Advanced light modelling with GroIMP

Michael Henke[1,2], Gerhard Buck-Sorlin[1]

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

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

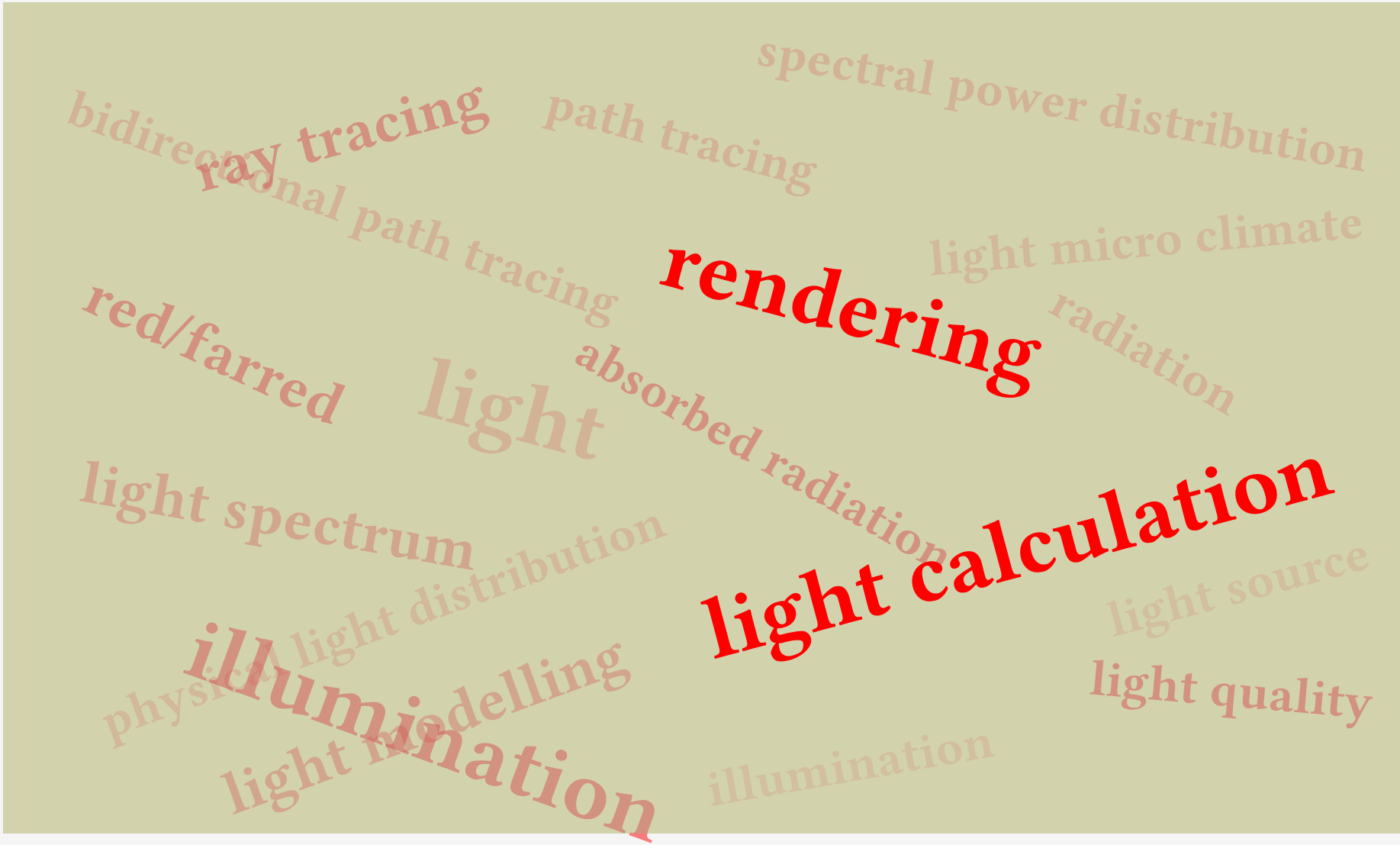
Tutorial and Workshop
"Functional-Structural Plan Modelling with GroIMP and XL"
combined with the 7th GroIMP user and developer meeting

Angers, 2015-05-06
Extended version (January 20, 2020)

Table of Contents

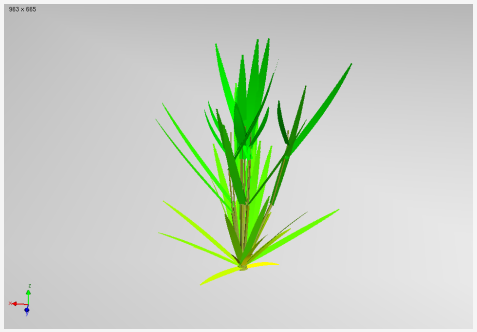
Disambiguation

Disambiguation



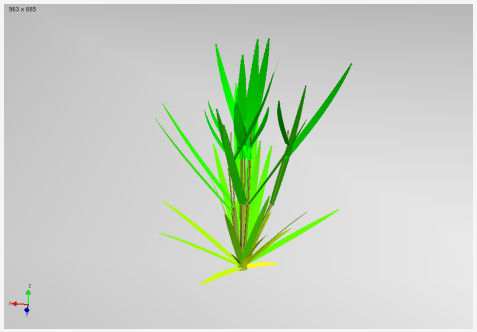
Rendering vs. Light calculation

3d model



Rendering vs. Light calculation

3d model

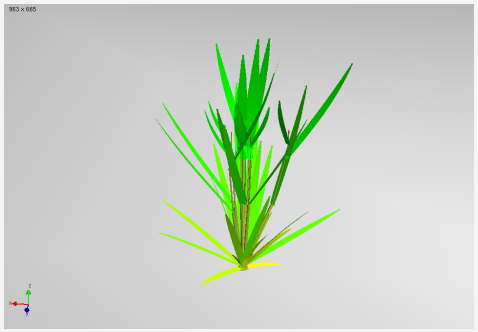


rendering

process of generating an image

Rendering vs. Light calculation

3d model



rendering

process of generating an image

light calculation

1	0.8234641
2	0.7027052
3	1.3539617
4	0.6167236
5	0.6495002
6	1.3293481
...	

process to calculate the light distribution

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?
  - Flux Light Model / GPU Flux
  - Dietger van Antwerpen (2011)

Requirements

- programmable graphics card with OpenCL support (SSE $>$ 4.1, Wikipedia: SSE4)

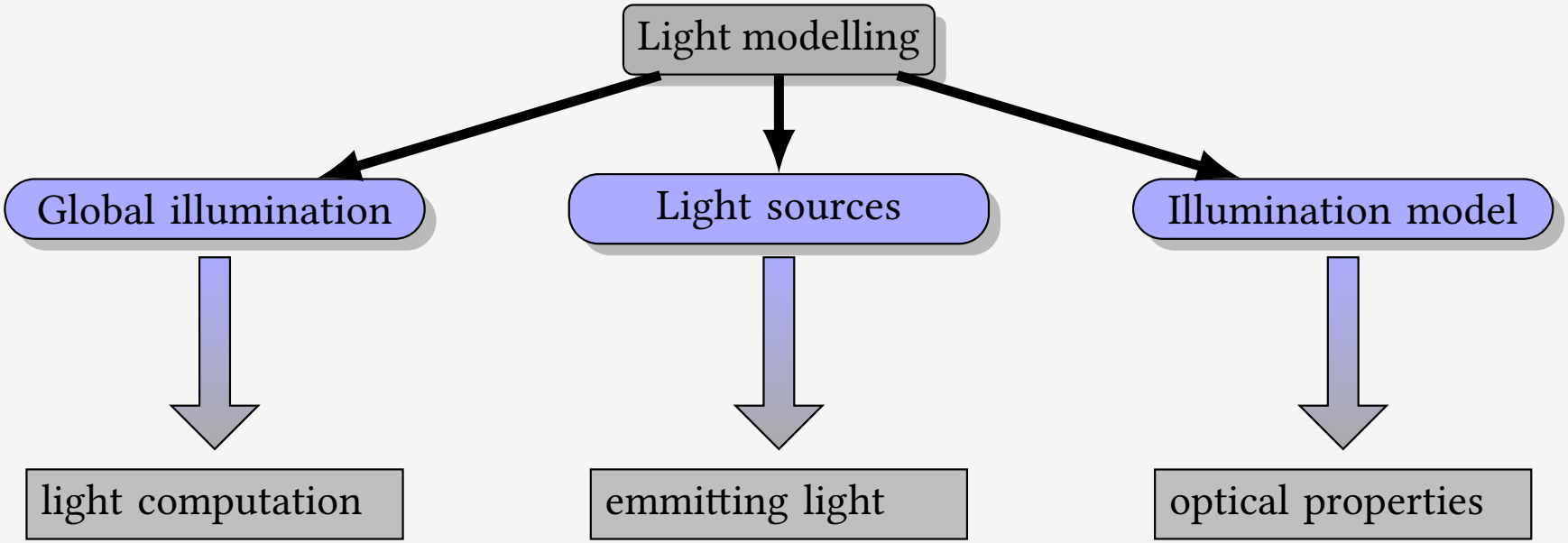
Examples: high end graphics cards

Nvidia GeForce GTX 980

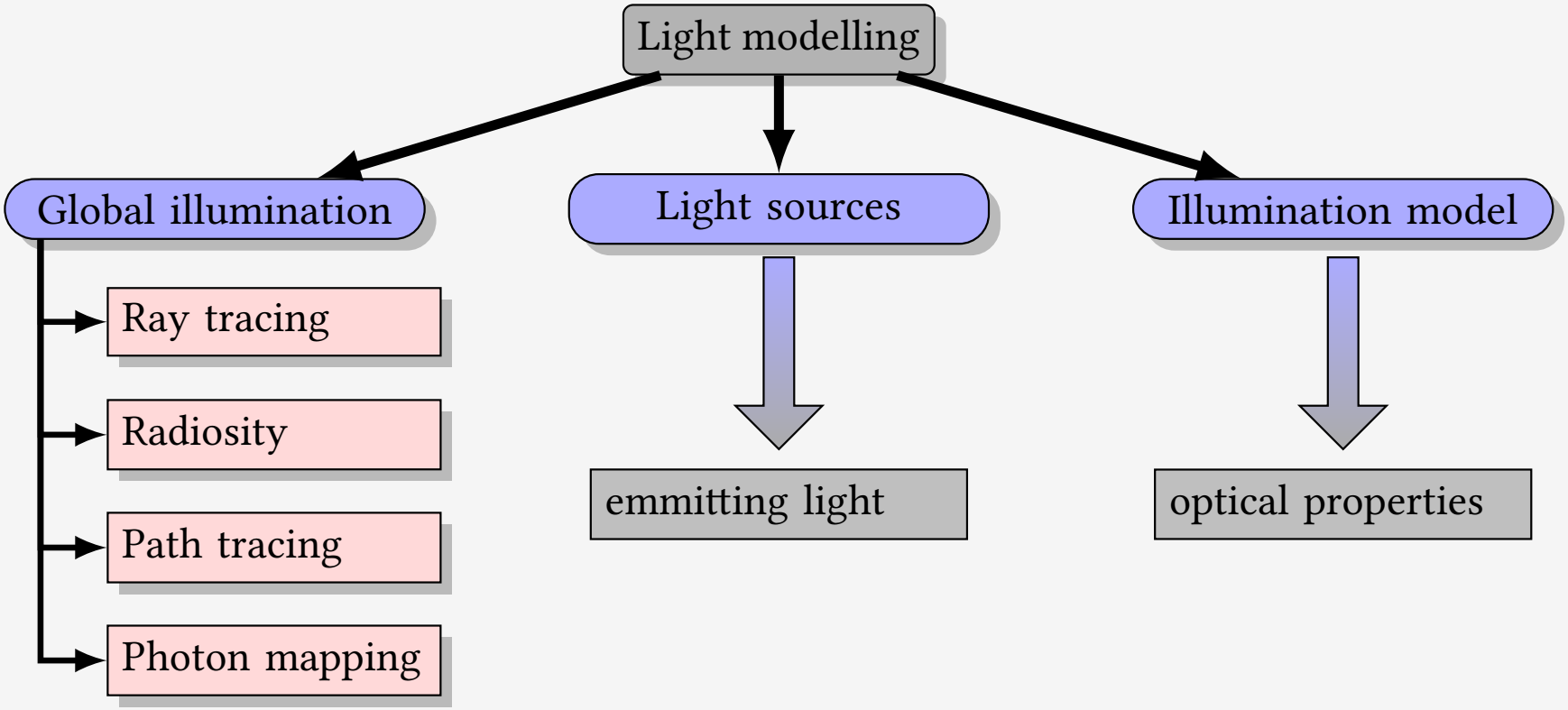
MSI Radeon R9 290

Table of Contents

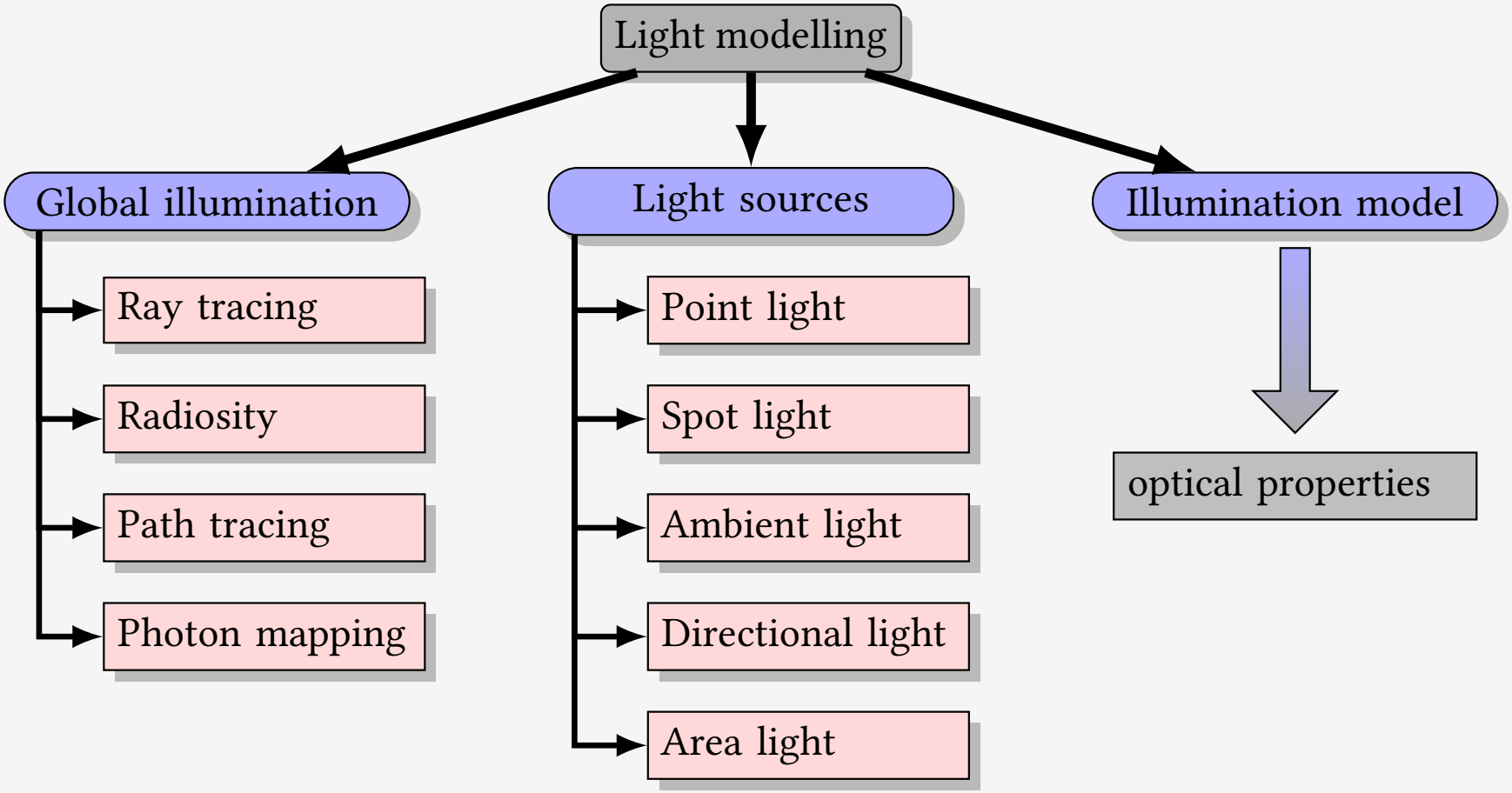
Light modelling



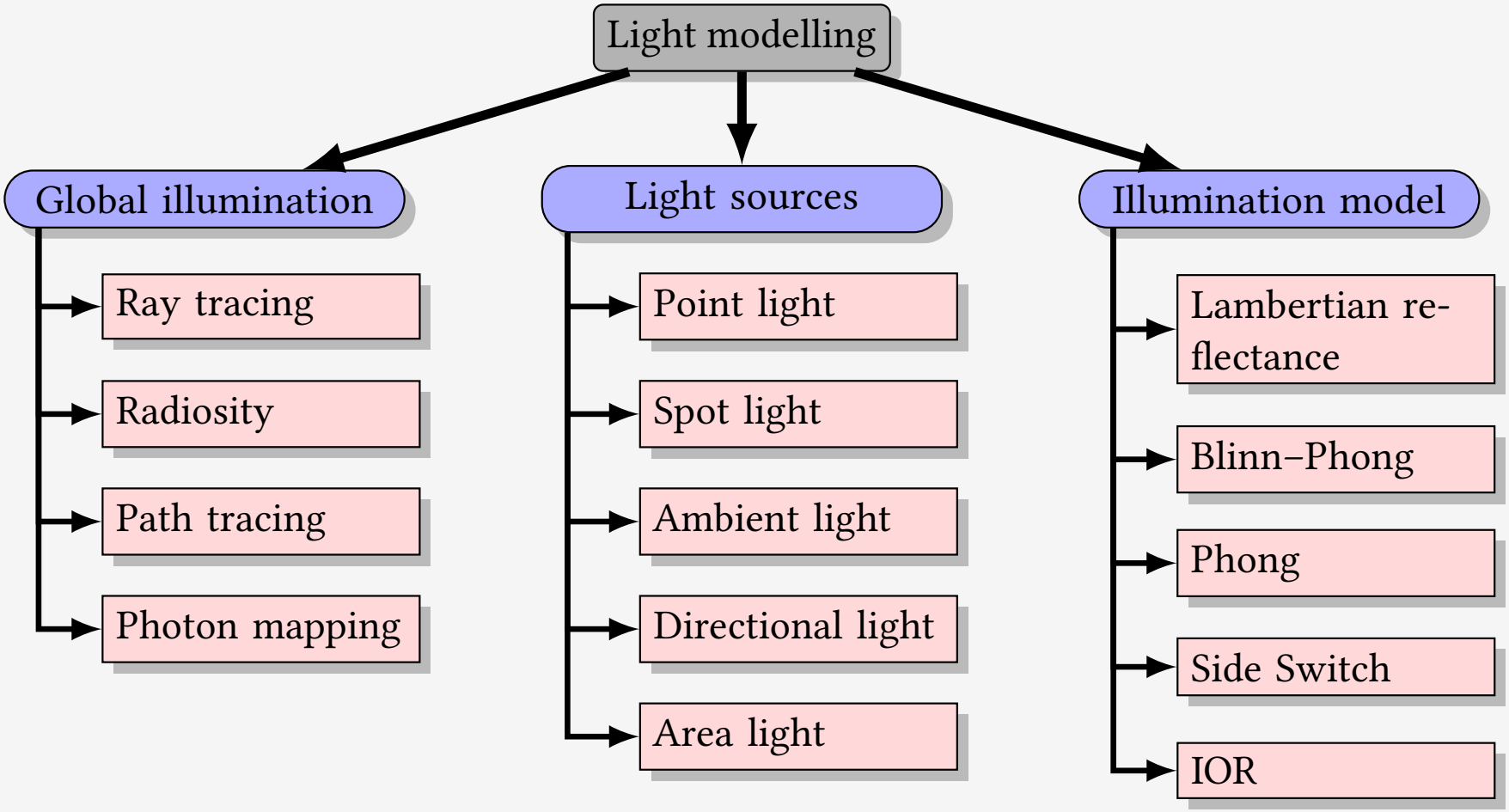
Light modelling



Light modelling

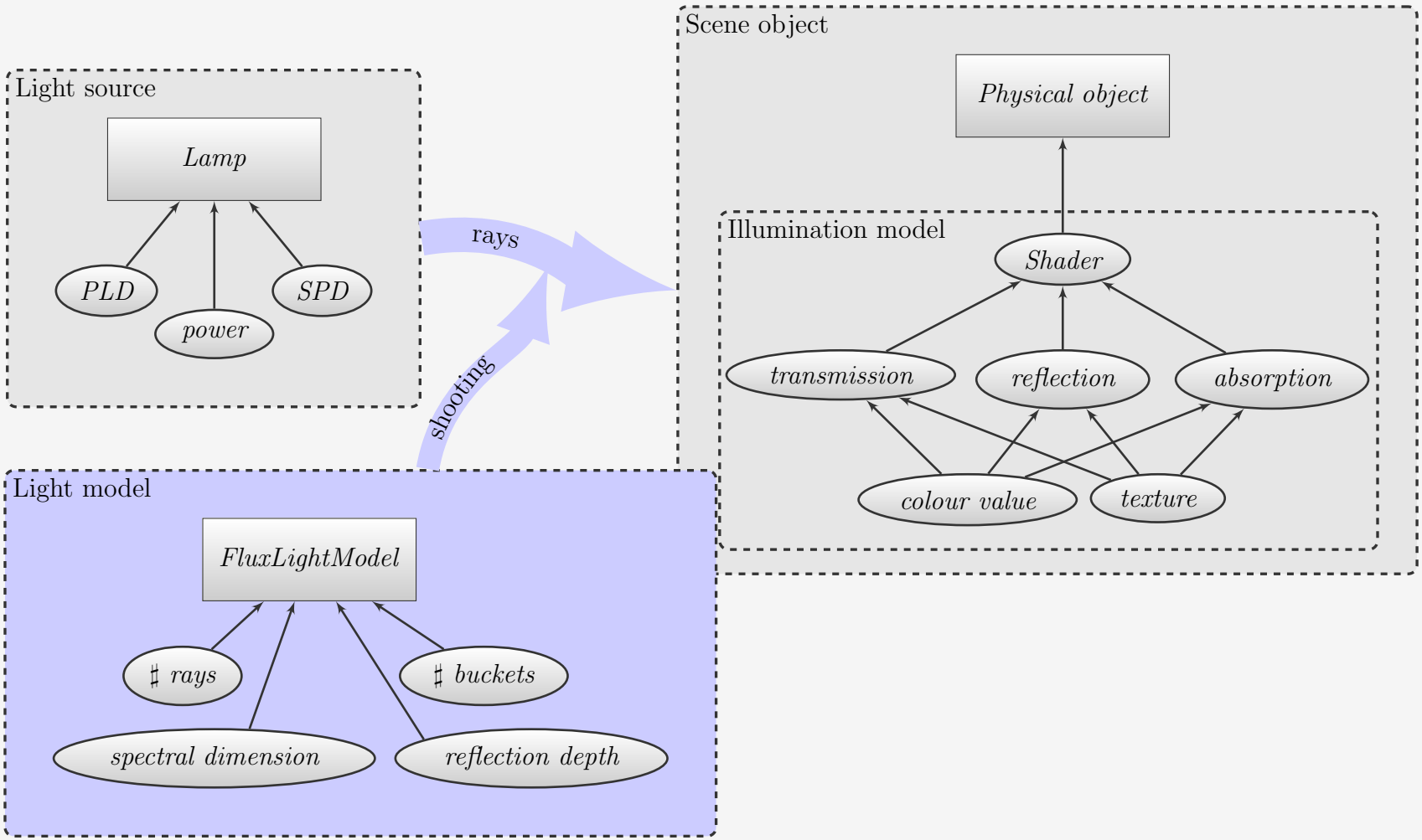


Light modelling



(Henke and Buck-Sorlin 2017)

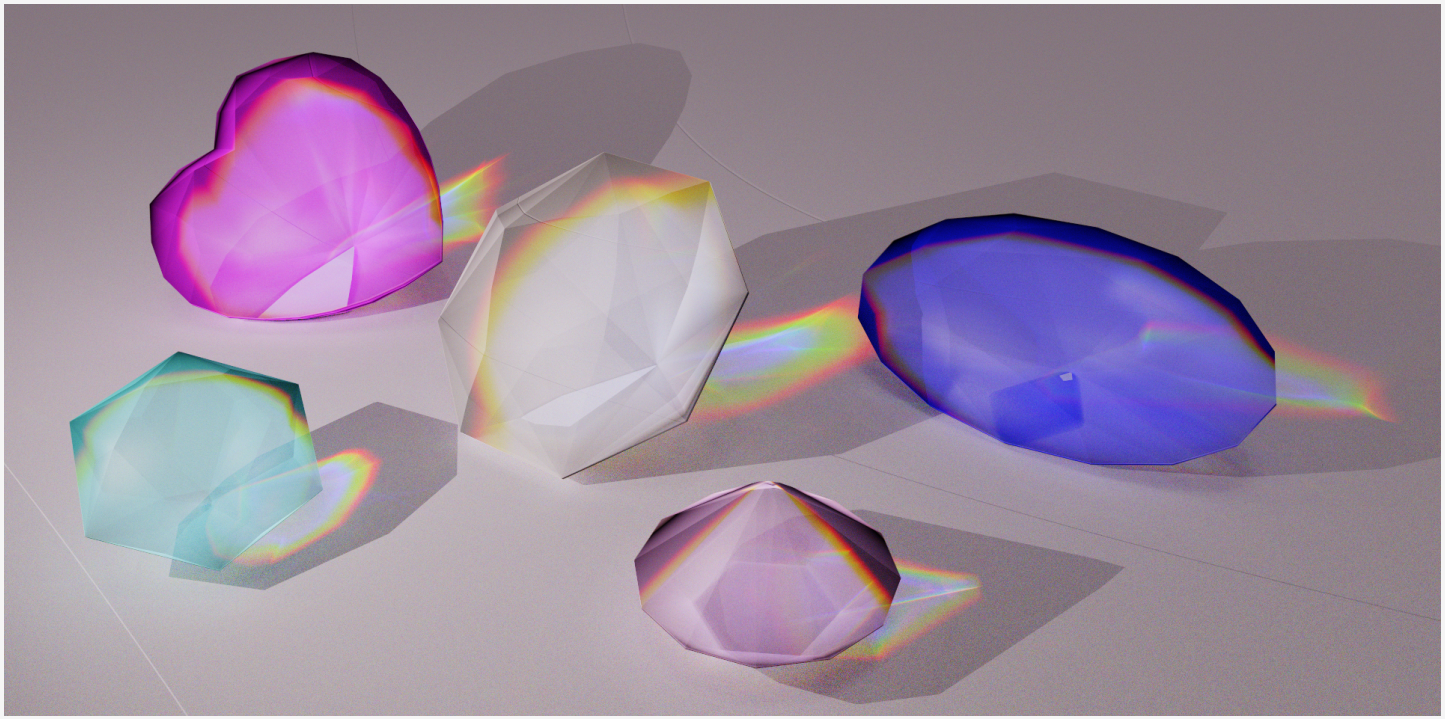
Progress in light modelling



(Henke and Buck-Sorlin 2017)

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 2017)

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.gpuflux.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.gpuflux.tracer.FluxLightModelTracer.
       MeasureMode;
4  import de.grogra.gpuflux.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.
           getSensedIrradianceMeasurement(x);
9      float absorbedPower = spectrum.integrate();
10     ...
11   }
12
13
14   x:Box ::> {
15     Measurement spectrum = lm.
           getAbsorbedPowerMeasurement(x);
16     float absorbedPower = spectrum.integrate();
17     ...
18   }
19 ]
```

Flux Light Model: Setup and Usage - III

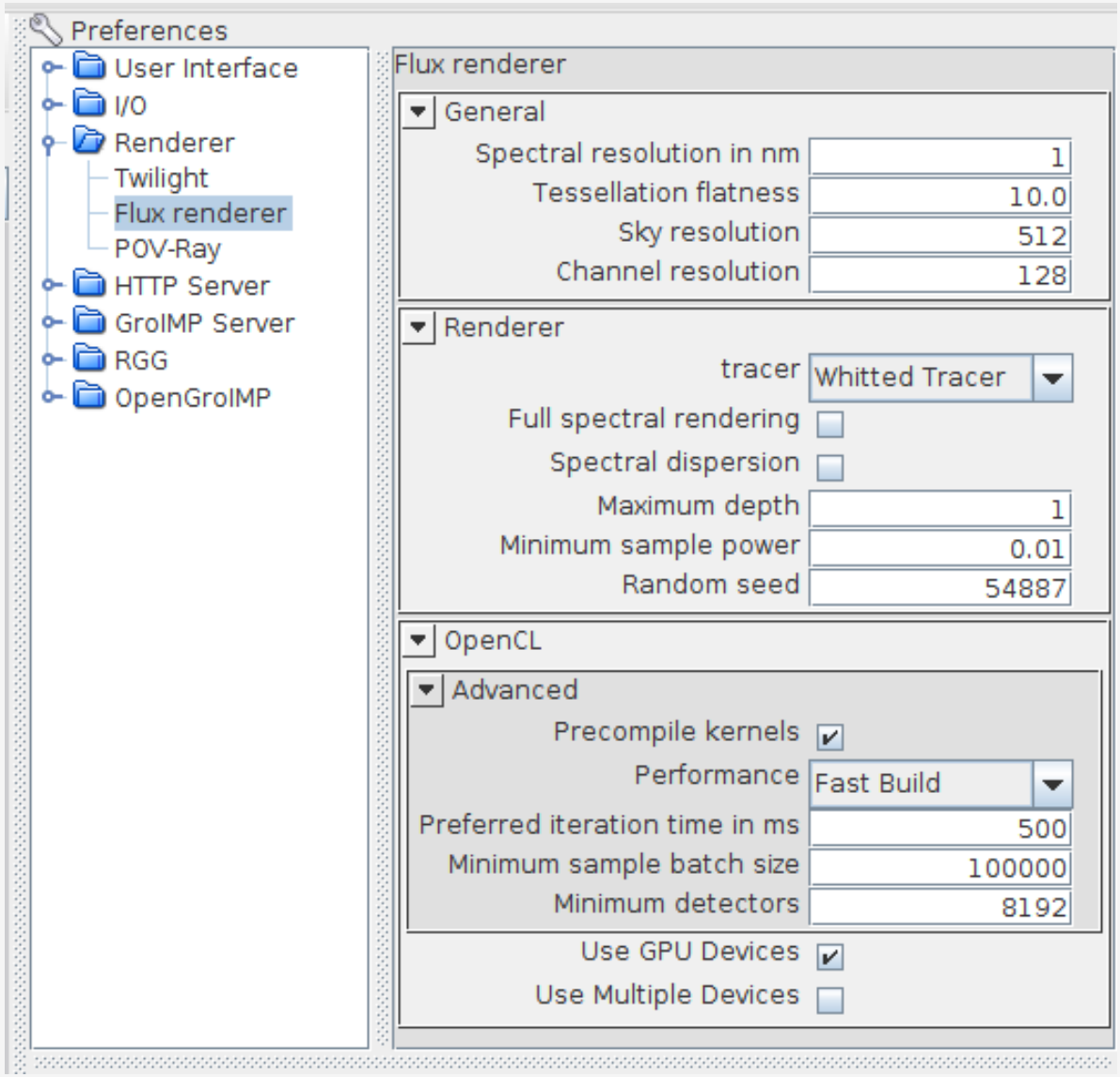
Absorbed power per bucket

```
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  }
```

Flux Light Model: Setup and Usage - IIII

