

Advanced light modelling with GroIMP

Michael Henke^{1,2}, Gerhard Buck-Sorlin¹

¹UMR1345 Institut de Recherche en Horticulture et Semences (IRHS) Équipe
Arboriculture Fruitière AGROCAMPUS OUEST Centre d'Angers - INRA - Université
d'Angers, France

²Department Ecoinformatics, Biometrics and Forest Growth, Büsgenweg 4, University of
Göttingen, Germany

Tutorial and Workshop

"Functional-Structural Plan Modelling with GroIMP and XL"
combined with the 7th GroIMP user and developer meeting

Angers, 2015-05-06





Table of Contents

Introduction

Light modelling

Model test

Application

Conclusions

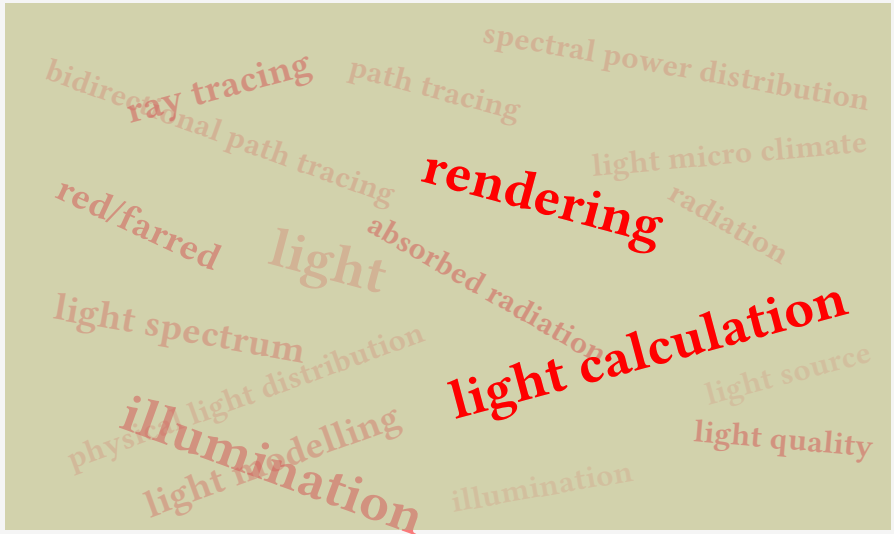


Disambiguation





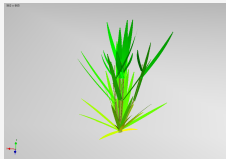
Disambiguation





Rendering vs. Light calculation

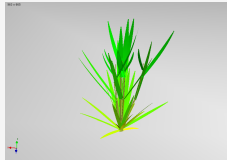
3d model





Rendering vs. Light calculation

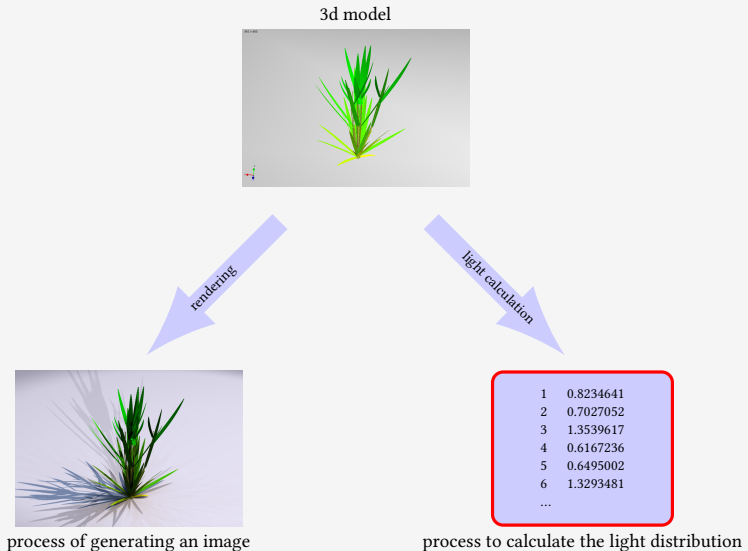
3d model



process of generating an image



Rendering vs. Light calculation





Light calculation - The "common/old" way

- ▶ LightModel
 - ▶ CPU-based → relatively slow
 - ▶ only three channels (e.g. for R, G, B)

```
1 LightModel lm = new LightModel(100000, 10); // rays, recursion depth
2 lm.compute();
3
4 [
5     x:Sphere ::> {
6         println("CPU LM = "+lm.getAbsorbedPower(x).integrate());
7     }
8 ]
```



Advanced light modelling

- ▶ And now - what is so new/advanced?
 - ▶ GPU-based → really fast
 - ▶ full light spectrum
 - ▶ supports multiple devices



Advanced light modelling

- ▶ And now - what is so new/advanced?
 - ▶ GPU-based → really fast
 - ▶ full light spectrum
 - ▶ supports multiple devices
- ▶ What is it good for?
 - ▶ precise calculation of (micro) light climate: light quality + light distribution
 - ▶ light quality → photosynthesis calculation
 - ▶ reduce computation time
 - ▶ increase accuracy



Advanced light modelling

- ▶ And now - what is so new/advanced?
 - ▶ GPU-based → really fast
 - ▶ full light spectrum
 - ▶ supports multiple devices
- ▶ What is it good for?
 - ▶ precise calculation of (micro) light climate: light quality + light distribution
 - ▶ light quality → photosynthesis calculation
 - ▶ reduce computation time
 - ▶ increase accuracy
- ▶ How is it called?
 - ▶ FluxLightMode
 - ▶ Dietger van Antwerpen (2011)



Table of Contents

Introduction

Light modelling

General

Light model

Light sources

Illumination model

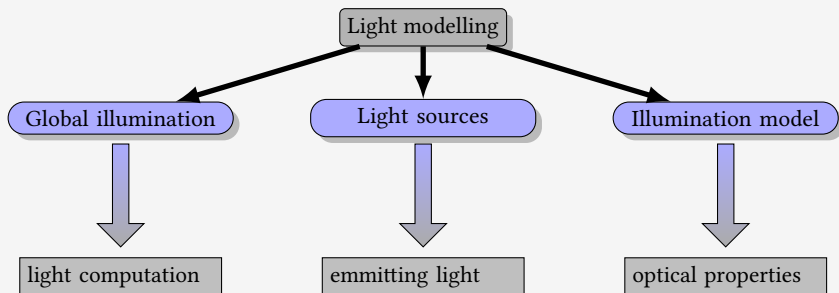
Model test

Application

Conclusions

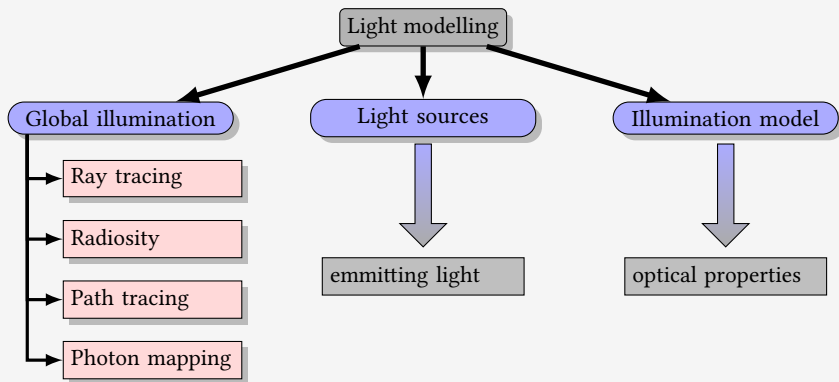


Light modelling



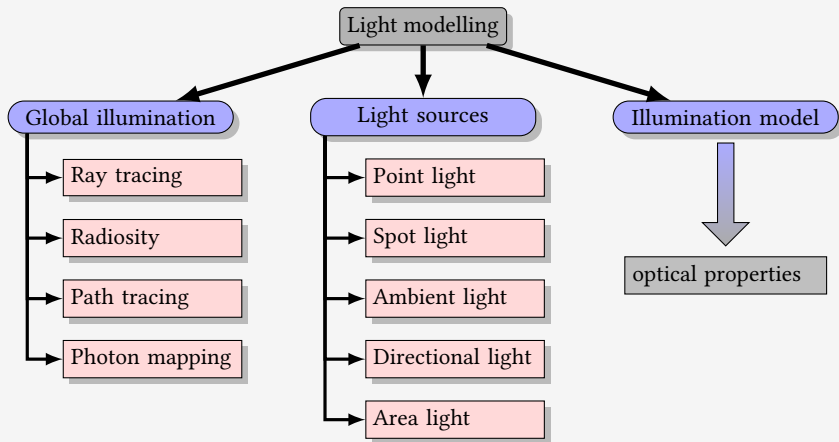


Light modelling



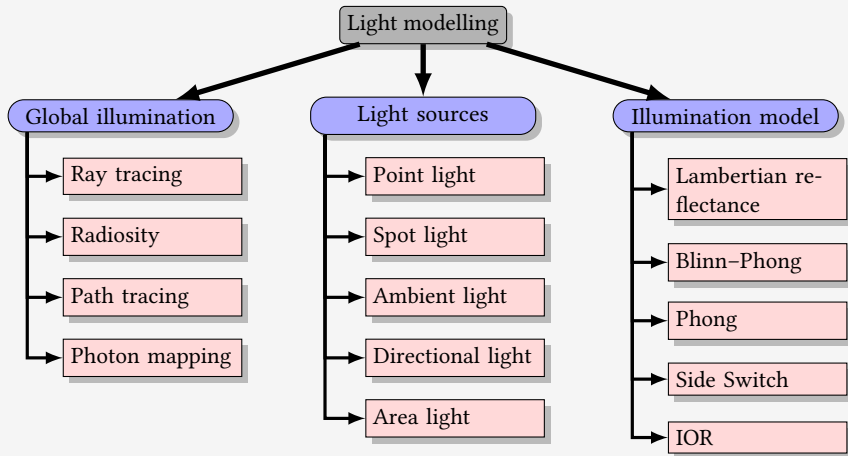


Light modelling





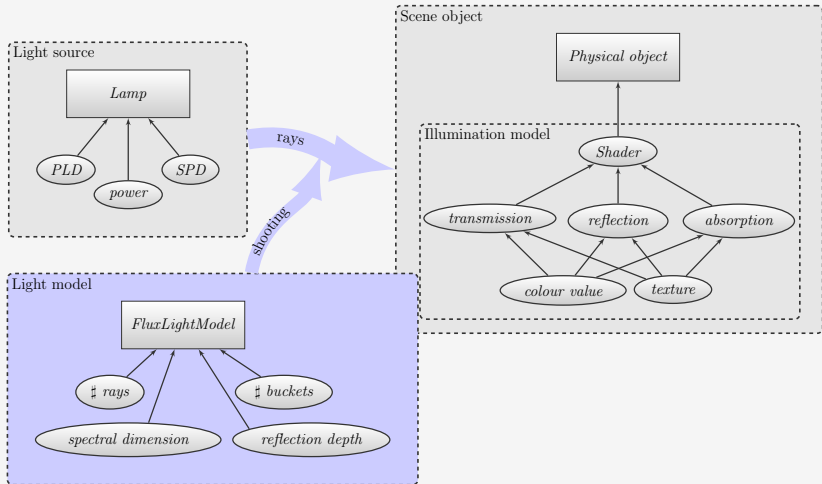
Light modelling



(Henke and Buck-Sorlin, unpubl.)



