RenderMan Bark Shader

Hannes Ricklefs d1132332

10th June 2005

1 Introduction

This Report outlines the outcome of the Major Animation Project that was taken as part of an MSc Computer Animation at the University of Bournemouth in 2005. The main aim of this project was to produce a photo realistic RenderMan shader representing tree bark. The main motivation behind the undertaking of this project was to learn procedural modeling and texturing as well as the development process of shader writing in general. In addition some parts of this project could be included in other projects and be reused in the authors Masters Project.

At first the idea was to create both an internal wood shader and an external Bark shader. However, the internal wood shader is already covered by the standard RenderMan shaders wood.sl and wood2.sl. Therefore, the author decided to concentrate mainly on the external representation of trees. This report is structured into different sections. The first section describes identified features of tree bark that will be in cooperated in the shader. The second sections describes current approaches from fellow researchers and the approach taken by the author. The third section gives speculation into different approaches that could have been applied to the same problem. The fourth section describes problems encountered during the development process and potential resolutions for these problems. The fifth section lists tools and scripts that were used to enhance the productivity during this project. Finally the last section gives concluding remarks about the undertaking of this project.

2 Wood Features

In order to create a photo realistic bark shader reference photographs were taken. As the main goal was to create a procedural shader that could be applied for many bark types, common features of bark had to be identified. Figure 1 shows the different identified, patterns by the author as well as Wang[11]. These patterns are as follows:

1. fracture Fig. 1 (a) - As a tree increases in girth, great tension on the bark can cause fractures which can be either vertical or horizontal (edges bending upwards)
2. furrowed cork Fig. 1 (b) - Deeply furrowed bark with a thick accumulation of cork cells
3. ironbark Fig. 1 (c) - Rough Bark becomes hard and compact
4. lenticel Fig. 1 (d) - Small oval rounded spots upon the trunk or branch of a tree, from which the underlying tissues may be protruded or cuppy. Lenticles are usually horizontal as shown.
5. tessellation Fig. 1 (e) - The bark fractures to form flakes or plates with deep furrows.

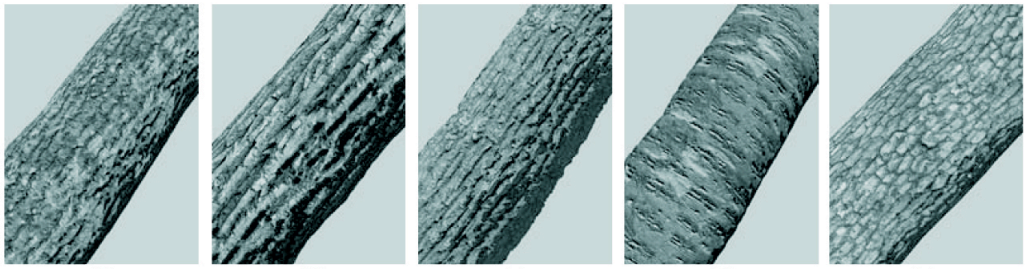


Figure 1: Bark Patterns

The author decided to focus mainly on patterns Fig. 1 (b,c,e) as the underlying pattern is very similar. In addition to the bark pattern, color had to be taken into consideration. In general the color of bark depends on many different aspects: the species, the influence of nature (weathered appearance), defects caused from nature or humans, and the growth of moss or other plants on top of the bark as shown in Figure 2.

Figure 2: Moss growth inside crack and on bark surface

3 Approaches

Before deciding on an approach to take the author investigated into papers that had been written about generating tree bark. The main papers found included: Modeling the Mighty Maple by Jules Bloomenthal [2], Interactive Modeling of Tree Bark by Wang et al. [11], Finite Element Model of Fracture Formation on Growing Surfaces by Federl and Prusinkiewicz [4], and Synthesizing Bark by Lefebvre and Neyret [4]. All of these papers have different approaches to the modeling of bark. In [6] the authors proposed a method that they claim to be physical, empirical and textural. They use textures to dress the inside and outside of fractures with detail. They then split the surface into circular strips and simulate the fracture creation and enlargement by the tension along a strip. Furthermore, they simulate in the orthogonal direction the propagation of fractures. Although the pattern generation of the cracks looks natural, the applied textures have a flat appearance. In [11] the authors provide an interesting approach where they use a bark texture file to identify certain patterns. These patterns are group into different features which can be edit through a special piece of software to enhance each feature. This approach results in the most realistic looking tree bark from all the referenced papers. However, the tweaking of the different features and the need for the specialised software would be something to consider for a project with a larger timescale. In Modeling the Mighty Maple, Jules Bloomthal [2] proposed to use a texture file as a bump

