (1-\alpha)\hat{X}_{n-1}, & otherwise \end{cases}$$

where $X$ and $\hat{X}$ are the measured and smoothed joint positions respectively, $d$ is a threshold that should be less than the typical jump distance of the input data and $\alpha$ should be chosen to minimize the input lag. These filtered joint positions are transmitted to the renderers. As the communication between the tracker and the renderers is inherently unreliable we apply further processing of the received joint positions.

To improve the user experience we should produce a hand position in every frame. However, the reliability of the tracking and the transmission between the components does not meet this requirement. Therefore we could only predict the frame by frame positions with additional filtering [2]. The standard approach would be a variation of Kalman filter, but the resource constraints of our handheld device does not allow it. A feasible approach is a double moving averaging [3] (DMA) filter, which can be tuned to be responsive but also predicts the lacking samples well enough for our purposes. The DMA filter tracks the first and second order moving averages of the input positions and produces locally smoothed

output positions:

$$MA_n^{'} = \frac{1}{N+1}\sum_{i=0}^{N} X_{n-i} MA_n^{''} = \frac{1}{N+1}\sum_{i=0}^{N} MA_n^{'}$$

, where $MA^{'}$ and $MA^{''}$ are the first and second order moving averages, and

$$\hat{X}_n = MA_n^{'} + (MA_n^{'} - MA_n^{''})$$

is the smoothed position. To predict the lacking position we adjust the trends according to the number of missing input frames as

$$\hat{X}_{n+k|n} = \hat{X}_n + \frac{2k}{N-1}(MA_n^{'} - MA_n^{''}).$$

This filtering approach is resource friendly and also successfully eliminates the small gaps in the tracking. Finally when the tracked skeleton is completely lost we reset the filter.

## 5. Virtual world rendering

The rendering of the virtual world is performed on the handheld device. The VR application acts as a client and reads tracking messages continuously over the wireless network. The head rotation is defined by the rotation sensor of the device, which can be easily accessed through the device platform API. We should give special attention of two things: the coordinate space of the tracker and the basic rotation of the device. In our case the device returned identity rotation for the head if the glasses were looking straight downwards. Thus when starting the application we orient the user according to the coordinate system of the Kinect sensor, namely we place the user right in front of the infrared camera and orient its head to look horizontally in the direction of the camera (in the camera $z$ axis). This is the rotation sensor calibrating step, where the head orientation returned by the sensor in the calibration pose is stored and rotation values are corrected with this orientation in each frame.

In practice looking exactly horizontally is not a straightforward thing to ask from the user, but we found that only two rotations are needed to correct orientations. The first is a rotation that rotates the glasses from vertical view direction to horizontal view direction, which is a 90 degree rotation around its $x$ axis. The second rotation is a rotation around the up axis which can be determined by turning toward the Kinect camera if the user is right in front of the Kinect sensor. As we have head tracking data, the user can move until head track data returns zero for head $x$ position. This means that the user stands right in front of the camera, and after that we should rotate the head to look straight to the camera. We also rendered a sphere in the position of the Kinect sensor in the virtual world for a visual feedback of the quality of calibration. When using the Moverio glasses in see through mode we can see the real world sensor and the virtual sphere at the same time, if their positions match, the calibration was successful. This is basically the extrinsic calibration of the headmounted display. We did not need to calibrate intrinsic parameters, field of view in particular, as it is known for the Moverio device.

Virtual cameras are set up using the tracked camera position and rotation. As we need two separate rendering for the two eyes, we need to create two cameras with a slight horizontal offset. We referred to the average human interpupillary distance (IPD), which has a 63 mm mean value (comparison of multiple databases about IPD can be found in the work of Dodgson [4]). The image of the two eyes should be rendered side by side, this holds both for Google Cardboard like devices and for the Moverio glasses too.

Beside head position data, tracked hand positions are also read in each frame. We can use these positions to interact with the environment. They can serve as a three dimensional pointer, or we can attach virtual objects to them. Attaching a virtual object two the two hands also shows calibration errors as they should be located in the virtual world exactly where the users palms would be located.

