Adapting Hair and Face Geometry of Virtual Avatars with the Kinect Sensor

Hannes Plank
Supervised by: Stefan Hauswiesner

Graz University of Technology, Austria

Abstract

In this project, we developed a method for creating virtual head models that look like the user. The resulting application scans the user with a Microsoft Kinect sensor to obtain RGB-D images. Based on these images, a virtual face model can be adapted to resemble the users appearance.

A fast C++ implementation was created. To generate hair, several hair parameters were obtained. A method to personalize existing hair mesh templates was implemented.

1 Introduction

Virtual avatars are virtual 3D representations of humans. They can help users to feel more involved with an application and enable a number of future high-impact use cases. For example, users can see themselves in different clothes in virtual dressing rooms. Computer games could be customized to allow the user to be the main character of a game, or interact with other people and their personal avatars. High-end teleconferencing systems would allow participants to collaborate naturally in a 3D virtual meeting room.

We developed an implementation capable of creating virtual avatars using the popular Microsoft Kinect sensor. The implementation acquires several RGB-D frames of the user. Using a face tracking algorithm, it tracks the users head and fits a face tracking mesh for every frame. The gathered RGB-D frames are converted to 3D pointclouds. Using the head tracking information, the pointclouds are placed over a generic morphable 3D face. The avatar is morphed with an optimization algorithm, minimizing the distance between the avatar and the pointcloud.

To ensure compatibility with common work-flows, we use avatar data models that can be generated using DAZ 3D Studio. These avatars feature parametric head models which are customizable.

To generate realistic hair, an algorithm detects various hair parameters. Several hair template meshes for different hair lengths were created. By evaluating the hair parameters, the best template is chosen automatically. The template is morphed to adapt to the persons hairstyle.

The hair and face meshes are projectively textured by using the best captured RGB image.

To demonstrate the algorithm, a fast C++ implementation, using the Kinect SDK and OpenCV was developed.

2 Related Work

The Microsoft Kinect was the first affordable depth camera, available on the consumer market. Since the Microsoft Windows driver appeared, a lot of Kinect and RGB-D camera related research was accomplished.

There is an approach to scan a whole person with the Kinect [Sum+13]. This method however needs 15 minutes to generate a watertight mesh, with an algorithm similar to Kinect Fusion [Iza+11]. Our project is focused on makeing the avatar personalization process as intuitive and simple (from the users perspective) as possible.

Zollhofer et al. [Zol+14] present a very similar approach to our project. They built an interactive system which reconstructs facial data in real time, while giving the user feedback. Instead of morphing a pre-designed face template mesh, like in this project, this implementation combines 200 different heads into a statistical shape model. With depth fitting and regularization, they estimate the parameters of the head. We apply morphing for our implementation, which replaces the task of creating a statistical model.

If there is no depth information available, Jiang et at. [Jia+05] show that it is also possible to reconstruct faces by using 3D face shape databases. In their approach, the 2D face image is first aligned with a generic 3D face geometry. Since all face geometry is compressed by PCA, the key features of the 2D face are used to compute the 3D shape coefficients of the Eigen vectors. The face shape is reconstructed by using these coefficients.

Cao et al. [Cao+13] show, that animating abstract avatars can be performed just by using simple VGA cameras. It is possible to transfer facial expressions from a person in real time to any mesh. The paper demonstrates how far interaction with personalized avatars can go, and how much potential there is. As first step, the implementation requests the user to make several extreme facial expressions. The abstract avatars have geometrical models of each these extreme facial expressions. In the interactive initialization part, the algorithm registers the persons

extreme facial expressions. During animation, it interpolates the previously saved extreme expressions. This is also possible with the avatars created by our project, however additional face meshes for the extreme facial expressions would be required.

Chai et al. [Cha+13] show a very promising way to generate 3D volumes of human hair just by using an image sequence. The gathered hair model is based on strands. The model also enables physically plausible animation of hair. As mentioned in section 5, the image quality of the Kinect color camera was not sufficient to consider this hair generation method.

Blanz et al. [BV03] also use 2D images and a 3D scan database to obtain face geometry. Their implementation takes several 2D images of the same face to fit data to a morphable 3D face model. Their morphable 3D face model was created from a database of 3D scans. The goal of this 3D scanning methods is face recognition rather than realistic avatar generation.

