Photo Sphere Viewer

A JavaScript library to display Photo Sphere panoramas

Download

Currently v3.2.2

Presentation

Photo Sphere Viewer is a JavaScript library which renders 360° panoramas shots with Photo Sphere, the new camera mode of Android 4.2 Jelly Bean and above.

Photo Sphere Viewer is pure JS and based on Three.js (http://threejs.org/), allowing very good performances on WebGL enabled systems (most recent browsers) and reasonably good performances on other systems supporting HTML Canvas.

And it works with touch screens too !

Thanks to @JeremyHeleine (https://github.com/JeremyHeleine)

I forked the original Photo Sphere Viewer by Jérémy Heleine (http://jeremyheleine.me/#photo-sphere-viewer) to provide a better JS architecture and a bunch of new features.

Getting started

Download Photo Sphere Viewer

Dependencies

- three.js (http://threejs.org)
- doT.js (http://olado.github.io/doT) (@master)
- uEvent (https://github.com/mistic100/uEvent)
- D.js (http://malko.github.io/D.js)

Markup

Include all JS & CSS files in your page and you are ready.

```
<link rel="stylesheet" href="Photo-Sphere-Viewer/dist/photo-sphere-viewer.min.css">

<script src="three.js/three.min.js"></script>
<script src="D.js/lib/D.min.js"></script>
<script src="uevent/uevent.min.js"></script>
<script src="doT/doT.min.js"></script>
<script src="Photo-Sphere-Viewer/dist/photo-sphere-viewer.min.js"></script>

<script>
  var viewer = PhotoSphereViewer({ /* ... */});
</script>
```

PSV can also be used with CommonJS :

```
require(['photo-sphere-viewer'], function(PhotoSphereViewer) {
  var viewer = PhotoSphereViewer({ /* ... */});
});
```

Canvas rendering

In order to get Photo Sphere Viewer working on browsers without WebGL you will need some additional files from Three.js examples (also available in `three.js-examples` Bower package) :

- CanvasRenderer.js (https://github.com/mrdoob/three.js/blob/master/examples/js/renderers/CanvasRenderer.js)
- Projector.js (https://github.com/mrdoob/three.js/blob/master/examples/js/renderers/Projector.js)

Transition effect

In order to perform the transition effect between two panoramas (`transition.blur` option), Photo Sphere Viewer requires the following files from Three.js examples (also available in `three.js-examples` Bower package) :

- EffectComposer.js (https://github.com/mrdoob/three.js/blob/master/examples/js/postprocessing/EffectComposer.js)
- RenderPass.js (https://github.com/mrdoob/three.js/blob/master/examples/js/postprocessing/RenderPass.js)
- ShaderPass.js (https://github.com/mrdoob/three.js/blob/master/examples/js/postprocessing/ShaderPass.js)
- MaskPass.js (https://github.com/mrdoob/three.js/blob/master/examples/js/postprocessing/MaskPass.js)
- CopyShader.js (https://github.com/mrdoob/three.js/blob/master/examples/js/shaders/CopyShader.js)

Gyroscope support

In order to be able to respond to device gyroscope (`gyroscope` option), Photo Sphere Viewer requires the following files from Three.js examples (also available in `three.js-examples` Bower package) :

- DeviceOrientationControls.js (https://github.com/mrdoob/three.js/blob/master/examples/js/controls/DeviceOrientationControls.js)

Usage

Angles definition

Photo Sphere Viewer uses a lot of angles for it's configuration, most of them can be defined in radians by using a simple number (`3.5`) or in degrees using the "deg" prefix (`'55deg'`).

Positions definition

Some methods take a `position` parameter. It is an object with either `longitude` and `latitude` properties (radians or degrees) or `x` and `y` properies (corresponding to the pixel position on the source panorama file).

Options

Name	type	default	description
container	HTMLElement|string	**required**	HTML element which will contain the panorama, or identifier of the element.
panorama	String	**required**	Path to the panorama image.
caption	String	null	A text (can contain HTML) displayed in the navbar. If the navbar is disabled it will be shown anyway but with no button.
markers	Array	[]	List of markers (http://photo-sphere-viewer.js.org/markers.html).
autoload	boolean	true	Automatically load the panorama, if `false` you must use `load` method later.
min_fov	integer	30	Minimal field of view (corresponds to max zoom), between 1 and 179.
max_fov	integer	90	Maximal field of view (corresponds to min zoom), between 1 and 179.
default_fov	integer	`max_fov`	Initial field of view, between `min_fov` and `max_fov` .
fisheye	boolean|integer	false	Enable fisheye effect with `true` or specify effect strength (`true = 1.0`). ⚠ This mode can have side-effects on markers rendering.
default_long	double	0	Initial longitude, between 0 and 2π.
default_lat	double	0	Initial latitude, between -π/2 and π/2.
longitude_range	double[]		Viewable longitude range. Examples: `[0, Math.PI]` , `[-3*Math.PI/4, 3*Math.PI/4]` .
latitude_range	double[]	[π/2, -π/2]	Viewable latitude range.
time_anim	integer|boolean	2000	Idle time (milliseconds) before the panorama automatically starts rotating. `false` to deactivate.
anim_speed	string	'2rpm'	Automatic rotation speed in radians/degrees/revolutions per

