Bling3D

http://www.starplugins.com

http://www.starplugins.com/bling3d

Document Revision: 1602171600

Contents

Intended Audience

This document is designed for web-developers or technically-minded readers who have a solid understanding of HTML and CSS. Familiarity with JavaScript would also be useful, but not essential.

While Bling3D largely removes the necessity for 3D programming or graphics knowledge, having a basic understanding 3D concepts like positions, rotations and parent/child hierarchies will be an advantage.

In most cases, referring to the code examples in the download package will help to clarify things. You can also see the examples here:

http://www.starplugins.com/bling3d/home

How to Use This Document

This document contains technical information and example code snippets. While we have tried to be as comprehensive as possible, it would be cumbersome to including full working examples within the text, especially as the content would just duplicate what is already available in the download package and online.

It is recommended that this document is read in conjunction with studying the examples. You can also see the examples here:

http://www.starplugins.com/bling3d/home

- If using the above link to view examples (as opposed to the examples in the download package), click the 'View Standalone' button on that page to open a window containing just the example
- Use your browser's 'View Source' option to study the source code of the page

Overview

Bling3D allows you to create impressive 3D content for your websites with minimal effort. It doesn't require any specialist 3D or JavaScript knowledge to use – just add some HTML tags to your page and watch them transform into an interactive 3D scene with cameras, lights, reflections and other 3D goodies.

Despite standard availability of WebGL 3D graphics on modern browsers and devices, use of 3D in websites has been mostly limited to games, demos, experiments and self-contained micro-sites. It's a pity there hasn't been more significant uptake for 'regular' websites.

One problem is that implementing 3D in a useful and polished way isn't easy or familiar for most web-developers; there is little in common with WebGL 3D graphics and 'everyday' technologies like HTML or CSS.

Even when using 3D libraries like Three.js, additional skills, resources and time are needed. A solid understanding of 3D graphics techniques, 3D content creation and certain mathematics principles are typically required to create effective 3D content.

Bling3D bridges that gap by allowing you to create impressive 3D website content and interactivity using your existing skill sets, largely eliminating the need for additional coding or 3D knowledge.

You can now consider 3D as a routine part of your development process, allowing you to literally add another dimension to your websites.

HTML to 3D

By initializing 3D content from your HTML tags, Bling3D allows your page to retain semantic meaning. Image `alt` attributes, links, lists and paragraphs are all present, allowing web crawlers and screen readers to make full sense of your content – that's good for SEO and accessibility reasons.

While modern web crawlers will attempt to index content inside JavaScript files, you can't beat having everything laid out for them in HTML.

The hierarchy of HTML tags also determines the hierarchy of the 3D scene, a neat one-to-one relationship that makes constructing a 3D scene more intuitive.

Modules

Bling3D gets its functionality from *modules* and *module boosters*.

- Modules provide core functionality (e.g. a slider, a reflective floor etc.)
- Module boosters add additional optional capabilities to modules

Full descriptions of the modules and module boosters are provided later in the document.

Technical Overview

This section covers Bling3D's technical aspects to give you an idea of how Bling3D operates in your pages, the resources it uses and any pitfalls or limitations that you need to be aware of.

Dependencies

Bling3D uses the Three.js rendering library (threejs.org) on WebGL compatible browsers. While Three.js is a large library (around 99KB gzipped), it is available via Content Delivery Networks (CDNs) for efficient caching e.g.

https://ajax.googleapis.com/ajax/libs/threejs/r73/three.min.js

or

https://cdnjs.cloudflare.com/ajax/libs/three.js/r73/three.min.js

Bling3D is compatible with r72, r73 and r74 of Three.js. Three.js goes through relatively frequent updates and the version Bling3D uses will typically trail the latest version of Three.js by a month or so. Just continue using the last compatible version of Three.js until support for the latest version in Bling3D is confirmed.

Bling3D does not use any other libraries internally, just plain JavaScript, so it is compatible with your favorite JavaScript frameworks like jQuery etc.

Files List

Bling3D splits its functionality across a few files listed here (file names are case sensitive):

- Bling3D.js – A small initialization and loader file, included in the `<head>` section or before the end of the `<body>` section if you prefer
- Bling3D.core.js – All the core modules and functionality
- Bling3D.css – Basic CSS to create a viewport window. CSS classes can be overridden, although doing this externally to this file is recommended
- three.min.js – The Three.js WebGL rendering library, include from CDN or locally if you prefer.