```
1   ...
2
3   // sets the maximum negligible power quantum
4   lm.setCutoffPower(0.01); // default: 0.001
5
6   // disables the simulation of sensor
7   lm.setEnableSensors(false); // defaul: true
8
9   // sets the random seed for the random number generator
10  lm.setRandomseed(123456);
11
12  // disables dispersion
13  lm.setDispersion(false); // defaul: true
14
15  ...
```

Flux Light Model: Preferences



Scattering rays over light sources

- total light contribution of the light source determines the number of rays created for that light source

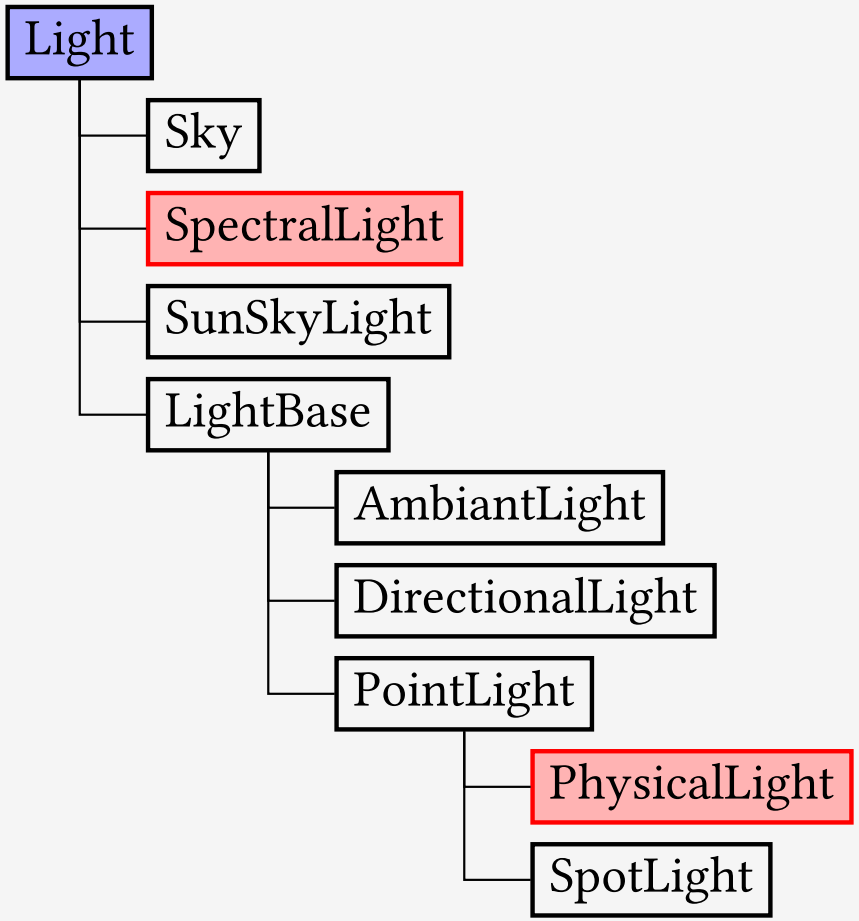
Example: a scene with

- two light sources with a power of 10W and 90W,
- a total number of 10.000 rays

→ the first light will create 1.000 rays, the second one 9.000 rays

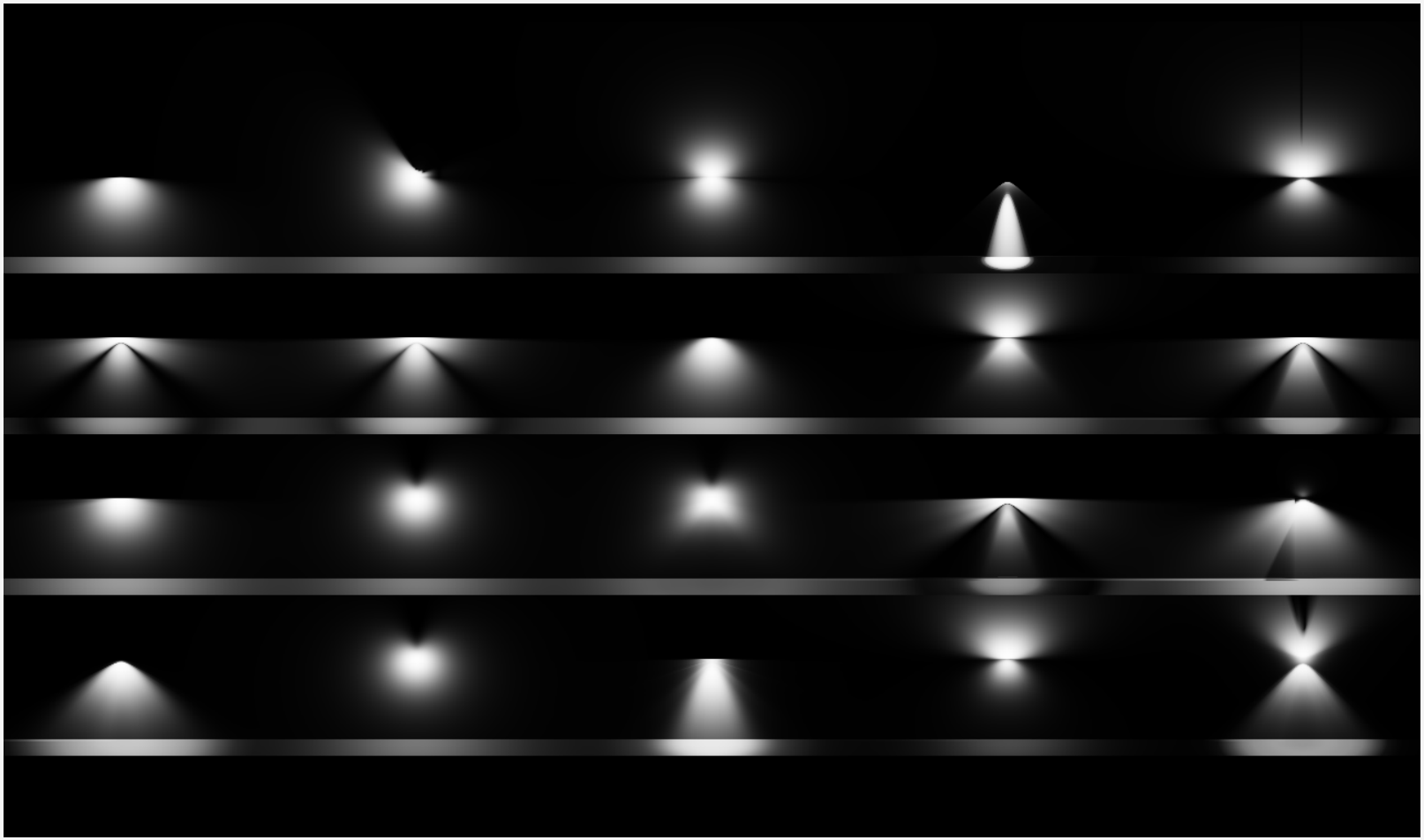
Available Light Nodes

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



New Light Source - *SpectralLight*

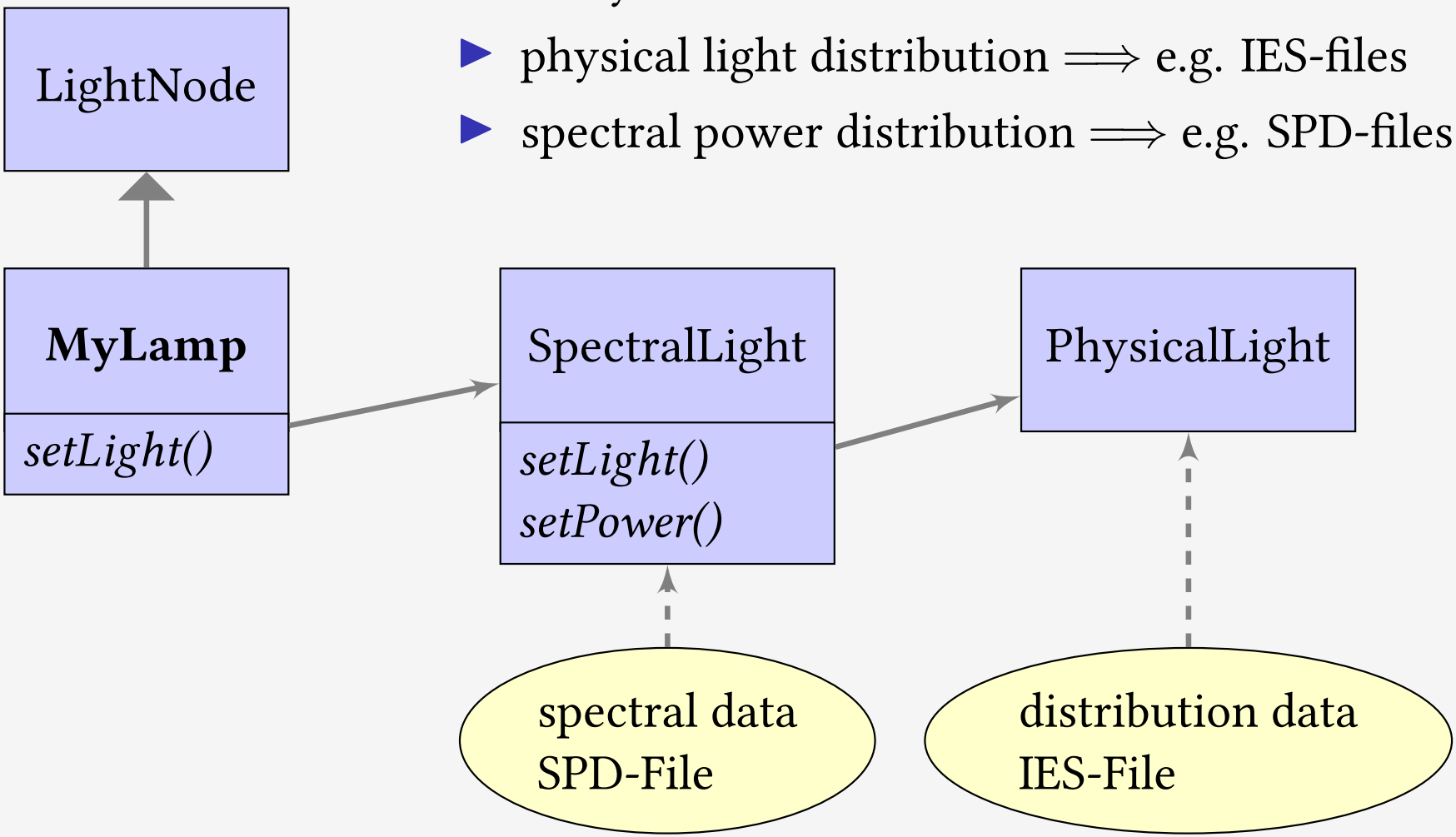
- generic lamp ($\Longrightarrow$ *LampDemo.gsz*)
- individually defined light source



Spectral Light Source

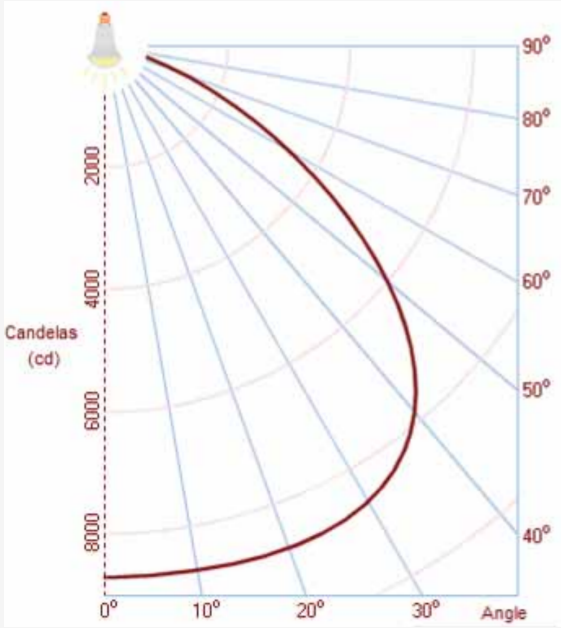
defined by two files:

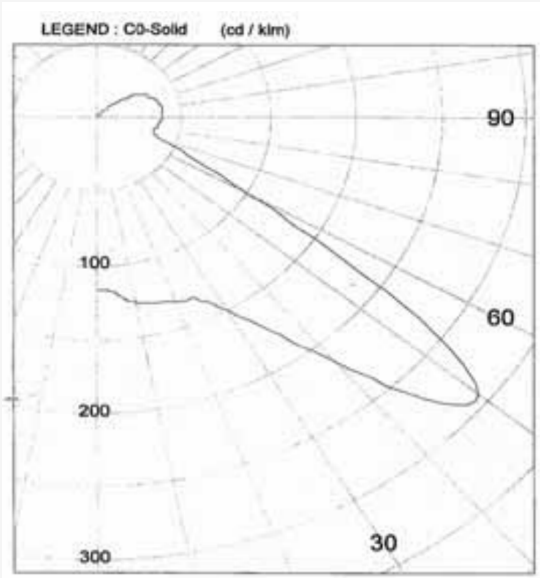
- physical light distribution $\Longrightarrow$ e.g. IES-files
- spectral power distribution $\Longrightarrow$ e.g. SPD-files



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, ...



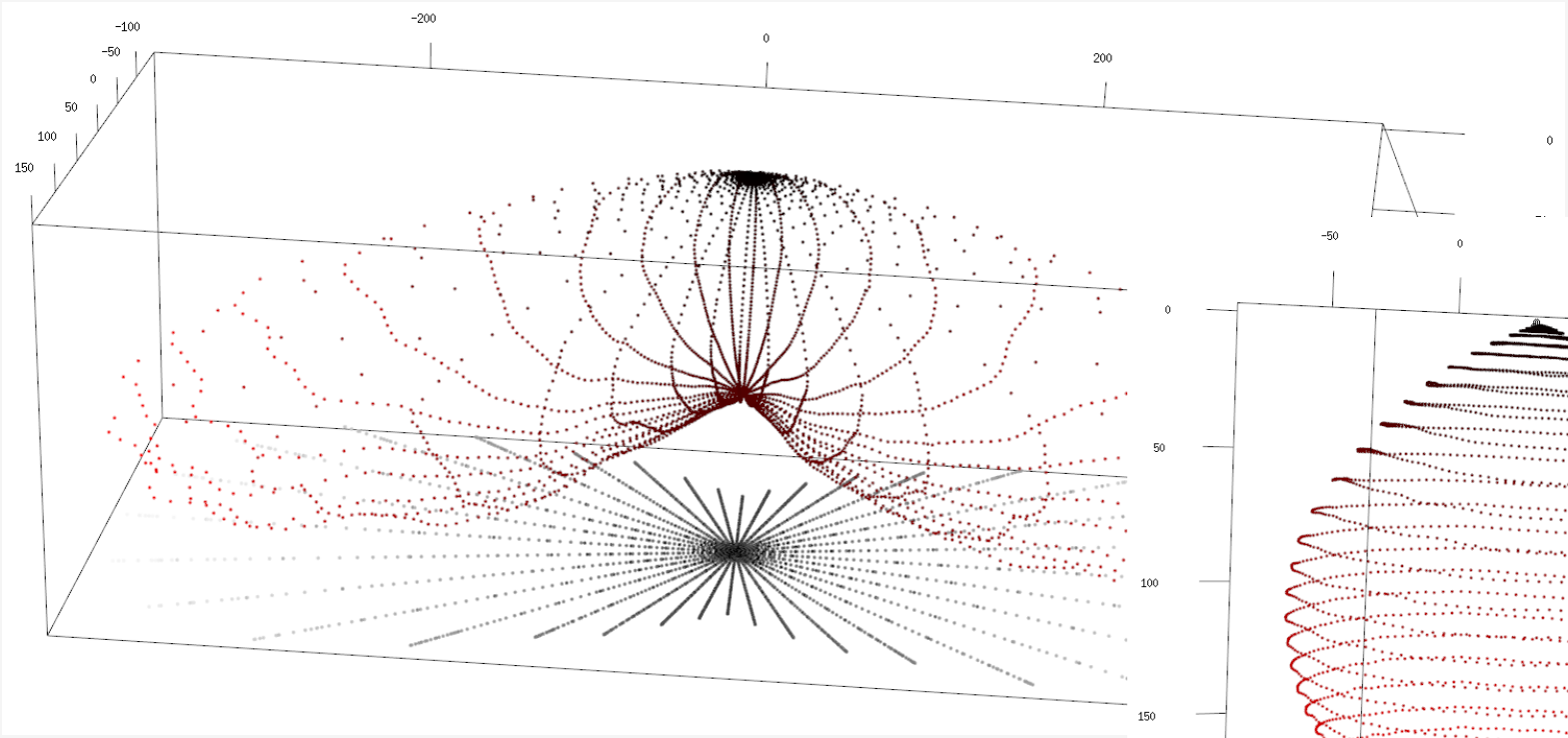


INTENSITY DATA (cd / klm)

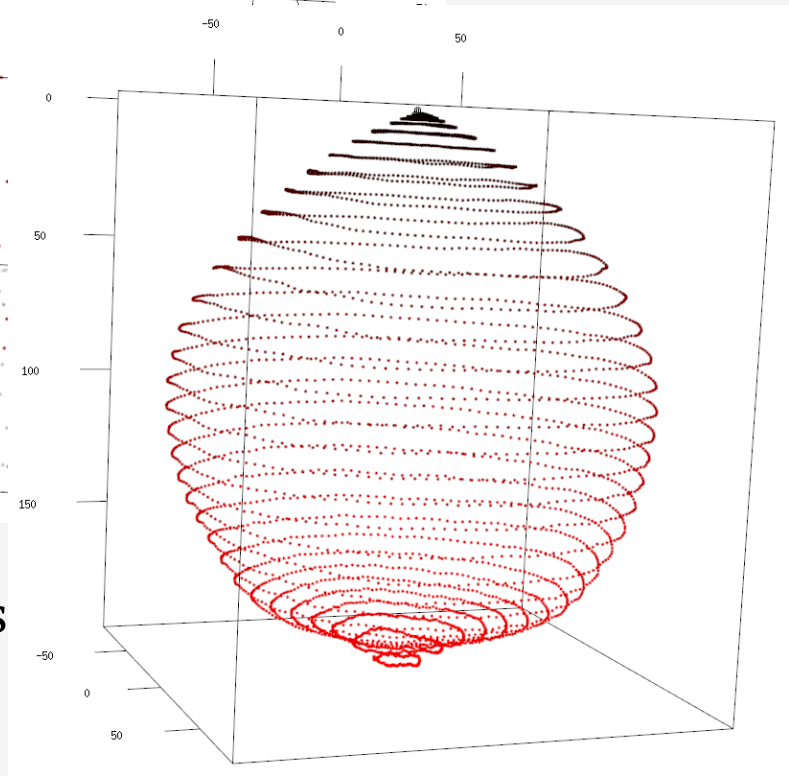
Gamma	C0	LMS.	Gamma	C0	LMS.
0	117		90	38	
5	119	12	95	39	42
10	127		100	39	
15	130	37	105	38	40
20	132		110	36	
25	135	64	115	34	34
30	150		120	31	
35	175	112	125	27	24
40	214		130	21	
45	268	204	135	14	11
50	290		140	8	
55	205	180	145	5	3
60	103		150	4	
65	59	62	155	3	2
70	38		160	3	
75	34	37	165	3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

Physical light distribution - 3D-Visualization

SON-T



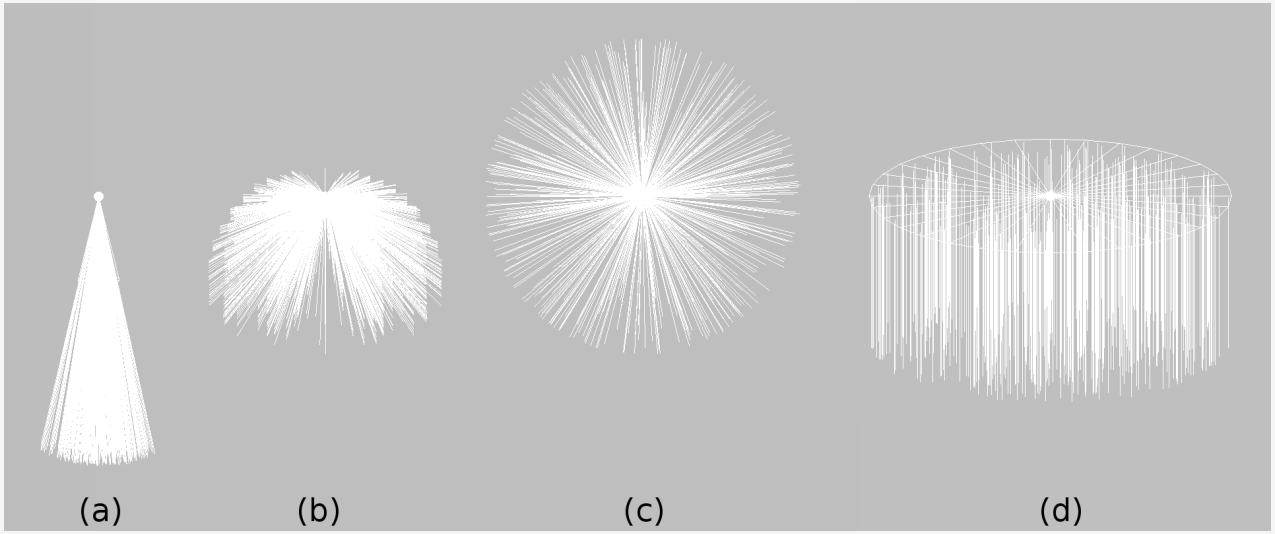
LED



- Example of 3d physical light distributions
- each point in each direction represents the power emitted per steradian

Visualization of physical light distribution for light nodes

- length of a ray equal to emitted power per steradian (sr)



spot spectral point directed

```
1 LightNode().(
2     setLight(new DirectionalLight().(
3         setVisualize(true),
4         setNumberofrays(250),
5         setRaylength(1.75)
6     ) ) );
```

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

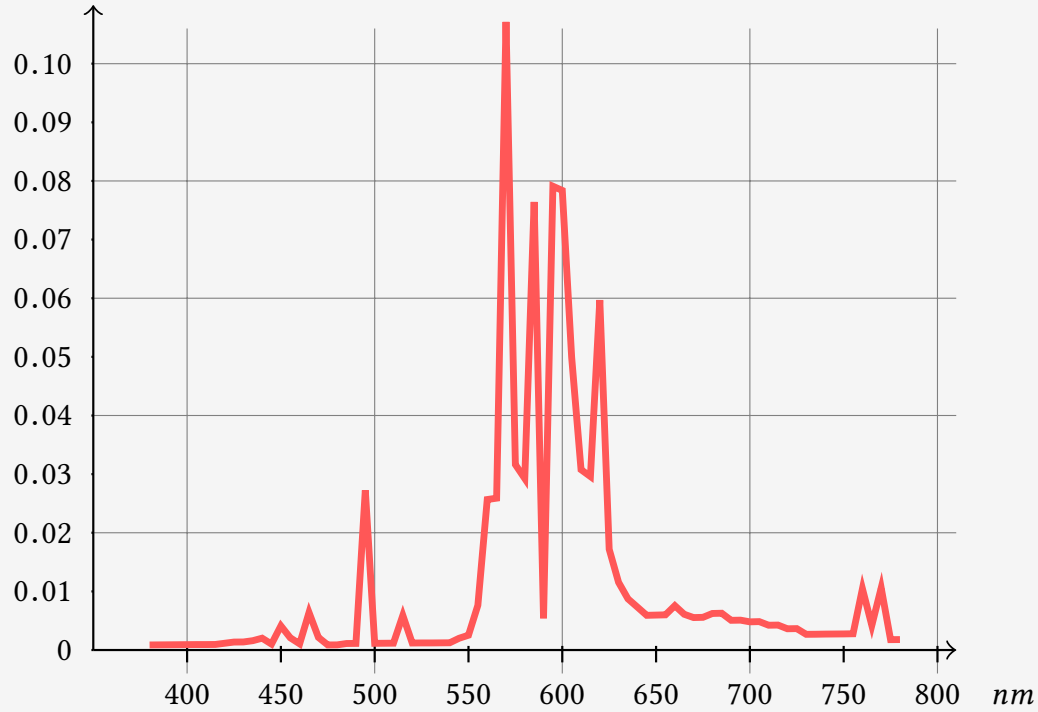
↬ How to convert IES to LUM?

Spectral power distribution

1. SPD: Spectral Power Distribution file
   - simple ASCII-file: two columns (wavelength, amplitude)
   - wavelength divided into, e.g., 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
...
```



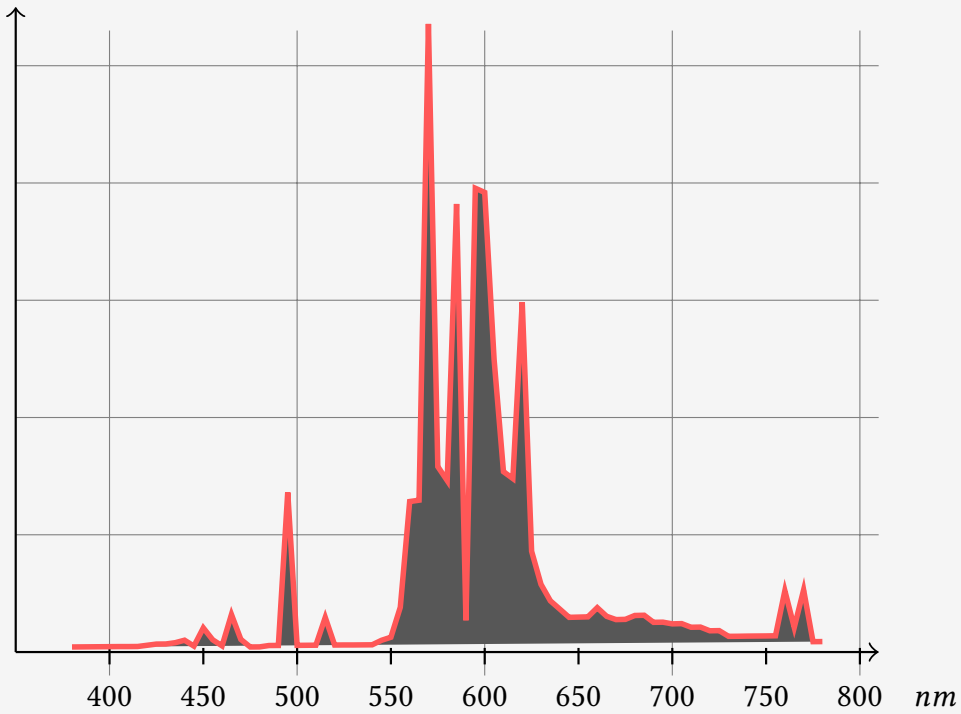
Spectral power distribution

- The total power of a lamp is distributed according to the spectral power distribution.
  $\rightarrow$ The final power distribution is proportional to the spectral power distribution, but integrates to the total power of x Watt.

Example:

power of the light source: 10 W distributed over a normed SPD

$\Longrightarrow \int SPD = 10$



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

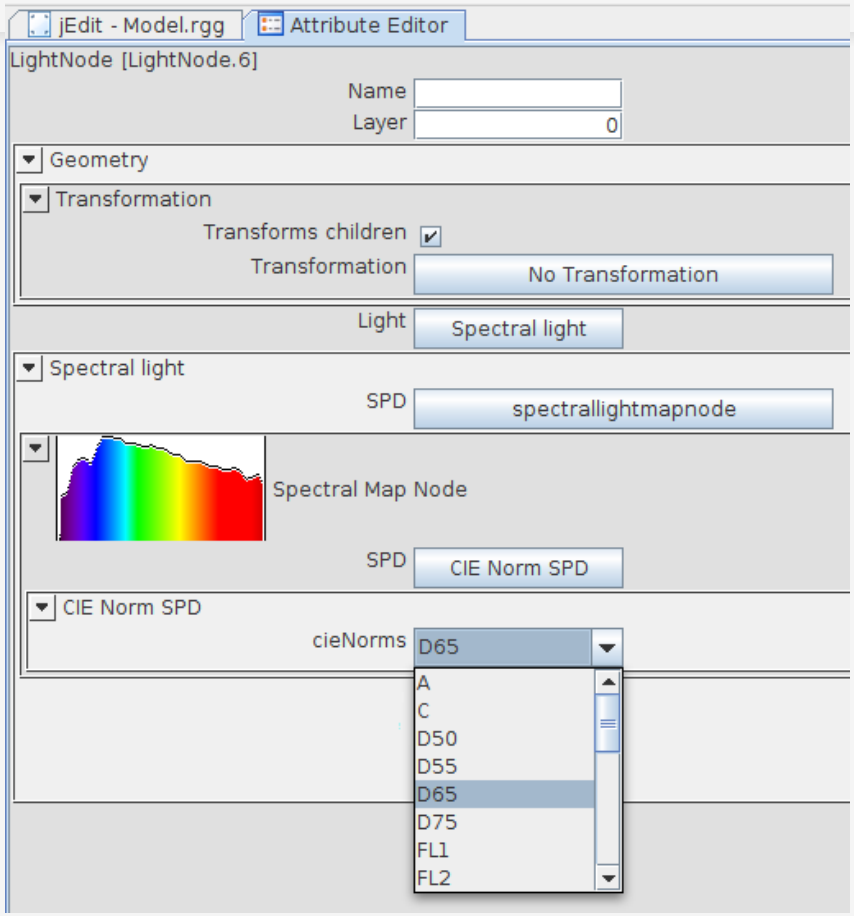
```
1 module MyLamp extends LightNode {
2   {
3     setLight(new SpectralLight().(
4       setPower(5), //[W]
5       setLight(new PhysicalLight(DISTRIBUTION)),
6       setSpectrum(SPECTRUM)
7     ));
8   }
9 }
```

↬ How to define references?

