Supplementary Materials: Getting started to QMwebJS

Accompanying documentation to: ”QMwebJS: An open source software tool to visualize and share time-evolving three-dimensional wavefunctions”

Edgar Figueiras, David Olivieri, Angel Paredes and Humberto Michinel

1 Introduction

In the first section of this document, a step-by-step guide is provided to show how to produce visualizations from 3D wavefunction simulations. In particular, we describe the simulation output formats and utilities for converting these files to an appropriate input for QMwebJS. For this, we illustrate a typical file convention that we use, but also indicate other alternatives. While QMwebJS could be used from an html file stored on a local computer, we describe its use from our free online site http://www.parvis3d.org.es/.

QMwebJS provides an intuitive graphical interface, both when manipulating 3D objects in the view-port canvas as well as using the side-panel control. Nonetheless, there are several options available that may not be immediately obvious. Thus, section 2 provides a brief user guide to the GUI control panel and relevant parameters as well as typical file handling.

It should be understood that QMwebJS is a javascript code. For use, it must be embedded within an html web page. For users that will only access our online visualizer (enabled with QMwebJS), it is not necessary to know anything about this; users would simply access the app through the web.

2 Detailed Workflow: How to Produce Visualizations

Figure 1 shows the principal steps of the data workflow for using QMwebJS. These steps are described in more detail in Supplementary Materials, but the essential points are the following:

- Simulate evolution of 3D wavefunction $|\psi|^2$; output are 3D matrices stored in binary files, one for each time point t,
- Use these matrices for the particle sampling algorithm with the utility function, `particle_sample.py`; produces particle positions and associated $|\psi|^2$ that are stored in JSON format,
- Load the JSON to a QMwebJS enabled web page (for example our online site http://www.parvis3d.org.es/). After editing, models and images can be exported directly from the from client-browser

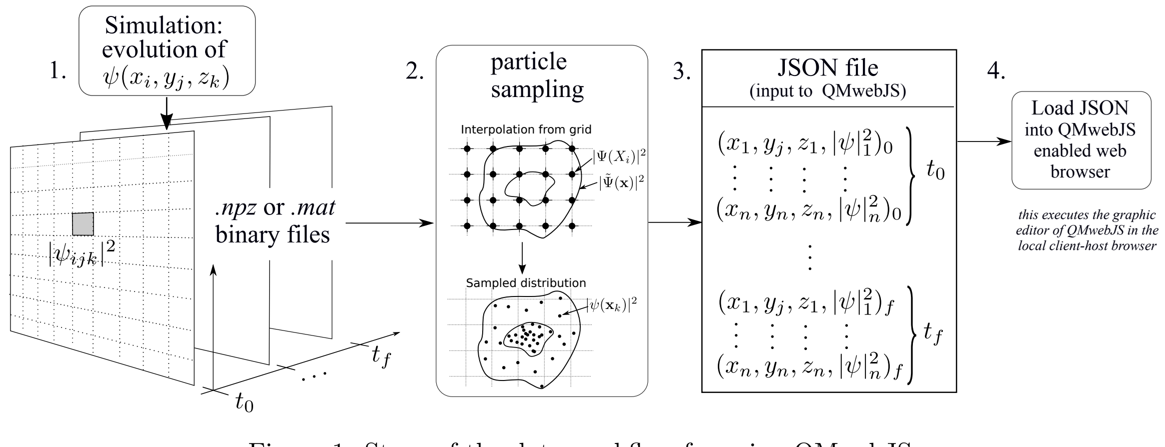

1.
Simulation: evolution of $\psi(x_i, y_j, z_k)$
$|\psi_{ijk}|^2$
.npz or .mat binary files
t_0
t_f
2.
particle sampling
Interpolation from grid
$|\Psi(X_i)|^2$
$|\tilde{\Psi}(\mathbf{x})|^2$
Sampled distribution
$|\psi(\mathbf{x}_k)|^2$
3.
JSON file
(input to QMwebJS)
$(x_1, y_j, z_1, |\psi|^2_1)_0$
$(x_n, y_n, z_n, |\psi|^2_n)_0$
t_0
$(x_1, y_j, z_1, |\psi|^2_1)_f$
$(x_n, y_n, z_n, |\psi|^2_n)_f$
t_f
4.
Load JSON into QMwebJS enabled web browser
this executes the graphic editor of QMwebJS in the local client-host browser

Figure 1: Steps of the data workflow for using QMwebJS.