Only Bling3D.js and Bling3D.css need to be included in the page – other files will automatically load as appropriate.

On browsers that do not support WebGL (typically older browsers), Bling3D will fail fast and not attempt to load either Bling3d.core.js or three.min.js.

Order of Execution

- Page loads Bling3D.js and Bling3D.css
- Bling3D waits for the DOM to load
- Bling3D detects whether WebGL is available. If not available, it sets appropriate CSS classes (see Displaying Fallback Content) and exits
- Bling3D loads Three.js
- Bling3D loads Bling3D.core.js
- Bling3D Scans the HTML to create the 3D scene(s)

Displaying Fallback Content

Sometimes WebGL won't be available (generally on older browsers and devices). Bling3D allows you to display alternative content in such situations. There are two ways of displaying alternative content:

1. Display different HTML for non-WebGL browsers
2. Display the same HTML. On WebGL browsers, the HTML would be 'hidden' and used to create the 3D scene. On non-WebGL browsers, the HTML will be displayed as-is.

Displaying Different HTML

The default behavior of the included Bling3D CSS is to allow you to easily hide and show sections of your page depending on WebGL availability. There are three CSS classes you can use to do this:

- `.bling3d-show-if-webgl` (only displays element if WebGL is available)
- `.bling3d-show-if-not-webgl` (only displays element if WebGL is *not* available)
- `.bling3d` (does nothing, but element will receive a class indicating WebGL availability)

Any elements containing any of the above three classes will also obtain one of the following classes after WebGL availability has been established:

- `.bling3d-webgl-false` (WebGL is not available)
- `.bling3d-webgl-true` (WebGL is available)

So, your page content might be something like this:

```
<div class="bling3d-show-if-not-webgl">Sorry! No WebGL here.</div>

<div class="bling3d-show-if-webgl">Great, you have WebGL!</div>
```

Displaying the Same HTML

Another way to display fallback content is to use exactly the same HTML for both setting up the 3D scene and for displaying as regular 2D HTML content. Typically, this would be done by overriding the included Bling3D CSS classes so that elements are not hidden when WebGL is not available. For example:

```
.bling3d-show-if-webgl.bling3d-webgl-false {
    display: 'block';
}
```

Then it's a case of styling the elements as desired. There is a fully working example of the same HTML approach here:

http://starplugins.com/bling3d/example-shared-html

You could also use CSS media selectors to show or hide 3D content based on screen size.

CPU, GPU and Battery Consumption

3D graphics are demanding of hardware resources, so Bling3D has been designed to minimize consumption of these resources where possible – no one wants to visit a website that will cause their device's battery to run flat after 30 minutes or cause their laptop fan to go into overdrive.

When a Bling3D viewport is not interacted with for a few seconds, it enters a 'sleep' mode where 3D rendering and calculations are paused. The viewport consumes little more resources than a regular static image in this mode.

Specifying Colors

In most cases where colors can be specified, Bling3D will accept them as a numbers or as CSS color strings. Here are several valid ways of specifying bright blue:

- 0x0000ff
- 255

- 'blue'
- 'rgba(0,0,255,1)'
- '#0000FF'

Screen Resolution and Device Pixel Ratios

When device manufacturers started using very high resolution displays in their portable devices (e.g. iPad 'retina' display with 2048 × 1536 pixels), they were faced with the problem that many websites and apps are/were designed to fill screens of much lower resolution. This means a typical website would have occupied less than one quarter of an iPad retina display, too small to comfortably use.

This issue was solved by having a 'device pixel ratio' (DPR), which allows devices to pretend they have a lower resolution display than their real or native resolution. For example, iPad retina has a DPR of 2 which means every website pixel (CSS pixels) is mapped to two native resolution pixels. This effectively doubles the website in size to use the majority of the screen space.

Fonts and other elements can still look razor sharp because they are drawn using the device's native resolution. Websites can also deliver alternative image sizes to better use the native resolution for crisper images by using the HTML `srtcset` attribute on images.

Bling3D uses the device's native resolution by default for sharper looking 3D content. One caveat is that drawing more pixels is performance hungry, so you can turn down the pixel ratio if required (see Bling3D.Viewport)

Basic Example Page

The following example creates a page with a Bling3D viewport containing a circular carousel with full mouse and touch input. Don't worry if you don't understand some of the HTML attributes being used as they will be explained later in the document.

