Introduction.

My project consits into the design, the implementation and the testing of a visual editor for realizing *3D* graphics with `WebGL`.

This developed tool is very useful, in fact it permits to every one to use `WebGL` without knowing any programming rule.

I named it `WGLEditor`, `WebGL` Graphics Layers–based `Editor`.

The name derives to the fact that I divided the features of a *3D* graphic into 4 categories, or layers:

– The *Structure Layer*, that includes all geometrical properties (e.g., *hight*, *width*, *diameter*, *tessellation*) of a mesh, its linear transformations (translation, rotation and scaling) and its hierarchy relationships;

– The *Appearance Layer*, that includes each property concerning the look of a mesh: its, material, the 4 fundamental colors (ambient, diffuse, specular and emissive), the texturing, the wireframe effect, the shadowing, the shading, the visibility, etc.

– The *Motion Layer*, that includes both the animation and the physical mechanisms.

– The *Scene And User Interaction Layer*, that includes the setting, namely: the lights, the camera, the panorama (by means the so called sky–box), the background color, the ambient color, the the shadowing mechanism, etc.

The `WGLEditor` scheme: an overview..

In **Figure 1** the block diagram of `WGLEditor` is represented.

What is represented between the client and the server entities is the client–side of the developed system, that consists into a smart *GUI* mostly based on `Bootstrap`[1] and some derived projects found on `GitHub`.

Into the other side of the server icon are represented the 5 essential modules of the server–side part:

– *A `PostgreSQL` managing library*, that permits to store permanently the informations sent by the user and to extract them when required.

  The great relevance of this module consists into the fact that *it lets the informations employment be asynchronous by their production.*

– *A `BabylonJS` code producer library*, that uses the informations stored into the database to creating the `javascript` code, based onto the `BabylonJS` library[2]. Actually `BabylonJS` is just the main used library, in fact actually the produced code is based even onto other libraries: `CannonJS` per the physcal effects, `HandJS` for the camera and some further classes needed for the procedural texturing operations.

– *A file uplading managing library*, that permits to handle in a systematic and reusable way the uploading of the image files that the client sends for texturing purposes.

– *A tree data structure managing library.* In order to know and navigate the hierarchical relationships among meshes, `WGLEditor` uses a tree–like data structure that indexes the actual mesh objects stored into the database.

– A complete `BabylonJS` documentation, composed by 55 pages, that shows all details about what is the code produced by the above mentioned *`BabylonJS` code producer library*.

1. A very powerful `css` library. For more details please refer to this source:
http://getbootstrap.com/

2. Avaliable here:
https://doc.babylonjs.com/

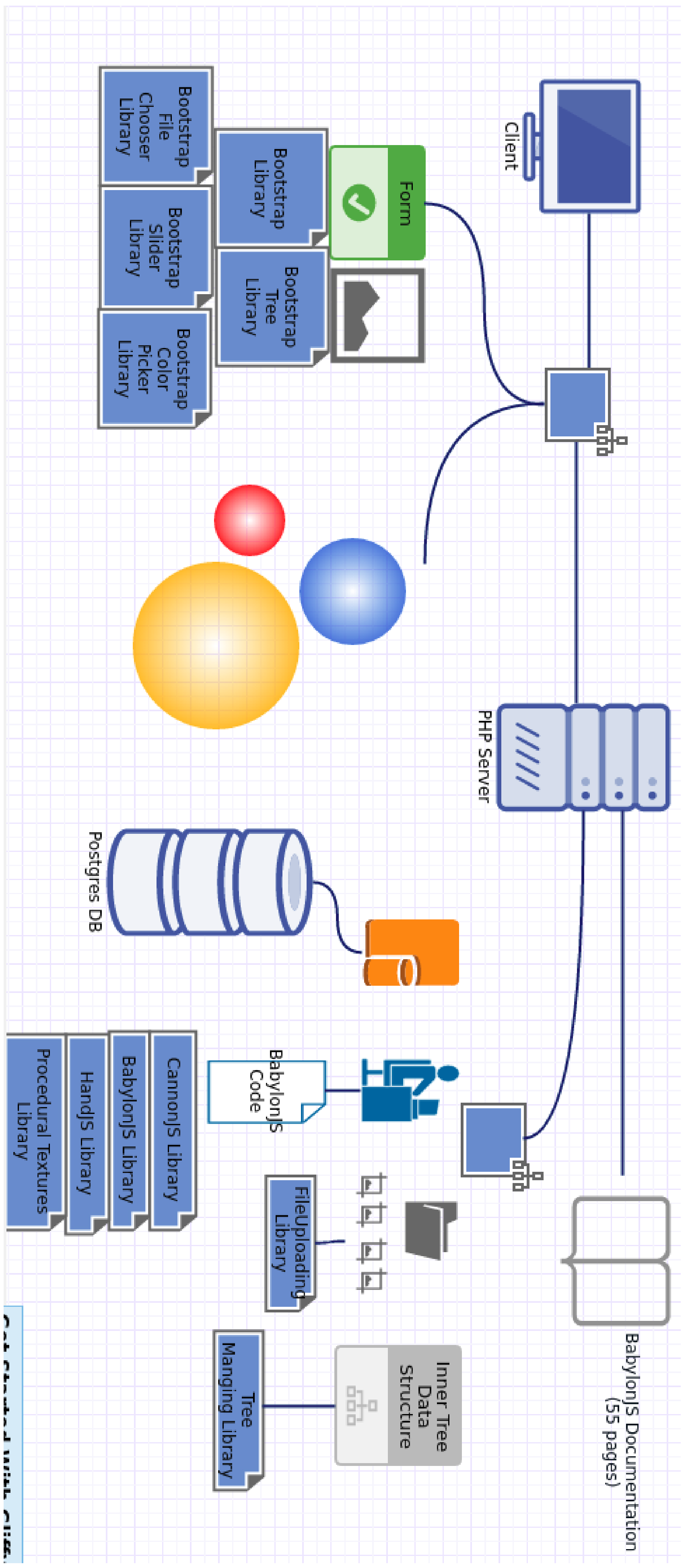

Client
Form
Bootstrap Library
Bootstrap Tree Library
Bootstrap File Chooser Library
Bootstrap Slider Library
Bootstrap Color Picker Library
PHP Server
Postgres DB
BabylonJS Code
CannonJS Library
BabylonJS Library
HandJS Library
Procedural Textures Library
FileUploading Library
Tree Manging Library
Inner Tree Data Structure
BabylonJS Documentation (55 pages)

Figure 1.