Progress in light modelling

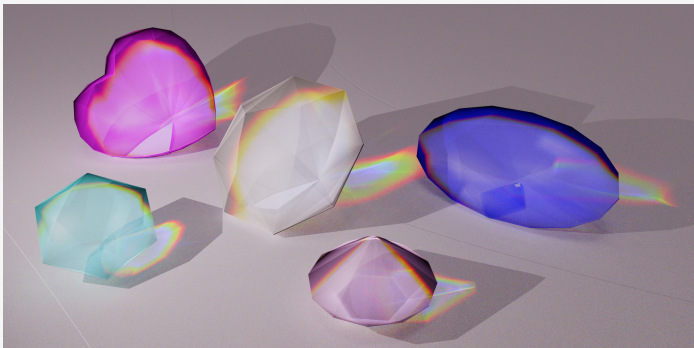


(Henke and Buck-Sorlin, unpubl.)



Integrated Light Model

- ▶ High performance light model (GPU Flux)
path tracer, bidirectional path tracer and spectral render
- ▶ Supports full spectral rendering, dispersion



(Henke and Buck-Sorlin, unpubl.)



Integrated Light Model

- ▶ supports three different modes of measuring spectral power
 - ▶ regular RGB
 - ▶ full discretized spectral measurements
 - ▶ weighted integration

$mode \in \{RGB, FULL_SPECTRUM, INTEGRATED_SPECTRUM\}$

```
1 import de.grogra.gpuflex.tracer.FluxLightModelTracer.MeasureMode;  
2  
3 ...  
4  
5 lm.setMeasureMode(MeasureMode.FULL_SPECTRUM);
```



Flux Light Model: Setup and Usage - I

import
needed classes

initialization

parame-
terization

```
1 import de.grogra.imp3d.spectral.IrregularSpectralCurve;
2 import de.grogra.ray.physics.Spectrum;
3 import de.grogra.gpuflex.tracer.FluxLightModelTracer.
  MeasureMode;
4 import de.grogra.gpuflex.scene.experiment.Measurement;
5
6
7
8 const int RAYS = 10000000;
9 const FluxLightModel lm = new FluxLightModel(RAYS, 10);
10
11
12
13 protected void init () {
14     lm.setMeasureMode(MeasureMode.FULL_SPECTRUM);
15     lm.setSpectralBuckets(81);
16     lm.setSpectralDomain(380, 780);
17 }
```




Flux Light Model: Setup and Usage - II

computation

evaluation sensor

evaluation box

```
1 public void run () [  
2     {  
3         lm.compute();  
4     }  
5  
6  
7     x:SensorNode ::> {  
8         Measurement spectrum = lm.  
9             getSensedIrradianceMeasurement(x);  
10        float absorbedPower = spectrum.integrate();  
11        ...  
12    }  
13  
14    x:Box ::> {  
15        Measurement spectrum = lm.  
16            getAbsorbedPowerMeasurement(x);  
17        float absorbedPower = spectrum.integrate();  
18        ...  
19    }  
20 ]
```



Flux Light Model: Setup and Usage - III

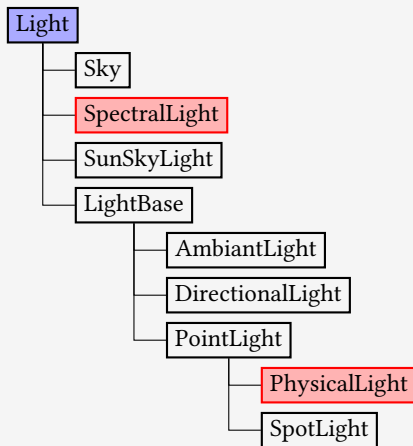
Absorbed power per bucket

```
1  ...
2
3  Measurement spectrum = lm.getAbsorbedPowerMeasurement(x);
4
5  //absorbed power for the first bucket: 380-385 nm
6  float ap380_385 = spectrum.data[0];
7
8  //accumulate absorbed power for the first four 50 nm buckets
9  float b0 = 0, b1 = 0, b2 = 0, b3 = 0;
10 for(int i:(0:9)) {
11     b0 += spectrum.data[i];
12     b1 += spectrum.data[i + 10];
13     b2 += spectrum.data[i + 20];
14     b3 += spectrum.data[i + 30];
15 }
```



Available Light Nodes

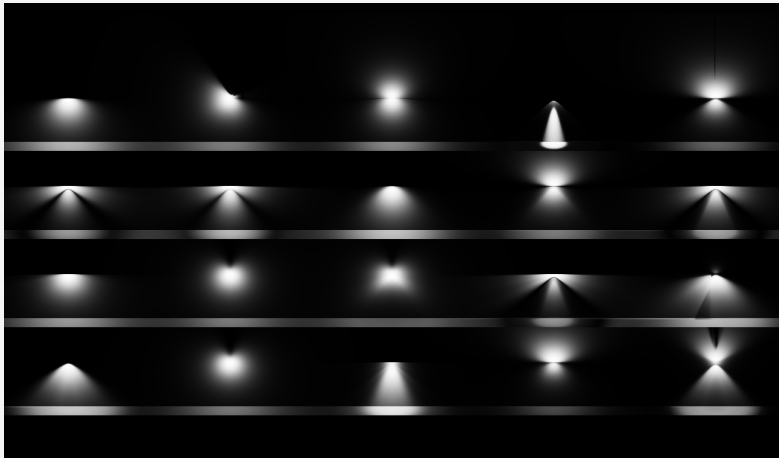
- ▶ current light nodes in GroIMP
- ▶ implementing *Light* interface
- ▶ *SpectralLight* required for spectral light calculation





New Light Source - *SpectralLight*

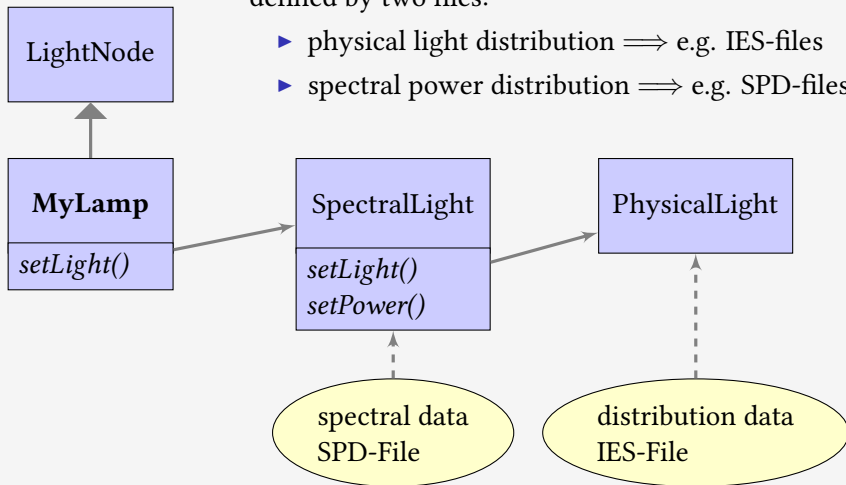
- ▶ generic lamp ($\implies$ *LampDemo.gsz*)
- ▶ individually defined light source



Spectral Light Source

defined by two files:

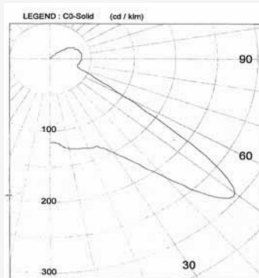
- ▶ physical light distribution $\Rightarrow$ e.g. IES-files
- ▶ spectral power distribution $\Rightarrow$ e.g. SPD-files





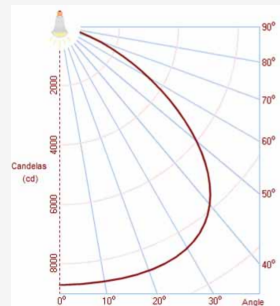
Physical light distribution

- ▶ polar distribution diagram, also called a polar curve
- ▶ luminous intensity values with increasing angles from the imaginary axis of the lamp
- ▶ file formats: IES, LUM, LDT, ...



