View-dependent 3D Projection using Depth-Image-based Head Tracking

Jens Garstka
smartoptics GmbH, Bochum

Gabriele Peters
FernUniversität in Hagen

View-dependent 3D Projection using Depth-Image-based Head Tracking

Jens Garstka
University of Hagen
Chair of Human-Computer-Interaction
Universitätsstr. 1, 58097 Hagen, Germany
jens.garstka@fernuni-hagen.de

Gabriele Peters
University of Hagen
Chair of Human-Computer-Interaction
Universitätsstr. 1, 58097 Hagen, Germany
gabriele.peters@fernuni-hagen.de

Abstract

The idea of augmenting our physical reality with virtual entities in an unobtrusive, embedded way, i. e., without the necessity of wearing hardware or using expensive display devices is exciting. Due to the development of new inexpensive depth camera systems an implementation of this idea became affordable for everyone. This paper demonstrates an approach to capture and track a head using a depth camera. From its position in space we will calculate a view-dependent 3D projection of a scene which can be projected on walls, on tables, or on any other type of flat surface. Our projector-camera system enables the viewer to explore the projected scene while walking around.

1. Introduction

In this paper we discuss a projection method, which enables a single person to view a projection of a three-dimensional scene. For the remains of this paper we regard the 3D rendering of the scene th be projected as given. The projection will be modified according to the viewing person's head position toward the projection plane. The approach neither forces the user to wear any kind of hardware like glasses or a head tracker nor needs expensive devices for the rendering of the scene, such as holograms or head-mounted displays.

Instead, we will use a low-cost depth camera to track the viewing person's head and create an adapted projection on a computer display or a projector.

2. Related work

In 2004 Göktürk and Tomasi used a time-of-flight camera to recognize and track a head with the generated depth-image [2]. Their approach is based on a knowledge-based clustering algorithm. They collect a training set of depth images in a preliminary step. During this training a feature vector is stored for each depth image. This feature vector is also referred to as the signature. Subsequently for each depth image a signature is created and compared against the signatures in the database. For the most likely matches a correlation metric is calculated between these and the image to find the best match.

Late in 2006, Nintendo released a game console with a new controller called the Wii Remote. This wireless controller is used as a pointing device and detects movements with its three axis accelerometers. Due to the low cost of this controller and the possibility to be connected via Bluetooth, numerous alternative applications, such as those proposed by Lee [4], Schlömer et al. [10], or Lin et al. [5] came up. One of the applications described by Lee is a view-dependent 3D projection using head tracking with infrared emitting diodes on glasses.

A more closely related approach to our paper is the "Real-Time 3D Head Tracking Based on Time-of-Flight Depth Sensor" by Ehsan Parvizi and Jonathan Wu [8]. The paper describes a head detection algorithm based on contour analysis on depth images. Like in the first mentioned paper a 3D time-of-flight depth sensor is used. The segmentation is done with a straightforward depth thresholding technique.

Approaches using a classical stereo matching on unstructured images are described in [7] and [9] and exhibit a higher complexity.

3. Used device

Two of the papers mentioned in the previous section use a 3D time-of-flight depth camera. Currently available time-of-flight depth cameras are quite expensive and have some disadvantages, including low resolution and sensitiveness to ambient light.

Since Microsoft launched the game controller Kinect for Xbox 360 in November 2010, a low cost alternative exist. It was developed in cooperation with PrimeSense and has a different concept than the Wii Remote: it creates a depth-image. On this account we use the Microsoft Kinect as a low cost depth camera solution. But any other depth capture device can also be used.

3.1. The Microsoft Kinect

The Microsoft Kinect, formerly known by its code name Project Natal, is a game controller for the Microsoft Xbox 360. It is mainly composed of three elements: First the illumination unit, which creates a projection of an infrared (IR) pattern with a known static and irregular dot distribution. Second the IR camera to capture the infrared pattern and to calculate a 640 × 480 pixel depth-image. And third the RGB camera with the same resolution [1].

