



Escher for Nuke User's Guide

Release 1.2.0

artixels

February 2015

Rev. B (20150228)

Escher for Nuke User Guide ("This Document") Copyright © 2012-2015 Artixels. All Rights Reserved.

Use of this document and the Escher software is subject to an End User License Agreement (the "EULA"), the terms of which are incorporated herein by reference. This document and the Escher software may be used only in accordance with the terms of the EULA. This document, the Escher software and all intellectual property rights relating thereto are and shall remain the sole property of Artixels.

Artixels assumes no responsibility or liability for any errors or inaccuracies that may appear in this document and its content is subject to change without notice. The content of this document is furnished for informational use only. No part of this document may be reproduced, stored in a retrieval system or transmitted, in any form or by any means, electronic, mechanical, recording or otherwise, without the prior written permission of Artixels.

The names of other actual companies and products mentioned in this document (including, but not limited to, those set forth below) may be the trademarks or service marks, or registered trademarks or service marks, of their respective owners. No association with any company or product is intended or inferred by the mention of its name in this document.

Nuke™ compositing software © 2012 The Foundry Visionmongers Ltd. All Rights Reserved.

Nuke™ is a trademark of The Foundry Visionmongers Ltd.

Linux ® is a registered trademark of Linus Torvalds.

Windows ® is the registered trademark of Microsoft Corporation.

Mac, Mac OS, Leopard and Snow Leopard are trademarks of Apple, Inc., registered in the U.S. and other countries.

Maya ® is a registered trademark of Autodesk, Inc., in the USA and other countries.

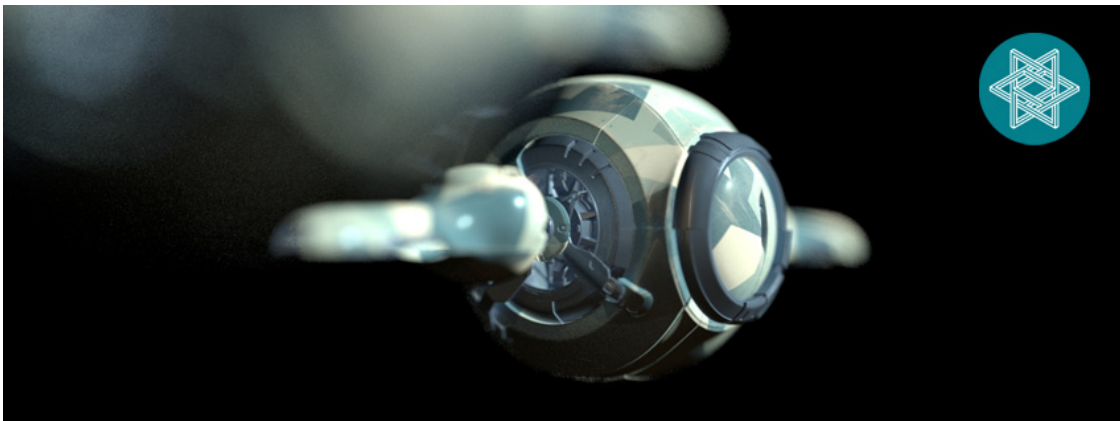
3D character model 'George' courtesy of Treehouse studio.

CONTENTS

1	Overview	1
1.1	Escher for Nuke	1
1.2	From plain 2D to 3D	1
1.3	2.5D shaders	4
1.4	Image generators	6
1.5	Other Escher nodes	8
1.6	Installation and Licensing	9
1.7	Support	9
2	Image Generators	11
2.1	caustix	11
2.2	skyWalker	15
3	Technical Operators	23
3.1	depthOp	23
3.2	normalOp	30
3.3	aaOp	34
3.4	SHOp	36
4	2.5D Shaders	39
4.1	SHeader	39
4.2	glossy	43
4.3	depthTrooper	46
4.4	depthVader	49
5	Deep Ops	55
5.1	Deep Normal Maps	55
5.2	deepShuffleOp	62
5.3	deepNormalOp	64
5.4	deepSHeader	66
5.5	deepGlossy	70

OVERVIEW

1.1 Escher for Nuke



Escher for Nuke is a suite of image generators, image-based lighting (IBL) and 2.5D creative tools designed for Nuke compositors who are looking for tools to explore the creative possibilities stashed between 2D and 3D. The goal of Escher is to enable creative 3D image making in a different way and at the same time taking full advantage of the organic nature of 2D image processing and manipulation.

Escher relies on 2.5D images input, both depth images (or gray-scale images) and 3D normal map images are regarded as 2.5D images to Escher. Depth image is interpreted by Escher as the distance of each pixel from the camera, so it essentially contains information more than just a plain 2D image. Normal map image provides the normal vector direction of each pixel which is good enough for basic shading. Usually, it's preferred to have both depth and normal vector passes rendered at higher resolution (2x up to 8x) than the beauty pass because of the aliasing consideration, **Escher 1.2** has brought new solution to this issue by introducing new Ops that support **DEEP data** passes directly and this practically eliminates the alias problems of 2.5D rendering.

Designed as a unique toolset which enables creative manipulations of 2.5D images to create 3D images, we believe Escher will quickly become your weapon of choice when it is about unseen creative visuals. In the following sections, we will explore a few examples of how Escher may be used.

1.2 From plain 2D to 3D

Escher interprets 2D gray-scale images as depth information, so the input is not limited to depth passes rendered from CGI renderers and it simply accepts any gray-scale image. Let us consider an image as shown in **Image 1.2A**, it's a plain gray-scale image and we may give it a twist by applying the GodRays node to create a more interesting image as



shown in **Image 1.2B**. This new image has better range of intensity and could be interpreted as a 2.5D depth image, so it is a good candidate input for Escher.

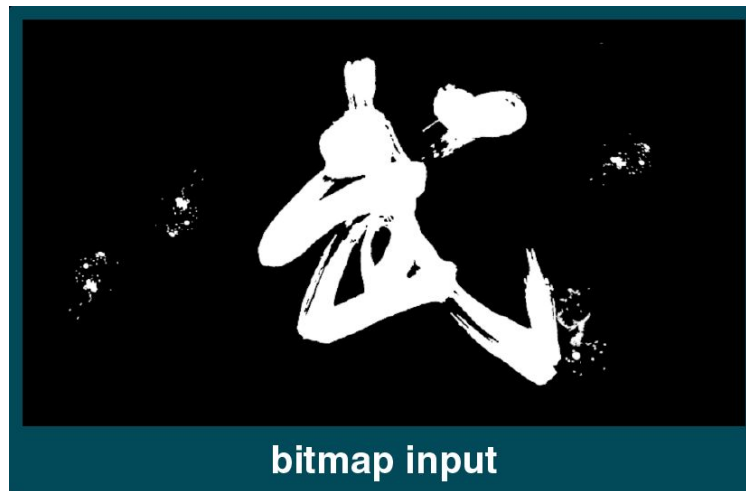


Image 1.2A plain gray-scale image input

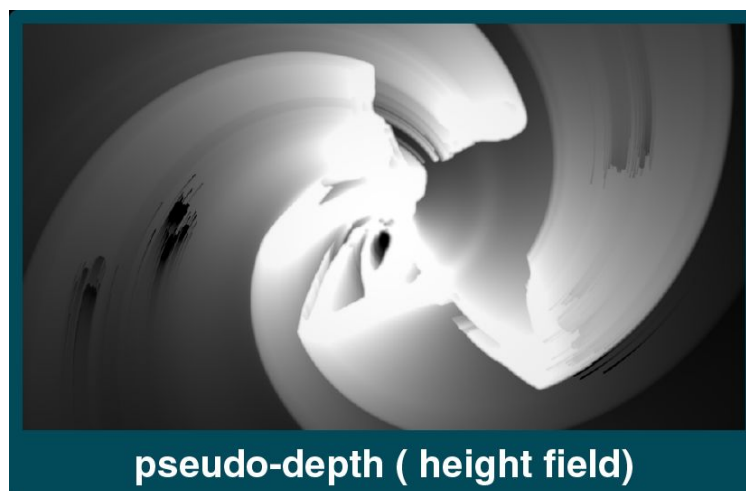


Image 1.2B pseudo-depth image output from Nuke's Godrays node

Escher has a node named **depthVader** which is able to output normal vector information based on a depth image, so we can quickly obtain a normal map image as shown in **Image 1.2C** which is ready to be shaded by Escher 2.5D shaders. In this example, we used Escher's **Glossy** shader (for high frequency lighting) to shade this normal map image with a lightprobe image and the result is shown in **Image 1.2D** and the overall workflow is shown in figure **1.2E**.

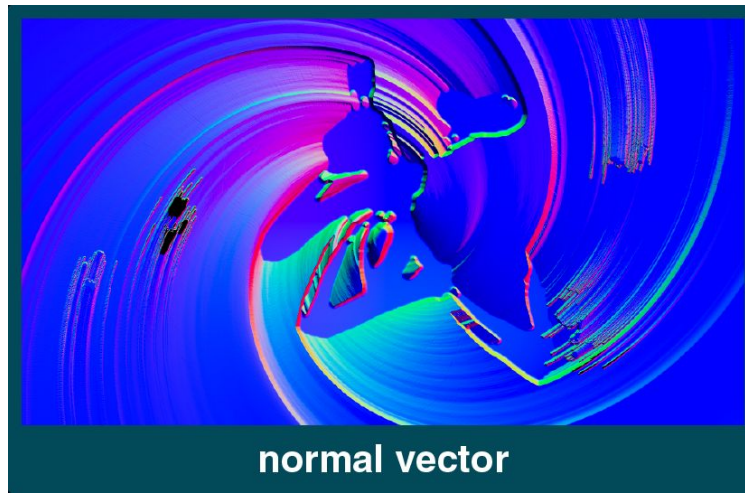


Image 1.2C normal map output from Escher's **depthVader** node



Image 1.2D rendered output from Escher's **glossy** node



Figure 1.2E typical Escher 2.5D shading process based on a plain gray-scale image

1.3 2.5D shaders

Imagine a typical case that the director wants to modify the background environment of a comp, then it is often necessary to request new renders from your CGI team due to the change in lighting. **With Escher for Nuke, you could quickly create new image elements to tweak your comp without going back to 3D.** Escher may be used to create new image elements for more flexible comp using only normal map passes from CGI renderers (includes Nuke's scanline renderer output).

Escher has a powerful and highly interactive Spherical Harmonics (SH) based 2.5D shader (designed for low frequency lighting) which takes a lightprobe image as light source to create a lambertian shaded image of the 2.5D normal map input. **Image 1.3A** shows the input normal map and **Image 1.3B** shows the output of Escher **SH**ader. In case you have the 3D asset available, you may also output the normal map image from Nuke's scanline renderer to take advantage of Escher 2.5D shaders too.

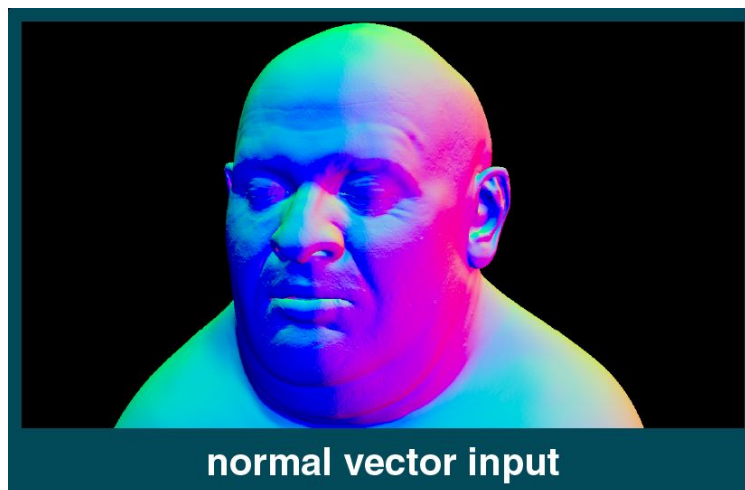


Image 1.3A normal map rendered from Maya

Note: Escher 2.5D shaders are highly interactive, we believe they can also be used for Look development inside Nuke too.

Using additional Escher 2.5D shaders, Nuke artists may take full advantage of 2.5D workflow to create full comp using limited CGI input passes, say diffuse RGB, depth and normal vector passes only. **Image 1.3C** shows a simple comp of using Escher 2.5D shaders with only raw RGB, depth map and normal vector passes only. **Figure 1.3D** outlines the workflow and shaders used to produce **Image 1.3C**.



Image 1.3B Escher's SHader rendered output



Image 1.3C a simple full-comp using only Escher 2.5D shaders

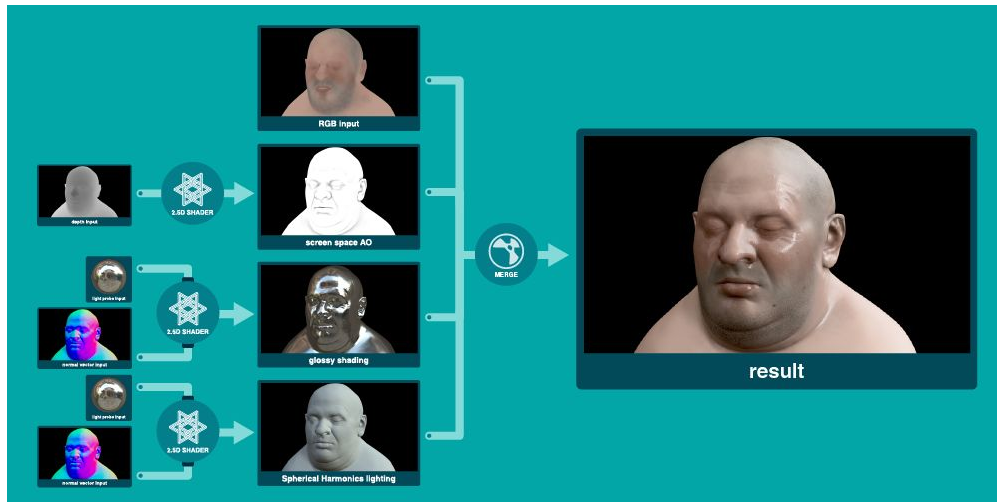


Image 1.3D workflow

1.4 Image generators

Escher's image generators are friendly to Nuke's 3D tools, for example, you may project Escher **caustix** node output onto geometry in the 3D scene as displacement map to create a 3D rippling surface. In addition, the distance between projection camera and the geometry can drive the depth parameter of **caustix** via expression, so that the generated patterns would be more interactive with the 3D environment.

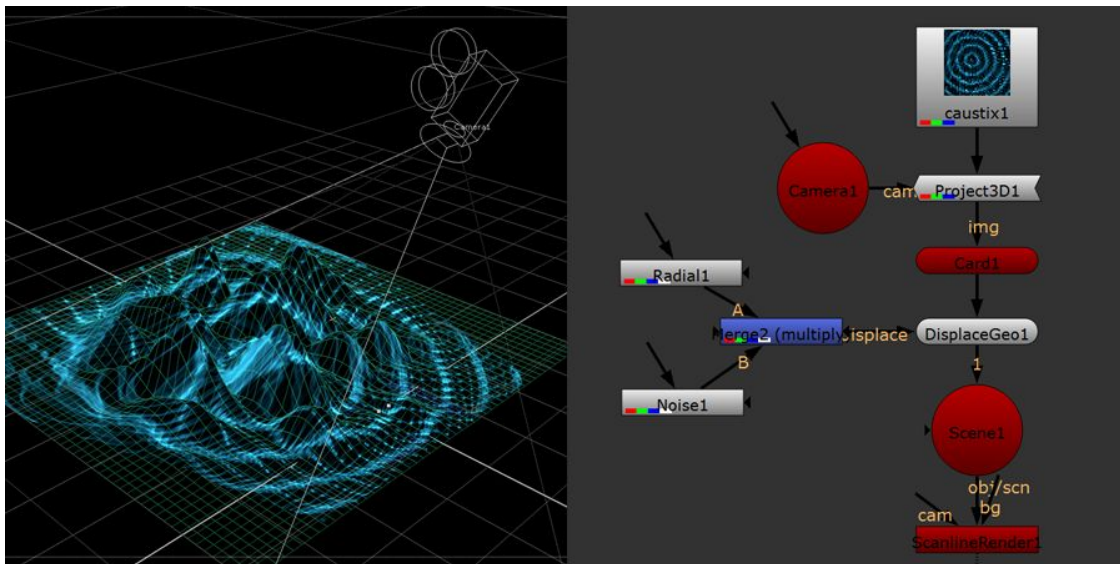


Image 1.4A Using caustix with Nuke's 3D sub-system

Image generators are useful for lighting and shading purposes in Escher workflow. **Figure 1.4B** shows a workflow of using skyWalker output to drive Nuke's 3D environment node. Of course, they can be used as lightprobes for Escher 2.5D shaders or other image based lighting purpose. **Figure 1.4C** and **Figure 1.4D** show the results of using them as lightprobes for Escher 2.5D shading.

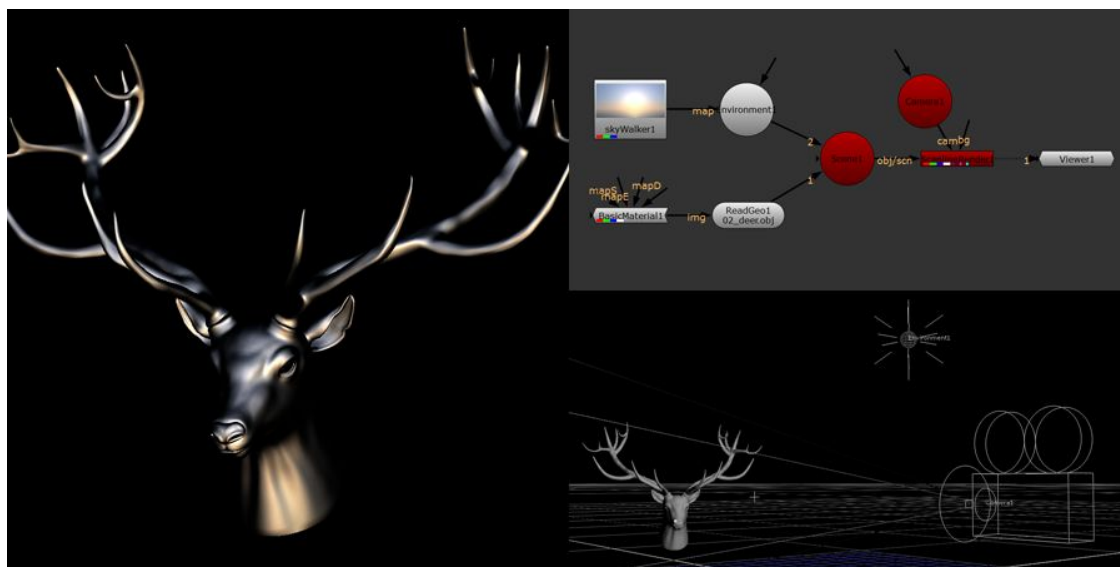


Figure 1.4B using skyWalker and SHader in Nuke's 3D system

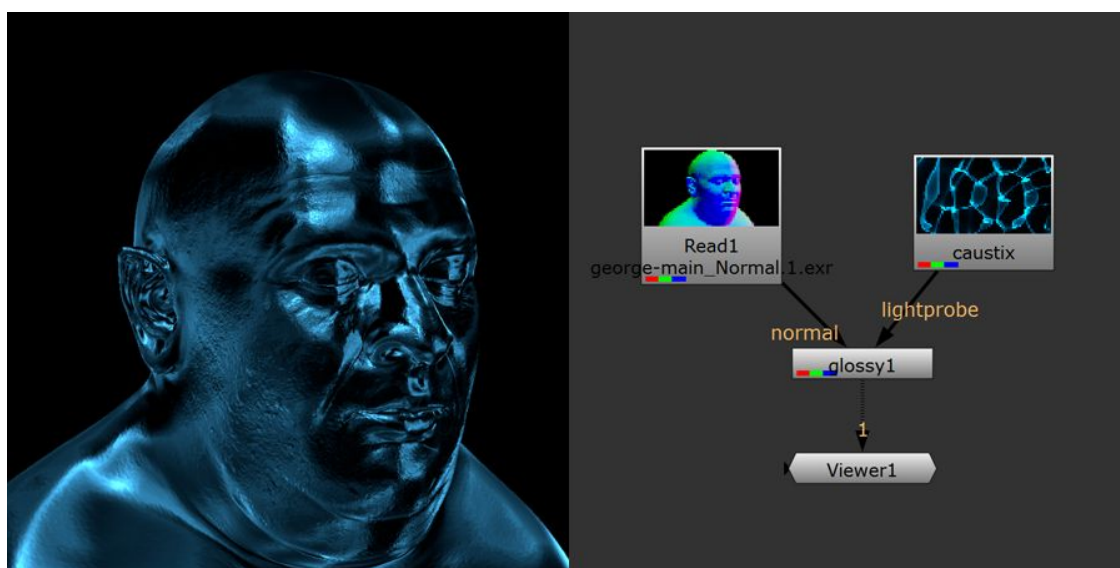


Figure 1.4C caustix output as lightprobe for Escher 2.5D Glossy shader



Figure 1.4D skyWalker output as lightprobe for Escher 2.5D SH shading

1.5 Other Escher nodes

There are other Escher nodes not covered in this overview, they include a very practical node dedicated for morphological anti-aliasing named **aaOp** and two technical nodes designed to do routine operations necessary for depth and normal vector data processing. **depthTrooper** is an image-space 2.5D shader which renders Ambient Occlusion like shading using only a depth image, **image 1.5A** shows a rendered output of **depthTrooper**.

depthVader is the node which outputs normal vector information based on depth input that we have briefly mentioned but this is only one of the many operations that it offers and we strongly encourage artists to explore the full potential of this node. **Image 1.5B** shows how the gradient blur mode of **depthVader** is used to fake a subsurface scattering effect.

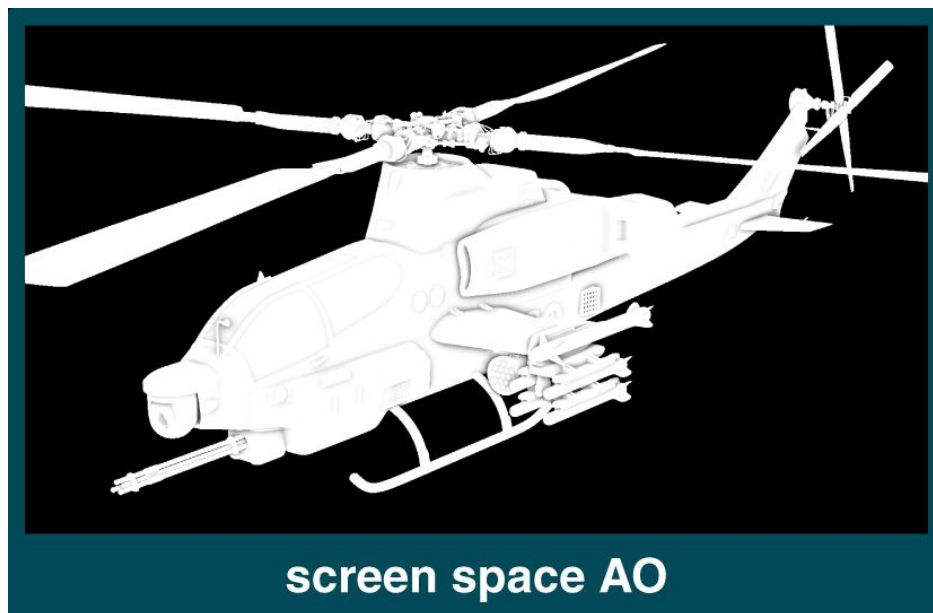


Image 1.5A DepthTrooper output



Image 1.5B Before and After with **depthVader**'s gradient blur

1.6 Installation and Licensing

Please refer to *Escher Installation and Licensing Guide* included in the installation package for all installation and licensing details.

1.7 Support

The official support forum of Escher for Nuke is available at:

<http://artixels.zendesk.com>

Access requires registration and all customers should receive their support forum access information together with their license.



IMAGE GENERATORS

2.1 caustix

2.1.1 Introduction

caustix is a high performance image generation plug-in designed for creating rippling caustics patterns. It simulates how distant light shines through a body of rippling water onto a plane of constant depth.

To accelerate the creative process, an intuitive GUI is included for direct manipulation of the pulse generators, users may control the position, amplitude and frequency of each pulse independently.



