



Università degli Studi di Milano Bicocca

Dipartimento di Informatica, Sistemistica e Comunicazione

Corso di Laurea Magistrale in Informatica

Spectral Clara Lux Tracer: physically based ray tracer with multiple shading models support

Relatore: Prof.re Gianluigi Ciocca

Co-relatore: Dott. Simone Bianco

Tesi di Laurea Magistrale di:

Fabrizio Daroni

Matricola 770157

Anno Accademico 2015-2016

Table of contents

List of figures	iv
List of tables	vi
Abbreviations	vii
Symbols	viii
1 INTRODUCTION	1
2 PRINCIPLES OF COLORIMETRY AND RADIOMETRY	7
2.1 COLORIMETRY	7
2.1.1 <i>Tristimulus values: CIE 1931 XYZ color space</i>	7
2.1.1.1 CIE 1931 XYZ Chromaticity coordinates	10
2.1.2 <i>CIE 1960 UCS color space</i>	11
2.1.3 <i>CIE 1964 UVW color space</i>	12
2.1.4 <i>CIE 1976 $L^* a^* b^*$ color space</i>	13
2.1.5 <i>Standard Illuminants</i>	13
2.1.6 <i>Macbeth Color checker</i>	14
2.1.7 <i>CAT: Chromatic Adaptation transform</i>	16
2.1.7.1 Von Kries Transform	16
2.1.7.2 CIECAT94 Transform	17
2.2 RADIOMETRY	19
2.2.1 <i>Radiant flux</i>	20
2.2.2 <i>Irradiance and radiant exitance</i>	20
2.2.3 <i>Radiance</i>	20
3 RAY TRACING: PHOTOREALISTIC RENDERING	22
3.1 THE RENDERING EQUATION	23
3.2 A BRIEF CLASSIFICATION OF RAY TRACING TECHNIQUES	24

3.2.1	<i>Path tracing</i>	25
3.3	BRDF AND BTDF	26
3.3.1	<i>Empirical models</i>	28
3.3.1.1	Phong model	28
3.3.1.2	Blinn-Phong model	29
3.3.2	<i>Physically based models</i>	29
3.3.2.1	Lambertian model	30
3.3.2.2	Oren-Nayar model	30
3.3.2.3	Specular reflection and transmission model	31
3.3.2.4	Torrance sparrow model	33
3.3.2.5	Measured BRDF	34
4	SCLT: OVERVIEW	36
4.1	CORE ARCHITECTURE	37
4.2	MODERN CROSS PLATFORM DEVELOPMENT	42
4.2.1	<i>Apple: iOS and OS X</i>	43
4.2.2	<i>Windows</i>	45
4.2.3	<i>Linux</i>	46
4.2.4	<i>Application architecture</i>	46
4.2.5	<i>Development tools, tests and continuous integration</i>	49
4.2.6	<i>Project size</i>	50
5	SCLT: PHYSICALLY BASED RENDERING	51
5.1	PHYSICALLY BASED RENDERING: IMPLEMENTATION DETAILS	51
5.1.1	<i>Diffuse and Specular models</i>	52
5.1.2	<i>Specular reflection/transmission models</i>	53
5.1.3	<i>Path tracing</i>	54
5.2	CASE STUDY 1: RGB SCENE USING WHITTED	56
5.3	CASE STUDY 2: SPECTRAL SCENES USING WHITTED	60
5.4	CASE STUDY 3: SPECTRAL SCENES USING PATH TRACING	68
6	SCLT: COLOR RENDERING INDEX	83
6.1	TEST SAMPLES METHOD	83
6.2	R96 _A METHOD	85
6.3	CRI: IMPLEMENTATION DETAILS	86
6.4	CASE STUDY: CRI CALCULATION	86

7	CONCLUSION	90
Appendix A	Spherical coordinates	93
Appendix B	SCLT scene XML file format	95
Appendix C	SCLT source code	107
References		109

List of figures

Figure 1. Big hero 6, the first Walt Disney animation movie produced with Hyperion.....	3
Figure 2. Monsters University, the first Pixar movie produced with RIS.	3
Figure 3. Example of a scene rendered using PBRT [8].....	4
Figure 4. Eye and photoreceptor cone cells anatomy [19].....	8
Figure 5. Visible light: a small part of the electromagnetic radiation spectrum [20].	8
Figure 6. CIE 1931 XYZ chromaticity diagram [23].	11
Figure 7. How works a general ray tracer.....	22
Figure 8. Ray tracing technique classification.	24
Figure 9. Path tracing follows the ray's path from camera to light source.	26
Figure 10. BRDF representation diagram [38].	27
Figure 11. Operating principle of a gonireflectometer [43].....	34
Figure 12. An example of a gonireflectometer [44].	35
Figure 13. Spectral Clara Lux Tracer logo.	36
Figure 14. Tracer classes hierarchy - UML class diagram.	38
Figure 15. ShadingModel classes hierarchy – UML class diagram.....	39
Figure 16. BRDF classes hierarchy – UML class diagram.....	41
Figure 17. SCLT OS X User Interface.....	44
Figure 18. SCLT iOS User Interface.	44
Figure 19. SCLT Windows 10 User Interface.	45
Figure 20. SCLT application architecture.....	46
Figure 21. Model-View-Controller architectural pattern.....	47
Figure 22. RGB scene: Whitted, Phong model, cube mapping, bump mapping.	58
Figure 23. RGB scene: Whitted, Blinn-Phong, multiple texture types.....	59
Figure 24. Spectral scene: Whitted, physically based BRDF, D65 illuminant.....	62
Figure 25. Spectral scene: Whitted, physically based BRDF, FL4 illuminant.	63
Figure 26. Spectral scene: Whitted, physically based BRDF, FL9 illuminant.	64
Figure 27. Spectral scene: Whitted, physically based BRDF, A illuminant.....	65

Figure 28. Measured BRDF data, illuminant D65, FL4, FL9, A.....	66
Figure 29. Spectral scene: Whitted, mesh (lambertian), D65 illuminant.....	67
Figure 30. First spectral path tracing scene: 20 samples per pixel, illuminant D65.....	70
Figure 31. First spectral path tracing scene: 200 samples per pixel, illuminant D65.....	71
Figure 32. First spectral path tracing scene: 800 samples per pixel, illuminant D65.....	72
Figure 33. First spectral path tracing scene: 4000 samples per pixel, illuminant D65.....	73
Figure 34. First spectral path tracing scene: 20000 samples per pixel, illuminant D65.....	74
Figure 35. Torrance Sparrow scene: PBRT 1000 samples, SCLT 10000 samples.....	75
Figure 36. Scene of case study 2 with path tracing: 20000 samples, illuminant D65.....	76
Figure 37. Scene of case study 2 with path tracing: 20000 samples, illuminant FL4.....	77
Figure 38. Spectral path tracing scene with D65 light inside it: 20000 samples.....	78
Figure 39. Spectral path tracing scene: 20000 samples per pixel, illuminant FL9.....	79
Figure 40. Spectral path tracing scene: mesh (lambertian), 10000 samples.....	80
Figure 41. Spectral path tracing scene: mesh (Torrance-Sparrow), 10000 samples.....	81
Figure 42. Spectral path tracing scene: mesh (reflection and transmission), 20000 samples.....	82
Figure 43. Example of CRI calculation.....	88
Figure 44. Whitted scenes with their CRI obtained from SCLT.....	89
Figure 45. Spherical coordinates representation.....	93

List of tables

Table 1. Macbeth color checker under D50 illuminant.	15
Table 2. Color Rendering Index for the F family of illuminants.	87
Table 3. Color Rendering Index for A and D65 illuminant.	88
Table 4. scene tag attributes list.	95
Table 5. viewReferencePoint tag attributes list.	96
Table 6. lookAtPoint tag attributes list.	96
Table 7. viewPlane tag attributes list.	96
Table 8. light tag attributes list.	97
Table 9. traceIModelType attributes list.	97
Table 10. object tag attributes list.	98
Table 11. origin tag attributes list.	99
Table 12. radius tag attributes list.	99
Table 13. color tag attributes list.	99
Table 14. vertex tag attributes list.	100
Table 15. pointOnPlane tag attributes list.	100
Table 16. material tag attributes list.	102
Table 17. objFile tag attribute list.	102
Table 18. min tag attribute list.	103
Table 19. max tag attribute list.	103

Abbreviations

PBR	Physically Based Rendering
SCLT	Spectral Clara Lux Tracer
CIE	Commission Internationale de l'Éclairage
RGB	Red Green Blue
CRI	Color Rendering Index
SPD	Spectral Power Distribution
CMF	Color Matching Function
CCT	Color Correlated Temperature
CAT	Chromatic Adaptation Transform
BRDF	Bidirectional Reflectance Distribution Function
BTDF	Bidirectional Transmittance Distribution Function
MERL	Mitsubishi Electric Research Laboratories
GCC	Gnu Compiler Collection
UWP	Universal Windows Platform
MVC	Model View Controller
GCD	Grand Central Dispatcher
UI	User Interface
XAML	eXtensible Application Markup Language
CI	Continuous Integration
LOC	Lines Of Code
AABB	Axis Aligned Bounding Box

Symbols

λ	Wavelength
$E(\lambda)$	Illuminant SPD
$S(\lambda)$	Generic object spectrum
$CMF(\lambda)$	Color matching function
$\bar{x}(\lambda), \bar{y}(\lambda), \bar{z}(\lambda)$	Color matching function component
$V(\lambda)$	Photopic luminous efficiency function
X, Y, Z	Tristimulus value (CIE 1931 color space)
x, y, z	XYZ chromaticity coordinates
u, v	CIE 1960 color space chromaticity coordinates
U^*, V^*, W^*	CIE 1964 color space coordinates
L, a, b	CIE 1976 color space coordinates
$u_{c,i}, v_{c,i}$	Von Kries CAT components
ϕ	Radiant flux
E	Irradiance
L	Radiance
p	Point
ω_i	Incident light direction (vector)
ω_o	Outgoing light direction (vector)
$L_i(p, \omega)$	Incident radiance
$L_o(p, \omega)$	Exitant radiance
$L_e(p, \omega_o)$	Emitted radiance
$f_r(p, \omega_i, \omega_o)$	Bidirectional Reflectance Distribution Function
$f_t(p, \omega_i, \omega_o)$	Bidirectional Transmittance Distribution Function

$\omega_i(\theta_i, \phi_i)$	Spherical coordinates of incident light direction
$\omega_o(\theta_o, \phi_o)$	Spherical coordinates of outgoing light direction
R	Reflectance spectrum
δ	Dirac delta function
η	Index of refraction
F_r	Fresnel reflection
X	Random variable
$P(x)$	Cumulative Distribution Function
$p(x)$	Probability Density Function
$E_p[f(x)]$	Expected value of a function
F_N	Monte Carlo estimator
R_i	Special CRI
R_a	General CRI

Dedicated to my family

Dedicated to Pierantonio

Chapter 1

Introduction

Physically based rendering (PBR) is one of the most advanced and trending field in computer graphics. PBR uses physically correct lighting and shading models to treat light as it behaves in the real world. As a consequence of the fact that what could be seen in a computer graphics application like a videogame, a CAD, a medical model, is decided by how light is represented, with PBR it is possible to reach a new level of realism.

In this thesis, a new PBR engine has been created from the ground up to explore some of the possibilities of PBR and colorimetry in computer graphics rendering: Spectral Clara Lux Tracer (SCLT).

SCLT is a cross-platform ray tracing engine that supports multiple physically based lighting and shading models. It is also entirely based on spectral distribution data, to achieve the maximum level of color fidelity.

What's ray tracing? Ray tracing is a computer graphics technique that could generate realistic images by tracing the paths of light in a scene. From the definition it's easy to see that this technique combines well with PBR.

Its main drawback is that it requires a huge amount of computing power to generate a single frame.

In fact, ray tracing is usually not a real time rendering technique.

SCLT supports different types of ray tracing:

- Whitted model;
- Path tracing.

The first one is a classical ray tracing model. It can render materials like glass or mirror, which are difficult to obtain in the standard 3D real time pipeline. The second one is a technique able to

render scenes with a high level of realism. The scenes generated using SCLT with path tracing will show realistic physical phenomenon, difficult to obtain with the Whitted model.

As previously stated SCLT puts a lot of attention on the color fidelity of the objects rendered in a scene. All color calculations are based on spectral data and the CIE 1931 XYZ tristimulus values. In this way SCLT is able to represent accurately physical phenomena strictly correlated with color, like metamerism, which are difficult to obtain with the standard RGB color calculations.

SCLT also supports the calculation of the Color Rendering Index (CRI). This index describes the ability of a light source to accurately render all frequencies of its color spectrum when compared to a perfect reference light of a similar type. CRI is rated on a scale from 0 - 100. The lower the CRI, the less accurately colors will be reproduced [1]. So SCLT has the ability to evaluate how much accurately a light source is showing colors in a scene.

With these features SCLT is a perfect tool for industrial light design and production. In fact, it could be used to evaluate the quality of a new light product using the CRI. Light designers could also evaluate the color fidelity of their new light products on rendered scenes using PBR and spectral data to get the most accurate results. As a cross platform engine, with support for Apple iOS/OS X devices and PC with Microsoft Windows 10, gives to the user the freedom to choose the preferred platform without any restriction.

What is the state-of-the-art of PBR and ray tracing? How does SCLT compare to other physically based ray tracer engine?

PBR and ray tracing are already used in conjunction in production environment. For example, the two most famous animation studios, Walt Disney Animation Studio and Pixar, use PBR and ray tracing for the production of their animation movies.

In particular, Walt Disney Animation Studio is using its own in-house PBR path tracer to produce its latest animation film since 2014: *Hyperion* [2] [3]. Obviously, this rendering engine is well optimized and implements new innovative techniques [2]:

Hyperion handles several million light rays at a time by sorting and bundling them together according to their directions. When the rays are grouped in this way, many of the rays in a bundle hit the same object in the same region of space. This similarity of ray hits then allows us – and the computer – to optimize the calculations for the objects hit.

Pixar also improved its rendering engine, RenderMan. In fact, it now includes a new rendering framework, RIS [4], based on ray tracing and optimized for PBR.



Figure 1. Big hero 6, the first Walt Disney animation movie produced with Hyperion.



Figure 2. Monsters University, the first Pixar movie produced with RIS.

At the moment, SCLT could not be compare with these engines, as they are the result of years of development by the teams of the two animation studios.

SCLT couldn't be compared also with other performance orientated rendering engine. Even if, as previously described, ray tracing is usually not a real time technique, some almost real-time ray tracer engine have been created thanks to the research progress in computer graphics. One of them is Brigade, an impressive real time path tracing engine [5]. On the other hand, SCLT takes a lot of hours to render a single image of a scene with medium complexity.

SCLT could be compared more accurately with another engine available on the market: LuxRender [6], an open source physically based rendering engine. This engine, like SCLT, uses physically correct lighting and materials models and has many other features: tone mapping, queueing and network rendering, multiple camera models, various effects like depth of field and motion blur. It is distributed as a cross platform application available for Microsoft Windows, Apple OS X and Linux operating systems.

In fact, LuxRender, like SCLT, is born from the study of the most famous open source PBR engine available: PBRT by Matt Phar and Greg Humphreys [7].

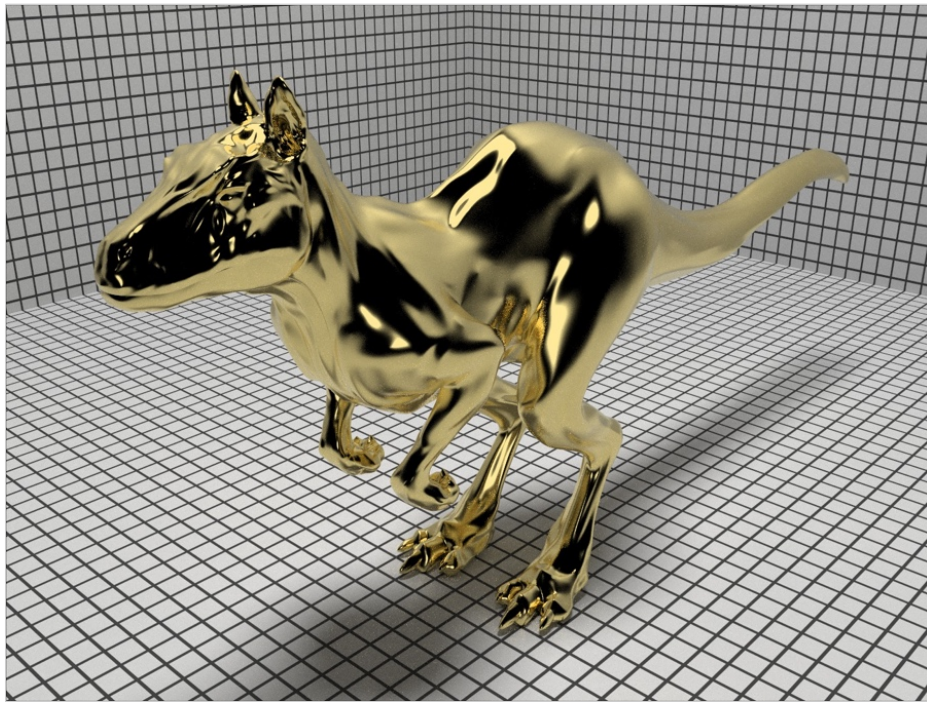


Figure 3. Example of a scene rendered using PBRT [8].

One of the key feature of SCLT that distinguish it from the other engines is the CRI: none of the previously cited engines support its calculation.

More in details, the main features of SCLT are:

- rendering of scenes using PBR and empirical lighting models. In this way it is possible to compare physically based spectral scenes with standard empirical RGB scenes. In particular, for PBR, SCLT implements the following BRDF models: Lambertian, Oren-Nayar [9], Torrance-Sparrow [10], specular reflection, specular transmission and measured BRDF [11] [12];
- support for different ray tracing techniques:
 - Whitted ray tracing, created by Turner Whitted in 1979 [13] [14], a technique where different kinds of rays are traced: reflection rays for mirror surfaces, refracted rays for transmissive surfaces (e.g. glass) and shadow rays. It is implemented using PBR and empirical models;
 - Path Tracing, a technique presented by J. T. Kajiya in 1986 [15]. It uses Monte Carlo integration method to find a numerical solution to the rendering equation previously described. It is implemented using only PBR models;
- color calculation using spectral data and the CIE XYZ color space in PBR scenes, to achieve high color fidelity;
- calculation of the CRI for light sources contained in PBR scenes, using two methods: Test samples method (CIE Commission Internationale de l'Eclairage, [16]) and R96_a (Bodrogi, 2004 [17]);
- cross-platform support: SCLT is a native application for any Apple iPad device starting from iPad 2 (iOS 9.0), any Apple computer that supports OS X El Capitan 10.11 and any PC that supports Windows 10.

In the following chapters a brief introduction to the fundamental concepts related to physically based rendering, radiometry, colorimetry and ray tracing will be showed. Then a detailed description of SCLT will be presented, focusing on:

- its general implementation and architecture;
- its main features previously listed;
- a series of case studies showing some rendered scenes obtained from SCLT and the CRI calculation on different type of lights.

SCLT is an education tool and an open source project: all the source code is available in a public Github repository under MIT License (<https://github.com/chicio/Spectral-Clara-Lux-Tracer>), and could be explored and improved by everyone.

Chapter 2

Principles of colorimetry and radiometry

In this chapter the main principles of colorimetry and radiometry will be described. They constitute the basis for some of the main key features of SCLT. In particular, color calculation and the lighting/shading techniques implemented in SCLT use the notions presented in the following paragraphs.

2.1 Colorimetry

Color is a perception tightly correlated to the observer that try to describe it. A lot of factors can influence how it is perceived: age, fatigue, physiological factors.

As a sensation, color is difficult to be described and measured. This is why colorimetry has been created.

Colorimetry is the science that try to physically describe the perception of human color. The main authority in this field of study is the Commission Internationale de l'Eclairage, usually abbreviated CIE. It was established in 1913, and since then it was responsible for defining and specifying new colorimetry standard via its publications. One of its main purpose is to the define organization of colors: color spaces. In 1931 CIE defined the basis of colorimetry using tristimulus values: the CIE XYZ color space.

2.1.1 Tristimulus values: CIE 1931 XYZ color space

As previously described, color is a perception.

Specifically, human eyes are sensitive to a small part of the entire electromagnetic radiation spectrum, in a range of wavelength from 380 nm to 780 nm. They contain three different kind of photoreceptor in the retina, called cone cells, with sensitive peak in different range of spectrum:

short wavelength cone S, middle wavelength cone M and long wavelength cone L (Watanabe et al., 2013 [18]). The range of electromagnetic radiation spectrum previously described is usually called visible light.

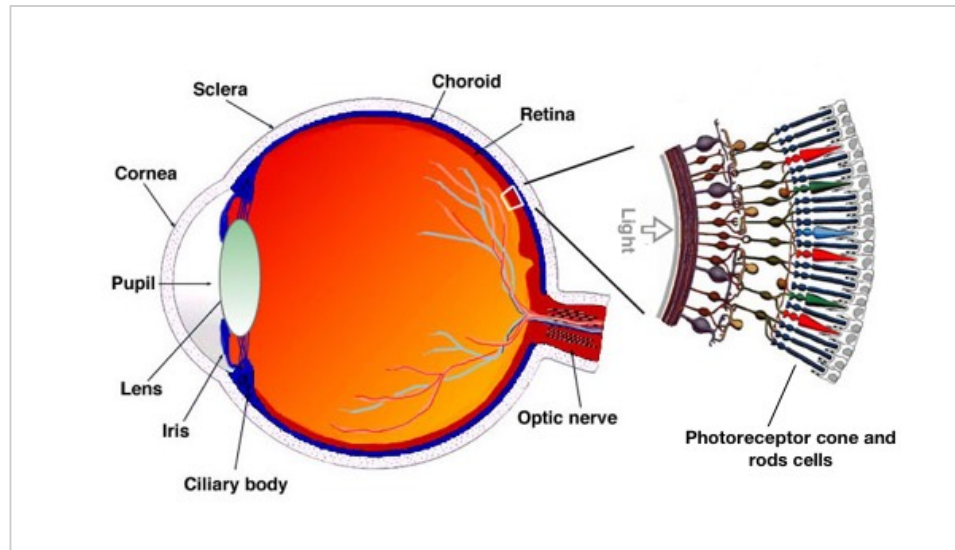


Figure 4. Eye and photoreceptor cone cells anatomy [19].

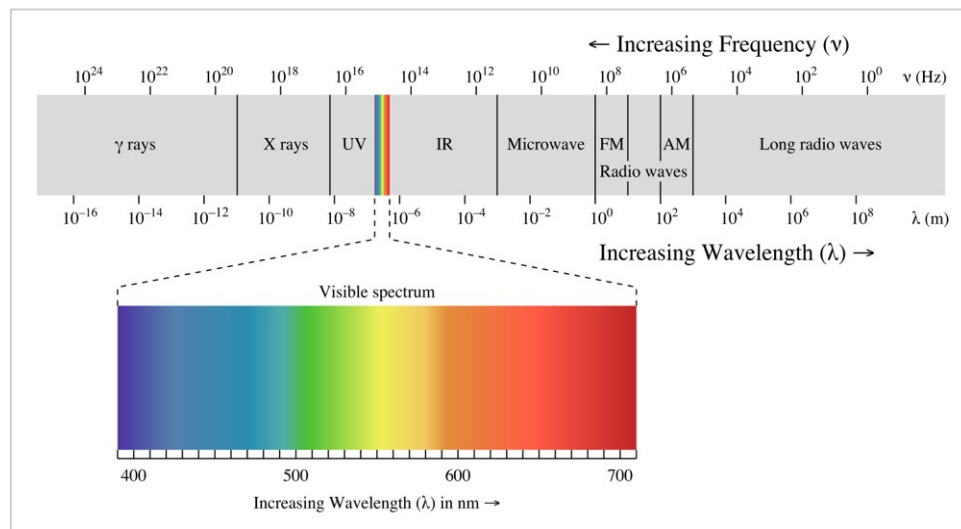


Figure 5. Visible light: a small part of the electromagnetic radiation spectrum [20].

Color could be represented using just three values. This is the first of the three principles described by Hermann Grassman's laws [21]:

- all color could be specified using three different independent variables;
- stimuli with different spectral composition could produce the same color match. These are called metamerism;
- If one component of the color change, it must change accordingly.

CIE used these laws the spectral power distribution (SPD) of lights and the spectrum of objects to defined the CIE tristimulus values and CIE XYZ color space, to identify a color numerically. These are defined as the integration of the product of an illuminant SPD $E(\lambda)$, an object spectrum $S(\lambda)$ and the color matching functions $CMF(\lambda) = \{\bar{x}(\lambda), \bar{y}(\lambda), \bar{z}(\lambda)\}$ multiplied by a normalizing factor (that is choose to give 1 for luminance Y) (Kang, 2007 [22]).

$$X = \frac{1}{\int \bar{y}(\lambda) E(\lambda) d\lambda} \int \bar{x}(\lambda) E(\lambda) S(\lambda) d\lambda \quad (1a)$$

$$Y = \frac{1}{\int \bar{y}(\lambda) E(\lambda) d\lambda} \int \bar{y}(\lambda) E(\lambda) S(\lambda) d\lambda \quad (1b)$$

$$Z = \frac{1}{\int \bar{y}(\lambda) E(\lambda) d\lambda} \int \bar{z}(\lambda) E(\lambda) S(\lambda) d\lambda \quad (1c)$$

A spectral power distribution is a distribution function of wavelength that describes the amount of light at each wavelength.

Object spectrum could be obtained using specific instruments, for example a spectrophotometer. Color matching functions are also called CIE standard observers. They are the numerical representation of the chromatic response of a human observer. These functions are obtained with an experiment: and observer must match a test sample with three matching stimuli $r(\lambda)$, $g(\lambda)$ and $b(\lambda)$. The values obtained are then scaled with the photopic luminous efficiency function $V(\lambda)$ established by CIE in 1924 (Kang, 2007 [22]).

Actually, two type of standard observer are available:

- 2° standard observer, also called CIE 1931 Standard Observer, that represent a human observer with a 2° field of view, with matching stimuli at 700, 546.1 and 435.8 nm;

- 10° standard observer, also called CIE 1964 Standard Observer, that represent a human observer with a 10° field of view, with matching stimuli at 645.2, 526.3 and 444.4 nm.

Illuminant SPD usually describes the spectral composition of a light source. Multiple light sources SPD are available and will be discussed in section 2.1.5.

Spectrum and SPD are usually represented as a set of sample over the visible light wavelength range with a specific sampling rate. The most used sampling rate is 5 nm. So the integrals in formula (1) could be calculated with a Riemann sum:

$$X = \frac{1}{\sum \bar{y}(\lambda) E(\lambda) \Delta\lambda} \sum \bar{x}(\lambda) E(\lambda) S(\lambda) \Delta\lambda \quad (2a)$$

$$Y = \frac{1}{\sum \bar{y}(\lambda) E(\lambda) \Delta\lambda} \sum \bar{y}(\lambda) E(\lambda) S(\lambda) \Delta\lambda \quad (2b)$$

$$Z = \frac{1}{\sum \bar{y}(\lambda) E(\lambda) \Delta\lambda} \sum \bar{z}(\lambda) E(\lambda) S(\lambda) \Delta\lambda \quad (2c)$$

where $\Delta\lambda$ is the sampling interval. When it is constant, it corresponds to:

$$\Delta\lambda = \frac{\lambda_t}{(n - 1)} \quad (2d)$$

where λ_t is the total range of spectrum and n is the number of samples.

2.1.1.1 CIE 1931 XYZ Chromaticity coordinates

Color has three main properties: hue, chroma and luminance.

Hue is how color is perceived: red, orange, green, blue and so on. Chroma (sometimes called also saturation) describes how much close a color is to gray or the pure hue. Luminance is related to the degree of lightness of a color.

Sometimes it is useful to represent a color regardless of its luminance, focusing only on hue and chroma property.

A color in CIE XYZ could be specified only for this two properties using the chromaticity coordinate, that are defined as a normalization of tristimulus values:

$$x = \frac{X}{X + Y + Z} \quad (3a)$$

$$y = \frac{Y}{X + Y + Z} \quad (3b)$$

$$z = \frac{Z}{X + Y + Z} \quad (3c)$$

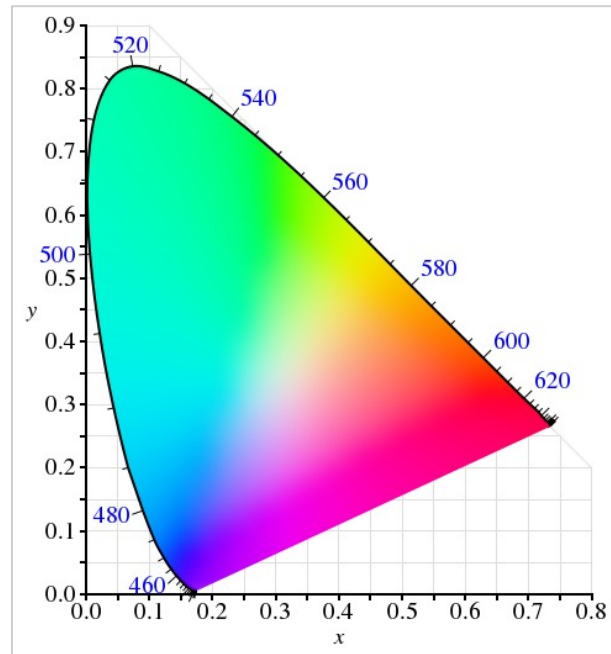


Figure 6. CIE 1931 XYZ chromaticity diagram [23].