Thus it is possible to use the Kinect as contact-free motion-, movement-, and gesture-recognition device. Recent research was made possible through the $3000-award that Adafruit Industries in New York granted the first person who was able to get RGB and distance values from the Microsoft Kinect.[1] One week later, the first Open Kinect driver was released.[2] Since this date, many applications related to the Microsoft Kinect have been devised.[3]

3.2. Depth calculation with the Microsoft Kinect

The basic principle of Kinect's depth calculation is based on stereo matching. Since classical stereo matching requires well structured images, the Kinect creates its own structure. Based on structured infrared light a static pseudo-random pattern is projected on the environment.

The stereo matching requires two images: one image is captured by the infrared camera, and the second image is the projected hardwired pattern. Those images are not equivalent, because there is some distance between projector and camera. So the images correspond to different camera positions, and that allows us to use stereo triangulation to calculate the depth.

As projector and camera are placed on a line with its principal axes parallel and with the epipolar lines parallel to the x-axis (the images are rectified), it is possible to use a single line (even a sub-pattern contained in a line) to search for matching points and to calculate the depth.

4. Head tracking method

In our algorithm only the depth-images are used. There is no classical face recognition and head tracking based on RGB data. We use the depth-image to find local minima of distance and the surrounding gradients to identify a blob with the size of a head. To measure and match the sizes in the depth-image we need to transform the raw data of the depth-image into a metric Euclidean Space.

4.1. Depth-image raw data

The raw data depth $d_{raw}(u, v)$ of this image is 11 bit (0-2047) per pixel. These discrete values do not have a linear distribution. To map them to the coordinate space of a projection, a fitting function between those values and a suitable linear space is necessary.

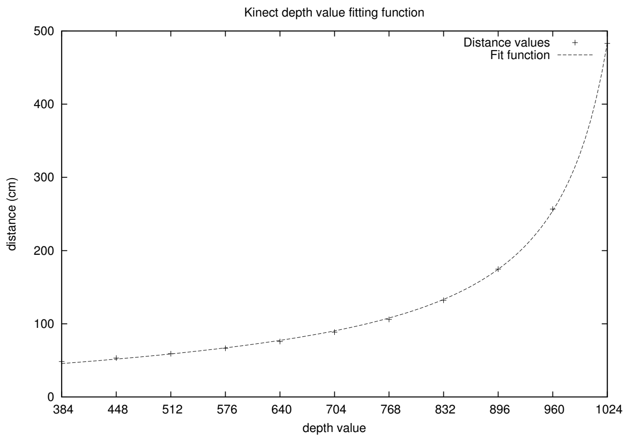

Figure 1. Fitting function between the raw data of the depth-image and the measured distance in centimeters of an object from the front edge of the IR camera.

To approximate this fitting function we measured the real world distances that belong to the Kinect depth values from 384 to 1024. This interval has been sampled in steps of 64, as shown in Figure 1. We begin with an approximated fitting function that has been proposed by Stephane Magnenat in a Google Newsgroup[4]:

$$d''(u, v) = 0.1236 \cdot \tan(d_{raw}(u, v)/2842.5 + 1.1863) \quad (1)$$

The conversion of the formula to centimeters and the approximation using a scaled Levenberg-Marquardt algorithm [6] with our measured values result in the fitting function illustrated in Figure 1:

$$d'(u, v) = 45.28 \cdot \tan(d_{raw}(u, v)/694.77) + 17.72 \quad (2)$$

The values may vary slightly among different controllers. However, their re-estimation will not be important as long as the absolute distance values are not needed.

4.2. Depth-image preprocessing

As mentioned in section 3.2, it is mandatory for stereo triangulation that projection and sensor do not have the same optical axis. But this necessity leads to the unpleasant side effect of "shadows" – regions without infrared patterns – for which no depth data is available. In Figure 2 this effect is depicted.