Focus on the client side.

In **Figure 2** the banner, the canvas area and the notices strip of the application *GUI* are represented.

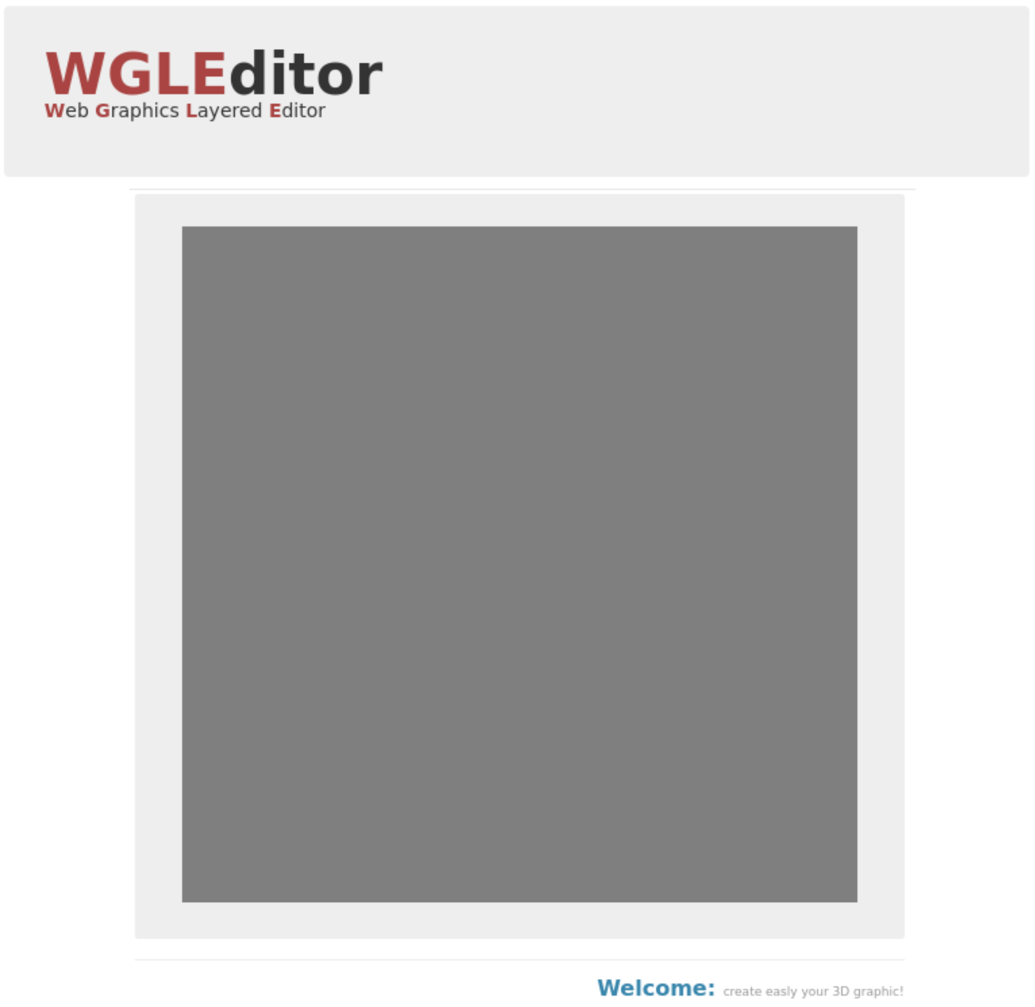

WGLEditor
Web Graphics Layered Editor
Welcome: create easly your 3D graphic!

Figure 2.

After them there are 3 sliding panels containing respectively:

– the meshes data sheet ballot, a form that permits to add, review and update all the set details of a such mesh.

  In more details it is composed by a tabbed pane composed by three tabs, respectively for the *Structure Layer* properties, the *Appearence Layer* ones and the *Motion Layer* ones.

