Innovations Project Report

Investigation into the creation and use of displacement and surface shaders

ANNE OKANE

National Centre for Computer Animation

BOURNEMOUTH UNIVERSITY

2008

Abstract

This innovations project is looking into the creation of displacement and surface renderman shaders. It will examine the environment necessary for creating and testing these shaders using SL (a specialized programming language) alongside the PRman implementation of the Renderman standard. How this code can then be imported and implemented within a 3D modelling and animation package such as Maya or Houdini.
In the report the creation of a displacement and surface shader that can be used together to create a visually interesting surface will be examined from the ground up.

Introduction

Within a 3d environment, the creation of models is referred to as geometric modelling and is concerned with the manipulation and creation of points within a surface. Appearance modelling or texturing however, is concerned with the individual pixel values at points on the surface called micro polygons, which the surface has been divided into by the renderer using the REYES algorithm. A fuller explanation of micro polygons and the REYES algorithm can be found in Rendering for Beginners, Raghavachary(P32,33).
Creating interesting visual representations of a 3d scene is ultimately the job of the CG artist. The renderer is the interpreter for these artistic decisions, taking geometrical information about the models and processing it together with scene information such as lights, cameras, and shader informations such as colour and opacity. The output from the renderer creates the believable or interesting images originally initiated by the artist.

Motivation

I want to create a surface shader and a displacement shader in order to have maximum control over the appearance of an object. I hope to use these shaders in my Major Project to create a cracked surface where the inner part of the cracks reveals a different surface than the outer part. For example, a dry dusty surface that has cracked apart to reveal a wetter surface inside. I have always enjoyed creating surfaces in more traditional art, such as painting and textiles. As I have no experience in making custom computer generated surfaces, exploring a new media from a more traditional knowledge base will be innovative for me as an artist. It will allow me to broaden my knowledge and understanding of 3d computer graphics.

Aim

My aim is to create a surface and displacement shader that can be applied to an

object to create a visually pleasing result. I hope to be able to import the shader I write into a modelling package, which will allow me to create an animation of a sphere rotating to show these shaders combined.

Proposed method

To use a renderman compliant renderer together with a rib file, to test and develop a shader in order to produce a shader that can displace a surface to form cracks, and change the surface appearance within those cracks.

Where to start?

Essential Renderman, Stephenson(2007,P.15) states
You should attempt to render and extend the [shader] examples [in each chapter] as this is the only method by which familiarity can be gained with the rendering process.
As a starting point I decided to follow the examples given in his book to create some basic shaders and gather some understanding of how the shader code and the renderer interact to create the image. The images created in this way can be found in Appendix C.

About this report

The following report reflects my personal understanding of the subject that I have gleaned throughout the course of this project. Writing rib and shader files is for me a complex subject that has been touched on in this report at a very basic level. Fuller explanations for writing rib and shader files can be found in any of the books listed in the bibliography at the end of the report.

The rendering environment and depencancies.

In order for the renderer to work its magic and turn information into an image it relies on two main things, The RIB File and the shaders. Therefore in order to create an image I need to create a RIB file and a Shader File.

So what is a Rib File?

A rib file is text file that contains all the information about the scene/environment, this includes all geometry, lights and the camera information, it is also where your shader needs to be declared so that the renderer can find and use it to render the image.
The RIB file format is explained by Anthony A.Apodaca and Larry Gritz (1999,P.58),
The Renderman Interface has two official language bindings, one for the C programming language and another that is a metafile format (the Renderman Interface Bytestream, or RIB). Details of both of these bindings can be found in

Appendix C of the Renderman Interface Specification.
They also state,
RIB is not a programming language itself. It does not have any programming structures, like variables or loops. It is simply a transcription of a series of calls to the C API binding into a textual format.

Creating a Rib file

A rib file can be written in any text editor and needs to be saved with the file extension .rib.

Listing 1.0
myFile.rib

```
line 1      Display "myFile.Tiff" "file" "rgba"
line2     Projection "perspective" [45]
line 3    WorldBegin
line 4      Translate 0 0 4
line5     Sphere 1 -1 1 360
line6      WorldEnd
```

Explaining the basic rib file

Line 1

Makes a call to the display command, it is here that I can specify what I want the renderer to do with the image created, Note that things that are specified to a command have to be enclosed in "".
Rudy Cortes and Saty Raghavachary(2008,P.19) expand on the Display command.
The *Display* command tells the renderer what to do with the final computed pixel values. You can store them in a file or display them in a window of some sort. The target for the pixels is also referred to as the *display driver.* Each renderer can support different display drivers, but no matter which they support, they need to be able to handle two basic drivers: the file driver and the framebuffer driver. The framebuffer driver will send the pixels to a window drawn on the screen.
In the above example I have specified that I want the renderer to send the information to a file, to name that file *myFile* and store it in the tiff format with the rgba channels. By default the image *myFile.tiff* will be stored in the same directory that the rib file is in.

Line 2

The command Projection gives us access to the viewing portion of the camera.
Rudy Cortes and Saty Raghavachary(2008,P.18) explain the projection command.
The projection can be of two types: *orthographic* and *perspective*. When using

an *orthographic* projection, you don't need to provide a field of view (fov) parameter, which is a float value that represents the angle of view in degrees. This line in the rib file tells the renderer to use a perspective camera, which means that things closer to the camera will be drawn larger while things further away will be drawn smaller, with a field of view of 45 degrees.

Line 3 and line 6.
Ian Stephenson(2008,p.19) describes the worldBegin and WorldEnd commands, "The command WorldBegin informs the renderer that the setup phase is complete, and it should prepare to draw. The related command at the end of the file WorldEnd informs it that the scene is finished, and it can write out the final image."

Line 4.
All objects within a scene are described relative to the camera, and world space is defined with the camera as the origin of that space, therefore any object I create within the scene will be drawn exactly where the camera is. Here I use the transform command to move the object away from the camera allowing us to see it. The command Transform 0 0 4, specifies that the object should be moved by a value of 0 units on the x axis, 0 units on the y axis and 4 units along the z axis.

Line 5.
The command sphere tells the renderer to create a simple geometric shape, the command takes four parameters, the first parameter is the radius, Ian Stephenson(2007) explains the next two parameters concisely, by stating.
"The second and third parameters of the Sphere command allow you to clip the bottom and top of the sphere, creating a ring. Any part of the sphere below the second parameter or above the third is removed. For example, the command Sphere 2 -0.5 1 360 would draw a sphere of radius two with most of the bottom removed, and a little of the top clipped."
The last parameter refers to the sweep of the sphere, when set to 360 this creates a complete sphere, setting it to a value of 45 would create a partially drawn sphere, rather like a piece of cake.

Detailed information on Rib specifications can be found in the following books.
Rendering for beginners by Saty Raghavachary.
Essential Renderman by Ian Stephenson.
Advanced Renderman by Larry Gritz and Tony Apodaca.

What is a Shader?

Saty Raghavachary(2005,P31) describes shaders
"Shaders are external programs (small files on disk) that are used by Renderman in conjunction with the RIB files in order to calculate shading for objects in your scene."

A shader is a collection of code that allows the programmer to apply changes to the geometry information stored in the rib file. The information contained in the rib file and accessed by the shader, describes the geometry surface broken into micro polygons, and is used by the renderer to compute the pixel values of the image.
The original information about the surface comes from either a rib file or an animation and modelling package like Houdini or Maya.
There are different types of shader and each type can access special variables often referred to as Global Variables. These globals are the mechanism that allows information to be passed between the shader, the rib file, and the renderer.
There are two types of global variables, the input and the output variables.
Global variables that store information specified in the rib file about the object, are commonly referred to as the input variables as they contain information about the object that can be accessed as input from the shader.
Global variables that hold modified values that have been calculated from the input variables inside the shader are often referred to as the ouput variables. For example, Cs is the global input variable that stores the colour information of the object defined in the rib file and Ci is the global output variable that holds the new color information calculated in the shader.

The type of information passed between the rib file and the shader depends upon the type of shader that you decide to write, as each shader type is defined with a list of different variables. A full list of variables can be found in Essential Renderman, Stephenson (2007.inside back cover).

