General representation of light in 3D

1 – LS : Light Sourcer

2 – SPD: Spectral Power Distribution

1 Light Source

How to store light source information in computer ?

The project is based on Rendering equation :

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

From it and after some implementations, 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. Besides that LS only needs method *getNumberOfBeams()* which is used for color (power on camera) deduction. With this 2 methods, we can load any valid LS and immediately start using it.

Therefore LS should implement this interface (Java):

```
interface LightSource extends java.io.Serializable{
        Beam getNextBeam();
        double numberOfBeams();
}
```

Beam is data class, and generally looks like this:

```
public class Beam{
        SpatialData sd;
        LightSource parent;
        ColorData c;
}
```

Where SpatialData sd is some representation of Beams origin and direction, LightSource parent is reference for LS that generated this beam and ColorData c is another data class, that contains representation of beams color.

In my implementations, I directly replaced SpatialData sd *with 2 variables :* Vector3 origin *and* Vector3 direction *, and* ColorData c *with* double lambda *and* double power. *To use LS with different beam's* SpatialData sd*, you have to Transform it to be compatible with your renderer, which could slow it down significantly (depends on how different are your representations) and besides some representations having less memory requirements then mine (for example* Vector3 origin *and* Double angleA, andlgeB*), I thing its better to use faster (all this math needs vectors) 2-vector representation for all LS.*

In my implementations, I generate wave lengths with power and not RGB values, because this way, we can easily produce different optical effect just by adding know physical principles into renderer. Down side of this is that it introduces need for SPD[2]s and little bit of Color science.

If Beam isnt compatible with your renderer, the easiest solution is to wrap it in LS that is. This way you have to transform Beam data only once → when getNextBeam() is called (wrapper returns transformed beam).

1 – LS : Light Sourcer
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.

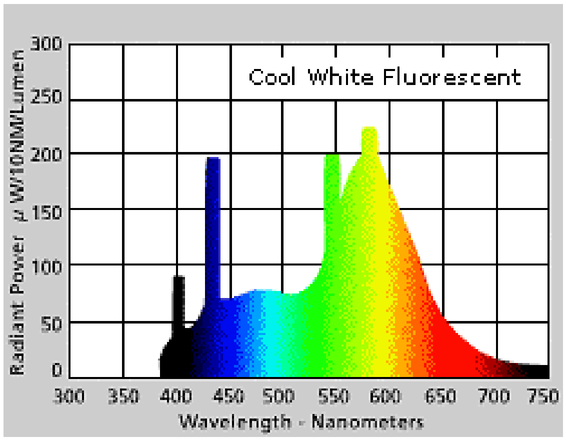

Illustration 1: SPD of flourescent lamp, source: http://www.lrc.rpi.edu/education/learning/terminology/spectralpowerdistribution.asp

In this project, SPDs are used for color deduction and wavelength generations. All SPDs integrals should be equal to 1.

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

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

1 – LS : Light Sourcer
2 – SPD: Spectral Power Distribution

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 ColorScience - it calcualte XYZ from SPD, using cone sensitivity defined in specific implementation of *Color*, and then it converts XYZ to RGB.

1 – LS : Light Sourcer
2 – SPD: Spectral Power Distribution

4 Math

Using Vectors [x,y,z] (4. value is 1 implicitly) for points, directions and normals, 4x4 row major order matrices for transformations.

1 – LS : Light Sourcer
2 – SPD: Spectral Power Distribution

5 Non uniform distributions

How to turn uniform random distribution to nonuniform random distribution ?

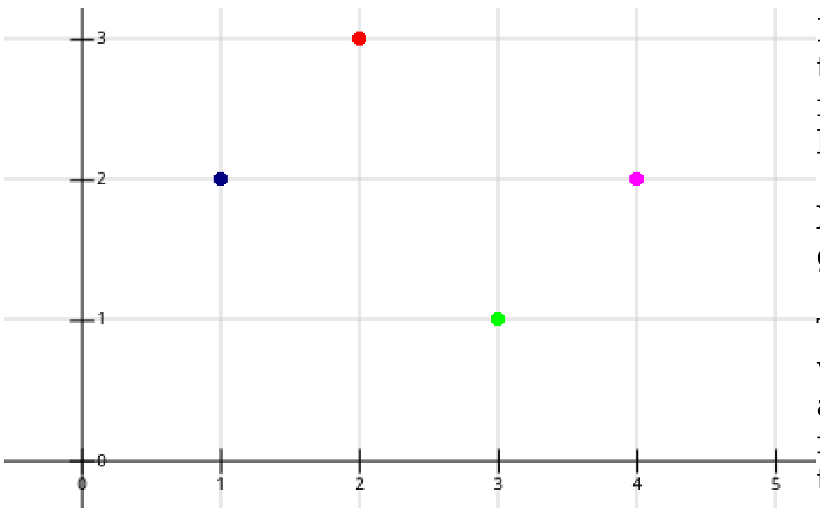

Illustration 2: some samples

In many cases, we want to create weighted random distribution based on some samples.

