A11 – GLSL lights

In this assignment, you have to write the code for determining the light direction and color for 7 different illumination models. The application contained in `index.html`, has a user interface similar to the one of Example 04. Depending on the requested shader, the interface below changes to allow the user playing with the elements specific for the selected model. Shader code to compute the final pixel color should be written in file `shaders.js`. The code, written in GLSL, can use the following variables to find the elements necessary to compute the final color:

The shader can find the required information in the following variables:

```
vec3 fs_pos;                // Position of the point in 3D space

vec3 Pos;                   // Position of the light (if not direct)
vec3 Dir;                   // Direction of the light (if not point)
float ConeOut;              // Outer cone (in degree) of the light (if spot)
float ConeIn;               // Inner cone (in percentage of the outher cone) of the light (if spot)
float Decay;                // Decay factor (0, 1 or 2)
float Target;               // Target distance
vec4 lightColor;            // color of the light

vec4 ambientLightColor;     // Ambient light color. For hemispheric, this is the color on the top
vec4 ambientLightLowColor;  // For hemispheric ambient, this is the bottom color
vec3 ADir;                  // For hemispheric ambient, this is the up direction
vec4 SHconstColor;          // For spherical harmonics, constant term
vec4 SHDeltaLxColor;        // For spherical harmonics, ΔLx color
vec4 SHDeltaLyColor;        // For spherical harmonics, ΔLy color
vec4 SHDeltaLzColor;        // For spherical harmonics, ΔLz color

vec3 normalVec;             // direction of the normal vector to the surface
```

The light direction is returned into:

```
vec3 OlightDir;
```

and intensity is returned into:

```
vec4 OlightColor;
```

ambient light contribution is returned into:

```
vec4 ambientColor;
```

The following GLSL standard procedure can be helpful in solving this exercise:

normalize()
cos()
radians()
pow()
dot()
length()
clamp()
reflect()
max()

whose reference can be found at pages 7 and 8 of the official WebGL reference card from Kronos Group (`webgl20-reference-guide.pdf`, enclosed in this ZIP file for convenience).

GLSL will be presented in depth during the exercise sessions later in the course. However it has a syntax that is very similar to Javascript or C. Its main feature is having the possibility of operating on vectors and matrix with common math operators, and to extract vector components with the ".dot" notation (es: `normalVec.x` to denote the `x` component of the vector contained in variable `normalVec`). At this point, if you cannot figure out the syntax yourself, and you do not want to wait to have it covered in the course, you can refer to the following tutorial:

https://cgvr.cs.uni-bremen.de/teaching/cg2_07/literatur/glsl_tutorial/index.html

Starting from the section "Data Types and Variables", at around 1/3 of the page. Please ignore what is presented before since it refers to a very old version of OpenGL which uses concepts that are now deprecated and not valid for WebGL.

To check the correctness of your shader, you can visually compare your results with the ones that can be obtained by *Example E04*, available on the Beep page of this course, after properly setting the same type of lights and materials.

The first two light models have been implemented to guide you in writing the code