Predefined standard CIE norms (spectral power distributions)

- CIE standard illuminant:
  - A, C
  - D55, D60, D65, D75
  - FL1-FL12
  - HP1-HP5

```
1  import de.grogra.imp3d.spectral.
       CIENormSpectralCurve;
2
3  module MyLamp extends LightNode {
4    {
5      setLight(new SpectralLight(
6        new SpotLight(),
7        new CIENormSpectralCurve(
               Attributes.CIE_NORM_D55
               ),
8        500 //power
9      ));
10   }
11 }
```



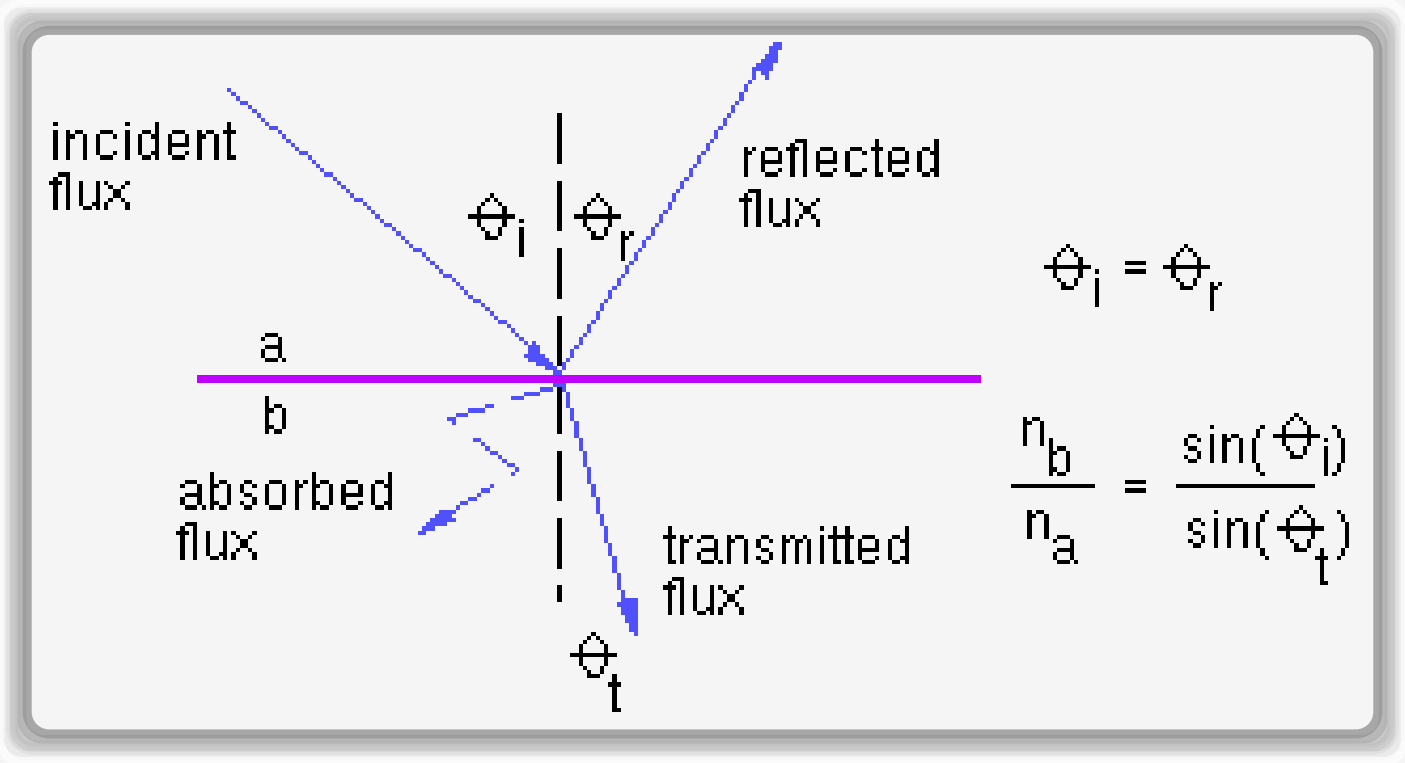
Area light

- diffuse distribution cast in the positive direction of the local z-axis of the parallelogram
- does not (yet) support user defined physical light distribution

```
1 module AreaLamp extends Parallelogram() {
2     {
3         setLight(new AreaLight().(setPower(100)));
4         setLength(1); // 1 m
5         setAxis(0.5f, 0, 0);
6     }
7 }
```

Illumination model

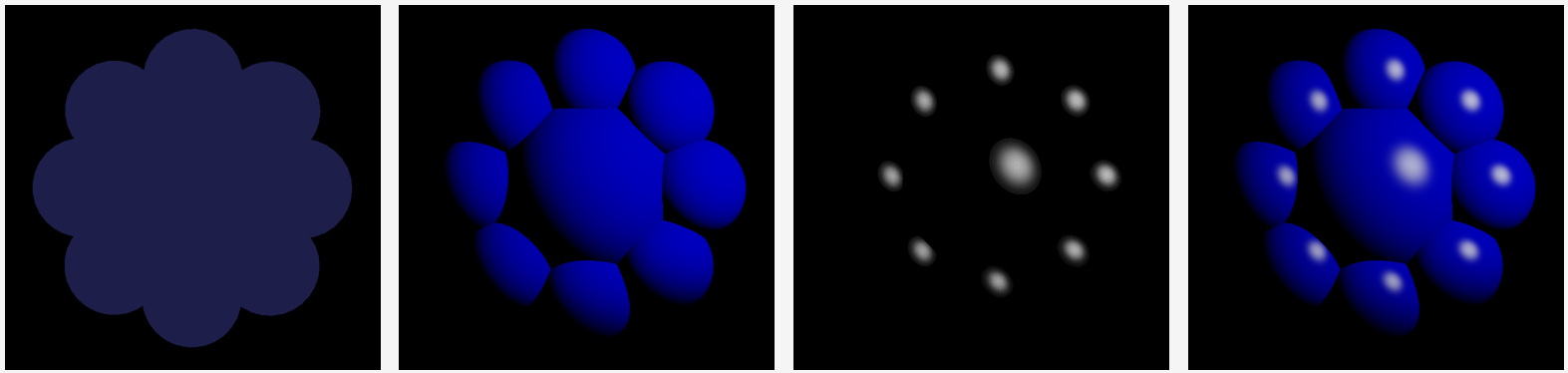
local illumination



Shader

definition of optical properties

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



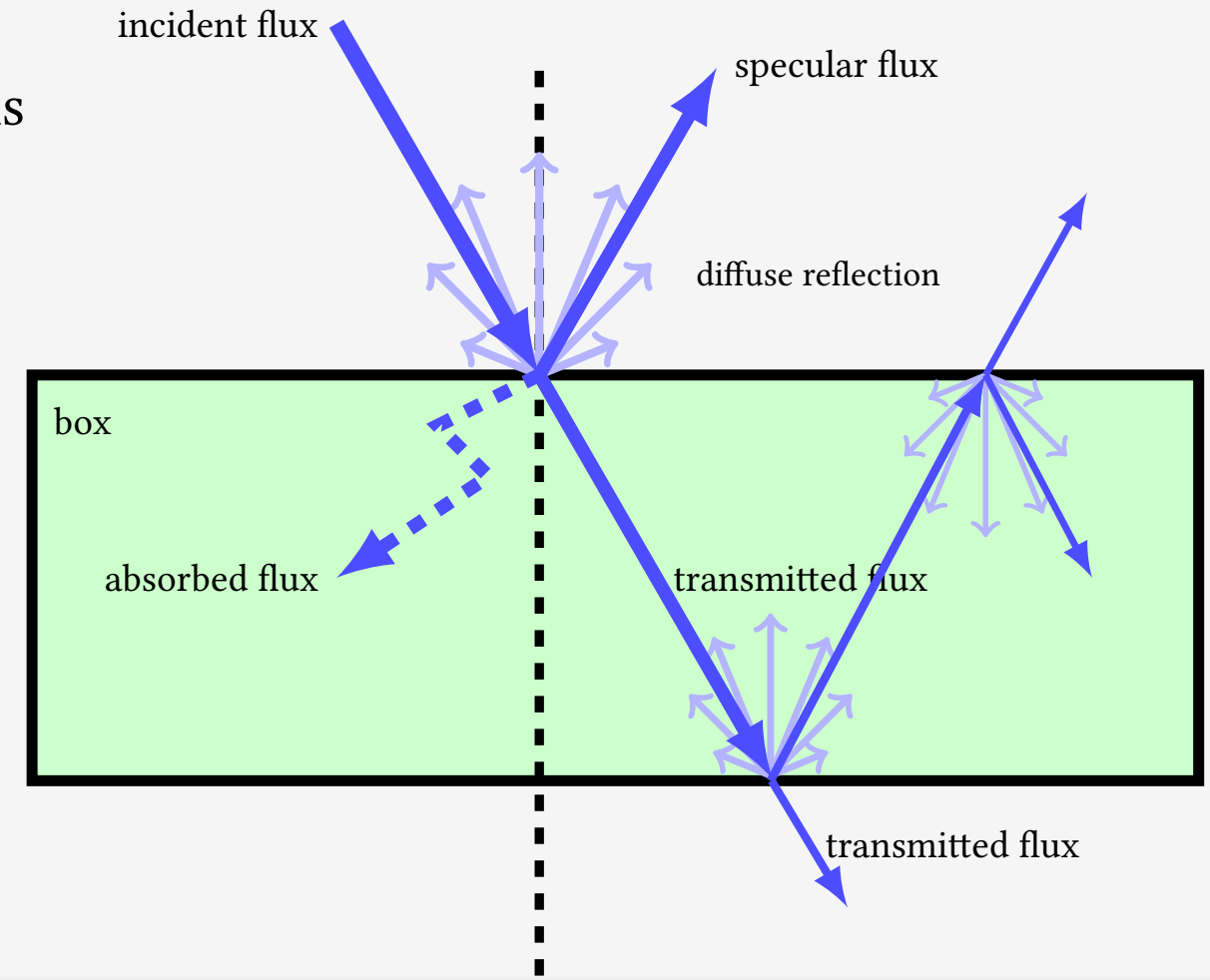
ambient + diffuse + specular = phong

(Henke and Buck-Sorlin 2017)

- Lambertian reflectance → only diffuse reflection

GroIMP Phong reflection

- for volumes multiple internal reflections are simulated
- Interior: Vacuum → photons are transmitted without altering their direction
- number of reflections is determined by:
  - recursion depth
  - cutoff power



Specular reflection vs Shininess

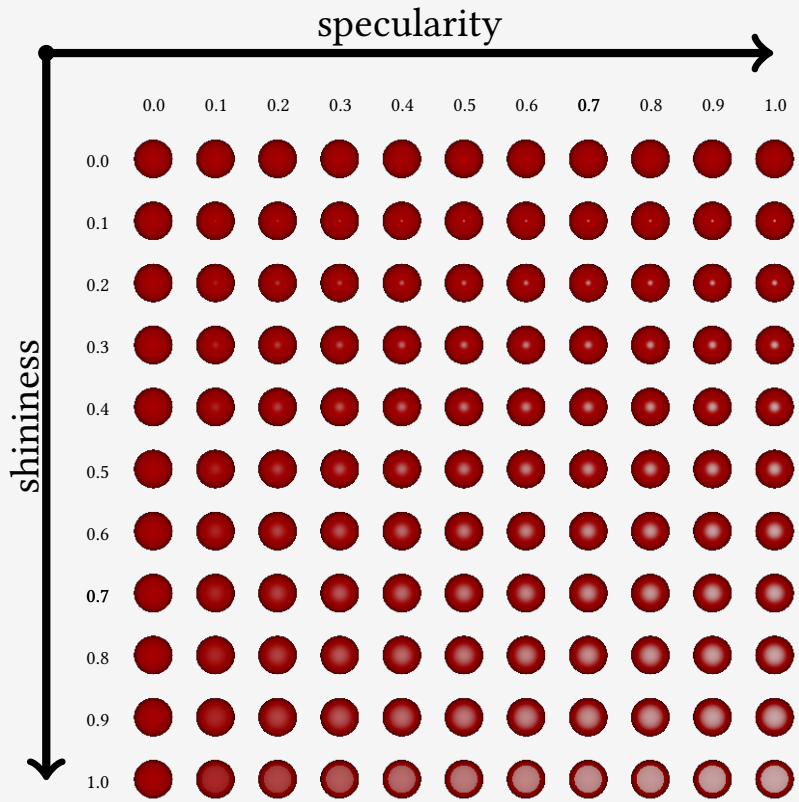
Specular reflection

Light from a light source reflects from the surface of an object directly into a camera.

Shininess

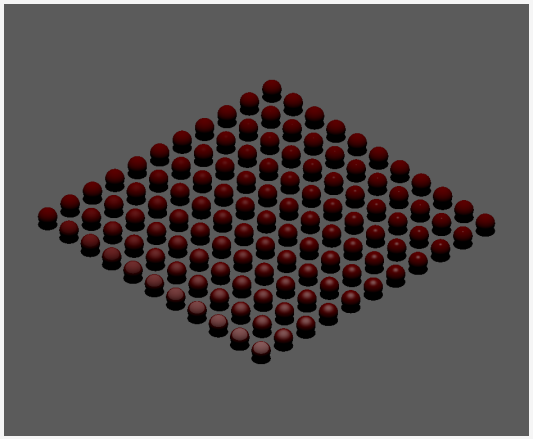
The size of the specular highlight on an object. If an object is very smooth, the specular highlight will be small. For dull or rough objects, the size will be larger.

Modern Phong shader uses roughness and glossiness instead.

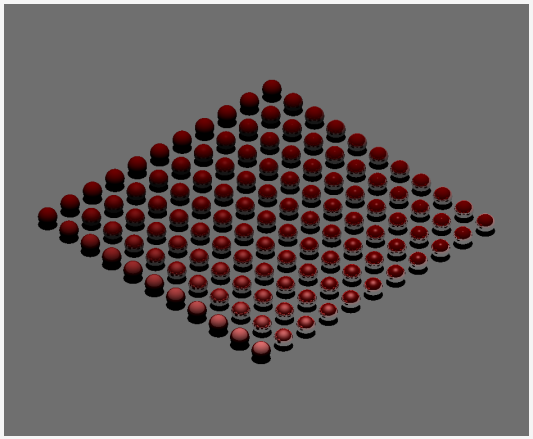


Differences in interpretation of Shininess and Specularity by light models and renderers

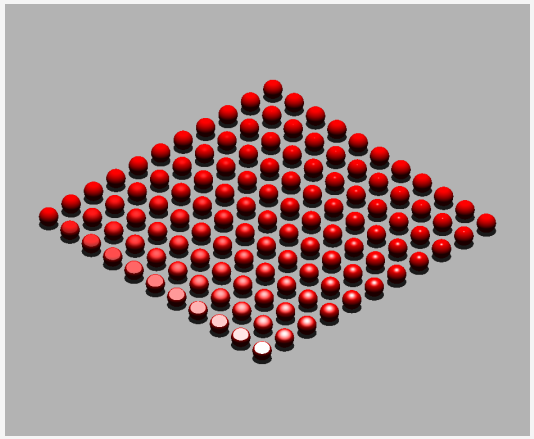
- CPU and GPU renderer and light model interprete the Shininess and Specularity differently



Flux renderer



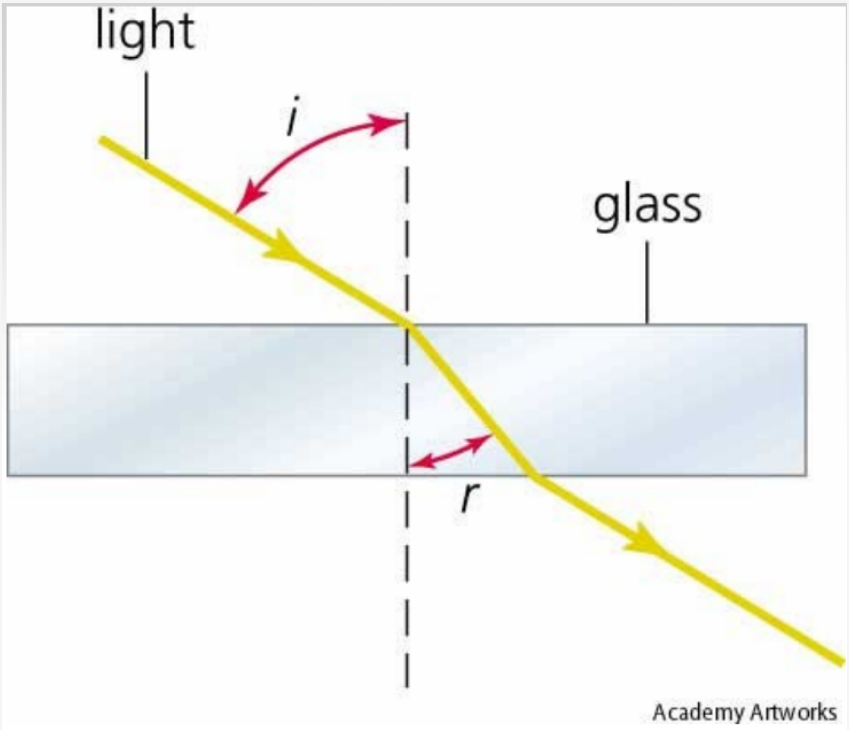
Twilight renderer



POV-Ray renderer

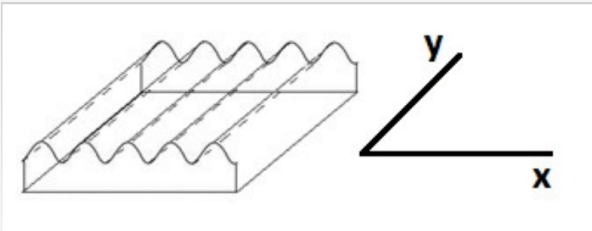
Refraction handling (CPU LM only)

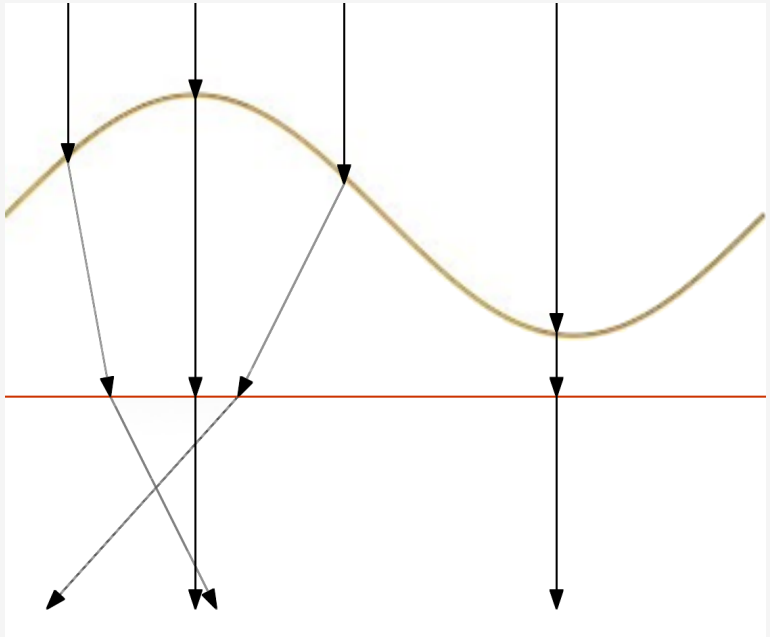
- GroIMP has supported to set the IOR value (index of refraction) for the interior of a solid object for years.
- But the (CPU) light model didn't properly handle it. So refraction wasn't possible. This has been fixed.



Application of refraction (CPU LM only)

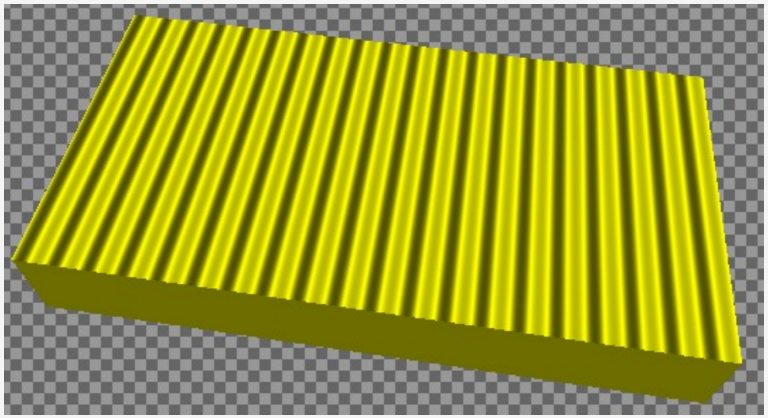
- Light scattering with corrugated glass
- Glass with a wave pattern on one side will scatter light in one direction due to refraction
- Can be used to make e.g. sun light more diffuse

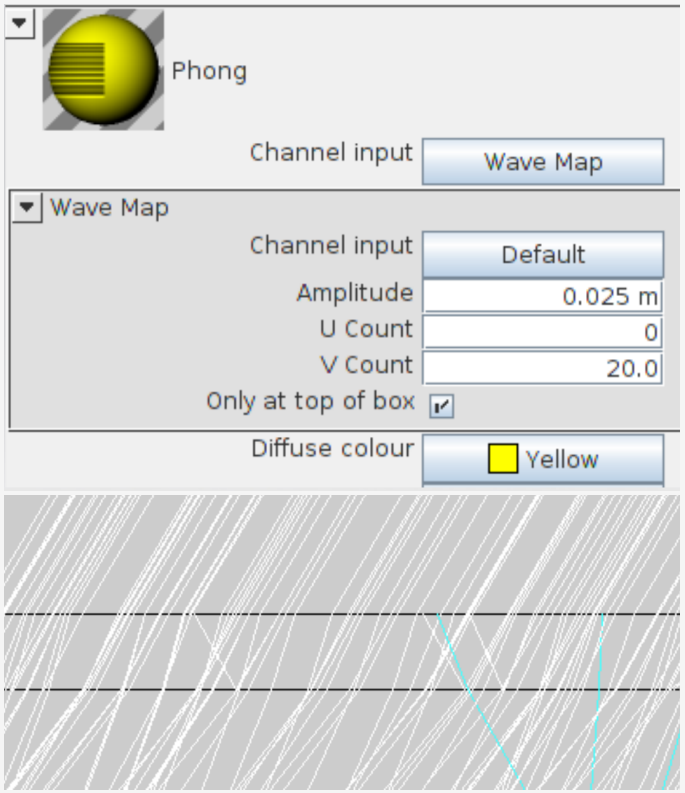




Application of refraction (CPU LM only)

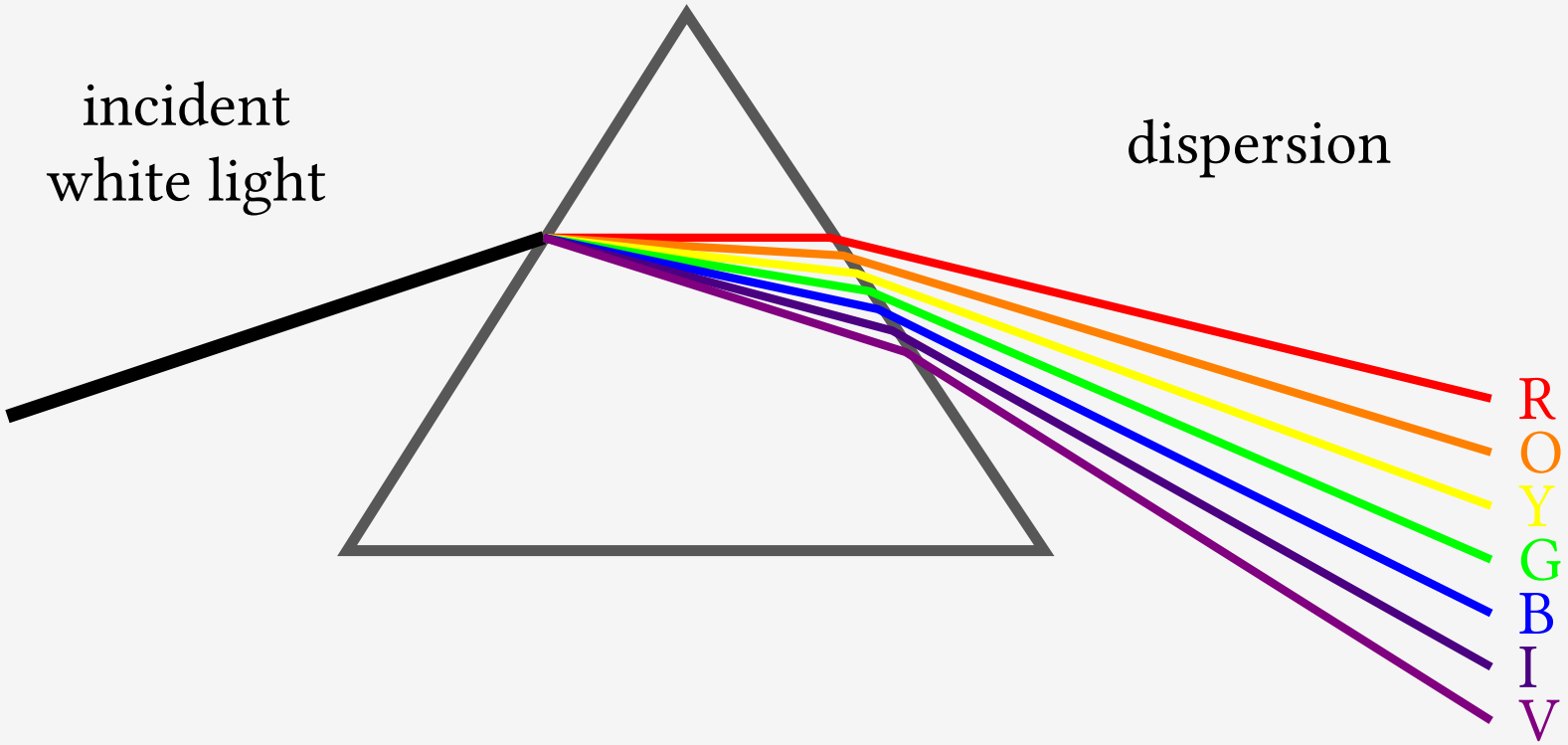
- Wave map: A special bump map for corrugated glass
- A special bump map is the wave map which can be applied easily to a geometry





Reflection with Dispersion

- variation of index of refraction with wavelengths
- Interior: Constant IOR → transmitted photons will refract to/from the normal based on the IOR



Phong: Colours and Transparency

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.setDiffuseTransparency(new Graytone (0.0));
7
8  //myShader.setTransparencyShininess(new Graytone (0.5));
9  //myShader.setTransparency(new Graytone (0.5));
10 //myShader.setAmbient(new Graytone (0.5));
11 //myShader.setEmissive(new Graytone (0.5));
```

Graytone(0.5) $\Longleftrightarrow$ *RGBColor(0.5, 0.5, 0.5)*

- weights are not automatically normalized
- If they add up to less then 100% all is well, the remainder is just absorbed.
- However, values greater than 100% will give false results

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));
```

red reflector

```
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));
```

semi-transparent

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

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));
```

total absorber

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

red reflector

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

Easy case: equal for the whole range!

Not preferable to use them when Flux LM is used!

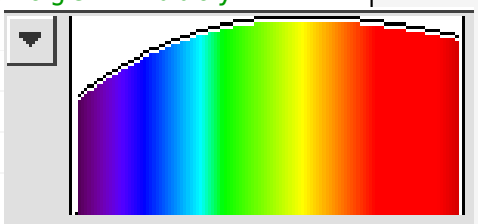
Spectral Shaders are needed!

Predefined SPDs

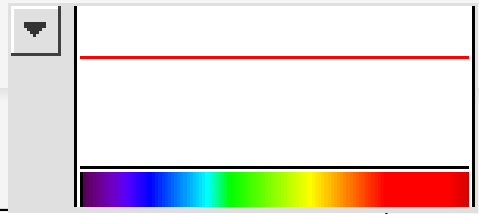
- BlackbodySpectralCurve
- ConstantSpectralCurve
- RGBSpectralCurve
- RegularSpectralCurve

LightNode: daylight

```
module MySun extends LightNode {
  {
    setLight(
        new SpectralLight(
        new PhysicalLight(
              DISTRIBUTION).(
              setVisualize(true)),
        new BlackbodySpectralCurve
              (5003), //[K]
        //5003K horizon light D50
        //6504K average midday
              light
        POWER
        ) );
  }
}
```



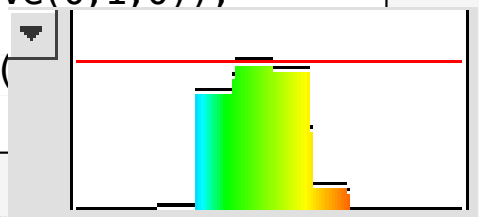
ConstantSPD



```
Phong s = new Phong();
  s.setDiffuse( new ChannelSPD(new
        ConstantSpectralCurve(0.2)));
```

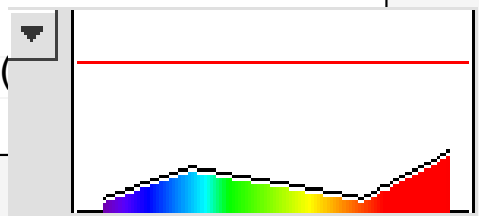
RGBSpectralCurve

```
ChannelSPD c = new ChannelSPD(new
      RGBSpectralCurve(0,1,0));

Phong s = new Phong(
  s.setDiffuse(c);
```



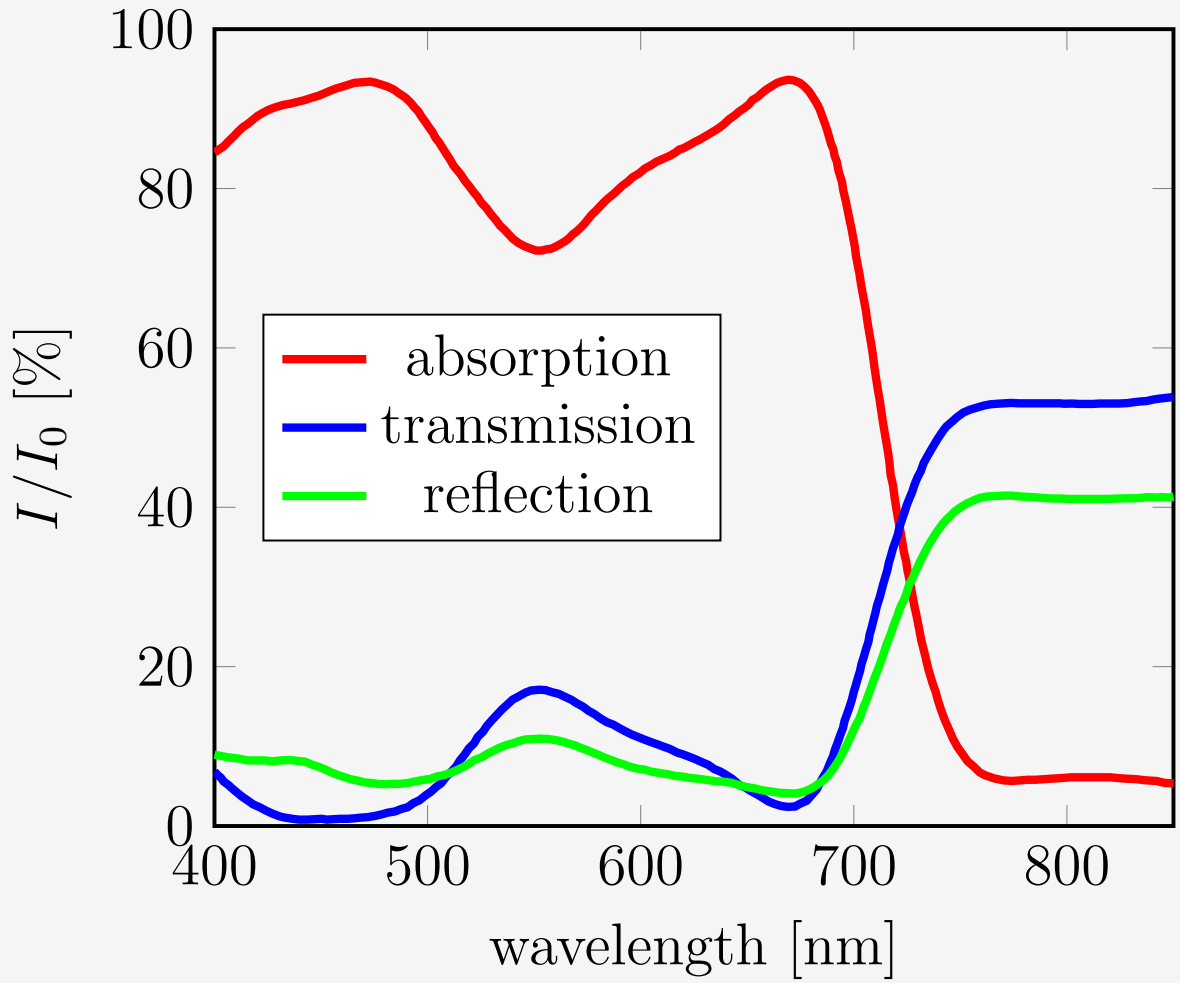
RegularSpectralCurve

```
ChannelSPD c = new ChannelSPD(new
      RegularSpectralCurve(new float
      [] {0.1, 0.3,0.2,0.1,0.4}, 400,
       700));

Phong s = new Phong(
  s.setDiffuse(c);
```



Absorption spectrum of leaf pigments

- typical soybean leaf



Kasperbauer, 1987

User defined SPDs

- IrregularSpectralCurve

WAVELENGTHS	350	360	370	380	...	850
AMPLITUDES	0.1	0.0	0.1	0.2	...	0.2

defined in GroIMP by simple arrays

```
1  public const float[] WAVELENGTHS = {350, 360, 370, 380, ..., 850};
2  public const float[] AMPLITUDES = {0.1, 0.0, 0.1, 0.2, ..., 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 }
```

User defined SPDs

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 }
```