Shaders can do much more than merely process and change the information passed to it from the rib file, they are designed to allow a programmer to create unique parameters within them that in turn allow users to influence directly the calculations computed within the shader; these parameters can be used to include accessing texture files whose information can then be utilised within the shader.
The use of parameters within the shader are used to incorporate information from outside of the rendering environment contained in the rib file, creating the ability for this independent information to become part of the rendering process. This is a powerful and flexible feature of the renderman API.

How to write my Shader

Shaders are written in a high level programming language generally referred to as SL, a shading language similar to C.
There are conventions that need to be followed for shader writing, the first thing that needs to be defined is the shader type, this is required to enable the renderer to know which where to use the shader. For example, a displacement shader is

always processed before a surface shader. In the book Advanced Renderman, Anthony A.Apodaca and Larry Gritz (1999,P.161)state
The Renderman Interface specification describes several types of shaders, distinguished by what quantities they compute and at what
point they are invoked in the rendering pipeline
they go on to describe these shader types.
Surface **shaders** describe the appearance of surfaces and how they react to the lights that shine on them.
Displacement shaders describe how surfaces wrinkle or bump.
Light Shaders describe the directions, amounts, and colours of illumination distributed by a light source in the scene.
Volume Shaders describe how light is affected as it passes through a participating medium such as smoke or haze.

Immediately after specifying the type of shader I will be writing, I need to specify a name that I want my shader to be called, this name has to be defined in the rib file to tell the renderer to find and use this specific shader.
A list of parameters follows the shader type and name. These must be enclosed in parenthesis; these parameters are created by the programmer for the end users and allow for flexibility of the shader aesthetic. For example, a shader created with a paramater for brightness will create an attribute that can be accessed in the modelling package called brightness, which allows the user to control the amount of brightness that is being used in the calculation of the image.
After the parenthesis curly braces are used, everything between these braces will be evaluated by the renderer.

Detailed information on the SL shading language can be found in the following books.
Advanced Renderman, (1999.P159/182)
The Renderman shading language guide(2008.chapter 2)

Writing my first Shader

The following code is from Ian Stephensons essential Renderman(2007.P.131).

Listing 1.1

```
Line 1    surface test()
Line 2    {
Line 3    Oi=Os;
Line 4    Ci=Cs *Oi;
Line 5    }
```

Explaining my first Shader, listing 1.1.
Line 1 - Specify the type of shader, followed by the name of the shader itself, which could be a descriptive name for the surface, such as blackRock, or

redStone. The parenthesis is empty as no parameters for the shader are required.

Line 2,5 The curly braces define the body of the shader.

Line 3 Assigning the input opacity to the output opacity.

Line 4 – Assigning to the output colour, the input colour multiplied by the output opacity.

Stephenson (2007, P.136 explains concisely why this needs to be done, Normally colour values range between 0 and 1, so 1, 0, 0 would be bright red. However, if this colour was applied to a surface that was totally transparent, then the surface could not carry any colour all (0, 0, and 0). If the surface was 50% transparent, then half the output colour would come from the surface (0.5, 0, 0) and half would be from whatever was behind the surface. Multiplying the surface colour by the opacity takes this into account. The simple rule is that you should always multiply the output colour by the opacity.

What do I do with the shader file now?

The high level SL programming code has to be changed into a lower language that the renderer can understand, this can be done through a process known as compilation, each renderer has its own compiler, the extensions and commands needed to use it are dependent on which renderer you have. I am using a PRman compliant renderer so I need to add .sl as an extension. Listing 1.1 saved as *test.sl.*
This will create another file in the directory called *test.slo*, which contains the lower level language that the renderer can understand. This .slo file is called by the rib file when it is sent to the renderer.

Writing my first RIB file

I created this basic rib file, and saved it as testMyShader.rib, the following code is taken from Essential Renderman, Stephenson (2007,P.133).
Listing 1.2

```
Line 1      Display "test.tiff" "file" "rgba"
Line 2          Projection "perspective" "fov" [45
Line 3      LightSource "ambientlight" 1
Line 4                  "intensity" [0.2]
Line 5      LightSource "spotlight" 2
Line 6          "from" [-1 1 0 ]
Line 7          "to" [0 0 3]
Line 8          "intensity" [3]
Line 9      Translate 0 0 3
```

Line 10 *WorldBegin*
Line 11 *Color [1 0 0]*
Line 12 *Surface "first"*
Line 13 *Sphere 1 -1 1 360*
Line 14 *WorldEnd*

Understanding the rib file, listing 1.2

Line 3 & 5 - Here an ambient light and a spotlight are created within the scene using the
command LightSource, specifying which type of light should be created, along with some specifications for the intensity of those lights. Note that the spotlight parameters include from and to commands, these tell the renderer where the spotlight is starting from and where it finishes.
Line 11 – This specifies the default colour of the object, the rib file passes this colour to the shader within the variable Cs, if the shader doesn't specify any changes to the colour value; the resulting image will have the object shaded with this color.
Line 12 – Specify the shader type and its name.

How do I get my shader and rib file to create an image.

Now I have three files in my directory, test.sl, test.slo and testMyShader.rib.
The execution of the above rib file using the command *render testMyShader.rib* would tell the renderer to collate information from both the rib file and the previously compiled test.slo file to create an image called test.tiff, and store it in the same directory where the rib file is situated. *See figure 1.*
It should be noted that the rib file relies on the .slo file created once the .sl file has been compiled.

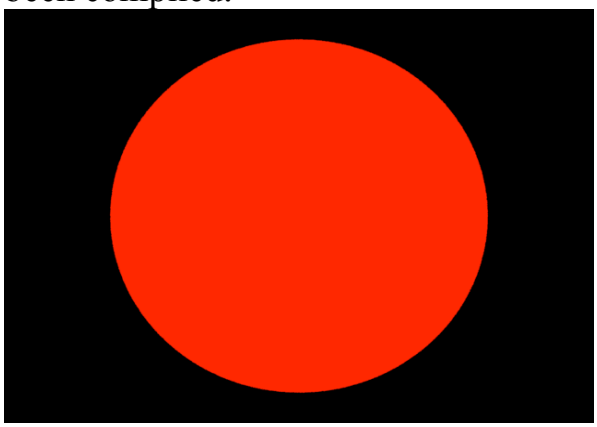


Figure 1_ my first shader

I can now setup a shader development process as outlined by Ian Stephenson(2007,P134).

"Edit myShader.sl
Shader MyShader.sl
Render testMyShader.rib
Viewer testMyShader.tiff "

Creating a displacement shader

Speaking with a lecturer Phil Spicer about input and output variables, lead to a brief examination of the vex level in Houdini (a graphical way of creating renderman shaders), this revealed that a displacement shader has access to quite a large list of input variables, but can only ouput two variables which are P(the point on the surface) and N(the surface normal). See figure 2.