			second/minute. More
anim_lat	double	`default_lat`	Latitude at which the automatic rotation is performed.
navbar	boolean|array		Enable or disable the navigation bar, you can also choose which buttons are displayed and even add custom buttons. ℹ See below.
lang	Object	View	Text of navbar buttons tooltips.
loading_img	String	null	Path to an image displayed in the center of the loading circle.
loading_txt	String	'Loading...'	Text displayed in the center of the loading circle, only if `loading_img` is not provided.
mousewheel	boolean	true	Listen to mouse wheel events to zoom in and out.
mousemove	boolean	true	Listen to mouse click+move events to rotate the view.
keyboard	boolean	true	Enabled keyboard navigation in fullscreen.
gyroscope	boolean	false	Enable gyroscope navigation and add a navbar button when the device supports it.
size	Object	null	The final size if the panorama container (e.g. `{width: 500, height: 300}`. By default the size of `container` is used and is followed during window resizes.
transition	Object	View	Configuration of the transition effect between panoramas.

Advanced options

Name	**type**	**default**	**description**
move_speed	double	1	Speed multiplicator for manual moves.
sphere_segments	integer (multiple of 2)	64	Number of horizontal and vertical segments used to render the sphere, decrease if you encounter performance issues.
usexmpdata	boolean	true	Read real image size from XMP data, must be kept `true` if the panorama has been cropped after shot.
pano_data	object		Manually define cropping config (if

			`usexmpdata = false` or no XMP tag is found) ℹ example.
cache_texture	integer	5	Number of texture objects to cache into memory, this is to prevent network overload when calling `setPanorama` multiple times.
tooltip	object	View	Configuration of the tooltip. This only needs to be changes of the CSS is modified.
move_inertia	boolean	true	Enabled smooth animation after a manual move.
click_event_on_marker	boolean	false	A click on a marker will trigger a `click` event as well as `select-marker` .

Methods

The `PhotoSphereViewer` function returns an object with the following methods:

Actions

`.destroy()`

Destroy the viewer.

> The memory used by the ThreeJS context is not totally cleared. This will be fixed as soon as possible.

`.load() -> Promise`

Inits panorama loading if `autoload` was false.

`.render()`

Performs a full render of the viewer, useful if you manually modified some properties or updated the markers.

`.setPanorama(path [, position] [, transition]) -> Promise`

Loads a new panorama image, optionnally changing the camera position and activating or not the transition animation.

```
viewer.setPanorama(
  'panorama-2.jpg',
  {
    longitude: Math.PI,
    latitude: 0
  },
  true
);
```

`.rotate(position [, render])`

Rotates the view to specific longitude and latitude. By default this triggers and render.

```
viewer.rotate({
  longitude: Math.PI,
  latitude: - Math.PI / 4
});
```

`.animate(position, duration | speed) -> Promise`

Rotates the view to specific longitude and latitude with a smooth animation. The last param can be a duration in milliseconds or a speed with the same format as `anim_speed` parameter.

```
viewer.animate({
  x: 1200,
  y: 500
}, 2000);
```

`.zoom(level)`

Zooms to a specific level between `max_fov` (default: 0) and `min_fov` (default: 100).

```
viewer.zoom(45);
```

`.zoomIn() / .zoomOut()`

Increases or decreases to zoom level by `1` .

`.showPanel(content [, noMargin]) / .hidePanel()`

Opens the right panel with the specified HTML content.

```
viewer.showPanel(html);
```

`.showTooltip(config) / .hideTooltip()`

Displays a tooltip on the viewer.

Name	**type**	**default**	**description**
content	string	**required**	HTML content of the tootlip
top & left	int	**required**	Position of the tip of the arrow of the tooltip, in pixels.
position	string	'top center'	Tooltip position toward it's arrow tip. Accepted values are combinations of `top`, `center`, `bottom` and `left`, `center`, `right` with the exception of `center center` .
className	string		Additional CSS class added to the tooltip.

