Photorealistic physically based render engines, a comparative study.

Dsic-98

Author: Francisco Pérez Roig
Director: Ramón Mollá Vaya
Escuela Técnica Superior de Ingeniería Informática ETSINF
Universidad Politécnica de Valencia UPV
fraperoig@fiv.upv.es
September 2011

Index

Pages

Index. **2-3**
Acknowledgements. **4**
Overview. **5-6**
Structure. **7-8**
Hardware and software features. **9-10**
    Hardware Specification. 9
    Software Specifcation. 9-10

Render Engines. **10-40**
    Rendering Bias. 10-11
    Feature Chart. 12-15
    Election Criteria. 16-18
    Summary. 19
    Renderers Technical Information. 19-40
    Biased Render Engines Technical Information. 20-32
        MentalImages Mentalray v3.8.1.33. 20-24
        ChaosGroup Vray core v1.90.03. 25-27
        Dna Research 3Delight v9.0.84. 28-32
    Unbiased Render Engines Technical Information. 32-38
        NextLimit MaxwellRender v2.5. 32-35
        RandomControl FryRender v1.5. 36-38

Render Engines Comparison. **41-112**
    Lineal Workflow. 41-44
    Image Error Metrics. 45-54
    Ambient Occlusion. 55-60
    Caustics. 61-65
    Depth of Field. 66-69
    Displacement Mapping. 70-73
    Global Illumination. 74-95
    Image Based Lightning IBL. 96-101
    Motion Blur. 102-105
    Sub Surface Scattering. 106-109
    Conclusions. 110-112

Glossary of Terms. **113-196**
    Ambient Occlussion. 113-115
    Bump Mapping. 116-119
    Depth of field 120-124
    ISO 125-126
    Motion Blur 127-129
    Caustics. 130-131
    Cornell Box. 132-136

Displacement Mapping. 137-141
Global Illumination. 142-157
Image Based Lighting IBL. 158-161
IES profiles. 162-165
Normal Mapping. 166-170
Render Passes. 171-175
SSIM index 176-187
Sub Surface Scattering. 188-191
Volumetric Lighting. 192-196

Contact. 197

Acknowledgements

This study would not have been possible without the approval of my project supervisor Ramón Mollá, for which I am really grateful.

I would like to thank my parents and girlfriend for their personal support and great patience at all times.

Last, but by no means least, I thank my friends all around the world from CgTalk site for their support and help.

For any errors or inadequacies that may remain in this work, of course, the responsibility is entirely my own.

Overview

Tha main purpose of this paper entitled "Photorealistic physiscally based render engines, a comparative study." is an exhaustive study of nowadays' physically based render engines.

The term "*physically based*" indicates the use of physical models and approximations that are more general and widely accepted outside rendering. A particular set of related techniques have gradually become established in the rendering community.

The basic concepts are moderately straightforward, but intractable to calculate; and a single elegant algorithm or approach has been elusive for more general purpose renderers. In order to meet demands of robustness, accuracy and practicality, an implementation will be a complex combination of different techniques.

Therefore, this paper will cover in a rigorous manner all relevant render techniques available to obtain hiperealistic syntetic images, to be applied to the most capable render engines in the market. This group of most capable render engines will be obtained as a result of a preliminary comparison between all render engines found in the market, attending to technical, economical, and availability factors.

From that group, an exhaustive comparison of the photorealistic techniques available will be done, in order to show up which renderers work better in each lighting situation.
Not only quality or accuracy factors will be considered, render time will also be an important variable to take into consideration.
That will be the main part of this study, and therefore the clear explanation of these techniques will be mandatory.

A list of these main techniques being covered in this study is shown below.

- Ambient Occlusion.
- BSSRDF materials.
- Bump mapping.

- Camera Lens Effects.
    1. Camera Depth of Field.
    2. Motion Blur.

- Displacement mapping.
    1. Traditional Displacement.
    2. Sub-pixel Displacement.

- Global Illumination.
- IES profiles
- Image-Based Lighting (IBL).
- Normal mapping.
- MultiPass Rendering.

- Photon mapping.
    1. Caustics.
    2. Volume Lighting.
        -Volume Fog.
        -Volume Caustics.

To ensure the correct execution of this task, several standard scenes will be modeled in order to get accurate results between render engines in each situation. These are scenes like CornellBox, Sky Dome ...

Photorealistic techniques shown above will then be applied to this scenes.
It will be necessary to accurately choose the parameters in each render engine, so that resulting images and rendertimes are objective, not being conditionated by the author's poor parameter choice.

Once the comparison is done and the rendering techniques are exposed, nowaday's photorealistic rendering field will have been studied in depth both in a theorical and practical manner.

Structure

This study contains three clearly defined main sections.

- Hardware, Software and Render Engines' Features.
- Comparison
- Glossary of Terms

The first section, is splitted in two parts.
On one hand, a list the technical features of the Workstation's hardware being used throughout this study will be presented. That involves processor type, RAM memory amount, etc. It is worth noting that all renders produced in this study have been done with this workstation, therefore rendertimes are conditioned by this particular Hardware setup.

On the other hand, "Software Features" chapter consists first in a brief exposition of the operating system and modelling production software that is used, and the reasons for chosing it.
Then, all physically based render engines available found in the market will be compared according to the documentation provided by their producers or vendors and a list of minimun technical features requirements will be exposed.
As a result of analysing each renderer's technical features, and refusing the use of those that do not meet the minimum requirements, a final list of render engines will be obtained. These will be the render engines that will be installed in our system to develope the following sections.

Once the final render engines are chosen, a side to side comparison of each technique with those render engines is done. For each technique, several scenes are built and identically reproduced in all render engines, so the results shown for each of them accurately represent the same scene.
Those scenes will be then rendered with different parameters. The resulting images and their rendertimes are then presented.

All that done, a quality versus render time ratio is obtained showing the limitations of each render engine in each area.

Next, the conclusions chapter will summarize the results obtained with each render engine, pointing out the weaknesses of each one, so the reader can easily notice what renderer is more efficient or physically accurate in each situation.

Finally, a proposal for further studies will be exposed. That will give an idea of what fields in 3D rendering would be worth investigating further, and a brief understanding of what will presumably be the future of photorealistic render engines in the short and middle term.

For the correct understanding of the above sections, it was necessary to build a glossary of terms containing an explanatory but brief description of all techniques that are used throughout the paper. Several scenes of each technique are built and rendered so the reader could easily familiarize with the technical aspects of each technique and have a better consideration of the conclusions obtained in the comparison.

The addition of this last section allowes to clearify the comparison section, where only rendered image data and related conclusions should be shown.

Doing it that way, allows the reader to easily appreciate the differences between the different techniques and render engines in the comparison section side by side, without the inconveniences of the inclussion of theorical text paragraphs between images.

Hardware & Software features

Hardware Specification.

The following Workstation will be used in all purposes throughout this paper without exception. Therefore, rendertimes will be conditioned by this specific setup in addition to the Software/Operating System being used [See Software Specification Section]

- **Motherboard** : ASUS KGPE-D16 Dual Socket G34 AMD SR5690 SSI EEB 3.61 Dual 8/12 Core AMD Opteron 6000 series Server Motherboard
- **Processors** : 2x AMD Opteron 6128 Magny-Cours 2.0GHz 8 x 512KB L2 Cache 12MB L3 Cache Socket G34 115W 8-Core Server Processor
- **Cpu Cooling** : 2x Noctua NH-U12DO A3
- **Memory** : 8x Kingston ValueRAM 2GB 240-Pin DDR3 SDRAM ECC Unbuffered DDR3 1333 Server Memory (4 Modules for each cpu in quad-channel)
- **Graphics** : ATI 100-505606 FirePro V4800 1GB GDDR5 PCI Express 2.0 x16 Workstation Video Card
- **Power Supply** : Corsair Professional Series AX850 850W ATX12V v2.31 / EPS12V v2.92 80 PLUS Gold Certified Modular Active PFC Power Supply
- **Primary Storage** : OCZ Vertex 2 OCZSSD2-2VTXE60G 2.5" 60GB SATA II MLC Internal Solid State Drive (SSD)
- **Data Storage** : Seagate Barracuda 7200.12 1TB 3.5" SATA 3.0Gb/s Internal Hard Drive -Bare Drive

Software Specification.

Operating System.

Microsoft Windows 7 professional x64 operating system will be treated as the primary OS used for all purposes throughout this document.

A 64bit version of the OS is chosen basicaly for two reasons.
Firstly it will be necessary to take advantage of the 16GB of Ram available in the workstation when rendering dense polygoned scenes that need to be loaded in RAM, whereas using a 32bit system will lead to a limited amount of 4GB.
On the other hand, we could expect a major performance improvement when taking advantage of the 64bit computing capability of the two AMD Opteron 6128 processors.

Microsoft software is used just because render engines tend to be compiled for Windows first.

A similar performance could be expected when using Unix systems if supported by the render engine, but some of the studied in this paper are not compiled for Unix platforms [see render engines section], and it is important to unify the system configuration used between all render softwares to ensure a good comparison.

Production Software.

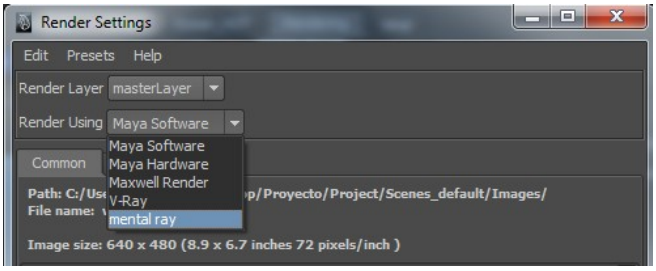
Autodesk Maya 2011 sp1 x64 is the production software chosen for modelling and rendering tasks in this study.

Maya has been a 3D tool of choice for companies producing the top film, games, and television content throughout the world for the last decade. Itself an Academy Award® winner for Scientific and Technical Achievement in 2003, Maya has been extensively used by the vast majority of nominees and winners for the Academy Award for Best Visual Effects since 1999 [1]

It also integrates quite well with most popular render engines that take part on the comparison [see Render Enines section] so it is possible to render the different scenes conceiled in this paper with the different render engines directly in Maya's interface.

Maya plug-ins are available for most of these render engines.
That gives the possibility to assign shaders to the objects in the scene, set the illumination and configure its render settings in Maya's own viewport without the need of third party standalone applications.

The following image shows Maya's render settings tab where you can choose between different renderers to use in the scene (The own Maya's renderers and Maxwell Rener, V-Ray and Mentalray in this case).



Render Layer masterLayer

Render Using Maya Software

Path: C:/Us p/Proyecto/Project/Scenes_default/Images/

File name:

Image size: 640 x 480 (8.9 x 6.7 inches 72 pixels/inch)

Maya 2011 Render Settings Tab

There is no need to export the geometry and import it then into the different render engine's applications and that highly improves flexibility and productivity, avoiding the use of several different softwares.

In fact, Mentalray 3.8 [see render engine section] render engine is included and fully integrated with Maya. That saves from the need of purchasing another license.

From the point of view of hardware capabilities, Nvidia Quadro 600 video card installed in our workstation is certified by Autodesk for this software, and that ensures the best possible performance when managing big scenes with high polygonal count.

References

[1] Autodesk Maya 2011. Top reasons to buy, © 2010 Autodesk, Inc.

Render Engines.

Once described which hardware and software configuration will be used throughout this study, it is time to decide which render engines to choose between all renderers available in the market for the comparison.

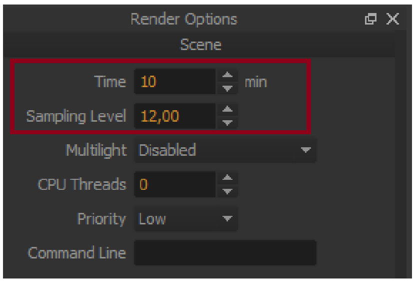
First, it is important to divide the most relevant render engines available in two different groups, biased and unbiased renderers [see Rendering Bias section]. These use a totally different nature of render algorithms so different aspects will be considered when choosing both kinds of renderers.

Next, a feature chart will show the capabilities of each render engine in both technical and commercial terms to ensure that the renderers chosen are the most capable, flexible, robust and photorealisticaly accurate in the market.

Rendering Bias.

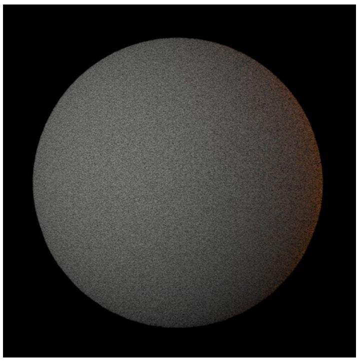
The term bias is used to define the error present in a rendering algorithm. It basicaly measures how easy the algorithm is to use and how well it works regardless of the scene being rendered.

Unbiased render engines give the possibility to choose the level of error desired for a given render or the computation time limit when the render will stop at a given render sampling (or quality) level. One example of this kind of render engines is MaxwellRender [see feature chart section], refer to the following image.

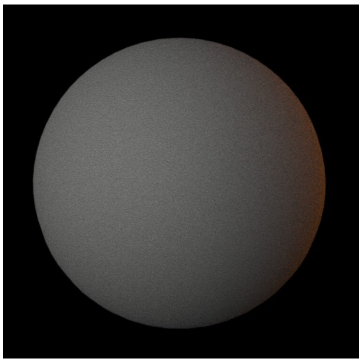


Time 10 min
Sampling Level 12,00
Multilight Disabled
CPU Threads 0
Priority Low
Command Line

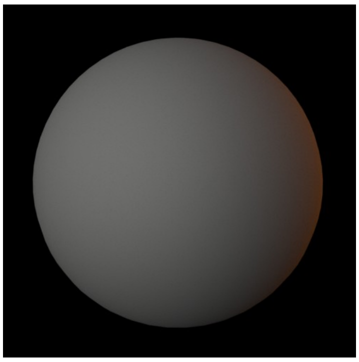
Maxwell Render render options



Sampling quality = 3



Samplig quality = 5



Samplig quality = 10

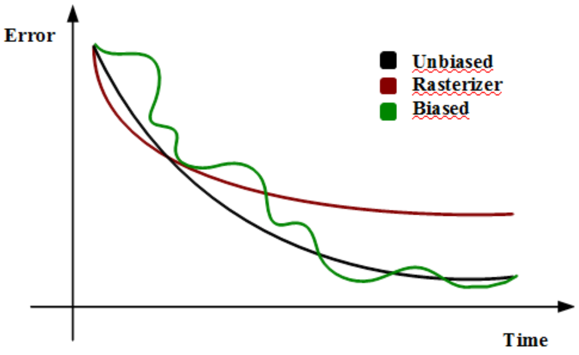
Note the noise grain in the first two images where the samplig quality is too low and therefore the error is too high. The third image shows no grain because it reached enough samplig quality.

In unbiased engines, less parameters need to be adjusted to produce an accurate photorealistic image, as long as there are no approximations performed by the render algorithm that need to be adjusted. The counterpoint of this type of algorithms is that they are generally more expensive computational speaking.

Biased engines, therefore tend to be less expenssive to render in general terms. That is achieved by making simplifying assumptions or approximations that improve performance while adding some sort of error or bias.

Generally speaking, there is a tradeoff between fast, biased methods and robust, unbiased methods [1].

Most commercial render engines use some sort of fast approximations that are neither biased nor unbiased (rasterizer, Reyes, path tracing...) but render engines that use that kind of algorithms tend to be called biased.
The next graph shows the relationship between the error and the computation time given a real biased (eg. Photon mapping) , an unbiased, and a common method used by most biased commercial engines (rasterizer) .



Error vs Computation Time graph.

Note how the biased method gets to a consistent solution but the error is not predictable at all. Rasterizer method is highly predictable and converge relatively fast to a not very accurate solution, that is the goal of most commercial render engines.
On the other hand, the unbiased method converge slower but to a more accurate solution.

That said, one of the purposes of this paper will be to compare between this two types of algorithms [see election critera section] to study whereas a biased render engine can be fast and still accurate and an unbiased render can render photorealistic images with good performance.

Refer to the chart [see Feature chart section] to see a list of biased and unbiased photorealistic render engines.

References

[1] Keenan Crane, “Bias in Rendering”, kcrane@uiue.edu

Feature chart.

At this point, is mandatory to decide a list of minimum technical render capabilities that the render engines used in this paper should be able to handle.
Once this list is described, render engines available in the market will be filtered so they pass these minimum requirements.
From the resulting render engines, a more in depth study of their technical features will be made, in order to filter the most robust and capable.

Non-technical specifications will be compared from the resulting renderers.
Only the ones with a large community, providing best support, and licensed for multi-threading systems will be finally used in this study.

General technical features.

The list below shows several photorealistic tecniques, and technical specifications that must be considered in the process of selecting our final render engines.
These techniques will be explored in depth for each resulting render engines, and are the base of the comparison that will be covered in the following chapters.

Therefore, is mandatory that the final render engines used in this paper meet most if not all of this features.

- Caustics
- Volume caustics
- Volume shadows
- Global Illumination
- Image-Based Lightning
- IES/ Elumdat lightning
- Light Mapping
- Ambient Occlussion*
- Bump mapping
- Normal mapping
- Sub surface scattering
- 3D motion blur
- Depth of field
- Sub-pixel Displacement Map
- Render passes
- Multithreaded
- 64-bit compiled

**Ambient occlussion will only be considered for biased render engines, as it is a lightning simulation technique present in unbiased renderers by default. [See ambient Occlussion chapter in Glossary of Terms section].*

A detailed explanation of each technique is shown in Glossary of Terms chapter at the end of this paper. Please refer to that section for further information of a given technique.

With the above list, the following chart shows each render engine and the features it supports.

Render Engine	Biased/Unbiased[1]	GI algorithm	Caustics[2]	Volume Caustics[3]	Volume Shadows[3]	GI[4]	IBL[5]	IES / Elumdat[6]	Ambient Occlusion[7]	Bump Mapping[8]	Normal Mapping[9]	real SSS[10]	Motion Blur[14]	Depth of field[11]	Sub-pixel Displacement[12]	Passes[13]	Multithread	64-bit
Dna Research **3Delight**	Biased/ **Reyes**	Raytracing	yes	yes	yes	yes	yes	no	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
Graffiti Software **Arc+Render**	Biased	N/A	no	no	no	yes	no	no	no	yes	no	no	no	no	no	no	n/a***	no
SitexGraphics **AIR**	Biased	Raytracing	yes	no	no	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
DctSystems **Angel**	Biased/ **Reyes**	Raytracing	no	no	no	yes	no	no	no	no	no	no	no	no	yes	no	n/a***	no
Aqsis	Biased/ **Reyes**	Reyes local GI	no	no	no	yes	no	no	yes	yes	yes	no	yes	yes	yes	yes	yes	yes
Accurender Ntx	Biased	Raytracing	yes	no	no	yes	yes	yes	no	yes	yes	no	no	yes	no	no	yes	yes
Splutterfish **Brazil**	Biased	N/A	no	no	no	yes	yes	yes	yes	yes	yes	yes	yes	yes	no	yes	yes	yes
dRad	Biased	radiosity	no	no	no	yes	no	no	no	no	no	no	no	no	no	no	n/a***	no
Google **Elvishray**	Biased	final gathering, irradianc	yes	no	no	yes	no	no	no	yes	no	no	no	no	no	no	n/a***	no
RandomControl **Fryrender**	Unbiased	Unbiased	yes	yes	yes	yes	yes	yes	yes (built-in)	yes	yes	yes	yes	yes	yes	yes	yes	yes
Igneus	Biased	Raytracing, irradiance c	yes	yes	yes	yes	yes	no	no	yes	yes	no	no	yes	no	no	yes	no
Radiance	Biased	Deterministic and Stoch	no	no	no	yes	no	no	no	no	no	no	no	no	no	no	n/a***	no
Glare Technologies **Indigo**	Unbiased	pathtracing, bi-directio	yes	yes	yes	yes	yes	no	yes(built-in)	yes	no	yes	yes	no (faked)	no	yes	yes	yes
Integra **Inspirer**	Biased	bi-directional Monte Car	no	no	no	yes	yes	no	no	no	no	no	no	no	no	no	yes	yes
JaTrac	Biased	photon mapping with irra	no	no	no	yes	no	no	no	yes	no	no	no	no	no	no	n/a***	no
jrMan	Biased/ **Reyes**	Reyes	no	no	no	yes	no	no	no	yes	yes	no	no	no	yes	no	n/a***	no
Bunkspeed **Shot**	Biased	final gathering, raytracin	yes	no	no	yes	yes	no	yes	yes	yes	no	no	yes	yes	no	yes	yes
Kerkythea	Biased&Unbiased	Ray Tracing,Path Tracin	yes	yes	yes	yes	yes	yes	no	yes	yes	yes	no	yes	no	no	yes	no
Kray tracing	Biased	Path tracing, photon ma	yes	no	no	yes	no	no	no	yes	yes	no	yes	yes	no	no	yes	yes
Lucille	Biased	Monte Carlo Ray Tracin	no	no	no	yes	yes	no	no	no	no	no	no	no	no	no	n/a***	no
Lux Render	Unbiased	Bidirectional Pathtracin	yes	no	no	yes	yes	yes	no	yes	yes	yes	yes	yes	yes	no	yes	yes
NextLimit **Maxwellrender**	Unbiased	Unbiased	yes	yes	yes	yes	yes	yes	yes (built-in)	yes	yes	yes	yes	yes	yes	yes	yes	yes
MentalImages **Mentalray**	Biased (raytracing, ra	Raytracing, Photon map	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
Sunfish studio **Meridian**	Biased	N/A	no	no	no	yes	no	no	yes	yes	no	yes	yes	yes	yes	no	yes	no
Muse	Biased/ **Reyes**	Monte Carlo Ray Tracin	yes	no	no	yes	yes	no	yes	yes	yes	yes	yes	yes	yes	no	yes	yes
Hxa Minilight	Biased	Monte-carlo path-tracin	no	no	no	yes	no	no	no	no	no	no	no	no	no	no	n/a***	no
Hxa **Perceptum**	Biased	Monte-carlo raytracing	yes	no	no	yes	no	no	no	yes	no	no	no	yes	no	no	n/a***	no
Pixie	Biased / **Reyes**	photon mapping and irra	yes	no	no	yes	yes		yes	yes	yes	yes	yes	yes	yes	no	yes	yes
Per. Vision Raytracer **PovRay**	Biased	Ray tracing	yes	no	yes	yes	no	no	no	yes	yes	no	no	yes	no	no	n/a***	no
Pixar **Renderman**	Biased / **Reyes**		yes	yes	yes	yes	yes	no	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
RenderPark	Biased&Unbiased	Galerkin radiosity, Stoc	yes	no	no	yes	no	no	no	no	no	no	no	no	no	no	n/a***	no
RenderSpud (developement)	Biased&Unbiased	Metropolis light transpor	yes	yes	yes	yes	yes	no	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
SunFlow	Biased	Photon Mapping , irradia	yes	no	no	yes	yes	no	yes	yes	yes	no	yes	yes	yes	no	yes	no
Thea Render	Biased	Photon Mapping,Irradian	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	no	yes	yes	yes	yes	yes
Toxic	Biased	Photon Mapping,Irradian	yes	no	no	yes	no	no	no	no	no	no	no	no	no	no	no	no
ChaosGroup **Vray**	Biased	path tracing, irradiance	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
WinOsi	Biased	'Iterative Two Pass Optic	yes	no	no	yes	yes	no	no	no	no	no	yes	yes	no	no	yes	no
Yafaray	Biased	Raytracing, Photon Map	yes	yes	yes	yes	yes	no	yes	yes	yes	no	yes	yes	yes	no	yes	no**
Yasrt	Biased	Raytracing	no	no	no	yes	no	no	no	no	no	no	no	yes	no	no	yes	no

1 Refer to Bias in Rendering section in Software Features chapter.
2 Refer to Caustics section in Glossary of Terms.
3 Refer to Volume lightning section in Glossary of Terms.
4 Refer to Global Illumination section in Glossary of Terms.
5 Refer to Image Based Lightning section in Gloassary of Terms.
6 Refer to IES profiles section in Glossary of Terms.
7 Refer to Ambient Occlussion section in Glossary of Terms.
8 Refer to Bump Mapping section in Glossary of Terms.
9 Refer to Normal Mapping section in Glossary of Terms.
10 Refer to SubSurface Scattering section in Glossary of Terms.
11 Refer to Depth of Field section in Glossary of Terms.
12 Refer to Displacement Mapping section in Glossary of Terms.
13 Refer to Layer Rendering section in Glossary of Terms.
14 Refer to Motion Blur section in Glossary of Terms.

**** Yafaray windows 64bit binary not available.**
***** Renderer documentatio does not provide related information**

Render Engines Feature Chart

References

[1] *http://www.3delight.com/en/*
[2] *http://www.arcrender.com/*
[3] *http://www.sitexgraphics.com/html/air.html*
[4] *http://www.dctsystems.co.uk/RenderMan/angel.html*
[5] *http://www.aqsis.org/*
[6] *http://www.accurender.com/*
[7] *http://www.splutterfish.com/sf/WebContent/B4Max*
[8] *http://users.softlab.ece.ntua.gr/~jpanta/Graphics/HierarchicalRadiosity/*
[9] *http://code.google.com/p/elvishray/*
[10] *http://fryrender.com/*
[11] *http://www.igneus.co.uk/*
[12] *http://radsite.lbl.gov/radiance/HOME.html*
[13] *http://www.indigorenderer.com/*
[14] *http://www.integra.jp/en/inspirer/index.html*
[15] *http://wfmh.org.pl/thorgal/jatrac/*
[16] *http://www.jrman.org/*
[17] *http://www.bunkspeed.com/hypershot/*
[18] *http://www.kerkythea.net/joomla/*
[19] *http://www.kraytracing.com/*
[20] *http://lucille.sourceforge.net/*
[21] *http://www.luxrender.net/*
[22] *http://www.maxwellrender.com/*
[23] *http://www.mentalimages.com/products/mental-ray.html*
[24] *http://sunfishstudio.com/software.htm*
[25] *http://viarender.com.sapo.pt/products/muse/index.htm*
[26] *http://www.hxa.name/minilight/*
[27] *http://www.hxa7241.org/perceptuum/*
[28] *http://www.renderpixie.com/*
[29] *http://www.povray.org/*
[30] *https://renderman.pixar.com/*
[31] *http://www.cs.kuleuven.ac.be/cwis/research/graphics/RENDERPARK/*
[32] *http://renderspud.blogspot.com/*
[33] *http://sunflow.sourceforge.net/*
[34] *http://www.thearender.com/*
[35] *http://www.toxicengine.org/*
[36] *http://www.chaosgroup.com/en/2/vray.html*
[37] *http://www.winosi.onlinehome.de/*
[38] *http://yafray.org/*
[39] *http://www.yasrt.org/*

Election criteria.

Once the table is built we can clearly group a number of valid candidates, and get rid of the render engines that do not fit most of our general technical requirements, please refer to “general technical requirements” section above this document.

First of all, render engines not compiled fot 64 bit platforms are eliminated from the table. Comparison between render engines would not be significative if both 32bit and 64bit softwares are mixed together. Both Opteron 6128 processors in the workstation that is being used [see hardware features section] offer 64bit architecture and therefore it is not logical not taking advantage of the 64bit software capabilities in terms of performance.

The same occures with non multithreaded render engines, it is mandatory to use render engines in our comparison that offer good scalability for multithreaded systems close to 100% utilization. That will provide a boost of performance in the 16 core system used in this paper. Mixing both multithreaded and non multithreaded renderers would lead to a situation where rendertime results are impossible to compare.

After applying this filter the resulting table with the valid candidates stays as follows.

Render Engine	**Biased/Unbiased[1]**	**GI algorithm**	**Caustics[2]**	**Volume Caustics[3]**	**Volume Shadows[3]**	**GI[4]**	**IBL[5]**	**IES / Elumdat[6]**	**Ambient Occlusion[7]**	**Bump Mapping[8]**	**Normal Mapping[9]**	**real SSS[10]**	**Motion Blur[14]**	**Depth of field[11]**	**Sub-pixel Displacement[12]**	**Passes[13]**	**Multithread**	**64-bit**
Dna Research **3Delight**	Biased/ **Reyes**	Raytracing	yes	yes	yes	yes	yes	no	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
SitexGraphics **AIR**	Biased	Raytracing	yes	no	no	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
Accurender Ntx	Biased	Raytracing	yes	no	no	yes	yes	yes	no	yes	yes	no	no	yes	no	no	yes	yes
Splutterfish **Brazil**	Biased	N/A	no	no	no	yes	yes	yes	yes	yes	yes	yes	yes	yes	no	yes	yes	yes
RandomControl **Fryrender**	Unbiased	Unbiased	yes	yes	yes	yes	yes	yes	yes (built-in)	yes	yes	yes	yes	yes	yes	yes	yes	yes
Glare Technologies **Indigo**	Unbiased	pathtracing, bi-direct	yes	yes	yes	yes	yes	no	yes(built-in)	yes	no	yes	yes	no (faked)	no	yes	yes	yes
Bunkspeed **Shot**	Biased	final gathering, raytrac	yes	no	no	yes	yes	no	yes	yes	yes	no	no	yes	yes	no	yes	yes
Lux Render	Unbiased	Bidirectional Pathtraci	yes	no	no	yes	yes	yes	no	yes	yes	yes	yes	yes	yes	no	yes	yes
NextLimit **Maxwellrender**	Unbiased	Unbiased	yes	yes	yes	yes	yes	yes	yes (built-in)	yes	yes	yes	yes	yes	yes	yes	yes	yes
Mentalimages **Mentalray**	Biased (raytracing, ra	Raytracing, Photon m	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
Muse	Biased/ **Reyes**	Monte Carlo Ray Trac	yes	no	no	yes	yes	no	yes	yes	yes	yes	yes	yes	yes	no	yes	yes
Pixie	Biased / **Reyes**	photon mapping and i	yes	no	no	yes	yes		yes	yes	yes	yes	yes	yes	yes	no	yes	yes
Pixar **Renderman**	Biased / **Reyes**		yes	yes	yes	yes	yes	no	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
RenderSpud (developement)	Biased&Unbiased	Metropolis light transp	yes	yes	yes	yes	yes	no	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
Thea Render	Biased	Photon Mapping,Irradi	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	no	yes	yes	yes	yes	yes
ChaosGroup **Vray**	Biased	path tracing, irradianc	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes

Valid candidates Feature Chart.

Subsurface scattering phenomena is present in a vast amount of materials in the real world [see Glossary of Terms], and is a technique responsible of te photorealistic appearance of materials rendered in nowaday's render engines.
Therefore, renderers not providing this feature are eliminated. Accurender Ntx and Bunkspeed Shot both do not offer sss at this point of the table state. They do not offer motion blur nor ambient occlussion, so are good candidates to go out of the table.

The same applies for Glare Technologies Indigo renderer, whose documentation does not refer to sub-pixel displacement [see Glossary of Terms section]. It also does not offer normal mapping, a common rendering technique available in most production render engines. [see Glossary of Terms section].

Next, render engines without volumetric lightning [see Glossary of Terms] are eliminated. This important feature would be covered in the comparison of the final render engines, so renderers that do not meet this criteria should be eliminated from our valid candidates.

Render Engine	**Biased/Unbiased**[1]	**GI algorithm**	**Caustics**[2]	**Volume Caustics**[3]	**Volume Shadows**[3]	**GI**[4]	**IBL**[5]	**IES / Elumdat**[6]	**Ambient Occlusion**[7]	**Bump Mapping**[8]	**Normal Mapping**[9]	**real SSS**[10]	**Motion Blur**[14]	**Depth of field**[11]	**Sub-pixel Displacement**[12]	**Passes**[13]	**Multithread**	**64-bit**
Dna Research **3Delight**	Biased/ **Reyes**	Raytracing	yes	yes	yes	yes	yes	no	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
RandomControl **Fryrender**	Unbiased	Unbiased	yes	yes	yes	yes	yes	yes	yes (built-in)	yes	yes	yes	yes	yes	yes	yes	yes	yes
NextLimit **Maxwellrender**	Unbiased	Unbiased	yes	yes	yes	yes	yes	yes	yes (built-in)	yes	yes	yes	yes	yes	yes	yes	yes	yes
Mentalimages **Mentalray**	Biased (raytracing, ra	Raytracing, Photon m	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
Pixar **Renderman**	Biased / **Reyes**		yes	yes	yes	yes	yes	no	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
RenderSpud (developement)	Biased&Unbiased	Metropolis light transp	yes	yes	yes	yes	yes	no	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
Thea Render	Biased	Photon Mapping,Irradi	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	no	yes	yes	yes	yes	yes
ChaosGroup **Vray**	Biased	path tracing, irradianc	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes

Filtered Feature Chart

Thea Render do not offer Motion Blur [see Glossary of Terms]. It is an important feature in production photorealistic rendering. That render engine offers good support for other features, but this specific technique will be compared in depth in this study. That said, Thea render won't be further present in the table and won't be considered for the more in-depth study later in the following pages.

IES profile support [See Glossary of Terms section] is important when rendering complex sorts of light sources, but it does not provide any sort of performance effect in the rendered image in terms of render time. It is not a crucial feature, and both renderers that provide support for it and those who do not, should be considered at this point of the table state.

Finally, RenderSpud is deleted from the table as it is under developement at the moment.
It looks promising, as it will cover all photorealistic render techniques shown in this paper, and it will benefit from biased and unbiased algorithms depending on the rendering situation. Unfortunately, it is not available yet.

The candidate's table stays as follows and contains most of the render engines that will be used for the comparison in this document.

Render Engine	**Biased/Unbiased**[1]	**GI algorithm**	**Caustics**[2]	**Volume Caustics**[3]	**Volume Shadows**[3]	**GI**[4]	**IBL**[5]	**IES / Elumdat**[6]	**Ambient Occlusion**[7]	**Bump Mapping**[8]	**Normal Mapping**[9]	**real SSS**[10]	**Motion Blur**[14]	**Depth of field**[11]	**Sub-pixel Displacement**[12]	**Passes**[13]	**Multithread**	**64-bit**
Dna Research **3Delight**	Biased/ **Reyes**	Raytracing	yes	yes	yes	yes	yes	no	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
RandomControl **Fryrender**	Unbiased	Unbiased	yes	yes	yes	yes	yes	yes	yes (built-in)	yes	yes	yes	yes	yes	yes	yes	yes	yes
NextLimit **Maxwellrender**	Unbiased	Unbiased	yes	yes	yes	yes	yes	yes	yes (built-in)	yes	yes	yes	yes	yes	yes	yes	yes	yes
Mentalimages **Mentalray**	Biased (raytracing, ra	Raytracing, Photon m	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
Pixar **Renderman**	Biased / **Reyes**		yes	yes	yes	yes	yes	no	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes
ChaosGroup **Vray**	Biased	path tracing, irradianc	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes	yes

Final Candidates Feature Chart.

From this final list, it is necessary to point out some other aspects like license pricing and official support, as well as other extra specific features.
Taking in consideration the information shown in the table below, a final renderers list will be built containing the render engines that are being used in this comparison study.

Render Engine	**Price**	**Support**	**Renderman Compliant**	**Other features**
Dna Research **3Delight**	2000$	450$	yes	Pointbased GI, Stereo Rendering, Procedural geometry,public SDK
RandomControl **Fryrender**	1190$	n/a	no	Interactive Render preview, SDK not public
NextLimit **Maxwellrender**	995$	n/a	no	Interactive Render preview, multilight system, public SDK
Mentalimages **Mentalray**	Host application.	n/a	no	Interactive Render preview, public SDK, progressive IBL rendering, fast blur rasterization, MetaSL
Pixar **Renderman**	3500$	700$	yes	Interactive Render preview via custom node, public SDK, deep shadows, point based colour.
ChaosGroup **Vray**	999$	n/a	no	Interactive Render preview via VrayRT, public SDK, vrayFur.

Final Candidates extra characteristics Chart.

From this table we could divide render engines in three groups:

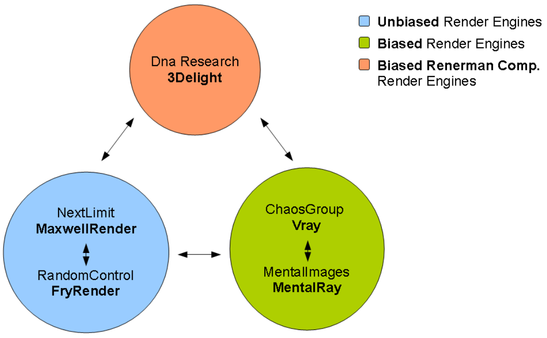
-Biased Renderman Compliant (3Delight & Renderman)
-Biased non Renderman Compliant (Vray & MentalRay)
-Unbiased (MaxwellRender & FryRender)

Focusing in the first group, a big difference of price is noticeable regarding the other render engines. Between both of them, Pixar Renderman renderer is almost 1300$ more expensive and offer quite the same features than 3Delight renderer. They are both Reyes core architectured, and full Renderman Shading Language compliant. Despite its price, choosing a Reyes based render engine is a good idea in this study, because it offers great benefits in some areas compared with raytracing based render engines, as will be discussed later in this study.
Staying with both Reyes renderers for the comparison won't be necessary for the purposes of this paper, so we will stick up for Dna Research 3Delight so it is cheaper, meets the minimum technical specifications [See feature Chart section], and offers all Reyes based benefits needed for the comparison.

According to the biased non-Renderman compliant group, both Vray and Mentalray renderers pass the minimum technical specifications requirements.
In terms of pricing, MentalRay is the production built-in renderer offered by Autodesk with Maya 2011 so no extra licensing is needed for this render engine.
Biased non-renderman compliant render engines are the most common types of renderers in the market, [see feature Chart section] so it is a good idea to leave both biased renderers for the comparison.
That done, differences between two biased non renderman compliant render engines could be observed throughout each rendering technique in this document.

Finally, both unbiased render engines offer quite the same characteristics and licensing price is almost equivalent between them.
It will be a good idea, however, to compare side to side two unbiased render engines, as we will be doing with biased non-renderman compliant renderers.
That said, we will stick up for both of them for the comparison.

The following image summarizes the final renderer's list in a schematic form.



Final Renderers Diagram

Summary.

Once evaluated the features for each of the most relevant render engines currently available and according to the criteria exposed in the above sections [see Election Criteria section], the final list of render engines that are being used in this study stays as follows.

Biased Render Engines.

MentalImages **MentalRay** *Version 3.8.1.33, Aug 20 2010, revision 124793.*
ChaosGroup **V-Ray** *Core Version 1.90.03 (V-Ray for Maya ver. 1.50.SP1 Jun 23 2010).*
Dna Research **3Delight** for Maya *version 9.0.84 x64.*

Unbiased Render Engines.

NextLimit **MaxwellRender** *v2.5.1 x64 (MaxwellRender for Maya ver. 2.5.11).*
RandomControl **FryRender** *v1.5 (FryRender for Maya ver. 1.5).*

Renderers Technical information.

The following pages show an overview of technical information for the final render engines being used in this paper, result of the analysis and conclusions drawn in the last previous sections.
For each renderer, the technical specifications follow the same schematic shown below, and could be consulted in further detail in each renderer's official documentation [see References in this section].

0. Overview
1. Rendering

    1.1 Algorithms
    1.2 Shading and Color
    1.3 Shadows
    1.4 Textures

2. Geometry
3. Performance
4. Integration

Please note that the technical specification documents shown here are written by the companies that produce these software packages, and therefore show clear signs of commercial purposes inside the overview section.

Biased Render Engines Technical Specification.

MentalImages MentalRay *v3.8.1.33*. *[1]*

0. Overview

Mental ray is founded on a network transparent scene and rendering database. Scene data can be provided as files using the .mi scene description language, in either ASCII or mixed ASCII/binary form. The supported geometric primitives include triangle and arbitrary polygon meshes, trimmed free-form surfaces, hierarchical subdivision surfaces and fast subdivision meshes, and specialized hair geometry. User-supplied functions, typically written in MetaSL® or C/C++ programming language, allow to procedurally create textures, materials, new lighting models, volume and lens effects, as well as geometry and displacement. They can also be used to control global illumination simulation through photon shaders and the Photon Map™. The free-form surface processing and approximation capabilities, and the mathematical accuracy of mental ray are superior to those of competing rendering software products.

Mental ray comes with a number of standard shader packages, many of them in source code. They range from shaders for basic illumination tasks to comprehensive and optimized packages for common applications, like shaders for typical materials used in architecture and design, subsurface scattering and car paint shaders, as well as physical sun and sky lighting.

1. Rendering

1.1 Algorithms

- **Ray tracing** architecture for automatic, accurate, and general simulation of reflections, refractions, shadows, and complex illumination.

- **Rasterizer** for fast, high quality rendering of directly visible objects, with efficient support for motion blur and displacement mapping, and with separate control of shading quality and visibility anti-aliasing

- **Scanline** rendering method for fast rendering of scenes in low to medium complexity

- Physically correct simulation of G**lobal Illumination**
    - **Photon mapping** combines forward and backward ray tracing and simulates all possible light paths, without the need to prepare and combine each effect separately
    - **Final gathering** for efficient and easy to use computation of one or more diffuse indirect light bounces, optionally combined with photon mapping
    - C**austics** computation using dedicated photon maps (light patterns caused by reflected or refracted light)
    - Interaction with participating media: V**olume caustics** and multiple volume scattering in diffuse media like fog (e.g. light beams in fog)

- Use of **graphics hardware (GPUs)** for high quality rendering, with support for:
  - Cg shaders
  - Phenomena
  - order independent transparency
  - motion blur

- **Anti-aliasing**
  - **Strictly deterministic low-discrepancy sampling** ensures faster convergence to the correct solution. Algorithms are completely deterministic (no animation flickering due to randomness; re-rendering a frame produces exactly the same result).
  - **Automatic anti-aliasing** of all features such as transparency, refractions, reflections, shadows, area lights, textures, global illumination, caustics, volume caustics, and multiple volume scattering
  - **Adaptive recursive oversampling** and under-sampling (fewer than one sample per pixel taken initially, automatically refined where necessary)
  - Strictly deterministic low-discrepancy **jittering** to avoid "stair casing" at sharp edges

- **Motion blur**, **Depth of field**
  - Object motion blur by motion transformation matrices
  - Motion deformations by per-vertex motion vectors
  - Multi-segment vertex motion blur for curved motion blur
  - Full motion blur of reflections, refractions, light sources, shadows, global illumination, and caustics
  - Depth of field implemented in lens shaders
  - Motion and depth output for post-processing motion blur and depth of field

- **Light mapping:** baking of illumination into texture maps, per vertex and per texel
- **Multipass rendering** for compositing of layers with sub-pixel accuracy, collecting sample data from multiple render passes, including user-defined data. Pass data can be merged into the final image with a custom merging function.

1.2 Shading and Color

- Programmable shaders
  - Shading language: **C and C++**; any existing development environments (compilers and debuggers) including their optimization features can be used
    Shaders have full **access to the scene graph**, and may extend the scene graph at runtime (for example, to build procedural geometry or perform on-demand creation of geometry)

- **Shade trees** and shader graphs: shaders can be linked into graphs such that one or more outputs of a shader feeds one or more inputs of another shader
- Encapsulated cooperating **shader graphs** (Phenomena) that package one or more shader graphs and scene insertion points into an easy-to-use "black box" with simple external parameters, hiding internal complexity without the need to write a program or plug-in
- **Shader types:** material, texture, light, volume, shadow, atmosphere, lens, environment, photon, photon emitter, displacement, output, contour, geometry, inheritance, and state shader
- **Volume shaders** implement non-geometric volume effects such as fire or smoke, in both scanline and full ray tracing modes
- **Ray marching** for volume shading (sampling space iteratively or recursively, for example to render visible light cones)
- Arbitrarily many and **arbitrarily large frame buffers**, stored on disk during rendering to conserve memory
- **Arbitrary frame buffer types**, including color, depth, motion, normals, labels, and coverage
- **HDRI** (high dynamic range imaging) using RGBE, float, and half float data types
- Incremental **image preview** during rendering
- More than 30 **image types** for textures and output, including **OpenEXR**
- Full support for **IES and Eulumdat** light profile data with accurate illumination estimation (light profiles for commercial light fixtures are available from the light vendors to describe accurate lighting parameters of their products.)
- Support for **color spaces** with higher perceptual precision than RGB
- Support for arbitrary **BRDFs**
- **Spectral rendering**, with arbitrarily many samples per color spectrum
- **Contour rendering** for cartoon animation, with fine control over contour placement, overlaps, width, color, transparency, and other user-definable criteria

- Shaders:
    - Extensive base **shader library**
    - **Subsurface scattering**
    - Glossy reflections & refractions
    - Metallic car paint
    - Physically correct shaders
    - **Spectral rendering** shaders
    - Contour lines
    - Depth of field
    - 2D motion blur
    - **Cg shaders** for hardware accelerated rendering

1.3 Shadows

- **Ray traced shadows** for correct computation of shadows, including transparency
- **Shadow maps** for fast shadowing where the accuracy of ray traced shadows and area light sources is not required
- **Detail shadow maps** for fast, high quality shadowing of hair, partial shadows, and

transparent shadows

- **Area lights** create shadows with variable and accurate softness: spheres, disks, rectangles, cylinders (fluorescent lights), and arbitrarily shaped geometric light objects; no fixed manual blurring of shadow maps required
- Effects such as **colored shadows** cast by stained glass using shadow shader graphs or Phenomena
- **Volume cast shadows** and volume self-shadowing by applying volume shaders to shadows or shadow ray segments

1.4 Textures

Texture **tiling and caching**, loading only small, currently needed portions of textures
More than 30 supported texture **image formats**, including HDR formats
Mipmapping and pyramid textures for fast and smooth texture interpolation
Reflection maps for fast and simple distant environments such as sky cloud images
Memory-mapped textures and memory-mapped pyramid textures (only the pieces of the texture that are actually required are loaded/unloaded on demand from disk to memory)
High quality **ellyptic texture filtering**

2. Geometry

- Free-form surfaces in Bezier, B-spline (including NURBS), Cardinal, Taylor, and arbitrary basis matrix representations; rational and non-rational, up to degree 21
- Any number of trimming curves, hole curves, and special curves and points (that control tessellation) can be applied to free-form surfaces
- Connectivity between surfaces
    - Polygons
- Hierarchical subdivision surfaces with support for:

    - **Quads** (Catmull-Clark scheme)
    - **Triangles** (Loop scheme)
    - Sharp features: **creases and points**
    - Optional **wavelet-based multiresolution editing** and data compression/LOD (mental matter)
    - Optional conversion from and to **NURBS** (mental matter)

- **Trimming**
- **Approximation, controllable** by parametric subdivisions, edge length, analytic distance, and angle
- **Displacement mapping**, including very precise sub-pixel displacement that resolves even the finest detail of the displacement map, without excessive memory consumption and with very easy-to-use quality control settings, using mental ray's data flow mechanism
- Texture surfaces and multiple texture coordinate systems
- **Special hair primitives** for optimized rendering of hair and fur
- **Multiple instancing** of objects and light sources
- **Object and instance flags** to control visibility, shadows, reflections, etc. per object instance: for example, there may be different versions of an object for beauty passes, reflection

passes, and shadow passes, even within a single rendering operation

- **On-demand loading** of object files and on-demand procedural object creation using geometry shaders and placeholders: geometry is not created until it is needed, the geometry cache maintains only objects currently or recently in use to reduce rendering times and concurrent memory requirements

3. Performance

- Full data flow architecture which maintains a scene/data cache to store only data which is currently needed, creating data on demand wherever possible. This enables mental ray to render scenes with great complexity with limited amounts of available memory.
- Multi-threading parallelism for scalable exploitation of multi-processor machines
- Network parallel rendering for scalable, efficient rendering on heterogeneous networks of machines regardless of operating system, byte order, and word size (32 and 64 bits) differences, using an adaptive load-balancing algorithm for tessellation, rendering, and other computations
- Rasterizer for fast, high quality rendering of directly visible objects
- BSP tree (kd-tree) acceleration method for fast ray tracing, with memory caching for very complex scenes
- Scanline rendering method for fast rendering of scenes in low to medium complexity
- Incremental changes for animation and interactive rendering, defining, transmitting, and tessellating only the differences between frames
- No-master option in network parallel rendering, relieving the master host from performing rendering computations, and helping to reduce memory consumption on the master host
- Full support for 64 bit architectures (see supported platforms)
- Full support for multi-core processors with close to 100% scaling efficiency
- Satellites for network parallel rendering in many OEM applications, using a number of machines to increase rendering performance without requiring additional licenses

4. Integration

- Optional developer library form available for integration into OEM products
- Translators and plugins are available for a variety of front-end applications
- C language API for entire scene description language and other features, such as API calls for support of parallelism in shaders
- Simple, efficient, and full-featured hierarchical scene description language (.mi), fully text-based with optional binary vector data for increased performance
- Material and parameter inheritance
- OEM specific software protection and licensing technology

Chaos Group Vray core v1.90.03. *[2]*

0. Overview

Chaos Group provides state of the art rendering solutions for the Visual FX, Film, Media and Entertainment, Architecture, Automotive design, Product design, Television, and other industries. The V-Ray rendering engine was acknowledged for its ability to deliver high quality photorealistic images in the shortest time possible. This state of the art ray-tracing technology is capable of reducing costs in every production pipeline due to the doubled rendering power and saving in terms of human, time and CPU resources.

Now, V-Ray for Maya enables the rendering of even larger scenes with greater complexity and artists can rely on a faster rendering process. The comprehensive list of features provided by V-Ray for Maya includes true 3D Motion Blur, Sun & Sky procedural lighting system, Physical camera for matching live footage, Environment Fog, a set of Sub-Surface Scattering shaders and many others. With official release of V-Ray for Maya in August 2009 V-Ray rendering solutions confirmed their image of indispensable tool that is always there help artists transform even the most ambitious creative visions into reality for both less time and lower cost without sacrificing quality.

1. Rendering

1.1 Algorithms

- Physically accurate full G**lobal Illumination** algorithms:
  - **path tracing**
  - **irradiance cache**
  - **photon maps**
  - **light cache**

- Reusable GI solutions for accelerated rendering of walk-through animations and animations with dynamic objects.
- Physically accurate **area lights**
- Efficient illumination from **HDR** environments
- Procedural **sun & sky** models
- Extensible with custom lights through the V-Ray SDK

1.2 Shading and Color

- Efficient shading system specifically optimized for raytracing.
- Physically plausible materials
- Blurry reflections/refractions
- Accurate hilights

- Set of fast **Sub-surface scattering** shaders
- Support for efficient material layering
- Extensible with custom shaders through the V-Ray SDK
- Physically accurate area lights
- Efficient illumination from HDR environments

- Procedural sun & sky models
- Extensible with custom lights through the V-Ray SDK

- Depth-of-field with bokeh effects
- Accurate motion blur
- Physical camera model
- Custom cameras through the V-Ray SDK

1.3 Shadows

- Different GI algorithms: path tracing, irradiance cache, photon maps, light cache
- Reusable GI solutions for accelerated rendering of walk-through animations and animations with dynamic objects
- Physically accurate area lights
- Efficient illumination from HDR environments
- Procedural sun & sky models

1.4 Textures

- Three different image sampling methods
- Full-scene anti-aliasing
- Progressive path tracing
- Support for additional render elements (diffuse, reflection, GI, etc)
- Extended matte/shadow capabilities
- Texture baking of any render element (GI, lighting, etc)

2. Geometry

- Efficient geometry handling
- True **instance** rendering
- On-demand dynamic geometry creation (.vrmesh files, converter for .OBJ, .PLY, .GEO files included)
- On-demand geometry loading from disk files
- **Displacement mapping**
- Catmull-Clark and Loop subdivision surfaces
- Extensible with custom geometric primitives through the V-Ray SDK
- Particle rendering & particle istancer

3. Performance

- Vray Technical Feature's Document do not specify performance information for this render engine.
  However, there are some aspects that could be considered after testing this software.
    - Multi-threading parallelism for scalable exploitation of multi-processor machines
    - Full support for 64 bit architectures.
    - Full support for multi-core processors with close to 100% scaling efficiency

4. Integration

- V-Ray specific frame buffer with integrated color corrections and display of multiple rendering elements
- Direct rendering to disk for extremely large images, either as OpenEXR files or as .vrimg files.
- All features accessible from within Maya (no external applications required) Standalone version included (10 licenses)
- Support for many of the standard Maya shaders, lights, materials, procedural and utility textures

DNA Research 3Delight v*9.0.84*. *[3]*

0. Overview

High quality, high performance rendering software and tools based on the industry proven RenderMan® standard and practices. **3Delight** has been used in numerous productions and is recognized as a benchmark in rendering technology.

3Delight is our most complete **RenderMan-compliant** suite of tools for the most demanding production environments. Suitable for experienced users and larger studios looking for the utmost flexibility.

3Delight for Maya is the most powerful RenderMan-compliant rendering solution for **Autodesk® Maya®**. Easy to use by the independent artists while totally flexible to support the most elaborate rendering pipe-line.

1. Rendering

1.1 Algorithms

- **REYES** Rendering Algorithm

  The core rendering algorithm in 3delight is reyes based. This algorithm has proven its worth during more than two decades of outstanding quality pictures.
  Images produced using this technique have a feel that is difficult to produce using ray traced based solutions.

- Two Render Modes

  Both direct rendering in *Maya* (including rendering into *Maya*'s render view) and RIB export are supported. Direct rendering is useful in normal lighting and rendering work while RIB export can be used for render farm rendering. RIBs exported by *3Delight For Maya* are optimized for size and can be written in compressed binary form.

- HyperShade and RenderMan Shaders Support

  Both RenderMan shaders and *Maya*'s HyperShade nodes can be assigned to objects. HyperShade nodes are automatically converted into *human-readable* SL code and compiled for rendering. New HyperShade nodes can easily be added by providing *3Delight For Maya* with their SL code.

- **Motion Blur and Depth of Field**

  Multi-Segment motion blur and realistic camera shutter simulation contribute to high quality rendered images. Compared to other rendering software, motion blur in *3Delight For Maya* is *fast*. Depth of field is fully supported and simulates a realistic camera bokeh.

- High Quality Anti-Aliasing

Edge anti-aliasing, motion blur and depth of field quality are all controlled using very simple and *predictable* controls. **"Pixel Samples", "Pixel Filter"** and **"Filter Width"** are the most common parameters one needs to know. Contrary to other rendering packages, such as ray tracers, increasing pixel samples for higher quality anti-aliasing does not affect performance significantly.

- Ray Tracing
  Reflections and refractions are accurately rendered.
- Global Illumination
- Photon maps, caustics, final gathering and image based lighting are supported.

1.2 Shading and Color

RenderMan® Shading Language Support

3Delight offers programmable shading and lighting using the standard RenderMan® Shading Language. Surface, displacement, light, volume and imager shaders are fully supported. Matrices, arrays, normals, vectors and all the standard shadeops are supported. Shader interrogation from within other shaders is also supported (message passing) as well as output variables. The language can be extended using DSO shadeops, providing the increased flexibility of C/C++ programming. *3Delight*'s optimizing compiler can either produce compiled object-code (for better performance) or byte-code (for better portability). In both cases, run-time execution is performed in an SIMD manner for best performance. See User's Manual section 3.2 Using the shader compiler - shaderdl.

- **Subsurface Scattering**
  Automatic subsurface scattering can be enabled on a per-object basis and delivers impressive results, fast.

1.3 Shadows

Both ray tracing and shadow maps can be used to render shadows. Shadow maps can be rendered in "deep" mode for realistic shadows in hair and fur. Additionally, *3Delight For Maya* can automatically generate "cube shadow maps" for point lights. Contrary to other rendering packages, cube shadow maps are not stored as six separate shadow maps but in one special shadow map.

1.4 Textures

- Complete File Format Support
  - 3delight has support for most common image file formats:

    Input TIFF, OpenEXR, JPEG, PSD (PhotoShop files), IFF, PIC, SGI, PIC (SoftImage files), GIF and Radiance.
    Output TIFF, PSD, OpenEXR, Cineon, IFF, PNG, Radiance and '.tdl' texture files.

- **Output Formats**

  3Delight For Maya can save rendered images in many file formats, including: TIFF, IFF, OpenEXR, cineon, bmp, sgi, softimage and PSD.

- **Input Formats**

  Maya's textures used in **HyperShade** shaders are automatically converted to 3Delight ".tdl" textures. When using RenderMan shaders, **3Delight**'s texture converter (*tdlmake*) can be used to convert the most common images formats to ".tdl" textures.

- **HDR Images**

  tdlmake can convert high dynamic range probes (both normal and "twofish" probes) images into environment maps suitable for image based lighting.

2. Geometry

- **Geometry Support.**

  3Delight supports a complete set of **RenderMan® geometry**: subdivision surfaces, polygons, patches (B-spline, Bezier, Catmull-Rom and others), NURBS (with trim-curves), Curves (for Fur & Hair), quadrics and procedural geometry. Also, user-defined variables, including vertex and facevarying variables, attached to geometry are fully supported.

- **Supported Geometry**

  - *Maya* Hair and Paint Effects

    Rendered using 3Delight's efficient `RiNuCurves' primitive.

  - Polygons

    All types of polygons, including those with holes, are supported. Additionally, polygonal geometry can be tagged as a subdivision surface and rendered as such.

  - NURBS (*Maya* surfaces)

    All NURB surfaces are supported. Trim curves on surfaces are also fully supported and rendered to sub-pixel accuracy. All surfaces are rendered **smooth**.

  - Hierarchical Subdivision Surfaces

    Fully supported, including both creasing and partial creasing on edges and vertices as well as per-level UV sets. As always, all hierarchical subdivisions are rendered smooth and to sub-pixel accuracy.

  - Particles

    "Point", "blobby", "sphere" and "patch" particles are supported. "Point" particles are rendered using 3Delight's efficient **lightweight particle** primitive.

  - Curves

    Maya curves can be tagged as renderable on a per object basis and rendered using 3Delight's RiNuCurves primitive.

  - Additionally, UV sets, texture reference objects and normals (if any) are correctly assigned to all primitives.

  - Geometric Displacements

- Displacements are efficiently rendered to sub-pixel accuracy. Hypershade displacement shaders as well as RenderMan shaders can be assigned to geometry.

3. Performance

- **Multi-threading** and **Multi-processing.**

  The renderer is fully multi-threaded and will take all the available processors/ core to render images. Additionally, processes can be started on remote machines without additional task scheduling software.

4. Integration

- As with all serious production tools, *3Delight For Maya* has been designed with flexibility in mind. In fact, no other rendering plug-in on the market achieves the same balance between integration and configurability.

1. We implemented most of the RenderMan interface in MEL, which means that you can call RenderMan commands in your MEL scripts. Other packages mainly provide "RIB boxing" capability.

2. 3Delight For Maya was written mostly using the MEL RenderMan interface which means that one can modify it at will. Only core functionalities have been programmed in C++ for performance reasons.

- **Import and Export**
  - Output Formats

    3Delight For Maya can save rendered images in many file formats, including: TIFF, IFF, OpenEXR, cineon, bmp, sgi, softimage and PSD.

  - Input Formats

    Maya's textures used in **HyperShade** shaders are automatically converted to 3Delight ".tdl" textures. When using RenderMan shaders, **3Delight**'s texture converter (*tdlmake*) can be used to convert the most common images formats to ".tdl" textures.

  - HDR Images

    tdlmake can convert high dynamic range probes (both normal and "twofish" probes) images into environment maps suitable for image based lighting.

- 3Delight Plug-ins
  - **Maya 3Delight** comes with 3Delight For Maya, a full featured plug-in that uses 3delight as a renderer.
  - **Softimage 3Delight** For Softimage, a full featured plug-in that uses 3delight as a renderer, is available.

  - **Massive1 3Delight** is supported by Massive “out of the box”.
  - **Houdini2 3Delight** is supported by Houdini “out of the box”.

Unbiased Render Engines Technical Specification.

NextLimit MaxwellRender v2.5 *[4]*

0. Overview

Maxwell Render™ is a rendering engine based on the mathematical equations governing light transport, meaning that all elements, such as emitters, materials and cameras, are derived from physically accurate models. Maxwell Render is unbiased, so no tricks are used to calculate the lighting solution in every pixel of a scene; the result will always be a correct solution, as it would be in the real world. Maxwell Render can fully capture all light interactions between all elements in a scene, and all lighting calculations are performed using spectral information and high dynamic range data.

Due to its very nature, Maxwell Render enables users to create accurate and extremely realistic images. Maxwell Render is a recognized standard in architectural visualization, product design, jewelry, film production, scientific research and other high-end rendering markets, and the leader in render quality.

Maxwell Render is a rendering engine that accepts models and scenes created in 3D or CAD applications. Several of these applications are directly supported through a Maxwell Render plug-in; others can be used in conjunction with Maxwell Render by importing the geometry into Maxwell Studio, a component of the software.

1. Rendering

1.1 Algorithms

Physically correct
Unbiased
Maxwell Fire

- Fast Interactive Rendering
- Preview scene changes in seconds
- Integrated in Plug-ins and Maxwell Studio
- Multilight™
- Real time interactive emitters, Sky and HDRIs with key frame
- Animated intensities and colors

Camera Features

SimuLens™
- Diffraction
- Lens Scattering
- Vignetting

Shift Lens
Polygonal/ Circular diaphragm
Shutter speed
Focal length
F-Stop
Depth of Field (DOF)
Auto exposure
Z-clip planes

1.2 Shading and Color

Material Features

Stacked Layers

- Stack layers on top of each other
- Create extremely complex materials
- Layer blending (Normal, Additive)
- Weight mapping

Subsurface Scattering
ThinSSS
Fresnel simulation
Spectral dispersion

Advanced absorption and transmittance

- With attenuation distance
- Mapped (smooth, color)
- Clipped (explicit, black & white)

Advanced anisotropy
Matte material
Bump and normal mapping
Complex IOR files
Coating layers

- Thin film interference/ iridescence
- Thickness mapping

Emission layers (composite emitters)
Advanced Physical Sky system

- Aerosols
- Pre-sets
- Real-life parameters
- Sky to HDRI conversion
- Multilight™ controllable

Sky Dome
HDR/ MXI environment system

- Multi-channel control
- Multilight™ controllable

Advanced illumination system

- Color + Luminance emitters
- Temperature emitters (Kelvin)
- HDR/MXI emitters
- Luminary pre-sets
- Real-life units
- IES and EULUMDAT support
  Multilight™ controllable

1.3 Shadows

Unbiased Physically correct shadow calculation.

1.4 Textures

Color spaces

- sRGB, Adobe98, Apple, PAL, NTSC

Render layers

32 Bits output **EXR, TIFF, HDR, MXI**

2. Geometry

Export/Import extensions.
Obj, Stl, Lwo, Nf, Xc2, Mxs, Dxf, 3Ds, Xml, Fbx, Ply, Dae

3. Performance

Resume render
Render channels
Instances
Luminary pre-sets
Real-life units
Physically correct
Unbiased
Maxwell Fire

- Fast Interactive Rendering
- Preview scene changes in seconds
- Integrated in Plug-ins and Maxwell Studio

Multilight™

- Real time interactive emitters, Sky and HDRIs with key frame.
  Animated intensities and colors

4. Integration

Platform Support.
3D Application plug-ins

- Autodesk 3ds Max
- Graphisoft Archicad
- Maxon Cinema4D
- Form Z
- Lightwave 3D
- Autodesk Maya
- Modo
- Rhinoceros
- Google Sketchup Pro
- SolidWorks
- Autodesk Softimage

Networking Features

- Hybrid network support
  - Windows 32, Windows 64, Mac OSX and Linux 64.
- Full monitoring
- Auto-detect RenderNodes
- Cooperative/ Animated/ Single job modes
- Batch rendering/ Render queue
- Priority control

Hierarchy support
Flexible and customizable UI
Real time OpenGL previews

- Sky/Sky Dome
- Environment
- DOF indicator

Real time compass/ sun position
Material Editor
UV mapping tools
Export OBJ option
Material wizards
Material browser
MXM gallery integration
Orthographic rendering

RandomControl FryRender v1.5 *[5]*

0. Overview

Fryrender is the physically-based light simulator developed by RandomControl.
It is a photo-realistic render engine where all the elements involved in the generation of the final image (materials, lights and cameras) are based on physically accurate models.