Switch shader

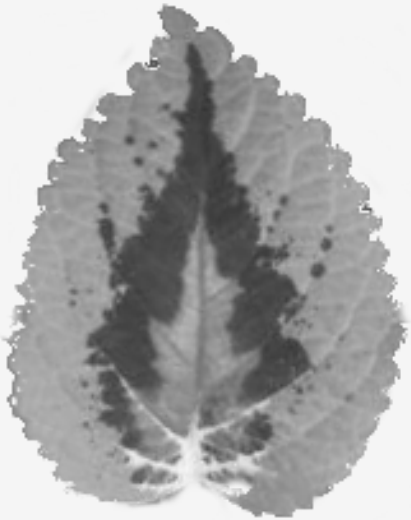
- SideSwitchShader: different shader for upper and lower side
- AlgorthmSwitchShader: different shader for light calculation and visualization

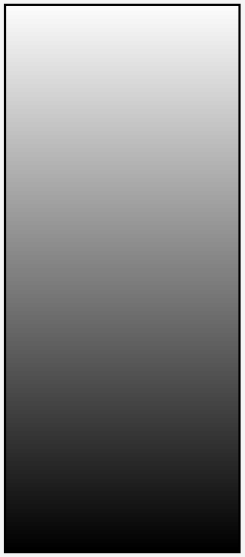
```
1  const Phong AdaxialS = new Phong();
2  static {
3      AdaxialShader.setSpecular(new Graytone(0.01)); // reflected specular
4      AdaxialShader.setDiffuse(new Graytone(0.05)); // reflected diffuse
5      AdaxialShader.setTransparency(new Graytone(0.04));  // transmitted
6  }
7
8  const Phong AbaxialS = 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 // distinctive upper (adaxial) and lower (abaxial) surfaces
16 const SideSwitchShader leafS = new SideSwitchShader(AdaxialS, AbaxialS);
17
18 //switch between gui (GREEN) and radiation (leafS) shader
19 const AlgorithmSwitchShader leafA = new AlgorithmSwitchShader(GREEN, leafS);
```

Special shader - Images

- image as texture → allows inhomogeneous area

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

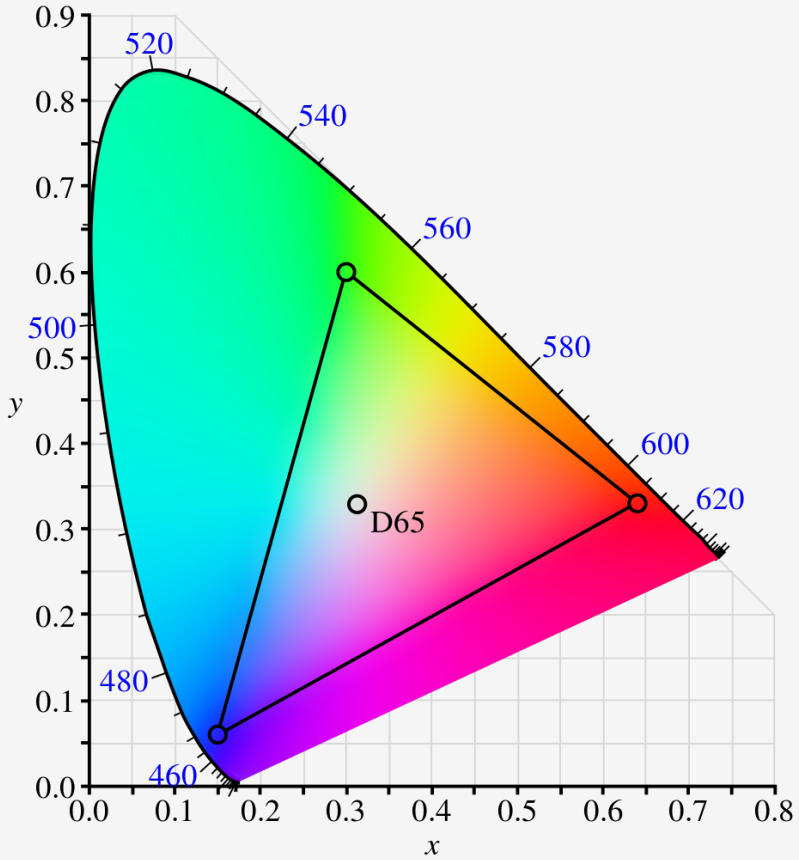




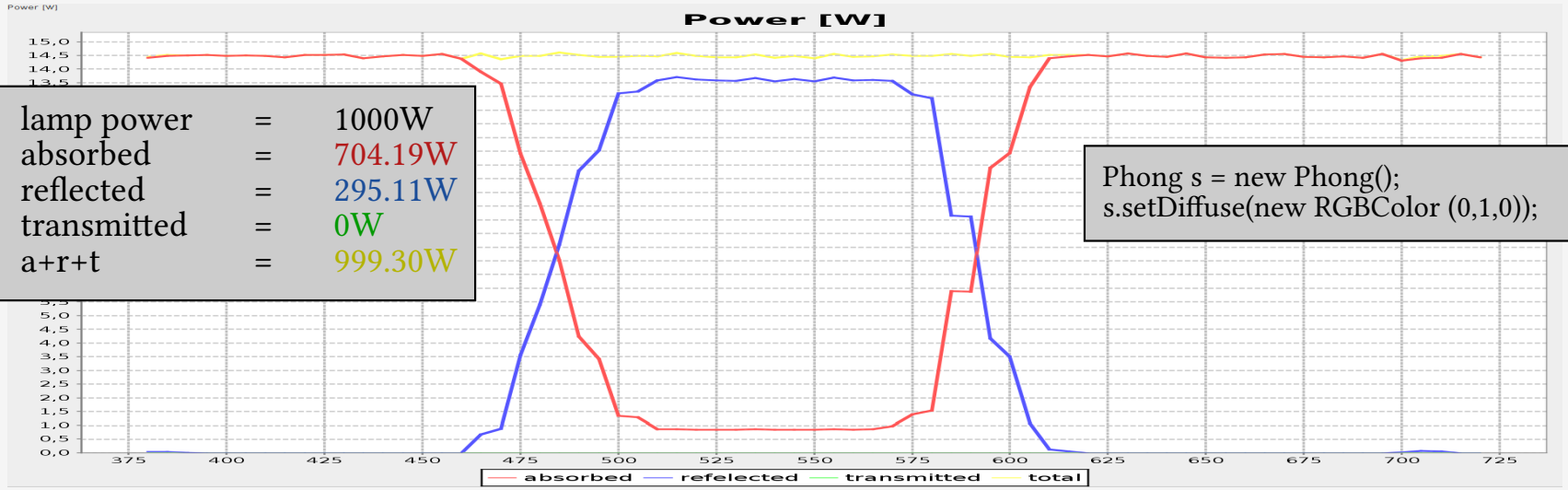
Colour spaces: sRGB vs. spectral

- colour spaces are not equal
- no one-to-one conversion
  - wavelength spectrum doesn't have all colours, such as pink and grey
  - multiple wavelengths can combine to give the same colour

sRGB colour space

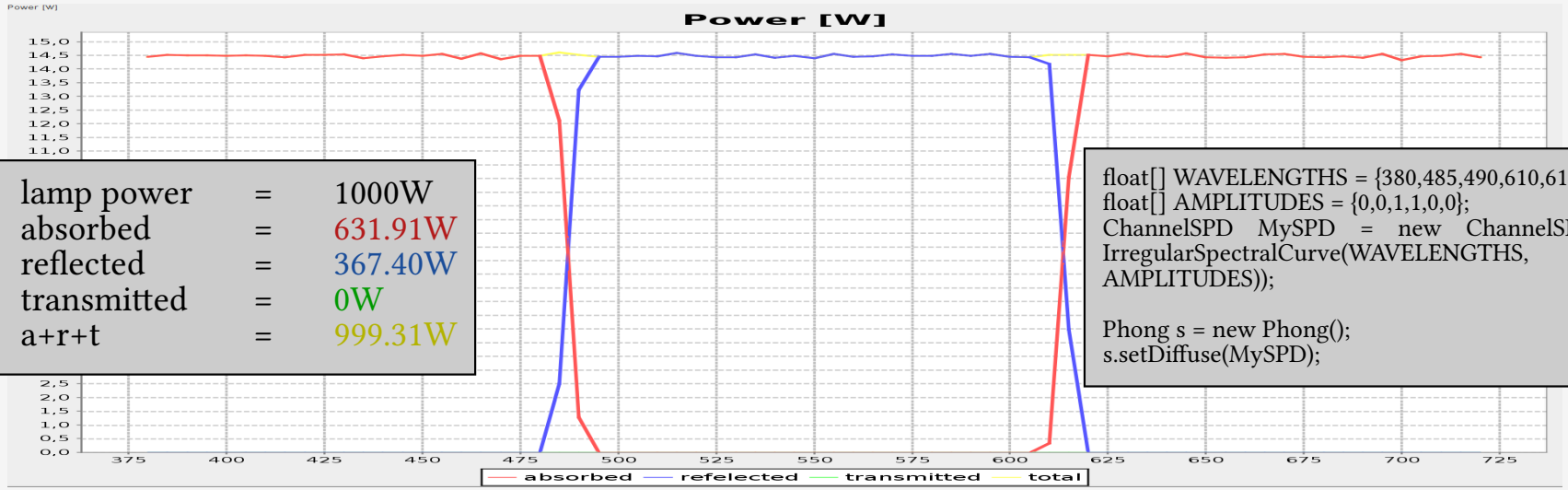


RGB vs. Spectral Shader



lamp power = 1000W
absorbed = 704.19W
reflected = 295.11W
transmitted = 0W
a+r+t = 999.30W

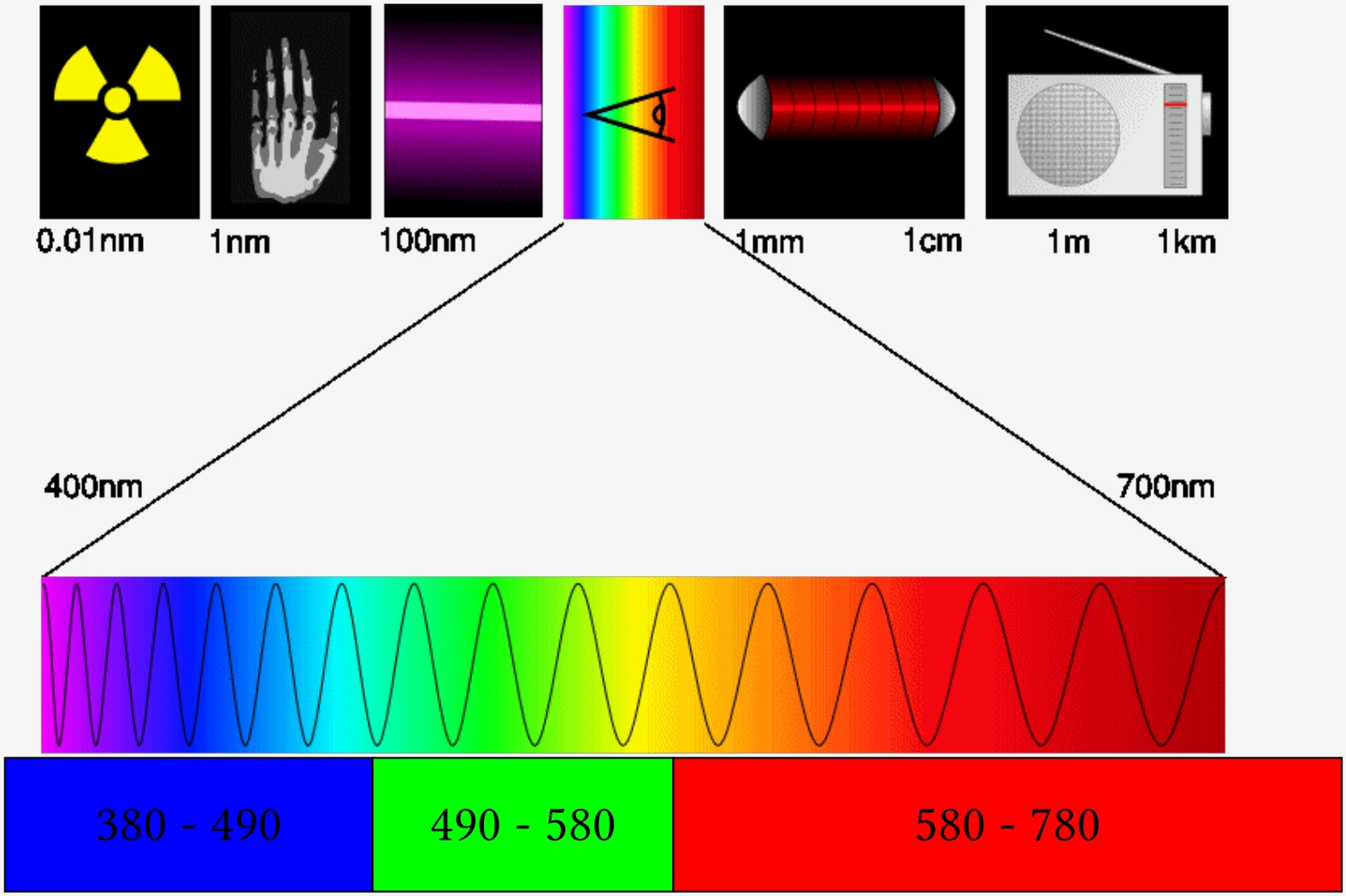
```
Phong s = new Phong();
s.setDiffuse(new RGBColor (0,1,0));
```



lamp power = 1000W
absorbed = 631.91W
reflected = 367.40W
transmitted = 0W
a+r+t = 999.31W

```
float[] WAVELENGTHS = {380,485,490,610,615,720};
float[] AMPLITUDES = {0,0,1,1,0,0};
ChannelSPD MySPD = new ChannelSPD(new
IrregularSpectralCurve(WAVELENGTHS,
AMPLITUDES));

Phong s = new Phong();
s.setDiffuse(MySPD);
```

Visible Spectrum 2 sRGB



Definition of Domains

- light source vs. shader vs. light model

  $\rightarrow$ should be defined over the same domain!

  e.g., different domain on LM will lead to different results (constant light and shader): $LM.setSpectralDomain(MIN_W, MAX_W)$;

domain LM = 420-680nm
absorbed = 518.98W
reflected = 480.53W

domain LM = 380-720nm
absorbed = 631.89W
reflected = 367.40W

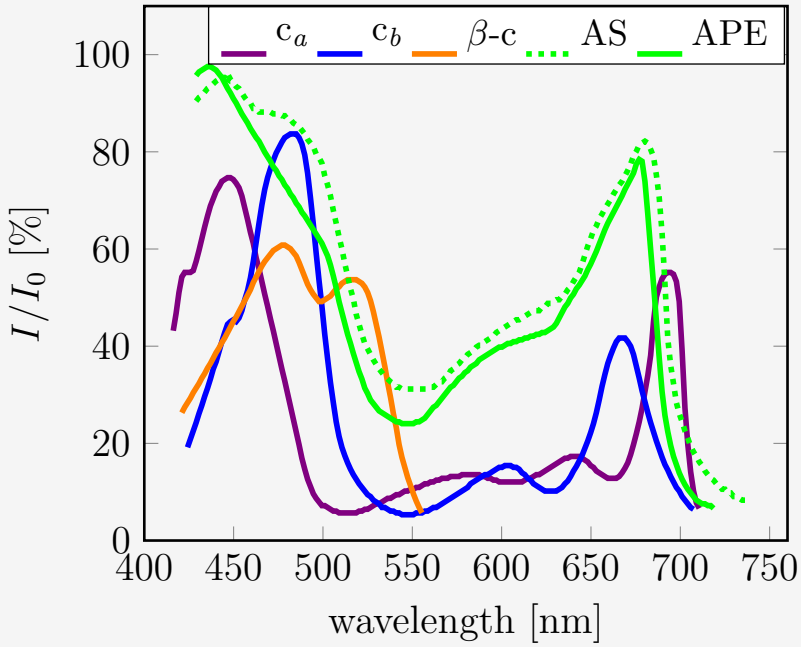
Note: Do not deviate too much from visual ligh! When the wavelength deviates too much from visual light, ray optics is no longer the most adequate tool for describing flow of electromagnetic radiation.

Spectral photosynthesis calculation I

- Light models use Watt as unit
- conversion from the absorbed spectrum to APE is complicated
- PAR is species specific (depending on the content and configuration of photochemically active pigments)

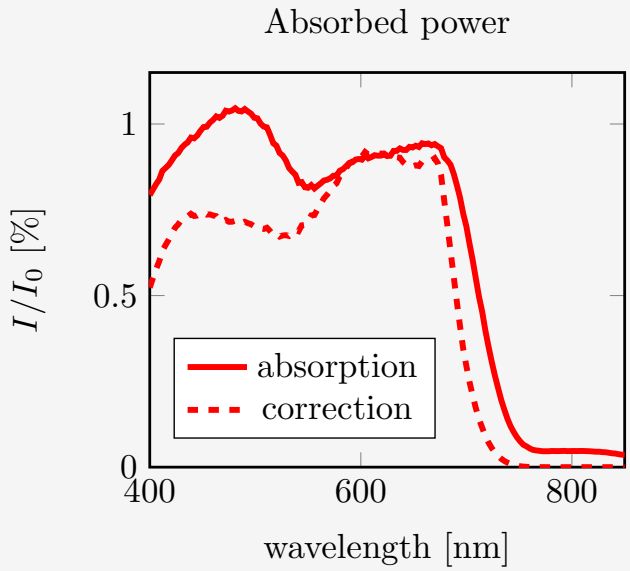
  → McCree Curve

  → McCree - correction!



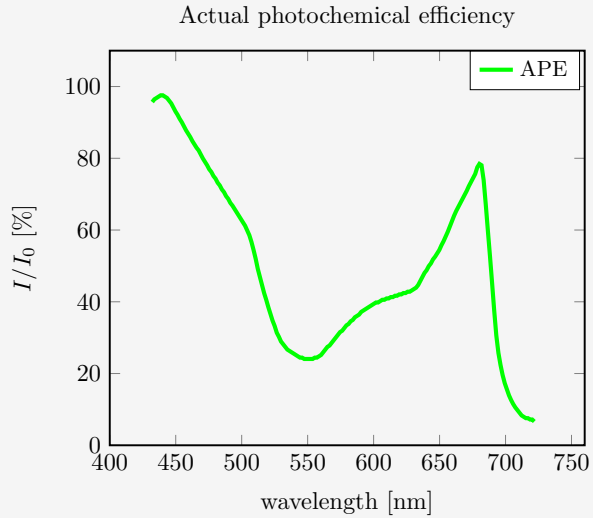
c_a = chlorophyll a
c_b = chlorophyll b
β-c = β-carotene
AS = absorption spectrum = relative light absorption
APE = actual photochemical efficiency = action spectrum

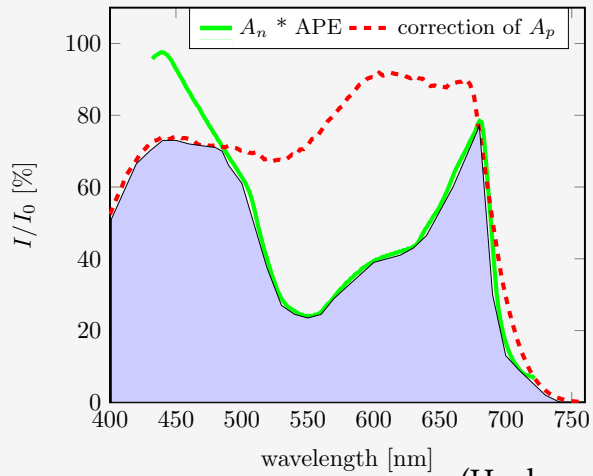
Spectral photosynthesis calculation II



$$\text{YPF} = \int \text{McCreeCorrected}(A_p)$$

$$A_n = \text{Thornley}(\text{YPF}, \mathit{age}, \mathit{temp})$$



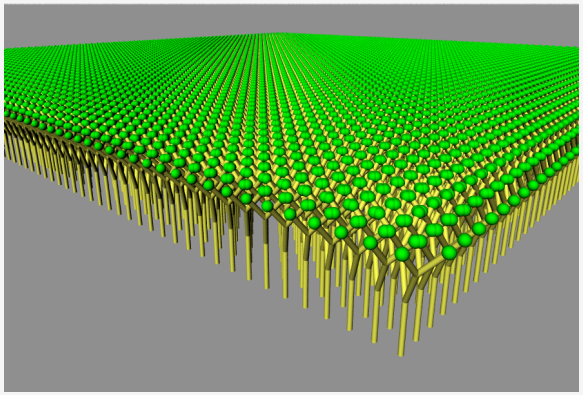


(Henke and Buck-Sorlin 2017)

Support for infinite canopies

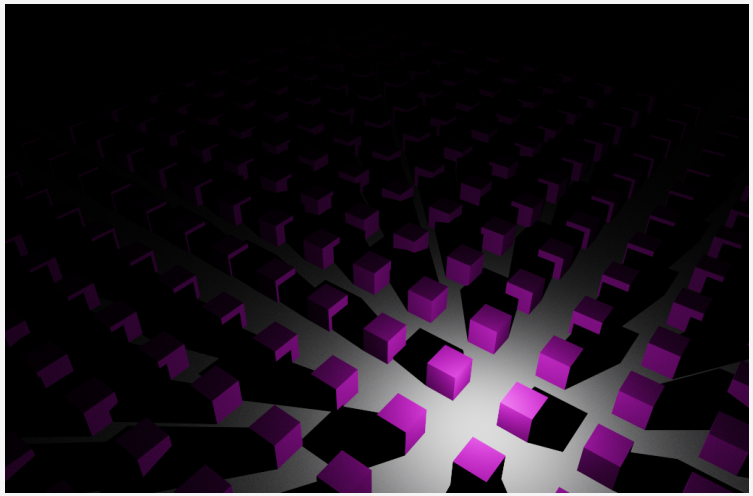
- *GridClonerNode(xCount, xDistance, yCount, yDistance)*
- In the scene graph there are only the original plants plus the GridClonerNode
- For the light model and the raytracer, the geometry is repeated in a grid-like manner.
- This doesn't cost memory, but of course it makes the intersection computation more demanding.
- Light absorption from the clones is recorded at the originals.

```
protected void init () [
  Axiom ==> GridClonerNode(25, 1.5, 25, 1.5)
    for (int x : 0 : 2) //three times in x direction
      for (int y : 0 : 2) ( //and three times in y
        [ Translate(x*0.5, y*0.5, 0) A(1) ]
      );
]
```

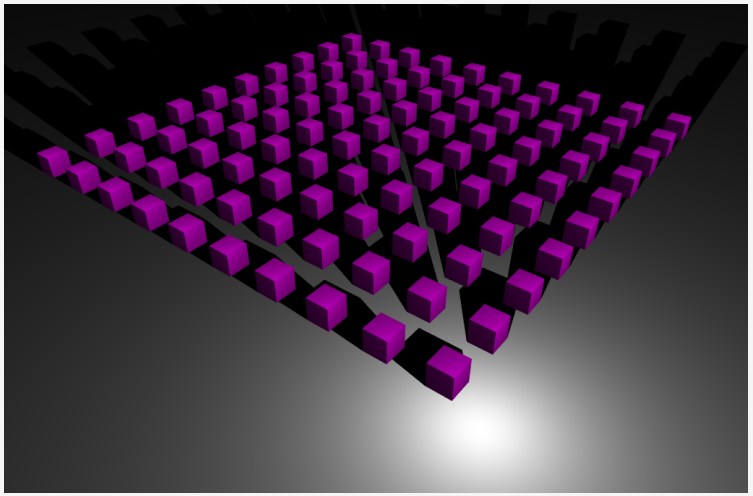


Differences in interpretation of infinite canopies by light models and renderers

- CPU and GPU renderer and light model interpret the *GrinClonerNode* differently
  - CPU (Twilight) implementation: repeats objects only into first quadrant
  - GPU (Flux) implementation: repeats objects into all fout quadrants



Flux renderer



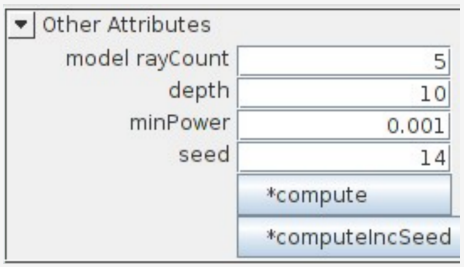
Twilight renderer

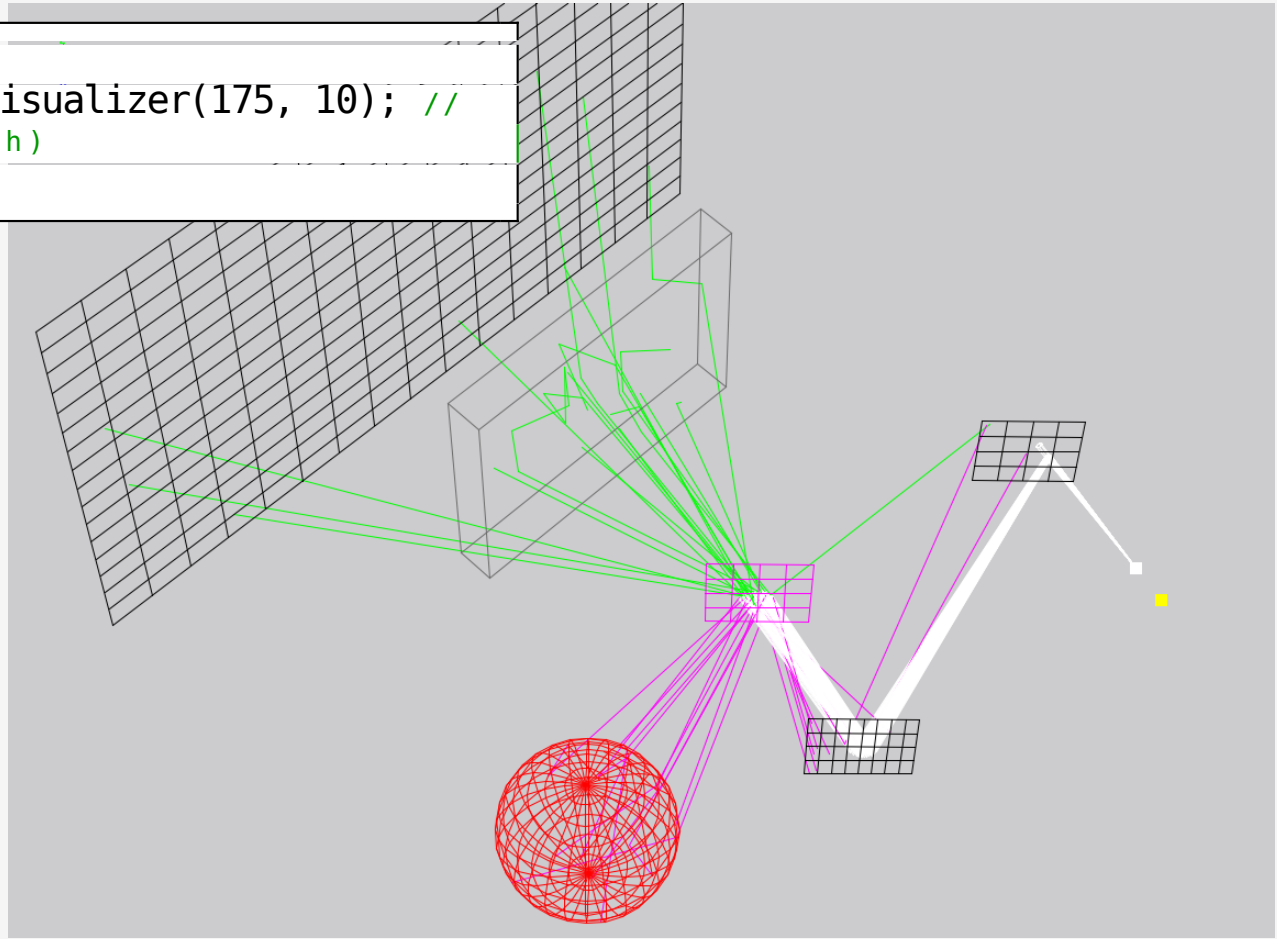
Table of Contents

Light visualization for CPU LM

- Visualization of light rays

```
1 protected void init () [
2     Axiom ==> LightModelVisualizer(175, 10); //
          (ray count, depth)
3 ]
```

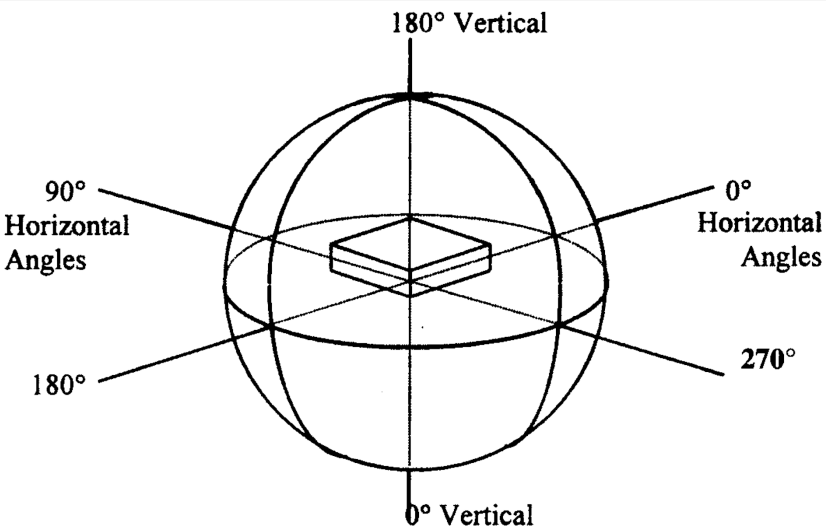


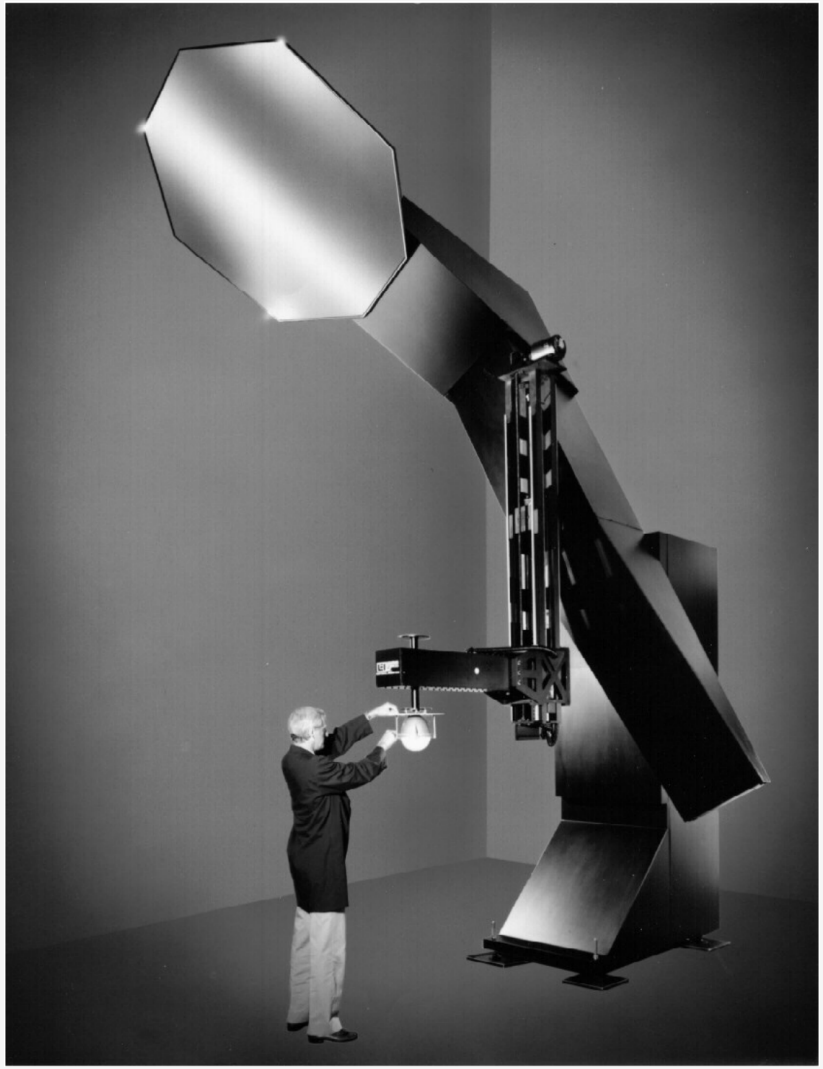


Note: Only rays that hit an object are visualized!

