WAIT: Workshop on the Advances in Information Technology, Budapest, 2018

Stylized facial video processing using Kinect face tracking

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 present an NPR post processing method for RGBZ cameras. Our stroke based NPR technique uses geometry buffers to reconstruct geometric features that are used to calculate stroke properties like stroke direction and bending. We mainly focus on improving the quality on human face areas with the help of facial tracking and face geometry reconstruction techniques provided by the Kinect SDK. Our method is implemented on the GPU and provides high quality, temporal coherent hatching strokes for faces at interactive frame rates.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

1. Introduction

A wide range of computer graphics techniques deals with non-photorealistic (NPR) or stylistic rendering. These techniques try to reproduce the hand made feel of traditional drawings and paintings. Some of these techniques address 3D animations, others postprocess traditional videos. Our goal is to process a video but make stylization using a stroke based NPR technique that was developed for rendering 3D scenes. To reconstruct the 3D geometry seen on the video we use the depth data of a Kinect One device. RGBZ cameras were already used as a base for this NPR technique but it had several limitations and could not reproduce the quality of images based on 3D renderings.

In many cases the video to be processed contains a closeup of a human face. An obvious example is a video chat stream. As human faces play serious role in video processing many method has been developed to identify facial areas or even track the face motion and reconstruct its 3D geometry. These techniques can be used to help the NPR post processing of videos, and in this paper we investigate such a direction.

Though the Kinect is not a very wide spread device despite its relatively low price, and its development has been stopped, but RGBZ technology will likely become a standard in future handheld devices and probably in webcams too. Depth sensing using depth sensors or stereo technologies is also an active research area in the film industry, so we hope that our solution will have relevance in the near future.

2. Previous work

Geometry reconstruction based on depth images has a wide literature. Fusion based techniques [5] continuously update a volumetric representation of the scene by fusing the images taken from different viewpoints. It has been successfully used for Kinect data [10], but the reconstructed geometry must be static, and the size of the object is also limited. Size and resolution limitations were addressed by using hierarchical volumes [17] and dynamically caching large scenes [3]. However volume representation has high memory needs and the volume should be converted to a triangle mesh for traditional rendering pipelines. Fusion techniques can only be used for static scenes, so not applicable for our purposes.

Though reconstructing a point cloud from a depth map is trivial, the noise and bad resolution of range cameras necessitate the enhancement of the depth map. Depth upsampling has a wide literature [6]. The most commonly used techniques are bilateral filtering with region growing to fill the holes of invalid pixels [4]. Joint bilateral filtering upsamples the depth map while reconstructs object boundaries based on the high resolution color image [1]. These techniques usually also smooth the noise of depth maps.

With an upsampled and noise reduced depth map in our hand, geometry reconstruction is much more precise which

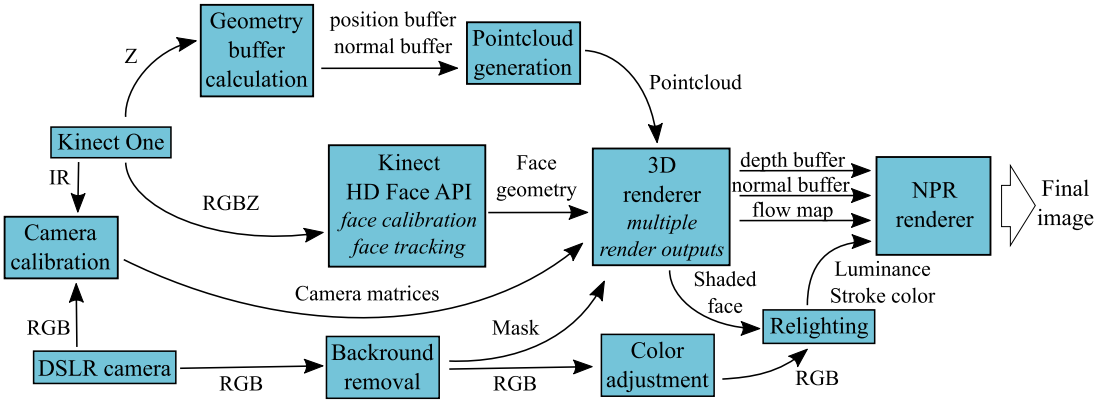

Figure 1: *The main components of our algorithm.*

enables video relighting. Many techniques were proposed that used the Kinect sensor for relighting and placing a recorded user in a virtual environment [12]. Many of these can reconstruct the surface color of the subject [14] or ever perform more precise BRDF reconstructions [19]. Our approach also uses the depth map to reconstruct the geometry of the captured scene, and we also use a simple relighting technique.