2.1.2 CIE 1960 UCS color space

The CIE 1960 color space, also called Uniform Color Space, is a chromaticity space that is focused, like the chromaticity coordinates of the CIE XYZ, on hue and chroma properties of a

color. It was proposed by David L. MacAdam in 1937 [24], and it is widely used to calculate the correlated color temperature, described in section 2.1.5.

The formula used to convert CIE XYZ tristimulus values to CIE 1960 UCS is the following:

$$u = \frac{4X}{X + 15Y + 3Z} \quad (4a)$$

$$v = \frac{6X}{X + 15Y + 3Z} \quad (4b)$$

The formula to convert from CIE XYZ chromaticity coordinates to CIE 1960 UCS is:

$$u = \frac{4x}{12y - 2x + 3} \quad (5a)$$

$$v = \frac{6x}{12y - 2x + 3} \quad (5b)$$

2.1.3 CIE 1964 UVW color space

In 1963 Günther Wyszecki created the CIE 1964 UVW color space [25]. It is based on the previously described CIE 1960 UCS color space. The formulas used to obtain the coordinates are:

$$U^* = 13W^*(u - u_0) \quad (6a)$$

$$V^* = 13W^*(v - v_0) \quad (6b)$$

$$W^* = 25Y^{1/3} - 17 \quad (6c)$$

U^* and V^* are a transformation of the CIE XYZ chromaticity coordinates such that a white point with CIE XYZ chromaticity coordinates (u_0, v_0) maps to the origin. W^* correspond to the luminance index.

2.1.4 CIE 1976 L* a* b* color space

The CIE L* a* b* color space is based on the CIE XYZ color space. L^* defines the lightness, a^* defines the red/green value and b^* defines the yellow/blue value [26]. The a^* and b^* have no specific range, while L^* is defined in a range 0 - 100.

This color space is able to represent the entire human color vision. In fact, it is usually compared with the CIE XYZ color space.

The formula used to convert coordinates from CIE XYZ to CIE L* a* b* is [27]:

$$L = 116 f(Y/Y_r) - 16 \quad (7a)$$

$$a = 500(f(X/X_r) - f(Y/Y_r)) \quad (7b)$$

$$b = 200(f(Y/Y_r) - f(Z/Z_r)) \quad (7c)$$

In the previous formula (X_r, Y_r, Z_r) correspond to the tristimulus values of a reference white point, a set of values that represent the white color for a specific environment.

$f(a)$ is defined as:

$$f(x) = \begin{cases} \sqrt[3]{a} & a > \varepsilon \\ \frac{\kappa a + 16}{116} & a \leq \varepsilon \end{cases} \quad (8)$$

where κ e ε correspond in the CIE standard to the following value:

$$f(x) = \begin{cases} 0.008856 & \text{Actual CIE standard} \\ 216/24389 & \text{Intent of the CIE standard} \end{cases} \quad (9)$$

2.1.5 Standard Illuminants

As previously described in section 2.1.1, an illuminant represents a specific SPD that could be associated with a specific type of light source. CIE defined some families of standard illuminants. Here are reported the ones used by SCLT:

- illuminant A: it is equivalent to a bulb lighting (CIE Commission Internationale de l'Eclairage, 1999 [28]). Its SPD is calculated according to the Planck's radiation law:

$$E(\lambda, T) = \frac{c_1}{\lambda^5} \frac{1}{e^{(c_2/\lambda T)} - 1} \quad (10)$$

where $c_1 = 2\pi hc^2$ and $c_2 = hc/k$. The constant used in the formula above are: the Planck's constant $h = 6.62670 \times 10^{-34} J \cdot s$, the speed of light $c = 2.99792458 \times 10^8$ m/s and the Boltzmann's constant $k = 1.3806505 \times 10^{-23} J/K$;

- illuminants D: this family of illuminants are the mathematical representation of various phase of daylight (Judd et al., 1964 [29]). The most commons are: D65, which represent noon daylight, and D50, which represent horizon daylight;
- illuminants F: this family of illuminant embody various types of fluorescent light (CIE Commission Internationale de l'Eclairage, 2004 [30]). The most important illuminant of this family is FL4, used as reference to calibrate the CRI.

One useful property of illuminants is the Color Correlated Temperature (CCT): the color temperature of a black body radiator, an ideal physical body that absorb all incident light, that has almost the same color as the illuminant.

2.1.6 Macbeth Color checker

The Macbeth color checker chart is a set of colors which can be used as a reference for studies in the field of colorimetry.

It was originally name Color Rendition Chart, and it was originally described in an article by C. S. McCamy, H. Marcus and J. G. Davidson in 1976 [31]. They were colleagues at the Macbeth company. A few years ago this company has been acquired by X-Rite, another company specialized in everything related to color science.

It is composed of 24 elements. Spectral data of each element are available with sampling at 1 nm. It is also possible to obtain their coordinates for various color spaces under different illuminants. In table 1 are reported all the 24 elements under D50 illuminant (Pascale, 2006 [32]).




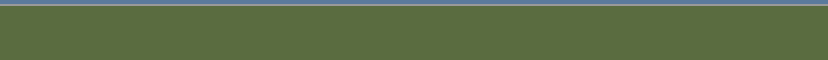















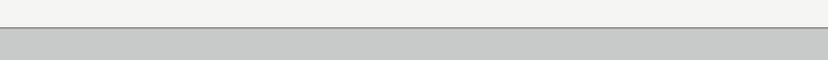
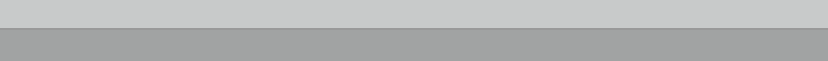



N.	Color name	Color
1	Dark skin	
2	Light skin	
3	Blue sky	
4	Foliage	
5	Blue flower	
6	Bluish green	
7	Orange	
8	Purplish blue	
9	Moderate red	
10	Purple	
11	Yellow green	
12	Orange yellow	
13	Blue	
14	Green	
15	Red	
16	Yellow	
17	Magenta	
18	Cyan	
19	White 9.5	
20	Neutral 8	
21	Neutral 6.5	
22	Neutral 5	
23	Neutral 3.5	
24	Black	

Table 1. Macbeth color checker under D50 illuminant.

2.1.7 CAT: Chromatic Adaptation transform

Chromatic adaptation is a visual mechanism that let the eyes adapt to changes in the spectral composition of the light. In this way human eyes are able to preserve the appearance of an object under different light conditions. A typical example is a piece of paper. Its appearance remains the same, white, under daylight, tungsten or fluorescent light (CIE Commission Internationale de l'Eclairage, 2004 [33]).

To simulate the same mechanism on digital devices, Chromatic Adaptation Transform (CAT), has been created. A CAT takes tristimulus values of a test sample under a test illuminant and calculates the corresponding tristimulus values under a reference illuminant. In the next sections two CAT are described: Von Kries Transform and CIECAT1994.

2.1.7.1 Von Kries Transform

Von Kries Transform was developed by Johannes Von Kries. This CAT is used in the calculation of the CRI using the “test method sample” described in section 6.1. The formula used to calculate it for a specific sample is:

$$u_{c,i} = \frac{10.872 + 0.404(c_r/c_t)c_{t,i} - 4(d_r/d_t)d_{t,i}}{16.518 + 1.481(c_r/c_t)c_{t,i} - (d_r/d_t)d_{t,i}} \quad (11a)$$

$$v_{c,i} = \frac{5.520}{16.518 + 1.481(c_r/c_t)c_{t,i} - (d_r/d_t)d_{t,i}} \quad (11b)$$

$$c = \frac{(4 - u - 10v)}{v} \quad (11c)$$

$$d = \frac{(1.708v - 1.418u + 0.404)}{v} \quad (11d)$$

The subscripts mean:

- r reference illuminant;
- t test illuminant;
- (t, i) test illuminant * sample;
- c current test color sample.

2.1.7.2 CIECAT94 Transform

The CIECAT94 is an evolution of the Von Kries transform that takes into account the luminance level. This CAT is used to calculate the CRI with the R96_a method described in section 6.2. Here is reported the standard procedure to calculate this CAT (CIE Commission Internationale de l'Eclairage, 2004 [33]) (Westaland et al., 2004 [34]). The subscripts stand for:

- w white (illuminant);
- rw reference white (illuminant);
- c test sample computed under reference white (illuminant).

The main three step are:

1. calculate the RGB values of the test sample under the test illuminant using the following matrix:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 0.40024 & 0.7076 & -0.08081 \\ -0.2263 & 1.16532 & 0.04570 \\ 0 & 0 & 0.91822 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \quad (12)$$

2. calculate the values R_c , G_c and B_c under the reference illuminant are computer using the equations:

$$R_c = (Y_o \xi_{rw} + n) K^{1/\beta_1(R_{rw})} \left[\frac{R + n}{Y_o \xi' + n} \right]^{\beta_1(R_w)/\beta_1(R_{rw})} - n \quad (13a)$$

$$G_c = (Y_o \eta_{rw} + n) K^{1/\beta_1(G_{rw})} \left[\frac{G + n}{Y_o \eta' + n} \right]^{\beta_1(G_w)/\beta_1(G_{rw})} - n \quad (13b)$$

$$B_c = (Y_o \zeta_{rw} + n) K^{1/\beta_2(B_{rw})} \left[\frac{B + n}{Y_o \zeta' + n} \right]^{\beta_2(B_w)/\beta_2(B_{rw})} - n \quad (13c)$$

3. calculate the tristimulus values adapted using the inverse of the matrix used to convert the values to RGB:

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \begin{bmatrix} 1.85995 & -1.12939 & 0.2199 \\ 0 & 0.63881 & 0 \\ 0 & 0 & 1.08906 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \quad (14)$$

The values ξ_{rw} , η_{rw} , ζ_{rw} and ξ_w , η_w , ζ_w correspond to the test and reference illuminants chromaticity correlates of the their chromaticity coordinates.

$$\xi = (0.48105x_0 + 0.78841y_0 - 0.080811)/y_0 \quad (15a)$$

$$\eta = (-0.27200x_0 + 1.11962y_0 + 0.04570)/y_0 \quad (15b)$$

$$\zeta = 0.91822(1 - x_0 - y_0)/y_0 \quad (15c)$$

The values ξ' , η' and ζ' are the adapting ξ , η and ζ values and are calculated:

$$\begin{bmatrix} \xi' \\ \eta' \\ \zeta' \end{bmatrix} = \alpha \begin{bmatrix} \xi_w \\ \eta_w \\ \zeta_w \end{bmatrix} + (1 - \alpha) \begin{bmatrix} \xi_{rw} \\ \eta_{rw} \\ \zeta_{rw} \end{bmatrix} \quad (16)$$

and α is:

$$\alpha = 0.1151 \log_{10} L_a + 0.0025(L^* - 50) + (0.22D + 0.510) \quad (17)$$

In this formula L_a is the adapting luminance (cd/m^2) in test field, and L^* the CIELAB lightness of the sample. For object colors $D = 1$, and for luminous color (typically CRT) $D = 0$. The α value must be less than one.

R_w , G_w , B_w and R_{rw} , G_{rw} , B_{rw} are calculated with the formulas reported below:

$$\begin{bmatrix} R_w \\ G_w \\ B_w \end{bmatrix} = L_a \begin{bmatrix} \xi_w \\ \eta_w \\ \zeta_w \end{bmatrix} \quad (18a)$$

$$\begin{bmatrix} R_{rw} \\ G_{rw} \\ B_{rw} \end{bmatrix} = L_{ra} \begin{bmatrix} \xi_{rw} \\ \eta_{rw} \\ \zeta_{rw} \end{bmatrix} \quad (18b)$$

L_{ra} is set to 63.66 cd/m².

β_1 and β_2 functions are:

$$\beta_1(I) = \frac{6.469 + 6.362I^{0.4495}}{6.469 + I^{0.4495}} \quad (19a)$$

$$\beta_2(I) = 0.7844 \left(\frac{8.414 + 8.09I^{0.5128}}{8.414 + I^{0.5128}} \right) \quad (19b)$$

Last but not least the formula to calculate K:

$$K = \frac{[(Y_0\xi' + n)/(20\xi' + n)]^{(2/3)\beta_1(R_r)}}{[(Y_0\xi_{rw} + n)/(20\xi_{rw} + n)]^{(2/3)\beta_1(R_{rw})}} \frac{[(Y_0\eta' + n)/(20\eta' + n)]^{(1/3)\beta_1(G_r)}}{[(Y_0\eta_{rw} + n)/(20\eta_{rw} + n)]^{(1/3)\beta_1(G_{rw})}} \quad (20)$$

The noise factor is 0.1 in all the formulas above.

2.2 Radiometry

As seen in the previous section, eyes are sensitive to a small subset of the entire electromagnetic radiation spectrum with wavelength between 400 nm and 700 nm, the visible light. Therefore, light is a form of electromagnetic radiation.

The set of studies and techniques that try to describe and measure how the electromagnetic radiation of light is propagated, reflected and transmitted is called radiometry.

In the next few paragraphs some basics quantities of radiometry are presented. These are a fundamental part of computer graphics, as most of the algorithm used in this field, and presented in this thesis, try to estimate these quantities.

2.2.1 Radiant flux

Sometimes simply called flux, it describes the amount of radiant energy emitted, reflected or transmitted from a surface per unit time. The radiant energy is the energy of an electromagnetic radiation. The unit measure of flux is joules per seconds $\frac{J}{s}$, and it is usually reported with the Greek letter ϕ .

2.2.2 Irradiance and radiant exitance

Other two important quantities of radiometry are irradiance and radiant exitance. The first one described flux arriving at a surface per unit area. The second one describe flux leaving a surface per unit area (Pharr et al., 2010 [35]).

Formally irradiance is described with the following equation:

$$E = \frac{d\phi}{dA} \quad (21)$$

where the differential flux is computed over the differential area. It is measured as units of watt per square meter $\frac{W}{m^2}$.

2.2.3 Radiance

Radiance is the last and most import quantity of radiometry. To really understand its definition, it is useful to give the definition of solid angle.

A solid angle is an extension of a 2D angle in 3D on a unit sphere. It is the total area projected by an object on a unit sphere centered at a point p . It is measured in steradians. The entire unit sphere corresponds to a solid angle of 4π (the surface area of the unit sphere). A solid angle is usually indicated as Ω , but it is possible also to represent it with ω , that is the set of all direction vectors anchored at p that point toward the area on the unit sphere and the object (Pharr et al., 2010 [35]). Now it is possible to give the definition of radiance, that is flux density per unit solid angle per unit area:

$$L = \frac{d\phi}{d\omega dA^\perp} \quad (22)$$

In this case dA^\perp is the projected area dA on a surface perpendicular to ω . So radiance describe the limit of measurement of incident light at the surface as a cone of incident directions of interest $d\omega$ becomes very small, and as the local area of interest on the surface dA also becomes very small (Pharr et al., 2010 [35]).

It is useful to make a distinction between radiance arriving at a point, usually called incident radiance and indicated with $L_i(p, \omega)$, and radiance leaving a point called exitant radiance and indicated with $L_o(p, \omega)$. This distinction will be used in the equations described in the next chapters. It is important also to note another useful property, that connect the two types of radiance:

$$L_i(p, \omega) \neq L_o(p, \omega) \quad (23)$$

Chapter 3

Ray tracing: photorealistic rendering

Ray tracing is one of the most powerful computer graphics technique. It is able to generate images that expose a visual realism difficult to obtain with the standard 3D real-time rendering pipeline. How does it work?

The general ray tracing algorithm is based on the fact that, in the real world, light starting from a source bounces between objects surfaces and arrives to the human eyes that, as previously seen, decode it to get the color of objects.

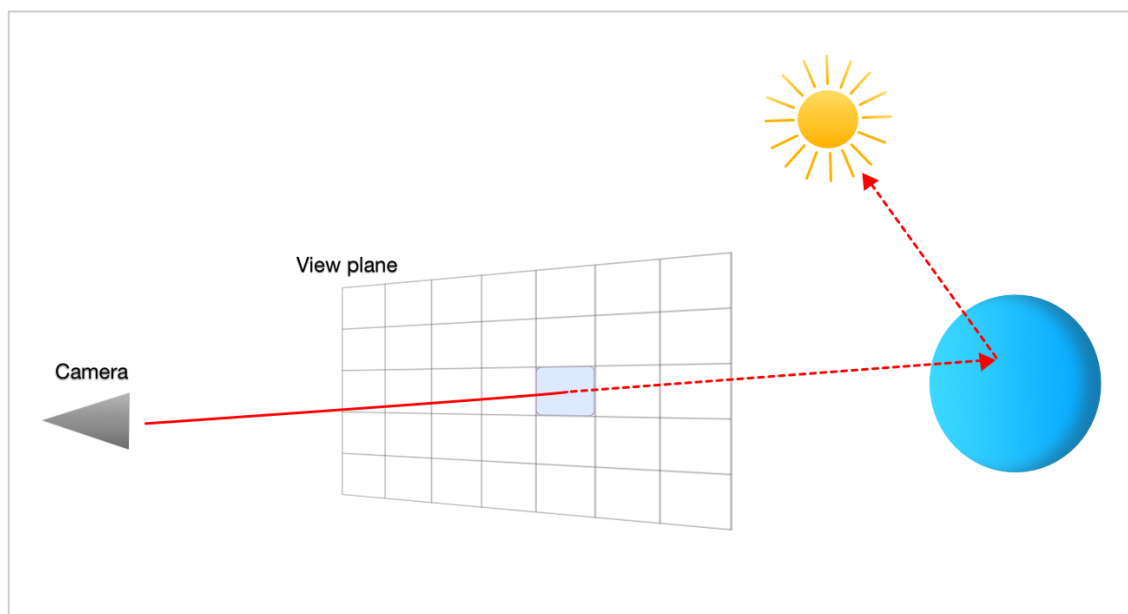


Figure 7. How works a general ray tracer.

The ray tracing algorithm follows an inverse path: starting from the eyes, usually defined as a camera widely used in computer graphics, the algorithm traces rays of light, that bounce in the scene and arrive to the light source, and calculate the color of objects on which the rays bounced using a surface reflection model. These models will be described in section 3.3. As it follows all the various paths of light, direct and indirect, ray tracing, with all its variants, is a global illumination model.

In the next paragraph a brief classification of ray tracing algorithms will be introduced. Only a subset of them is implemented in SCLT: Whitted and path tracing. But before starting with the classification, it is useful to defined what ray tracing (and all the global illumination models) try to solve: the rendering equation.

3.1 The Rendering equation

The rendering equation was introduced by James Kajiya in 1986 [15]. Sometimes it is also called the LTE, Light Transport Equation. It is the equation that describes the equilibrium distribution of radiance in a scene (Pharr et al., 2010 [36]).

It gives the total reflected radiance at a point as a sum of emitted and reflected light from a surface. This is the standard formulation of the rendering equation:

$$L_o(p, \omega_o) = L_e(p, \omega_o) + \int_{\Omega} f_r(p, \omega_i, \omega_o) L_i(p, \omega_i) \cos \theta_i d\omega_i \quad (28)$$

where the meaning of each component is

- p is a point on a surface in the scene;
- ω_o is the outgoing light direction;
- ω_i is the incident light direction;
- $L_o(p, \omega_o)$ is the exitant radiance at a point p ;
- $L_e(p, \omega_o)$ is the emitted radiance at a point p ;
- Ω is the unit hemisphere centered around the normal at point p ;
- $\int_{\Omega} \dots d\omega_i$ is the integral over the unit hemisphere;
- $f_r(p, \omega_i, \omega_o)$ is the Bidirectional Reflectance Distribution Function. A detail description of this function, and its correlated Bidirectional Transmittance Distribution Function, and

the various models that are used to calculate it are described in section 3.3. Their implementation in SCLT will be discussed in chapter 5;

- $L_i(p, \omega_i)$ is the incident radiance arriving at a point p ;
- $\cos \theta_i$ is given by the dot product between ω_i and the normal at point p , and is the attenuation factor of the irradiance due to incident angle.

The purpose of all global illumination models, so also ray tracing with all its variants, is to solve this equation.

3.2 A brief classification of ray tracing techniques

The first basic type of ray tracing presented is what is usually called Whitted ray tracing. This model was invented, as the name suggest, by Turner Whitted in 1979 [13] [14]. It assumes that when a ray, that starts from the camera, intersects an object, a local surface reflection model is used to calculate the direct light component. But the algorithm goes on, and three type of rays are traced: reflection ray, refraction ray and shadow ray. As the name suggest, the first is used for reflective material, the second one is used for refractive material, and the third is used to calculate if a point is not directly reached from light, so it is shadowed.

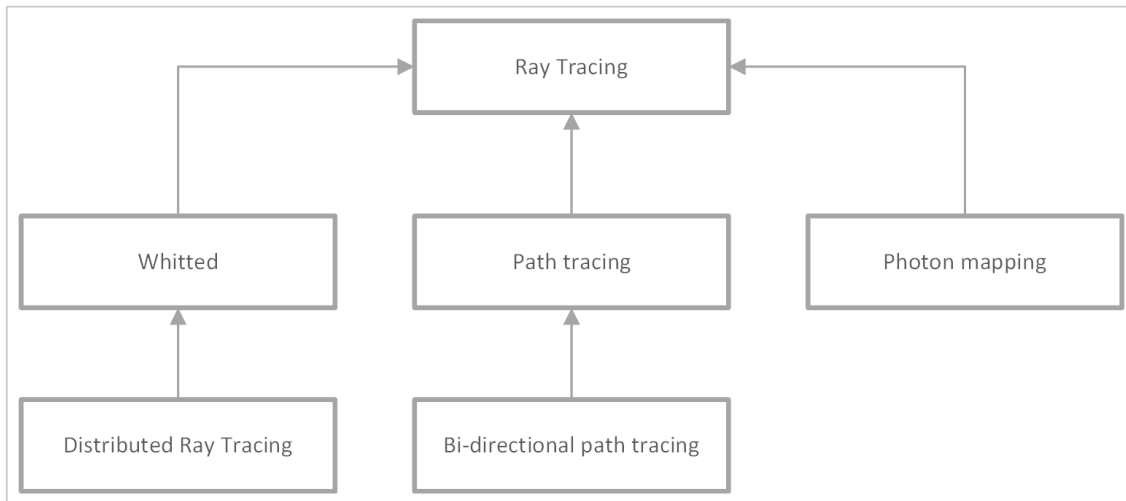


Figure 8. Ray tracing technique classification.

The Whitted ray tracing is not the most realistic ray tracing technique. Other effects and improvements could be added as a consequence of tracing more and more rays. First of all, more shadow rays could be traced for each point into the scene.

Instead of checking for their intersection with a single point on the light surface, usually its center (or the origin in case of a point light), multiple points could be sampled uniformly and a shadow ray test is executed for each point: in this way it is possible to calculate the percentage of shadow in a single point and generate what is usually called soft shadow.

Tracing more rays of different types, other effects could be generated like antialiasing, depth of field, glossy reflection. The set of all this technique is usually called distributed ray tracing.

Another technique that improve the visual realism is photon mapping. In this technique photons are traced in the scene. The information collected are then used in addition to another ray tracing technique to calculate the final rendered scene.

3.2.1 Path tracing

The path tracing is the technique with the maximum level of visual fidelity with respect to all the ray tracing techniques available. It was presented by James Kajiya in 1986 [15] as a solution to the rendering equation presented in section 3.1. It uses Monte Carlo integration technique (Pharr et al., 2010 [36]) to give a numerical solution to this equation.

More details about this technique will be explained in the next chapters. Here a general description of the algorithm will be presented.

For each pixel a number of samples is taken. Each one of these samples is calculated tracing a ray from the camera and following its bounces in the scene until it reaches a light source. Then these samples are used with the Monte Carlo technique to calculate each pixel's color in the final image. This technique is the only one that is able to generate a lot of visual effects without the need to trace secondary rays. It also has the ability to naturally simulate some physical phenomenons that usually require specific techniques to be rendered. For example, caustic could be generated without extra calculation.

A variant of path tracing is bi-directional path tracing (Lafortune et al., [37]). In this technique the rays are traced in two directions: from the camera to the light source, and from the light source to the camera. In this way the scene could be rendered with a high level of visual realism at a faster speed than path tracing.

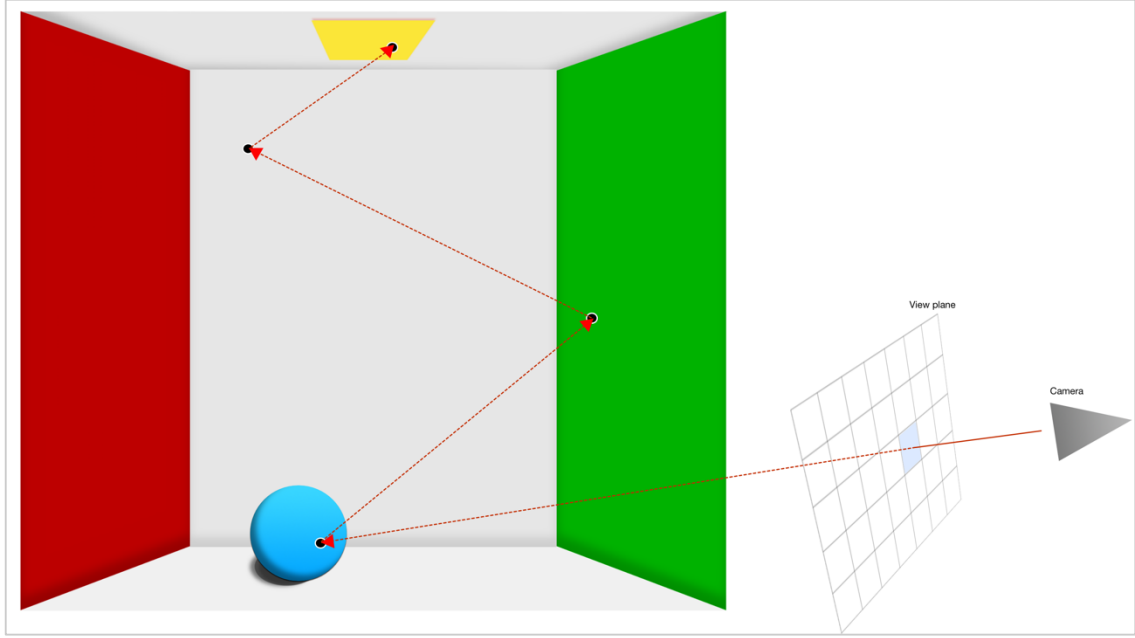


Figure 9. Path tracing follows the ray's path from camera to light source.

3.3 BRDF and BTDF

One of the main component of the rendering equation described in section 3.1 is the Bidirectional Reflectance Distribution Function (BRDF). This function describes how light is reflected from a surface. It represents a constant of proportionality between the differential exitant radiance and the differential irradiance at a point p (Pharr et al., 2010 [35]). The parameter of this function are: the incident light direction, the outgoing light direction and a point on the surface.

The formula for this function is the following:

$$f_r(p, \omega_i, \omega_o) = \frac{dL_o(p, \omega_o)}{dE(p, \omega_i)} \quad (29)$$

The BRDF has two important properties:

1. it is a symmetric function, so for all pair of directions $f_r(p, \omega_i, \omega_o) = f_r(p, \omega_o, \omega_i)$;
2. it satisfies the energy conservation principle: the light reflected is less than or equal to the incident light.

Some specific material surfaces, for example glass, reflect and transmit light at the same time. So a fraction of light goes through the material. For this reason, there's another function, the Bidirectional Transmittance Distribution Function, BTDF, defined in the same way as the BRDF, but with the directions ω_i and ω_o placed in the opposite hemisphere around p (Pharr et al., 2010 [35]). It is usually indicated as $f_t(p, \omega_i, \omega_o)$.

How are the BRDF and BTDF calculated? A set of models has been developed, each one representing a specific property of different materials.

These models use the incident light direction ω_i and the outgoing light direction ω_o in spherical coordinate $\omega_i(\theta_i, \phi_i)$, $\omega_o(\theta_o, \phi_o)$, relative to the tangent space for a surface point, where the normal is aligned with the y axis. This happens because the BRDF is defined in a coordinate reference system local to the point on the surface considered, so the tangent space is an obvious choice.

Further details about spherical coordinate calculation are contained in appendix A.

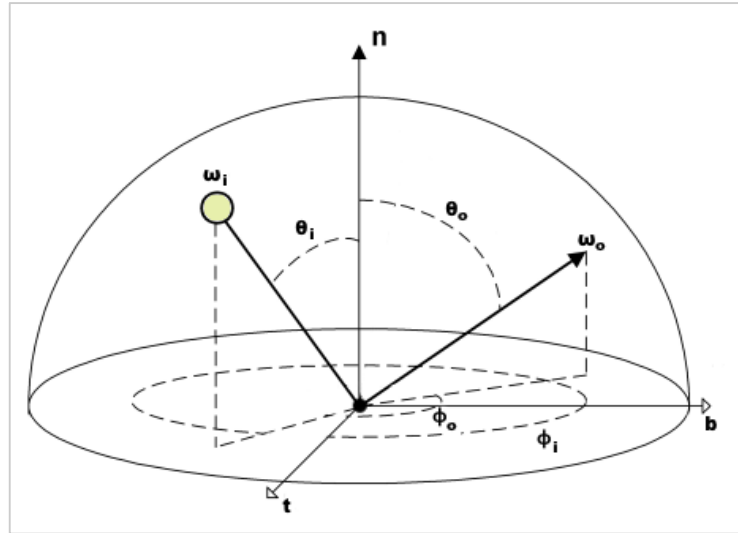


Figure 10. BRDF representation diagram [38].

In the following paragraphs a theoretical background of some of these models is showed. These are the ones implemented in SCLT. In this way it will be more easy to understand the implementation of a part of its core functionalities.

3.3.1 Empirical models

The first two models presented are Phong and Blinn-Phong. They are called empirical because they try to describe how a surface reflects light as an approximation of observed phenomenon.