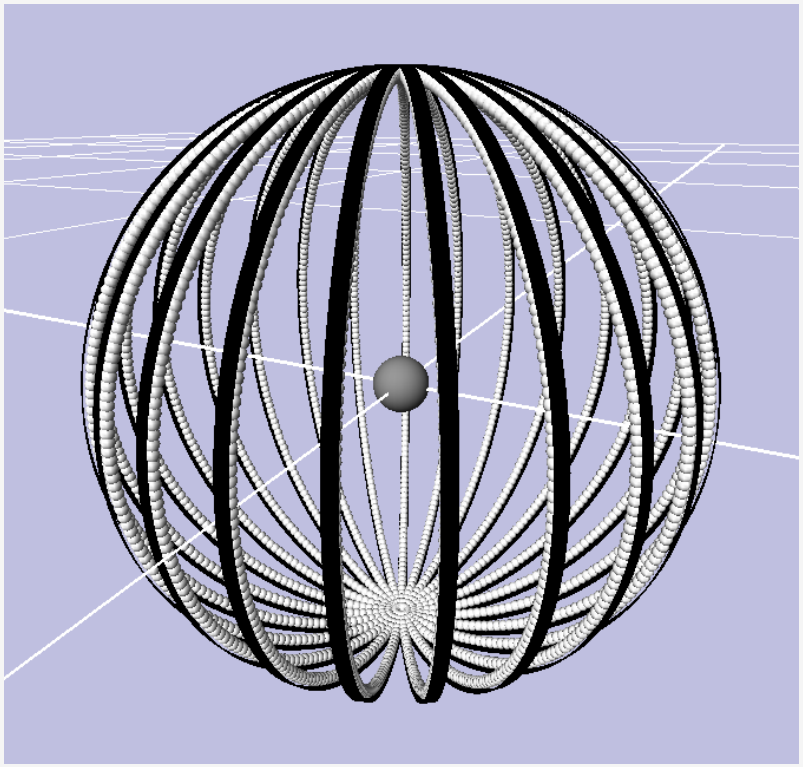
Lamp Test Environment

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





Lamp Test Environment



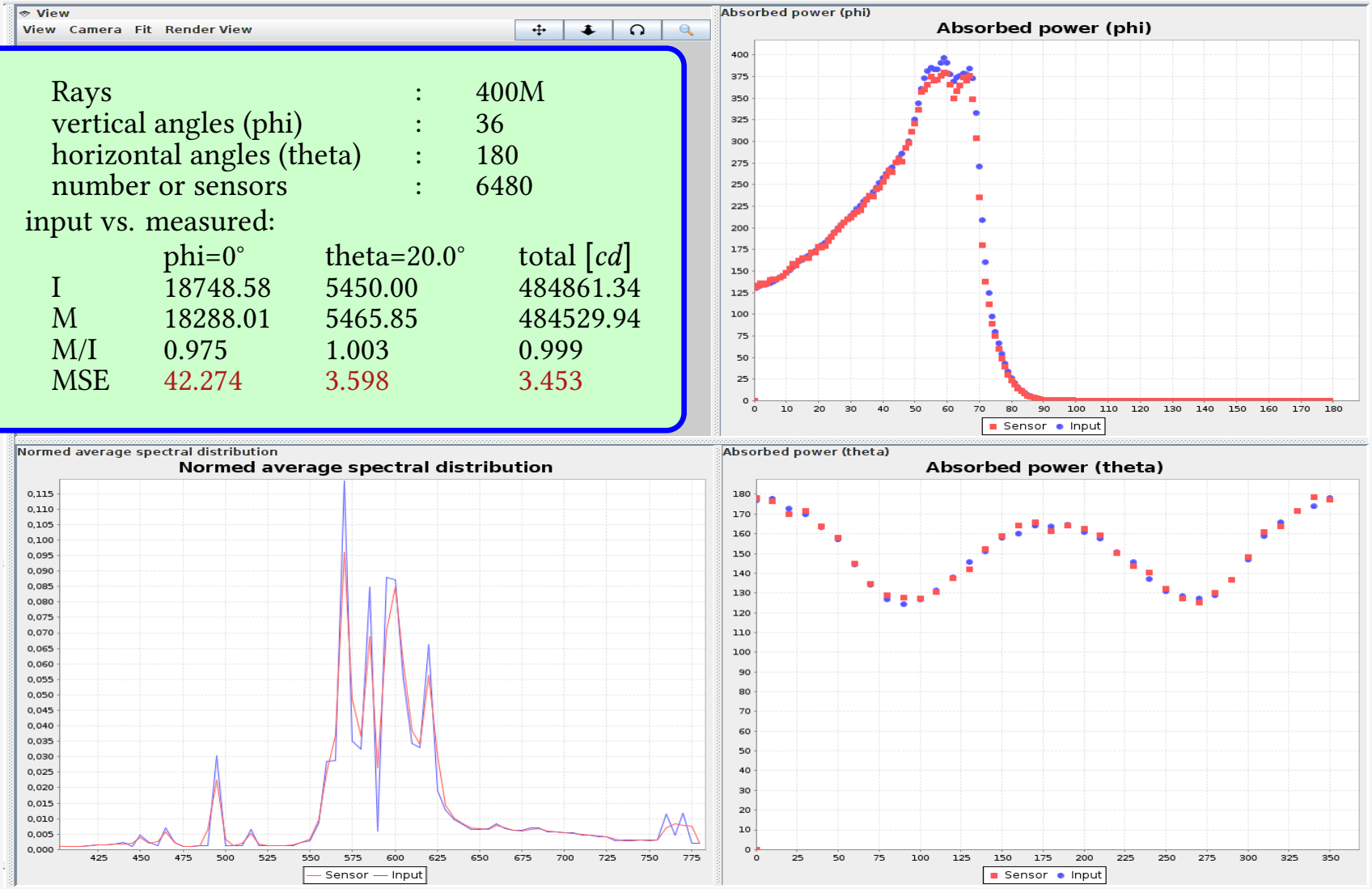
- model: *LampTestEnvironment.gsz*
- input: ies + spd file
- setup
  - light source in the centre of the scene
  - spherical net of sensors
  - spectral resolution: freely definable, e.g. 380:5:780 *nm*
- output: charts and datasets of distributions
- no additional conversions are applied

Lamp Tests: SONT

Rays	:	400M
vertical angles (phi)	:	36
horizontal angles (theta)	:	180
number or sensors	:	6480

input vs. measured:

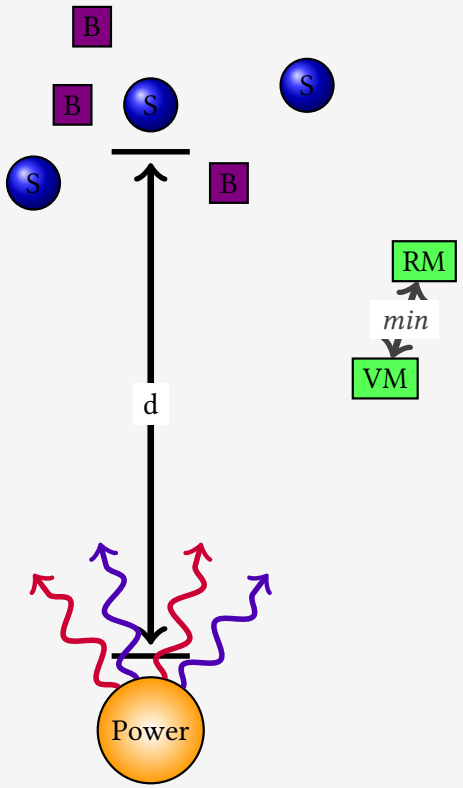
	phi=0°	theta=20.0°	total [cd]
I	18748.58	5450.00	484861.34
M	18288.01	5465.85	484529.94
M/I	0.975	1.003	0.999
MSE	42.274	3.598	3.453



Output Power Value

Output power: influential factor → determines a proper value

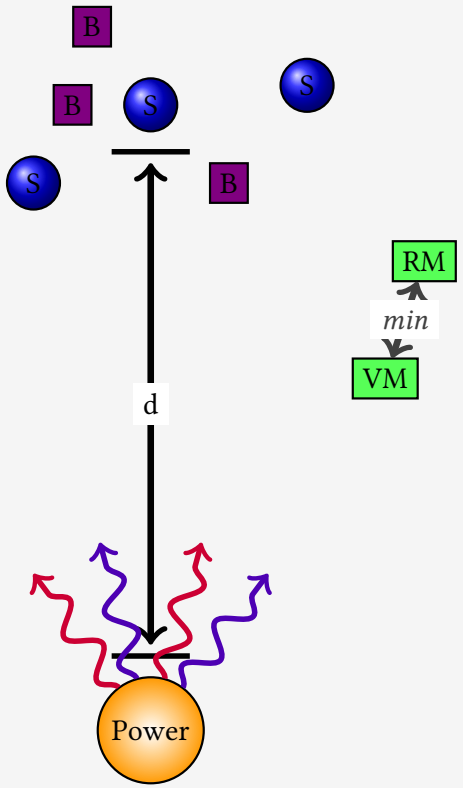
- power of the light node
- distance between light source and object
- used spectral power distribution
- optical properties of the object
- position and distribution of sensors (position of real measurement equals measurement point in the simulation)



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)



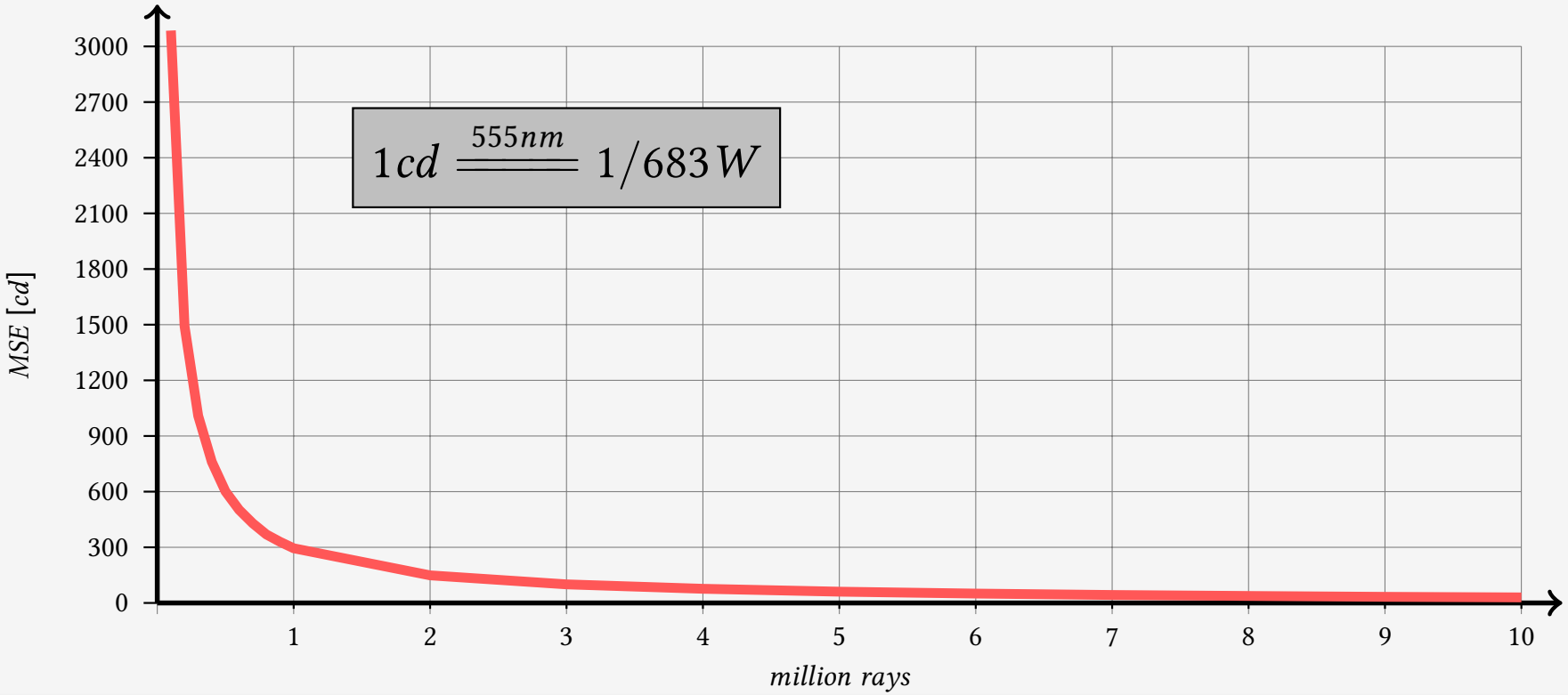
Consequences:

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

LED - Ray Statistic

Model accuracy is a function of number of rays used

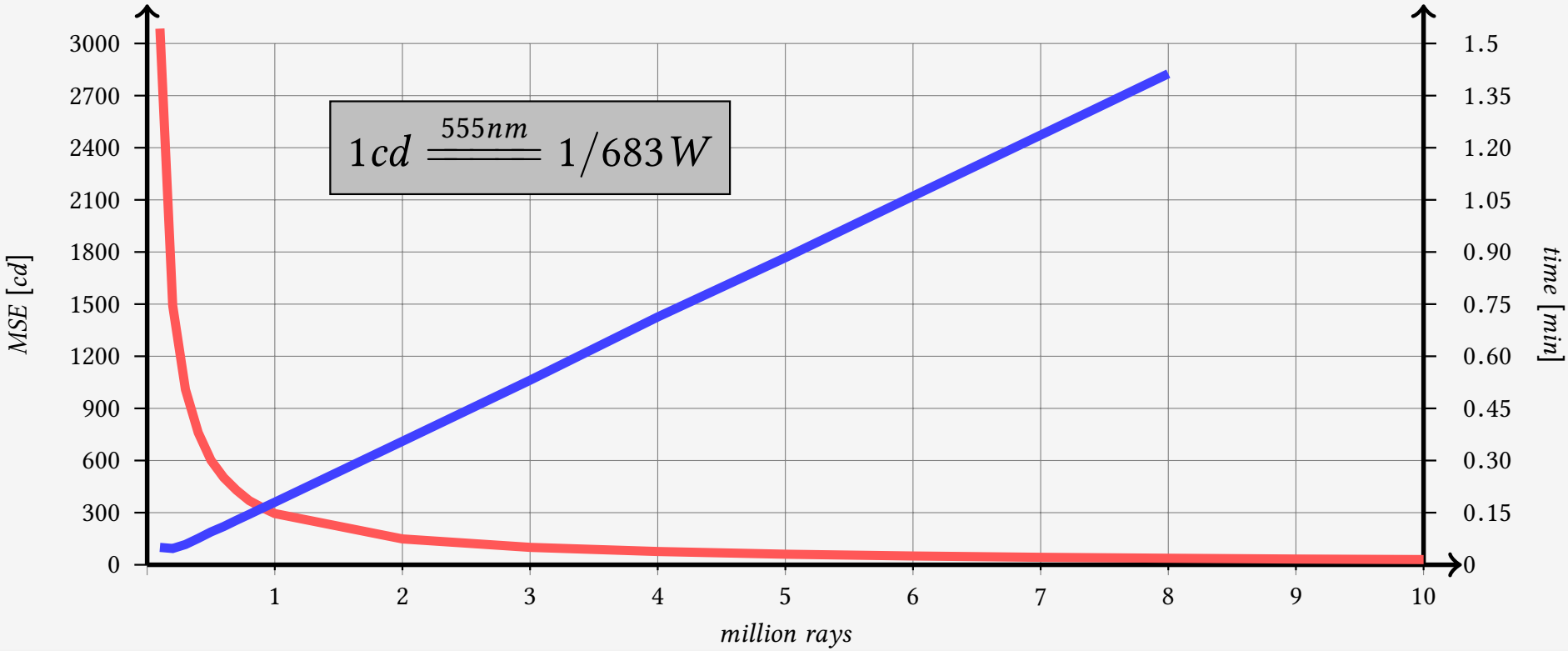
- aim: realistic physical distribution with minimal computational investment
- realistic → minimal mean square error (MSE)



LED - Ray Statistic

Model accuracy is a function of number of rays used

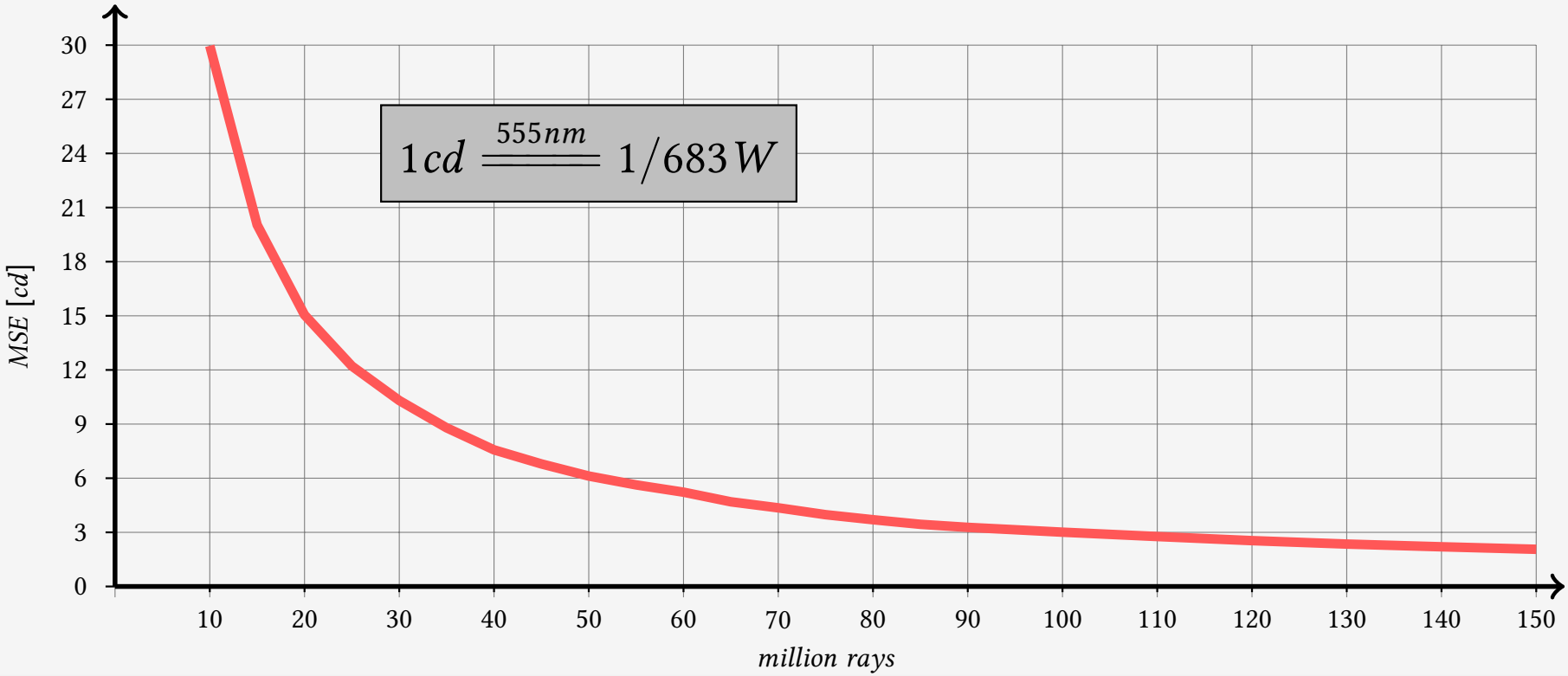
- aim: realistic physical distribution with minimal computational investment
- realistic → minimal mean square error (MSE)



LED - Ray Statistic

Model accuracy is a function of number of rays used

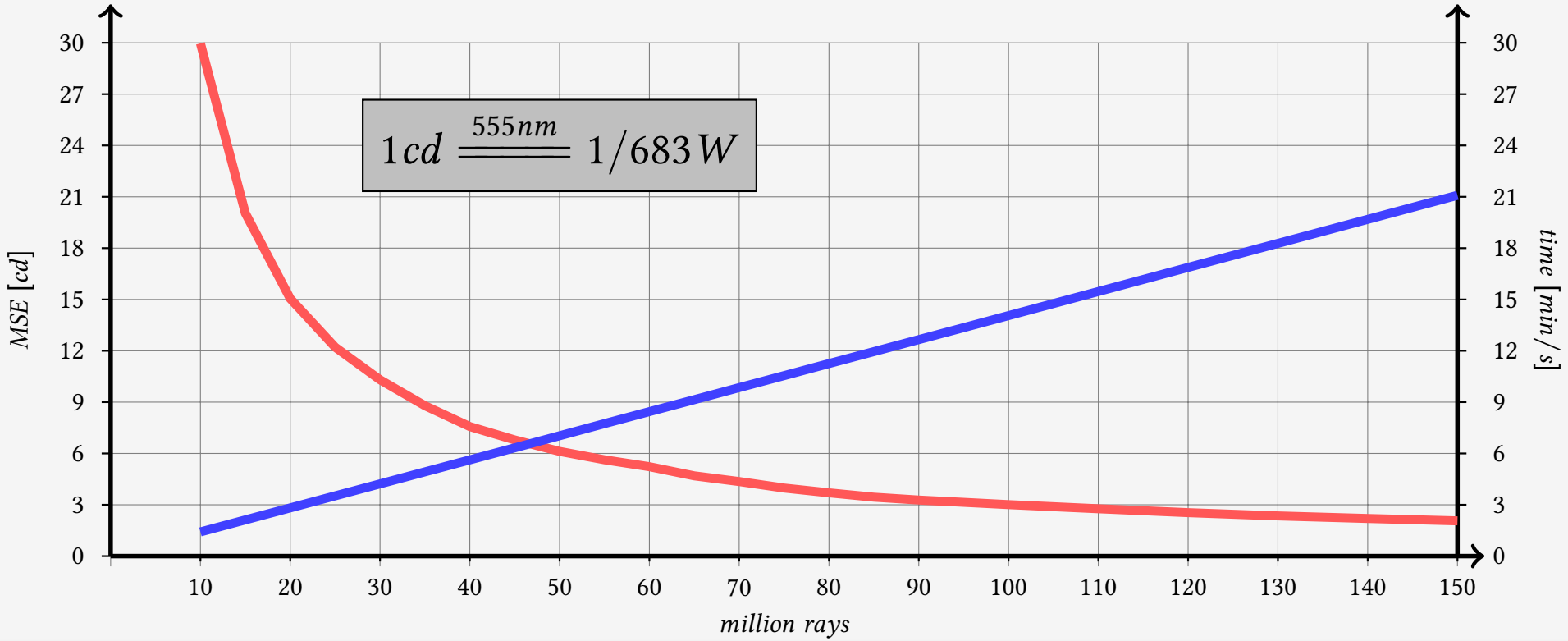
- aim: realistic physical distribution with minimal computational investment
- realistic → minimal mean square error (MSE)



LED - Ray Statistic

Model accuracy is a function of number of rays used

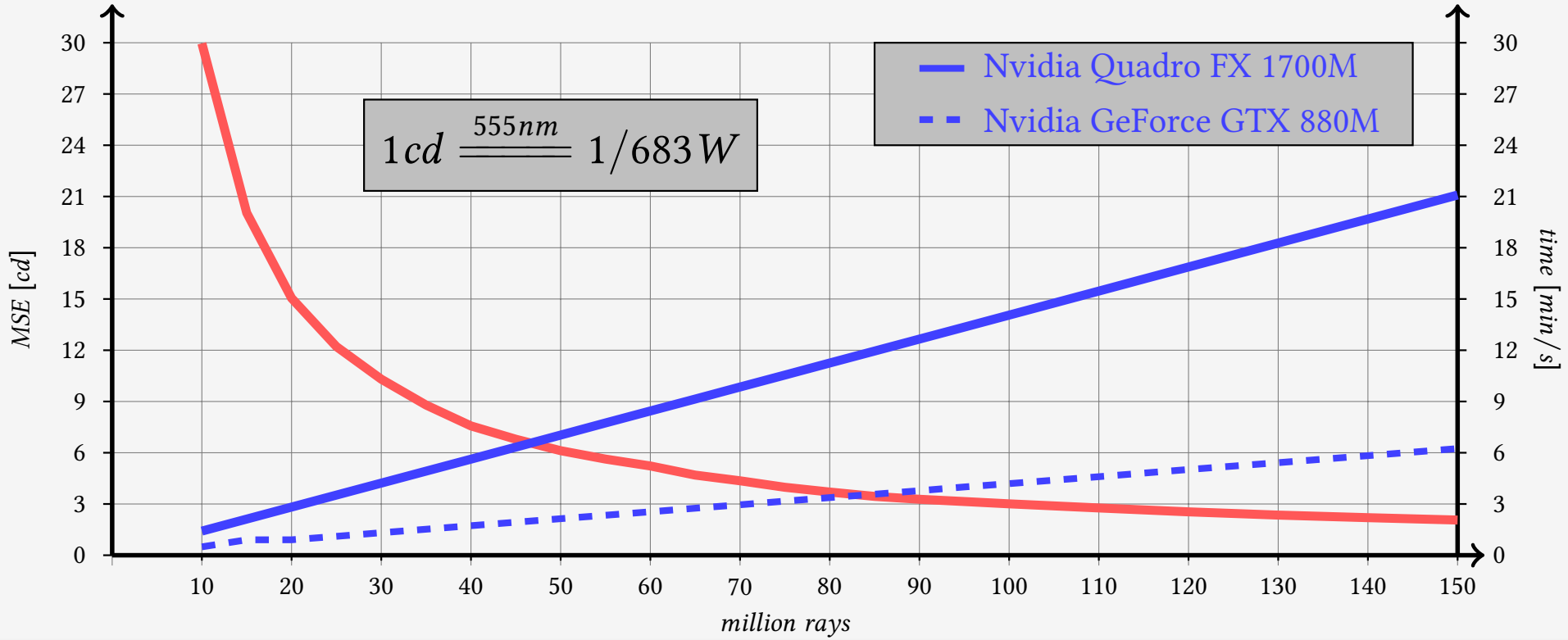
- aim: realistic physical distribution with minimal computational investment
- realistic → minimal mean square error (MSE)



LED - Ray Statistic

Model accuracy is a function of number of rays used

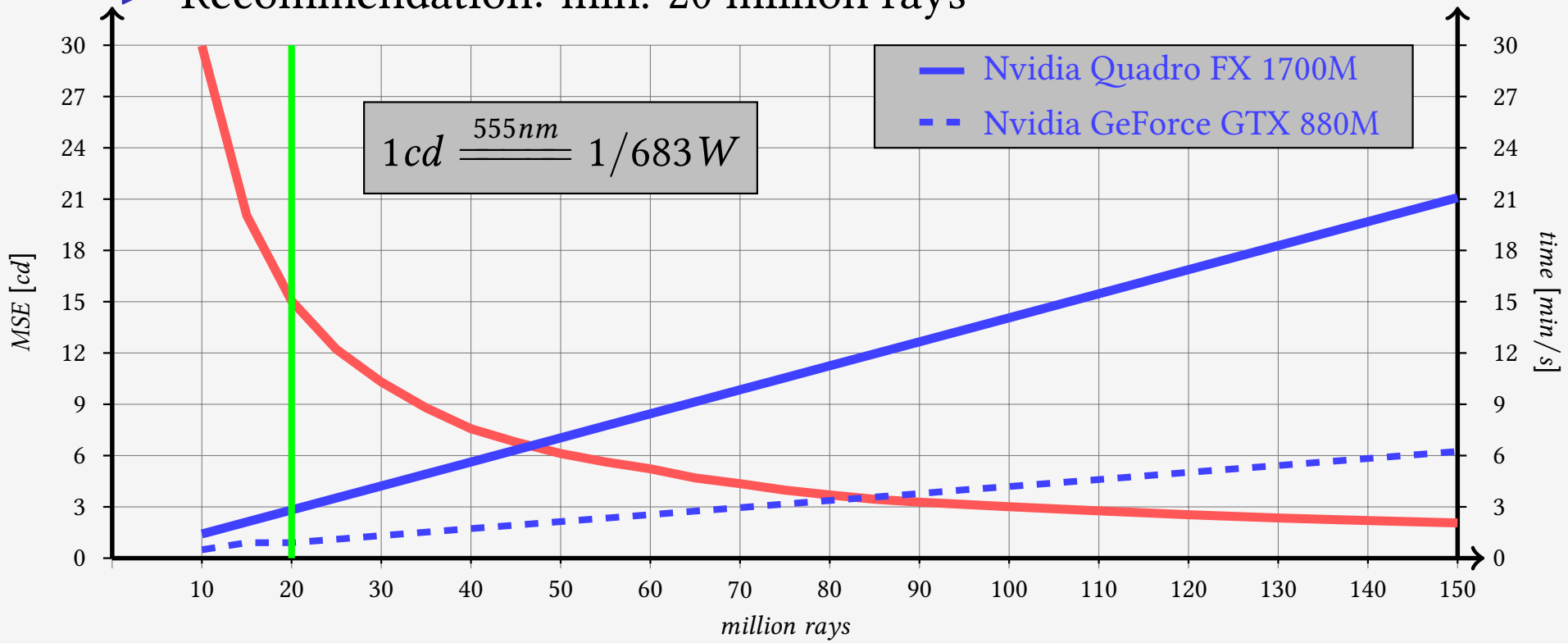
- aim: realistic physical distribution with minimal computational investment
- realistic → minimal mean square error (MSE)



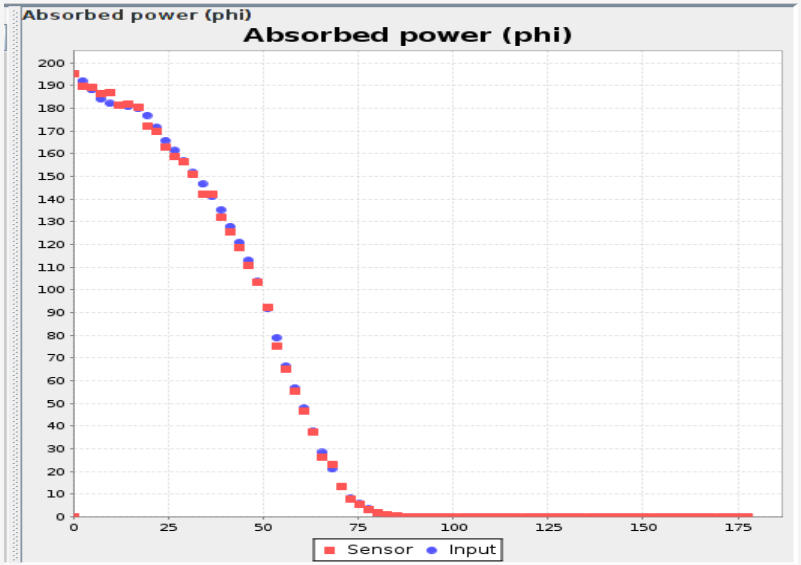
LED - Ray Statistic

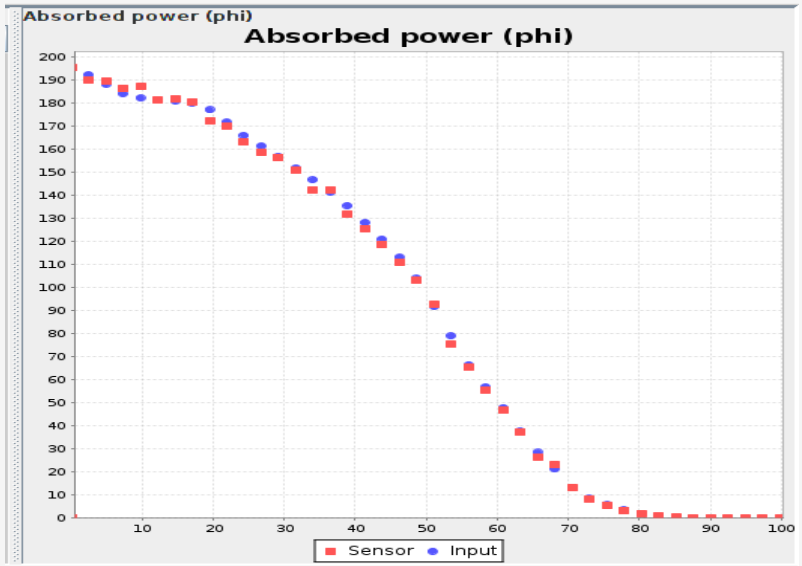
Model accuracy is a function of number of rays used

