Cel-Shading with OSL in RenderMan 21 for Maya

Francesca Galluzzi

August, 2017

Bournemouth University

MSc Computer Animation and Visual Effects

Cel-Shading with OSL in RenderMan 21 for Maya

Francesca Galluzzi

August, 2017

Francesca Galluzzi

Cel-Shading with OSL in RenderMan 21 for Maya

August, 2017

Bournemouth University

MSc Computer Animation and Visual Effects

Abstract

Cel-shading is a particular type of non-photorealistic rendering that consists of representing a 3D scene in a 2D style similar to the comic books or animes one. After an in-depth analysis of previous research in the field and currently available technologies, a practical solution is proposed for a simple and efficient shader written in OSL and integrated into the RenderMan for Maya environment.

Contents

1 Introduction

Non-photorealistic rendering (NPR) has been subject of studying for numerous year. As opposed to photorealistic rendering, the goal of NPR is to produce a stylised image, whether it is in the style of a drawing, a painting, a comic book or any other art style. Generally, photorealistic rendering is the preferred choice in productions which caused the research in NPR rendering to be slowed down. In later years, however, great advances have been made in this field so that NPR can finally be considered a valid option for a quality productions. The main problems that need to be solved when working with stylised images are the possibility for artists to fully customise their style, without any sort of limitations, while, at the same time, keep the process as automated as possible in order to have affordable production times. This last aspect is what caused NPR to fall in the background for a long period of time, as there is not yet a complete technology that responded to such needs.

Cel-shading is a particular type of non-photorealistic rendering that consists of representing a 3D model, or any 3D scene, as a flat plain surface, using two or three tones to describe the shape of the represented objects, namely a basic tone that represent the general colour of the object, a darker tone for the shadows and, eventually, a lighter tone for the highlights. This shading technique is often paired with an edge detection system that defines the outline and all the necessary lineart that is required to complete the final look. Therefore, cel-shading is used to reproduce the style of comic books or traditional 2D animations, like animes.

The goal of this project is to create a simple tool for cel-shading that easily integrates into the RenderMan environment, specifically into RenderMan for Maya, as this is one of the most commonly used softwares for 3D animation. The main requirements on which the entire development will focus are simplicity of use for the end user, mainly the look-dev artist, and efficiency in terms of render quality and render times.

In the next section, it will be presented some previous research in the field of NPR and cel-shading followed by a section dedicated to the explanation of the technology related to the project itself. Afterwards, an entire section will be dedicated to the description and analysis of the produced project and finally there will be some

concluding thoughts on the project, on some current problems to be fixed and on how it can be expanded with more functionalities.

2 Previous Work

Initial research in NPR can be traced back as far as 1999, in the paper proposed by Costa Sousa M. and Buchanan J. W. [SB99]. Starting from previous works in the modeling of traditional artistic media and styles, they proposed a graphite pencil 3D rendering system defined in four steps. The first one is simulating the drawing material, like the effect of the graphite on the paper. Secondly, they focused on modeling the drawing primitives, such as the individual types of strokes and marks, in order to create the basic texture. Third step is simulating the basic techniques used by artists and illustrators for creating art pieces in pencil and finally they modeled a control mechanism for the final drawing composition.

A new framework for image processing was later proposed by Hertzmann A. et. al. [Her+01], using a process called “image analogies”. The objective of this framework is to produce a wide variety of filters for image manipulation based on some input images defined by the user. These input images represent the desired initial state and end result so that the same end result can be applied to other images in similar initial states. The final effects that can be produced vary from tradition blurring to high quality texture production, from artistic filters to the production of higher resolution images.

DeCarlo D. et. al. [DeC+03], on the other hand, focused their research on how to better convey shapes using lines. They proposed the use of suggestive contour in addition to the basic contours and creases. Suggestive contour is defined as the set of lines drawn on clearly visible parts of the surface, as an extension of the true contours of the object. The paper presents two methods for computing these lines, in particular an algorithm that finds the zero crossings of the radial curvature

Some years later, Barla P. et. al. [Bar+06] developed a toon shading technique based on 2D textures. These textures contains the traditional information about colours and lights in one dimension and the desired tone detail, that depends on the depth or surface orientation, in the second dimension. The result is the possibility to simulate effects such as levels-of-abstraction, aerial perspective, depth-of-field, backlighting, and specular highlights. Moreover, they presented a solution for the simplification of the normals by interpolating the original normal field with the one generated by an abstraction of the original shape.