INTENSITY DATA (cd / klm)

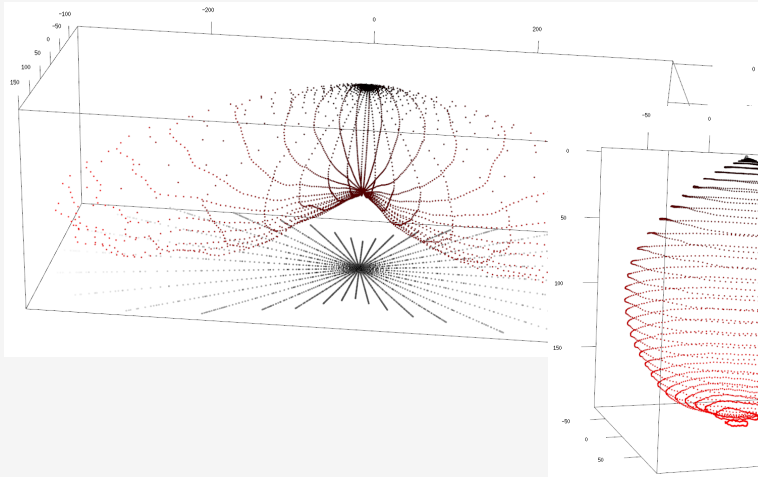
Gamma	C0	LMS.	Gamma	C0	LMS.
0	117		90	38	
5	119	12	95	39	42
10	127		100	39	
15	130	37	105	38	40
20	132		110	36	
25	135	64	115	34	34
30	150		120	31	
35	175	112	125	27	24
40	214		130	21	
45	268	204	135	14	11
50	290		140	8	
55	205	180	145	5	3
60	103		150	4	
65	59	62	155	3	2
70	38		160	3	
75	34	37	165	3	1
80	35		170	3	
85	37	40	175	2	0
90	38		180	2	



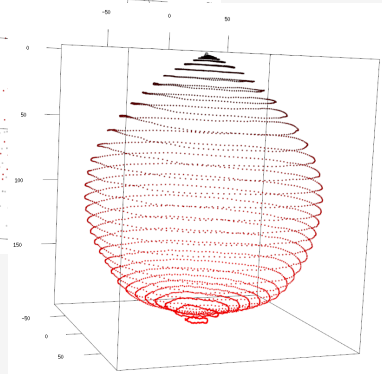


Physical light distribution - 3D-Visualization

SON-T



LED





Physical light distribution - File Formats

1. IES files

- ▶ IES Illuminating Engineering Society
- ▶ ASCII file: see IES-Specification

2. LUM files

- ▶ Luminance file
- ▶ simple ASCII file:
- ▶ first two lines number of angles,
- ▶ followed by rows of values for each angle
- ▶ can be directly imported by GroIMP

3. LDT files

- ▶ EULUMDAT file
- ▶ simple ASCII file

↪ How to convert IES to LUM?



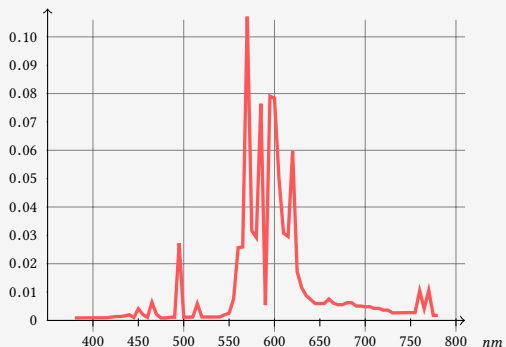
Spectral power distribution

1. SPD: Spectral Power Distribution file

- ▶ simple ASCII-file: two columns (wavelength, amplitude)
- ▶ wavelength divided into 5 *nm* buckets, amplitudes are normalized
- ▶ can be directly imported by GroIMP

SON-T

```
380 0.000967721
385 0.000980455
390 0.000993188
395 0.001005921
400 0.001018654
405 0.001031387
410 0.001044120
415 0.001056854
420 0.001283504
425 0.001515248
430 0.001533074
435 0.001772458
...
```





Parameterisable light node - Code: Using arrays

```
1  const double[][] DISTRIBUTION = {  
2    {131.25, 131.67, 132.37,...},  
3    {131.36, 131.81, 132.11,...},  
4    ...  
5  };  
6  
7  static const float[] WAVELENGTHS = {380,385,...};  
8  static const float[] AMPLITUDES = {0.000967721, 0.000980455, ...};  
9  
10 module MyLamp extends LightNode() {  
11   {  
12     setLight(new SpectralLight(  
13       new IrregularSpectralCurve(WAVELENGTHS, AMPLITUDES)).(  
14         setPower(10), //[W]  
15         setLight(new PhysicalLight(DISTRIBUTION))  
16       ) //end SpectralLight  
17     ); //end setLight  
18   }  
19 }
```



Parameterisable light node - Code: Using file references

Definition of references

```
1 const LightDistributionRef DISTRIBUTION = light("distri01");  
2 const SpectrumRef SPECTRUM = spectrum("equal");
```

set them via the constructor

```
1 module MyLamp extends LightNode {  
2   {  
3     setLight(new SpectralLight(new PhysicalLight(DISTRIBUTION), SPECTRUM, 5));  
4   }  
5 }
```

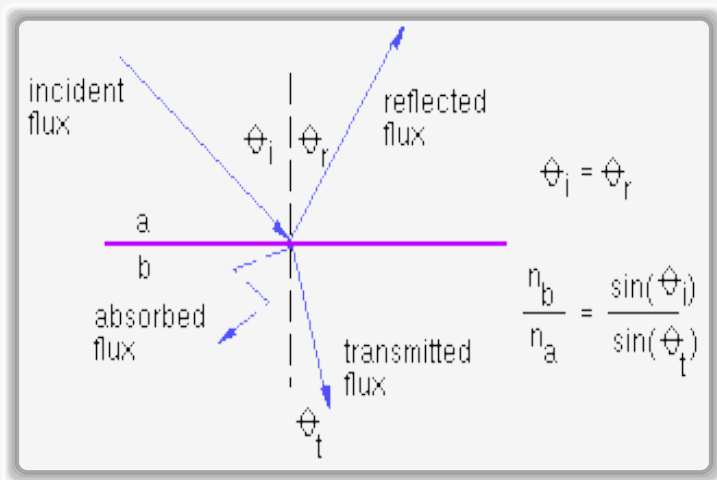
or use the set-methods

```
1 module MyLamp extends LightNode {  
2   {  
3     setLight(new SpectralLight().(  
4       setPower(5), //[W]  
5       setLight(new PhysicalLight(DISTRIBUTION)),  
6       setSpectrum(SPECTRUM)  
7     ));  
8   }  
9 }
```

↔ How to define
references?

Illumination model

local illumination

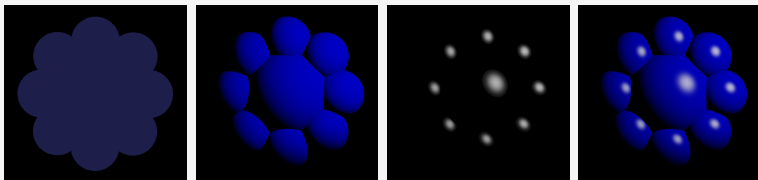




Shader

definition of optical properties

- Phong reflection model, B.T. Phong, 1973



ambient + diffuse + specular = phong

Henke and Buck-Sorlin unpubl.

- Lambertian reflectance → only diffuse reflection



Phong: Textures, Colours and Transparency

set an image as texture

```
1 Phong myShader = new Phong();
2 ImageMap image = new ImageMap();
3 image.setImageAdapter(
4     new FixedImageAdapter(image("leaf").toImageAdapter().getBufferedImage())
5 );
6 myShader.setDiffuse(image);
```

set specific properties

```
1 Phong myShader = new Phong();
2 //myShader.setDiffuse(new Graytone (0.5));
3 myShader.setDiffuse(new RGBColor (0,1,0));
4 //myShader.setSpecular(new Graytone (0.5));
5 //myShader.setShininess(new Graytone (0.5));
6 myShader.setTransparency(new Graytone (0.0));
7
8 //myShader.setTransparencyShininess(new Graytone (0.5));
9 //myShader.setDiffuseTransparency(new Graytone (0.5));
10 //myShader.setAmbient(new Graytone (0.5));
11 //myShader.setEmissive(new Graytone (0.5));
```



Special Phong Shader - I

mirror

```
Phong s = new Phong();  
s.setDiffuse(new Graytone(0));  
s.setSpecular(new Graytone(1));  
s.setShininess(new Graytone(1));
```

glass

```
Phong s = new Phong();  
s.setDiffuse(new Graytone(1));  
s.setSpecular(new Graytone(0.5));  
s.setShininess(new Graytone(0.5));  
s.setDiffuseTransparency(new  
    Graytone(0.95));
```

semi-transparent

```
Phong s = new Phong();  
s.setDiffuse(new Graytone(1));  
setDiffuseTransparency(new  
    Graytone(0.5));
```

total absorber

```
Phong s = new Phong();  
s.setDiffuse(new Graytone(0));
```

total red

```
Phong s = new Phong();  
s.setDiffuse(new RGBColor(1,0,0));
```



Special Phong Shader - I

mirror

```
Phong s = new Phong();  
s.setDiffuse(new Graytone(0));  
s.setSpecular(new Graytone(1));  
s.setShininess(new Graytone(1));
```

glass

```
Phong s = new Phong();  
s.setDiffuse(new Graytone(1));  
s.setSpecular(new Graytone(0.5));  
s.setShininess(new Graytone(0.5));  
s.setDiffuseTransparency(new  
    Graytone(0.95));
```

semi-transparent

```
Phong s = new Phong();  
s.setDiffuse(new Graytone(1));  
setDiffuseTransparency(new  
    Graytone(0.5));
```

total absorber

```
Phong s = new Phong();  
s.setDiffuse(new Graytone(0));
```

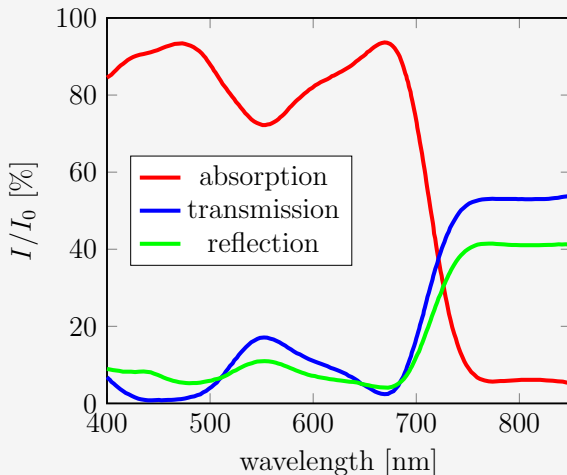
total red

```
Phong s = new Phong();  
s.setDiffuse(new RGBColor(1,0,0));
```

easy case: equal for the whole range!