On fryrender, when you hit the render button, a simulation of the real behavior of light happens inside. Because of this, it is possible to achieve hyper-realistic results with minimal effort.

1. Rendering

1.1 Algorithms

Spectral **Unbiased** Rendering

- All internal computations use spectral data instead of RGB data.
- Automatic RGB-Spectrum conversion (the user doesn't need to deal with spectrums directly).
- Unbiased, physically-based and accurate light simulation.
- All light components converge to the final image progressively.

Physical Lighting

- Automatic Full Global Illumination.
- Physical Sun & sky.
- HDRI environment.
- Image-Based Lighting.
- Mesh emitters.

Camera Simulation

- True optics and Reflex camera simulation.
- Focal length.
- Diaphragm aperture (f-stop).
- Shutter-speed.
- ISO film.
- Focal distance and auto-focus.
- Camera clipping (z-clip).
- Film shift.
- Camera motion blur.
- Object motion blur.
- Advanced lens effects.
- Lens distortion.
- Panoramic cameras.
- Orthogonal cameras.

1.2 Shading and Color

Physical Materials

- Advanced material creation system.
- Global or mapped roughness.
- Global or mapped anisotropy.
- Fresnel.
- Custom IOR.
- Dielectric absorption.
- Opacity mapping.
- Dispersion.
- Bump / Normal mapping.
- Sub-Surface Scattering.
- Single-Sheet SSS.
- Plastic or thin-film interference coatings.

Physical Atmosphere

- Physically-based Sun & sky simulation.
- Time-step day-cycle animations.
- Accurate geo-location and celestial body positioning simulation.

1.3 Shadows

Spectral Unbiased Rendering

- All internal computations use spectral data instead of RGB data.
- Automatic RGB-Spectrum conversion (the user doesn't need to deal with spectrums directly).
- Unbiased, physically-based and accurate light simulation.
- All light components converge to the final image progressively.

1.4 Textures

Map Editor

- .JPG, .BMP, .PNG, .TGA, .TIF, .HDR.
- Support for basic handy procedurals.
- Space warp for HDR maps.
- Per-map UV modifiers.
- Per-map RGB modifiers.

Interactive Tonemapping

- All tonemapping features provide real-time feedback.
- Lens bloom.
- Lens glare.

- Camera vignetting.
- Real Camera Response operators.

Image-Based Lighting

- Easy-to-use and highly efficient **IBL** support.
- High-Dynamic Range input.
- **HDR** imagery warping.

2. Geometry

Run-time Geometry

- Efficient **Micro-Poly Displacement Mapping**.
- Geometry Instancing (supporting translation, rotation and non-uniform scale).

3. Performance

Technology and Performance

- **Multi-processor / Multi-core / Multi-thread.**
- Based on highly optimized and clean code.
- SSE support.
- Distributed computing over the LAN through NetWarrior.
- Built to match up perfectly with other RandomControl applications such as SWAP and Arion.

Network Rendering

- Direct connection with NetWarrior.
- .ZIP Archive with resource gathering.
- .DSI merge for cooperative rendering.

NetWarrior Support

- Co-operative still frame rendering.
- Distributed animation rendering.
- Automatic .DSI merge for still frames.
- Maximized use of network resources.
- Distributed rendering works across all supported platforms (hybrid rendering).
- Visual and user-friendly monitoring through NetMonitor

4. Integration.

Supported Modelling Platforms

- Autodesk 3ds Max.
- Maxon Cinema 4D.
- NewTek LightWave 3D.
- Autodesk Maya.
- McNeel Rhinoceros 3D.
- Google SketchUp.
- SoftImage XSI.
- Luxology Modo.

User Interface

- Powerful yet simple and intuitive GUI.
- Real-time OpenGL viewport.
- Open/Close scenes.
- Render/Resume renders.
- Environment & Atmosphere controls.
- Material editor.
- Map editor.
- Interactive tonemapping controls.
- LightMixer faders.
- One-Click-Away Network Rendering.
- Render Region.
- Exportation to **.OBJ**, **.RCS** and **.FRY**.

LightMixer

- Each light source can be sent to a separate layer.
- Independent power and RGB controls per layer.
- Master power and RGB controls.

Output Image Formats

- Conventional RGB output (8-bpp or 16-bpp).
- Raw High-Dynamic Range (HDR) formats.
- .JPG, .BMP, .PNG, .TGA, .TIF, .HDR.
- Optional Illustration (Toon) Core.
- Compositing capabilities (alpha, matte, depth...).
- OpenEXR and .RPF, with multiple compositing channels.
- Straight QTVR for panoramic cameras.

Multi-Platform Support

- 32-bit and 64-bit binaries are provided for all Operating Systems supported.
- Windows XP, 2000, Vista and 7. (Standalone and render slaves).

References

*[1]	Mentalray Technical Specifications, Features, © mental images GmbH*
http://www.mentalimages.com/products/mental-ray/about-mental-ray/features.html

*[2]	Vray For Maya key features,* Copyright ©2011, Chaos Software
http://www.chaosgroup.com/en/2/vray_maya_features.html

*[3]	3Delight Technical Specification,* © 2006-2011 dna research
http://www.3delight.com/en/index.php?page=3delight_documentation

*[4]	Maxwell Render Product/Features,* © 2011 Next Limit S.L.
http://www.maxwellrender.com/mw2_features.php

*[5]	Fryrender Technical Specifications,*© 2011 RandomControl, SLU.
http://www.randomcontrol.com/fryrender-tech-specs

Render Engines Comparison.

Lineal Workflow.

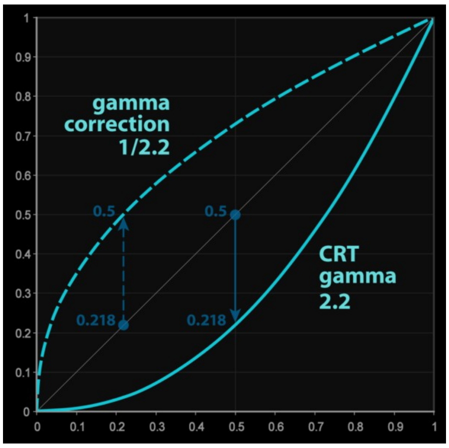
Before making a comparison between render softwares, it is crucial to determine a common lineal workflow to follow. That will give the artist full control of rendered images of a scene regardless of the device in which the rendered image is displayed.

Most of the photorealistic rendering software (and all of which will be studied in this paper) use some sort of lineal data model. In effect, it could be stated that lineal math is applied when a 3D renderer is calculating what the resulting image should be like. However, our display devices do not use lineal math when visualizing this data as images. They use logarithmic math due to the technology used in recording and displaying images. As an example, a lineal calculation of 2 + 2 = 4 done by the render software could be interpreted as 2 + 2 = 10 by the display device.

In order to make the lineal data which generates a synthetic image show on a display device that needs to visualize non-lineal data, we need to convert the lineal data image. This is done by using a display device parameter called gamma,and this process is commonly known as gamma correction.

The gamma value varies depending on the display device but the most common gamma for laptops and computer display devices these days is approximately gamma 2.2, also known as sRGB. That said, it is necessary to gamma correct the lineal image, output of the rendering software calculations, according to the value of gamma of the final display device.

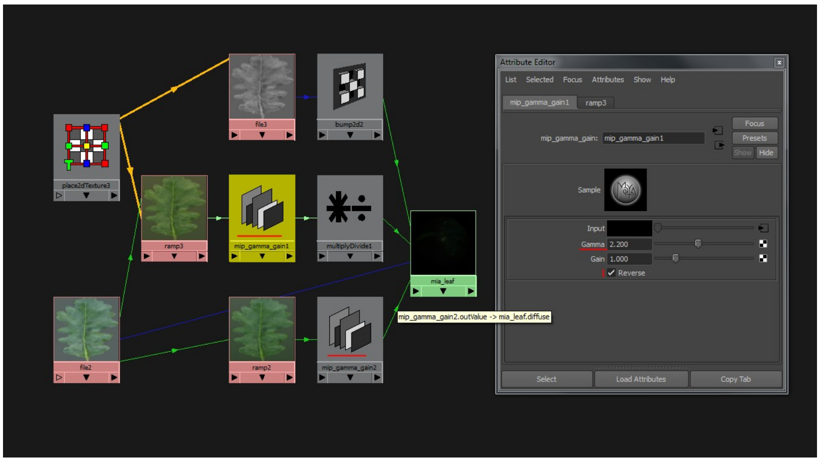
To achieve that, a gamma correction curve is applied and baked into the image file by the rendering or composition software so that the image is shown correctly on the display devices. The gamma correction value for sRGB devices is approximately the inverse of 2.2 (1/2.2 = 0.4545).



Gamma Correction Curve

That said, it's essential to take the following aspects into consideration when shading and lightning a scene using a photorealistic render engine. Please note that this will be applied to all the rendered images shown in this study, regardless of the rendering software that is being used. Accurate comparison between images could not be achieved otherwise.

- **1.** Correct all rgb textures that are not in lineal color space to lineal space (Gamma 0.4545), before applying them to any shader.



mip_gamma_gain: mip_gamma_gain1

Input

Gamma 2.200

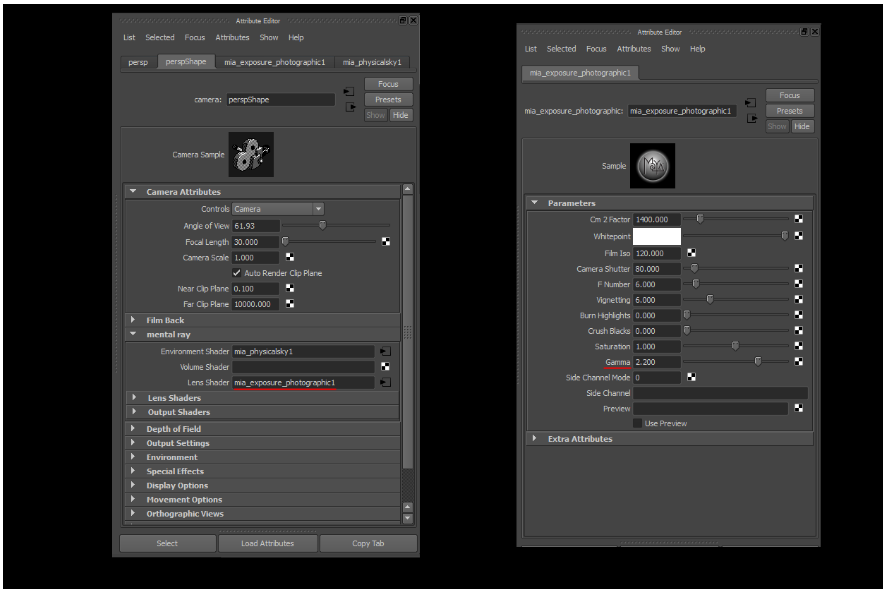
Gain 1.000

☑ Reverse

Texture *Gamma Correction Nodes in MentalRay.*

- **2.** Light and render your scene without color correction.

- **3.** Gamma correct the output image. Output rendered images in this paper will be gamma corrected using the rendering software if not otherwise noted to keep things simple. However, that could be done by using an external post production application such as Fusion or Nuke with greater flexibility.



camera: perspShape

Controls Camera

Angle of View 61.93

Focal Length 30.000

Camera Scale 1.000

☑ Auto Render Clip Plane

Near Clip Plane 0.100

Far Clip Plane 10000.000

Environment Shader mia_physicalsky1

Volume Shader

Lens Shader mia_exposure_photographic1

mia_exposure_photographic: mia_exposure_photographic1

Cm 2 Factor 1400.000

Whitepoint

Film Iso 120.000

Camera Shutter 80.000

F Number 6.000

Vignetting 6.000

Burn Highlights 0.000

Crush Blacks 0.000

Saturation 1.000

Gamma 2.200

Side Channel Mode 0

Side Channel

Preview

☐ Use Preview

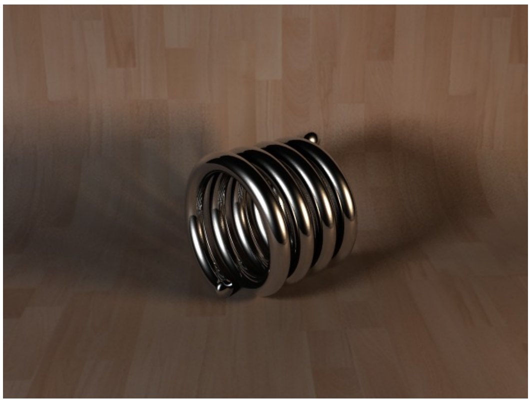
MentalRay Gamma Correcting cammera settings.

As a summary, it could be said that it is essential to keep lineal data in the rendering pipeline and apply gamma correction after the output image is calculated.

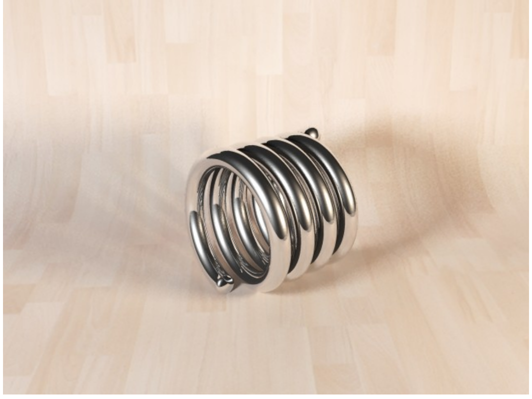
The first problem resides in already gamma corrected textures applied to our shaders. It is necessary to correct those textures to its lineal format before applying them anywhere in the scene. Note that further gamma correction of the entire image should be performed, but already gamma corrected textures will lead in wrong calculations made by the rendering software (which operate using lineal data models exclusively), that will lead to unexpected results.

However, textures that only contain 0-1 luminance values (bump map [See BumpMapping section] or a glossiness map) should not be de-gammed, only images that are providing RGB color information should be corrected. In addition, image files that are already in a lineal color space (floating point images, HDRs [See IBL Image Based lightning section]) won't need that correction.

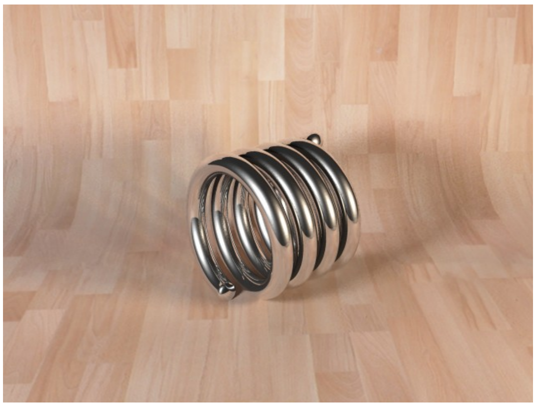
If non gamma-corrected textures are used, we will also prevent the application of two gamma correction curves for that specific texture (one from the gamma corrected texture and another from the rendering software gamma correction) which will make the image look all washed out.



No gamma corrected image (Vray)



Gamma corrected image
with gamma corrected textures
(Vray)



Correct Image. Gamma corrected
output without gamma corrected textures
(Vray)

Moreover, if any additional operations were to be applied to this image (using a second graphics manipulation software such as Photoshop or a compositing package), we do not have the original and lineal pixel values stored in the file any more – which we need if we intend to process the image further – and in effect, we are working with a compromised image. Instead, by keeping all images in the graphics pipeline stored in lineal space and always rendering with gamma correction at 1.0, will solve the issue.

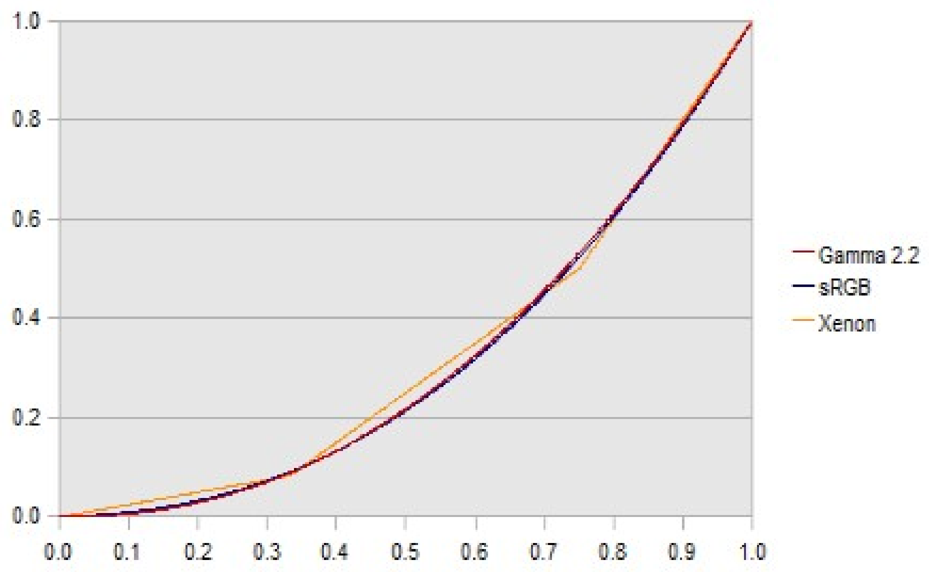
Anyway, the aim of this paper is comparing different render engines, so no post processed images will be shown here.

Keeping in mind that lineal data in the rendering pipeline should be preserved will ensure that internal rendering mathematical calculations are being correctly made.

Image Error.

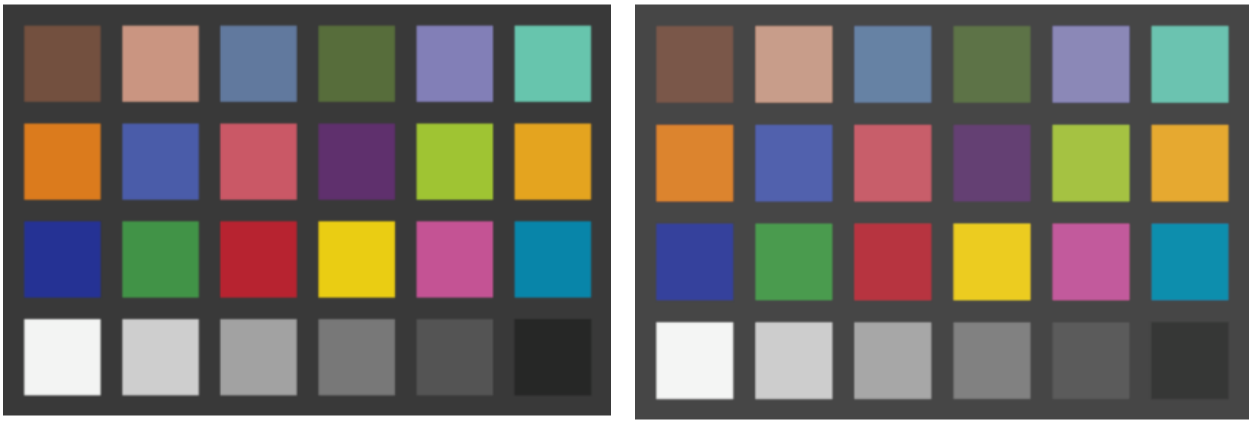
Correctly evaluating image error in rendered images is mandatory for a good comparison between render engines. There are some aspects, however, that must be taken into consideration before writting any error metrics, beacause thay can be deceiving if not calculated properly.

To ilustrate this problem, we will take a graph of the gamma 2.2 curve [see Lineal Workflow section] compared to the sRGB curve and the Xenon gamma curve (Gamma curve of Microsoft's Xbox 360). *[1]*



Gamma graph showing Xenon gamma curve

We want to determine wether the Xenon gamma curve is a good aproximation (error is minimum) of the sRGB curve. The images below show two color checkers with both sRGB (leften image) and Xenon (righten image) curves applied to them side-by-side.



sRGB vs Xenon gamma curves applied to a color checker

Both images look close enough to be hard to determine if they are different images or far enough for the righten image to be a bad aproximation of the leften one.

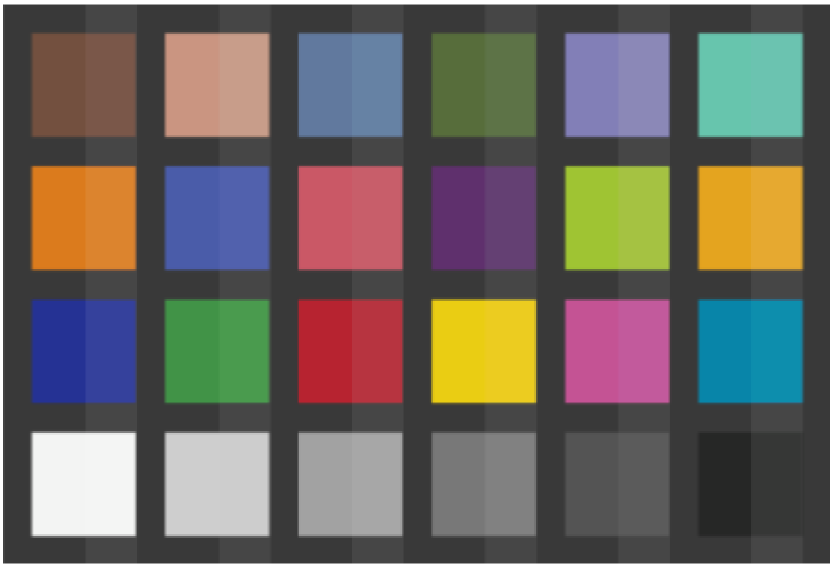
Then, for each x value, we take the difference between the two curves, add them all up, and divide by the total number of x values. Then you divide by the maximum value of the curve, which in this case is 1. That is the absolute average error calculation method.

Once the calculations are made, the final result is 1.62%.
Since the average error is actually only 1.62%, the Xenon curve could be deceivingly assumed to be clearly good enough.

Furthermore, let us calculate the worst error. The absolute error between the curves peaks is 3.71%.

Based on these observations, one would assume that the Xenon is a good approximation of the sRGB curve. Average error is 1.62% and worst error 3.71%

Unfortunately, this approach is very misleading.
The bellow image shows both images interleaved with each other so that the difference can be easily noticed.



Interleaved image of color checkers.

(sRGB vs Xenon curves).

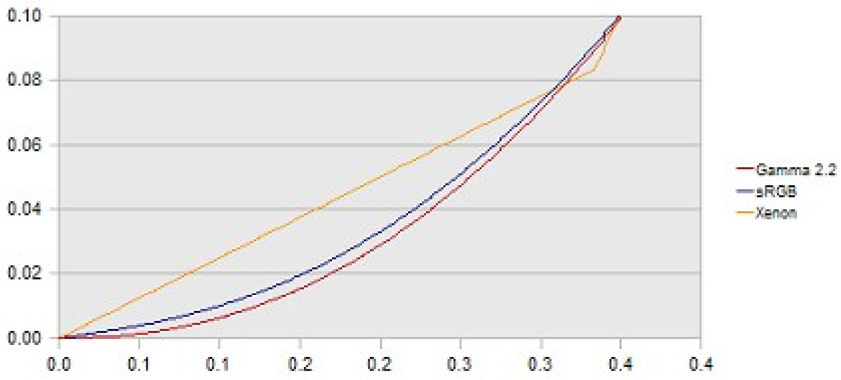
At first glance, it does not seem that the Xenon curve is a valid approximation of the sRGB curve. Therefore, the above observations seem to fail to give a correct conclussion. The cause of this is all about the absolute average error, and must be avoided in these cases as it is being explained in the following paragraphs..

In fact, it is necessary to use an error metric that takes into consideration the logarithmic nature of electronic displays [see Lineal Workflow section]. The absolute error calculations must then be replaced by relative error metrics.

For each curve's x values, the difference between the two values is taken, and then divided by the reference value.Doing that for all x values, and then dividing by the number of values give you the relative error.

With the new calculations, the total relative error between gamma curves is 37.37%, much higher than the percentage of 1.62% result of applying absolute error metrics.

At the bottom end of the curve, values often have 223% relative error. The error in the bottom end of the curve is therefore being heavily under-represented when using absolute error metrics.



Close-up of gamma curves bottom part.

As stated above, relative error metrics give a much better accurated results when comparing images and should be used without exceptions.

The following section sumarizes all said above in four points that clearly define the aspects to take into consideration when evaluating image error.

1. The only way to evaluate perceptual difference between two images is to interleave them. Putting two images side-by-side is not enough. They should be layered and differenced if possible.

2. Absolute error metrics must be avoided at any time. Even if absolute error metrics are valid for many uses, it is not suitable for evaluating the perceptual difference between images.

3. Relative error should be used when possible. It will be larger than absolute error, representing the difference between two values relative to a reference value.

4. If values differ by more than an order of magnitude, relative error must be used. Otherwise lower values will be underrepresented.

5. When possible, a metric that takes into consideration human perceptual similarity should be used.

Having these considerations into account, it is the time to define a specific procedure for evaluating the error present in the different rendered images produced by the render engines conceiled in this study.

References

[1] John Hable, The Problem With Graphics Research: Error Metrics. March 2011

Rendering error evaluation procedure.

If working in lineal space and therefore following a Lineal Workflow [See Lineal Workflow section] , noise patterns and artifacts (both antialiasing and lighting) are the sources of error between rendered images that should be evaluated to empirically compare them in terms of quality.

It is necessary to distinguish the different sources of noise that could be present in a rendered image. Note that each of these sources of noise are controlled independently by render engine settings, but are also affected by the antialiasing method of choice.

1. Global Illumination artifacts from insufficient photon or final gather samples. (The famous disco-ball effect).

2. Aliasing noise along visible thin edges and bright spots of light.

3. Shadow noise where depth map shadow maps with low resolution, or raytraced shadows without enough latitude are used.

4. Raytrace effects undersampling noise (As an example, not giving an Ambient Occlussion [See AO section] pass enough samples).

5. Moire noise from textures that are not properly filtered in Rendertime.

6. Temporal noise from differences in sampling between frames (For example Radiosity [See GlobalIllumination section] flicker between frames).

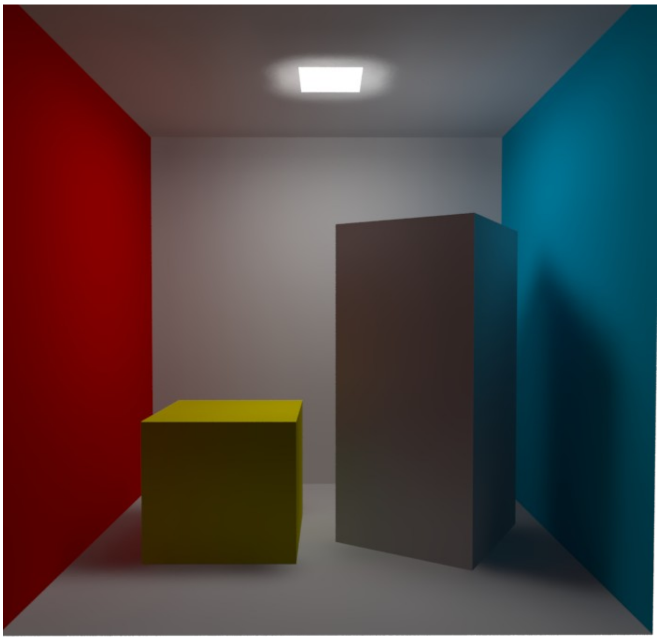
7. Grainy noise from camera lens effects with insufficient samples (DOF [see Depth of Field section], motion-blur [see Motion-Blur section]).

The mixture of noises present in rendered images shown in the previous paragraph makes impossible to separately evaluate sources of noise in final rendered scenes. Therefore it is necessary to isolate rendering techniques when comparing render engines and then evaluate the global noise of the resulting renders relative to a reference noise-free image.

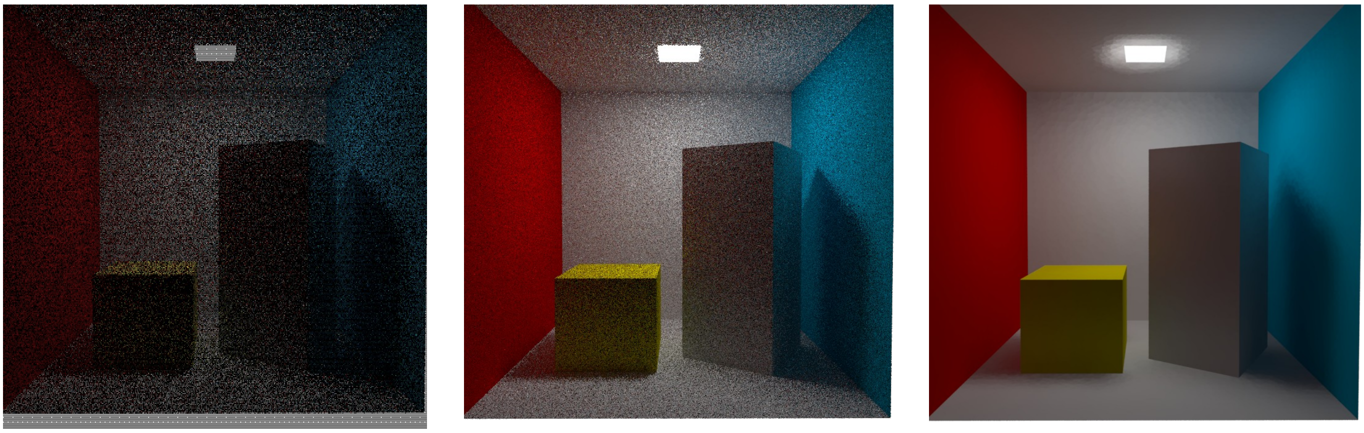
That reference image must be rendered using a theorical sampling limit (Infinite sampling quality leads to infinite render time).
When the sampling quality limit is reached for a given scene, the render process can be stopped and the resulting image is the reference render that will be used for further following the error evaluation procedure. Note that this sampling limit will be high enough, and therefore time-consuming, so that no other render that takes part in the comparison wether if it is biased or unbiased reaches this level of quality at any point of the rendertime range.

A first approach to accomplish this was done as follows.

The reference render was taken as a baseline to compare with the renders overlayed as a "difference" layer so differences between the two could be isolated.

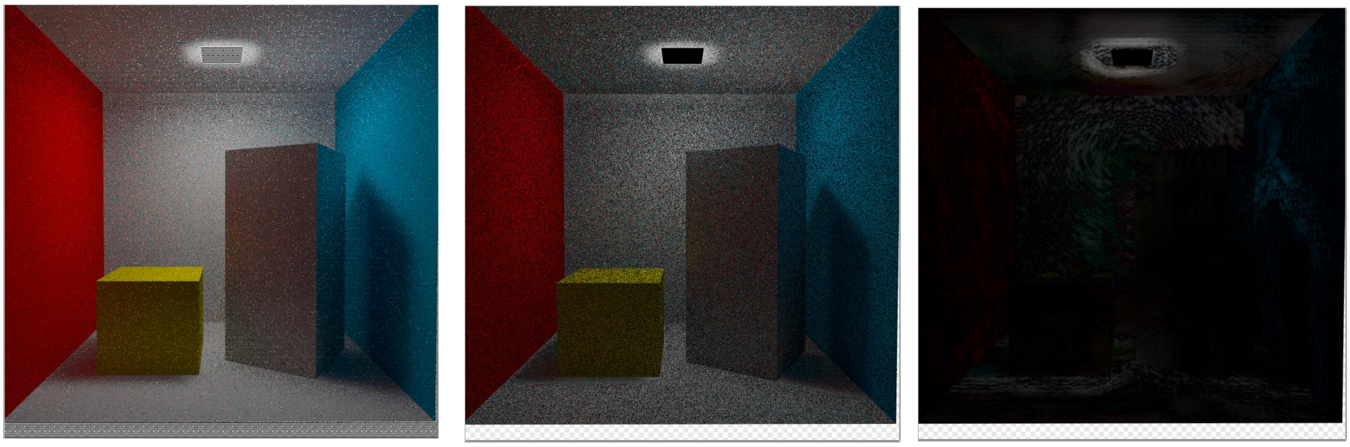


Reference rendered image of a CornellBox using unbiased algorithms (Rendered with , Sampling quality:).

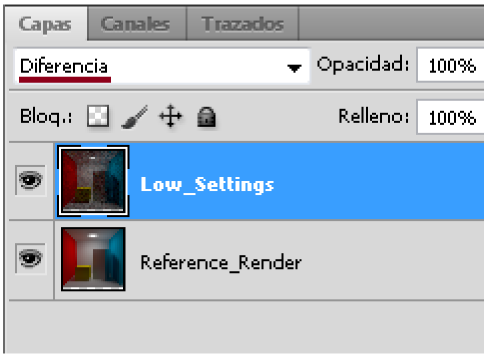


Three scenes rendered using Pathtracing algorithm with different levels of noise.

As said before, the reference render will be overlayed as a "difference" layer with the renders that take part in the comparison to isolate the differences between the two. The following images show the result of the difference between the reference and the three Pathtracing CornellBoxes.



Difference between Reference and target render in three scenes with different levels of noise.



Diferencia

Opacidad: 100%

Relleno: 100%

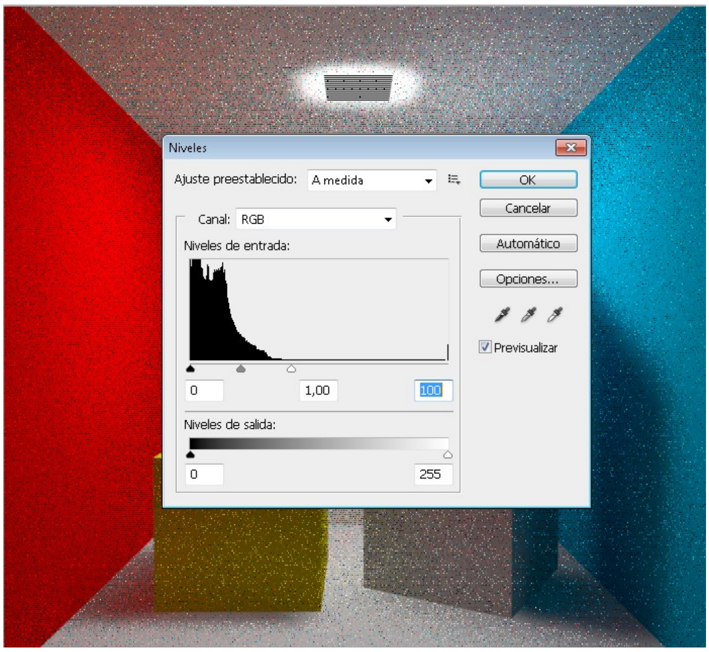
Low_Settings

Reference_Render

Photoshop layer settings for the noisiest
of the three CornellBox scenes.

In most cases, it was necessary to apply a levels filter to the resulting differenced images beforehand to brighten them up in a consistent way since the values might not have a big enough range. That could lead to images with nearly the same amount of noise to show the same level of noise at the end of the error evaluation procedure, and that must be avoided.

Note that if level filter is to be applied, the same level settings must be applied to all images that take part in a given technique comparison in order to get correct results and maintain the coherence between images.


Niveles

Ajuste preestablecido: A medida

Canal: RGB

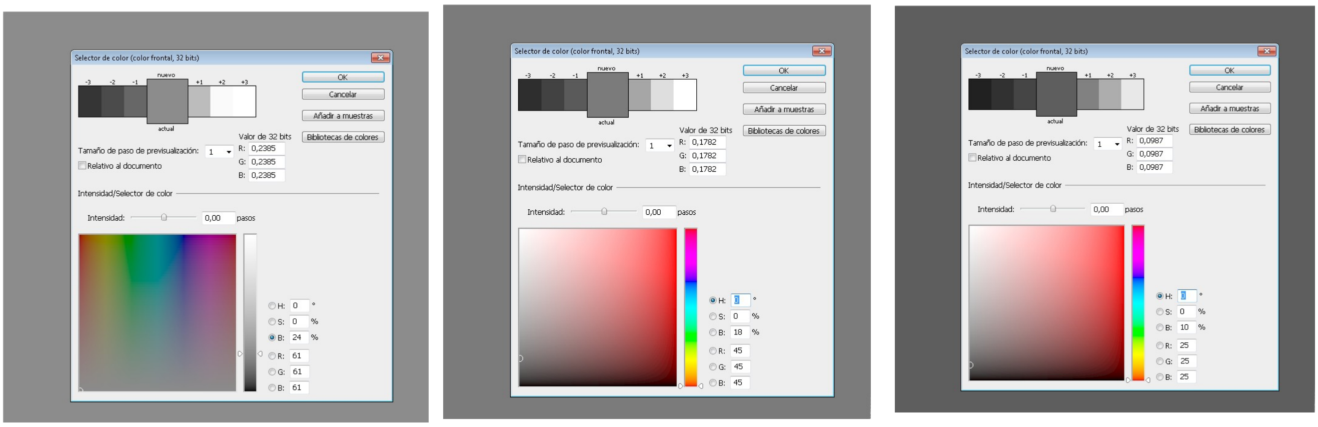
Niveles de entrada: 0 1,00 100

Niveles de salida: 0 255

☑ Previsualizar

Level adjustment of layer-differenced image.

Once the renders are layered-differenced and their levels adjusted, an average blur algorithm is applied to them in order to generate a single color for the frame. This single color will be the result of average filtering the colour of each pixel in the image.


Selector de color (color frontal, 32 bits)

Tamaño de paso de previsualización: 1
☐ Relativo al documento
Valor de 32 bits R: 0,2385 G: 0,2385 B: 0,2385
Intensidad: 0,00 pasos
○ H: 0 ° ○ S: 0 % ● B: 24 % ○ R: 61 ○ G: 61 ○ B: 61

Selector de color (color frontal, 32 bits)

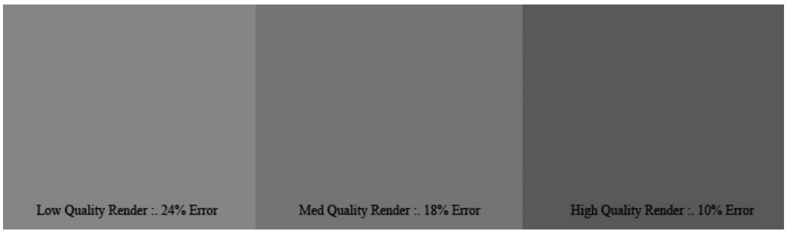
Tamaño de paso de previsualización: 1
☐ Relativo al documento
Valor de 32 bits R: 0,1782 G: 0,1782 B: 0,1782
Intensidad: 0,00 pasos
● H: 0 ° ○ S: 0 % ○ B: 18 % ○ R: 45 ○ G: 45 ○ B: 45

Selector de color (color frontal, 32 bits)

Tamaño de paso de previsualización: 1
☐ Relativo al documento
Valor de 32 bits R: 0,0987 G: 0,0987 B: 0,0987
Intensidad: 0,00 pasos
● H: 0 ° ○ S: 0 % ○ B: 10 % ○ R: 25 ○ G: 25 ○ B: 25

Average Filter applied to the three layer-differenced CornellBox scenes.
Note how the leften image's value is the highest in value.

Then, the three images will be desaturated and as a result, whichever filtered image has the higher value (V-value) is the one furthest away from the baseline "perfect" image and hence is the noisiest.



However, this first approach could not give a correct estimation of rendered image error because human-eye perception was not considered throughout the process.
For applications in which images are ultimately to be viewed by human beings, the only "correct" method of quantifying visual image quality is through subjective evaluation.

Back in 2004, under the assumption that human visual perception is highly adapted for extracting structural information from a scene, Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli introduced an alternative complementary framework for quality assessment based on the degradation of structural information. It was the Structural Similarity Index Metric. SSIM [See SSIM chapter].

The SSIM index is a full reference metric, in other words, the measuring of image quality based on an initial uncompressed or distortion-free image as reference. SSIM is designed to improve on traditional methods like peak signal-to-noise ratio (PSNR) and mean squared error (MSE), which have proved to be inconsistent with human eye perception.

The SSIM metric is calculated on various windows of an image. The measure between two windows *x* and *y* of common size *N*×*N* is: [1]

$$\mathrm{SSIM}(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

with

- μ_x the average of x;
- μ_y the average of y;
- σ_x^2 the variance of x;
- σ_y^2 the variance of y;
- σ_{xy} the covariance of x and y;
- $c_1=(k_1L)^2$, $c_2=(k_2L)^2$ two variables to stabilize the division with weak denominator;
- L the dynamic range of the pixel-values (typically this is $2^{\#bits\ per\ pixel}-1$);
- $k_1=0.01$ and $k_2=0.03$ by default.

In order to evaluate the image quality this formula is applied only on luma. The resultant SSIM index is a decimal value between -1 and 1, and value 1 is only reachable in the case of two identical sets of data. [2]

However, this methods presents some disadvantages that make it useless for rendering evaluation purposes: [See SSIM chapter for further details]

- Does not fare well when used on translated images (shifted, rotated). Small translations between renders in the same scene may occure when using different render engines.

- Does not correlate well for Gaussian-contaminated samples vs. blurred and JPEG-artifacted samples.

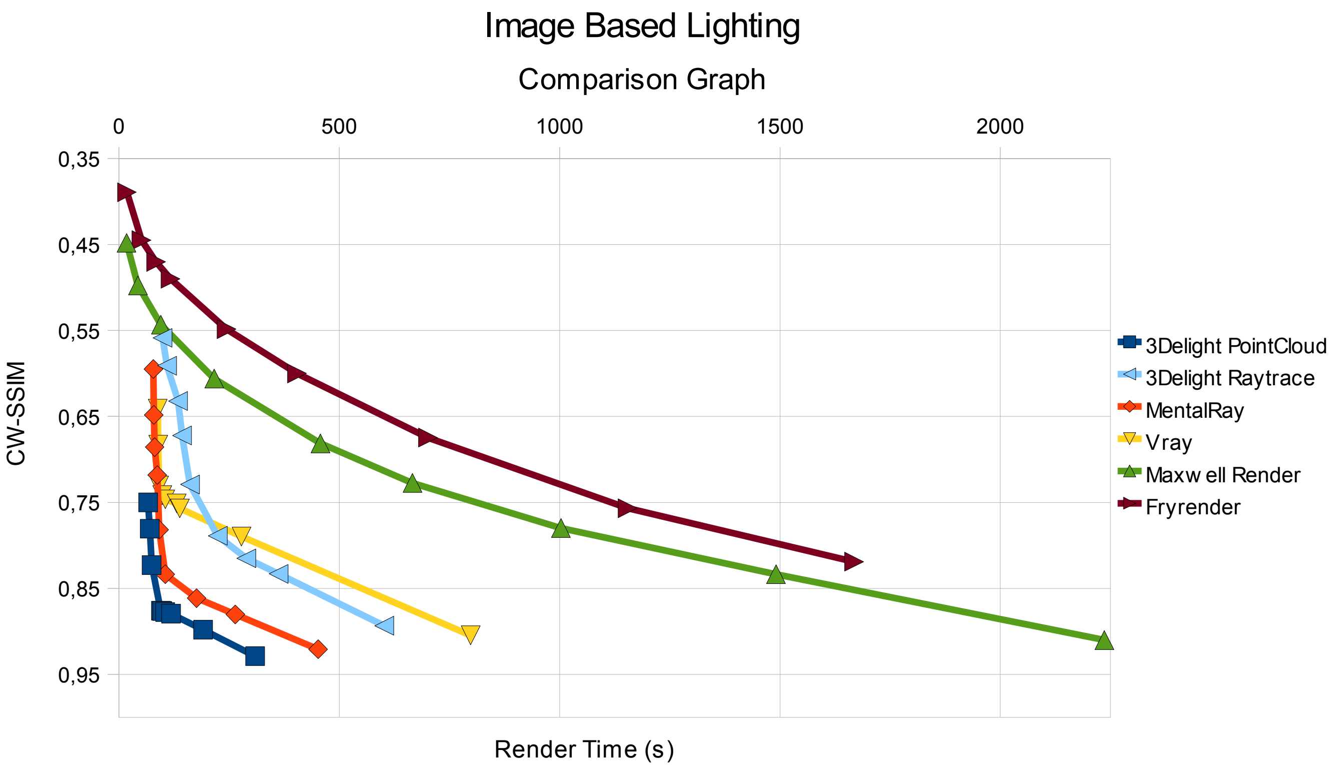
- Did not take into account any Human Vision System (HVS) features.

For that reason, it is necessary to use an improvement of this technique called " Translation insensitive image similarity in complexwavelet domain" (CW-SSIM) [See SSIM chapter], proposed in 2005, that solves the above problems.[3]

With CW-SSIM index it is now possible to evaluate the error of any isolated rendering technique.

However, the aim of this study is comparing render engines and therefore it is not enough to evaluating image error but it is necessary to evaluate how much time does a render engine take to achieve those error levels.
For that purpose, the rendertime of each render image must be stored so that the two values (rendertime, error) can be represented in a graph.

The graph is with Rendertime in the X-axis and CW-SSIM value in an inversed Y-axis.
Several images of a given scene with different settings on each renderer are rendered and the resulting values (rendertime, CW-SSIM index) are drawn in the graph.
Graph points could be interpolated to create a curve for each render engine.



Graph representing different errors vs rendertimes of a given scene rendered with different render engines.

As a result, for a given rendertime, the differences CW-SSIM value for each renderer can be observed.

References

[1] Z. Wang, A. C. Bovik, H. R. Sheikh and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," IEEE Transactions on Image Processing, vol. 13, no. 4, pp. 600-612, Apr. 2004.

[2] Loza et al., "Structural Similarity-Based Object Tracking in Video Sequences", Proc. of the 9th International Conf. on Information Fusion

[3] Translation insensitive image similarity in complex wavelet domain. Zhou Wang and Eero P. Simoncelli. Lab for Computational Vision, New York University, New York, NY 10003

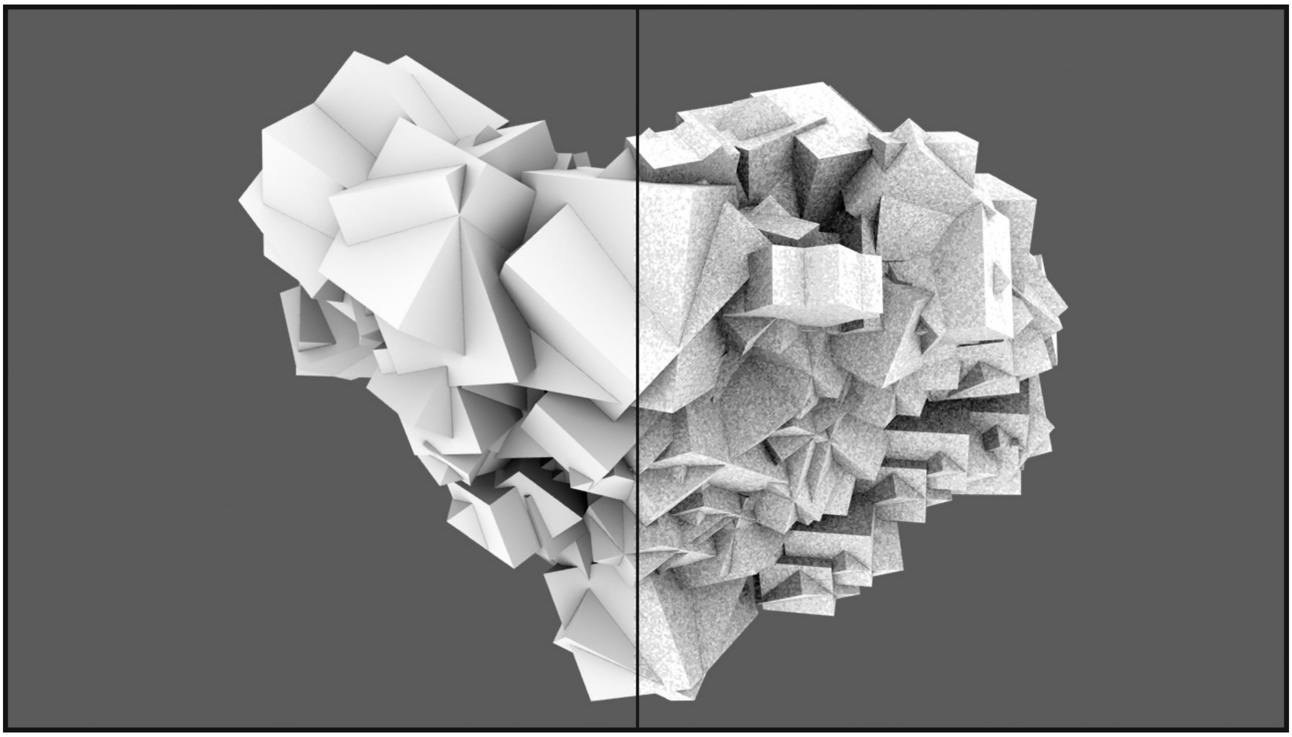
Render Engines Comparison.

Ambient Occlussion Comparison.

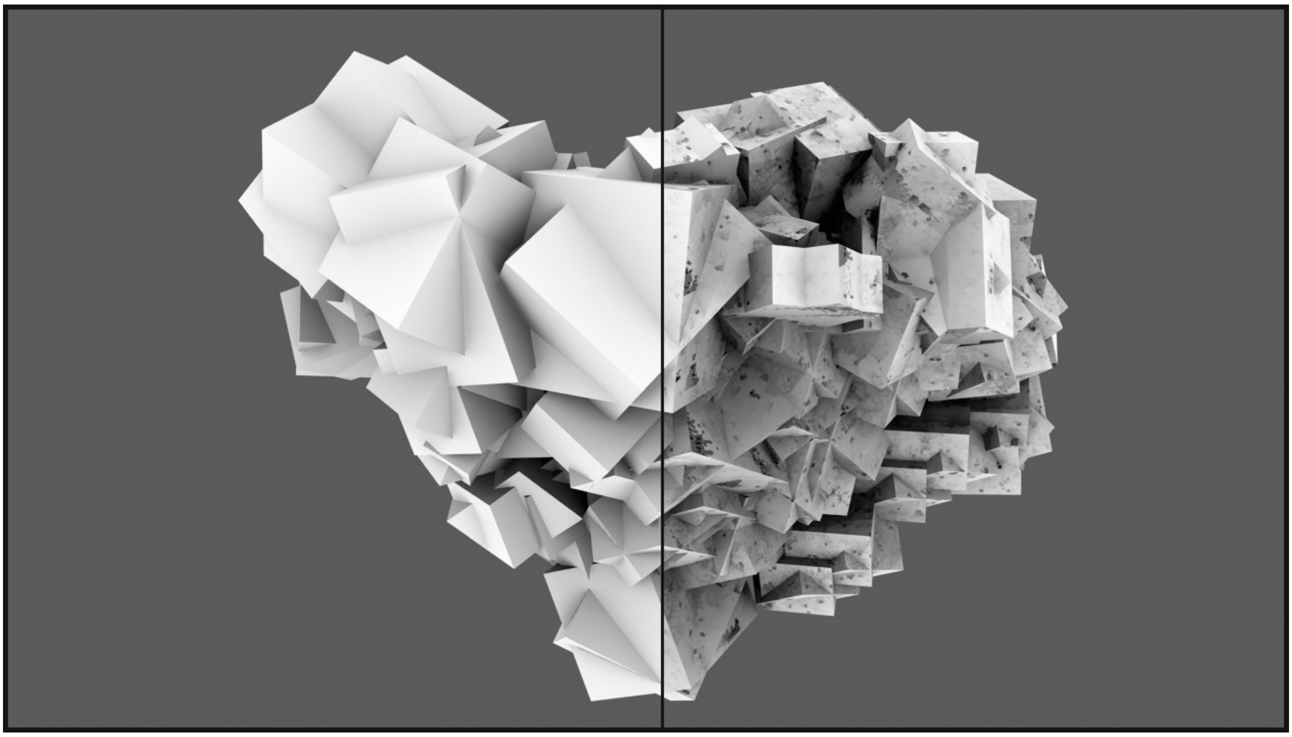
Ambient occlusion refers to the blocking of indirect or diffuse light on an object. It refers to the darker areas of the object, typically creases, cracks and crevices. *[See AmbienOcclussion in Glosary of Terms].*

It is caused by indirect light's inability to bounce around and illuminate areas that are blocked by a nearby object that absorbs the light rays.
These subtle variations in lighting are visual clues for our eyes to detect surface details and distinctions that would otherwise be washed out and unnoticeable.

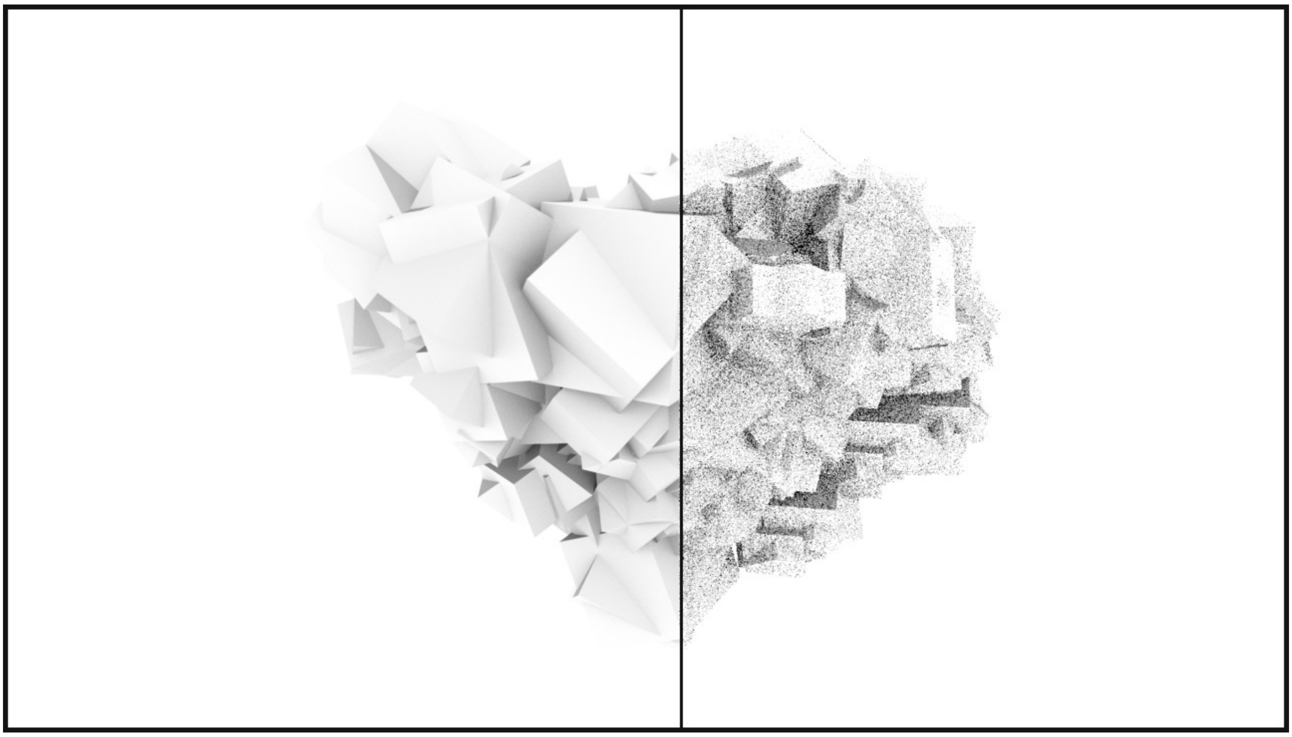
The following comparison involves different methods for the computation of Ambient Occlussion depending on the software used. These could be divided into Raytracing PointCloud, and Unbiased methods.
The following pictures are render images used in this comparison, example of each kind of method, showing up the differences between a low quality noisy image and a good quality, noise free image. Note how the level of quality in Ambient occlussion is determined by the amount of noise each image contains, apart from point clound rendering method, where the appearance of blotchy artifacts indicate a poor quality render.



Raytrace computation of Ambient Occlussion.
(3Delight Raytrace 1024 and 8 Occlight samples respectively)



PointCloud computation of Ambient Occlussion.
(3Delight Point Cloud rendering 0,1 and 1 MaxSolidAngle Value respectively)



Unbiased computation Global Illumination (Ambient Occlussion layer)
(MaxwellRender 8 and 2 samples respectvely)

Please refer to "Renders" subfolder for a complete list of rendered images used in this part of the study for each rendere engine involved in the comparison.

The following table shows CW-SSIM vs RenderTime values in nine different renders for each render engine involved in this study.

Please note that values given in the table have been rounded for a better and faster appreciation .

3Delight Raytrace

	1	2	3	4	5	6	7	8	9
Render Time (s)	5	6	8	14	35	74	146	215	286
CW-SSIM	0,88	0,9	0,92	0,93	0,95	0,97	0,98	0,98	0,99

Samples	8	16	32	64	128	256	512	768	1024

Occlight samples (SpotLight)

3Delight PointCloud

Render Time	2	2	22	26	27	29	32	38	50
CW-SSIM	0,84	0,94	0,97	0,97	0,99	0,99	0,99	0,99	0,99

MaxSolidAngleValue	1	0.75	0.5	0.35	0.3	0.25	0.2	0.15	0.1

ptc_occlusion shader *

MentalRay

Render Time	19	23	33	55	97	182	356	729	1061
CW-SSIM	0,93	0,95	0,96	0,98	0,99	0,99	0,99	0,99	1

Samples	2	8	16	32	64	128	256	512	768

Vray

Render Time	28	32	32	39	58	81	110	144	183
CW-SSIM	0,83	0,92	0,92	0,94	0,95	0,96	0,96	0,97	0,97

Subdivs	1	10	20	30	40	50	60	70	80

Maxwell Render

Render Time	8	18	42	79	148	312	1013	1108	1257
CW-SSIM	0,87	0,9	0,91	0,92	0,93	0,94	0,95	0,96	0,96

Sampling Level	2	4	6	8	10	12	14	15	16

Fryrender

Render Time	40	81	122	163	204	306	407	508	1002
CW-SSIM	0,5	0,56	0,58	0,58	0,58	0,61	0,62	0,63	0,71

Passes	2	4	6	8	10	15	20	25	50

Lineal rendertime vs passes
1 pass = 20,5 sec

For a more precise representation of the values shown, refer to the excel sheet located in the corresponding section's folder named "tables.ods"

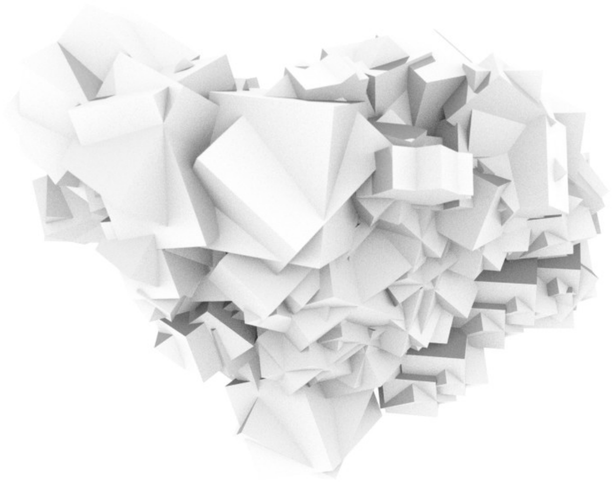
The following code shows the point cloud Ambient Occlussion algorithm used with 3Delight renderer for this part of the comparsion.

Ptc_occlusion shader *

```
surface ptc_occlusion(
                        string ptc_file = "default.ptc";
                        float biasValue = 0.1;
                        float clampValue = 0;
                        string hitsidesValue = "both";
                        float samplebaseValue = 1;
                        float maxdistValue = 0;
                        float maxsolidangleValue = 0.5; )
{
                        normal Nf = faceforward( normalize(N), I );

                        Ci = 1 - occlusion( P, Nf,
                                    filename, ptc_file, "pointbased", 1,
                                    bias, biasValue, "clamp", clampValue, "hitsides", "both", "samplebase", 1 , "maxdist", maxdistValue, "maxsolidangle", maxsolidangleValue  );
}
```

It should be noted that the Ambient Occlusion channel in unbiased render engines has little or no use at all in compositing.
It is just provided as an easy way to create clay-like illustrations. Given the nature of unbiased render engines *[See Bias in Rendering section]* , ambient occlussion is computed as part as the global illumination unbiased equation.

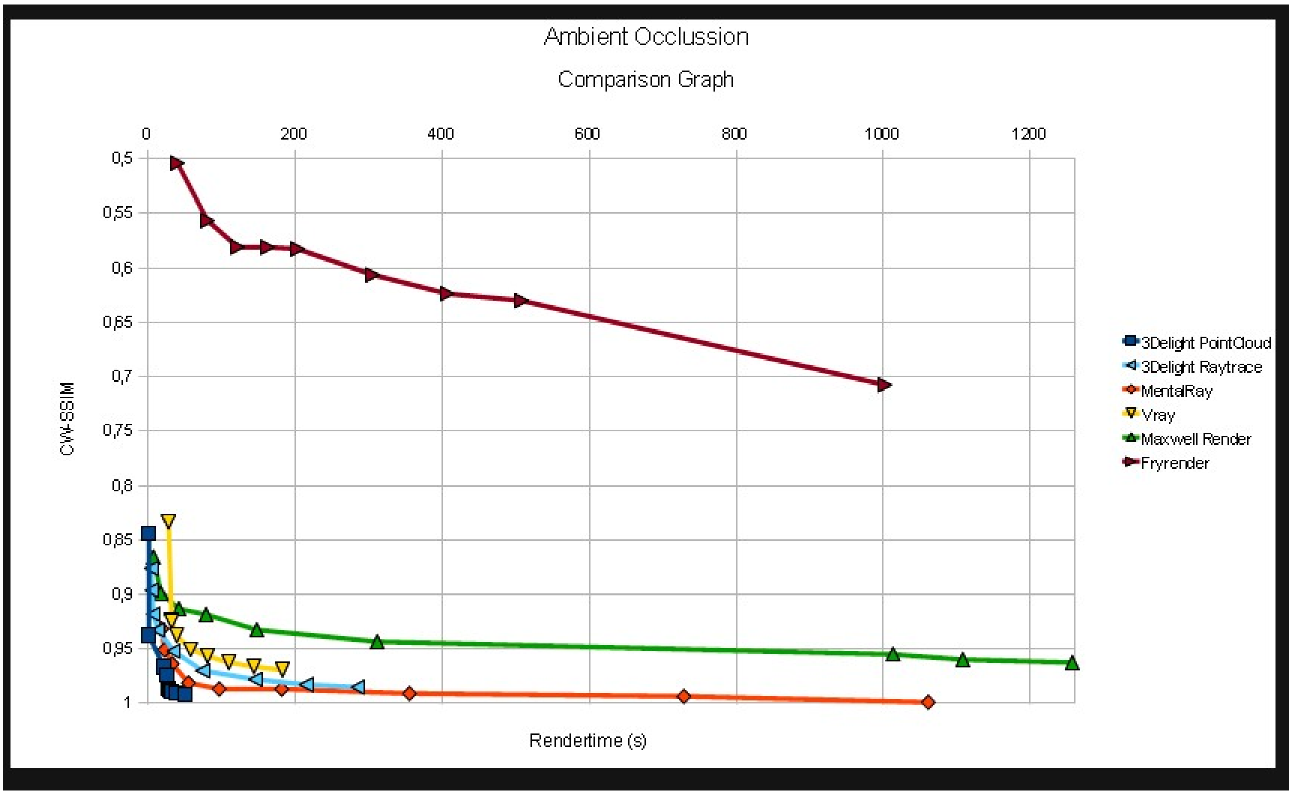


Ambient Occlussion Channel in an Unbiased Renderer
(Maxwell Render 16 sampes).