Bringing the focus back to the implementation of the optimal edge definition, Lee, Y. et. al. [Lee+07] described a GPU-based algorithm for rendering a 3D model as a line drawing, based on the insight that a line drawing can be understood as an abstraction of a shaded image. The basic idea is to use the tone boundaries obtained with the toon shading to define where to draw the highlight lines and the dark boundaries. The lines produced by the method can include silhouettes, creases, and ridges, along with a generalization of suggestive contours that responds to lighting as well as viewing changes.

Whited B. et. al. [Whi+12] from Walt Disney Studios broke down barriers in the NPR filed when they presented the Meander Animation Tool. This system was designed to combine the strengths of CG animation, like temporal coherence, spatial stability, and precise control, with traditional animation's expressive and pleasing line-based aesthetic. The power of this tool is the ability for artists to draw over simple renders a set of lines that are then automatically animated to match the initial 3D animation using vector fields derived from the animation itself.

Bénard P. et. al. [Bén+13] around the same time proposed a different tool for the production of stylised animations. The main problem that their research focused on was keeping temporal coherence during the animation of stylised scenes. Extending the work proposed by Hertzmann A. et. al. [Her+01], the final tool allowed the artist to manually paint a set of keyframes of an animation and then automatically interpolate the intermediate frames in order to still maintain stylistic continuity.

Continuing the research on the subject of stylised animations, Bénard P. et. al. [Bén+14] also proposed a method for accurately computing the visible contours of a smooth 3D surface. The proposed approach is to generate, for each viewpoint, a new triangle mesh with contours that are topologically equivalent and geometrically close to those of the original smooth surface. The main problem tackled in this paper is the consistency of contours, so a new type of triangle is introduced to solve it.

3 Technical Background

3.1 OSL

The code for the shader has been developed following the official Open Shading Language 1.8 language specification, released in February 2017 [Gri17].
Both the OSL files are shaders of type `surface`, which is a type of shader that determines the basic material properties of a surface and how it reacts to light. In the particular case of this project, it was necessary to compute the orientation of the surface in relation to the light direction or the viewing direction in order to determine the final colouring of the surface itself.

OSL comes with a set of predefined variables and functions that easily allowed to define and implement the computation algorithm.
The table is Figure 3.1 shows the list of global variables available inside the shaders.

Variable	Description
`point` **P**	Position of the point you are shading. In a displacement shader, changing this variable displaces the surface.
`vector` **I**	The *incident* ray direction, pointing from the viewing position to the shading position `P`.
`normal` **N**	The surface "Shading" normal of the surface at `P`. Changing `N` yields bump mapping.
`normal` **Ng**	The true surface normal at `P`. This can differ from `N`; `N` can be overridden in various ways including bump mapping and user-provided vertex normals, but `Ng` is always the true surface geometric normal of the surface at `P`.
`float` **u**, **v**	The 2D parametric coordinates of `P` (on the particular geometric primitive you are shading).
`vector` **dPdu**, **dPdv**	Partial derivatives $\partial P/\partial u$ and $\partial P/\partial v$ tangent to the surface at `P`.
`point` **Ps**	Position at which the light is being queried (currently only used for light attenuation shaders)
`float` **time**	Current shutter time for the point being shaded.
`float` **dtime**	The amount of time covered by this shading sample.
`vector` **dPdtime**	How the surface position `P` is moving per unit time.
`closure color` **Ci**	Incident radiance — a closure representing the color of the light leaving the surface from `P` in the direction `-I`.

Fig. 3.1: The Figure shows the setup of the shading network in Maya

The variable `normal N`, being the normal of the surface in each shading point, represents the orientation of the surface and can be directly compared with the light direction vector. The variable `vector I`, instead, represents the camera vector, that is the viewing vector also to be compared with the surface normals.
In terms of predefined functions, two are particularly noteworthy.

```
float dot (vector A, vector B)
```
Returns the inner product of the two vectors, i.e., $A \cdot B = Ax\,Bx + Ay\,By + AzCz$

The **`dot`** function is used to compute the dot product between two vectors. When comparing a surface normal with the light direction vector, for example, this function allows to verify how similar or different the two vectors are.

```
type step (type edge, type x)
```
Returns 0 if $x <$ *edge and 1 if* $x \geq$ *edge.*

Once the dot product is defined, the obtained values is then compare to a threshold value using the **`step`** function to determine which category the surface belongs to.