Absorption spectrum of leaf pigments

- ▶ typical soybean leaf



Kasperbauer, 1987



Special Phong Shader - II

by simple arrays

```
1 public const float[] WAVELENGTHS = {350, 360, 370, ..., 850};
2 public const float[] AMPLITUDES = {0.1, 0, 0, 0.1, 0.2, ..., 0.4, 0.3, 0.2};
3
4 const ChannelSPD mySPD = new ChannelSPD(
5     new IrregularSpectralCurve(WAVELENGTHS, AMPLITUDES));
6
7 const Phong MyShader = new Phong();
8 static {
9     MyShader.setDiffuse(mySPD);
10 }
```



Special Phong Shader - III

or defined by functions

```
1  const int MIN_WAVELENGTH = 350; // [nm]
2  const int MAX_WAVELENGTH = 850; // [nm]
3  const int BUCKETS = 50;
4  const int DELTA_W = (MAX_WAVELENGTH - MIN_WAVELENGTH) / BUCKETS;
5
6  public const float[] WAVELENGTHS = new float[BUCKETS];
7  public const float[] AMPLITUDES = new float[BUCKETS];
8  static {
9      for(int i=0; i<BUCKETS; i++) {
10         WAVELENGTHS[i] = MIN_WAVELENGTH + i * DELTA_W;
11         AMPLITUDES[i] = i/(float)(BUCKETS+1);
12     }
13 }
14
15 const ChannelSPD linSPD = new ChannelSPD(
16     new IrregularSpectralCurve(WAVELENGTHS, AMPLITUDES));
17
18 const Phong MyShader = new Phong();
19 static {
20     MyShader.setDiffuse(linSPD); // linear increasing
21 }
```



Side switch shader

- ▶ leaves have distinctive upper (adaxial) and lower (abaxial) surfaces

```
1 const Phong AdaxialShader = new Phong();
2 static {
3     AdaxialShader.setSpecular(new Graytone(0.01)); // reflected specular
4     AdaxialShader.setDiffuse(new Graytone(0.05)); // reflected diffuse
5     AdaxialShader.setTransparency(new Graytone(0.04)); // transmitted
6 }
7
8 const Phong AbaxialShader = new Phong();
9 static {
10     AbaxialShader.setSpecular(new Graytone(0.02)); // reflected specular
11     AbaxialShader.setDiffuse(new Graytone(0.07)); // reflected diffuse
12     AbaxialShader.setTransparency(new Graytone(0.01)); // transmitted
13 }
14
15 const static SideSwitchShader leafS = new SideSwitchShader(AdaxialShader,
    AbaxialShader);
```



Special shader - Images

- inhomogeneous area





Table of Contents

Introduction

Light modelling

Model test

Light sources

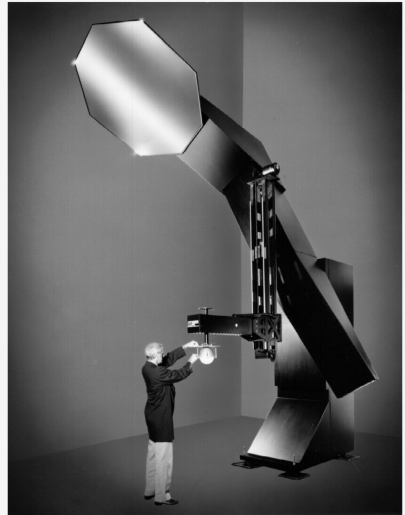
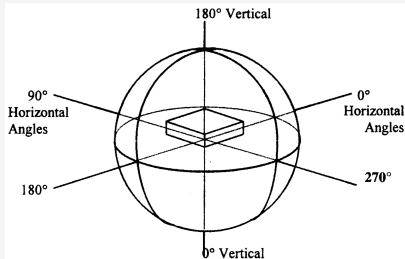
Illumination model

Application

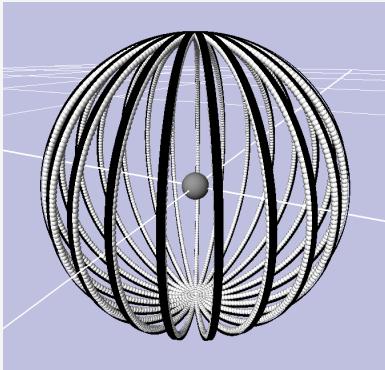
Conclusions

Lamp Test Environment

- ▶ simulates a **Goniophotometer**
- ▶ compares input values with simulated values
 - ▶ physical light distribution
 - ▶ spectral power distribution
- ▶ conducts measurements in a sphere



Lamp Test Environment



- ▶ model:
LampTestEnvironment.gsz
- ▶ input: ies + spd file
- ▶ set-up
 - ▶ light source in the centre of the scene
 - ▶ spherical net of sensors
 - ▶ spectral resolution:
freely definable, e.g.
380:5:780 *nm*
- ▶ output: charts and datasets of distributions
- ▶ no additional conversions are applied



Lamp Tests: SONT

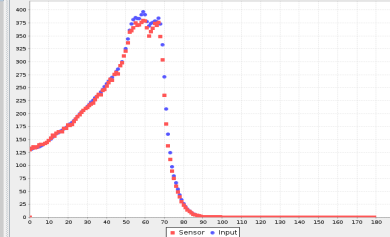
View
View Camera Fit Render View

Rays : 400M
 vertical angles (phi) : 36
 horizontal angles (theta) : 180
 number of sensors : 6480
 input vs. measured:

	phi=0°	theta=20.0°	total [cd]
I	18748.58	5450.00	484861.34
M	18288.01	5465.85	484529.94
M/I	0.975	1.003	0.999
MSE	42.274	3.598	3.453

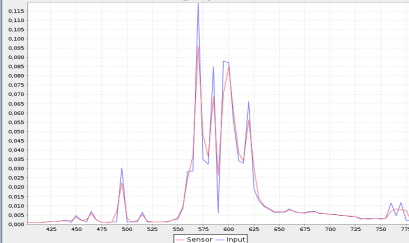
Absorbed power (phi)

Absorbed power (phi)



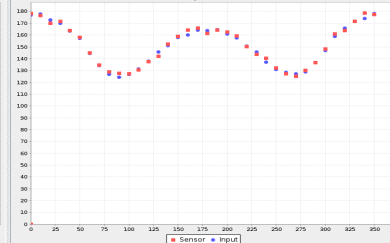
Normed average spectral distribution

Normed average spectral distribution



Absorbed power (theta)

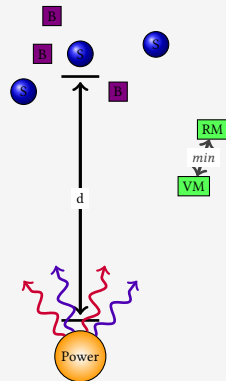
Absorbed power (theta)



Output Power Value

Output power: influential factor → determines a proper value

- ▶ power of the light node
- ▶ distance between light source and object
- ▶ used spectral power distribution
- ▶ optical properties of the object
- ▶ position and distribution of sensors (position of real measurement equals measurement point in the simulation)





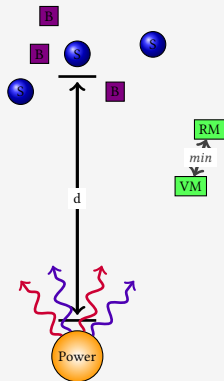
Output Power Value

Output power: influential factor → determines a proper value

- ▶ power of the light node
- ▶ distance between light source and object
- ▶ used spectral power distribution
- ▶ optical properties of the object
- ▶ position and distribution of sensors (position of real measurement equals measurement point in the simulation)

Consequences:

- ▶ need for precise data of overall power output
- ▶ ...and for information about setup of measurement

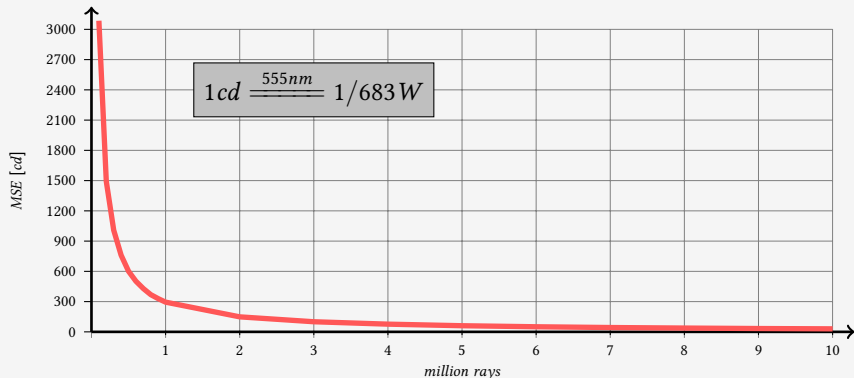




LED - Ray Statistic

Model accuracy as a function of number of rays used

- ▶ aim: realistic physical distribution with minimal computational investment
- ▶ realistic → minimal mean square error (MSE)

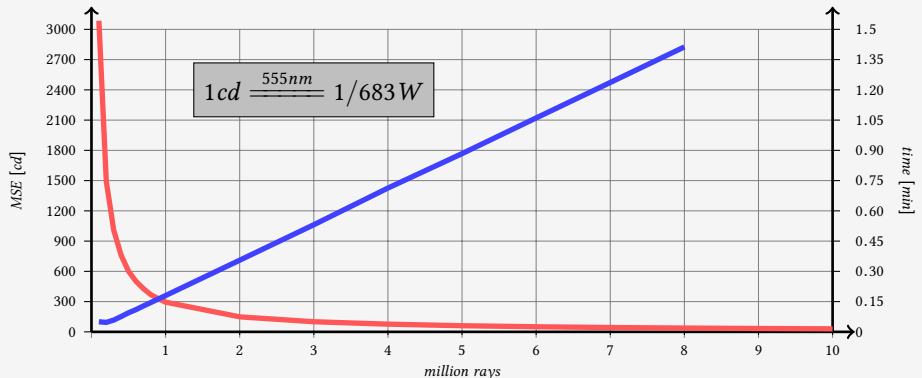




LED - Ray Statistic

Model accuracy as a function of number of rays used

- ▶ aim: realistic physical distribution with minimal computational investment
- ▶ realistic → minimal mean square error (MSE)