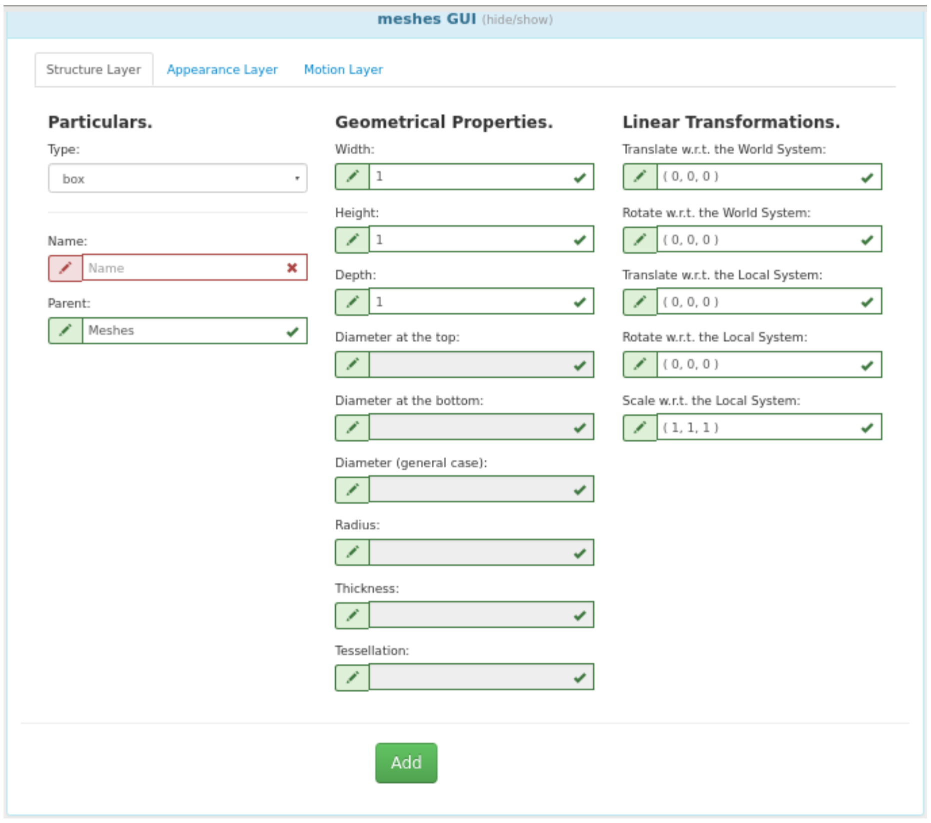
  All this is shown in the **Figure 3**, **Figure 4** and **Figure 5**.

– the scene navigator, a nice representation of the tree–like data structure managed by the above mentioned library. It is represented in **Figure 6**.

– the scene data sheet ballot, a form that permits to update and/or review all the set details of the scene.

  In more details it is composed by a tabbed pane composed by three tabs, respectively for the panorama properties, the ligth sources ones and the camera ones. Namely this ballot covers the remaining layer.

All this is shown in the **Figure 7**, **Figure 8**, **Figure 9** and **Figure 10**.



Particulars.

Type: box

Name: Name

Parent: Meshes

Geometrical Properties.

Width: 1

Height: 1

Depth: 1

Diameter at the top:

Diameter at the bottom:

Diameter (general case):

Radius:

Thickness:

Tessellation:

Linear Transformations.

Translate w.r.t. the World System: (0, 0, 0)

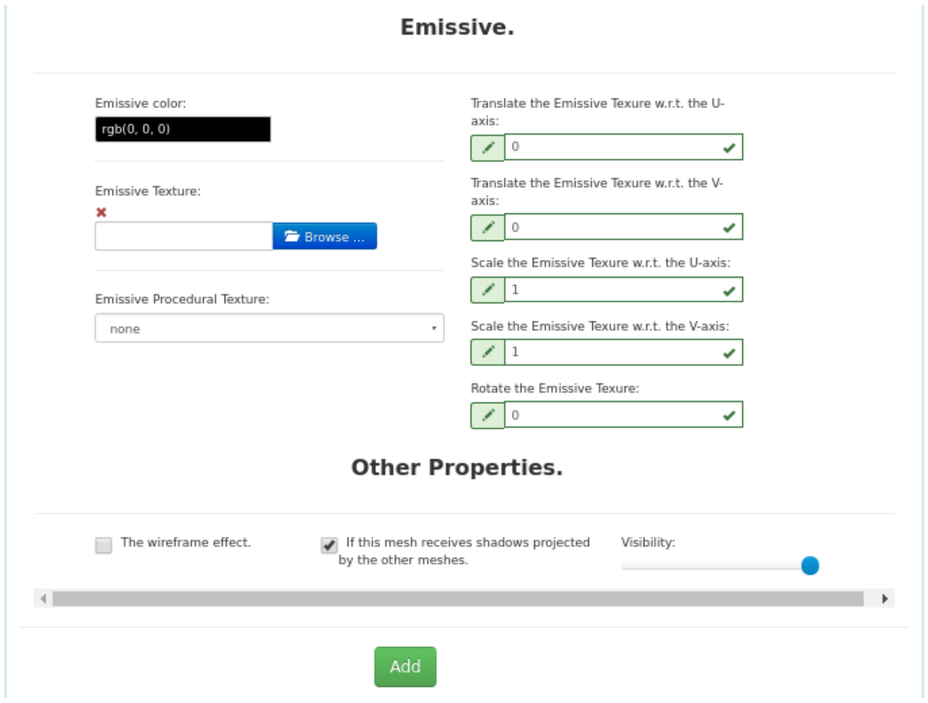
Rotate w.r.t. the World System: (0, 0, 0)

Translate w.r.t. the Local System: (0, 0, 0)

Rotate w.r.t. the Local System: (0, 0, 0)

Scale w.r.t. the Local System: (1, 1, 1)

Figure 3.



Emissive.

