General representation of light in 3D

1 Light Source

How to store light source information in computer ?

The project is based on Rendering equation :

$$L_o(\mathbf{x}, \omega_o, \lambda, t) = L_e(\mathbf{x}, \omega_o, \lambda, t) + \int_\Omega f_r(\mathbf{x}, \omega_i, \omega_o, \lambda, t)\, L_i(\mathbf{x}, \omega_i, \lambda, t)\, (\omega_i \cdot \mathbf{n})\, \mathrm{d}\omega_i$$

From it, we determined that all light sources should have method *getNextBeam()* from which you can get information „from where to where is beam going“, some information about its color and which light source emitted it. This is how this project tries to model different types of lights.

Method *getNextBeam()* in LS[1]es implemented in this project works this way:
1) generates coordinates [x,y,z] – beams origin
2) generate angle α and β – where is it shining (direction)
3) generate wave length – from which we will deduce color

Point 1) and 2) effectively give you a vector, but they are separated for easier generation (So you can have different distributions for origins and directions)

We generate wave length and no RGB values, because this way, we can easily produce different optical effect just by adding know physical principles into renderer (if we add refraction of beams, it will(or not :D) work correctly without any more work). Down side of this is that it introduces need for SPD[2]s and little bit of Color science. This is why every Light Source has besides *getNextBeam()* , Constructor taking SPD and *get/setSPD* methods.

For some future shenanigans with color intensity, we would like to know how many beams had LS generated, thus *getNumberOfBeams()* is present here too.

1- LS : Light source
2 – SPD: Spectral Power Distribution

2 Spectral Power Distribution

Spectral Power Distribution or SPD for short, represents how much energy is illuminated/absorbed/… on each wavelength.

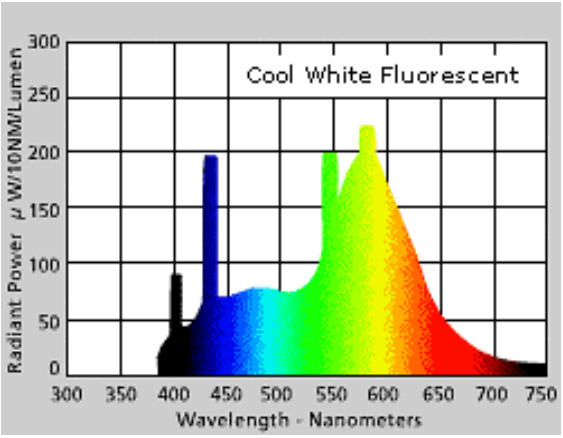


Illustration 1: SPD of flourescent lamp, source: http://www.lrc.rpi.edu/education/learning/terminology/spectralpowerdistribution.asp

As you can see it is neat graph with lots of info. For example, you can divide each Y by corresponding wavelength power and normalize is (so max value would be 1) and use this as graph of ,,number of photons per wavelength“, which is useful for generating beams wavelength. Put simple, because it is 2D graph, you can easily transform it to something else, what describes relations between wavelength and mentioned ,,something else“.

In this project, SPDs are used for color deduction and wavelength generations.

In our *interface SpectralPowerDistribution* , we have method *getNextLamnbda()*, which should translate SPD to „random distribution“ of lambdas (wavelengths) - wavelength generations.

Then it has *getValue(double lnb)*, which should return SPDs power on wavelength lmb – used for color creation.

3 Color

Human eyes have 3 cones (S,M,L) which perceive color. They are sensitive to different, overlapping parts of visible light spectrum, and are more sensitive to different wavelengths in their corresponding part of light spectrum. This means that color does not depend only on which wavelengths are hitting your eye, but also on how many fotons with same/different wave length are there.

This is why there are things like CIE standard observer, and why we have *interface Color* with *SPDtoRGB(SPD a)* method. It takes SPD and outputs RGB color using some ColorScience voodoo (it calcualte XYZ from SPD, using cone sensitivity defined in specific implementation of *Color*, and then it converts XYZ to RGB).

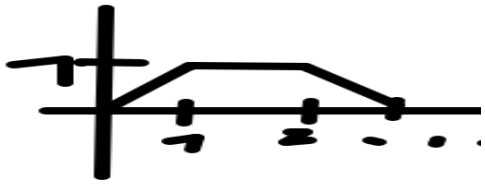
Then something with color intensity … but idk

4 Math

Most of math functions for 3D are from THE INTERNET, and I haven't bothered to find time to understand them.

5 Non uniform distributions

How to turn uniform random distribution to nonuniform random distribution ?



Imagine that you want to make nonuniform random distribution that look like this. You want as many ones as twos , but only half as many 0,5s and 2,5s as ones.

Illustration 2: distribution

How to turn uniform to “nonuniform“ distribution:

Lets *f(x)* be the function on the graph. Let *i* be its integral (from 0 to $a = 3$ in this case). *R* is uniform random distributor in interval [0,1).

$i=\int_0^{a=3} f(x)$ $v=i*R$, so *v* is some value in [0,i).

Now we want to solve $\int_0^n f(x)=v$ for n. Our output is $\frac{n}{a}$ or n/3 in this case.

But why ?!

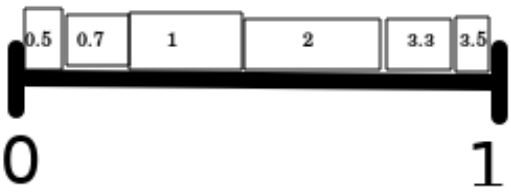


Illustration 3: distribution on line

You can have uniformly any number form [0,1). Imagine you got 0,1; 0,6;0,7; 0,9. If you make mark on line on the picture for these number, you see that 0,1 is below rectangle with 0,5; 0,6 and 0,7 are below 2 and 0,9 is below 3,5. If you look closely, the rectangles width determines how many times it is picked by uniformly distributed numbers. The width depends on integral for rectangle value from picture 2. Now just change line from 0-1 to 0-*i*, where *i* is integral form before, or sum of integrals for all values on graph 2. Now just get *v =R*i*, look at number in rectangle over v, and voila, you got your nonuniform distribution.

In reality you want to make steps by 1, so you can just sum, so you have to find *z-y=1*, *Iy<=v<=Iz*, *y<=n<=z* and return *n + (v-Ia)/(Ib-Ia)* to smooth it (and div it all with *a* so result would be in [0,1)).

This part of document will be reworked, because right now it is mess...

6 Renderer

Camera

~~Method *computeBeam(Beam b)* detects if beam hit camera, and if it hit from right direction (from somewhere where is camera looking), it will add this beams wavelength to SPD on camera pixel it went through.~~

Then *getPixels()* will compute collor from *SPD* and *Color* on each pixel, and output x*y matrix of RGB colors.