Ambient Occlussion is ruled by one parameter that can be regarded as the radius of the darkening around the corners in the scene. Note that it makes no sense at all to combine an Ambient Occlussion layer with an unbiased render as it has no physical meaning at all.

Anyway, it was a good way to isolate the effect, so unbiased renderers could be included in this section of the comparison.

Once the table is built, a graph can be drawn for a more intuitive representations of the results obtained. Please refer to "Error Metrics" chapter in this document for a more detailed explanation.



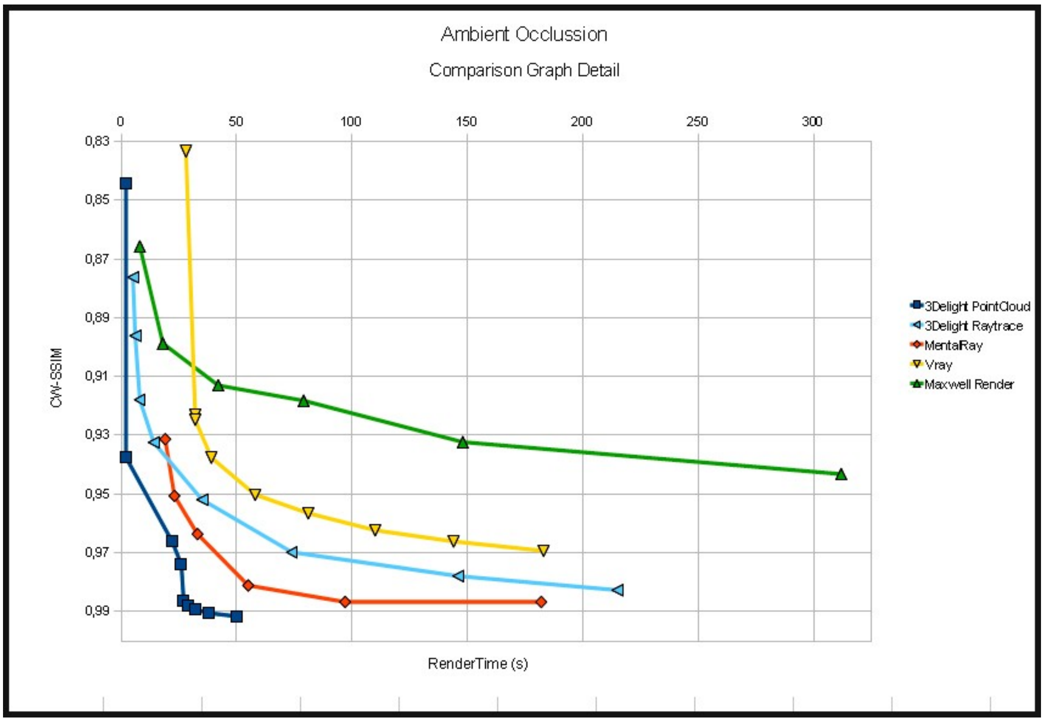
Ambient Occlussion Comparison Graph.

As seen on the above graph, FryRender stays in the worst position throughout all the time range. As stated before, given the nature of unbiased render engines Ambient Occlussion has no physical meaning, and the whole GI algorithm is being calculated in those cases where an Unbiased Renderer is used.
It could be expected then that unbiased render engines give a worse result compared with biased renderers, when isolated Ao is to be computed.

That is clearly represented in the green and red lines (Maxwell render and FryRender respectively). MaxwellRender stays above 0,95 CW-SSIM value after 1000 seconds of rendering, while Fryrender stays above 0,71 in the first 1000 seconds.

As for unbiased renderers, a clearly defined better performance in Ambient Occlussion calculation may be expected then from MaxwellRender compared to FryRender.

Biased render engines, however, present a better overall performance opposited to unbiased renderers.
For a better representation of the biased renderers, a zoomed version of the previous graph focusing on biased renderers' data is shown.



Ambient Occlussion Detailed Comparison Graph.

Note how 3Delight PointCloud method shows the best performance between all biased methods. A CW-SSIM value of 0,97 is reached after the first 20 seconds of rendering and it gets to 0,99 (almost equal to the Reference render) in no more than 27 seconds.

On the other hand, 3Delight Raytraced method gets worse results. It reaches a CW-SSIM value of 0,99 at 288 seconds (Whereas pointCloud method does in 27), and stays slower than PointCloud in all the TimeRange.

About Mentalray, it offers an overall performance inbetween 3Delight PointCloud and Raytrace, it reaches a CW-SSIM value of 0,97 in less than 50 seconds, while reaching 0,99 in 98 seconds. It would be a great choice when preprocessing the scene to get the PointCloud *[See PointCloud section]* is not feasible (For example in really big detailed scenes or rendered in such a high resolution that the point cloud would take too much time to compute compared to the rendering time itself).

Lastly, Vray render offers a worse overall performance from biased renderers, reaching a CW-SSIM value of 0,97 in 144 seconds, far later than the other biased engines (70 seconds after 3Delight raytrace and up to 122 seconds agter 3Delight PointCloud rendering method).

Anyway, as happens with all biased renderers included in this comparison, its performance stays in a much better place in all the time range compared with unbiased render engines.
That could be expected beforehand due to the nature of unbiased engines. As said earlier in this document, Ambient Occlussion has no physical meaning at all, so unbiased render engines compute the whole Global Illumination to render these images whereas biased render engines calculate Ambient Occlussion separately without the need to solve the global illumination equation.

All that said, 3Delight PointCloud is the better choice when an intensive calculation of Ambient Occlussion is needed, for example in scenes where lots of small objects that produce creases, cracks and crevices are present.

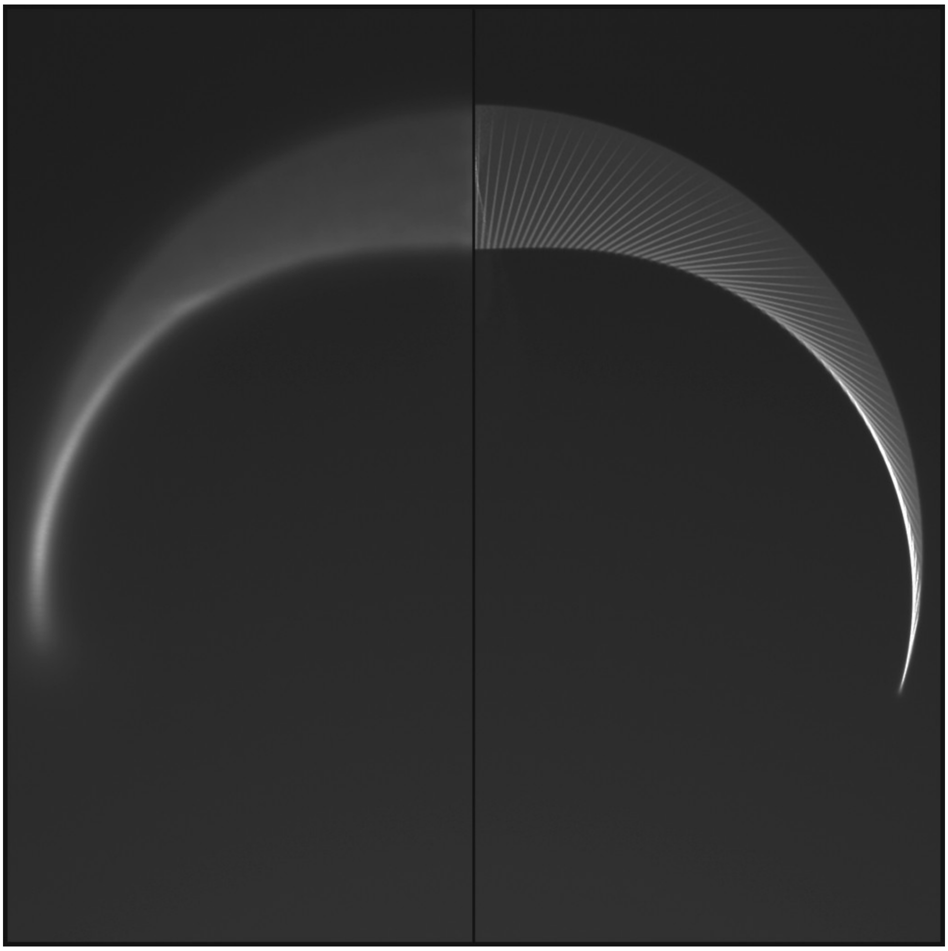
Caustic Comparison.

Caustics are light patterns that are created when light rays emitted from a light source are bent by one intermediate medium and illuminate a diffuse surface via one or more specular reflections or transmissions (refractions). *[See Caustics chapter in Glossary of Terms].*

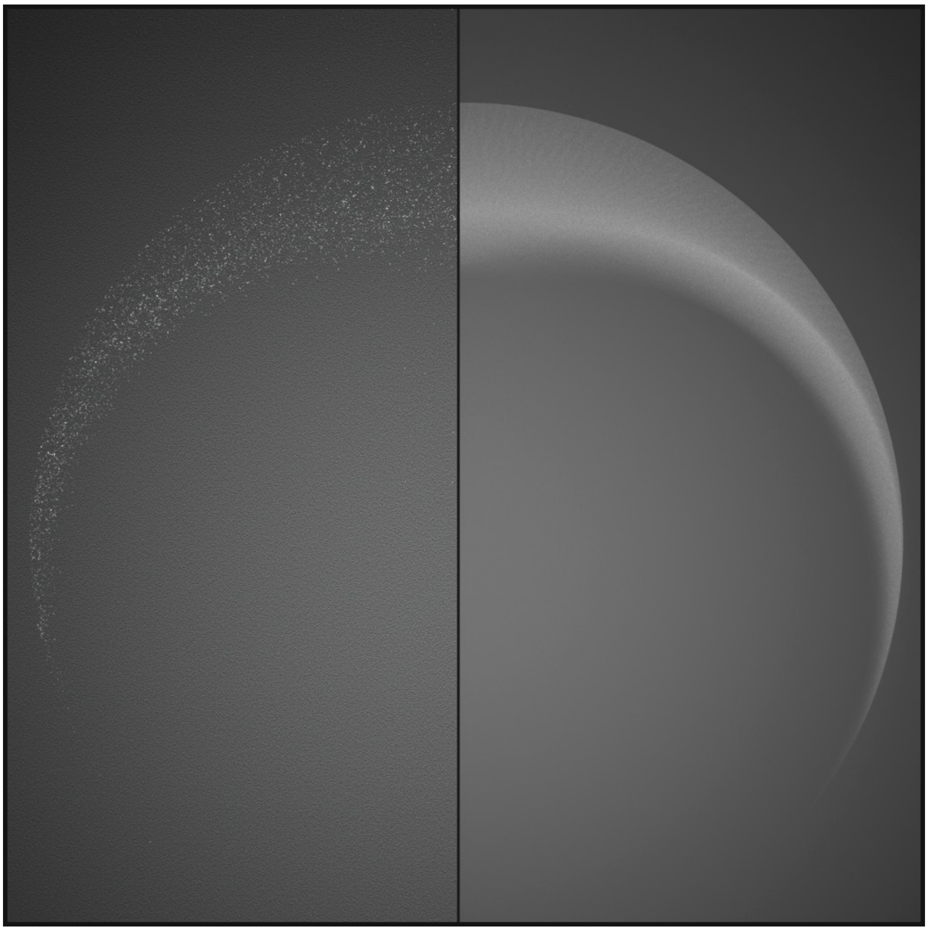
The following comparison involves different methods for the computation of Caustics depending on the software being used. These could be divided into Photon Mapping methods, and Unbiased methods.

Photon Mapping *[See Photon Mapping chapter in Glossary of Terms]* methods are used by Biased render engines such as the ones involved in this study (MentalRay, Vray and 3Delight), whereas Caustics in Unbiased render engines are produced natively with the global illumination calculation (Metropolis Light Transport GI method for example). For more information about photon mapping and global illumination methods in unbiased renderers please refer to "Global Illumination" chapter in Glossary of Terms.

The following images are render images used in this comparison, as an example of each kind of method, showing up the differences between noisy and blurred caustics produced by a low quality render versus sharp well defined caustics produced by a high quality noise free render.



Low quality vs High quality Caustics in Biased Renderers.
(From left to right, MentalRay 50K and 4M photons emitted)



Low quality vs High quality Caustics in an Unbiased renderer.
(From left to right, MaxwellRender 6 samples and 20 samples)

Please refer to "Renders" subfolder for a complete list of rendered images used in this part of the study for each rendere engine involved in the comparison.

Note how the level of quality in Caustics is mainly determined by the amount of photons emitted from the light source in Biased renderers. The more blurred and noisy each image is, the less photons are emited, or less samples are computed in biased and unbiased render engines respectively.

On the other hand, clearly different type of caustics are achieved with biased and unbiased render engines. Due to the physical nature of unbiased render engines, and according to the previous images, note how unbiased renderers produce more soft physically accurate caustics as image samples are increased.
However, biased renderers produce sharper caustics as the photon emission amount increases. That is an inherent consequence of photon-mapping's caustics generation algorithm.

For most situations though, photon mapping caustics would produce such similar results to unbiased renderers that it is a method worth using when higher rendertimes are not acceptable, as will be discussed thorugh the following pages in this chapter.

The following table shows CW-SSIM vs RenderTime values in nine different renders for each render engine involved in this study.

3Delight

	1	2	3	4	5	6	7	8	9
Render Time (m,s)	4	7	17	30	48	161	335	528	784
CW-SSIM	0,46	0,59	0,67	0,71	0,74	0,8	0,83	0,84	0,85
Caustic Photons	90K	900K	3M	6M	9M	30M	60M	90M	130M

Estimator = 85
Values above 160M exceeded maximum RAM usage

MentalRay

Render Time	7	7	9	27	47	87	128	167	205
CW-SSIM	0,58	0,62	0,77	0,88	0,91	0,94	0,96	0,97	0,97
Caustic Photons	500	5K	50K	500K	1M	2M	3M	4M	5M

Caustics Accuracy= 100; Caustic Filter = Cone; Filter Kernel = 1.1;

Maxwell Render

Render Time	16	26	45	59	306	648	1324	2085	3074
CW-SSIM	0,17	0,19	0,2	0,22	0,44	0,51	0,57	0,6	0,63
Samples	3	4	5	6	10	12	14	15	16

Vray

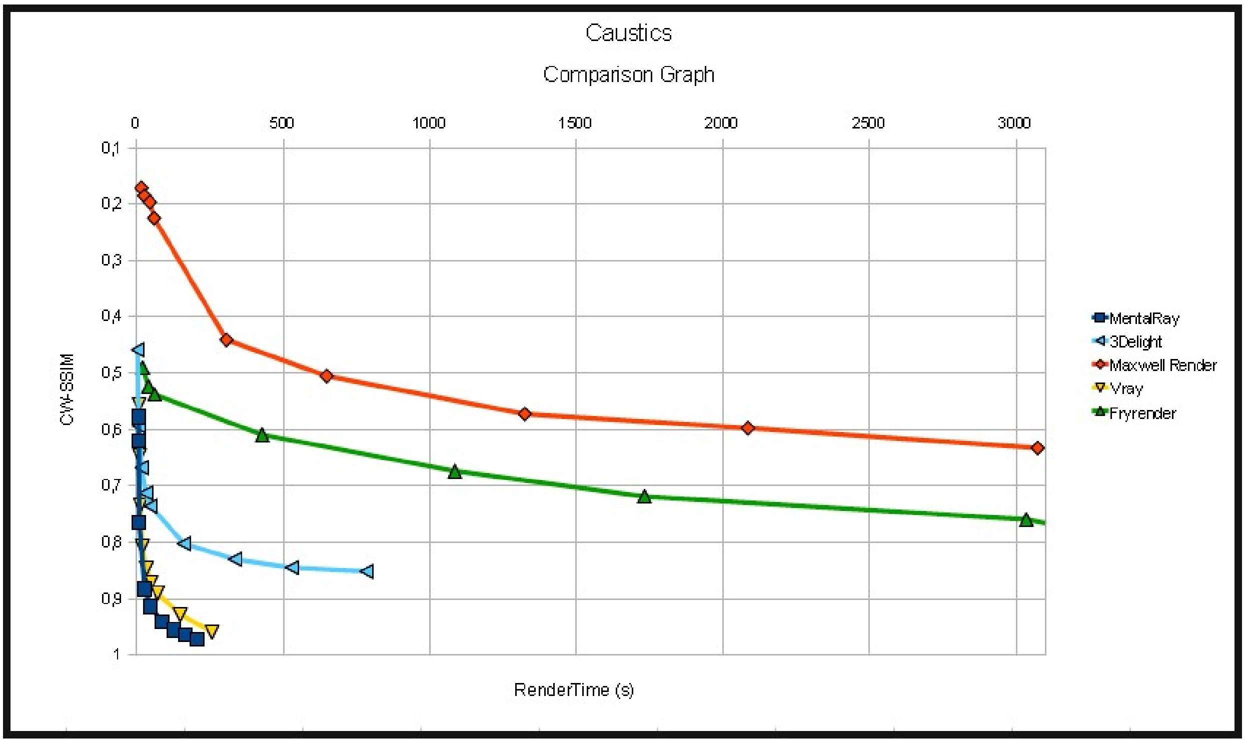
Render Time	8	9	12	19	33	50	71	148	257
CW-SSIM	0,56	0,65	0,74	0,81	0,85	0,87	0,9	0,93	0,96
Caustics Subdivs	500	1K	2K	4K	6K	8K	10K	15K	20K

Max photons = 100

Fryrender

Render Time	20	40	61	428	1085	1358	1732	3035	3475
CW-SSIM	0,49	0,52	0,54	0,61	0,67	0,.6938	0,72	0,76	0,8
Passes	1	2	3	25	65	82	106	151	211

Please note that the values given in the table have been rounded for a better and faster appreciation . For a more precise representation of the values shown, refer to the excel sheet located in the corresponding section's folder named "tables.ods".
Once the table is built, a graph can be drawn for a more intuitive representations of the results obtained. Please refer to "Error Metrics" Chapter in this document for a more detailed explanation.



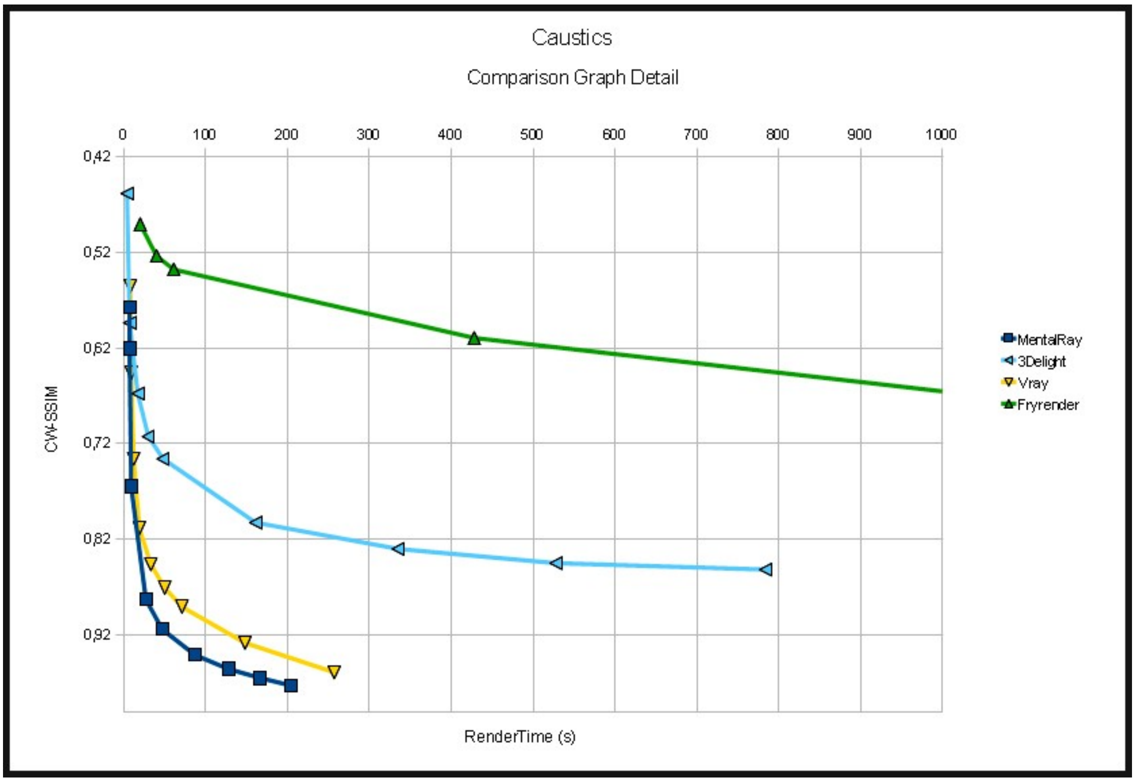
Caustics Comparison Graph.

As seen from the above image, both unbiased render engines present quite similar characteristics. However, Fryrender performs better when intensive caustics generation is needed . Both curves are almost parallel throughout the timerange (500 – 3000 secs) so it could be stated that it gets to the same level of quality (CW-SSIM value) about 1600 seconds before MaxwellRender.

The fastest of both renderers for the computation of this technique, FryRender, takes about 3500 seconds to get to a CW-SSIM value of 0,8 (quite inacceptable as a final production-quality render).

Biased render engines, however, present a much better overall performance opposited to unbiased renderers, reaching levels of detail up to 0,97% in a fraction of the time unbiased render engines do. Anyway, as stated in previous pages of this chapter, unbiased render engines for caustics generation are most suitable when smooth physically accurated caustics are needed (due to the nature of unbiased global illumination calculation), if there are no time restrictions in the pìpeline.

For a better representation of biased renderers, a zoomed version of the previous graph focusing on biased renderers' data is shown.
Note how the green curve (FryRender) stays way up the slowest of the biased renderers in caustics calculation (3Delight).



3Delight photon mapping for caustics calculation performs much worse that the other biased render engines (Vray and Mentalray). In fact, it reaches a quality level of 0,80% by the time the other biased render engines stay above 0,92%
Furthermore, RAM usage *[See Hardware Specification Section]* when 3Delight is used is exceeded when a value greater than 160M in caustics photons is set.
That said, a quality level greater 0,85% could not be achieved with the workstation being used in this study.

On the other hand, Mental Ray and Vray perform better and without such an intense RAM memory usage. A value greater than 0,9 is reached before 47 and 71 seconds respectively.

As seen in the detailed graph above, MentalRay reaches higher quality values in less rendertime though, being the best render engine for caustics generation in this study.

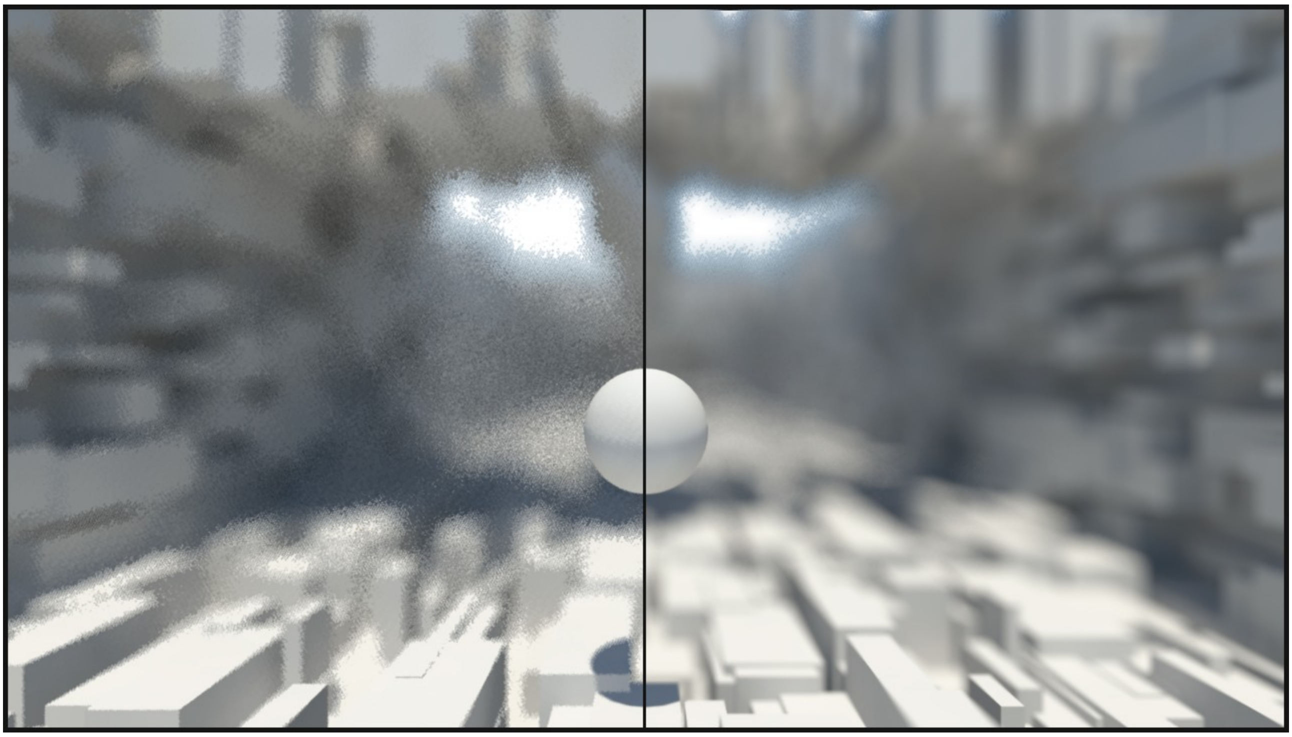
Depth of Field Comparison.

In optics, particularly as it relates to film and photography, depth of field (DOF) is the distance between the nearest and farthest objects in a scene that appear acceptably sharp in an image. Although a lens can precisely focus at only one distance at a time, the decrease in sharpness is gradual on each side of the focused distance, so that within the DOF, the unsharpness is imperceptible under normal viewing conditions. *[See Depth of Field Chapter in Glossary of Terms]*.

Common camera parameters have been chosen for all renders that take part in the Depth of Field comparison. That done, the amount of DOF between render engines does not vary so that the obtained results are coherent.

- Lens Aperture = f1
- Shutter Speed = 1 / 8000
- Focal Length = 22

The following image is an example of Depth of Field generation renders used in the comparison for a particular biased render engine, showing up the differences between noisy Depth of Field produced by a low quality render versus smooth noise free Depth of Field produced by a high quality render.



Low quality vs High quality Depth of Field in Biased Renderers.
(From left to right, MentalRay 2 and 256 samples computed).

Please refer to "Renders" subfolder for a complete list of rendered images used in this part of the study for each rendere engine involved in the comparison.

Note how the level of quality in Depth of Field is mainly a matter of image noise. The more noisy each image is, the less depth of field samples, or less image samples are being computed in biased and unbiased render engines respectively.

The following table shows CW-SSIM vs RenderTime values in nine different renders for each render engine involved in this study.

3Delight

	1	2	3	4	5	6	7	8	9
Render Time (m,s)	13	14	15	13	14	15	16	18	33
CW-SSIM	0,86	0,92	0,94	0,77	0,84	0,88	0,95	0,96	0,97
Pixel Samples (filter)	1-1 (sync)	2-2 (sync)	3-3 (sync)	1-1 (gauss)	2-2 (gauss)	3-3 (gauss)	4-4 (gauss)	5-5 (gauss)	10-10 (gauss)

MentalRay

Render Time	48	66	81	115	175	332	477	602	785
CW-SSIM	0,66	0,52	0,71	0,57	0,76	0,61	0,78	0,79	0,79
Samples	2	4	8	16	32	64	96	128	160

Vray

Render Time	68	208	365	407	441	482	498	541	607
CW-SSIM	0,93	0,84	0,93	0,89	0,93	0,93	0,93	0,94	0,94
Subdivs	1	2	3	4	5	6	7	8	9

Maxwell Render

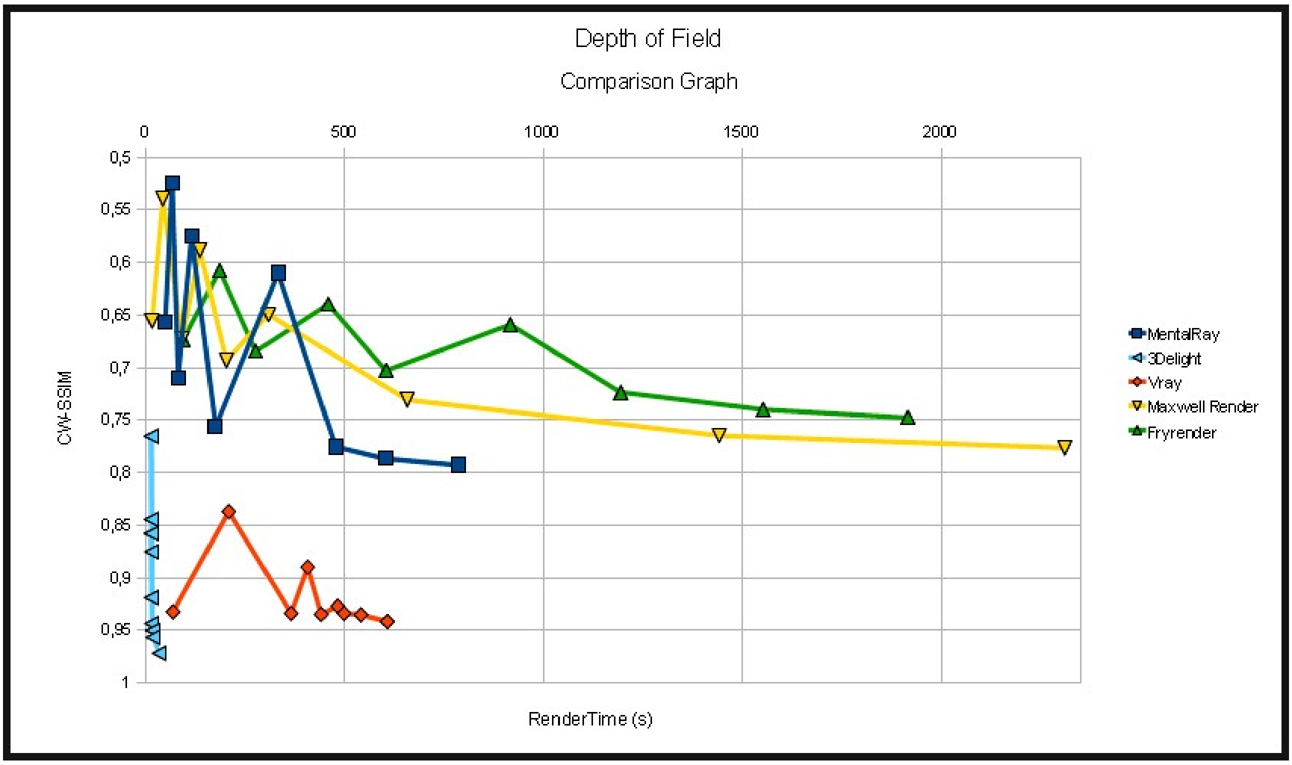
Render Time	15	42	91	136	202	308	657	1442	2311
CW-SSIM	0,66	0,54	0,67	0,59	0,69	0,65	0,73	0,76	0,78
Sampling Level	3	5	7	8	9	10	12	14	15

Fryrender

Render Time	94	185	276	458	604	916	1194	1552	1916
CW-SSIM	0,67	0,61	0,68	0,64	0,7	0,66	0,72	0,74	0,75
Passes	2	4	6	10	14	20	26	34	42

Please note that the values given in the table have been rounded for a better and faster appreciation . For a more precise representation of the values shown, refer to the excel sheet located in the corresponding section's folder named "tables.ods".

Once the table is built, a graph can be drawn for a more intuitive representations of the results obtained. Please refer to "Error Metrics" Chapter in this document for a more detailed explanation.



Depth of Field Comparison Graph.

As can be easily seen from the above image, 3Delight performs much better than any other render engine in this study. Depth of field computation is almost costless for this render engine, as it reaches a value of quality (CW-SSIM value) of 95% in just 16seconds.

Besides that, 3Delight curve in the graph shows how stable this render engine is for this type of calculation, whereas the other render engines show clear signs of inestability (see cuve pikes) during the first hundreds of seconds of rendertime.

The key to such a good behaviour of 3Delight resides in its Reyes Algorithm Core, that is capable of computing depth of field, motion blur, and displacement, as will be shown later in this chapter, almost without rendertime cost.

About the other render engines, we notice how Vray biased render engine performs quite good compared to the other renderers. It shows some signs of inestability in the first 400 seconds of rendering, but it does not fall below a value of 84% in CW-SSIM.
After those six and a half minutes, it stabilizes, reaching a CW-SSIM value of 94% in about ten minutes.
Note how slow is this compared to 3Delight, that reaches 95% in a fraction of the time (16 seconds).

Mental Ray, respectively, takes a bad place in this comparison, staying above unbiased render engines, but too close to them to compare them together. It stabilyzes after the first 500 seconds, reaching a quality level of 80% in 785 seconds

Maxwell render, on the other hand, stabilyzes somewhat sooner, in about 300 seconds, and reaches a value of 78% in 2311 seconds, a bit better than its unbiased companion, Fryrender, which stabilyzes quite later (about 1300 seconds opposite to 300 seconds of MaxwellRender) but stays close to MaxwellRender in terms of image quality after the first 1500 seconds.
There is a difference of about 2% between them beyond that point.

All that said, seems clear that 3Delight render engine should be used when strong depth of field is needed, in order to produce noise free renders without almost any time cost. That will otherwise be impossible with the biased render engines included in this study, nor with the unbiased renderers.

Displacement Mapping Comparison.

Displacement mapping is a rendering technique that variates the positions of surface elements (vertex). [See Displacement Mapping chapter in Glossary of Terms]. This leads to effects not possible with bump mapping, such as surface features that occlude each other and nonpolygonal silhouettes.

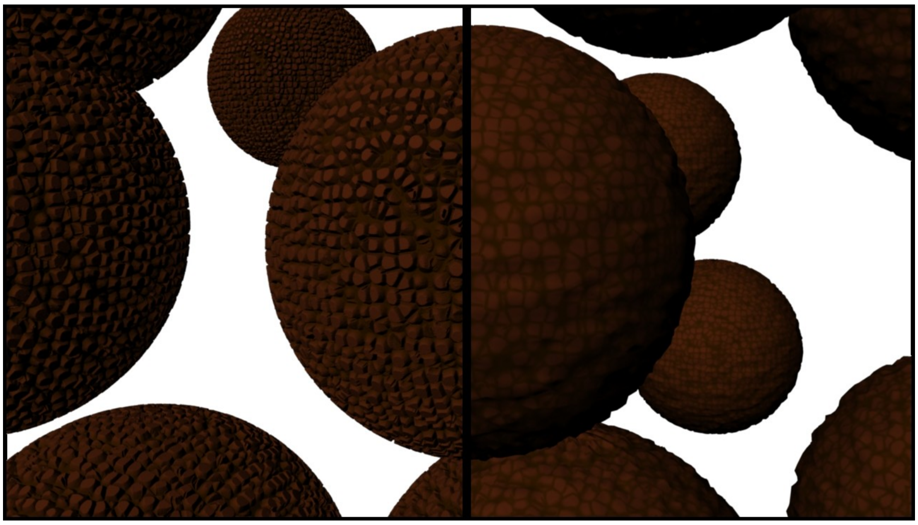
The usual implementation of displacement mapping iteratively tessellates a base surface, pushing vertices out along the normal of the base surface, and continuing until the polygons generated are close to the size of a single pixel.

The following comparison involves different methods for the computation of Displacement Mapping depending on the software used. These could be divided into Biased, and Unbiased methods, taking special consideration in 3Delight's Reyes rendering implementation.

The following pictures are render images used in this comparison, example of each kind of method, showing up the differences between a low quality and a good quality image.

Note how the level of quality in Displacement mapping is determined by the surface's deformation accuracy relative to the image map used for the appliance of this technique. Therefore, high quality renders will produce images with surfaces displaced accurately according to the displacemnet map, while low quality renders will deform surfaces poorly, as polygon retesselation won't be detailed enough to deform the surface accuately.

In other words, poorly displaced surfaces won't contain enough vertex count to be sculpted accordingly , relative to the image map used for displacement.



Tesselation differences using the same displacement map.
(MentalRay native displacement calculation).

Please refer to "Renders" subfolder for a complete list of rendered images used in this part of the study for each rendere engine involved in the comparison.

The following table shows CW-SSIM vs RenderTime values in nine different renders for each render engine involved in this study.

Please note that values given in the table have been rounded for a better and faster appreciation .

3Delight

	1	2	3	4	5	6	7	8	9
Render Time	3	3	3	4	4	5	6	13	31
CW-SSIM	0,55	0,58	0,63	0,7	0,79	0,86	0,91	0,97	1
Shading rate	20	15	10	5	2	1	0,5	0,1	0,05

1/Micro polygons per pixel

MentalRay

Render Time	52	53	55	61	80	148	*	*	*
CW-SSIM	0,49	0,57	0,7	0,83	0,9	0,94	…	…	…
Max Subdivisions	0	1	2	3	4	5	6	7	8

Max Subdivisions (Length = 0,25)

* Ram usage exceeded, unable to complete render.

Vray

Render Time	38	39	39	57	123	179	263	660	*
CW-SSIM	0,61	0,61	0,68	0,72	0,73	0,74	0,75	0,75	…
Edge Length	500	250	100	10	5	4	3	2	1

Keep contiunity= On, Max Subdivs=512

* Ram usage exceeded, unable to complete render.

Maxwell Render

Render Time m,s	658	675	725	805	855	916	988	1809	1824
CW-SSIM	0,7	0,75	0,83	0,.8785	0,89	0,9	0,91	0,91	1
Precision	1	2,5	5	10	12	15	20	25	Adapt

Sampling level = 7

Render without displacement = 58s

Fryrender

Render Time	686	805	829	892	902	905	998	1081	1500
CW-SSIM	0,73	0,74	0,74	0,81	0,84	0,86	0,89	0,89	0,96
Refinement	32	64	128	256	512	512*	1024*	1024	2048

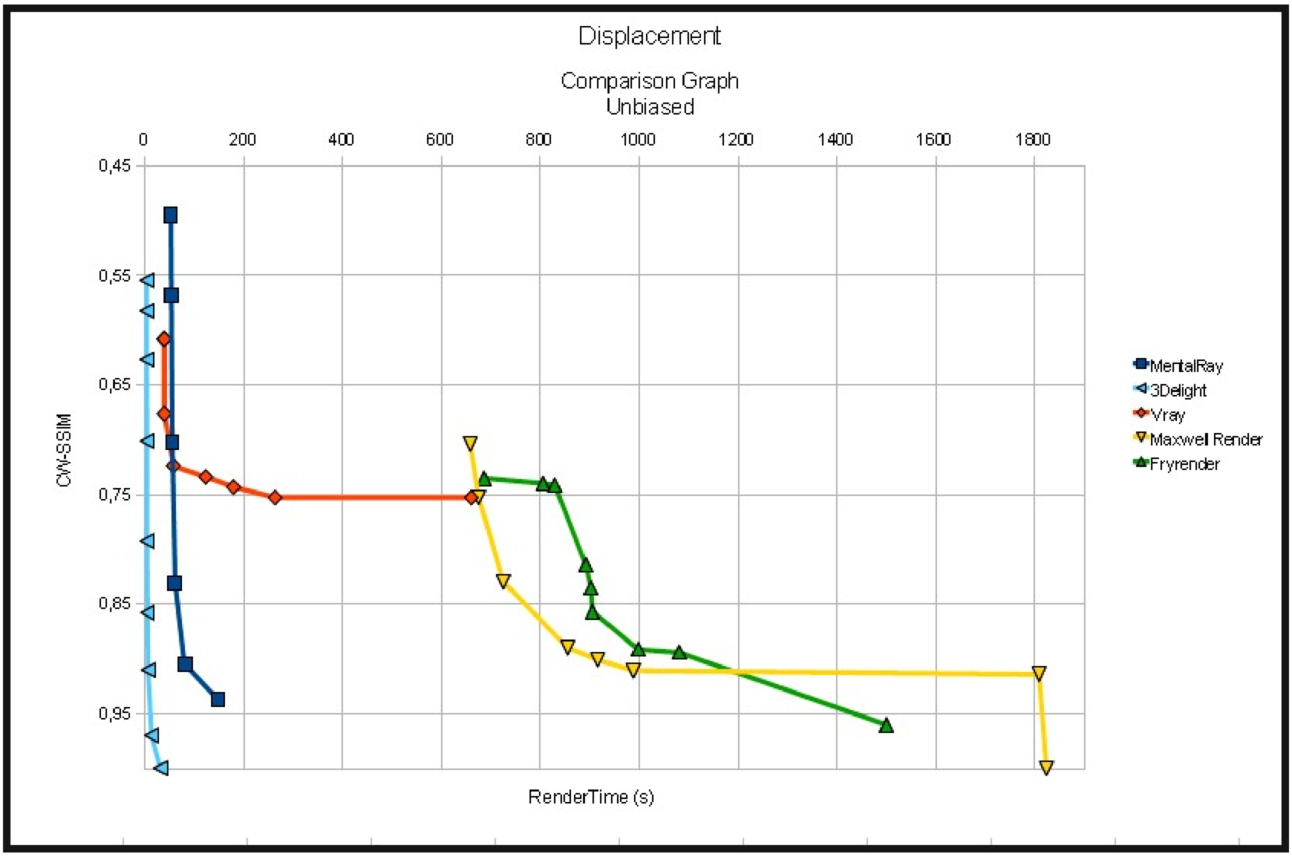
Sampling level = 7, Smooth MPDM

* = Enhaced MPDM enabled

Render without displacement = 29s

For a more precise representation of the values shown, refer to the excel sheet located in the corresponding section's folder named "tables.ods"

Once the table is built, a graph can be drawn for a more intuitive representation of the results obtained. Please refer to "Error Metrics" chapter in this document for a more detailed explanation.



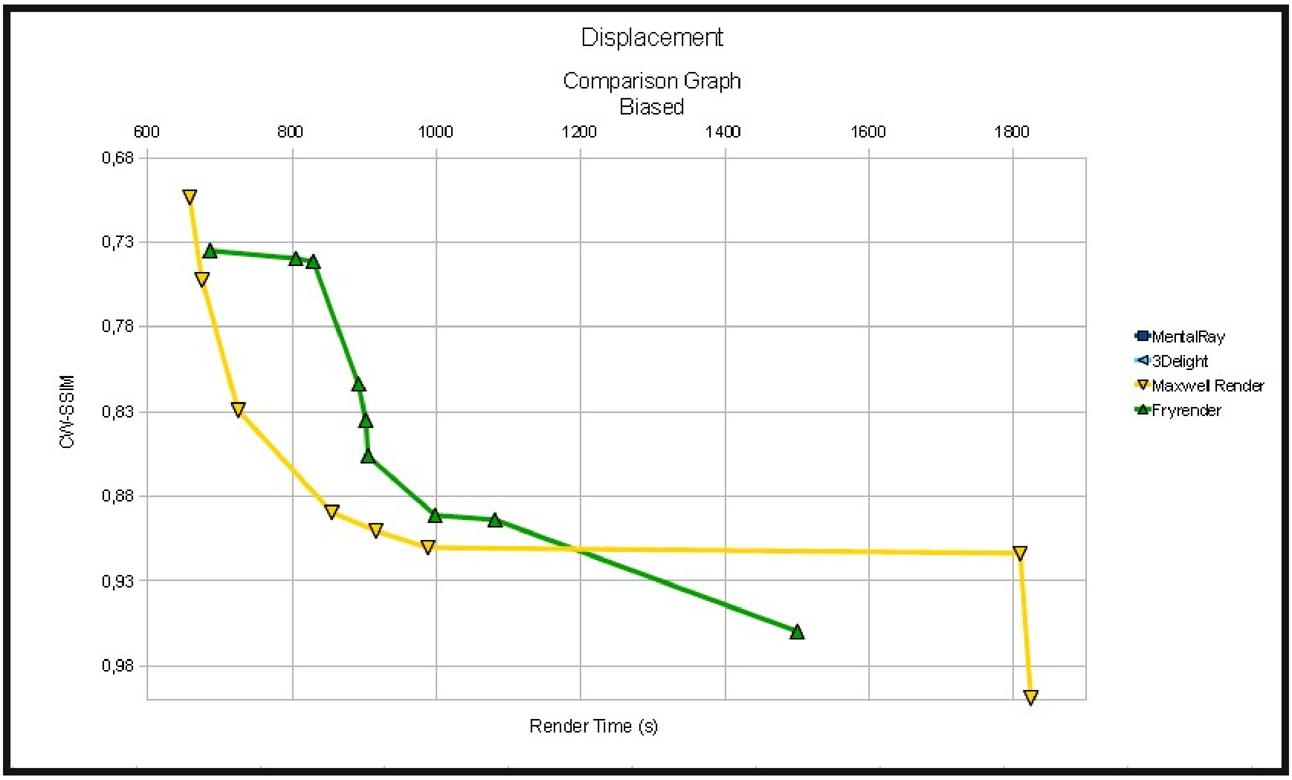
Displacement Mapping Comparison Graph.

Note how the Reyes based render engine (3Delight) performs the best in displacement calculation. Surfaces are displaced in almost costless with this render engine, as can be seen from the above graph.
It reaches a CW-SSIM value of 100% in 31 seconds, while its other competitors won't start even rendering before that time.

The next render engine, MentalRay, starts rendering at 49% CW-SSIM value in 52 seconds, and does not reach a value of 0,94 until two and a half minutes. Beyond that point, the maximum RAM usage is exceeded *[See Hardware specification section]* and it was impossible to render our scene with higher level of displacement quality with the workstation used throughout this document. That is an important counterpoint that must be taken into consideration when big scenes are to be rendered, and enough displacement detail is needed.

Vray render engine, however, is the worst competitor in displacement calculation from the biased renderers used in this study. It reaches a CW-SSIM value of 75% in 10 minutes of render, and could not be possible to go beyond that point because of memory limitations. It exceeded the maximum RAM usage at 75% of image quality, while MentalRay exceeded it at 94%

Unbiased render engines, however, start rendering about 600 seconds later than biased render engines, suffering from a big hit in performance compared to biased renderers. The next graph focuses on unbiased render engines to offer a better appreciation of their developement over time.



Displacement Mapping Detailed Coparison Graph.

During the first interval of 20 minutes, MaxwellRender performs better overall than FryRender. The latter shows an irregular behaviour throughout the timerange, but goes beyond MaxwellRender after 20 minutes of rendering, reaching a CW-SSIM value of 96% in 1500 seconds, while MaxwellRender stays around at that time 90%. (It reaches a CW-SSIM value of 91% in 1809 seconds.)

Without a big impact in rendertime (15 seconds), however, MaxwellRender reaches a CW-SSIM of 100%. That can be achieved because of MaxwellRender's displacement special feature, Adaptative Displacement.

The adaptive option in MaxwellRender locks the precision value to the given texture detail (at half pixel accuracy), which has the advantage of always creating the most detailed displacement that a given texture can provide. The user does not have to guess what the maximum precision value should be for that texture, or worry about exceeding it (which would increase render times but would not necessarily increase image detail, see example above).

The adaptive mode should be used with care, because using a very large-resolution texture to represent some simple detail will result in unnecessarily long render times. The larger your texture, the longer the render times with Adaptive mode on because it will always render the maximum amount of detail for that particular texture.

Global Illumination Comparison.

Global illumination refers to techniques that take indirect lighting, such as inter-object reflections, into account when computing the light distribution in a scene.

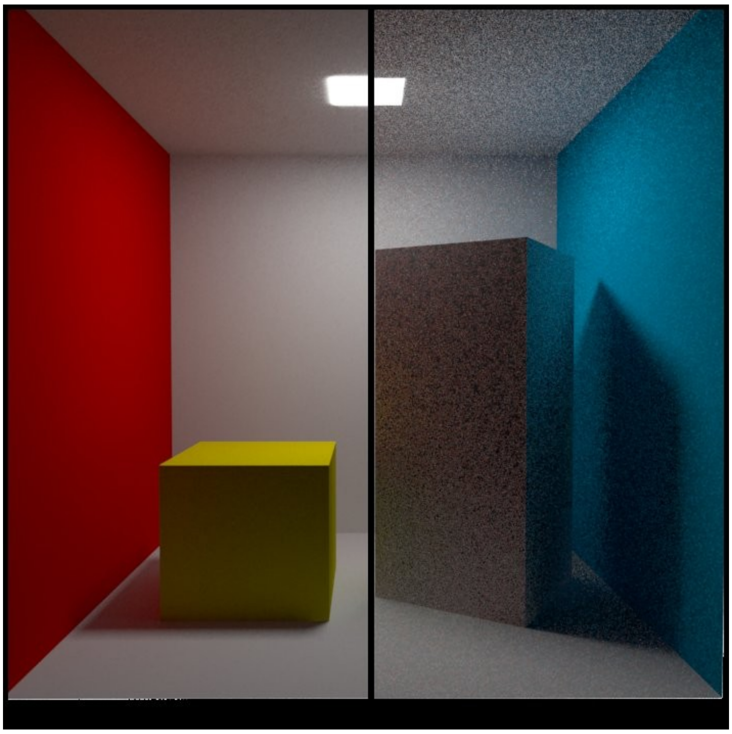
Therefore, such techniques take into account not only the light which comes directly from a light source (direct illumination), but also subsequent cases in which light rays from the same source are reflected by other surfaces in the scene, whether reflective or non (indirect illumination).
[See Global Illumination Chapter in Glossary of Terms].

Radiosity, Raytracing, Photon mapping, Pathtracing, Metropolis Light Transport, Irradiance Caching, are examples of algorithms used in global illumination, some of which may be used together to yield more accurate results.

For a complete list of the render engines included in this comparison, and Global Illumination algorithms they provide, please refer to *Software Specification* Chapter of this study.

The render engines used in this study implement several approaches for computing indirect illumination with different trade-offs between quality and speed:

- **Brute force** - this is the simplest approach; indirect illumination is computed independently for each shaded surface point by tracing a number of rays in different directions on the hemisphere above that point.



Differences between Low and High quality GI in BruteForce calculation.
(Vray Render)

Advantages:

- this approach preserves all the detail (e.g. small and sharp shadows) in the indirect lighting;

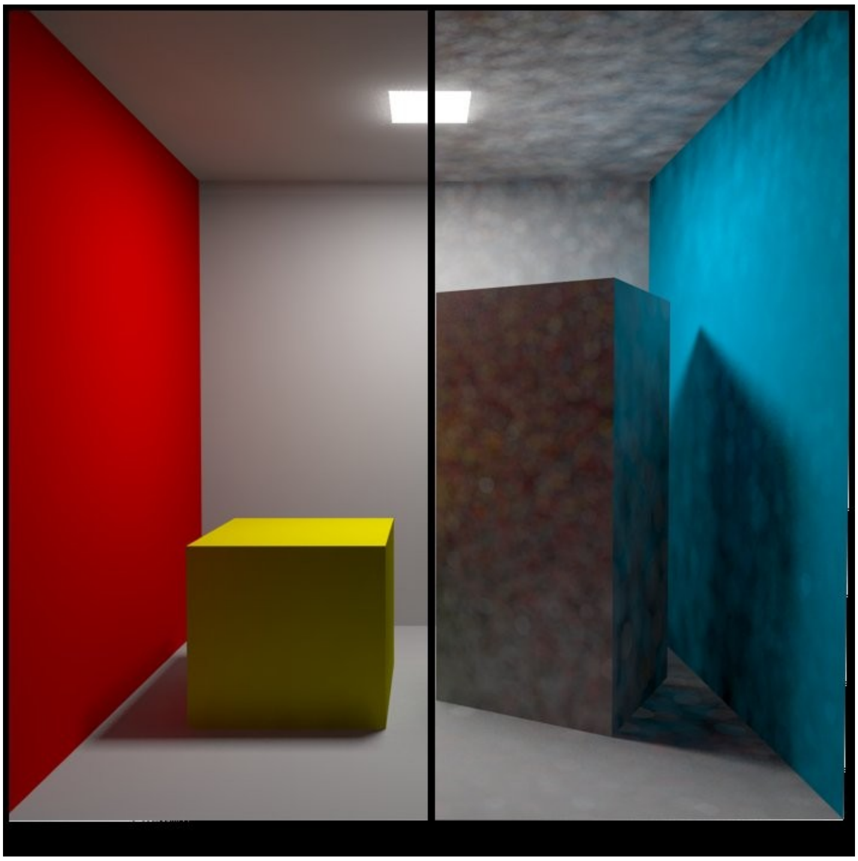
  - it is free from defects like flickering in animations;
  - no additional memory is required;
  - indirect illumination in the case of motion-blurred moving objects is computed correctly.

  Disadvantages:

  - the approach is very slow for complex images (e.g. interior lighting);
  - it tends to produce noise in the images, which can be avoided only by shooting a larger number of rays, thus slowing it even more.

- **Final gather.** Diffuse reflections occur when light bounces of in all directions from non-reflective surfaces. In most cases this effect should somehow be taken into account to get a nice, realistically looking rendering. Final Gather is a method to approximate the diffuse reflections that occur in the real-world, while trying to keep render times as low as possible by using a smart calculation model.

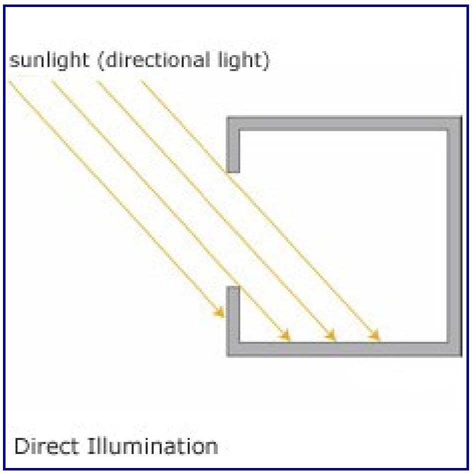
  Final Gather works best for scenes that have (relative) uniformly distributed light, typically: daylight exteriors and interior scenes with a fairly large area for incoming daylight or other lightsources with a large area. With Final Gather the calculations of light are divided in two components:

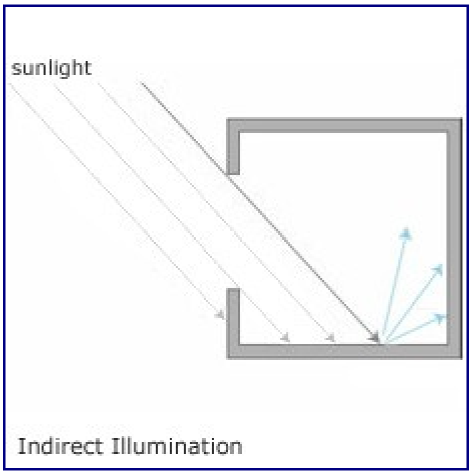


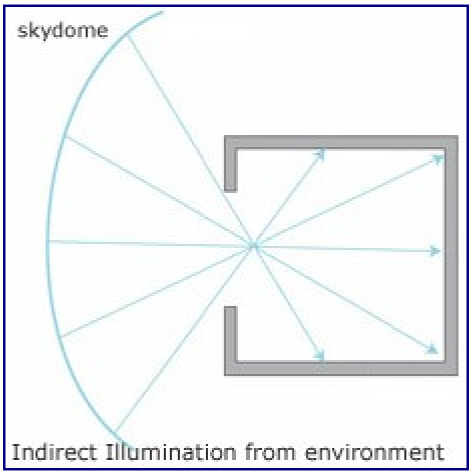
Differences between Low and High quality GI in FinalGather calculation. (MentalRay Render)

  1. Direct Illumination
  2. Indirect Illumination
     - Indirect Illumination from other surfaces (light bounces of direct light)

- Indirect Illumination from the environment (eg: skydome)







Advantages:

- Fewer photons need to be cast in the photon tracing phase since the demandos on irradiance estimate accuracy are much lower (500-2000 of them are averaged). This makes photon tracing phaster, and reduces the amount of memory required for the photon map.
- The illumination accuracy is higher than for a direct photon map lookup, specially near edges and corners of objects.

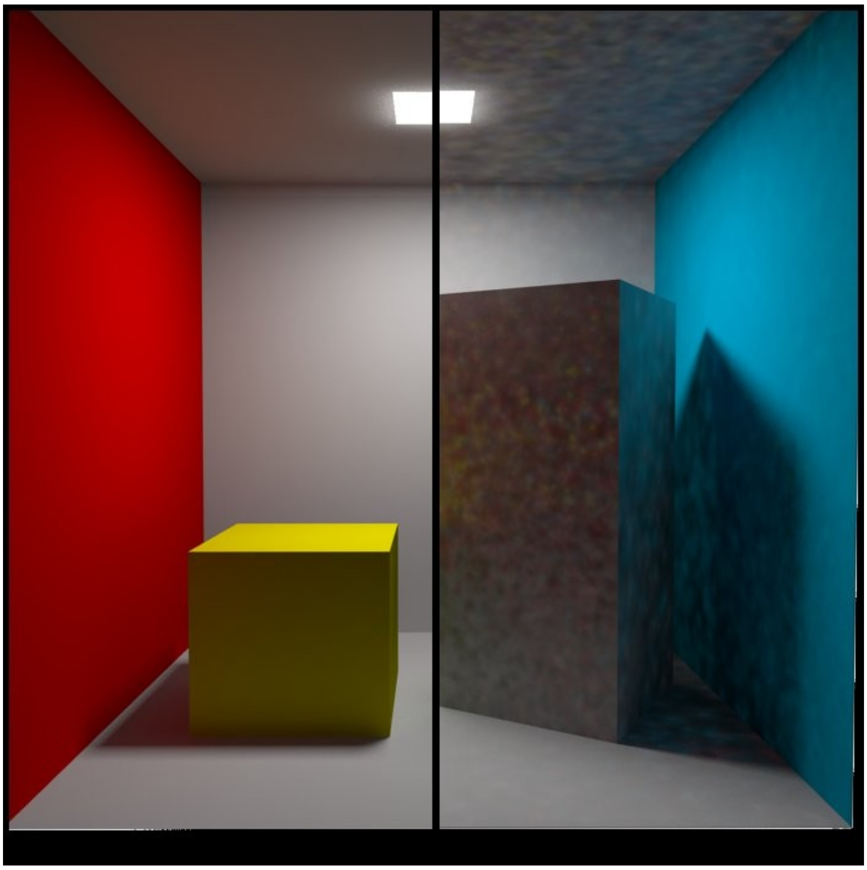
Disadvantages:

- Without global illumination, rendering takes much longer than with just direct illumination. It is not a real problem here, where global illumination is compared.
- With global illumination, depending on the accuracy settings, the rendering pass can take longer than without final gathering, depending on the required accuracy of the solution and the number of photons used to estimate irradiance. The performance increase or decrease is a trade-off between faster photon map generation and higher quality, and slower rendering.

- **Irradiance map** - this approach is based on irradiance caching; the basic idea is to compute the indirect illumination only at some points in the scene, and interpolate for the rest of the points.

  Advantages:

  - the irradiance map is very fast compared to direct computation, especially for scenes with large flat areas;
  - the noise inherent to direct computation is greatly reduced;
  - the irradiance map can be saved and re-used to speed up calculations of different views for the same scene and of fly-through animations;
  - the irradiance map can also be used to accelerate direct diffuse lighting from area light sources.



Differences between Low and High quality GI in Irradiance Map calculation. (MentalRay Render)

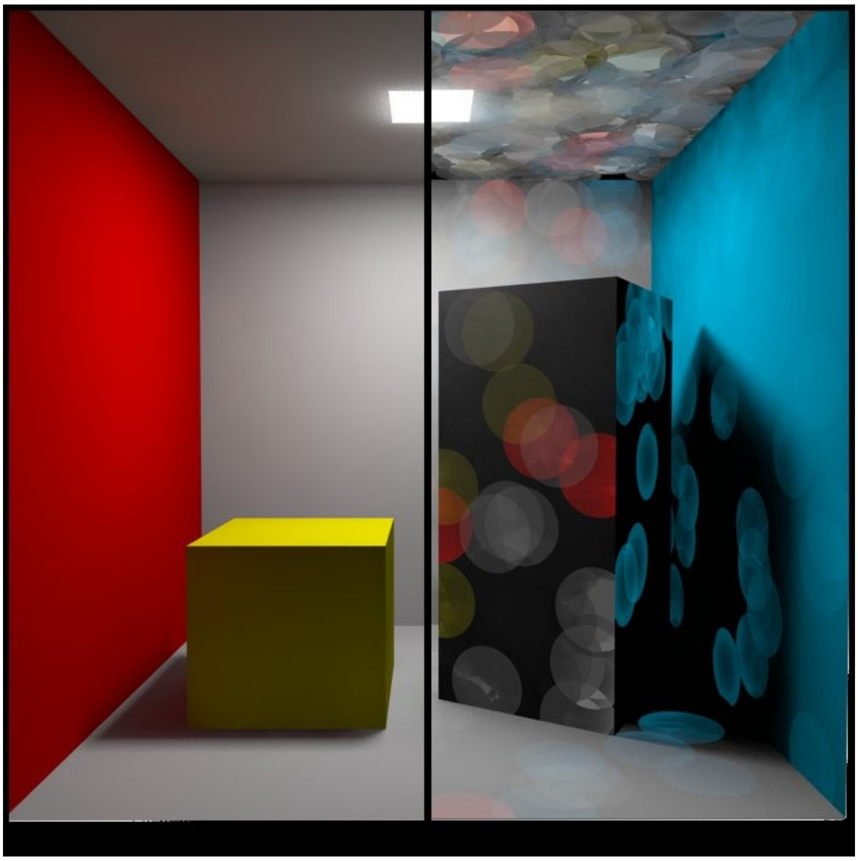
Disadvantages:

  - some details in indirect lighting can be lost or blurred due to the interpolation;
  - if low settings are used, flickering may occur when rendering animations;
  - the irradiance map requires additional memory;
  - indirect illumination with motion-blurred moving objects is not entirely correct and may lead to noise (although in most cases this is not noticeable).

- **Photon map** - this approach is based on tracing particles starting from the light sources and bouncing around the scene. This is useful for interior or semi-interior scenes with lots of lights or small windows. The photon map usually does not produce good enough results to be used directly; however it can be used as a rough approximation to the lighting in the scene to speed the calculation of GI through direct computation or irradiance map.

  Advantages:

  - the photon map can produce a rough approximation of the lighting in the scene very quickly;
  - the photon map can be saved and re-used to speed up calculation of different views for the same scene and of fly-through animations;
  - the photon map is view-independent.