X is what we want to get, Y is its weight.

That means that we want same amount of *1* as *4*, but only half as many *3* as *1* (or *4*) and three times more *2* than *3*.

Easy solution is to compute the sum of functional values form 1 to 4 of this sample, which is 8, and then produce random number η form [0,1) , multiply it with found sum and then sum functional values until our sum is > η*8. Our random number from this weighted distribution is the last X used in sum. *I use sum instead of integral, because we do not care about step size in this case.*

Generalization:

We have samples on uniform set {a,...,b} with values of f(x), x ∈ {a,...,b}

1) Calculate ***I*** = $\sum_{n=a}^{b} f(n)$

2) Produce ***i*** = ***η*** * ***I***, ***η*** is random number form [0,1)

3) Find greatest ***r*** for $\sum_{n=a}^{r} f(n) < i$ where ***r*** is our result

Problem with this solution is that it creates "Choppy" distribution,and if we want to fill gaps between two adjacent elements ***k,l*** ∈ {a,...,b} so that their f(x) wold be ,for example, linearly changing, we have to add more samples between k,l , what add more iterations.

We can deal with "choppynes" by adding more samples which would reduce "choppynes", but not in all cases, or by replacing gaps with some function that is easy to integrate which would remove "choppynes" completely.

1 – LS : Light Sourcer

2 – SPD: Spectral Power Distribution

Example of replacing gaps with linear function:

To find our function $g_n(x)$ for gap filling between A and B we will use this formula $\mathbf{y - y_1 = m*(x-x_1)}$. In our case, $\mathbf{m = (f(B) - f(A))/(B-A)}$, $\mathbf{y_1 = f(A)}$ and $\mathbf{x_1 = A}$.

y - f(A) = m*(x – A) is y = $\frac{f(B)-f(A)}{B-A}*(x-A)+f(A)$ let c1 = (f(B) – f(A))/(B-A), c2 = (f(B)*A + f(A)*B)/(B-A) then **g_n(x) :y = c1*x – c2, x ∈ <A,B)** and $\int g_n(x)=c1\int x dx-c2\int 1\, dx=\frac{(c1)}{2}*x-c2*x=x*(\frac{(c1)}{2}-c2)$.

Now we want to find **r** from $\int_a^r \sum_{n=a}^b g_n(x)$ *. We can do similar thing as in beginning (but with I equals sum of all g_n() integrated), find greatest **r1** that sum of all integrals of $g_n(x)$ on each ones <A,B) is less than **i**, and solve $\int_A^{r_2} g_n(x)=i-r_1*I$ what is $r_2*(\frac{c_1}{2}-c_2)-A*(\frac{c_1}{2}-c_2)=i-r_1*I$ where $r_2=\frac{(i-r_1*I)}{(\frac{c_1}{2}-c_2)}+A$ so $\mathbf{r = r_1 + r_2}$.

For highest sped on dense sets, we can precompute all integrals and constants, then just quickly retrieve r_1 and constants of next g_n and calculate r_2.

1 – LS : Light Sourcer

2 – SPD: Spectral Power Distribution

6 Renderer

Right-handed system, x – is horizontal (bottom of monitor), y is vertical, z is depth (exiting from monitor towards viewer is positive value, entering into monitor is negative value)

It will most likely be bidirectional path tracer, with SPD and Power as color representation which enables easy use of known light interaction rules and thus should result in easily created photo-like images.

Renderer will be composed of following elements:

Scene

Contains SceneObjects, Cameras (CameraManager) and lights (LightSourceManager), manages light/camera path tracing.

SceneObject

Something that can interact with beams (light/camera). Provides ray intersection function, which returns new beam using relevant material function. Scene should contain SceneObjectManager containing all SO.

Camera

Can create image from detected beam data, Can create camera beams, is aware of “master light source” - LightSourceManager

Scene should contain CameraManager, CameraManager contains all other sensors (Cameras)

Light source

Can create light beams with some color data (wavelength, power), knows how many beams created.

Scene should contain 1 LightSourceManager, which contains all other LS

Materials

Have material function which takes beam and returns new one

1 – LS : Light Sourcer
2 – SPD: Spectral Power Distribution