(Image `alt` attributes omitted for brevity).

```
<!DOCTYPE html>
<html>
<head>
    <title>Bling3D Basic Carousel Example</title
    <link rel="stylesheet" type="text/css" href="css/Bling3D.css">
    <script src="js/Bling3D.js"></script>
</head>
<body>
    <div data-b3d-viewport="itemRotation:{y:-90}" data-b3d-viewport-input >
        <img data-b3d-image-tile src="images/tiles/image1.jpg" />
        <img data-b3d-image-tile src="images/tiles/image2.jpg" />
        <img data-b3d-image-tile src="images/tiles/image3.jpg" />
        <img data-b3d-image-tile src="images/tiles/image4.jpg" />
        <img data-b3d-image-tile src="images/tiles/image5.jpg" />
        <img data-b3d-image-tile src="images/tiles/image6.jpg" />
        <img data-b3d-image-tile src="images/tiles/image7.jpg" />
        <img data-b3d-image-tile src="images/tiles/image8.jpg" />
        <img data-b3d-image-tile src="images/tiles/image9.jpg" />
        <img data-b3d-image-tile src="images/tiles/image10.jpg" />
        <img data-b3d-image-tile src="images/tiles/image10.jpg" />
    </div>
</body>
</html>
```

This will create the output shown below:

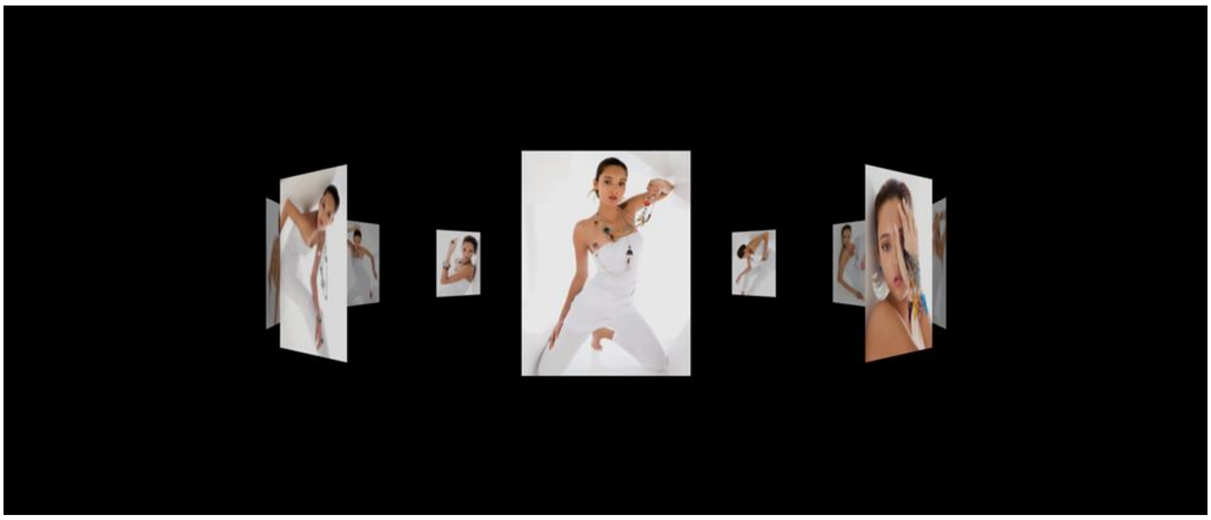

Figure 1 Simple carousel output

Limitations

Text Handling

HTML and CSS do a great job of displaying text. Bling3D doesn't attempt to reinvent the wheel and provides only basic text-handling within a 3D scene. It is assumed you will still use HTML and CSS for most text content and best text quality.

HTML Elements

The 3D environment is not the same as the HTML page environment. In fact, the HTML parser just treats the 3D scene like a static image. You cannot use any HTML elements inside the 3D scene. If you want HTML functionality, you will need to overlay the elements on top of the 3D viewport element.

Image Sizes

GPUs have limited amounts of RAM dedicated to images and other video/graphics related items. Some GPUs may use a portion of standard system memory if they do not have dedicated graphics memory. Using very large images is not advised as this will consume the GPU's memory quickly and force mitigating actions that will reduce performance as the system struggles to manage its graphics memory deficit.