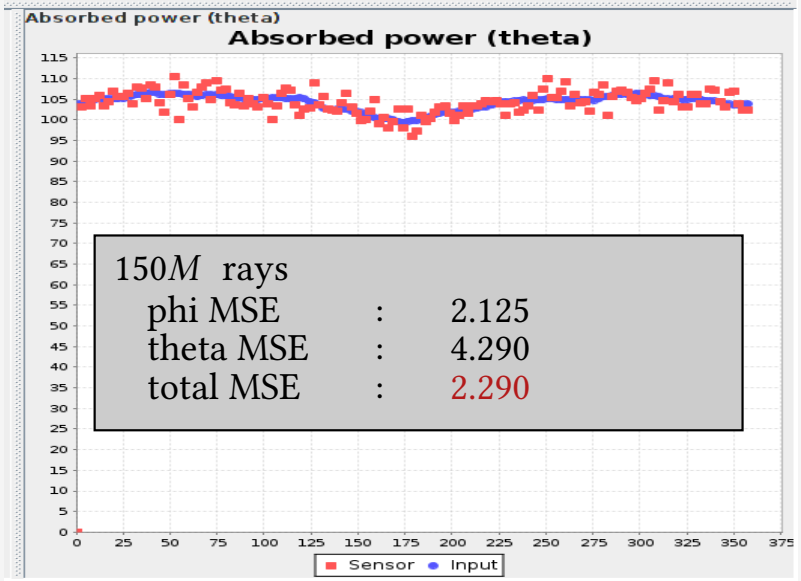
- aim: realistic physical distribution with minimal computational investment
- realistic → minimal mean square error (MSE)
- Recommendation: min. 20 million rays



SONT - Ray Statistic: 150M rays

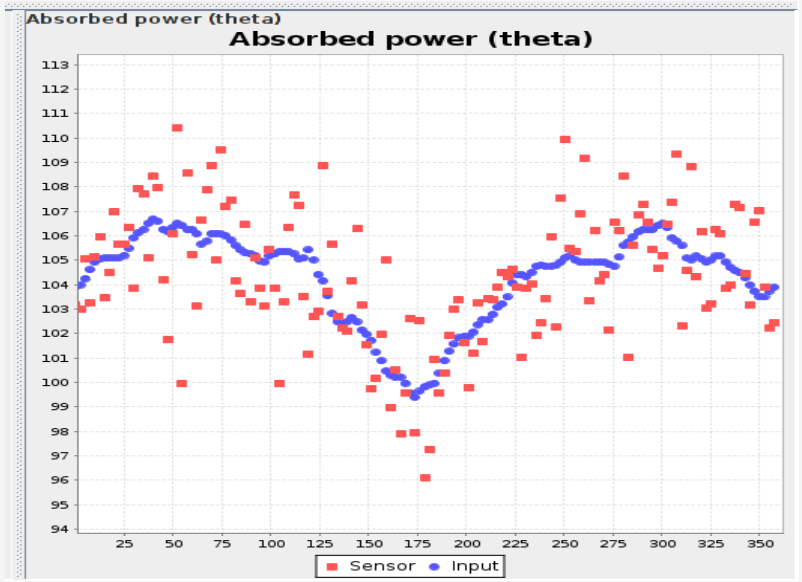






150M rays

phi MSE	:	2.125
theta MSE	:	4.290
total MSE	:	2.290



Reproducibility - I

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

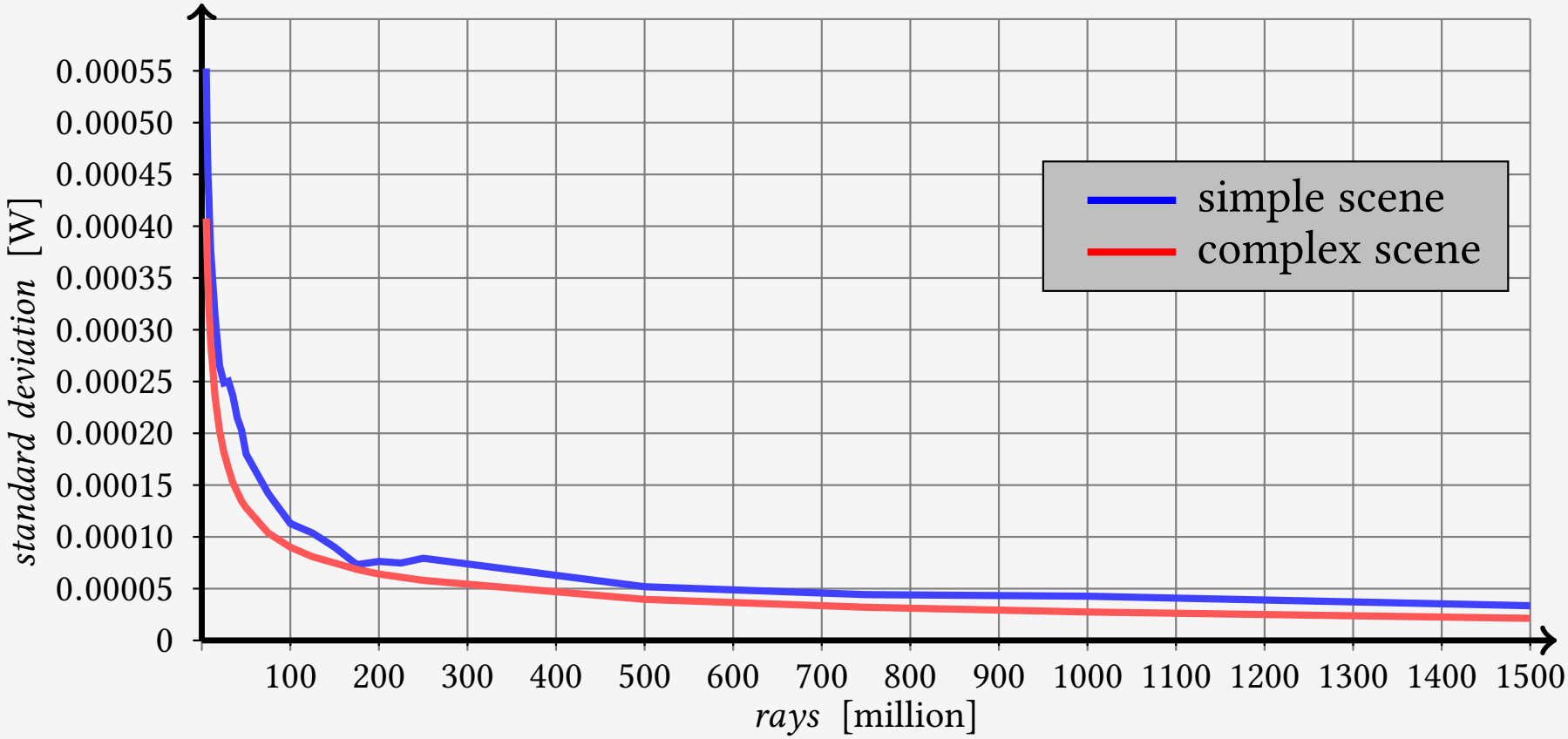
```
fixed scene
for an increasing number of rays do
    for an number of repetitions do
        run the LM with a new random seed
        obtain measurements for all objects
    end
    calculate the mean of all standard deviations
end
```

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

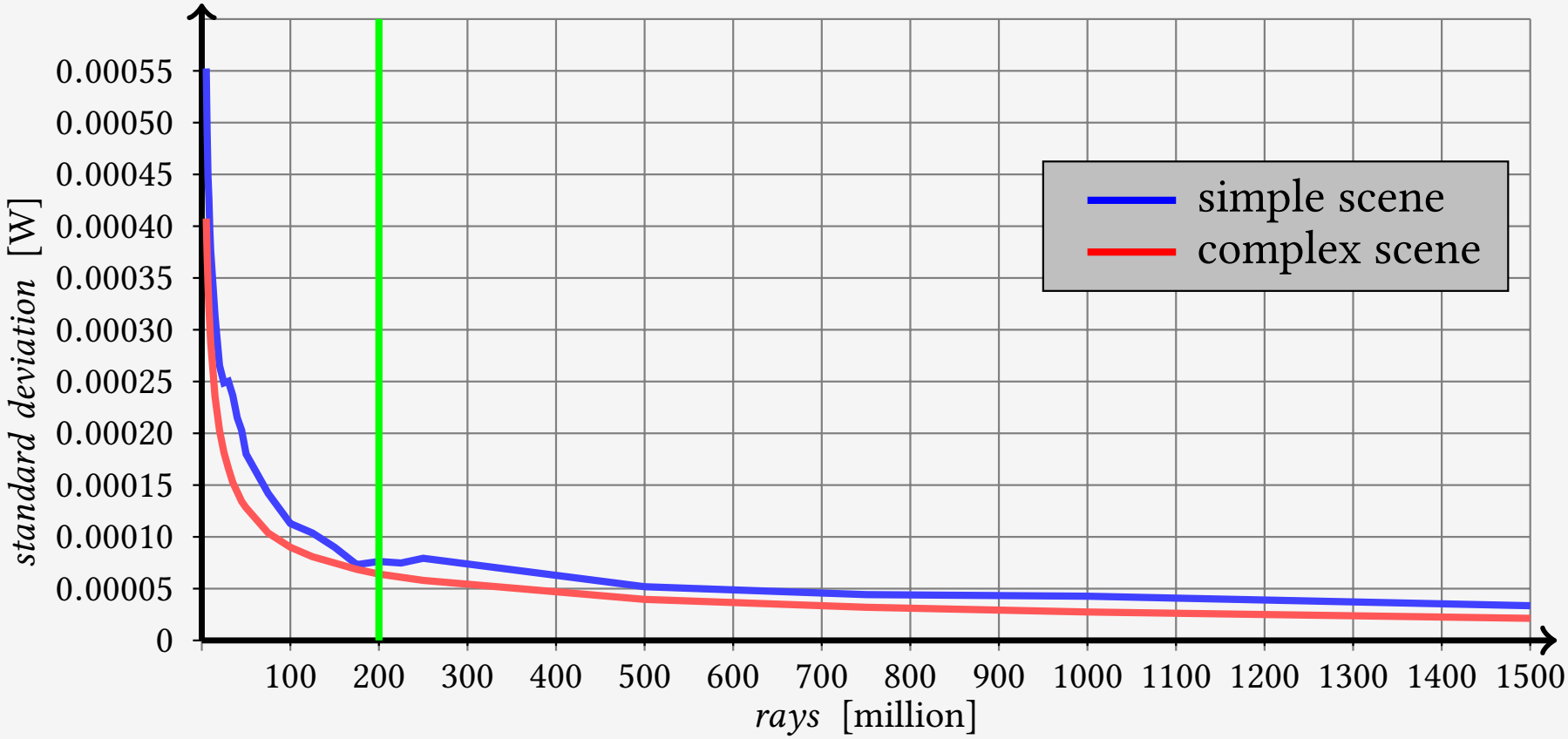
- mean standard deviation of 25 repeated light model runs
- increasing number of rays (five million to 1.5 billion rays)



(Henke and Buck-Sorlin 2017)

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 2017)

Ray Statistic - Influencing Factors

limitation factors $\rightarrow$ 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)

$\Longrightarrow$ proportional to increasing complexity of the scene
(...use as many rays as possible)

Sensor Object

- sensing light without interaction
- colour determines which wavelength should be observed; default: white ($\rightarrow$ all)

```
1  Axiom ==> SensorNode().(setRadius(0.005), setColor(1, 0, 0));
2
3
4  x:SensorNode ::> {
5    Measurement spectrum = lm.getSensedIrradianceMeasurement(x);
6    float absorbedPower = spectrum.integrate();
7    ...
8  }
```

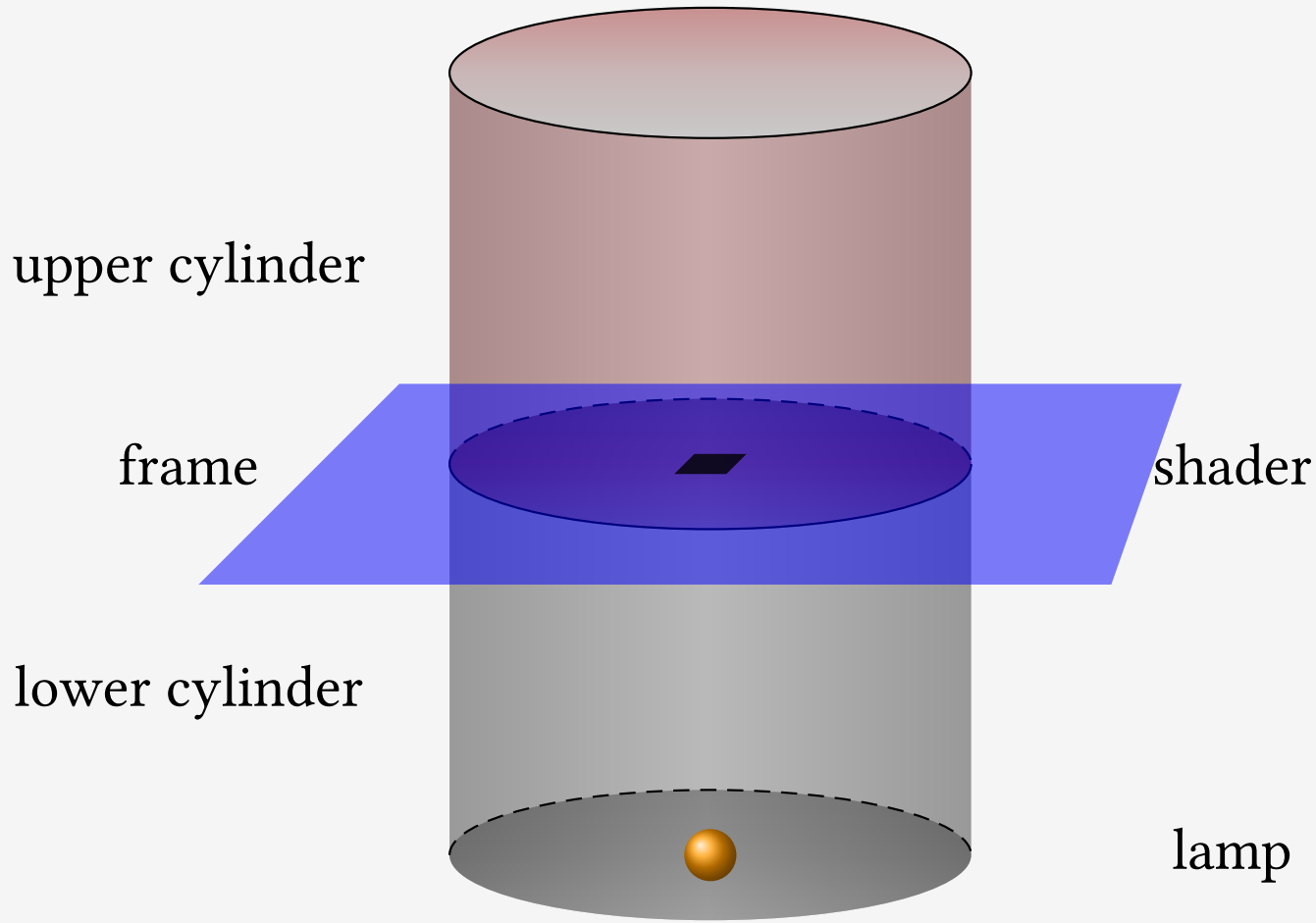
- can be en/disabled with *lm.setEnableSensors(true/false)*; default: enabled

Sensor vs. Primitive Object

Differences in usage (when used as sensors)

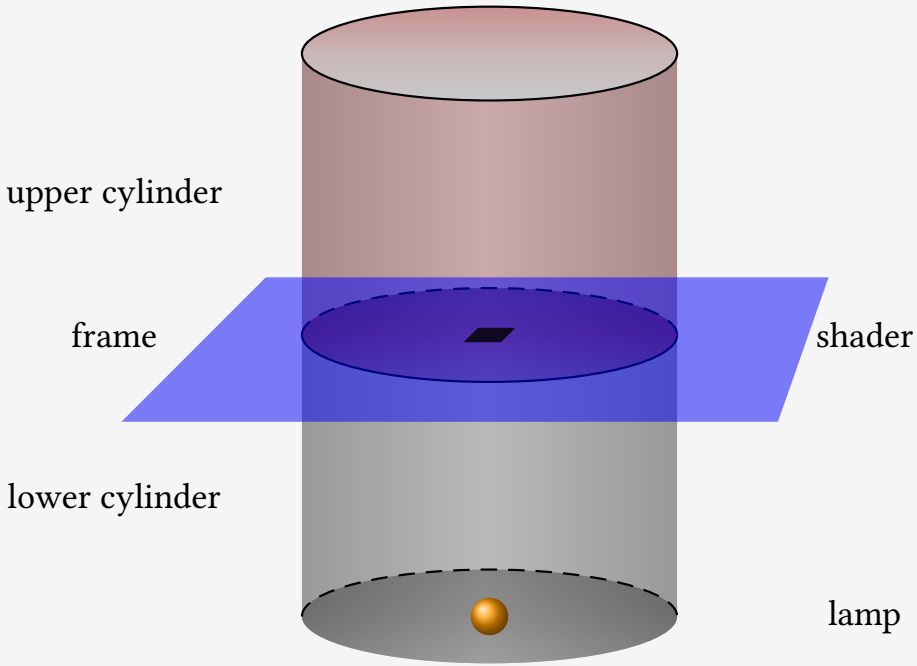
Property	Sensor	Primitive Object
colour	white	black
transparency	–	has to be turned off
absorbed power returned	square meter	surface area
differences	100%	higher
overlapping	not problematic	problematic
(↬ charts)		
obtain spectrum	*getSensedIrradianceMeasurement*	*getAbsorbedPowerMeasurement*

Shader Test Environment

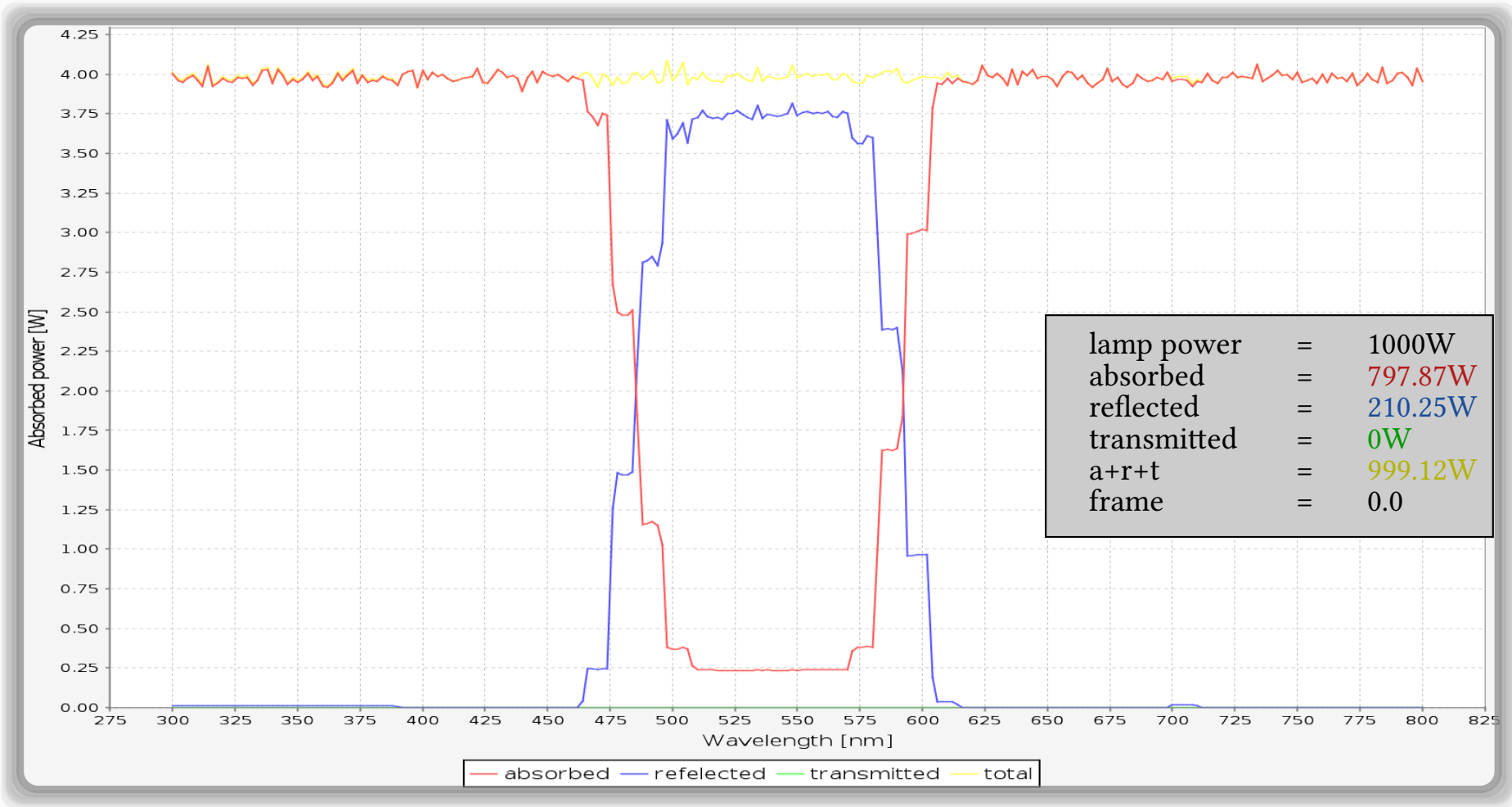


Shader Test Environment

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

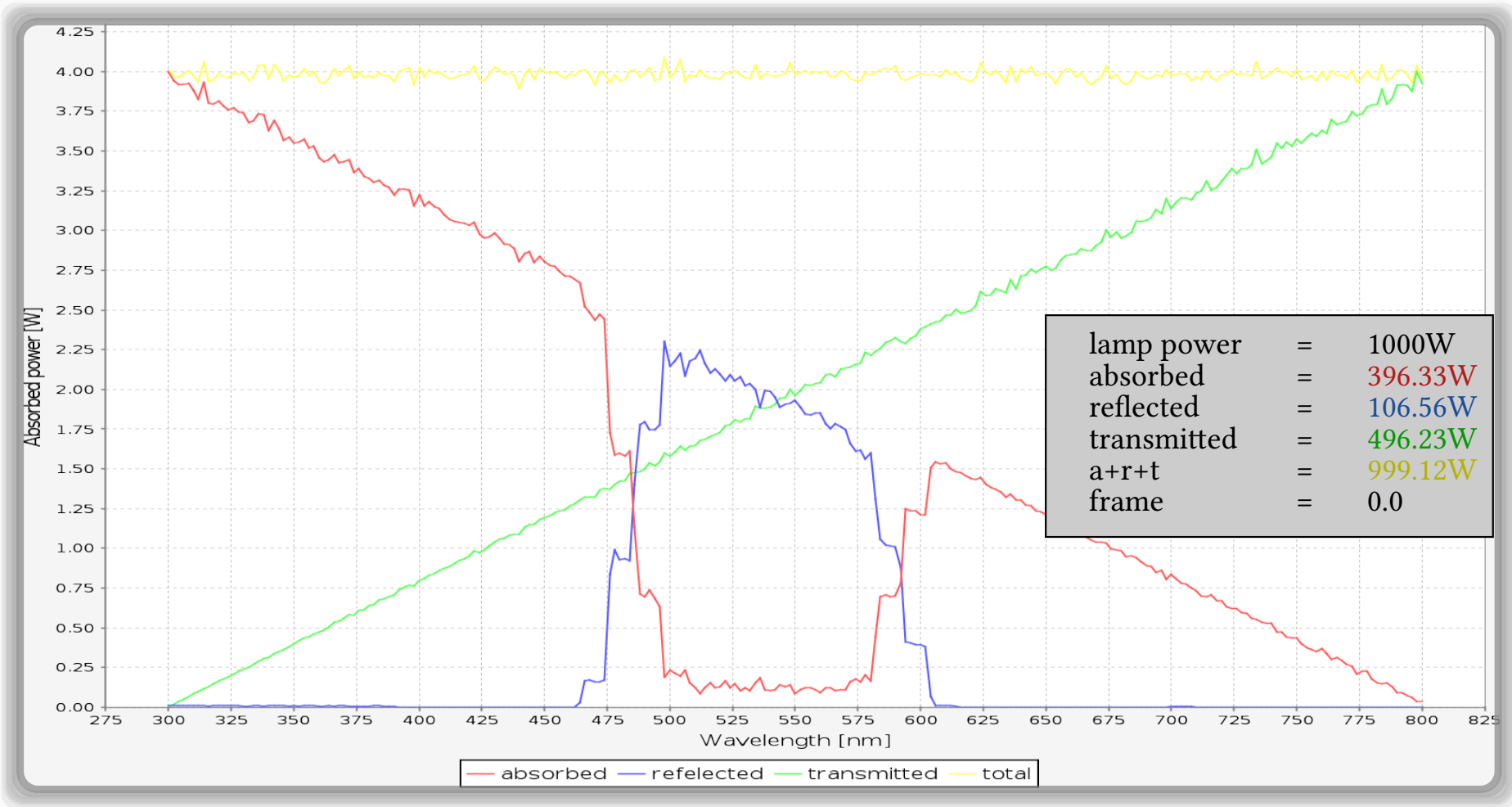


Shader Test - Example I



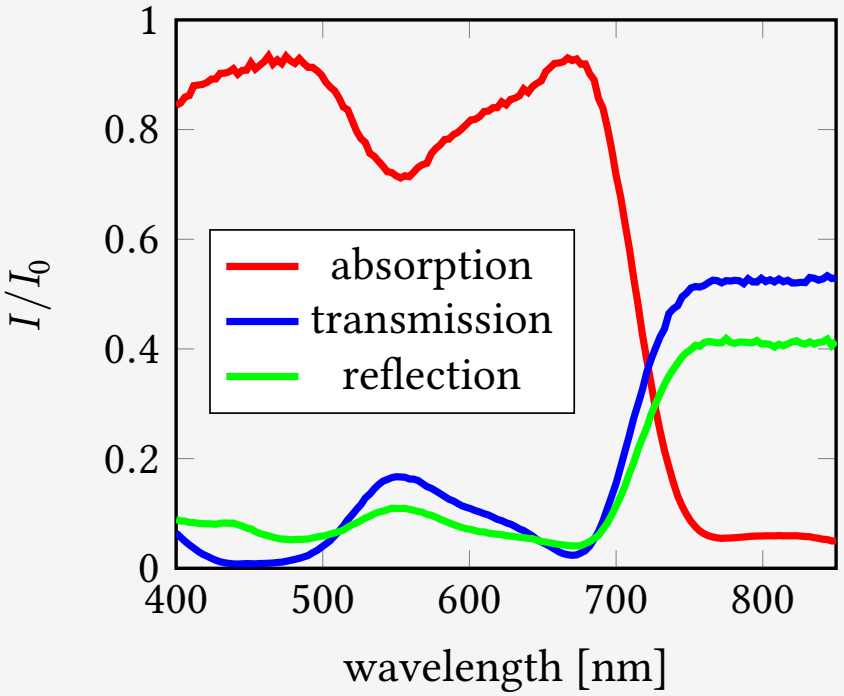
s.setDiffuse(new RGBColor(0, 1, 0));