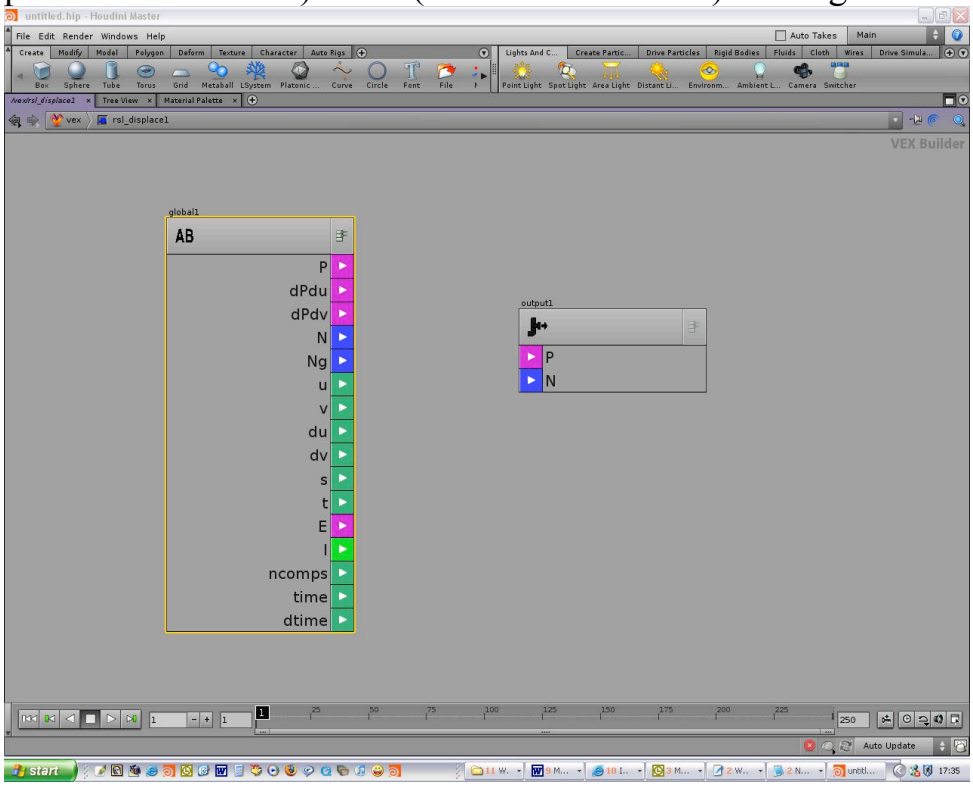


Figure 2 Input and output variables available for displacement shaders

A surface shader has access to a larger list of input variables, but can only ouput two variables, these are Ci(the colour value) and Oi(The opacity value). See figure 3.

One of the globals that can be accessed by a displacement shader is P which holds the co-ordinates of the micro olygon points of the surface. These can be modified within the shader and assigned to the ouput variable P, subsequesntly the renderer uses these modified values in P, as the co-ordinates of the surface and these are then used in the final image. It should be noted that when writing other shader types the point P can be an input variable but not an ouput, this means that displacement cannot be performed in other shader types although in some

renderers a surface shader also has the capability to modify P.

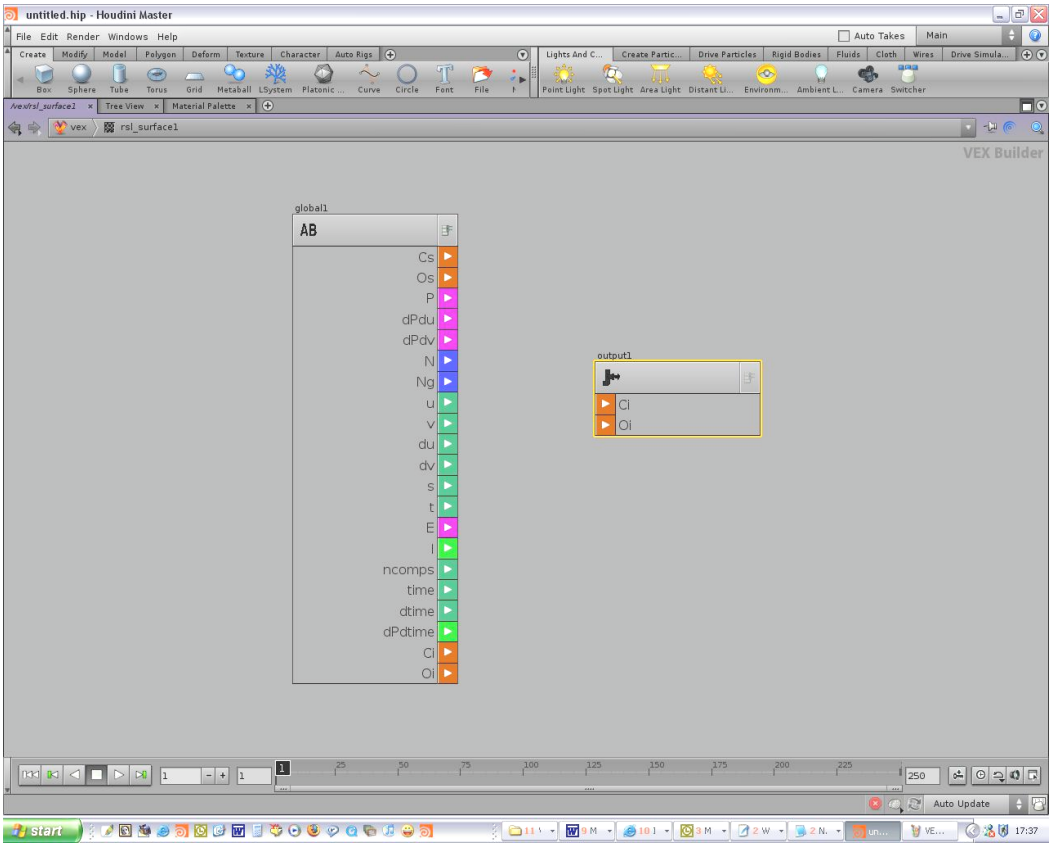


Figure 3 Input and output variables available to a surface shader.

A basic displacement shader is illustrated below, the code is from Essential Renderman, Stephenson(2007,P.192).

Listing 1.3

```
Line 1      Displace simple)
Line 2      float km=0.1;)
Line 3      {
Line 4      Normal NN=Normalize(N);
Line 5      float mag=o;
Line 6      mag=sin(s*10*2*PI)*sin(t*10*2*pi);
Line 7      P=P+mag*km*NN;
Line 8      N=calculatenormal(P);
Line 9      }
```

Understanding the displacement shader, listing 1.3

Line 1 Here the shader type is followed by the name and the opening parenthesis to enclose the parameter list.

Line 2 Creating a parameter that can be modified by a user to affect the displacement.

Line 3&9 Curly braces define the body of the shader.

Line 4 The surface normal is modified by the function normalize, which returns the normal with unit length, which is assigned it to the variable NN.

Line 5 Creation of a variable which will control the magnitude of the displacement.

Line 6 Assigning a value to the variable mag.

Line 7 Change the value of P (moving the actual point on the surface).

Line 8 Recalculate the surface normal of the new point P.

When compiled and used in a rib file this shader creates the displacement we can see
In figure 4 on several pieces of geometry.

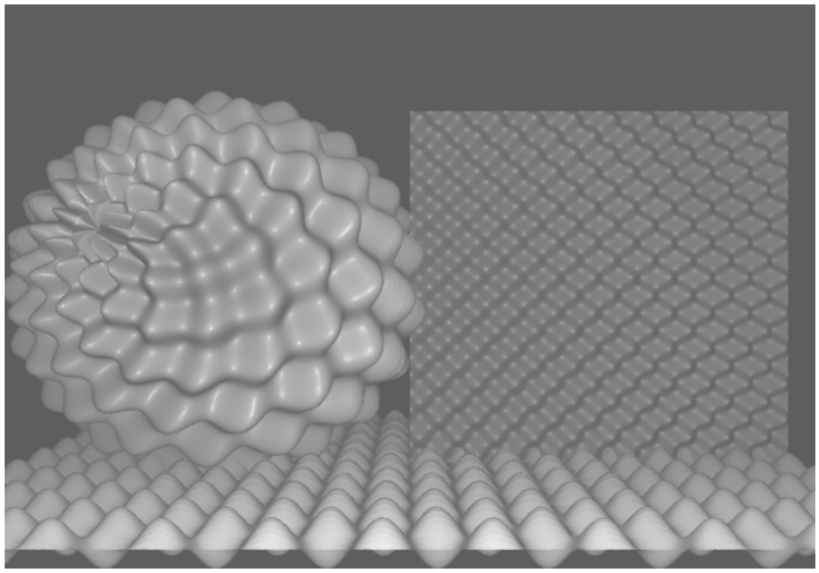


Figure 4 Displacement on different surfaces.

Figure 4 was created using a found rib file[see ref:1]

Using Texture files.