3.3.1.1 Phong model

The Phong model was proposed by Bui Tuong Phong in 1973 [39].

It describes surface light reflection as a sum of different components that try to account for the possible light behavior on a material, using the following parameters:

1. K_e that described the emissive component of the material;
2. K_a that described the ambient component of the material;
3. K_d that described the diffuse component of the material;
4. K_s that described the specular component of the material.

The equation to compute the amount of light at a specific point p on a surface in a scene with j light sources is given by the following formula:

$$I_p = K_e I_{max} + K_a I_{amb} + \sum_{i=1}^j K_d I_{i,d}(l_i \cdot n) + K_s I_{i,s}(r_i \cdot v)^n \quad (30)$$

The term $(l_j \cdot n)$ is the attenuation term of the diffuse component that depends on the cosine of the incident angle of the light direction l_j with respect to the surface normal n , and is calculated as the dot product between these two vectors.

The term $(r_i \cdot v)^n$ controls the specular component and is the dot product of the reflection vector r_i and the view direction, described by the vector v . The first one is calculated with the following formula:

$$r_i = 2(n \cdot l_i) n - l_i \quad (31)$$

The vector v is calculated as a difference between the point of view of the camera and the point p . The exponent of the specular control term is used to control the size of the specular highlight.

This model is usually used in Whitted ray tracing to render scene defined with RGB values. For this reason, in SCLT, as will be described later, the Phong model will be use to explore the rendering of images with standard RGB colors.

3.3.1.2 Blinn-Phong model

The Blinn-Phong model is an evolution of the Phong model, proposed by James Blinn in 1977 [40].

In particular, the specular component is approximated using the angle γ between the normal and the half vector h of the observer direction vector v and light direction vector l . So h is calculated with the following formula:

$$h = \frac{l + v}{|l + v|} \quad (32)$$

and the specular component is calculated as:

$$I_s = K_s I_s (h \cdot v)^n \quad (33)$$

As for the Phong model, also the Blinn-Phong model will be used in conjunction with the Whitted ray tracing for scene with standard RGB colors.

3.3.2 Physically based models

Physically based BRDF models are based on the laws of physics related to light interaction with different surfaces. In the following paragraphs the description of a series of physically based models is presented. They could be used to render a wide range of materials, including glass, plastic and diffuse surfaces. All these models are implemented in SCLT.

Last but not least, the models presented in this thesis are all isotropic: the surface reflectance properties don't change with respect to rotation of the surface around the surface normal. The models that account for a preferred tangent direction are called anisotropic. SCLT doesn't support any anisotropic model.

3.3.2.1 Lambertian model

The first physically based BRDF presented is the Lambertian model. It models a perfect diffuse surface that reflects incident light equally in all direction. Although not completely physically plausible (in fact sometimes it is included in the empirical models), it is a good approximation to many real surfaces.

Given the SPD of an object, declared here as R , the standard formula to calculate the Lambertian model is:

$$f_r(p, \omega_i, \omega_o) = \frac{R}{\pi} \quad (34)$$

The π term is used to ensure that the Lambertian model follows the principle of conservation of energy: in this way outgoing light distributed over all the hemisphere doesn't exceed incident light.

3.3.2.2 Oren-Nayar model

The Oren-Nayar model is used to represent diffuse surfaces. It was defined by Michael Oren and Shree K. Nayar in 1994 [9]. It is part of a large set of models named “Microfacets models”, because they describe surfaces as composed of a collection of small microfacets with specific geometric properties.

This model is more realistic than the Lambertian one. It accounts for the fact that rough surfaces appear brighter as the light direction approaches the viewing direction. It describes the surface as composed with a collection of v-shaped Lambertian facets, distributed with a Gaussian distribution with a parameter σ that described the standard deviation from the orientation angle (Pharr et al., 2010 [41]).

The model accounts for local lighting effects between microfacets, like shadowing, interreflection and masking, and it is described by the following formula (where R is the SPD of the object):

$$f_r(p, \omega_i, \omega_o) = \frac{R}{\pi} (A + B \max(0, \cos(\phi_i - \phi_o)) \sin \alpha \tan \beta) \quad (35)$$

Given σ in radians the various terms of the previous equation are defined as:

$$A = 1 - \frac{\sigma^2}{2(\sigma^2 + 0.33)} \quad (36)$$

$$B = \frac{0.45\sigma^2}{\sigma^2 + 0.09} \quad (37)$$

$$\alpha = \max(\theta_i, \theta_o) \quad (38)$$

$$\beta = \min(\theta_i, \theta_o) \quad (39)$$

3.3.2.3 Specular reflection and transmission model

Specular reflection BRDF model and specular transmission BTDF models describe a smooth surface that generates, as the name suggest, phenomenon of reflection and refraction of light.

For an incident light direction ω_i , light is reflected or refracted in a single direction ω_o , that must be calculated and used as a new ray to be traced in all the kinds of ray tracing algorithms previously described. For reflection, this direction is given by equation 31, the same used in Phong and Blinn Phong models. For transmission, the outgoing direction is given by Snell's law, that relates the angle between the normal and the transmission direction θ_t , and the angle between the normal and the incident light direction θ_i :

$$\eta_i \sin \theta_i = \eta_t \sin \theta_t \quad (40)$$

In the previous formula η is the refractive index of a material, and describe how much more slowly light travels through a material than in vacuum (Pharr et al., 2010 [41]). The formula used to compute the transmission direction is:

$$t = \omega_i \frac{\eta_i}{\eta_t} + \left(\frac{\eta_i}{\eta_t} \cos \theta_i - \sqrt{1 - \left(\frac{\eta_i}{\eta_t} \right)^2 (1 - \cos^2 \theta_i)} \right) n \quad (41)$$

Usually materials are never completely reflective or transmissive. They are described simultaneously with each one of the models in this paragraph. How is it calculated the distribution

between reflected and refracted light? The Fresnel equations are the answer. They describe the percentage of light reflected F_r from a surface. There are two kinds of Fresnel equations, one for dielectrics, i.e. materials that don't conduct electricity, and conductors. SCLT implements only dielectrics, glass in particular, so only the equation for these set of materials is described here. The formula for Fresnel dielectric reflection is:

$$F_r = \frac{(r_{\parallel}^2 + r_{\perp}^2)}{2} \quad (42a)$$

where the terms r_{\parallel}^2 and r_{\perp}^2 are calculated with the following equations:

$$r_{\parallel} = \frac{\eta_t \cos \theta_i - \eta_i \cos \theta_t}{\eta_t \cos \theta_i + \eta_i \cos \theta_t} \quad (42b)$$

$$r_{\perp} = \frac{\eta_i \cos \theta_i - \eta_t \cos \theta_t}{\eta_i \cos \theta_i + \eta_t \cos \theta_t} \quad (42c)$$

As a consequence of conservation of energy principle, the light transmitted from a material surface is:

$$F_t = 1 - F_r \quad (42a)$$

Recall that each one of the two models are described by a specific outgoing direction. This causes the BRDF and BTDF to be 0 everywhere except at the specular reflection or transmission direction. Which function could be used to describe this behavior? The Dirac delta function δ . This is a function that is zero everywhere except at zero, with an integral of one over the entire real line (Dirac, 1958 [42]). An important property of this function is that (Pharr et al., 2010 [41]):

$$\int f(x) \delta(x - x_0) dx = f(x_0) \quad (43)$$

Using this property, it is possible to solve the rendering equation and find the BRDF and BTDF value using the Dirac delta function centered at the specular reflection or refraction direction vector.

It is now possible to define the equations of the two models.

For specular reflection the BRDF is described by the following formula:

$$f_r(p, \omega_i, \omega_o) = R F_r \frac{\delta(\omega_i - r)}{|\cos \theta_i|} \quad (44)$$

where r is the specular reflection vector for ω_o described in equation 31. The extra cosine term at the denominator is needed to cancel out the one taken from the rendering equation.

For specular transmission the BTDF is described by the following formula, where t is the transmission vector for ω_o described in equation 41:

$$f_t(p, \omega_i, \omega_o) = R \left(\frac{\eta_i}{\eta_t} \right)^2 (1 - F_r) \frac{\delta(\omega_i - t)}{|\cos \theta_i|} \quad (45)$$

3.3.2.4 Torrance sparrow model

The Torrance-Sparrow model was developed by K. E. Torrance and E. M. Sparrow in 1967 [10]. It belongs to the microfacets models set, like Oren-Nayar. It models a surface as a collection of perfectly smooth microfacets. The equation that describes this model is:

$$f_r(p, \omega_i, \omega_o) = R \frac{D(\omega_h) G(\omega_o, \omega_i) F_r}{4 \cos \theta_o \cos \theta_i} \quad (46)$$

The term $D(\omega_h)$ is the distribution function that describes the probability that a microfacet has a specific orientation in relation to the half vector ω_h . Specifically, this model uses the Blinn distribution. This distribution is the same used in the empirical model previously presented, Blinn Phong, proposed by James Blinn in 1977 [40]. To ensure its physical correctness, the distribution is normalized, so that the integral over the projected area of all microfacets is equal to 1. Its equation is the following (Pharr et al., 2010 [41]):

$$D(\omega_h) = \frac{e + 2}{2\pi} (\omega_h \cdot n)^e \quad (47)$$

The term $G(\omega_o, \omega_i)$ is a geometric attenuation term that assumes that the microfacets are distributed along v-shaped grooves. The equation used to describe it is:

$$G(\omega_o, \omega_i) = \min \left(1, \min \left(\frac{2(n \cdot \omega_h)(n \cdot \omega_o)}{\omega_o \cdot \omega_h}, \frac{2(n \cdot \omega_h)(n \cdot \omega_i)}{\omega_o \cdot \omega_h} \right) \right) \quad (48)$$

The term F_r is the Fresnel equation term, the same used for dielectric previously presented for the specular reflection and transmission model.

3.3.2.5 Measured BRDF

BRDF could also be measured, and the data obtained could be used for rendering materials with different properties. The device used for measuring a BRDF is the gonioreflectometer.

Various BRDF data formats are available online. For example, there's MERL database with over a hundred of BRDF data ready to be used.

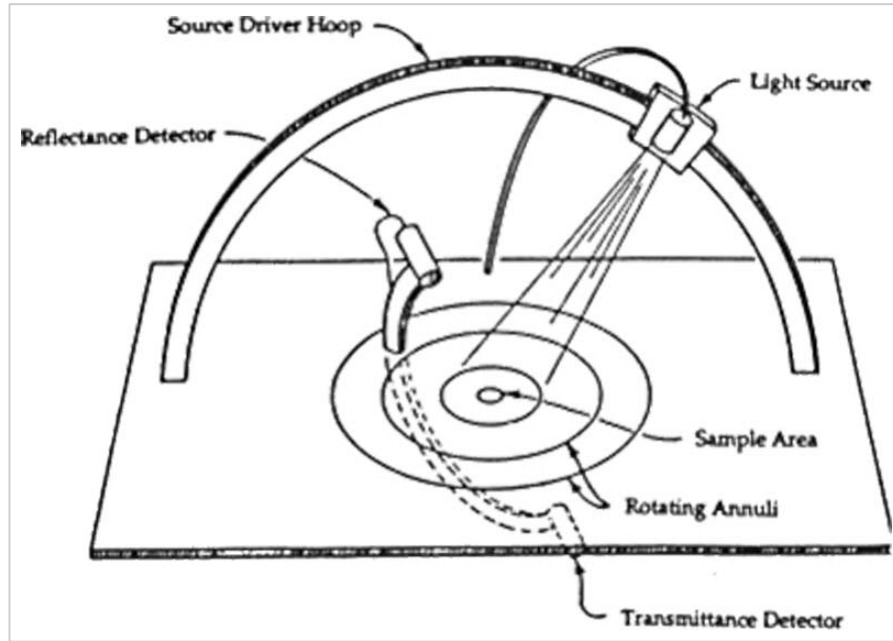


Figure 11. Operating principle of a gonioreflectometer [43].



Figure 12. An example of a gonioreflectometer [44].

This database stores data in RGB format only. SCLT uses measured BRDF only in conjunction with spectral data, to account for the maximum level of color fidelity.

This is why the measured BRDF data set chosen is the Cornell University Reflectance Data (Cornell Univ. Computer Graphics Dept., 1998 [12]).

This kind of irregular spectral data, sampled at 10 nm, is indexed with incident and outgoing light directions recorded as spherical coordinates. This data could be used with a simple nearest samples interpolation algorithm or with a specific data mapping.

A specific mapping could be useful to do a more precise selection of spectrum samples from the measured data for each direction. As a result, it is possible to avoid the generation of artifacts on the rendered surface. This is way SCLT uses a specific data mapping. Details will be given in the next chapters.

Chapter 4

SCLT: overview

All the notions, definitions and equations previously described constitute the foundation for the ray tracing engine developed for this thesis: SCTL, Spectral Clara Lux Tracer. All its source code is available in a public Github repository under MIT License (<https://github.com/chicio/Spectral-Clara-Lux-Tracer>).

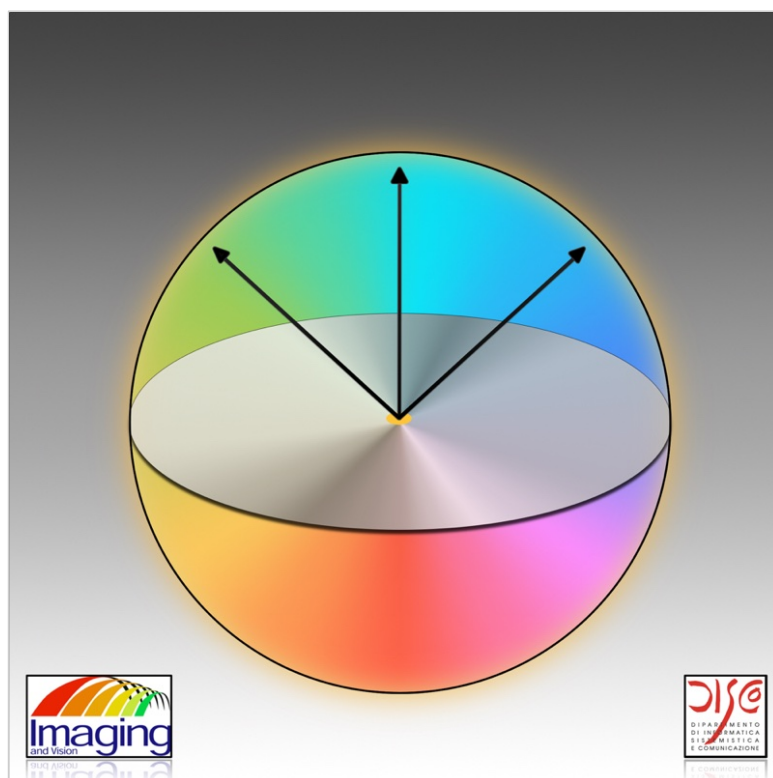


Figure 13. Spectral Clara Lux Tracer logo.

SCLT supports classical RGB rendering techniques with empirical BRDF models, but its key features are:

- physically based rendering, using physically based BRDF models and spectral data;
- calculation of CRI for the illuminant used in PBR scenes rendered with spectral data.

In this way SCLT is able to render different scenes with an obsessive focus on color fidelity and visual realism. In the following chapters all its features will be described, focusing on the implementation details. First of all, a general overview of SCLT architecture will be showed in the next paragraph.

4.1 Core architecture

As previously said, SCLT supports multiple types of rendering, based on different kinds of color data, standard RGB or spectral data. So it is a requirement that SCLT must be able to change the technique used to render a scene in a convenient way. This is also the reason why SCLT must adhere to the separation of concerns design principle, that requires the subdivision of a software into modules, each one accounting for a specific set of information and operations [45]. SCLT is composed of different modules, each one created to manage a specific part of the ray tracing rendering engine.

The main class that starts the rendering process is *SCLT* class. It contains the main loop of the ray tracing general algorithm. This loop goes through each pixel of the view plane and trace the current camera ray with a *Tracer* instance to obtain the color of the pixel (or a part of a pixel, in case of rendering with antialiasing) of the image with the method *getColor(Ray ray, int bounce)* that has two parameters: the first one is a ray to be traced, implemented in the *Ray* class and described by an origin and a direction, the second one is the maximum number of bounces that a ray is allowed to execute, and it is decrement on each bounce of a ray. These bounces could be generated for different reason: the bouncing of a ray between reflective/refractive material in a Whitted ray tracing model, or for the general number of maximum number of bounce allowed in path tracing. Obviously in *SCLT* class there's also the general setup needed to render a scene: the definition of the view plane (*ViewPlane* class) and the definition of the camera (*Camera* class).

Different kinds of tracer are implemented in SCLT, each one accounting for different features of the engine. There's a base *Tracer* class that implements a method common to all the tracer

specialization: *closestIntersection(Ray ray)*. This method is the one used to calculate the closest intersection of a ray with the objects in the scene. The nearest object will be the one to render. This is a standard method common to all ray tracing model implemented in SCLT. This is the reason why it is contained in this base class.

There are two specializations of the *Tracer* class, each one accounting for a different color rendering technique: *TracerRGB* and *TracerSpectrum*. As the name suggest the first one is used to render a scene using standard RGB data, the second one using spectral data.

In particular, the *TracerSpectrum* class uses the *CIE1931XYZ* class to compute the tristimulus values for spectral PBR scenes.

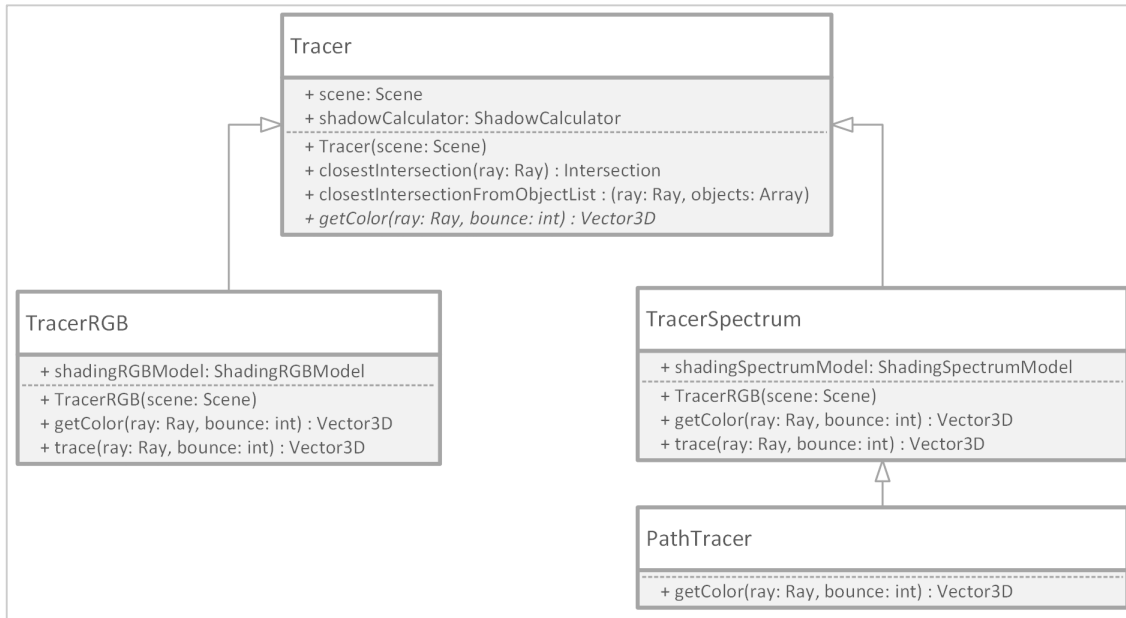


Figure 14. Tracer classes hierarchy - UML class diagram.

The *Tracer* subclasses implement two methods: the previously described *getColor(Ray ray, int bounce)* and the method *trace(Ray ray, int bounce)*. The last one is the method that effectively executes the trace of a ray.

Tracer specialized classes have a property, *shadingModel*, that stores the current shading model to be used by the tracer in the color calculation.

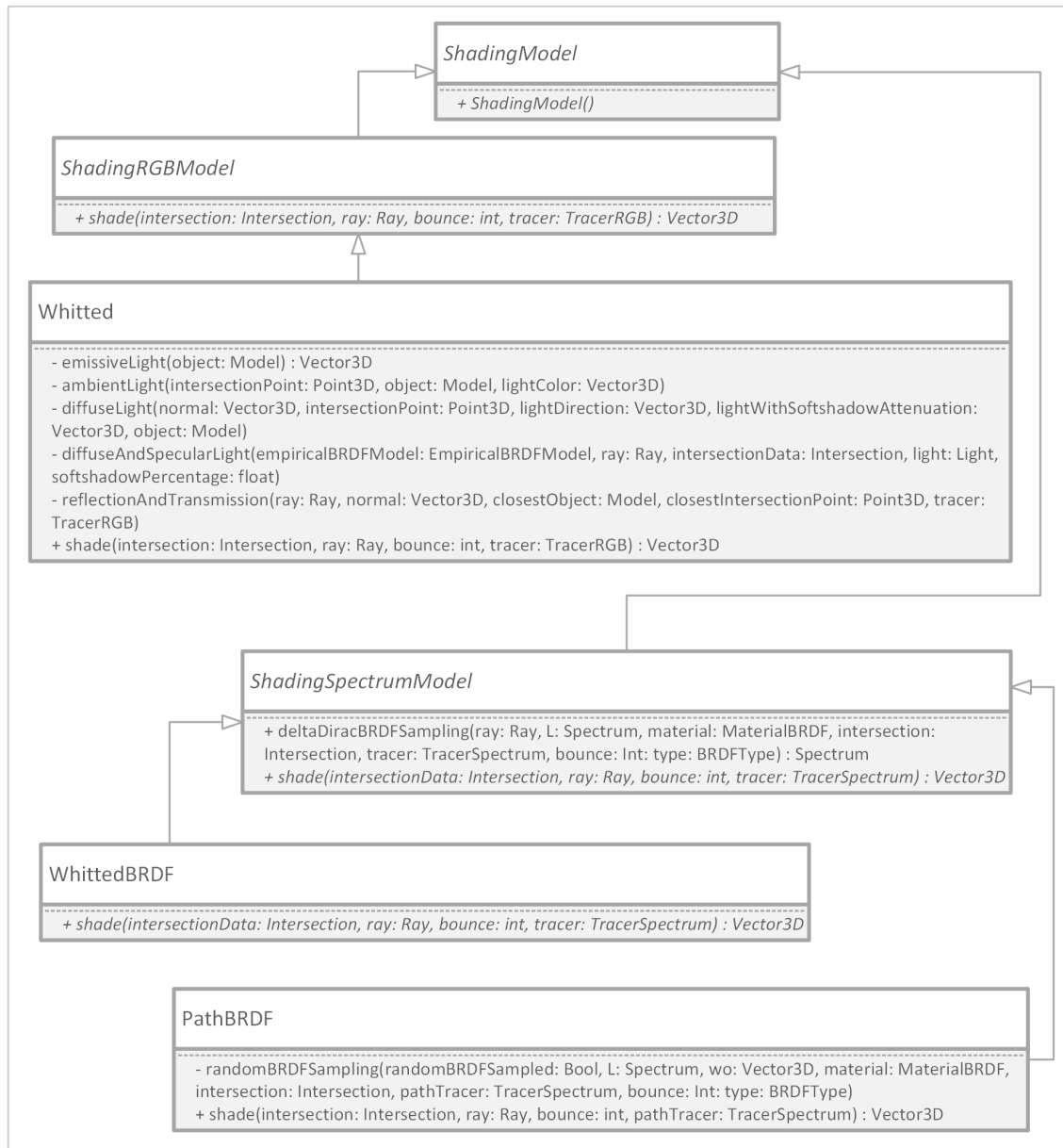


Figure 15. ShadingModel classes hierarchy – UML class diagram.

For the shading models there's a base *ShadingModel* class. This class is extended with two specializations, the *ShadingRGBModel* and *ShadingSpectrumModel*. All the shading models implemented in SCLT must extend one of these classes to declare their support for RGB or spectral data.

SCLT support the following shading models, each one implemented with its own class:

- *Whitted*, a class that implements the Whitted ray tracing using empirical BRDF models Phong/Blinn-Phong and RGB data;
- *WhittedBRDF*, a class that implements the Whitted ray tracing using physically based BRDFs and spectral data;
- *PathBRDF*, a class that implements the Path tracing model using, as for the WhittedBRDF, using physically based BRDFs spectral data to render the scene.

To support some of these models a specialized tracer class could be needed. This is way it is possible to subclass the *TracerRGB* and *TracerSpectrum* classes to customize a part of the trace or color calculation process. For example, to support path tracing a specialization of the *TracerSpectrum* class, the *PathTracer* class, has been created to customize the color calculation and the trace process for this model.

All empirical BRDF models are contained in the *EmpiricalModel* class: this choice was made because the main focus of SCLT is spectral rendering, and RGB rendering was added as a proof of concept to do some comparison with PBR scene.

Physically based models are all implemented as a specialization of the *BRDF* base class. As for the tracer and shading component, the addition of a new BRDF model requires just the creation of a new class that extends the previously defined *BRDF* base class. SCLT supports by default the following physically based BRDF models:

- Lambertian;
- Oren Nayar;
- Specular reflection;
- Specular transmission;
- Torrance Sparrow;
- Measured.

These models has been described in section 3.3.2. As the main focus of SCLT is to render spectral scene, all BRDF models are implemented with support for spectral data only.

The various BRDF are associated in the *MaterialBRDF* class to create different types of material. There's also a *MaterialRGB* class used in RGB scene. All of them are a specialization of the *Material* class.

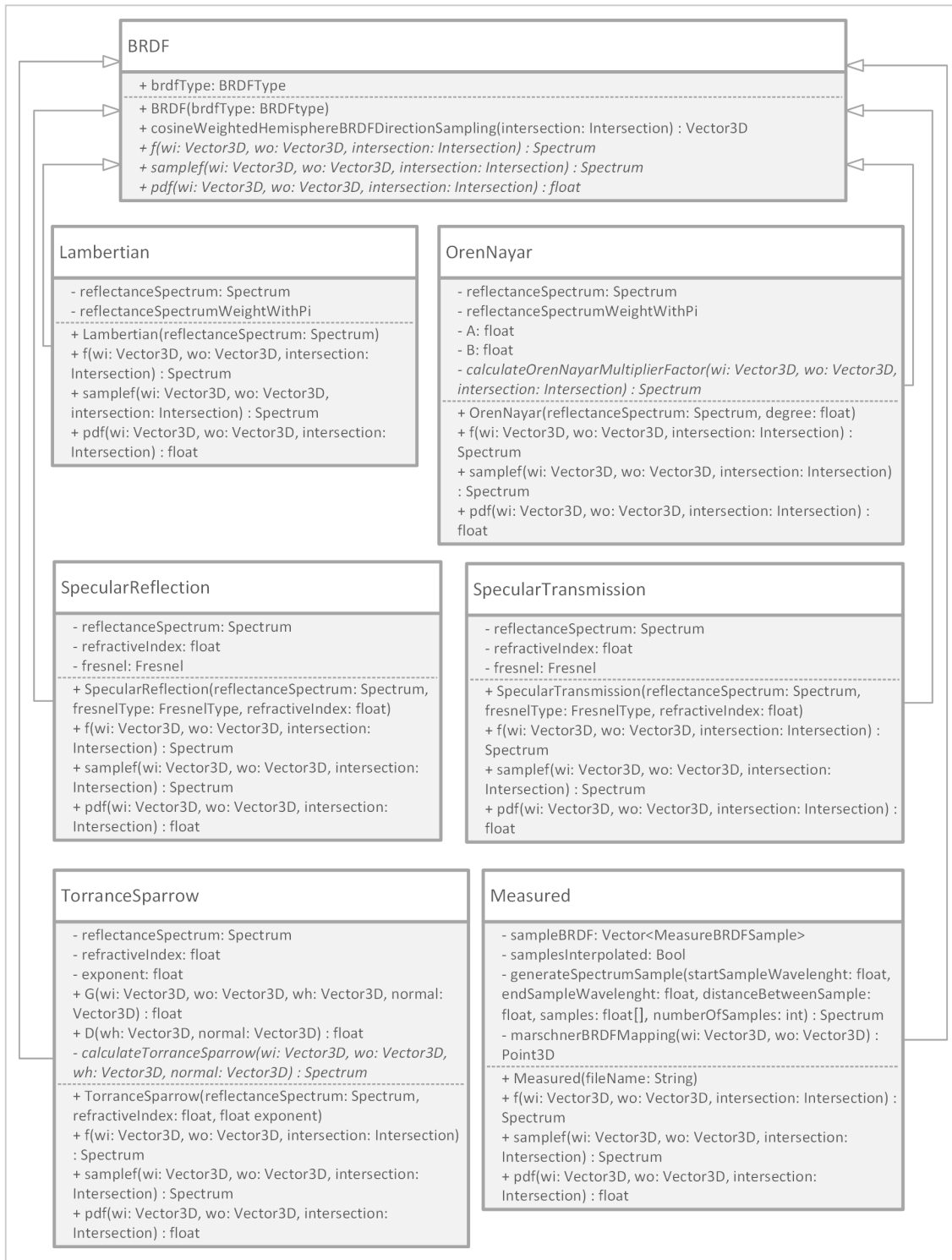


Figure 16. BRDF classes hierarchy – UML class diagram.

All the previous components use data taken from the *Scene* class, that contains the scene definition with the informations of the tracer and shading model to be used during rendering. The *TracerModelFactory* and a *ShadingModelFactory* classes create an instance of the tracer and shading model defined in the scene description. There is one single instance of the *Scene* class created in the *SCLT* class, based on the user selection, that is passed as a pointer through all the classes previously seen that need it.