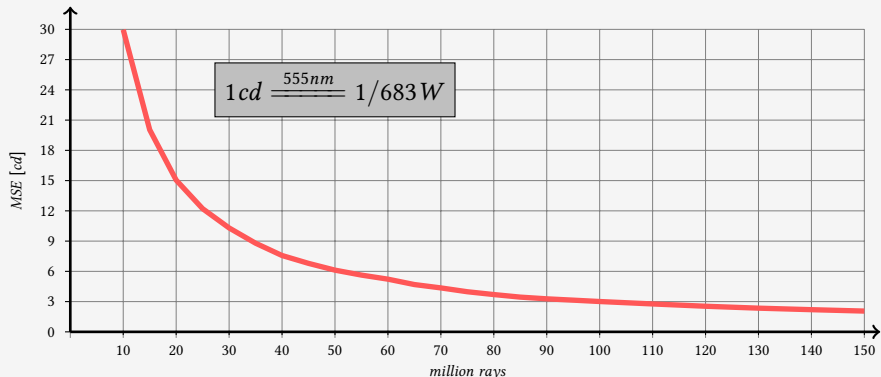




LED - Ray Statistic

Model accuracy as a function of number of rays used

- ▶ aim: realistic physical distribution with minimal computational investment
- ▶ realistic → minimal mean square error (MSE)

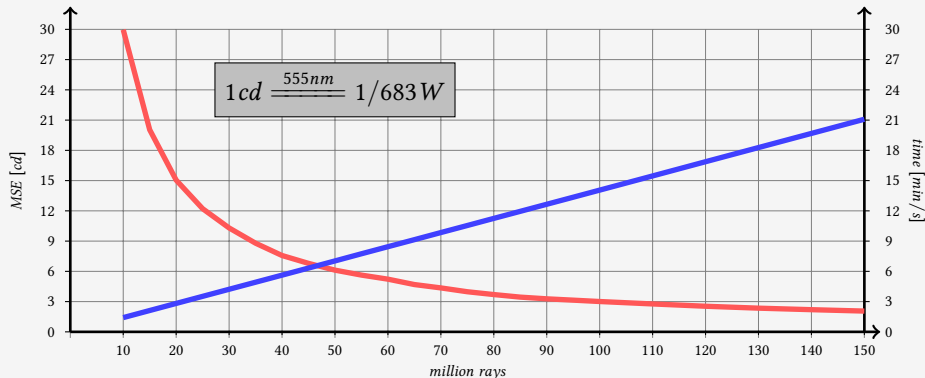




LED - Ray Statistic

Model accuracy as a function of number of rays used

- ▶ aim: realistic physical distribution with minimal computational investment
- ▶ realistic → minimal mean square error (MSE)

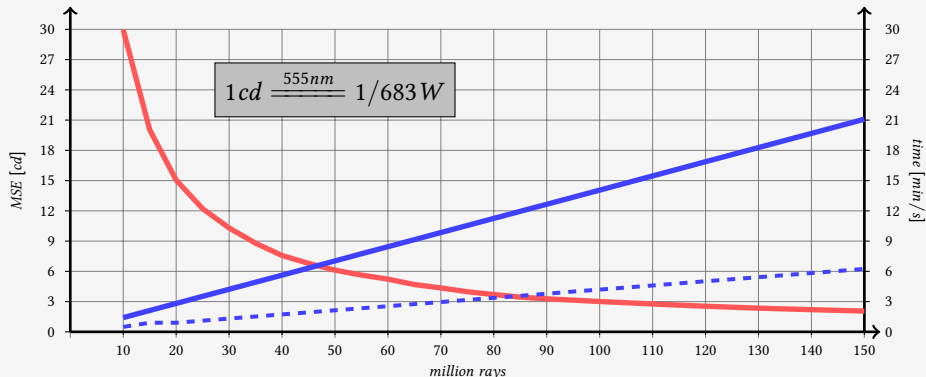




LED - Ray Statistic

Model accuracy as a function of number of rays used

- ▶ aim: realistic physical distribution with minimal computational investment
- ▶ realistic → minimal mean square error (MSE)

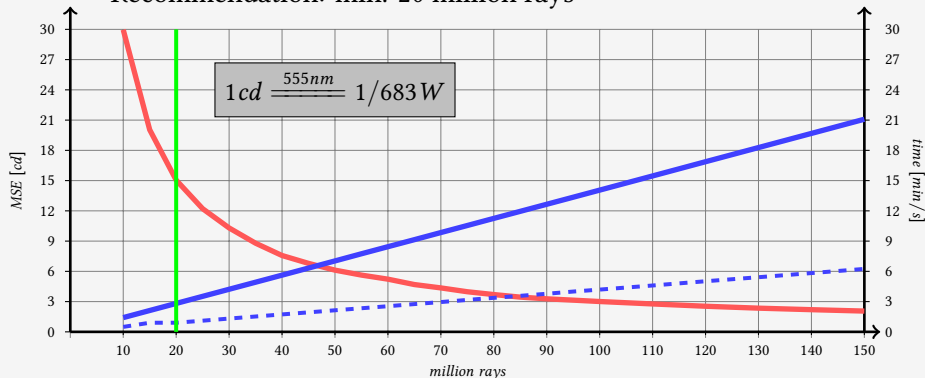




LED - Ray Statistic

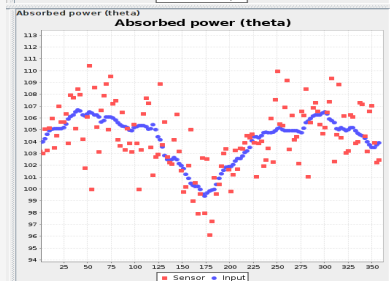
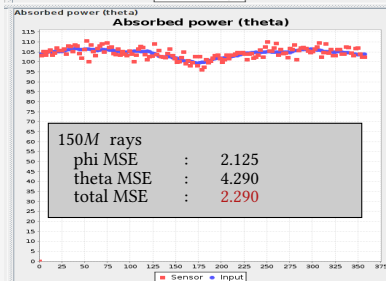
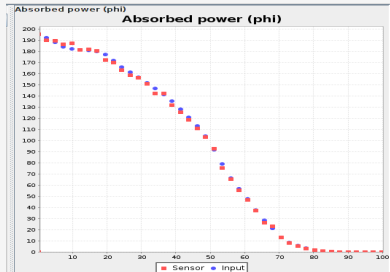
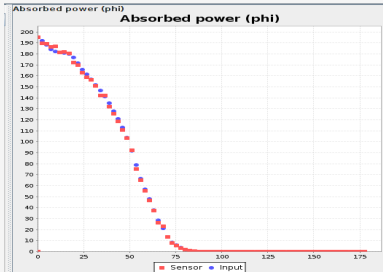
Model accuracy as a function of number of rays used

- ▶ aim: realistic physical distribution with minimal computational investment
- ▶ realistic → minimal mean square error (MSE)
- ▶ Recommendation: min. 20 million rays





SONT - Ray Statistic: 150M rays





Reproducibility - I

- ▶ we saw: increased number of rays → increased accuracy



Reproducibility - I

- ▶ we saw: increased number of rays $\rightarrow$ increased accuracy
- ▶ but this does not tell much about reproducibility



Reproducibility - I

- ▶ we saw: increased number of rays → increased accuracy
- ▶ but this does not tell much about reproducibility
- ▶ → repetitions needed!



Reproducibility - I

- ▶ we saw: increased number of rays → increased accuracy
- ▶ but this does not tell much about reproducibility
- ▶ → repetitions needed!

fixed scene



Reproducibility - I

- ▶ we saw: increased number of rays → increased accuracy
- ▶ but this does not tell much about reproducibility
- ▶ → repetitions needed!

fixed scene

```
for an number of repetitions do  
    run the LM with a new random seed  
    obtain measurements for all objects  
end
```



Reproducibility - I

- ▶ we saw: increased number of rays → increased accuracy
- ▶ but this does not tell much about reproducibility
- ▶ → repetitions needed!

fixed scene

for an increasing number of rays do

for an number of repetitions do

run the LM with a new random seed

obtain measurements for all objects

end

calculate the mean of all standard deviations

end



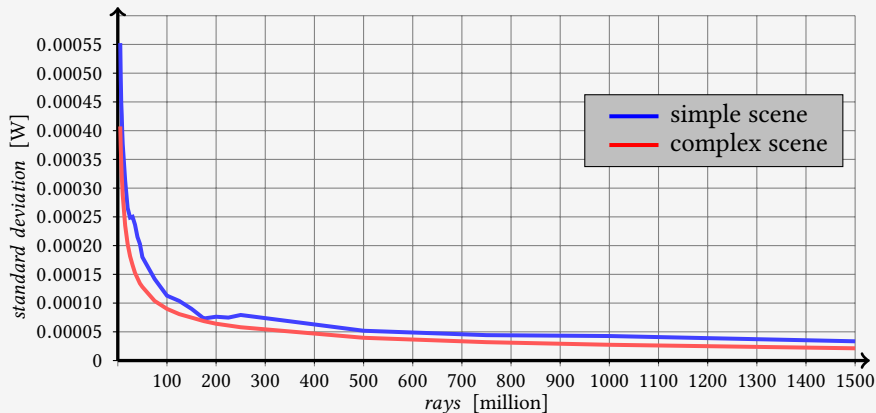
Reproducibility - II

```
1  const int[] RAYS = new int[] {10,20,50,100};
2  const int REPETITIONS = 25;
3
4  public void runTest () {
5      double[][] data = new double[MAX_OBJ][REPETITIONS];
6      double[] dataSD = new double[MAX_OBJ];
7      double[] dataset = new double[RAYS.length];
8
9      for(int i = 0; i < RAYS.length; i++) {
10         println("rays = "+(RAYS[i]*1000000));
11         for(int j = 0; j < REPETITIONS; j++) {
12             FluxLightModel lm = new FluxLightModel(RAYS[i]*1000000, 10);
13             lm.setSeed(SEEDS[j]); // !!!
14             lm.compute();
15             [
16                 x:B ::> data[x.i][j] = lm.getAbsorbedPower(x).integrate();
17             ]
18         }
19         for(int k = 0; k < MAX_OBJ; k++) dataSD[k] = sd(data[k]);
20         dataset[i] = mean(dataSD);
21     }
22 }
```