It should be stressed that all loading, processing, and graphical editing is performed in the client-browser. In fact, the web page does not need to be connected to the Internet. The loading of the JSON data is stored in memory on the client-browser.

Step 1: Output preparation for the Simulator To illustrate an output format we typically employ, we consider the simulation example provided in the manuscript for Hydrogen decay. For this, we implemented a python script `schrodHdecay.py` and a Matlab script `Hydrogen_decay.m` for integrating the time evolution of $\psi|^2$.

For our purpose here, all the details of the simulation are not relevant, it is sufficient to show parts of the time evolution (the full scripts are available as accompanying files).

```
//.... more code not shown
    x=(nx-(float(Nx)/2.))*float(wwx)/float(Nx)
    y=(ny-(float(Ny)/2.))*float(wwy)/float(Ny)
    z=(nz-(float(Nz)/2.))*float(wwz)/float(Nz)

    //....  code not shown
    self.X,self.Y,self.Z=np.meshgrid(x, y, z)
    //...  (more code not shown)
    t=0
    tc=1000 # arbitrary time constant, as explained above
    ampl = 0.
    for qq in range(nsteps):
        Initial_state   = np.exp(-1j*tc*E3*t)* np.exp(-t)*psi322
        Final_state = np.exp(-1j*(E2*tc*t+initphase)) * np.sqrt(1.-np.exp(-2*t))*psi210
        f =  Final_state + Initial_state
        psisqr = 1e2 * np.abs(f)**2
    if np.mod(qq,nstepsplot)==0:
            np.savez(out_dir + '/psi_'+ countstr + '.npz',  psi=psisqr)
            cnt+=1
        t+=dt
```

The Matlab code segment would be something like the following (for the Hydrodgen problem).

```
%  ... more code
t=[0:dt:tmax];
tc=1000;      % arbitrary time constant, as explained above
counter_for_psi=0;
for j=1:nsteps   % propagation loop
    if mod(j-1,nstepsplot)==0
    initphase=pi;
     f=exp(-1i*tc*E3*t(j))*exp(-t(j))*psi321
               +exp(-1i*(E2*tc*t(j)+initphase))*sqrt(1-exp(-2*t(j)))*psi210;
    psi2_evolution= abs(f).^2;
    save(psi_output_file, 'psi2_evolution');
end
```

As can bee seen in both cases, the output format at time t is the 3D cube of values $|\psi|^2$ defined on the mesh points x, y, z. In this case, one file is saved for each time point. In the case of Python, the output format is a binary Numpy file (we give extension .npz), while the output for Matlab is the standard binary file (we assign a .mat extension)

An example of the collection of output files from the Python simulation would be as follows.

```
(base):~//hdecay/resdata$ ls
psi001.npz  psi041.npz  psi081.npz  psi121.npz  psi161.npz  psi201.npz  psi241.npz
psi006.npz  psi046.npz  psi086.npz  psi126.npz  psi166.npz  psi206.npz  psi246.npz
...
```

Step 2: Output preparation for the Simulator The next step is to use the values of $|\psi|^2$ from these files defined on the 3D mesh for each time to perform particle sampling. As described in the manuscript, this algorithm determines particle positions $\mathbf{r} = (x, y, z)$ by sampling from the $|\psi|^2$. Thus, each particle position is stored together with the closest value of $|\psi|^2$ to help provide coloring information (see main text). Therefore, the output matrix M at time t is given as:

$$M = \begin{bmatrix} x_{t1} & y_{t1} & z_{t1} & |\psi_{t1}|^2 \\ x_{t2} & y_{t2} & z_{t2} & |\psi_{t2}|^2 \\ \vdots & \vdots & \vdots & \vdots \\ x_{tn} & y_{tn} & z_{tn} & |\psi_{tn}|^2 \end{bmatrix} = (\mathbf{r}_t, |\psi_t|^2)$$

To accomplish this operation, we have written a python utility program, `particle_sample.py`, This program can read either the Numpy or Matlab binary files (which store the values of $|\psi|^2$ on the 3D mesh). Apart from the input data type, the script can produce an output file (with particle pos and amplitude: $(\mathbf{r}_t, |\psi_t|^2))$ for each time point, or it can produce a single file consisting of all time points. This routine saves $(\mathbf{r}_t, |\psi_t|^2))$ in JSON because it can be handled directly by the WebGL low level libraries.