Last but not least, there's a *ColorRenderingIndex* class used to compute the CRI of a scene. It uses a set of classes that implements all the different color spaces presented in chapter 2 to execute the calculation. One of them is the *CIE1931XYZ* class previously defined. It also uses a class, *ChromaticAdaptationTransform*, that implements the various chromatic adaptation transform described earlier. There is also a set of classes that simply contains almost all the spectral data used in SCLT. These spectral data are used during CRI calculation and to defined the SPD of materials and light sources. Finally, the class *ColorMatchingFunction* is used by *CIE1931XYZ* class, defined the color matching function used by SCLT.

Generally speaking, the entire structure presented was defined with the focus to let SCLT be a “pluggable and extensible system”: it would be easy to add new tracers or shading models as a specialization of the base classes presented previously.

Other components of SCLT are not listed here, because their implementation is straightforward. For example, the SPD of illuminants and objects is represented with the *Spectrum* class, that is just a wrapper for an array containing the value for each wavelength.

4.2 Modern cross platform development

There's a part of SCLT, not strictly correlated to computer graphics, that differentiates it from other ray tracing engines. Usually the result of the rendering in a standard ray tracing engine is saved to disk as an image, and the rendering process is started and controlled from command line. So most of the ray tracing engine available online are only command line application. This is not the case of SCLT, that has been developed as a command line application for Linux operating system, but also as a full application able to run with a specific native interface on Apple device, on the mobile operating system iOS and the desktop operating system OS X, and also on Windows platform.

The implementation of the core rendering parts of SCLT has been done using C++ language. Using only functions available in the C++ standard library, it is possible to compile the core classes with

different compilers: g++ of the Gnu Compiler Collection (GCC) on Linux platforms, Visual C++ on Microsoft platforms and LLVM on Apple platforms. The scene input in each application is done using a specific XML file format. See appendix B for more details on SCLT scene definition. In the following paragraphs a brief description of the specific implementations on each platform is reported.

4.2.1 Apple: iOS and OS X

On Apple platforms the main languages used to develop native applications are Objective-C and Swift.

The first one is a superset of C language with Smalltalk-like messaging features. The second one has recently been added as a development choice by Apple (the language was introduced for the first time to developers in 2014). It was presented as a powerful language with a mix of features inherited from: C#, Ruby, Rust, and again Objective-C.

So it's clear that there was a big question before starting with the development: which one of the two languages is the correct choice? The answer is simple: Objective-C. Even if it is clear that Apple will pay more attention to Swift in the future, as it tries to attract more developers (because it is a more "immediate" language), Objective-C will be supported for a very long time as some specific application will always require some of its feature, as for example the easy integration with C/C++ languages. In particular, for C++, an extension of Objective-C, Objective-C++, allows to freely mix the two languages with very few limitations.

So the complete architecture of SCLT on Apple devices is composed of three different levels:

1. *SCLT Engine Core*, containing the core classes and feature of the ray tracer engine;
2. *Rasterizer*, a class that executes the raster of the image data obtained from SCLT Core;
3. *User interface (UI) and image display*.

One last question is related to the device supported. It is clear that an application like SCLT is much more usable on a device with an adequate screen size and resolution. This is why SCLT supports any Apple computer running OS X El Capitan 10.11 or higher and any iPad model that could run iOS 9.0 or higher. iPhone, the mainstream Apple device, is not supported by SCLT at the moment.

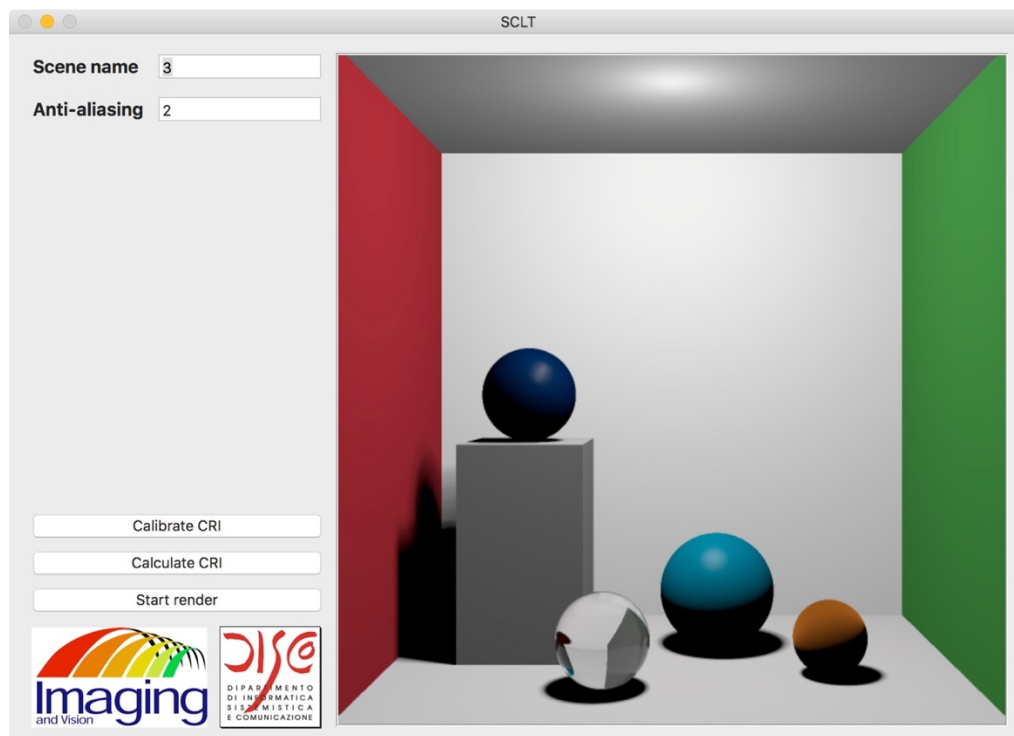


Figure 17. SCLT OS X User Interface.



Figure 18. SCLT iOS User Interface.

4.2.2 Windows

With the release of Windows 10, a lot has changed in the Microsoft development world. In fact, this major release of their operating system brings a new completely redesigned development platform: Universal Windows Application. In fact, it is now possible to develop a single application that could be installed on mobile phones, tables and desktop computer running Windows 10. There are two main languages available to be used to develop UWP applications: C# and C++. The first one is the “mainstream” Microsoft language. It was created as a competitor for Java, and it has become the main development language on the Windows platform. The second one has been the reference language on the Windows platform for years. It has been surpassed by C#, but its presence remains strong and a lot of applications are still written with it. Also in this case the choice is simple: C++. In this way the SCLT C++ core could be easily integrated into a Windows UWP application.

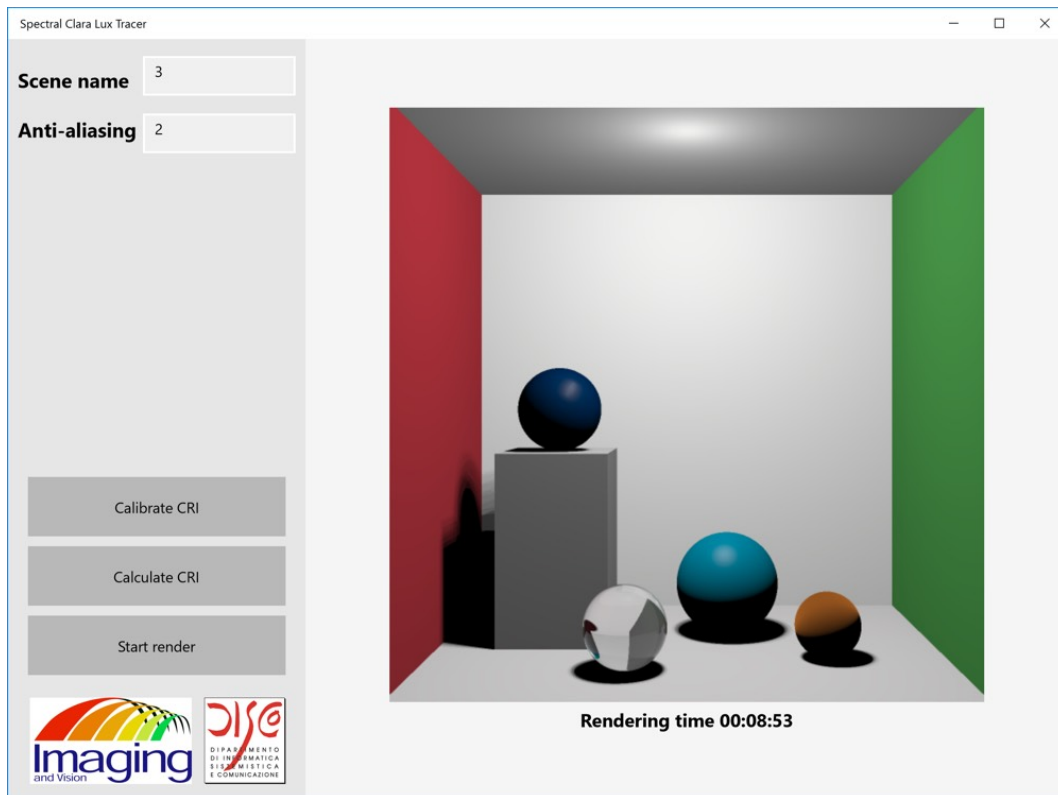


Figure 19. SCLT Windows 10 User Interface.

The architecture of SCLT UWP application is similar to the one previously seen for Apple devices, with three main components: the *SCLT Core Engine*, a *Rasterizer* specific for the Windows platform and last but not least the *User Interface and image display*. Related to the last component there is an important thing to be noted: the user interface for the UWP application has been developed using XAML, an xml-based language used on the Microsoft platform to develop the UI, and it is the same for computers and tablet devices. This is possible due to the fact that XAML let the developer create responsive layout, that adapt themselves on the device on which it is displayed.

4.2.3 Linux

Linux is the most used operating system for scientific researches. As previously stated, SCLT is also an educational tool. For this reason SCLT is also available as a command line application, to let students and academic researchers use it on their preferred distro.

As a consequence of the fact that Linux is also the most used operating system on servers, this version of SCLT could be used to generate the most sophisticated scenes using much more powerful machine with respect to classic desktop/laptop computers.

The rasterizer component in this case is a third party library: LodePNG [46]. This is a good choice because in this way the SCLT core engine could be used as a standalone application not dependent on any operating system feature.

4.2.4 Application architecture

In the following diagram it is reported the complete architecture of SCLT.

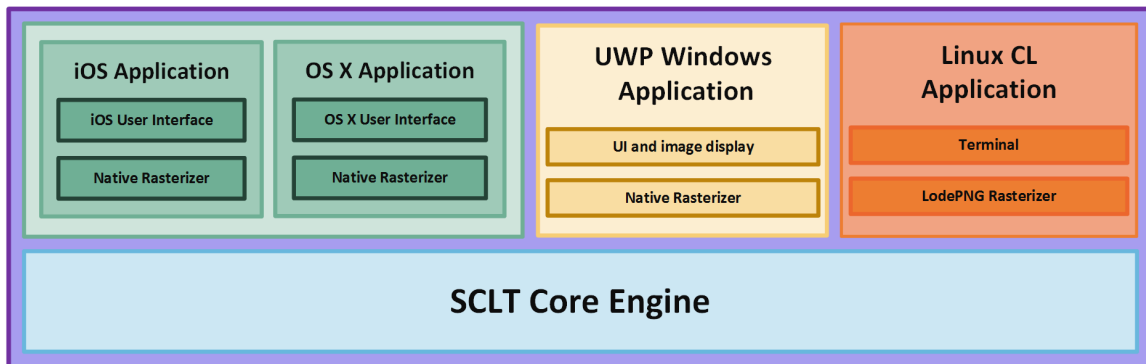


Figure 20. SCLT application architecture.

As could be easily seen, SCLT is a true cross platform application. It has been developed without the need of any external cross-platform framework.

The diagram previously showed is an overview of all the components used in the various native application versions of SCLT previously described. It is interesting to describe the architectural pattern used in the various native SCLT application versions. In fact, they use the same pattern: Model View Controller (MVC). In this way it has been possible to separate the SCLT core components, as previously stated written in C++, from the user interface and the specific platform application code.

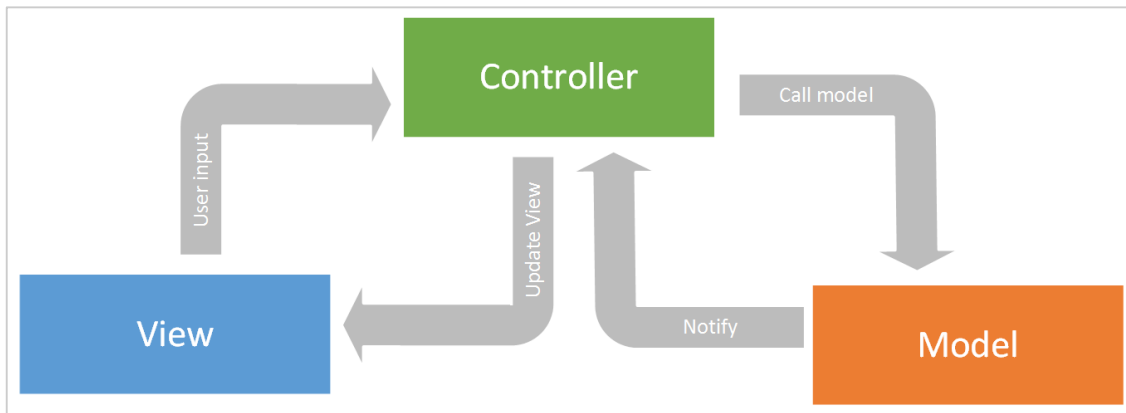


Figure 21. Model-View-Controller architectural pattern.

iOS and OS X SCLT applications have a similar structure: there's a *MainViewController* class, that is a subclass of the *NSViewController* for OS X and a subclass of *UIViewController* for iOS, that manage SCLT rendering process status (and eventually CRI calculation) and the user input to setup the ray tracing process. These classes are contained in the two main MVC frameworks available on Apple platforms: Cocoa [47] for OS X and Cocoa Touch [48] for iOS.

Each one of these controllers uses Grand Central Dispatcher (GCD), an Apple framework to support concurrent code execution and avoid in this way the block of the UI [49]. The view part is composed of:

- the UI to get the user input;
- the UI to show the rendering result.

These UI components are platform specific and are taken from the UI frameworks previously described. The model is composed of:

- all SCLT C++ core code. This one is shared between the iOS and OS X applications using an Apple embedded framework. In this way, during the build process, it could be automatically targeted for compilation on the specific platform architecture needed (X86 - 64 bit Intel for OS X and armv7 - arm64 for iOS);
- a native rasterization class for each platform that creates an image, as a result of the ray tracing rendering data obtained from the SCLT core classes, to be displayed to the user.

The Windows 10 version of SCLT has a similar structure. There's a *MainPage* class that is the controller of the application. This class uses the concurrency task to support concurrent code execution and avoid the UI block [50].

The View/UI part is created using, as previously seen, XAML and native Windows 10 UWP objects. The model is composed of:

- all SCLT C++ core code. For Windows, this classes are included in the main project, because it is possible to switch the compilation target directly from the main project itself (X86 32 bit or 64 bit);
- a native rasterization class as for the apple platform, that uses specific Windows functions to generate the rendered image of the ray traced scene.

The Linux command line version doesn't follow the MVC pattern. In fact, it includes only the model: the SCLT core rendering engine classes. The additional components needed for this version are the *main* method, which is the designated start of the application, and a *rasterizer* method, used to generate the image of the rendered scene. In fact, this two methods are contained in the *main.cpp* source file. This file could be found in the folder that contains the core engine classes, but it is used only by command line version of SCLT. The command syntax of the command line version is: *./SpectralClaraLuxTracer <scene filename path> <antialiasing> <width> <height> <result image name>*. The parameters are self explanatory.

4.2.5 Development tools, tests and continuous integration

SCLT has been developed using platform specific tools. This is a consequence of the fact that each operating systems, as previously seen, uses different languages and targets different computer architectures.

For each platform, their main IDEs has been used in the development:

- XCode for Apple;
- Microsoft Visual Studio for Windows;
- CLion for Linux.

All the source code of SCLT has been managed using Git as source version control system, and it is actually hosted in a public Github repository (see appendix C for the Github repository url).

All versions of SCLT have a suite of test cases, written using platform specific frameworks. This test cases are used for continuous integration (CI): each time a new commit, containing modification to the SCLT source code, is pushed to the Github repository, all the test cases for each platform are executed and a general build of each application version is created to check that the compile process works as expected. The platform used for CI are:

- Travis [51], for Apple platform, because it gives to the developer the ability to use some pre-configured OS X virtual machine with the latest version of XCode installed. In this way it is possible to generate the various test build for OS X and iOS (using the iOS simulator);
- Appveyor [52], for Windows platform, because it is the only continuous integration service to support Windows 10.

The setup and the build process of the command line version of SCLT has been done using CMake [53], a cross platform build tool. In fact, the command line version could be easily ported to other platforms, if the CMakeList file is configured correctly.

4.2.6 Project size

SCLT is a single developer - medium size project. It is interesting to see how many lines of code (LOC) there are for each one of its main parts. The count has been done using CLOC [54]. The LOC in SCLT core classes are 10365. As previously stated, all core classes are written in C++.

The LOC contained in Apple platform applications are 804. There's no difference between iOS and OS X: the application targets are part of the same XCode project.

The LOC in UWP Windows platform application are 2451. In this case CLOC includes some configuration files generated by Visual Studio.

The entire SCLT project is composed of 15325 LOC. The count has been done including the scene XML created and CI configuration files.

If the count is done including also the third party libraries used, the entire SCLT project is composed of 20429 LOC.

Finally, it is interesting to note the lines of comments counted by CLOC: 6516. In fact, SCLT is a well documented project. Each function is associated with a detailed description of what it does, its parameters, and its return value.

Chapter 5

SCLT: physically based rendering

This section contains a description of the implementation details of PBR in SCLT.

Then some scenes rendered will be presented. They will show all the features of SCLT previously described. The first two scenes presented will be rendered using RGB data and Whitted model, to show typical images that could be obtained from a standard ray tracing engine. The other scene presented show how SCLT could render physically based scenes, with all the BRDF models previously described.

5.1 Physically based rendering: implementation details

One of the core main feature of SCLT is the set of physically based BRDF models it supports.

All these models, as previously seen, are a specialization of the BRDF base class. This class has a property, *brdfType*, that specify which type of BRDF the models implements. Each BRDF model defined in SCLT belongs to one of these families:

- diffuse;
- specular;
- reflection;
- transmission.

All BRDF models implements three methods. The first one, $f(const\ Vector\&\ wi, const\ Vector\&\ wo, const\ Intersection\&\ intersection)$, implements the BRDF model equations. The second one $samplef(Vector*\ wi, const\ Vector\&\ wo, const\ Intersection\&\ intersection)$, implements the model equations like the previous one, and it samples the ω_i direction so that some ray tracing models,

like path tracing, could use it to trace the path of a ray to the light source. The last one $pdf(const\ Vector\&\ wi, const\ Vector\&\ wo, const\ Intersection\&\ intersection)$ is used again in some specific models, like path tracing, to solve the rendering equation using Monte Carlo technique. This one has been hinted during path tracing explanation and will be detailed in section 5.1.3. In the following paragraphs is reported a description of how the various models are implemented in SCLT.

5.1.1 Diffuse and Specular models

The first model of the diffuse family implemented is also the simplest one: Lambertian model. As could be seen in the equation presented in the paragraph 3.3.2.1 the implementation is straightforward: the object spectrum is divide by pi. A different approach must be taken for the Oren-Nayar and Torrance-sparrow models. Their equations are heavily based on the use of the light directions ω_i and ω_o in spherical coordinates. To calculate them usually a change of coordinate reference system transformation of this directions to a tangent space with origin on the point of ray intersection is usually required. In this way the calculation of spherical coordinates could be consistent for each point of intersection and for each direction.

But sometimes, when possible, it is better to rely on trigonometry and vector operations to avoid the calculation costs of the transformation. This is why these models are all implemented using operation that don't use spherical coordinates directly. As an example, remembering the Oren-Nayar formulation, its $\cos(\phi_i - \phi_o)$ term could be calculated as the dot product of the projection of the two angles onto the tangent space with the following formula [55]:

$$\cos(\phi_i - \phi_o) = \| \omega_i - n(\omega_i \cdot n) \| \| \omega_o - n(\omega_o \cdot n) \| \quad (49)$$

A different approach is needed for the measured BRDF model. As previously described in section 3.3.2.5, the data sets used in SCLT are the Cornell University Reflectance Data. These data sets are indexed using spherical coordinates. This is why, first of all, the generation of the tangent space axis is needed. For the y axis, the normal at the surface intersection point will be used. For tangent and bitangent vectors, different approaches are available. Usually a coordinate system could be created strictly from one vector, cutting down one of the component and inverting the remaining two. This would be a good technique to use, because the measured BRDF like all the other diffuse models presented in this thesis are isotropic, so they don't need the tangent to be

oriented in a specific direction. This useful technique is used in other part of SCLT, but for the measured BRDF another way has been explored: it is possible to calculate the cross product between the normal and ω_i direction to obtain a tangent vector candidate, and then calculate the cross product between it and the normal to obtain the bitangent. This is the preferred method used by SCLT to calculate the tangent space for the measured BRDF model, and execute the change of coordinate transform for the ω_i and ω_o direction to convert them to spherical coordinates. Now a step further must be taken: a direct read of a sample from the data sets that is the nearest to the spherical coordinate obtained from the previous calculation is not good for rendering (Pharr et al., 2010 [41]).

For this reason, a specific remap of the spherical coordinate must be used. SCLT uses the one defined by Stephen Maschner in its PhD dissertation [11]. The remap is given by the following formula (considering that in SCLT y is the up axis):

$$\text{remap}(\theta_i, \phi_i, \theta_o, \phi_o) = (\sin \theta_i \sin \theta_o, \cos \theta_i \cos \theta_o, \Delta\phi) \quad (50)$$

This remap has two important properties:

- isotropicity of the BRDF is taken into account. This is given by the $\Delta\phi$. Using this term, pairs of directions (ω_i, ω_o) , that have different ϕ values, could be associated with the same set of samples, because they are at the same $\Delta\phi$ distance even if the ϕ values are not the same. In this way the BRDF doesn't depend on the tangent direction used to convert the ω_i and ω_o ;
- the reciprocity of the BRDF is taken into account.

After the remapping the BRDF measured data could be used during rendering.

5.1.2 Specular reflection/transmission models

The last models implemented in SCLT are the reflection and transmission models.

They are used together to create a glass material. An important implementation detail must be described. *MaterialBRDF* class, the one used to create materials using physically based BRDF models, has two main methods: *f(const Vector3D& wi, const Vector3D& wo, const Intersection& intersection)* and *samplef(Vector3D* wi, const Vector3D& wo, const Intersection& intersection,*

const BRDFType type). The first one accounts for all the BRDF associated with a material and sum up their contribution. The second one sample a specific type of BRDF and gives the SPD of the material and the new ω_i direction to be traced. For all other type of BRDF one method or the other could be used to get the total SPD of the material, based on the type of tracer used. But this is not the case for the specular reflection/transmission models: all the types of ray tracing model supported by SCLT use the second method, because a new direction must always be sampled, to account for the reflection or refraction caused by this BRDF models.

5.1.3 Path tracing

In section 3.2.1 a general description of the path tracing algorithm has been given. In particular, the Monte Carlo Integration was cited as a way to find a numerical solution to the integral contained in the rendering equation.

Monte Carlo Integration techniques are powerful mathematical tools that could be used to approximate the value of an integral. How does it work? Before giving its description, some definitions are needed.

1. a random variable X is a value chosen by some random process. It could be discrete or continuous;
2. the Cumulative Distribution Function, CDF, $P(x)$ of a random variable is the probability that it will take a value that is less than or equal to some value from its distribution:

$$P(x) = Pr\{X \leq x\} \quad (51)$$

3. in case a random variable is continuous, the probability density function PDF gives the probability that a random variable takes a particular value. It is usually defined as the derivate of the CDF (see equation 52), and usually integrate to 1 over its domain;

$$p(x) = \frac{dP(x)}{dx} \quad (52)$$

4. the expected value $E_p[f(x)]$ of a function f is its average value over some distribution of values. It is defined as (with D corresponding to the function domain):

$$E_p[f(x)] = \int_D f(x)p(x)dx \quad (53)$$

5. the variance of a function is its deviation from its expected value.

Now it possible to define the Monte Carlo integration technique: given a set of uniform random variables $X_i \in [a, b]$, the expected value of the Monte Carlo estimator F_N , reported in equation 54, is equal to the integral of the function over the interval $[a, b]$ (Pharr et al., 2010 [56]).

$$E[F_N] = E\left[\frac{b-a}{N} \sum_{i=1}^N f(X_i)\right] = \int_a^b f(x)dx \quad (54)$$

When the random variables are not uniform the PDF must be taken into account and the estimator become:

$$F_N = \frac{1}{N} \sum_{i=1}^N \frac{f(X_i)}{p(X_i)} \quad (55)$$

How is it related to path tracing, and how it works in detail?

The path tracing calculates N samples for each pixel. Each sample is the color obtained from tracing all the bounces of a ray from the camera to a light source (or to a max bounce number, and in this case the color is black, or SPD equal to 0 in case of spectral data).

The direction of a ray bounce, that occurs when a ray hit an object, is just a new ray selected accordingly to a specific distribution. All the colors obtained from the various sample are then averaged. The result of this process is the expected value for the Monte Carlo estimator that correspond to the one of the rendering equation, presented in section 3.1.

SCLT uses this general path tracer implementation with an addition: the distribution from which a new ray is selected is not uniform, and is chosen from a distribution with a shape that is similar to that of the BRDF distribution. This technique is called importance sampling. In fact, SCLT uses:

- a cosine weighted distribution for all the diffuse BRDFs;
- a distribution with similarity with the Blinn distribution for the Torrance-Sparrow BRDF.

The key feature of the SCLT path tracer is in the data used for color estimation: all calculations are done using spectral power distributions for object materials and for light sources. The value obtained for each pixel is then converted to tristimulus values and then sRGB color to be displayed. To regulate the overall power of the light in the scene, a brightness parameter has been used. Also sRGB gamma correction is applied, as for the other model contained in SCLT that uses spectral data, to better decode the colors obtained from spectral data.

In this way SCLT path tracer is a valid candidate to be used in color fidelity evaluation on a rendered scene, and useful to visualize color phenomenon like metamerism (even if the last one could be seen also with Whitted Ray tracing model). Obviously the rendered scene with path tracer are the most impressive that SCLT could generate.

5.2 Case study 1: RGB scene using Whitted

Even if, as previously seen, the main focus of SCLT is on spectral rendering, it has also the ability to render standard RGB scene using empirical lighting model.

Two scenes have been created and rendered using 4X antialiasing. The first is rendered using Phong model (Figure 19), and the second one using Blinn-Phong (Figure 20). There are objects composed of different materials: diffuse, specular, glass and mirror. SCLT supports, only for RGB rendering, some texture mapping techniques.

Texture mapping was pioneered by Edwin Catmull in his PhD thesis in 1974 [57]. A texture is a 1D, 2D or 3D array of texels. Texels are texture samples.

They are classified as:

- color texel, that contains RGBA data. The texture composed of color texel is a color map;
- alpha texel, that contains alpha values;
- normal texel, that contains normal directions values. The texture composed of normal texel could be a normal map or a bump map.

The two RGB scenes presented contain different kind of texture mapping.

The first scene shows the implementation of cube mapping technique. It was invented by Ned Green in 1986 [58].

It uses six color maps, that are applied to a cube that contains the whole scene rendered. These color map are used to represent the view of the world in all directions. Rays traced in the scene that doesn't intersect any 3D object, samples a color from one of the color maps using basic algebra. In fact, the component with the biggest magnitude tells which of the 6 sides of the cube must be sampled. The other two components are used for sampling the color map. The skybox that contains the first scene has been generated using this technique.

The second scene shows an example of procedural texture generated using Perlin noise. A procedural texture is a texture that generates on the fly the texel for a specific sampled position. Perlin noise is an algorithm that could be used to generate procedural texture. It was invented by Kevin Perlin in 1983 [59] [60].

The basic idea is to combine pseudorandom noisy signals to generate natural randomic effects (eg. clouds, lava). The blue/white sphere at the top of the cube and the orange/red sphere in the center of the scene are rendered using this technique.

The second scene shows also an example of bump mapping. Bump mapping is a technique introduced by James Blinn in 1978 [61] to simulate rough surfaces. During lighting calculation, normals are recovered from a bump map and are used to change the perceived light. Normal could also be geneareted procedurally using Perlin Noise. The ruby sphere on the bottom right is an example of this technique (using Perlin noise to generate the normal map procedurally).



Figure 22. RGB scene: Whitted, Phong model, cube mapping, bump mapping.



Figure 23. RGB scene: Whitted, Blinn-Phong, multiple texture types.

5.3 Case study 2: spectral scenes using Whitted

Three scenes are presented in this case study. They are rendered using Whitted model, spectral data, and using 4X antialiasing.

All color calculations are done using 81 samples for each SPD used: the distance between each wavelength is 5 nm.

The first scene contains all the physically based BRDFs supported by SCLT: Lambertian, Oren-Nayar, specular reflection and refraction, Torrance-Sparrow and measured BRDF.

It is rendered using different illuminants, to see the effect of each of them on the various color rendered.

In Figure 24, the scene just described is rendered using illuminant D65. The colors, in the entire scene, looks natural because, as previously seen, this illuminant simulates the light condition at open air comparable to sunlight. In Figure 25 the same scene is rendered using illuminant FL4. As could be easily seen, the color of the cyan sphere, rendered using Torrance-Sparrow BRDF, and of the krylon blue sphere rendered with measured BRDF are changed from the previous one as a consequence of the FL4 SPD.