Stylistic rendering also has a large literature. NPR methods can be grouped to object space and image space methods. Object space methods use stylistic texturing [13] or generates strokes on object surfaces [18]. Screen space stroke based methods directly work in image space, which ensures even screen space stroke density, but temporal coherency is a great issue. Latest solutions use examples drawn by artists and try to transfer these styles to the final animations [2]. Temporal coherency is also a great issue here as well as the distortion or blurring of the transferred style. Neural network based methods were successfully used for still images [9] and were specialized for stylization of human faces [15]. Texture synthesis based techniques are more controllable and faster than neural network based solutions [7] and were also extended to specially handle human faces [8]. Style transfer and synthesis methods give great freedom for various styles, but can not provide realtime performance and usually not suitable for animations.

The NPR technique we use is a screen space stroke based method that is suitable for displaying hatching or brush based styles [11]. Though it works in screen space it requires buffers that samples the underlying geometry. These buffers (normal, depth and flow buffers) are easy to generate with a 3D renderer, but usually not present in case of real-life video footage. In this paper we investigate the possibility to use RGBZ data of the Kinect sensor as an input of this technique. We also use human face tracking and face geometry reconstruction to improve the quality at facial areas. We should note that recently a similar idea, namely face detection based image segmentation, was used in style synthesis for videos of animated human faces.

3. Overview

The main components of our method are shown on Figure 1. We use two capture devices: a Kinect RGBZ camera and a Canon entry level digital single lens reflex camera (DSLR). We used the second camera because the Kinect has a fixed field of view (horizontally 60 degree), which is large enough to capture a full body from a two meter distance, but not efficient for facial recording. Moving closer to the Kinect doesn't help, as depth sensing only works from a minimum half meter distance, and moving close to a camera with large view angle results distorted faces. On the other hand the Canon is an interchangeable lens camera, so one can choose a lens with proper focal distance. We used an 50mm lens, and as our camera has a cropped format sensor it leads to 27 degree vertical field of view.

We recorded our input videos in front of a green screen, which helped us removing unnecessary background elements. We used continuous studio lighting to properly expose the green background and the subject, which enables smaller aperture and ISO values and higher shutter speed which reduces sensor noise and blur. Our setup is shown on Figure 3. Using two separate cameras requires proper camera calibration as the depth values of the Kinect sensor will be reprojected to the DSLR camera space. To reduce the amount of shadowing the two camera's nodal point should be placed as near as possible. It is also beneficial to rotate them to the same view direction.

Reprojection is based on point cloud generation from the depth image and rendering it with the camera parameters of the DSLR camera. The Kinect SDK has a face tracking and face geometry extracting feature, which will be used to replace the noisy facial part of the point cloud. This face geometry is also rendered from the point of view of the DSLR

Figure 2: *Image from the color camera, its green screen mask, and the image after background removal.*

camera and combined with the reprojected images. During rendering we store not only color but depth and normal information too which will be the input of our NPR rendering module. Additionally we can also perform relighting on our face model, thus we can use a clear real world lighting but a dramatic virtual lighting at the end.

Figure 2 shows our DSLR capture, its background mask and the removed green background. This mask is saved as it will be used later. The main steps of our workflow are explained in details in the following sections.

Figure 3: *Our camera setup using a Kinect One and a DSLR.*

4. Depth reprojection

To register the depth image from the Kinect and the color image from the DSLR we need to reproject the depth image to the image plane of our RGB camera. This can be done with generating a point cloud from the depth image and rendering it with the camera transformation and projection matrices of the RGB camera. The depth map returned by the Kinect, showed on Figure 5, stores Z coordinate values defined in the camera space of the Kinect's infra red camera. If we know the projection matrix of this camera, we can find the camera space position for each pixel using the pixel coordinates and this 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 position buffer.