In any case, the sampling data is saved in a file called `3dData.3d`. This file is then loaded into the QMwebJS.

Using the particleSample utility code The python script, `particleSample.py`, that we provide converts either matlab or python simulation data, as described above. Parameters can be passed via the command line in the standard way as follows:

```
usage: particleSampling.py [-h] -m MATTYPE [-t THRESHOLD] [-n NPARTICLES]
                           [-t0 TMIN] [-tf TMAX] [-s STEPSIZE]

The Particle Sampling
optional arguments:
  -h, --help            show this help message and exit
  -m MATTYPE, --mattype MATTYPE
                        The simulator output type: either .mat or .npz
  -t THRESHOLD, --threshold THRESHOLD
                        The amplitude threshold value for selected particles
  -n NPARTICLES, --nparticles NPARTICLES
                        The number of sampled particles
  -t0 TMIN, --tmin TMIN
                        The minimum time point to sample from
  -tf TMAX, --tmax TMAX
                        The maximum time point to samples from
  -s STEPSIZE, --stepsize STEPSIZE
                        The timestep for sampling
```

Step 3: Loading particle data file into QMwebJS enabled browser We provide a hosted website (http://www.parvis3d.org.es/) that is powered by QMwebJS and can be used for free. Figure 2 shows how this site and the instances of the buttons.