Figure 26 shows again the same scene, this time rendered with illuminant FL9. This illuminant is part of a subset of the F family known as full-spectrum light fluorescent lamp.

This subset includes illuminant FL7 and FL8. These illuminants simulate fluorescent lamps that approximate the entire visible light spectrum.

For this reason, lights with an SPD of this kind have high CRI, and they are considered as one of the most accurate light source with respect to the natural visible light. In fact, as could be seen, the result of the rendering is much more similar to the one of Figure 24. Figure 27 shows again the same scene for the last time, rendered with the standard A illuminant. As previously seen, this illuminant simulates a typical domestic tungsten-filament light source.

This is why this image rendered is interesting: what it shows is the scene as it will be illuminated by a common lamp. It's easy to see how color are affected by the illuminant SPD, and the color appearance of some of the objects rendered is different from their original in the rendered image with the D65 illuminant.

The second scene presented in this case study shows all types of measured BRDF available in SCLT. As stated in section 3.3.2.5, the data set used in SCLT is the Cornell reflectance data. There are four measured BRDF: krylon blue, cayman, garnet red, and acryl blue. Figure 28 contains the scene with these measured BRDFs, rendered with the illuminants used in the previous scene.

The third scene presented shows another feature of SCT that is not strictly correlated with PBR: the ability to render polygonal mesh. A polygonal mesh is a set of connected polygons (generated from vertices and edges), that defines the shape of a model. The most common polygon used is the triangle.

Figure 29 shows a scene with all lambertian BRDFs, that contains a modified version of the Stanford dragon 3D model contained in the Stanford 3D scanning repository [62].

The original mesh from the Stanford repository contains 871414 triangles. This means that for each ray, all these triangles must be tested to check if there's an intersection, leading to poor performance in rendering speed.

This is a consequence of the fact that in the standard ray tracing algorithm, the ray-objects intersection test has $O(n)$ complexity, where n is the number of objects to be tested. To improve the performance of the mesh rendering, the original Stanford dragon has been simplified using Meshlab [63]. In this way the mesh used in the scene is composed only by 7423 faces, but maintains a good level of details if compared to the original model. The simplified mesh was not enough to obtain reasonable performance. This is why SCLT support a technique to speed up the ray intersection process: the axis aligned bounding box (AABB) [64].

The polygon mesh is bounded in a box, aligned to the world coordinates system of the scene. During a ray intersection test, instead of checking all polygons of the mesh, a first test is done: the ray is tested for intersection with the bounding box. If it is not intersected, then the ray doesn't intersect any of the polygon of the mesh inside it, so they can be skipped in the intersection test. The box intersection is calculated from the ray plane intersection generic equation. As a consequence of the fact that the box is aligned with the coordinate system axis, the ray plane intersection equation is simplified to improve its calculation speed.

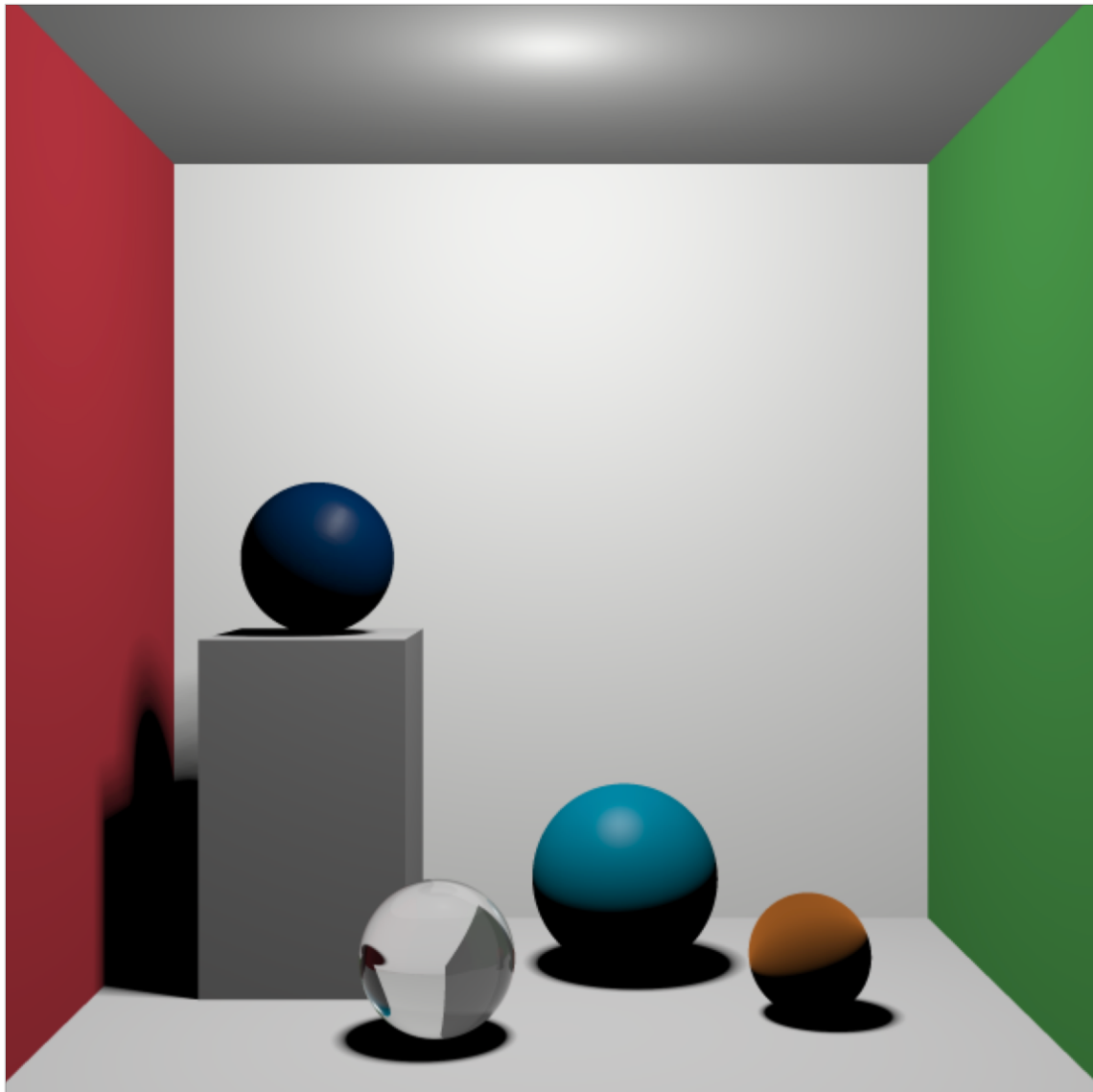


Figure 24. Spectral scene: Whitted, physically based BRDF, D65 illuminant.

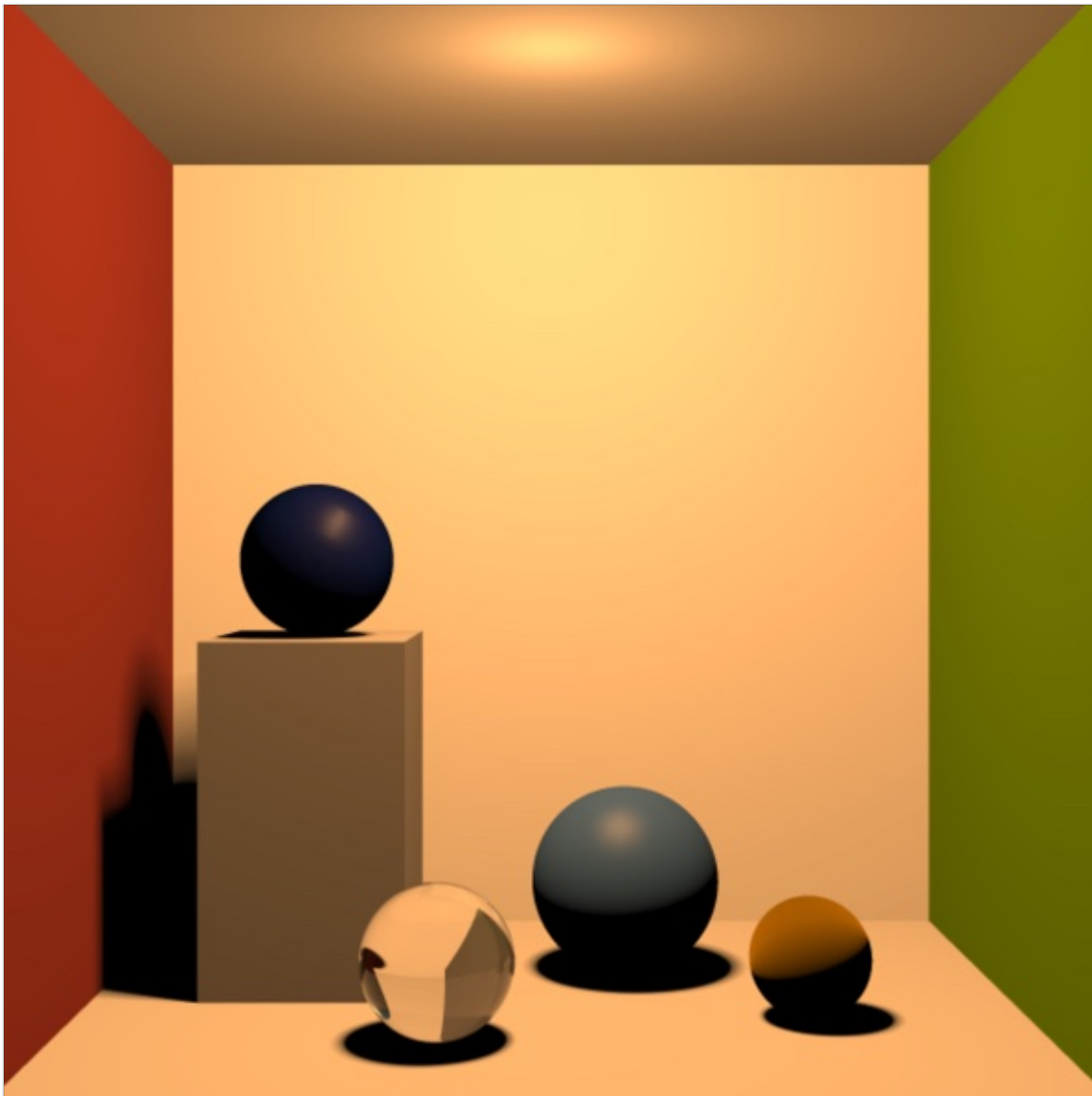


Figure 25. Spectral scene: Whitted, physically based BRDF, FL4 illuminant.

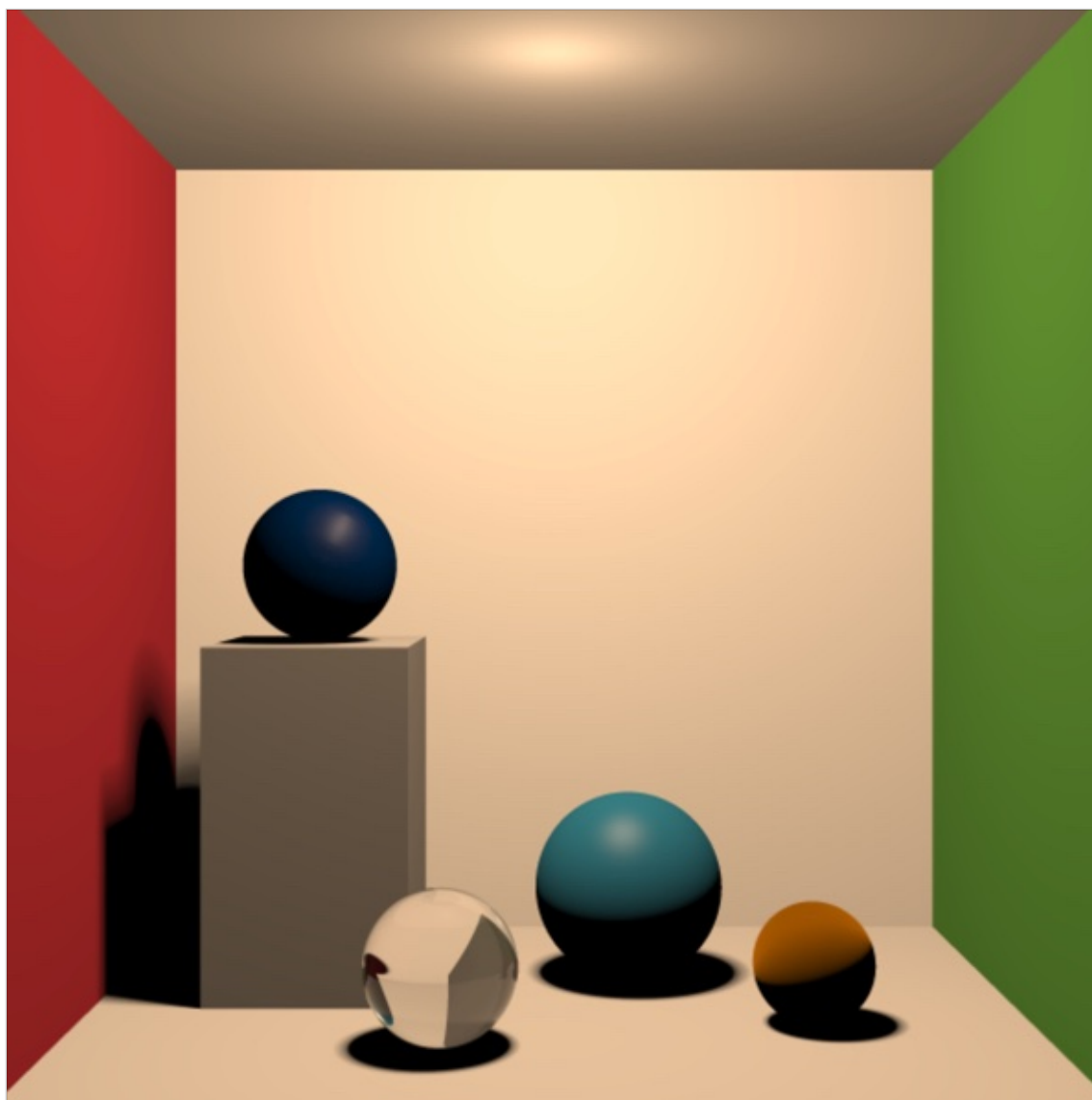


Figure 26. Spectral scene: Whitted, physically based BRDF, FL9 illuminant.

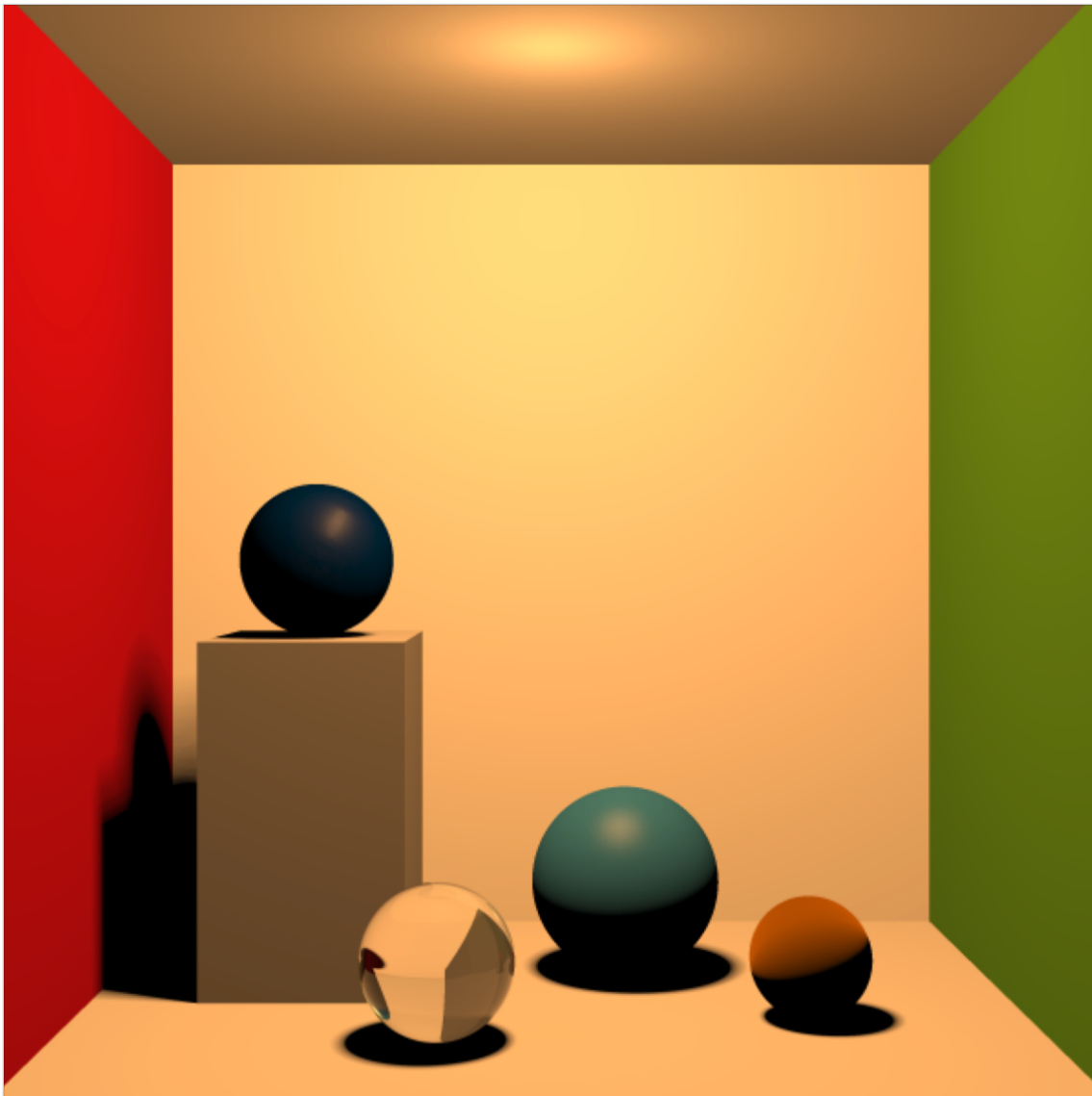


Figure 27. Spectral scene: Whitted, physically based BRDF, A illuminant.

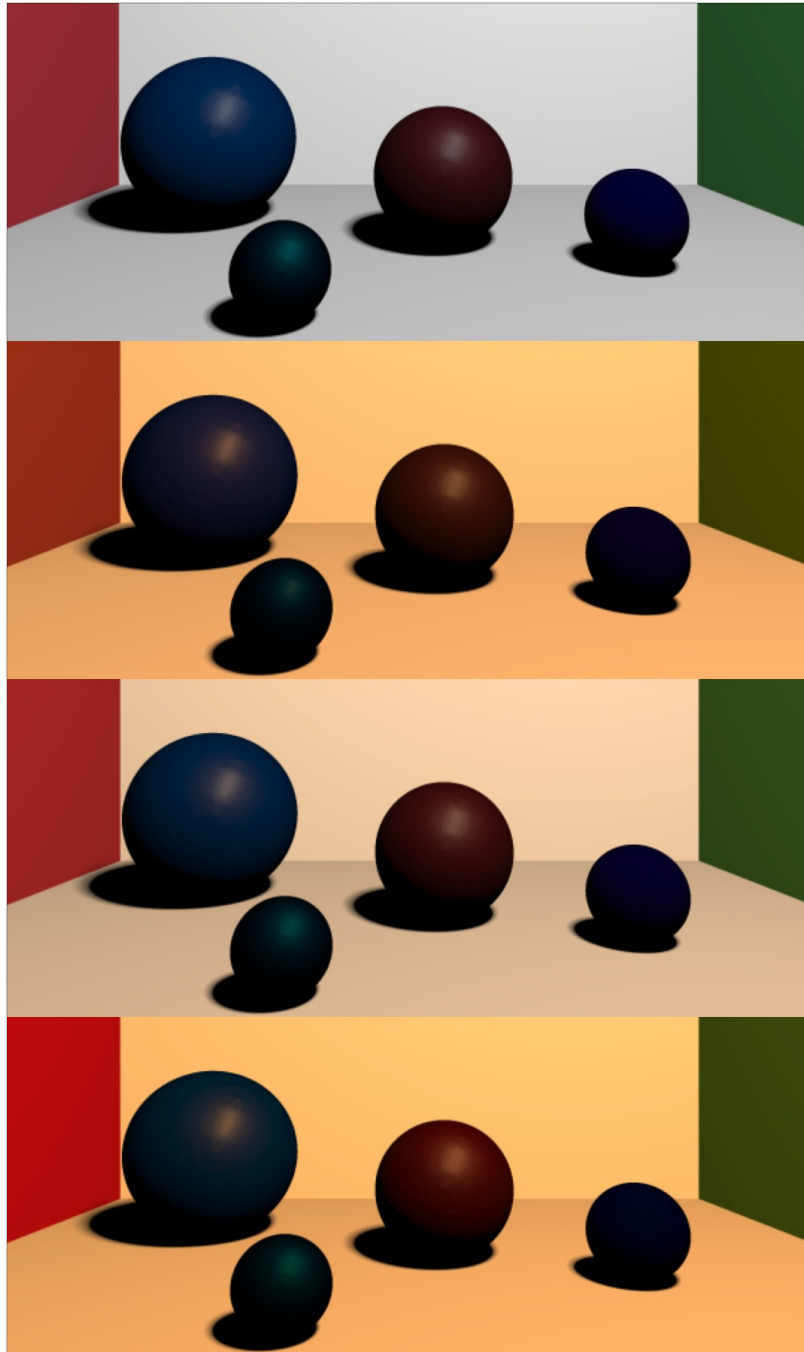


Figure 28. Measured BRDF data, illuminant D65, FL4, FL9, A.

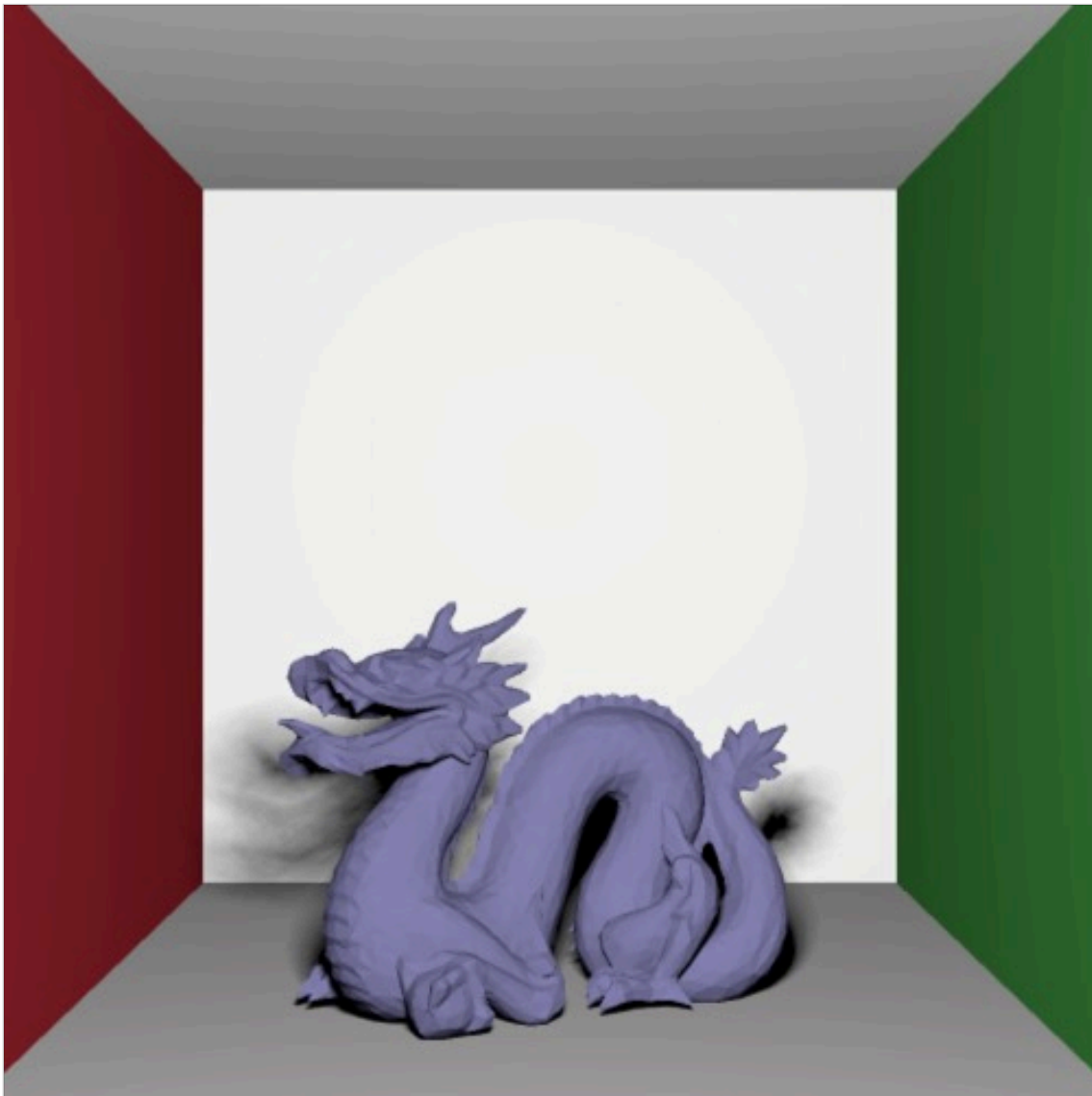


Figure 29. Spectral scene: Whitted, mesh (lambertian), D65 illuminant.

5.4 Case study 3: spectral scenes using path tracing

This case study is related to one of the key feature of SCLT: spectral path tracing.

All images are rendered using 2X antialiasing and, as for the Whitted spectral scenes, using 81 samples for each SPD.

Various images are presented that contains rendering of different scenes using this model. It is worth noting the level of realism of the objects in the scenes:

- all the glass spheres rendered show caustics on their nearest surfaces;
- all objects show natural soft shadows by default, without the need for artifacts or secondary specific rays.

Usually, to get a scene completely clean from noise, thousands of samples are needed, however after just 50 samples the image becomes recognizable, and after 4000 the scene has a good level of sharpness.

The first scene presented contains two glass sphere, one orange diffuse sphere obtained using Oren-Nayar BRDF and one blue diffuse sphere obtained using Lambertian BRDF. It is presented in multiple version based on the number of samples: 20 (Figure 30), 200 (Figure 31), 800 (Figure 32), 4000 (Figure 33) and 20000 (Figure 34). In this scene there aren't any objects that use the Torrance Sparrow BRDF.

This is because with path tracing, the Torrance Sparrow BRDF gains an incredible level of realism. As a consequence of the fact that it is a BRDF with reflective microfacets, a glossy effects shows up when a sphere is rendered with path tracing. To be sure that the sphere obtained is correct, a new scene has been created: one sphere in the center of a Cornell box, with black and white Macbeth spectrum respectively for the diffuse and specular components. A scene with a similar setup has been created also using the most famous physically based rendering engine cited in the introduction: PBRT [7].

The only difference in the two scenes are: the spectrum used for some of the wall of the Cornell box, neutral 8 on SCLT scene and white in the PBRT, and the color of the light, defined with D65 spectrum in SCLT and with simple RGB values in PBRT.

It is worth noting that PBRT uses Russian roulette, and other techniques, to render a scene as fast as possible, and with a good level of sharpness using few samples. So in this example PBRT uses 1000 samples, but SCLT needs 10000 samples to get the scene with a similar result.

As could be seen from the image obtained in Figure 35, SCLT rendered the sphere with Torrance Sparrow in the same way as PBRT does. The glossy effect on the sphere is clearly visible, with the wall of the Cornell box reflected on each side of it.

Now one good question arises: how does the scene presented in case study 2 appear if it is rendered using path tracing instead of Whitted model?

Figure 36 shows the result. The scene is rendered using 20000 samples and D65 as illuminant. It's easy to see the level of realism of the scene. All the spheres appear more naturally. Their shadows lose the fake appearance that is visible in the Whitted version. The Torrance Sparrow sphere gains a realistic glossy effect (the same visible in the scene reported in Figure 32), the blue measured BRDF sphere appears more uniform, with the specular highlight more defined, and finally the glass sphere is more coherent with respect to the box environment.

In Figure 37 the same scene is rendered using spectral path tracing and the illuminant FL4. The results in term of colors are the same obtained using the Whitted model.

It is worth noting that in all the path traced scene, the light source is rendered as an object inside it. This allow the user of SCLT to generate scene with great visual effects, mixing an emitting light object inside the scene with the others. Figure 38 shows an example, in which a scene is rendered with the light source placed inside it with a low level of brightness.

Another scene is presented in Figure 39. It contains a new Cornell Box with a mix of BRDFs (including garnet red measured BRDF), rendered with illuminant FL9.

The last path tracing scene presented contains, as the last Whitted scene presented in the previous paragraph, the dragon polygonal mesh from the Stanford scan repository [62]. It has been rendered applying different BRDFs: lambertian, in Figure 40, Torrance-sparrow, in Figure 41 and Specular reflection and transmission, in Figure 42.