Using the position buffer we estimate geometry normals with finite differences:

$$dx = Position_{x+1,y} - Position_{x-1,y}$$

$$dy = Position_{x,y+1} - Position_{x,y-1}$$

$$N = \frac{dx \times dy}{|dx \times dy|},$$

where $Position_{x,y}$ is the value stored in the position buffer under pixel coordinates x, y. The resulting normal vector values can also be written to an image called normal buffer. The

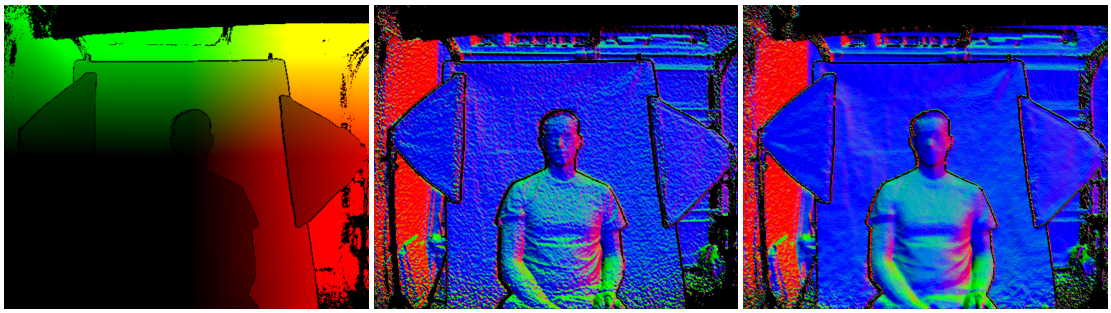

Figure 4: *Position and normal buffers reconstructed from the depth image and the smoothed normals using edge preserve filtering.*

Figure 5: *Depth output of the Kinect One.*

position and normal buffer together is called geometry buffer as it stores all necessary geometric information. The normal buffer calculated this way is rather noisy due to the noise of the depth map.To reduce this, we used a simple edge preserving blur filter on the depth map to smooth out most of the noise. This filter is similar to a bilateral filter, but uses importance sampling thus no spatial weighting is needed, and for the color difference weighting we used a triangle function instead of a Gaussian. Figure 4 shows the position and normal buffer before and after depth smoothing.

Each pixel of the geometry buffer defines a point in camera space with a camera space normal vector, thus the buffer defines a point cloud. We can visualize this by rendering a point primitive for each pixel. Figure 6 shows the point cloud in the 3D virtual space using normal vectors as colors. If we know the view and projection matrix of the DSLR camera, we can render the point cloud from the DSLR point of view. We calibrated the Kinect and the DSLR cameras together, thus we know the transformation matrix that transforms from the Kinect camera space to the DSLR camera space. We also now the intrinsic parameters of the DSLR, thus its projection matrix can be defined. For the near and far plane we should choose values that enclose the point cloud.

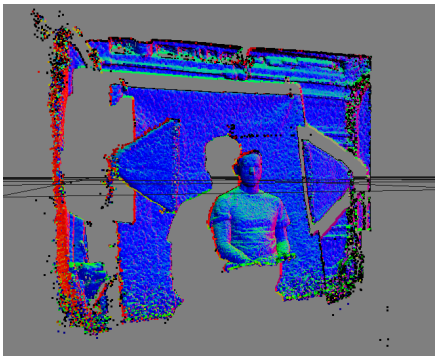

Figure 6: *Point cloud generated from the depth image using its normals as color.*

The first column of Figure 7 shows the reprojected depth and normal vectors. A shadowing caused by the nodal point difference of the two cameras can clearly be seen. This shadow does not disturb us too much as we would like to remove the background, thus using the background mask calculated from green screen keying the background depth and normals can be removed as shown on the second column. It can be seen that fine details are lost due to the low effective resolution of the projected depth map and the smoothing filter used for noise reduction.

5. Replacing the face geometry

For our NPR effect to work correctly we need to reconstruct the true geometric normals preferably with high details. This cannot be done with enhancing the reprojected depth map as it does not contain enough information. On the other hand we know that a significant area of the image contains a human face. Modern face tracking methods usually work with a three dimensional geometry of the tracked face, and this geometry is constructed from an average face model by the