Due to the arrangement with the IR projector to the right of the IR camera, the shadows in depth-images are always on the left side of convex objects and on the inner right

[1] At http://www.adafruit.com/blog/ from 04.11.2010

[2] At http://www.adafruit.com/blog/ from 10.11.2010

[3] For an overview see http://www.adafruit.com/blog/

[4] http://groups.google.com/group/openkinect/browse_thread/thread/31351846fd33c78/e98a94ac605b9f21

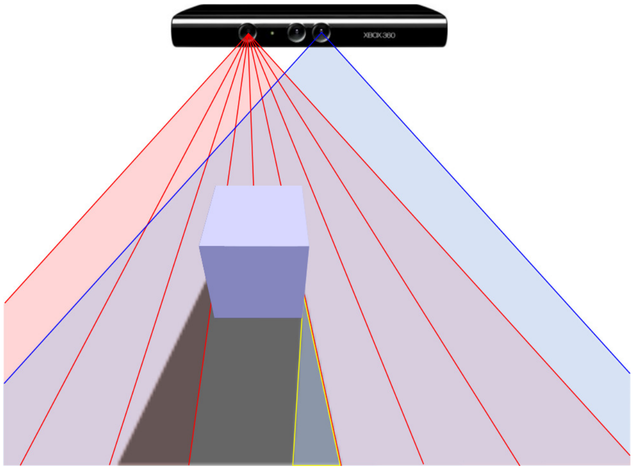

Figure 2. “Shadows” caused by the camera configuration. The red polygon shows the region of the IR pattern projection and the blue polygon shows the visible area from the point of view of the IR camera. The yellow framed area is visible by the IR camera with no projected pattern. In this area it is not possible to calculate depth values. We call this area a shadow.

side of concave objects. Since we search for heads to track, which are indeed concave, we can assume shadows on the left side of potential heads. Thus we can fill those shadows while processing the depth-image line-by-line from top to bottom and from left to right by using the last known depth value instead of the raw value 2047, which represents the unknown depth.

In addition to the shadows, the depth-images contain noise. The reason for that is the limited resolution of the IR camera. A single infrared point of the captured pattern may not be assigned to a single pixel. Therefore the position must be interpolated by the portion of luminosity of the two adjacent pixel. That means, in turn, that the calculation is sensitive to external infrared radiation, e. g., direct sun light. With increasing distance from the Kinect, the differences in centimeters between two possible depth values increase as well.

For example: the depth value 1010 corresponds to a distance of $\approx 402\,cm$, while the next value 1011 corresponds to a distance of $\approx 407\,cm$ (see Figure 1).

As already mentioned, we will use local minima in a depth-image and surrounding gradients to identify a head. To enhance their stability, we need to reduce this noise. The used algorithm should be fast enough to smooth the depth-image in real time (30 frames per second with a resolution of 640 × 480). For this reason we use a straight forward averaging filter utilizing the integral image filter from Viola and Jones [11]. The integral image provides the sum $i(u, v)$ of the pixel values above and to the left of the current depth value $d'(u, v)$, inclusive. It can be computed in linear time with a dynamic programming approach. Afterwards we can calculate the sum of any rectangular area with the width $2w+1$ and the height $2h+1$, and at the same time the arithmetic average of this rectangular in constant time, resulting in the smoothed depth image d:

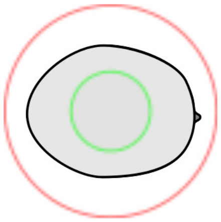

Figure 3. Top view of a human head facing right with the inner (green) and outer (red) bound for length and width.

$$\begin{aligned} A &= i(u-w, v-h) \\ B &= i(u+w, v-h) \\ C &= i(u-w, v+h) \\ D &= i(u+w, v+h) \end{aligned} \tag{3}$$

$$d(u,v) = \frac{A - B - C + D}{(2w+1)(2h+1)}$$

4.3. Head tracking

To decide if a blob found in the depth-image might be a head, we must define inner and outer bounds for its length, width and height. The head of an adult is approx $20\,cm \times 15\,cm \times 25\,cm$ (length×width×height). While we can assume the head in an upright position, the orientation is unclear. This is the reason why an inner bound should be $< 15\,cm$ and an outer bound should be $> 20\,cm$ for length and width, respectively. An inner bound of $10\,cm$ and an outer bound of $25\,cm$ turns out to be suitable (see Figure 3).