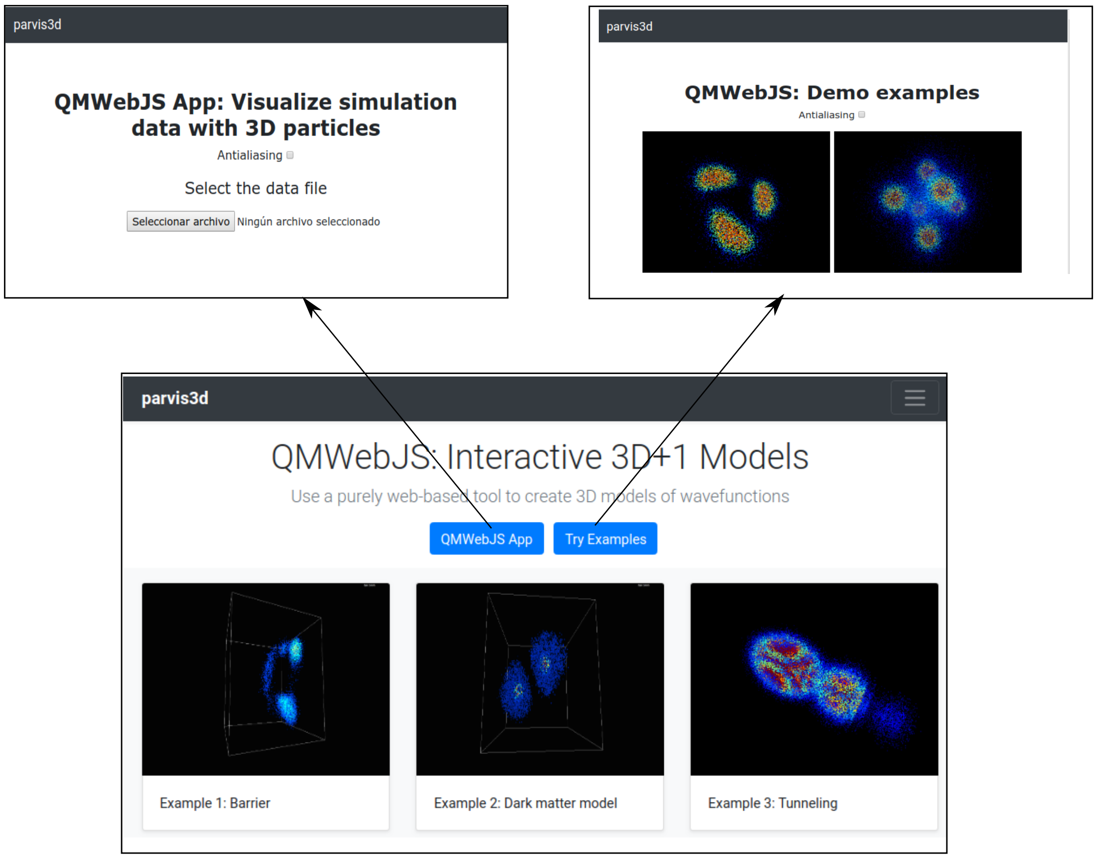

parvis3d
QMWebJS App: Visualize simulation data with 3D particles
Antialiasing
Select the data file
Seleccionar archivo
Ningún archivo seleccionado
parvis3d
QMWebJS: Demo examples
Antialiasing
parvis3d
QMWebJS: Interactive 3D+1 Models
Use a purely web-based tool to create 3D models of wavefunctions
QMWebJS App
Try Examples
Example 1: Barrier
Example 2: Dark matter model
Example 3: Tunneling

Figure 2: The hosted website app for using and trying QMwebJS.

3 Details of the QMwebJS Application

This section provides a quick-guide for using QMwebJS. A users could download the QMwebJS javascript code and insert it into an html document to run it locally or serve it from a web server. The important issue is that all processing is done within the browser and a server is not necessary. For ease, we provide a fully functioning implementation of QMwebJS at http://www.parvis3d.org.es/ so that no installation/download is required.

3.1 Data interaction

Once data is loaded into a QMwebJS, a the particle data is rendered with default parameters within a canvas. In particular, this main view consists of two parts: the 3D canvas (enabled by WebGL) and a graphical user interface (GUI) menu. The 3D canvas is fully interactive, allowing intuitive rotation, translation and manipulation of the 3D particle system. The GUI menu is used to control several parameters of the visualization and timeline.

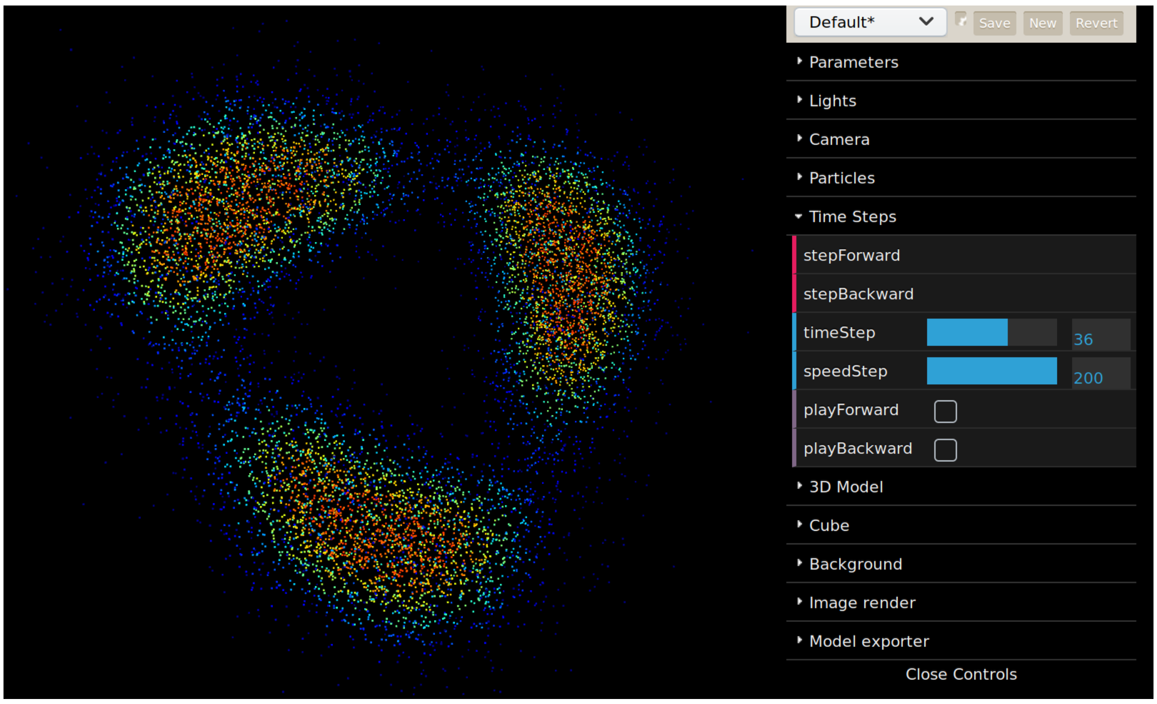


Figure 3: The QMWebJS environment. The two main parts are shown: the WebGL enabled 3D interactive canvas and the collapsible control menu.