Emissive color: rgb(0, 0, 0)

Emissive Texture:

Emissive Procedural Texture: none

Translate the Emissive Texure w.r.t. the U-axis: 0

Translate the Emissive Texure w.r.t. the V-axis: 0

Scale the Emissive Texure w.r.t. the U-axis: 1

Scale the Emissive Texure w.r.t. the V-axis: 1

Rotate the Emissive Texure: 0

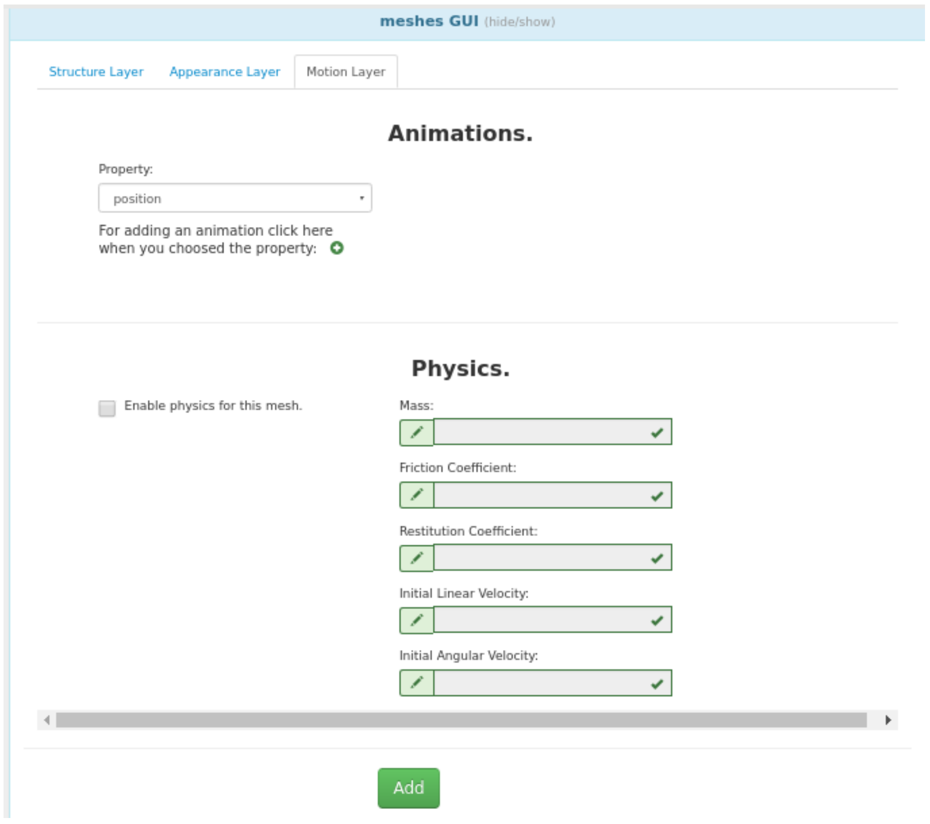
Other Properties.

The wireframe effect.

[✓ If this mesh receives shadows projected by the other meshes.]

Visibility:

Figure 4.



meshes GUI (hide/show)

Structure Layer | Appearance Layer | Motion Layer

Animations.

Property: position

For adding an animation click here when you choosed the property:

Physics.

Enable physics for this mesh.

Mass:

Friction Coefficient:

Restitution Coefficient:

Initial Linear Velocity:

Initial Angular Velocity:

Figure 5.

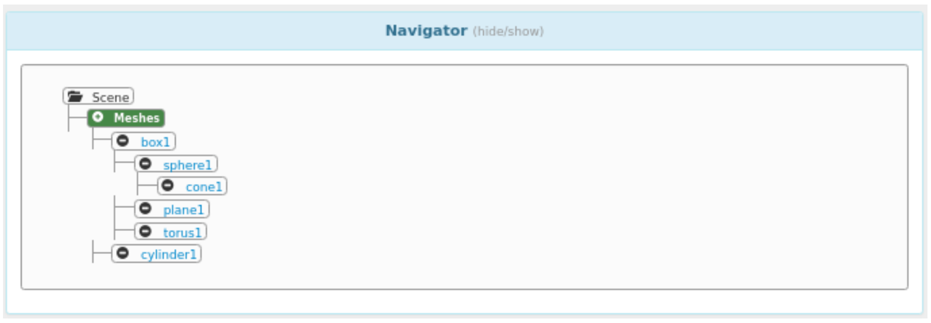


Figure 6.