There is also a way to reconstruct faces from a single 2D image with a generic face mesh [Mag+13]. It exploits the global similarity of faces and combines shading information with generic shape information. With a Kinect depth camera available, there is not much potential to use this technique additionally. However some geometry adaptions, mentioned in section 5.3, are only influenced 2D texture evaluation.

3 The Procedure

The virtual avatars are created by using RGB-D data to morph a generic face mesh. At first, multiple RGB-D frames are captured with the Kinect sensor. They are converted to pointclouds and are aligned with a morphable face mesh.

After morphing the head, some processing is done to improve the avatar and add hair geometry. Additional processing steps include the projective texture mapping, and positioning the eyes.

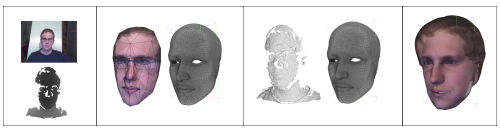

Figure 1: The acquired data, the facetracking mesh aligned with the morphable generic head mesh, the pointclouds aligned with the head mesh, morphed result with eye and hairmesh.

3.1 Data gathering

The data gathering software captures the RGB-D frames and tracks the rotation and position with a head tracking algorithm [Mic13]. Several frames are captured by the Kinect RGB-D camera. The user presses a key to capture a frame in a ten second interval. The optimal distance from the sensor is one meter. Best results are obtained with a bright diffuse illumination and a neutral background.

The following data is captured per frame:

- A color image.
- The depth information. Using the known camera parameters, a depth image is converted to a 3D pointcloud. Due to the face tracking, it is possible to align the pointclouds.
- A facetracking mesh. The Kinect facetracker fits a Candide 3 [Ahl01] mesh to the face of the user. The resulting low polygon facetracking mesh has the same orientation and scale of the pointclouds.
- The rotation and translation of the head, calculated by the Kinect facetracking algorithm.
- A projection matrix for projective texturing.

3.2 Face mesh production

The implementation goes through the following processing steps as illustrated in Figure 1:

1. The Candide 3 headtracking meshes are in the same coordinate system as their corresponding pointclouds. Each pointcloud has one headtracking mesh, so the pose of the users head is saveed in the pose of the headtracking mesh. The orientation and position of the facetracking meshes is used to calculate a transformation for each point cloud. The transformed pointclouds are aligned in the same coordinate system. The morphable face template mesh is scaled and placed close to the pointclouds..
2. $$\begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix} + \begin{pmatrix} d_x \\ d_y \\ d_z \end{pmatrix} * f \qquad (1)$$

   The morphing equation (1) transforms a vertex v in direction d by a factor f. The external morphing data from [Pro14] contains groups of vertices with the same factor f_n. Each vertex is assigned a direction vector v_n. The direction vector's magnitude defines how much influence a change in factor factor f_n has.

   To be able to morph the head template, all morphing equations are compiled into equation system (2).

$$\begin{pmatrix} p_{1_x} & 0 & 0 & \vdots \\ 0 & p_{1_y} & 0 & \vdots \\ 0 & 0 & p_{1_z} & \vdots \\ \cdots & \cdots & \cdots & p_{n_{x/y/z}} \end{pmatrix} = \begin{pmatrix} v_{1_x} & 0 & 0 & \vdots \\ 0 & v1_y & 0 & \vdots \\ 0 & 0 & v_{1_z} & \vdots \\ \cdots & \cdots & \cdots & v_{n_{x/y/z}} \end{pmatrix} + \begin{pmatrix} f_1 & 0 & 0 & \vdots \\ 0 & f_1 & 0 & \vdots \\ 0 & 0 & f_1 & \vdots \\ \cdots & \cdots & \cdots & f_n \end{pmatrix} \qquad (2)$$

   The morphing factors are retrieved by solving the equation system. This can be seen as a minimization of the distances between the vertices of template

mesh v and their nearest neighbours p in the pointclouds. When the morphing factors are known, the morphing equation 1 can be applied to each vertex v. This enables to create a personalized, watertight DAZ 3D face mesh. This format can be incorporated into existing workflows.

3. The mesh is projectively textured with a color image and saved in Wavefront obj mesh file format.

4 Improvements

The avatars generated by the introduced algorithm resemble the scanned person, however a better result is received by some improvements.