Shader Test - Example II



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

Shader Test - Example III

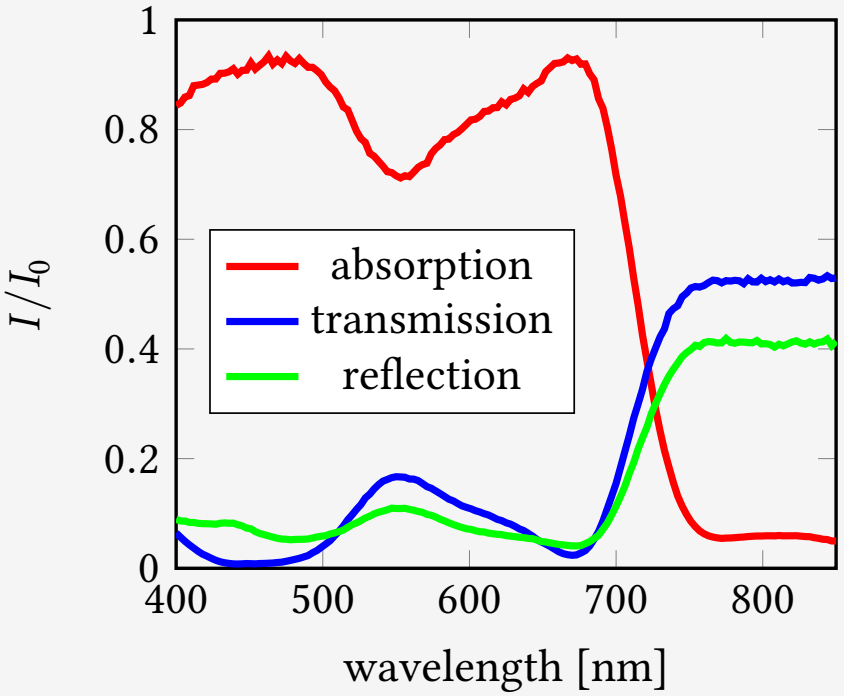


simulation

input: reflection and transmission

measured: absorption

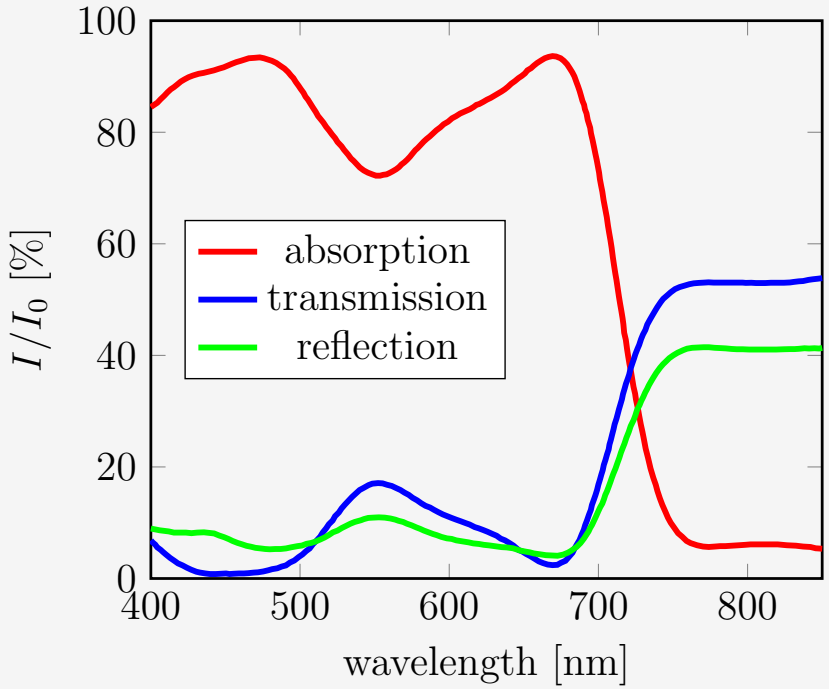
Shader Test - Example III



simulation

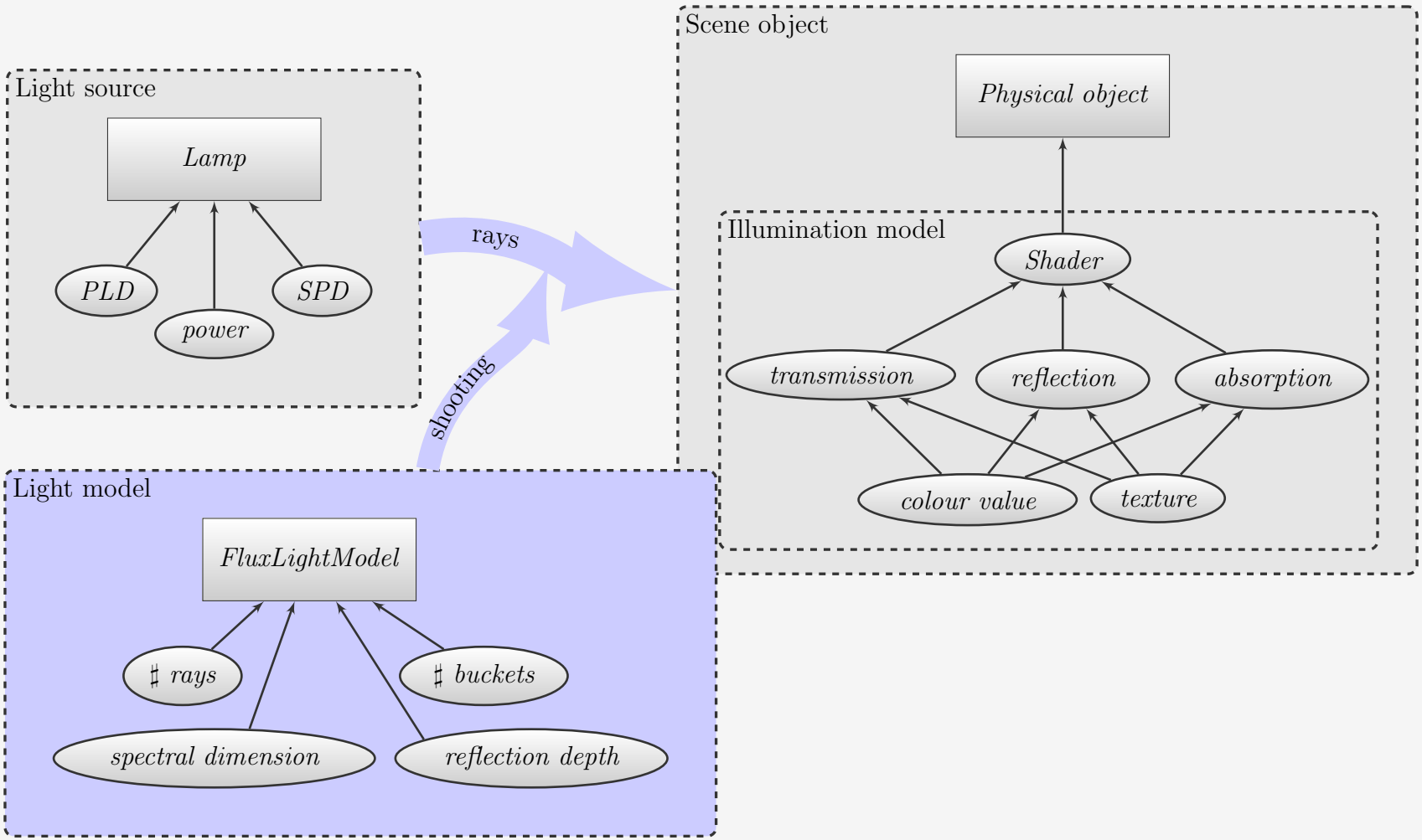
input: reflection and transmission

measured: absorption



soybean leaf: Kasperbauer, 1987

Progress in light modelling



(Henke and Buck-Sorlin 2017)

Differences CPU vs. GPU model

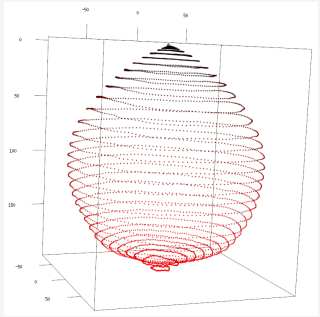
- Parametrization of the light models
  - number of rays
  - recursion depth
  - cutof power

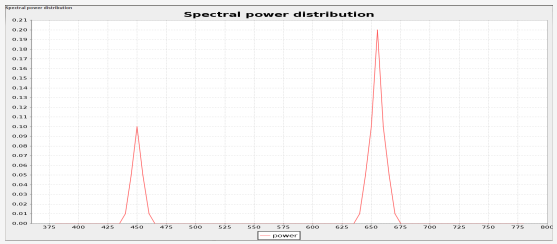
  } default for both: 30000, 10, 0.001
- Differences caused by ”spectral vs. non spectral” calculation
  - light emitted by the light sources
  - absorption, reflection and transmission will be different
  - interpretation of ranges or RGB
- Differences in implementation
  - in the Flux LM, directional light is assumed to be an approximation for a very distant light; for Twilight the distance can be set

Do not mix spectral and RGB shader!
Do not mix spectral and RGB colours in one shader!

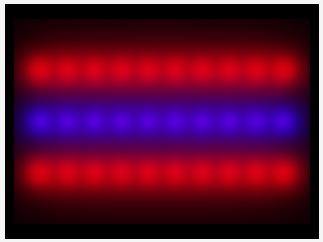
Table of Contents

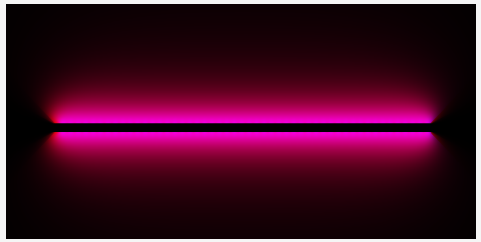
Climate-KIC project - LED grids in tomato stands



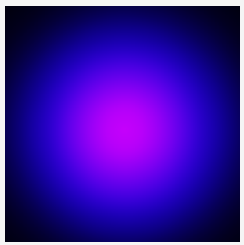


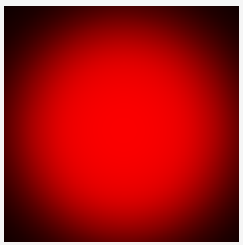
Measured physical distribution, measured spectrum



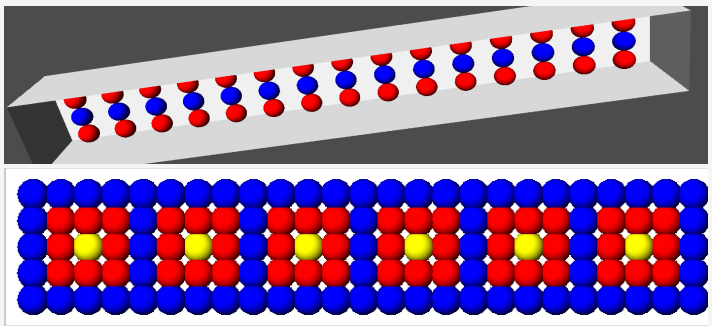


Simulated double-sided LED panel, as used in the greenhouse

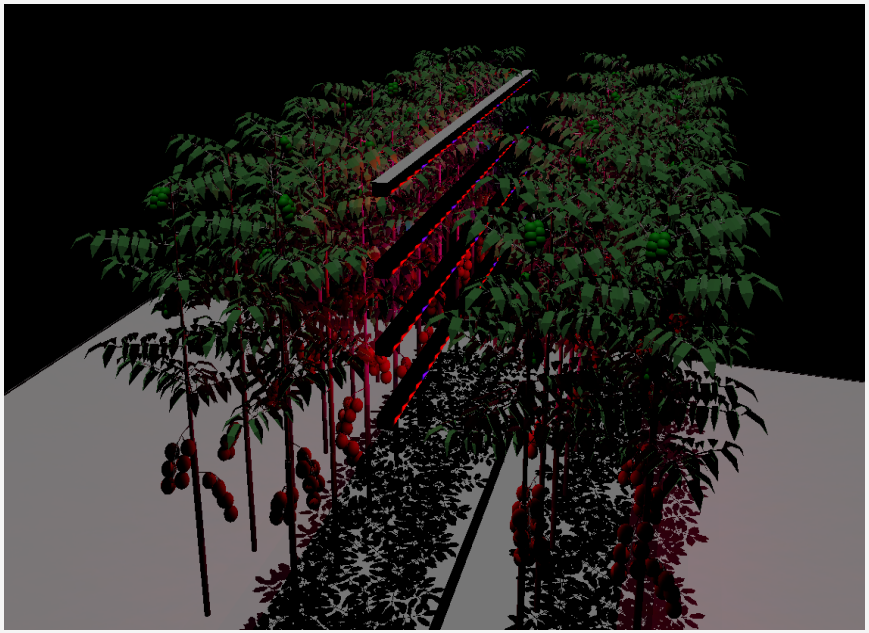




Model of single blue and red LED



Arbitrary arrangement of LEDs in panels



(de Visser and Henke)

Table of Contents

Summary

- Models
  - full spectral light model and renderer
  - generic light source
  - reduced computation time $\rightarrow$ increased accuracy
- Documentation
  - This presentation
  - FluxLightModel API
  - Henke and Buck-Sorlin 2017
- available with GroIMP $\geq$ v.1.5

Thank you for your attention!

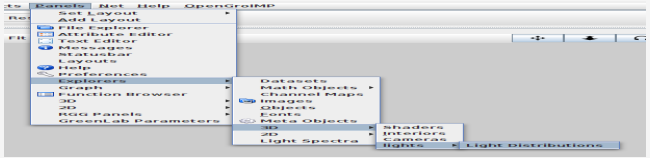
www.grogra.de

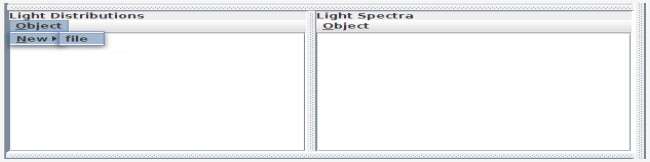
Table of Contents

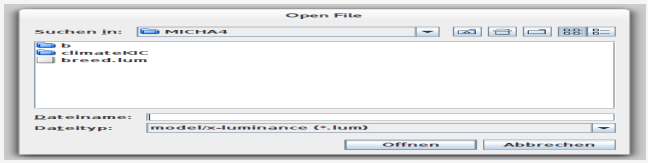
Define Light and Spectrum References

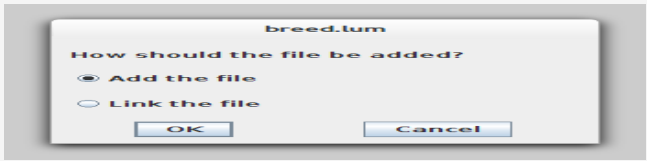
defining references

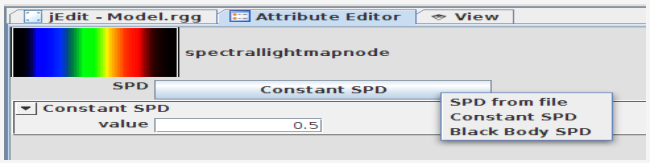
a) Open *Panels* → *Explorers*

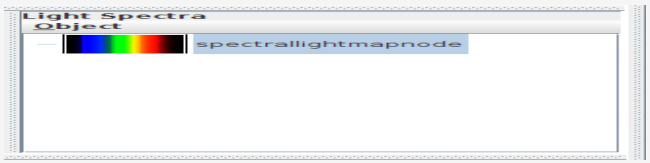










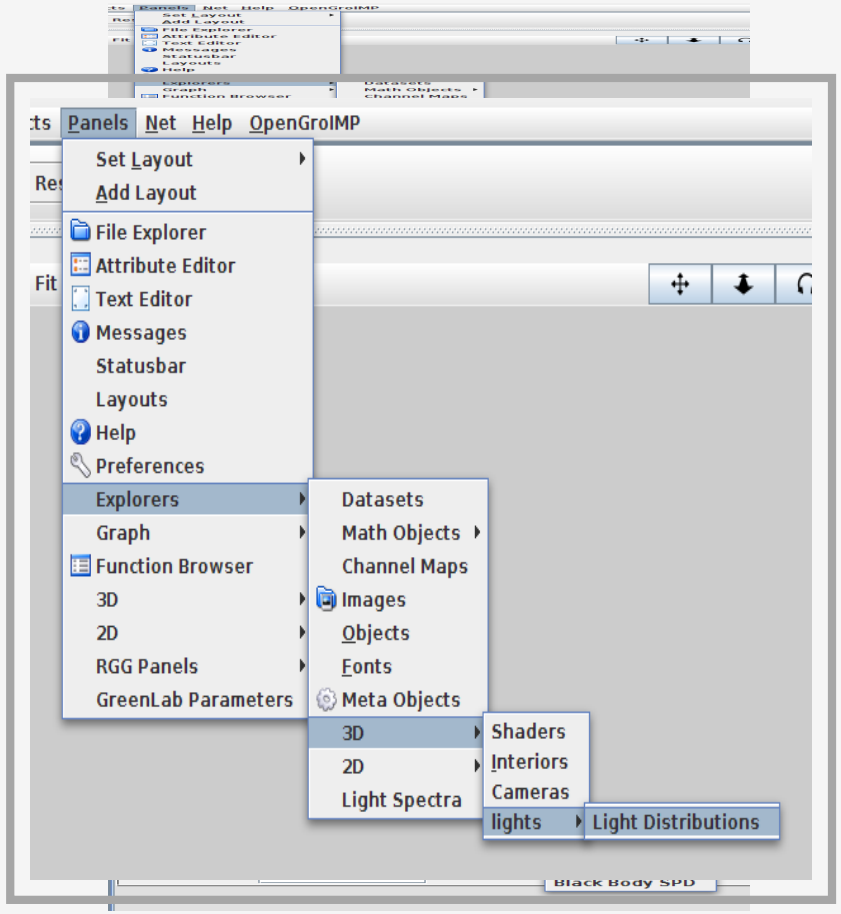


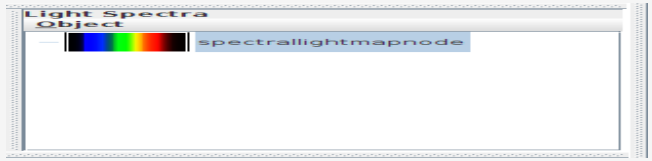
Define Light and Spectrum References

defining references

a) Open *Panels* → *Explorers*

b) Create new distribution object
Object → *New* → *file*





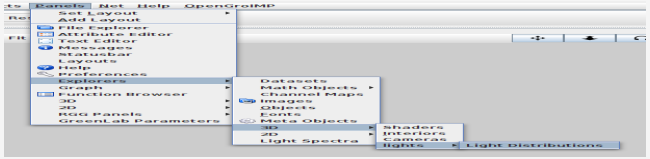
Define Light and Spectrum References

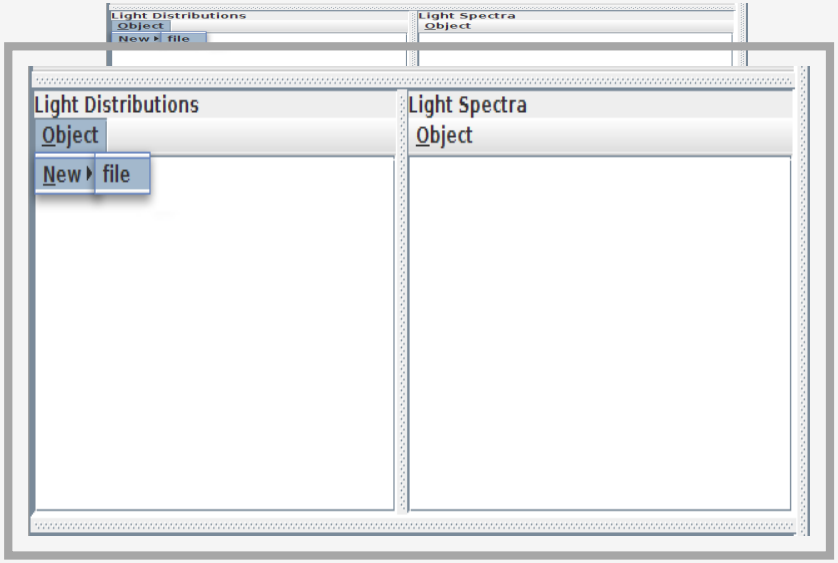
defining references

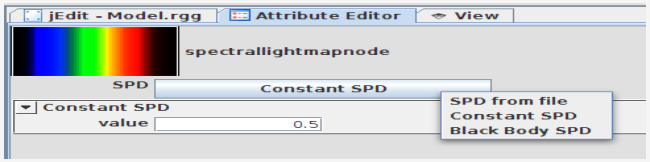
a) Open *Panels* → *Explorers*

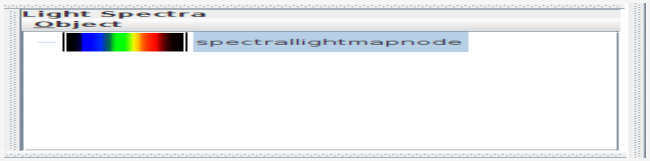
b) Create new distribution object
Object → *New* → *file*

   1. Choose a file









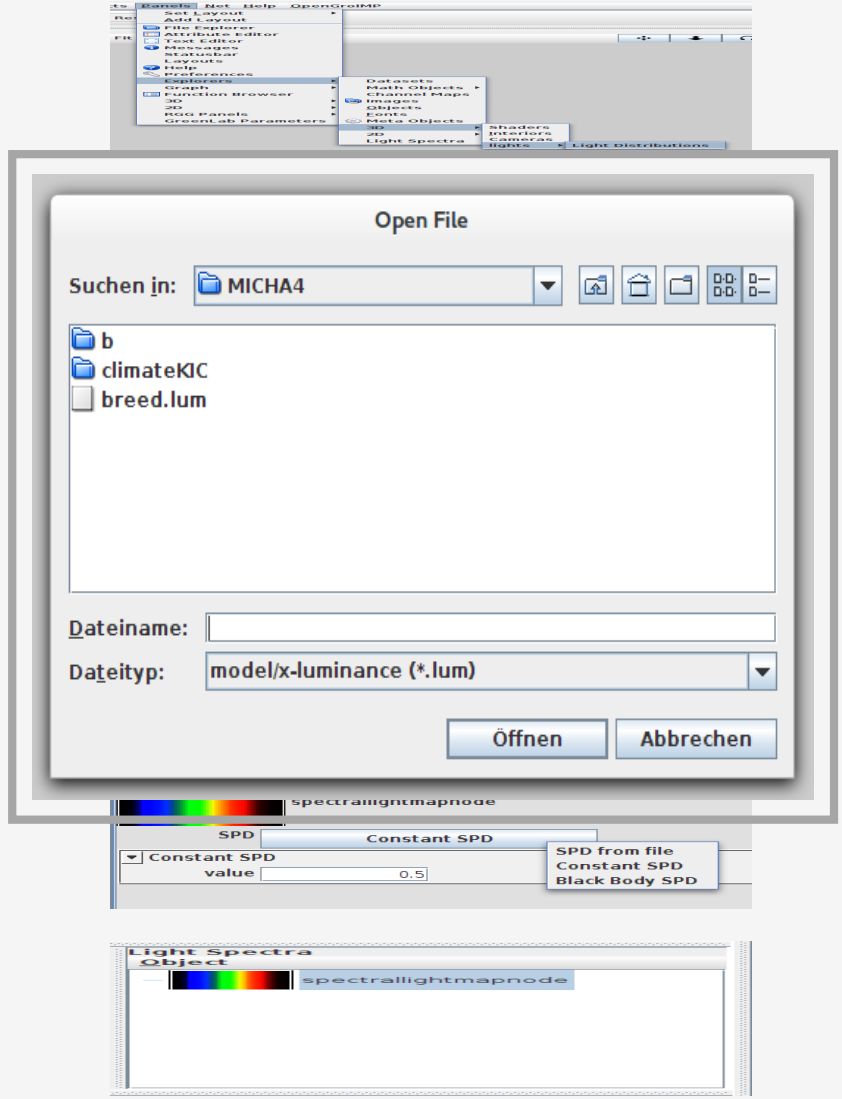
Define Light and Spectrum References

defining references

a) Open *Panels* → *Explorers*

b) Create new distribution object
Object → *New* → *file*

1. Choose a file
2. Add or link the file



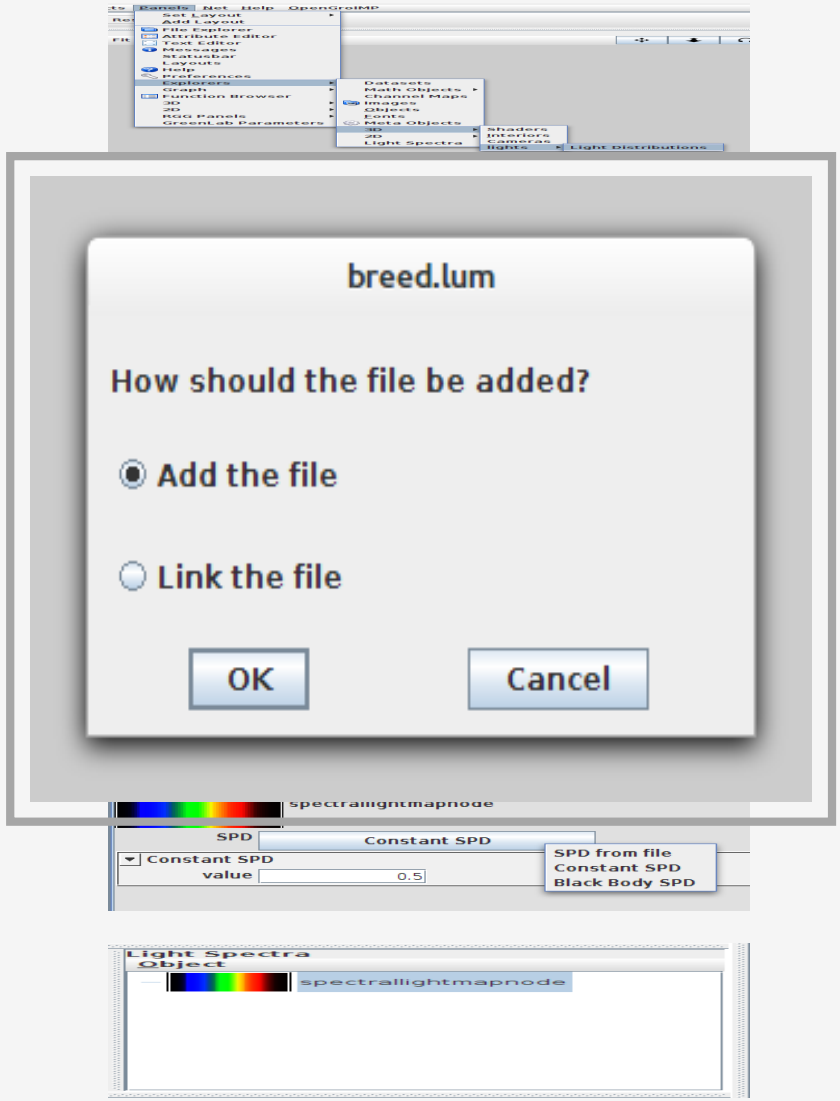
Define Light and Spectrum References

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



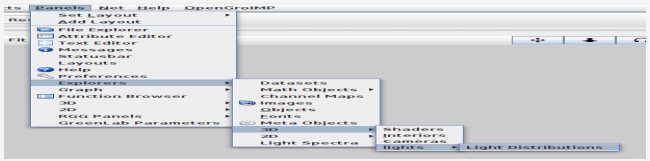
Define Light and Spectrum References

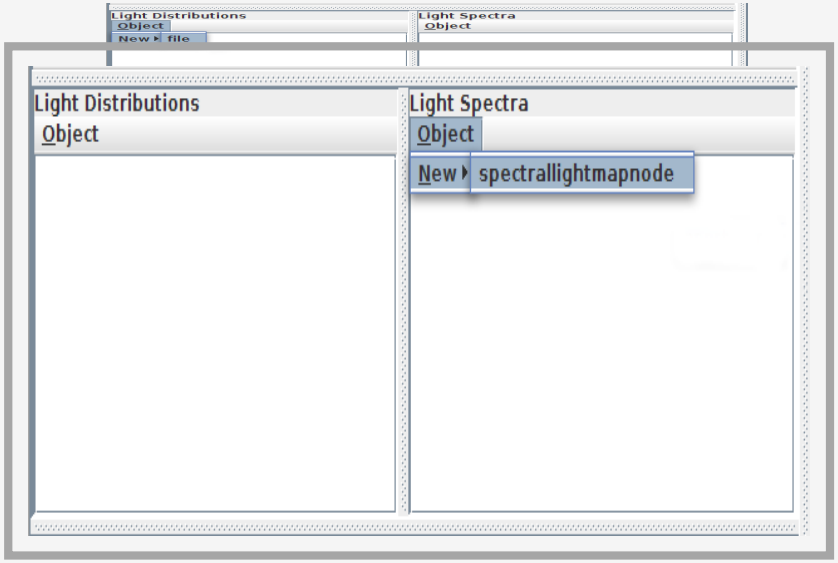
defining references

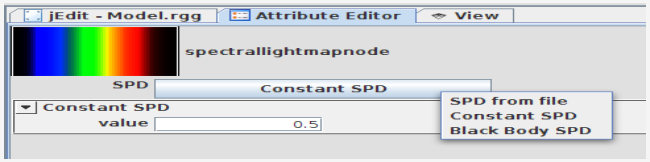
a) Open *Panels* → *Explorers*

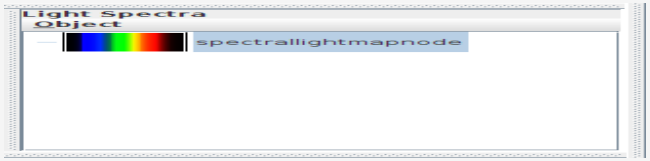
b) Create new distribution object
Object → *New* → *file*
   1. Choose a file
   2. Add or link the file

c) Create new spectra object
Object → *New* → *spectrallightmapnode*
   1. Switch to the Attribute Panel









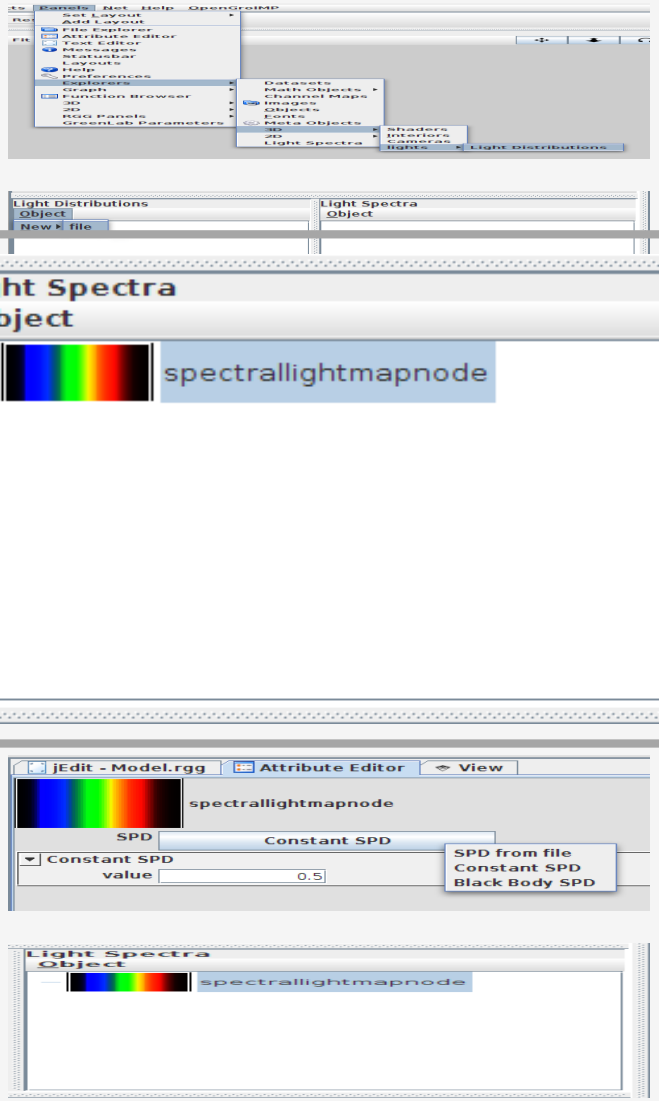
Define Light and Spectrum References

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*"