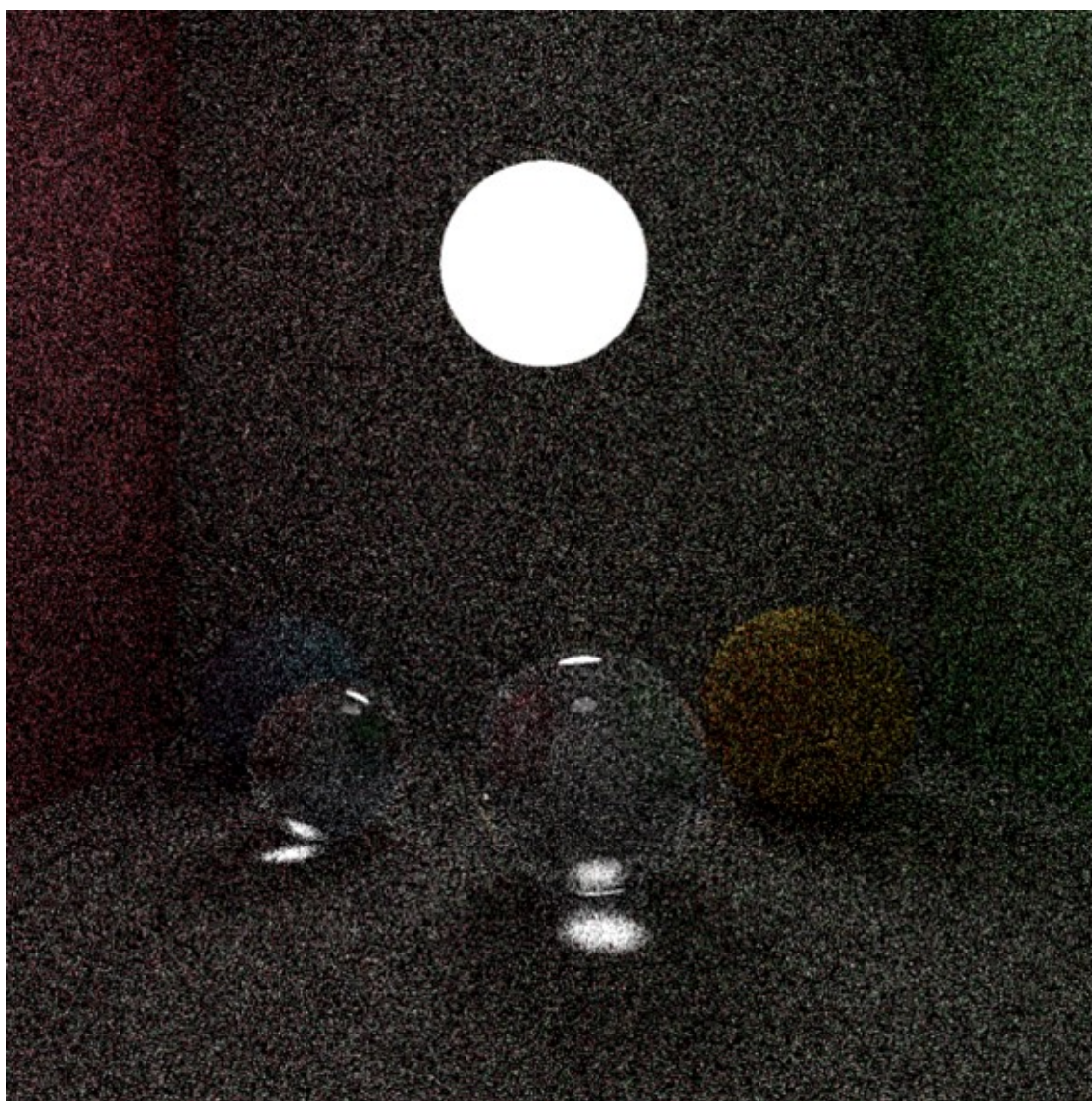


Figure 30. First spectral path tracing scene: 20 samples per pixel, illuminant D65.

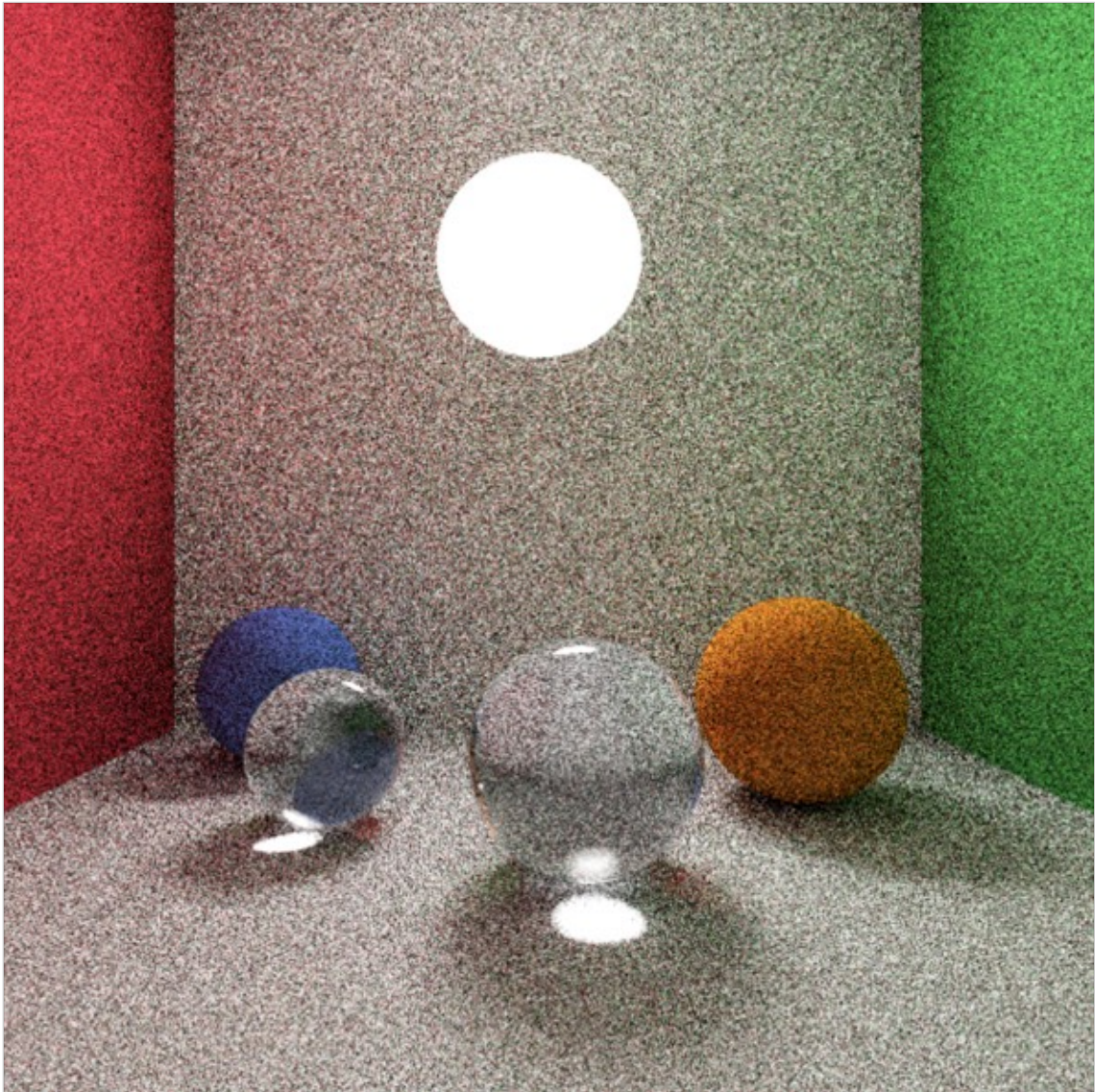


Figure 31. First spectral path tracing scene: 200 samples per pixel, illuminant D65.

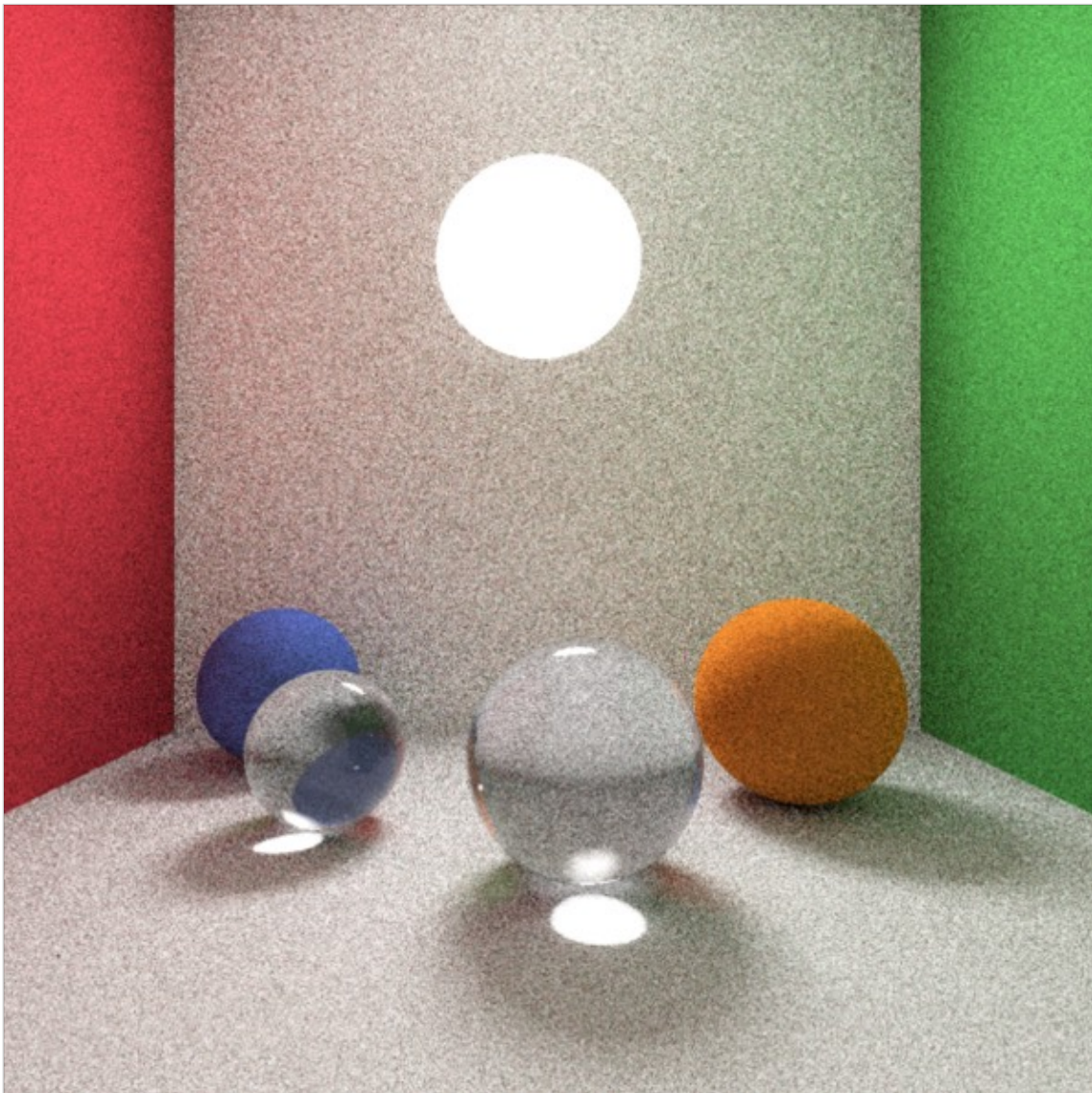


Figure 32. First spectral path tracing scene: 800 samples per pixel, illuminant D65.

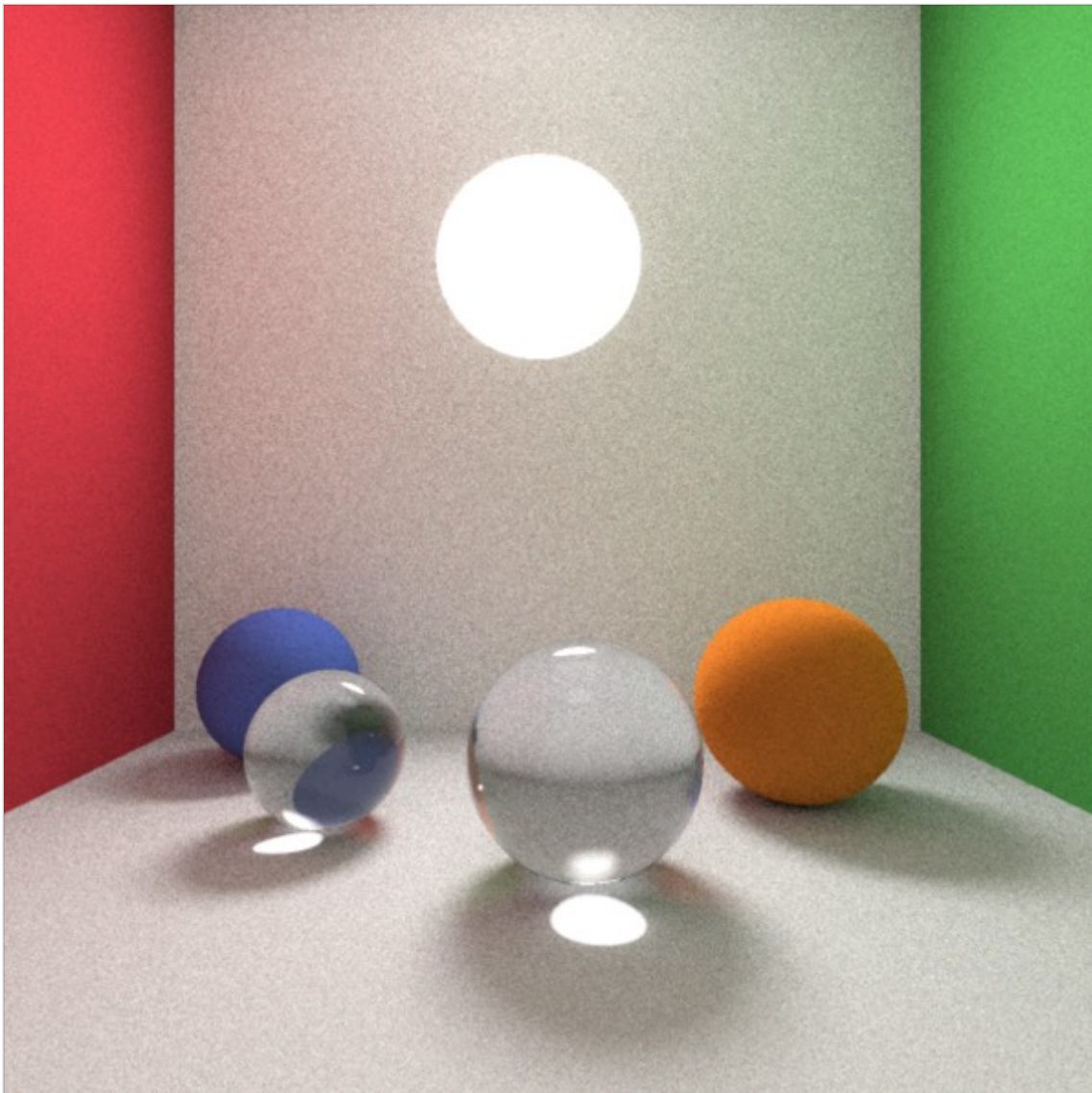


Figure 33. First spectral path tracing scene: 4000 samples per pixel, illuminant D65.

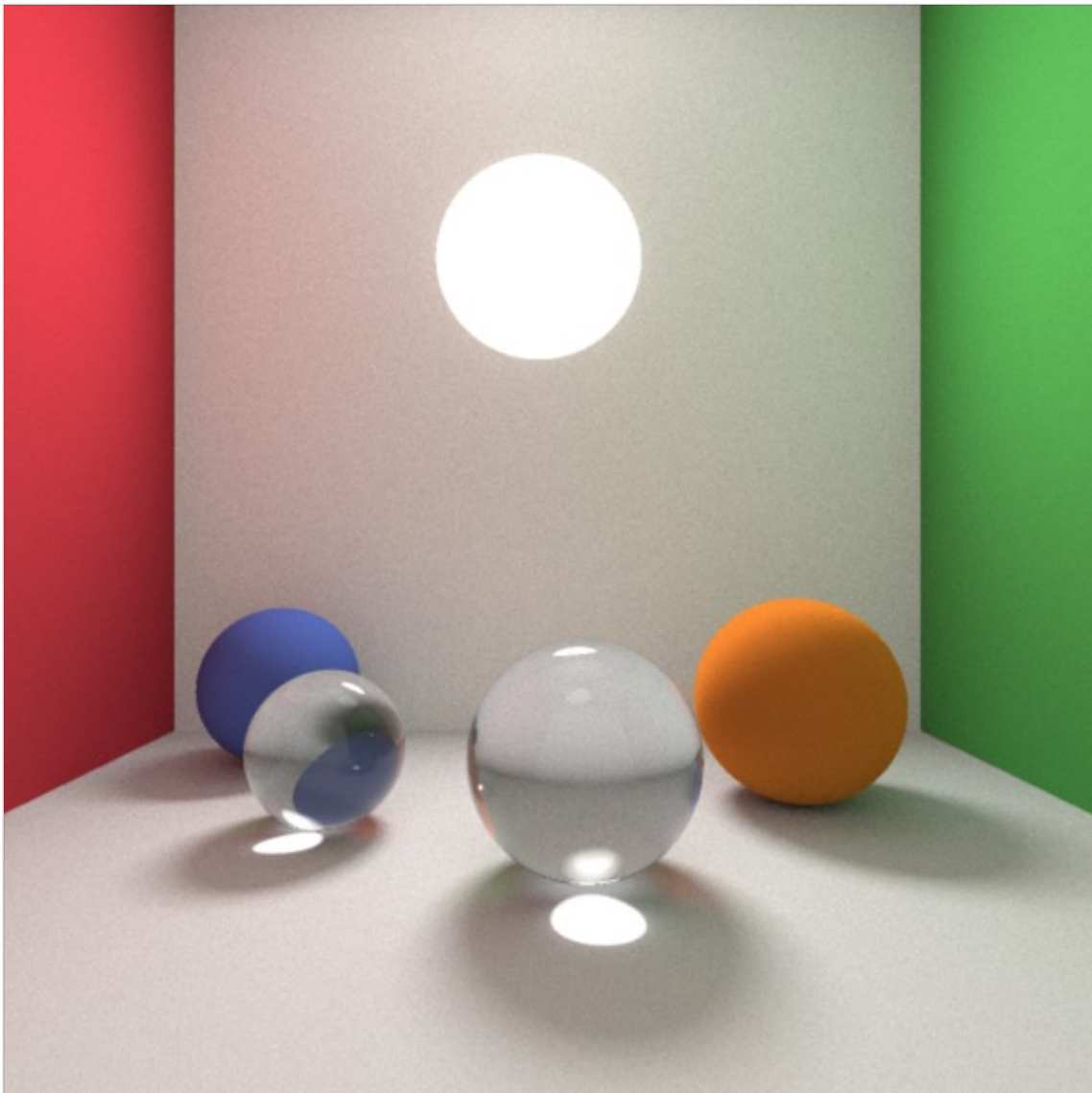


Figure 34. First spectral path tracing scene: 20000 samples per pixel, illuminant D65.

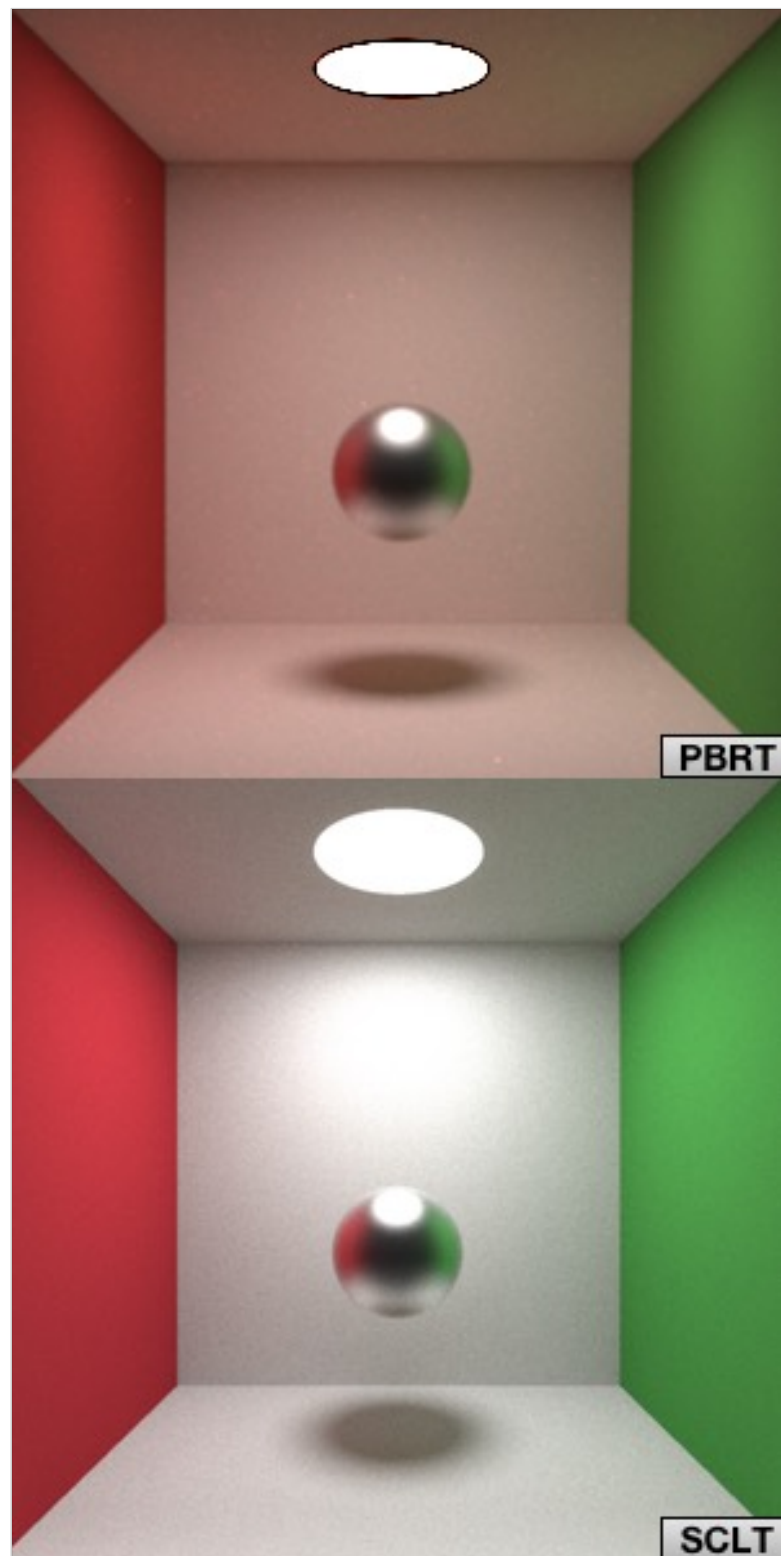


Figure 35. Torrance Sparrow scene: PBRT 1000 samples, SCLT 10000 samples.



Figure 36. Scene of case study 2 with path tracing: 20000 samples, illuminant D65.

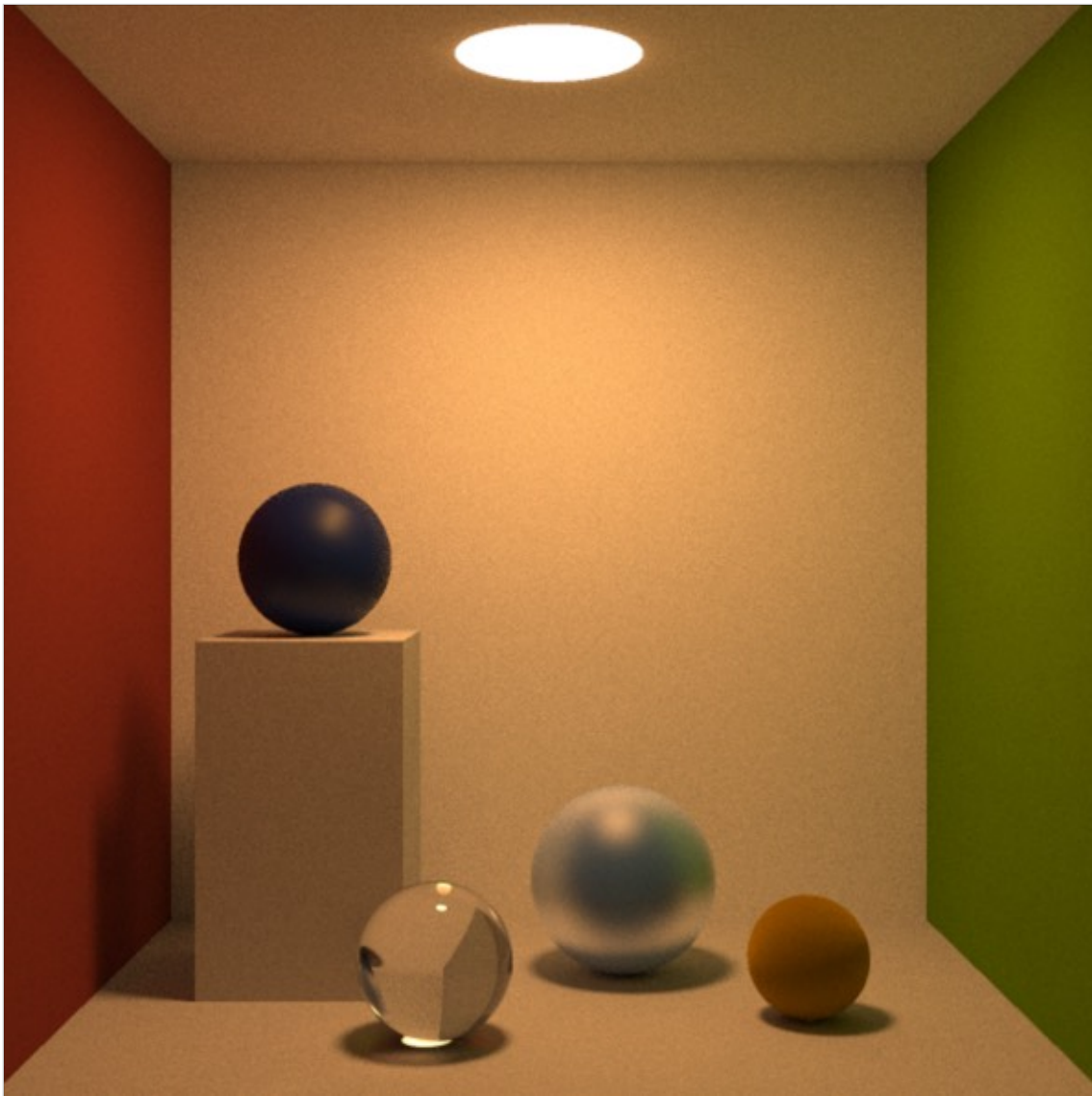


Figure 37. Scene of case study 2 with path tracing: 20000 samples, illuminant FL4.



Figure 38. Spectral path tracing scene with D65 light inside it: 20000 samples.

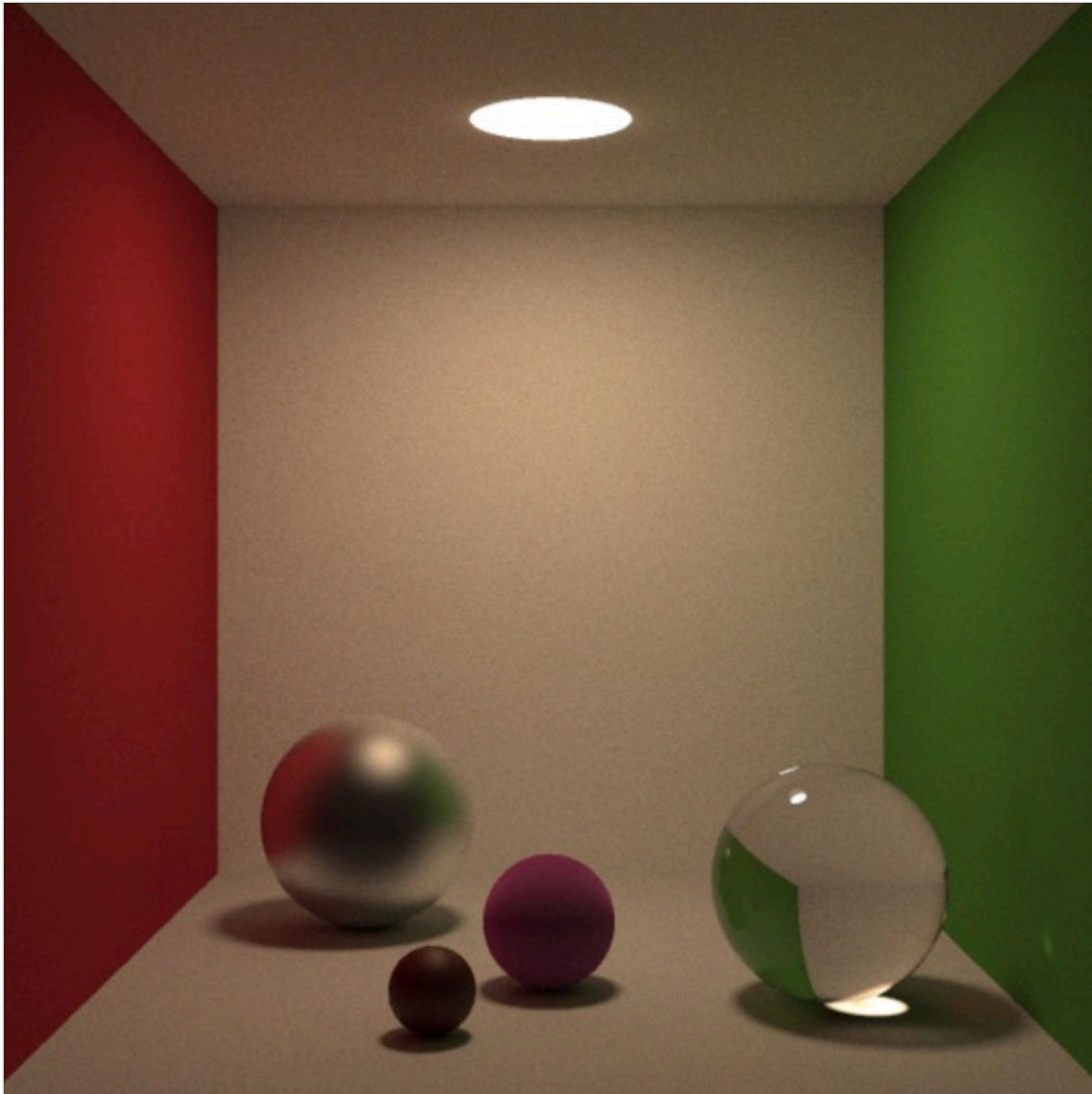


Figure 39. Spectral path tracing scene: 20000 samples per pixel, illuminant FL9.