Image 2.1A rendered output of caustix

2.1.2 Quick Start

1. Connect the **caustix** node to the Viewer, four on-screen controls (pulse) will be displayed on the Viewer.

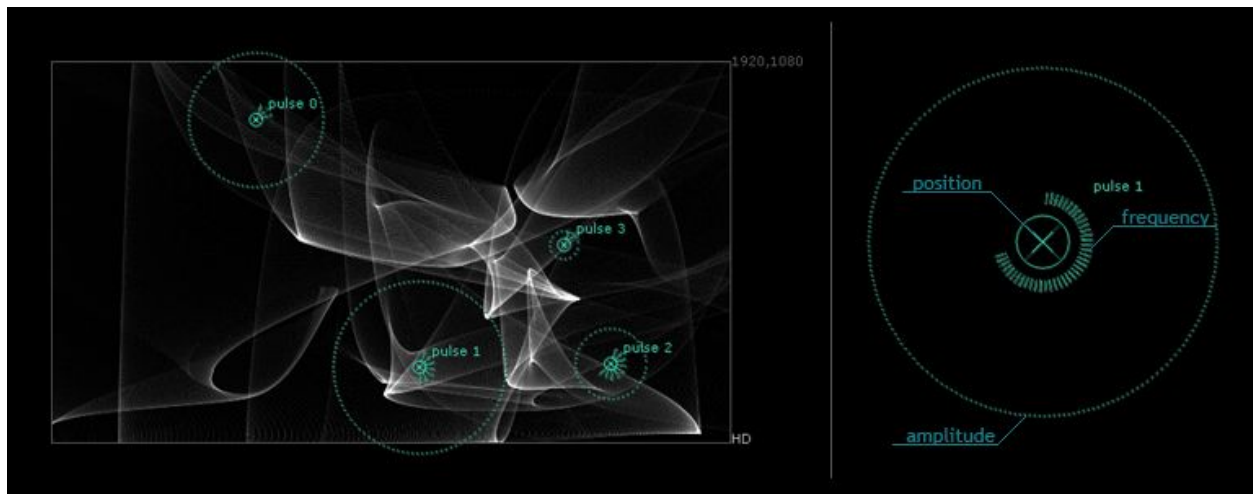


Figure 2.1B GUI of caustix

2. Each pulse has a centre point from which emits **caustix**, users may re-position each pulse by click-and-drag this centre point.
3. Apart from the positions of each pulse, there are two controls which determine the look of the caustics pattern: the dashed circle around the pulse controls the *amplitude*, the larger the circle, the higher the amplitude is.
4. The second control is a dial which controls the *frequency*, turning it clockwise to increase the frequency.
5. Additional controls are available on the node properties panel which allow more precise and delicate manipulation of **caustix**. All the controls can be animated by using the standard animation tools.

2.1.3 Parameters

caustix has 6 sub-panels, **render settings**, 4 individual **pulse control**, and **postprocess**.

- **format** Image output size and format.

render settings

- **extent** A scale factor which provides an overall zooming in/out alike effect but the pulses are kept stationary relative to the image boundary (**Figure 2.1D**).
- **depth** The depth of the image plane from the rippling water surface (**Figure 2.1E**).
- **timestep** It determines the time-step of simulation between each frame, smaller value gives slower motion.

individual pulse control

- **as sink** Check to turn the pulse into a sink (i.e. reversed pulse direction).
- **amplitude** Pulse's oscillation amplitude.
- **phase** Phase shift/offset of the pulse.
- **frequency** Pulse's oscillation frequency.

postprocess

- **color** Gives an overall color to the caustics pattern.
- **intensity** Adjusts the brightness.
- **blur** Blurriness control.

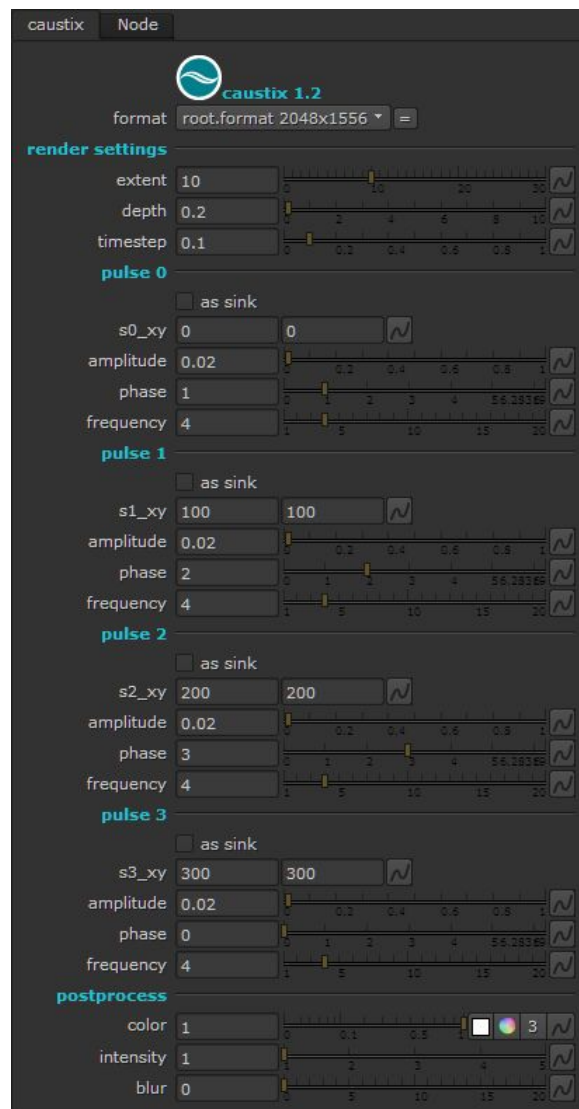


Figure 2.1C control panel of caustix

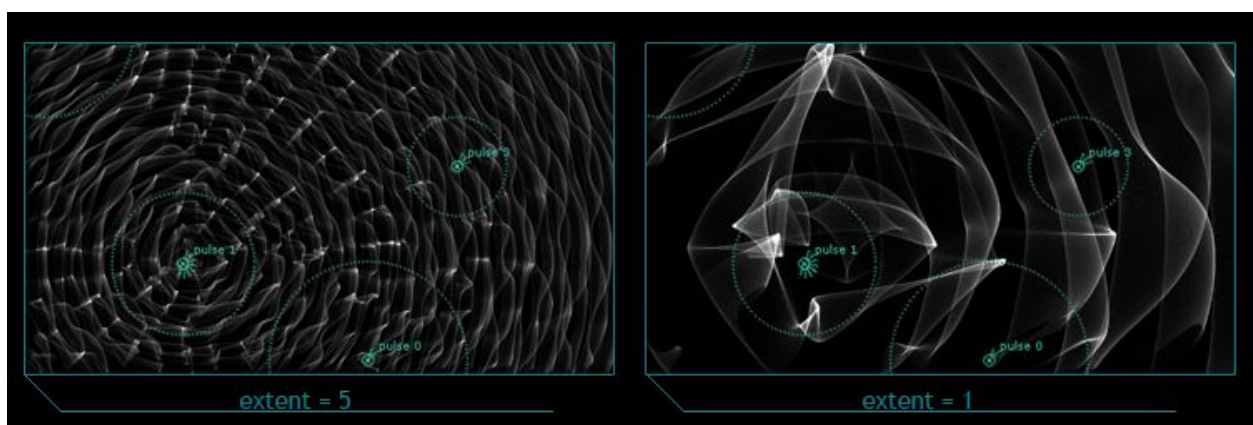


Figure 2.1D different values of extent

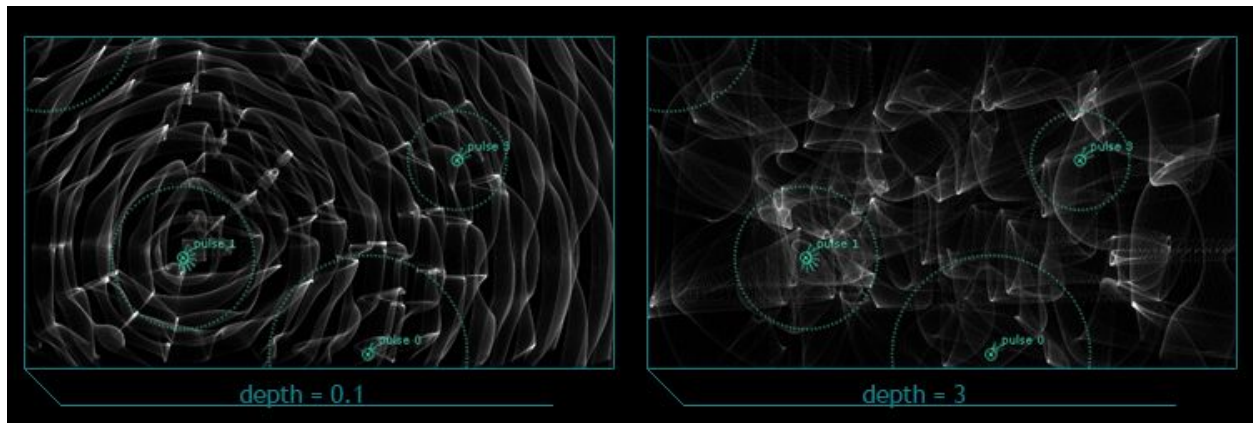


Figure 2.1E different values of depth

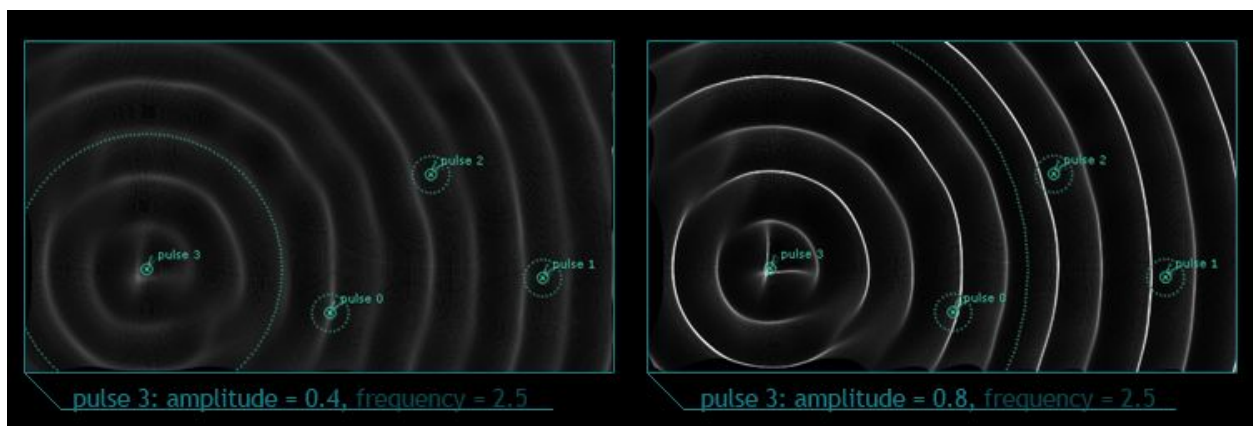


Figure 2.1F different values of amplitude

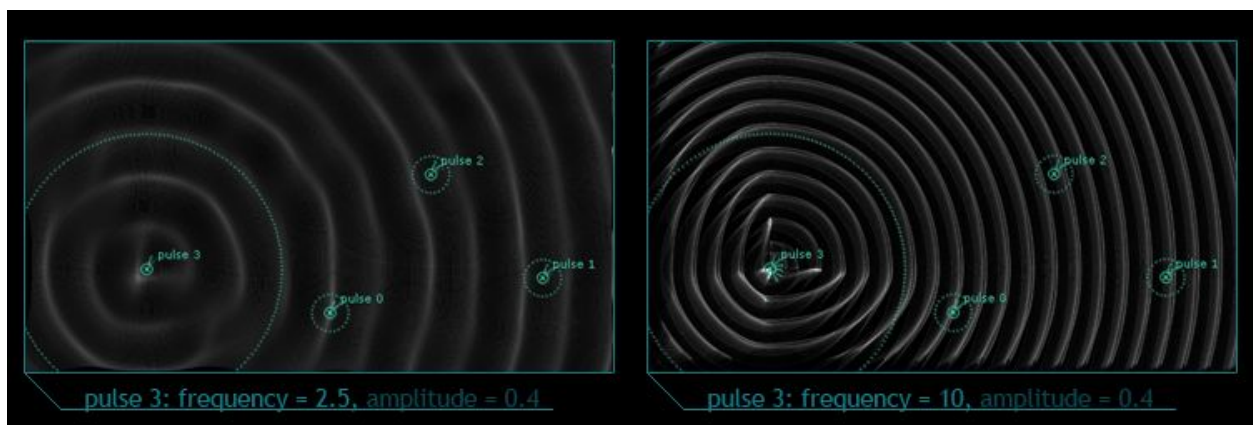


Figure 2.1G different values of frequency



2.2 skyWalker

2.2.1 Introduction

skyWalker is an image generation plug-in for creating physically plausible daylight. It supports four different output modes including pinhole camera, and 3 light probe formats, i.e. mirror ball, angular and latlong.



Image 2.2A rendered output of **skyWalker**

2.2.2 Quick Start

1. Create a **skyWalker** node and connect the **skyWalker** node to the Viewer, a sky image is generated.
2. Choose the sky *image format* in **output** sub-panel.
3. Choose the desired render mode: *pinhole, mirror ball angular or latlong*.
4. Adjust the sky attributes in the sky sub-panel : *sunZenith, turbidity, intensity* and *ground reflection*.
5. Adjust the camera/image plane attributes in the camera sub-panel : *horizontal fov, camZenith and camAzimuth*.

2.2.3 Parameters

skyWalker has 3 sub-panels, **output**, **sky**, and **camera**.

output

- **format** Image output size and format.
- render mode

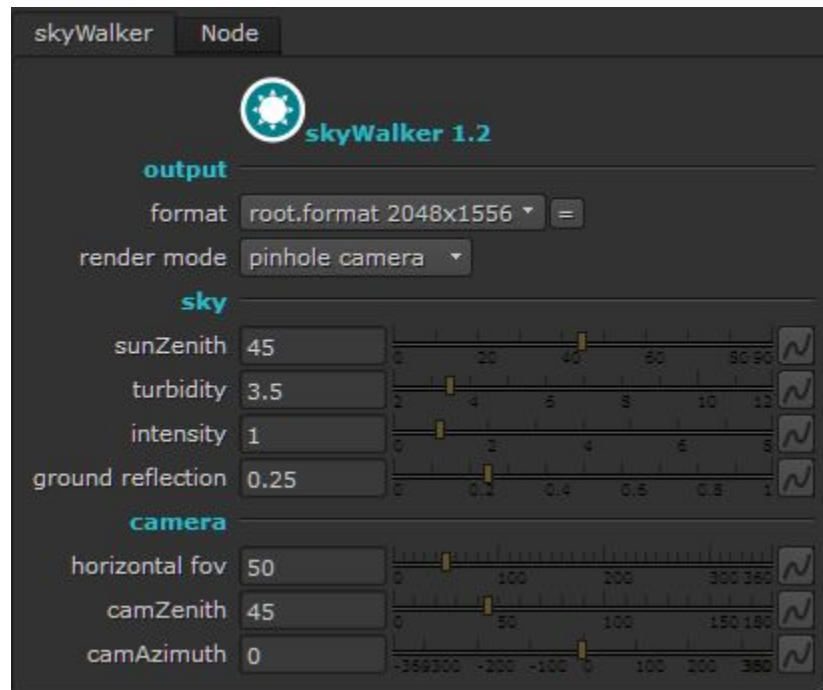


Figure 2.2B control panel of skyWalker

- *pinhole(default)* - It is similar to regular CGI pinhole camera, it displays what a camera looks through a window.(Figure 2.2C)
- *mirror ball* - It outputs a mirror ball reflection rendering of the skydome.(Figure 2.2D)
- *angular* - It is similar to mirror ball, but the radial dimension is mapped linearly with the angle, so the pixels around the edge are not squeezed too much, giving a better sampling result.
- *lat long* - Mainstream latlong format, the latitude and longitude of a skydome are unwrapped to form a rectangular image, polar area is spread out.

sky

- **sunZenith** Refers to the angle between the sun and the zenith, expressed in degrees.(Figure 2.2E)
- **turbidity** The clarity and amount of haze in the atmosphere. it ranges from 2 (a clear sky) to 12(an overcast sky), turbidity affects the color and intensity of the sky, the strength and color of the glow around the sun.(Figure 2.2F)
- **intensity** The overall brightness of the sky.
- **ground reflection** Skydome is only defined for upper hemisphere, so the ground reflection serves to fill up the lower hemisphere.(Figure 2.2G)

camera

- **horzFov** The horizontal angle of field of view, expressed in degrees. It is the angular extent of the skydome at the centre position. The horizontal fov is only applicable to *pinhole* and *lat long* mode.(Figure 2.2H)
- **camZenith** The angle of the elevation of the camera.(Figure 2.2I, Figure 2.2J)
- **camAzimuth** The rotating angle of the camera at vertical axis.(Figure 2.2K, Figure 2.2L)