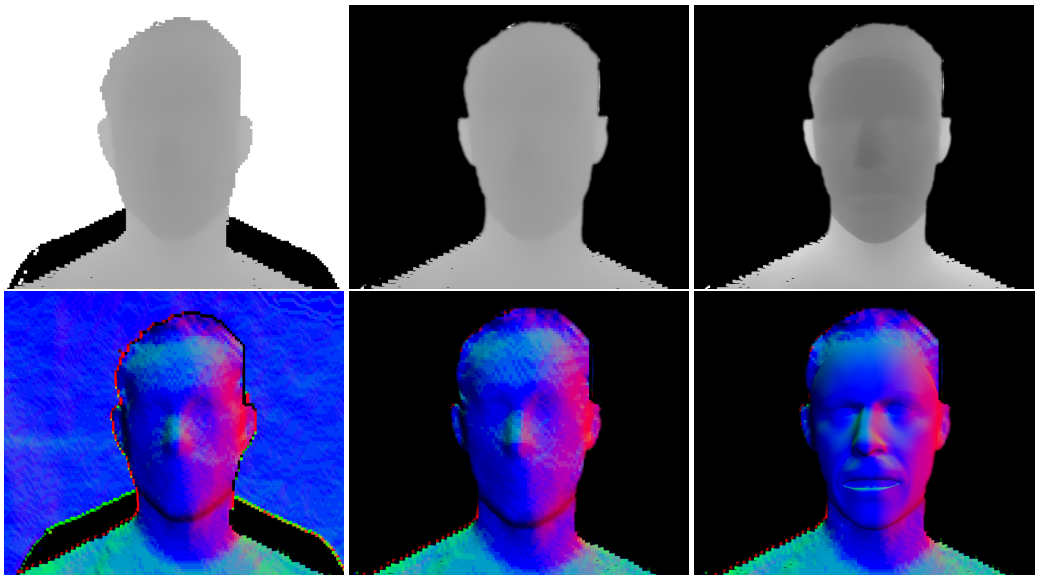

Figure 7: *Projected depth (first row) and normals (second) in the DSLR screen space. The first column shows the projected values, the second column shows the result of background removal, while the third column shows the final buffer with the tracked face geometry.*

tracker itself. The user specific face model with the current tracked expression can usually be acquired as an output of the tracker.

The Kinect also has a face tracking module, and as it works from an RGBZ data instead of color only, it can produce quite detailed results. The Kinect SDK has a low resolution (LDFace) and a high resolution (HDFace) face module. As the high resolution module is also realtime, we used the HDFace interface of the SDK. Using the HDface module requires a face calibration step. Without this step only the face orientation can be retrieved stably, the geometry will contain information about an average human face. The calibration step is easy and fast: the user should look straight to the camera, look to both sides and finally should slightly tilt up its head. If the capture of these main viewpoints was successful the module can reproduce the user's facial geometry. It is important to have a high frame rate during the calibration step, so it should be run as a separate process.

After the calibration is done the face tracking module can give back the exact face geometry of the user for each frame. The geometry is defined in the Kinect depth camera frame, the same frame we defined our point cloud in. To replace the face areas of the depth buffer we only have to render the face geometry together with the point cloud and write its depth values. We should note that a slight offset in the camera z direction should be used to overcome z-fighting artifacts. Normal vectors are not provided by the HDFace interface, but as the geometry is given by an indexed triangle list, adjacency information is present, so smooth normals can be calculated efficiently. Figure 8 shows the face geometry of one frame and combining it with the point cloud. The third column of Figure 7 shows the projected depth and normal values with the face area replaced by the HDFace geometry.

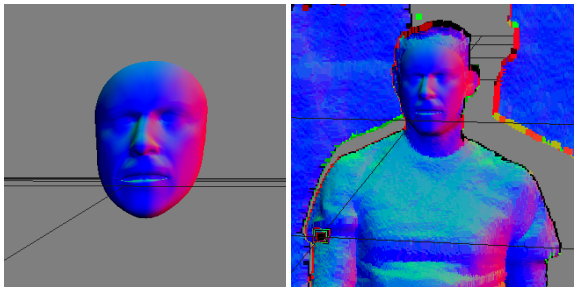

Figure 8: *The geometry returned by the Kinect HDFace face tracker and rendering it together with the point cloud.*

6. NPR renderer

Our NPR renderer is a stroke based screen space method. It places hatching strokes with even density in screen space and reduces this density in areas where lighter tones needed according to lighting. The input of the algorithm is a set of buffers, namely: luminance buffer, depth buffer, normal buffer, flow map. Luminance is used for toning, depth and normals are used to orient and bend the lines according to the underlying geometry. The algorithm orients the lines to the principal curvature directions, which can be calculated from

Figure 9: *From left to right: a darkened version of the color image, the lit face geometry, the relighted color image, NPR rendering the relighted image with a pencil style.*

the second order derivatives of the depth map or the first order derivatives of the normal map. By providing a smooth normal map we can eliminate large constant eras in curvatures caused by linear depth interpolation.