After experimenting a little with displacement shaders I decided that the next step would be to use a texture to drive the displacement. This wasn't as straightforward as I thought it would be. The following render, (see figure 5), was created by assigning a texture to the variable mag, and was my first attempt at using a texture map to displace the surface. It was created by using the line, mag=float texture(cracks.tx)"

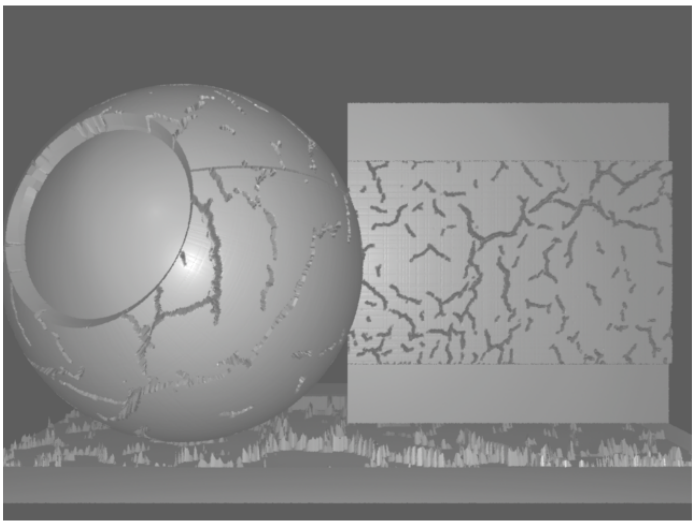


Figure 5 Displacment using a texture

It can be seen that the texture isn't being mapped to the surface correctly. I discussed this with one of my classmates who informed me that a picture ahs to be formatted into a file format that the renderer supports. This can be done using the command *txmake* from the command line, making sure that you are in the directory where the image that needs to be converted is stored. *Txmake* takes two arguments, the name of the picture including its extension, followed by the name you want the new image to have followed by the extension .tx.

For ease of use any picture for converted also had to be square in a power of 2, for example 512 by 512, 1024 by 1024. This is because the texture needs to be accessed using the s and t co-ordinates of the surface which are always between 0 and 1, therefore the picture has to have co-ordinates between 0 and 1. The function *texture()* can be used in the shader to grab the values for each point on the texture that correlate to the co-ordinates of the surface, it does this by using the current shading sample's s and t co-ordinates to find the value of the texture at the corresponding point and stores that as a colour or as a float.
Using *color texture();* will look up a colour value, whilst using *float texture();* will return a greyscale value. The channel this greyscale value is taken from can be specified after the file name in square brackets. This is often used to access the alpha channel in a picture by specifying float texture ("myfile.tx"[3]);

At this point I have created a displacement shader that takes a picture and uses it to
displace the surface. (See figures 6). Original tiff pictures used in the shaders after they were converted can be found in Appendix A.

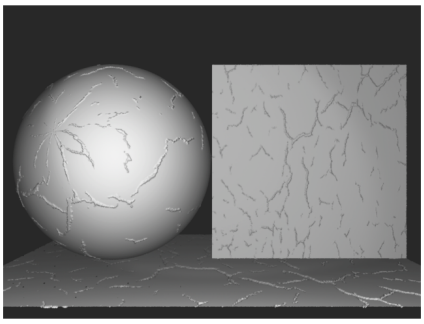


Figure 6 displacement using a texture

I now have a shader that will create the cracks in my surface. I decided to start exploring writing a surface shader to use with my displacement as the next development step.

Creating a Surface Shader.

Reading Essential Renderman I discovered that you can achieve creating two different surface types within one surface by using a texture not as a colour, but as a control. The following code is from Stephenson(2007, P184, Listing 24.6).

*Ci=Oi *(Ct*(Ka*ambient()+surfaceType*diffuse(Nf))+(1-surfaceType)*specular(Nf,V,roughness));*

In this code the value of the diffuse lighting is being affected by adding the value of surfaceType, (which is a greyscale value) to it, whilst the specular lighting is having the same value removed from it, this creates areas in the surface where light calculations are influenced by the texture values stored in surfaceType.
I also discovered from reading Advanced Renderman (Apodaca and Gritz,P.185, 8.1.2)
that you can use texture mapping with alpha compositing. Below is the code I have used, based on the above source.

```
LINE 1      CT=CS;
Line 2          C=color texture("crack2Alpha.tx");
Line 3          a=float texture("crack2Alpha.tx[3]");
Line 4          Ct=C + (1-a) *Ct;
```

Line 1 – assigns the value stored in Cs, which is the input color from the rib file to the
color variable Ct;
Line 2 – takes a colour value from the texture and stores it in the colour variable C.
Line 3 – takes a greyscale value from the same picture but from the alpha channel of the
picture which is either 1 or 0.

Line 4 – this calculates the value stored in C and adds to it 1-the greyscale value from the
alpha channel, then multiplies this by the values held in Ct, before assigning the resulting value back to Ct.
It should be noted that a displacement shader is applied first followed by the surface shader, in the rib file we can just state.
Displacement "dis1"
Surface "matte"

Using these codes together in my shader with some modifications I created the following render. See fig 7.

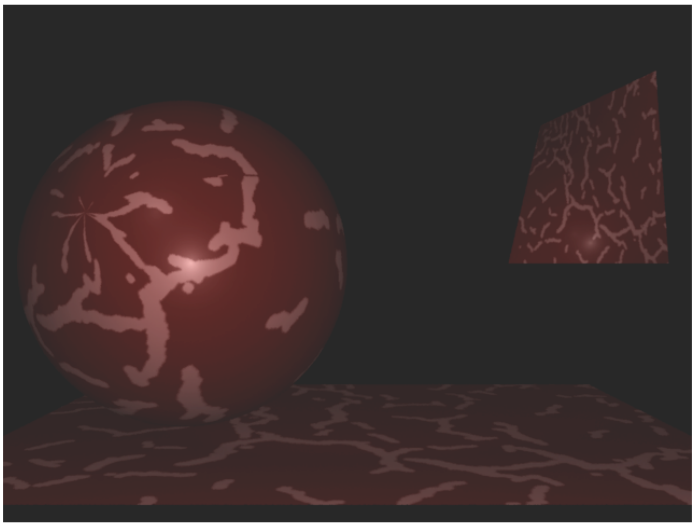
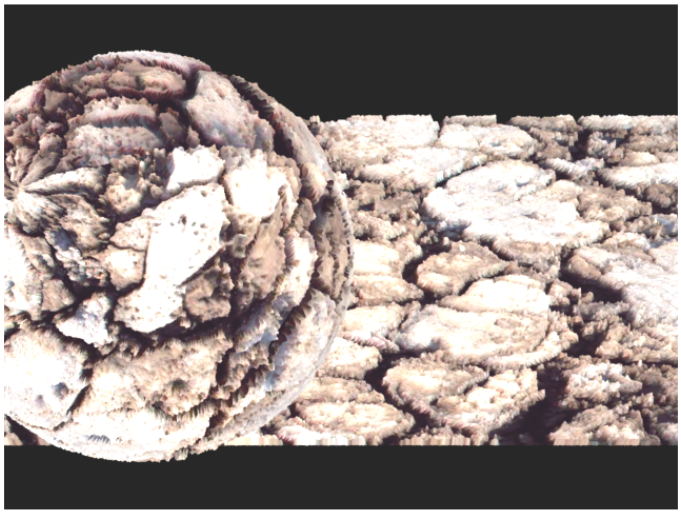


Figure 7 alpha compositing and texture control of the surface type.

Shader for animation

Through much trial and error I eventually created a shader that displaces the surface and allows me to control the change in surface type through the alpha channel. This can be seen in the video product of the shader in use and also in



the render below. See figure 8.

Figure 8 surface change with displacement

This version of the shader has been rendered out as a video and it can be seen that where the texture is changed to shiny, it is a flat color with no input from the texture that is on the rest of the surface.
I spent about 2 days trying to get my shader into maya, as there were problems with loading renderman, I even spent over an hour with the IT technician who tried to resolve the issue. Time was running short so I decided to try and import it into Houdini instead. As I am not familiar with Houdini, I asked Phil Spicer, how to get my shader into it. Fantastically he had written a script that does all the hard work, I only had to run the command *rhGo9.1 myShader.sl*, from the directory where my .sl files were stored for both my shaders. Then I only had to open Houdini and go to file, refresh libraries, and my shaders were there and ready to use. Absolutely wonderful.

A Last Adjustment

I wanted to create a texture change that was influenced by the original texture as well, so that it keeps more consistancy with the uncracked parts of the surface. I managed to achieve this in the next version of the shader with help from Ian Stephenson to clarify my code. This effect can be seen in figure 9.
Unfortunately I didn't have time to render out an animation of this last version. There are problems with the shader, I have artefacts showing through where the displacement is, this is something that I will have to address outside of the project as I would like to use the develop and use the shader further.