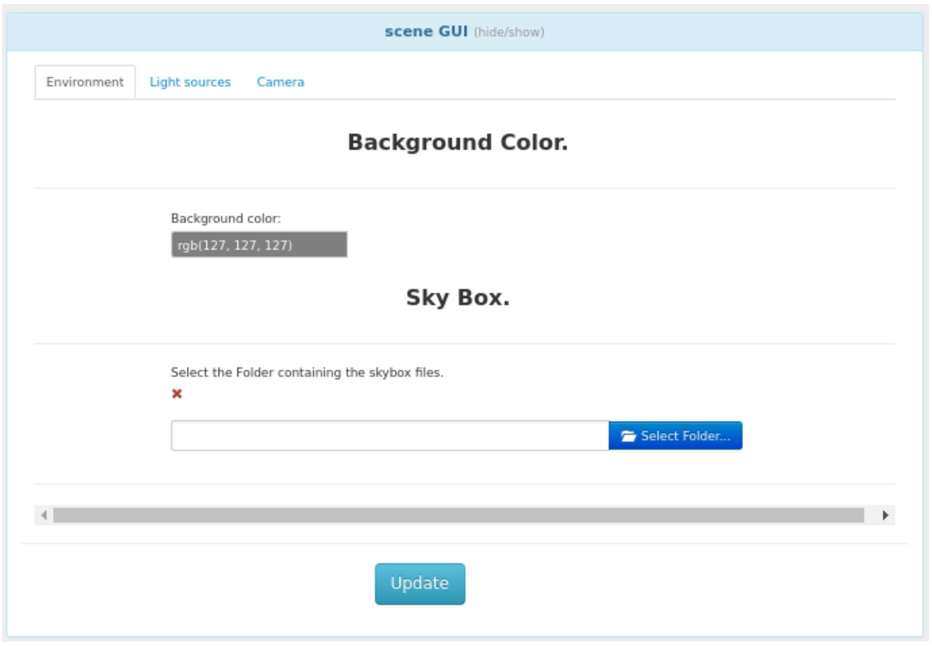
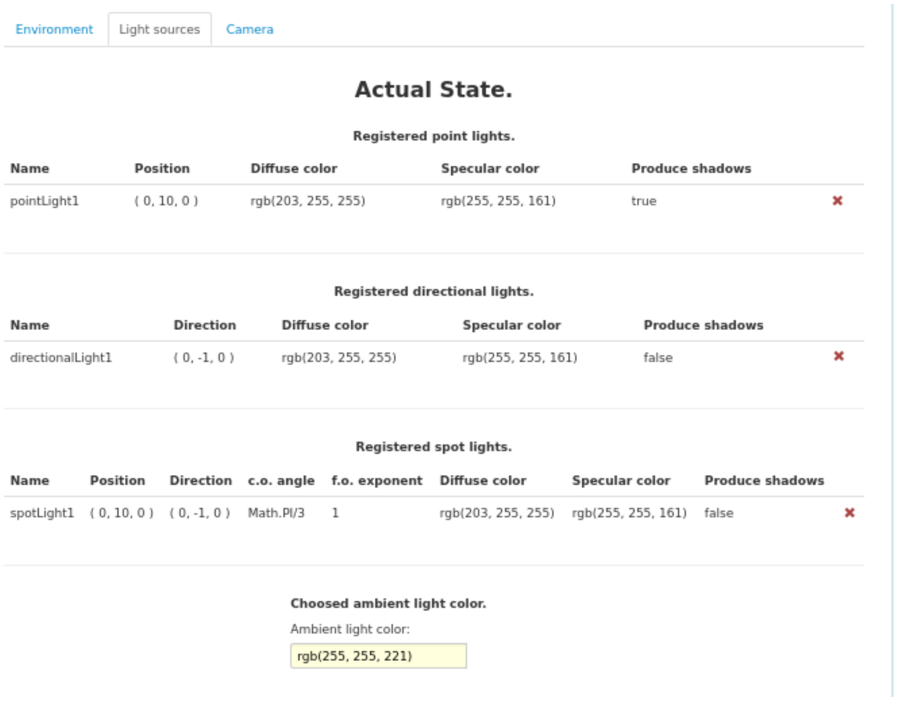


Figure 7.



Environment | Light sources | Camera

Actual State.

Registered point lights.

Name	Position	Diffuse color	Specular color	Produce shadows	
pointLight1	(0, 10, 0)	rgb(203, 255, 255)	rgb(255, 255, 161)	true	✖

Registered directional lights.

Name	Direction	Diffuse color	Specular color	Produce shadows	
directionalLight1	(0, -1, 0)	rgb(203, 255, 255)	rgb(255, 255, 161)	false	✖

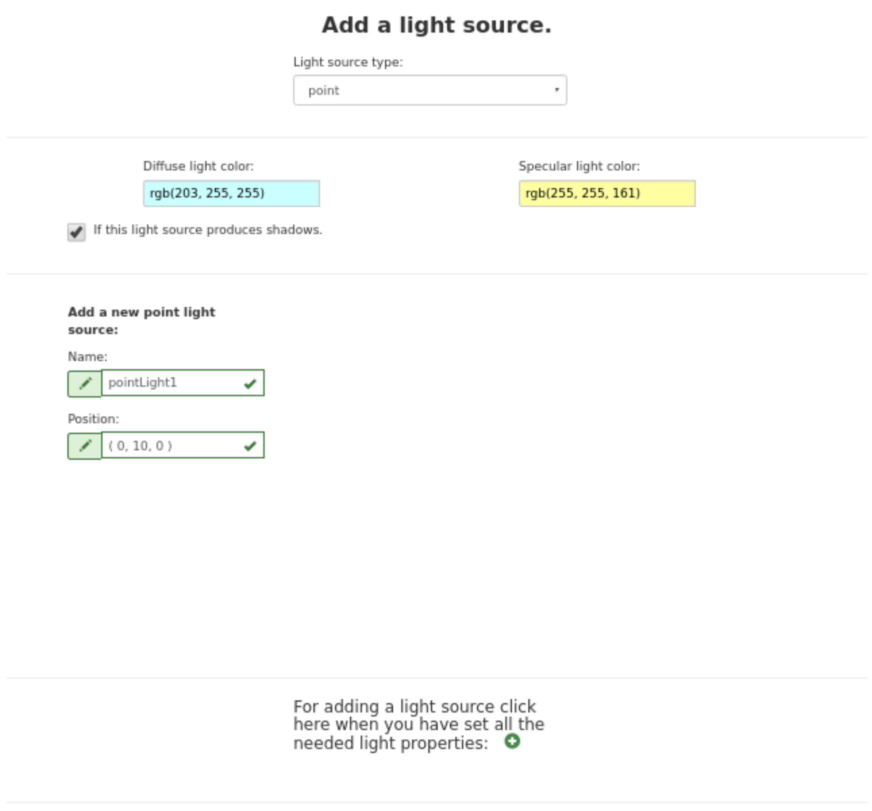
Registered spot lights.

Name	Position	Direction	c.o. angle	f.o. exponent	Diffuse color	Specular color	Produce shadows	
spotLight1	(0, 10, 0)	(0, -1, 0)	Math.PI/3	1	rgb(203, 255, 255)	rgb(255, 255, 161)	false	✖

Choosed ambient light color.

Ambient light color: rgb(255, 255, 221)

Figure 8.



Add a light source.

Light source type: point

Diffuse light color: rgb(203, 255, 255)

Specular light color: rgb(255, 255, 161)

✓ If this light source produces shadows.