Normal and depth maps can be passed to the NPR renderer without modifications, but preparing the luminance map needs attention. Hand drawn art usually uses contrast dramatic lighting, but to record a movie in high quality and to perform stable facial tracking we need a bright even lighting. Thus the RGB image from the DSLR should be post processed. The simplest method is to use a brightness and contrast filter to make the lighting more plastic. However we tried a more complex solution that gives more freedom to the user in final lighting.

Relighting an image is a common task in photography retouching and in movie visual effects. These methods add lighting with a falloff within a ellipse to imitate positional light sources. However as the underlying geometry is not known the light bleeds through object borders. We can use masks to reduce this artifact but drawing masks in each frame automatically is yet an unsolved problem. Even with mask the light reflectance is not realistic as the position and normal of the pixels are not known. However as we know the geometry of the face, so we can make a much realistic relighting. We render the face geometry with a diffuse white BRDF using one light source defined in Kinect camera space and add the result to the luminance map. As our original lighting was even and bright first we make it darker and increase contrast to imitate ambient lighting before adding our virtual lighting to it. Figure 9 shows the adjusted luminance map, the face model rendered with a single light source illuminating from the side, and the combined luminance map. The position, direction and color of the light source is defined by the user, and more virtual light sources can be used.

Our NPR renderer uses a flow map as an input to move the lines with the surfaces to avoid the so called shower door effect. With only a single RGB camera flow can be calculated with an optical flow algorithm. Unfortunately the results of such methods are not perfect: for large constant areas it fails to calculate the flow, and even on characteristic areas of the image the result can be noisy or inaccurate. These imperfections become obvious when using our NPR renderer as strokes looks like sliding on the surface.

As the topology of the HDFace geometry does not change, we can calculate camera space movement vectors for each vertex using the position in the previous frame. This movement can be transformed to screen space movement and written to a flow buffer. Using the normal, depth and flow buffers our strokes move with the surface and follow its geometry on face areas. On the remaining part of the image our geometry data is imperfect which causes sudden line rotations, flickering and sliding on the surface. As when we are looking at such a video our eye is focusing on the face, these artifacts are not as disturbing as they would be on the facial areas. The rightmost image of Figure 9 shows a rendering with a pencil drawing style using our technique.

Figure 10 shows renderings with (left) and without (right) using the tracked face geometry. The orientation of lines can follow the underlying geometry with the new technique quite well, while using normals calculated from the depth map results in chaotic stroke directions. Figure 11 shows NPR rendering of two different expressions.

7. Implementation

We integrated our method in a form of computing nodes in an interactive, node based compositor software that uses OpenGL for acceleration [16]. All image operators are implemented as full screen quad renderings where the fragment shader does the processing. To create the point cloud we render a point primitive for each pixel of the Kinect depth buffer, and a geometry shader transforms it to its place or discards invalid pixels. All host codes were written in C++, we used the Kinect SDK and the CanonSDK to handle our input devices.

8. Conclusions and future work

We created a system that can process a video of a human face and outputs a stylized version. We used a Kinect One and a DSLR camera to reconstruct the geometry based on RGBZ input. The Kinect FaceAPI was used to replace the facial areas with a tracked face geometry that is free of noise and can produce accurate flow values. Our screen space NPR renderer uses the depth, normal, illumination and flow buffers and calculates a stylized stroke based output image with temporal coherency. We also used the face geometry to relight the face in post. All calculations were accelerated on the GPU and the entire system runs at interactive frame rates: around 10-15 frames per second.

Our system is still in experimental stage but the results are pleasing. In the future we would like to achieve more precise camera calibration as some misalignment was noticeable. We would like to use a smoother blending of the face geometry to reduce the sharp edges around the face in the final image. We also investigate methods to enhance the non-facial parts of our geometry buffers, as they are still affected by a large amount of spatial and temporal noise. This noise ruins temporal coherency of stroke directions, and produces false relighting on these areas. Optical flow calculation is also a problem for the point cloud geometry. Our relighting method assumes even lighting on the original video, which we could approximately achieve within our controlled environment. However surface albedo should be extracted using inverse rendering techniques, which would make relighting more accurate and gives more freedom in model lighting.