Precise alignment of the pointcloud data and the morphing template is crucial for good results. The mesh produced by the Kinect face tracker, is in the same coordinate system as the pointcloud data. This means the necessary pointcloud transformation can be calculated by aligning the face tracking mesh with the morphing template. At first the generic unfitted Candide 3 [Ahl01] face tracking mesh was used. To take individual face proportions into account, a fitted Candide 3 mesh as seen in Figure 2, was used instead. This leads to better pointcloud alignment and improves fitting the textures.

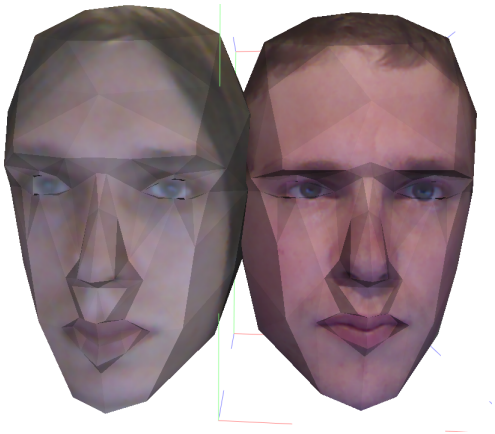

Figure 2: Fitted Candide 3 facetracking meshes, obtained by the Kinect SDK facetracker.

Aligning the facetracking mesh with the morphable facemesh is not trivial. Various alignment methods were evaluated. Scaling along all dimensions did not produce realistic looking faces, since the persons face proportions got distorted. When the aligning aims to minimize the distance to the facial features, the rest of the head is distorted tremendously.

Simply aligning the face tracking mesh by its center and scaling by a calculated factor in every dimension as seen in Figure 3, produced the best results. The result of the alignment is shown in Figure 4.

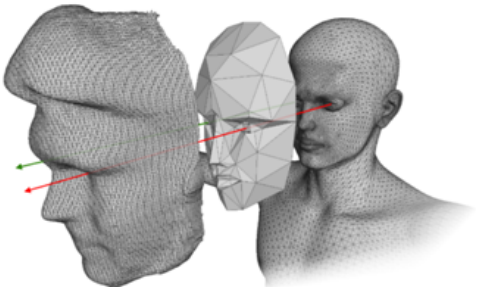

Figure 3: The pointcloud data is placed on the virtual avatar template with the help of the facetracking mesh.

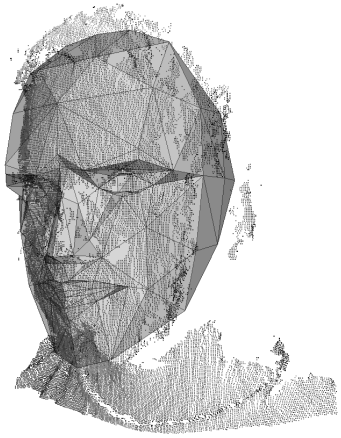

Figure 4: Pointcloud data from the Kinect sensor overlayed with the Candide 3 facetracking mesh.

4.1 Optimization

Since the creation of the virtual avatar is computationally complex, a fast implementation was needed. Early versions used the Kinect Fusion algorithm to obtain high quality depth maps. Our experiments came to the conclusion, that simple RGB-D images are sufficient.

We developed a fast C++ implementation using OpenCV and the EIGEN Library. The solver of the Eigen Library was utilized to solve the morphing equation. The Kinect SDK is used to capture the RGB-D frames and perform headtracking. The implementation has an interface to process live RGB-D frames from arbitrary cameras. It is possible to align the pointclouds and capture new frames at the same time in seperate threads. For direct and fast display and further processing, there is an interface to retrieve the geometry data arrays in an OpenGL compatible format. An external obj file format importer/exporter was used to load the data from a Kinect capturing framework and to produce the output meshes.

5 Hair Reconstruction

Realistic virtual avatars need hair. There are various approaches, however only a few parameters can be obtained by the data from the Kinect sensor. The resolution of the Kinect RGB camera is not good enough to extract orientations of hair strands. Therefore only color, shape and geometry of the frontal hair are obtained.

To not completely rely on morphing, several hair templates for different length of hair were created. An algorithm detects each hair type by evaluating the color input images. The hair template as seen in Figure 11, is optimized by modifying the shape by scaling or morphing.