map. He generates the bump map through digitising a plaster face by x-raying the cast and using the digitised x-ray as a depth map. This approach in addition to the aforementioned mentioned approach produces very natural looking bark. In the last paper [4], Pavol Federl et al. proposed to synthesize patterns of fractures through finite element methods. This approach is the heaviest in terms of computation and as claimed in the paper runs in the order of a few hours per frame.

The author made the decision very early on to concentrate on purely procedural textures and therefore investigated the possibilities in this area. As the fractures that appear on tree bark represent a non regular cellular pattern the author began to investigate voronoi patterns [5, 3, 1]. The voronoi pattern samples the space into equal cells, the center point of each cell will than be randomly moved around and a line is drawn at the half way mark between neighbouring cells, Fig. 3.

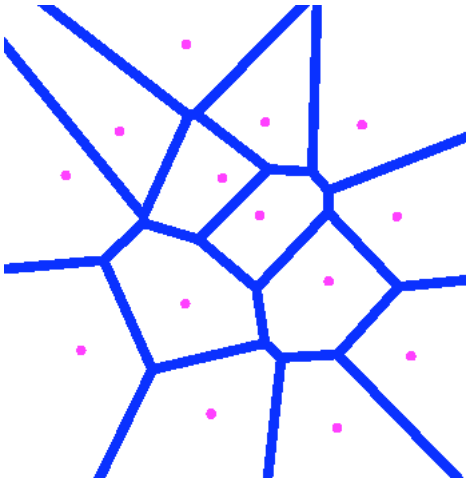
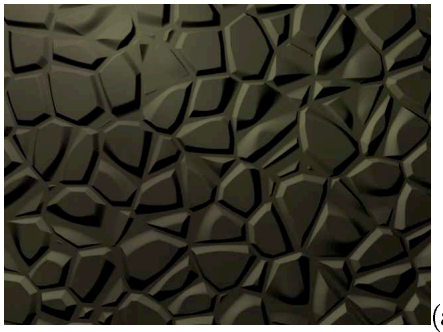


Figure 3: Voronoi Pattern

In order to give the voronoi diagram a more organic look fractal Brownian noise [1, 3] and turbulence[1, 3] were added. Fractal Brownian noise was added to the cracks, Fig. 4 (b), and turbulence to the overall surface, Fig. 5 (b). As real bark has larger and smaller cracks that interact with each other the idea of using a fractal approach to the voronoi pattern was in cooperated [7]. The basic idea is to apply the same pattern with a higher frequency and a lower amplitude as seen in Fig. 5 (a).


(a)

(b)

Figure 4: Base Voronoi (a) and fBm applied to the cracks (b) of the Bark2.sl shader

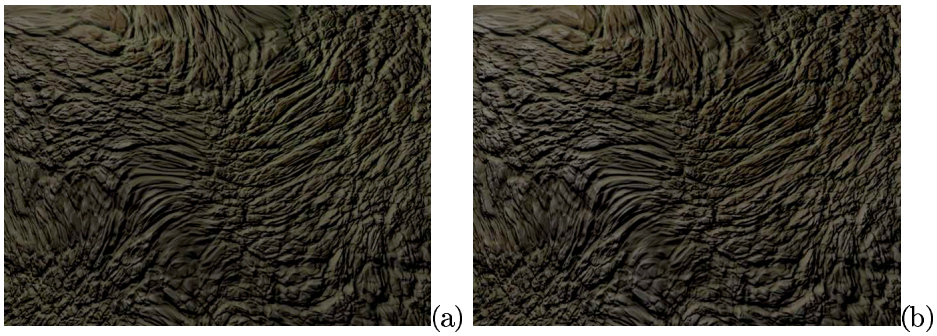


Figure 5: Fractal Voronoi (a) Turbulence (b) of the Bark2.sl shader

As for color the author made the decision to give the shader artist the ability to provide three colors: baseColor, middleColor, and topColor. Furthermore, to include the aspect of plants growing on the surface the artist can provide a color for moss that grows inside the cracks and on top of the surface, Fig. 6. The height of these colors can be controlled by passing in respective values. The decision of using three colors is based on the reference pictures taken, the reference pictures can be viewed in the folder named reference handed in with this report. As it can be seen on these pictures, bark color can be categorised into three different areas. The bottom of the crack, the crack region and the top surface. In order to give the color of the bark a more organic look each color is taken and two more colors are created one slightly darker and one slightly lighter than each of the three colors.