9. Acknowledgment

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. Bogumil Bartczak and Reinhard Koch. Dense depth maps from low resolution time-of-flight depth and high resolution color views. In *Proceedings of the 5th International Symposium on Advances in Visual Computing: Part II*, ISVC '09, pages 228–239, Berlin, Heidelberg, 2009. Springer-Verlag.

2. Pierre Bénard, Forrester Cole, Michael Kass, Igor Mordatch, James Hegarty, Martin Sebastian Senn, Kurt Fleischer, Davide Pesare, and Katherine Breeden. Stylizing animation by example. *ACM Trans. Graph.*, 32(4):119:1–119:12, 2013.

3. Jiawen Chen, Dennis Bautembach, and Shahram Izadi. Scalable real-time volumetric surface reconstruction. *ACM Trans. Graph.*, 32(4):113:1–113:16, July 2013.

4. L. Chen, H. Lin, and S. Li. Depth image enhancement for kinect using region growing and bilateral filter. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, pages 3070–3073, Nov 2012.

5. Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 303–312, New York, NY, USA, 1996. ACM.

6. Iván Eichhardt, Dmitry Chetverikov, and Zsolt Jankó. Image-guided tof depth upsampling: A survey. *Mach. Vision Appl.*, 28(3-4):267–282, May 2017.

7. Jakub Fišer, Ondřej Jamriška, Michal Lukáč, Eli Shechtman, Paul Asente, Jingwan Lu, and Daniel Sýkora. StyLit: Illumination-guided example-based stylization of 3d renderings. *ACM Transactions on Graphics*, 35(4), 2016.

8. Jakub Fišer, Ondřej Jamriška, David Simons, Eli Shechtman, Jingwan Lu, Paul Asente, Michal Lukáč, and Daniel Sýkora. Example-based synthesis of stylized facial animations. *ACM Transactions on Graphics*, 36(4), 2017.

9. L. A. Gatys, A. S. Ecker, and M. Bethge. Image style transfer using convolutional neural networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2414–2423, June 2016.

10. Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, and Andrew Fitzgibbon. 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, pages 559–568, New York, NY, USA, 2011. ACM.

11. Zoltán Lengyel, Tamás Umenhoffer, and László Szécsi. Realtime, coherent screen space hatching. In *VII. Magyar Számítógépes Grafika és Geometria Konferencia*, pages 131–137, Febr 2014.

12. Yohei Ogura and Hideo Saito. *Relighting for an arbitrary shape object under unknown illumination environment*, volume 9475 of *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pages 433–442. Springer Verlag, 2015.

13. Emil Praun, Hugues Hoppe, Matthew Webb, and Adam Finkelstein. Real-time hatching. In *Proceedings of SIGGRAPH 2001*, pages 579–584. ACM Press, 2001.

14. T. Richter-Trummer, D. Kalkofen, J. Park, and

Figure 10: *Comparison of stroke directions using our technique (left) and using the normal buffer only (right).*

Figure 11: *Two NPR renderings using our technique showing two different facial expressions.*

D. Schmalstieg. Instant mixed reality lighting from casual scanning. In *2016 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 27–36, Sept 2016.

15. Ahmed Selim, Mohamed Elgharib, and Linda Doyle. Painting style transfer for head portraits using convolutional neural networks. *ACM Trans. Graph.*, 35(4):129:1–129:18, July 2016.

16. L. Szirmay-Kalos and L. Szécsi. General purpose computing on graphics processing units. In A. Iványi, editor, *Algorithms of Informatics*, pages 1451–1495. MondArt Kiadó, Budapest, 2010. http://sirkan.iit.bme.hu/s̃zirmay/gpgpu.pdf.

17. László Szirmay-Kalos, Balázs Tóth, and Tamás Umenhoffer. Efficient voxel marking for hierarchical volumetric fusion. In *Proceedings of the 37th Annual Conference of the European Association for Computer Graphics: Posters*, EG '16, pages 9–10, Goslar Germany, Germany, 2016. Eurographics Association.

18. Tamás Umenhoffer, László Szécsi, and László Szirmay-Kalos. Hatching for motion picture production. *Comput. Graph. Forum*, 30(2):533–542, 2011.

19. Hongzhi Wu and Kun Zhou. Appfusion: Interactive appearance acquisition using a kinect sensor. *Computer Graphics Forum*, 34(6):289–298, 2015.