WebGL prefers image sizes whose width and height are 'powers of two' (POT). It can apply better filtering effects to such images for improved images quality (especially when reducing in size), and also repeat-tile these images over surfaces. Here is a list of powers of two:

1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048 etc.

So, good image sizes might include: 256 × 256, 512 × 512, 256 × 128, 128 × 64 etc.

You might not have the option of using POT image sizes, so Bling3D will still handle non POT images.

In order of preference, here are the image sizing options:

- POT for both width and height
- POT for width or height
- Arbitrary sizes for width and height

You will have the option to make Bling3D convert non POT images to POT internally to receive the benefits of this image sizing format.

Image Rendering Accuracy

When images are drawn inside a 3D scene they will be subject to various lighting, filtering, scaling and distortion effects. The image appearance may also vary between devices depending on the model of GPU. This means that images will not be pixel-perfect renditions of the original source image. This is not a bug and cannot be fixed. Use regular HTML if you want to display 'perfect' images.

Antialiasing

Not all devices support antialiasing for smooth edges. This means you may notice a jagged edge or 'stair-casing' effect, especially where there is significant contrast between an edge and its background.

On very high-resolution displays that don't have antialiasing (e.g. iOS with retina display) the stair-casing effect will be less pronounced due to the very small pixels.

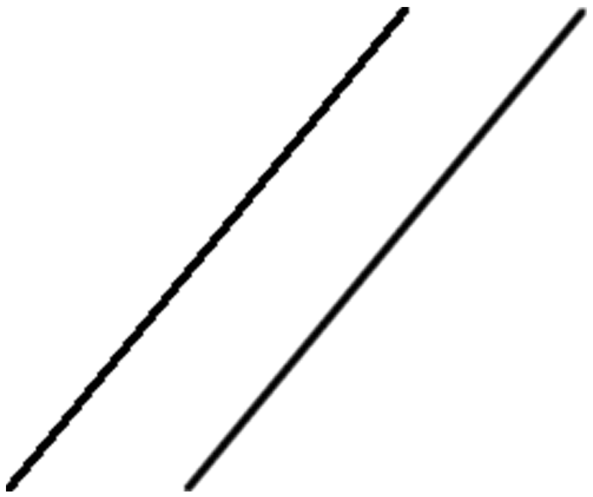

Figure 2 Non-antialiased line, antialiased line

Browser Support

Browser support for Bling3D is limited to those that support WebGL:

- Chrome
- FireFox
- Safari
- Internet Explorer 11
- Microsoft Edge

On incompatible browsers and devices, Bling3D will fail gracefully and allow you to display alternative content.

Certain default touch behaviors may be disabled on Bling3D viewports on some devices, for example, pinch zooming. This is largely controlled via CSS and the `touch-action` property. The default setup case found in Bling3D.css is `touch-action: pan-y` to allow vertical page swiping/scrolling to work, but to allow other touch actions to control the 3D scene.

Coordinate Systems

When we use positions and sizes, we need to know which 'space' the values are being used in. If you are developing web-pages that contain Bling3D viewports, you will be mostly concerned with two coordinate spaces:

Screen Space / Screen Coordinates

This is just the familiar CSS/HTML coordinate space that web developers use every day. The vertical axis goes from top to bottom, the horizontal axis goes left to right. Each unit is equivalent to a CSS pixel.

World Space / World Coordinates

This is the 3D coordinate space used inside Bling3D viewports. The units are irrelevant; they could refer to centimetres, feet, fathoms or nothing at all, but just use the same units throughout. Notice in the diagram below that the origin is in the center.

Bling3D uses a 'right-handed' 3D coordinate system as shown in the diagram below, with X going left to right, Y going from bottom to top, and Z coming out of the screen towards the viewer.