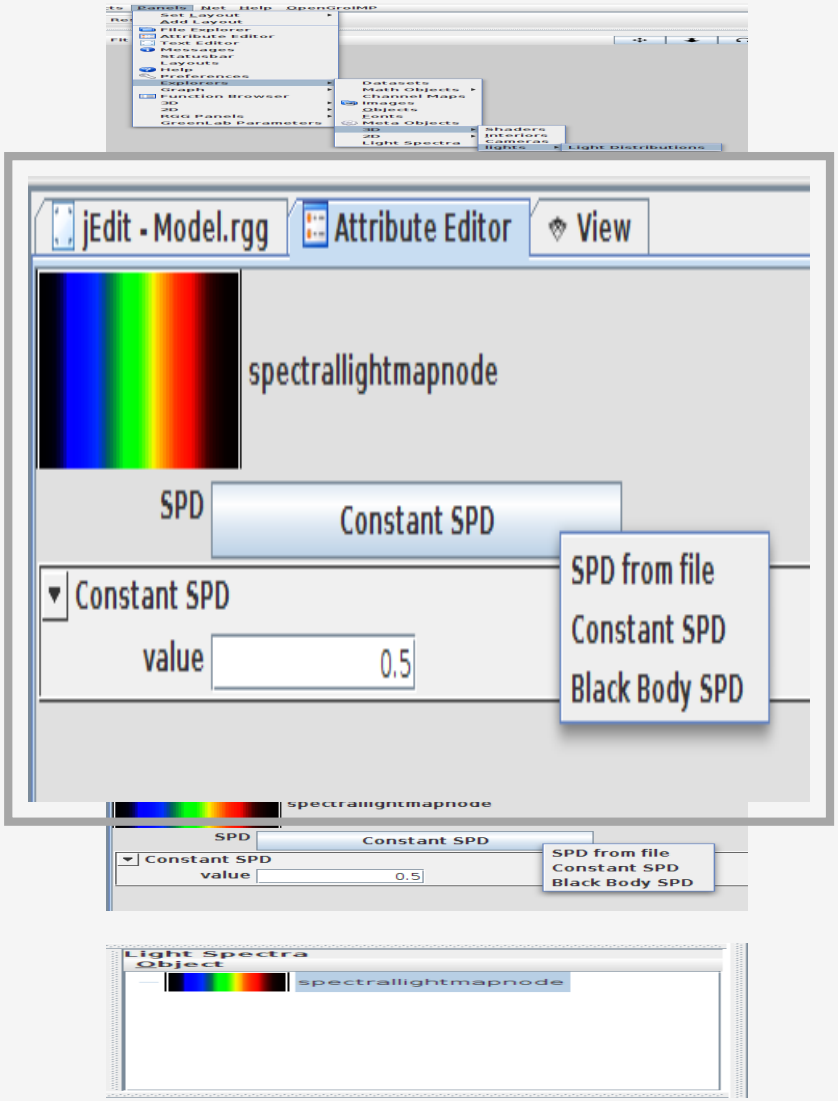
Define Light and Spectrum References

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

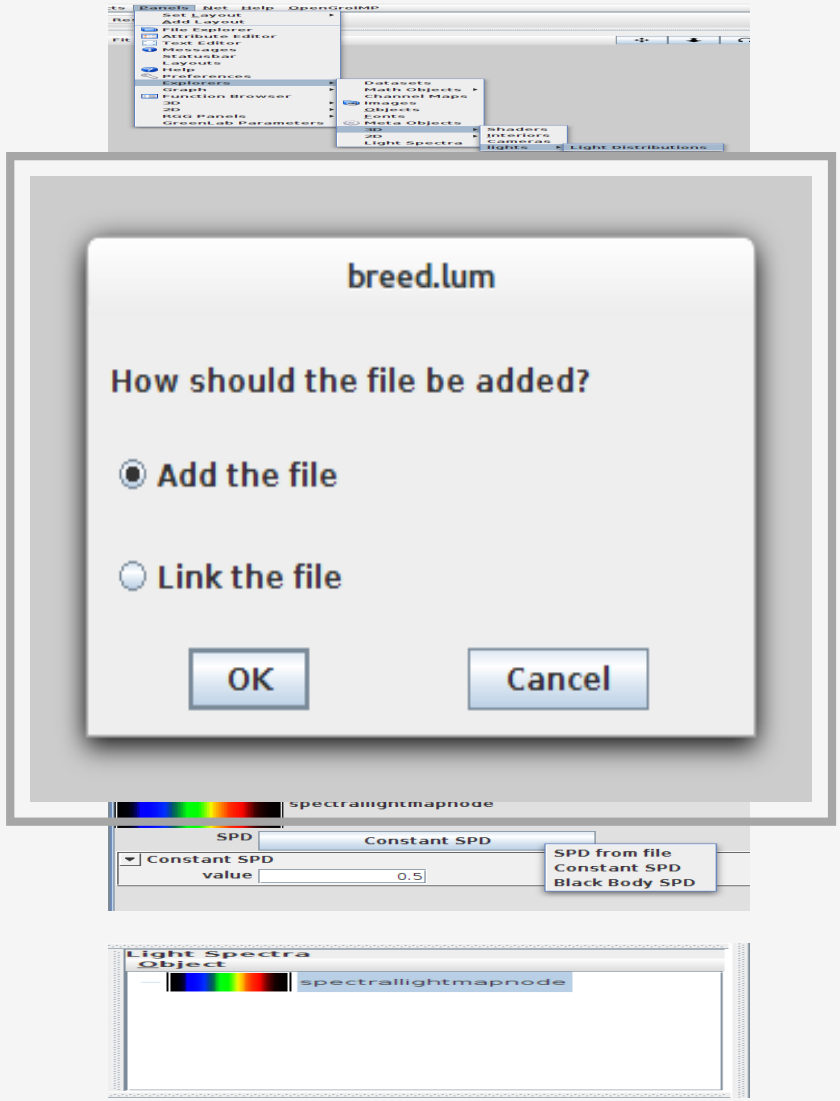
defining references

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-explanatory names



Define Light and Spectrum References

defining references

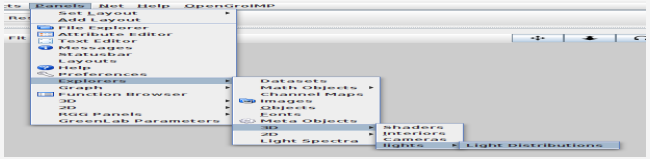
a) Open *Panels* → *Explorers*

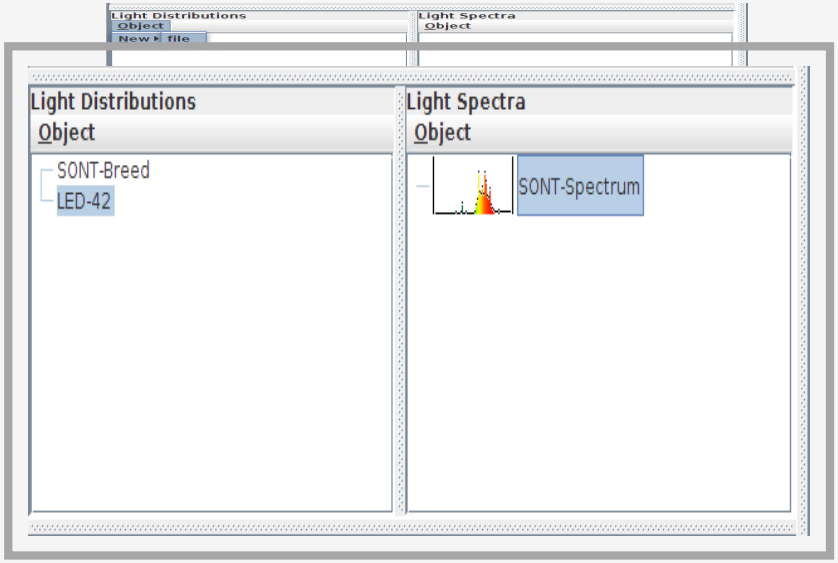
b) Create new distribution object
Object → *New* → *file*
   1. Choose a file
   2. Add or link the file

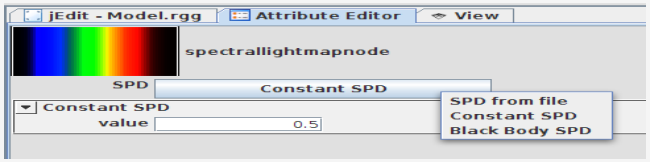
c) Create new spectra object
Object → *New* → *spectrallightmapnode*
   1. Switch to the Attribute Panel
   2. Select "*SPD from file*"
   3. Add or link the file

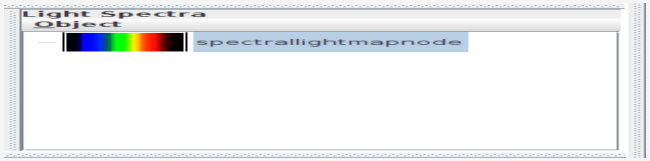
d) Choose self-explanatory names

↬ Back









Relevant Commands I

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

Relevant Commands II

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

Relevant Lighting Metrics

	Human Vision	Photosynthetically Active Radiation (PAR)	Electromagnetic Radiation
"Power"	Luminous Flux $[lm]$	Photosynthetically Active Photon Flux (PPF) $[\mu mol/m^2/s]$	Radiant Flux $[W]$
"Intensity"	Illuminance $[lx = lm/m^2]$	Photosynthetically Active Photon Flux Density (PPFD) $[\mu mol/m^2/s]$ Daily Light Integral (DLI) $[mol/m^2/day]$	Irradiance $[W/m^2]$
"Efficiency"	Luminous Efficacy $[lm/W]$	Photometry Photosynthetically Active Photon Efficacy $[\mu mol/\mathcal{J}, mol/kWh]$ $1\mu mol/\mathcal{J} = 3,6 mol/kWh$	Wall-Plug Efficiency or Radiant Efficiency [-]
	Photometry	Quantum Sensing	Radiometry

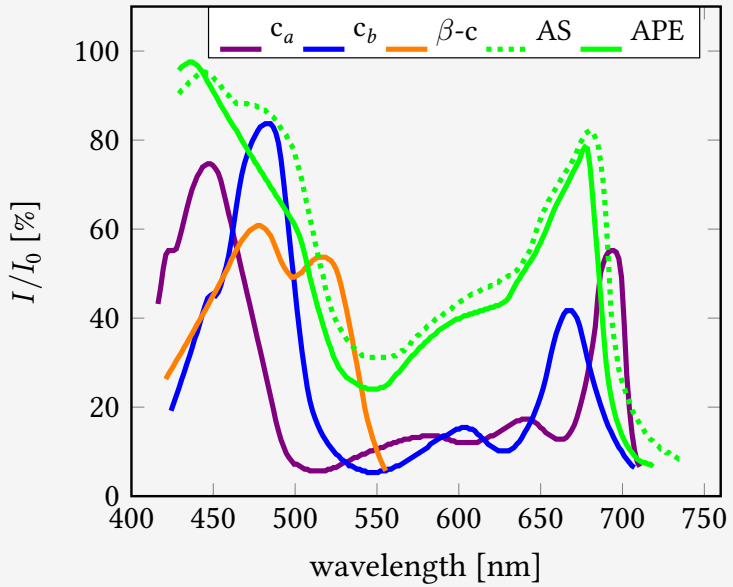
$1 mol = 6,02 * 10^{23}$ particles $(1 mol = 1,000,000 \mu mol)$

Photosynthetically active light spectrum: $400 - 700 nm$

PAR is species dependent!!

Photosynthesis rate as a function of absorbed wavelength

The composition of photochemically active pigments determines the action spectrum of a species and thus photochemical efficiency.



Legend:

c_a = chlorophyll a,

c_b = chlorophyll b,

β-c = β-carotene,

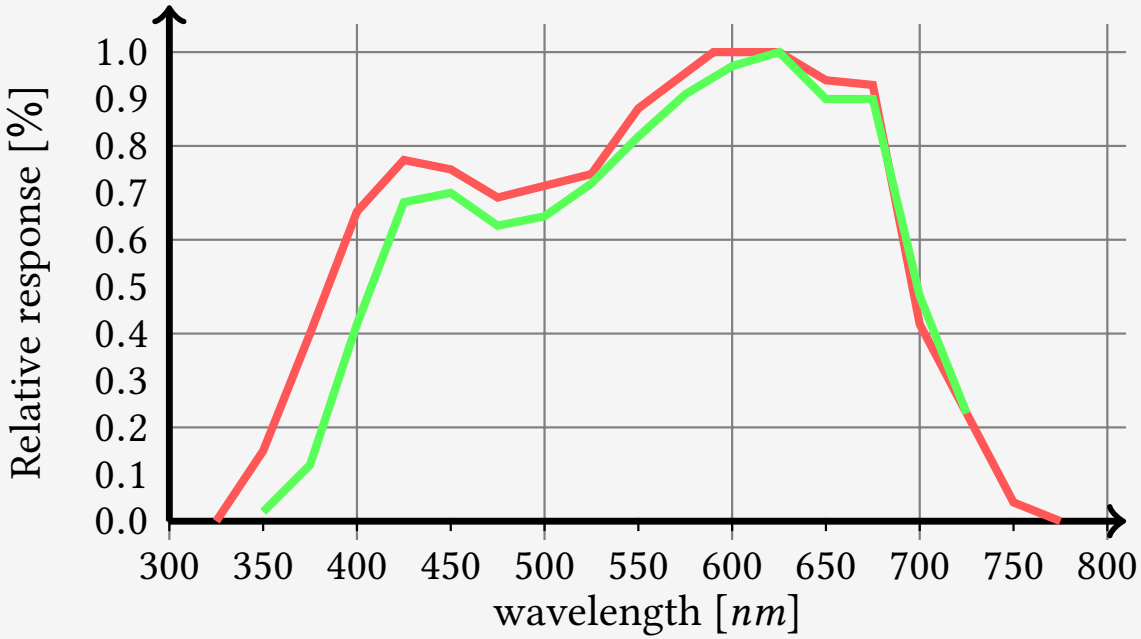
AS = absorption spectrum = relative light absorption,

APE = actual photochemical efficiency = action spectrum.

I/I_0 refers to the radiation absorbed, transmitted or reflected relative to incident radiation at the same wavelength.

Data taken from Karp 2013.

Relative quantum efficiency curve - McCree Curve



The relative quantum efficiency curve, also known as the McCree Curve, as determined by the average plant response for photosynthesis rate; Red: (Wikipedia.org: Photosynthetically_active_radiation). Green: Mean relative quantum yield of field plant species McCree 1972, Tab. IV.

Conversion: Candela $\Longleftrightarrow$ Watt

at a wavelength of 555 nm

$$1\ cd = 1\ lm/sr = 1/683\ W/sr$$
$$1\ lm = 1/683\ W$$
$$1\ W = 683\ lm$$

Total luminous intensity

$$I_v(\lambda) = 683.002 * \bar{y}(\lambda) * I_e(\lambda)$$

where $\bar{y}(\lambda)$ luminous intensity function (Sharpe et al. 2005)

Integration over the whole spectrum of wavelengths

$$\int_{\lambda=380}^{780} I_v(\lambda)$$

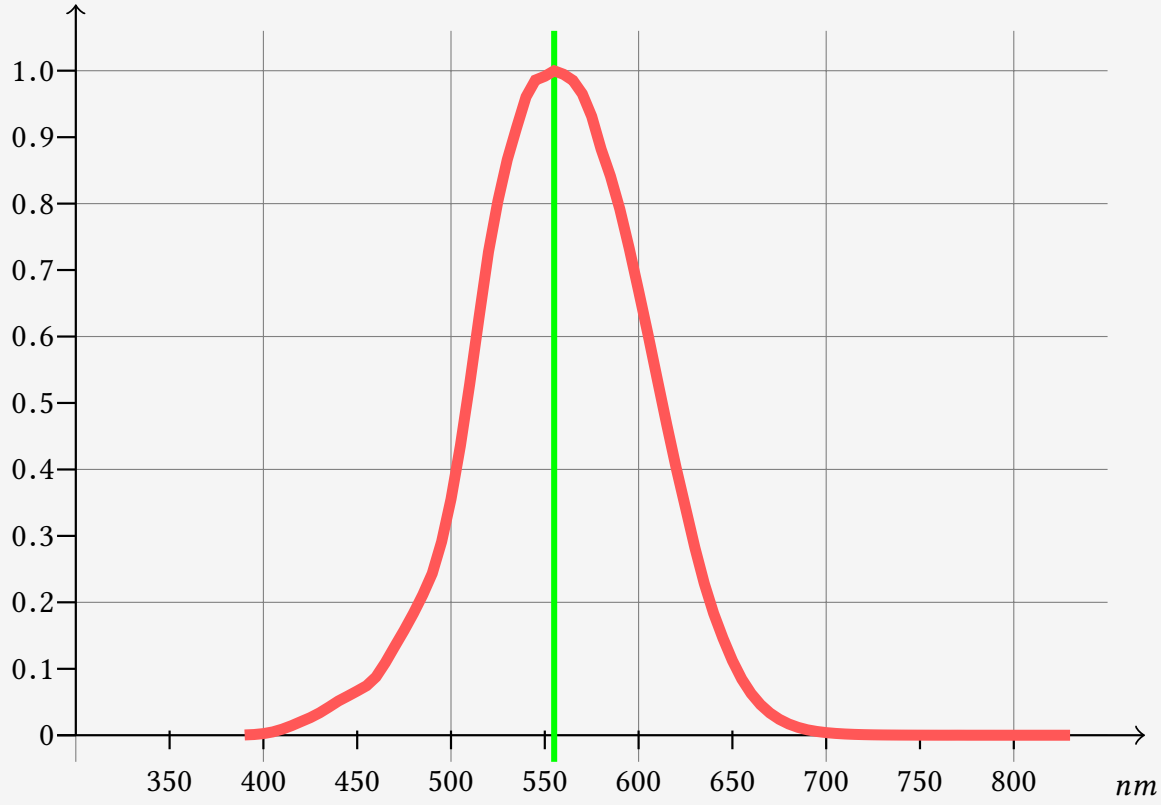
approximation

$$\sum_{\lambda=380}^{780} I_v(\lambda),\ \lambda \equiv 0(mod 5)$$

$I_e(\lambda)$ can be obtained by the spectral raytracer (getAbsorbedPowerMeasurement)

Luminous intensity function

- Sharpe, Stockman, Jagla & Jägle (2005): $2 - degV^*(l)$
- visual perception of brightness by the human eye



Conversion: Candela $\Longleftrightarrow$ Watt - Example

a single red LED at a wavelength of 650 nm with an output of 1600cd has
$1600cd/683.002 * 0.0665381 = 0.146W$ (output power)

Transformation: 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 ...
    ...
```

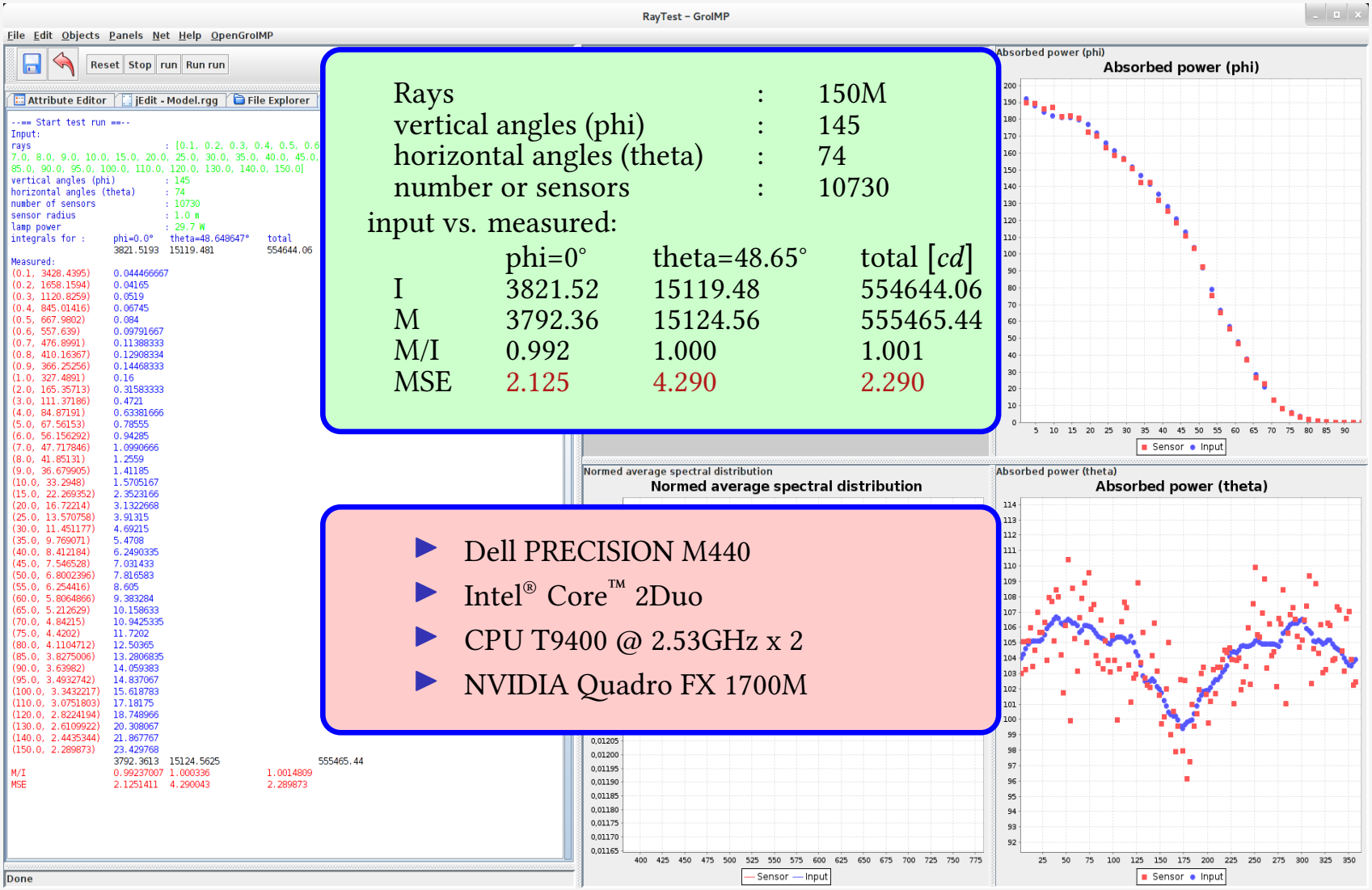
LUM file

```
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

Ray Test Scenario




```
--== Start test run ==--
Input:
rays                          : [0.1, 0.2, 0.3, 0.4, 0.5, 0.6
7.0, 8.0, 9.0, 10.0, 15.0, 20.0, 25.0, 30.0, 35.0, 40.0, 45.0,
85.0, 90.0, 95.0, 100.0, 110.0, 120.0, 130.0, 140.0, 150.0]
vertical angles (phi)         : 145
horizontal angles (theta)     : 74
number of sensors             : 10730
sensor radius                 : 1.0 m
lamp power                    : 29.7 W
integrals for :     phi=0.0°    theta=48.648647°    total
                    3821.5193   15119.481           554644.06

Measured:
(0.1, 3428.4395)    0.044466667
(0.2, 1658.1594)    0.04165
(0.3, 1120.8259)    0.0519
(0.4, 845.01416)    0.06745
(0.5, 667.9802)     0.084
(0.6, 557.639)      0.09791667
(0.7, 476.8991)     0.11388333
(0.8, 410.16367)    0.12908334
(0.9, 366.25256)    0.14468333
(1.0, 327.4891)     0.16
(2.0, 165.35713)    0.31583333
(3.0, 111.37186)    0.4721
(4.0, 84.87191)     0.63381666
(5.0, 67.56153)     0.78555
(6.0, 56.156292)    0.94285
(7.0, 47.717846)    1.0990666
(8.0, 41.85131)     1.2559
(9.0, 36.679905)    1.41185
(10.0, 33.2948)     1.5705167
(15.0, 22.269352)   2.3523166
(20.0, 16.72214)    3.1322668
(25.0, 13.570758)   3.91315
(30.0, 11.451177)   4.69215
(35.0, 9.769071)    5.4708
(40.0, 8.412184)    6.2490335
(45.0, 7.546528)    7.031433
(50.0, 6.8002396)   7.816583
(55.0, 6.254416)    8.605
(60.0, 5.8064866)   9.383284
(65.0, 5.212629)    10.158633
(70.0, 4.84215)     10.9425335
(75.0, 4.4202)      11.7202
(80.0, 4.1104712)   12.50365
(85.0, 3.8275006)   13.2806835
(90.0, 3.63982)     14.059383
(95.0, 3.4932742)   14.837067
(100.0, 3.3432217)  15.618783
(110.0, 3.0751803)  17.18175
(120.0, 2.8224194)  18.748966
(130.0, 2.6109922)  20.308067
(140.0, 2.4435344)  21.867767
(150.0, 2.289873)   23.429768
                    3792.3613   15124.5625          555465.44
M/I                 0.99237007  1.000336            1.0014809
MSE                 2.1251411   4.290043            2.289873
```

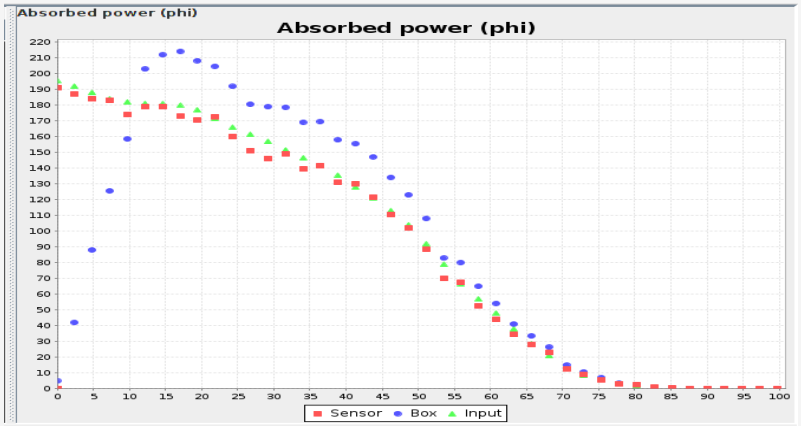
Rays	:	150M
vertical angles (phi)	:	145
horizontal angles (theta)	:	74
number or sensors	:	10730

input vs. measured:

	phi=0°	theta=48.65°	total [*cd*]
I	3821.52	15119.48	554644.06
M	3792.36	15124.56	555465.44
M/I	0.992	1.000	1.001
MSE	2.125	4.290	2.290

- Dell PRECISION M440
- Intel® Core™ 2Duo
- CPU T9400 @ 2.53GHz x 2
- NVIDIA Quadro FX 1700M

Comparison Box vs. Sensor

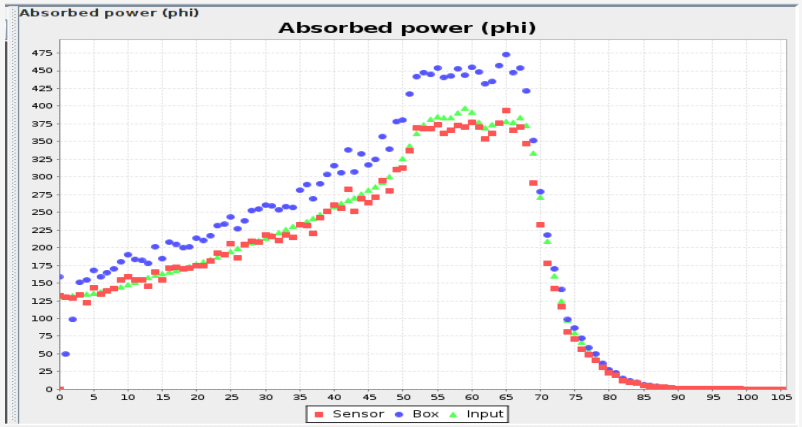


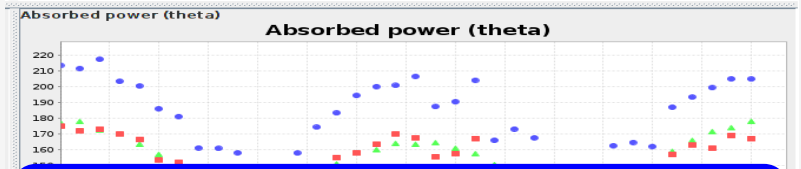
LED

Rays	:	50M
vertical angles (phi)	:	145
horizontal angles (theta)	:	74
number or 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





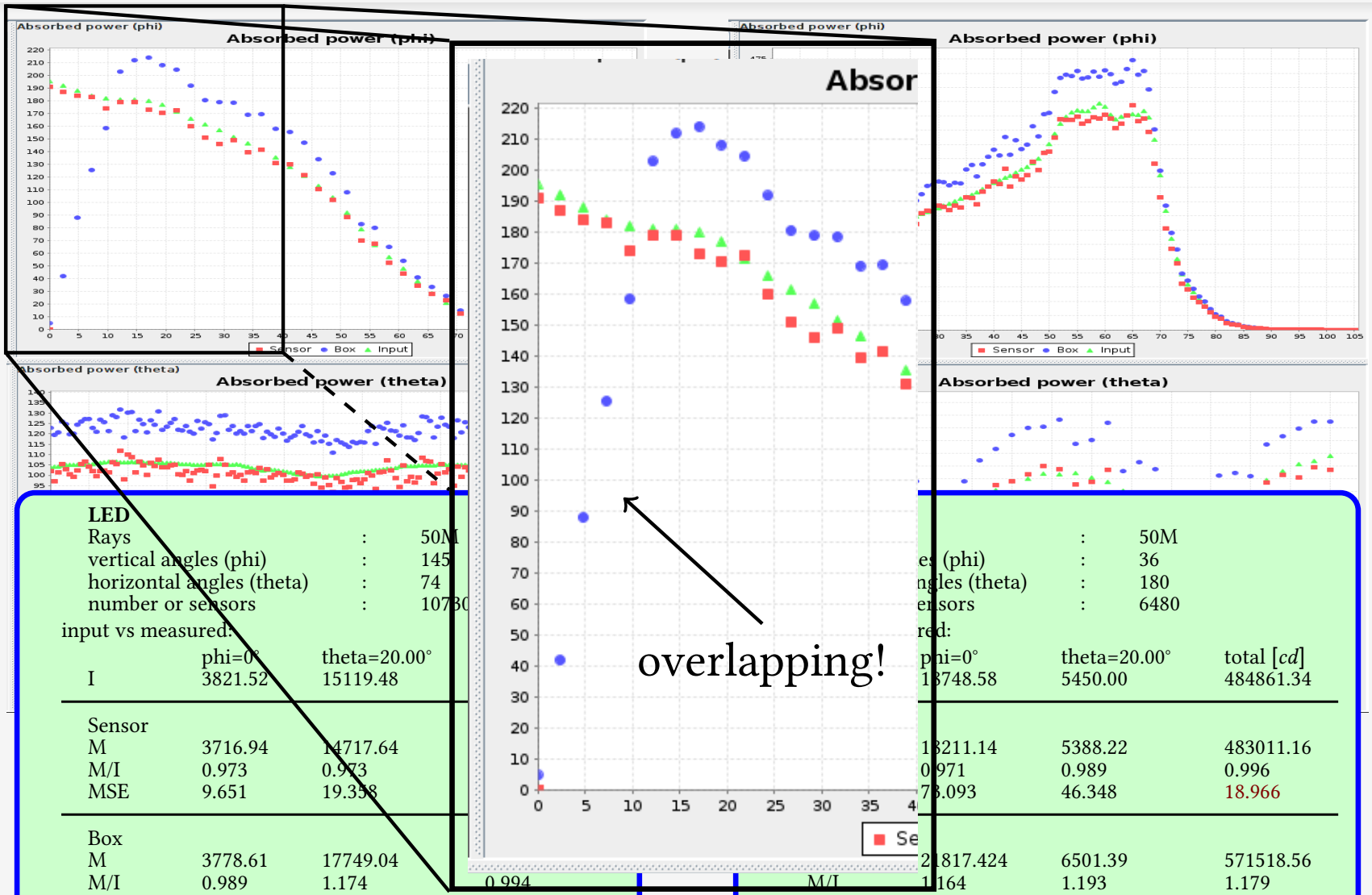
SONT

Rays	:	50M
vertical angles (phi)	:	36
horizontal angles (theta)	:	180
number or 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



LED

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

input vs measured:

	phi=0°	theta=20.00°	total [*cd*]
I	3821.52	15119.48	
Sensor			
M	3716.94	14717.64	
M/I	0.973	0.973	
MSE	9.651	19.358	
Box			
M	3778.61	17749.04	
M/I	0.989	1.174	0.994
MSE	1143.13	341.54	1187.79

	:	50M
es (phi)	:	36
ngles (theta)	:	180
ensors	:	6480

red:

	phi=0°	theta=20.00°	total [*cd*]
	13748.58	5450.00	484861.34
	13211.14	5388.22	483011.16
	0.971	0.989	0.996
	73.093	46.348	18.966
	21817.424	6501.39	571518.56
M/I	1.164	1.193	1.179
MSE	955.677	926.649	766.348

Common Problems

- configuration
  - incompatible bit-versions of Java and libraries (32 vs 64 bit)
  - two graphics cards (onboard and real) → disable the onboard card
  - 'use multiple devices' may produce errors (at least for rendering) → disable this option

Common Problems

- configuration
  - incompatible bit-versions of Java and libraries (32 vs 64 bit)
  - two graphics cards (onboard and real) → disable the onboard card
  - 'use multiple devices' may produce errors (at least for rendering) → disable this option
- application
  - wrong/different model setup compared with reality
  - misunderstanding of *setDiffuse* - it is the amount of reflected light
  - mixed units: Watt, Candela, $\mu mol\ photons\ m^{-2}s^{-1}$ v
  - Phong shader does not normalize the transmission and reflection. Absorption and transmission don't surpass 100% be physically plausible.
  - Mixing spectra and RGB colours in one shader is not advised!

Interesting extensions

- add a function *getSensedIrradiance()* to a measurement so that one can find out how much light in total reaches an object
- add functions to get the amount of reflected and transmitted light of an object
- extend the measurement functions to get results for each reflection level (to distinguish between direct light and reflected light)
- *setPower()* function of a light currently distributes the power equally over the whole spectrum ignoring that different wavelengths have different powers
- adding new shaders: opaque, mirror (perfect specular reflection implemented as a separate brdf type or shader)
- addind new light sources, e.g., box- or cylinder-shaped, curved area lights

Table of Contents

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*(λ), 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.
- Henke, M., Buck-Sorlin, G.H. (2017) Using a full spectral raytracer for the modelling of light microclimate in functional-structural plant modelling. Computing and Informatics, 36:1492-1522
- Karp, G.: Cell and Molecular Biology: Concepts and Experiments (7. ed.). John Wiley & Sons, 2013
- 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
- McCree, K.J. (1971/72) The action spectrum, absorbance and quantum yield of photosynthesis in crop plants. Agricultural Meteorology. 9:191-216
- Phong, B.T. (1975) Illumination for computer generated pictures, Communications of ACM 18, 6:311–317