3.2 RenderMan for Maya (RfM)

From the official RenderMan documentation [Stu17], it reads that RenderMan for Maya (RfM) provides easy access to RenderMan features via either a Maya-centric workflow (including Maya-style pattern nodes, etc.) or in conjunction with the other tools. The plugin also connects Maya with RenderMan Pro Server, through its RIB-out functionality and advanced integration with the renderer. RfM provides seamless access to RenderMan's speed, power, and stability for Maya users, who can maintain a simple, Maya-centric workflow, or take advantage of the plugins' flexibility to create an optimal, customized pipeline.

The main most interesting feature, in the case of this project, is the possibility of easily integrating OSL shaders into the shading network and take advantage of the predefined Maya nodes to take care of the more tedious aspects of the look development process. This allows to focus the attention on the development of the core of the shading algorithm rather than understanding and developing the correct tools for the deployment of the shader onto the desired geometry.

4 Proposed Solution

The goal of the project was to offer an easy to use tool for toon shading, with few key parameters accessible to the end users for refinements, but also to guarantee efficiency and low render times.
To achieve the first requirement, the core technology of the shader has been developed using OSL. OSL shaders can be integrated into various renderers, but with the latest release of RenderMan for Maya, version 21.5, it is actually possible to register OSL files as Maya nodes and access them in the material presets. Moreover, by registering the node, it is possible to take advantage of the shader metadata to create a custom user interface for the plug-in. Unfortunately, the latest version that was available on the local machine while developing this project was RenderMan for Maya version 21.3, so the OSL code has not been registered in Maya. The entire procedure, however, is described in detail by Christos Obretenov in the Official RIS Tech Series - Course #2 - Class 07 on the RenderMan Community Forum [Obr17].
For the second requirement, instead, of all the techniques discussed above for stylised rendering, the most simple ones have been chosen to be developed. The end result is not completely clean, but rendering times are noticeably short and could presumably be considered for a real time rendering environment.

4.1 OSL Code

The final look is defined by two OSL surface shader: one for the base colour and shading, that represent the actual cel-shading, and another for the black outline, to complete the comic like look.

For the first shader, the basic structure follows the one described by Junya Christopher Motomura used in the look development of the game Guilty Gear Xrd [Mot15]. A simple step function is used to evaluate whether a point on the surface of the object is lit by the light in the scene of not. The comparison is done by calculating the dot product of the normal of the surface at that point and the light direction vector. The result is then compared to a threshold value set by the user so that the shaded area can be customised in width. The smaller the threshold, the bigger the shaded area is.
Furthermore, the same dot product is used to identify the areas on the surface whose

normal is similar in direction with the light vector. Those areas are the highlighted areas and, as well as with the shaded areas, the user can personalise the threshold between a standard surface and an highlighted one. The smaller the threshold, the smaller the highlighted area is. Below it is presented the pseudo code for the cel-shader.

```
1: if STEP(highlightThreshold,DOT(LightDirection, N)) then
2:     Cout ← (1 − BaseColor) ∗ HighlightIntensity
3: else
4:     if STEP(shadowThreshold,DOT(LightDirection, N)) then
5:         Cout ← BaseColor
6:     else
7:         Cout ← BaseColor ∗ ShadeIntensity
```

The outline shader is constructed following the idea proposed by Michel Anders on his blog [And12] on how to write a simple toon shader in OSL. Similarly to the base colour shader, the outline is generated by comparing each surface normal to the incident ray vector, represented by the global variable `vector I`. If the angle between the normal of the surface in a point and the camera vector is greater than a threshold value set by the user, then the shading point is considered part of the outline and is coloured in black. Otherwise, the original colour of the surface is used. Below it is presented the pseudo code for the outline shader.

```
1: if STEP(OutThreshold,DOT(-I, N)) then
2:     Cout ← BaseColor
3: else
4:     Cout ← 0
```

4.2 Shading network setup

The final step to complete the toon shading is integrating the OSL files into the Maya scene using the RenderMan infrastructure.
As shown in Figure 4.1, the OSL files are loaded into the shading network using the PxrOSL plug-in. This utility node automatically reads inputs and outputs of the shader and adapts the interface accordingly. The two shaders are then mixed together using the PxrBlend node. The inputs are fused in one single colour output using the `multiply` operation so that the black outline prevails over the base colour whereas everything else remains unaltered. A valid alternative to obtain the same result could have been connecting the `Cout` output of the base colour shader to the `Base Color` input of the outline shader and then using the `Cout` output of the outline shader as the final RGB result.
Either way, the final result is then connected to the PxtConstant material plug-in. The reason for choosing this material is that any path tracing computation of the