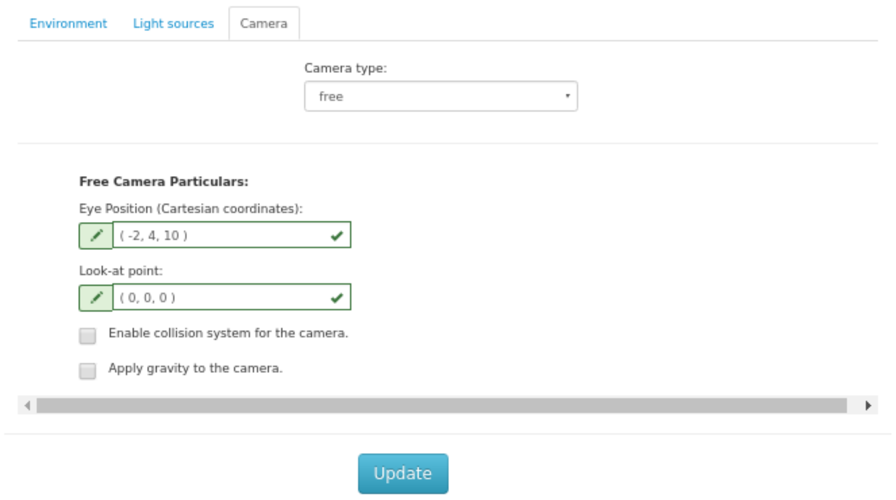
Add a new point light source:

Name: pointLight1

Position: (0, 10, 0)

For adding a light source click here when you have set all the needed light properties:

Figure 9.



Camera type: free

Free Camera Particulars:

Eye Position (Cartesian coordinates): (-2, 4, 10)

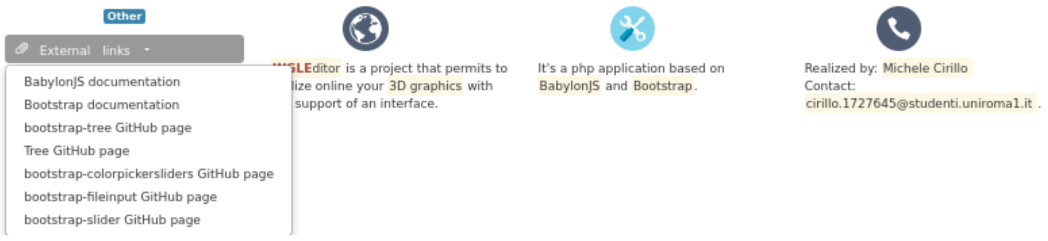
Look-at point: (0, 0, 0)

Enable collision system for the camera.

Apply gravity to the camera.

Figure 10.

Finally there is the footer, shown in the **Figure 11**.



BabylonJS documentation
Bootstrap documentation
bootstrap-tree GitHub page
Tree GitHub page
bootstrap-colorpickersliders GitHub page
bootstrap-fileinput GitHub page
bootstrap-slider GitHub page

GLEditor is a project that permits to lize online your 3D graphics with support of an interface.

It's a php application based on BabylonJS and Bootstrap.

Realized by: Michele Cirillo
Contact: cirillo.1727645@studenti.uniroma1.it .

Figure 11.

Focus on the server side.

Please refer to the above mentioned documentation of 55 pages.

In it there are all the `BabylonJS` codes reproduced bt `WGLEditor`.

An explicative example.

In **Figure 12** all the mesh types provided by `WGLEditor` are represented.

In **Figure 13** are shown the effects of the addition to the scene of three lights – a point one put at $(0, 10, 0)$, a directional one directed along $(0, -1, 0)$ and a spot one directed along $(0, -1, 0)$ – and enabling the shading and the shadowing processes.

In **Figure 14** all the texturing types provided by `WGLEditor` (in this case, all the possible ones!) and the wireframe effect are represented.

In **Figure 15** a better view is shown.

In **Figure 16** and **Figure 17** the effects of to apply the physics engine are shown.

The meshes, that now are physical objects and not just geometrical ones, fall onto the big plane (that doesn't fall because it has zero mass).

But, because the plane is not perfectly hrizontal and there isn't friction, the fallen object will glide until they'll reach the edge of the big plane and finally they'll fall into the oblivion.

In **Figure 18** are shown the effects of to apply the sky–box texture effect.

In **Figure 19** are shown the effects of to use the arc–rotate camera instead of the free one.

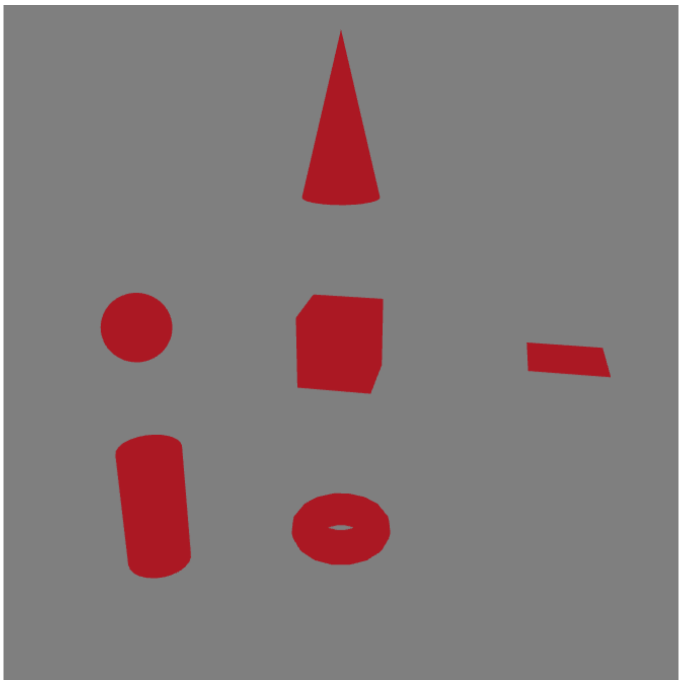


Figure 12.