3.2 GUI menu and functions

This menu consists of 10 collapsible tab groups. Each group controls functional aspects of the visualization, such as the timeline, camera and lighting properties, particle object type, and particle color. Since so many parameters can be fixed, the top bar provides a way to save and load session parameters.

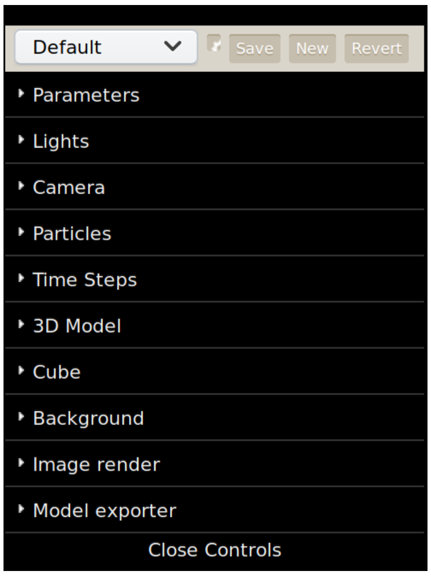
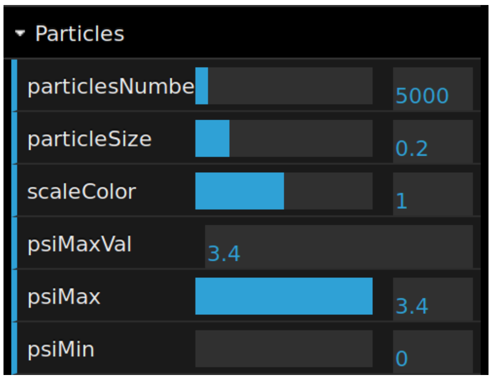


Figure 4: GUI menu with all the tabs collapsed.

3.3 The Particle Tab

The Particles Tab controls the parameters associated with particle data types and display properties. The following parameters settings can be controlled:

1. *particlesNumber*: selects the total number of particles that will be displayed in the 3D canvas.
2. *particlesSize*: sets the particle size.
3. *scaleColor*: provides fine tuning of the color range; this can have an important impact on the final renderization.
4. *psiMaxVal*: adjust the maximum $|\psi|^2$ value useful for the color range.
5. *psiMax*: another adjustment control for the maximum $|\psi|^2$ value
6. *psiMin*: adjusts the minimum $|\psi|^2$ value; once again, this is used for controlling the color range that impacts directly the final renderization.



▾ Particles

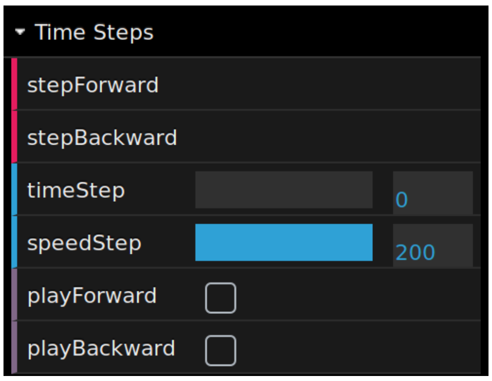
particlesNumbe	5000
particleSize	0.2
scaleColor	1
psiMaxVal	3.4
psiMax	3.4
psiMin	0

Figure 5: Particle tab.

3.4 Timeline Tab

The Timeline tab is used to adjust various parameters of the visualization along the timeline simulation.

1. *stepForward*: renders the next Time Step of the simulation.
2. *stepBackward*: renders the previous Time Step of the simulation.
3. *timeStep*: a timeline scrub-bar that advances the timeline.
4. *speedStep*: sets the time-step rate if *playForward* or *playBackward* are enabled.
5. *playForward*: automatically renders the simulation forward in time.
6. *playBackward*: automatically renders simulation backward in time.



▾ Time Steps

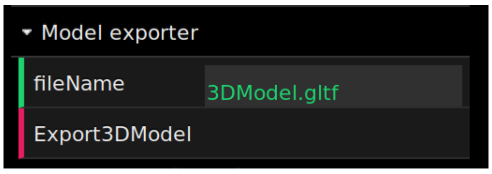
stepForward	
stepBackward	
timeStep	0
speedStep	200
playForward	☐
playBackward	☐

Figure 6: Time Steps tab.