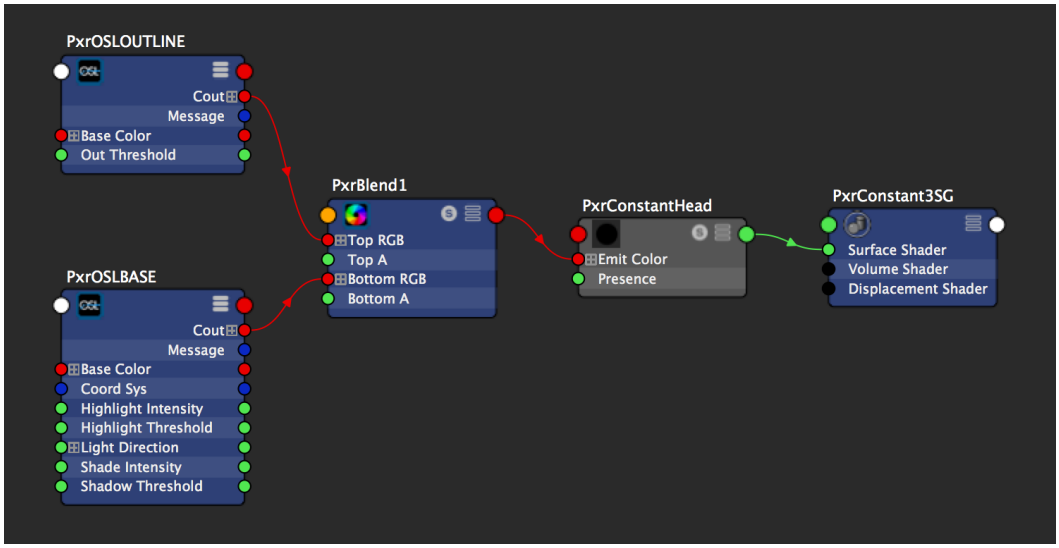

Fig. 4.1: The Figure shows the setup of the shading network in Maya

light in the scene is unwanted since only the global light direction in the scene is necessary and it is already used inside the OSL file.
In case of a texture being prepared for the 3D model, it can be used to map the base colour of each shading point of the object when computing the cel-shading. The texture can be brought in the shading network with a standard File node and then connected to the `Base Color` input of the toon shader. The shader will automatically brighten up or darken the necessary areas using the texture as its starting point.

5 Conclusion

The final product respects the original goals set for the project, that were simplicity of use and efficiency. Moreover, the final aesthetics is comic book like, as expected from a toon shader. The images below demonstrate the results achieved on a plain 3D model, in Figure 5.1, and on a textured 3D model, in Figure 5.2.

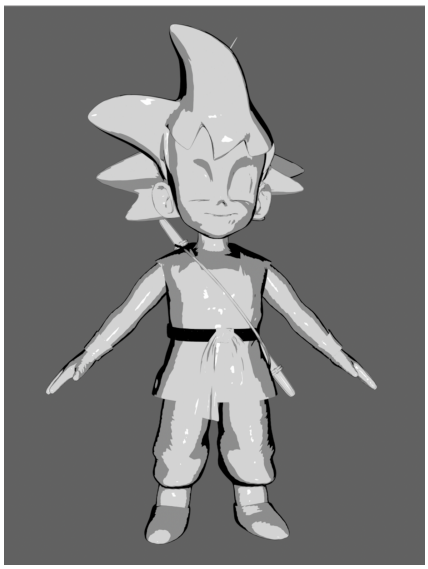

Fig. 5.1

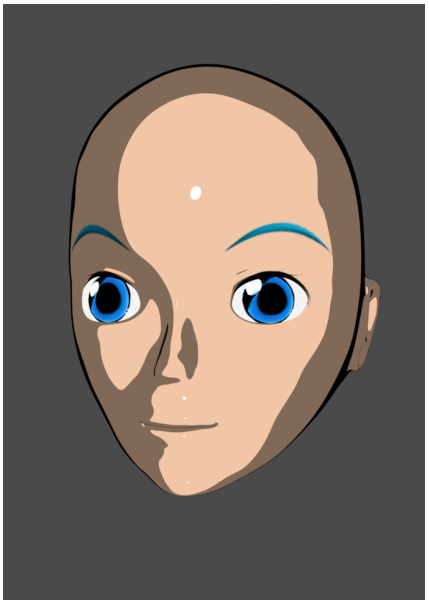

Fig. 5.2

The shader has been tested on simple animations as well and the results are demonstrated in the attached video.