Figure 9 the final surface shader used with the displacement shader

I did some experimentation with using noise on the surface as I felt that this would give the shader more flexibility, but ran out of time to fully explore this idea. Renders from these experiments and others can be found in Appendix A.

Original Pictures used to create the .tx files used can be found in Appendix B. Renders created from Essential Renderman Stephenson(2007,[part 3])can be found in Appendix C.

Analysis

I feel that I have achieved a visually pleasing image using my shaders, which I intend to develop further.

With hindsight I think that I could have achieved the same result without using the alpha channel at all. It should be a relatively simple task to just sample the texture for a greyscale image and change the surface where the pixel value is darkest.

As my shader stands at the moment, it needs a picture that has an alpha channel in it to drive the surface change. The same picture also drives the displacement and this could lead to unexpected results. In the displacement shader, the displacement pushes further inwards as the values get darker, and if the alpha is masking the wrong part of the image the wrong part of the surface becomes shiny.

I wonder if I could have generated the cracks procedurally rather than using a texture.

Conclusion

Looking back at my aim I feel that I have achieved what I set out to do, but more positively, I feel that I have achieved a good foundation for shading writing, which I hope to take further into my major project. I have discovered that although based in programming the rendering process is accessible to artists and doesn't exclude non-programmers.
I have found during this project that I enjoy shader writing although intially the interdependencies between the rib file, shader, and renderer were a little difficult to totally comprehend. The flexibility and amount of control that can be achieved with shader writing is well worth the time and effort spent, and I hope to develop my shader writing further to use in my major project.

Acknowledgements

I would like to thank Phil Spicer, for his clarification of topics, helpful explanations, and the lovely script for getting shaders into Houdini.
Ian Stephenson, for his help and explanations about code issues, along with Luke Harris for lending me his book and his helpful suggestions.
Thanks also to Jon Macey for creating his website which was a big help for this project, and my tutor Hammadi Nait-Charif for his encouragement.

BIBLIOGRAPHY

BOOKS

Apodaca, A. A., Gritz, L. 2000. Advanced RenderMan. USA : Morgan Kaufmann Publishers.

Cortes,R.,Raghavachary,S.2008.The Renderman shading language guide.USA:Thomson Course Technology PTR.

Raghavachary, S.2005.Rendering for beginners.Oxford:Focal Press.

Stephenson, I. 2003. Essential RenderMan.second edition. London: Springer-

Verlag.

Websites

Macey Jon .[online]. displacement shaders.pdf. available at http://nccastaff.bournemouth.ac.uk/jmacey/Renderman/index.html

[accessed 25th march 2007]

References

Ref:1 http://nccastaff.bournemouth.ac.uk/jmacey/Renderman/index.html8 Displacement Shaders pdf examples Displcement.tgz_FILES/TextureDisp/ShaderTest.rib.

PICTURES

PICTURE 1 http://www.cgtextures.com/[07.02.08] Ref *REPTILES0009_3_L*

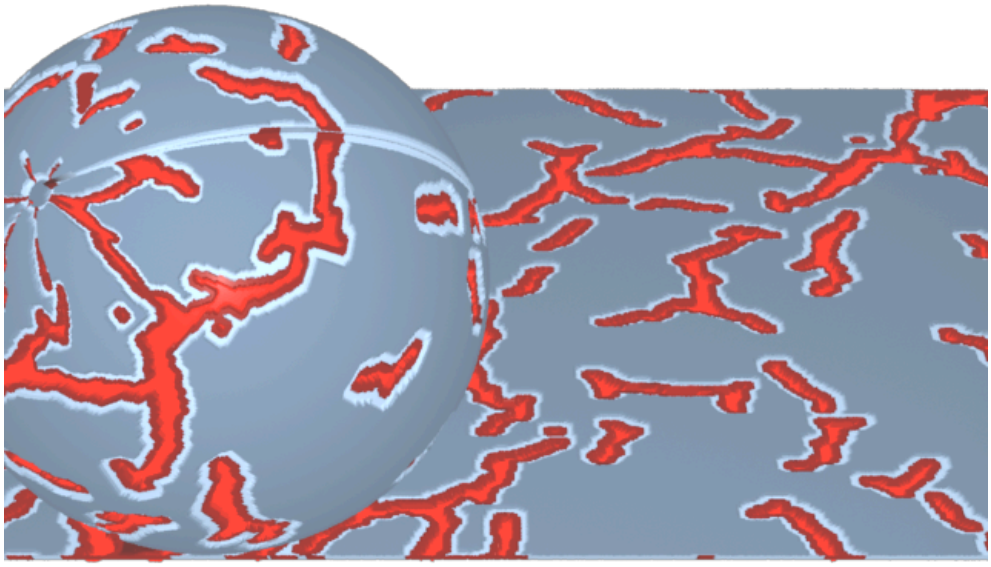
PICTURE 2 http://www.cgtextures.com/[07.02.08]

PICTURE 3 http://www.cgtextures.com/[07.02.08]