5.1 Texture atlas

Since several data frames are captured, all hair parameter detection can be performed on multiple frames and then averaged. To explore a reduction in the processing time, and eliminating false classifications, a texture atlas was created.

It was necessary to associate color information with each vertex of the pointcloud data. The vertices were projected into the corresponding texture. The pointcloud data from all frames was aligned in the same coordinate space. Just by removing the z-coordinate, the complete pointcloud data got projected into 2D. By interpolating the color information between the projected vertices, a texture atlas was obtained.

For the depth data, a texture atlas was created as well. Since the pointclouds are previously aligned in the same coordinate system, a simple plane projection was sufficient to create the depth-atlas. A depth and a RGB texture atlas can be seen in Figure 5.

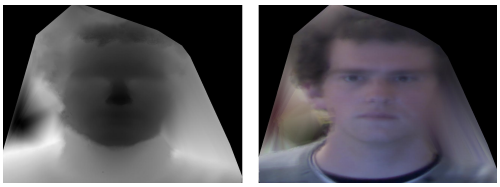

Figure 5: The texture atlas of the color and the depth images.

Performing all detections on all captured images proved to be more robust. Calculating the variance from the atlas for each image was a good method to eliminate erroneous frames. It was also possible to vote on the frames, and calculate a new weighted average texture atlas. Frames with a high similarity to the original texture atlas were assigned better weights than others. The improvement can be seen in Figure 6.

However, hair segmentation using the atlas turned out to be more difficult, because information is lost during the averaging process.

Figure 6: Original texture atlas on the left, weighted texture atlas on the right

5.2 Hair Segmentation

To procedurally generate a realistic hair mesh, information about the hair is required. Various methods were implemented to segment the hair on captured RGB images.

The first attempt was to segment hair with k-means clustering. By projecting the virtual avatar into the frame, the face could be masked. The trick of the k-means clustering in this application was to find a cluster containing hair. The area on the forehead above the hairline was masked. After the k-means clustering, the two most prominent clusters on forehead were selected. These clusters were likely hair clusters.

Figure 7: Red is the most prominent cluster, blue the second prominent cluster.

K-means clustering as seen in Figure 7, was not robust enough and did not consider the geometrical topology of the hair.

The Growcut algorithm [Vl05] as seen in Figure 8, takes sets of pixels of the image as input and performs iterative

clustering with cellular growth from these points. Pixels similar to the hair color were chosen for the hair as input. Random pixels in the face and on the background were defined as starting points for the face cluster.

Figure 8: Clustering with grow cut

Since these results were not good enough, segmenting the RGB images with a Graphcut implementation [GPS89] was evaluated. Even with the depth information and different color spaces, the results were not good enough.

Because the goal of this project was an applicable and robust implementation, a different strategy was chosen:

The hair length only gets estimated to choose a predefined hair template. The template later gets personalized by morphing. Only the following parameters had to be obtained from the RGB images

Hairline

To find the point where the hair starts on the forehead, the hairline need to be estimated. The position of the facial features on the RGB images are known, since certain vertices just need to be projected into these images. The maximum magnitude of the color gradient between the eyebrows and the top of the head was searched on the 2D images. It turned out to be a very robust approximation of the hairline. The results of the hairline detection can be seen in Figure 9.

Hair color

A good approximation of the hair color was the average color of the pixels above the hairline. The pixels under the hairline were identified as skin color and used during the hair template morphing.

Figure 9: Red: Calculated hairline Green: Averaged hairline parameter.

Hair length

The obtained hair length in this implementation is only a rough estimate to select a template with proper hair length. Three templates were created with Sculptris [Pix13]. The hair length was estimated efficiently by comparing the pixel color with the previously calculated hair color. Masking the face and the background helped a lot to get a robust estimate of the hair length. All parameters were obtained for every captured RGB-D frame separately and then averaged.

5.3 Hair Template Adaption

The hair length is classified into three categories. Depending on the detected category, a hair template mesh is loaded. Figure 10 shows some examples for possible hair templates. The template is positioned on the head of the virtual avatar as seen in Figure 12. The mesh gets projectively textured with a Kinect color image. Since the hair color is known, vertices with a different projected color are morphed.

All morphing operations face to the center of the head. The operation is applied per vertex, but the surrounding vertices are moved as well. The movement is linearly depended to the distance from the center vertex. This eliminates sharp creases and softens morphing. Several passes are performed. The projective texture is mapped again at the end of each pass, to ensure the morphing is stopped when the hair template is in shape. Figure 13 shows the result after the morphing.