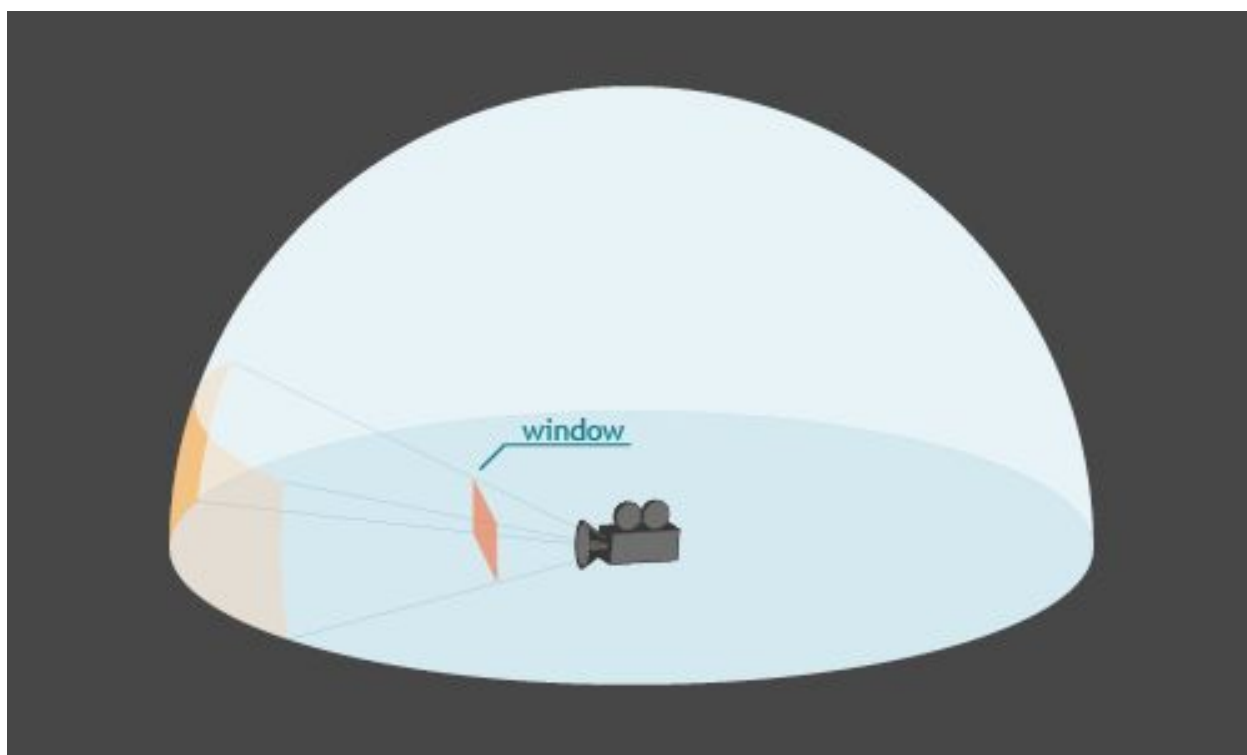


Figure 2.2C pinhole format

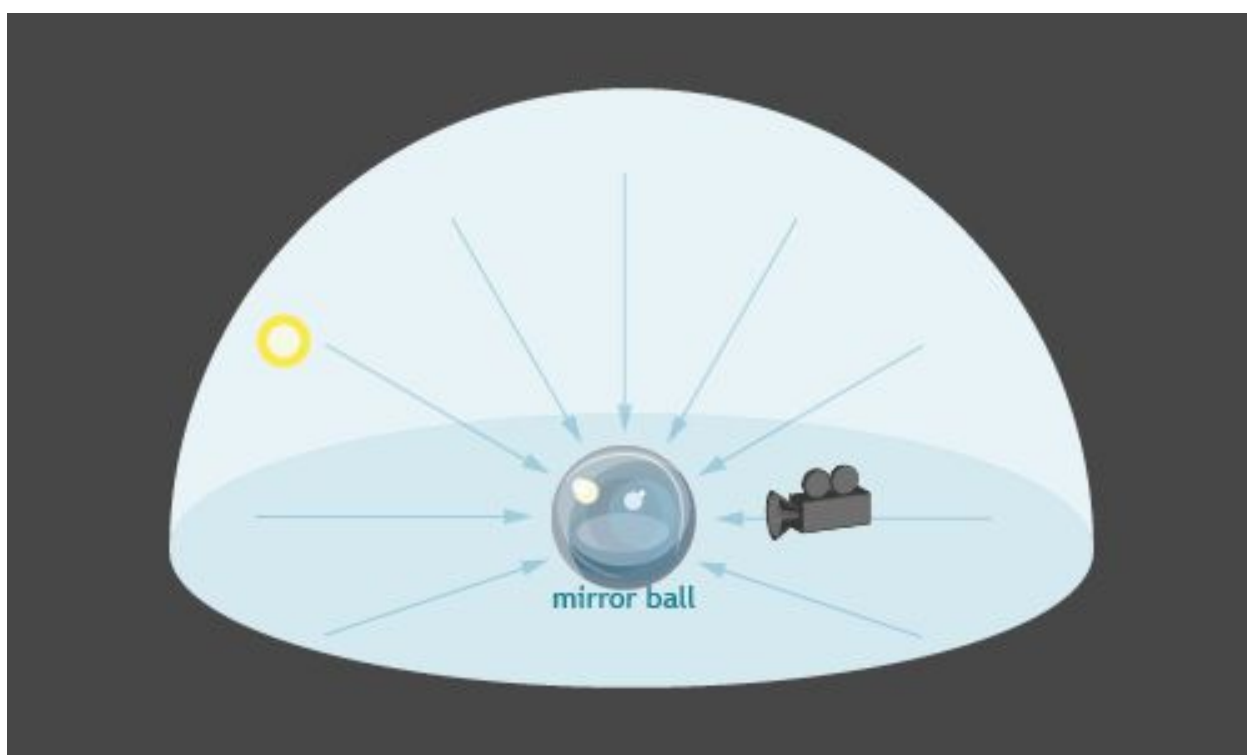


Figure 2.2D mirror ball / angular format

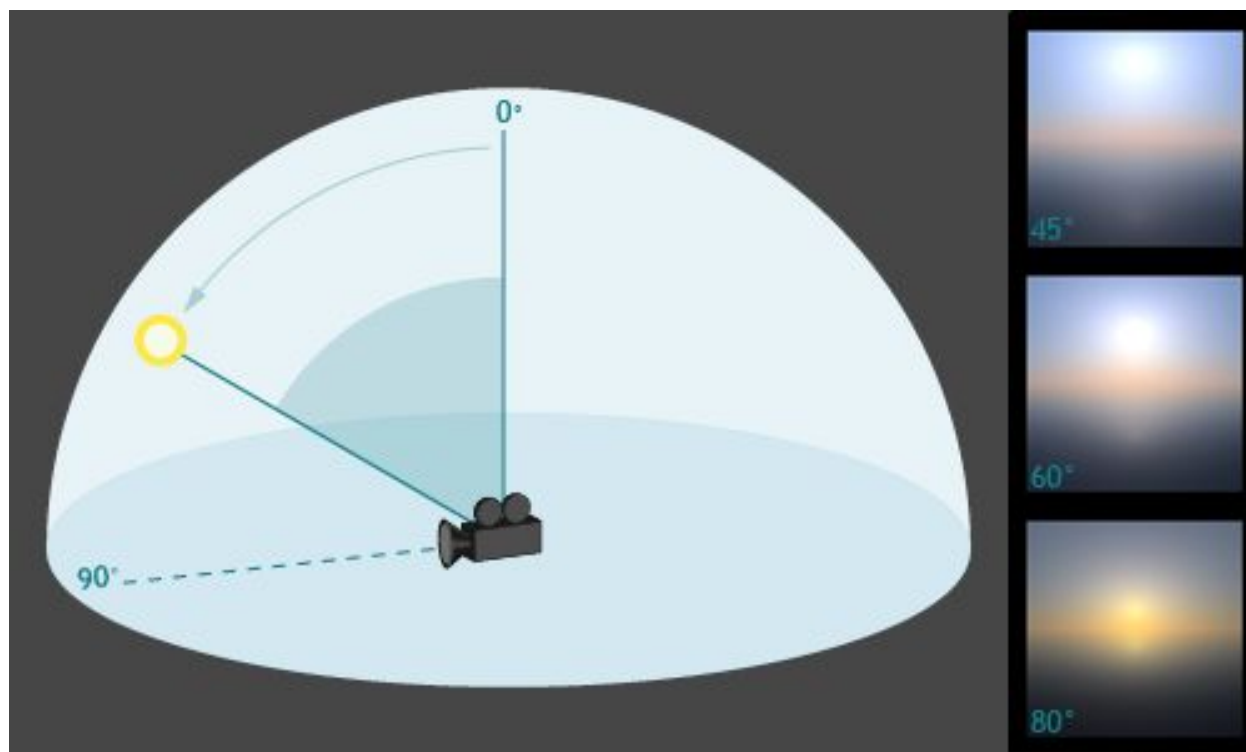


Figure 2.2E annotated diagram of sunZenith

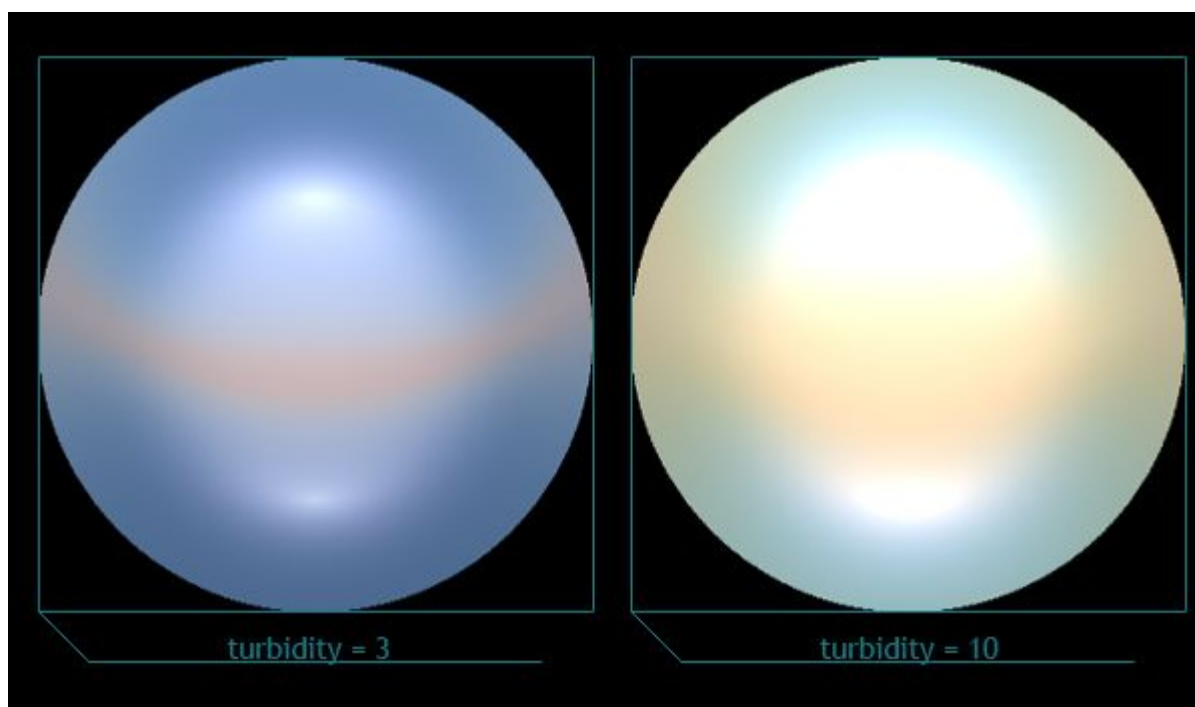


Figure 2.2F different values of turbidity

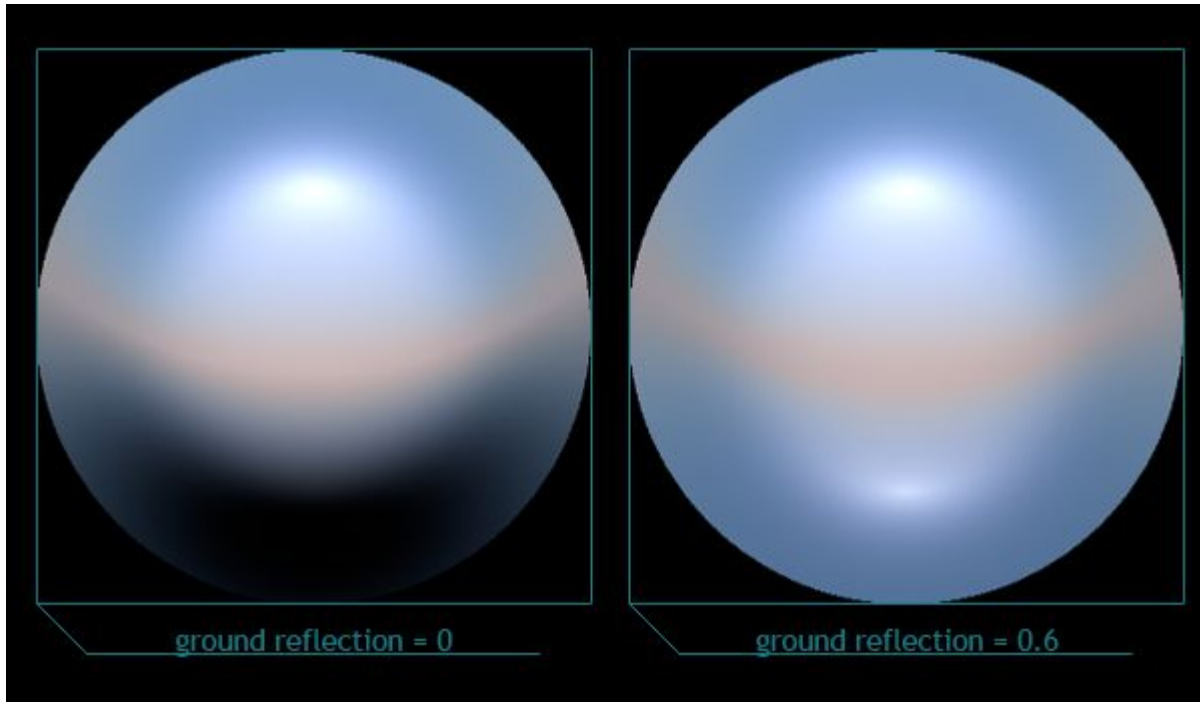


Image 2.2G different values of ground reflection

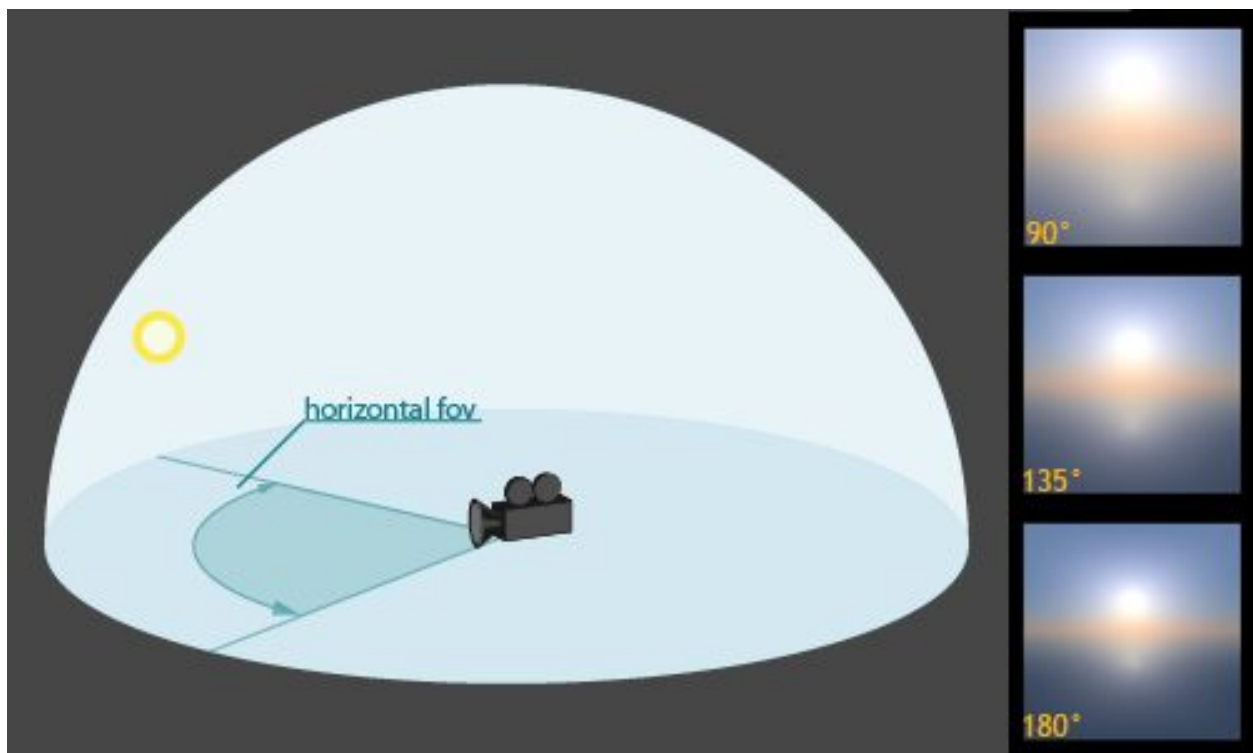


Figure 2.2H annotated diagram of horzFov

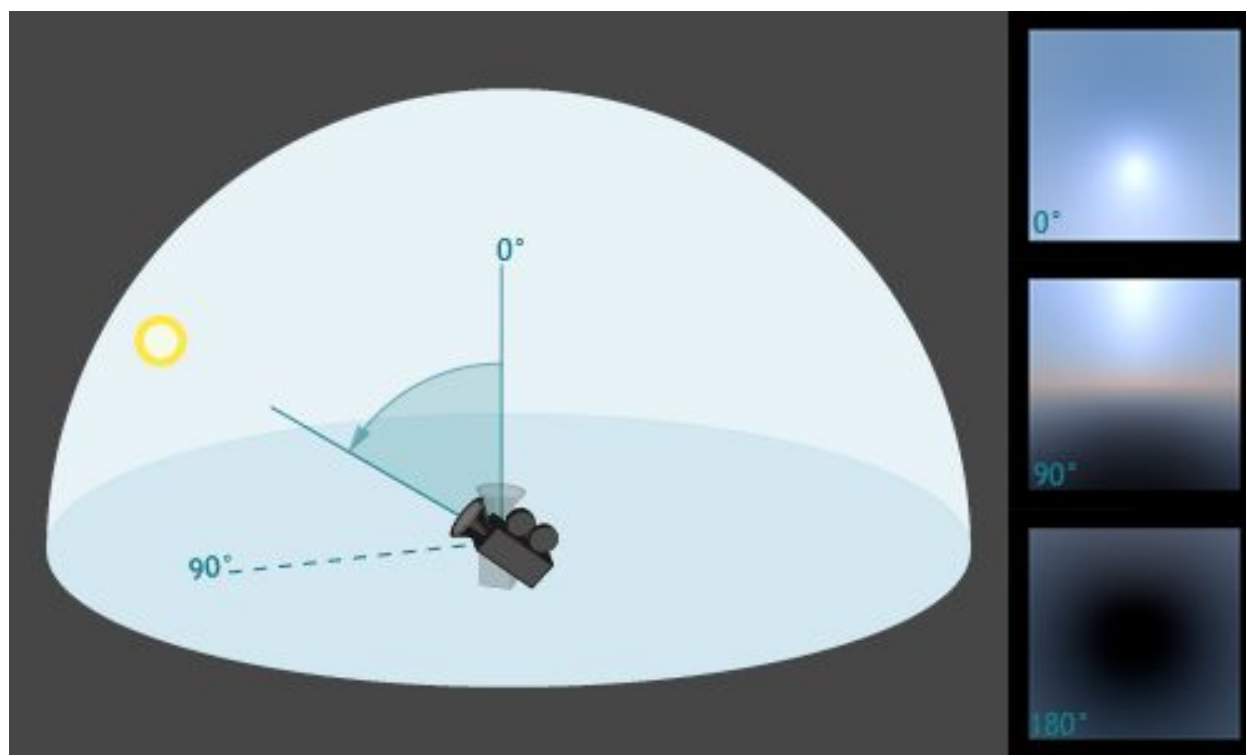


Figure 2.2I annotated diagram of camZenith(pinhole / latlong)

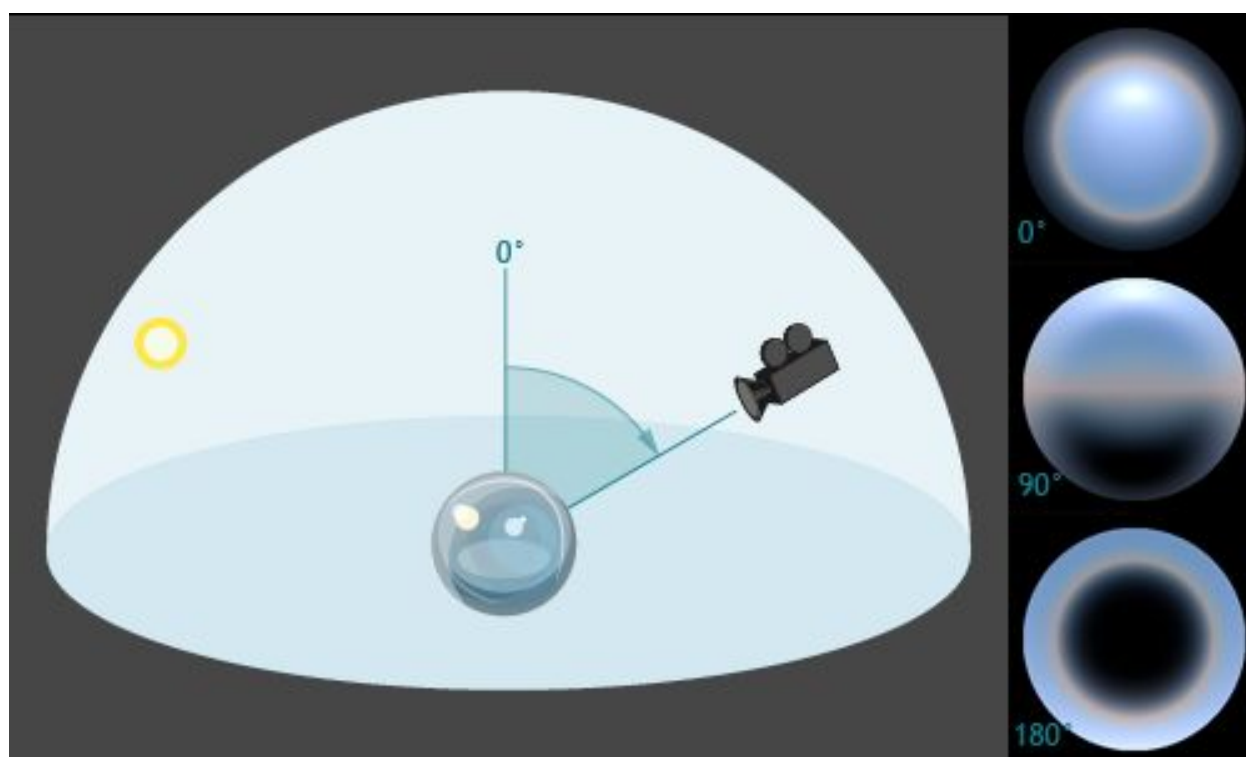


Figure 2.2J annotated diagram of camZenith(mirror ball / angular)

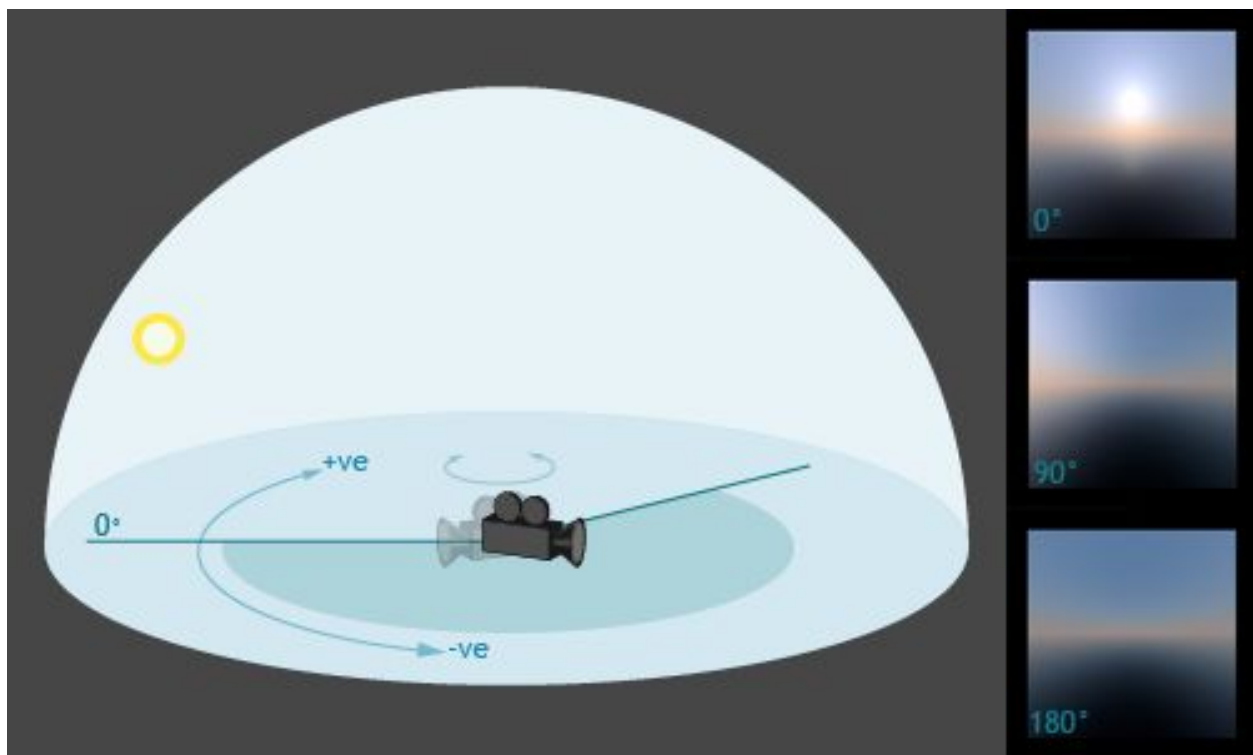


Figure 2.2K annotated diagram of camAzimuth(pinhole / latlong)

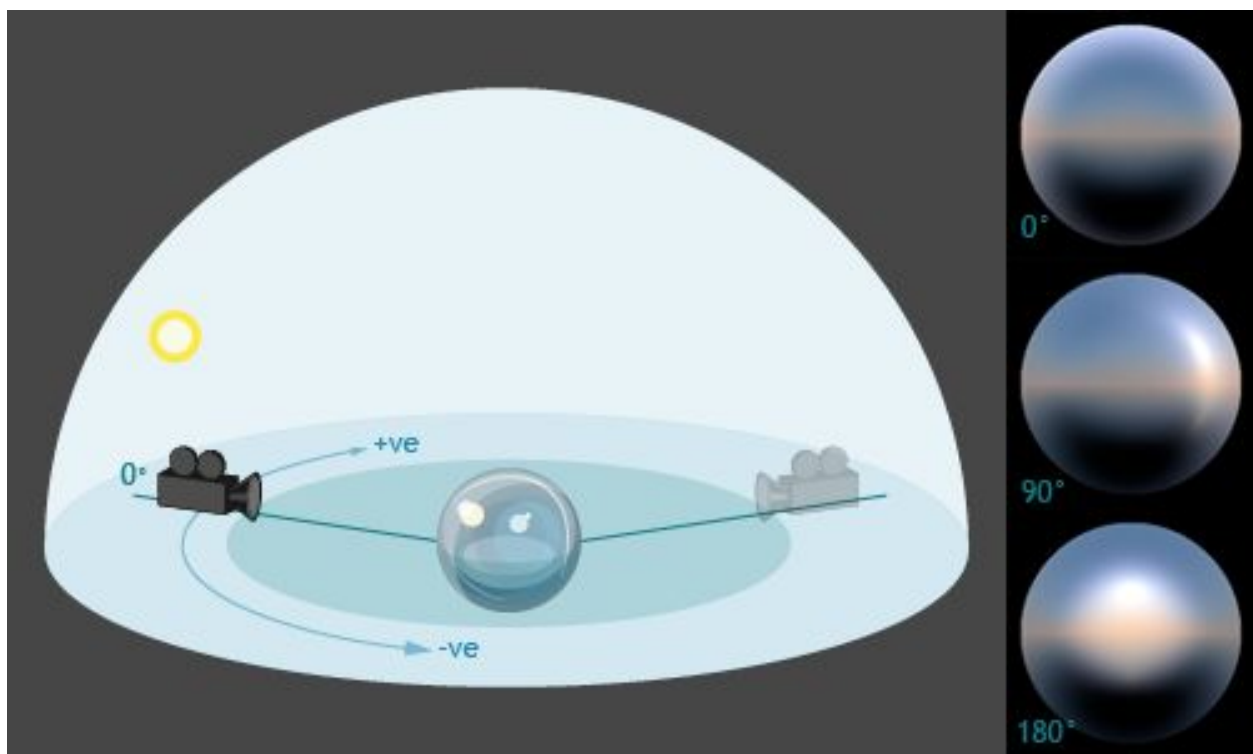


Figure 2.2L annotated diagram of camAzimuth(mirror ball / angular)



TECHNICAL OPERATORS

3.1 depthOp

3.1.1 Introduction

depthOp is a plug-in node specifically designed for depth data processing, it allows users to obtain the min/max depth values of each individual frame and the overall sequence-wide min/max values quickly. As different CGI renderers might output depth information differently, **depthOp** provides a stack of operations (we call them microOps) for easy conversion between different depth encoding standards. All depth-related plug-ins in escher assume a '0 to 1' normalised depth range. (close: 0.0, far: 1.0).

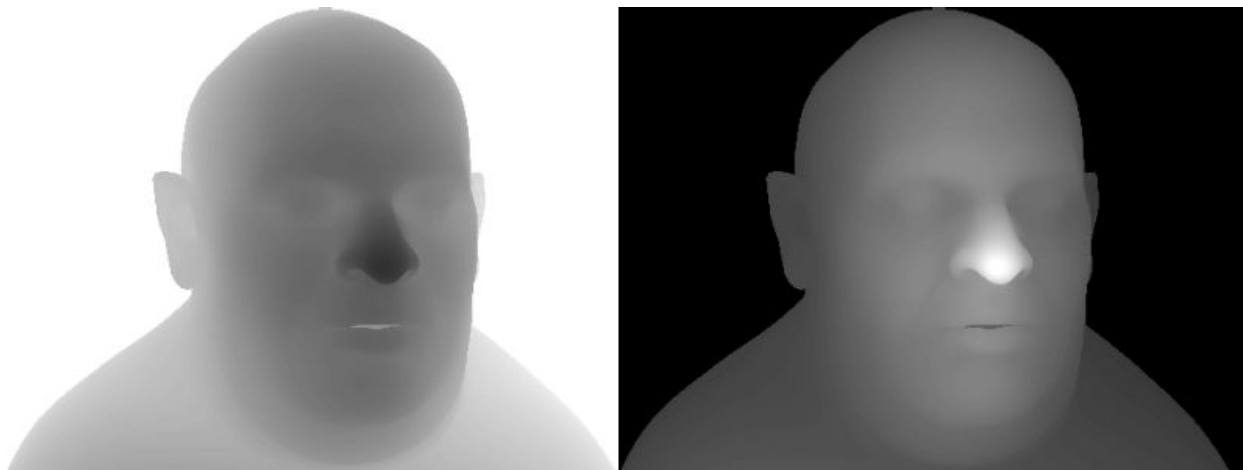


Image 3.1A rendered outputs of **depthOp**

3.1.2 Input

depthOp only takes one input, depth map.

3.1.3 Parameters

depthOp has two sub-panels, **statistics** and **microOps**.

statistics



- **analyse** Analysis of the incoming depth channel, finding the *absolute minimum*, *2nd minimum*, *absolute maximum* and *2nd maximum* depth values of each individual frame and the overall sequence.
- **multi-threaded** Check to enable multi-threaded processing of the analysis stage. Checked ON is recommended when the upstream nodes contain processing nodes, checked OFF is recommended when there is only a read upstream node.

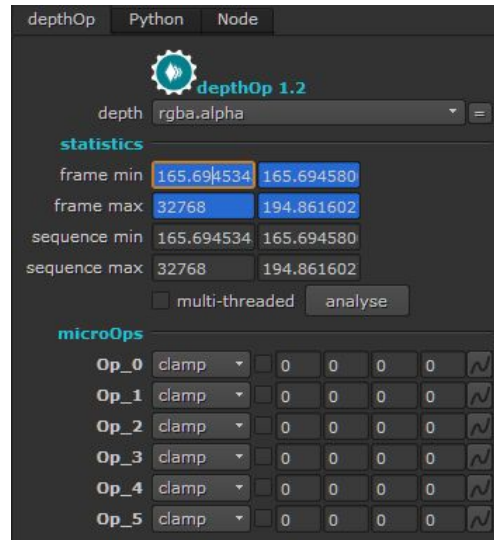


Figure 3.1B control panel of **depthOp**, statistics section

microOps

This version of **depthOp** offers six different microOps, and users may use the microOps according to their needs, the operations always start from the top (i.e. Op_0) and proceed downwards; each of them could be enabled or disabled independently by the checkbox on each row.