## 6. Inspecting the user in the virtual world

In many VR applications there is a need to inspect the user interacting with the virtual world. In such systems the camera view seen by the user is also displayed to an external monitor, so others not interacting in the virtual world can see what the user does at the moment. This gives only a first person view perspective, in some cases the scene should be investigated from an other viewpoint.

We can render the scene from arbitrary viewpoint, however, the user will not be visible in the virtual world. Some high end application use full body motion capture and apply this motion to a virtual avatar. Other systems use camera arrays and reconstruct a point cloud of the user in real time, and mixes this point cloud with the virtual environment. These systems can be rather expensive.

We used the depth and color image of the Kinect sensor to reconstruct a point cloud of the user in real time without any additional hardware. The workstation that handles head and hand tracking also processes depth data, reconstructs the point cloud and renders the scene from an arbitrary viewpoint. To do this, first we should separate the user from its environment. The Kinect API provides us a special mask image, that contains ID values for the bodies it tracks. Thus for each body we have a binary mask. This mask is not necessarily precise enough, thats why we covered the wall behind the user with green fabric, which increases foreground mask quality.

If we know the projection matrix of this camera, we can find the camera space position for each pixel using the pixel coordinates and its depth value. The projection matrix is known, as we know the view angle and the aspect ratio of the camera. Near and far plane values does not affect the final camera space positions, so we can choose basically any

valid values for them. The formulas are simple and have the following form:

$$C_n = M_{proj}^{-1} \cdot (H.xy, -1, 1)$$

$$C_n = \frac{C_n}{C_n.w}$$

$$C = \frac{C_n \cdot Z}{n}$$

For each pixel of the depth image we construct homogeneous coordinates from the device space pixel coordinates $H.xy$ by setting the z coordinate to -1 (which stands for the near plane in case of an OpenGL projection matrix) and the w coordinate to 1. Then we multiply this with the inverse of the projection matrix of the Kinect depth camera. This will lead to a camera space point on the near plane ($C_n$) seen from the given pixel. We should make a homogeneous division, and finally the camera space position $C$ is the near plane position multiplied by the ratio of the measured depth $Z$ and near plane distance $n$. The final camera space position can be written to a new image called geometry buffer. Each pixel of the geometry buffer defines a point in camera space, thus the buffer defines a point cloud. We can visualize this by rendering a point primitive for each pixel. The color values of these points can be read from the RGB camera of the Kinect sensor.

Rendering the point cloud in the virtual environment we can see the user navigating in three dimensional space, we can see its body motions and even facial expressions. Of course this point cloud only samples the nearest surface points from the depth sensor, but this is usually sufficient for inspection.

## 7. Results

We prepared a virtual test scene: a room with pillars, a three dimensional character, and some interactive elements. We implemented our VR application in Unity [1], which provides a comfortable multi-platform solution, thus our PC and Android applications could use the same project. On the workstation side we separated the virtual world rendering tasks used for inspection and the Kinect handling tasks and implemented them in separate applications. The tracking application was implemented in C++, its purpose is to read the tracked locations and broadcast them over the local network. The broadcasting is needed as not only the handheld device, but the inspecting application also uses these data. Kinect color and depth stream processing, foreground extraction and point cloud generation was also implemented in the tracker server and the point cloud was also streamed through the network to the inspecting application. In practice this application was run on the same machine as the tracker, so a fast loopback communication could be achieved. Though this setup would enable rendering the point cloud on the handheld device too, due to performace reasons we disabled this function.

We attached two colored virtual cubes to the hand positions, they were rendered in both on handheld and on desktop side. Thus both the user and the inspectors had a visual feedback of where the user's virtual palms are. We placed interactive elements like a ball attached to an invisible point in space by a spring. When the user hits the ball with the virtual cubes, physics simulation is used to swing the ball. Figure 6 shows screen captures of our test scene from two viewpoints: on the right the Kinect camera's viewpoint was used, while on the left an arbitrary viewpoint was used. Note that the point cloud only approximates geometry from the Kinect's point of view, but still helps a lot to locate the user in virtual space using other viewpoints too.