Because there are only images of the front side of the head captured, the back of the head neede to be textured as well. To be able to also morph the back of the hair template, the back of the head was textured projectively with the front texture. However the face of the back texture was

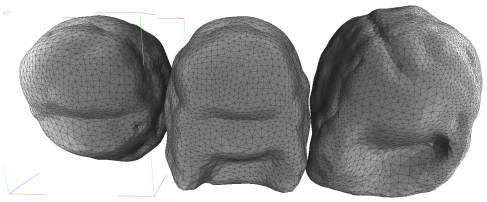

Figure 10: Hair templates for three different hair lengths.

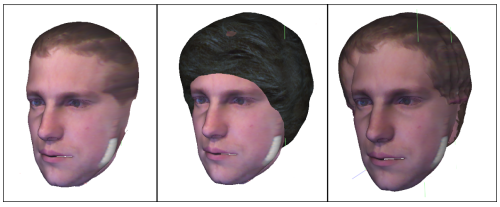

Figure 11: Face mesh, face mesh with hair template, face mesh with adapted hair template

masked and the hair color was interpolated. As a side effect, the backside of the head also has a hair texture which blends with the front texture. The interpolation was simply done by replacing the face with the color of the hair as seen in Figure 14. There is room for improvement by using a better interpolation method.

6 Results

The results were retrieved in a scene with a clear background and bright lightning.

The final result as seen in Figure 15 consists of three textured meshes. The three meshes are the face, eyes and hair. They are separate meshes for exchangeability and to ease future animation. The meshes are saved in the .obj format, however the C++ implementation offers an interface to use the mesh data live and interactively.

6.1 Execution time

The time to generate a mesh from 7 RGB-D frames on a Pentium Dual Core CPU is about 45 seconds. However most of the computation time is needed by I/O operations with the slow OBJ fileformat. In an realistic application, the templates are previously loaded. Since the results do not have to be exported to an OBJ mesh, the system can work significantly faster. The most crucial and computationally intensive part of the implementation is solving the morphing equation. This part only takes 1.5 seconds using the solver of the EIGEN library.

6.2 Problems

One problem is the false morphing of the cheeks of the heads. In Figure 15, you can see white spots on some cheeks. This is caused by the lack of depth data in these

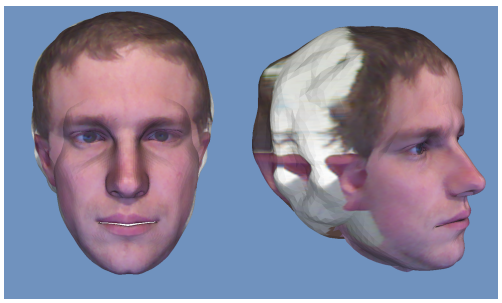

Figure 12: Virtual avatar with unmorphed hair template.

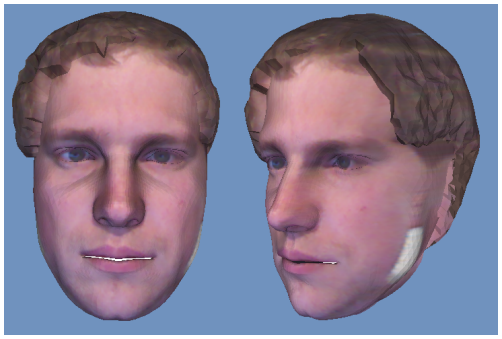

Figure 13: Virtual avatar with morphed hair.

regions. The RGB-D image are usually gathered just from the front side of the face. This might either be solved by removing certain morphing vectors, or reshaping the face mesh the same way as the hair mesh. Replacing the white spots with the skin color might also be a viable option.

Another problem is the low texture resolution on the side of the hair. This is a consequence of the dual projective texturing. Textures are projected on the front and back, but not on the sides. During the development of the implementation, using the best suiting texture for every polygon was evaluated, but the desired OBJ mesh export did not support multi-texture blending. However all the information can be obtained by the C++ interface, and a future interactive implementation can use shader programs to blend all RGB frames for optimal mesh texturing.