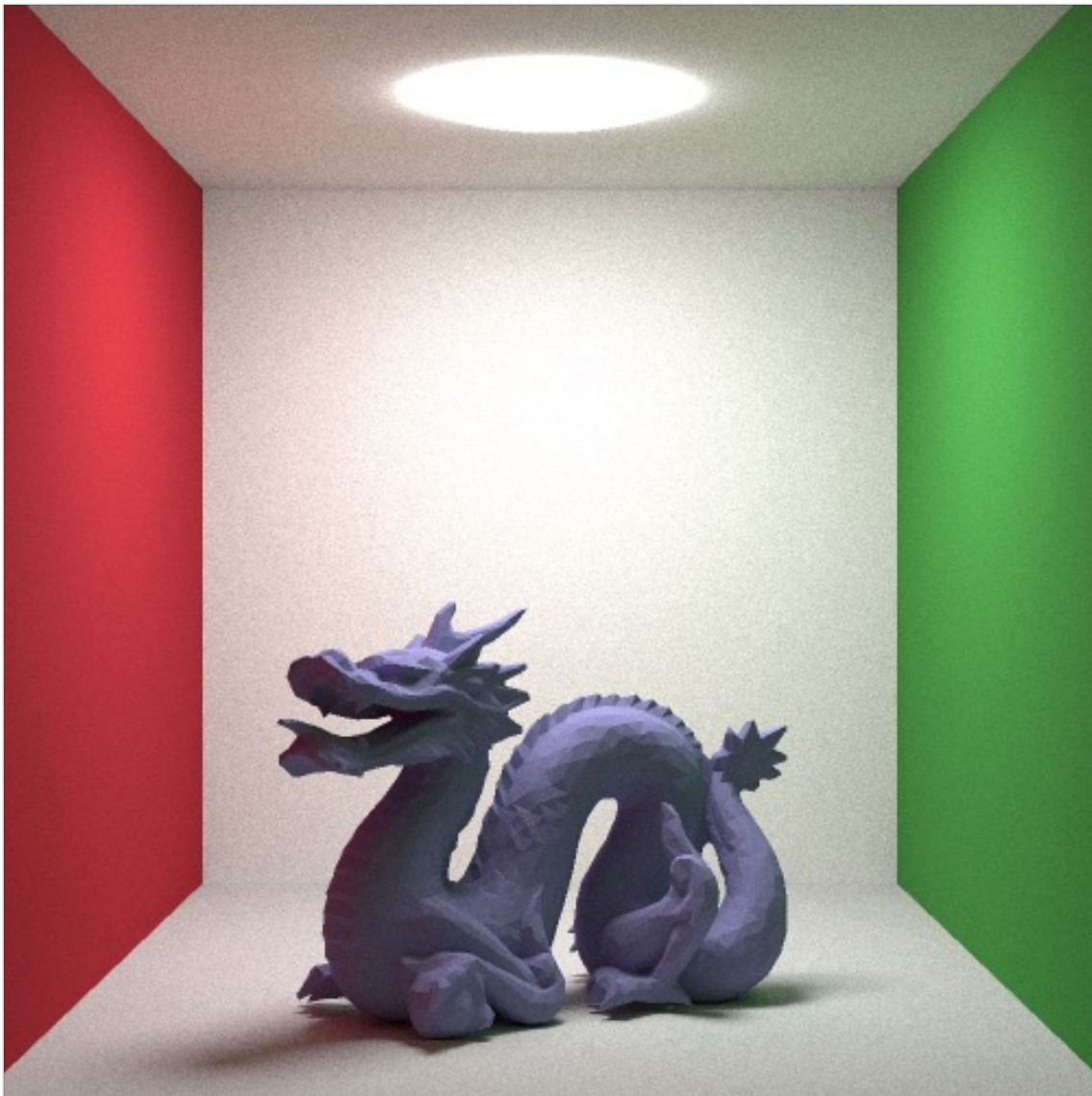


Figure 40. Spectral path tracing scene: mesh (lambertian), 10000 samples.

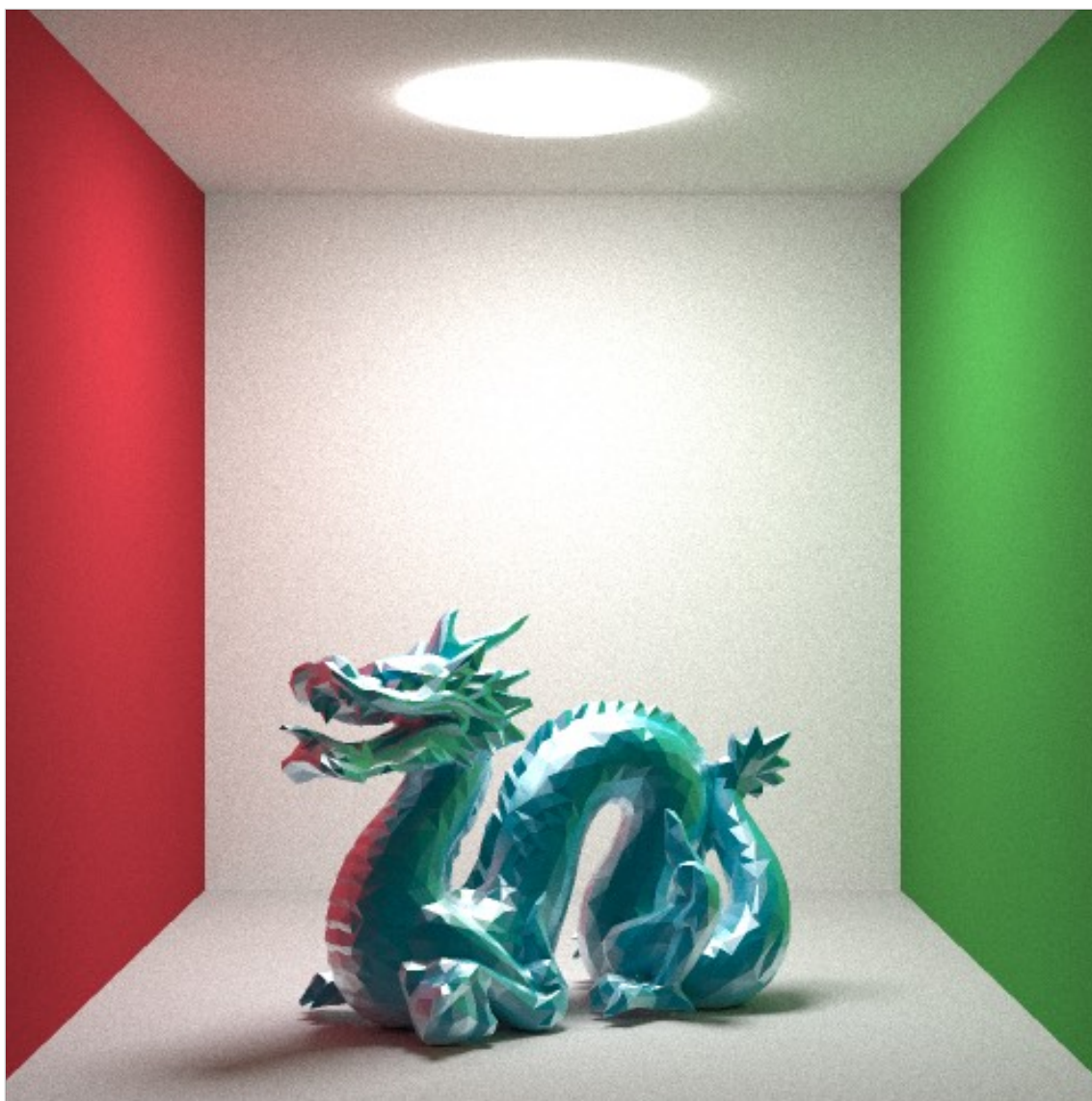


Figure 41. Spectral path tracing scene: mesh (Torrance-Sparrow), 10000 samples.

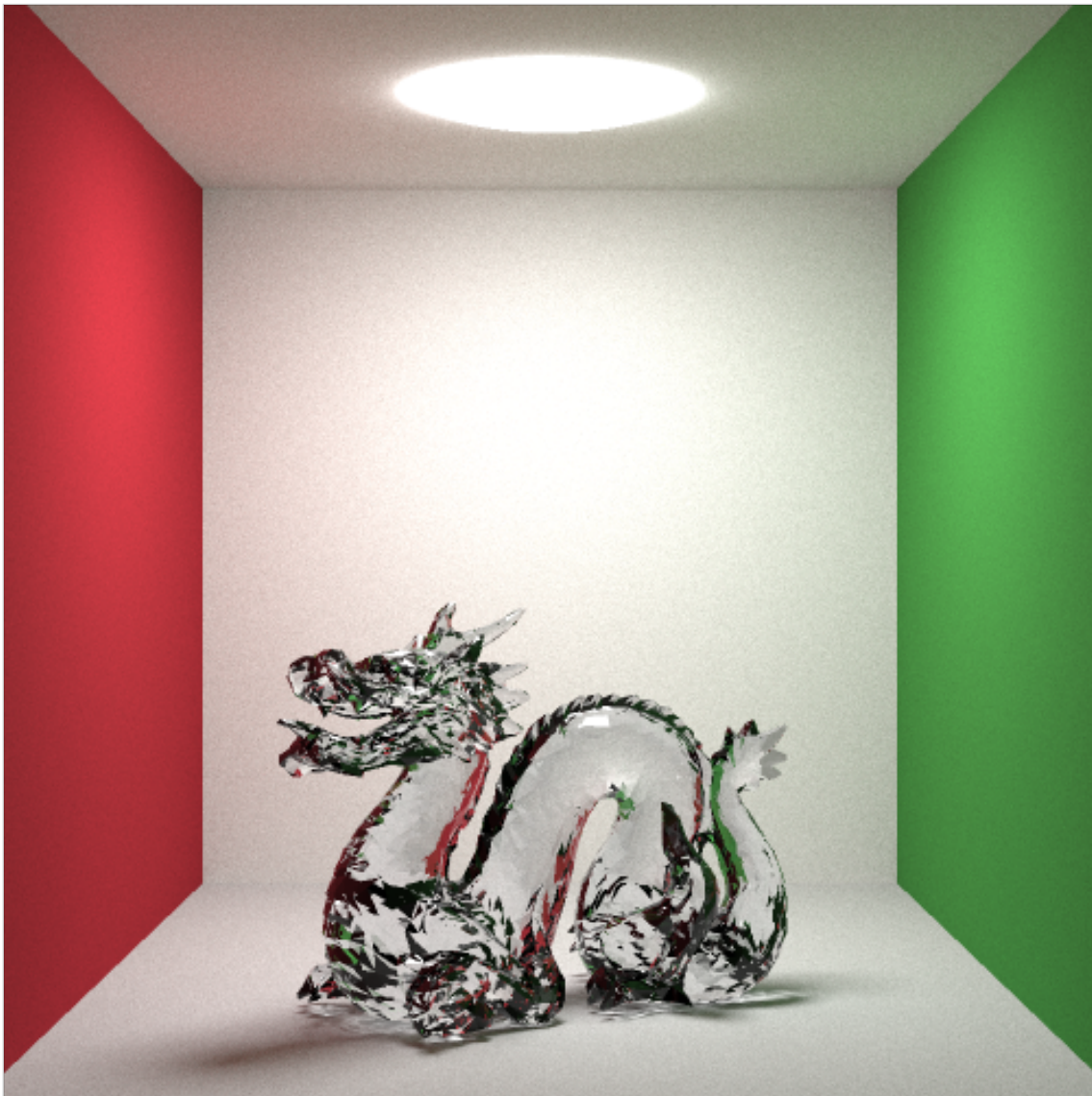


Figure 42. Spectral path tracing scene: mesh (reflection and transmission), 20000 samples.

Chapter 6

SCLT: Color Rendering Index

As reported in the first chapter, the second Grassman's law describes a phenomenon called metamerism, that is the possibility that the same color could be perceived from objects with different SPD when they are illuminated with the same light source that has a specific SPD. This fact is mostly correlated with the use of artificial light sources.

How is it possible to evaluate the ability of a light source to reveal all frequencies of its color spectrum when compared to a reference light? The answer is simple: Color Rendering Index (CRI).

Even if it has been criticized and CIE is trying to replace it with other indexes, for example Color Quality Scale (CQS) and CIECAM02, it is used in standard commercial applications to evaluate fidelity of color of a light source, and it is usually indicated on available commercial light products. In this thesis CRI will be used in a different way. It will be used to evaluate the accuracy with which an illuminant will reproduce color in a scene rendered using SCLT.

CRI is defined to be in a range 0-100: an illuminant with CRI of 100 will reproduce color with an accuracy similar to that of natural light.

SCLT supports CRI calculation using two different methods: Test samples method and R96_a. CIE defined the CRI as [65]:

“Effect of an illuminant on the color appearance of objects by conscious or subconscious comparison with their color appearance under a reference illuminant.”

6.1 Test samples method

Test samples method is the standard procedure used to calculate CRI.

It compares the color rendering of the test source with one obtained from an ideal light based on the CCT of the light source to be tested. It uses samples taken from the Munsell atlas.

The steps needed to obtain the CRI, indicated with R_a , are (CIE Commission Internationale de l'Eclairage [16]):

1. convert the SPD of the light source into tristimulus values using the 2° standard observer;
2. convert the tristimulus values obtained with the previous step into CIE 1931 XYZ chromaticity coordinates;
3. calculate the CIE 1960 color space coordinates (chromaticity coordinate) using the CIE 1931 XYZ Chromaticity coordinates calculated before;
4. calculate the CCT of the test source. This could be estimated from its planckian locus in the CIE 1960 color space coordinate. The planckian locus is the path that a black body radiator follows in a specific chromaticity space. For the purpose of this thesis, it will be calculated with the McCamy's approximation algorithm, with the formula:

$$CCT = -449n^3 + 3525n^2 - 6823.3n + 5520.33 \quad (24)$$

where n is:

$$n = \frac{x - 0.3320}{y - 0.1858} \quad (25)$$

5. choose the reference illuminant: if CCT is under 5000 K use the planck's radiation law (equation (10)) to estimate the SPD of the black body, otherwise use standard illuminant D65;
6. convert the SPD of the samples to tristimulus values using the test light source and the reference light source;
7. apply a Von Kries CAT to each sample obtained from the previous step;
8. convert the samples adapted with Von Kries to the CIE 1964 color space;
9. calculate the Euclidean distance ΔE_i between pair of coordinate of the same sample under the test and under reference;
10. calculate for each sample the special CRI using the following formula:

$$R_i = 100 - 4.6\Delta E_i \quad (26)$$

11. find the general CRI R_a by calculating the arithmetic mean of the special CRIs.

6.2 R96_a method

The R96_a is an update on the CRI described in the previous paragraph. CIE started to work on it in 1991 and released it to the word in the technical committee 1-33. It never reached a real application in industry, due to disagreements between researchers and industries. For this thesis it is interesting to calculate the R96_a index for a simple fact: it uses the SPD of 10 colors taken from the Macbeth color checker as sample for the calculation. These SPDs are the same used for rendering the objects contained in the various scenes presented earlier.

The step needed to calculate the R96_a are (Bodrogi, 2004 [17]):

1. convert the test light source SPD to tristimulus values using the 2° standard observer and convert them into CIE 1931 XYZ Chromaticity coordinates;
2. calculate the CCT of the test source;
3. choose the reference illuminant in the following list, based on the minimum difference with the CCT of the test light source: 2700K, 2950K, 3450K or 4200K black body radiator, D50 (5003K), D65 (6504K);
4. convert each sample SPD under test and reference illuminant to tristimulus values using the 2° standard observer;
5. apply CIECAT94 to adapt each sample under test and reference illuminant to D65;
6. convert the previous obtained samples to CIE 1976 L* a* b* color space;
7. calculate for each sample in CIE 1976 L* a* b* the special CRI with the following formula:

$$R_i = 100 - 3.248\Delta E_i \quad (27)$$

8. find the general CRI $R96_a$ by calculating the arithmetic mean of the special CRIs.

6.3 CRI: implementation details

How does SCLT calculate the CRI?

The implementation is contained in the *ColorRenderingIndex* class. The spectrum of the illuminant of the scene is passed into two functions:

1. *testSampleMethod(Spectrum<constant::spectrumSamples> spectrumTestLight)*
2. *r96aMethod(Spectrum<constant::spectrumSamples> spectrumTestLight)*

These two methods correspond to the two CRI calculation processes previously described: the test samples method and the R96_a method.

As these methods require various color spaces, described in paragraph 2.1, a class for each of them has been implemented, with various conversion methods. These classes are:

1. *CIELab*
2. *CIEUCS*
3. *CIEUVW*
4. *CIE1931XYZ*

6.4 Case study: CRI calculation

As previously seen, SCLT supports different illuminants by default, in particular the illuminant A, the F family, D50 and D65, to let the user render scenes and execute CRI computation for light source with different SPD. This is why some tests have been done to evaluate the difference between the two CRI calculation methods previously showed, and the accuracy of SCLT with respect to the expected CRI value for the illuminants supported by default.

As for the color calculation in the physically based spectral rendering showed before, all the calculation has been done using 81 samples for each SPD taken from the Munsell or from the Macbeth color checker.

Obviously, as SCLT supports also RGB scene/rendering, a check is done if the user tries to calculate the CRI on a scene of this type: it could be calculated only on spectral scenes.

The first set of illuminants tested is the F family. This is not a random choice: in this family there is the FL4 illuminant, used by the Commission International de l'Eclairage to calibrate the Color

Rendering Index calculation in the test samples method. In the following table the value obtained from SCLT for the two methods and the expected value for the test method samples are reported.

Illuminant	CRI test samples method - CIE expected value	SCLT CRI test method samples	SCLT CRI R96a
FL1	76	74.67	69.75
FL2	64	64.64	69.73
FL3	57	57.23	64
FL4	51	50.92	57.60
FL5	72	64.12	58.32
FL6	59	58.49	55.53
FL7	90	90.31	88.91
FL8	95	88.13	96.13
FL9	90	90.48	89.74
FL10	81	79.77	81.27
FL11	83	83.04	75.19
FL12	83	83.22	78.26

Table 2. Color Rendering Index for the F family of illuminants.

As could be seen from the data, SCLT has generally a high degree of accuracy if a comparison is done between the CIE test samples method expected values and the SCLT computed values.

Only a discrepancy on illuminant FL5 and FL8 could be noted. This could be accounted to the SPD data used for these two illuminants, maybe not so precise and accurate. From the same table it is worth noting that the R96a method generates discontinuous values, with discrepancy with almost all the CRI test samples method.

The fact that this method has a higher computational cost than the test samples method, due to the use of the CIECAT94, plus the fact that CIE has never approved it as a standard, leave its implementation to be just a “toy experiment” to be used only for evaluation and comparison with the standard test samples method. It must also be noted that the CIECAT94 requires a lot of parameters whose values are difficult to be retrieved. SCLT sets them to a default value. For

example, the adapting luminance of the test field is defined to have the same value of the adapting luminance of the reference field.

Illuminant	CRI test samples method - CIE expected value	SCLT CRI test method samples	SCLT CRI R96a
A	98	99.74	86.52
D65	100	100	100

Table 3. Color Rendering Index for A and D65 illuminant.

Table 3 shows the CRI for A and D65 illuminants. Again the SCLT implementation of the test samples method is highly accurate. Finally, the D65 gives a value of 100: as it represents the SPD of a natural light, it accounts for the maximum level of color rendering accuracy and the CRI value remarks this fact.

Figure 43 shows an example of SCLT CRI calculation. In Figure 44 all the spectral scene rendered with the Whitted model are presented with their CRI obtained from SCLT.

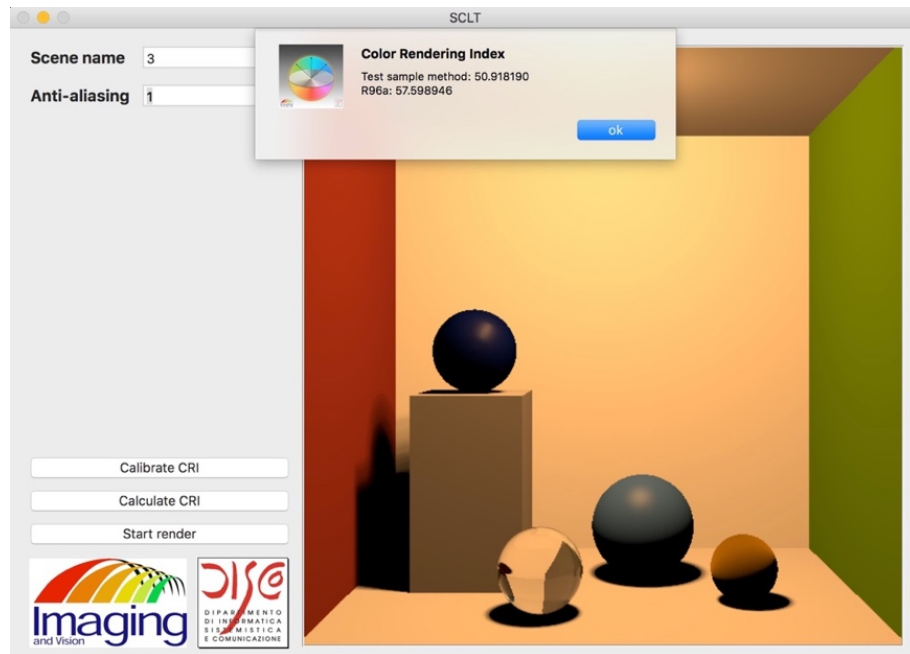


Figure 43. Example of CRI calculation.

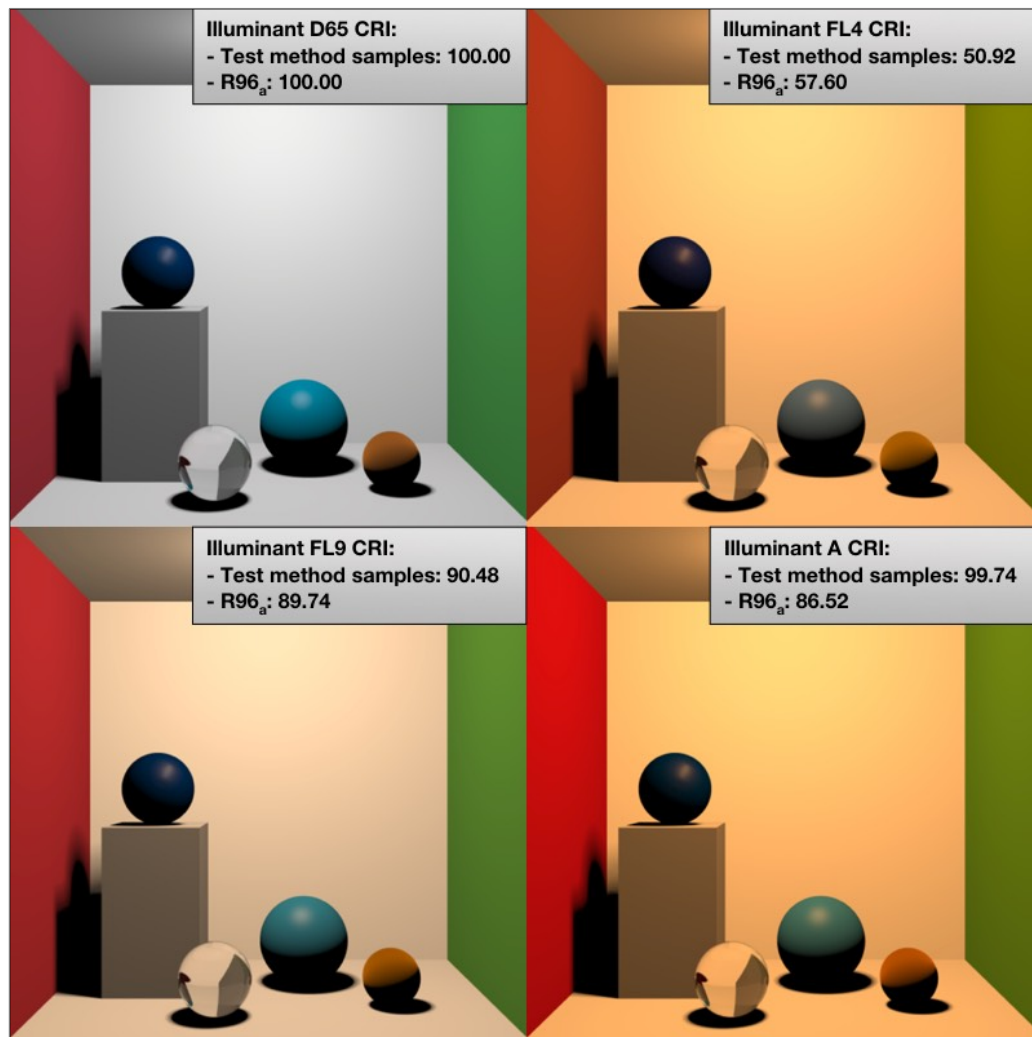


Figure 44. Whitted scenes with their CRI obtained from SCLT.

Chapter 7

Conclusion

Physically based rendering is the future of computer graphics.

SCLT shows how these techniques could improve the images rendered in a ray tracing engine. Its main purpose is to render PBR scenes using spectral data to achieve the maximum level of color fidelity.

In fact, the use of tristimulus values as a basis for color calculation gives SCLT the ability to explore phenomenon like metamerism, difficult to be showed with standard RGB color processing. The ability to calculate the CRI for an illuminant gives to the user the ability to evaluate the general color fidelity of a scene rendered.

The ability to render PBR scenes using spectral data, and the CRI calculation support, let SCLT be a perfect tool for industrial light design and production. In fact, SCLT could be used to quantify the ability of a light product to reveal colors. The support of multiple devices and operating systems gives to the user the freedom to choose the preferred platform.

Even if some great features has been showed during this thesis, SCLT is far from being a perfect 3D engine. A lot of improvements could be added and explored to let SCLT become a real performant engine, useful not only in education but also in a production environment.

In particular, the main focus area to be improved are:

- rendering speed and performance;
- additional BRDF models;
- light source.

For what concerns speed and performance, at the moment SCLT is slow.

In fact, to obtain the path tracing images, with a number of samples like the one showed in the previous chapters, SCLT takes a lot of hours of computation. The improvements in this area could be of two types.

The first one is related to the ray tracing models implemented, in particular path tracing. At the moment SCLT path tracing is the standard version of this algorithm, that doesn't account for any kind of explicit direct light calculation. Other engines, like for example the previously cited PBRT, or smallPT in one of its variant [66], take into account this kind of calculation. Another technique, Russian roulette, can be added to modify the standard path tracing algorithm.

The second one is related to the ray intersection test. As stated in paragraph 5.3, this part of a ray tracing engine is the most computational expensive one. In fact, for each ray, the intersection test has $O(n)$ complexity, where n is the number of objects to be tested. To improve this part some data structure, called acceleration data structures, could be used to speed up the process:

- k -d tree, a data structure to organize the scene into a hierarchy of spatial set subdivisions;
- bounding volume hierarchy, a data structure to organize the scene into a hierarchy of primitive set subdivisions.

These structures could bring the intersection test to time complexity $O(\log n)$.

The second area of improvement is related to the BRDF models supported by SCLT: at the moment there are only isotropic BRDFs. It is possible to implement additional anisotropic models, like for example, Ward BRDF, introduced by Gregory J. Ward in 1992 [67], and Ashikhmin-Shirley BRDF, introduced by Michael Ashkhmin and Peter Shirley in 2000 [68]. In this way SCLT could be able to render a wider set of surfaces (e.g. brushed metals).

The third area of improvement is related to light. First of all, at the moment SCLT doesn't implement any kind of light sampling, like for example the PBRT engine. SCLT also uses an empirical brightness parameter to adjust the total brightness of light in a scene for the path tracing model. This could be substituted with a more realistic calculation, based on the various characteristics of a light source like its area and power.

Even if some of the previous showed enhancement could improve the general experience in using SCLT, it remains a good starting point for everyone interested in the study of physically based rendering and colorimetry.

As previously stated, SCLT is also a good education project: all its source code is open source and has been published on a Github repository under MIT License (<https://github.com/chicio/Spectral-Clara-Lux-Tracer>).

Appendix A

Spherical coordinates

A vector in a three dimension space, described with the cartesian coordinate system, is defined as $v = (v_x, v_y, v_z)$, with each component related to a specific dimension. In BRDF calculation it is useful to described a vector also in the spherical coordinate system.

With this kind of system, a vector is specified with a magnitude ρ and a pair of angle θ and ϕ . The first one is the polar angle, measured from a zenith direction. The second one is the azimuth angle of the vector projection on a tangent plane to the origin and orthogonal to the zenith, measured from a fixed reference direction on that plane.