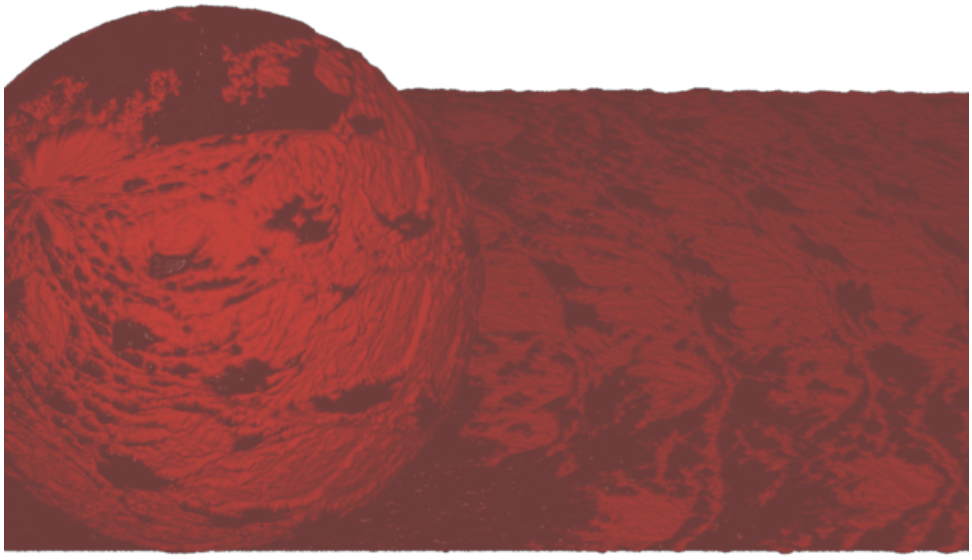
Appendix A

Some renders from the development of my shader

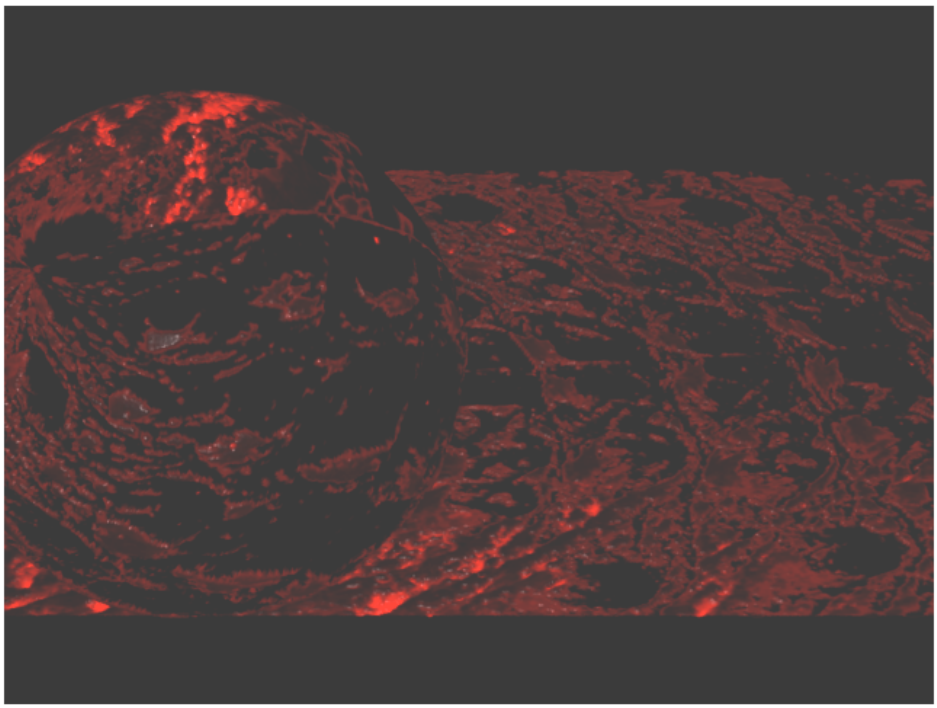
Render 1 using a simple drawn texture



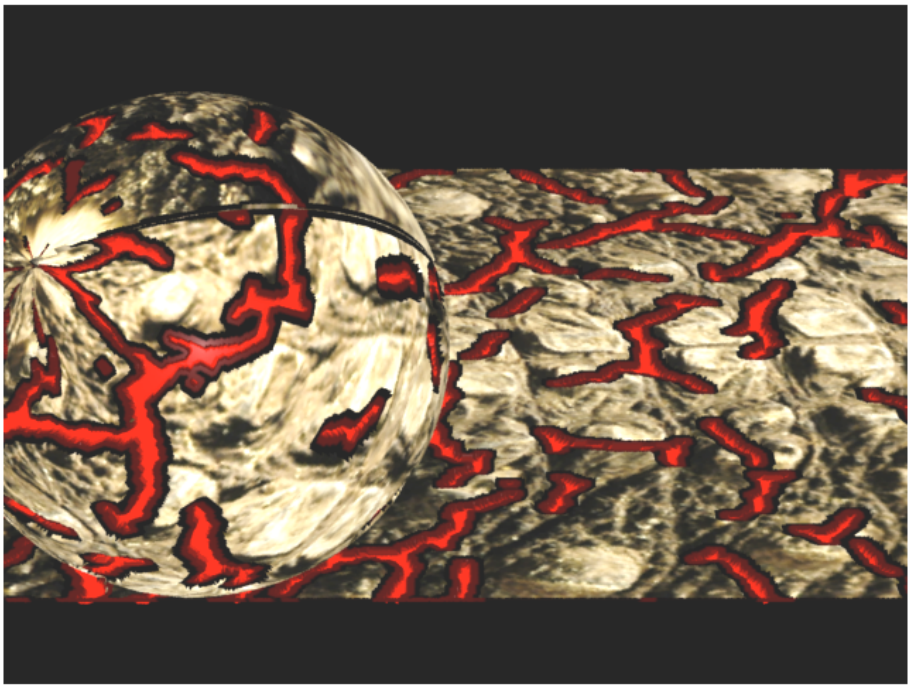
Render 2 using the colour form the rib file with the texture



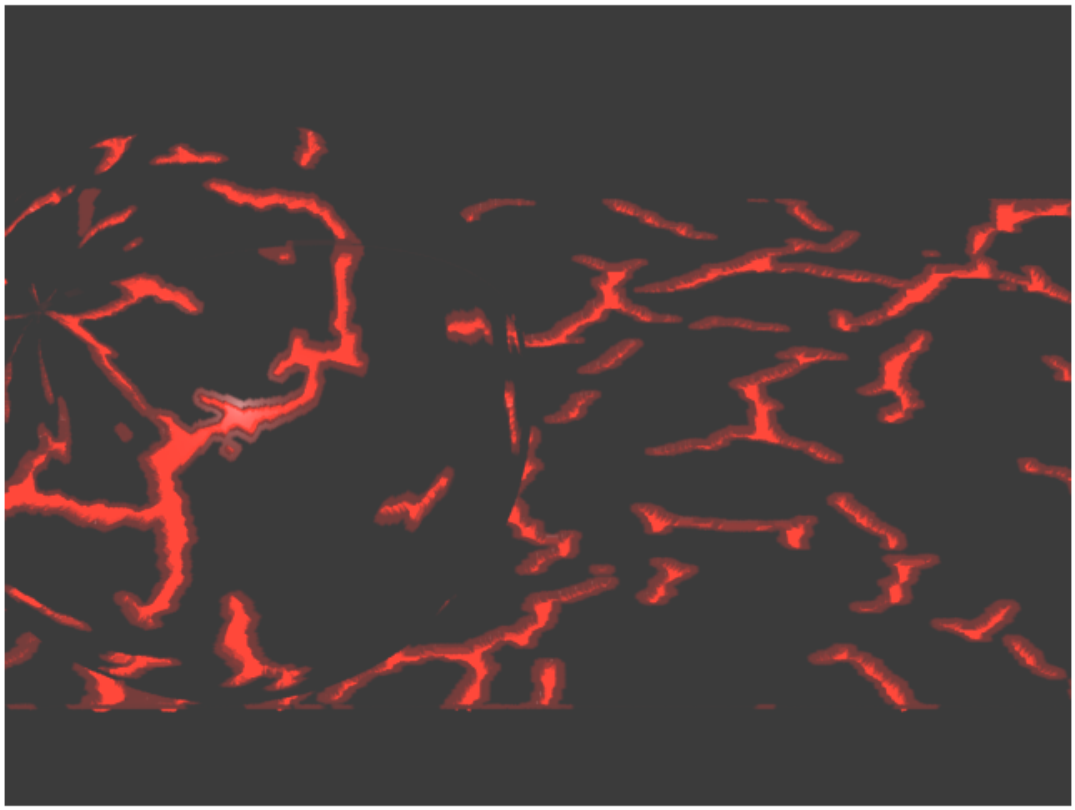
Render 3 lava like



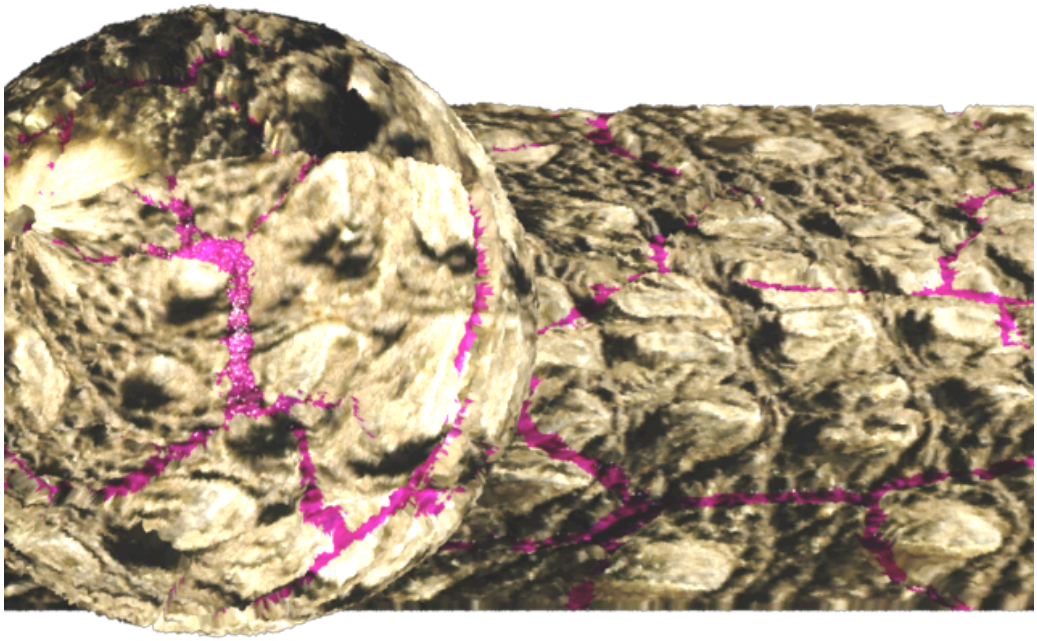
Render 4 Adding texture where the alpha isn't



