A09 – Make the objects

In this assignment, you have to write the code for solving the rendering equation for 7 different illumination and BRDF 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 informations in the following variables:

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

vec3 LAPos;                 // Position of first (or single) light
vec3 LADir;                 // Direction of first (or single) light
float LAConeOut;            // Outer cone (in degree) of the light (if spot)
float LAConeIn;             // Inner cone (in percentage of the outher cone) of the light (if spot)
float LADecay;              // Decay factor (0, 1 or 2)
float LATarget;             // Target distance
vec4 LAlightColor;          // color of the first light

vec3 LBPos;                 // Same as above, but for the second light
vec3 LBDir;
float LBConeOut;
float LBConeIn;
float LBDecay;
float LBTarget;
vec4 LBlightColor;

vec3 LCPos;                 // Same as above, but for the third one
vec3 LCDir;
float LCConeOut;
float LCConeIn;
float LCDecay;
float LCTarget;
vec4 LClightColor;

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

float SpecShine;            // specular coefficient for both Blinn and Phong
float DToonTh;              // Threshold for diffuse in a toon shader
float SToonTh;              // Threshold for specular in a toon shader

vec4 diffColor;             // diffuse color
vec4 ambColor;              // material ambient color
vec4 specularColor;         // specular color
vec4 emit;                  // emitted color

vec3 normalVec;             // direction of the normal vecotr to the surface
vec3 eyedirVec;             // looking direction
```

Final color is returned into:

```
vec4 out_color;
```

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).

To check the correctness of your shader, you can visually compare your results with the ones that can be obtained by *Example E04ex*, available on the Beep page of this course, after properly setting the same type of lights and materials.