Differences between Low and High quality GI in PhotonMap calculation. (MentalRay Render)

Disadvantages:

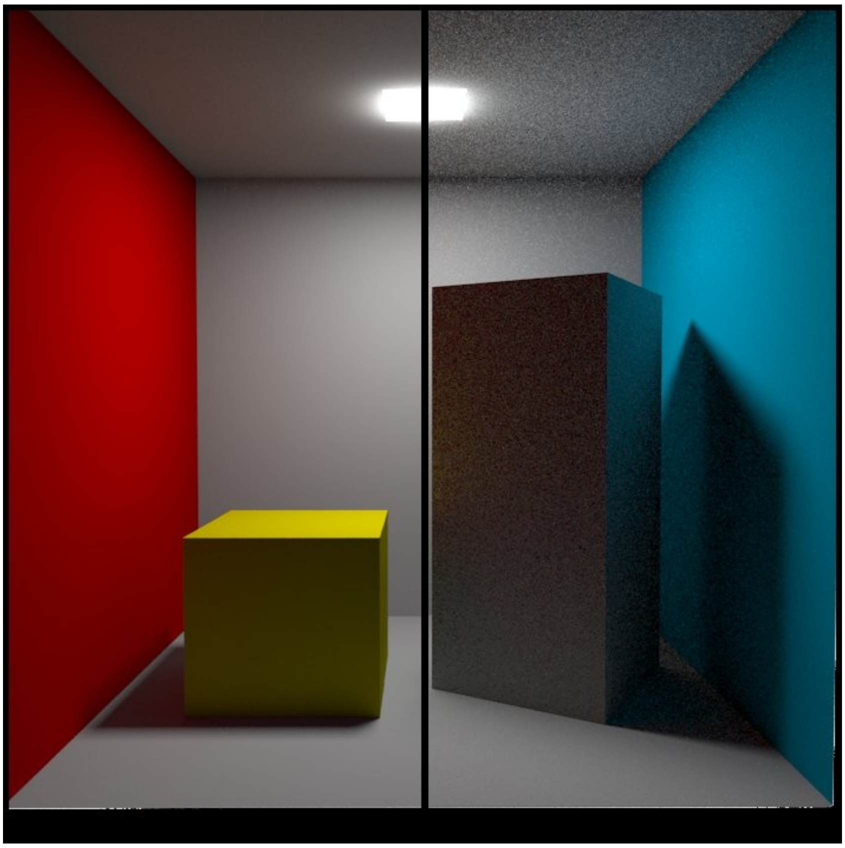
- the photon map usually is not suitable for direct visualization;
- requires additional memory;
- in V-Ray's implementation, illumination involving motion-blurred moving objects is not entirely correct (although this is not a problem in most cases).
- the photon map needs actual lights in order to work; it cannot be used to produce indirect illumination caused by environment lights (skylight).

- **Light cache** - light caching is a technique for approximating the global illumination in a scene. It is very similar to photon mapping, but without many of its limitations. The light map is built by tracing many many eye paths from the camera. Each of the bounces in the path stores the illumination from the rest of the path into a 3d structure, very similar to the photon map. The light map is a universal GI solution that can be used for both interior or exterior scenes, either directly or as a secondary bounce approximation when used with the irradiance map or the brute force GI method.

  Advantages:

  - the light cache is easy to set up. We only have the camera to trace rays from, as opposed to the photon map, which must process each light in the scene and usually requires separate setup for each light.

- the light-caching approach works efficiently with any lights - including skylight, self-illuminated objects, non-physical lights, photometric lights etc. In contrast, the photon map is limited in the lighting effects it can reproduce - for example, the photon map cannot reproduce the illumination from skylight or from standard omni lights without inverse-square falloff.
- the light cache produces correct results in corners and around small objects. The photon map, on the other hand, relies on tricky density estimation schemes, which often produce wrong results in these cases, either darkening or brightening those areas.
- in many cases the light cache can be visualized directly for very fast and smooth previews of the lighting in the scene.



Differences between Low and High quality GI in Light Cache calculation.
(Vray Render)

Disadvantages:

- like the irradiance map, the light cache is view-dependent and is generated for a particular position of the camera. However, it generates an approximation for indirectly visible parts of the scene as well - for example, one light cache can approximate completely the GI in a closed room;
- like the photon map, the light cache is not adaptive. The irradiance is computed at a fixed resolution, which is determined by the user;
- the light cache does not work very well with bump maps; use the irradiance map or brute force GI if you want to achieve better results with bump maps.

  - lighting involving motion-blurred moving objects is not entirely correct, but is very smooth since the light cache blurs GI in time as well (as opposed to the irradiance map, where each sample is computed at a particular instant of time).

- **Spherical Harmonics** - this approach is based on rendering light information, already precomputed in *.vrsh file by spherical harmonics baking engine. This method is suitable for flight-through animations or animations with changing environment. It is applicable to open scenes only.

  Advantages:

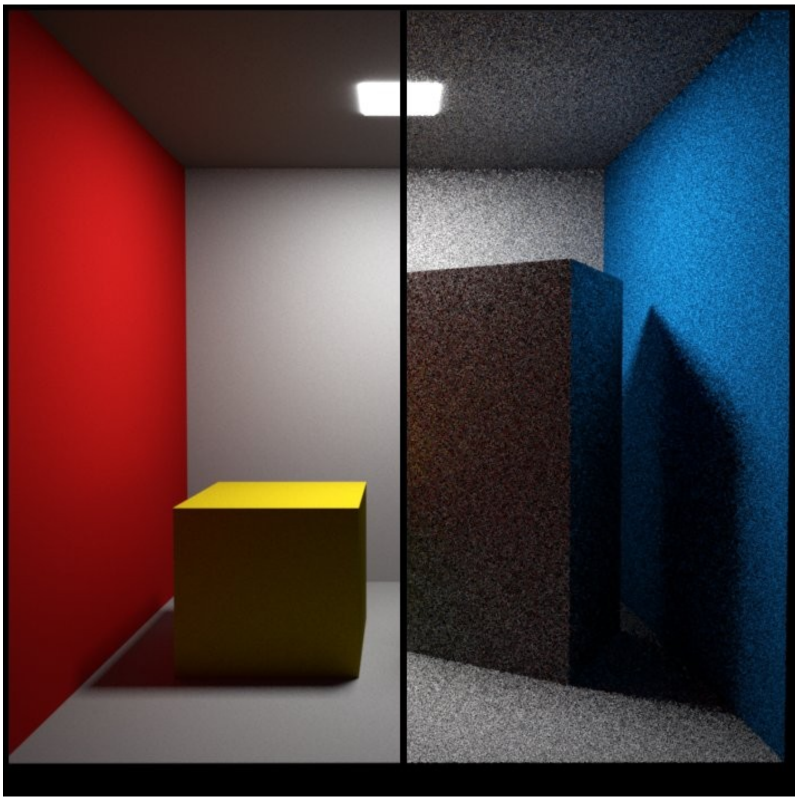
  - this method produces smooth result (no noise at all).
  - spherical harmonics are saved to file and could be used to speed up the calculations in flight-through animations and animations with per frame changing environment.
  - it is view-independent method.

  Disadvantages:

  - this method is not suitable for high frequency lighting.
  - it can'T be used for closed scenes.
  - it could produce unwanted artifacts on closely placed surfaces, due to their coarse subdivision.
  - it captures diffuse indirect illumination only

  - **Metropolis light transport (MLT)** - is a method for computing physically accurate behaviour of light who described an application of a variant of the Monte Carlo method called the Metropolis-Hastings algorithm to the rendering equation for generating images from detailed physical descriptions of three dimensional scenes.

    The procedure constructs paths from the eye to a light source using bidirectional path tracing [see path tracing section], then constructs slight modifications to the path. Some careful statistical calculation (the Metropolis algorithm) is used to compute the appropriate distribution of brightness over the image.



Differences between Low and High quality GI in MLT calculation. (FryRender render)

Advantages:

- Produce very accurate results.
- The only artifact these methods produce is noise.
- Renderers using exact methods typically have only few controls for specifying image quality.
- Typically require very little additional memory.

Disadvantages:

- Unbiased methods are not adaptive and so are extremely slow for a noiseless image.
- Some effects cannot be computed at all by an exact method (for example, caustics from a point light seen through a perfect mirror).
- It may be difficult to impose a quality requirement on these methods.
- Exact methods typically operate directly on the final image; the GI solution cannot be saved and re-used in any way.

- **Point Cloud** - a point cloud is a cloud of points in 3d space that contains one or more channels of data (lighting, occlusion, area, etc) at each point.

  Point clouds are important for many purposes beyond point-based rendering, as they are simply a useful structure for caching 3d textures, and 3Delight is very efficient at generating point clouds because of the way the REYES algorithm dices geometry.

Point Cloud capable renderers can simply write a point to the point cloud, that contains the point's location in space, its normal, and any data that the user specifies.

First, a pre-pass generates a point cloud of the scene geometry and illumination in a process informally called "baking a point cloud." Each point includes data about the area of the micropolygon and its surface normal. Once that data is generated it can be cached and reused.

The second phase occurs at render time, when the baked point cloud is referred to by shaders attached to the scene geometry to calculate the point-based effect. Each of these points is then approximated with an oriented disk in space.

It is simple to compute the analytical color and occlusion contribution of a disk without having to use ray tracing. For efficiency, distant disks, which individually have a weak contribution to the overall result, are clustered and treated as one entity. The inter-disk occlusion calculation also computes the average non-occluded normal which is used for an environment lookup for image based lighting.

This is analogous to the same functionality in ray-traced occlusion. An added benefit of point-based rendering is that color bleeding calculations can be computed nearly as quickly as occlusion due to the fact that shaders only need to be run when the colors are baked into a point.

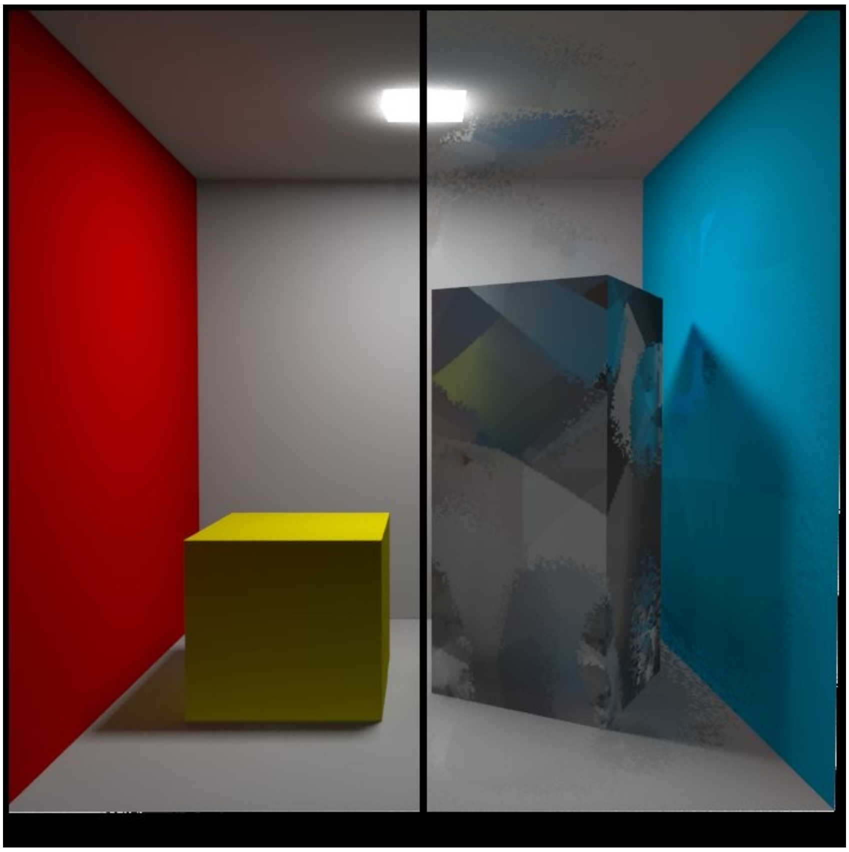
Advantages:

- Each point includes data about the area of the micropolygon and its surface normal. Once that data is generated it can be cached and reused.

- An added benefit of point-based rendering is that color bleeding calculations can be computed nearly as quickly as occlusion due to the fact that shaders only need to be run when the colors are baked into a point cloud, which is a big time saving over ray tracing. The only thing the algorithm needs is for the point baking pass to include the radiance at each point (surface color and illumination).
- Color bleeding calculations can be computed nearly as quickly as occlusion due to the fact that shaders only need to be run when the colors are baked into a point.
- Noise free images - high frequency "spotty" artifacts associated with ray tracing are avoided.
- Faster computation times - 4x to 10x speedup versus ray tracing large scenes.
- Color bleeding and image-based lighting (including support for HDRI) - nearly as fast as ambient occlusion.
- Lowered memory requirements - geometry does not need to be visible for ray tracing.
- Displaced surfaces - no penalty is incurred for rendering displacements, a huge advantage over ray tracing.
- Lights and objects can be easily included or excluded from computation.

Disadvantages:

- The effect can sometimes compute too much occlusion and color bleeding, because the points are being used to approximate the actual scene geometry. This can lead to an over darkening from occlusion or stray color bleeding from hidden surfaces.



Differences between Low and High quality GI in PointCloud GI calculation.

(3Delight Render)

Primary and secondary bounces

The indirect illumination controls in biased render engines are divided into two large sections: controls concerning primary diffuse bounces and controls concerning secondary diffuse bounces. A primary diffuse bounce occurs when a shaded point is directly visible by the camera, or through specular reflective or refractive surfaces. A secondary bounce occurs when a shaded point is used in GI calculations.

Once the GI methods available in this study have been explained, the following pages will show the results obtained in the comparison for GI calculation of each render engine.

Please refer to "Renders" subfolder for a complete list of rendered images used in this part of the study for each rendere engine involved in the comparison.

The following table shows CW-SSIM vs RenderTime values in nine different renders for each render engine and Global Illumination method involved in this study.

Please note that values given in the table have been rounded for a better and faster appreciation .

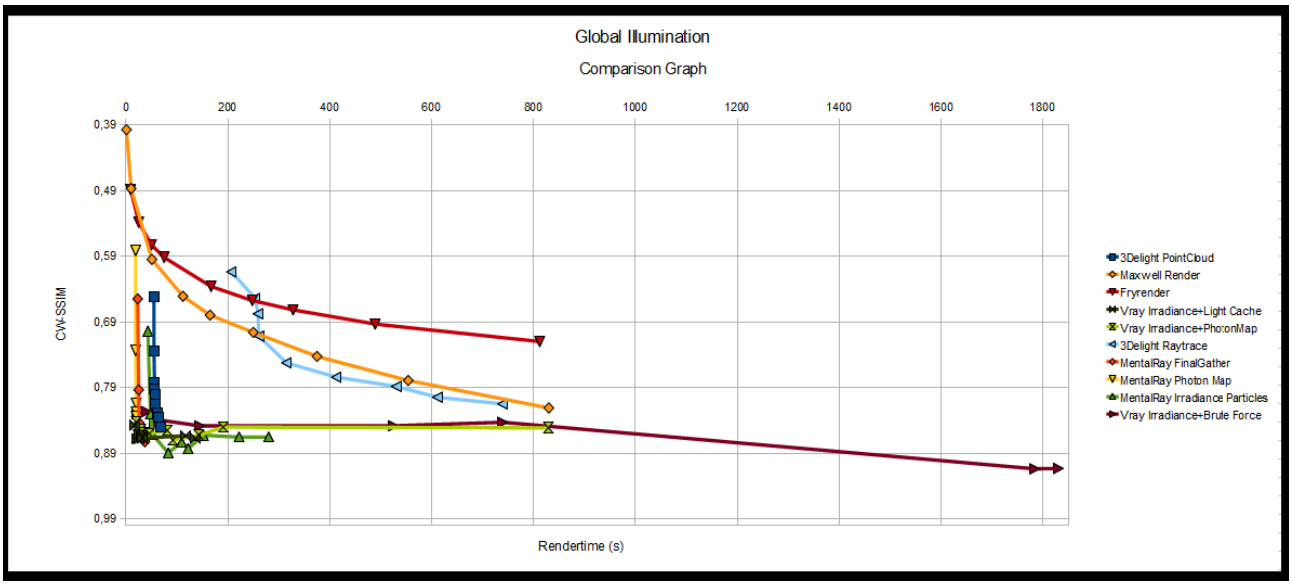
	1	2	3	4	5	6	7	8	9	
3Delight Raytrace										
Render Time (m,s)	207	254	259	261	315	413	530	611	739	
CW-SSIM	0,61	0,65	0,68	0,71	0,75	0,77	0,79	0,8	0,81	
ColorBleeding Samples	4	8	16	32	64	96	128	160	192	
3Delight PointCloud										
Render Time	55	55	56	56	57	58	62	64	69	
CW-SSIM	0,65	0,73	0,78	0,79	0,8	0,82	0,83	0,84	0,85	
MaxSolidAngle Value*	5	1	0,5	0,4	0,3	0,2	0,1	0,08	0,05	Pixel Samples = 3-3; Shading rate=1.0; Pixel Filter = Sinc; AO maxSolidAngle = 0.03;
MentalRay FinalGather										
Render Time	23	25	26	30	32	33	34	36	37	
CW-SSIM	0,65	0,79	0,84	0,86	0,86	0,86	0,87	0,87	0,87	
Accuracy	1	10	30	50	60	70	80	90	100	Secondary Difuse Bounces = 1; Point Density = 1,5 ; Point Interpolation = 20; Fg Filter = ;1
MentalRay Photon Map										
Render Time	19	19	20	20	20	24	28	33	38	
CW-SSIM	0,58	0,73	0,81	0,83	0,83	0,86	0,86	0,86	0,86	
GI Photons	1k	10k	50k	75k	100k	250k	500k	750k	1M	GI Accuracy 5M; Direct Illumination Shadow Effects = enabled
MentalRay Irradiance Particles										
	*	*	*	*	**	**	**	***	***	
Render Time	43	48	52	83	109	122	152	222	280	
CW-SSIM	0,7	0,83	0,86	0,89	0,87	0,88	0,86	0,87	0,87	** = Importon Density = 0.5;
Irradiance Rays	8	64	128	512	32	64	128	64	128	Indirect passes = 1; Interpoints = 32;
Vray Irradiance+Brute Force										
	*	*	*	**	***	***	***	*	**	
Render Time	21	40	55	55	145	525	740	1785	1830	
CW-SSIM	0,83	0,83	0,85	0,84	0,85	0,85	0,84	0,91	0,91	** = Brute Force Subdivs = 100;
Min/Max Rate	Mn -4 Mx -3	Mn -3 Mx -1	Mn -3 Mx 0	Mn -3 Mx 0	Mn -4 Mx -3	Mn -3 Mx -2	Mn -3 Mx -1	Mn -3 Mx 1	Mn -3 Mx 1	Brute Force Depth = 3 ; Irradiance Interpolation Samples = 20 ; Irradiance Subdivs= 100;
Vray Irradiance+Light Cache										
	*	***	*	***	***	*	**	*	**	
Render Time	18	22	26	26	29	33	33	117	136	
CW-SSIM	0,85	0,87	0,87	0,86	0,87	0,86	0,87	0,86	0,87	** = Light Cache Subdivs = 500;
Subdivs	Mn -4 Mx -3	Mn -4 Mx -3	Mn -3 Mx -1	Mn -3 Mx 2	Mn -3 Mx -1	Mn -3 Mx 0	Mn -3 Mx 0	Mn -3 Mx 1	Mn -3 Mx 1	Light Cache sample size = 0.020 ; Irradiance Interpolation Samples = 20 ; Irradiance Subdivs= 100;
Vray Irradiance+PhotonMap										
	*	*	*	*	*	**	**	*	*	
Render Time	20	29	34	46	81	92	143	191	829	
CW-SSIM	0,84	0,85	0,86	0,86	0,86	0,87	0,86	0,85	0,85	

Subdivs	Mn -4 Mx -3	Mn -3 Mx -2	Mn -3 Mx -1	Mn -3 Mx 0	Mn -2 Mx 0	Mn -3 Mx -1	Mn -3 Mx 0	Mn -1 Mx 0	Mn -3 Mx 1	Brute Force Depth = 3 ; Irradiance Interpolation Samples = 20 ; Irradiance Subdivs= 100;
										Photon Map Bounces = 100; Retrace Bounces = 100; Store Direct Light = on;

Maxwell Render										
Render Time	1	10	51	112	165	250	375	554	830	
CW-SSIM	0,4	0,49	0,59	0,65	0,68	0,71	0,74	0,78	0,82	
Sampling Level	1	4	8	10	11	12	13	14	15	
Fryrender										
Render Time	9	25	50	75	167	248	328	489	812	
CW-SSIM	0,49	0,54	0,57	0,59	0,64	0,66	0,67	0,69	0,72	
Passes	1	3	6	9	20	30	40	60	100	

For a more precise representation of the values shown, refer to the excel sheet located in the corresponding section's folder named "tables.ods"

Once the table is built, a graph can be drawn for a more intuitive representations of the results obtained. Please refer to "Error Metrics" chapter in this document for a more detailed explanation.

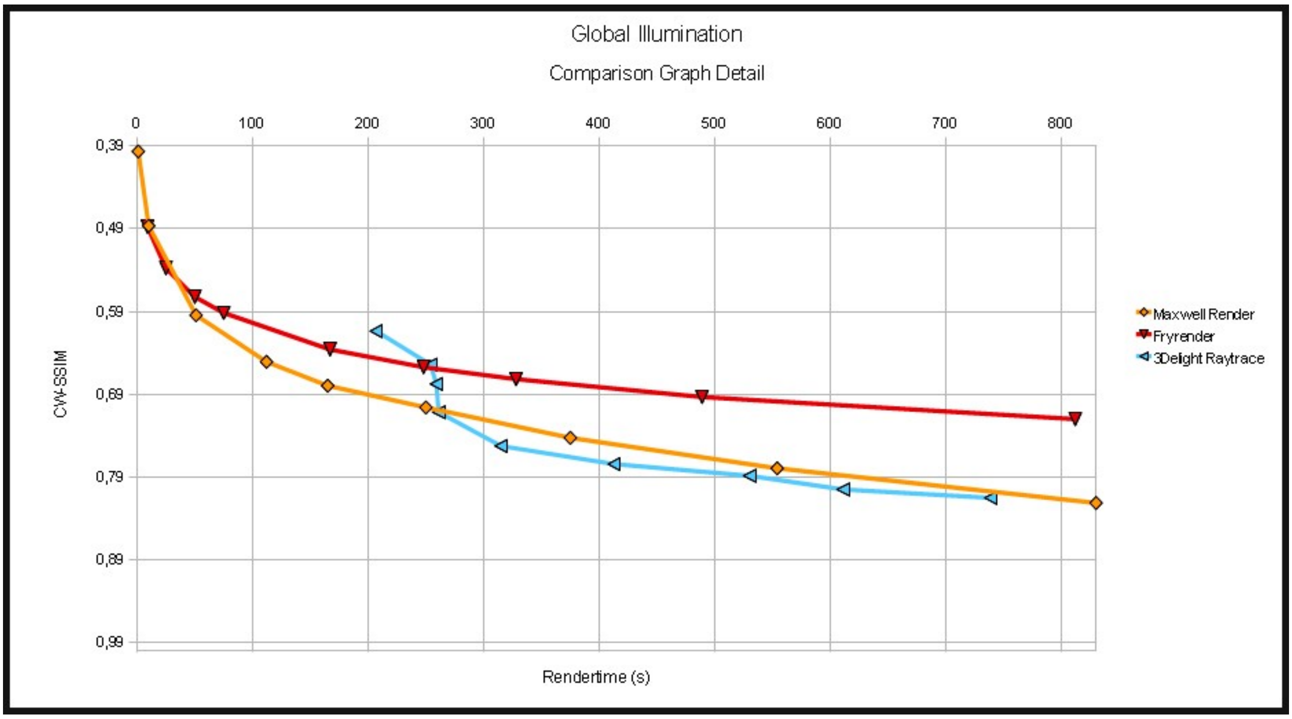


Global Illumination Comparison Graph.

The above graph shows a complete representation of global illumination methods compared in this chapter. Please note the legend on the righten side of the image for color curve associations.

Because of the lineal scale taken to build the graph, curves often appear too close together to clearly notice the differences between render engines. Therefore, it would be a good idea to build a more detailed graph focusing on those render engines and methods that are too close together, in order to offer a clearer representation of the results obtained in the comparison.

The following image represents the values obtained for 3Delight Raytrace, MaxwellRender and FryRender in the Global Illumination comparison.



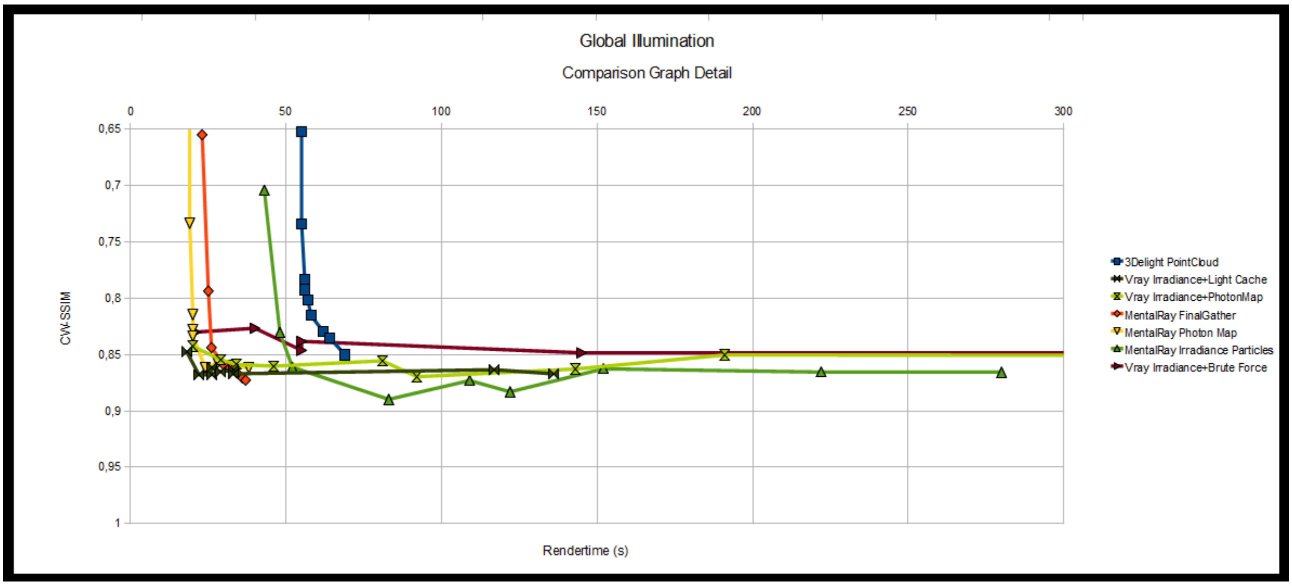
Global Illumination Detailed Comparison Graph no.1

Although being a biased render engine 3Delight Raytrace implementation of Global Illumination stays closer to unbiased results than to biased GI performance. Traditional raytracing Global Illumination calculation in a Reyes based render engine should be avoided if possible, and PointCloud based solutions should be used instead.
That will be explained later in the next graph, with 3Delight PointCloud's respresentation.

However, note that 3Delight raytrace stays in a slightly better position than unbiased renderers after the first 5 minutes of rendering, followed by MaxwellRender.

On the other hand, FryRender starts similar to MaxwellRender, but after the first minute of rendering it starts separating from the other unbiased renderer taking the worst possition in the comparison.

The other biased methods used in this comparison converge much faster to a good result, as can be seen from the below graph, which focuses in Vray, MentalRay and 3Delight PointCloud GI render methods.



Global Illumination Detailed Comparison Graph no.2

Note that Vray's BruteForce + Light Cache is the worse competitor here. It reaches a CW-SSIM value of 80% in 56 seconds, quite similar to 3Delight PointCloud. However, 3Delight PointCloud implementation goes beyond that CW-SSIM value without much time effort. It reaches a quality of 85% in 69 seconds.
Please, refer to "References" section in this chapter for the complete implementation of the Light Shader used in 3Delight for PointCloud Global Illumination calculations.

MentalRay Photon map implementation of Global Illumination, however, takes about 22 seconds to reach a quality level of 85% (47 seconds earlier than 3Delight PointCloud). Reaching a CW-SSIM value of 86% 16 seconds later, at 38".

Even better than that, MentalRay Finalgather reaches a CW-SSIM value of 87% in 37 seconds, though being slightly slower in the first seconds of rendering (Reaches 85% CW-SSIM value in about 26 seconds, ~4 senconds later than MentalRay using photon mapping).

Vray render engine methods perform something better than MentalRay FinalGather and the other biased methods in the first seconds of rendering, but they tend to stop improving the image quality given a certain moment.
For example, Vray Irradiance + Light Cache reaches a CW-SSIM value around 87% in the first 22 seconds and stays in that value through all the timerange. The same happens with Vray Irradiance + BruteForce (85% beyond the first 55 seconds), and Vray Irradiance + PhotonMap (~85% beyond the first 29 seconds).

References.

Ptc_Uber_Light Shader*

```
light
   ptc_uber_light (
            string filename = "<project>/3delight/<scene>/ptc/<scene>.#.ptc";

            string envmap = "";
            string coordsyst = "world";
            string envspace = "world";
            string AO_hitsides = "both";
            float AO_maxdist = 1e2,
            AO_int = .5,
            AO_coneangle = 180,
            AO_falloff = 1,
            AO_falloffmode = 1,
            AO_bias = 0.01,
            AO_clamp = 1,
            AO_maxsolidangle = 0.05,
            AO_maxvariation = 0,
            AO = 0;
            string DO_hitsides = "both";
            float DO_maxdist = 1e2,
            DO_int = .5,
            DO_coneangle = 180,
            DO_falloff = 1,
            DO_falloffmode = 1,
            DO_bias = 0.01,
            DO_clamp = 1,
            DO_maxsolidangle = 0.05,
            DO_maxvariation = 0,
            DO = 0;
            normal DO_dir = normal "world" (0,1,-2);

            string RO_hitsides = "both";
            float RO_maxdist = 1e15,
            RO_int = 1,
            RO_falloff = 1,
            RO_falloffmode = 1,
            RO_bias = 0.01,
            RO_clamp = 1,
            RO_maxsolidangle = 0.05,
            RO_coneangle = 30,
            RO = 0;
            string GRefl_hitsides = "both";
            float GRefl_maxdist = 1e15,
            GRefl_int = 1,
            GRefl_falloff = 1,
            GRefl_falloffmode = 1,
            GRefl_bias = 0.01,
            GRefl_clamp = 1,
            GRefl_sort = 1,
            GRefl_maxsolidangle = 0.05,
            GRefl_coneangle = 30,
            GRefl = 0;
            string GRefr_hitsides = "both";
            float GRefr_maxdist = 1e15,
            GRefr_readme = 0,
            GRefr_int = .3,
            GRefr_ior = 1.1,
            GRefr_falloff = 1,
            GRefr_falloffmode = 1,
            GRefr_bias = 0.01,
            GRefr_clamp = 1,
            GRefr_sort = 1,
            GRefr_maxsolidangle = 0.05,
            GRefr_coneangle = 0.1,
            GRefr = 0;
            string IBL_hitsides = "both";
            float IBL_maxdist = 1e15,
            IBL_int = 1,
            IBL_coneangle = 180,
            IBL_falloff = 1,
            IBL_falloffmode = 1,
            IBL_bias = 0.01,
            IBL_clamp = 1,
            IBL_maxsolidangle = 0.05,
            IBL_maxvariation = 0,
            IBL = 0;
            string GI_hitsides = "both";
            float GI_maxdist = 1e15,
            GI_falloff = 1,
            GI_falloffmode = 1,
            GI_bias = 0.045,
            GI_clamp = 1,
            GI_sort = 1,
            GI_coneangle = 180,
            GI_maxsolidangle = .2,
            GI_int = 1,
            GI = 0;
            color SSS_dmfp = color(8.51, 5.57, 3.95);

            color SSS_al = color(0.83, 0.79, 0.753);
```

```
        float SSS = 0,
        SSS_ior = 1.5,
        SSS_scale = 1,
        SSS_weight = .4;
        float reuse = 0,
        rebake = 0;
        string rebaked_file = "<project>/3delight/<scene>/ptc/<scene>.#.rebake.ptc",

        rebaked_coordsyst = "world";
        output varying color GI_Cl = 0;
        output varying color GRefl_Cl = 0;
        output varying color GRefr_Cl = 0;
        output varying color RO_Cl = 0;
        output varying color IBL_Cl = 0;
        output varying color AO_Cl = 1;
        output varying color DO_Cl = 1;
        output varying color SSS_Cl = 1;
        output varying color aov_envcolor = 1)
  {
  normal shading_normal = normalize(Ns);

   if(AO == 1) {
    if (reuse != 1) {
    AO_Cl = AO_int*(1-occlusion( // performs occlusion with parameters from GUI, inverse AO
and multiply by our intensity
        Ps, shading_normal, 0,
        "pointbased", 1,
        "filename", filename,
        "coordsystem", coordsyst,
        "coneangle", radians(AO_coneangle),
        "hitsides", AO_hitsides,
        "maxdist", AO_maxdist,
        "falloff", AO_falloff,
        "falloffmode", AO_falloffmode,
        "bias", AO_bias,
        "clamp", AO_clamp,
        "maxsolidangle", AO_maxsolidangle,
        "maxvariation", AO_maxvariation));

    } else {
    texture3d(rebaked_file, Ps, shading_normal,
    "occlusion", AO_Cl,
    "coordsystem", rebaked_coordsyst);
    }
    outputchannel("aov_AO", AO_Cl);
    Cl = AO_Cl;
    Ol = Ol;
    if(rebake == 1) {
      bake3d(rebaked_file, "", Ps, shading_normal,

      "occlusion", AO_Cl,
      "coordsystem", rebaked_coordsyst,
      "interpolate", 1);
    }
   }

  if(DO == 1) {
    if (reuse != 1) {
    DO_Cl = DO_int*(1-occlusion( // performs occlusion with parameters from GUI, inverse AO
and multiply by our intensity
        Ps, DO_dir, 0,
        "pointbased", 1,
        "filename", filename,
        "coordsystem", coordsyst,
        "coneangle", radians(DO_coneangle),
        "hitsides", DO_hitsides,
        "maxdist", DO_maxdist,
        "falloff", DO_falloff,
        "falloffmode", DO_falloffmode,
        "bias", DO_bias,
        "clamp", DO_clamp,
        "maxsolidangle", DO_maxsolidangle,
        "maxvariation", DO_maxvariation));

    } else {
    texture3d(rebaked_file, Ps, DO_dir,
    "occlusion", DO_Cl,
    "coordsystem", rebaked_coordsyst);
    }
    outputchannel("aov_DO", DO_Cl);
    Cl = DO_Cl;
    Ol = Ol;
    if(rebake == 1) {
      bake3d(rebaked_file, "", Ps, DO_dir,
      "occlusion", DO_Cl,
      "coordsystem", rebaked_coordsyst,
      "interpolate", 1);
    }
   }

   if(RO == 1) {
      vector RO_refl = reflect(I, shading_normal); // Uses the reflected vector to compute
occlusion, thus reflection occlusion
    if (reuse != 1) {
    RO_Cl = RO_int*(1-occlusion(Ps, RO_refl, 0,

        "pointbased", 1,
        "filename", filename,
        "coordsystem", coordsyst,
        "hitsides", RO_hitsides,
          "clamp", RO_clamp,
        "maxdist", RO_maxdist,
        "falloff", RO_falloff,
        "falloffmode", RO_falloffmode,
        "coneangle", radians(RO_coneangle),
        "maxsolidangle", RO_maxsolidangle,
        "bias", RO_bias));

    } else {
    texture3d(rebaked_file, Ps, RO_refl,
    "occlusion", RO_Cl,
    "coordsystem", rebaked_coordsyst);
    }
    outputchannel("aov_RO", RO_Cl);
    Cl = RO_Cl;
    Ol = Ol;
    if(rebake == 1) {
      bake3d(rebaked_file, "", Ps, RO_refl,
      "occlusion", RO_Cl,
      "coordsystem", rebaked_coordsyst,
      "interpolate", 1);
    }
   }

   if(GRefl == 1) {
       vector GRefl_refl = reflect(I, shading_normal); // Again, using a reflection vector for
computation
    if (reuse != 1) {
    GRefl_Cl = GRefl_int*indirectdiffuse(Ps, GRefl_refl, 0,

        "pointbased", 1,
        "filename", filename,
        "coordsystem", coordsyst,
        "hitsides", GRefl_hitsides,
        "clamp", GRefl_clamp,
        "sortbleeding", GRefl_sort,
        "maxdist", GRefl_maxdist,
        "falloff", GRefl_falloff,
        "falloffmode", GRefl_falloffmode,
        "coneangle", radians(GRefl_coneangle),
        "bias", GRefl_bias,
        "maxsolidangle", GRefl_maxsolidangle);

    } else {
    texture3d(rebaked_file, Ps, GRefl_refl,
    "reflection", GRefl_Cl,
    "coordsystem", rebaked_coordsyst);
    }
    outputchannel("aov_GRefl", GRefl_Cl);
    Cl = GRefl_Cl;
       Ol = Ol;
    if(rebake == 1) {
      bake3d(rebaked_file, "", Ps, GRefl_refl,
      "reflection", GRefl_Cl,
      "coordsystem", rebaked_coordsyst,
      "interpolate", 1);
```

```
  }
}

if(GRefr == 1) {
  if (reuse != 1) {
  extern point Ps;
  extern normal N;
  extern vector I;
  // normalizing
  vector IN = normalize(I);
  // stores reflection and refraction vectors
  vector reflDir, refrDir;
  float eta = (IN.shading_normal < 0) ? 1/GRefr_ior : GRefr_ior;

  float kr, kt;
  // computes fresnel effect
  fresnel(IN, shading_normal, eta, kr, kt, reflDir, refrDir);

  kt = 1 - kr;

  vector RF = normalize(refrDir);
  GRefr_Cl = kt*GRefr_int*indirectdiffuse(Ps, RF, 0, //glossy refractions

          "filename", filename,
          "pointbased", 1,
          "hitsides", GRefr_hitsides,
          "coneangle", radians(GRefr_coneangle),
            "clamp", GRefr_clamp,
           "coordsystem", coordsyst,
           "sortbleeding", GRefr_sort,
          "maxdist", GRefr_maxdist,
          "falloff", GRefr_falloff,
           "falloffmode", GRefr_falloffmode,
          "bias", GRefr_bias,
            "maxsolidangle", GRefr_maxsolidangle);


  } else {
  texture3d(rebaked_file, Ps,  shading_normal,
  "refraction", GRefr_Cl,
  "coordsystem", rebaked_coordsyst);
  }
  outputchannel("aov_GRefr", GRefr_Cl);
  Cl = GRefr_Cl;
  Ol = Ol;
  if(rebake == 1) {
     bake3d(rebaked_file, "", Ps, shading_normal,

     "refraction", GRefr_Cl,
     "coordsystem", rebaked_coordsyst,
     "interpolate", 1);
  }
```

```
}

if(IBL == 1) {
  if (reuse != 1) {
  color IBL_envcol = 0;
  vector IBL_envdir = 0;
  IBL_Cl = IBL_int*occlusion( // performs occlusion from HDRI with parameters from GUI

         Ps, shading_normal, 0,
         "pointbased", 1,
         "filename", filename,
         "coordsystem", coordsyst,
         "hitsides", IBL_hitsides,
         "maxdist", IBL_maxdist,
         "coneangle", radians(IBL_coneangle),
         "falloff", IBL_falloff,
         "falloffmode", IBL_falloffmode,
         "bias", IBL_bias,
         "clamp", IBL_clamp,
         "maxsolidangle", IBL_maxsolidangle,
         "maxvariation", IBL_maxvariation,
         "environmentmap", envmap,
         "environmentcolor", IBL_envcol,
         "environmentdir", IBL_envdir,
         "environmentcolor", aov_envcolor);

  } else {
  texture3d(rebaked_file, Ps, shading_normal,
  "ibl", IBL_Cl,
  "coordsystem", rebaked_coordsyst);
  }
  outputchannel("aov_IBL", IBL_Cl);
  Cl = IBL_Cl;
  Ol = Ol;
  if(rebake == 1) {
     bake3d(rebaked_file, "", Ps,  shading_normal,

     "ibl", IBL_Cl,
     "coordsystem", rebaked_coordsyst,
     "interpolate", 1);
  }
}

if(GI == 1) {
  if (reuse != 1) {
  GI_Cl = GI_int*indirectdiffuse(Ps,  // performs color bleeding with parameters from GUI

         shading_normal, 0,
         "pointbased", 1,
         "filename", filename,
         "hitsides", GI_hitsides,
         "coneangle", radians(GI_coneangle),
         "clamp", GI_clamp,
         "coordsystem", coordsyst,
         "sortbleeding", GI_sort,
         "maxdist", GI_maxdist,
         "falloff", GI_falloff,
         "falloffmode", GI_falloffmode,
         "bias", GI_bias,
         "maxsolidangle", GI_maxsolidangle,
         "environmentmap", envmap,
         "environmentspace", envspace);

  } else {
  texture3d(rebaked_file, Ps, shading_normal,
  "indirectdiffuse", GI_Cl,
  "coordsystem", rebaked_coordsyst);
  }
  outputchannel("aov_GI", GI_Cl);
  Cl = GI_Cl;
  Ol = Ol;
  if(rebake == 1) {
     bake3d(rebaked_file, "", Ps,  shading_normal,

     "indirectdiffuse", GI_Cl,
     "coordsystem", rebaked_coordsyst,
     "interpolate", 1);
  }

  }

  if(SSS == 1) {
  if (reuse != 1) {
  float SSS_mult = SSS_weight;
  SSS_mult = (SSS_mult - 1)*(-1); // SSS_weight should be 0 for SSS and 1 for diffuse.

  SSS_Cl = subsurface( // performs subsurface with ptc

         Ps, shading_normal,
         "coordsystem", coordsyst,
         "filename", filename,
         "diffusemeanfreepath", SSS_dmfp,
         "albedo", SSS_al,
         "scale", SSS_scale,
         "ior", SSS_ior) * SSS_mult; // Multiply by the SSS_mult to control SSS intensity


  } else {
  texture3d(rebaked_file, Ps, shading_normal,
  "subsurface", SSS_Cl,
  "coordsystem", rebaked_coordsyst);
  }
  outputchannel("aov_SSS", SSS_Cl);
  Cl = SSS_Cl;
```

```
    Ol = Ol;
    if(rebake == 1) {
       bake3d(rebaked_file, "", Ps,  shading_normal,

       "subsurface", SSS_Cl,
       "coordsystem", rebaked_coordsyst,
       "interpolate", 1);
    }
  }

}
```

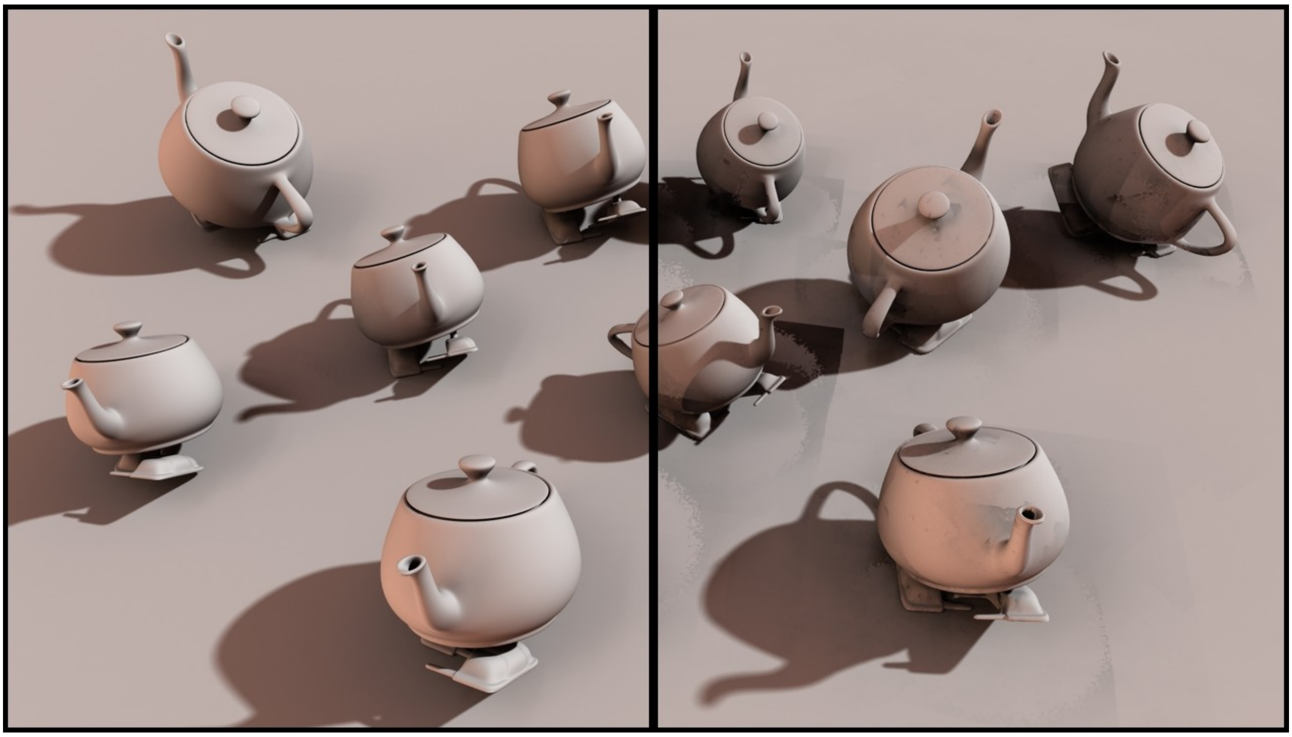
Image based lighting.

Image-based lighting (IBL) is the process of illuminating scenes and objects with images of light from the real world. *[See IBL chapter in Glosary of Terms]*.

It evolved from the reflection-mapping technique, in which we use panoramic images as texture maps on computer graphics models to show shiny objects reflecting real and synthetic environments. IBL is analogous to image-based modeling, in which we derive a 3D scene's geometric structure from images, and to image-based rendering, in which we produce the rendered appearance of a scene from its appearance in images. When used effectively, IBL can produce realistic rendered appearances of objects and can be an effective tool for integrating computer graphics objects into real scenes.

The following comparison involves different methods for the computation of Image Based Lighting, depending on the software used. These could be divided into Raytracing image based lighting, PointCloud, and Unbiased methods.

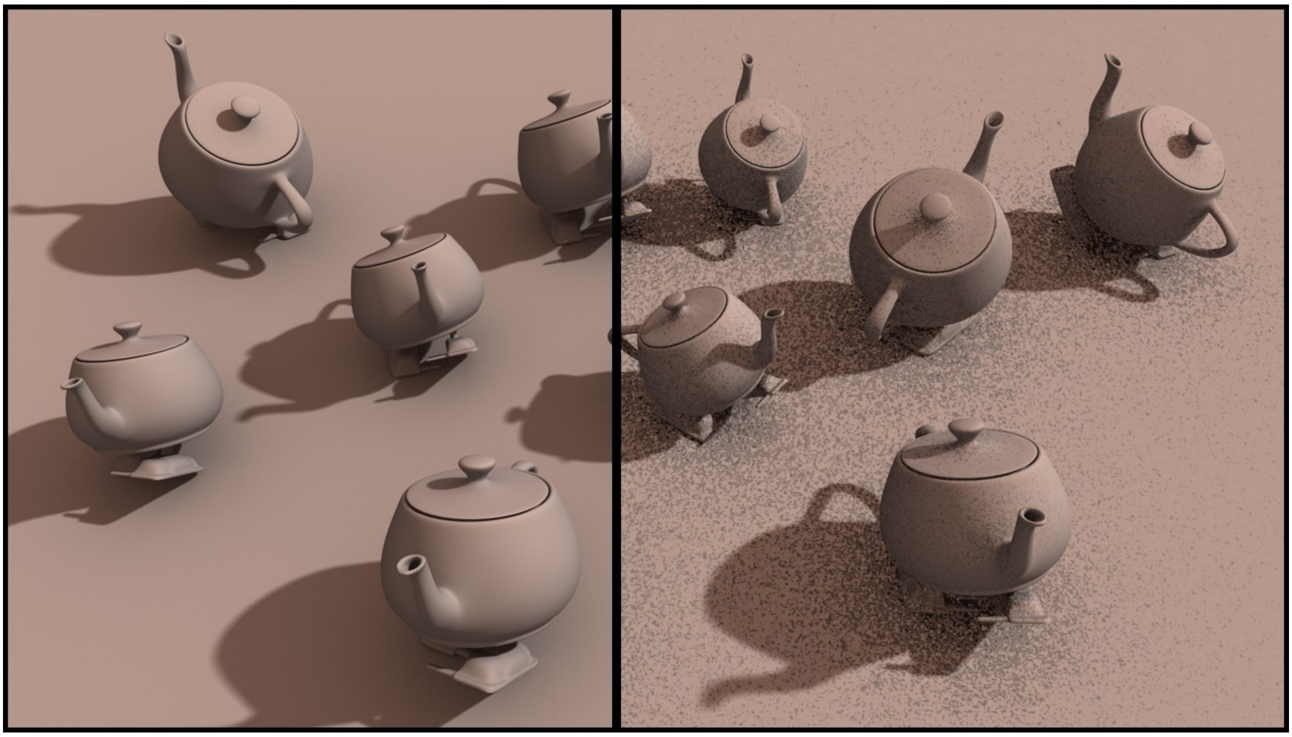
The following pictures are render images used in this comparison, example of each kind of method, showing up the differences between a low quality noisy image and a good quality, noise free image. Note how the level of quality in IBL lighting is determined by the amount of noise each image contains, apart from point clound rendering method, where the appearance of blotchy artifacts indicate a poor quality render.



Picture 1. High vs Low quality pointCloud render.
(3Delight PointCloud)

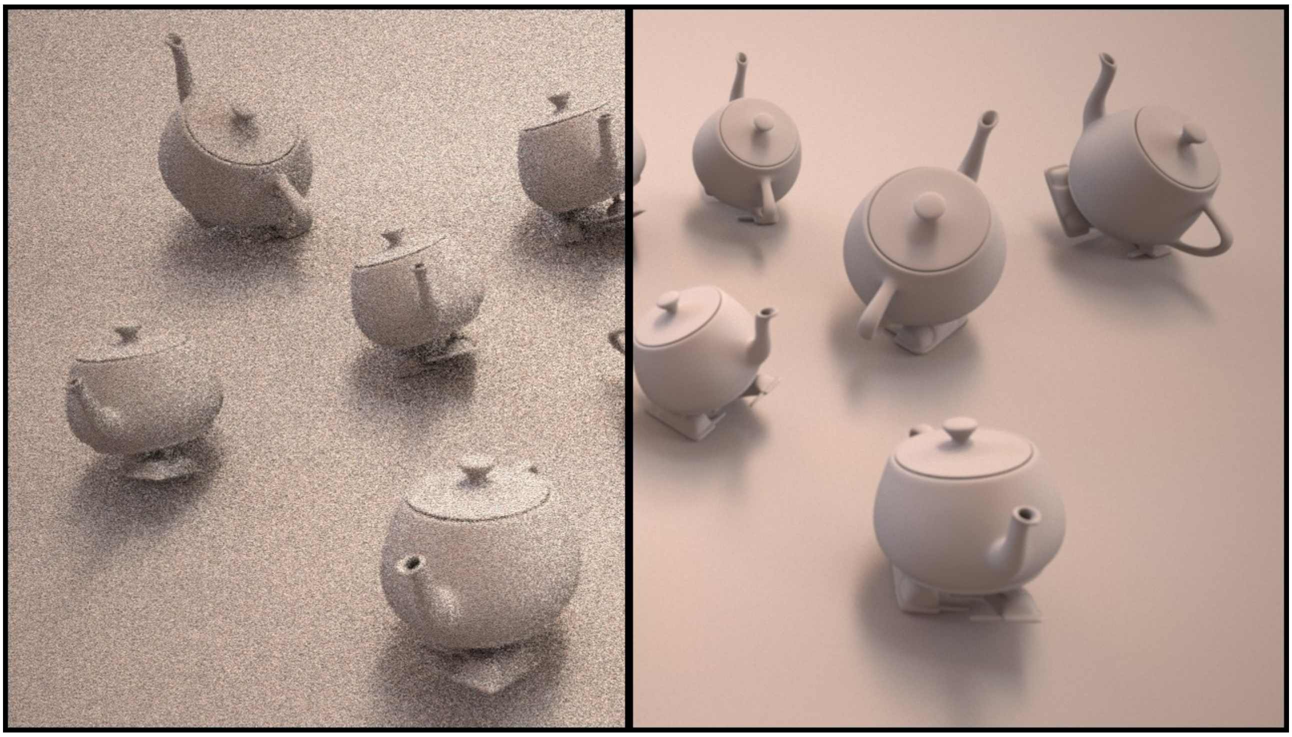
Note how the lack of quality in point cloud rendering results in the appearance of artifacts in the image mostly in areas where light occludes.

The following images show the differences between high quality IBL and a low quality version of the same scene.

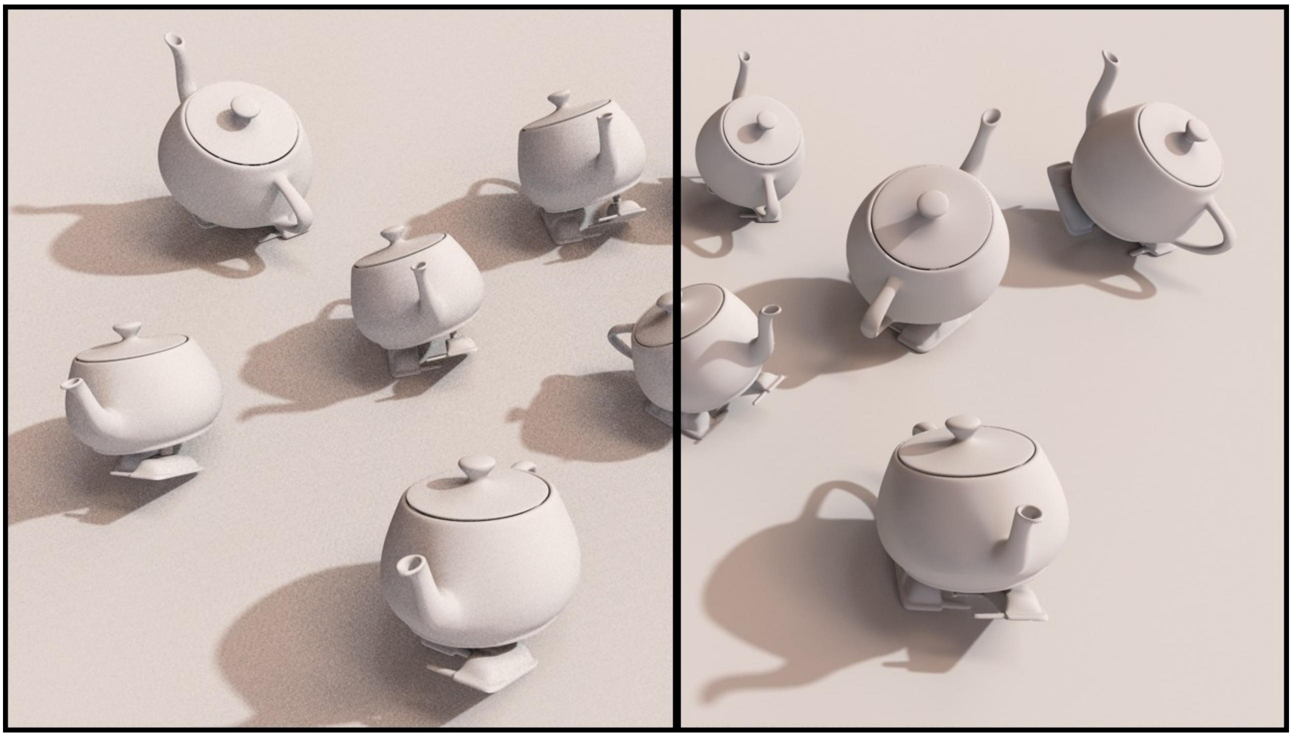


Picture 2. IBL quality differences.
(3Delight Raytracing)

In this and the following cases, the lack of quality is determined in terms of grain (image noise) without the appearance of artifacts.



Picture 3. IBL quality differences.
(MaxwellRender)



Picture 4. IBL quality differences.
(Mental Ray Radiosity (Final Gather))

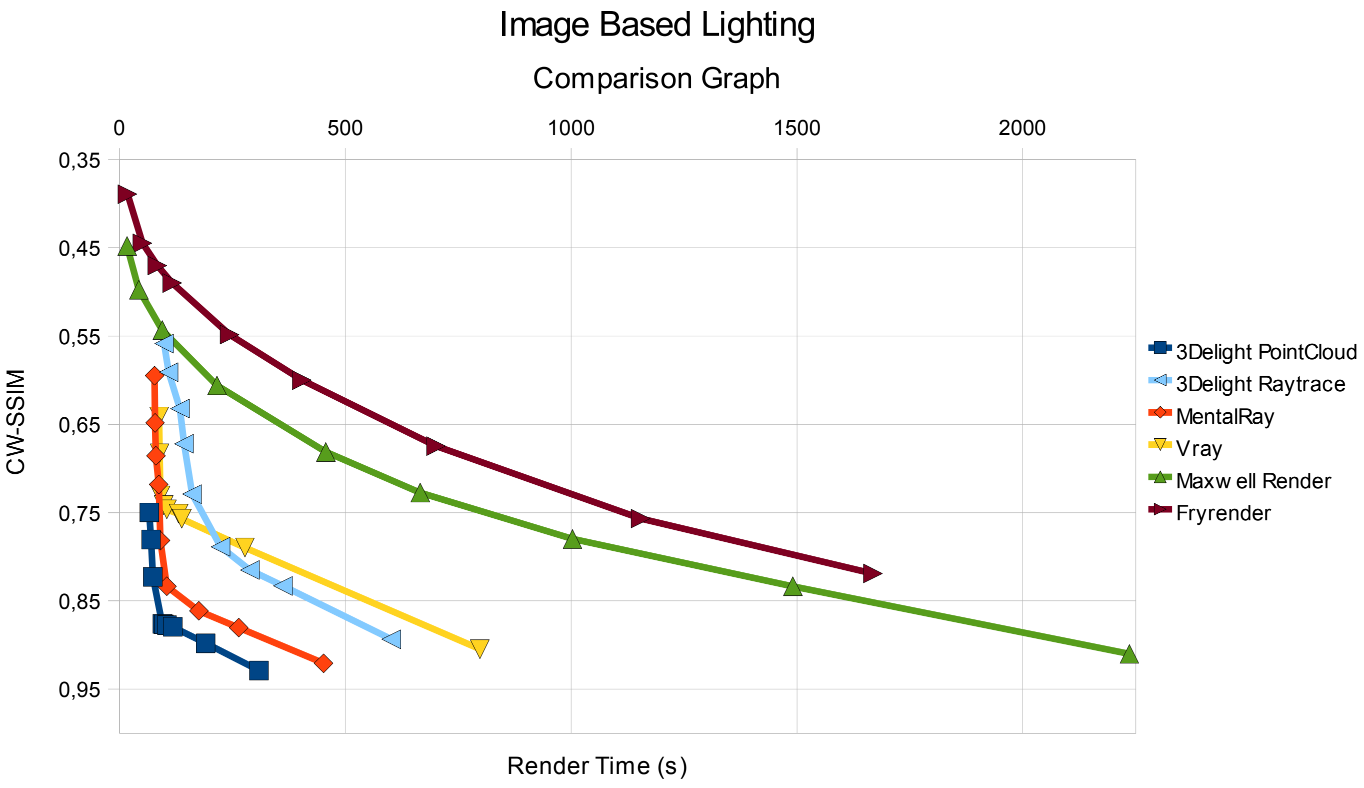
Please refer to "Renders" subfolder for a complete list of rendered images used in this part of the study for each rendere engine involved in the comparison.

	1	2	3	4	5	6	7	8	9	
3Delight Raytrace										
Render Time (m,s)	99	109	134	143	161	224	289	362	602	
CW-SSIM	0,56	0,59	0,63	0,67	0,73	0,79	0,82	0,83	0,89	
Occlusion Samples	2	4	8	16	32	64	96	128	256	
3Delight PointCloud										
Render Time	66	70	74	95	98	105	118	191	309	
CW-SSIM	0,75	0,78	0,82	0,88	0,88	0,88	0,88	0,9	0,93	
MaxSolidAngleValue	30	1	0,5	0,1	0,09	0,07	0,05	0,02	0,01	Samples=8, BaseSamples=128
MentalRay										
Render Time	78	79	81	87	91	105	176	264	452	
CW-SSIM	0,59	0,65	0,69	0,72	0,78	0,83	0,86	0,88	0,92	
env. lighting quality	0,05	0,1	0,2	0,4	0,5	0,8	1	1,2	1,5	Fg density=0,20 Fg Accuracy=30
Vray										
Render Time	88	89	91	98	105	131	138	278	798	
CW-SSIM	0,64	0,68	0,73	0,74	0,75	0,75	0,76	0,79	0,9	
Min,Max Rate	-8...-7	-6...-5	-4...-3	-3...-2	-3...-1	-3...0	-2...0	0...0	-1...1	Primary Rays = Irradiance Cache = 50subdivs, Interp. Samples = 20 Secondary Rays = Light Cache (8 passes, 500 subdivs, 0.02 sample size)
Maxwell Render										
Render Time	17	43	95	216	457	666	1003	1491	2236	
CW-SSIM	0,45	0,5	0,54	0,61	0,68	0,73	0,78	0,83	0,91	
Sampling Level	4	6	8	10	12	13	14	15	16	
Fryrender										
Render Time	18	51	84	117	244	404	701	1154	1668	
CW-SSIM	0,39	0,44	0,47	0,49	0,55	0,6	0,67	0,76	0,82	
Passes	1	3	5	7	15	25	45	75	105	

The previous table shows CW-SSIM vs RenderTime values in nine different renders for each render engine involved in this study.

Please note that values given in the table have been rounded for a better and faster appreciation . For a more precise representation of the values shown, refer to the excel sheet located in the corresponding section's folder named "tables.ods"

Once the table is built, a graph can be drawn for a more intuitive representations of the results obtained. Please refer to "Error Metrics" chapter in this document for a more detailed explanation.



IBL comparison graph.

The above graph shows clearly the differences between the render engines involved in the comparsion.

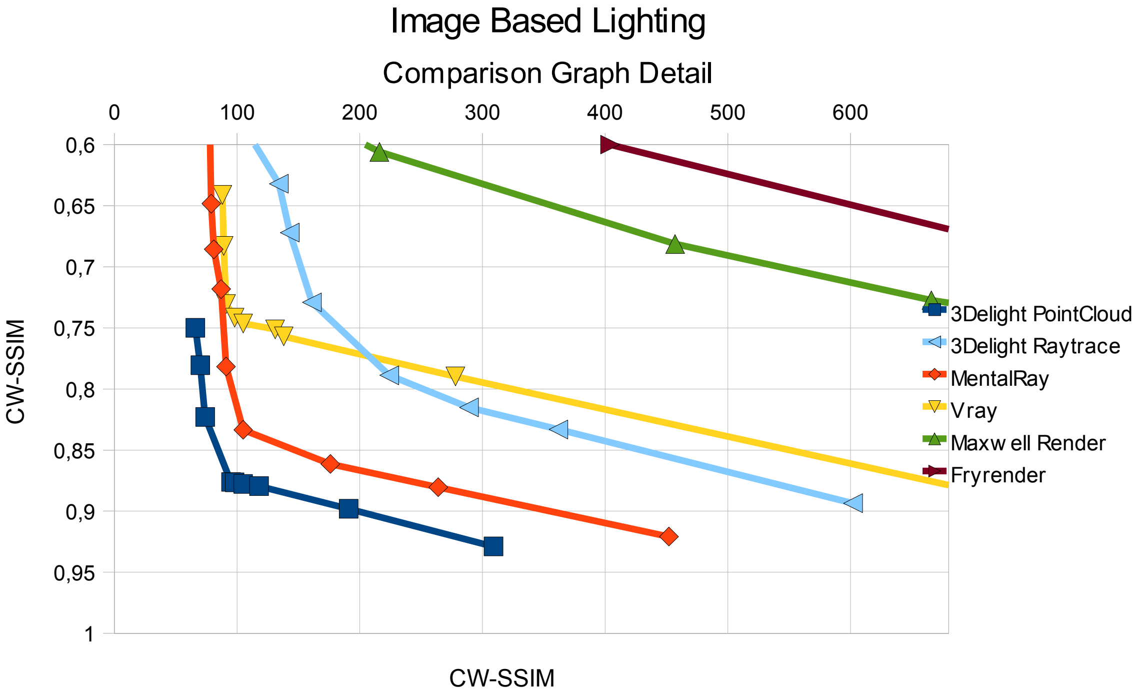
From unbiased renderers, it could be easily noticed how similar both curves are, being also close from each other. They stay in a worse place than biased render engines, however, MaxwellRender performs something better than FryRender through the time range.
About 85% CW-SSIM value is reached at around 1668 seconds opposited to 82% from FryRender at the same time.