5.1 Improvements

The most obvious improvement that can be applied to project is to update to the latest version of RenderMan for Maya in order to be able to register the shader as a plug-in amongst the RenderMan materials. The advantages of this operation are that it becomes easier to set up the scene with the toon shader and that it would be possible to customise the interface of the plug-in. Thus, the OSL code should be updated to include the metadata required to produce the desired user interface.

A second improvement can be applied in regard to the control of the light di-

rection vector. Currently, it is only possible to manually enter the values of the vector's components, which is not intuitive from the user point of view. A simple workaround for this problem can be connecting the values of the light direction vector's components to the values of the amounts of rotation around the axis of a light placed in the scene. However, this process is quite tedious and requires particular attention in correctly matching the values. For these reasons, automating the process of reading the light direction vector directly in the shader code is an advancement in the usability of the tool itself.

5.2 Further Developments

New functionalities can also be considered to be implemented inside the shader to further expand the tool and obtain cleaner results.

One problem consists of having an unwanted amount of details in the shaded areas. Very often it is desired that the shadows have a simpler shape even if the shaded object has a complex geometry.
This issue has been dealt with in the development of the the game Guilty Gear Xrd [Mot15]. Their solution can be narrowed down to manually control the direction of the surface normals in each point of the geometry until the desired result is achieved. Another more automated solution has been proposed by Gulbrandsen O.[Gul10]. It consists of creating an invisible and more simple geometry around the object that has to be shaded and using the surface normals of the shell geometry to average the directions of the original surface normals. The result is shown in Figure 5.3 and the referenced paper also proposes a practical implementation in OSL.

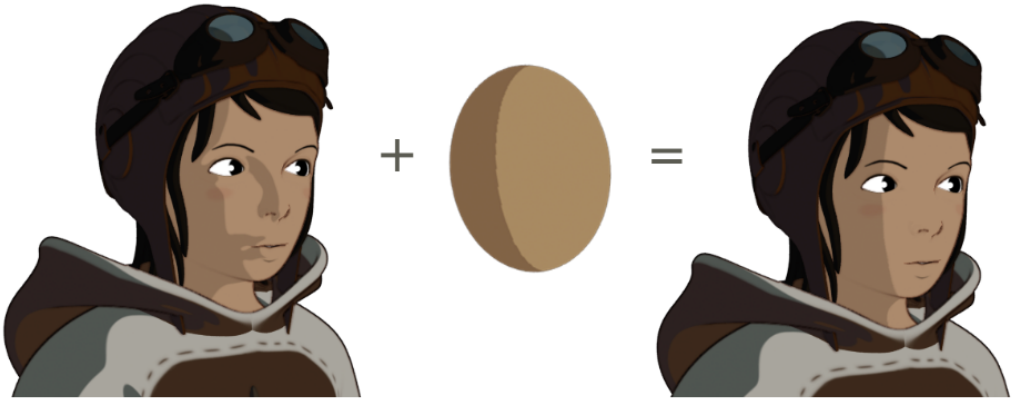

Fig. 5.3

A second aspect that heavily influences the aesthetic of the end result is the quality of the 2D-like lineart. The current project only traces the external outline, also defined as the general silhouette, of the shaded object, but often this is not enough to fully

describe the shape of the object itself so internal lines need to be defined.
There are various possible solutions available to solve this problem. In the game Guilty Gear Xrd [Mot15] they decided to map a texture for the lineart to the UV space of the model so that the artist could manually modulate the line in each point by controlling its width in each point. Using the texture in the UV space instead of directly applying it to the model also removes the need for a high resolution texture necessary for close ups. As before, this solution is heavily based on the manual tweaking of the parameters from the artists, which is very time consuming and tedious.
There have been studies that focused on the development of automated algorithms for extracting the internal lines and some of these algorithms have been presented in the 2008 SIGGRAPH Line Drawing class [Rus+08]. However, Davies M. [Dav15] conducted an in-depth research on the subject and produced a RIS Integrator that is capable of detecting edges. Being a project developed in the same framework as the toon shader proposed in this thesis, it would be interesting to integrate the two projects into one and further expand toon shading in the RenderMan environment.