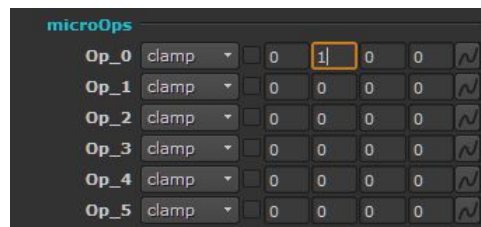


Figure 3.1C control panel of **depthOp**, microOps section

- *clamp* - Constrict the depth range.
- *offset* - Shift pixel values of the depth map either positively or negatively.
- *scale* - Multiply pixel values of the depth map by an input value.
- *inverse* - Invert the pixel values of depth by the equation: $1 / z$.
- *remap* - Lay out the original depth map range to a self-defined range.
- *replace* - Replace a range of values with an input value.



On each row of, there is a *<1> microOp selector*, *<2> on/off checkbox* and *<3> parameters*. Different microOps use different number of parameters and details are provided in the table below :

microOps	opParameter_0	opParameter_1	opParameter_2	opParameter_3
clamp :	minimum value	maximum value	/	/
offset :	value of offset	/	/	/
scale :	value of scale	/	/	/
inverse :	/	/	/	/
remap :	input minimum	input maximum	output minimum	output maximum
replace :	range minimum	range maximum	replace value	/

Table A - microOps and their parameters

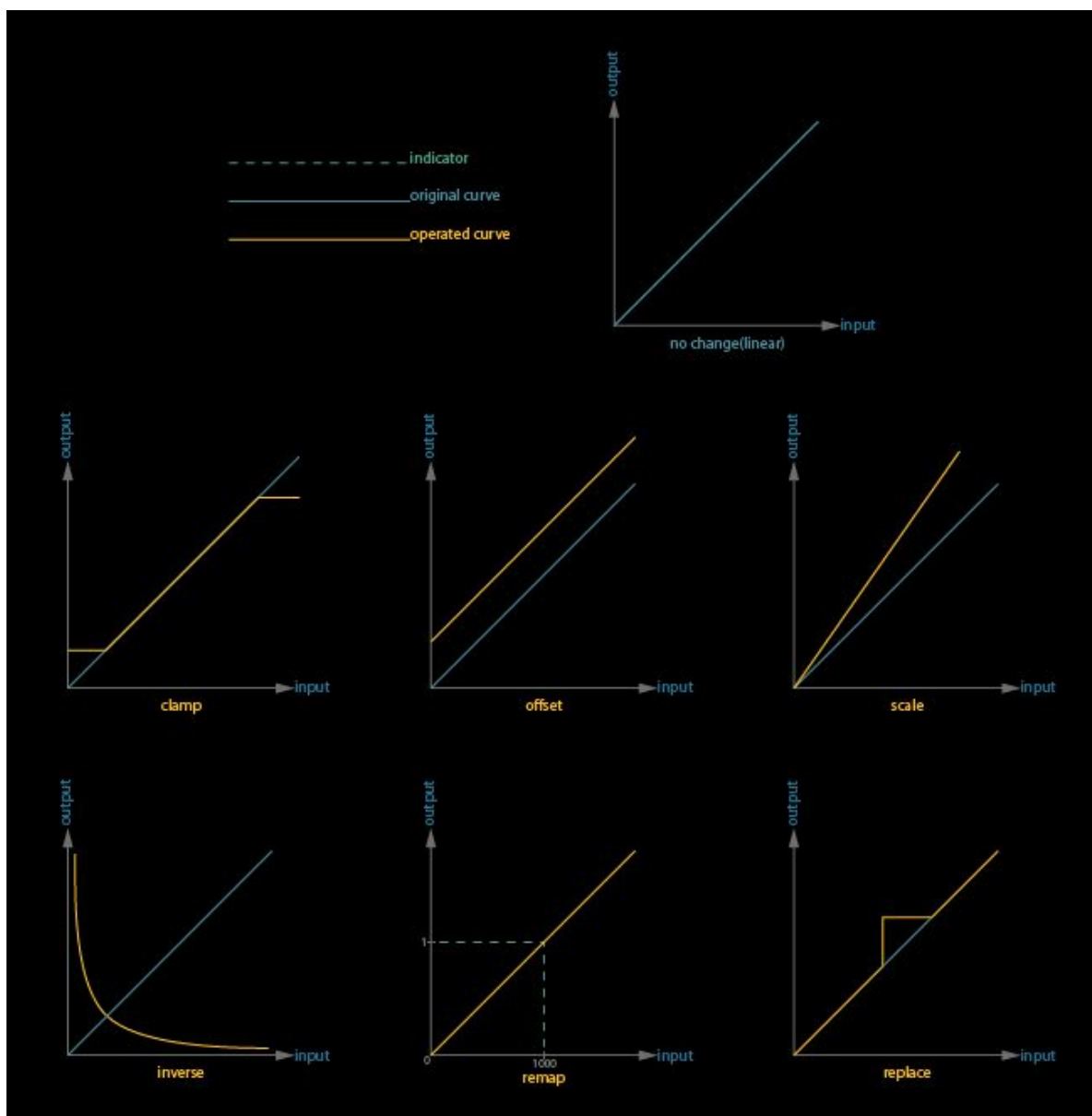


Figure 3.1D curves of different microOps in depthOp



3.1.4 Example

The example shows the workflow of how to convert an unnormalised depth map to a *normalised* one.

1. This is an unnormalised depth map, the depth value is far beyond 0-1 (Figure 3.1E).



Figure 3.1E depth map

2. To normalise the depth, firstly connect a **depthOp**, choose the depth channel to be converted (Figure 3.1F).

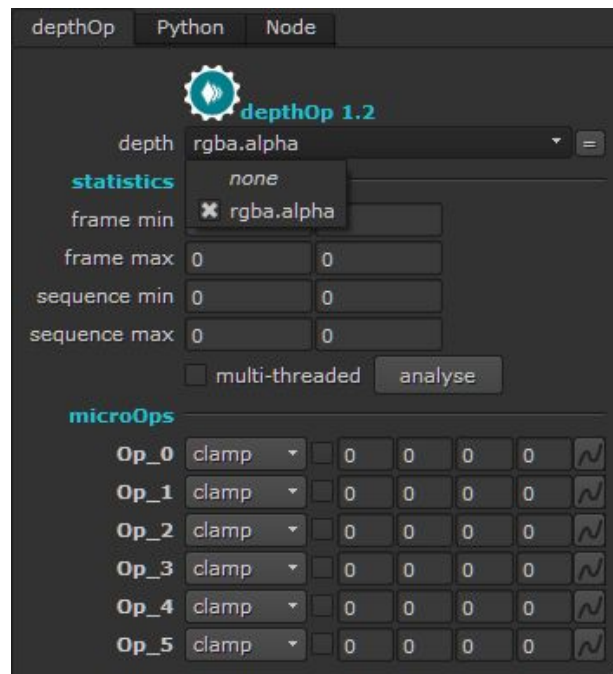


Figure 3.1F select the correct channel

3. Analyse the image/sequence to find the statistics of the depth values(**Figure 3.1G**)

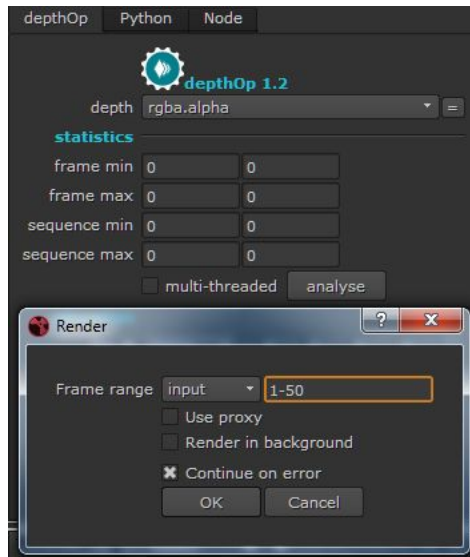


Figure 3.1G input the frame range for analysis

4. 10.0388078 and 16.1382246 are found as the *second minimum* and *second maximum* depth value of the pass. So, to normalise these to 0 - 1, we choose *remap* in microOps section(**Figure 3.1H**), and key in 10 and 17 for the remap range(1st, 2nd column), and 0 - 1 for the target output range(3rd and 4th column).

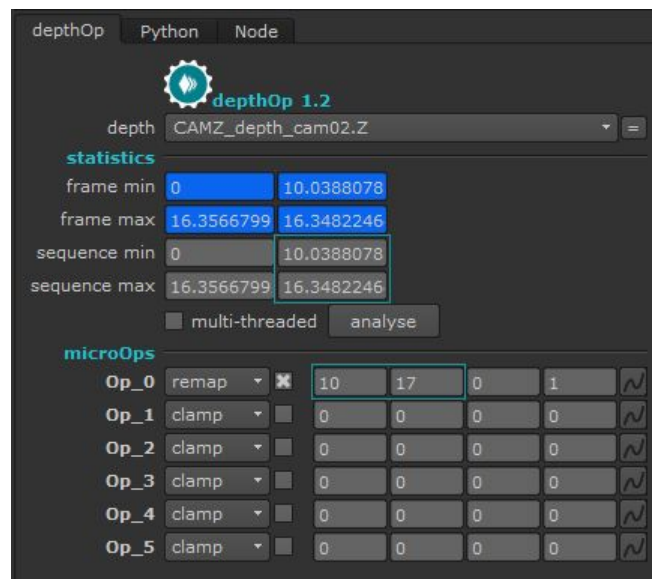


Figure 3.1H input the parameters of remap according to analysed data



5. A remapped depth map is resulted(**Figure 3.1I**)

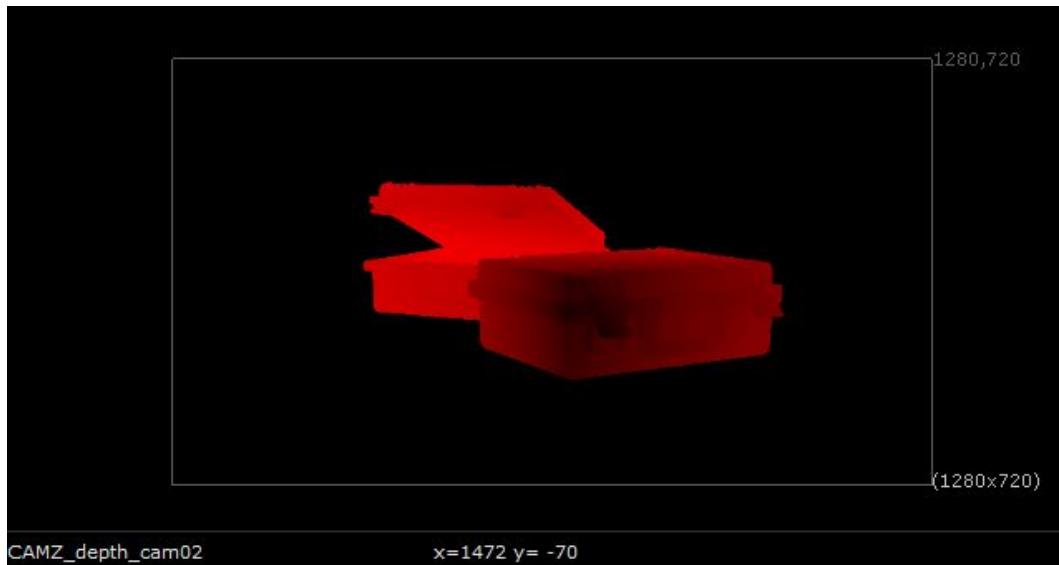


Figure 3.1I remapped depth map

6. But in blank area, the value is outside 0-1 range(**Figure 3.1J**)

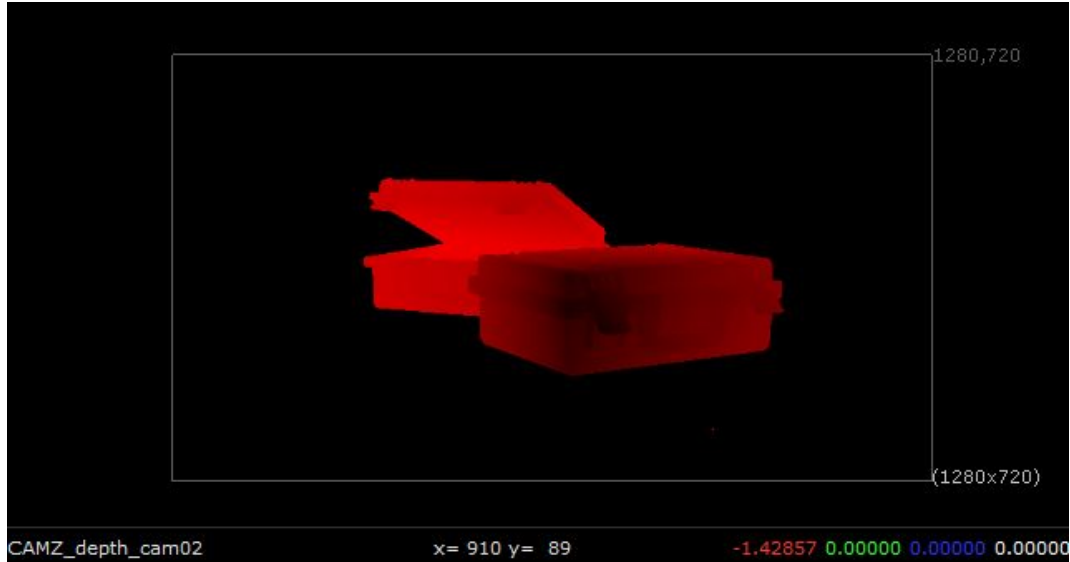


Figure 3.1J unclamped depth map



7. So *clamp* should be chosen, and key in 0 - 1 for the clamp range (1st, 2nd column). Any numbers bigger than 1 would be clamped to 1, any lower than 0 would be 0(**Figure 3.1K**)

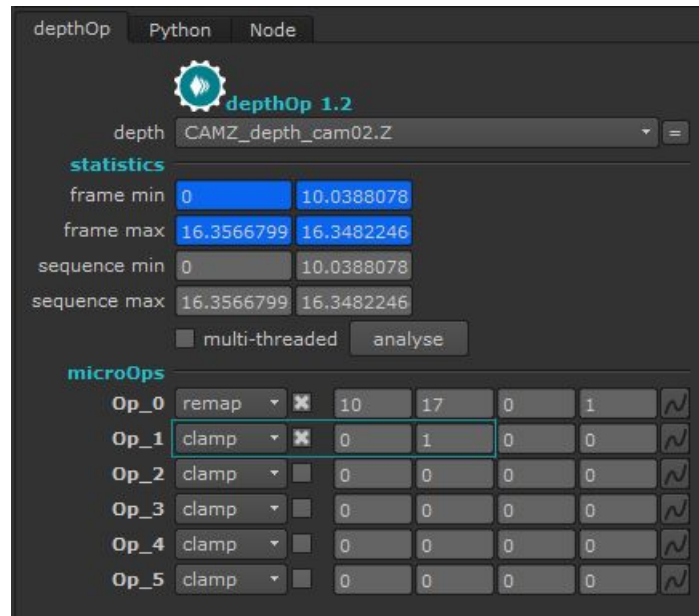


Figure 3.1K clamp the depth map

8. The depth values are now *normalised* (**Figure 3.1L**).

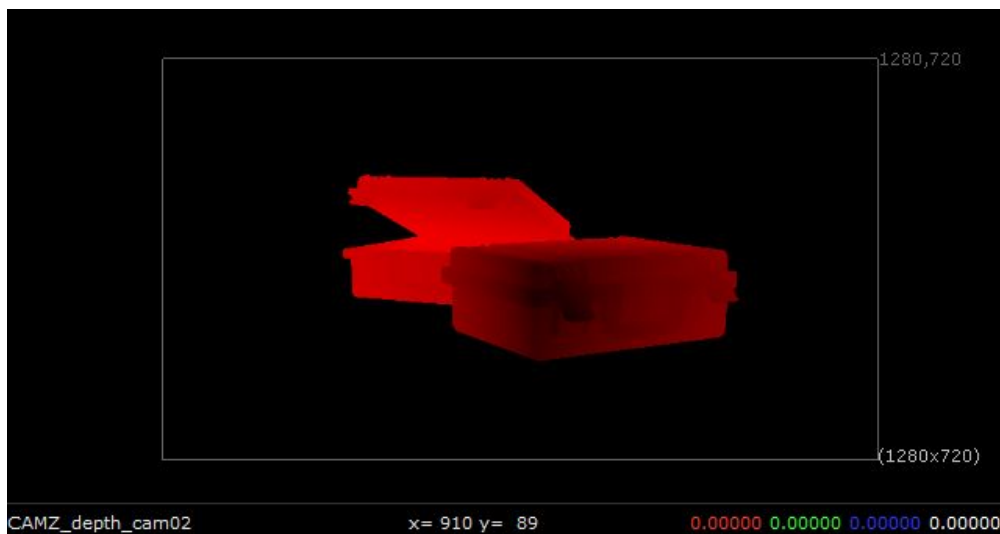


Figure 3.1L normalised depth map



3.2 normalOp

3.2.1 introduction

normalOp is a plug-in node specifically designed for 3D normal vector processing, it allows users to apply transformation based on transformation matrices found in the metadata from upstream or from a custom user supplied matrix. This version of **normalOp** also includes a smoothing function which averages each pixel's normal vector with the neighborhood. Similar to **depthOp**, **normalOp** also has 2 microOps for convenient day-to-day quick conversion.



Figure 3.2A converted normal vector from object space(left) to camera space(right)

3.2.2 input

normalOp only takes one input, normal map.

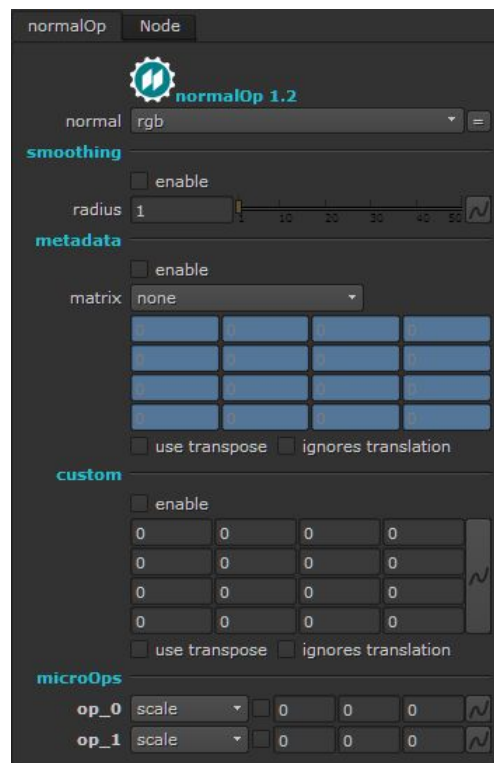


Image 3.2B control panel of **normalOp**



3.2.3 parameters

normalOp has four sub-panels, **smoothing**, **metaData**, **custom** and **microOps** :

smoothing

Each pixel's normal vector will be averaged by its neighborhood, the area of the neighborhood is controlled by the *radius* value.

metadata

- *enable* if checked, applies the following matrix data to the Deep Normal data.
- *matrix normalOp* will try to locate if there's any transformation matrices found from the upstream metadata, if yes, they will be listed in the pull-down menu and users may select the one they need.
- *use transpose* - apply the transformation with the matrix transposed.
- *ignores translation* - make certain translation component will not be applied to the normal data.

custom

Users may further apply a custom transformation if needed.

microOps

This version of **normalOp** offers two different microOps, and users may sequence the microOps according to their needs, the operations always start from the top (i.e.op_0) and proceed downwards; each of them could be enabled or disabled independently by the checkbox on each row.

- *scale* - A scalar to multiply the current normal map, normal axis can be flipped by inputting *-1*.
- *normalise* - Length of the normals are expected to be *1*, otherwise there may be artifacts on the shaded surface, check to normalise the map.

microOps	opParameter_0	opParameter_1	opParameter_2
scale :	X	Y	Z
normalise :	/	/	/

Table A - microOps and their parameters

3.2.4 example

This example shows the workflow of how to convert a normal map of *object space* to a new one of *camera space*.

1. This is a normal map in object space(**Figure 3.2C**).



Figure 3.2C normal map in object space



2. In this case , the normal map has a *worldToCamera matrix* channel in the metadata, which is available for conversion to *camera space* (Figure 3.2D).

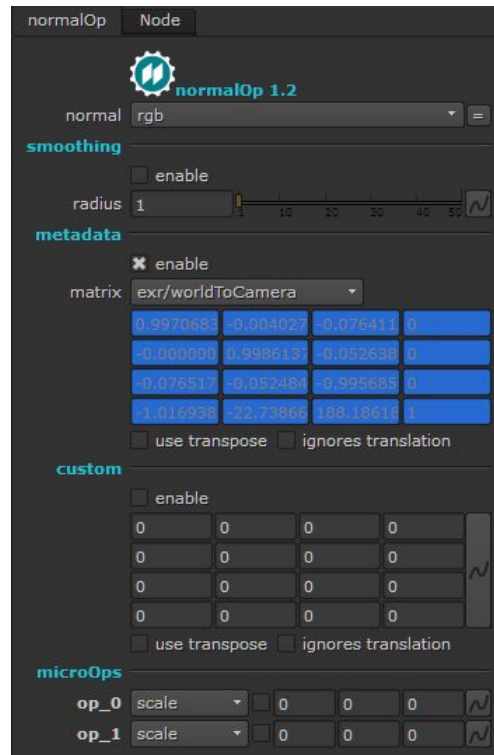


Figure 3.2D draw world-to-camera matrix data from the .exr.

3. This is the converted normal map but the map's z axis is pointing away (there is no bluish color)(Figure 3.2E).

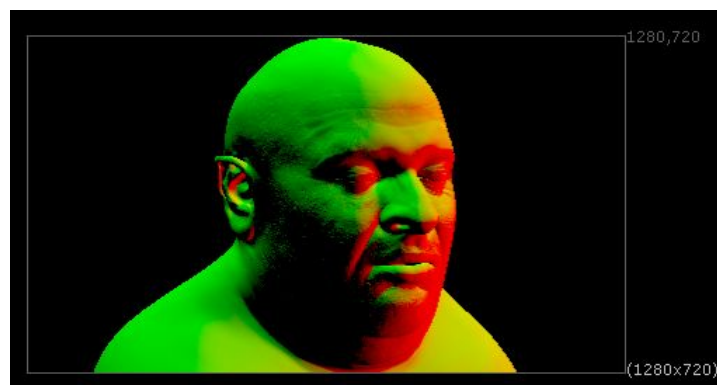


Figure 3.2E converted normal map, but z axis is pointing away.



4. So we need to flip the z-axis by multiplying -1 to the Z, by using the *scale* microOp (Figure 3.2F)

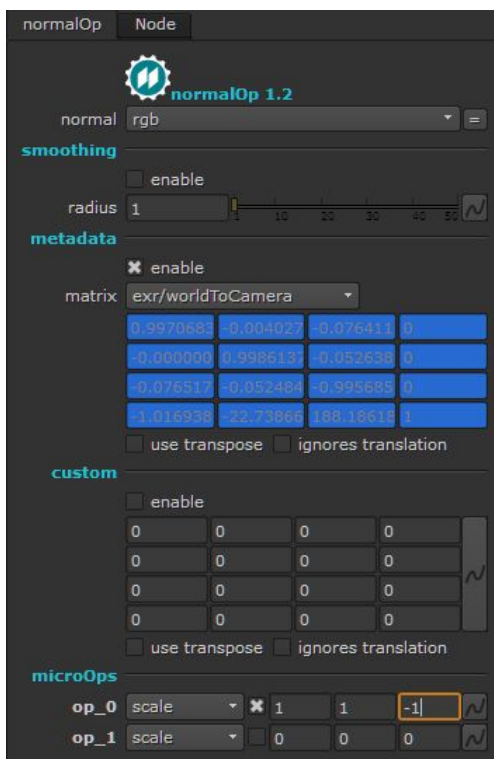


Figure 3.2F use the microOp scale to flip the z-axis

5. Normal vectors are now remapped to *camera space* (Figure 3.2G).



Figure 3.2G screen space normal map



3.3 aaOp

3.3.1 introduction

As both depth and normal based image manipulations in escher rely on non anti-aliased (or unfiltered) inputs, **aaOp** is a plug-in designed to apply anti-aliasing to these processed images based on a morphological approach.