Figure 6: Moss growth based on height

This results in nine colors that are are interpolated through the RenderMan spline function, based on the height value of the point to be shaded[1, 9]. The height value gets passed in from the

displacement shader and has to be normalised in order to interpolate the full spectrum of the nine colors in the surface shader. The actual reflection of light of the surface is based on the material Clay [1] which provides an almost non reflective surface as tree bark.

During the development of the shader Ian Stephenson, Lecturer at the NCCA mentioned that it would be interesting to apply some change to the space the shader was declared in, to produce more interesting affects. The most interesting affects where produced through the RenderMan function call skew [1, 9]. The skew function takes as parameter an angle and two vectors and "skews" points between these two vectors depending on the angle. Figure 7 as well as the pictures in the test_render/beauty folder handed in with this report show the effect.

Figure 7: Skew applied to Figure 5(b)

4 Different Approaches

During the development of the bark shader many different approaches where considered, the most promising ones are outlined in this section.

4.1 Hypertextures

The new "hype" in the shader area evolves around Hypertextures [8, 1, 3]. The main problem with Hypertextures is that they are incredibly hard to control and are very complex and expensive in terms of computation. However, the use of a Hypertexture could be an advantage when controlling the flow of the texture between branches. Through developing a clever look-up function that stretches the value in a specific direction depending on the change on the surface. This could be taken further, as in Houdini and Mantra for example the artist can write 3D texture files. These files are then used to read 3D point information which makes the look-up very fast. However, the writing of 3D texture files is not possible in RenderMan and a custom RSL function would have to be written to access data from a 3D data file created from Mantra. Given enough time and experience this approach would have been considered over the proposed approach in section 3 as it is it represents a novel idea for generating tree bark.

4.2 Du Dv continuity

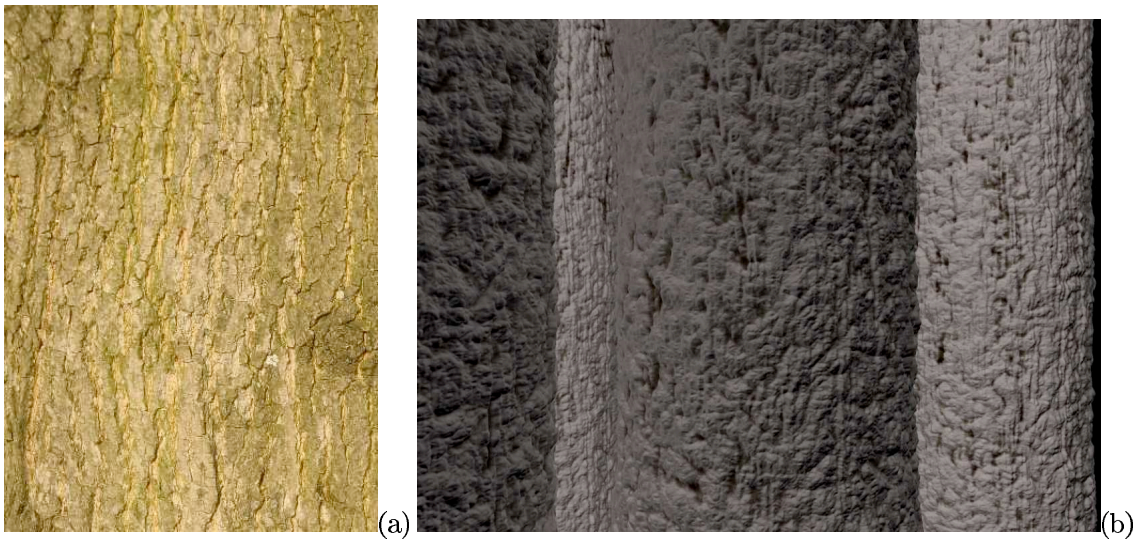
The next approach was to consider the derivative Du and Dv which represent the amount of change in surface direction in relation to u and v. If either of these two values change a considerable amount the surface changes and a different mapping/orientation of the procedural pattern should be applied. In the case that the trunk and the branch were modeled separately this approach could be applied as follows. The trunk is modeled in its own coordinate system. This coordinate system is then passed to the shader of the branch which maps its own points to the coordinate system of the trunk. It has to be taken into consideration that this mapping only applies to the region where the branch is growing out of the trunk in order to have a continuous flow through the texture. The problem with this approach is that when mapping from the branch to the trunk the values from the branch will be mapped to the trunk values at the root. This could be resolved by either using a different coordinate system such as "NDC", "screen", "raster", or applying an offset.