To calculate the size in pixel of these inner and outer bounds, we need to determine the camera’s field of view. At a distance of $100\,cm$ between the camera and a plane orthogonally to the viewing direction, we measured a visible width of $102.5\,cm$.

$$p := \frac{640\,px \cdot w}{1.025 \cdot d} \tag{4}$$

describes how many pixel p match the width w in a distance d. This results in the lower and the higher bounds

$$b_l(d) = \frac{640\,px \cdot 10\,cm}{1.025 \cdot d} \approx \frac{6250\,cm}{d} px \tag{5}$$

and

$$b_h(d) = \frac{640\,px \cdot 25\,cm}{1.025 \cdot d} \approx \frac{15600\,cm}{d} px. \tag{6}$$

To find a possible head-position, the depth-image is processed line by line. For each line v' find a u' where $d(u', v')$

is a local minimum and where

$$\forall f \in \left(1, \ldots, \frac{b_l(d(u',v'))}{2}\right) : \\ d(u'+f,v') - d_{min}(u',v') < 10\,cm \\ d(u'-f,v') - d_{min}(u',v') < 10\,cm \tag{7}$$

and

$$\begin{aligned} d(u' + \frac{b_h(d(u',v'))}{2}, v') - d_{min}(u',v') > 20\,cm \\ d(u' - \frac{b_h(d(u',v'))}{2}, v') - d_{min}(u',v') > 20\,cm \end{aligned} \tag{8}$$

In other words: the minimum depth value and the preceding and following depth values within the inner bounds must have a smaller difference than $10\,cm$, and the minimum depth value and the depth values at the outer bounds must have a larger difference than $20\,cm$.

We do not use the u'-value as horizontal center of the head. Instead we calculate the positions u_1 and u_2 of the lateral gradients, where the threshold difference of $20\,cm$ to the minimum distance $d(u',v')$ is exceeded

$$\begin{aligned} d(u_1,v') - d(u',v') &\leq 20\,cm \\ d(u_1-1,v') - d(u',v') &> 20\,cm \\ d(u_2,v') - d(u',v') &\leq 20\,cm \\ d(u_2+1,v') - d(u',v') &> 20\,cm \end{aligned} \tag{9}$$

and keep the arithmetic mean $\overline{u}(v') = \frac{u_1+u_2}{2}$ as an element of a possible vertical head axis.

Further we assume, that we need a number of subsequent lines where we can find those points. Therefore we have to count the number n of subsequent lines. If we are able to calculate $\overline{u}$ for the current line v', we increase $n = n + 1$. Otherwise we set $n = 0$

To calculate the required number of subsequent lines in which those $\overline{u}$ need to be found, we assumed a head height of at least $25\,cm$. Further we need a distance to measure the corresponding number of lines. The average distance of the points found in the last n subsequent lines is

$$\overline{d} = \frac{1}{n}\sum_{i=0}^{n-1} d(\overline{u}(v'-i), v'). \tag{10}$$

Thus the number of lines for this average distance is

$$n_{max} = \frac{640\,px \cdot 25\,cm}{1.025 \cdot \overline{d}} \approx \frac{15600\,cm}{d}px. \tag{11}$$

If $n \geq n_{max}$ we have the position $(\tilde{u}, \tilde{v})$ of the center of a head:

$$\begin{aligned} \tilde{u} &= \frac{1}{n}\sum_{i=0}^{n-1} \overline{u}(v'-i) \\ \tilde{v} &= v' - n/2 \end{aligned} \tag{12}$$

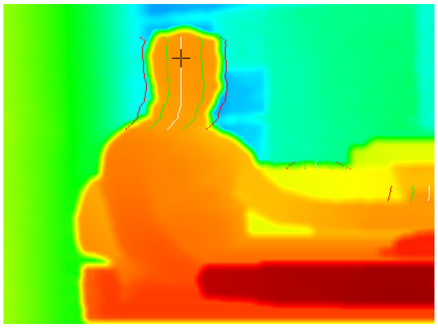

Figure 4. A colored depth-image. Near elements are red, far elements are blue. The vertical green and red lines show the inner and outer boundaries of 10 cm and 25 cm with reference to the detected minimum. The white line shows the calculated $\overline{u}$. Finally the cross marks the calculated head position $(\tilde{u}, \tilde{v})$.