Note that the perceived directions and positions can appear to change (even though they haven't) if different camera angles or camera positions are used, for example:

- If the camera was to spin 90 degrees about its Y axis, then objects moving along the X axis would appear to move in and out of the screen, whereas objects moving along the Z axis would appear to move side to side on the screen
- If the camera pans to the left, then everything in the scene will appear to move to the right

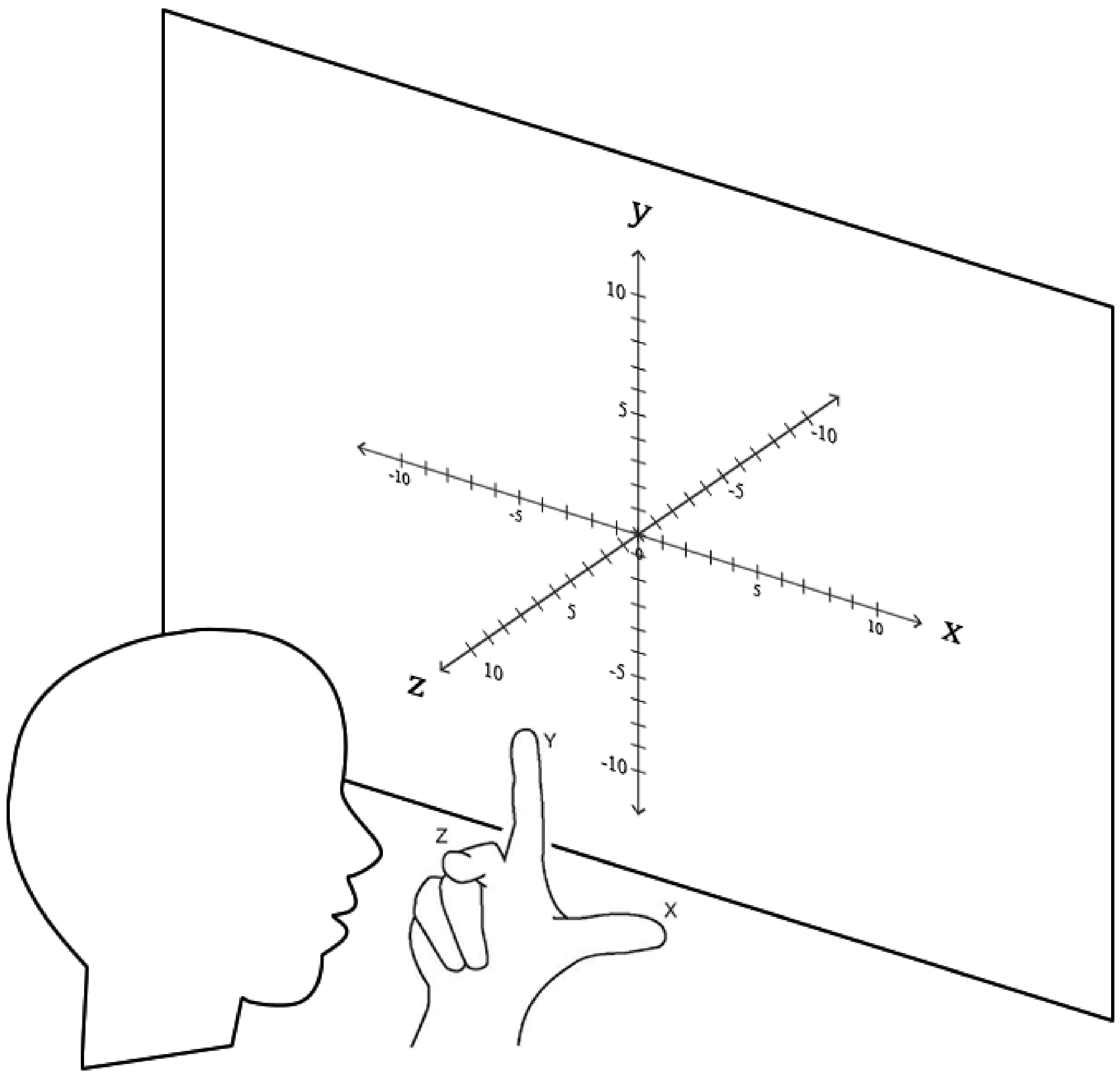

Figure 3 World coordinate system

Materials

Some modules allow you to specify a 'material' (not to be confused with 'material design' used in the Android OS and other frameworks). Materials in Bling3D and Three.js refer to the way surfaces are rendered, specifically how they are affected by lighting and other effects.