`.toggleFullscreen()`

Enters or exits the fullscreen mode.

`.startAutorotate() / .stopAutorotate() / .toggleAutorotate()`

Enables or disables the automatic rotation.

`.startGyroscopeControl()` / `.stopGyroscopeControl()` / `.toggleGyroscopeControl()`

Enables or disables the gyroscope navigation.

`.startKeyboardControl()` / `.stopKeyboardControl()`

Enables or disables the keyboard controls (done automatically when entering fullscreen).

`.hideNavbar()` / `.showNavbar()` / `.toggleNavbar()`

Hides or show the navbar.

`.preloadPanorama(panorama) -> Promise`

Put a panorama image in the cache for future use with `.setPanorama` .

`.clearPanoramaCache([panorama])`

Removes an image from the cache or clears the entire cache.

Getters & setters

`.setCaption(html)`

Changes the content of the navbar caption.

`.getNavbarButton(id)`

Returns an existing navbar button by its identifier. ℹ See below.

`.getPosition()`

`.getZoomLevel()`

`.getSize()`

`.isAutorotateEnabled()`

`.isGyroscopeEnabled()`

`.isFullscreenEnabled()`

Events

On the same viewer object you can use the `on` method to listen to various events.

`ready`

Triggered when the panorama image has been loaded and the viewer is ready to perform the first render.

```
viewer.on('ready', function() {
  // do something when the viewer is ready to render
});
```

`render`

Triggered on each viewer render **which happens A LOT when rotating, zooming, etc.**

`click [data]`

Triggered when the user click on the viewer on a "free space" (everywhere excluding the navbar, the side panel or an existing marker). The `data` contains info about the clicked point :

Name	description

target	Original target of the click event.
client_x & client_y	Position in pixels from the top-left corner of the viewer.
latitude & longitude	Corresponding spherical coordinates in radians.
texture_x & texture_y	Corresponding position on the panorama image in pixels.

`autorotate [enabled]`

Triggered when the automatic rotation is enabled/disabled.

```
viewer.on('autorotate', function(enabled) {
  // enabled is true or false
});
```

`zoom-updated [level]`

Triggered when the zoom level changes.

```
viewer.on('zoom-updated', function(level) {
  // level is from 0 to 100
});
```

`fullscreen-updated [enabled]`

Triggered when the fullscreen mode is enabled or disabled.

```
viewer.on('fullscreen-updated', function(enabled) {
  // enabled is true or false
});
```

`gyroscope-updated [enabled]`

Triggered when the gyroscope mode is enabled or disabled.

```
viewer.on('gyroscope-updated', function(enabled) {
  // enabled is true or false
});
```

`position-updated [position]`

Triggered when the view longitude and/or latitude changes.

```
viewer.on('position-updated', function(position) {
  // position has longitude and latitude
});
```

`size-updated [size]`

Triggered when the viewer size changes.

```
viewer.on('size-updated', function(size) {
  // size has width and height
});
```

`panorama-load-progress [panorama, progress]`

Triggered while a panorama image is loading, with loading progression.

```
viewer.on('panorama-load-progress', function(panorama, progress) {
  // progress goes from 0 to 100
});
```

`open-panel` / `close-panel`

`show-tooltip` / `hide-tooltip`

Navbar customization

`navbar` is an array which can contain the following core buttons: `autorotate` , `zoom` , `download` , `markers` , `gyroscope` , `fullscreen` , as well as `caption` and objects to create custom buttons :

Name	description
id	The unique identifier of the button.
title	The button tooltip.
content	The content of the button.
className	A CSS class added to the button element.
onClick	Function called when the button is clicked.
disabled	If the button must be disabled by default.
hidden	If the button must be hidden by default.

This example uses some core buttons and a custom one.

```
new PhotoSphereViewer({
  /* ..... */,
  navbar: [
    'autorotate',
    'zoom',
    'markers',
    {
      id: 'my-button',
      title: 'Hello world',
      className: 'custom-button',
      content: 'Custom',
      onClick: function() {
       alert('Hello from custom button');
      }
    },
    'caption',
    'fullscreen'
  }
);
```

Altering the buttons

After the viewer creation you cannot add or remove buttons but you can change their visibility. Use the `getNavbarButton(id)` method the get a button by its id (works for core buttons too). You will get an object with the following methods: `disable()` , `enable()` , `hide()` , `show()` .

```
viewer.getNavbarButton('my-button').hide();
```