4.3 Color generation

As the voronoi pattern generates unique cells around the area the idea came to use the cell id as a value that could be passed to the hue value of the color [7]. However, this approach was not considered as the generated surface color was to cellular based and therefore the decision was made to base the color on the height value of the displacement.

4.4 Texture files

The author made the decision very early on that he wants to generated the bark pattern purely through procedural textures. It would be wrong on the other hand not to look into using texture files. Fig. 8(a) shows the texture file that was used to create Fig. 8 (b). This approach has many benefits. In comparison to the procedural approach the texture file renders by a magnitude of 4, faster than the procedural texture. In addition the texture file is less pruned to aliasing. The main problem with the texture is that it has to be tile-able and in order to seamlessly flow around the tree. As the benefit of texture files became so obvious the author included a displacement shader (Bark_Texture.sl) that takes as one of its inputs a texture file. The texture file displacement shader was written to work with the same surface shader (Bark_Colour.sl) as the procedural displacement shader (Bark2.sl) and therefore has an output value of the displacement height.



(a) (b)

Figure 8: (a) Texture file (b) Texture with turbulence added

5 Problems

This project was the first investigation of the author into the field of shader writing. Hence, the possibility of discovering problems was planned for. This section outlines the various problem encountered during the writing of the bark shader. Even though some may sound simple they were due to the inexperience of the author in shader writing. The first main issue was that the objects appeared to be moving through the texture. This was simply resolved by transforming the point to be shaded into the relevant shader space. The same problem but with a slightly different outcome was encountered at a later stage. The problem was that the object and shader were moving at the same speed but it seamed as if the object was moving faster than the texture resulting in a stretch of the texture at certain rotations (see the mov folder midrange.mov). There were two solutions to this problem, the first was in relation to the voronoi pattern. The author wanted to have control about the appearance of the voronoi pattern meaning being able to stretch the pattern in any direction therefore he passed a vector that was applied to the point being shaded. The problem of the texture "half" moving through the object was because as with the first problem the vector was not transformed into the same space as the point to be shaded. The second solution to the problem was to move the shader declaration within the same AttributeBegin AttributeEnd block as the geometry. Even though this solution sounds straight forward it was one of these bugs that one spends hours over especially at 3'o clock in the morning.

As aforementioned one of the main goals while creating this shader was to give the artist as many possibilities to change the appearance of the bark as possible. One of these aspects was to change the voronoi pattern. Through the limited experience with voronoi patterns and shader writing it took a fair amount of time to figure out that the pattern could be scaled by passing in a vector to affect the appearance. The other problems when using voronoi patterns was and still is the continuity of the cracks along branches as seen in Fig. 9 (a).