Reproducibility - III

- ▶ mean standard deviation of 25 repeated light model runs
- ▶ increasing number of rays (five million to 1.5 billion rays)

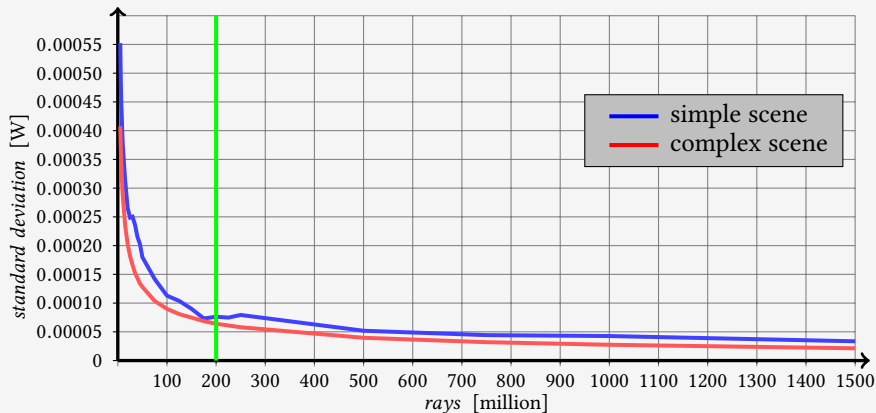


(Henke and Buck-Sorlin, unpubl.)



Reproducibility - III

- ▶ mean standard deviation of 25 repeated light model runs
- ▶ increasing number of rays (five million to 1.5 billion rays)



(Henke and Buck-Sorlin, unpubl.)



Ray Statistic - Influencing Factors

limitation factors → find a suitable number of rays

- ▶ hardware(!)
- ▶ complexity of the scene (number of objects, complexity of objects)
- ▶ number of light sources
- ▶ measured spectrum (number of buckets)
- ▶ recursion depth
- ▶ optical properties of objects (shader: e.g. transparency, emission)

⇒ proportional to increasing complexity of the scene
(...use as many rays as possible)



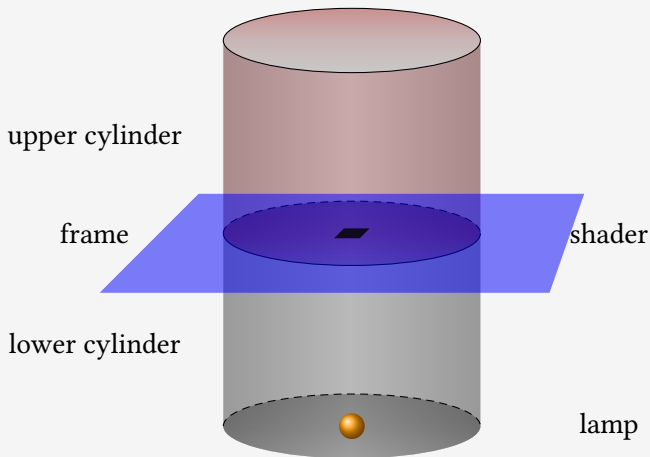
Sensor vs. Primitive Object

Differences in usage (when used as sensors)

Property	Sensor	Primitive Object
colour	white	black
transparency	–	has to be turned off
absorbed power returned	square meter	surface area
differences	100%	higher
overlapping (↔ charts)	not problematic	problematic
obtain spectrum	<code>getSensedIrradianceMeasurement</code>	<code>getAbsorbedPowerMeasurement</code>



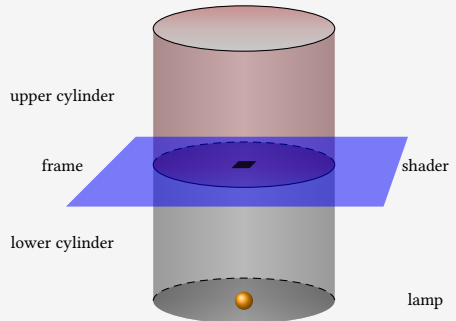
Shader Test Environment





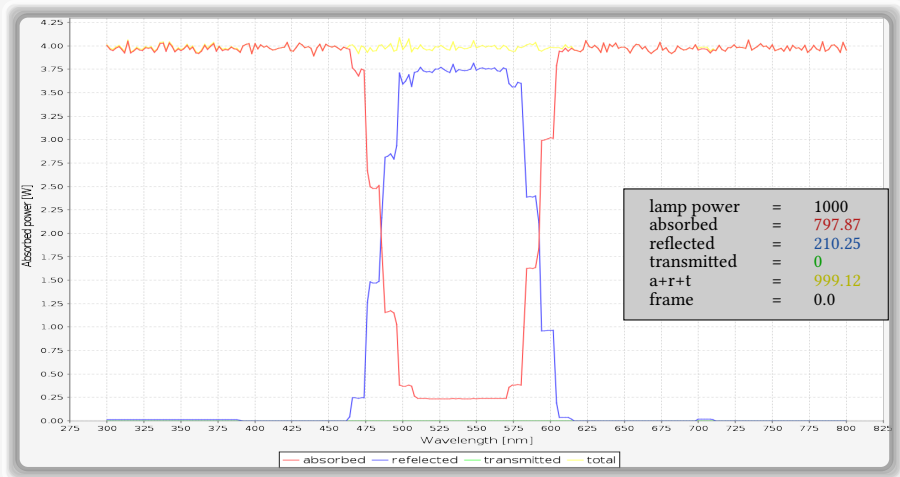
Shader Test Environment

- ▶ setup
 - ▶ two cylinders + frame
 - ▶ SpotLight, 1000W
 - ▶ spectrum: 380 : 5 : 780
 - ▶ #rays : 1M
- ▶ input
 - ▶ shader
 - ▶ optional: light source
- ▶ output
 - ▶ charts





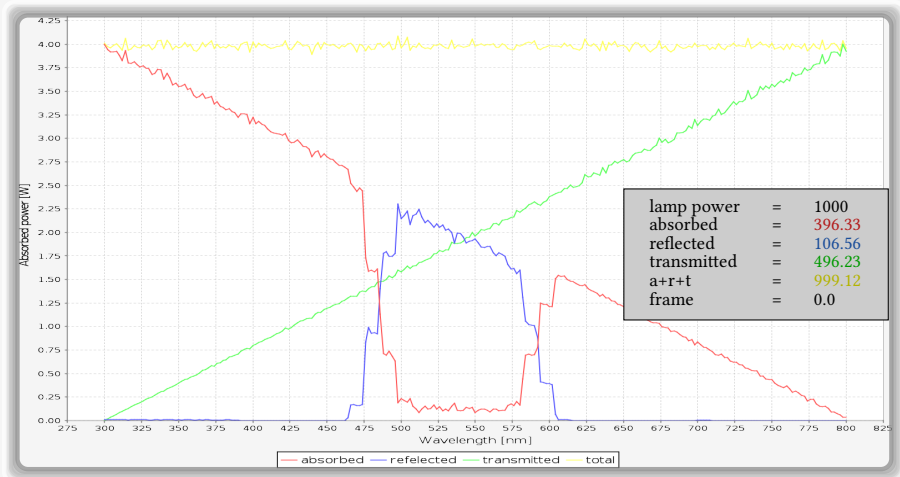
Shader Test - Example I



```
s.setDiffuse(new RGBColor(0,1,0));
```



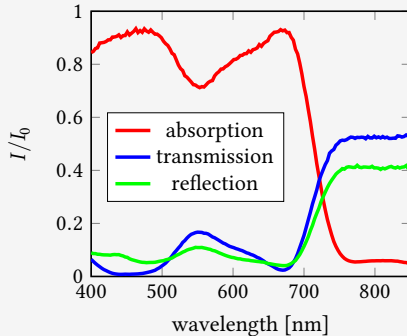

Shader Test - Example II



s.setDiffuse(new RGBColor(0,1,0)); lin. increased transparency



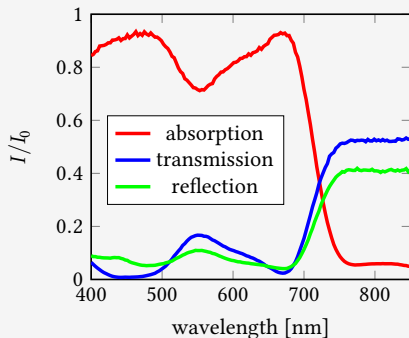
Shader Test - Example III



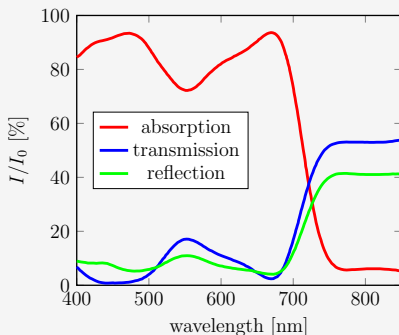
simulation



Shader Test - Example III



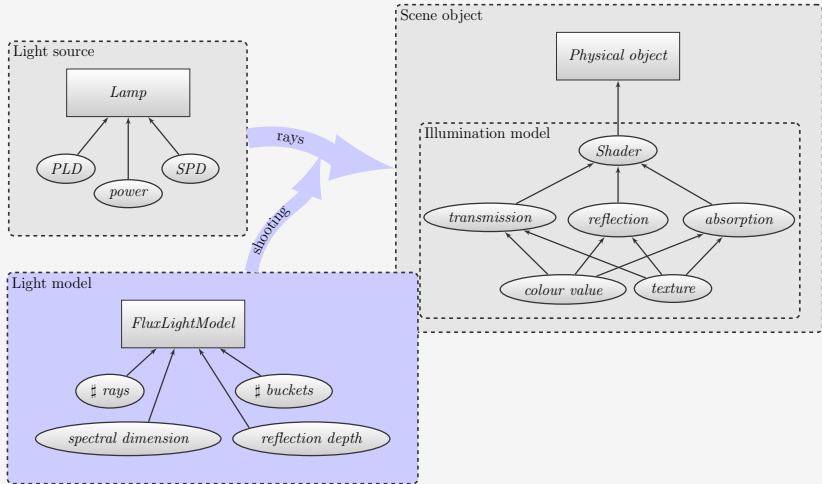
simulation



soybean leaf: Kasperbauer, 1987



Progress in light modelling



(Henke and Buck-Sorlin, unpubl.)