Figure 4 shows a colored depth-image with drawn in bounds and the final head position.

4.4. Transformation into Cartesian coordinate system

Once we have the head position, we can transform the planar coordinates $\tilde{u}$ and $\tilde{v}$ into Cartesian coordinates, where the z-axis is along the viewing direction of the infrared camera. From section 4.3 we can derive the three Euclidean coordinates

$$\begin{aligned} x &= (\tilde{u} - 320) \cdot \frac{1.025 \cdot \overline{d}}{640}, \\ y &= (\tilde{v} - 240) \cdot \frac{1.025 \cdot \overline{d}}{640}, \text{ and} \\ z &= \overline{d}. \end{aligned} \tag{13}$$

So the head position in Cartesian camera coordinates is

$$\mathbf{h} := \begin{pmatrix} x \\ y \\ z \end{pmatrix}. \tag{14}$$

The scaling depends on the scaling of $\overline{d}$. By using the distance equation (2) from section 4.1, the values x, y and z have a metric scale.

4.5. Postprocessing of the head position

Although the values of the depth-image are smoothed, the gradients used in equation (9) have bumpy outlines and may not be stable from one frame to another. To compensate this, we use two filters:

1. Three separate median filters on the x-, y- and z-values of $\mathbf{h}$ over the five latest values to eliminate outliers or single frames where we were not able to find a head position.

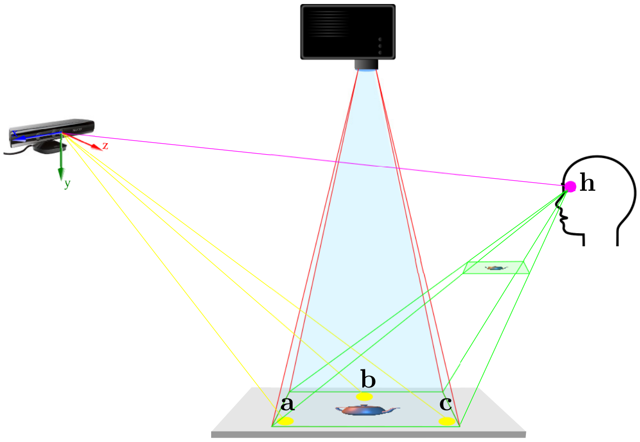

Figure 5. System configuration with camera, projector, projection surface and viewing person. The head position in camera coordinates is $\mathbf{h}$. The depicted yellow points $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$ are used in a precedent calibration step. The projection is rendered through a viewing frustum volume that is asymmetrically skewed (green).

2. An infinite impulse response filter to attenuate small movements.

$$\overline{\mathbf{h}} = (1-\alpha)\overline{\mathbf{h}} + \alpha\mathbf{h}, \text{ with } \alpha = \frac{1}{4} \tag{15}$$

   The initial value of $\overline{\mathbf{h}}$ is the first head position found.

With both filters enabled, the detected position is sufficiently stable.

5. Projection of the 3D scene

The projection of the 3D scene can be used on any surface, from a simple computer display up to a projection on a table.

5.1. Calibration

To transform the head position $\mathbf{h}$ from the camera coordinate space to the systems world coordinate space, a calibration is required. Therefore three spheres with known world coordinates will be displayed ($\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$ in Figure 5). We define $\mathbf{a}_z = \mathbf{b}_z = \mathbf{c}_z = 0$ for the centers of those spheres. Since there is no need for the 3D scene to be displayed in a realistic scaling, it is sufficient for the x- and y-values of those sphere midpoints to be elements of the same Euclidean space. If proper scaling is required, the distance or the coordinates of the projected spheres must be measured in a metric scale.

Now, within a merged view of the colored depth-image and the RGB image (see Figure 6), the user can click on the spheres. With the u and v coordinates from the relative mouse position and the depth value $d(u, v)$ from the depth-image, the coordinates of these spheres in camera space can be calculated (see 4.4). With those coordinate pairs from the three spheres we calculate the transformation matrix $\mathbf{T}$ using the solution of Horn [3].