Materials can be specified in Bling3D as follows:

- **Basic** – Objects are rendered 'as is', with no lighting or other effects. Objects can still fade into the background if fog is used
- **Lambert** – Draws objects that react to the lights around them. Surfaces exposed to light will be brighter than those that aren't
- **Phong** – A more sophisticated form of lit object that will show highlights and look 'shinier' than Lambert objects

Some modules may define their own materials for other effects; these will be documented individually in their module descriptions.

Modules

Modules contain JavaScript code that Initialize 3D objects and place them in the scene

The latter part of this section lists the modules available in Bling3D. The module names in the headings refer to the JavaScript class name used in the module.

Using Modules via HTML

To use a module within HTML, apply the appropriately named HTML5 `data-`attribute to HTML elements to trigger the module.

Typically, the `data-`attribute will retain the same name as the JavaScript class of the module, but all lowercase with a hyphen instead of *camelCase*, and prefixed by 'b3d', for example:

Bling3D.Viewport

```
<div data-b3d-viewport >
    <!-- Further content here. -->
</div>
```

Bling3D.ImageTile

```
<img data-b3d-image-tile src="..." />
```

The 'b3d' prefix is to avoid clashing with non-related JavaScript and plugin use of `data-` attributes.

Setting Module Properties via HTML

To set module setup properties to the HTML tag, enter them into the `data-` attribute as you would JavaScript properties, but without any opening and closing braces (spaces added for clarity, but not required):

```
<div data-b3d-viewport = " clearColor:'white', wakeTime:5 " >
```

Notice how colons appear between property names and their values, and how commas separate different properties. Text values are surround by single quotes, numbers are not. Similarly, if using true or false, no quotes are required. If you are starting and ending your `data-` attributes in single quotes, then use double quotes for the text values (otherwise you will get HTML errors):

```
<div data-b3d-viewport = ' clearColor:"white", wakeTime:5 ' >
```

Some modules have lots of properties, but you don't have to set all of them as default values will be used if you don't set any yourself. In fact, this is the recommended approach; adding properties in steps rather than trying to get all of them right at once.

Using Images with Modules

For modules that can use images, Bling3D allows you to specify which images to use in a variety of ways for maximum flexibility and best image quality across devices.

Using Image Tags to Specify Images

The simplest method to use images is to use regular `<img>` tags as the module trigger, for example:

```
<img data-b3d-image-tile src="image.jpg">
```

If you want to specify alternative images in a responsive design, Bling3D also works with the HTML `srcset` attribute, for example:

```
<img data-b3d-image-tile src="image.jpg" srcset="image1.jpg 1x, image2.jpg 2x">
```

If using non image elements to trigger modules, you can still use `<img>` tags to specify the image source for better SEO, for example:

```
<li data-b3d-image-tile >
    <img src="image.jpg">
</li>
```

In the above case, Bling 3D will search for the image contained within the surrounding `<li>` element.

Note: If using cross-domain images, ensure that the server delivering the images is setup for CORS (see

Images and Cross Origin Resource Sharing (CORS))

Using Other Tags to Specify Images

You can also specify the images in non-image tags by using `src` and `srcset` as properties of the module instead of attributes of an `<img>` tag, for example:

```
<span data-b3d-image-tile ="src:'image.jpg',srcset:'image1.jpg 1x,image2.jpg 2x' ">
</span>
```

Using Other Tags with Image Tags to Specify Images

Sometimes you might want to specify completely different images for WebGL and non-WebGL devices. This can be done by specifying `src` and `srcset` as properties of the module for the WebGL images, but have a child image element for the non-WebGL image, for example:

```
<li data-b3d-image-tile="src:'webgl.jpg', srcset:'webgl1.jpg 1x, webgl2.jpg 2x' ">
    <img src="no-webgl.jpg">
</li>
```

The properties will have priority over the image element on WebGL devices.

Specifying Image Repeat

Images can be made to tile across a surface a certain number of times. Modules that allow image repeating provide the following properties:

Properties

Property	Description	Type	Default Value
`ImageRepeatX`	Number of times image will repeat horizontally across surface.	Number	`0`
`imageRepeatY`	Number of times image will repeat vertically across surface.	Number	`0`

Images and Cross Origin Resource Sharing (CORS)

If using cross-origin images (images served from a different domain to the web page), then use the `crossorgin` attribute on the image tags:

```
<img data-b3d-image-tile src="image.jpg" crossorigin="anonymous">
```

Most CDNs (Amazon S3, Cloudflare, Flikr, Google etc.) can deliver images via CORS.