Table of Contents

Introduction

Light modelling

Model test

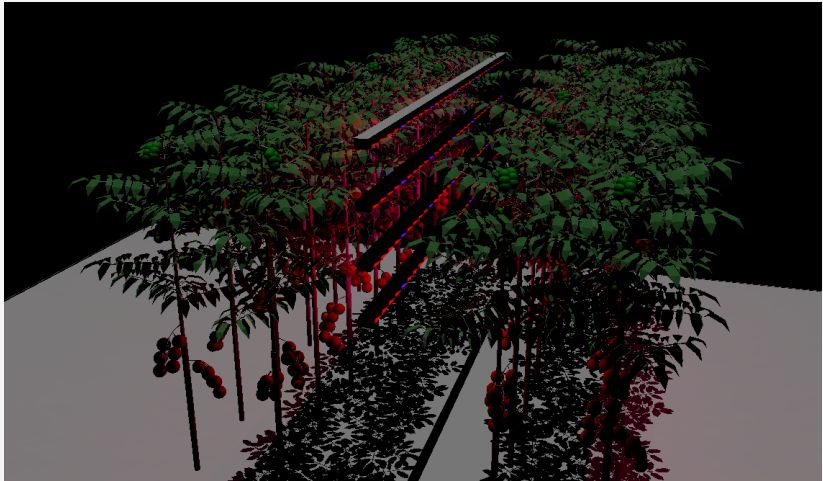
Application

Conclusions



Carbon LED project

LED-grids within tomato stands



(Henke, de Visser)



Table of Contents

Introduction

Light modelling

Model test

Application

Conclusions



Summary

- ▶ Models
 - ▶ full spectral light model and renderer
 - ▶ generic light source
 - ▶ reduced computation time → increased accuracy
- ▶ Documentation
 - ▶ This presentation
 - ▶ FluxLightModel API
 - ▶ Henke and Buck-Sorlin (under revision)
- ▶ available with GroIMP v.1.5



Thank you for your attention!

www.grogra.de







Table of Contents

Appendix

GroIMP

Calculations

Test results

Common Problems

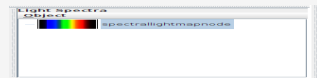
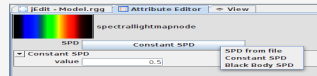
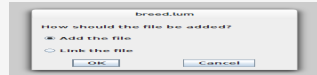
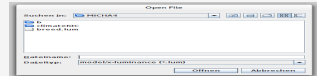
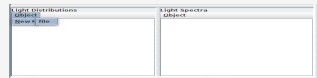
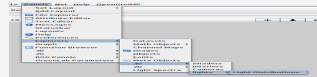
References



Define Light and Spectrum References

defining references

a) Open *Panels* → *Explorers*

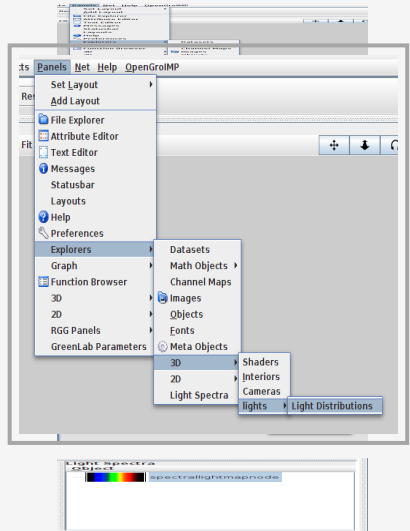




Define Light and Spectrum References

defining references

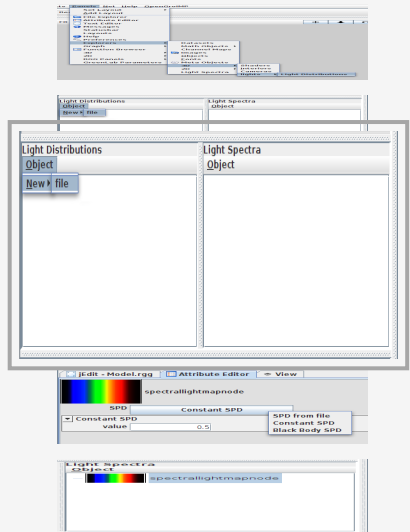
- a) Open *Panels* → *Explorers*
- b) Create new distribution object
Object → *New* → *file*



Define Light and Spectrum References

defining references

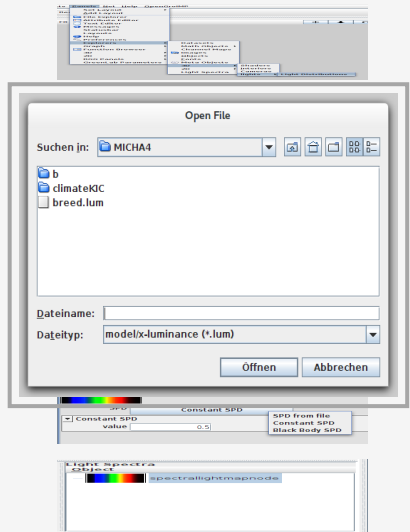
- a) Open *Panels* → *Explorers*
- b) Create new distribution object
Object → *New* → *file*
 1. Choose a file



Define Light and Spectrum References

defining references

- a) Open *Panels* → *Explorers*
- b) Create new distribution object
Object → *New* → *file*
 1. Choose a file
 2. Add or link the file

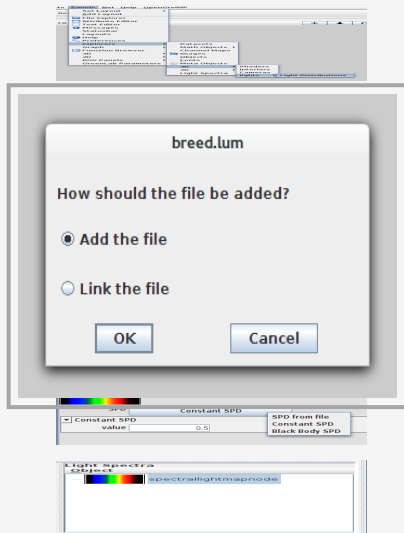




Define Light and Spectrum References

defining references

- a) Open *Panels* → *Explorers*
- b) Create new distribution object
Object → *New* → *file*
 - 1. Choose a file
 - 2. Add or link the file
- c) Create new spectra object
Object → *New* → *spectrallightmapnode*

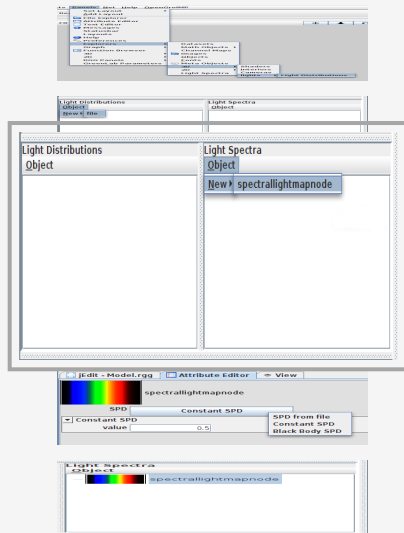




Define Light and Spectrum References

defining references

- a) Open *Panels* → *Explorers*
- b) Create new distribution object
Object → *New* → *file*
 1. Choose a file
 2. Add or link the file
- c) Create new spectra object
Object → *New* → *spectrallightmapnode*
 1. Switch to the Attribute Panel

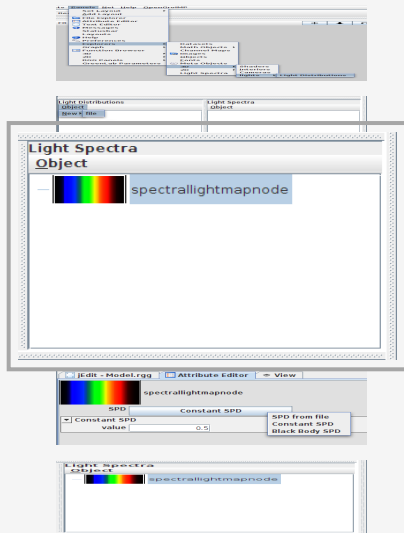




Define Light and Spectrum References

defining references

- a) Open *Panels* → *Explorers*
- b) Create new distribution object
Object → *New* → *file*
 - 1. Choose a file
 - 2. Add or link the file
- c) Create new spectra object
Object → *New* → *spectrallightmapnode*
 - 1. Switch to the Attribute Panel
 - 2. Select "SPD from file"

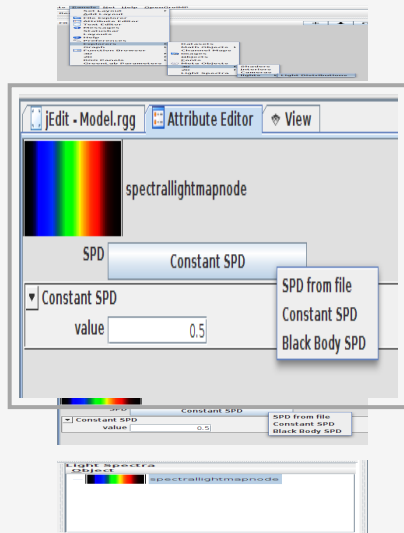




Define Light and Spectrum References

defining references

- a) Open *Panels* → *Explorers*
- b) Create new distribution object
Object → *New* → *file*
 - 1. Choose a file
 - 2. Add or link the file
- c) Create new spectra object
Object → *New* → *spectrallightmapnode*
 - 1. Switch to the Attribute Panel
 - 2. Select "SPD from file"
 - 3. Add or link the file