References

[And12] Michel Anders. *A Toon OSL Shader For Blender*. `http://blog.michelanders.nl/2012/11/a-toon-osl-shader-for-blender_93.html`. [Online; accessed 21-August-2017]. 2012 (cit. on p. 8).

[Bar+06] Pascal Barla, Joëlle Thollot, and Lee Markosian. „X-toon: an extended toon shader“. In: *Proceedings of the 4th international symposium on Non-photorealistic animation and rendering*. ACM. 2006, pp. 127–132 (cit. on p. 3).

[Bén+13] Pierre Bénard, Forrester Cole, Michael Kass, et al. „Stylizing animation by example“. In: *ACM Transactions on Graphics (TOG)* 32.4 (2013), p. 119 (cit. on p. 4).

[Bén+14] Pierre Bénard, Aaron Hertzmann, and Michael Kass. „Computing smooth surface contours with accurate topology“. In: *ACM Transactions on Graphics (TOG)* 33.2 (2014), p. 19 (cit. on p. 4).

[Dav15] Martin Davies. „A Practical Investigation of RenderMan RIS API“. MA thesis. Bournemouth University, 2015 (cit. on p. 13).

[DeC+03] Doug DeCarlo, Adam Finkelstein, Szymon Rusinkiewicz, and Anthony Santella. „Suggestive contours for conveying shape“. In: *ACM Transactions on Graphics (TOG)* 22.3 (2003), pp. 848–855 (cit. on p. 3).

[Gri17] Larry Gritz. *Open Shading Language 1.8 - Language Specification*. Sony Pictures Imageworks Inc., et al. 2017 (cit. on p. 5).

[Gul10] Ole Gulbrandsen. „Controlling the dark side in toon shading.“ In: *SIGGRAPH Posters*. 2010, pp. 121–1 (cit. on p. 12).

[Her+01] Aaron Hertzmann, Charles E Jacobs, Nuria Oliver, Brian Curless, and David H Salesin. „Image analogies“. In: *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM. 2001, pp. 327–340 (cit. on pp. 3, 4).

[Lee+07] Yunjin Lee, Lee Markosian, Seungyong Lee, and John F Hughes. „Line drawings via abstracted shading“. In: *ACM Transactions on Graphics (TOG)*. Vol. 26. 3. ACM. 2007, p. 18 (cit. on p. 4).

[Mot15] Junya Christopher Motomura. *GuiltyGearXrd's Art Style : The X Factor Between 2D and 3D*. `http://www.gdcvault.com/play/1022031/GuiltyGearXrd-s-Art-Style-The`. [Online; accessed 21-August-2017]. 2015 (cit. on pp. 7, 12, 13).

[Obr17] Christos Obretenov. *Official RIS Tech Series - Course #2*. https://community.renderman.pixar.com/article/1770/ris-educational-series-course-2.html. [Online; accessed 21-August-2017]. 2017 (cit. on p. 7).

[Rus+08] Szymon Rusinkiewicz, Forrester Cole, Doug DeCarlo, and Adam Finkelstein. *SIGGRAPH 2008 Class: Line Drawings from 3D Models*. http://gfx.cs.princeton.edu/proj/sg08lines/. [Online; accessed 21-August-2017]. 2008 (cit. on p. 13).

[SB99] Mario Costa Sousa and John W Buchanan. „Computer-Generated Graphite Pencil Rendering of 3D Polygonal Models“. In: *Computer Graphics Forum*. Vol. 18. 3. Wiley Online Library. 1999, pp. 195–208 (cit. on p. 3).

[Stu17] Pixar Animation Studios. *RenderMan for Maya*. https://rmanwiki.pixar.com/display/REN/RenderMan+for+Maya. [Online; accessed 21-August-2017]. 2017 (cit. on p. 6).

[Whi+12] Brian Whited, Eric Daniels, Michael Kaschalk, Patrick Osborne, and Kyle Odermatt. „Computer-assisted animation of line and paint in Disney's Paperman“. In: *ACM SIGGRAPH 2012 Talks*. ACM. 2012, p. 19 (cit. on p. 4).