That said, it is worth noting that, although unbiased renderers offer a worse performance than biased render engines in IBL computation, their unbiased nature provide renders obtained with this kind of software a more physically accurated look in shadows and occluded objects obtained right away from the HDRI source.

Picture 3. is a good example of that, where shadows are blurred and oriented according to the HDRI image, while Picture 1., for example, shows sharp shadows oriented in a different way, result of the creation of an auxiliary light source to fake HDRI shadowing typical from biased render engines in this kind of scenes.

St Peters Cathedral Probe used as HDRI light source for this comparison.
(St. Peter's Basilica, Rome, 1500 × 1500, Dynamic range: 200,000:1) [1]

Biased render engines, however, offer a better performance than unbiased renderers. The above graph shows how close they are from each other, so it would be a good idea to build a more detailed graph focusing on those biased render engines to have a clerarer representation of the results obtained in the comparison.



Detailed IBL comparison graph.

From the above graph the following conclussions could be extracted.

3Delight PointCloud approach to IBL performs the best, reaching a CW-SSIM value of 93% in 309 seconds, while Mentalray, the next render engine in terms of IBL calculation performance, reaches 92% in 452 seconds.

Vray stays close to MentalRay in the first 100 seconds of rendering, but beyond that point, it gets slower than MentalRay as can be noticed from the above graph.
3Delight raytrace, however, stays in a worse place in the first couple of hundred seconds of rendering , but beyond around 220 seconds it gets slightly better than Vray.

References.

[1] Paul Debevec. Rendering Synthetic Objects Into Real Scenes: Bridging Traditional and Image-Based Graphics With Global Illumination and High Dynamic Range Photography, Proceedings of SIGGRAPH 98, pp. 189-198 (July 1998, Orlando, Florida).

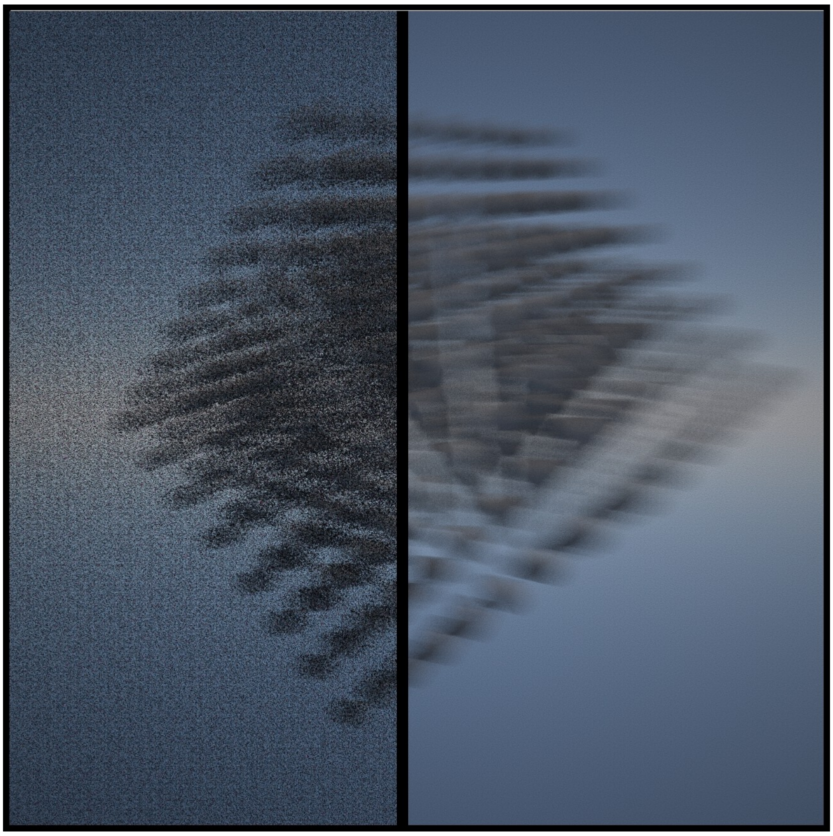
Motion Blur Comparison.

Motion blur is the apparent streaking of rapidly moving objects in a still image or a sequence of images such as a movie or animation. *[See Motion Blur chapter in Glossary of Terms]* It results when the image being recorded changes during the recording of a single frame, either due to rapid movement or long exposure time.

When the 3D camera creates an image, that image does not always represent a single instant of time. As objects in a scene move, an image of that scene must represent an integration of all positions of those objects, as well as the camera's viewpoint, over the period of exposure determined by the shutter speed.
This comparison involves different methods for the computation of Motion Blur depending on the software used. These could be divided into biased and Unbiased methods.

The following pictures are render images used in this comparison, example of each kind of method, showing up the differences between a low quality noisy image and a good quality, noise free image. Note how the level of quality in Motion Blur is determined by the amount of noise or grain each image contains.

It is worth noting that FryRender render engine does not compute real object motion blur, so this technique could not be included for that render engine in this chapter. A separate motion vector pass could be obtained in Fryrender though, giving the possibility to post-produce that separate pass with the final render, to obtain object motion blur.
That is by no means physically accurate, but could be used as a workaround for producing object motion blur when fryrender is to be used.



Low sampling vs High sampling in MotionBlur calculation.
(Maxwell Render built-in MotionBlur)

Please refer to "Renders" subfolder for a complete list of rendered images used in this part of the study for each rendere engine involved in the comparison.

The following table shows CW-SSIM vs RenderTime values in nine different renders for each render engine involved in this study.

Please note that values given in the table have been rounded for a better and faster appreciation .

	1	2	3	4	5	6	7	8	9

3Delight

Render Time (m,s)	22	21	22	22	21	22	21	22	22
CW-SSIM	0,69	0,81	0,85	0,87	0,76	0,84	0,89	0,91	0,96

Pixel Samples (filter)	1-1 (sync)	2-2 (sync)	3-3 (sync)	4-4 (sync)	1-1 (gauss)	2-2 (gauss)	3-3 (gauss)	4-4 (gauss)	8-8 (gauss)

MentalRay

					*		*		*
Render Time	71	91	92	94	103	110	115	134	144
CW-SSIM	0,85	0,86	0,87	0,92	0,93	0,93	0,94	0,95	0,96

Time Samples	1	2	3	20	25	30	35	40	45

*Raytracer disabled, Scanline render enabled.

Vray

Render Time	41	106	201	314	442	508	549	582	598
CW-SSIM	0,77	0,86	0,9	0,93	0,93	0,96	0,96	0,96	0,96

Subdivs	1	2	3	4	5	6	7	8	9

Maxwell Render

Render Time	63	147	244	434	753	1174	1819	1872	4426
CW-SSIM	0,29	0,37	0,41	0,45	0,47	0,54	0,59	0,67	0,74

Sampling Level	1	2	3	4	5	6	7	8	9

Shutter speed (1/s) = 40

Fryrender

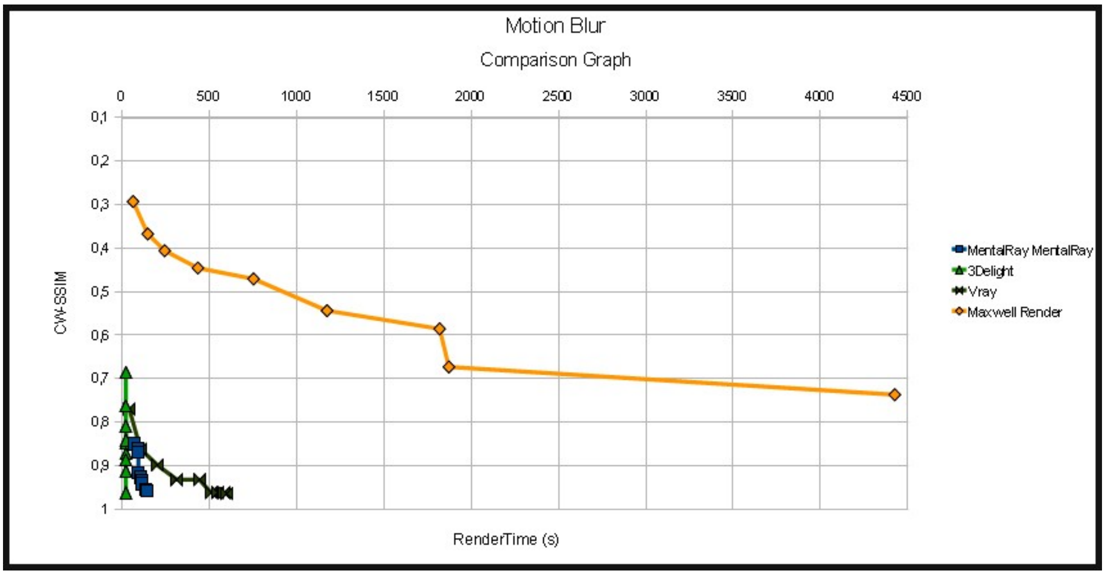
Render Time	***	***	***	***	***	***	***	***	***
Error (%)	***	***	***	***	***	***	***	***	***
CW-SSIM	***	***	***	***	***	***	***	***	***

Passes	***	***	***	***	***	***	***	***	***

*** = Object MotionBlur Not Supported

For a more precise representation of the values shown, refer to the excel sheet located in the corresponding section's folder named "tables.ods"

Once the table is built, a graph can be drawn for a more intuitive representations of the results obtained. Please refer to "Error Metrics" chapter in this document for a more detailed explanation.



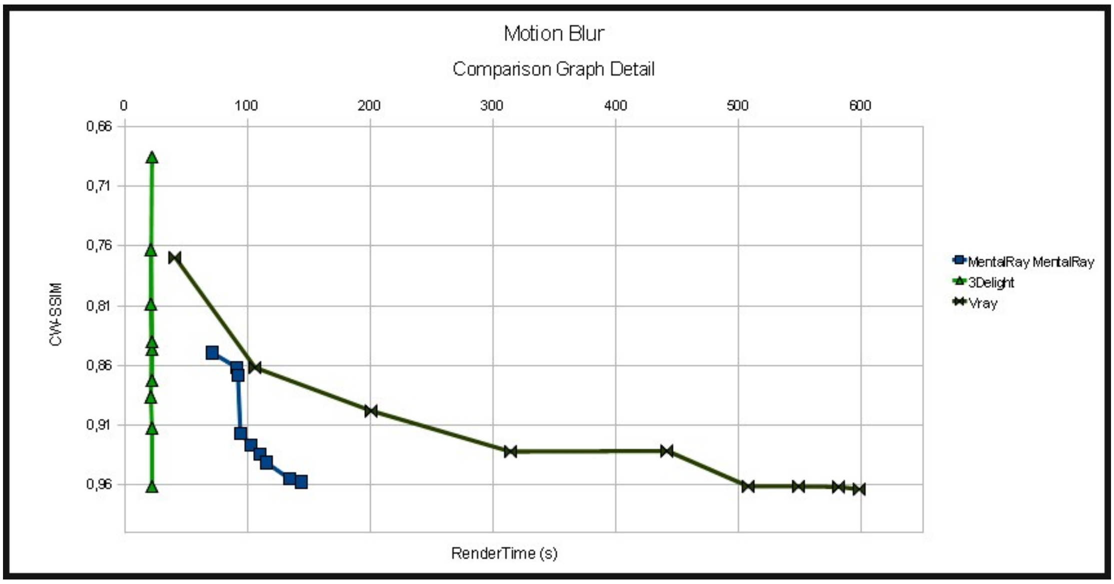
Motion Blur Comparison Graph.

The above table shows a big difference in terms of rendering performance between the Unbiased Renderer MaxwellRender, and the rest of Biased renderers involved in this study.

MaxwellRender stays in a clear defined worse place throughout the time range, taking 4428 seconds to get to a CW-SSIM level of 74%. That makes a huge difference compared to biased render engines, which in the worst case, reach a 77% CW-SSIM value in a fraction of the time (41 seconds opposited to 4428 needed to get to a value 3% better than MaxwellRender).

That said, it could be stated that MaxwellRender should be avoided when intense motionBlur is to be rendered and strict time restrictions are to be applied.

Due to the big difference between biased and unbiased render engines, it is difficult to compare biased renderers accurately, so they appear drawn too close together. For that reason, the next graph is built focusing on just the biased render engines to make their differences clearer to notice.



Motion Blur detailed Comparison Graph.

Note how 3Delight performance in MotionBlur calculation is virtually perfect. The scene used in this part of the comparison takes 22 seconds to render in 3Delight whatever Pixel Samples are used. That makes MotionBlur calculation costless compared to the other render engines involved in this comparison, and puts 3Delight in the better place by far.
That extremely good behaviour of 3Delight occurs due to the Reyes core of 3Delight, which leads to costless MotionBlur calculation.

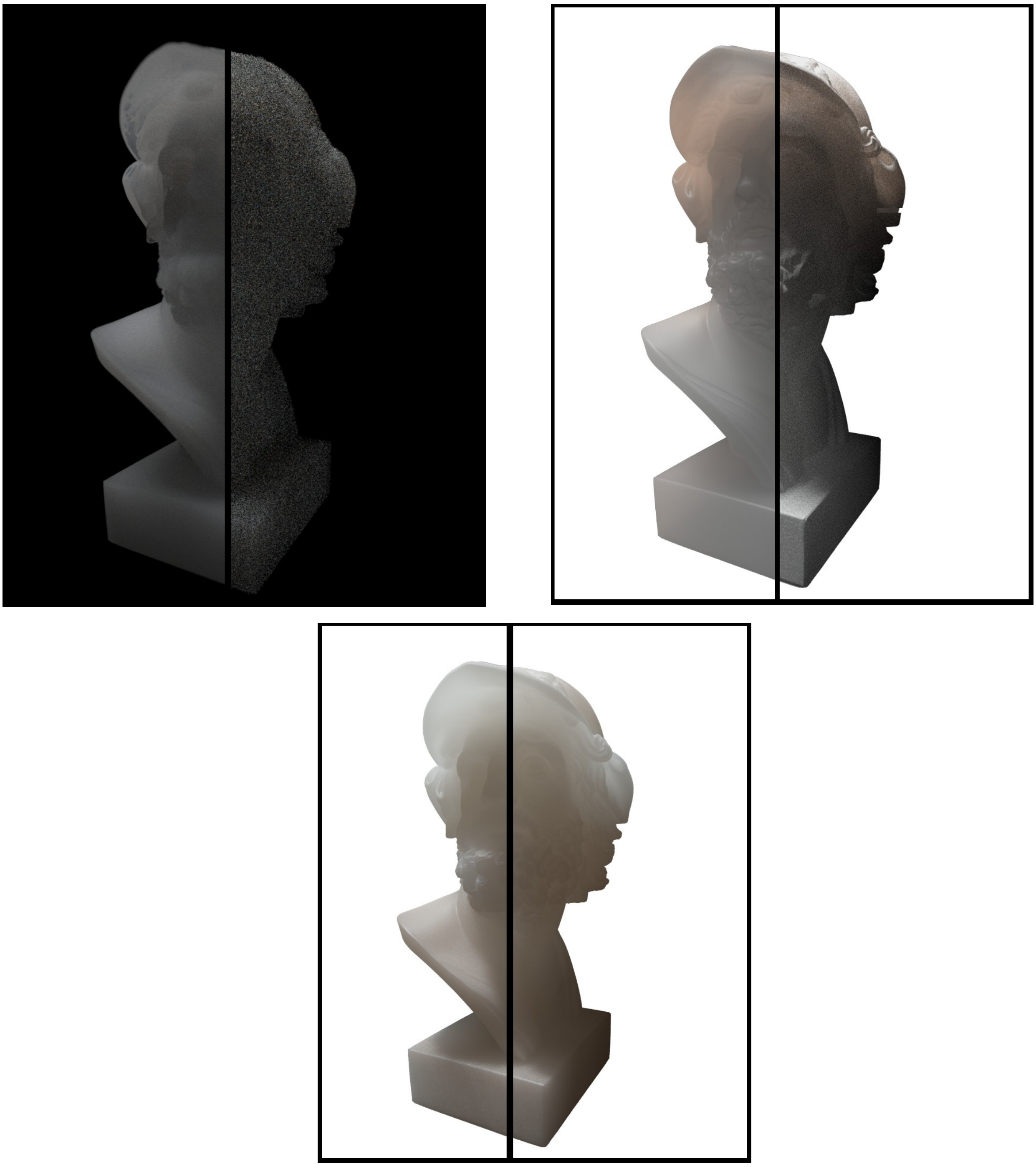
On the other hand, MentalRay stays in a better place than Vray Renderer, though being far from getting to 3Delight results. It reaches a CW-SSIM value of 96% in a quarter of the time Vray does. (144 seconds opposited to 568 needed by Vray).

All that said, 3Delight render engine is the most suitable software package when intense MotionBlur is to be computed, by far. Nevertheless, the other biased render engines perform quite good compared to MaxellRender, which should be avoided with scenes where intense MotionBlur is needed and time restrictions need to be applied.

Subsurface Scattering.

Subsurface scattering, also known as SSS, is a phenomenon of light transport through translucent objects. *[See Subsurface Scattering chapter in Glossary of Terms]*.
Light penetrates the translucent surface at some point, and it is scattered by interacting with the particles of the material until exits the surface at a different point.

Interaction between light and the object's particles lead to several reflections of light inside the material at irregular angles before passing back out of the material. As a result, some light is absorbed and some is scattered back to the surface.



Subsurface quality differences.
(MaxwellRender MentalRay & 3Delight respectively).

The previous pictures are render images used in this comparison, example of each kind of method, showing up the differences between a low quality noisy image and a good quality, noise free image.

Note how the level of quality in Subsurface scattering is determined by the amount of noise or grain each image contains.

Please refer to "Renders" subfolder for a complete list of rendered images used in this part of the study for each rendere engine involved in the comparison.

The following table shows CW-SSIM vs RenderTime values in nine different renders for each render engine involved in this study.

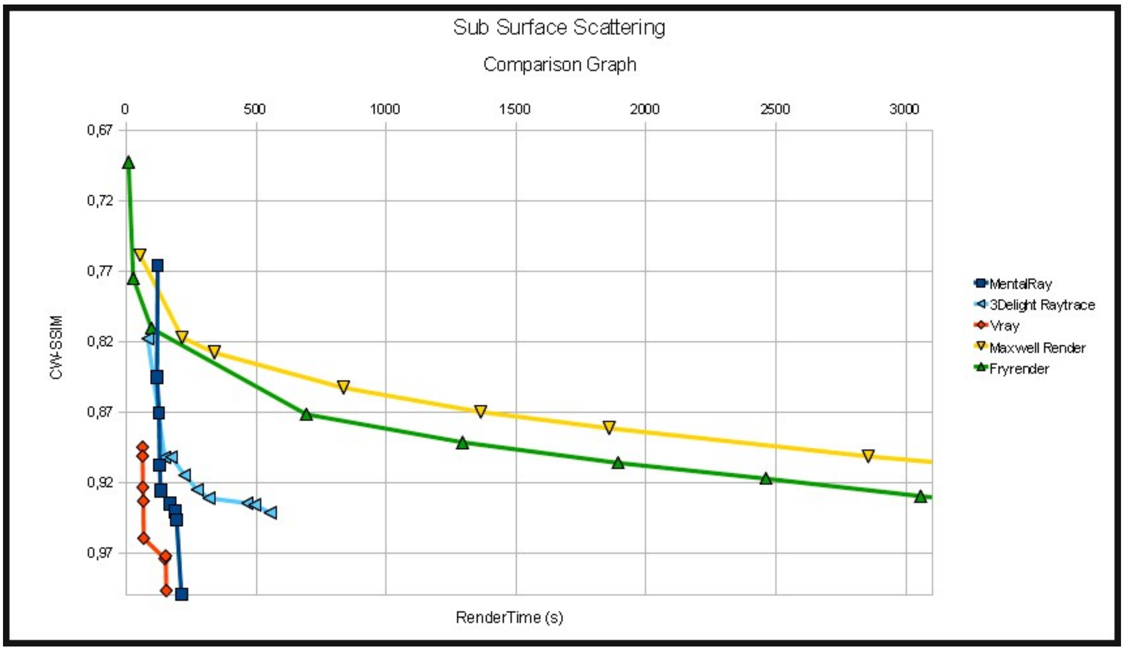
Note that values given in the table have been rounded for a better and faster appreciation .

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
3Delight Raytrace									
Render Time (m,s)	83	148	174	225	273	317	467	494	553
CW-SSIM	0,82	0,9	0,9	0,91	0,93	0,93	0,93	0,94	0,94
Samples	0	1	2	3	4	5	6	7	8
MentalRay									
Render Time	121	118	126	129	132	169	188	194	212
CW-SSIM	0,77	0,85	0,87	0,91	0,93	0,94	0,94	0,95	1
Samples	8	32	128	512	1024	1537	2048	3072	4096
Vray									
Render Time	63	63	64	66	67	148	148	152	154
CW-SSIM	0,9	0,9	0,92	0,93	0,96	0,97	0,97	0,97	1
Subdivs	2	2	8	8	8	512	512	512	512
Maxwell Render									
Render Time	53	215	339	836	1364	1858	2855	3539	3601
CW-SSIM	0,76	0,82	0,83	0,85	0,87	0,88	0,9	0,91	0,92
Sampling Level	2	5	6	8	9	10	11	11,5	12
Fryrender									
Render Time	9	27	97	693	1294	1892	2461	3056	3612
CW-SSIM	0,69	0,78	0,81	0,87	0,89	0,91	0,92	0,93	0,94
Passes	1	3	7	14	21	28	35	42	49

Pixel Samples = 3,3; Shading Rate = 1: Pixel Filter = sinc;

For a more precise representation of the values shown, refer to the excel sheet located in the corresponding section's folder named "tables.ods"

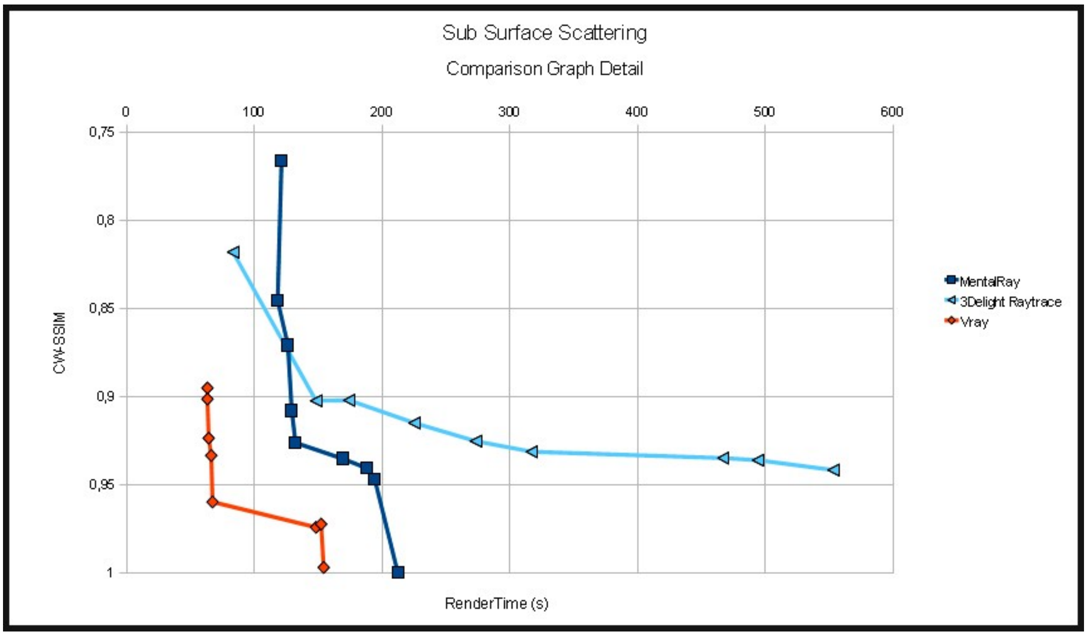
Once the table is built, a graph can be drawn for a more intuitive representations of the results obtained. Please refer to "Error Metrics" chapter in this document for a more detailed explanation.



Subsurface Scattering Comparison Graph.

The above graph shows clearly differences between the render engines involved in the comparsion in SSS calculation.
From unbiased renderers, it could be easily noticed how similar both curves are, being also close from each other. They stay in a worse place than biased render engines, however, FryRender performs something better than MaxwellRender through the time range.
MaxwellRender reaches about 92% CW-SSIM value is reached at 3601 seconds opposited to 94% from FryRender at a few seconds later (3612 seconds of render).

Biased render engines, however, offer a better performance than unbiased renderers. The above graph shows how close they are from each other, so it would be a good idea to build a more detailed graph focusing on those biased render engines to have a clearer representation of the results obtained in the comparison.



Subsurface Scattering Detailed Comparison Graph

From the above graph the following conclussions could be extracted.

3Delight approach converges the slowest of the three biased renderers. In fact, it reaches a CW-SSIM value of 93% in 273 seconds and does not get much better than that in the following minutes. It improves by 2% in the next 290 seconds (Reaches 95% in 563 seconds) and will take longer to get better than this.

On the other hand, Vray performs the best, reaching a CW-SSIM value of almost 100% in 154 seconds, while Mentalray, the next render engine in terms of SSS calculation performance, reaches 100% in 252 seconds.

That said, it seems that 3Delight Reyes implementation of SSS won't be suitable in situations where noise must be avoided completely. It could be used in production animation though, where a little amount of noise is not noticeable.

Vray and MentalRay, however, reach high levels of quality at good renderTimes compared to the other render engines, so better results should be expected from both renderers, being Vray the best choice beforehand.

Conclusions.

Once each technique has been compared, a brief summary of the results obtained will be shown here as a guide to follow when choosing a render engine for a specific application.

There are several rendering techniques that 3Delight Reyes renderer performed far better than any of the other render engines. These are geometry Displacement, Motion blur and Depth of field. The calculation of the three techniques mentioned above are achieved almost costless with 3Delight so applications that require an intense use of these techniques will be better rendered this renderer. *[See Displacement, MotionBlur and Depth of Field comparison chapters]*

In addition to this, 3Delight's PointCloud rendering method offered the best performance in Ambient Occlussion and Image-Based Lighting calculation.
Therefore is the most suitable render engine for the calculation of this effects though staying close to the other biased renderers.

However, 3Delight did not performed so well in applications such as Global Illumination calculation, Caustics, SubSurfaceScattering and generally speaking those techniques where photon calculation is required, even with the use of PointCloud rendering (in the case of GI).

In those techniques, one of the other biased render engines (MentalRay and Vray) should be used. They perform quite similar in all techniques involved in this study but to point out some differences the latter was more suitable for SSS and Depth of Filed generation and Mentalray performed better in Caustics generation and Motion Blur.

On the other hand, unbiased render engines perform somewhat worse (quality vs rendertime) than biased renderersin all techniques exposed in this study.
In applications without time restrictions , however, unbiased render engines are the best choice in terms of lighting quality and physical accuracy.

The biggest difference between biased and unbiased render engines is obtained at MotionBlur calculation *[See MotionBlur comparison chapter]* so the use of this kind of render engines with scenes that require an intense use of MotionBlur would not be recommended.

Unbiased rendering *[See "biasing in rendering" chapter]* algorithms will eventualy converge to a physicaly accurated solution. *[1]* However, their lack of bias in rendering calculations will undoubtedly introduce extra noise that then takes longer to get rid of.
Therefore, unbiased renderers are interesting when the application requires a high level of lighting accuracy and the extra rendertime needed to obtain a noise-free image is not a problem.

Between both unbiased renderers used in this study, however, MaxwellRender performed something better in most applications. Please refer to each technique for an in-depth study of the results obtained with both of these render engines.

References.

[1] Mike Farnsworth , RenderSpud renderer, 18th October 2006

Further Research.

Given the constant evolution nature of 3D render engines, and their application in production film, advertisements, etc. there are several matters related to this study in which future researches may deepen.

- GPU rendering.

Un-biased, physically based GPU renderers use the video cards' GPU in the workstation to solve the rendering equation. They provide photorealistic results really fast compared to CPU-only based render engines, as GPU computation architecture is specialy designed to rapidly manipulate and alter memory in such a way so as to accelerate the calculations involved in obtaining 3D renders .

In addition, these new methods in rendering provide a completely interactive workflow which allows to edit the lighting, materials, camera settings, depth of field realtime with instant feedback on the results.

- Memory usage in renders.

Being a common bottleneck in today's rendering workstations, it would be interesting to include a comparison of RAM usage between nowadays rendering engines, exploring methods of data handling such as BSP trees.

Binary space partitioning (BSP), for example, is a method for recursively subdividing a space into convex sets by hyperplanes.
This subdivision gives rise to a representation of the scene by means of a tree data structure known as a BSP tree. [1]
Originally, this approach was proposed in 3D computer graphics to increase the rendering efficiency by precomputing the BSP tree prior to low-level rendering operations. Some other applications include performing geometrical operations with shapes (constructive solid geometry) in CAD

Most render engines in the market offer BSP tree (kd-tree) acceleration method for fast ray tracing, with memory caching for very complex scenes.

They also provide on-demand loading of object files and on-demand procedural object creation using geometry shaders and placeholders: geometry is not created until it is needed, the geometry cache maintains only objects currently or recently in use to reduce rendering times and concurrent memory requirements.

- Stereo rendering and its optimization.

Given the arise of stereoscopic rendering in the film industry, it might be a good idea to do a research in stereoscopy rendering and which methods do 3D render engines provide for its optimization.

Stereoscopy (also called stereoscopic or 3-D imaging) refers to a technique for creating or enhancing the illusion of depth in an image by presenting two offset images separately to the left and right eye of the viewer. Both of these 2-D offset images are then combined in the brain to give the perception of 3-D depth. [2]

References.

[1] Chin, N., and Feiner, S., Near Real-Time Shadow Generation Using BSP Trees, *Computer Graphics (SIGGRAPH '89 Proceedings)*, 23(3), 99-106, July 1989.

[2] Dodgson, N.A. (August 2005). "Autostereoscopic 3D Displays". IEEE Computer 38 (8): 31–36. doi:10.1109/MC.2005.252. ISSN 0018-9162.

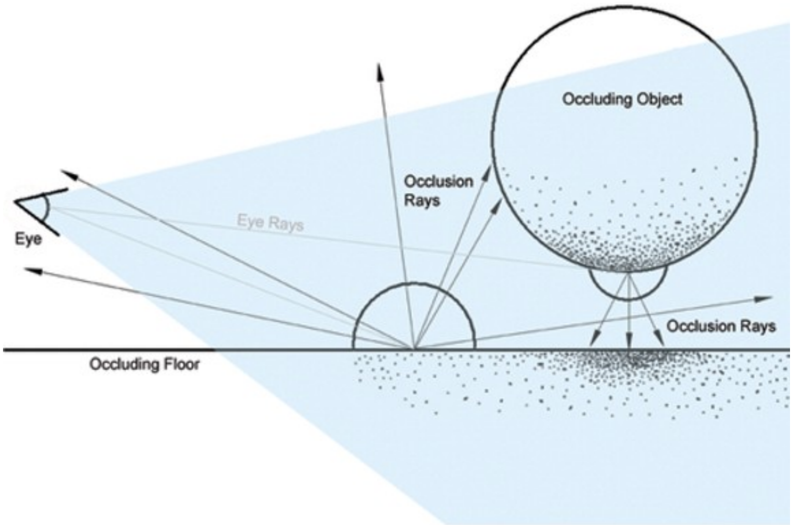
Glossary of terms.

Ambient Occlusion.

In 2002, Landis[1] and Bredow[2] presented in the SIGGRAPH course "RenderMan in Production" a technique that they had been using in the movies Pearl Harbor and Stuart Little 2, respectively, and they called it ambient occlusion. This technique was ported to production rendering for movies that mixed live footage with computer generated imagery (CGI), that is, programmed and rendered in a ray tracing environment as, for example, Renderman.
Since then the concept has popularized under the name of ambient occlusion
and it is included as a shader in most commercial renderers. [3]

Ambient occlusion refers to the blocking of indirect or diffuse light on an object. It refers to the darker areas of the object, typically creases, cracks and crevices.
Ambient occlusion is caused by indirect light's inability to bounce around and illuminate areas that are blocked by a nearby object that absorbs the light rays.
These subtle variations in lighting are visual clues for our eyes to detect surface details and distinctions that would otherwise be washed out and unnoticeable.
Ambient occlusion adds realism to your scene by adding shadows in crevices, nooks and crannies, and so on. [4]

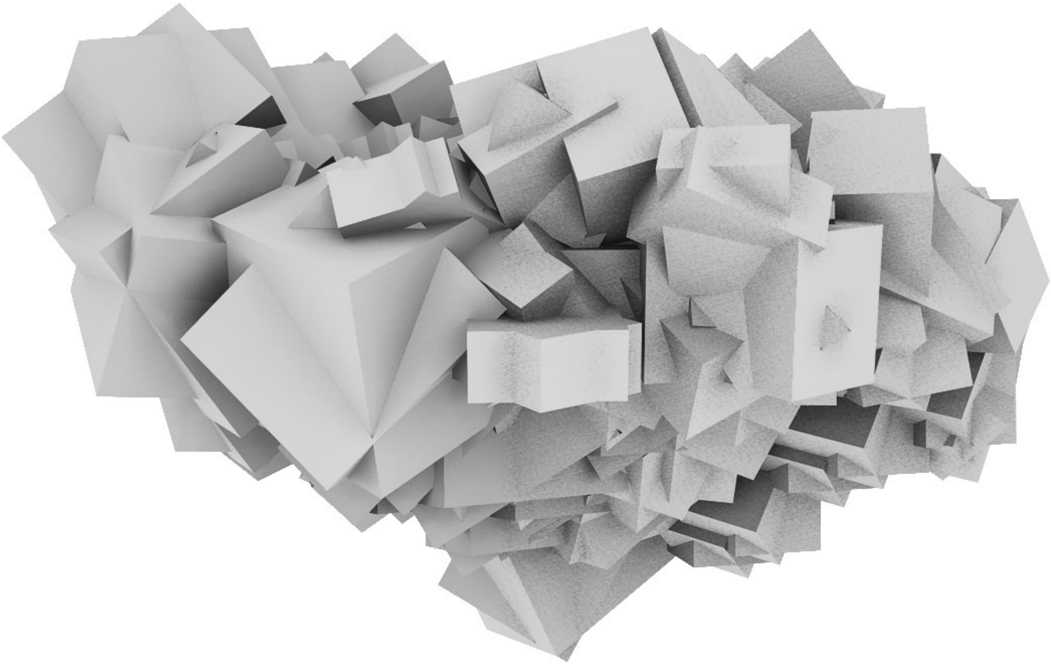
It is most often calculated by casting rays in every direction from a point on a surface.
As a result, points surrounded by a large amount of geometry are rendered dark, whereas points with little geometry on the visible hemisphere appear light.
[5]



An occlusion shader has been mapped to the floor and the sphere. A hemisphere of rays are emitted above each point that test the amount of shadowing each point should receive.

Ambient occlusion schematic diagram [6]

Depending on the amount of rays casted the resulting image would produce either poor or smooth shadows. That is commonly refered as to sampling quality.

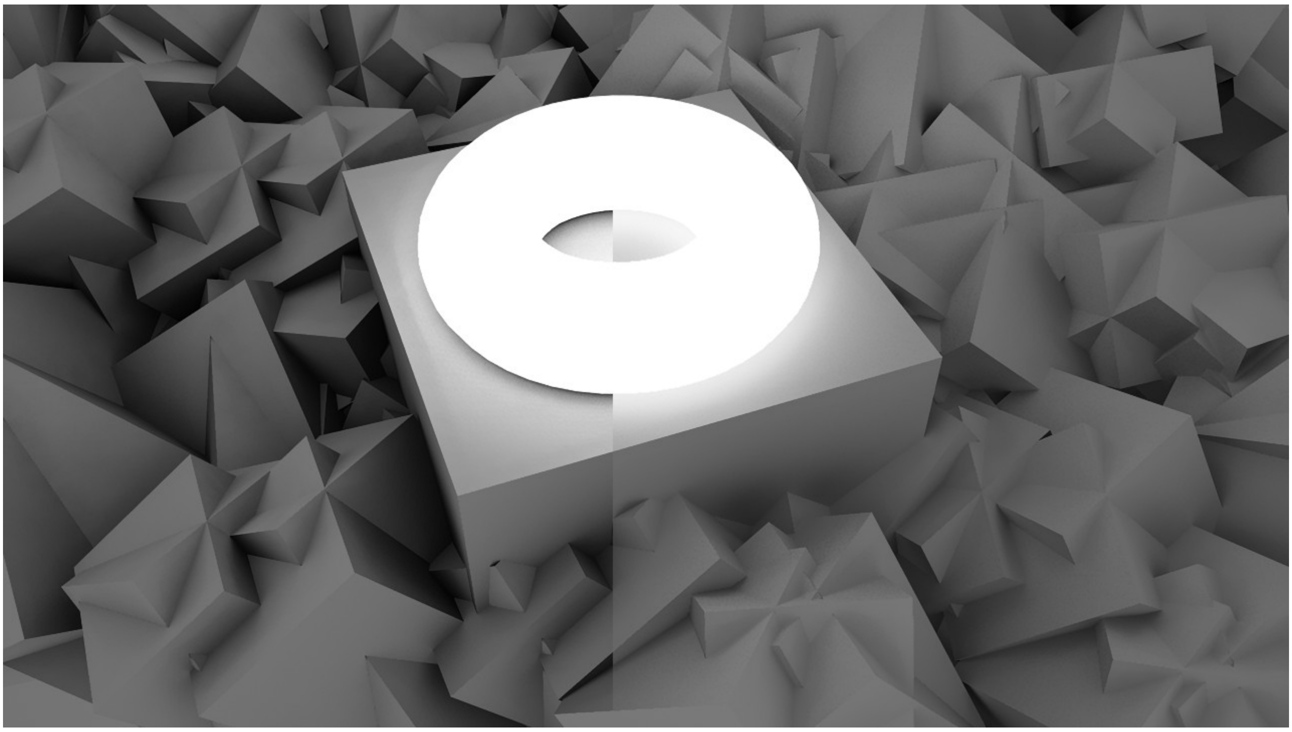


High sampling quality vs low sampling quality.

Some of the current photorealistic render engines include a more sophisticated ambient occlusion shader to improve realism.
Instead of doing simple occlusion, which can only add darkness of varying degree, those improved methods actually look at the color of the surrounding objects, and use that color rather than darkness. This is commonly know as color bleed.

Therefore, using this method would help fixing wrong contact shadows present between bright areas that lit each other.
Since this involves shading each of the points hit, this is not as computationaly fast as traditional ambient occlusion, but it has the additional efect of resolving both bright and dark details producing more accurate results.

MentalImages MentalRay v3.6 and above is a good example of a render engine that offers this newer type of ambient occlusion.



Traditional ambient occlusion vs advanced ambient occlusion.

Note the difference betwen the leften and the righten side of this image, showing the wrong contact shadows under the toroid and the very dark shaded areas compared to the correctly lit surface and much softer looking contact shadows produced by the color bleed technic respectively.

References

[1] Hayden Landis. Production-ready global illumination, july 2002. Course 16: RenderMan in Production. ACM SIGGRAPH 2002 Course Notes.

[2] Rob Bredow. Renderman on film, july 2002. Course 16: RenderMan in Production. ACM SIGGRAPH 2002 Course Notes.

[3] Universitat Politécnica de Catalunya
Departament de Llenguatges i Sistemes Inform#atics
Fast Photorealistic Techniques to
Simulate Global Illumination in
Videogames and Virtual Environments
Alex Mendez Feliu
Ph. D. dissertation
Advisor: Mateu Sbert Casasayas

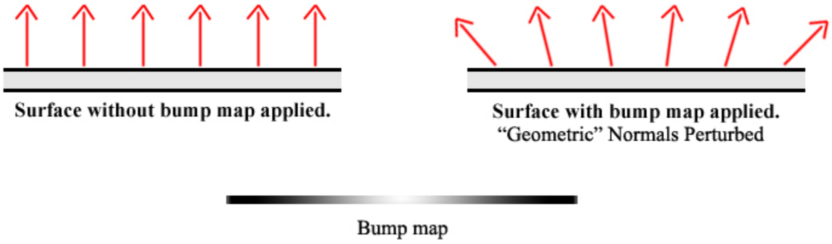
[4] Autodesk Maya 2010 Documentation, Ambient Occlusion section.

[5] Yafaray Renderer 0.1.1 Documentation, Renderer Characteristics.

[6] MentalRay v3.7 Documentation, Ambient occlusion concepts

Bump Mapping.

In 1978, James Blinn [1] presented a method of performing what is called bump mapping. Bump mapping simulates the bumps or wrinkles in a surface without the need for geometric modifications to the model. The surface normal of a given surface is perturbed according to a height map, the bump map.

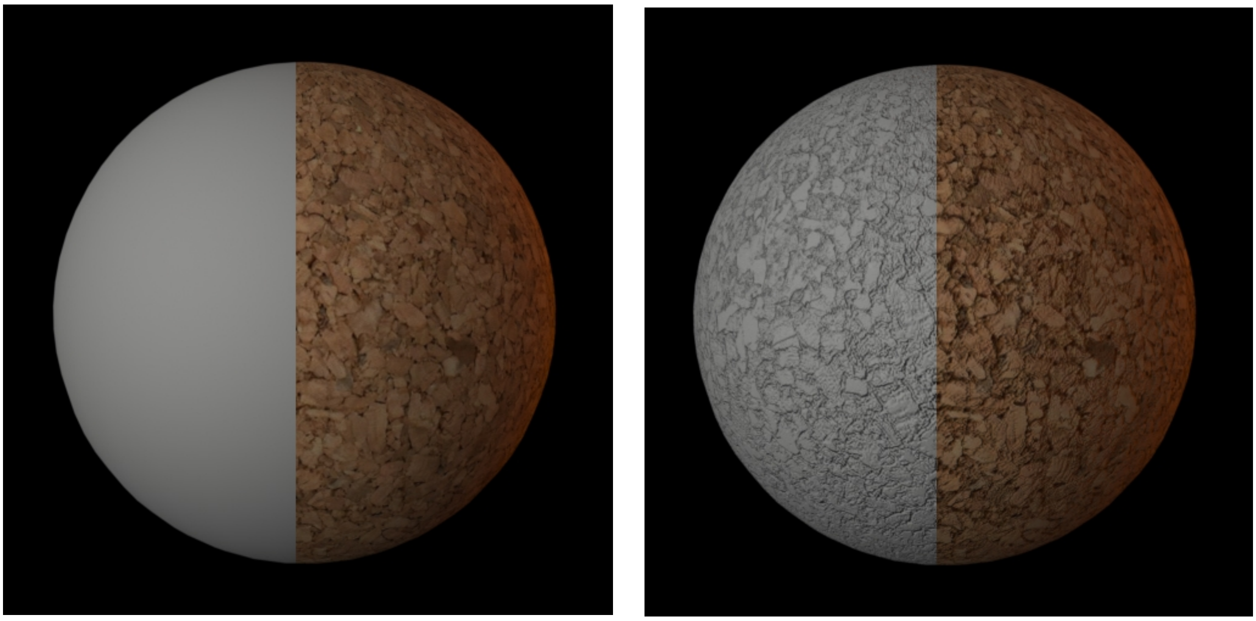

Surface without bump map applied.
Surface with bump map applied.
“Geometric” Normals Perturbed
Bump map

Normal perturbation caused by a bump map applied to a planar surface.

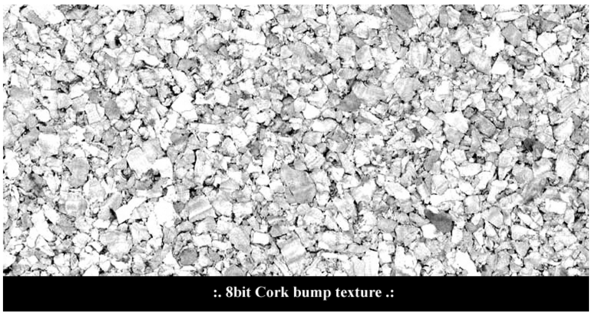
Note in the image above that geometric normals in the planar surface have been perturbed by the bump map, and point to a different direction.

The interaction of the surface and the lights in the scene make the brightness of the pixels on the surface change in response to this bump map giving the appearance of bumps and depressions in the surface.

The perturbed normal is then used instead of the original geometrical normal when shading the surface. James Blinn decided to change the original surface normals, instead of calculate perturbed normals, used in shading, separately. One problem with this method was that the simulated wrinkles did not extend all the way to silhouette of the object.



No bump mapping vs global bump mapping.



Cork bump texture

As seen in the above image, bump maps are usually seen as greyscale images, just because only height information is required. That said, increasing pixel depth of your bump map will let you define more levels of height, resulting in a more detailed bump texture.
8bits textures are commonly used, being able to define up to 256 levels of grey but anyway, it is sometimes useful to use 16bit textures to produce highly detailed bumps on your surface.

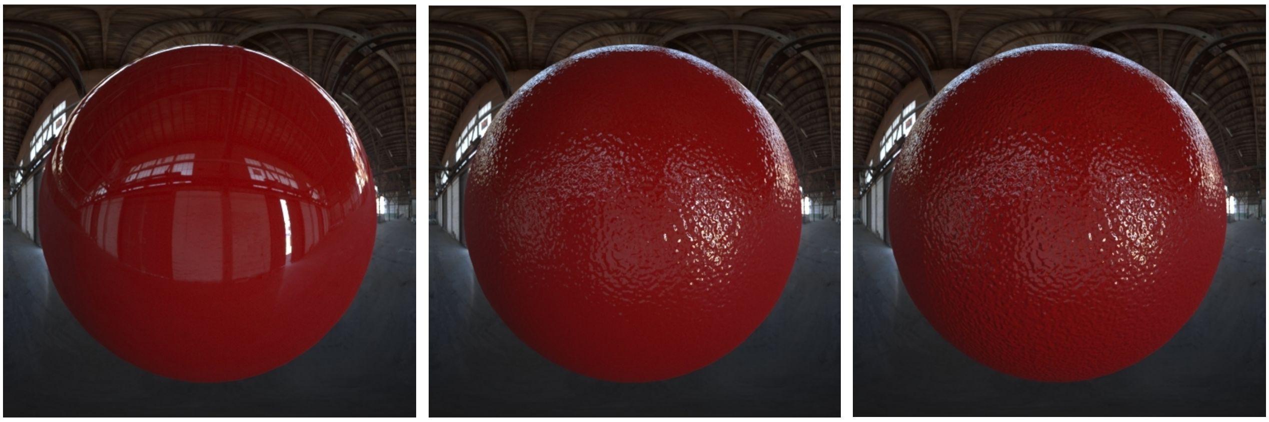
The leften image shows a sphere partly with some sort of cork texture applied and without any bump mapping.
On the other hand, the righten image shows the same sphere with bump mapping applied in both diffuse and specular channels.

Note that there is no geometrical difference between the two spheres, both are simple primitives consisting in 2380 triangles each.
When no bump map is applied, the image looks really fake, too smooth and without any depth. However, when bump map is properly applied the sphere becomes way more photorealistic, as small bumps and depressions have been added giving it a real cork look.

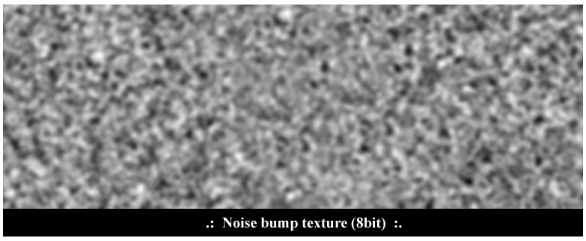
To implement bump mapping, a way must be found to perturb the surface normal that is independent of the surface's orientation and position. This requirement arises from the observation that if a bump mapped object was being animated, we would not want the surface wrinkles to change shape or appearance. Thus we should base the perturbation function on the local surface derivatives. [2 3 4]

Recent photo-realistic render engines accept selective bump mapping being possible to apply this technique to the difuse, highlights and reflection component of the surface separately [5], instead of global bumpmap, that applies the bump mapping method to all components together.

This possibility introduces the ability to render surfaces with no difuse bump, achieving the effect of a surface whose difuse surface is smooth, but covered by a bumpy lacquer coating.



No bump vs no difuse bump vs global bump
in some kind of shiny plastic material.



Noise bump texture

To show the results when using this selective bump mapping an hdr image has been added [see image based lighting (IBL) section] for both background and reflections channels and to illuminate the scene, just to get more realistic reflections in our spheres, so the difference between each would be clearly noticeable.

Note how the appearance change in the two righten above images. When no difuse bump is calculated, some kind of top shiny, reflective and bumpy, transparent coating is present but the difuse component of the surface is not being affected by the bump map.
However, in the last image where global bump map tecnique is applied, the difuse component is being affected by the height map, resulting in a shiny bumpy reflective sphere, being bumpy in difuse terms as well.

While a grayscale bump map can simulate only the up or down direction of the grooves, a normal map [see normal map section] has the additional advantage of specifying an angle, or the direction of the grooves.

Normal mapping is available in most recent photorealistic render engines and is strongly recommended if you want to give the impression of very strong bumps on a surface.
A bump map is usually enough for smaller bumps, but if you are trying to simulate bumps that are too large just raising the color range of a bump map, that may not be enough.

Back in 1998 bump maps were first implemented into hardware-accelerated realtime rendering. Scientists like I. Ernst, H. R¨usseler, H. Schulz, and O. Wittig. [6], G. Miller, M. Halstead, and M. Clifton [7], M. Peercy, J. Airey, and B. Cabral [8] built or proposed dedicated graphics hardware to implement bump maps.

When this was the case, and rendering in realtime was required, bump mapping is often referred to as several extra “passes” in the rendering process.

Researchers like T. McReynolds, D. Blythe, B. Grantham, and S. Nelson [9] use multipass techniques to implement bump maps with traditional graphics hardware.

References

[1] James Blinn, "Simulation of Wrinkled Surfaces", Computer Graphics, (Proc. Siggraph), Vol. 12, No. 3, August 1978, pp. 286-292.

[2] Max and Becker, "Bump Shading for Volume Textures", IEEE Computer Graphics and Animation, July 1994, pp. 18-20.

[3] Schlag, "Fast Embossing Effects on Raster Image Data", Graphics Gems IV, AP Professional, 1994, gem VIII.1, pp. 433-437.

[4] Watt and Watt, Advanced Animation and Rendering Techniques, Addison-Wesley, 1992, pp. 199-201.

[5] MentalRay architectural design visualization shader library, doc 1.7.6, 2007 Mental Images

[6] I. Ernst, H. R¨usseler, H. Schulz, and O. Wittig. Gouraud bump mapping. In Eurographics/ SIGGRAPH Workshop on Graphics Hardware, pages 47–54, 1998.

[7] G. Miller, M. Halstead, and M. Clifton. On-the-fly texture computation for realtime surface shading. IEEE Computer Graphics & Applications, 18(2):44–58, March–April 1998.

[8] M. Peercy, J. Airey, and B. Cabral. Efficient bump mapping hardware. In Computer Graphics (SIGGRAPH ’97 Proceedings), pages 303–306, August 1997.

[9] T. McReynolds, D. Blythe, B. Grantham, and S. Nelson. Advanced graphics programming techniques using OpenGL. In Siggraph 1998 Course Notes, July 1998.

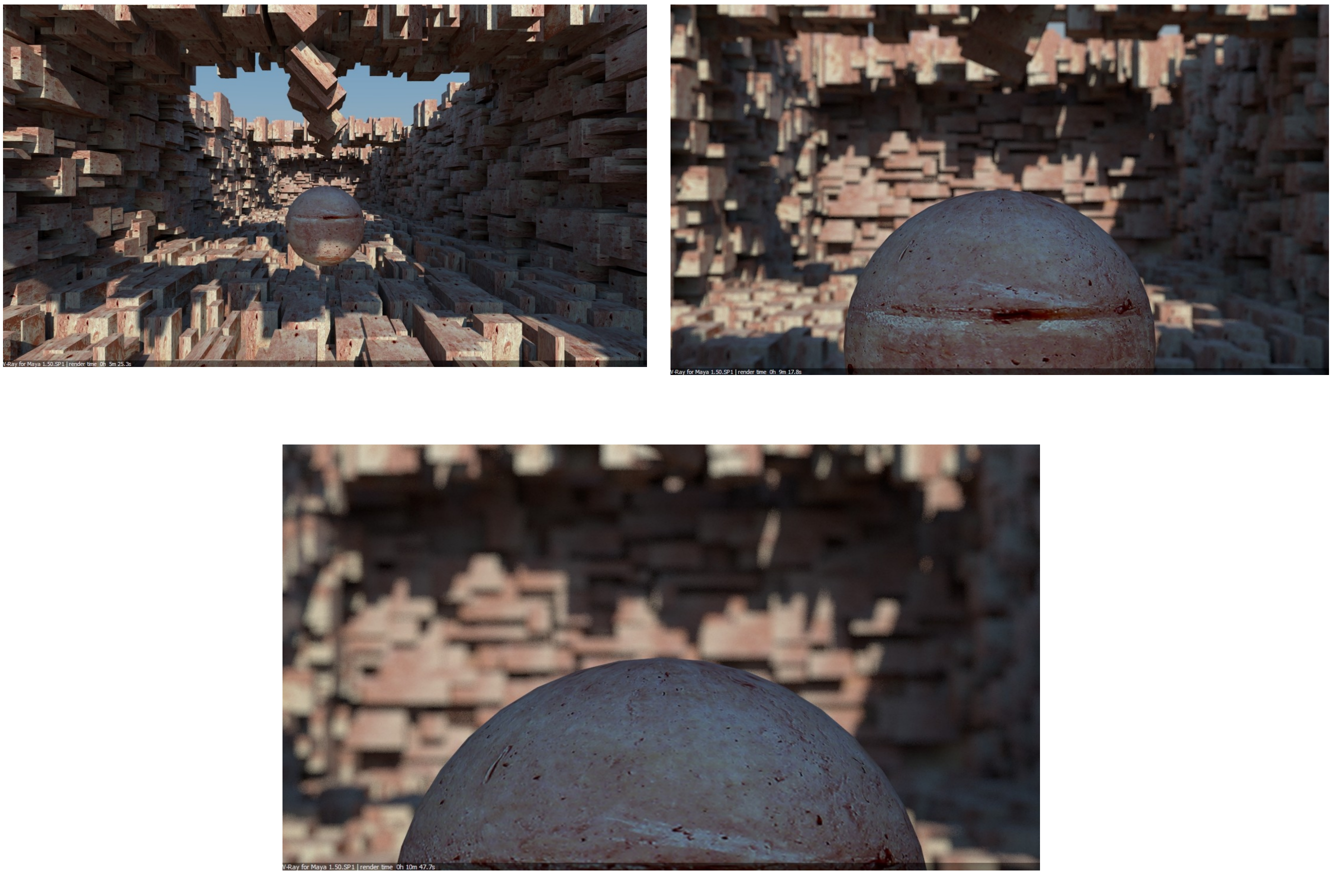
Depth of Field.

In optics, particularly as it relates to film and photography, depth of field (DOF) is the distance between the nearest and farthest objects in a scene that appear acceptably sharp in an image. Although a lens can precisely focus at only one distance at a time, the decrease in sharpness is gradual on each side of the focused distance, so that within the DOF, the unsharpness is imperceptible under normal viewing conditions. [1]

In some cases, it may be desirable to have the entire image sharp, and a large DOF is appropriate. In other cases, a small DOF may be more effective, emphasizing the subject while de-emphasizing the foreground and background. In cinematography, a large DOF is often called deep focus, and a small DOF is often called shallow focus.

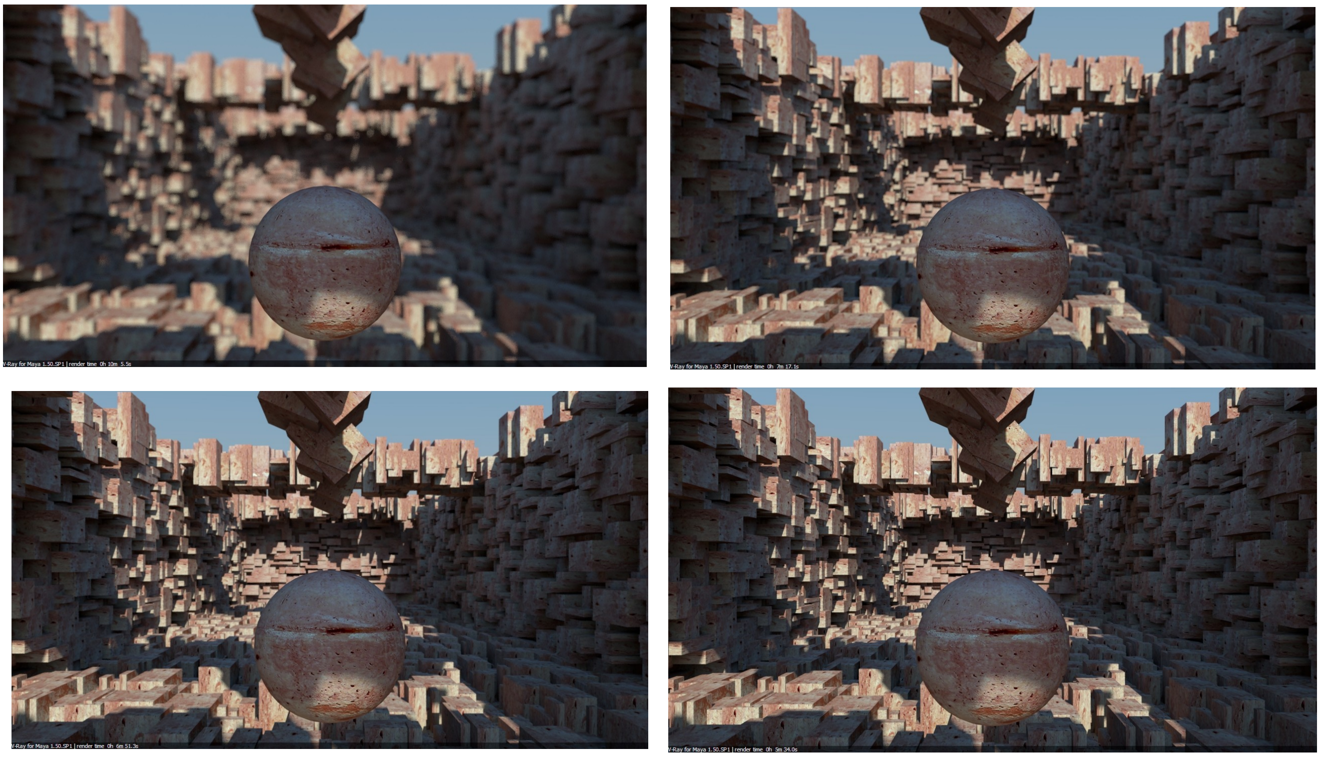
The DOF is determined by the camera-to-subject distance, the lens focal length, the lens f-number, and the format size or circle of confusion criterion.
All these four parameters will be present in the photorealistic render engine of our choice, and should be considered accordingly.

For a given format size, at moderate subject distances, DOF is approximately determined by the subject magnification and the lens *f*-number. For a given *f*-number, increasing the magnification, either by moving closer to the subject or using a lens of greater focal length, decreases the DOF; decreasing magnification increases DOF. [2]



Images showing differences using focal length values.

For a given subject magnification, increasing the *f*-number increases the DOF; decreasing *f*-number decreases DOF.



Images showing differences using f-number values.

As could be stated from the above images, for a given subject framing and camera position, the DOF is controlled by the lens aperture diameter, which is usually specified as the f-number, the ratio of lens focal length to aperture diameter.

Reducing the aperture diameter (increasing the *f*-number) increases the DOF; however, it also reduces the amount of light transmitted, and increases diffraction, placing a limit on the extent to which DOF can be increased by reducing the aperture diameter.

That limit is not present in 3D render engines, where we can increase the exposure of the rendered image without a limit, nor introducing any noise.

The amount of light transmitted (exposure of the image) has not changed in the four above images, because the ISO [see ISO section] exposure and shutter speed settings had been changed accordingly as explained below in the next paragraphs.

The exposure of a photograph is controlled by only three camera settings: ISO speed, lens aperture (f-number or f-stop) and shutter speed

Cameras can adjust these settings in steps of one, half or one third "stops." An increase of one stop (full stop) results in a doubling of brightness. A decrease of one stop results in the halving of brightness. [3]

This is a list of common f-numbers in steps of one stop. The brightness decreases down the list.

- f1 (square root 1) f1.4 (square root 2)
- f2 (square root 4) f2.8 (square root 8)
- f4 (square root 16) f5.6 (square root 32)
- f8 (square root 64) f11 (square root 128)
- f16 (square root 256) f22 (square root 512)
- f32 (square root 1024)

A smaller aperture (larger f-number) increases the depth-of-field, bringing more objects into focus. Focusing errors can be covered-up if the depth-of-field is large enough. A large aperture is used to throw the background out of focus, drawing attention to the in-focus subject.

This is a list of common shutter speeds in steps of one stop, measured in seconds. The brightness decreases down the list. For simplicity (to get round numbers), the speeds don't exactly halve each time.

- 1 1/2 1/4 1/8 1/15 1/30
- 1/60 1/125 1/250 1/500 1/1000 1/2000
- 1/4000 1/8000

A fast shutter speed is used to get clear photos, free from motion-blur [See MotionBlur section]. There are times when a slower shutter speed is used to purposely blur out moving objects.

This is a list of common camera ISO [see ISO section] sensitivities in steps of one stop. The brightness decreases down the list. The ISO scale is linear, so the speeds simply halve for each stop down the list.