3.5 Model Exporter

This tab exports the 3D object in the canvas to a standard GLTF model file. In this way, the exported model could be opened by any 3D editor and render engine.

1. *fileName*: sets the filename of 3D model to be exported.
2. *Export3DModel*: performs the export to .gltf format and saves to the name given in fileName.



▾ Model exporter

fileName	3DModel.gltf
Export3DModel	

Figure 7: Model exporter tab.

3.6 3D Model

The 3D Model tab is used to select the primitive particle object type. QMwebJS can use a fully 3D primitive icosphere but at a memory performance cost when compared to the pseudo-3D objects used by default (these are actually 2D objects that are scaled depending upon viewport parameters). The advantage of fully 3D icospheres is the ability to assign material properties and complex lighting.

1. *HQMode*: enables 3D primitive objects (here given as High Quality mode); so that particles are represented as icospheres.
2. *Enable3DModel*: sets HQ mode but disables if parameters change.
3. *objectsNumber*: fixes the number of particle icospheres
4. *objectSize*: sets the icosphere size
5. *modelSize*: sets the model scale size; for exporting 3D models.
6. *particlesWireframed*: sets the icosphere faces to transparent.

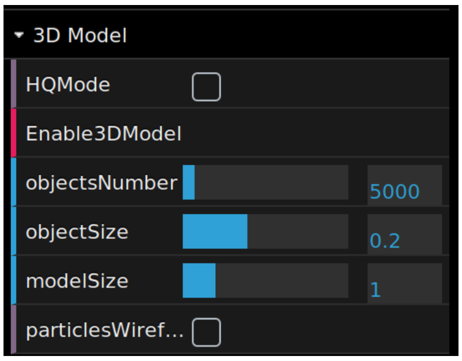

▾ 3D Model
HQMode ☐
Enable3DModel
objectsNumber 5000
objectSize 0.2
modelSize 1
particlesWiref... ☐

Figure 8: 3D Model tab.

3.7 Image Render

This tab controls the manner that the final rendering will be made. In particular, the following information can be set: the output image size, the time interval, the time step, autoplay features, and setting an orbital camera. When enabled, the model will advance along the timeline and each frame will be saved to compressed zip file.

1. *renderName*: sets the output filename
2. *renderResize*: enables resizing the renderWidth and renderHeight.
3. *renderWidth*: sets the output image width.
4. *renderHeight*: sets the output image height.
5. *setStartEnd*: enables automatic advance and capture of along timeline in the specified time interval.
6. *captureStart*: sets start time (if *setStartEnd* is enabled).
7. *captureEnd*: sets the final time (if *setStartEnd* is enabled).
8. *autoPlayForward*: initiates automatic advance of timeline between specified intervals.
9. *startCapturing*: initiates the render recording.
10. *stopAndSave*: Stops the recording at this particular time and outputs collected images.
11. *orbitalCamera*: enables the orbital camera setup.
12. *orientation*: Sets the axis of orientation for the orbital camera.
13. *orbitalSpeed*: sets the orbital speed.
14. *cameraZoom*: sets the camera position.

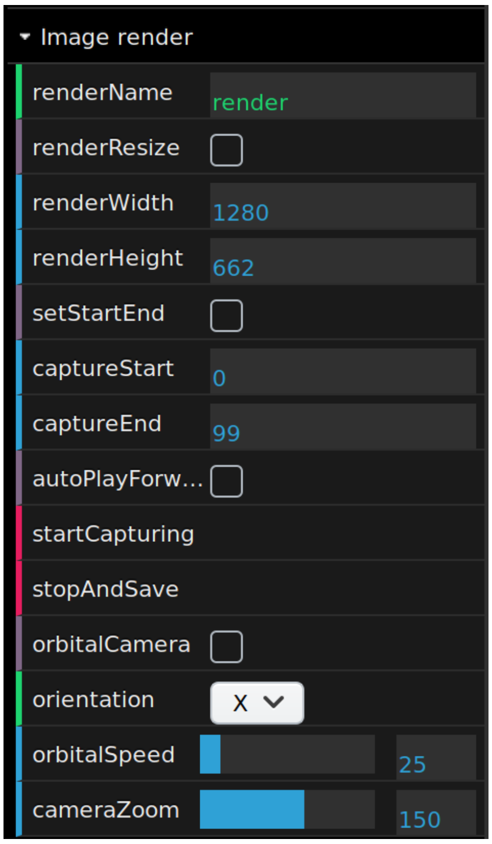

▾ Image render
renderName render
renderResize ☐
renderWidth 1280
renderHeight 662
setStartEnd ☐
captureStart 0
captureEnd 99
autoPlayForw... ☐
startCapturing
stopAndSave
orbitalCamera ☐
orientation X
orbitalSpeed 25
cameraZoom 150

