RP1 – what was done

From 24.10.2017 – 11.12.2017

1 Light Sources

1.1) LightSource.java

```
public abstract class LightSource implements java.io.Serializable{

        public LightSource(SpectralPowerDistribution spd){...}

        public SpectralPowerDistribution getSpectralPowerDistribution(){...}

        public void setSpectralPowerDistribution(SpectralPowerDistribution spd){...}

        public double getNumberOfBeams(){...}

        public abstract float[] getNextBeam();

        public class Beam{      public float[] n;
                                public LightSource origin;
                                public Beam(float[] n, LightSource ls){...}
                        }

        public abstract Beam getNextBeamC();

}
```

This is base class for every Light source. It is serializable so when you create your own light source, you can use it as ,,plugin".

The constructor currently has 1 parameter - SpectralPowerDistribution spd. SPD is good representation of color spectrum (wavelengths) of LS.

Most important function is - public abstract float[] getNextBeam();, which has to be override by children. It should **return vector *V[6]*** (with length of 6). **V[0,1,2] is source position of beam**, **V[4,5] are angle of beam to X axis** (in XY pane) **and** angle to **Y axis** (int YZ pane). **V[6] is lambda** (wavelength) of this beam obtained from SPD.

Function getNumberOfBeams() returns how many times was getNextBeam() called.

Class Beam is here **so we can output getNextBeam() + reference to LS via Beam getNextBeamC().** *(which does not need to be abstract ..., it just returns new Beam(getNextBeam(), this); , it could also do beam++)*

1.2) UniformPointLightSource.java

```
public class UniformPointLightSource extends LightSource{

        public UniformPointLightSource(SpectralPowerDistribution spd, float[] position){...}

        public float[] getNextBeam(){...}

        public float[] getPosition(){...}

        public void setPosition(float[] position){...}

}
```

This is **simplest LS.** It **shines in all direction with same intensity**. *In future referred to as UPLS.*

The constructor currently has 2 parameter - SpectralPowerDistribution spd and float[] position. SPD is here because of *LightSource* Constructor. **position is** array of 3 floats(will be changed to doubles) which represents **[x,y,z] coordination in 3D** space – position of this LS.

Function public float[] getNextBeam() returns vector V as described in *LightSource*. *V[0,1,2]* is same as position, *V[3,4]* = [rndrAX.nextFloat() * 360, rndrAY.nextFloat() * 360], where rndrAX and rndrAY are 2 separate instances of java.util.Random() . *V[5]* = spd.getNextLamnbda().

This LS is tested in /test/upls/Test.java , and it appears that it woks as described.

1.3) SimpleSpotLight.java

```
public class SimpleSpotLight extends LightSource{

        public SimpleSpotLight(SpectralPowerDistribution spd, float[] position, float cone_direction[],
                                        float cone_angle){...}

        public float[] getNextBeam(){...}


        public float[] getPosition(){...}       public void setPosition(float[] position){...}

        public float[] getDirection(){...}      public void setDirection(float[] direction){...}

        public float getAngle(){...}            public void setAngle(float angle){...}

}
```

This LS **is** works as **Spot Light** (directed light – it creates cone of light), **with** no fading, which means that it have **same intensity in all directions within the cone**. *In future referred to as SSL.*

This **constructor** works pretty much **same as UPLS, with 2 more parameters**. **cone_direction** **is** directional **vector which describes what direction is center of light cone pointed to.** **cone_angle** **is angle from center of light cone.** It *2 is total light cone angle.

getNextBeam() works same as in UPLS, but V[3,4] is obtained by generating angles in [0,cone_angle*2) and them adding them to angles generated from cone_direction - cone_angle.

This LS is tested in /test/ssl/TestSSL.java , and it appears that it woks as described.

1.4) **FadingSpotLight.java**

```
public class FadingSpotLight extends SimpleSpotLight{

        public FadingSpotLight(SpectralPowerDistribution spd, float[] position, float cone_direction[],
                                        float cone_angle, double fade_per_angle){...}

        public float[] getNextBeam(){...}

        public double getFade(){...}        public void setFade(double fade){...}

}
```

This LS **works as SSL**, **but i looses intensity** as you get further from cone_direction, by fade_per_angle% per angle. *In future referred to as FSL.*

Constructor **parameter fade_per_angle describes how much less beams should be genereated in directions further from cone_direction per angle.**

getNextBeam() works same as in SSL, but generating angles in [0,cone_angle*2) uses nonuniform distribution created from cone_angle **and** fade_per_angle**, as described in Rp.pdf chapter 5.**

This LS is tested in /test/fsl/TestFSL.java , and it appears that it woks as described.