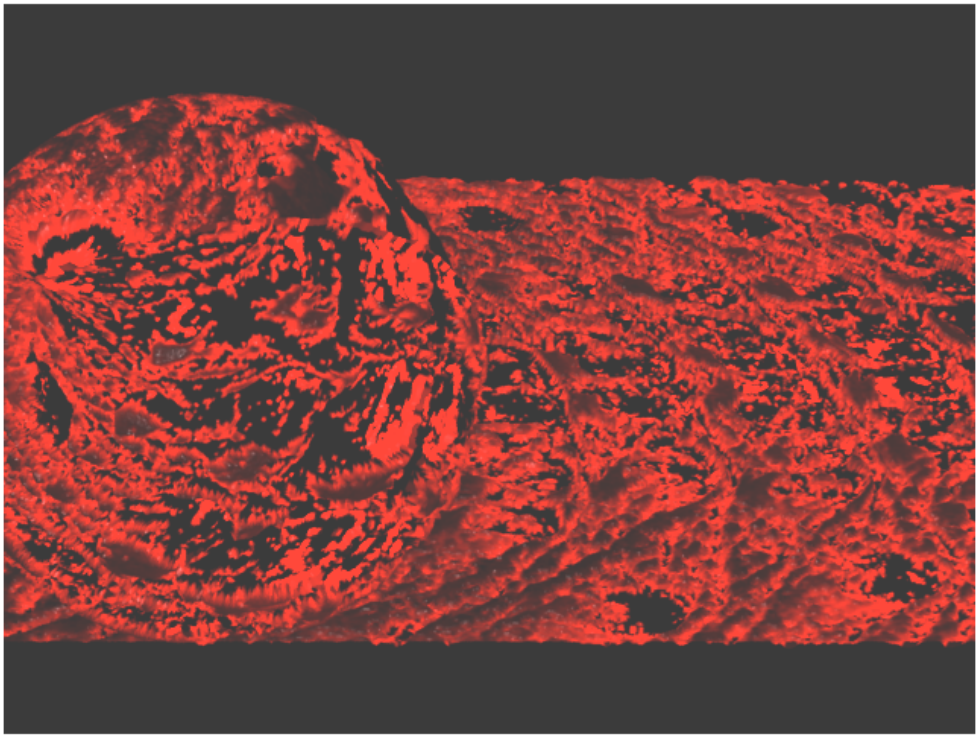
Render 4 Simplified texture to clarify which part of the texture is being used blended with the background colour from the rib file.



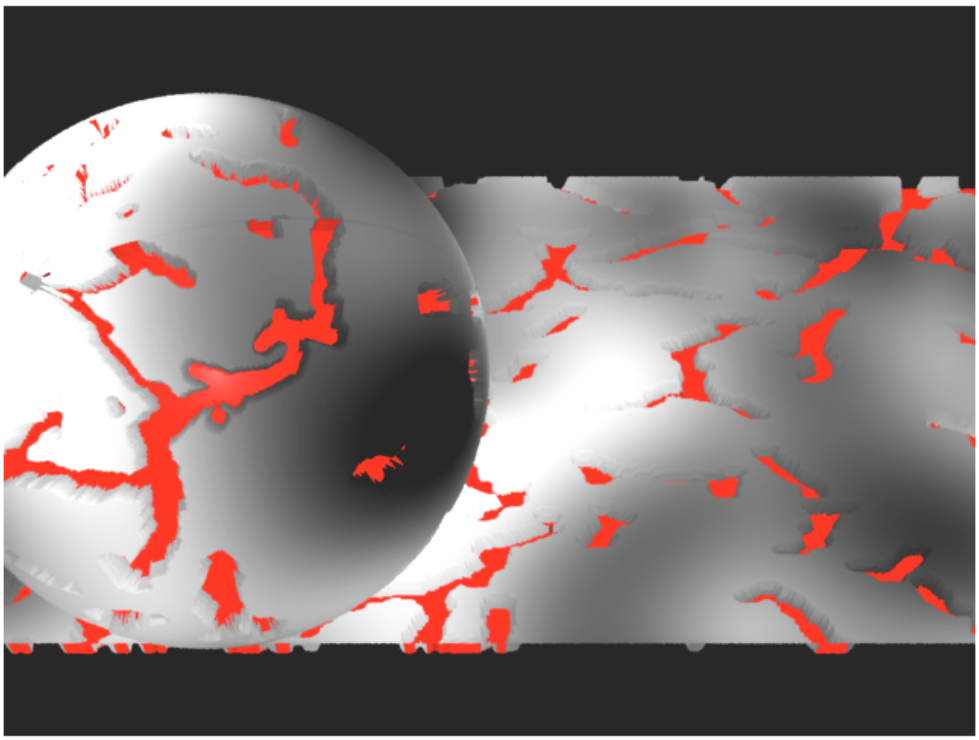
Render 6 Early experiment with using the alpha channel to drive a surface change along with a texture
for the parts not affected by the surface change..



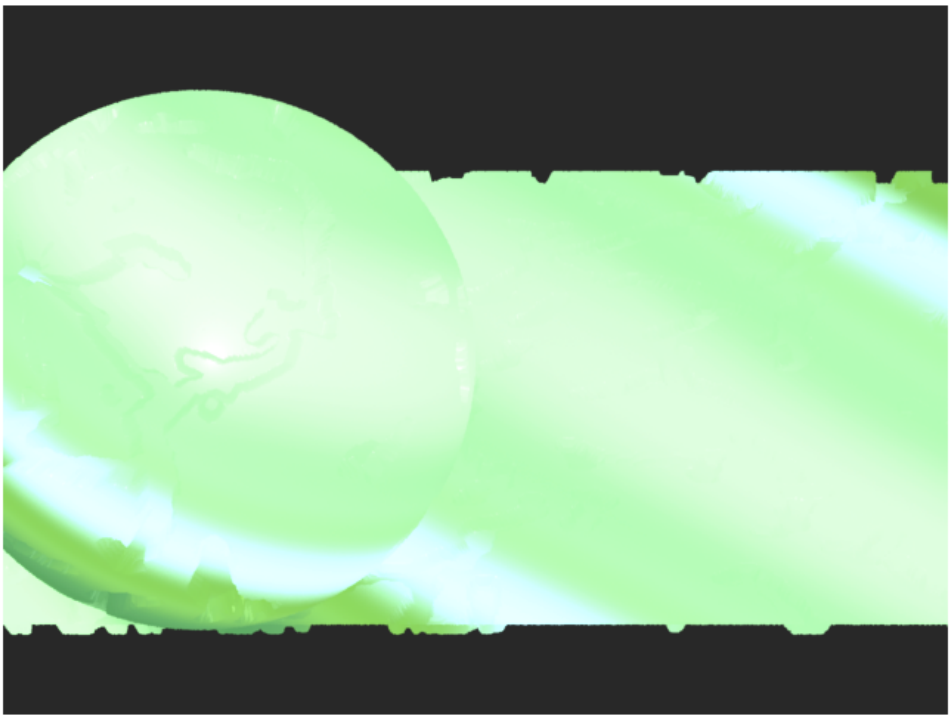
Render 7 Using a crocodile texture with alpha compositing to drive the surface change.