Image 3.3A rendered output of aaOp(right)

3.3.2 quick start

1. Create an **aaOp** node from escher.
2. Connect the aliased image/sequence to **aaOp**.
3. Choose the channel set to be anti-aliased.
4. Choose either RGB or luminance mode.
5. Tweak the thresholds to achieve the desired anti-aliasing level.

3.3.3 parameters

- **input** Choose the channel set to be anti-aliased.
- **mode** There are two modes for **antialiasing**, *RGB* and *luminance*.
 - RGB(default)* - Allows threshold to be set for individual color channels.
 - luminance* - Uses overall luminance level for antialiasing.
- **threshold_0** This value is used in Luminance mode or for the RED channel in RGB mode.
- **threshold_1** This value is used for the GREEN channel in RGB mode.

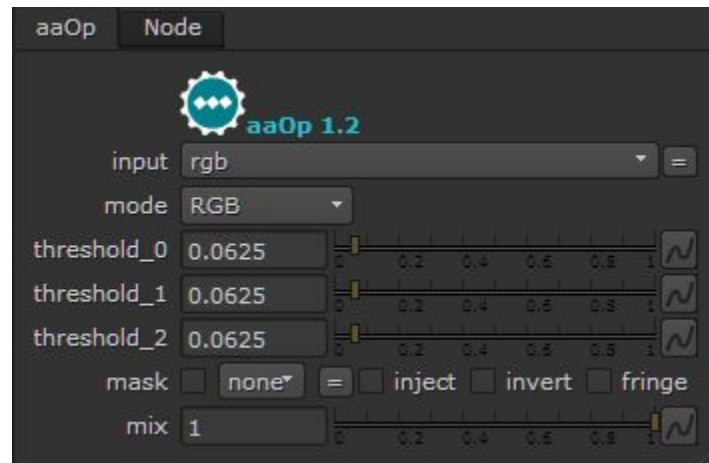


Figure 3.3B control panel of aaOp

- **threshold_2** This value is used for the BLUE channel in RGB mode.

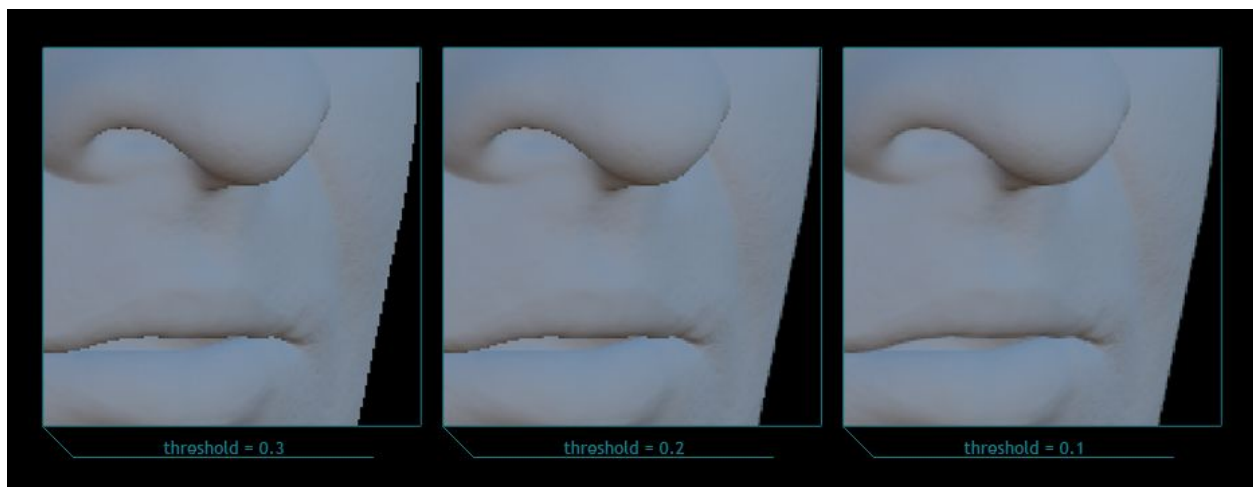


Figure 3.3C comparison of different threshold values (in luminance mode)



3.4 SHOp

3.4.1 introduction

Spherical Harmonics (SH) transform is often used in CGI pipeline for environment light information compression and efficient lambertian shading purposes, **SHOp** is a pipeline utility plug-in which allows users to compute SH coefficients of a given lightprobe input right inside Nuke. **SHOp** outputs the SH coefficients into the metaData channel such that existing SH-based tools may reference them easily. In addition, users may consider to embed these SH coefficients into their OpenEXR outputs such that no further SH transform is necessary and the new lightprobe OpenEXR images have the SH information right inside them.

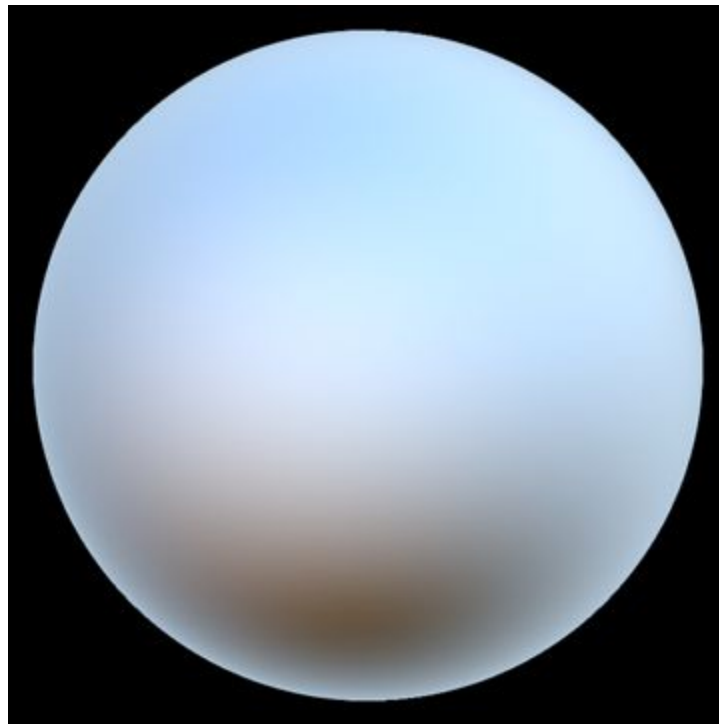


Image 3.4A irradiance environment map rendered from **SHOp**

3.4.2 quick start

1. Create an **SHOp** node from escher.
2. Read a light probe and connect it to **SHOp**.
3. Identify the light probe type.
4. Set the output, either **irradiance environment map** or **pass through**.
5. Set how many SH bands are desired.
6. Connect a SHader node for the shading or
7. Connect a write node to output a new light probe with coefficients metadata written.
8. Users may examine the SH coefficients in the 2nd tabs.



3.4.3 input

SHOp only takes one input, a light probe. The light probe can be in *mirror ball*, *angular* or *latlong* format.

3.4.4 parameters

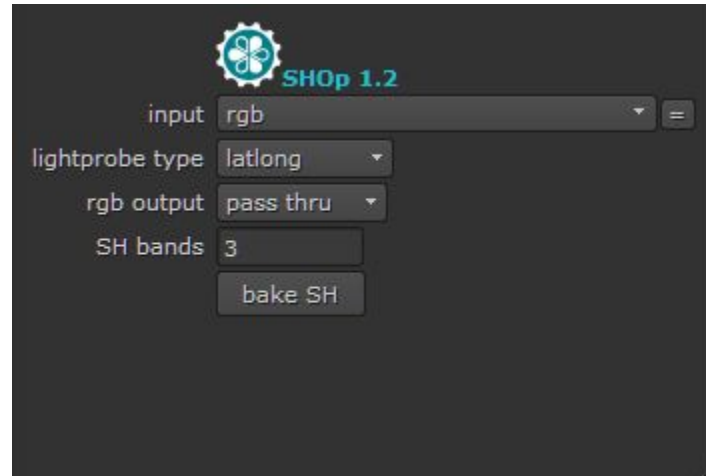


Figure 3.4B control panel of **SHOp**

SHOp tab

- **input** Choose the channel of the connected lightprobe.
- **lightprobe type** Choose the correct light probe type for the input.
 - *mirror ball*
 - *angular*
 - *lat long*
- **rgb output** Output method of the calculated lightprobe(**Figure 3.4C**)
 - *pass through* (default) - No rendering, simply inserts the SH coefficients into metadata.
 - *irradiance* - renders an irradiance environment map.
- **SH bands**

Number of SH bands to project(**Figure 3.4D**). In most cases, 3 bands should be enough for lambertian shading purpose.
- **bake SH** Bake SH coefficients into metaData, so SHOp can pass SH coefficients without an input of lightprobe; baked SH coefficients can also be embedded as metaData in an EXR output.

SH coefficients tab

Bands of SH coefficients are listed here(**Figure 3.4E**)

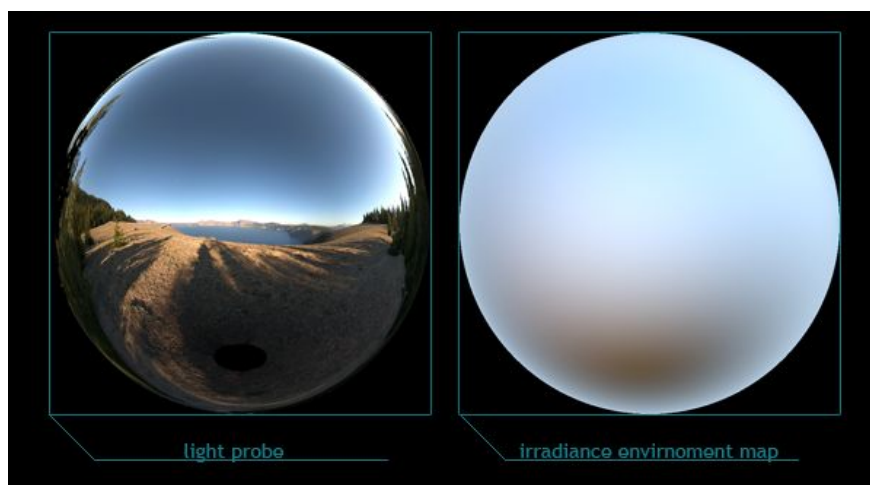


Figure 3.4C a light probe and an irradiance environment map generated by SHOp

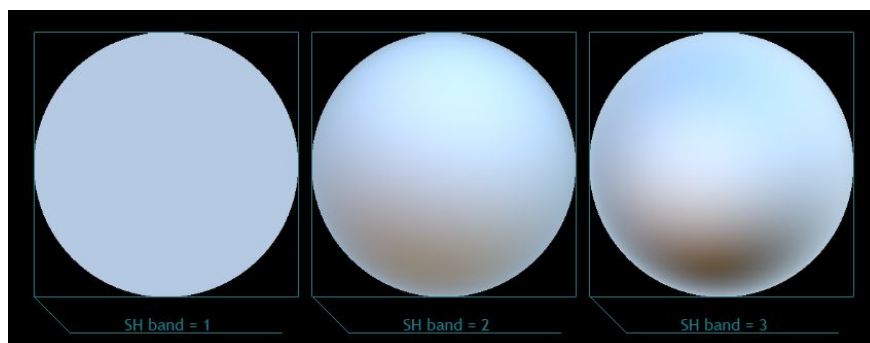


Figure 3.4D SH bands comparison

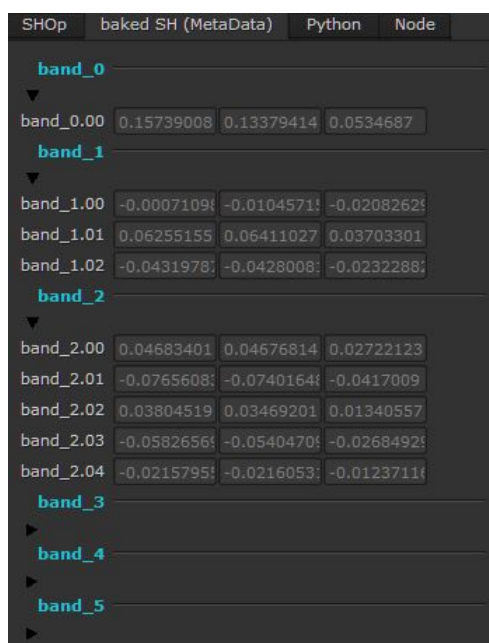


Figure 3.4E SH coefficients tab

2.5D SHADERS

4.1 SHader

4.1.1 introduction

SHader is a 2.5D shading plug-in driven by SH and is also considered as a partner plug-in of **SHOp**. It is designed to use the Spherical Harmonics (SH) coefficients found in metadata from upstream to apply Lambertian shading to a normal map. Apart from using computed SH coefficients from upstream, users may also input their own SH coefficients manually for creative shading purposes.



Image 4.1A rendered output of **SHader**

4.1.2 quick start

1. Create an **SHOp** node from escher.
2. Read a light probe and connect to **SHOp**.

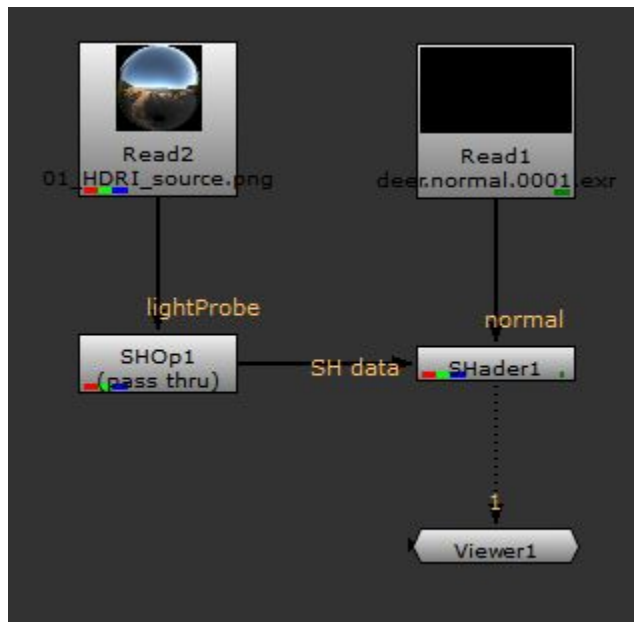


Figure 4.1B node graph of SHOp and SHader

3. Set the parameters as needed in **SHOp**.
4. Create a **SHader** node from escher.
5. Connect the **SHOp** to the input, *SH data*, of **SHader**.
6. Read an normal map and connect to the input, *normal*, of **SHader**.
7. Select the channel in which the normal map locates.
8. Choose the SH coefficient source, from *SHOP* directly, baked *metaData* of *SHOP*, or *maually* input.
9. To manually input SH coefficients, toggle to the SH coefficient tab and tune the values, each row represents the SH coefficients of Red, Green and Blue.
10. Tune the material setting as desired.

4.1.3 input

SHader takes two inputs, normal and SH data.

normal - Normal map used for **SHader** should be unfiltered or non anti-aliased. The normal map can also be pre-processed with escher's *normalOp* if desired.

SH data - This input is optional. SH data could come from escher **SHOp**, it provides **SHader** with SH coefficients to approximate the lighting. If SH data source is set manual, this input can be left unplugged.

4.1.4 parameters

SHader tab

SHader tab has 2 sub-panels, **SH Data** and **material**.

normal - Choose the correct channel set where the normal map locates.

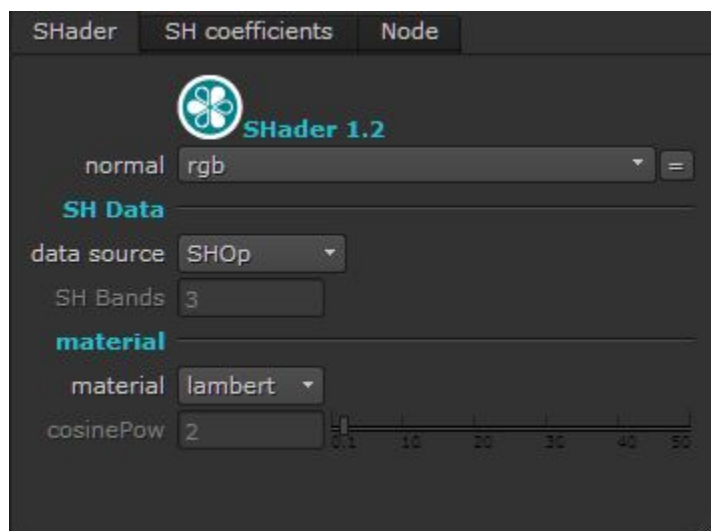


Figure 4.1C control panel of SHader

SH Data

- **data source** *SHOp(default)* - Pull SH coefficients directly from SHOp with a lightprobe input.
metaData - Pull the baked SH coefficients from the input(SHOp or exr)'s metaData.
manual - Input the SH coefficients manually in the SH coefficients tab.
- **SH bands** Number of SH bands used in shading calculations(**Figure 4.1D**).
P.S. When data source is from metaData, the maximum number of SH bands is determined by its input.

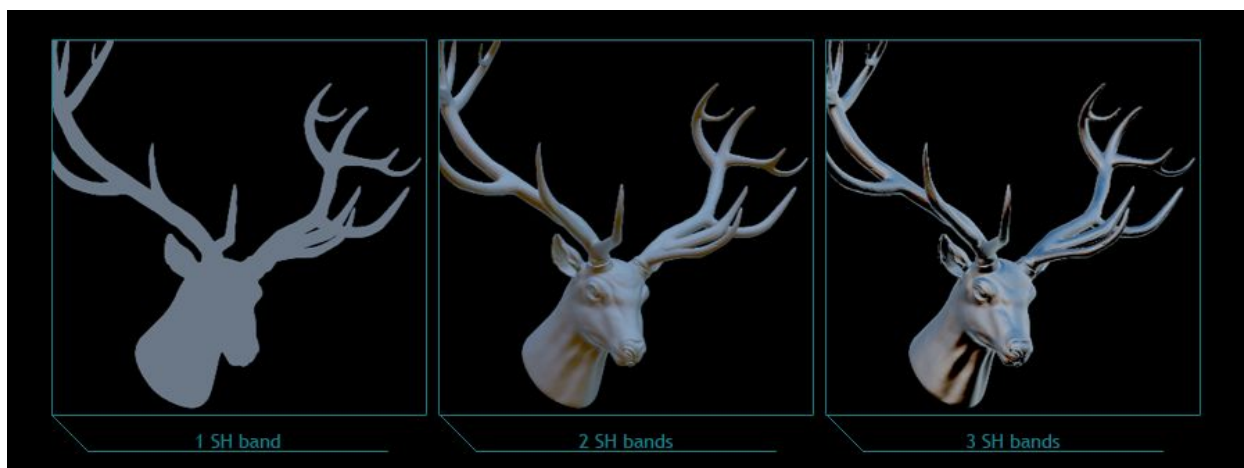


Figure 4.1D SH bands comparison (SH bands in 'phong' material)

material

- **material**
lambert(default) - Lambertian shading model.



phong - Phong shading model (users should not expect a good-looking output from it due to the low-frequency nature of the SH approach).



Figure 4.1E lambert and phong comparison

- **cosinePow** (for phong only) The cosine power factor.

SH coefficients tab

Users may examine or input the SH coefficients here. Each SH band contains (2*SH bands - 1) coefficients, and each coefficient could be regarded as a light (Figure 4.1F).

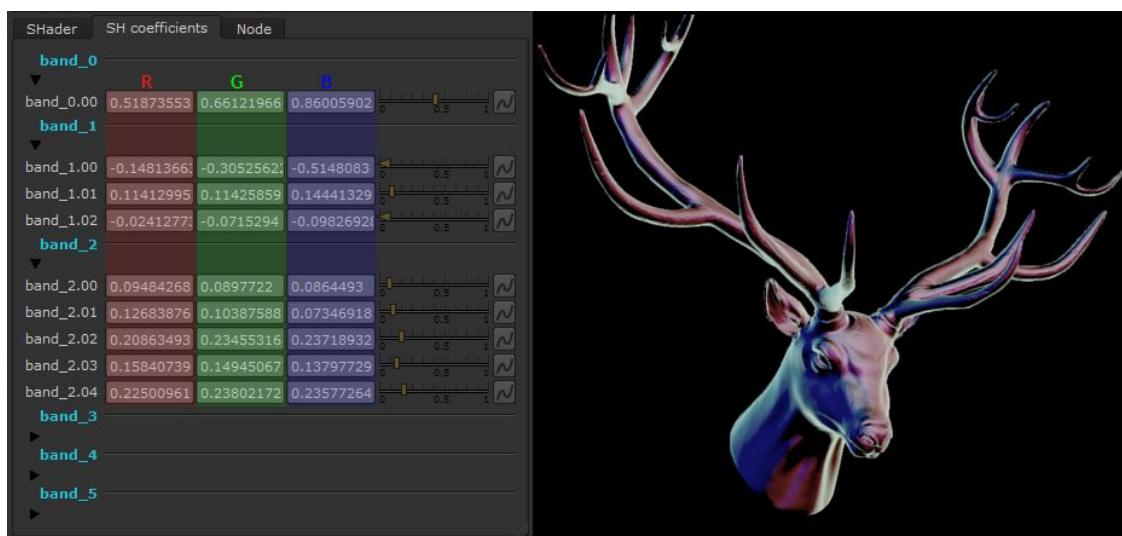


Figure 4.1F SH coefficients tab with manually input SH coefficients



4.2 glossy

4.2.1 introduction

glossy is 2.5D shading plug-in based on image based lighting and generic glossy shading model with importance sampling and pre-filtered samples.



Image 4.2A rendered output of **glossy**

4.2.2 quick start

1. Create a **glossy** node from escher.
2. Read a light probe image.
3. Read a normal map.
4. Connect the environment map to the lightprobe input of **glossy**.
5. Connect the normal to the normal input of **glossy**.
6. Choose the appropriate channel set for the light probe and normal map.

4.2.3 input

glossy takes two inputs, light probe and normal.

light probe - **glossy** assumes the light probe should be in mirror ball, angular or lat long format.

normal - Normal map used for **glossy** should be unfiltered or non anti-aliased. The normal map can also be pre-processed with escher's *normalOp* if desired.

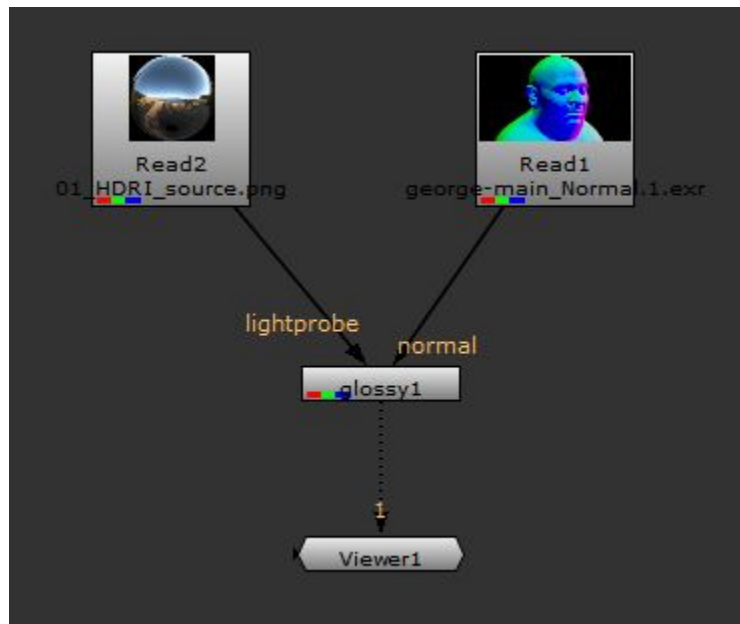


Figure 4.2B node graph of glossy

4.2.4 parameters

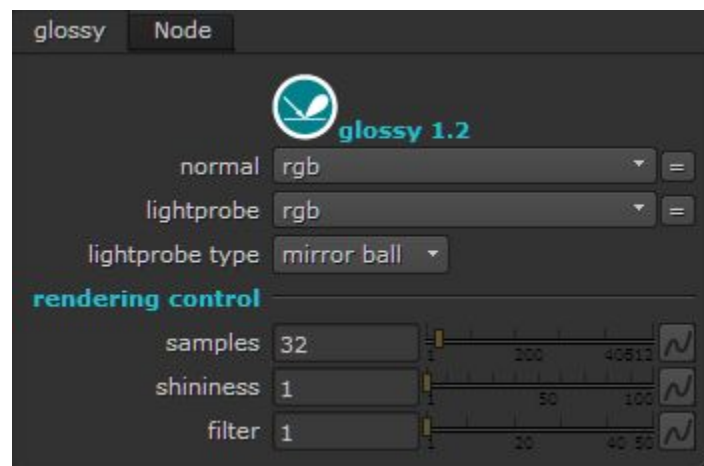


Figure 4.2C control panel of glossy

- **normal** Choose the appropriate channel set of the normal map.
- **lightprobe** Choose the appropriate channel set of the light probe.
- **lightprobe type** Select the correct light probe type for the input.
 - *mirror ball*(default)
 - *angular*
 - *lat long*

rendering control

- **samples** Number of samples for each shading(Figure 4.2D).