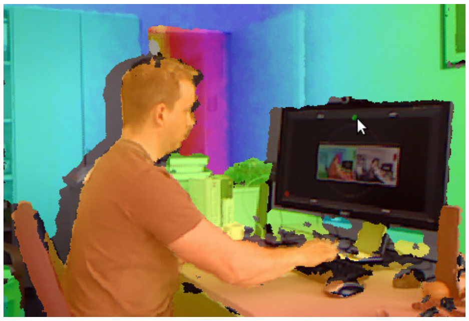

Figure 6. Merged view of colored depth-image and RGB image while selecting the second calibration sphere (green). The intended projection surface in this example is the computer display.

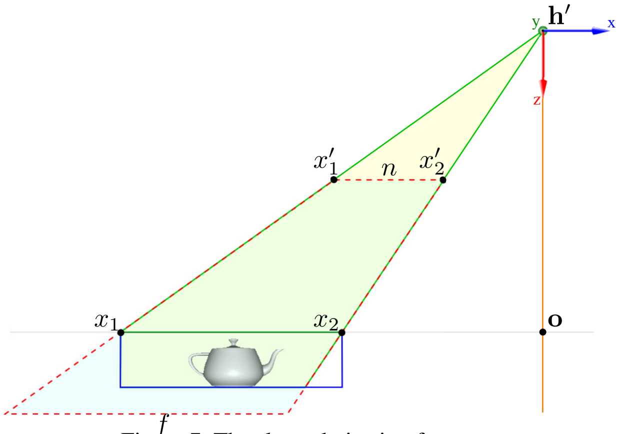

Figure 7. The skewed viewing frustum

5.2. Transformations

There are two different transforms needed: (1) the head position has to be transformed from the Kinect's camera coordinate space into world space coordinates and (2) the 3D scene has to be projected onto the 2D projection space.

(1) To transform the head position from the Kinect's camera coordinate space into world space coordinates the only thing to do is to multiply our head position with the previously calculated transform: $\mathbf{h}' = \mathbf{T} \cdot \overline{\mathbf{h}}$.

(2) To calculate the perspective projection matrix for the 2D projection we regard $\mathbf{h}'$ as camera with the viewing direction to

$$\mathbf{o} = \begin{pmatrix} \mathbf{h}'_x \\ \mathbf{h}'_y \\ 0 \end{pmatrix} \tag{16}$$

which is depicted in Figure 7. We skew and scale the viewing frustum (the red dashed line), in such a way that the lateral surfaces of the frustum intersect with the projection plane (gray) at the outer edges of the favored visible region (dark green line). This region must correspond to the visible region during calibration.

Let x_1 and x_2 be the outer x-values and y_1 and y_2 be the outer y-values of this visible region. As previously defined the projection plane is at $z = 0$, initially.

To be able to view the projection from a flat angle where $\mathbf{h}'_z \approx 1$ we set the near clipping plane $n = \mathbf{h}'_z + 1$ near to the head position. The far clipping plane f has some larger value. So we can derive the x- and y-values for the near clipping plane from the visible region of the far clipping plane:

$$\begin{aligned} x'_1 &= (x_1 - \mathbf{h}'_x)/n \\ x'_2 &= (x_2 - \mathbf{h}'_x)/n \\ y'_1 &= (y_1 - \mathbf{h}'_y)/n \\ y'_2 &= (y_2 - \mathbf{h}'_y)/n \end{aligned} \tag{17}$$

Resulting from the positioning of the near clipping plane as just described the distance d_n between the near clipping plane and the camera is $d_n = 1$. The distance between the far clipping plane and the camera is $d_f = f - \mathbf{h}'_z$.

Now we can define the projection matrix for the projection of the 3D scene as follows:

$$\mathbf{P} = \begin{bmatrix} \frac{2d_n}{x'_2 - x'_1} & 0 & \frac{x'_2 + x'_1}{x'_2 - x'_1} & 0 \\ 0 & \frac{2d_n}{y'_1 - y'_2} & \frac{y'_1 + y'_2}{y'_1 - y'_2} & 0 \\ 0 & 0 & -\frac{d_f + d_n}{d_f - d_n} & -\frac{2d_f d_n}{d_f - d_n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{18}$$

5.3. Realization with OpenGL

The matrix in equation (18) is equivalent to the projection matrix used by the OpenGL function `glFrustum`. We use OpenGL in our implementation to display a 3D scene.