1.5) CircleLight.java

```
public class CircleLight extends LightSource{

        public CircleLight(SpectralPowerDistribution spd, float[] position, float direction[], float radius)
                                { ...}

        public float[] getNextBeam(){...}

                                                        .

                                                        .

                                                        .
}
```

This is currently the most advanced LS. **It is circle** in pane (so it cant have bumbs or dents) witch shines in all direction with same intensity in 90 degrees around direction. In other words, **it shines from within its circle from 1 side**. *In future referred to as CL.*

Constructor has 3 special parameters. **position** – which is center of this cirlce, **direction** - where it is pointed and **radius** – which is this circle radius. **This constructor creates SSL with position , direction and it uses constant 90 as cone_angle. So it is like SSL, but instead of shining from 1 point, it shines from within the circle.**

getNextBeam() calls SSL.getNextBeam() and then it changes it like this. It makes perpendicular vector to direction, and then it multiplay it by random number from (0,radius]. We multiply this by position which gives us *[x′,y′,z′]* within the circle. Then we rotate *[x′,y′,z′]* around direction by random angle. *V[0,1,2]* obtained from SSL is replaced by *[x′,y′,z′]* and *V* is returned.

This LS is tested in /test/cl/TestCL.java , and it appears that it woks as described.

2 SpectralPowerDistribution

2.1) SpectralPowerDistribution.java

In future referred to as SPD.

SPD is used for generation of wavelengths for **getNextBeam()** functions.

```
public interface SpectralPowerDistribution {

        public double getNextLamnbda();

        public double getValue(double lambda);

        public double[] getFirstLastZero();

}
```

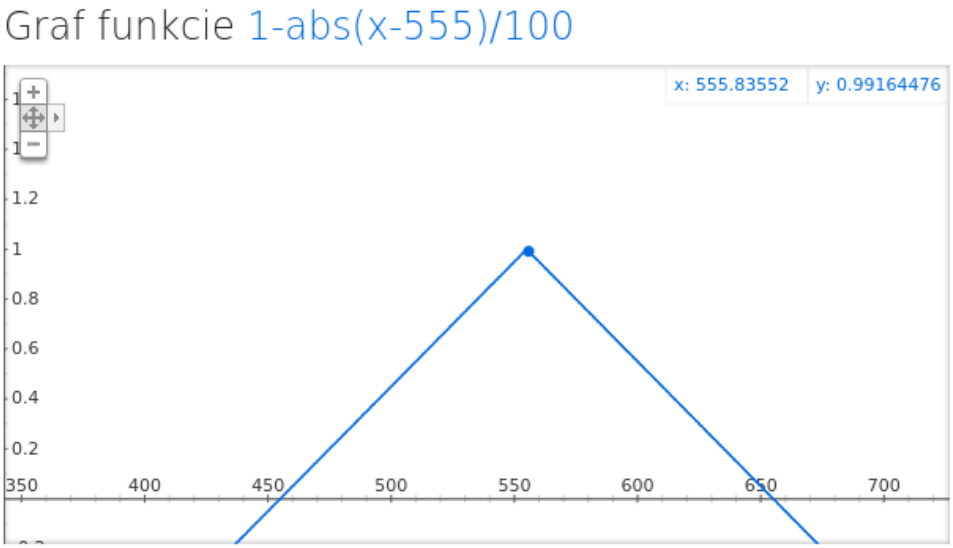
The key function for wavelengths generation in **getNextBeam()** **is** **getNextLamnbda()**. It should return some number which can be used as beam wavelength and is accepted by your *Color* implementation.

Functions **getValue(double lambda)** (return energy on that wavelength) and **getFirstLastZero()** (marginal wavelengths with no energy) **are used for color creation.**

How are SPD data stored are up to each implementations.

2.2) SPD1.java

Is implementation of SPD which graph looks something like



and **getFirstLastZero() returns [450,650].**

This SPDs **getNextLamnbda() returns lambdas using Nonuniform random distribution** created **from** its **graph**.

3 Color

3.1) Color.java

```
public interface Color {

	public int[] SPDtoRGB(SpectralPowerDistribution spd);

}
```

It has 1 function - SPDtoRGB, which must be able to convert SPD to RGB color.

3.2) CIE1931StandardObserver.java

This is implementation of CIE standard observer for CIE 1931 color space.

The observer data were taken from http://jcgt.org/published/0002/02/01/ and the SPDtoRGB was made according to http://www.brucelindbloom.com/index.html?Eqn_XYZ_to_xyY.html .

Using this with SPD1 results in very blue color.

4 Renderer

There is attempt to make camera, but it is not clear how ...