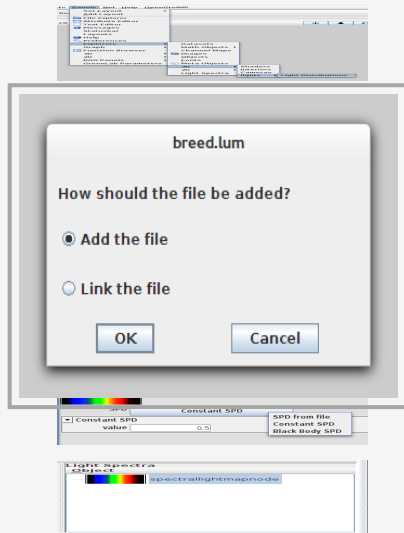




Define Light and Spectrum References

defining references

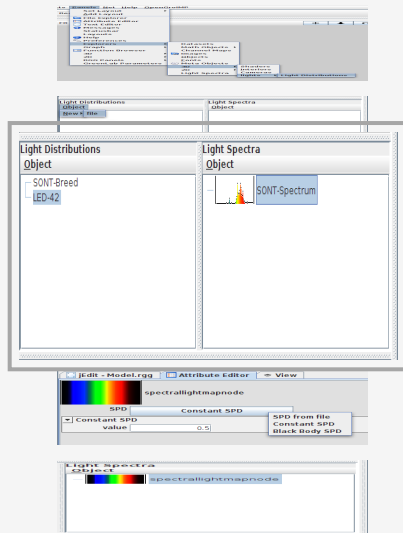
- a) Open *Panels* → *Explorers*
- b) Create new distribution object
Object → *New* → *file*
 1. Choose a file
 2. Add or link the file
- c) Create new spectra object
Object → *New* → *spectrallightmapnode*
 1. Switch to the Attribute Panel
 2. Select "SPD from file"
 3. Add or link the file
- d) Choose self-explaining names



Define Light and Spectrum References

defining references

- a) Open *Panels* → *Explorers*
- b) Create new distribution object
Object → *New* → *file*
 1. Choose a file
 2. Add or link the file
- c) Create new spectra object
Object → *New* → *spectrallightmapnode*
 1. Switch to the Attribute Panel
 2. Select "SPD from file"
 3. Add or link the file
- d) Choose self-explaining names
- e) ↩ Back





Relevant Commands I

Class	Description
FluxLightModel	
<i>compute()</i>	Compute the light distribution
<i>getAbsorbedPowerMeasurement(x)</i>	Returns the power absorbed by a node during the last call to <i>compute()</i>
<i>getSensedIrradianceMeasurement(x)</i>	Returns the irradiance sensed by a node during the last call to <i>compute()</i>
<i>setMeasureMode(MeasureMode)</i>	Sets the current measurement mode
<i>setSpectralBuckets(value)</i>	Sets the spectrum discretization resolution in buckets
<i>setSpectralDomain(min, max)</i>	Sets the simulated spectral domain



Relevant Commands II

Class	Description
Measurement	
<i>Measurement</i> (length)	Constructor; length, the dimension of the measurement vector
<i>integrate</i> ()	Returns the sum of the measurement vector elements
LightNode	
<i>setLight</i> (x)	Sets the <i>Light</i> to be used by this <i>LightNode</i>
Library	
<i>spectrum</i> (name)	Returns a <i>SpectrumRef</i> instance which refers to the spectrum named <i>name</i>
<i>light</i> (name)	Returns a <i>LightDistributionRef</i> instance which refers to the light distribution named <i>name</i>



Conversion: IES to LUM

IES file

```
IESNA:LM-63-2002
[TEST] LVE2326600
[TESTLAB]
[MANUFAC] PHILIPS
[LUMCAT]
[LUMINAIRE] GP-TOPlight DRB
[LAMP] LED TOP LIGHT
[BALLAST] 196W
[ISSUEDATE] 2013-08-07
[OTHER] B-Angle = 0.00 B-Tilt = 0.00
TILT=NONE
1 450.00 1 37 145 1 2 0.050 1.250 0.060
1.0 1.0 187.00
0.00 2.50 ... 87.50 90.00
0.00 2.50 ... 357.50 360.00
195.42 192.25 188.15 184.10 182.14 ...
195.42 192.42 188.15 184.12 181.94 ...
...
```

LUM file

```
37
145
195.42 192.25 188.15 184.10 182.14 ...
195.42 192.42 188.15 184.12 181.94 ...
...
```

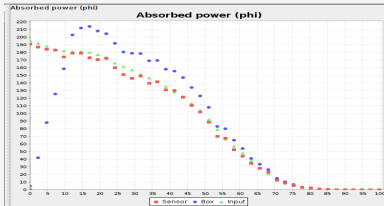
- ▶ remove everything of the IES file except the red parts
- ▶ first two lines number of angles,
- ▶ followed by rows of values for each angle

↪ [Back](#)





Comparison Box vs. Sensor



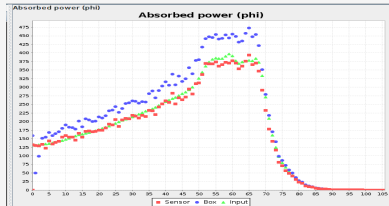
LED
 Rays : 50M
 vertical angles (phi) : 145
 horizontal angles (theta) : 74
 number of sensors : 10730

input vs measured:

	phi=0°	theta=20.00°	total [cd]
I	3821.52	15119.48	554644.06

Sensor			
M	3716.94	14717.64	542202.90
M/I	0.973	0.973	0.978
MSE	9.651	19.358	9.731

Box			
M	3778.61	17749.04	551289.10
M/I	0.989	1.174	0.994
MSE	1143.13	341.54	1187.79



SONT
 Rays : 50M
 vertical angles (phi) : 36
 horizontal angles (theta) : 180
 number of sensors : 6480

input vs measured:

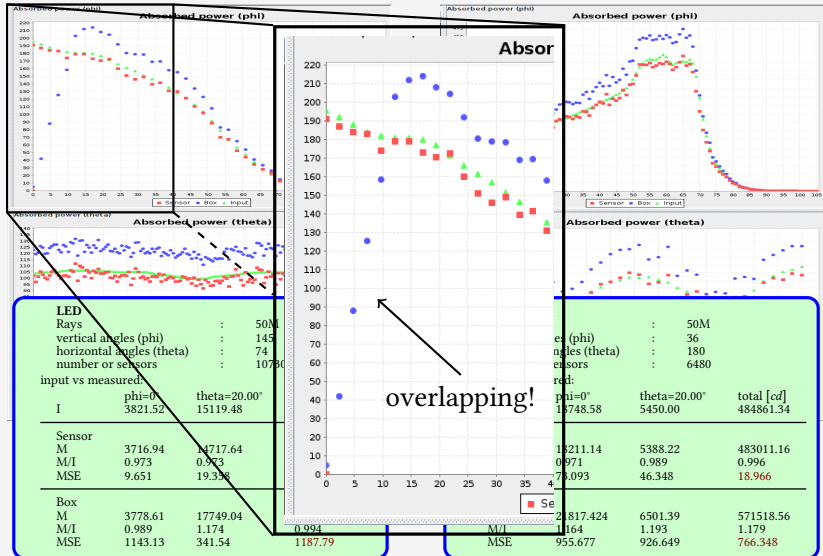
	phi=0°	theta=20.00°	total [cd]
I	18748.58	5450.00	484861.34

Sensor			
M	18211.14	5388.22	483011.16
M/I	0.971	0.989	0.996
MSE	73.093	46.348	18.966

Box			
M	21817.424	6501.39	571518.56
M/I	1.164	1.193	1.179
MSE	955.677	926.649	766.348



Comparison Box vs. Sensor





Common Problems

- ▶ configuration
 - ▶ incompatible bit-versions of Java and libraries
 - ▶ two graphics cards (onboard and real) → disable the onboard card
 - ▶ 'use multiple devices' produces errors (at least for rendering) → disable this option



Common Problems

- ▶ configuration
 - ▶ incompatible bit-versions of Java and libraries
 - ▶ two graphics cards (onboard and real) → disable the onboard card
 - ▶ 'use multiple devices' produces errors (at least for rendering) → disable this option
- ▶ application
 - ▶ wrong/different model set-up compared with reality
 - ▶ misunderstanding of *setDiffuse* - it is the amount of reflected light
 - ▶ mixed units: Watt, Candela, $\mu \text{ mol photons m}^{-2} \text{ s}^{-1}$



Table of Contents

Appendix

References



References I

- ▶ ANSI/IESNA LM-63-02, IES-Specification, Standard File Format for the Electronic Transfer of Photometric Data
- ▶ The Basics of Efficient Lighting, A Reference Manual for Training in Efficient Lighting Principles, National Framework for Energy Efficiency (Australia), First Edition, December 2009
- ▶ Sharpe, L.T., Stockman, A., Jagla, W., & Jägle, H. (2005) A luminous efficiency function, $V^*(\lambda)$, for daylight adaptation. Journal of Vision 5:948-968.
- ▶ van Antwerpen, D.G. (2011) High Performance Spectral Light Transport Model for Agricultural Applications, Poster HPG
- ▶ van Antwerpen, D.G. (2011) Unbiased physically based rendering on the GPU, Master thesis, Delft University of Technology



References II

- ▶ R Development Core Team (2005) R: A language and environment for statistical computing, reference index version 2.2.1. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL <http://www.R-project.org>.
- ▶ Phong, B.T. (1975) Illumination for computer generated pictures, Communications of ACM 18, 6:311–317.
- ▶ Kasperbauer, M.J. (1987) Far-red light reflection from green leaves and effects on phytochrome-mediated assimilate partitioning under field conditions. Plant Physiology, 85(2):350–354.
- ▶ Hneke, M., Buck-Sorlin, G.H. (submitted) Using a full spectral raytracer for the modelling of light microclimate in functional-structural plant modelling