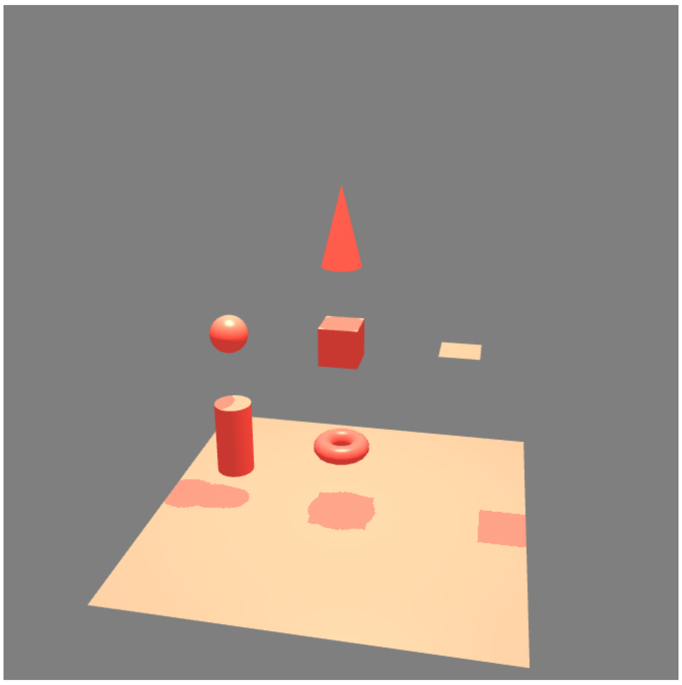


Figure 13.

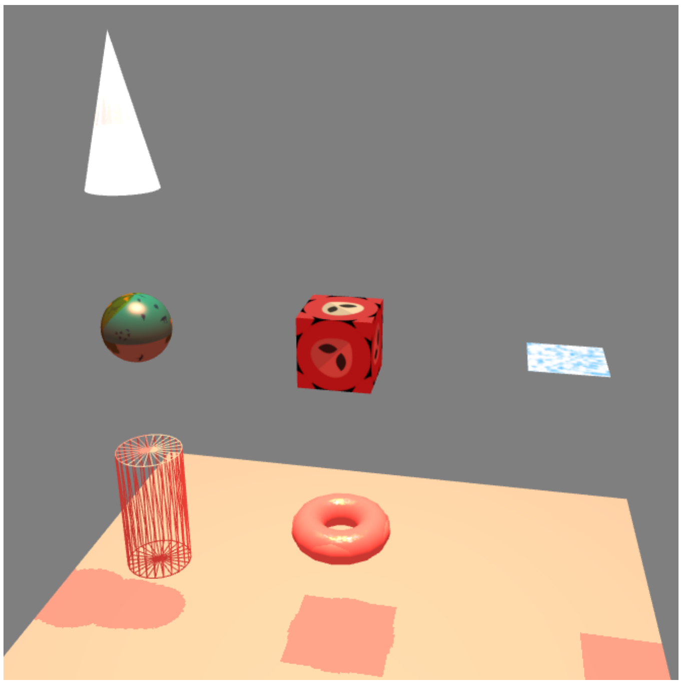


Figure 14.

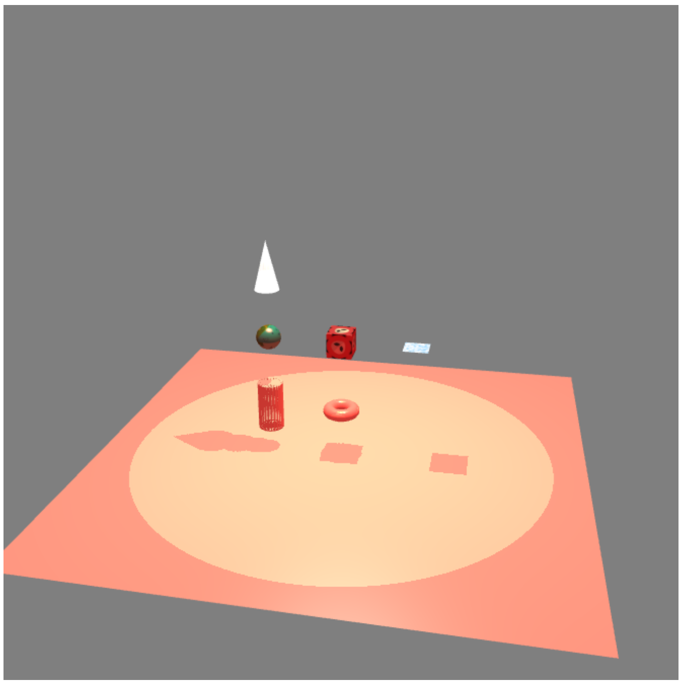


Figure 15.