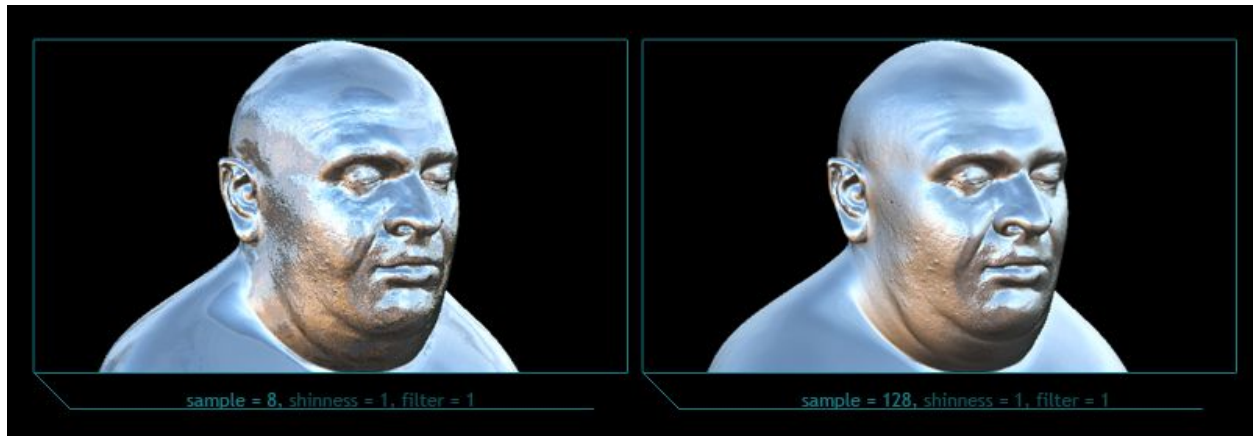


Figure 4.2D comparison of samples

- **shininess** Glossiness of the shading(Figure 4.2E).

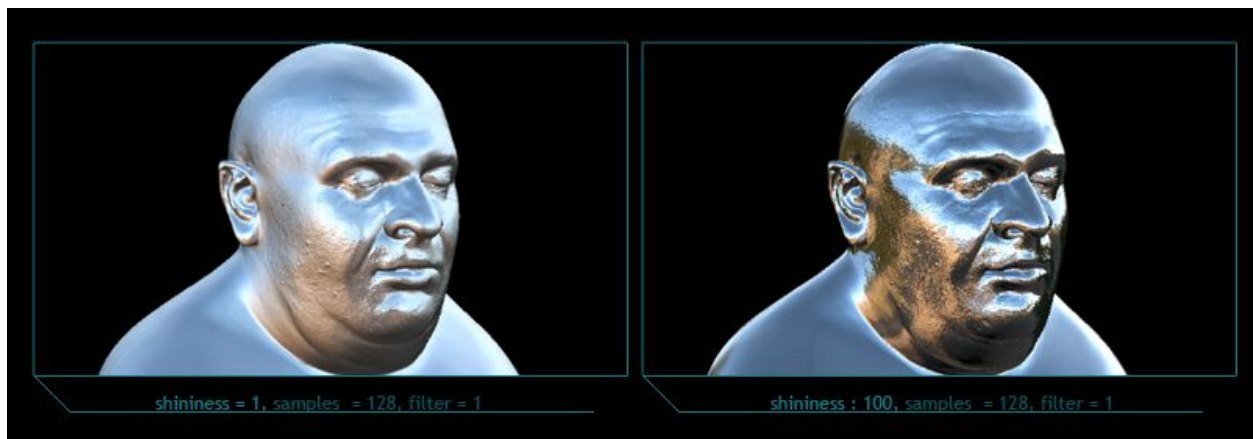


Figure 4.2E comparison of shininess

- **filter** Maximum amount of filtering applied to each sample.



4.3 depthTrooper

4.3.1 introduction

depthTrooper plug-in uses depth gradient information derived from the input depth map to create ambient occlusion alike shading at interactive speed. As **depthTrooper** uses a purely image based approach, it is in no way comparable to the quality of 3D ray-traced ambient occlusion but this plug-in should bring new creative possibility to compers.



Image 4.3A rendered output of **depthTrooper**

4.3.2 quick start

1. Create a **depthTrooper** node from escher.
2. Read the depth map and connect it to **depthTrooper**.
3. In the control panel, choose the channel in which the depth map locates.
4. Clip the depth range by inputting *min* and *max*.
5. Adjust other parameters to tune the occlusion.

4.3.3 input

depthTrooper only takes one input, depth map.

Depth map used for **depthTrooper** should be normalised, and unfiltered or non anti-aliased. The depth map can also be pre-processed with escher's *depthOp* if desired.

P.S. **depthTrooper** is designed to work for a normalised depth range [0 - 1]. 0 is the closest to camera, 1 is the most distant.

4.3.4 parameters

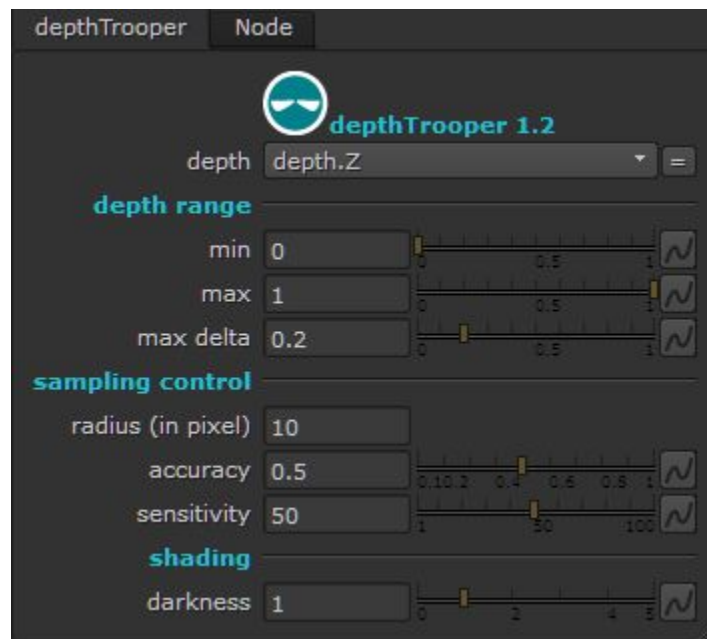


Figure 4.3B control panel of **depthTrooper**

- **depth** - Choose the appropriate channel of the depth map from the input.

depth range

- **min and max** These 2 values define the global depth range to be included for processing.
- **max delta** As **depthTrooper** compares one pixel's depth value with its neighbors, users may sometimes want to exclude neighbors which have big depth difference. In short, this value controls the local depth range included for processing for each pixel(Figure 4.3C).

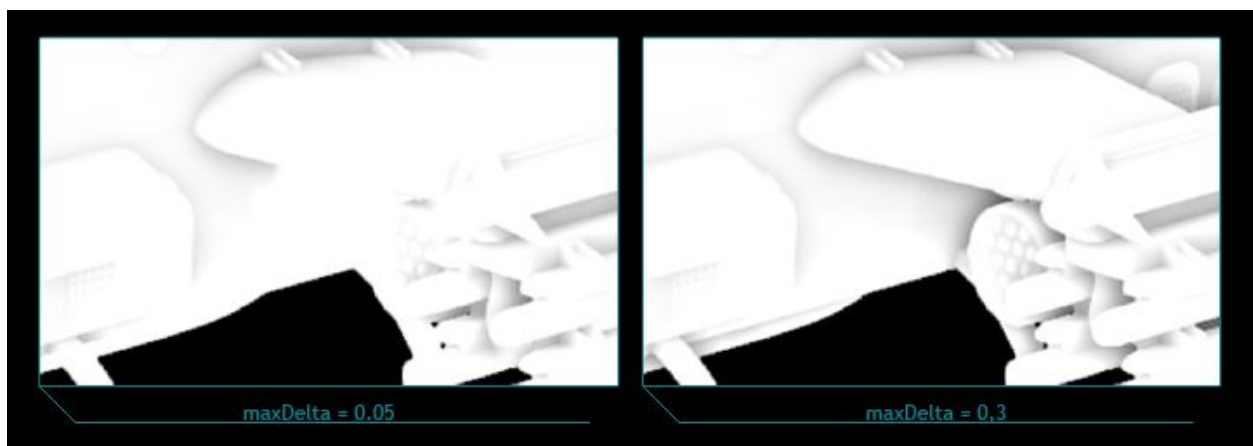


Figure 4.3C comparison between different values of *max delta*

sampling control

- **radius(in pixel)** This value defines the amount of neighborhood pixels to be included for occlusion shading. As **depthTrooper** is purely image based, so the area is always measured in image space(Figure 4.3F).

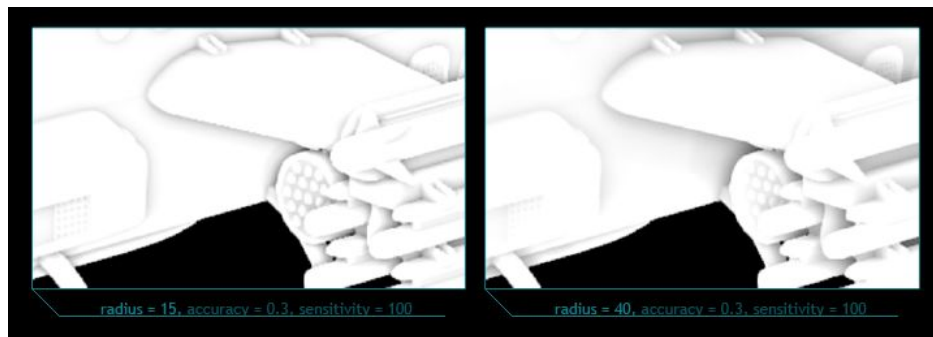


Figure 4.3D comparison between different values of *radius*

- **accuracy** **depthTrooper** relies on this value to determine its sampling strategy. When this parameter is set at a low value, the plug-in fully samples closer pixels and sparsely samples those distant ones. With a value of 1, **depthTrooper** will sample all pixels in the neighborhood(Figure 4.3E).

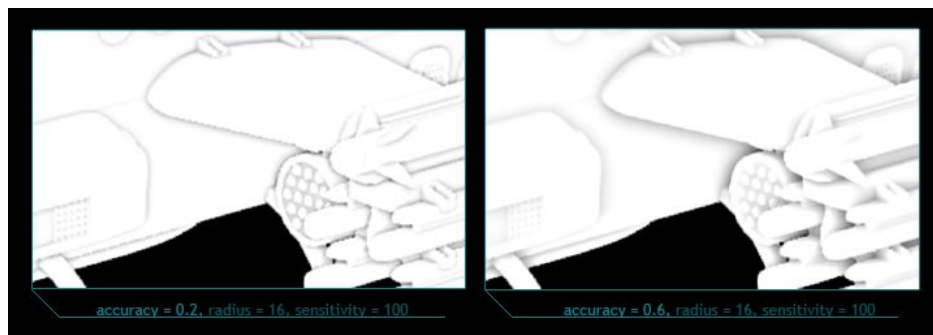


Figure 4.3E comparison between different values of *accuracy*

- **sensitivity** This value practically controls the details of the shading, a low value instructs **depthTrooper** to shade only if an obvious depth difference is observed from its neighborhood and a high value instructs the plug-in to shade even when a minimal depth difference is detected from its neighborhood(Figure 4.3F).

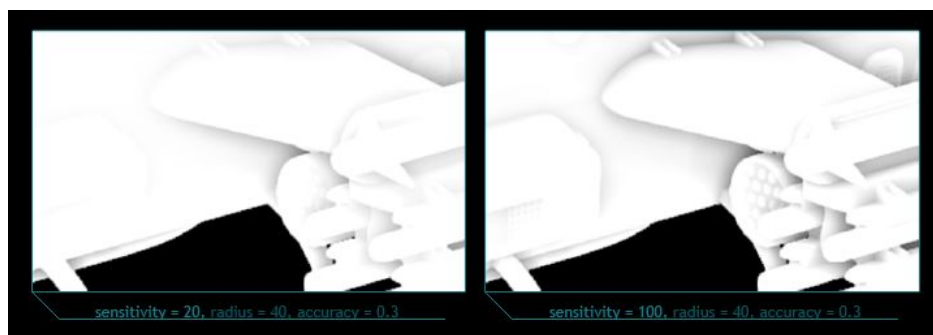


Figure 4.3F comparison between different values of *sensitivity*

shading

- **darkness** Controls the magnitude of the achromatic shading.

4.4 depthVader

4.4.1 introduction

depthVader plug-in is a multi-purpose plug-in designed to explore creative possibilities with 2.5D image making. It offers a gradient blur function which helps compers to fake subsurface scattering shading using depth gradient information. **depthVader** also produces other image data derived from the depth information, they include depth gradient and normal vectors. We have also designed a very user-friendly user interface widget called depth sampler which helps compers to sample depth data information while keeping creative focus on the main composite. Depth samplers could also be driven by Nuke's tracker via expression or to be used with other depth-related software for convenient depth information collection.



Image 4.4A rendered output of depthVader

4.4.2 quick start

1. Create a **depthVader** node from escher.
2. Read the depth map and connect it to the *depth* input of **depthVader**.
3. Read the colour map and connect it to the *rgb* input **depthVader**.
4. Select the channel where your depth map and colour map locate.
5. Choose the mode for **depthVader**.
6. To add gradient blur, select *gradient blur* mode and tune parameters in *blur control*.
7. To recognize the depth range in effect, select *depth range guide* mode.
8. To make a slice out of the depth, select *depth slice mode*.



9. To convert the depth map to a normal map, select *normal map* mode.
10. To view the incoming depth, select *depth* mode.
11. To view the depth gradient, select the *depth gradient* mode and toggle R,G,B channel to view different mode of the depth gradient.

4.4.3 input

depthVader takes two inputs, depth and rgb.

- **depth** Depth used for **depthVader** should be normalised, and unfiltered or no anti-aliased. The depth map can also be pre-processed with escher's *depthOp* if desired.
- **rgb** A colour map for the **depthVader** to take place.

4.4.4 parameters

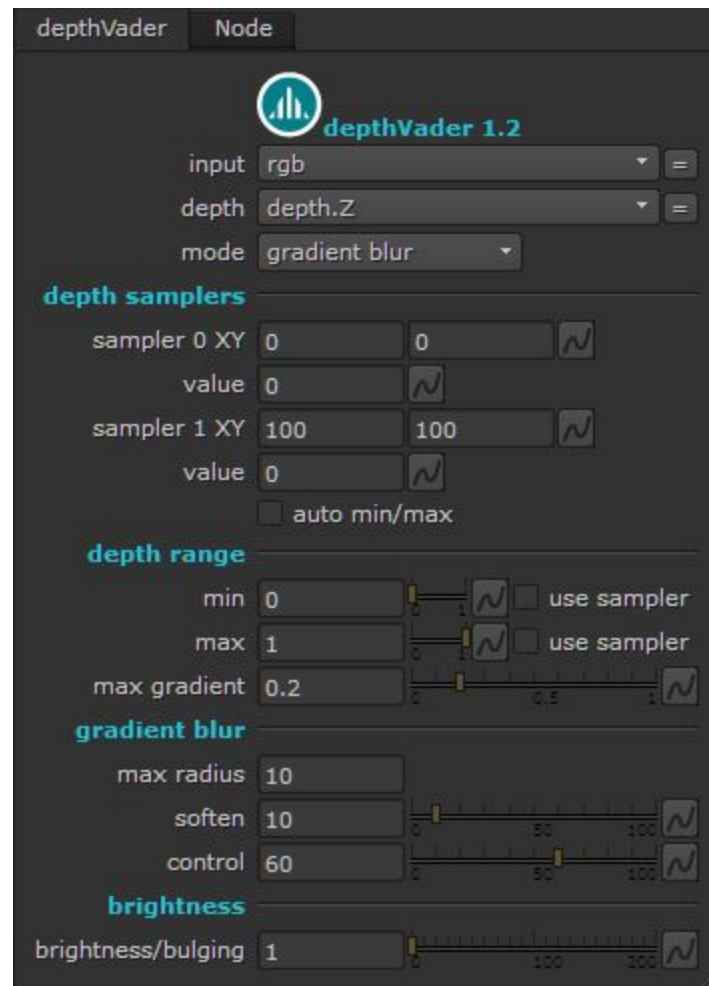


Figure 4.4B control panel of **depthVader**

- input - Choose the appropriate channel set of the colour map from *rgb* input.
- depth - Choose the appropriate channel of the depth map from *depth* input.



- **mode** - *gradient blur* - A fake sub-surface scattering effect.
 - depth range guide* - Visualises the area in effect, greyed region indicates non computed area.
 - depth slice* - Allows users select a slice out of the depth input.
 - normal map* - Outputs a normal map based on depth gradient information.
 - incoming depth* - Views the incoming depth map.
 - incoming depth gradient* - Views the gradient information of the incoming depth map, the brighter the area, the greater the gradient.

RED channel : depth gradient in X-direction, GREEN channel : depth gradient in Y-direction, BLUE : overall depth gradient.

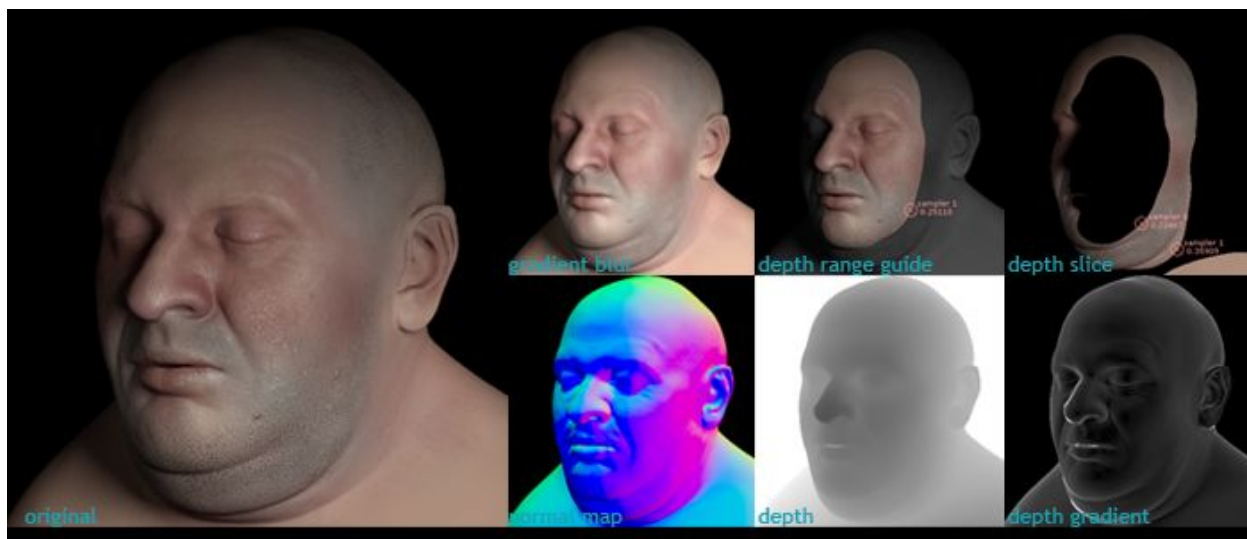


Figure 4.4C different modes of **depthVader**

depth samplers

depth range

- **min and max** These 2 values define the global depth range to be included for processing.
 - if *use sampler* on *min* is checked on, it takes values from *sampler 0* or the minimum between two samplers if *auto min/max* is on.
 - if *use sampler* on *max* is checked on, it takes values from *sampler 1* or the maximum between two samplers if *auto min/max* is on.
- **max gradient (applicable to *gradient blur* mode only)** As **depthVader** compares one pixel's depth value with its neighbors, users may sometimes want to exclude neighbors which have big depth difference. In short, this value controls the local depth range included for processing for each pixel.

blur control

- **max radius** This value defines the amount of neighborhood pixels to be included for blurring. It is measured in pixels(Figure 4.4E).
- **soften** Softness of the blur(Figure 4.4F).
- **control** This parameter suppresses the blur strength according to the depth gradient value of the current pixel. In short, areas which are flat to the camera would be blurred more while areas with strong gradient would receive less blurring(Figure 4.4G).



Figure 4.4D depth samplers

The samplers evaluate the depth value of the current pixel without switching the Viewer's input to the depth channel(Figure 3.D).

- *XY* is the coordinates of the samplers.
- *value* reflects the depth value of the current pixel.

auto min/max - Without this option checked, *min* of depth range will always receive from sampler 0 and *max* will always receive from sampler 1. It could happen that depth value obtained by sampler 1 is smaller than the one by sampler 0, so it defeats the samplers' purpose. This *auto min/max* option makes certain that the correct min-max range is sent.



Figure 4.4E different values of max radius



Figure 4.4F different values of soften



Figure 4.4G different values of control



- **brightness** (applicable to *gradient blur*, *normal map*, *depth* and *depth gradient* mode)
 - Apart from *normal map* output, it simply adjusts the brightness of image
 - for *normal map*, it has the effect of enhancing the features of the normal vectors output.

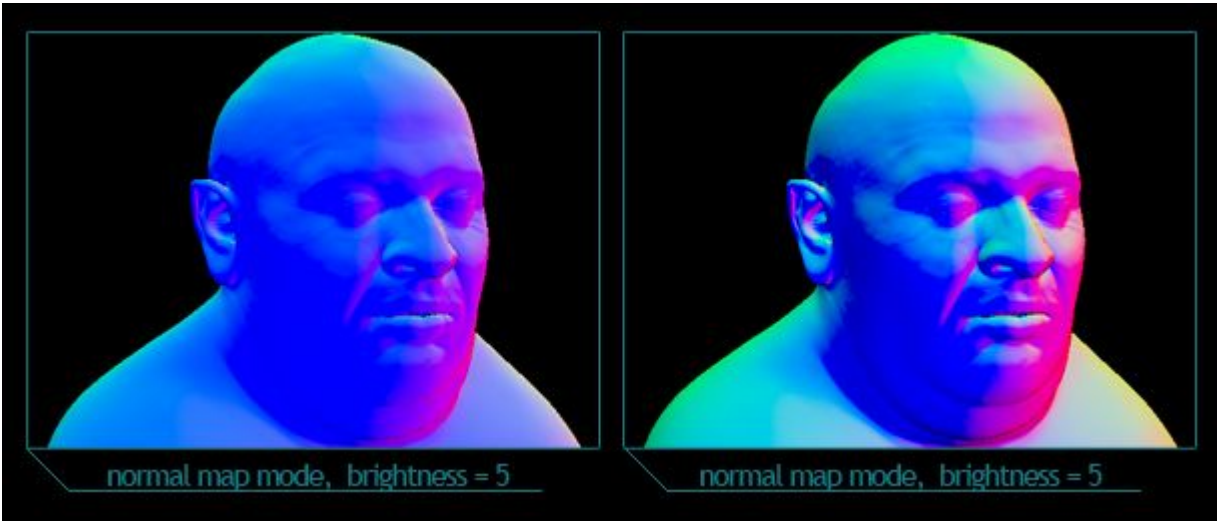


Figure 4.4H different values of brightness in normal map mode

5.1 Deep Normal Maps

5.1.1 introduction

Escher's 2.5D Deep shaders take Deep Normal Maps as their inputs. Deep Normal Maps are similar to regular Deep RGB images where each pixel stores multiple samples of normal vector information (**figure 5.1A**), these additional samples automatically enable well anti-aliased 2.5D rendering which is not possible with regular normal maps. Today's most modern renderers support deep images output well and in this chapter we have outlined brief information of how to correctly render Deep Normal passes inside Maya with **RenderMan/RIS**, **Arnold** and **vRay**.