OpenGL is a state machine. When a state value is set, it remains set until some other function changes it. One of this states is the projection matrix. It is applied to any geometry drawn. When we calculate the head position in world coordinates and the six values x'_1, x'_2, y'_1, y'_2, d_n and d_f from above, we can use

```
gluLookAt(h.x,h.y,h.z,h.x,h.y,0,0,1,0);
glFrustum(x1,x2,y1,y2,dn,df);
```

to set the state of the OpenGL projection matrix. With the first three parameters of `gluLookAt` we set the position of the head $\mathbf{h}'$. With the second three parameters we set the point $\mathbf{o}$ to look at. The third three parameters describe the direction of the y-axis as 'up'. Finally `glFrustum` creates the matrix of equation (18) and multiplies the current projection matrix state with it.

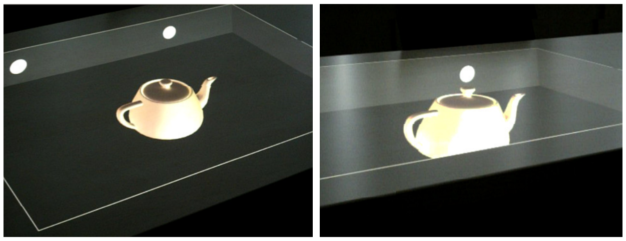

Figure 8. Resulting impressions of the projection for different viewpoints of the viewing person. The images are taken with a digital photo camera placed directly in front of the persons eyes. Left: the top view of the teapot in the box with two of the virtual light sources visible. Right: the cut-off of the teapot viewed from a low viewpoint.

5.4. Box-Layout

In Figure 7 a rendered 3D scene (here: only one object) is shown below the projection plane. Furthermore it is placed in a box (the blue lines). There are two reasons for this:

1. When a 3D scene would be projected directly on the projection plane it would be truncated on top, when the viewer takes a low viewpoint, resulting in an unrealistic impression. When the scene is displayed in our box-layout it will never be truncated on top. Rather, when the scene is viewed from a low viewpoint with respect to the projection plane, the scene will be cut off on the lower side, as one would expect.

2. The illumination of a given 3D scene rarely matches the natural illumination of the room in which the projection takes place. By placing stylized light sources into the box, the viewers' perception is influenced resulting in a more realistic impression.

Both effects are illustrated in Figure 8. Although the benefits of the box-layout have been illustrated here by means of a horizontal projection plane on a table, teh same principles hold true for any orientation of a projection plane such as a vertical wall.

6. Results

In Figure 8 and Figure 9 a few pictures of our testing scenes are shown. They were taken while moving around a table which was used as a projection plane for the projector installed on the ceiling. While Figure 8 shows the OpenGL teapot, Figure 9 shows a more complex, textured mesh of a little Sokrates statue. These pictures can not reflect the effect seriously in any way. So we decided to take a short movie walking around this table, which is able to illustrate the resulting effect more compelling. This video can be found here:

http://www.mci.fernuni-hagen.de/conferences/procams2011/kinevision.mp4

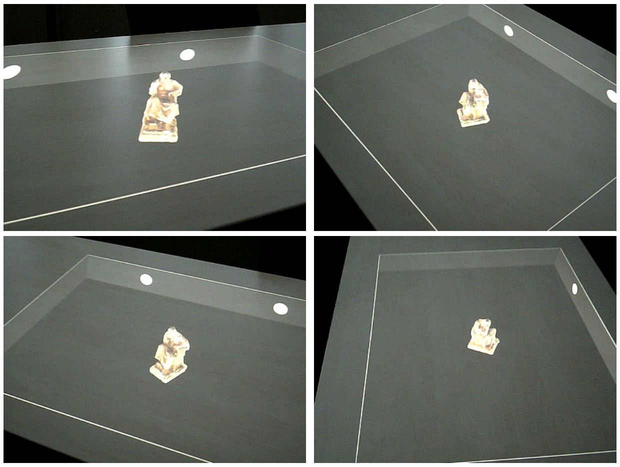

Figure 9. Four views of a different, more complex object.