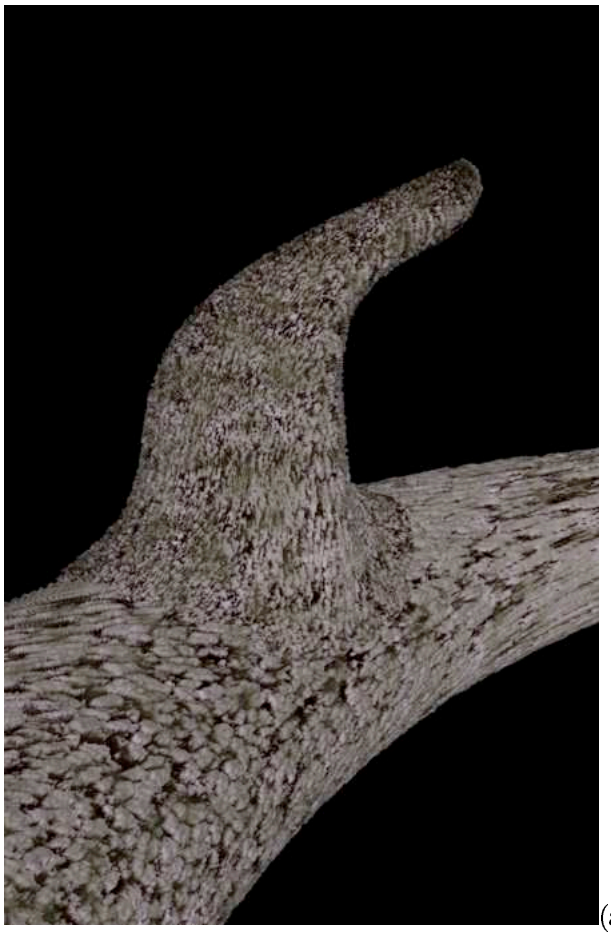
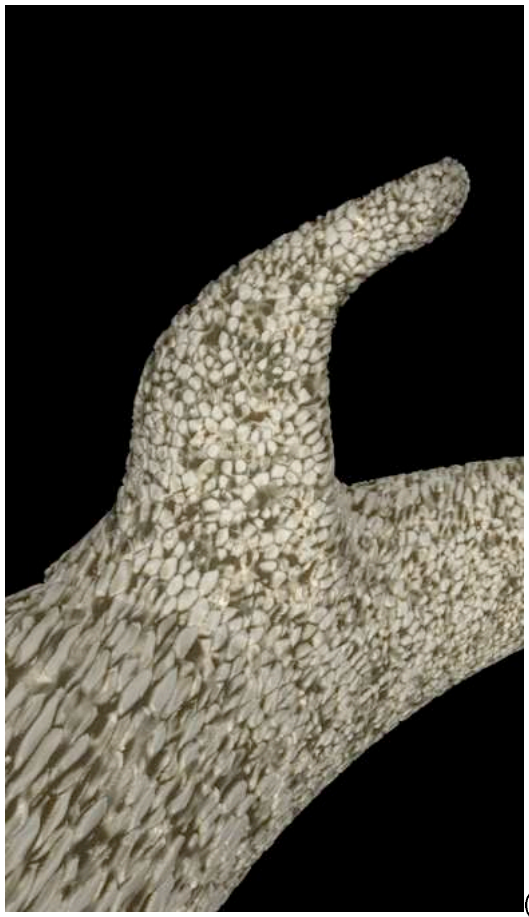

(a)

(b)

Figure 9: Branch continuity problem (a) object space (b) shader space

The branch continuity problem can be resolved by applying the shader in shader space which will guarantee to flow between the two pieces of geometry Fig (b). Through discussions with some of Lecturers at the NCCA the decision was made that this aspect is not of major importance as in a production environment the continuity between branches would be painted by an artist. However, through providing the possibility of specifying the space in which to shade objects leaves the final decision with the artist.

As stated in section 3 the color of the surface is dependent on the height generated by the displacement shader. The problem here was that the color in the surface shader is created through a spline function that interpolates between the given colors depending on a value from 0 - 1. The issue that arose was that the height value passed from the displacement shader could be in any range but not normalized between 0 and 1. Therefore the generated height value needed to be normalized. The decision was made to normalize the height value before passing it to the surface shader. In order to normalize the height value to a range of 0 - 1 the min and max value of all the height values needed to be calculated. As it is impossible to remember values between shader calls a new approach, not involving shading language had to be considered. The first idea was to generate sample values, record these and write a function that depending on the "crack_detail" value returns a normalized height value. This works fine for a single piece of geometry but fails when trying to apply the recorded sample values to a different geometry. The shader writing software Cutter provides the useful feature to write one printf line inside a shader and when executing a rib file the software collects these print statements and on completion returns the min, max and average of that value. Hence, the author has written a perl script that behaves similarly. It adds

a printf statement into the shader, collects all values and returns the min, max and average to the user, see section 6 for more details. After calculating the values the artist can pass the min and max values to the shader and the shader will automatically normalize the height value.