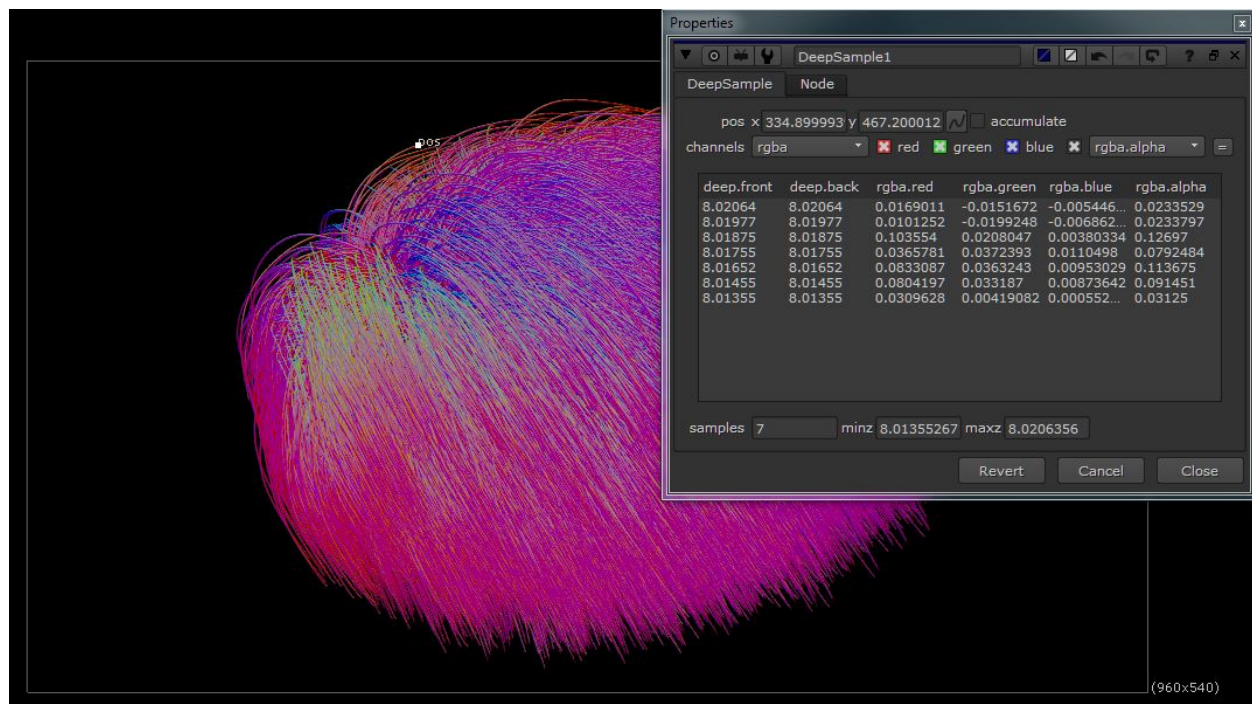


Image 5.1A Deep Normal Map rendered using Pixar's RenderMan/RIS



5.1.2 deep normal output from *RenderMan for Maya*

Pixar's RenderMan for Maya has robust support of deep images rendering and it outputs them using Pixar's .dtex format. Nuke's DeepReader node supports .dtex directly but it relies on Pixar's software library component in order to decode .dtex images, so please consult Nuke User Guide for the details. We used the following software version for this document:

- Maya 2015 ext. 1 + SP5
- RenderMan RMS 19.0 (build 1417965)

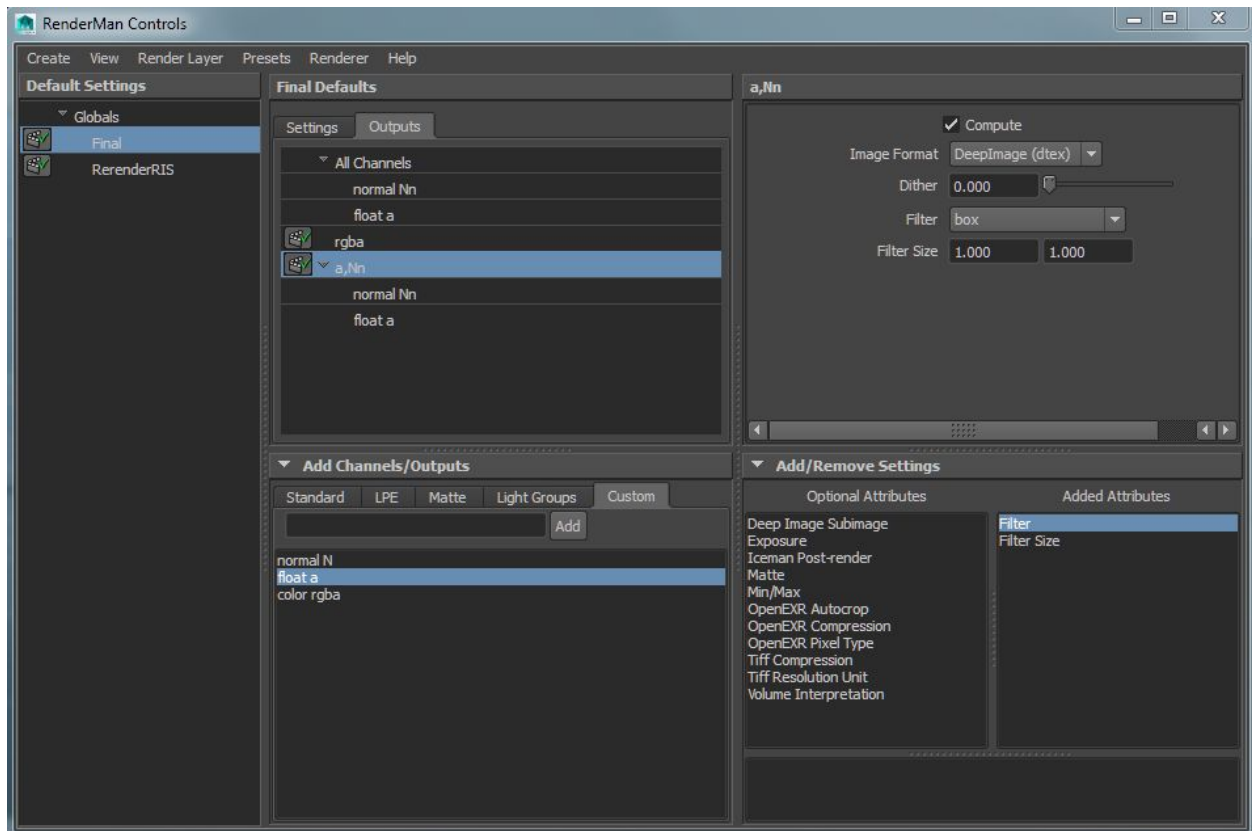


Figure 5.1B RenderMan Advanced Controls in Maya

5.1.3 adding Deep Normal output in *RenderMan for Maya*

It is fairly straight-forward to add a Deep Normal pass to RenderMan output. Inside Maya with RMS loaded, users may select '**RenderMan -> Advanced RenderMan Controls**' from the menu bar to invoke the advanced controls and from its '**Add Channels/Outputs**' pane, new output using '**normal Nn**' from '**Standard**' tab can be added and then a custom '**float a**' (this refers to the deep channel) should be added in order to complete a Deep Normal output, i.e. a, Nn. As this output will be used as normal vector data, so filtering must not be applied and users should add additional '**Settings**' using '**Add/Remove Settings**'. As shown in **figure 5.1B**, on the upper right pane, '**Filter**' and '**Filter setting**' have been added, the choice of **box filter** together with **filter size = (1,000, 1,000)** ensures this output will not be filtered. Please be reminded that the '**Deep Image (dtex)**' must be selected as the image format for this newly added output.



5.1.4 deep samples control in *RenderMan for Maya*

In RenderMan for Maya, there is no parameter available to control deep samples output (this is usually available as deep sample merge options in other renderers). Fortunately, we may add almost any kind of RenderMan options through the Ri Injection Points available from the Advanced tab of Render Settings as shown in **figure 5.1C**.

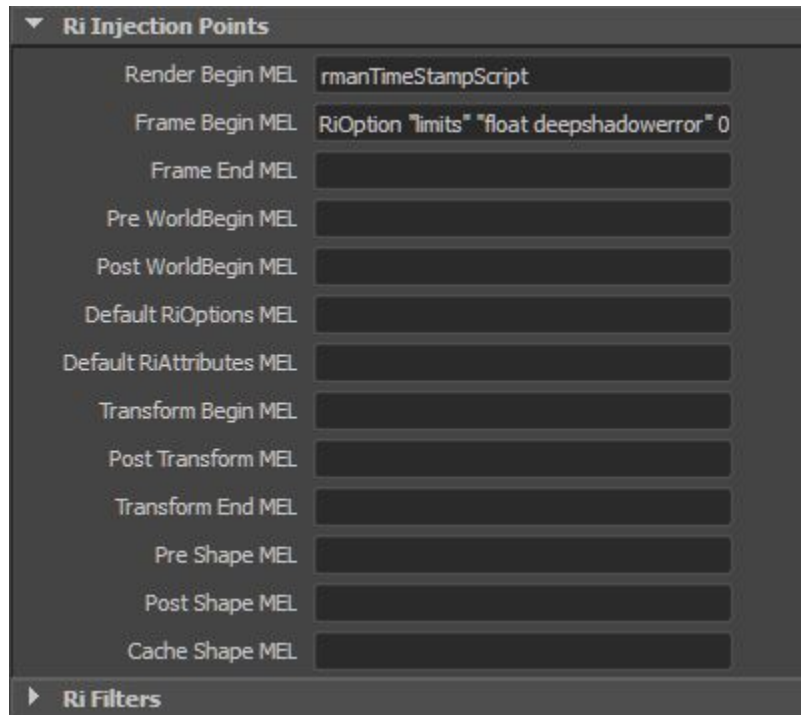


Figure 5.1C Using RiInjection for deep sample control

Users may insert the following line to **'Frame Begin MEL'** entry.

RiOption "limits" "float deepshadowerror" 0.0001;

This **'deepshadowerror'** option functions as a threshold value for deep sample merging, the lower the value it is, the fewer deep samples will be merged. According to Pixar's engineers, this value serves purely as one of many reference parameters that RenderMan considers during its deep samples computation (other considerations include memory consumption and storage for example), so even if we set its value to 0.0, some deep samples might still be merged.

5.1.5 deep normal output from *Arnold for Maya*

Arnold for Maya outputs deep images using OpenEXR 2.0 format which is commonly known as DeepEXR and Nuke has native support of OpenEXR 2.0 deep images. We used the following software version for this document:

- Maya 2015 ext. 1 + SP5
- MtoA 1.1.2.1 (Arnold Core 4.2.2.0)

5.1.6 adding Deep Normal output in *Arnold for Maya*

In order to add Deep Normal pass to Arnold output, users may simply add an **additional AOV: N** to the Active AOVs list via the **AOV Browser** available in Render Settings of Arnold for Maya as shown in **figure 5.1D**.

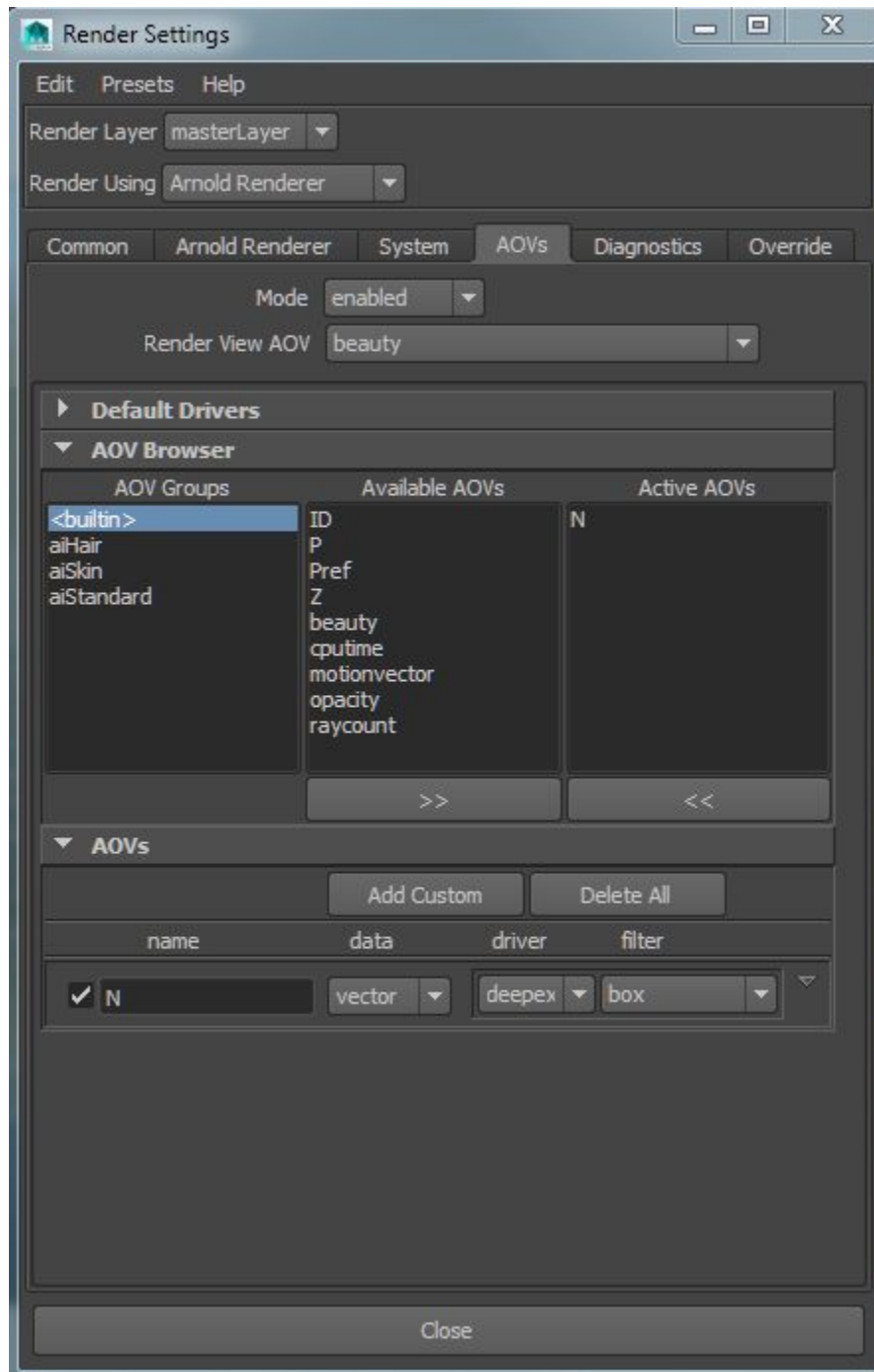


Figure 5.1D AOV Browser of Arnold's Render Settings



In addition, users have to select **'vector'** as data and **'deepexr'** as the driver under the AOVs list as shown in the lower part of **figure 5.1D**.

5.1.7 deep samples control in *Arnold for Maya*

Arnold relies on the display driver setting to configure how the data are written. When users select the AOV **'N'** from the **AOV browser**, the corresponding **aiAOVDriver** will be displayed as a tab in the Attribute Editor of Maya as shown in **figure 5.1E**.

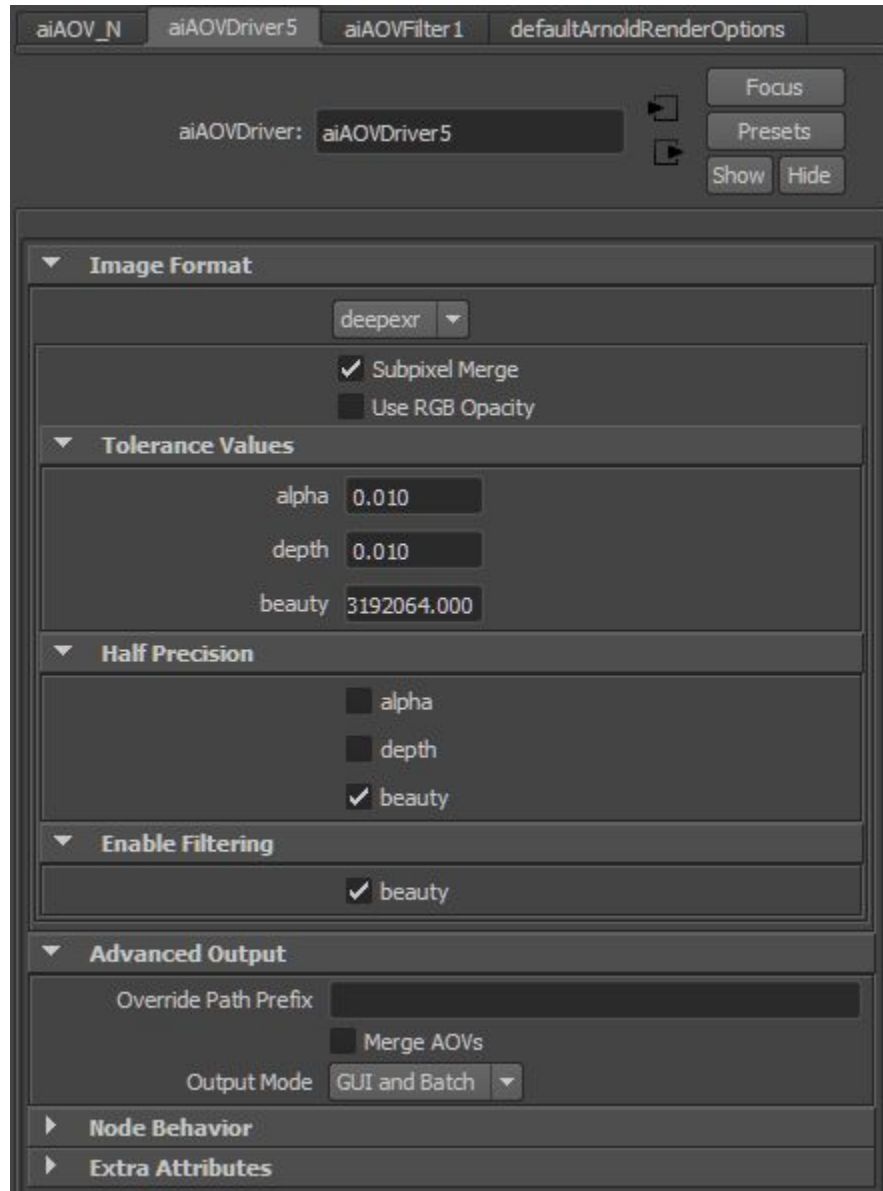


Figure 5.1E Subpixel merge of deep samples in Arnold for Maya

In the driver's panel (**figure 5.1E**), both **'alpha'** and **'depth'** parameters under **'tolerance values'** can be used to control how deep samples should be merged. Tolerance values refer to the allowed minimum difference between samples, whenever the difference is smaller than the tolerance value, a merge will happen (**in short, the larger the tolerance values are, more samples would be merged**) and please refer to Arnold user guide for further details.



5.1.8 deep normal output from *vRay for Maya*

Version 3.0 of vRay for Maya has provided deep images output support using OpenEXR 2.0 format which is commonly known as DeepEXR and Nuke has native support of OpenEXR 2.0 deep images. Whenever users choose ‘**exr (deep)**’ as image output format, vRay assumes deep image output. We used the following software version for this document:

- Maya 2015 ext. 1 + SP5
- vRay for Maya 3.0.1

5.1.9 adding Deep Normal output in *vRay for Maya*

In order to add Deep Normal pass to vRay output, users may simply add an **additional Render Element: vrayRE_Normals** to the **Added render elements** list via the **Render Elements** available in Render Settings of vRay for Maya as shown in **figure 5.1F**.

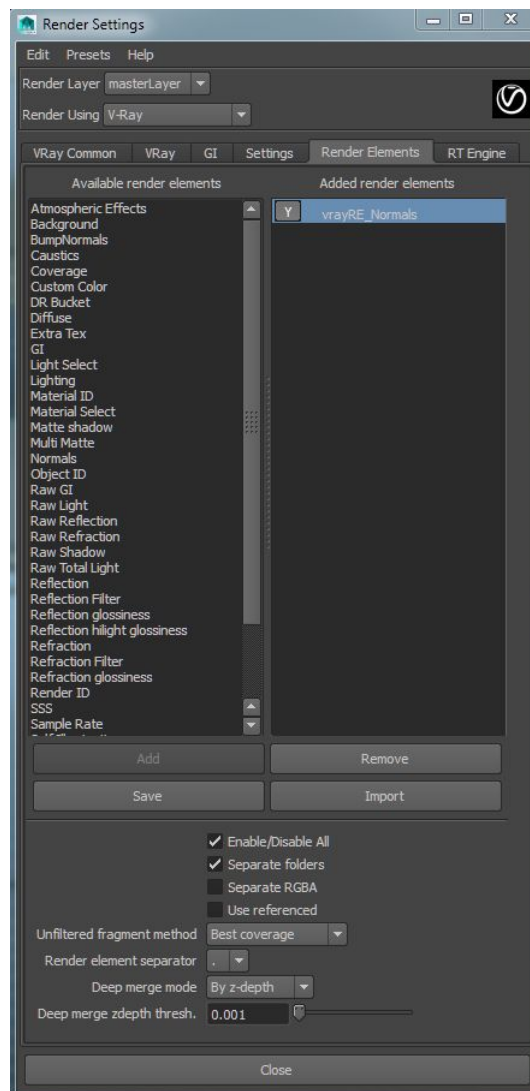


Figure 5.1F Using vRay's Render Elements



5.1.10 deep samples control in vRay for Maya

From the same Render Elements panel (figure 5.1F), users may find both ‘Deep merge mode’ and the ‘Deep merge zdepth thresh.’ at the bottom which offer similar merging controls available as in Arnold renderer and please refer to vRay for Maya user guide for further details.

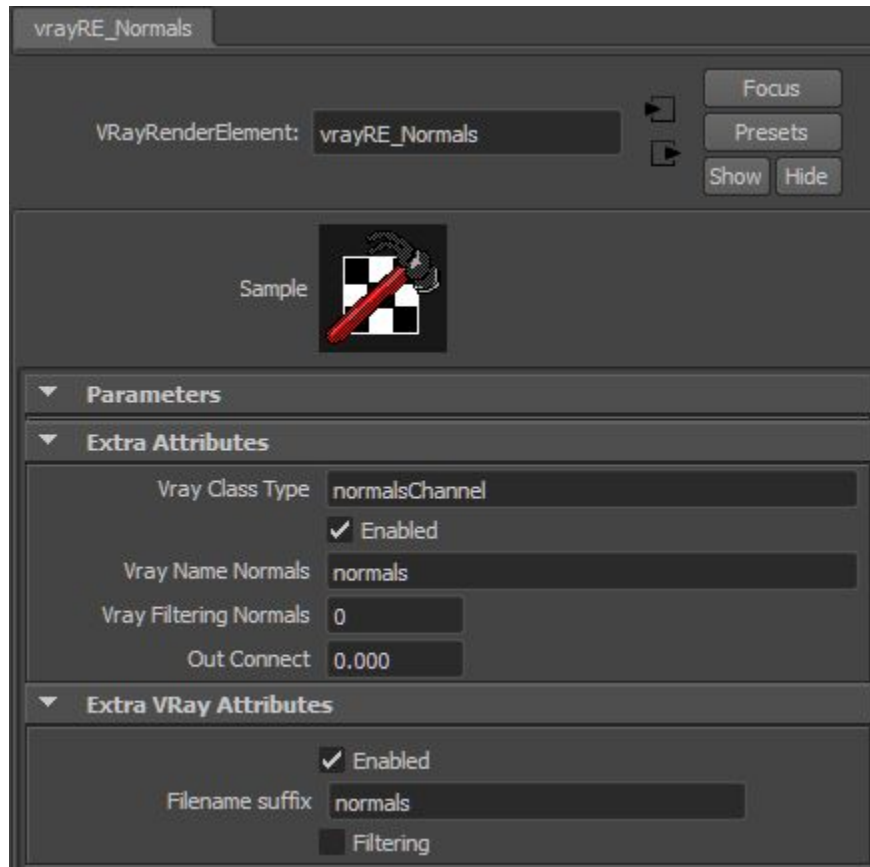


Figure 5.1G Filtering control of a Render Element

As filtering must not be applied to deep normal data, users are reminded to **turn off filtering option** via the attribute editor of the corresponding Render Element as shown in figure 5.1G above.



5.2 deepShuffleOp

5.2.1 introduction

deepShuffleOp is a utility node which allows shuffling of deep data channels from one or two deep data sources. It's a very handy node if you work with .dtx file because sometimes the alpha channel precedes the RGB and this node helps you to re-shuffle them.

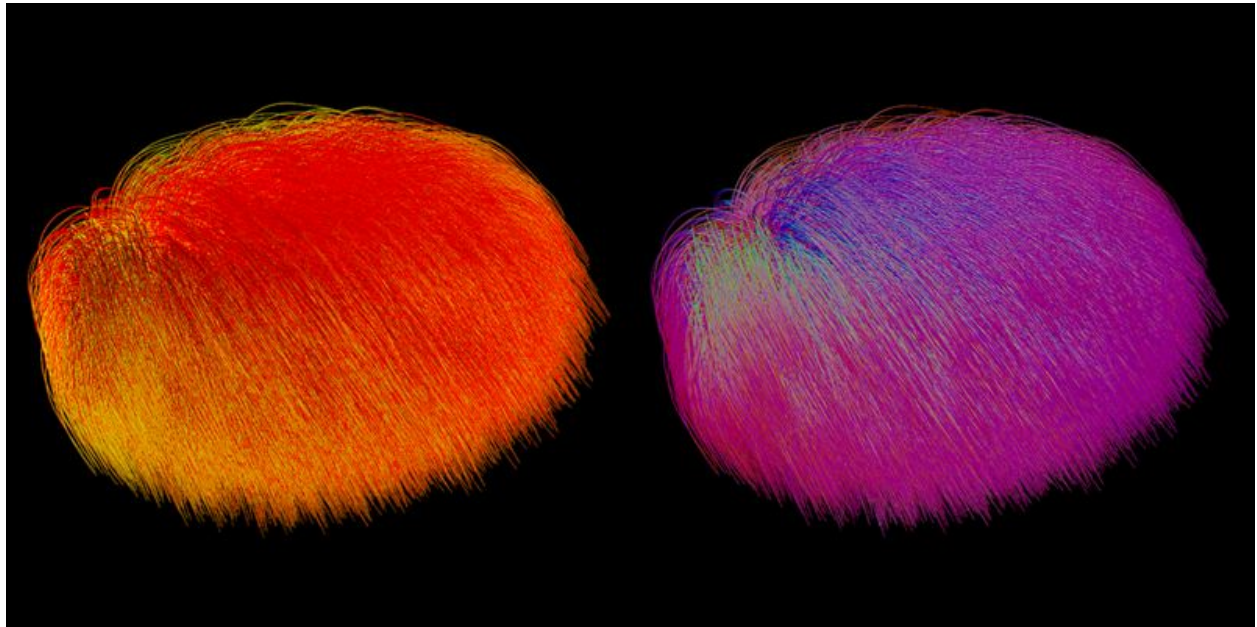


Image 5.2A Left Deep Data re-shuffled to correct Deep Normal Maps

5.2.2 quick start

1. Create a **deepShuffleOp** node from escher.
2. DeepRead one or two deep image files.
3. Connect the main deep image to the *deep_0* input of **deepShuffleOp**.
4. (Optional) Connect an optional deep image to the *aux* input of **deepShuffleOp**.
5. Choose the appropriate shuffling combination on control panel.