- ISO 6400 ISO 3200 ISO 1600 ISO 800
- ISO 400 ISO 200 ISO 100 ISO 50

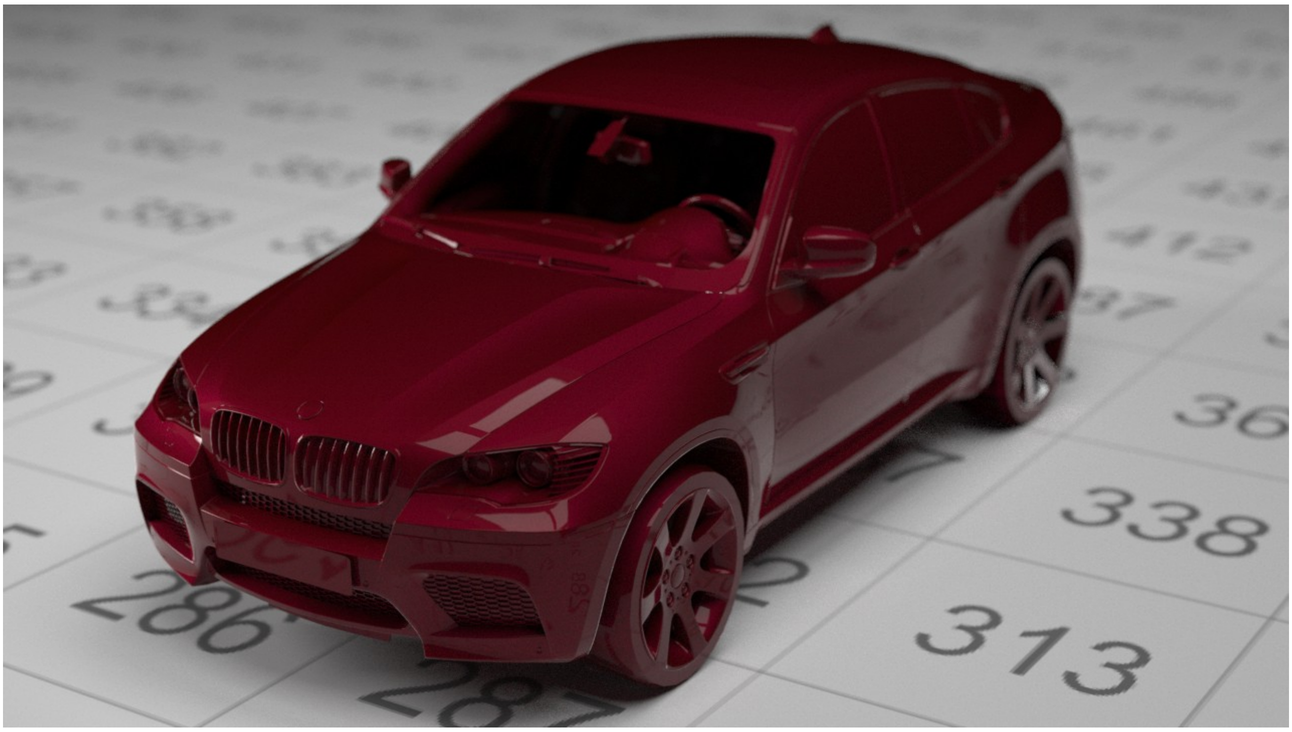
The above three lists decrease in brightness by one stop, each step down the list. If one setting is changed, the same exposure setting is maintained by simply adjusting one of the other settings in the opposite direction.

Let's say a setting of ISO 400, f2, 1/60 seconds results in the correct exposure. However the depth-of-field at f2 is too shallow and causes too many objects to be out of focus. [4]

It would be necessary to decrease the aperture to for example f5.6 for more depth-of-field. That's a 3 stop decrease down the list.
Therefore, it would be necessary to make it up with an equivalent increase up the list in shutter or ISO exposure.
One possibility could be to increase shutter brightness one stop, going from 1/60 to 1/30 seconds. And subsequently making up the rest of the brightness by increasing ISO two stops, from ISO 400 to ISO 1600.

The new setting of ISO 1600, f5.6 and 1/30 seconds will give the same exposure but with greater depth-of-field and more motion-blur.

By understanding the three lists above, settings can be easily adjusted so ISO, motion-blur and depth-of-field is correctly balanced. It is simply a case of balancing changes: any adjustment up or down the list in one setting, needs to be balanced by an equal adjustment down or up the list for the other settings.



Depth of Field technique applied to a scene.
(Rendered with MaxwellRender v2.5)

Note the sharpness in the front of the vehicle compared to the bottom part of it, emphasizing the foreground of the image.

References

[1] Couzin, Dennis. 1982. Depths of Field. SMPTE Journal, November 1982, 1096–1098

[2] Langford, Michael J. 1973. Basic Photography. 3rd ed. Garden City, NY: Amphoto

[3] Shipman, Carl. 1977. SLR Photographers Handbook. Tucson: H.P. Books

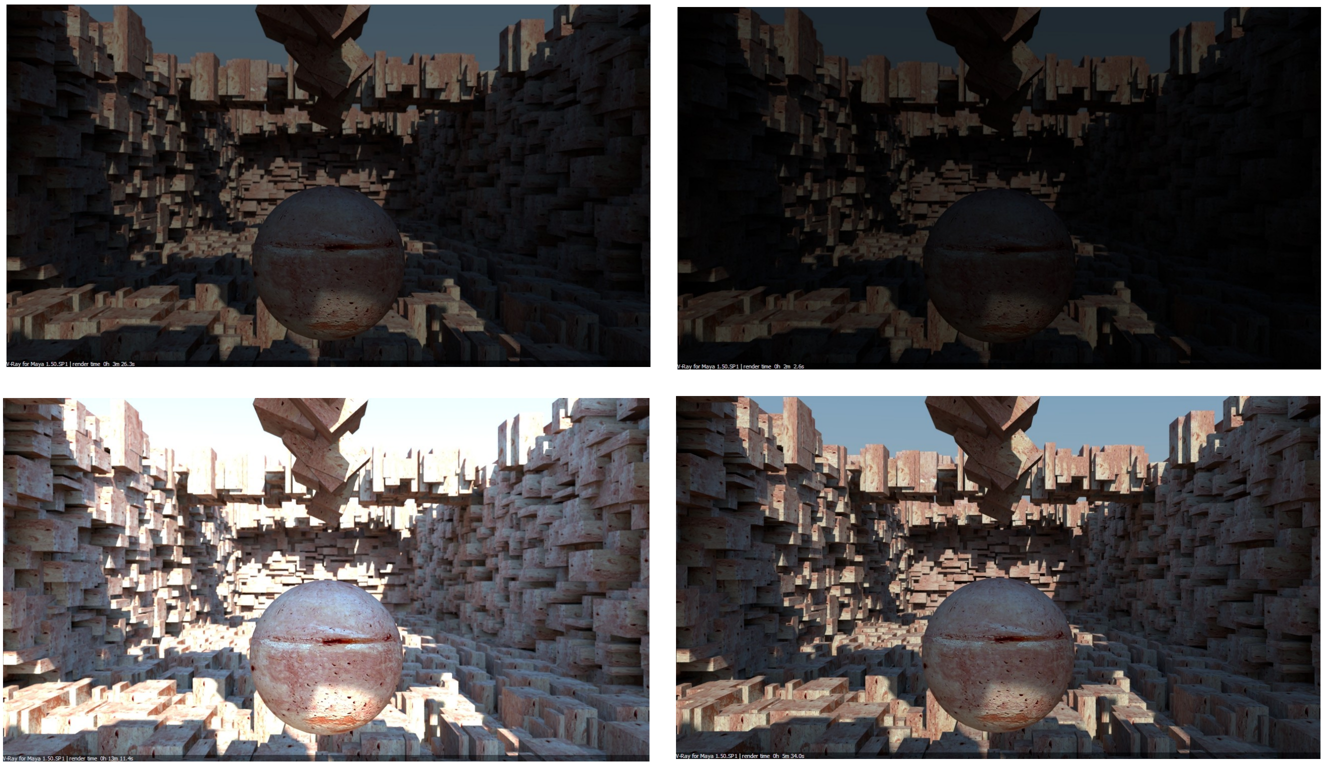
[4] Suite101: Relationship Between Camera ISO, Aperture and Shutter Speed

ISO.

Film speed is the measure of a photographic film's sensitivity to light, determined by sensitometry and measured on various numerical scales, the most recent being the ISO system.

Relatively insensitive film, with a correspondingly lower speed index requires more exposure to light to produce the same image density as a more sensitive film, and is thus commonly termed a slow film. Highly sensitive films are correspondingly termed fast films. A closely related ISO system is used to measure the sensitivity of digital imaging systems. In both digital and film photography, the reduction of exposure corresponding to use of higher sensitivities generally leads to reduced image quality (via coarser film grain or higher image noise of other types). In short, the higher the film speed, the grainier the image will be.

That is not the case of ISO in rendering softwares, where not a real camera is being used. In 3D rendering, high ISO values do not lead to any noise in the final image. The images below show different ISO values in the same rendering scene.


V-Ray for Maya 1.50.SP1 | render time 0h 3m 26.3s
V-Ray for Maya 1.50.SP1 | render time 0h 2m 2.6s
V-Ray for Maya 1.50.SP1 | render time 0h 13m 11.4s
V-Ray for Maya 1.50.SP1 | render time 0h 5m 34.0s

Increasing ISO values in a 3D scene.
(Rendered with MentalRay)

Note that no noise is present even in the bottom left image, with high film sensitivity.

This is a list of common camera ISO sensitivities in steps of one stop. The brightness decreases down the list. The ISO scale is linear, so the speeds simply halve for each stop down the list. [2]

- ISO 6400 ISO 3200 ISO 1600 ISO 800
- ISO 400 ISO 200 ISO 100 ISO 50

The above three lists decrease in brightness by one stop, each step down the list.

References

[1] A.L.M. Sowerby (Ed.) (1961). *Dictionary of Photography: A Reference Book for Amateur and Professional Photographers* (19th ed.). London: Iliffe Books Ltd.. pp. 582–589.

[2] *Suite101: Relationship Between Camera ISO, Aperture and Shutter Speed*

Motion Blur.

Motion blur is the apparent streaking of rapidly moving objects in a still image or a sequence of images such as a movie or animation. It results when the image being recorded changes during the recording of a single frame, either due to rapid movement or long exposure time.

When the 3D camera creates an image, that image does not always represent a single instant of time. [1]
Because of technological constraints or artistic requirements, the image may represent the scene over a period of time. As objects in a scene move, an image of that scene must represent an integration of all positions of those objects, as well as the camera's viewpoint, over the period of exposure determined by the shutter speed.

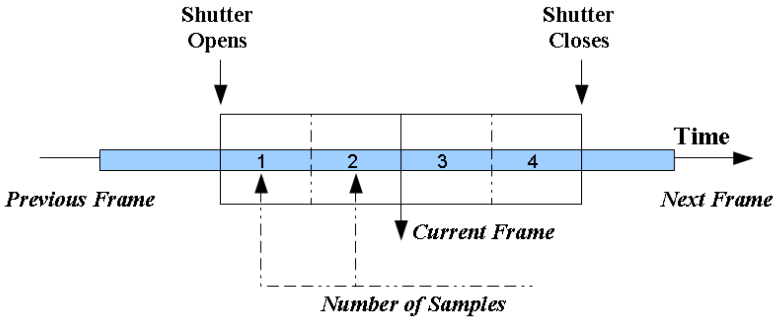
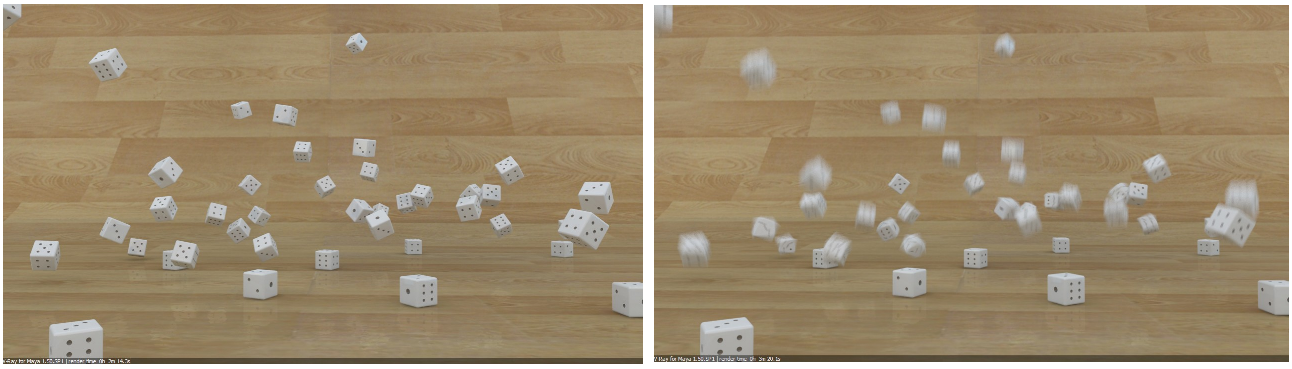


Diagram showing ShutterSpeed invoved in MotionBlur.

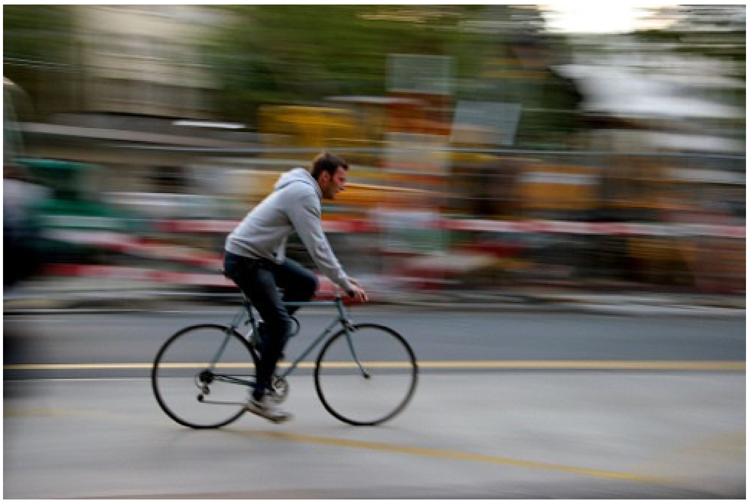
On the other hand, the number of samples indicate the accuracy of the MotionBlur. When more samples per frame are computed, the MotionBlur effect will be smoother, as the renderer will compute motion blur every 1/(FPS*Number of sampes) seconds, and then interpolate all this number of samples.

In such an image, any object moving with respect to the camera will look blurred or smeared along the direction of relative motion. This smearing may occur on an object that is moving or on a static background if the camera is moving. In a real photograph, this looks natural because the human eye behaves in much the same way, that is the reason of Motion Blur being that important in photorealistic render engines.


V-Ray for Maya 1.50.SP1 | render time 0h 2m 14.3s
V-Ray for Maya 1.50.SP1 | render time 0h 3m 20.1s

Same scene rendered with and without MotionBlur effect.
(Rendered with MentalRay)

Because the effect is caused by the relative motion between the camera, and the objects and scene, motion blur may be avoided by panning the camera to track those moving objects. In this case, even with long exposure times, the objects will appear sharper, and the background more blurred.



Real photograph of background blurred due to MotionBlur.

Without this simulated effect each frame shows a perfect instant in time (analogous to a camera with an infinitely fast shutter), with zero motion blur.

For still camera rendering, the amount of motion blur in an image is controlled by the Shutter Speed parameter. The lower the Shutter Speed is, the longer the shutter is open, and the more pronounced the motion blur will be.

However, for animations, the shutter speed is generally fixed to a certain speed, usually 24 frames per second (1/24). Since film cameras cannot control the amount of motion blur by changing the shutter speed, they instead have a rotating disc with an adjustable pie-shaped cut-out on it, which controls how long each frame is exposed. The width of the cut-out is called Shutter Angle, and is expressed in degrees. [2]

Fully open (180 degrees) will yield the maximum amount of motion blur, while a very narrow setting (say, 15 degrees) will produce very subtle motion blur.
MaxwellRender, 3Delight and Vray are a few examples of photorealistic render engine that makes use of this technique to produce MotionBlur.

It is worth noting that this feature automatically translates your usual ISO [see ISO section] / Shutter speed settings in combination with the Shutter Angle, so your animation exposure will match your still image exposure, while producing the proper amount of motion blur.

References

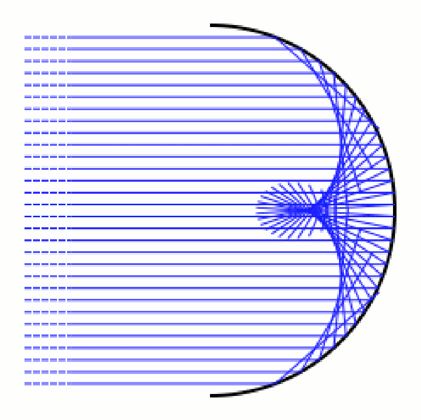
[1] Michael Potmesil Bell Laboratories Indranil Chakravarty .Modeling motion blur in computer-generated images. SIGGRAPH '83

[2] NextLimit Technologies, MAXWELL RENDER 2.5 USER MANUAL

Caustics

The word *caustic*, comes from the Greek καυστός, burnt, via the Latin *causticus*, burning.

Caustics are light patterns that are created when light rays emitted from a light source are bent by one intermediate medium and illuminate a diffuse surface via one or more specular reflections or transmissions.
Caustics can also refer to the curves to which light rays are tangent, defining a boundary of an envelope of rays as a curve of concentrated light.



Reflective caustic generated from a circle and parallel rays

This effect was first introduced and studied by Tschirnhausen [1] in 1682 and scientists like Huygens, Quetelet, Lagrange, and Cayley wrote later studies about it.

Most modern photorealistic render engines support caustics both normal and volumetric [see volumetric caustics section] but its simulation cannot be achieved efficiently using standard traditional raytracing methods like Monte Carlo [See Raytracing section] since predicting the potential specular paths to a light source from any given surface is a difficult and in many situations impossible task. [2]

An accurate computation of this effect can be easily obtained using a separate caustic photon map showing a radiance estimate. [See Photon Map section]

Usually render engines generate the photon map in a preprocessing step in which photons are emitted from the light sources and traced through the scene using photon tracing technics.

Depending on the density of photons used in the photon map, the simulation of caustics will be more or less acurate.

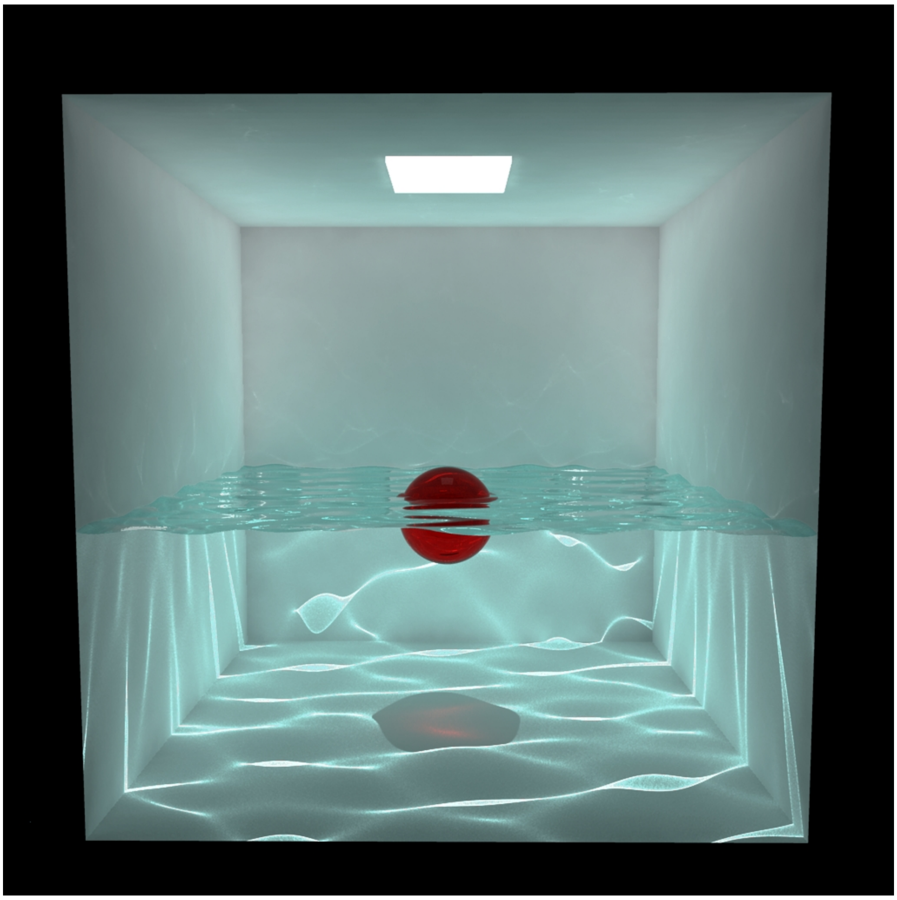
Some examples of this effect are:

- The light patterns created on the bottom of a swimming pool as light is refracted by the water surface and reflected by the diffuse pool bottom.

- Light being focused by a glass of water onto a diffuse table material.

- The light emanating from the headlights of a car: the light is emitted by the filament of a light bulb, reflected by a parabolic mirror reflector (thereby being focused in the forward direction), and reflected by the diffuse road surface. [3]

- The rainbow. Scattering of light by raindrops causes different wavelengths of light to be seen in arcs of differing radius. Each such arc is a directional caustic .



Example of caustics produced by a light emiting surface through a translucent medium in some sort of CornellBox [see Cornell Box section] scene.

References

[1] *Weisstein, Eric W., "Caustic" from mathworld.wolfram.com*

[2] *Global Illumination using Photon Maps, Henrik Wann Jensen. Department of Graphical Communication. The Technical University of Denmark, http://www.gk.dtu.dk/~hwj*

[3] *Fast Photorealistic Techniques to Simulate Global Illumination in Videogames and Virtual Environments, Alex Mendez Feliu. Universitat Politecnica de Catalunya, Departament de Llenguatges i Sistemes Informatics.*

Cornell Box

The Cornell Box is basicaly a rendering accuracy test and was created by Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaile. [1]

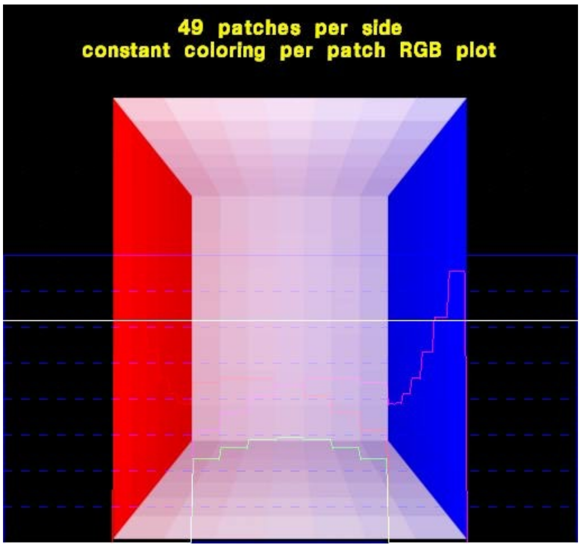
The goal of this test is to determine the accuracy of rendering software by comparing the rendered scene with a real photograph of the same scene.

This test symbolize an approach to physically based rendering. The Cornell box is a simple physical environment consisting of a box with specific characteristcs and a light source in the center of a white ceiling photographed with a calibrated CCD camera. The right wall is green, and the left wall red, while the back and floor walls are white.

The exact settings such as emission spectrum of the light source, reflectance spectra of all the surfaces, exact position and size of all objects inside the box, walls and light source and camera are then measured from the scene. All specifications of the geometry and material properties of the box are provided for public use courtesy of the Cornell University Program Of Computer graphics.

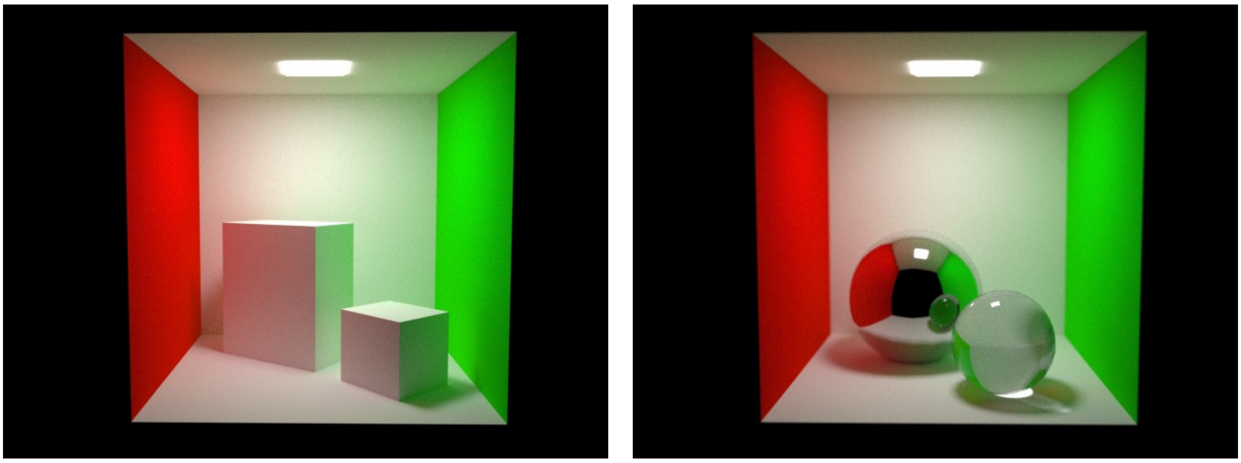
The same scene is afterwards modelled and rendered with some software renderer, and the output file is compared with the photography.

Back in 1984, form factors of the original Cornell Box were computed analytically, so no occluding objects were included inside the box.



Original Cornell Box [1]

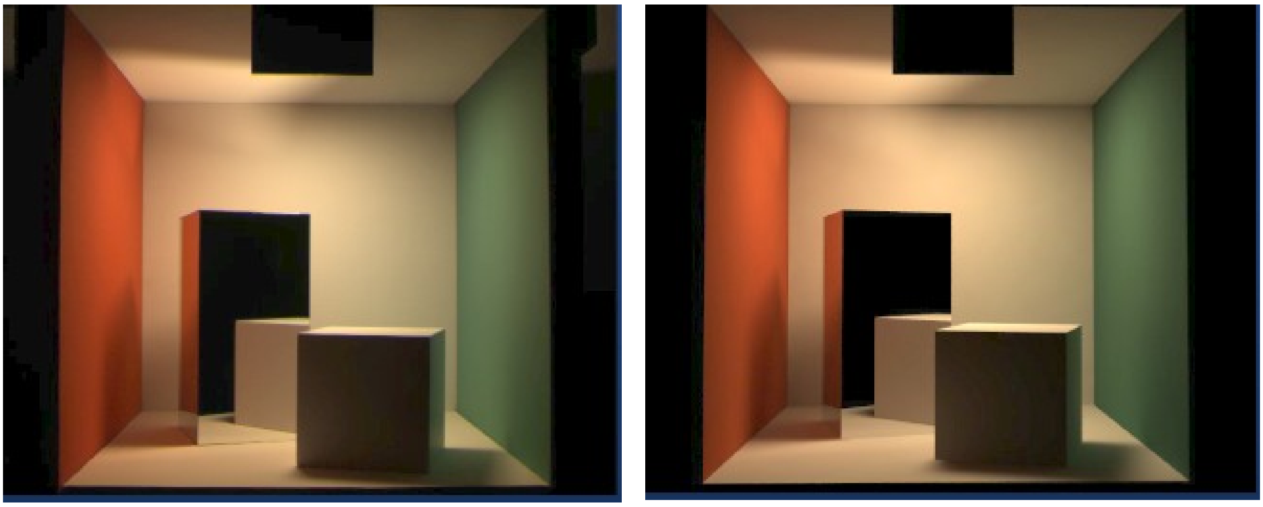
Nowadays objects are often placed inside the box. The first objects placed were two white boxes or two spheres, one with perfect mirror surface and the other made of glass material, to test photon mapping. [See Photon Map section]



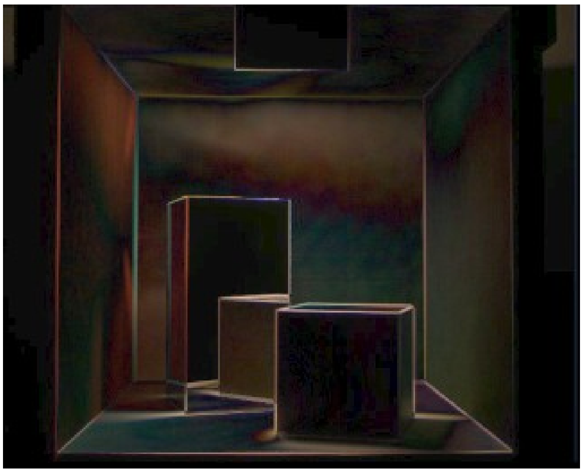
Objects commonly placed in a Cornell Box

As shown above, the physical properties of the box are designed to show diffuse interreflection. For example, some light reflects off the red and green walls and bounce onto the white walls, so parts of the white walls appear slightly red and green respectively.

The method used for comparing the real image vs the simulated one consists simply of a pixel by pixel subtraction of one image from the other.
Note that some discrepancies can be seen in the shadowing on large surfaces due to the meshing used in the rendering software. In other hand, edge effects stem from misregistration between the actual positions and dimensions of the physical objects and the coordinates used for rendering. [3]



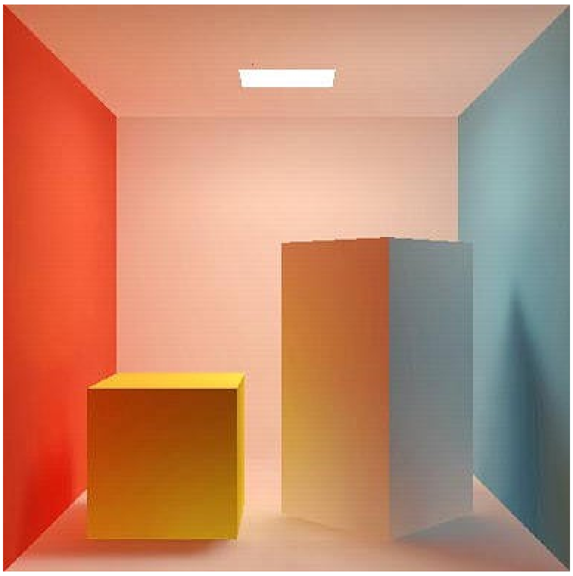
Real photograph of a Cornell Box vs a rendered image of the same environment. [3]



Pixel by pixel difference of real and simulated Cornell Boxes. [3]

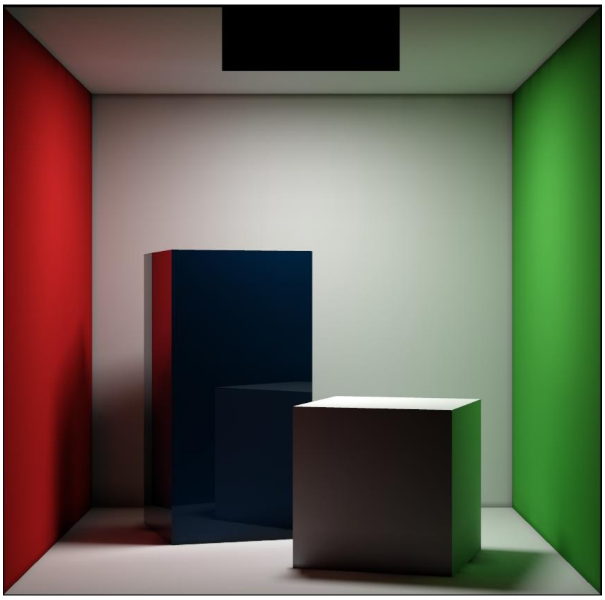
Several methods were later developed to improve the accuracy of the simulated scene.

Michael F. Cohen and Donald P. Greenberg [4] used scan conversion algorithms, available in hardware to calculate form factors as a radiosity solution, and called them hemi-cube form factors. Note that shadows from occluding objects are present in the following image due to the use of this method.



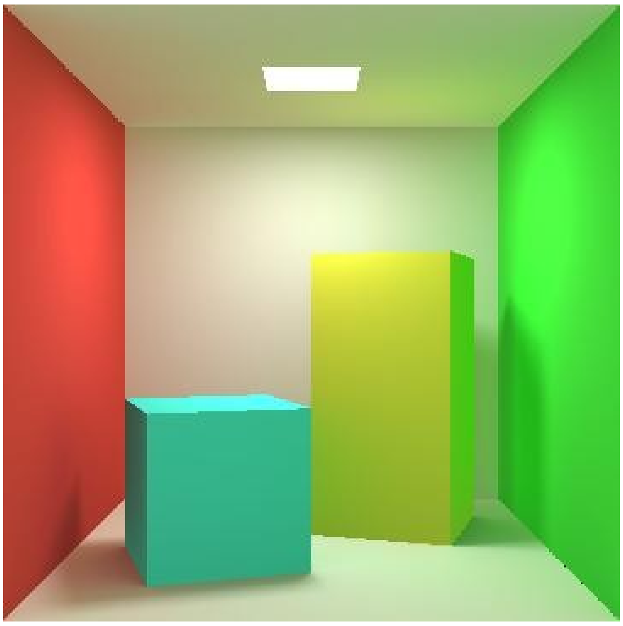
Hemi-cube form factors radiosity solution [4]

Francois Sillion introduced another solution using spherical harmonics to represent Bidirectional Reflectance Distribution Functions [See BRDF section] for surfaces in the simulated environment. Note that Sillion changed the right wall from blue, that helped demonstrate the familiar color bleeding effects of radiosity solutions, to green to get a more balanced solution. [2]



Spherical harmonics solution, François Sillion [2]

In 1992, Dani Lischinski, Filippo Tampieri, and Donald P. Greenberg [5] , developed a discontinuity meshing software to simulate a more accurate radiosity solution.



Simulation using discontinuity meshing software. [5]

References

[1] Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaile. Modeling the Interaction of Light Between Diffuse Surfaces. Siggraph 1984.

[2] History of the Cornell Box, Cornell University of Computer Graphics.

[3] Cornell Box Comparison, Cornell University of Coumputer Graphics.

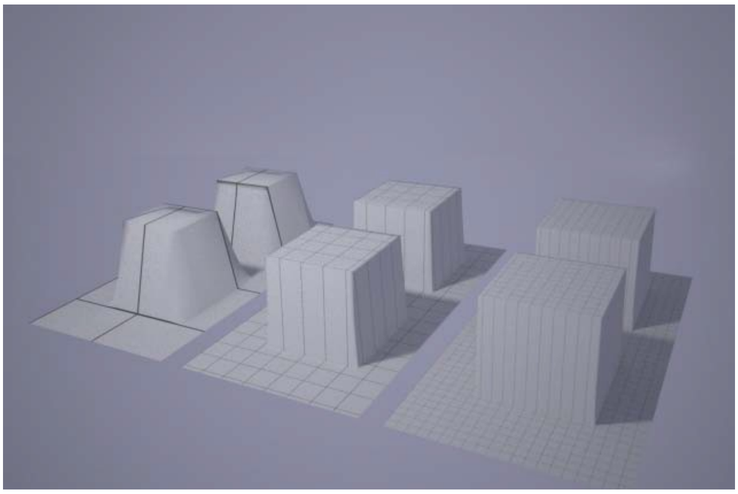
[4] The Hemi-Cube, A Radiosity Solution for Complex Environments. Michael F. Cohen and Donald P. Greenberg Vol. 19, No. 3, July 1985, pp. 31-40.

[5] *Dani Lischinski, Filippo Tampieri, and Donald P. Greenberg. Discontinuity meshing for accurate radiosity. IEEE Computer Graphics and Applications 12(6), 1992, pp. 25–39.*

Displacement Mapping.

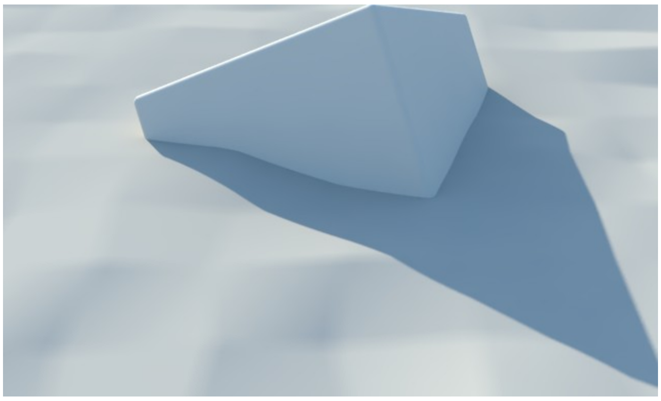
In 1984, Cook [1] introduced a method for adding small-scale detail to surfaces called displacement mapping.
Unlike bump mapping [see bump mapping section], which perturbs surface normals and thus affects only the shading of surfaces, displacement mapping variates the positions of surface elements. This leads to effects not possible with bump mapping, such as surface features that occlude each other and nonpolygonal silhouettes.

The usual implementation of displacement mapping iteratively tessellates a base surface, pushing vertices out along the normal of the base surface, and continuing until the polygons generated are close to the size of a single pixel. [2]

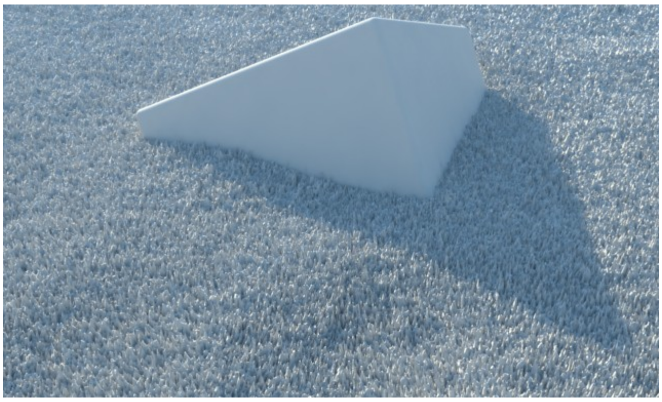


Differences between tessellation iterations in Displacement Mapping
MaxwellRender v2.5 Manual, NextLimit.

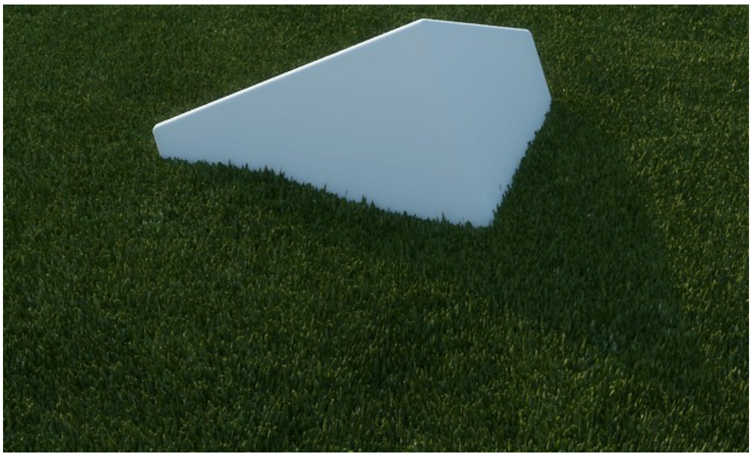
The following image shows a rendering of a grass carpet surface in which occlusion between features contributes to the illusion of depth.



Original plane mesh without displacement.
(Rendered in Vray for Maya)



Plane mesh with displacement map applied.
(Rendered in Vray for Maya)

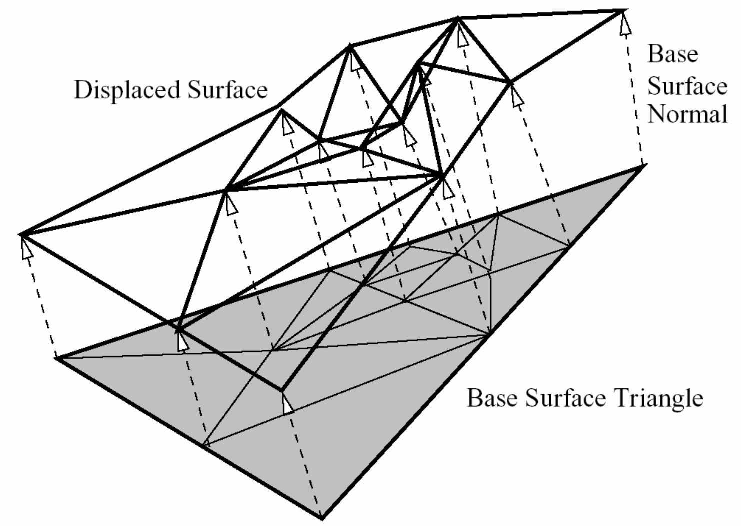


Displaced plane mesh with diffuse texture.
(Rendered in Vray for Maya)

As said earlier, in the case of displacement mapping the surface is actually modified, which leads to correct outline.

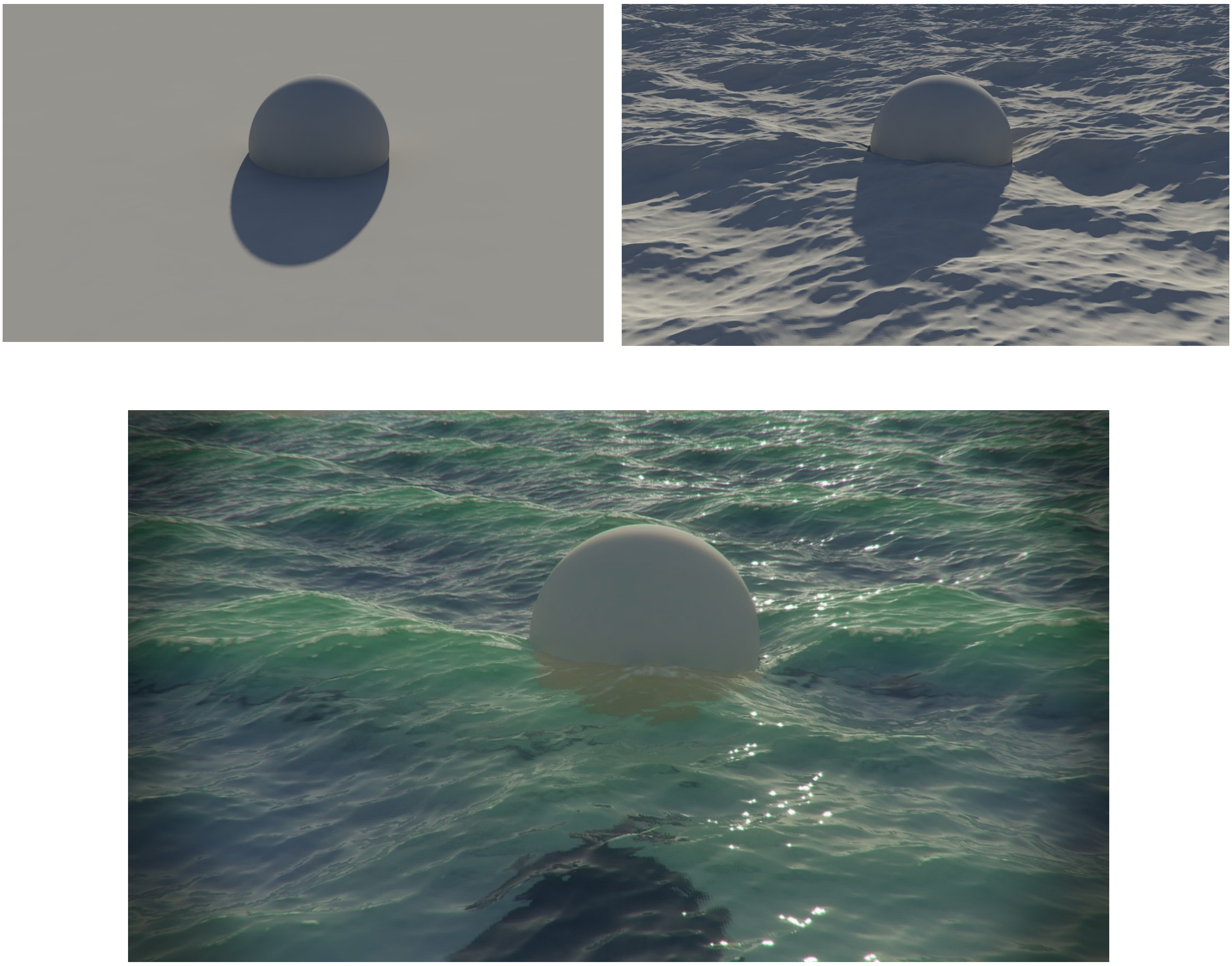
That is easily noticeable in the above images, where shadows and GI *[see global illumination section]* are correctly calculated. In the case of bump mapping, although the surface appears modified, the outline and the shadow stay the same, resulting in a non photorealistic result.

Note that displacement is different from other kinds of shading, since it needs to modify the actual object surface. Therefore an object must be displaced before it can be rendered.



Displacement mapping overview diagram.
(Doggett et Hirche, 2000) [5]

In the above image, the original triangle (shaded grey) is tesselated in several sub-triangles. The resulting vertex are then displaced according to the displacement map in the direction of the base triangle's normal.[6]



An image of a complex object created by displacement mapping a PLANE. The figure is ray traced with global illumination. 28K triangles are stored.

All that said, displacement mapping appears to be an important feature in render engines, where highly detailed models are required and finite memory capabilities are present.

However, the processing power of computers rendering the objects sets limits to the complexity of the models.
Render times can vary greatly. These three factors play an important role in render times: [8]

- The base mesh vs. Tesselation gain value.
- How many surfaces are displaced in the scene.
- The height of displacement (higher displacements will increase render times).

For example, a common usage of displacement may be for a brick wall seen from far away, taking up 30-40% of the rendered image. In this case, low height and low tesselation gain values can be used, and the impact on render times will be minimal. On the other hand, a close-up render of a displacement object taking up the whole image, using high gain values, will need more time to render clean.

Nowadays, most production quality photorealistic render engines offer some kind of high detailed displacement mapping feature, often called micro-polygon displacement.

Renderers using the Reyes algorithm, or similar approaches based on micropolygons, have allowed displacement mapping at arbitrary high frequencies since they became available almost 20 years ago.

The first commercially available renderer to implement a micropolygon displacement mapping approach through REYES was Pixar PhotoRealistic's Renderman.
Micropolygon renderers commonly tessellate geometry themselves at a granularity suitable for the image being rendered.
That is: the modeling application delivers high-level primitives to the renderer.
The renderer then tessellates this geometry into micropolygons at render time using view-based constraints derived from the image being rendered.

The difference between displacement mapping in micropolygon renderers vs. displacement mapping in a non-tessellating (macro)polygon renderers can often lead to confusion in conversations between people whose exposure to each technology or implementation is limited.
Even more so, as in recent years, many non-micropolygon renderers have added the ability to do displacement mapping of a quality similar to what a micropolygon renderer is able to deliver, naturally.
To distinguish between the crude pre-tessellation-based displacement these renderers did before, the term sub-pixel displacement got introduced to describe this feature.

Sub-pixel displacement commonly refers to finer re-tessellation of geometry that was already tessellated into polygons. This re-tessellation results in micropolygons or often microtriangles. The vertices of these then get moved along their normals to archive the displacement mapping.

True micropolygon renderers have always been able to do what sub-pixel-displacement achieved only recently, but at a higher quality and in arbitrary displacement directions.

References

[1] Cook, Robert L. 1984. “Shade Trees.” In *Computer Graphics (Proceedings of SIGGRAPH* 84) 18(3), pp. 223–231.

[2] Per-Pixel Displacement Mapping with Distance Functions, William Donnelly. University of Waterloo

[5] Doggett M., Hirche J. 2000. Adaptive View Dependent Tessellation of Displacement Maps. *Proceedings of Eurographics/SIGGRAPH Workshop on Graphics Hardware 2000*. Interlaken, Switzerland, August 21-22, 2000. ACM.

[6] Direct Ray Tracing of Displacement Mapped Triangles, Brian Smits Peter Shirley Michael M.

Stark University of Utah. besjshirleyjmstark@cs.utah.edu

[7] *Becker & Max, "Smooth Transitions between Bump Rendering Algorithms",*
University of California, Davis and Lawrence Livermore National Laboratory.

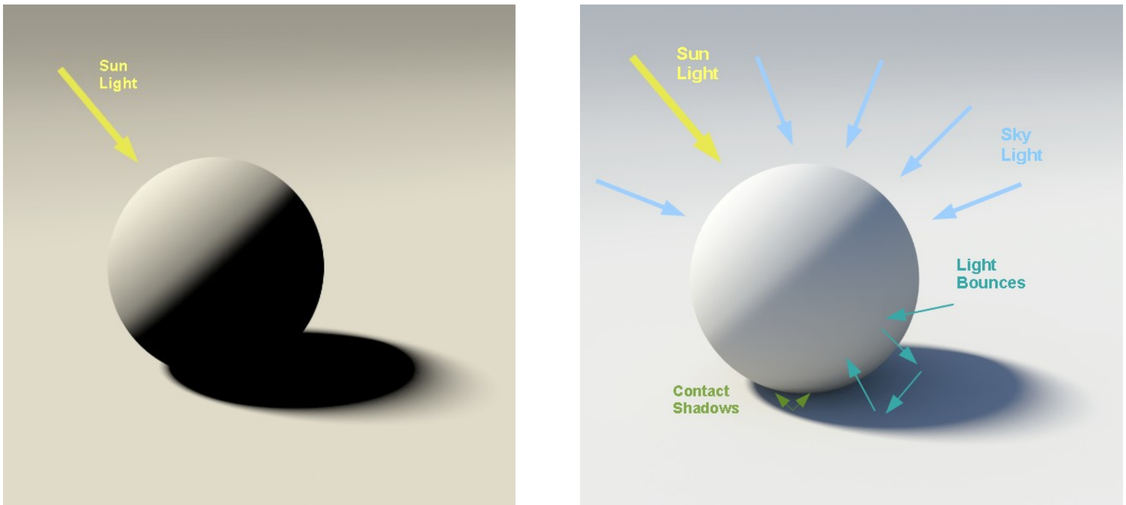
[8] NextLimit MaxwellRender v2.5 Manual. Displacement Section, NextLimit Technologies 2011.©
Copyright 2011 Next Limit SL

Global illumination

Global illumination refers to techniques that take indirect lighting, such as inter-object reflections, into account when computing the light distribution in a scene.

Therefore, such techniques take into account not only the light which comes directly from a light source (direct illumination), but also subsequent cases in which light rays from the same source are reflected by other surfaces in the scene, whether reflective or non (indirect illumination). [1]

The images below show the difference between direct and indirect illumination rendered using radiosity [see radiosity section].



Direct Illumination vs Indirect Illumination diagram.

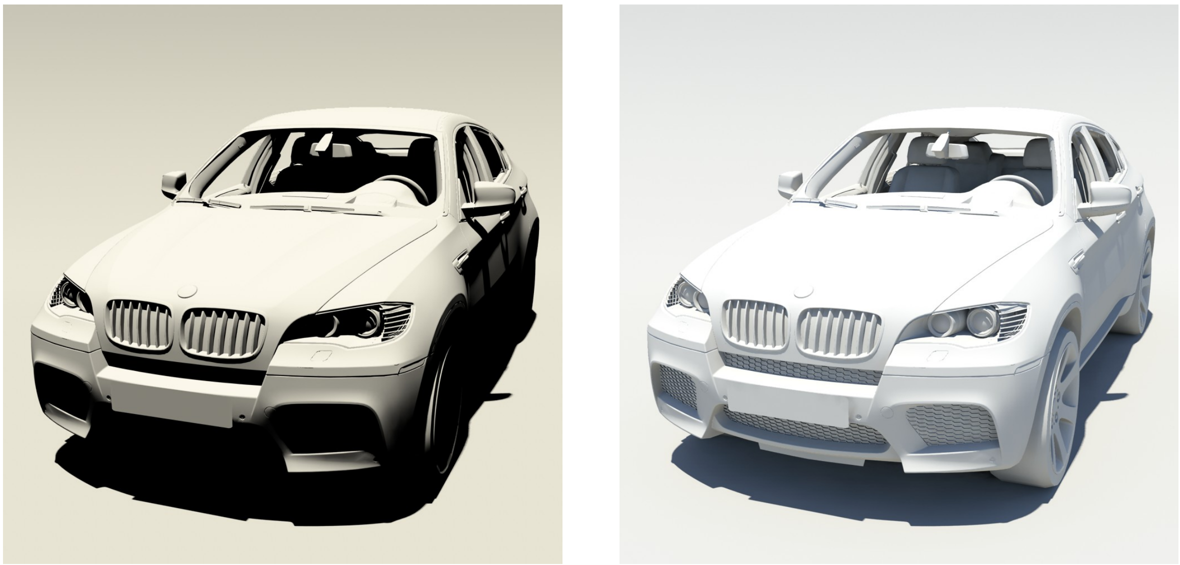
Global illumination techniques were first introduced to the computer graphics community in the late 70's early 80's [2] with the introduction of Raytracing [see raytracing section] and Radiosity [see radiosity section]. These techniques allowed for far higher realism by simulating specular and diffuse inter-object reflections, respectively.

In 1986 the global illumination problem was summarised by James Kajiya in the Rendering Equation. [3] Kajiya also proposed Path tracing [see pathtracing section] as a method to solve it. While there are numerous global illumination techniques available, they all approximate the same mathematical equation, each with their own particular benefits and drawbacks. In the next pages a brief introduction of each algorithm will be exposed.

Radiosity, Raytracing, Photon mapping, Pathtracing, Metropolis Light Transport, Irradiance Caching, are examples of algorithms used in global illumination, some of which may be used together to yield more accurate results.

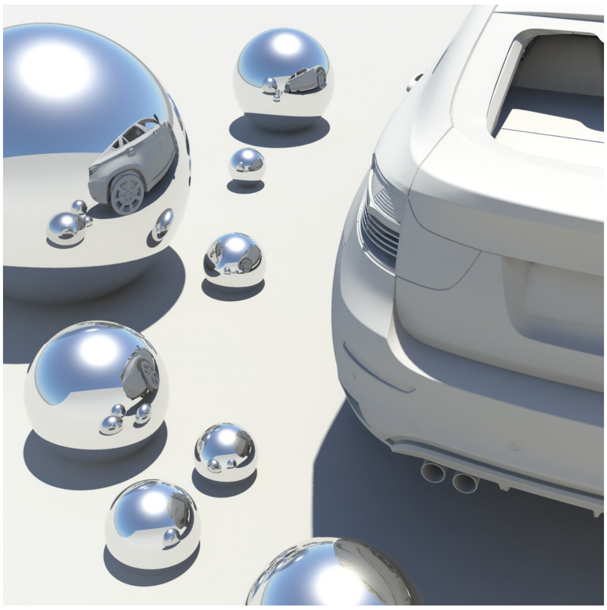
These algorithms model diffuse inter-reflection which is a very important part of global illumination; however most of these (excluding radiosity) also model specular reflection, which makes them more accurate algorithms to solve the lighting equation and provide a more realistically illuminated scene.

However, radiosity could be combined with a technique that computes specular reflection, such as raytracing, to take advantage of both techniques.



Direct vs Indirect outdoor lighting using radiosity.
(Rendered with MentalRay)

Note how the direct light image version does not accurately reproduce outdoor lightning. Only direct sun light is illuminating the car, producing too hard shadows (the interior of the car is completely dark). On the other hand, the indirect light version accurately reproduces outdoor lightning, as sun light bounces between surfaces illuminating surfaces that do not face the sun directly.
The next image shows the combination between radiosity and raytracing techniques, making possible to achieve inter-diffuse and specular reflections in the same scene.



Radiosity and raytracing techniques combination.
(Rendered with Mentalray)

Images rendered using global illumination algorithms often appear more photorealistic than images rendered using only direct illumination algorithms. However, such images are computationally more expensive and consequently much slower to generate. One common approach is to compute the global illumination of a scene and store that information with the geometry, i.e., radiosity. That stored data can then be used to generate images from different viewpoints for generating walkthroughs of a scene without having to go through expensive lighting calculations repeatedly.

References

[1] S. Pattanaik, Computational Methods for Global Illumination and Visualisation of Complex 3D Environments. PhD thesis, NCST Birla Institute of Technology & Science, Pilani, India, Feb. 1993.

[2] Whitted Turner. (1979) An improved illumination model for shaded display. Proceedings of the 6th annual conference on Computer graphics and interactive techniques

[3] Kajiya, J. T. The rendering equation. In Computer Graphics (SIGGRAPH '86 Proceedings) (Aug. 1986), D. C. Evans and R. J. Athay, Eds., vol. 20, pp. 143–150.

Raytracing

Ray tracing is a technique for generating an image by tracing the path of light through pixels in an image plane and simulating the effects of its encounters with virtual objects. The technique is capable of producing a very high degree of visual realism, usually higher than that of typical scanline rendering methods, but at a greater computational cost.

This makes ray tracing best suited for applications where the image can be rendered slowly ahead of time, such as in still images, and more poorly suited for real-time applications where speed is critical. [1] Ray tracing is capable of simulating a wide variety of optical effects, such as reflection, refraction and scattering effects.

Rendered Scene using raytracing for reflection and refraction effects.
(Rendered with MentalRay)

When a ray hits a surface, it could generate up to three new types of rays: reflection, refraction, and shadow. [2] A reflected ray continues on in the mirror-reflection direction from a shiny surface. It is then intersected with objects in the scene; the closest object it intersects is what will be seen in the reflection.

Refraction rays traveling through transparent material work similarly, with the addition that a refractive ray could be entering or exiting a material. To further avoid tracing all rays in a scene, a shadow ray is used to test if a surface is visible to a light. A ray hits a surface at some point. If the surface at this point faces a light, a ray (internally a line segment) is traced between this intersection point and the light.

If any opaque object is found in between the surface and the light, the surface is in shadow and so the light does not contribute to its shade. This new layer of ray calculation added more realism to ray traced images.

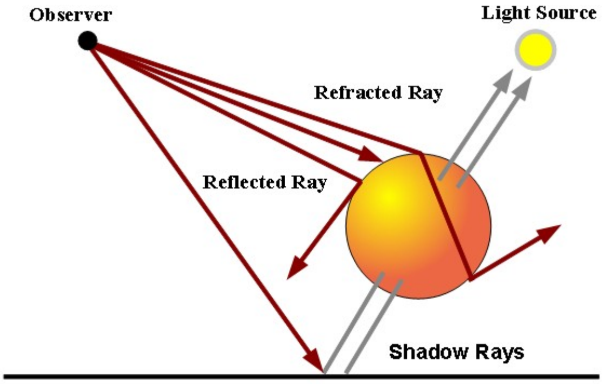


Diagram illustrating the ray tracing algorithm for rendering an image

Ray tracing's popularity stems from its basis in a realistic simulation of lighting over other rendering methods (such as scanline rendering or ray casting). Effects such as reflections and shadows, which are difficult to simulate using other algorithms, are a natural result of the ray tracing algorithm. The computational independence of each ray makes ray tracing suitable for parallelisation.

As noted in the first paragraph, a serious disadvantage of ray tracing is performance. Scanline algorithms and other algorithms use data coherence to share computations between pixels, while ray tracing normally starts the process anew, treating each eye ray separately. However, this separation offers other advantages, such as the ability to shoot more rays as needed to perform antialiasing and improve image quality where needed.

Although it does handle interreflection and optical effects such as refraction accurately, traditional ray tracing is also not necessarily photorealistic. True photorealism occurs when the rendering equation is closely approximated or fully implemented. [4] Implementing the rendering equation gives true photorealism, as the equation describes every physical effect of light flow. However, this is usually infeasible given the computing resources required. The realism of all rendering methods, then, must be evaluated as an approximation to the equation, and in the case of ray tracing, it is not necessarily the most realistic. Other methods, including photon mapping [see photon mapping section] are based upon ray tracing for certain parts of the algorithm, yet give far better results.

References

[1] Henrik Wann Jensen. "Global Illumination using Photon Maps", 1996.

[2] Tomas Nikodym (June 2010). "Raytracing algorithms for interactive applications". Czech Technical University, FEE.

[3] A. Chalmers, T. Davis, and E. Reinhard. Practical parallel rendering. AK Peters, Ltd., 2002.

[4] Whitted Turner. (1979) An improved illumination model for shaded display. Proceedings of the 6th annual conference on Computer graphics and interactive techniques

Radiosity

The radiosity method was one of the first widely used solutions to the global illumination problem. It was first introduced by Goral in 1984 [1] as a classical radiosity method being restricted to diffuse light transfer between surfaces, disregarding all other surface properties, such as specular reflection and refraction.

Unlike most other global illumination techniques, radiosity is view dependent, which means that although it can be costly to compute, once a scene's light distribution has been computed using the radiosity method, you can move your camera freely through a scene without having to recompute the illumination of the surfaces, making it particularly suitable for architectural walkthroughs and other scenarios with static environments.

Rendered Scene using radiosity.
(MentalRay v.3.5)

Many people in the computer graphics community tend to use the term radiosity when what they are actually talking about is global illumination or even just diffuse interreflection between objects in a scene. However, in research terms the radiosity method is a particular technique calculating light transfer using finite element methods, which makes it fundamentally different from methods such as path tracing [see path tracing section] and [photon mapping] though the results produced by these different methods can sometimes be similar.

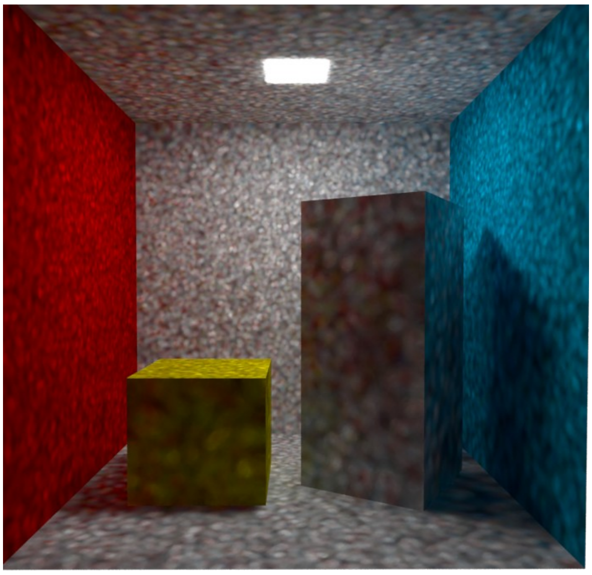
References

[1] Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaile. "Modeling the interaction of light between diffuse surfaces". SIGGRAPH '84: , New York, NY, USA, 1984. ACM Press.

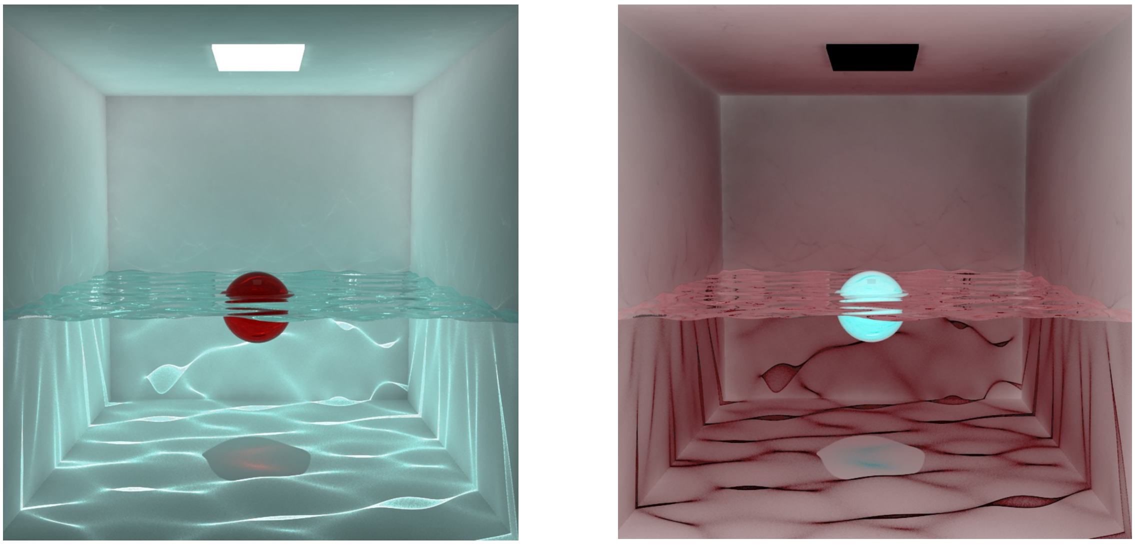
Photon Mapping

Photon mapping is a global illumination method to calculate the contribution of indirect illumination during rendering. It was first introduced by Henrik Wann Jensen [1] and simulates the way actual light disperses over a scene, from the light towards the eye, as opposed to regular ray tracing where the light rays are traced from the eye to the light source.