The main problem encountered during this Major Animation Project is the artifact of aliasing. Aliasing is one of the major problems in shader writing and especially for displacement shaders. For an extensive explanation of aliasing see the relevant chapters in [1, 3, 9]. The problem of aliasing in this bark shader is caused through various aspects. As the overall look of the bark should be as organic as possible many different layers of noise were added in the form of vfBm[1] , turbulence [1] and the fractal voronoi pattern. Even though the vfBm and turbulence functions are antialiased the high detail from the fractal voronoi approach causes aliasing. Swapping the smoothstep function when calculating the height of the displacement with the filteredsmoothstep function from the supporting materials of [1] did not resolve the issue. If anything it produced unexpected artifacts as shown in Fig. 10

Figure 10: filteredsmoothstep

To get a better understanding of the artifacts produced through aliasing see the movies provided in the mov folder that was handed in with this report. The main possibilities to reduce aliasing is to either reduce the displacement of the bark or to change the following values in the rib file: PixelSamples, PixelFilter and ShadingRate. The PixelSamples call controls the level of antialiasing by specifying the number of samples per pixel in the x and y directions [1]. The typical values range from 2 by 2 when simple antialiasing is needed, to reduce aliasing for this bark shader the PixelSamples need to be at least 16 by 16 or greater. The ShadingRate controls the frequency of shading a surface. If the ShadingRate is set to 1, a surface is shaded about once per pixel. Larger values cause cruder but faster shading [10]. For this bark shader the ShadingRate need to be as low as 0.05 to 0.001 in order to see a reduction in aliasing. It needs to be mentioned that all of these changes increase the rendering times, therefore the author has provided the artists with the texture file shader that aliases far less than the displacement shader and decreases the render times. The PixelFilter function specifies the type and width of the pixel reconstruction filter that is to be used [1, 10]. The best results have been achieved with the Catmull-Rom filter. The Gaussian filter produces very blurry images where as the sinc filter shows more artifacts.

In general a different problem continued throughout the whole project. The aspect of working towards something specific. Through not having a specific model to which to apply the shader to or the recreation of a specific bark pattern sometimes resulted in working in a loop. Although when looking back at the project not having a specific task proved helpful as through applying the shader to different pieces of geometry the author became more familiar with the shader and is now able to apply the shader and create realistic looking patterns in a relative short time.

6 Tools

This section outlines the tools that were used during the development of this shader.

- For rendering the following renderers where used: Pixars RenderMan and 3Delight. Where RenderMan (http://renderman.pixar.com) is Pixars production renderer and 3Delight (http://www.3delight.com is a "RenderMan" compliant renderer that is free for a single license.
- In order to create the various tree models Alias Maya Unlimited 6.0.1 was used.
- In order to generate Rib files of the models from Maya, Liquid (http://www.plastickitten.net/liquidwiki/index.ph and MTOR/Slim were used. Liquid has the advantage that it automatically creates interfaces to your shaders for the artist to interact with as shown Fig 11