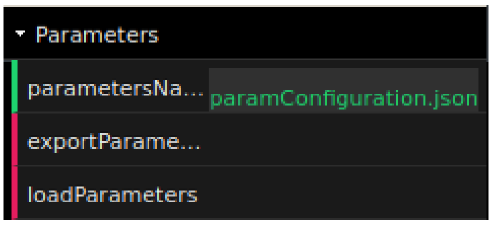
Figure 9: Image Render tab.

4 GUI secondary functions

4.1 Parameters

This tab provides utilities to save and load graphical parameters as configuration files. The parameters are saved to a JSON file (**paramConfiguration.json**) and could be modified in a text editor if desired (see example below of the file structure).

1. *parametersName*: sets the filename of the parameters configuration file.
2. *exportParameters*: generates a .json file that stores the values of the most significant parameters at the GUI menu.
3. *loadParameters*: Loads a paramConfiguration.json file and uses the parameters in it to edit the parameters of the visualization.



▾ Parameters

parametersNa... paramConfiguration.json

exportParame...

loadParameters

Figure 10: Parameters tab.

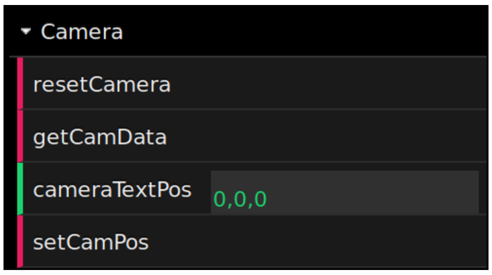
Listing 1: Example of paramConfiguration.json file

```
{"cameraX":2.26, "cameraY":-1.29," cameraZ":22,
"particlesNumber":70000, "particleSize":0.15, "scaleColor":1.57,
"psiMax":7, "objectsNumber":5000, "objectSize":0.2,
"modelSize":1, "enableCube":false, "bkgColor":"#000000",
"renderWidth":1004, "renderHeight":699,
"setStartEnd":true, "autoPlayForward":true}
```

4.2 Camera

This tab works as a fast camera editor, to reset, change and store the position of the camera.

1. *resetCamera*: sets the camera to its initial position.
2. *getCamData*: gets the current camera position.
3. *cameraTextPos*: displays the current camera position; it can be modified to change the current camera position in the graphical viewport canvas.
4. *setCamPos*: sets the current camera position in the graphical viewport canvas.



▾ Camera

resetCamera

getCamData

cameraTextPos 0,0,0

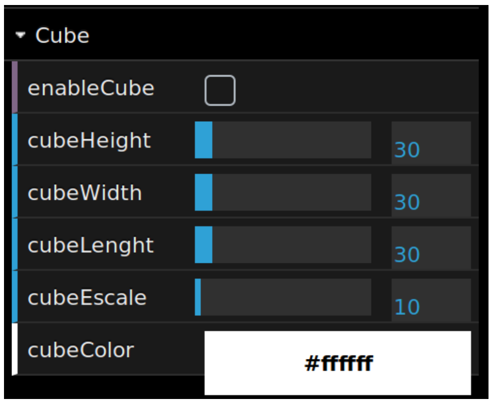
setCamPos

Figure 11: Camera tab.

4.3 Cube

By default, QMwebJS displays the 3D particle model within a wireframe cube. This cube can be useful to provide 3D perspective and orientation cues. This tab is provided to toggle this cube as well as to change its properties.

1. *enableCube*: toggles the display of the wireframe cube.
2. *cubeHeight*: sets the cube height.
3. *cubeWidth*: sets the cube width.
4. *cubeLenght*: sets the cube length.
5. *cubeEscale*: sets the cube scale
6. *cubeColor*: selects the cube color.