It is capable of emulating both light concentrations from refracted light, called caustics [see caustics chapter] and diffuse illumination from reflected light.
The algorithm works by first running a pre-processing step where the contribution of each light is stored in a *photon map*. Then this information is used during regular ray tracing. [2]

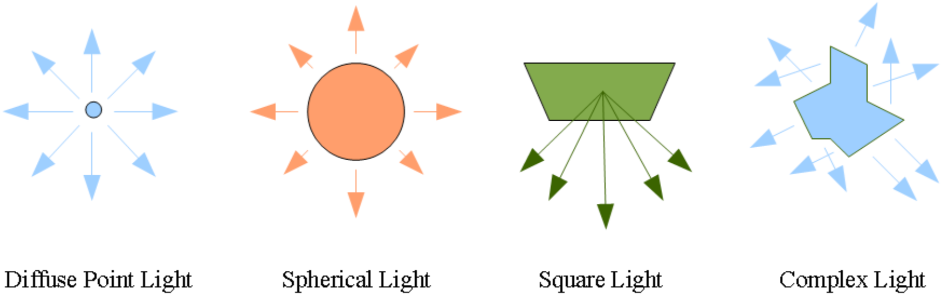


Photon map representation in a cornellbox scene.
Produced by not shooting enough photons.



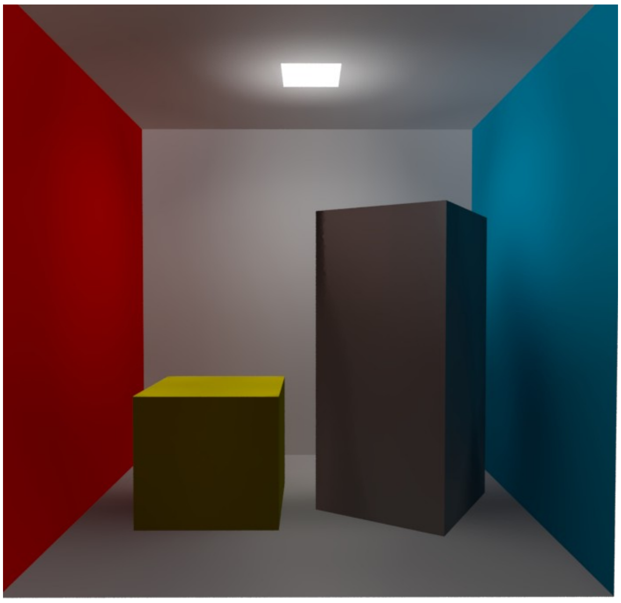
CornellBox showing accurate refractive and reflective caustics generated by photon mapping.
Inverted image to better show reflective caustics. (Rendered with MentalRay)

The photon map is generated by sending a swarm of light rays from each light into the scene. The light rays are reflected, refracted and attenuated by the objects in the scene. For each intersection, an entry about the lights direction and color, a "photon", is stored in the photon map.



Light photon direction given several light shapes.

Then during ray tracing [3] , the surrounding area around each intersection is checked in the photon map to gather information about the indirect illumination contribution at the intersection point.



CornellBox scene rendered using photon mapping.

References.

[1] Henrik Wann Jensen. "Global Illumination using Photon Maps" pages 21-30, 1996 Department of Graphical Communication The Technical University of Denmark

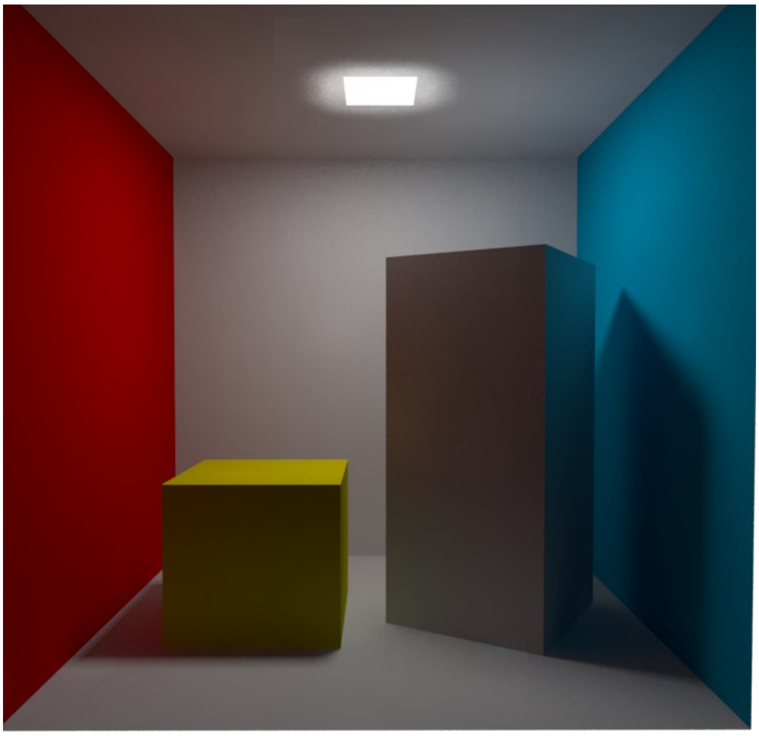
[2] Henrik Wann Jensen, "Realistic Image Synthesis Using Photon Mapping". A K Peters, Ltd., Massachusetts. July 2001

[3] Shirley, Peter. "Realistic Raytracing". A K Peters, Ltd. Massachusetts, 2000.

Path tracing

Path tracing is a rendering technique that attempts to simulate the physical behaviour of light by performing a generalisation of conventional raytracing [see raytracing section].
The image quality provided by path tracing is usually superior to that of images produced using conventional rendering methods at the cost of much greater computation requirements.

Path tracing naturally simulates many effects that have to be specifically added to raytracing or scanline methods, such as soft shadows, depth of field [see Camera chapter] , motion blur [see Camera chapter], caustics [see caustics chapter], ambient occlussion [see ao chapter] and indirect lighting. Implementation of a renderer including these effects is correspondingly simpler.



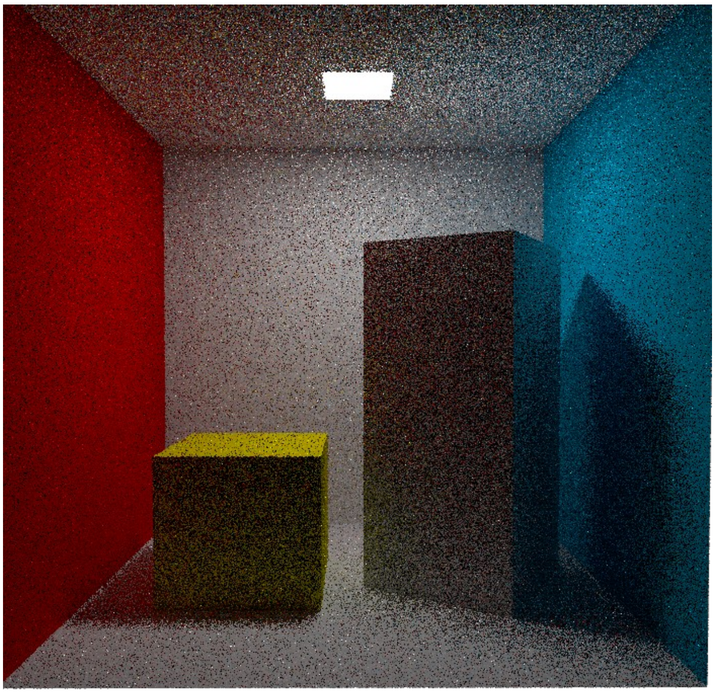
Rendered Scene using Pathtracing method.

In the real world, many small amounts of light are emitted from light sources, and travel in straight lines from object to object, changing colour and intensity, until they are absorbed by the observer.

This process is simulated by path tracing, except that the paths are traced backwards, from the camera to the light. The inefficiency arises in the random nature of the bounces from many surfaces, as it is usually quite unlikely that a path will intersect a light. As a result, most traced paths do not contribute to the final image.

That said, it is worth noting that Path tracing is not simply ray tracing with infinite recursion depth. In conventional ray tracing, lights are sampled directly when a diffuse surface is hit by a ray. In path tracing, a new ray is *r*andomly generated within the hemisphere of the object [1] and then traced until it hits a light, if that is the case. Therefore, this type of path can hit many diffuse surfaces before interacting with a light source.

Due to its accuracy and unbiased nature [see bias in render chapter], path tracing is sometimes used to generate reference images when testing the quality of other rendering algorithms. In order to get high quality images from path tracing, a large number of rays must be traced to avoid visible artifacts in the form of noise, increasing computational time and resources.



Visible artifacts produced by insufficient Pathtracing rays.

The famous render equation and its use in computer graphics was presented by James Kajiya in 1986 [2], containing what was probably the first description of the path tracing algorithm. A decade later, Lafortune [3] suggested many refinements, including bidirectional path tracing to accelerate the convergence of rendered images.

Bidirectional algorithms trace paths in both directions. In the forward direction, rays are traced from light sources until they are too faint to be seen or strike the camera. In the reverse direction , rays are traced from the camera until they strike a light or too many bounces (depth atribute) have occurred. This approach normally results in an image that converges much more quickly than using only conventional one direction pathtracing.

These methods generate one subpath starting at a light source and another starting at the lens, then they consider all the paths obtained by joining every prefix of one subpath to every suffix of the other. This leads to a family of different importance sampling techniques for paths, which are then combined to minimize variance. [4]

References

[1] Purcell, T J; Buck, I; Mark, W; and Hanrahan, P, "Ray Tracing on Programmable Graphics Hardware", Proc. SIGGRAPH 2002

[2] Kajiya, J T, The rendering equation, Proceedings of the 13th annual conference on Computer graphics and interactive techniques, ACM, 1986

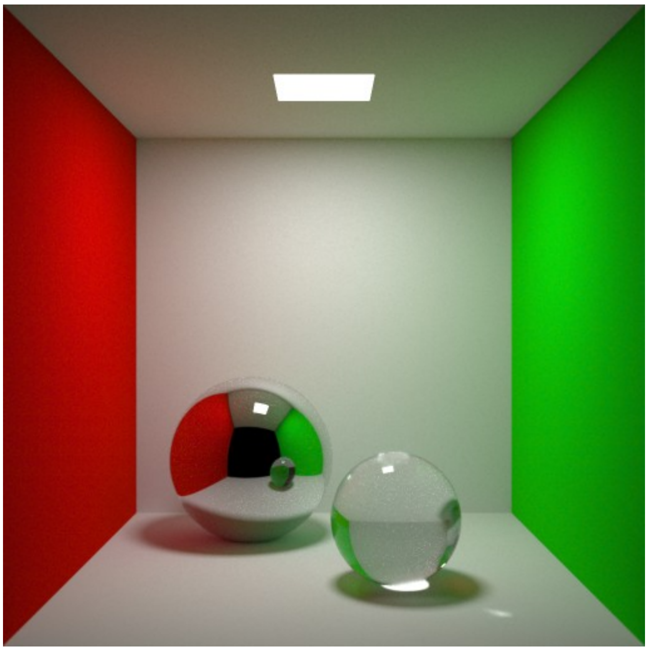
[3] Lafortune, E, ,Mathematical Models and Monte Carlo Algorithms for Physically Based Rendering (PhD thesis), 1996

[4] Veach, E., and Guibas, L. J. Metropolis light transport. In SIGGRAPH'97 (August 1997), p. 65–76.

Metropolis light transport (MLT)

The Metropolis light transport (MLT) is a method for computing physically accurate behaviour of light introduced by Veach and Guibas [1], who described an application of a variant of the Monte Carlo method [see MonteCarlo section] called the Metropolis-Hastings algorithm to the rendering equation for generating images from detailed physical descriptions of three dimensional scenes.

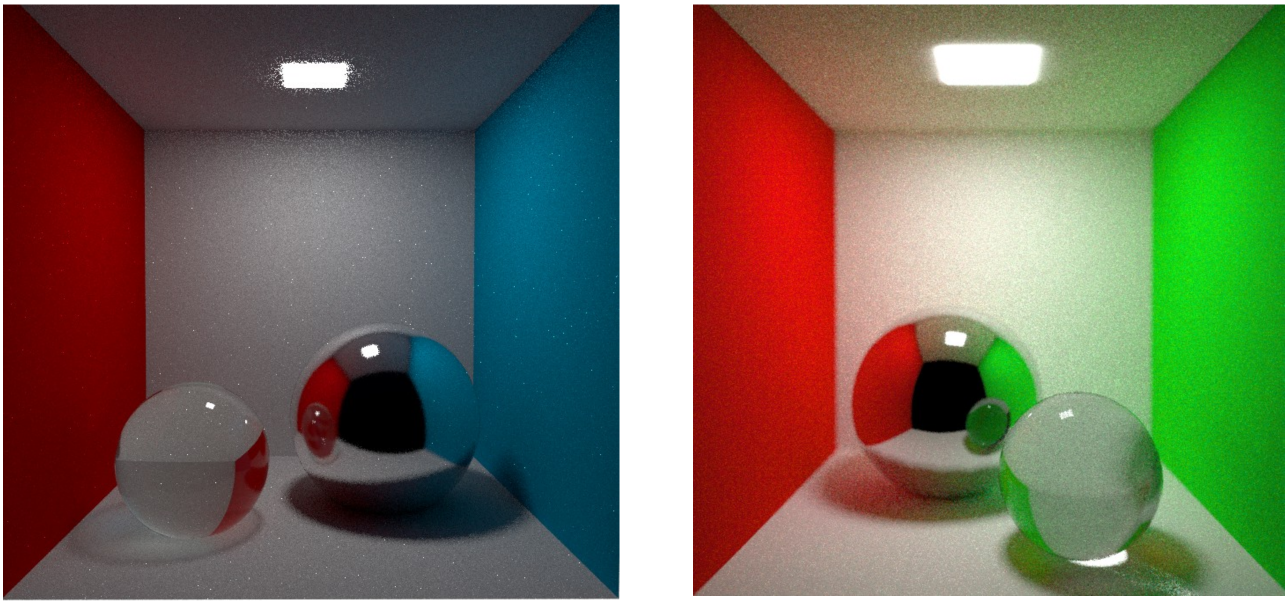
The procedure constructs paths from the eye to a light source using bidirectional path tracing [see path tracing section], then constructs slight modifications to the path. Some careful statistical calculation (the Metropolis algorithm) is used to compute the appropriate distribution of brightness over the image.



Rendered Scene using Metropolis Light Transport
(Maxwell Render v2.5)

This procedure has the advantage, relative to bidirectional path tracing, that once a path has been found from light to eye, the algorithm can then explore nearby paths; thus difficult-to-find light paths can be explored more thoroughly with the same number of simulated photons.
In short, the algorithm generates a path and stores the path's 'nodes' in a list. It can then modify the path by adding extra nodes and creating a new light path. While creating this new path, the algorithm decides how many new 'nodes' to add and whether or not these new nodes will actually create a new path.

Metropolis Light Transport is an unbiased method that, in some cases , converges to a solution of the rendering equation quicker than other unbiased algorithms, path tracing and bidirectional path tracing.



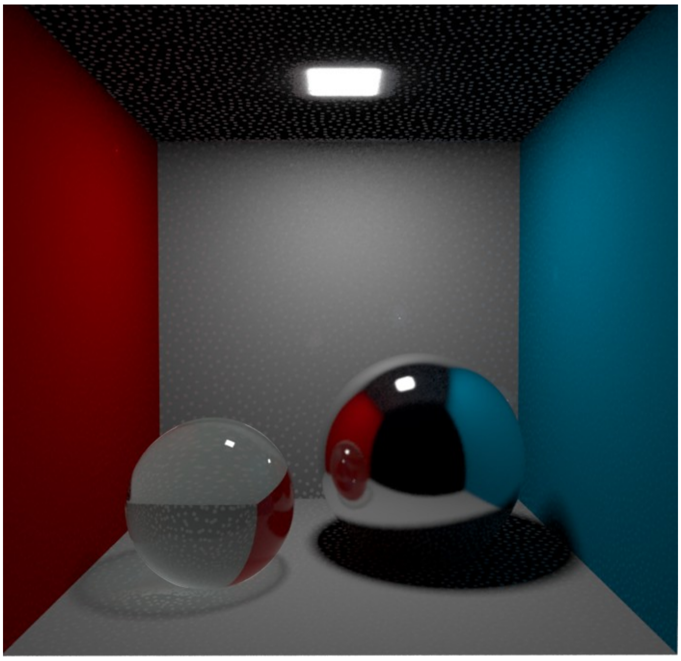
Convergence differences between MLT and traditional Pathtracing
Computation stopped at 1m15s

(Maxwell Render 2.5, Vray 1.5 sp1 respectively)

Irradiance Caching.

Irradiance caching is a ray tracing-based technique for computing global illumination on diffuse surfaces first introduced by Greg Ward in his Siggraph '88 paper [1]. Specifically, it addresses the computation of indirect illumination bouncing off one diffuse object onto another.
The sole purpose of irradiance caching is to make this computation reasonably fast. The main idea is to perform the indirect illumination sampling only at a selected set of locations in the scene, store the results in a cache, and reuse the cached value at other points through fast interpolation.

The irradiance caching algorithm has been successfully used to accelerate global illumination computation in the Radiance lighting simulation system. Its widespread use had to wait until computers became fast enough to consider global illumination in film production rendering. Since then, its use is ubiquitous.



Points indicating Irradiance Caches in a CornellBox Scene.

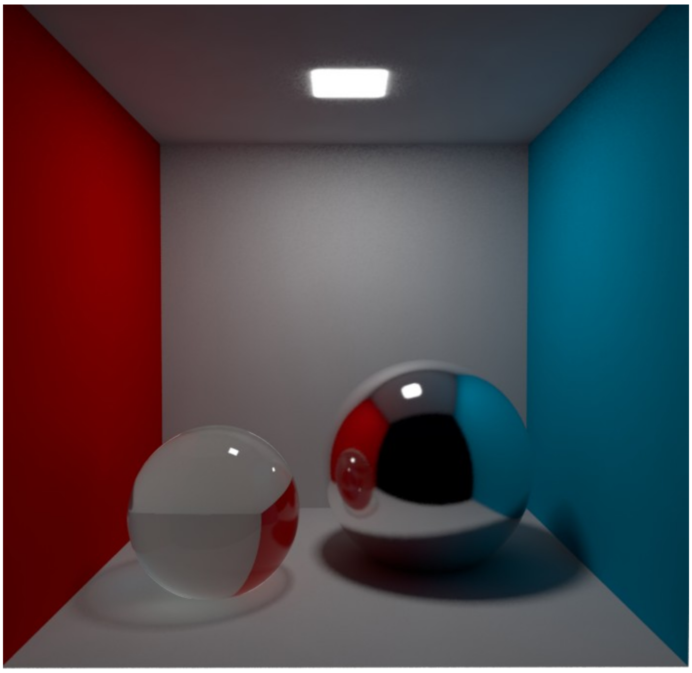
Solving the rendering equation with even just one bounce of indirect lighting can take a long time. The majority of time spent rendering a frame is in estimating the lighting integral.[2]
For example, rendering a single bounce of indirect lighting at 720p resolution with 256 sample rays for a Monte Carlo estimator requires about 237 million rays to be cast.

This doesn't even include the rays needed for sampling the lights for direct lighting, so in practice, the total will be even higher.

One interesting observation made by Greg Ward is that contrary to direct lighting, where shadows and lights can cause harsh changes, the indirect lighting on a surface tends to vary relatively slowly.

One way to picture why this is, is to imagine the computing average color from the what you can see from each of your eyes. Even though each eye has a slightly different view on the world, the images they see are nearly similar, and so the average color is also nearly the same.

In the image below, a scene has been rendered using Irradiance Caching. Note how the lighting varies in a very smooth fashion.



Rendered scene using Irradiance Caching.

(Vray render engine.)

The basic concept for irradiance caching is the following. For each point on a surface at which you want to evaluate irradiance, if the cache contains any valid entries then interpolate between them. Otherwise, calculate a new irradiance entry, and add it to the cache.

A cache entry contains the position and normal for the point on the surface where the irradiance was evaluated as well as the irradiance value itself. One important additional piece of information that the cache requires is the range over which the entry is considered potentially valid. This range could be calculated in a number of ways, but the most common one is to use the harmonic mean of the hit distance of the rays used for the estimator.

References

[1] Greg Ward, Francis Rubinstein and Robert Clear. "A Ray Tracing Solution to Diffuse Interreflection." Siggraph '88,

[2] Practical Global Illumination with Irradiance Caching 2009, 148 pages,
Jaroslav Krivanek Cornell University and Charles University, Prague
Pascal Gautron Thomson Corporate Research, France

Monte Carlo Ray Tracing

Recent advances in algorithms and compute power has made Monte Carlo ray tracing a widely used method for simulating global illumination. [1] This is a significant change from just a few years back when the (finite element) radiosity [see radiosity section] method was the prefered algorithm for most graphics researchers.

These raytracing algorithms generate random walks through the scene, where the pixel intensities are estimated by making the average of their conributions. In order to model all kind of reflections and refractions in the random walks, hybrid methods are used that try to exploit the advantageous properties of radiosity and ray tracing. [2]

Monte Carlo ray tracing has several advantages over finite element methods. [3]

- Geometry can be procedural
- No tessellation is necessary
- It is not necessary to precompute a representation for the solution
- Geometry can be duplicated using instancing
- Any type of BRDF can be handled
- Specular reflections (on any shape) are easy
- Memory consumption is low
- The accuracy is controlled at the pixel/image level

In addition one might add that Monte Carlo ray tracing methods can be very easy to implement. A basic path tracing algorithm which has all of the above advantages is a relatively straightforward extension to ray tracing. The main problem with MonteCarlo ray tracing is variance seen as noise in the rendered images. This noise can be eliminated by using more samples. [4]
Unfortunately the convergence of Monte Carlo methods is quite slow, and a large number of samples can be necessary to reduce the variance to an acceptable level.

References

[1] George S. Fishman. Monte Carlo: concepts, algorithms, and applications. Springer Verlag, New York, NY, 1996.

[2] Balázs Csébfalvi, "A Review of Monte Carlo Ray Tracing Methods." Department of Process Control, Technical University of Budapest,

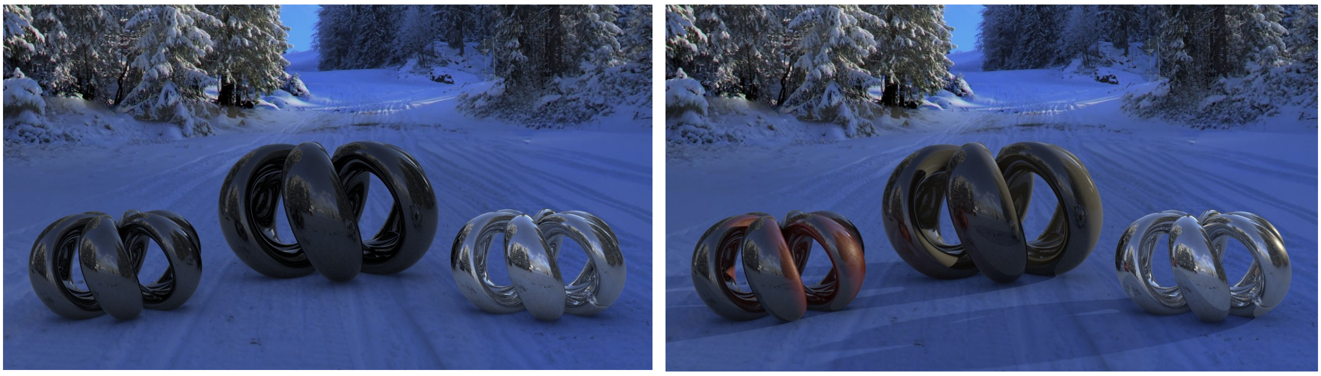
[3] Henrik Wann Jensen. Realistic Image Synthesis using Photon Mapping. AK Peters, 2001.

[4] Michael McCool. Anisotropic diffusion for monte carlo noise reduction. ACM Transactions on Graphics, pages 171–194, April 1999.

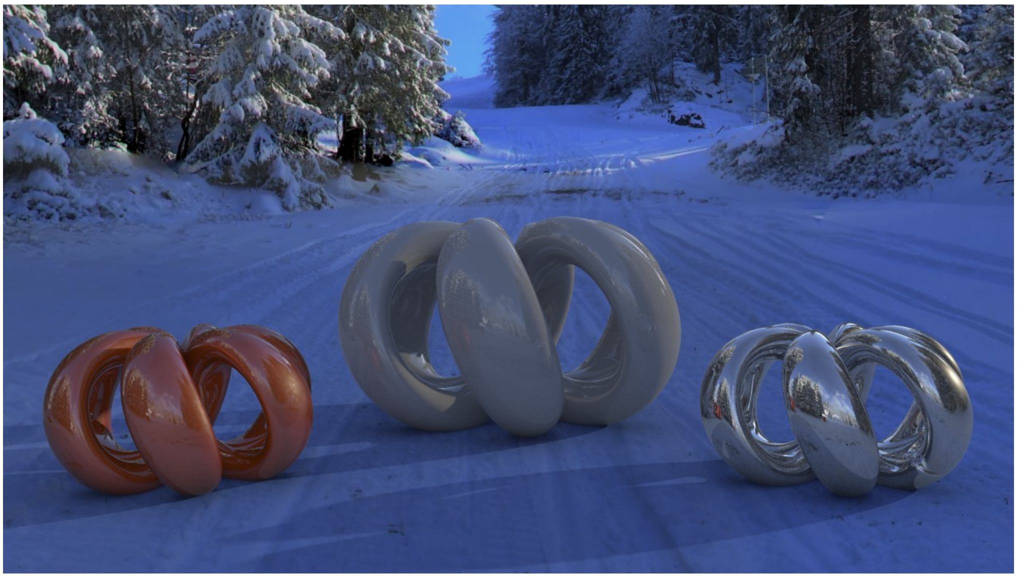
Image Based Lighting (IBL).

Image-based lighting (IBL) is the process of illuminating scenes and objects (real or synthetic) with images of light from the real world. It evolved from the reflection-mapping technique [1], [2] in which we use panoramic images as texture maps on computer graphics models to show shiny objects reflecting real and synthetic environments. IBL is analogous to image-based modeling, in which we derive a 3D scene's geometric structure from images, and to image-based rendering, in which we produce the rendered appearance of a scene from its appearance in images. When used effectively, IBL can produce realistic rendered appearances of objects and can be an effective tool for integrating computer graphics objects into real scenes. [4]

+ Traditional Directonal Light



+ IBL lighting.



Traditional Lighting vs IBL lighting.
(Rendered in MentalRay for Maya)

IBL lets us integrate computer-generated models into real-world environments according to the principles of global illumination.

It requires a few special practices for us to apply it, including taking omnidirectional photographs, recording images in high dynamic range, and including measurements of incident illumination as sources of illumination in computer-generated scenes.

The basic steps in IBL are:

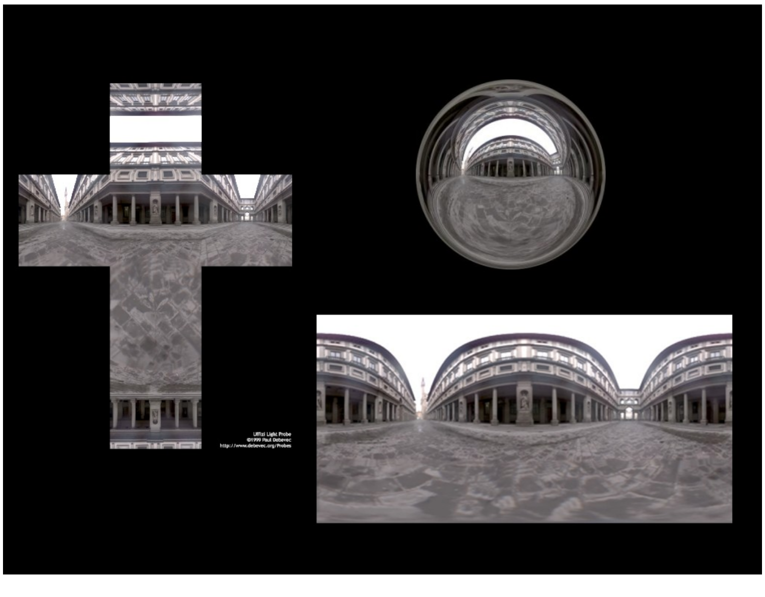
1. **Capturing real-world illumination as an omnidirectional, high dynamic range image;**

High Dynamic Range Imaging is a method to digitally capture and edit all light in a scene. [5] HDRI emerged from the movie industry, and it is now a mature technology available to everyone.

Depending on what application you are using, the names of the shapes of the HDR will change. Most 3D programs need an image that looks like an unwrapped world map. This file can be called “Equirectangular”, “Spherical”, “LatLong”, and also a “Latitude Longitude” file. These are called different names, but they are all exactly the same.

Another format is Vertical Cross, Horizontal Cross, and Cubic HDRs… these are all just six 90-degree views of a scene or panorama. They just happened to be place together in a certain shape (vertical or horizontal cross) or saved out as six separate cube faces as six different files. Retouching is pretty easy with these as long as you do not try to retouch any of the edge pixels. If you need to do some cloning or color correction across the seams, then convert the image to a spherical image and work on that. Then convert back to a cubic format if needed.

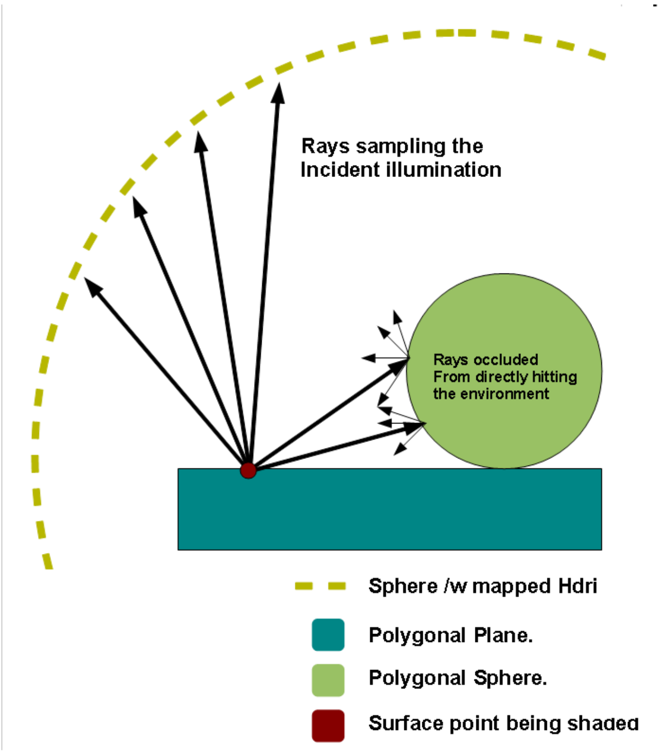
The last format is often called “Light Probe”. Light Probes are full 360x180-degree spherical (or LatLong) HDR files that have been converted into a ball shape. Visually, it appears that they are chrome balls, but in reality they have a mathematically much simpler distortion and they may or may not have been created and captured that way.
A few 3D programs directly support Light Probes (ball-looking) images, but almost all support LatLong shaped HDR images. When you make an HDR from a series of pictures of a chrome ball, this too, is considered a “light probe”. In fact, any fully immersive HDR panorama can be considered a light probe, no matter what unwrapping format it is in. They are also sometimes called "Light Maps" as well.



Light Probe, Vertical Cross, and Latlong hdri Formats.
Images freely available in Paul Debevec's website.

2. Mapping the illumination onto a representation of the environment;

Usually a big sphere is used surrounding all the scene. The hdri image is then mapped to that sphere that roughly represents the environment. Please see the image below for an easy representation of Ibl.



Incident surface illumination from IBL.

3. Placing all 3D objects inside the environment.

4. simulating the light from the environment illuminating the computer graphics object.

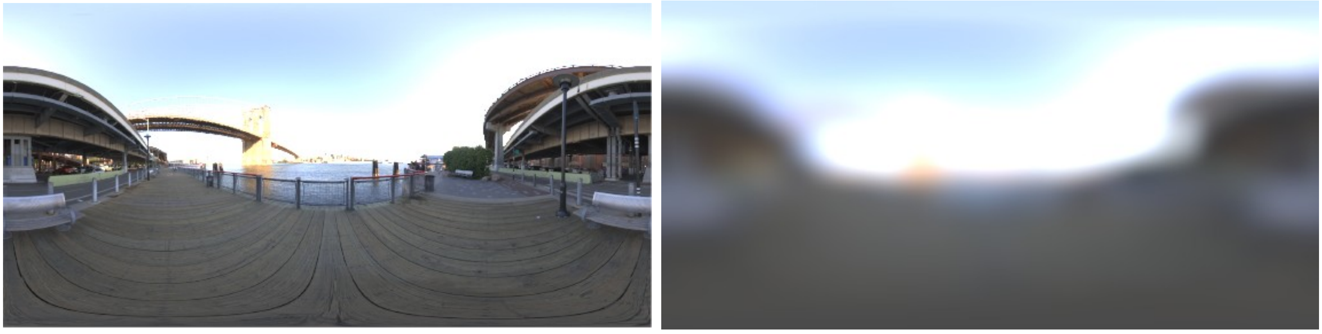
The lighting characteristics of the surrounding surface are then taken into account when rendering the scene, using the modeling techniques of Global Illumination [See Global Illumination Section]. This is in contrast to light sources such as a computer-simulated sun or light bulb, which are more localized.

Lighting a 3d scene with panoramic HDR imagery revolutionized computer graphics. [6]

Never before had it been so easy to replicate real lighting situations. Every photorealistic render engine includes a Global Illumination (GI) method that can be combined with HDRIs to create photorealistic renderings.

However, a common pitfall is that GI takes longer to render, especially when using high-res HDRIs. It takes ridiculously high quality settings to get rid of sampling noise or flickering.

That's why, in practice it’s much more efficient to use a blurred low-res image for diffuse lighting, and a high-res image for reflections only.



Reflection map vs Diffuse map Example
(Refl map: Real resolution: 2048x1024 32bit)
(Diff map: Real resolution: 512x256 32bit)

Another common problem is to capture the sun. It's just too bright, which makes it notoriously hard to capture. It's even harder to render with, because it causes the worst render artifacts. Much more convenient is a regular 3d directional light, allowing more control over intensity and shadow. Such advanced lighting rigs result in better renderings in less render time, but they can be complex to set up. Aside from preprocessing the HDR images the right way, you have to set up your renderer properly, becoming a tedious work.

References

[1] J.F. Blinn, “Texture and Reflection in Computer Generated Images,” Comm. ACM, vol. 19, no. 10, Oct. 1976, pp. 542-547.

[2] G.S. Miller and C.R. Hoffman, “Illumination and Reflection Maps: Simulated Objects in Simulated and Real Environments,” Proc. Siggraph 84, Course Notes for Advanced Computer Graphics Animation, ACM Press, New York, 1984.

[3] Paul Debevec, “Rendering Synthetic Objects Into Real Scenes: Bridging Traditional and Image-Based Graphics with Global Illumination and High Dynamic Range Photography,” Computer Graphics (Proc. Siggraph 98), ACM Press, New York, 1998, pp. 189-198.

[4] Paul Debevec, "Image-Based Lighting" . USC Institute for Creative Technologies. CGA 2002.

[5] Kirt Witte, "Hdri Frequently Asked Questions", Savannah College of Art and Design (SCAD).

[6] Christian Bloch, "Smart IBL overview", HDR Labs, 2007-2010.

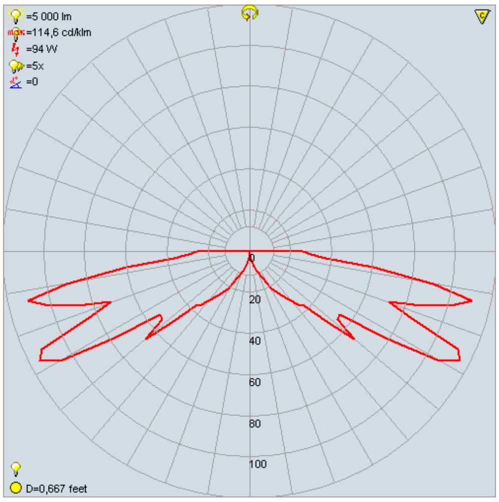
IES Profiles

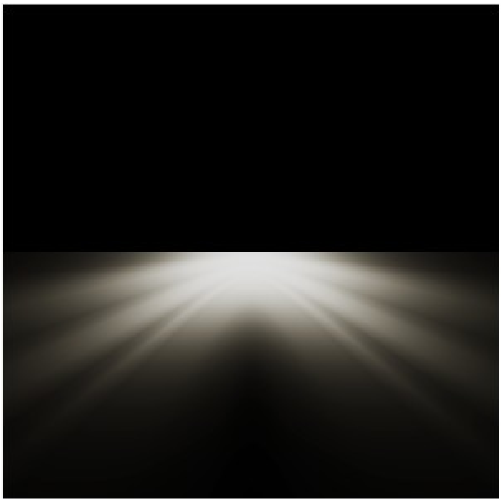
The **Illuminating Engineering Society of North America** (**IES**) is a non-profit learned society that was founded in New York City on January 10, 1906. The IES's stated mission is to improve the lighted environment by bringing together those with lighting knowledge and by translating that knowledge into actions that benefit the public. [1] Members of the IES are regarded as the top professionals in their industry and are globally respected for their knowledge. [2]

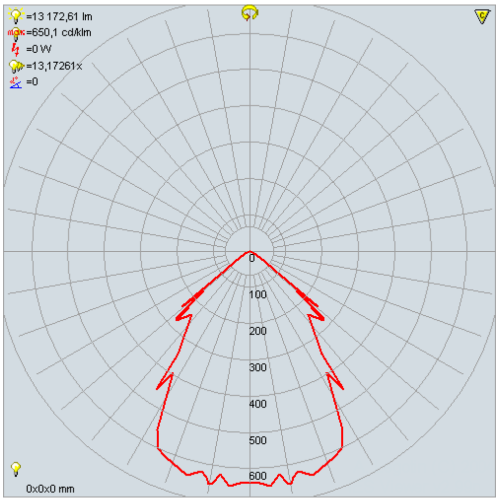
IE society developed a standard file format that holds information on the distribution of light intensity from a light source. This format was originally the standard of IESNA, whose most recent version is IES LM-6302. It is worth noting that Eulumdat data format is the equivalent to IES profiles in Europe, and the following lines also apply to the european standard.

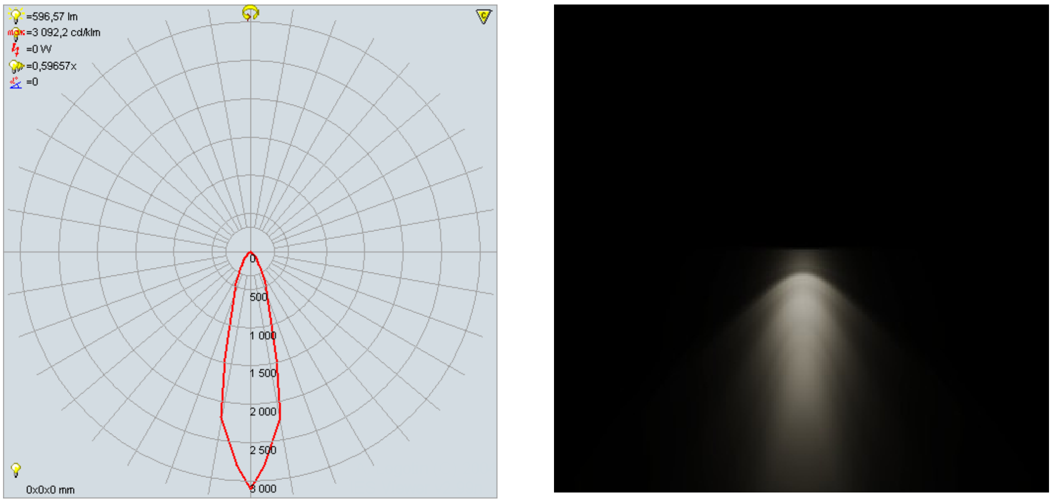
You can think of it as a digital profile of a real world light. In nowaday's render engines it can be used for creating lights with shapes and a physically accurate form.

IES standard file format was created for the electronic transfer of photometric data over the web. It has been widely used by many lighting manufacturers and is one of the industry standards in photometric data distribution. An IES file is basically the measurement of distribution of light (intensity) stored in ASCII format.









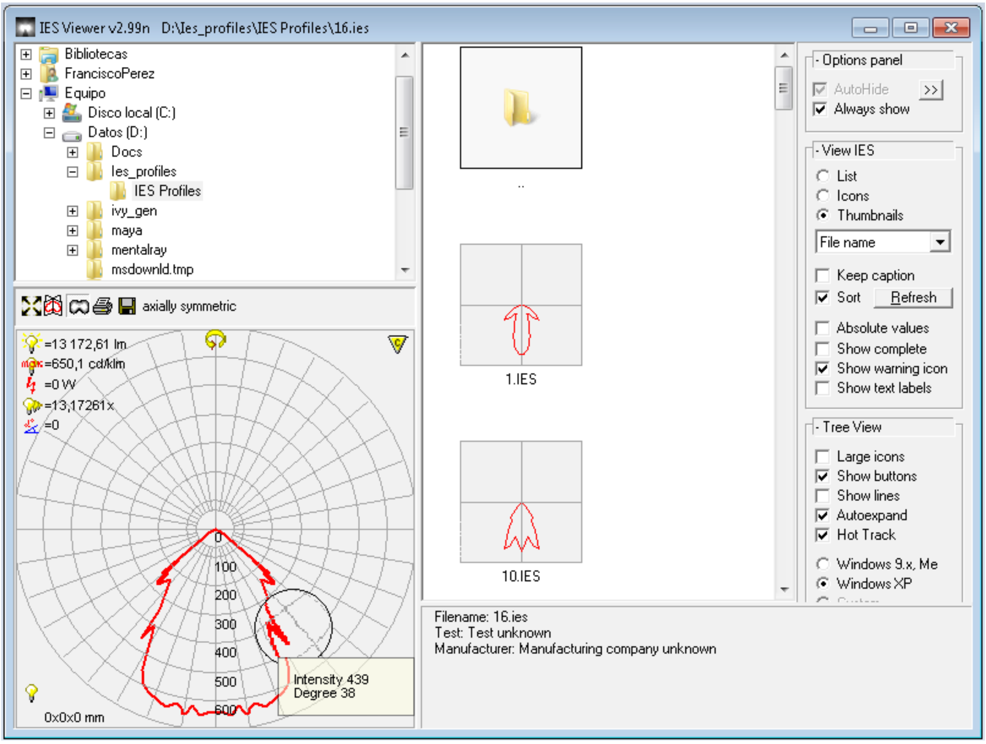
Examples of Ies profiles and its rendered appearance.

IES light files are created by many major lighting manufacturers and can be downloaded freely from their sites. One of such lighting manufacturer is Lithonia Lighting (www.lithonia.com) which has an extensive library of IES files in different categories. Other manufacturers like General Electric Lighting are worth mentioning.

```
IESNA:LM-63-1995
[TEST] BE1667
[DATE] 12-FEB-96
[MANUFAC] BEGA
[LUMCAT] 6339
[LUMINAIRE]  SURFACE MOUNTED WALL LUMINAIRE
[LAMP]  (1)  60W A-19 INC
TILT=NONE
 1   890   .89   73   1   1   2  -.1   0   0
 1   1   60
 0   2.5   5   7.5   10   12.5   15   17.5   20   22.5   25   27.5   30
32.5   35   37.5   40   42.5   45   47.5   50   52.5   55   57.5   60
62.5   65   67.5   70   72.5   75   77.5   80   82.5   85   87.5   90
92.5   95   97.5   100   102.5   105   107.5   110   112.5   115   117.5
120   122.5   125   127.5   130   132.5   135   137.5   140   142.5
145   147.5   150   152.5   155   157.5   160   162.5   165   167.5
170   172.5   175   177.5   180
 0
 178.4   176.7   170.8   160.9   147   133   114.6   96.13   80.57
67.13   56.54   47.61   40.87   36.97   36.86   39.76   45.47   54.35
66.39   75.84   79.96   73.53   62.87   52.35   44.33   39.14   36.31
33.87   29.1   22.59   15.67   9.13   5.317   2.654   1.382   .3416
.2   .09695   .2025   .175   .1518   .1455   .1328   .1307   .1139
.1097   .097   .1075   .1076   .07175   .1243   .1073   .1328   .1349
.1391   .1307   .1412   .1475   .156   .1813   .1876   .2087   .2466
.2466   .2782   .2993   .3035   .3035   .3035   .2993   .2993   .3035
.3035
```

ASCII content of a IES profile.

The shape of these profiles can be easily previewed making use of some programs like PhotometricViewer made by Andrey Legotin [4]. It allows you to view all the information associated with the light profiles and there's even a really handy render feature so you can see exactly how the light looks without having to setup a scene in your 3D package just for that .



=13 172,61 lm
=650,1 cd/klm
=0 W
=13,17261x
=0

Intensity 439
Degree 38

☑ AutoHide
☑ Always show

○ List
○ Icons
● Thumbnails

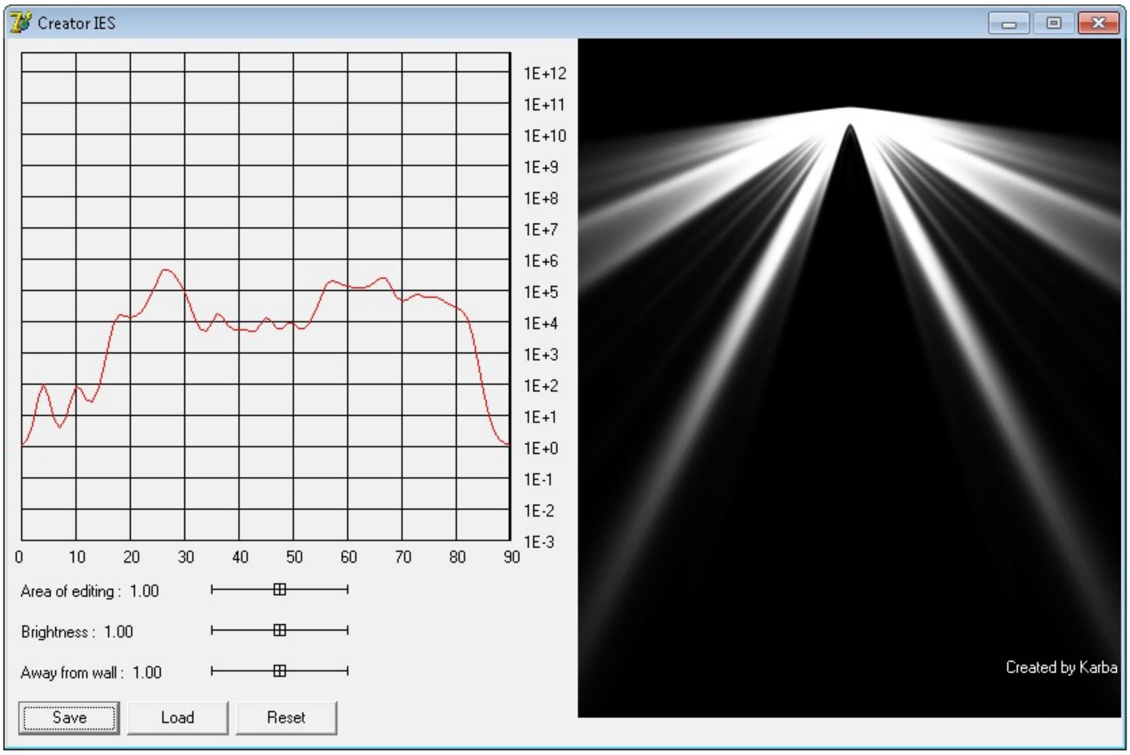
☐ Keep caption
☑ Sort
☐ Absolute values
☐ Show complete
☑ Show warning icon
☐ Show text labels

☐ Large icons
☑ Show buttons
☐ Show lines
☑ Autoexpand
☑ Hot Track
○ Windows 9.x, Me
● Windows XP

Filename: 16.ies
Test: Test unknown
Manufacturer: Manufacturing company unknown

Ies Viewer by Andrey Legotin Interface.

Other types of lightning software available allows you to interactively create a lightning profile, preview its rendered appereance, and save it as an IES/Elumdat profile. That brings the possibility to create your own light shapes without the need of scientifically measure real-world light sources.

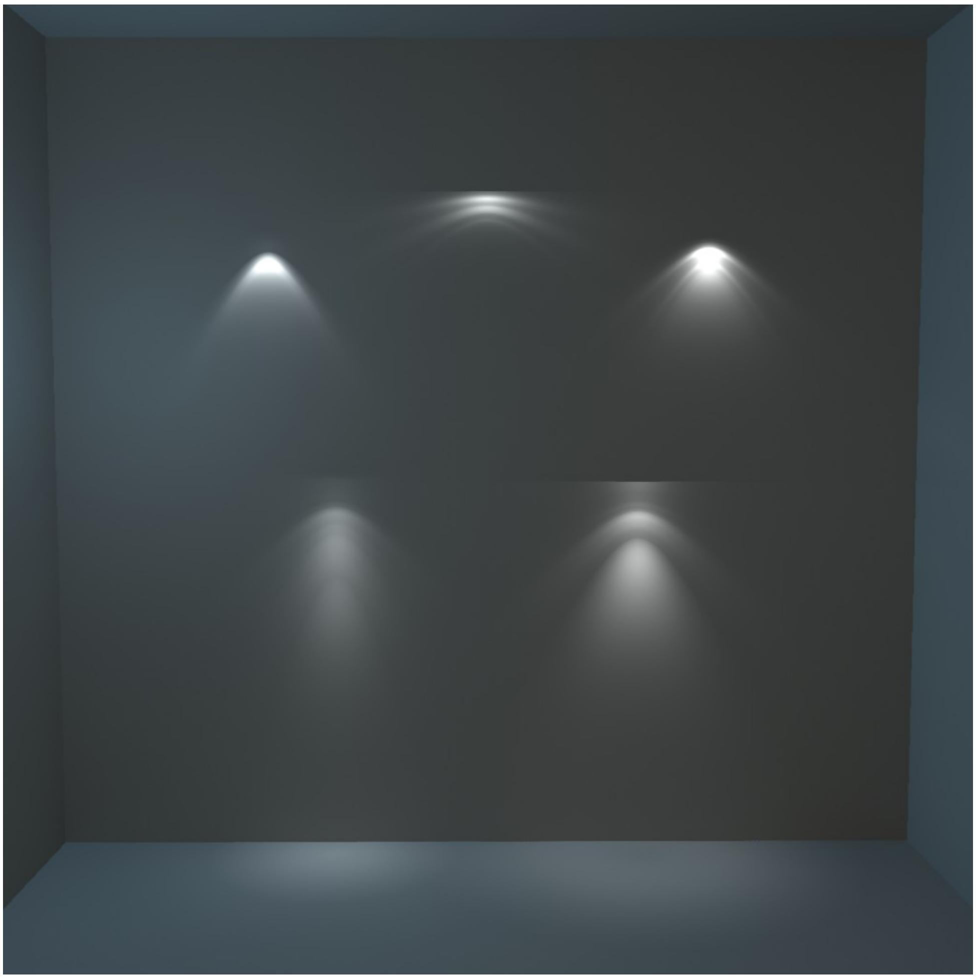


Area of editing : 1.00
Brightness : 1.00
Away from wall : 1.00

Light Profile generation software.

Custom shaped lights are not easy to achieve in nowaday's photorealistic renderers without the use of IES profiles. That is the main reason why light profiles are an important feature being a great way to add realism to your scene. [3]

The following image shows a scene rendered with MentalRay for Maya render engine used in this study showing the capabilities of IES profiles.



IES profiles applied to five point lights in a scene.
Rendered wih MentalRay (Rendertime 8sec).

References

[1] Illuminating Engineering Society , www.ies.org.
[2]Williams, Bill (2005). "A History of Light and Lighting". *Retrieved 2007-07-14.*
[3] Indigo Render Documentation, IES section. 2004-2011 Glare Technologies Limited
[4] IESViewer Photometric viewer, http://www.photometricviewer.com. 2001-2011 Andrey Legotin.

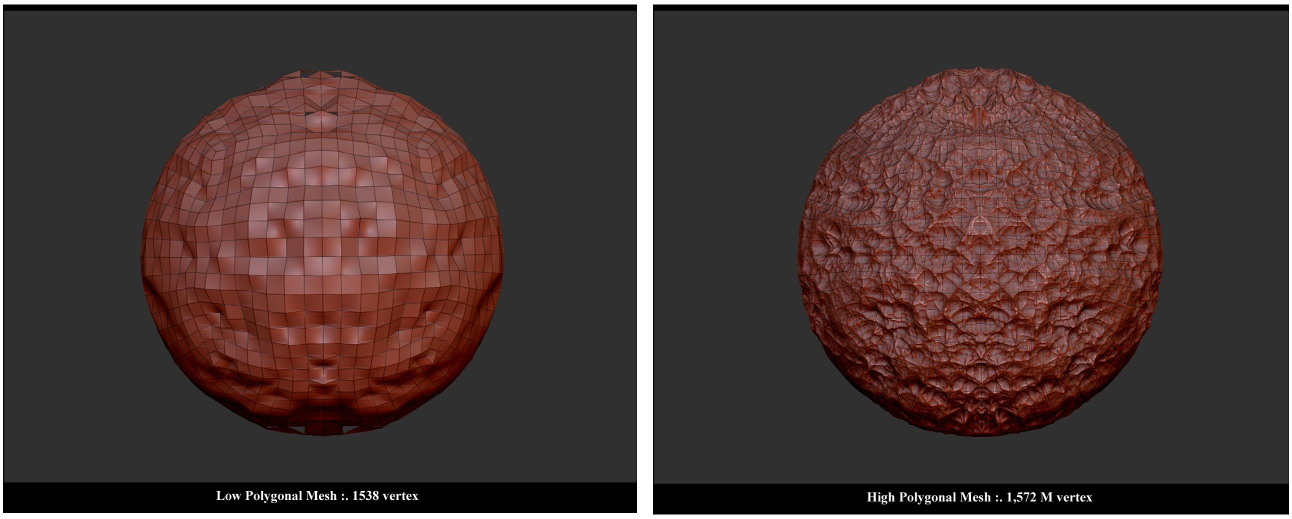
Normal Mapping.

In 1996, Krishnamurthy and Levoy [1] introduced the idea of extracting geometric details from a high polygon model to create a displacement map [see displacement section] applied to NURBS surfaces. This was the beginning of what is nowadays known as normal mapping.

Two years later, Cohen [2] and Cignoni [3] wrote two studies that went beyond the idea of Krishnamurthy and Levoy and normal maps appeared as a way of transfering details from high polygonal triangle meshes to low polygonal simplified surfaces.
Cohen showed an algorithm to build a normal map based on the details of the high polygonal surface, but considering how the simplified mesh was created as well, in order to correctly apply the normal map over the low polygonal surface.

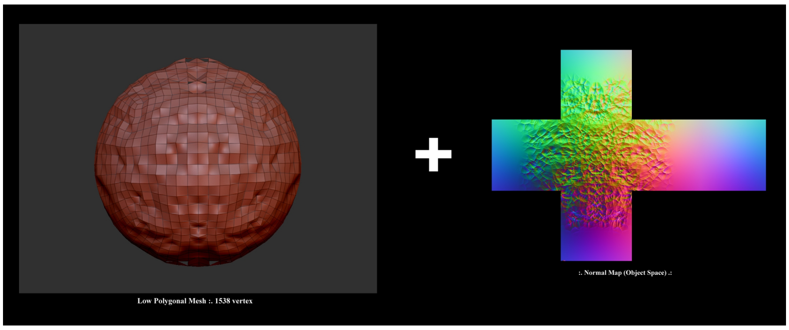
However, Cignoni presented a simpler approach that did not take into consideration how the low polygonal model is created, so the recreation of the details lost by the low polygonal mesh is not dependent on how it is created.

That said, normal mapping may be described as a rendering technique used to add details to a given surface without increasing its polygon count like displacement maps do [see displacement section]. The main use of this technique is to re-detail simplified meshes, rendering a highly detailed model from a low polygonal mesh, applying the normal map obtained from a high polygonal object previously built. The process of transferring normals from the high resolution model to the low polygonal model is often called baking.

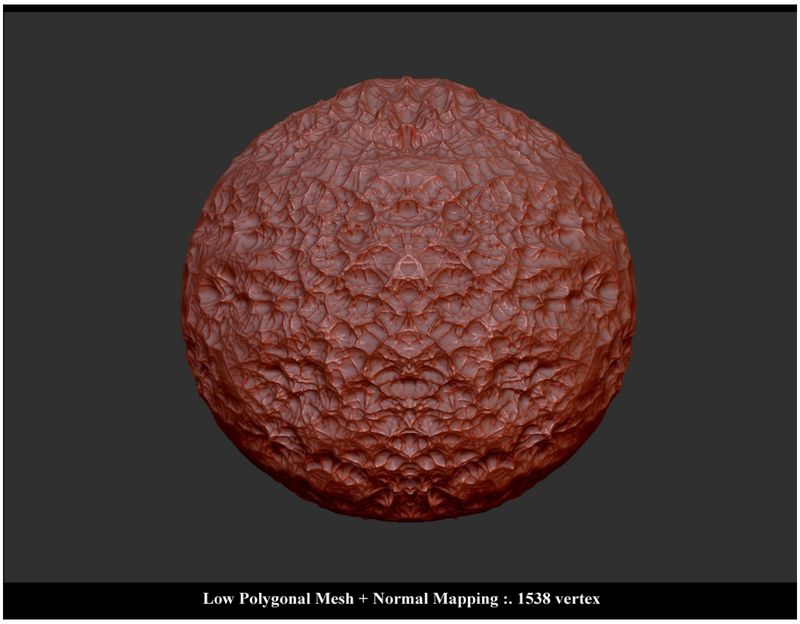

Low Polygonal Mesh :. 1538 vertex

High Polygonal Mesh :. 1,572 M vertex

Low polygonal mesh vs high polygonal mesh


Low Polygonal Mesh :. 1538 vertex
:. Normal Map (Object Space) .:

Low Polygonal surface and baked normal map



Low polygonal mesh with normal map applied

Note how normal mapping adds detail to the low polygonal mesh without increasing its 1538 vertex, giving the impression to be the same as the high polygonal sphere previously built with 1,572M vertex.

Another use of normal mapping is to simulate grooves and imperfections on a given surface. This technique is usually defined as an application of bump mapping [see bump mapping section] , but it is strongly recommended when the impression of very strong bumps or grooves is needed.

Usually in normal mapping an RGB image is used as the normal map, and each component corresponds to each coordinate of the surface normal from an hypothetical more detailed version of the object.
While bump mapping perturbs the existing surface normal of a model given a bump map (one channel image) [see bump mapping section] , normal mapping replaces the normal entirely depending on the normal map (rgb image), specifying its direction [4].

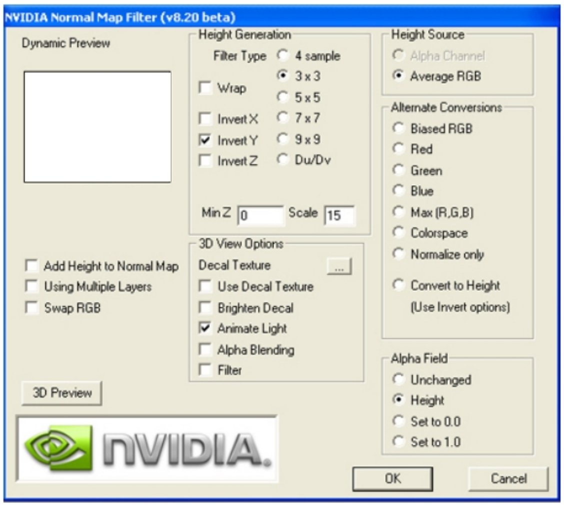
One advantage of using normal maps is that this technique is less expensive to compute than bump mapping, and normal maps can be generated more easily, for example by measurements [5], or surface simplification [6].

However, normal maps are attached to the underlying geometry, while bump maps can be applied to any surface. [7]

Some image software tools provide the utility to convert from bump map to normal map and vice-versa.

Nvidia's normal filter plug-in is the most commonly used and a good example of this kind of software tool that creates normal maps from bump textures and can be downloaded from http://developer.nvidia.com/object/photoshop_dds_plugins.html

The filter UI provides a powerful 3D preview and a variety of filtering options. For MIP-mapped normal maps, the NormalMap Filter should be used for preview only, and then the grayscale map exported via the DDS plugin with "Normal Map Settings..." enabled. This will create a normal map with maximal detail and fidelity for all texture levels, without undesirable filter aliasing.


NVIDIA Normal Map Filter (v8.20 beta)

Dynamic Preview

☐ Add Height to Normal Map
☐ Using Multiple Layers
☐ Swap RGB

Height Generation
Filter Type: ○ 4 sample ● 3 x 3 ○ 5 x 5 ○ 7 x 7 ○ 9 x 9 ○ Du/Dv
☐ Wrap
☐ Invert X
☑ Invert Y
☐ Invert Z
Min Z 0
Scale 15

3D View Options
Decal Texture ...
☐ Use Decal Texture
☐ Brighten Decal
☑ Animate Light
☐ Alpha Blending
☐ Filter

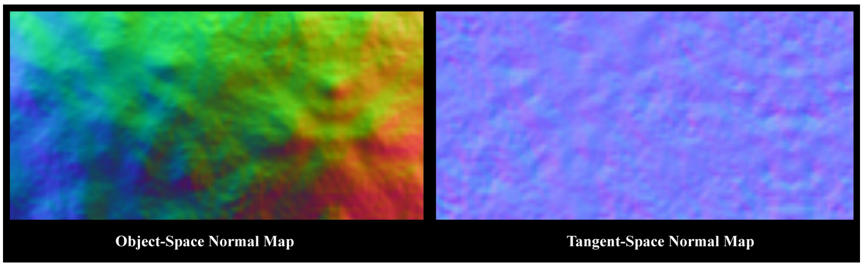
Height Source
○ Alpha Channel
● Average RGB

Alternate Conversions
○ Biased RGB
○ Red
○ Green
○ Blue
○ Max (R,G,B)
○ Colorspace
○ Normalize only
○ Convert to Height (Use Invert options)

Alpha Field
○ Unchanged
● Height
○ Set to 0.0
○ Set to 1.0

Nvidia's normal filter plug-in

Depending on how the normal map is created, it can be found in two varieties: object-space or world-space and tangent-space normal mapping. They differ in coordinate systems in which the normals are measured and stored. World-space is basically the same as object-space, except it requires the model to remain in its original orientation, neither rotating nor deforming. For that cause, world-space is rarely used.



Object-Space normal map vs Tangent-Space normal map

Note that the tangent-space normal map shows predominantly blue colors.
Objects with this kind of map applied could rotate and deform properly mantaining the details added by the normal map. It is then a good map to add when deformations are required, like rendering organic meshes. That said, one single map could be reused on differently shaped meshes.