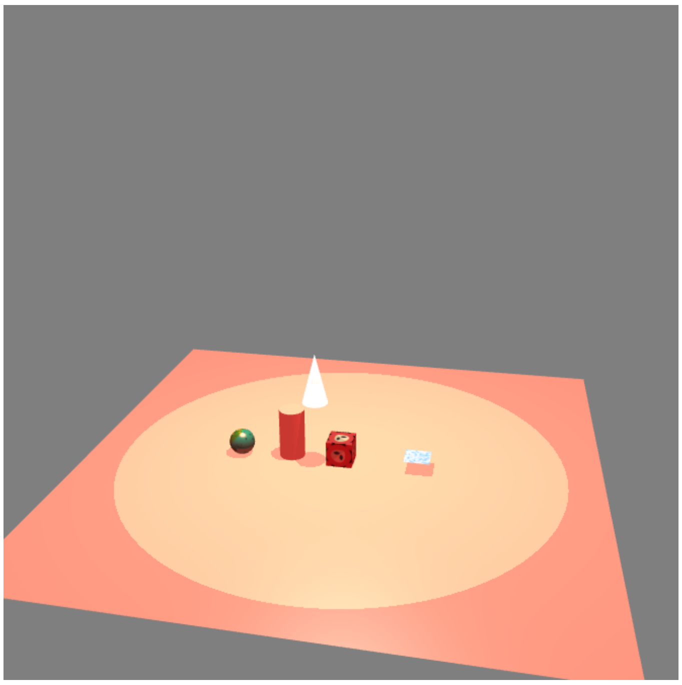


Figure 16.

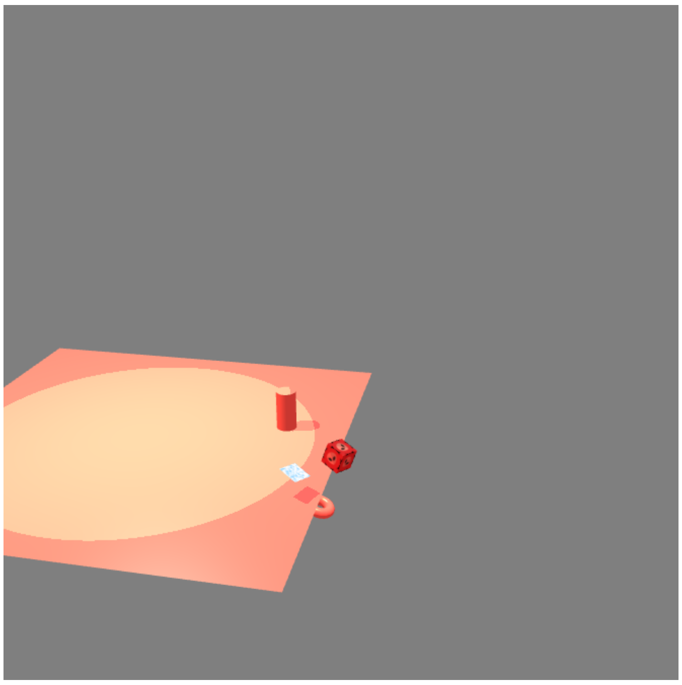


Figure 17.

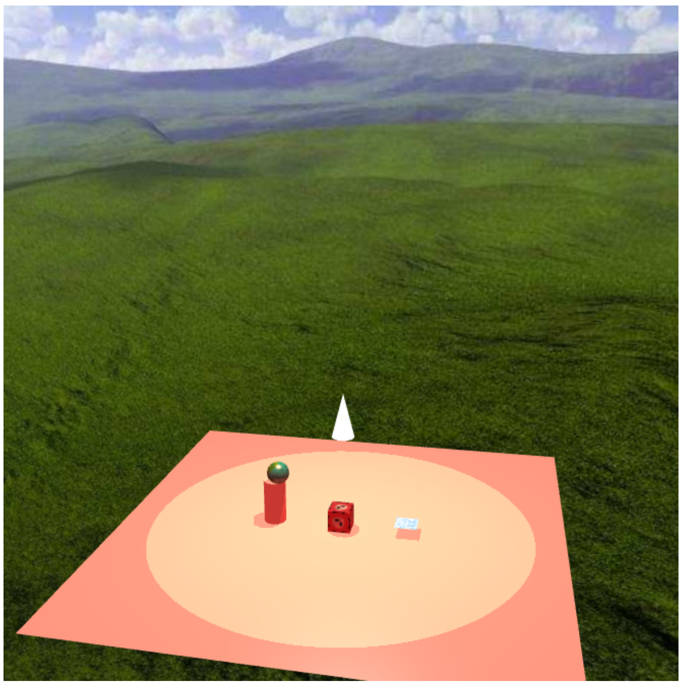


Figure 18.

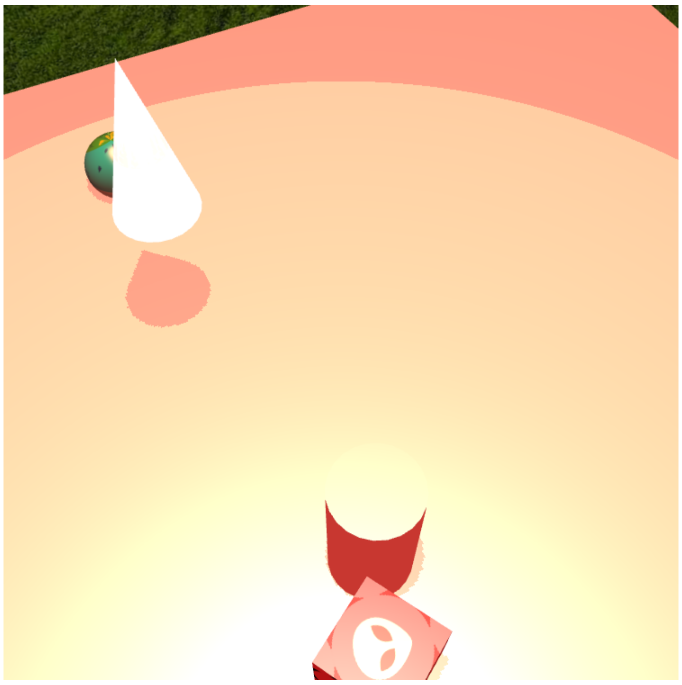


Figure 19.