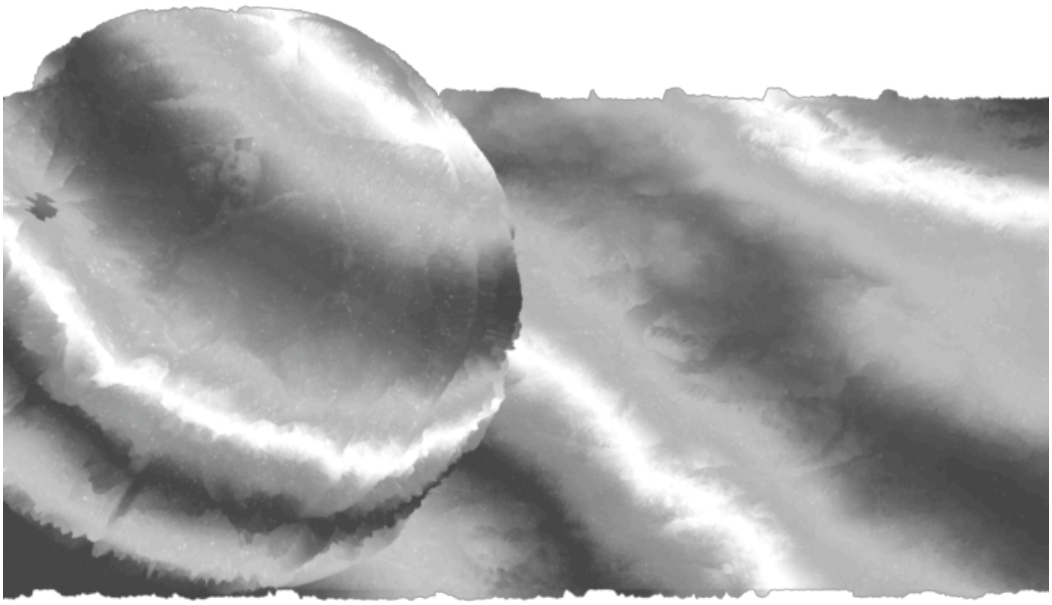
Render 8 Trying to create noise on the surface area not affected by the alpha driven surface change



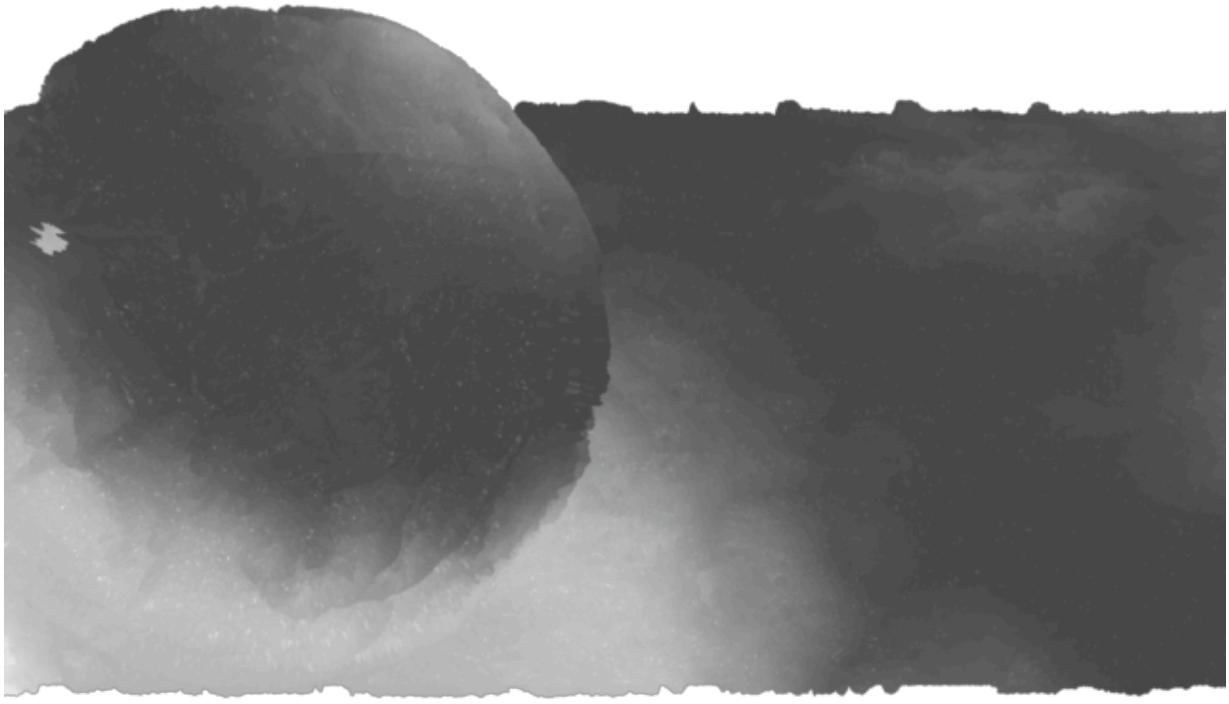
Render 9 Using marble from Essential Renderman Stephenson (2007,P.211), & my displacement shader



Render 10 Experiments with changing the marble shader



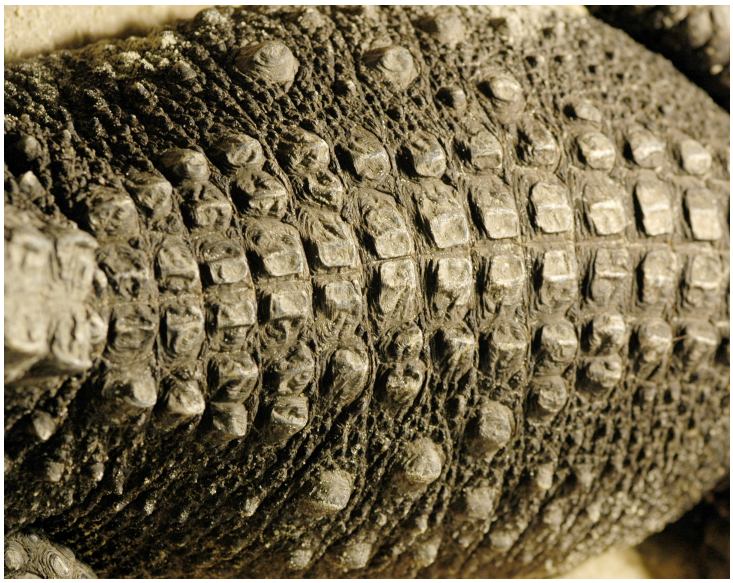
Render 11 Experimenting with noise effect.



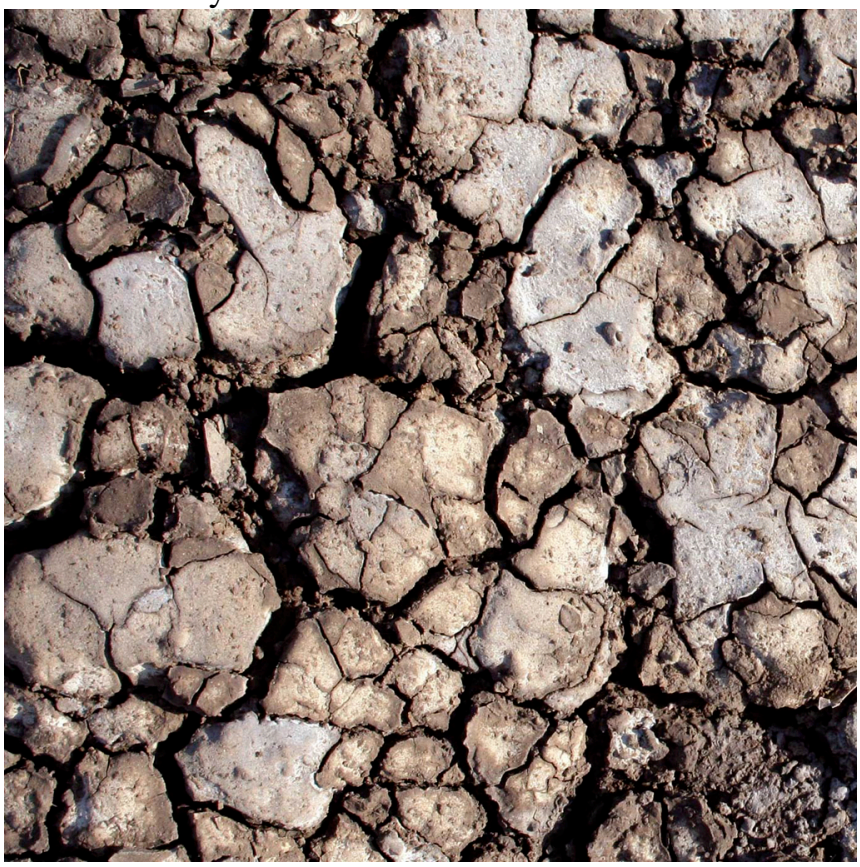
Appendix B

Source images converted to .tx and used in my shaders

Picture 1 - Crocodile texture



Picture 2 - Dry cracked mud



Picture 3
Dry cracked mud.

Appendix C

Renders from original code in Essential Renderman Stephenson(2007)

Familiarizing myself with the workflow for writing shaders and rib files