5.2.3 input

deepShuffleOp takes two inputs, namely **deep_0** and **aux** (optional).

deep_0 - The main source of deep data and the depth values are ALWAYS source from this input, so it determines the number of samples in each deep pixel.

aux - Auxiliary deep data source

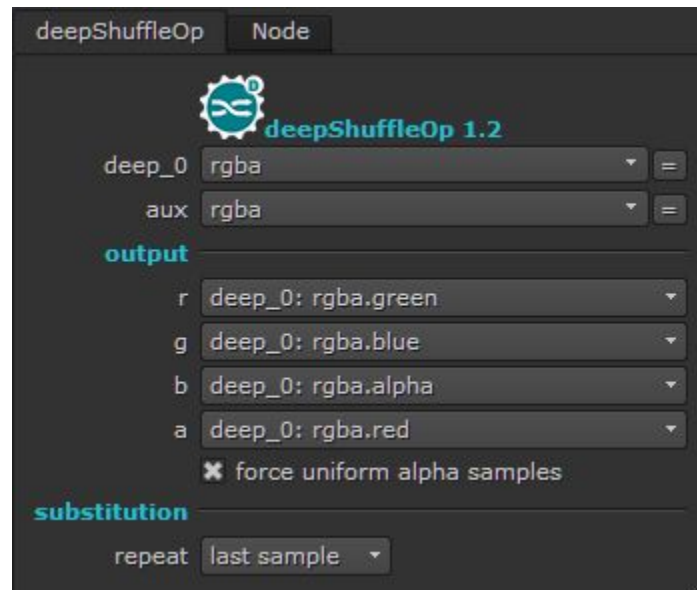


Figure 5.2B control panel of deepShuffleOp

5.2.4 parameters

- **deep_0** Choose the channel set from *deep_0* that you want to source from.
- **aux** Choose the channel set from *aux* that you want to source from.

output

- **r** The source channel that will be shuffled to output 'r'.
- **g** The source channel that will be shuffled to output 'g'.
- **b** The source channel that will be shuffled to output 'b'.
- **a** The source channel that will be shuffled to output 'a'.
- **force uniform alpha samples** with this option checked, deepShuffleOp will replace the 'alpha' samples with the ones that will ensure uniform contribution of each sample to the comp.

substitution

- **repeat** As 'depth' information will always be sourced from 'deep_0', so it happens that number of deep samples from 'aux' might not match 'deep_0', so this option instructs how deep samples should be duplicated/repeated from 'aux' when there are insufficient samples.



5.3 deepNormalOp

5.3.1 introduction

deepNormalOp is a utility node which follows the design of **normalOp** but works exclusively on Deep Normal Data only. This node will scans automatically for transformation matrix metadata from upstream (often available from the deep images) and apply them to the deep data if desired.

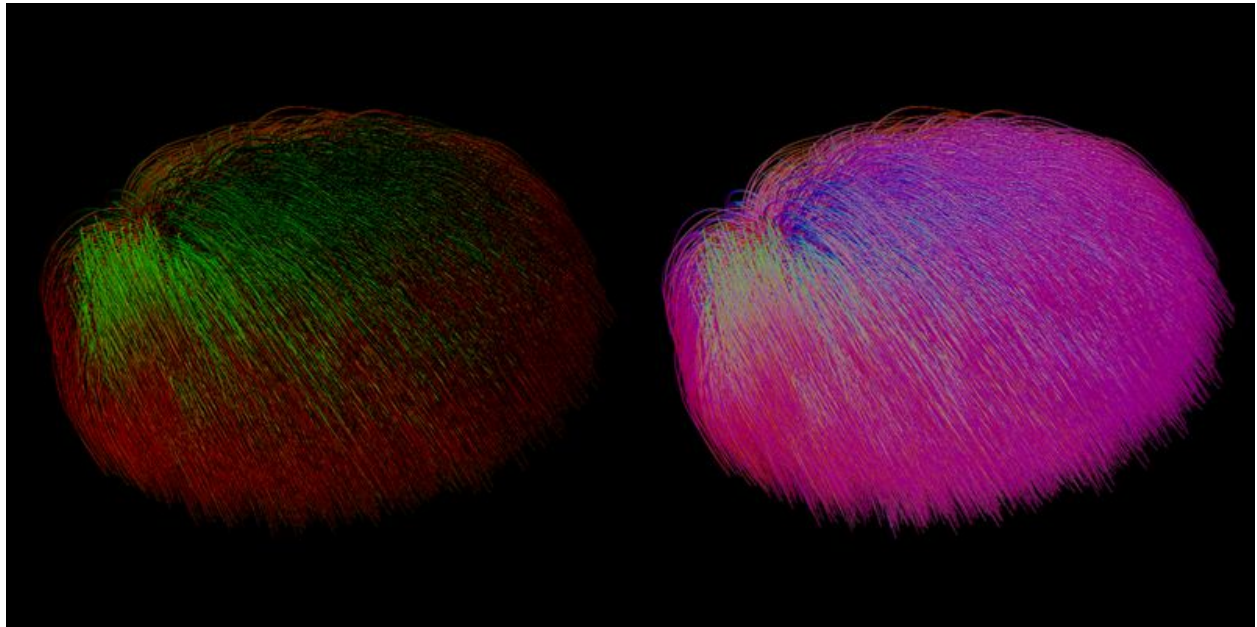


Image 5.3A Left Deep Data transformed to correct Deep Normal Maps using embedded camera matrix

5.3.2 quick start

1. Create a **deepNormalOp** node from escher.
2. DeepRead one image files.
3. Connect the deep image to the *deepNormal* input of **deepNormalOp**.
4. Choose the appropriate transformation operation on control panel.

5.3.3 input

deepNormalOp takes one single input, namely **deepNormal**.

deepNormal - The source of Deep Normal data and to be transformed by upstream metadata transformation matrix or custom matrix on the control panel. Additional Ops function exactly as **NormalOp**, please consult **NormalOp** for details.

5.3.4 parameters

- **deepNormal** Choose the channel set that you want to source Deep Normal data from.

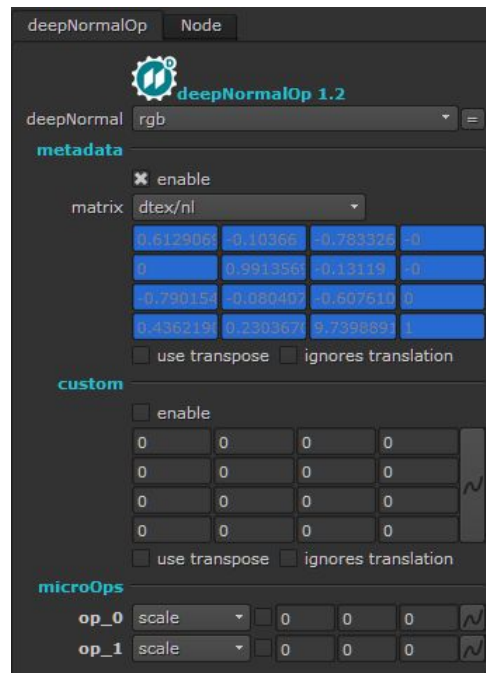


Figure 5.3B control panel of deepNormalOp

metadata

- *enable* if checked, applies the following matrix data to the Deep Normal data.
- *matrix* **deepNormalOp** will try to locate if there's any transformation matrices found from the upstream meta-data, if yes, they will be listed in the pull-down menu and users may select the one they need.
- *use transpose* - apply the transformation with the matrix transposed.
- *ignores translation* - make certain translation component will not be applied to the Deep Normal Data.

custom

Works like metadata matrix but users may manually supply a custom transformation.

microOps

This version of **deepNormalOp** offers two different microOps, and users may sequence the microOps according to their needs, the operations always start from the top (i.e.op_0) and proceed downwards; each of them could be enabled or disabled independently by the checkbox on each row.

- *scale* - A scalar to multiply the current normal map, normal axis can be flipped by inputting -1 .
- *normalise* - Length of the normals are expected to be 1 , otherwise there may be artifacts on the shaded surface, check to normalise the map.

microOps	opParameter_0	opParameter_1	opParameter_2
scale :	X	Y	Z
normalise :	/	/	/

Table A - microOps and their parameters

Please consult *normalOp* documentation for microOps details



5.4 deepShader

5.4.1 introduction

deepShader is the *Deep version* of **SHeader** specifically for shading Deep Normal map, so it shares the same shading controls of SHeader and the details can be found in *SHeader* chapter. What makes Deep 2.5D shaders special is its ability to shade multiple samples within a deep pixel and it does not only suppress the possibility of aliasing as seen in regular 2.5D shading but it also makes high quality shading of motion blurred normal vector pass possible.



Image 5.4A A Deep Normal data image rendered by *RenderMan/RIS*

5.4.2 quick start

1. Create a **deepShader** node from escher.
2. DeepRead in a Deep Normal image and connect it to 'deepNormal' input of **deepShader**.
3. Create a **SHeader** node from escher.
4. Read in a lightprobe image and connect it to the 'lightprobe' input of **SHeader**.
5. Connect **SHeader** output to 'SH data' input of **deepShader**.
6. Choose the appropriate channel set and type for the lightprobe and desired shading mode.



Image 5.4B rendered output of **deepSHader** using Lambert shading mode

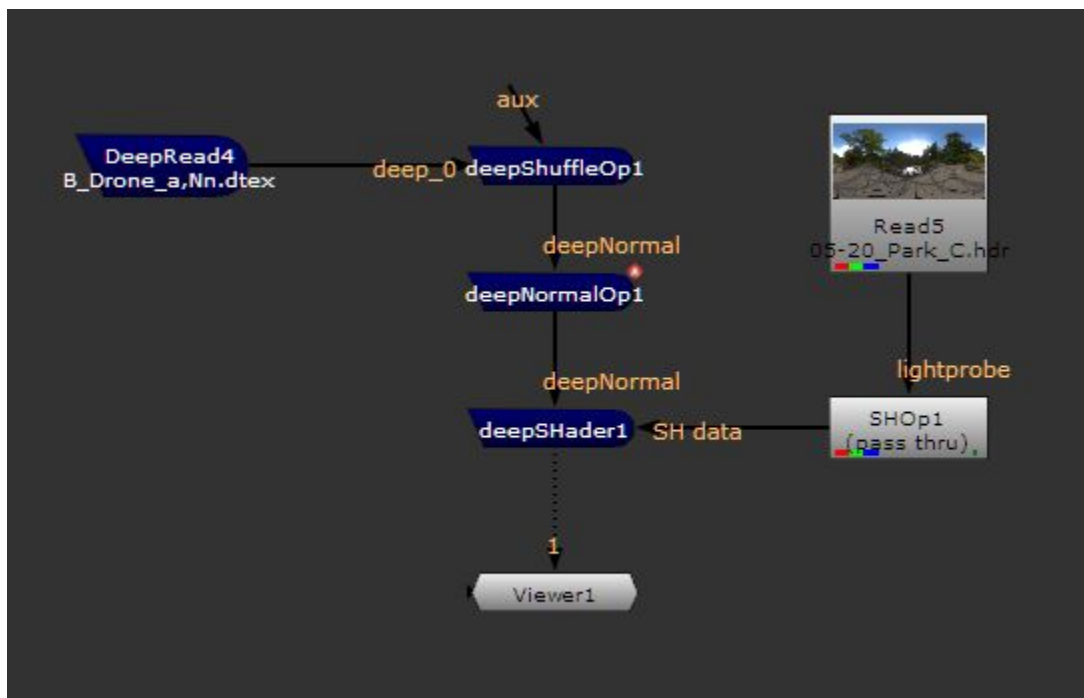


Figure 5.4C simple node graph of using **deepSHader**



5.4.3 input

deepShader takes two inputs, namely 'deepNormal' and 'SH Data'.

deepNormal - Deep Normal map used for **deepShader**. The Deep Normal map may be pre-processed with escher's *deepNormalOp* if desired.

SH Data - This input is optional it will normally rely on escher **SHOp** to feed SH Data, this connection provides **deepShader** with SH coefficients to approximate the low frequency lighting. If SH data source is set manual, this input can be left unconnected.

5.4.4 parameters

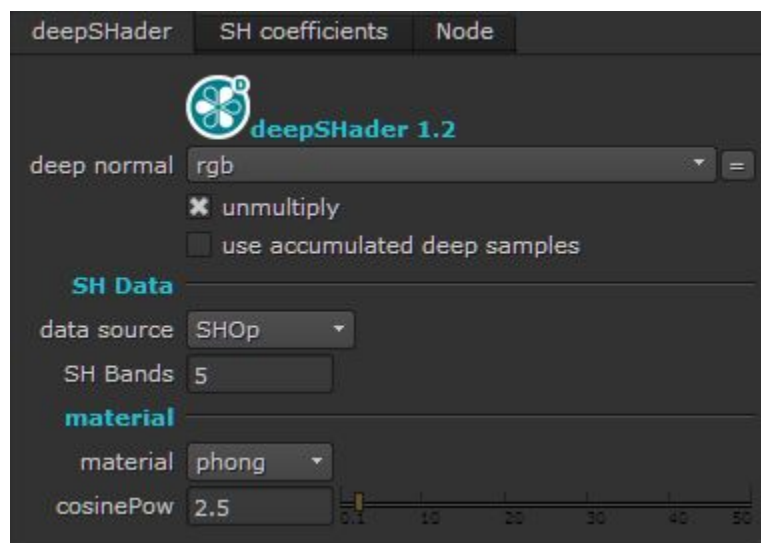


Figure 5.4D Control panel of deepShader

deepShader tab

deepNormal The source channel set of Deep Normal data.

unmultiply Un-multiply pixel sample value by the alpha sample.

use accumulated deep sample Accumulate preceding samples.

Renderer	unmultiply	accumulate
vRay	OFF	ON
Arnold	OFF	OFF
RenderMan/RIS	ON	OFF

Table 5.4 A - Recommended settings for renderers output

deepShader tab has 2 sub-panels, **SH Data** and **material**.

SH Data

- **data source** *SHOp(default)* - Pull SH coefficients directly from SHOp.
metaData - Pull the baked SH coefficients from the input(SHOp or exr)'s metaData.
manual - Input the SH coefficients manually in the SH coefficients tab.
- **SH bands** Number of SH bands used in shading calculations. (see *SHader* for more details)



material

- material

lamert(default) - Lambertian shading model.

phong - Phong shading model (users should not expect a good-looking output from it due to the low-frequency nature of the SH approach).

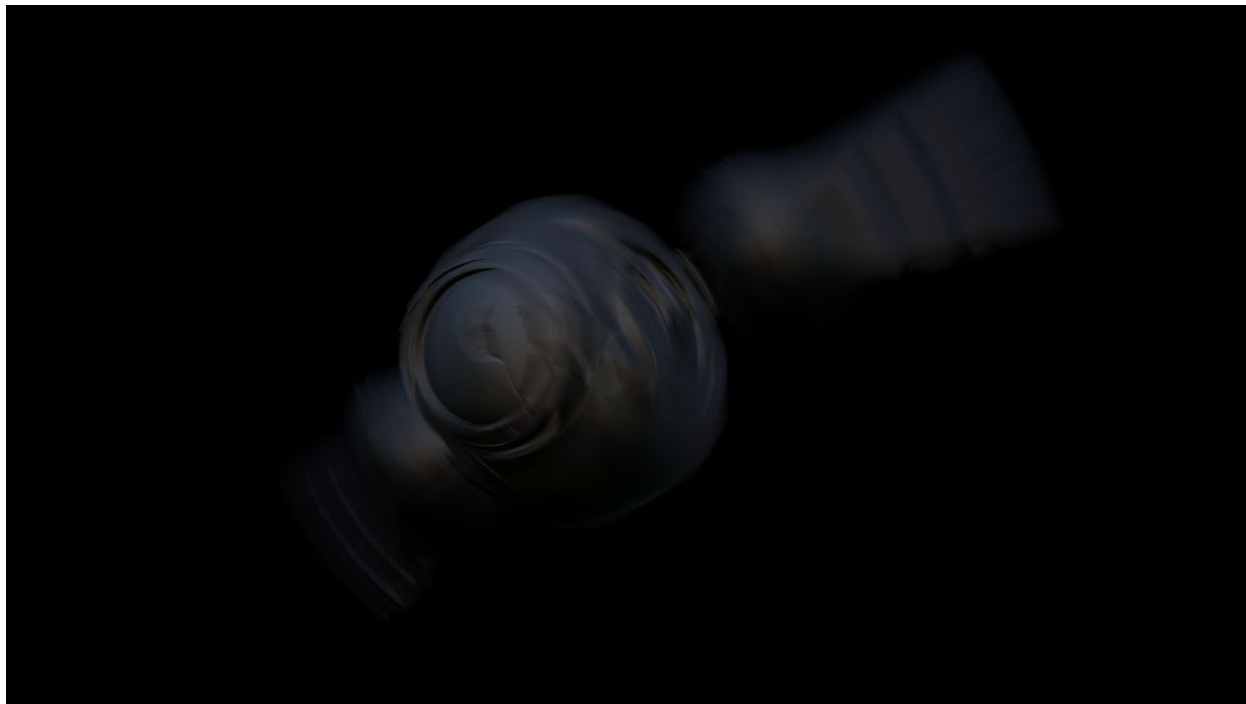


Figure 5.4E Phong shading mode

- **cosinePow** (for phong only) The cosine power factor.

SH coefficients tab

Users may examine or input the SH coefficients here. Each SH band contains (2*SH bands - 1) coefficients, and each coefficient could be regarded as a light, please refer to [SHader](#) for more details



5.5 deepGlossy

5.5.1 introduction

deepGlossy is the *Deep version* of **glossy** shader specifically for shading Deep Normal map, so it shares the same shading controls of glossy and the details can be found in *glossy* chapter. What makes Deep 2.5D shaders special is its ability to shade multiple samples within a deep pixel and it does not only suppress the possibility of aliasing as seen in regular 2.5D shading but it also makes high quality shading of motion blurred normal vector pass possible.



Image 5.5A A Deep Normal data image rendered by *RenderMan/RIS*

5.5.2 quick start

1. Create a **deepGlossy** node from escher.
2. DeepRead in a Deep Normal image and connect it to 'deepNormal' input of **deepGlossy**.
3. Read in a lightprobe image and connect it to the 'lightprobe' input of **deepGlossy**.
4. Choose the appropriate channel set and type for the lightprobe and desired shininess.

5.5.3 input

deepGlossy takes two inputs, 'lightprobe' and 'deepNormal'.

deepNormal - Deep Normal map used for **deepGlossy**. The Deep Normal map may be pre-processed with escher's *deepNormalOp* if desired.

lightprobe - **deepGlossy** assumes the light probe should be in mirror ball, angular or lat long format.



Image 5.5B rendered output of **deepGlossy** using Shininess set as 1 (quite rough)

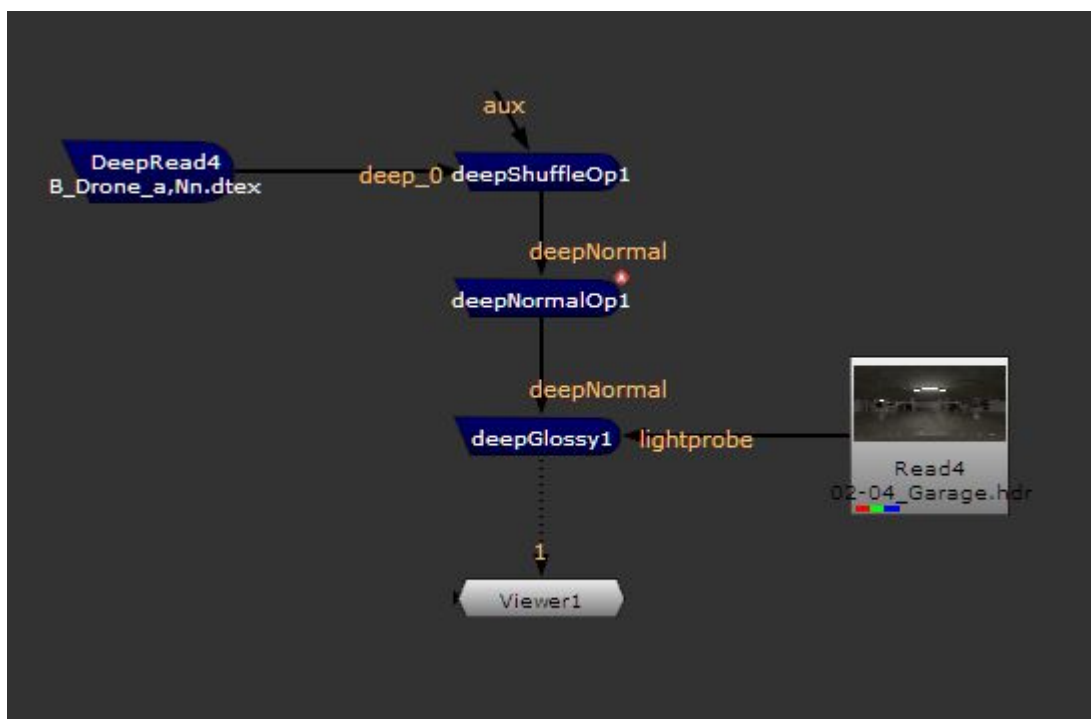


Figure 5.5C simple node graph of using **deepGlossy**



5.5.4 parameters

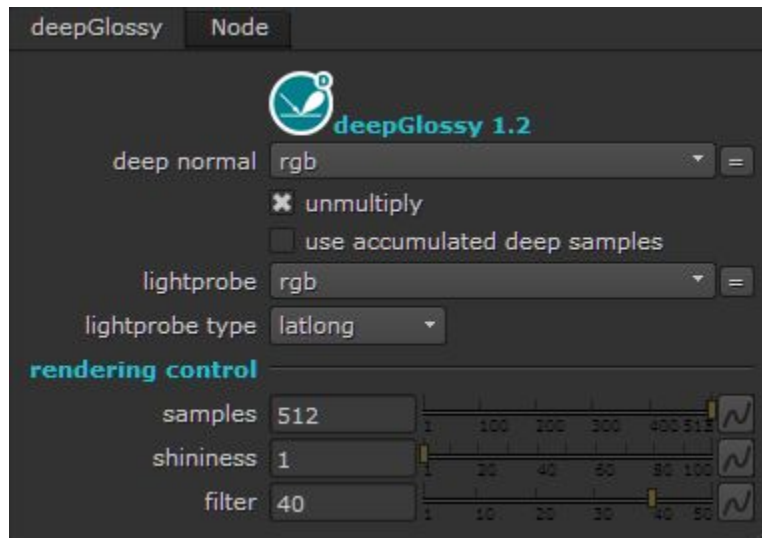


Figure 5.5D Control panel of deepGlossy

deepNormal The source channel set of Deep Normal data.

- **unmultiply** Un-multiply pixel sample value by the alpha sample.
- **use accumulated deep sample** Accumulate preceding samples.

Renderer	unmultiply	accumulate
vRay	OFF	ON
Arnold	OFF	OFF
RenderMan/RIS	ON	OFF

Table 5.5 A - Recommended settings for renderers output

- **lightprobe** Choose the appropriate channel set of the light probe.
- **lightprobe type** Select the correct light probe type for the input.
 - *mirror ball(default)*
 - *angular*
 - *lat long*

rendering control (see [glossy](#) for more details)

- **samples** Number of samples for each shading.
- **shininess** Glossiness of the surface.
- **filter** Maximum amount of pre-filtering applied to each sample.



Image 5.5E rendered output of **deepGlossy** using Shininess set as 90 (shiny)