Of course, any precomputed 3D scene can be projected in the same way.

7. Discussion

We introduced a novel approach for head tracking using depth-images. This enables us to detect the head orientation in space relative to the depth camera in real time. This is achieved by utilizing the recently introduced hardware Microsoft Kinect.

With the head orientation in space we compute a view-dependent 3D projection, which provides the viewing person with a realistic impression of the projected scene independently of his or her position in relation to the projection plane. The realism of the illusion is improved further by embedding the 3D scene in a virtual illuminated box.

7.1. Limitations

Our approach implies, that only one viewer is located in the view area of the Kinect.

Problems also exist with reflecting surfaces and solar radiation in the room where the projection takes place, because the Microsoft Kinect is unable to detect the IR pattern on bright and glossy surfaces. So the best results are achieved in a dimmed room.

A side effect of the use of the median filter and the infinite impulse response filter described in section 4.5 is, that fast movements of the observer will result in delays of the adaption of the projection to the new position of the observer. Although the delays are very small, they are perceived by the human brain and disturb the illusion.

7.2. Future Work

There are many applications for the proposed projection system. A conference table with a separate projection for each participant would be one conceivable scenario. This could be done by extending the head tracking system to multiple heads.

In addition, the illusion of the projection could be enhanced generating stereoscopic images in combination with shutter glasses. This makes the approach applicable to CAVE-like projector-based installations for low budgets.

References

[1] B. Freedman, A. Shpunt, M. Machline, and Y. Arieli. Depth mapping using projected patterns. U.S. Patent #2010/0118123 A1 issued May 13, 2010, filed 2008. 2

[2] S. B. Göktürk and C. Tomasi. 3d head tracking based on recognition and interpolation using a time-of-flight depth sensor. In *Proceedings of the 2004 IEEE computer society conference on Computer vision and pattern recognition*, CVPR'04, pages 211–217, Washington, DC, USA, 2004. IEEE Computer Society. 1

[3] B. K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A*, 4(4):629–642, 1987. 5

[4] J. C. Lee. Hacking the nintendo wii remote. *IEEE Pervasive Computing*, 7:39–45, July 2008. 1

[5] J. Lin, H. Nishino, T. Kagawa, and K. Utsumiya. Free hand interface for controlling applications based on wii remote ir sensor. In *Proceedings of the 9th ACM SIGGRAPH Conference on Virtual-Reality Continuum and its Applications in Industry*, VRCAI '10, pages 139–142, New York, NY, USA, 2010. ACM. 1

[6] J. Moré. The levenberg-marquardt algorithm: Implementation and theory. In G. Watson, editor, *Numerical Analysis*, volume 630 of *Lecture Notes in Mathematics*, pages 105–116. Springer Berlin / Heidelberg, 1978. 10.1007/BFb0067700. 2

[7] R. Newman, Y. Matsumoto, S. Rougeaux, and A. Zelinsky. Real-time stereo tracking for head pose and gaze estimation. In *Proceedings of the Fourth IEEE International Conference on Automatic Face and Gesture Recognition 2000*, FG '00, pages 122–, Washington, DC, USA, 2000. IEEE Computer Society. 1

[8] E. Parvizi and Q. M. J. Wu. Real-time 3d head tracking based on time-of-flight depth sensor. In *Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence - Volume 01*, ICTAI '07, pages 517–521, Washington, DC, USA, 2007. IEEE Computer Society. 1

[9] D. B. Russakoff and M. Herman. Head tracking using stereo. *Mach. Vision Appl.*, 13:164–173, July 2002. 1

[10] T. Schlömer, B. Poppinga, N. Henze, and S. Boll. Gesture recognition with a wii controller. In *Proceedings of the 2nd international conference on Tangible and embedded interaction*, TEI '08, pages 11–14, New York, NY, USA, 2008. ACM. 1

[11] P. Viola and M. J. Jones. Robust real-time face detection. *Int. J. Comput. Vision*, 57:137–154, May 2004. 3