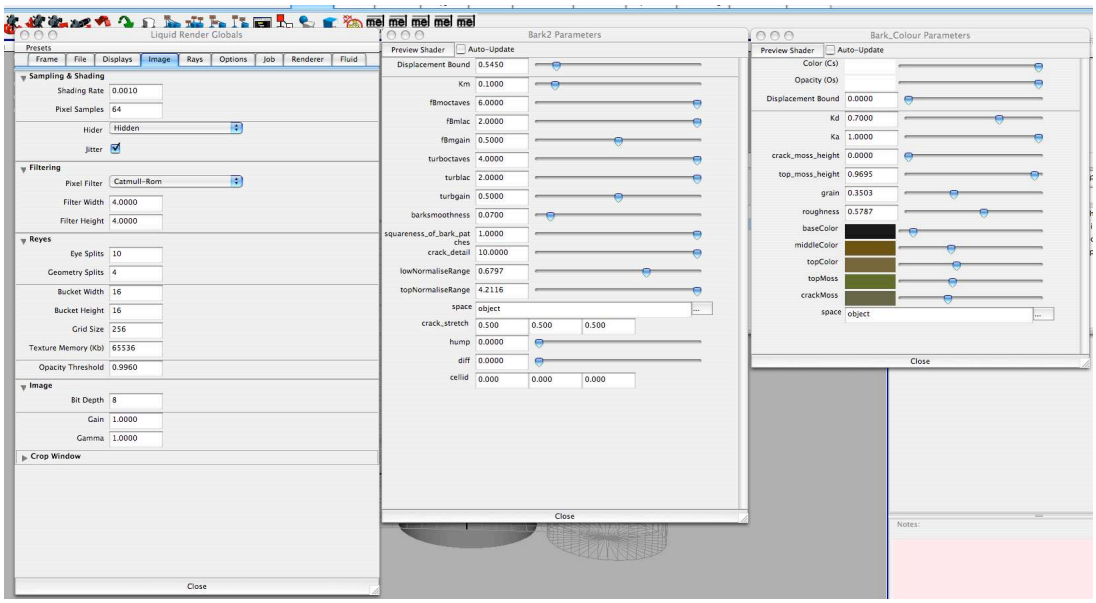


Figure 11: Liquid Interfaces

- In order to increase the productivity various Bash and Perl scripts were written. All of these can be accessed in the scripts folder handed in with this report. The most used was the normalrange.pl script that evaluates the range to normalise the height value of the displacement shader. In order to execute it the user has to add the following line "//printf" at the position of the shader code from which to print the value of interest. To execute it enter normalrange.pl <shader> <rib> <shaderdir without "/" at the end> <value to check> <Compiled shader name> in the commandline. For example normalrange.pl ../shader/Bark2.sl ../rib/trunk_cameraShape1.1.rib ../shader hump Bark2.sdl. In order to use MTOR with this shader a Slim Template would have been needed, however the turnseq.pl script looks for the standard MTOR Surface shader and replaces it with the Bark2 and Bark_Color shader. All the bash scripts were used to animate the texture which was quite important when identifying the influence of each of the different values or to render out simple turntables.
- The last tool and most heavily used was Cutter from http://fundza.com This tool is written for shader writers and provides many valuable features.

All of the above tools provided an excellent pipeline that was very helpful for the creation, development and debugging of the shader.

7 Conclusion

In general this project was a success as the author learned what he expected. He learned the strength and limitations of procedural textures and shader writing as well as learning the various tool that can be used during the shader writing process. The main conclusion was that it is incredible hard to recreate exact natural patterns using procedural textures, but they provide a great way to generate patterns close enough in a very short time. As said beforehand the tweaking in order to make the texture look even closer to the natural counterpart is a very time-consuming task.

The main aim was to create a shader that is highly controllable by the artist, this has been achieved as all the values that generate the appearance of the bark can be passed to the shader. Through providing two different ways of creating the displacement both procedurally (Bark2.sl) or via texture file (Bark_Texture.sl) adds another level of freedom to create the texture. However, it should be manifested that the main work went into creating the procedural texture and the texture file shader should be seen as a prove of concept. As said before the author is confident in stating that this shader provides a great deal of flexibility to create a tree bark texture, it would now be interesting to apply the shader to a specific shot!

References

[1] Anthony A. Apodaca and Larry Gritz. *Advanced RenderMan: Creating CGI for Motion Picture*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

[2] Jules Bloomenthal. Modeling the mighty maple. *SIGGRAPH Comput. Graph.*, 19(3):305–311, 1985.

[3] David S. Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin, and Steven Worley. *Texturing and Modeling: A Procedural Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.

[4] Pavol Federl and Przemyslaw Prusinkiewicz. Finite element model of fracture formation on growing surfaces. In *International Conference on Computational Science*, pages 138–145, 2004.

[5] Chris Gold. The voronoi web site.

[6] Sylvain Lefebvre and Fabrice Neyret. Synthesizing bark. In *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering*, pages 105–116, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[7] Abtabet Parrot. Abtabet parrot.

[8] K. Perlin and E. M. Hoffert. Hypertexture. In *SIGGRAPH '89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, pages 253–262, New York, NY, USA, 1989. ACM Press.

[9] Pixar Animation Studios. Renderman pro server documentation.

[10] Steve Upstill. *RenderMan Companion: A Programmer's Guide to Realistic Computer Graphics*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[11] Xi Wang, Lifeng Wang, Ligang Liu, Shimin Hu, and Baining Guo. Interactive modeling of tree bark. In *PG '03: Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, page 83, Washington, DC, USA, 2003. IEEE Computer Society.