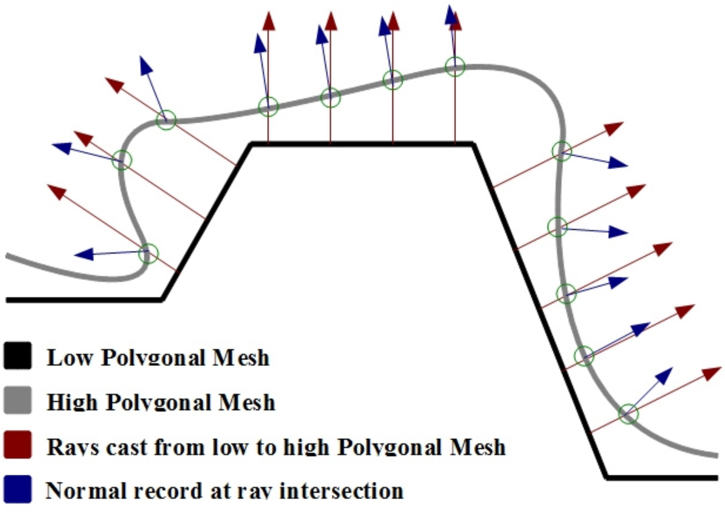
The main disadvantage of using this kind of normal map is the difficulty to avoid smoothing problems from the low-poly vertex normals. It has slightly slower performance than an object-space map as well.

On the other hand, object-space normal maps show several vivid colors. Objects with these maps applied can rotate, but usually shouldn't be deformed, unless the shader has been modified to support deformation. Each mesh shape would then require a unique map, being difficult to reuse it on differently shaped meshes. Comparing with tangent-space, it is easier to generate high quality curvature because it ignores the crude smoothing of the low polygonal vertex normals.

Most recent modeling applications offer the option to create a normal map from a detailed model. This is usually refered as to baking software. In addition, these applications often allow you to convert a grayscale bump map into a normal map.

When creating normal maps, the baking software uses the normals from your high-poly mesh and applies them into a normal map for the low polygonal mesh.
It usually starts projecting a certain numerical distance out from the low polygonal mesh, and sends rays inwards towards the high polygonal model. When a ray intersects the high resolution mesh, it records the mesh's surface normal into the normal map.

It is important to remark that the target mesh should not contain overlapping UVs or zero-sized triangles, which will post problems when rastering the low poly model in its UV coordinate system.


■ Low Polygonal Mesh
■ High Polygonal Mesh
■ Rays cast from low to high Polygonal Mesh
■ Normal record at ray intersection

Normal map record via ray intersection.

References

[1] Fitting Smooth Surfaces to Dense Polygon Meshes, Venkat Krishnamurthy and Marc Levoy, .SIGGRAPH 1996.

[2] Appearance Preserving Simplification, Cohen et al. SIGGRAPH 1998.

[3] A general method for preserving attribute values on simplified meshes, Cignoni et al. IEEE Visualization '98.

[4] A. Fournier, "Normal distribution functions and multiple surfaces," *Graphics Interface '92 Workshop on Local Illumination*, pp. 45--52, 1992

[5] H. Rushmeier, G. Taubin, and A. Gu´eziec. Applying shape from lighting variation to bump map capture. In Rendering Techniques '97 (Proceedings of Eurographics Rendering Workshop), pages 35–44, June 1997.

[6] J. Cohen, M. Olano, and D. Manocha. Appearance-preserving simplification. In Computer Graphics (SIGGRAPH '98 Proceedings), pages 115–122, July 1998.

[7] Realistic, Hardware-accelerated Shading and Lighting Wolfgang Heidrich, Hans-Peter Seidel Max-Planck-Institute for Computer Science, SIGGRAPH '99

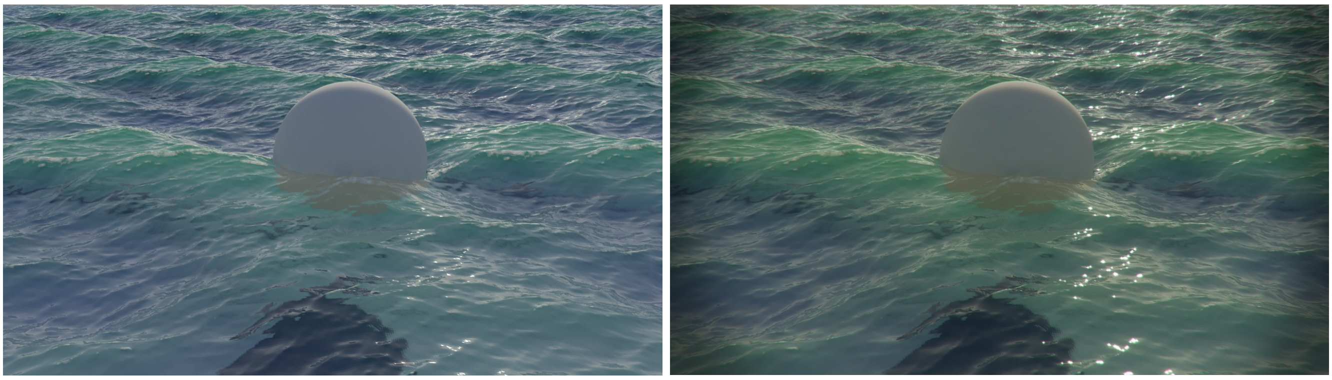
Render Passes.

An issue which is often overlooked or avoided is the necessity of fine-tuning projects in conjunction with a compositing package. Yet the level of interactive flexibility available within just about any 2D application can save precious time. [1]
While it is possible when working on full photorealistic 3D projects to render the final image directly from the 3D rendering engine, this methodology is rarely utilized in production environments.

Furthermore, as most work in broadcast and film utilizes 3D imagery as a subservient element to a live-action backplate, compositing is going to occur regardless. While many visual effects shots incorporate 3D, all visual effects shots are composited.

What also must be mentioned is that there are many effects which not only benefit from, but require a compositing application's specialized features: Color correction, film grain, effects such as depth of field [see camera section], fog, glows, motion blur [see camera section], heat distortion, optical effects... while some of these can be achieved in most 3D application sofware with varying degrees of success [refer to Feature Chart in Software Specs chapter], an experienced compositor with a strong application can often times take things further.

The image below shows the differences between the initial rendered image result of the rendering process with MentalRay photorealistic render engine, and the post-processed image.
Note that color correction, devignetting, optical glare, chromatic aberration, glow and sharpening effects were applied to the initial image using Eyeon Fusion 6 Software. (Composition effects were exaggerated to be clearly noticeable).

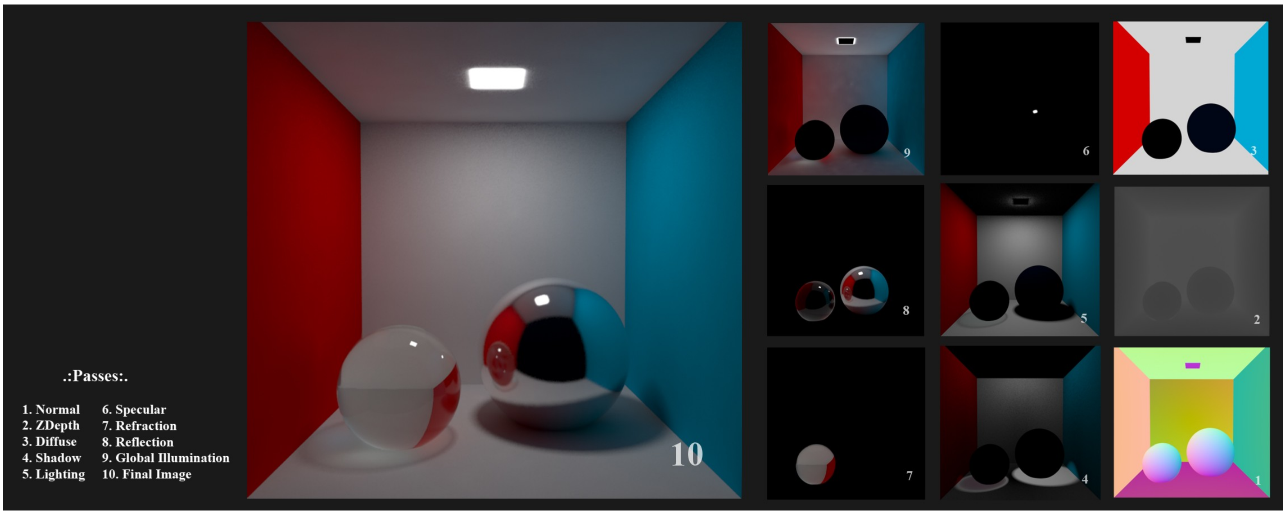


Differences between non-processed and post-processed images.
(MentalImages MentalRay render, Eyeon Fusion compositing)

A render pass creates layer of a scene that can be composited with any other passes to create a complete image. Passes also allow you to quickly re-render a single layer without re-rendering the entire scene. Later, you can composite the rendered passes using most current composition softwares.

It is worth noting that these passes should be rendered in 32bit image format if possible, so as much pixel information as available is preferable to have when applying post-processing algotithms to rendered images.

Refer to the image below to see the different passes composed to get the final CornellBox image [see CornellBox section].



Rendering in passes a CornellBox.
(Rendered in Maya's Vray)

Rendering in passes is therefore the process of rendering different attributes of your scene separately. While some people use the terms interchangeably [2], rendering in layers is different than rendering passes, being the latter a process of rendering different objects in your scene separately, so that a different image is rendered for each layer of objects.

All that said, rendering passes is an essential feature available in most current photorealistic render engines, and must be a requirement when rendering production images.
The following section shows the different types of rendering passes commonly available, offering a huge amount of flexibility when fine-tuning is needed.

Pass Types [2]

A **beauty pass** (sometimes called **diffuse pass** or **color pass**) is the main, full-color rendering of your subject, including diffuse illumination, color, and color maps. A beauty pass usually will not include reflections, highlights, and shadows, which are usually separate passes.

Highlight passes (sometimes called **specular passes**) isolate the specular highlights from your objects. You can render highlight passes by turning off any ambient light and making the object's diffuse shading and color mapping to pure black. The result will be a rendering of all the specular highlights in the scene, over a black background, without any other types of shading.

A **reflection pass** includes reflections of other objects or the surrounding environment, and can either replace or complement the highlight pass. To isolate reflections, usually all you need to do is turn off ambient, diffuse, and specular shading from a surface, so that only reflections appear.

A **shadow pass** is a rendering that shows the locations of shadows in a scene. A shadow pass often appears as a white shadow region against a black background, a black shadow against a white background, or a rendering with the shadow shape embedded in the alpha channel. **Cast shadows** are where an object casts a shadow onto another 3D object or darkens an area of a live-action plate. Separate shadow passes can depict **attached shadows**, where an object casts shadows onto itself, such as the shadow a character's nose casts onto his own face.

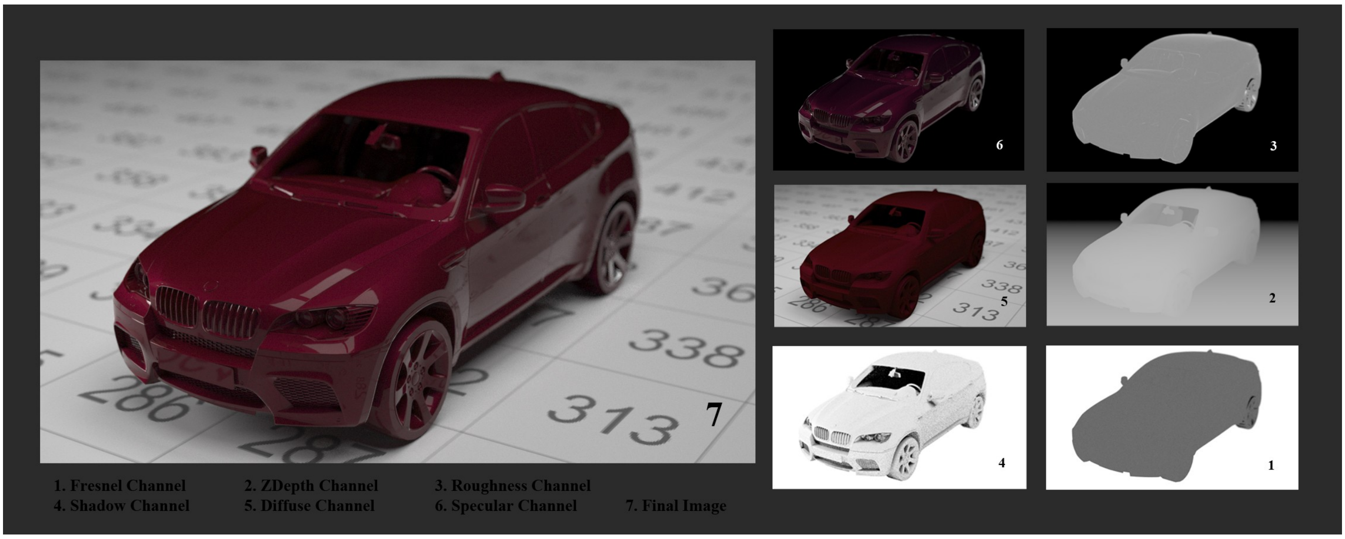
A **lighting pass** is an optional part of multi-pass rendering, that adds flexibility and control to the compositing process. Instead of rendering a beauty pass all at once, you could instead render multiple lighting passes, with each individual lighting pass showing the influence of one light (or one group of lights) on an element. Other lights are hidden or deleted when rendering the lighting pass.

Effects passes may sometimes be rendered, depending on the needs of your project. An effects pass is a separate rendering of a visual effect or a mask for a visual effect. An effects pass might be an optical effect, such as a light glow or lens flare, or a particle effect, such as a cloud of smoke or plume of jet exhaust.

A **depth map** (also called **Z-Depth** or a **depth pass**) is a pass that stores depth information at each point in your scene. Some productions use depth maps rendered in a special depth map file format. Other productions use simulated depth maps which are rendered as standard image files just like any other pass, but with a depth-fading effect over objects with constant white shading.

Passes can be rendered one at a time by rendering differently modified versions of your 3D scene, or some software can set them up automatically or render more than one pass type at once.

As an example, MaxwellRender v2.5 allows you to determine whether you want your channels to be exported as independent files (Separate), or embedded as one single file (Embedded) in the formats that allow extra buffers, like exr or tif. [3]



Rendering a Scene using separate channels.
(MaxwellRender 2.5)

Uses For Passes [2]

As said before, rendering passes play an important part when fine-tuning compositing is needed. However, scenes are split into different passes in professional work for some other important reasons, that are worth mentioned.

Memory can be saved by not putting every object into a rendering at once, making complex renderings possible on ordinary Pcs.

Changes can be made with little or no re-rendering. Often, just one element (such as a character) needs to be re-rendered, instead of a whole environment. Other adjustments, such as adjusting the darkness of a shadow or the color of a glow, can be made without any re-rendering, just by adjusting your composite.

Integration with live-action plates depends on separate passes. For example, if your character needs to cast a shadow onto the ground in a photograph, or appear reflected in a real pool of water.

Still Images can sometimes be used for some elements, especially if the camera isn't moving. For example, one still frame of a room might be composited behind every frame of a separately rendered animated character.

Recycling is often possible, where a separate element can be reused in different positions or times within a shot.

Reflections can be softly blurred in the composite, which eliminates the need for anti-aliasing or high-quality rendering for the reflection pass.

Particles can be used to create different effects in compositing. For example, if you wanted a natural refraction or heat-ripple effect from your particles, you could render them as a separate effects pass, then use the rendered particles as a mask for a glass distortion filter or warping of the background plate behind them.

Bump mapping can be applied selectively to reflections, beauty pass, or highlight pass, instead of having the same bump map on each element of the shader.

Glows can be created and manipulated in post by adding a blurred copy of your specular pass on top of the composite.

Depth of Field effects can be created without requiring any extra rendering time, by blurring separately rendered background layers, or by using a depth pass as a mask for a blur filter. You can also create atmospheric perspective (fading to blue or gray with distance) with a color correction masked by the depth pass.

References

[1] Alex Alvarez, "Rendering in Passes and Layers". 2002, The Gnomon Workshop, Inc.

[2] Jeremy Birn, "Digital Lighting & Rendering" Second Edition. Chapter 11, "Rendering Passes and Compositing".

[3] NextLimit Technologies, Maxwell Render 2.5.1 User Manual. Chapter 8. "Setting up the render output".

Structural Similarity Index.

Objective methods for assessing perceptual image quality traditionally attempted to quantify the visibility of errors or differences between a distorted image and a reference image using a variety of known properties of the human visual system.

However, for applications in which images are ultimately to be viewed by human beings, the only “correct” method of quantifying visual image quality is through subjective evaluation.
An image signal whose quality is being evaluated can be thought of as a sum of an undistorted reference signal and an error signal.
A widely adopted assumption is that the loss of perceptual quality is directly related to the visibility of the error signal. The simplest implementation of this concept is the MSE (Mean squared error), which objectively quantifies the strength of the error signal. But two distorted images with the same MSE may have very different types of errors, some of which are much more visible than others.

$$E(\mathbf{x},\mathbf{y}) = \frac{1}{M}\sum_{i=1}^{M}(x_i - y_i)^2$$

Mean squared error (MSE) calculation.

In practice, however, subjective evaluation is usually too inconvenient, time-consuming and expensive. The goal of research in objective image quality assessment is to develop quantitative measures that can automatically predict perceived image quality.

Back in 2004, under the assumption that human visual perception is highly adapted for extracting structural information from a scene, Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli introduced an alternative complementary framework for quality assessment based on the degradation of structural information.
As a specific example of this concept, they developed a Structural Similarity Index (SSIM) and demonstrated its promise through a set of intuitive examples, as well as comparison to both subjective ratings and state-of-the-art objective methods on a database of images compressed with JPEG and JPEG2000.1 [1]

SSIM basicaly compares local patterns of pixel intensities that have been normalized for luminance and contrast. Such an image quality metric can be used to benchmark image processing systems and algorithms. Objective image quality metrics can be classified according to the availability of an original (distortion-free) image, with which the distorted image is to be compared.

It has been proven that a very simple SSIM algorithm provides surprisingly good image quality prediction performance for a wide variety of image distortions [2].
A major drawback of the spatial domain SSIM algorithm is that it is highly sensitive to translation, scaling and rotation of images.

In spatial domain, the SSIM index between two image patches is calculated as follows:

$$S(\mathbf{x},\mathbf{y}) = \frac{(2\,\mu_x\,\mu_y + C_1)\,(2\,\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)\,(\sigma_x^2 + \sigma_y^2 + C_2)}$$

SSIM index calculation.

with

- μ_x the average of x;
- μ_y the average of y;
- σ_x^2 the variance of x;
- σ_y^2 the variance of y;
- σ_{xy} the covariance of x and y;
- $c_1=(k_1L)^2$, $c_2=(k_2L)^2$ two variables to stabilize the division with weak denominator;
- L the dynamic range of the pixel-values (typically this is $2^{\#bits\ per\ pixel}-1$);
- $k_1=0.01$ and $k_2=0.03$ by default.

$$\mu_x = \frac{1}{M}\sum_{i=1}^{M} x_i,\ \sigma_x^2 = \frac{1}{M}\sum_{i=1}^{M}(x_i-\mu_x)^2$$

$$\sigma_{xy} = \frac{1}{M}\sum_{i=1}^{M}(x_i-\mu_x)(y_i-\mu_y),$$

It can be shown that the maximum SSIM index value 1 is achieved if and only if x and y are identical.

To avoid translation problems in SSIM, Wang and Simoncelli [4] attempted to extend the current SSIM method to the complex wavelet transform domain and make it insensitive to these “non-structured” image distortions that are typically caused by the movement of the image acquisition devices, rather than the changes of the structures of the objects in the visual scene.

They proposed a complex wavelet domain image similarity measure, which is simultaneously insensitive to luminance change, contrast change and spatial translation. The key idea is to make use of the fact that these image distortions lead to consistent magnitude and/or phase changes of local wavelet coefficients. Since small scaling and rotation of images can be locally approximated by translation, the proposed measure also shows robustness to spatial scaling and rotation when these geometric distortions are small relative to the size of the wavelet filters. [3]

Compared with previous methods, the proposed measure, called CW-SSIM (complexwavelet structural similarity index) is computationally efficient, and can evaluate the similarity of two images without a precise registration
process at the front end.

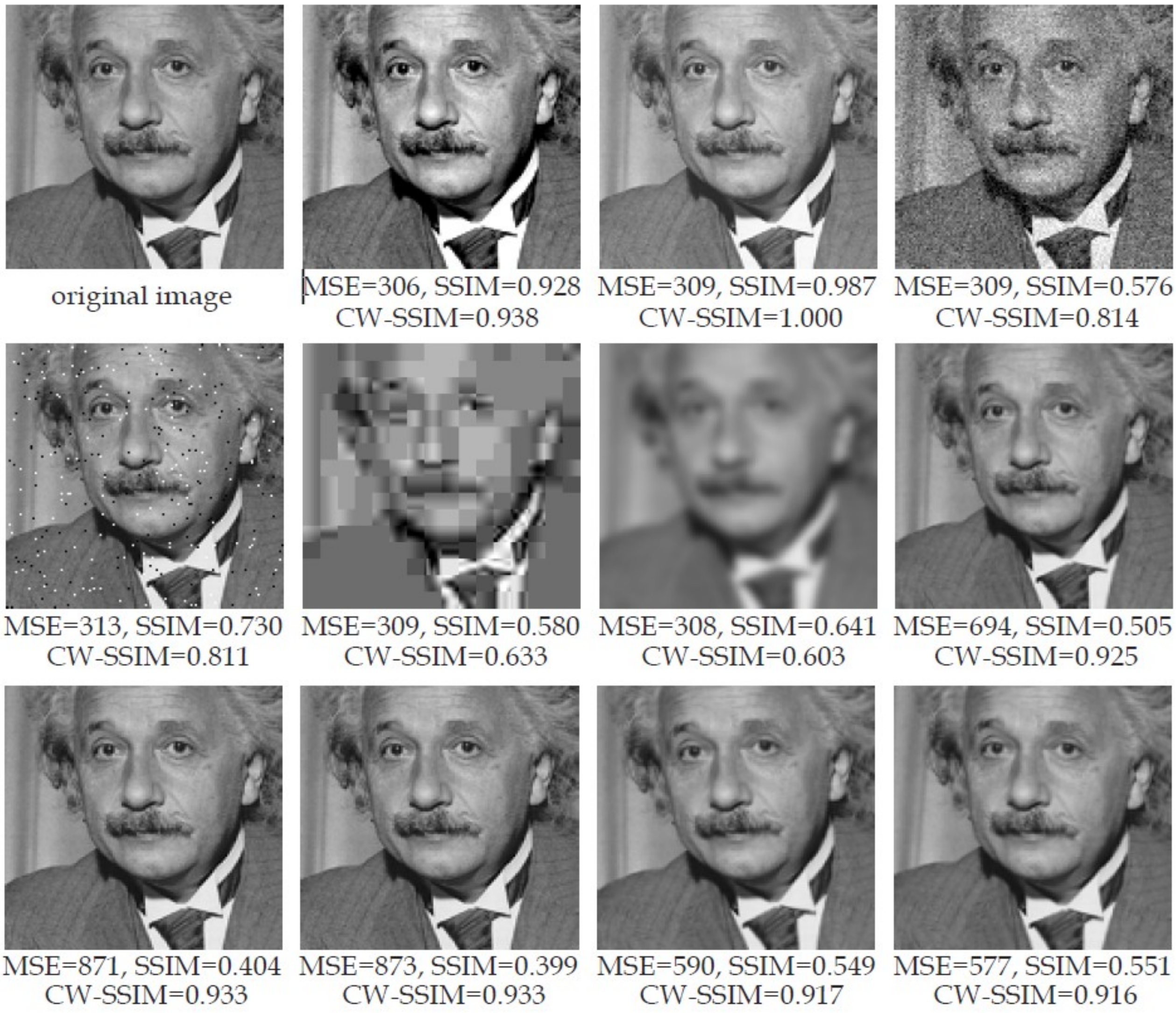
$$\boxed{\tilde{S}(\mathbf{c}_x,\mathbf{c}_y) = \frac{2\left|\sum_{i=1}^{N} c_{x,i}\, c_{y,i}^{*}\right| + K}{\sum_{i=1}^{N}|c_{x,i}|^2 + \sum_{i=1}^{N}|c_{y,i}|^2 + K}}$$

CW - SSIM index calculation.

Here c* denotes the complex conjugate of c and K is a small positive constant.
To better understand the CW-SSIM index, we rewrite it as a product of two components:

$$\underbrace{\frac{2\sum_{i=1}^{N} |c_{x,i}||c_{y,i}| + K}{\sum_{i=1}^{N} |c_{x,i}|^2 + \sum_{i=1}^{N} |c_{y,i}|^2 + K}}_{\text{magnitude comparison}} \cdot \underbrace{\frac{2\left|\sum_{i=1}^{N} c_{x,i}\, c_{y,i}^*\right| + K}{2\sum_{i=1}^{N} |c_{x,i}\, c_{y,i}^*| + K}}_{\text{phase comparison}}$$

CW - SSIM index two component calculation.



original image

MSE=306, SSIM=0.928 CW-SSIM=0.938

MSE=309, SSIM=0.987 CW-SSIM=1.000

MSE=309, SSIM=0.576 CW-SSIM=0.814

MSE=313, SSIM=0.730 CW-SSIM=0.811

MSE=309, SSIM=0.580 CW-SSIM=0.633

MSE=308, SSIM=0.641 CW-SSIM=0.603

MSE=694, SSIM=0.505 CW-SSIM=0.925

MSE=871, SSIM=0.404 CW-SSIM=0.933

MSE=873, SSIM=0.399 CW-SSIM=0.933

MSE=590, SSIM=0.549 CW-SSIM=0.917

MSE=577, SSIM=0.551 CW-SSIM=0.916

Comparison of image similarity measures for images with different types of distortions. (a) reference image (8bits/pixel, assumed to have perfect quality); (b) contrast stretch; (c) mean luminance shift; (d) Gaussian noise contamination; (e) impulsive noise contamination; (f) JPEG compression; (g) blurring; (h) spatial scaling (zooming out); (i) spatial translation (to the right); (j) spatial translation (to the left); (k) rotation (counterclockwise); (l) roation (clockwise). Images are cropped from 256£256 to 128£128 for visibility. Note that Images (b)-(g) have almost the same MSE values but drastically different visual quality, which is better predicted by SSIM and CW-SSIM. Also note that MSE and SSIM are both sensitive to translation, scaling and rotation (Images (h)-(l)), and CW-SSIM is robust to these distortions. [4]

References

[1] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: From error visibility to structural similarity," IEEE Trans. Image Processing, vol. 13, pp. 600–612, Apr. 2004.

[2] Z. Wang and E. P. Simoncelli, "Local phase coherence and the perception of blur," in Adv. Neural Information Processing Systems (NIPS03), vol. 16, (Cambridge, MA), MIT Press, May 2004.

[3] P. Y. Simard, Y. LeCun, J. S. Denker, and B. Victorri, "Transformation invariance in pattern recognition – tangent distance and tangent propagation," International Journal of Imaging Systems and Technology, vol. 11, no. 3, 2000.

[4] Zhou Wang and Eero P. Simoncelli "Translation insensitive image similarity in complexwavelet domain." Lab for Computational Vision, New York University, New York, NY 10003 March 2005

SSIM matlab implementation.

```
function [mssim, ssim_map] = ssim_index(img1, img2, K, window, L)

%========================================================================
%SSIM Index, Version 1.0
%Copyright(c) 2003 Zhou Wang
%All Rights Reserved.
%
%The author is with Howard Hughes Medical Institute, and Laboratory
%for Computational Vision at Center for Neural Science and Courant
%Institute of Mathematical Sciences, New York University.
%
%----------------------------------------------------------------------
%Permission to use, copy, or modify this software and its documentation
%for educational and research purposes only and without fee is hereby
%granted, provided that this copyright notice and the original authors'
%names appear on all copies and supporting documentation. This program
%shall not be used, rewritten, or adapted as the basis of a commercial
%software or hardware product without first obtaining permission of the
%authors. The authors make no representations about the suitability of
%this software for any purpose. It is provided "as is" without express
%or implied warranty.
%----------------------------------------------------------------------
%
%This is an implementation of the algorithm for calculating the
%Structural SIMilarity (SSIM) index between two images. Please refer
%to the following paper:
%
%Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image
%quality assessment: From error visibility to structural similarity"
%IEEE Transactios on Image Processing, vol. 13, no. 4, pp.600-612,
%Apr. 2004.
%
%Kindly report any suggestions or corrections to zhouwang@ieee.org
%
%----------------------------------------------------------------------
%
%Input : (1) img1: the first image being compared
%        (2) img2: the second image being compared
%        (3) K: constants in the SSIM index formula (see the above
%            reference). defualt value: K = [0.01 0.03]
%        (4) window: local window for statistics (see the above
%            reference). default widnow is Gaussian given by
%            window = fspecial('gaussian', 11, 1.5);
%        (5) L: dynamic range of the images. default: L = 255
%
%Output: (1) mssim: the mean SSIM index value between 2 images.
%            If one of the images being compared is regarded as
%            perfect quality, then mssim can be considered as the
%            quality measure of the other image.
%            If img1 = img2, then mssim = 1.
%        (2) ssim_map: the SSIM index map of the test image. The map
%            has a smaller size than the input images. The actual size:
%            size(img1) - size(window) + 1.
%
%Default Usage:
%   Given 2 test images img1 and img2, whose dynamic range is 0-255
%
%   [mssim ssim_map] = ssim_index(img1, img2);
%
```

```
%Advanced Usage:
%   User defined parameters. For example
%
%   K = [0.05 0.05];
%   window = ones(8);
%   L = 100;
%   [mssim ssim_map] = ssim_index(img1, img2, K, window, L);
%
%See the results:
%
%   mssim                        %Gives the mssim value
%   imshow(max(0, ssim_map).^4)  %Shows the SSIM index map
%
%========================================================================

if (nargin < 2 | nargin > 5)
  ssim_index = -Inf;
  ssim_map = -Inf;
  return;
end
if (size(img1) ~= size(img2))
  ssim_index = -Inf;
  ssim_map = -Inf;
  return;
end
[M N] = size(img1);
if (nargin == 2)
  if ((M < 11) | (N < 11))
            ssim_index = -Inf;
            ssim_map = -Inf;
    return
  end
  window = fspecial('gaussian', 11, 1.5);      %
  K(1) = 0.01;                                                                  % default settings
  K(2) = 0.03;                                                                  %
  L = 255;                         %
end
if (nargin == 3)
  if ((M < 11) | (N < 11))
            ssim_index = -Inf;
            ssim_map = -Inf;
    return
  end
  window = fspecial('gaussian', 11, 1.5);
  L = 255;
  if (length(K) == 2)
    if (K(1) < 0 | K(2) < 0)
                    ssim_index = -Inf;
                  ssim_map = -Inf;
                  return;
    end
  else
            ssim_index = -Inf;
          ssim_map = -Inf;
            return;
  end
end
if (nargin == 4)
  [H W] = size(window);
  if ((H*W) < 4 | (H > M) | (W > N))
            ssim_index = -Inf;
            ssim_map = -Inf;
    return
  end
  L = 255;
  if (length(K) == 2)
    if (K(1) < 0 | K(2) < 0)
                      ssim_index = -Inf;
                    ssim_map = -Inf;
                    return;
    end
  else
            ssim_index = -Inf;
          ssim_map = -Inf;
            return;
  end
end
if (nargin == 5)
  [H W] = size(window);
  if ((H*W) < 4 | (H > M) | (W > N))
            ssim_index = -Inf;
            ssim_map = -Inf;
    return
  end
  if (length(K) == 2)
    if (K(1) < 0 | K(2) < 0)
                      ssim_index = -Inf;
                    ssim_map = -Inf;
                    return;
    end
  else
            ssim_index = -Inf;
          ssim_map = -Inf;
            return;
  end
end
C1 = (K(1)*L)^2;
C2 = (K(2)*L)^2;
window = window/sum(sum(window));
img1 = double(img1);
img2 = double(img2);

mu1   = filter2(window, img1, 'valid');
mu2   = filter2(window, img2, 'valid');
mu1_sq = mu1.*mu1;
mu2_sq = mu2.*mu2;
mu1_mu2 = mu1.*mu2;
sigma1_sq = filter2(window, img1.*img1, 'valid') - mu1_sq;
sigma2_sq = filter2(window, img2.*img2, 'valid') - mu2_sq;
sigma12 = filter2(window, img1.*img2, 'valid') - mu1_mu2;
if (C1 > 0 & C2 > 0)
  ssim_map = ((2*mu1_mu2 + C1).*(2*sigma12 + C2))./((mu1_sq + mu2_sq + C1).*(sigma1_sq + sigma2_sq + C2));
else
  numerator1 = 2*mu1_mu2 + C1;
  numerator2 = 2*sigma12 + C2;
          denominator1 = mu1_sq + mu2_sq + C1;
  denominator2 = sigma1_sq + sigma2_sq + C2;
  ssim_map = ones(size(mu1));
  index = (denominator1.*denominator2 > 0);
  ssim_map(index) = (numerator1(index).*numerator2(index))./(denominator1(index).*denominator2(index));
  index = (denominator1 ~= 0) & (denominator2 == 0);
  ssim_map(index) = numerator1(index)./denominator1(index);
end
mssim = mean2(ssim_map);
return
```

CW-SSIM matlab implementation.

```
% Inputs:

%   IM1, IM2    Equally sized images to be compared (0-255 doubles)
%   LEVEL                    Level or "Height" of the pyramid decomposition to use
%                 Note: currently, this also determines the level that is
%                       used when computing cssim -> can't use the
%                       baseband <TO CHANGE>
%                 Note: higher pyramids add more low-frequency subbands
%   NUMOR       Number of orientations to use in the pyramid decomposition
%   GUARDB      Size of the boundry region to exclude
%   WINSIZE     Size of the sliding window (optional, default 7)
%   WTS         Weight vector [HP BP(1...LEVEL) LP], length LEVEL+2 (opt)
%
% Outputs:
%   CSSIM_WTD   Overall metric value with weighted subbands
%   CSSIM       Overall mean metric value of passband only
%   LEV_CSSIM   Vector of metric values for each level (from HP to BB)
%   BAND_CSSIM  Matrix of metric values for each orientation
%   CSSIM_MAP   RxCxOR Matrix of individual metric values (for RxC images)

% Original Author: Zhou Wang
% Current Changes Author: Alan Brooks
%
% 16-Mar-2005   slight modification to also pass back cssim_map's
% 26-Sep-2005   added winsize parameter
% 13-Oct-2005   added ability to access baseband
% 27-Oct-2005   removed mean from cssim_lp computations (since equ assumes)
%               added weights (need sub-function)
%  6-Jun-2006   fixed function name in main function (cosmetic)

function [cssim_wtd, cssim, lev_cssim, band_cssim, cssim_map, ...
    cssim_lp, cssim_map_lp, cssim_hp, cssim_map_hp, ...
    ssim_lp, ssim_map_lp] = ...
    cssim_index_multi(im1, im2, level, numOr, guardb, winsize, wts)

% build the steerable pyramids
[pyr1, pind] = buildSCFpyr(im1, level, numOr-1);
[pyr2, pind] = buildSCFpyr(im2, level, numOr-1);

%figure,showSpyr(abs(pyr2),pind);

% initialize
if ~exist('wts','var')
    % weight the subbands evenly if no weights given
    numWts = level + 2;
    wts = ones(numWts,1)/numWts;
else
    wts = wts/sum(wts);
end
if ~exist('winsize','var')
    nw = 7;
else
    nw = winsize;
end
window = ones(nw);
window = window./sum(sum(window));
C = 0.03;
band_cssim = zeros(level,numOr);
[sx,sy] = size(im1);
%gb = guardb;
```

```
% bandpass subbands at each level
for lev=1:level %(high to low frequency)
    gb = round(guardb/(2^(lev-1)));
    for i=1:numOr
        bandind = i+(lev-1)*numOr+1;
        [corr_band, varr_band] = ...
            calcCorrAndVarr(pyr1,pyr2,pind,bandind,gb,window);
        map = (2*abs(corr_band) + C)./(varr_band + C);
        cssim_map{lev,i} = map;
        band_cssim(lev,i) = mean2(cssim_map{lev,i});
    end % loop through orientations
end % loop over levels

% highpass and lowpass subbands
    % LP (baseband)
    gb = round(guardb/(2^(level-1)));
    bandind = size(pind,1);
    [corr_band, varr_band] = ...
        calcCorrAndVarr(pyr1,pyr2,pind,bandind,gb,window,1);
    map = (2*abs(corr_band) + C)./(varr_band + C);
    cssim_map_lp = map;

        % === EXTRA TEST CODE CALCULATING SSIM ON LP BAND ===
            % do same as W = fspecial('gaussian',srN,sigma)
        ndx = [1:nw]-nw/2-.5;
        sigma = 1.5;
        wts2 = ( (exp(-ndx.^2./(2*sigma^2)))'* ...
                (exp(-ndx.^2./(2*sigma^2)))   ) /(2*pi*sigma^2);
        wts2 = wts2./sum(wts2(:));
        im1_bb = pyrLow(pyr1, pind);
        im2_bb = pyrLow(pyr2, pind);
        maxLev = max([im1_bb(:); im2_bb(:)]);
        [ssim_lp ssim_map_lp] = ...
            ssim_index(im1_bb, im2_bb, [.01 .03], wts2, maxLev);
        % === END TEST CODE ===

    % HP
    bandind = 1;
    gb = guardb;
    [corr_band, varr_band] = ...
        calcCorrAndVarr(pyr1,pyr2,pind,bandind,gb,window);
    map = (2*abs(corr_band) + C)./(varr_band + C);
    cssim_map_hp = map;

% average over the orientations
lev_cssim = mean(band_cssim,2);

% add in the hp and lp (high-to-low frequnecy order)
cssim_lp = mean(cssim_map_lp(:));
cssim_hp = mean(cssim_map_hp(:));
lev_cssim = [cssim_hp lev_cssim' cssim_lp]';

% compute the weighted cssim including baseband and highpass
cssim_wtd = sum(lev_cssim.*wts(:))

% compute the original definition using only lowest freq passband
cssim = lev_cssim(level+1)




% === utility functions ===
function [corr_band, varr_band] = ...
    calcCorrAndVarr(pyr1,pyr2,pind,bandind,gb,window,remove_mean)
if ~exist('remove_mean','var'), remove_mean = 0; end
band1 = pyrBand(pyr1, pind, bandind);
band2 = pyrBand(pyr2, pind, bandind);
band1 = band1(gb+1:end-gb, gb+1:end-gb);
band2 = band2(gb+1:end-gb, gb+1:end-gb);
if remove_mean
    band1 = band1 - mean(band1(:)); % special for LP, remove mean
    band2 = band2 - mean(band2(:));
end
corr = band1.*conj(band2);
varr = abs(band1).^2 + abs(band2).^2;
corr_band = filter2(window, corr, 'valid');
varr_band = filter2(window, varr, 'valid');
```

Complex-Valued Steerable Pyramid matlab implementation.

```matlab
% [PYR, INDICES, STEERMTX, HARMONICS] = buildSCFpyr(IM, HEIGHT, ORDER, TWIDTH)

%
% This is a modified version of buildSFpyr, that constructs a
% complex-valued steerable pyramid  using Hilbert-transform pairs
% of filters.  Note that the imaginary parts will *not* be steerable.
%
% To reconstruct from this representation, either call reconSFpyr
% on the real part of the pyramid, *or* call reconSCFpyr which will
% use both real and imaginary parts (forcing analyticity).
%
% Description of this transform appears in: Portilla & Simoncelli,
% Int'l Journal of Computer Vision, 40(1):49-71, Oct 2000.
% Further information: http://www.cns.nyu.edu/~eero/STEERPYR/

% Original code: Eero Simoncelli, 5/97.
% Modified by Javier Portilla to return complex (quadrature pair) channels,
% 9/97.

function [pyr,pind,steermtx,harmonics] = buildSCFpyr(im, ht, order, twidth)
max_ht = floor(log2(min(size(im)))) – 2;

if (exist('ht') ~= 1)
  ht = max_ht;
else
  if (ht > max_ht)
    error(sprintf('Cannot build pyramid higher than %d levels.',max_ht));
  end
end

if (exist('order') ~= 1)
  order = 3;
elseif ((order > 15)  | (order < 0))
  fprintf(1,'Warning: ORDER must be an integer in the range [0,15]. Truncating.\n');
  order = min(max(order,0),15);
else
  order = round(order);
end
nbands = order+1;

if (exist('twidth') ~= 1)
  twidth = 1;
elseif (twidth <= 0)
  fprintf(1,'Warning: TWIDTH must be positive.  Setting to 1.\n');
  twidth = 1;
end

if (mod((nbands),2) == 0)
  harmonics = [0:(nbands/2)-1]'*2 + 1;
else
  harmonics = [0:(nbands-1)/2]'*2;
end

steermtx = steer2HarmMtx(harmonics, pi*[0:nbands-1]/nbands, 'even');

%-------------------------------------------------------------------

dims = size(im);
ctr = ceil((dims+0.5)/2);
```

```
[xramp,yramp] = meshgrid( ([1:dims(2)]-ctr(2))./(dims(2)/2), ...
   ([1:dims(1)]-ctr(1))./(dims(1)/2) );
angle = atan2(yramp,xramp);
log_rad = sqrt(xramp.^2 + yramp.^2);
log_rad(ctr(1),ctr(2)) =  log_rad(ctr(1),ctr(2)-1);
log_rad  = log2(log_rad);

%% Radial transition function (a raised cosine in log-frequency):
[Xrcos,Yrcos] = rcosFn(twidth,(-twidth/2),[0 1]);
Yrcos = sqrt(Yrcos);

YIrcos = sqrt(1.0 - Yrcos.^2);
lo0mask = pointOp(log_rad, YIrcos, Xrcos(1), Xrcos(2)-Xrcos(1), 0);
imdft = fftshift(fft2(im));
lo0dft =  imdft .* lo0mask;

[pyr,pind] = buildSCFpyrLevs(lo0dft, log_rad, Xrcos, Yrcos, angle, ht, nbands);

hi0mask = pointOp(log_rad, Yrcos, Xrcos(1), Xrcos(2)-Xrcos(1), 0);
hi0dft =  imdft .* hi0mask;
hi0 = ifft2(ifftshift(hi0dft));

pyr = [real(hi0(:)) ; pyr];
pind = [size(hi0); pind];
```

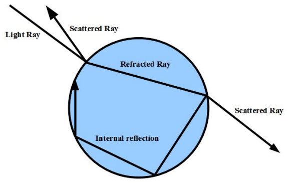
Subsurface Scattering.

Back in 2001, Henrik Wann Jensen, Stephen R. Marschner, Marc Levoy and Pat Hanrahan [2] introduced a new shading model for subsurface light transport.
It improved realism compared to the hard computer generated look achieved by traditional BRDF (Bidirectional Reflectance Distribution Function) , and was called Bidirectional Surface Scattering Reflectance Distribution Function, BSSRDF.

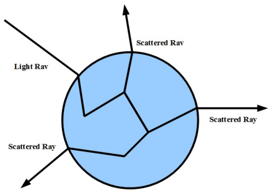
That was the first practical BSSRDF model for computer graphics, commonly known as subsurface scattering. Subsurface scattering, also known as SSS, is then a phenomenon of light transport through translucent objects.

Light penetrates the translucent surface at some point, and it is scattered by interacting with the particles of the material until exits the surface at a different point.

Interaction between light and the object's particles lead to several reflections of light inside the material at irregular angles before passing back out of the material. As a result, some light is absorbed and some is scattered back to the surface. [1]

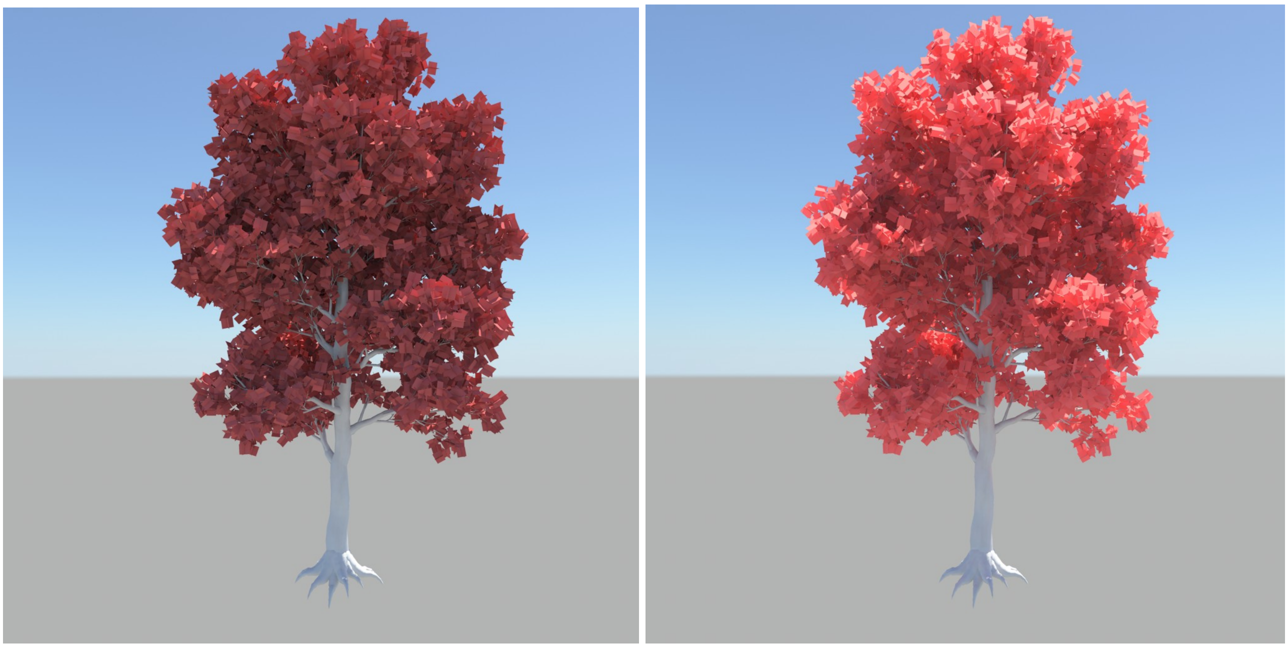


Traditional light scattering.



Subsurface scattering.

The following images show the difference between traditional scattering and subsurface scattering being applied to the same model that simulates translucency in leafs.



No subsurface scattering vs subsurface scattering applied to thin walled objects.
Rendered in MentalRay.

Subsurface material applied to dragon solid mesh.

Rendered in MentalRay.

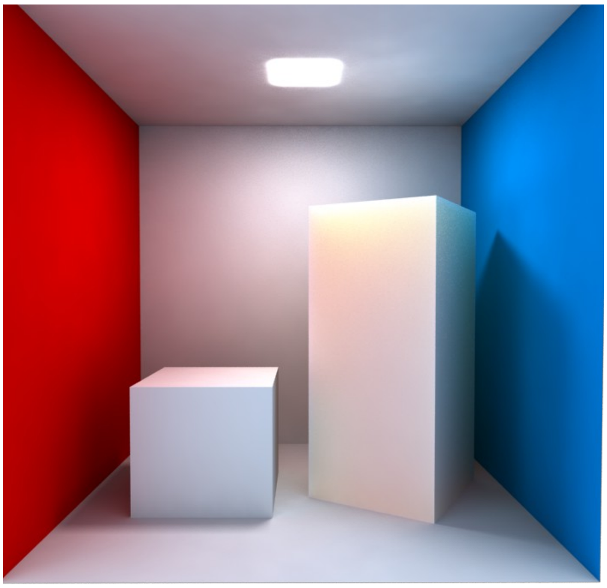
The model enabled efficient simulation of effects that BRDF models could not capture, such as color bleeding within materials and diffusion of light across shadow boundaries. The technique is efficient even for highly scattering media that were expensive to simulate using previously existing methods. They also described sampling techniques that allowed the model to be used within a conventional ray tracer [See raytracing section].

The face model Wann Jensen and Levoy used was pretty simple: few triangles and no texture, just a map separating the skin from the lip, a bumpmap [see bumpmap section] , and a light source.

The idea was to show the substantial difference that could be obtained with the new model without trying to fake subsurface scattering.

One year later, Wann Jensen and Buhler [3] introduced an efficient two-pass rendering technique for translucent materials. This approach was substantially faster than previous methods for rendering translucent materials, and it had the advantage that it integrated seamlessly with both scanline rendering [See scanline section] and global illumination methods [see global illumination section]. Tests made by Wann Jensen and Buhler showed an improvement factor of 150 compared to the BSSRDF model presented in 2001.

In conclussion, the new shading model was able to capture the soft appearance of many natural materials such as skin, milk and marble in a realistic way.



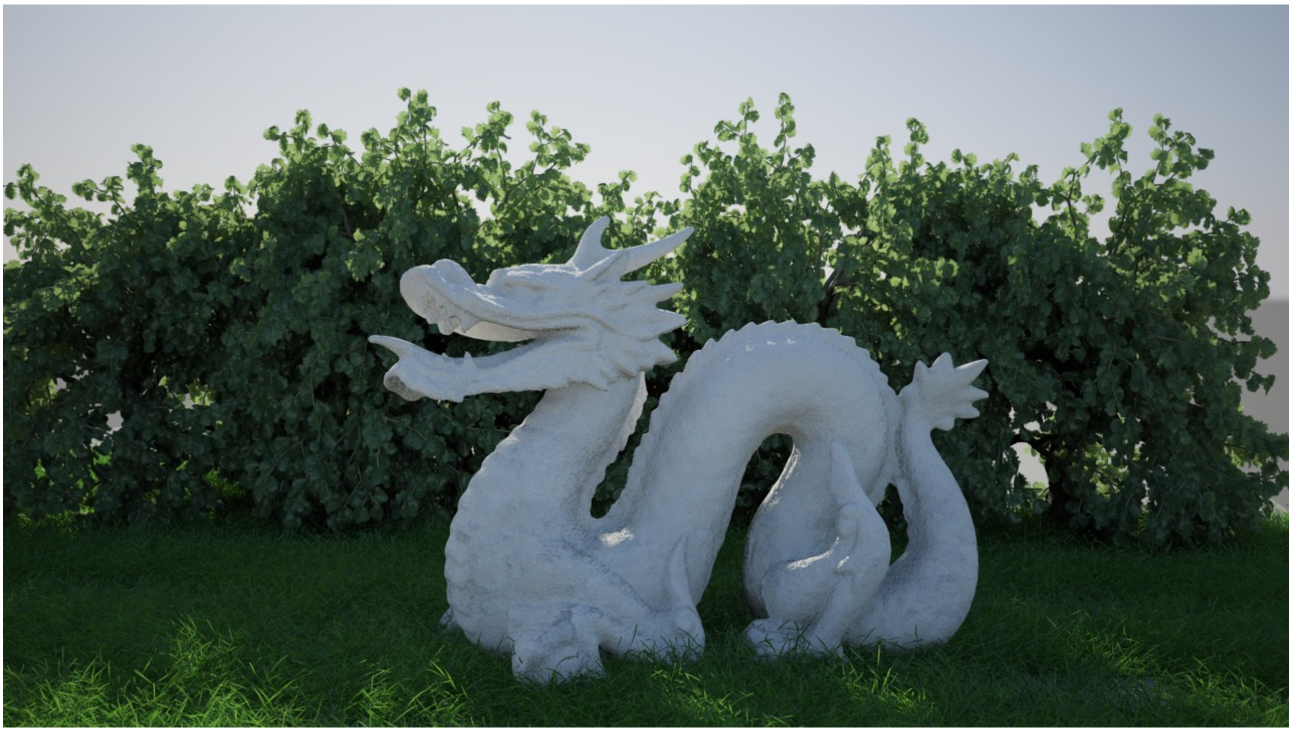
BRDF vs BSSRDF cubes inside a CornellBox [see CornellBox section]

Rendered in Vray

The BRDF model is a simplified version of the BSSRDF that does not account for subsurface light transport (it assumes that the light scatters at a single point on the surface). Note that the BRDF model does include subsurface scattering - the contributed is integrated into a single scattering term and a diffuse term. In contrast to the BRDF the BSSRDF does simulate subsurface light transport and gives the material a more natural translucent appearance. Note in particular the translucency in the upper part of the righten cube, where light is strongly penetrating the surface.

Simulating materials such as skin, marble, milk, vegetation, wax, and generally speaking all kind of translucent materials need the application of this technique and all photorealistic render engines nowadays should offer this feature, as light scattering inside materials is present in many situations in nature.

The following image shows only a specific application of BSSRDF applied to vegetation (grass and leafs), and a subtle subsurface effect shown in marble materials.



BSSRDF materials example applied throughout the scene.

Rendered in MentalRay.

References

[1]. MaxwellRender v2 Manual version 1.0 p.62. Copyright 2009 Next Limit SL

[2] Henrik Wann Jensen, Stephen R. Marschner, Marc Levoy and Pat Hanrahan: "A Practical Model for Subsurface Light Transport". Proceedings of SIGGRAPH'2001.

[3]. A Rapid Hierarchical Rendering Technique for Translucent Materials. Henrik Wann Jensen and Juan Buhler, Proceedings of SIGGRAPH'2002.

Volume Lightning.

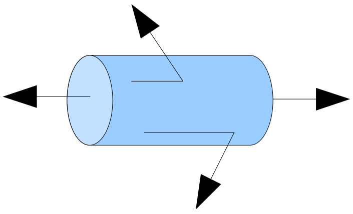
Volumetric lighting is a tecnique that reproduces beams of light shining through the environment. [1]

That technique commonly used in photorealistic render engines is a result of multiple volume scattering of light in a diffuse media (eg. Fog).

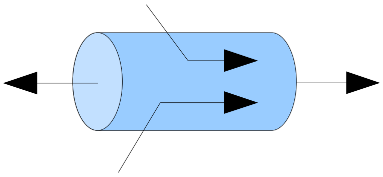
Real example of volume scattering of light.

Scattering has two effects. Depending on how light is redirected,

- Light along a line is scattered in a different direction.
- Light from some other direction is scattered into the direction of interest.



Out Scattering effect.



In Scattering effect.

As a result of this tecnique, light has the capability to give the effect of passing through a real three dimensional medium (such as fog, dust, smoke, or steam) that is inside its cone volume, just like in the real world.

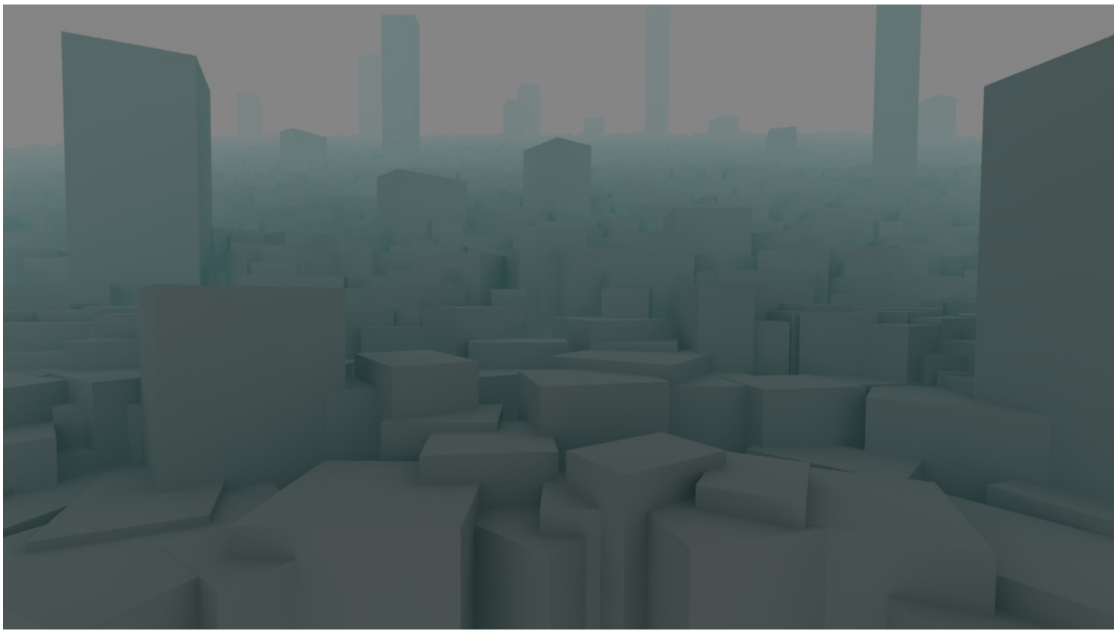
The appearance of many natural phenomena, such as human skin, clouds, fire, water, or the atmosphere, are strongly influenced by the interaction of light with volumetric media.

Therefore, efficiently rendering scenes with participating media has been an area of interest within computer graphics.

This problem is challenging, however, because accurately simulating light transport in participating media is computationally very expensive. [2]

It involves solving a recursive integral known as the rendering equation [6]. In the presence of scattering media the full three-dimensional radiative transfer equation must be considered, which is even more costly to compute.

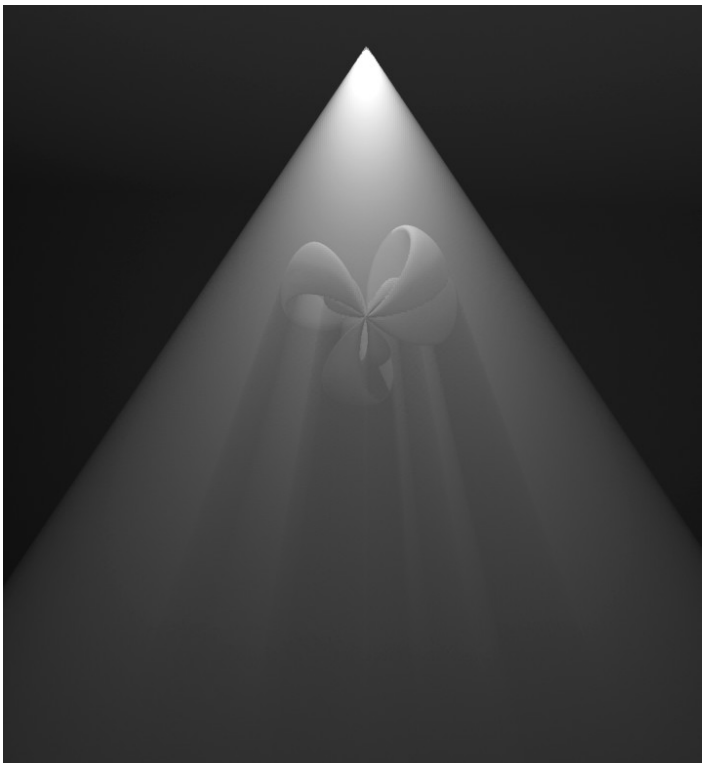
The most popular and successful algorithms to solve these equations are based on some form of stochastic ray tracing and Monte Carlo integration [See Global Illumination section]. Although these techniques tend to be very general and robust, they often suffer from noise. [7]



Volume scattering of light, fog. (Vray)

A basic approach to volumetric lighting would require two components: a light space shadow map, and a depth buffer.
Starting at the near clip plane of the camera, the whole scene would be traced and sampling values would be accumulated
into the input buffer. For each sample,it is determined if the sample is lit by the source of light being processed or not,
using the shadowmap as a comparison. Only lit samples will affect final pixel color.

In this tecnique, the emission of a given light source is considered as a cone. That cone is therefore considered as a container of the volume it fills.



Volumetric fog in participating media created by a SpotLight (MentalRay)

One way to optimize volumetric lighting effects is to render the lighting volume at a much coarser resolution than that which the graphics context is using.

This creates some bad aliasing artifacts, but that is easily touched up with a blur [See Motion blur section].

A more flexible framework, photon mapping [see Global Illumination section], was proposed in 1996 by Henrik Wann Jensen [3] which handled caustics in a natural manner on arbitrary geometry.

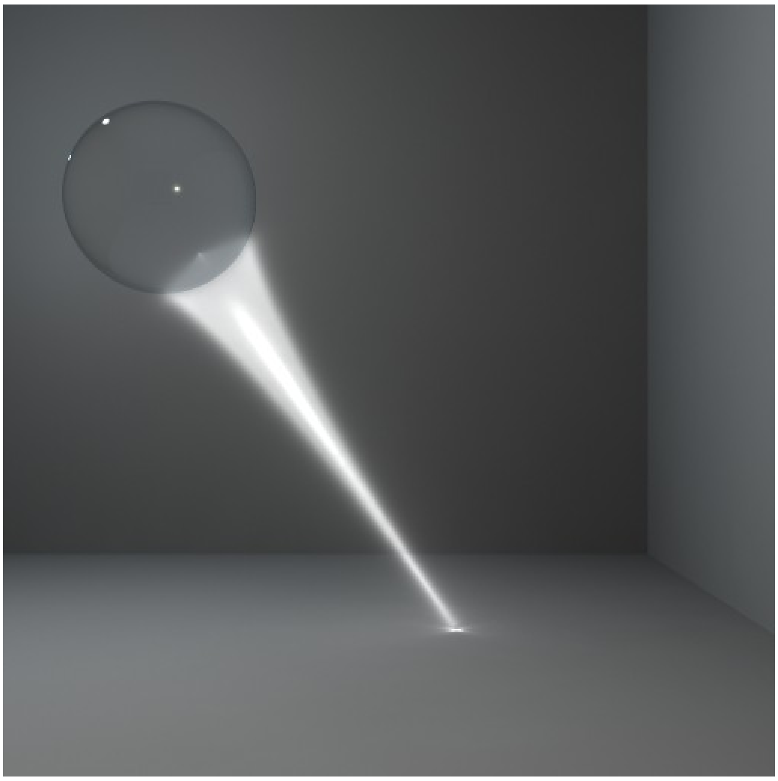
Two years later, in 1998, Henrik Wann Jensen and Per H. Christensen [4] developed a framework that was also able to support another application of volumetric lightning: volumetric caustics in participating media.

The downside of these approaches was that they suffered from noise that could only be overcome with a huge computational effort.

In 2008, Wojciech Jarosz and Matthias Zwicker [2] ,presented a new method for efficiently simulating the scattering of light within participating media. Using a theoretical reformulation of volumetric photon mapping.
Applying their method significantly reduced noise compared with conventional volumetric photon mapping using the same render time.

Volumetric caustics are complex light patterns formed by light when it first interacts with a specular surface and then scatters inside a participating medium.

Although this natural behaviour is simulated by existing techniques, accurate and physical correct simulation of this phenomena in nowadays render enginges is usually non trivial and consuming timewise.[5]



Volumetric Caustics in participating media (MentalRay)

References

[1] MentalImages Mentaltay 3.8 Documentation. Features Section, 2009.

[2] The Beam Radiance Estimate for Volumetric Photon Mapping. Wojciech Jarosz, Matthias Zwicker, Henrik Wann Jensen
University of California, San Diego. 2008

[3] Henrik Wann Jensen. Global Illumination Using Photon Maps. In Rendering Techniques ’96 (Proceedings of the Seventh Eurographics
Workshop on Rendering), pages 21–30, New York,
NY, 1996. Springer-Verlag/Wien.

[4] Henrik Wann Jensen and Per H. Christensen. Efficient simulation of light transport in scenes with participating media
using photon maps. In Computer Graphics (SIGGRAPH 98), pages 311–320. ACM Press, 1998.

[5] Scattering and light propagation, Symposium on Interactive 3D Graphics, ACM SIGGRAPH symposium on Interactive 3D
Graphics and Games,2010
Washington, D.C. Pages: 109-117

[6] KAJIYA J. T.: The Rendering Equation. In SIGGRAPH (New York, NY, USA, 1986), ACMPress, pp. 143–150.

[7] Irradiance Gradients in the Presence of Participating Media and Occlusions. Wojciech Jarosz, Matthias Zwicker,
Henrik Wann Jensen. University of California, San Diego. 2008

Contact

Pérez Roig, Francisco

Place and date of birth: **Valencia, 3rd December, 1985**

Nationality: **Spain**
E-mail address: **franciscoperezroig@gmail.com**
Telephone number: **(+34) 620 54 05 43**