Cube

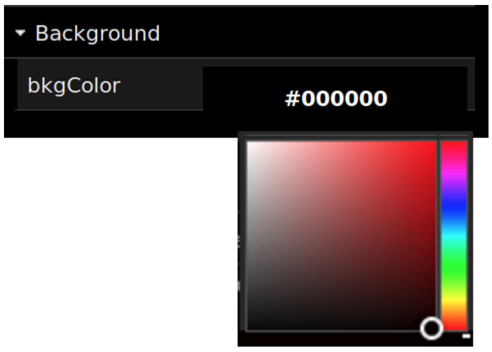
enableCube	☐
cubeHeight	30
cubeWidth	30
cubeLenght	30
cubeEscale	10
cubeColor	#ffffff

Figure 12: Cube tab.

4.4 Background

This tab provides adjustment of the background color. While the default black canvas background is often useful for onscreen viewing, lighter backgrounds can be useful for exporting image frames for videos and publications. Adjustments from this tab are immediately updated in the 3D canvas.

1. *bkColor*: deploys a color selector widget for setting the 3D canvas background. The color chosen influences the background color for exported images but not for the models exported to GLTF.



Background

bkgColor #000000

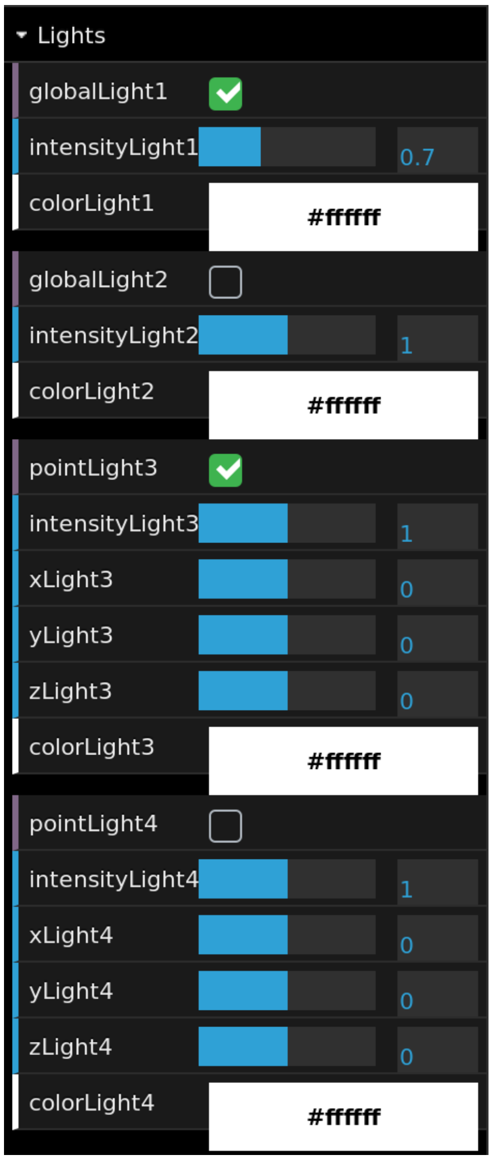
Figure 13: Background tab with color picker window.

4.5 Lights

Lighting is a key factor in order to obtain the best quality results. WebGL provides a large number of parameters for controlling lighting properties. This tab controls a subset of possibilities: a simple system with 4 lights, 2 global and 2 pointed. It should be noted that changes in the lighting parameters only affect models with the 3D models enabled.

1. *globalLight1-2*: sets environment lighting to have same global intensity without shadows; illumination is from all angles. The absolute intensity can also be adjusted to avoid over/under exposure. Color lighting can also be adjusted.

2. *pointLight3-4*: enables point lighting to act as directional sources. Changes to the position produces different effects and shadows, enhancing 3D perspective. As with the global lighting, both absolute intensity and color can be adjusted.



Lights

globalLight1	✓
intensityLight1	0.7
colorLight1	#ffffff
globalLight2	☐
intensityLight2	1
colorLight2	#ffffff
pointLight3	✓
intensityLight3	1
xLight3	0
yLight3	0
zLight3	0
colorLight3	#ffffff
pointLight4	☐
intensityLight4	1
xLight4	0
yLight4	0
zLight4	0
colorLight4	#ffffff

Figure 14: Lights tab.