7 Conclusion

Automatic avatar generation can be used to enrich games and telepresence systems. We demonstrated that the concept of template adaptation can be extended to generate hair models and avatar generation can be performed in a time frame that is suitable for the named applications. Moreover, we came to the conclusion, that shortcomings of the depth sensor can be compensated well enough. However a bad RGB sensor yields directly to bad results. Our system therefore needs decent lighting and a contrasting neutral background. We assume that average users are capable of meeting these two requirements. Due to the wide availability of commodity depth sensors, such as the

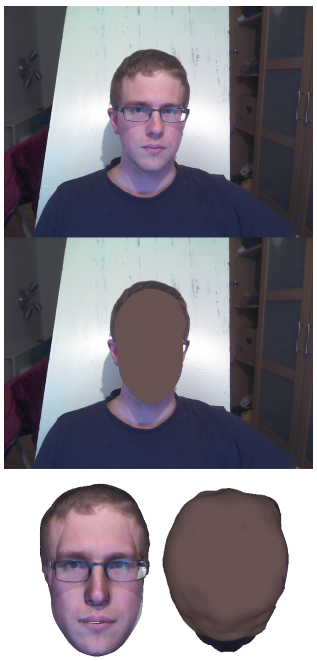

Figure 14: Texturing the backside. From top to bottom: Front texture, back texture with face automatically replaced, front and backside of the projectively textured mesh.

Figure 15: RGB Image and the final result.

Microsoft Kinect, we believe that particularly games and applications will contain automatic avatar generation features in the future.

References

[Ahl01] Jrgen Ahlberg. *CANDIDE-3 - An Updated Parameterised Face*. Tech. rep. 2001.

[BV03] V. Blanz and T. Vetter. "Face recognition based on fitting a 3D morphable model". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 25.9 (2003), pp. 1063–1074. ISSN: 0162-8828.

[Cao+13] Chen Cao et al. "3D Shape Regression for Real-time Facial Animation". In: *ACM Trans. Graph.* 32.4 (July 2013), 41:1–41:10. ISSN: 0730-0301.

[Cha+13] Menglei Chai et al. "Dynamic Hair Manipulation in Images and Videos". In: *ACM Trans. Graph.* 32.4 (July 2013), 75:1–75:8. ISSN: 0730-0301.

[GPS89] D. M. Greig, B. T. Porteous, and A. H. Seheult. "Exact maximum a posteriori estimation for binary images". In: (1989).

[Iza+11] Shahram Izadi et al. "KinectFusion: Real-time 3D Reconstruction and Interaction Using a Moving Depth Camera". In: *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*. UIST '11. Santa Barbara, California, USA: ACM, 2011, pp. 559–568. ISBN: 978-1-4503-0716-1. DOI: 10.1145/2047196.2047270. URL: http://doi.acm.org/10.1145/2047196.2047270.

[Jia+05] Dalong Jiang et al. "Efficient 3D reconstruction for face recognition". In: *Pattern Recognition* 38.6 (2005). Image Understanding for Photographs, pp. 787 –798. ISSN: 0031-3203.

[Mag+13] AshrafY.A. Maghari et al. "Reconstructing 3D Face Shapes from Single 2D Images Using an Adaptive Deformation Model". English. In: *Advances in Visual Informatics*. Ed. by HalimahBadioze Zaman et al. Vol. 8237. Lecture Notes in Computer Science. Springer International Publishing, 2013, pp. 300–310. ISBN: 978-3-319-02957-3.

[Mic13] Microsoft. *Kinect Headtracking*. 2013. URL: https://msdn.microsoft.com/en-us/library/jj130970.aspx.

[Pix13] Pixologic. *Sculptris*. 2013. URL: http://pixologic.com/sculptris.

[Pro14] DAZ Productions. *DAZ 3D*. 2014. URL: http://www.daz3d.com.

[Sum+13] E.A. Suma et al. "Rapid generation of personalized avatars". In: *Virtual Reality (VR), 2013 IEEE*. Mar. 2013, pp. 185–185.

[Vl05] Vladimir Vezhnevets and eta l. *"GrowCut" – Interactive Multi-Label N-D Image Segmentation By Cellular Automata*. 2005.

[Zol+14] Michael Zollhofer et al. "Interactive model-based reconstruction of the human head using an RGB-D sensor". In: *Computer Animation and Virtual Worlds* 25.3-4 (2014), pp. 213–222. ISSN: 1546-427X.