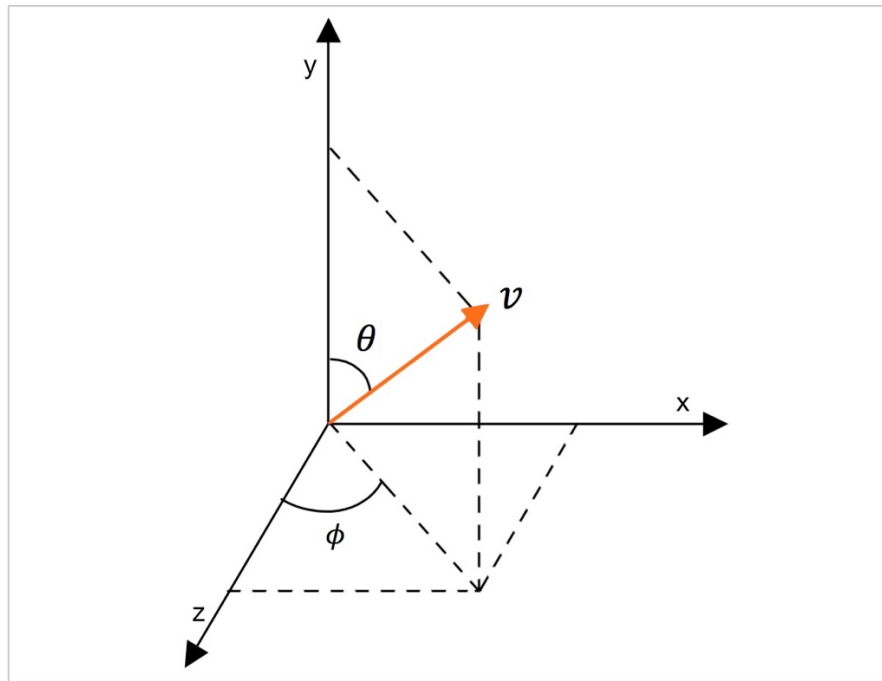


Figure 45. Spherical coordinates representation.

Assuming to have a normalized vector, and assuming a coordinates system with the up direction described by the y axis (as seen in the previous image) the formulas used to convert it from cartesian to spherical coordinate system are:

$$\phi = \arctan \frac{v_z}{v_x} \quad (\text{A1})$$

$$\theta = \arccos v_y \quad (\text{A2})$$

The inverse conversion, again assuming to have a normalized vector, from spherical to cartesian coordinate is possible using the following equation:

$$v_x = \cos \phi \sin \theta \quad (\text{A3})$$

$$v_y = \cos \theta \quad (\text{A4})$$

$$v_z = \sin \phi \sin \theta \quad (\text{A5})$$

Appendix B

SCLT scene XML file format

SCLT uses its own XML format to define scene. Each one of the tags contained in these XML files is related to: a component/object of the scene, an SCLT configuration property or a scene specific configuration property. In the following a list of these tags and their attributes is reported. This section could be considered a user guide manual.

B.1 scene

The scene tag is the root node of SCLT xml file format. It contains all the other tags that define a SCLT scene.

Attribute	Value	Description
<i>Type</i>	String: RGB or Spectral.	Define the type of scene. If not specified, default is Spectral.

Table 4. scene tag attributes list.

B.2 camera

The camera tag contains all the tags that define the information needed to create a camera in SCLT. This tag has no attributes. There must be only one camera tag for scene XML file.

B.3 viewReferencePoint

The viewReferencePoint tag defines a view reference point for the camera. It must be contained in a camera tag.

Attribute	Value	Description
x	float	x coordinate.
y	float	y coordinate.
z	float	z coordinate.

Table 5. viewReferencePoint tag attributes list.

B.4 lookAtPoint

The lookAtPoint tag defines a look at point for the camera. It must be contained in a camera tag.

Attribute	Value	Description
x	float	x coordinate.
y	float	y coordinate.
z	float	z coordinate.

Table 6. lookAtPoint tag attributes list.

B.5 viewPlane

The viewPlane tag defines the viewPlane. It must be contained in a camera tag. At the moment of this writing only the distance of the view plane could be customized in the scene XML file.

Attribute	Value	Description
d	float	View plane distance.

Table 7. viewPlane tag attributes list.

B.6 light

The light tag defines the light source of the scene. At the moment SCLT supports only one light per scene.

Attribute	Value	Description
<i>type</i>	String: area or point.	Defines the type of light used in the scene.
<i>spectrum</i>	String: illuminant spectrum name.	Defines the illuminant to be used in the scene. Used in spectral scene only.
<i>brightness</i>	Float	Brightness of the light (path tracing).

Table 8. light tag attributes list.

B.7 tracerModelType

The tracerModelType tag defines the type of tracer to be used to render the scene. This tag could have one of three values, related to the models supported by SCLT:

- TracerRGBModel;
- TracerSpectrumModel;
- PathTracerModel.

Attribute	Value	Description
<i>numberOfSamples</i>	float	Number of sample per pixel to be taken. (path tracing)

Table 9. tracerModelType attributes list.

B.8 shadingModelType

The shadingModelType defines the type of shading model to be used to render the scene. This tag could have one of three values, related to the models supported by SCLT:

- WhittedShadingModel;
- WhittedBRDFShadingModel;
- PathTracingBRDFShadingModel.

B.9 cubeMappingSkybox

The cubeMappingSkybox tag defines a skybox at infinite distance, using cube mapping texture technique, for RGB scenes. Its value could be TRUE or FALSE. If not defined, the default value is false and the cube mapping textures are not loaded.

The texture needed to create the skyboxes must be placed in the scenes folder and with name of the side of the skybox they represent.

B.10 empiricalLighting

The empiricalLighting tag defines the empirical light model to be used.

This tag is meaningful only for RGB scenes, as spectral scene uses physically based BRDF models. This tag could have two values:

- Phong;
- BlinnPhong.

B.11 objects

The object tag contains all object tags, that represent object definition. Every scene must contain an objects tag.

B.12 object

The object tag defines an object to be added to the scene.

Attribute	Value	Description
<i>type</i>	String: sphere, square and polygonal shape.	Defines the type of the object.
<i>skyboxSide</i>	String: bottom, left, right, back, front and top.	Defines the objects as a side of a skybox. Useful only on square objects.

Table 10. object tag attributes list.

B.13 origin

The origin tag is used to define of light and sphere object.

Attribute	Value	Description
x	float	x coordinate.
y	float	y coordinate.
z	float	z coordinate.

Table 11. origin tag attributes list.

B.14 radius

The radius tag is used to define the radius of light and sphere objects.

Attribute	Value	Description
r	float	Defines the radius measure.

Table 12. radius tag attributes list.

B.15 color

The color tag defines the color of the RGB color of the light. This tag is useful only in RGB scene, and is contained inside a light tag.

Attribute	Value	Description
r	float	Red color component.
g	float	Green color component.
b	float	Blue color component.

Table 13. color tag attributes list.

B.16 vertex

The vertex tag defines a vertex of square or polygonal shape objects. This tag is useful only inside an object tag define with a type square or polygonalshape values.

Attribute	Value	Description
<i>x</i>	float	x coordinate.
<i>y</i>	float	y coordinate.
<i>z</i>	float	z coordinate.

Table 14. vertex tag attributes list.

B.17 pointOnPlane

The pointOnPlane tag defines a point on plane associated with square or polygonal shape objects. This tag is useful only inside an object tag define with a type square or polygonalshape values.

Attribute	Value	Description
<i>x</i>	float	x coordinate.
<i>y</i>	float	y coordinate.
<i>z</i>	float	z coordinate.

Table 15. pointOnPlane tag attributes list.

B.18 material

The material tag determines the appearance of an object. It is useful only if defined inside an object tag.

Attribute	Value	Description
<i>type</i>	String with type of material. RGB scene: <ul style="list-style-type: none">• jade	Defines the type of material associated to an object.

	<ul style="list-style-type: none"> • bronze • violet • green • red • mediumGray • lightGray • matte • matteTextured • chrome • silve • glass • glasswater • rubyBumpMapped • flameMarble • blueTurbulence <p>Spectral scene:</p> <ul style="list-style-type: none"> • emissive • matteLambertian • matteOrenNayar • measured • plastic • glass • mirror 	
<i>textureName</i>	String	Defines the texture to be used. Textures are supported only in RGB scene.
<i>scale</i>	float	Defines the scale used with rubyBumpMapped material.
<i>spectrum</i>	String	Defines the spectrum used with matte lambertian or matte Oren Nayar spectral material.

<i>spectrumDiffuse</i>	String	Defines the diffuse spectrum used with plastic spectral material.
<i>spectrumSpecular</i>	String	Defines the spectrum used by plastic spectral material.
<i>degree</i>	float	Defines the degree used with matte Oren Nayar.
<i>name</i>	String	Defines the name of measured BRDF for measured material.
<i>interpolated</i>	bool	Defines if a measured BRDF material data must be interpolated.

Table 16. material tag attributes list.

B.19 mesh

The mesh tag is used to define an OBJ mesh to render in the scene.

Mesh must always contain an aabb tag, to define an axis aligned bounding box used to speed up the intersection test, and an objFile tag to get the OBJ file that contains the model. This tag has no attribute.

B.20 objFile

The objFile tag is used to define the OBJ model file used in a mesh.

Attribute	Value	Description
<i>name</i>	String	Defines the name of the mesh OBJ file to be loaded.
<i>backFaceCulling</i>	Bool	Defines if the polygons created from the OBJ file are face culled.

Table 17. objFile tag attribute list.

B.21 aabb

The aabb tag is used to define an axis aligned bounding box. This tag has no attribute.

B.22 min

The min tag is used inside an aabb tag to define the min extent of an AABB.

Attribute	Value	Description
<i>x</i>	float	x coordinate.
<i>y</i>	float	y coordinate.
<i>z</i>	float	z coordinate.

Table 18. min tag attribute list.

B.23 max

The max tag is used inside an aabb tag to define the max extent of an AABB.

Attribute	Value	Description
<i>x</i>	float	x coordinate.
<i>y</i>	float	y coordinate.
<i>z</i>	float	z coordinate.

Table 19. max tag attribute list.

B.24 RGB scene XML example

The following snippet show an example of a RGB scene XML file.

```
<scene type="RGB">
  <camera>
    <viewReferencePoint x="215" y="250" z="490" />
    <lookAtPoint x="250" y="250" z="250" />
    <viewPlane d="240" />
  </camera>
  <light type="area">
    <origin x="0" y="300" z="400" />
    <radius r="60" />
    <color r="255" g="255" b="255" />
  </light>
```

```

<cubeMappingSkybox>TRUE</cubeMappingSkybox>
<tracerModelType>TracerRGBModel</tracerModelType>
<shadingModelType>WhittedShadingModel</shadingModelType>
<empiricalLighting>Phong</empiricalLighting>
<objects>
  <object type="sphere">
    <origin x="300" y="300" z="250" />
    <radius r="90" />
    <material type="glass" />
  </object>
  <object type="sphere">
    <origin x="230" y="380" z="50" />
    <radius r="100" />
    <material type="jade" />
  </object>
  <object type="sphere">
    <origin x="110" y="150" z="150" />
    <radius r="90" />
    <material type="bronze" />
  </object>
  <object type="sphere">
    <origin x="340" y="130" z="250" />
    <radius r="60" />
    <material type="rubyBumpMapped" scale="4.0" />
  </object>
</objects>
</scene>

```

Code snippet 1. SCLT RGB XML scene file example.

B.25 Spectral scene XML example

The following snippet show an example of a spectral scene XML file.

The scene uses Path tracing as tracer model and shows almost all available tags and attributes for spectral rendering.

```

<scene type="Spectral">
  <camera>
    <viewReferencePoint x="512" y="250" z="1023" />
    <lookAtPoint x="512" y="250" z="512" />
    <viewPlane d="410" />
  </camera>
  <light type="area" brightness="50" spectrum="d65">
    <origin x="500" y="650" z="-990" />
    <radius r="1000" />
  </light>
  <tracerModelType>
    PathTracerModel
  </tracerModelType>
</scene>

```

```

</tracerModelType>
<shadingModelType numberOfSamples="40">
  PathTracingBRDFShadingModel
</shadingModelType>
<objects>
  <object type="polygonalshape">
    <vertex x="0" y="0" z="0" />
    <vertex x="0" y="0" z="1024" />
    <vertex x="1024" y="0" z="1024" />
    <vertex x="1024" y="0" z="0" />
    <pointOnPlane x="512" y="0" z="512" />
    <material type="matteLambertian" spectrum="neutral8" />
  </object>
  <object type="polygonalshape">
    <vertex x="0" y="0" z="0" />
    <vertex x="0" y="1024" z="0" />
    <vertex x="0" y="1024" z="1024" />
    <vertex x="0" y="0" z="1024" />
    <pointOnPlane x="0" y="512" z="512" />
    <material type="matteLambertian" spectrum="red" />
  </object>
  <object type="polygonalshape">
    <vertex x="1024" y="0" z="0" />
    <vertex x="1024" y="0" z="1024" />
    <vertex x="1024" y="1024" z="1024" />
    <vertex x="1024" y="1024" z="0" />
    <pointOnPlane x="1024" y="512" z="512" />
    <material type="matteLambertian" spectrum="green" />
  </object>
  <object type="polygonalshape">
    <vertex x="0" y="0" z="0" />
    <vertex x="1024" y="0" z="0" />
    <vertex x="1024" y="1024" z="0" />
    <vertex x="0" y="1024" z="0" />
    <pointOnPlane x="512" y="512" z="0" />
    <material type="matteLambertian" spectrum="white" />
  </object>
  <object type="polygonalshape">
    <vertex x="0" y="1024" z="1024" />
    <vertex x="0" y="1024" z="0" />
    <vertex x="1024" y="1024" z="0" />
    <vertex x="1024" y="1024" z="1024" />
    <pointOnPlane x="512" y="1024" z="512" />
    <material type="matteLambertian" spectrum="neutral8" />
  </object>
  <object type="polygonalshape">
    <vertex x="0" y="0" z="1024" />
    <vertex x="0" y="1024" z="1024" />
    <vertex x="1024" y="1024" z="1024" />
    <vertex x="1024" y="0" z="1024" />
    <pointOnPlane x="512" y="512" z="1024" />
    <material type="matteLambertian" spectrum="white" />
  </object>

```

```
<object type="sphere">
  <origin x="760" y="100" z="280" />
  <radius r="100" />
  <material type="matteOrenNayar"
    spectrum="orange"
    degree="50"/>
</object>
<object type="sphere">
  <origin x="540" y="90" z="450" />
  <radius r="90" />
  <material type="glass" />
</object>
<object type="sphere">
  <origin x="313" y="70" z="350" />
  <radius r="70" />
  <material type="glass" />
</object>
<object type="sphere">
  <origin x="200" y="90" z="150" />
  <radius r="90" />
  <material type="matteLambertian"
    spectrum="purplishBlue" />
</object>
</objects>
</scene>
```

Code snippet 2. SCLT Spectral XML scene file example.

Appendix C

SCLT source code

The entire SCLT source code is available on Github at the following url:

<https://github.com/chicio/Spectral-Clara-Lux-Tracer>

The entire source source code has been released under MIT License.

Third party library are reported (inside the code) with their specific license. Hereunder is reported the SCLT license.

The MIT License (MIT)

Copyright (c) 2016 Fabrizio Daroni.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,

WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.



References

- [1] Lowel-Light Mfg., “Color Temperature & Color Rendering Index DeMystified,” The Tiffen Company, 5 August 2014. [Online]. Available: http://lowel.tiffen.com/edu/color_temperature_and_rendering_demystified.html. [Accessed 8 June 2016].
- [2] Walt Disney Animation Studios, “Hyperion,” Walt Disney Animation Studios, 2 August 2015. [Online]. Available: <http://www.disneyanimation.com/technology/innovations/hyperion>. [Accessed 10 July 2016].
- [3] C. Eisenacher, G. Nichols, A. Selle and B. Burley, “Sorted deferred shading for production path tracing,” in *EGSR '13*, Aire-la-Ville, 2013, pp. 125-132.
- [4] Pixar Animation Studios, “About RIS,” Pixar Animation Studios, 1 February 2015. [Online]. Available: <https://renderman.pixar.com/resources/current/RenderMan/risOverview.html>. [Accessed 10 July 2016].
- [5] Otoy Inc., “Brigade: real-time path tracing,” Otoy Inc., 14 February 2015. [Online]. Available: <https://home.otoy.com/render/brigade/>. [Accessed 16 June 2016].
- [6] LuxRender Developer Community, “LuxRender: GPL Physically Based Renderer,” 11 February 2011. [Online]. Available: http://www.luxrender.net/en_GB/index. [Accessed 18 June 2016].
- [7] M. Pharr and G. Humphreys, “pbrt-v2,” Pharr, Matt and Humphreys, Greg, 6 October 2009. [Online]. Available: <https://github.com/mmp/pbrt-v2>. [Accessed 2 November 2015].
- [8] M. Pharr and G. Humphreys, “Scenes,” Pharr, Matt and Humphreys, Greg, 29 June 2010. [Online]. Available: <http://www.pbrt.org/scenes.html>. [Accessed 10 July 2016].
- [9] M. Oren and K. S. Nayar, “Generalization of Lambert’s Reflectance Model,” in *SIGGRAPH '94*, Orlando, 1994, pp. 239-246.
- [10] K. E. Torrance and E. M. Sparrow, “Theory for off-specular reflection from roughened surfaces,” *JOSA*, vol. 57, no. 9, pp. 1105-1114, 1967.

- [11] S. R. Marschner, "Inverse rendering for computer graphics," PhD Thesis, Cornell Univ., Dept. Computer Graphics, New York, 1998.
- [12] Cornell Univ. Computer Graphics Dept., "Reflectance Data: Cornell University Program of Computer Graphics," Cornell Univ., 01 January 1998. [Online]. Available: <http://www.graphics.cornell.edu/online/measurements/>. [Accessed 10 February 2016].
- [13] T. J. Whitted, "An improved illumination model for shaded display," *Communications of the ACM*, vol. 23, no. 6, pp. 343-349, 6 June 1980.
- [14] T. J. Whitted, "An improved illumination model for shaded display," in *SIGGRAPH '05*, Los Angeles, 2005, art. no. 4.
- [15] J. T. Kajiya, "The Rendering Equation," in *SIGGRAPH '86*, Dallas, 1986, pp. 143-150.
- [16] CIE Commission Internationale de l'Eclairage, "Method of Measuring and Specifying Colour Rendering Properties of Light Sources," Tech. Rep. 13.3-1995, CIE Commission Internationale de l'Eclairage, Vienna, 1995.
- [17] P. Bodrogi, "Colour rendering: past, present(2004), and future," in *CIE Expert Symposium on LED Light Sources: Physical Measurement and Visual and Photobiological Assessment*, Tokyo, 2004, pp. 10-12.
- [18] S. Watanabe, S. Kanamori, S. Ikeda, B. Raytchev, T. Tamaki and K. Kaneda, "Performance Improvement of Physically based spectral rendering using stochastic sampling," in *CCIW'13 Proceedings of the 4th international conference on Computational Color Imaging*, Chiba, 2013, pp. 184-198.
- [19] H. Kolb, "Simple Anatomy of the Retina by Helga Kolb," Moran Eye Center, 1 January 2012. [Online]. Available: <http://webvision.med.utah.edu/book/part-i-foundations/simple-anatomy-of-the-retina/>. [Accessed 28 April 2016].
- [20] P. Ronan, "File:EM spectrum.svg," Wikimedia Foundation, 5 August 2007. [Online]. Available: https://en.wikipedia.org/wiki/File:EM_spectrum.svg. [Accessed 30 April 2016].
- [21] C. J. Bartleson, "Colorimetry," in *Optical Radiation Measurements*, New York, Academic Press, 1980, pp. 33-148.
- [22] H. R. Kang, "Tristimulus specification," in *Computation color technology*, Bellingham, Washington: SPIE Publications, 2006, ch. 1, pp. 1-16.

- [23] BenRG, "File:CIE1931xy blank.svg," Wikimedia Foundation, 25 September 2009. [Online]. Available: https://commons.wikimedia.org/wiki/File:CIE1931xy_blank.svg. [Accessed 14 February 2016].
- [24] D. L. MacAdam, "Projective Transformations of I. C. I. Color Specifications," *JOSA*, vol. 27, no. 8, pp. 294-299, 1 August 1937.
- [25] G. Wyszecki, "Proposal for a New Color-Difference Formula," *JOSA*, vol. 53, no. 11, pp. 1318-1319, 1963.
- [26] X-Rite Inc., "A Guide to Understanding Color Communication," X-Rite Inc., 14 October 2007. [Online]. Available: https://www.xrite.com/documents/literature/en/110-001_understand_color_en.pdf. [Accessed 30 October 2015].
- [27] B. J. Lindbloom, "XYZ to Lab," 20 April 2003. [Online]. Available: http://www.brucelindbloom.com/index.html?Eqn_XYZ_to_Lab.html. [Accessed 10 March 2016].
- [28] CIE Commission Internationale de l'Eclairage, "CIE Standard Illuminants for Colorimetry," 03 June 1999. [Online]. Available: <http://www.cie.co.at/publ/abst/s005.html>. [Accessed 26 May 2016].
- [29] D. B. Judd, D. L. MacAdam, G. Wyszecki, H. W. Budde, H. R. Condit, S. T. Henderson and J. L. Simonds, "Spectral Distribution of Typical Daylight as a Function of Correlated Color Temperature," *JOSA*, vol. 54, no. 8, pp. 1031-1040, 1964.
- [30] CIE Commission Internationale de l'Eclairage, "Colorimetry," Tech. Rep. 15:2004, CIE Commission Internationale de l'Eclairage, Vienna, 2004.
- [31] C. S. McCamy, H. Marcus and J. G. Davidson, "A Color-Rendition Chart," *Journal of Applied Photographic Engineering*, vol. 2, no. 3, pp. 95-99, 1976.
- [32] D. Pascale, "RGB coordinates of the Macbeth ColorChecker," The BabelColor Company, 1 June 2006. [Online]. Available: <http://www.babelcolor.com>. [Accessed 13 November 2015].
- [33] CIE Commission Internationale de l'Eclairage, "A review of the chromatic adaptation transform," Tech. Rep. CIE 160:2004, CIE Commission Internationale de l'Eclairage, Vienna, 2004.

- [34] S. Westland and C. Ripamonti, "Chromatic-adaptation Transforms and Colour Appearance," in *Computational color science using MATLAB*, Chichester, West Sussex: John Wiley & Sons, 2004, ch. 6, pp. 86-88.
- [35] M. Pharr and G. Humphreys, "Color and radiometry," in *Physically based rendering: from theory to implementation*, 2nd Edition ed., Burlington, Massachusetts: Morgan Kaufmann, 2010, ch. 5, pp. 261-297.
- [36] M. Pharr and G. Humphreys, "Light transport I: surface reflection," in *Physically based rendering: from theory to implementation*, 2nd ed., Burlington, Morgan Kaufmann, 2010, ch. 15, pp. 760-770.
- [37] E. P. Lafortune and Y. D. Willems, "Bi-Directional Path Tracing," in *Compugraphics '93*, Alvor, 1993, pp. 145-153.
- [38] H. Jang, "Indoor lighting," 1 January 2003. [Online]. Available: <http://hinjang.com/articles/03.html>. [Accessed 4 May 2016].
- [39] B. T. Phong, "Illumination of Computer-Generated Images," *Communications of the ACM*, vol. 18, no. 6, pp. 311-317, 1975.
- [40] J. Blinn, "Models of light reflection for computer synthesized pictures," in *SIGGRAPH '77*, San Jose, 1977, pp. 192-198.
- [41] M. Pharr and G. Humphreys, "Reflection models," in *Physically based rendering: from theory to implementation*, 2nd ed., Burlington, Morgan Kaufmann, 2010, ch. 8, pp. 423-496.
- [42] P. Dirac, "Representations," in *The Principles of Quantum Mechanics*, 4th ed., Oxford, Clarendon Press, 1958, ch. 3, p. 58.
- [43] C. Moidel, "Bi-Directional reflectance distribution functions," Worcester Polytechnic Institute, 2 September 2006. [Online]. Available: http://web.cs.wpi.edu/~emmanuel/courses/cs563/write_ups/chuckm/chuckm_BRDFs_overview.html. [Accessed 23 April 2016].
- [44] D. Biliouris, W. W. Verstraeten, P. Dutré, J. A. Van Aardt, B. Muys and P. Coppin, "A Compact Laboratory Spectro-Goniometer (CLabSpeG) to Assess the BRDF of Materials. Presentation, Calibration and Implementation on *Fagus sylvatica* L. Leaves," *Sensors*, vol. 7, no. 9, pp. 1846-1870, 7 September 2007.

- [45] P. A. Laplante, “Designing Software,” in *What Every Engineer Should Know about Software Engineering (What Every Engineer Should Know)*, Boca Raton, Florida: CRC Press, 2007, ch. 4, pp. 85-86.
- [46] L. Vandevenne, “LodePNG,” 1 January 2005. [Online]. Available: <http://lodev.org/lodepng/>. [Accessed 27 December 2015].
- [47] Apple Inc., “Cocoa Application Layer,” Apple Inc., 27 May 2004. [Online]. Available: https://developer.apple.com/library/mac/documentation/MacOSX/Conceptual/OSX_Technology_Overview/CocoaApplicationLayer/CocoaApplicationLayer.html. [Accessed 18 June 2016].
- [48] Apple Inc., “Cocoa Touch Layer,” Apple Inc., 15 October 2008. [Online]. Available: <https://developer.apple.com/library/ios/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/iPhoneOSTechnologies/iPhoneOSTechnologies.html>. [Accessed 2016 June 18].
- [49] Apple Inc., “Grand Central Dispatch (GCD) Reference,” Apple Inc., 19 August 2009. [Online]. Available: https://developer.apple.com/library/ios/documentation/Performance/Reference/GCD_libdispatch_Ref/. [Accessed 18 June 2016].
- [50] Microsoft Corporation, “Asynchronous programming in C++,” Microsoft Corporation, 21 April 2016. [Online]. Available: <https://msdn.microsoft.com/windows/uwp/threading-async/asynchronous-programming-in-cpp-universal-windows-platform-apps>. [Accessed 18 June 2016].
- [51] Travis CI, GmbH, “Travis CI: builds apps with confidence,” Travis CI, GmbH, 1 September 2012. [Online]. Available: <https://travis-ci.com>. [Accessed 18 June 2016].
- [52] Appveyor Systems Inc., “AppVeyor: Continuous Delivery service for Windows,” Appveyor Systems Inc., 22 June 2012. [Online]. Available: <https://www.appveyor.com>. [Accessed 18 June 2016].
- [53] Kitware Inc., “CMake,” Kitware Inc., 1 January 2000. [Online]. Available: <https://cmake.org>. [Accessed 15 June 2016].
- [54] A. Danial, “Cloc: count lines of code,” Danial, AI, 1 September 2015. [Online]. Available: <https://github.com/AIDanial/cloc>. [Accessed 24 June 2016].

- [55] J. Hoxley, "D3DBook:(Lighting) Oren-Nayar," 28 September 2011. [Online]. Available: [http://content.gpwiki.org/D3DBook:\(Lighting\)_Oren-Nayar](http://content.gpwiki.org/D3DBook:(Lighting)_Oren-Nayar). [Accessed 07 November 2015].
- [56] M. Pharr and G. Humphreys, "Monte carlo integration I: basic concepts," in *Physically based rendering: from theory to implementation*, 2nd ed., Burlington, Morgan Kaufmann, 2010, ch. 13, pp. 638-733.
- [57] E. Catmull, "A subdivision algorithm for computer display of curved surfaces.," Phd Thesis, Utah Univ., 1974.
- [58] N. Green, "Environment mapping and other applications of world projections," *IEEE Computer Graphics and Applications*, vol. 6, no. 11, pp. 21-29, 1 November 1986.
- [59] K. Perlin, "An image synthesizer," in *SIGGRAPH '85*, San Francisco, 1985, pp. 287-296.
- [60] K. Perlin, "Improving noise," in *SIGGRAPH '02*, San Antonio, 2002, pp. 681-682.
- [61] J. Blinn, "Simulation of wrinkled surfaces," in *SIGGRAPH '78*, Atlanta, 1978, pp. 286-292.
- [62] Stanford computer graphics laboratory, "The Stanford 3D Scanning Repository," Stanford univ., 8 May 1999. [Online]. Available: <http://graphics.stanford.edu/data/3Dscanrep/>. [Accessed 5 June 2016].
- [63] Visual Computing Laboratory CNR, "MeshLab," Visual Computing Laboratory CNR, 24 December 2005. [Online]. Available: <http://meshlab.sourceforge.net>. [Accessed 15 June 2016].
- [64] Scratchapixel, "Ray-Box Intersection," Scratchapixel, 13 September 2015. [Online]. Available: <http://www.scratchapixel.com/lessons/3d-basic-rendering/minimal-ray-tracer-rendering-simple-shapes/ray-box-intersection>. [Accessed 15 June 2016].
- [65] CIE Commission Internationale de l'Eclairage, "International Lighting Vocabulary," Tech. Rep. 17.4-1987, CIE Commission Internationale de l'Eclairage, Vienna, 1987.
- [66] K. Beason, "smallpt: Global Illumination in 99 lines of C++," 12 November 2008. [Online]. Available: <http://www.kevinbeason.com/smallpt/>. [Accessed 10 December 2015].
- [67] G. J. Ward, "Measuring and modeling anisotropic reflection," in *SIGGRAPH '92*, Chicago, 1992, pp. 265-272.
- [68] M. Ashikhmin and P. Shirley, "An anisotropic phong BRDF model," *Journal of Graphics Tools*, vol. 5, no. 2, pp. 25-32, December 2000.