During our tests we found that head and hand tracking can be achieved without disturbing lags. Point cloud can not be streamed directly over the wireless network in real time without simplification and some compression. That is why we did not rendered the point cloud on the handheld device, though it could have provided valuable feedback to the user. Unfortunately reliable head tracking was limited in a roughly two square meter area. On the other hand as the Kinect has high field of view the user could crouch or jump safely. The user can also walk around a virtual object, head tracking is usually not lost even when the user back faces the camera, but in these situations hand tracking is lost or unreliable. These experiences show that our system provides similar tracking range as the Oculus and even the Playstation VR, but it is much smaller than the Vive's tracking range.

The small field of view of the Moverio glasses is a serious limitation, which can be eased with using a larger sized smart phone with VR headset. On the other hand even with shading the Moverio glasses the real world environment is still slightly visible, which is basically bad for immersion, but also makes the process more safe. The user always have a sense of the real world, and will not collide with real world obstacles by accident. The resolution is comparable with the Playstation VR's resolution, it is restricted by the smart phone being used, and is smaller that in case of Oculus and Vive.

Our system can track hand positions, but no orientations, while Oculus, Vive and Playstation VR can track both. Hand position tracking has the same limitations as in case of Oculus and Playstation VR, namely, if the hands are occluded (even by each other), they cannot be tracked. As all consumer approaches use controllers for hand tracking, they could place input buttons on these controllers too. In our case, no buttons are available. On the other hand, our system does not require any batteries and external tools. The head mounted display also works wirelessly, unlike Oculus or Vive (though an extension can be purchased to Vive, which replaces wired connection with wireless). The com-

plexity of applications developed for our system is limited by the handheld device.

## 8. Future work

Our system can be extended in several aspects. Tracking multiple users does not need significant changes in our system, as the Kinect V2 can track up to six bodies. Introducing several users in the same virtual world would definitely require the visualization of the users on the handheld devices too. Point cloud rendering would need the real time simplification and compression of the point cloud. An other solution would be to use virtual avatars for the users, which need full body tracking. As the Kinect sensor tracks all joint positions for multiple bodies, these locations can also be streamed through the network to the clients.

We also plan to modify the system to move all rendering to the desktop side, and send rendered image data to the handheld device. This would make the rendering process independent of the limited performance of the handheld device, thus we could use more complex scenes, materials and lighting. This would drastically improve user experience. In this case the handheld device should operate as a head tracking server too.

## Acknowledgements

The work was created in commission of the National University of Public Service under the priority project KÖFOP-2.1.2-VEKOP-15-2016-00001 titled „Public Service Development Establishing Good Governance" in the Ludovika Workshop 2017/162 BME-VIK „Smart City – Smart Government"

## References

1. Unity 3D. https://unity3d.com/.
2. Michael Adjeisah, Yi Yang, and Lian Li. Joint filtering: Enhancing gesture and mouse movement in microsoft kinect application. In *12th International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2015, Zhangjiajie, China, August 15-17, 2015*, pages 2528–2532, 2015.
3. R. G. Brown. *Smoothing, forecasting and prediction of discrete time series*. Englewood Cliffs, New Jersey: Prentice Hall, 1963.
4. Neil A. Dodgson. Variation and extrema of human interpupillary distance. *Proc.SPIE*, 5291:5291 – 5291 – 11, 2004.
5. Dante D'Orazio and Vlad Savov. Valve's vr headset is called the vive and it's made by htc. https://www.theverge.com/2015/3/1/8127445/htc-vive-valve-vr-headset, March 2015.
6. Cameron Faulkner. Samsung gear vr review. https://www.techradar.com/reviews/samsung-gear-vr-2017, January 2018.
7. Nick Pino. Oculus rift review. https://www.techradar.com/reviews/gaming/gaming-accessories/oculus-rift-1123963/review, January 2018.
8. Nick Pino. Playstation vr review. https://www.techradar.com/reviews/gaming/playstation-vr-1235379/review, January 2018.
9. Lily Prasuethsut. Epson moverio bt-200 review. https://www.techradar.com/reviews/gadgets/epson-moverio-bt-200-1212846/review, April 2015.
10. Sean Riley. Google cardboard review: Better than nothing. https://www.tomsguide.com/us/google-cardboard,review-4207.html, February 2017.
11. Matt Swider. Google daydream view review. https://www.techradar.com/reviews/google-daydream-view-review, October 2017.

**Figure 6:** *Users using our system to interact with the virtual world.*