Images specified via `src` or `srcset` properties (rather than image attributes) will automatically be set as cross origin.

Unfortunately, there is no way of getting around CORS security permissions when using Canvas and WebGL as they can access the pixel data directly and are therefore seen as a potential security threat.

If delivering images via your own custom CDN, you will need to ensure the server is setup to deliver content via CORS (not relevant if on the same domain as your website).

Setting Position and Rotation of Objects

Where modules initialize 3D objects, you will often want to change to position and rotation of the object. Some modules will reserve the following properties to do that:

Properties

Property	Description	Type	Default Value
`px`	X position in world coordinates (relative to parent)	Number	`0`
`py`	X position in world coordinates (relative to parent)	Number	`0`
`pz`	X position in world coordinates (relative to parent)	Number	`0`
`rx`	Rotation around object's x-axis (relative to parent)	Number	`0`
`ry`	Rotation around object's x-axis (relative to parent)	Number	`0`
`ry`	Rotation around object's x-axis (relative to parent)	Number	`0`

The example below would place the image tile at 10 units across the x-axis, with a 45° rotation about its y-axis:

```
<img data-b3d-image-tile = "px:10, ry: 45" src = "..." />
```

Note that the positions and rotations are relative to the object's parent.

Using Bitmap Text with Modules

Some modules allow you to specify text that will generated as bitmap planes that can be displayed in the 3D scene. The text may appear as individual objects or overlaid on other content such as image tiles.

The text is created via an internal 'text canvas'. Text formatting is limited compared to HTML and CSS, but should suffice for most basic text requirements. Note that any HTML/CSS styling applied to elements will not carry over into the generated text.

Modules that use bitmap text will reserve a `textCanvas` object property. The object contains the following listed below.

Properties

Property	Description	Type	Default Value
`align`	Horizontal text alignment within the canvas. Specify `'left'`, `'right'` or `'center'`	String	`'left'`
`backColor`	Color of the font background as a CSS color string. For no background color, use `'transparent'`.	String	`'white'`
`color`	Color of the font as a CSS color string.	String	`'black'`
`font`	Defines the font face and optional fallback font.	String	`'arial, sans-serif'`
`height`	Height of the internal canvas in pixels. Use `'auto'` to specify that the canvas will size itself to accommodate the text automatically.	Number - or - String	`'auto'`
`lineHeight`	Line spacing in pixels. Typically equal or larger than the font size. If `'auto'`, then line height will be set to 1.1 * font size.	Number	`'auto'`

Property	Description	Type	Default Value
`padX`	Padding inside left and right edge, specified in pixels. Mostly useful when a background color has been specified.	Number	`0`
`padY`	Padding inside top and bottom edge, specified in pixels. Mostly useful when a background color has been specified.	Number	`0`
`size`	Font pixel size	Number	`30`
`style`	Font style: • `'' (normal)` • `'italic'` • `'oblique'`	String	`''`
`variant`	Font variant: • `'' (normal)` • `'small-caps'`	String	`''`
`weight`	Font weight : • `'' (normal)` • `'bold'` • `'lighter'` • `'bolder'` • `100 - 900` Note: not all values supported depending on system/font.	String	`''`
`width`	Width of the internal canvas in pixels. Use `'auto'` to specify that the canvas will size itself to accommodate the text automatically.	Number - or - String	`'auto'`

Note that the canvas width and height does not necessarily correlate with the size of the text in world coordinates. That may be specified as a separate property in the module. The size of the text canvas is of more significance to the quality of the text appearance – larger canvases and font sizes will appear less pixelated and blurred when magnified.

Specifying Newlines

Newlines in text are specified as follows:

- Where text is sourced from HTML attributes, use `&#10;`
  `<div title="First Line&#10;Second Line">...</div>`
- Where text is sourced from inner HTML content, use `<br/>`
  `<p>First Line<br/>Second Line</p>`

Bling3D.Viewport

```
<div data-b3d-viewport = "..." >
    <!-- Content here. -->
</div>
```

This module creates 3D window in the page inside which the 3D scene appears. Its position, size and visibility are entirely controlled by CSS styling for maximum flexibility and easy page integration.

Properties

Property	Description	Type	Default Value
`antialias`	Enables antialiasing on supported hardware for smoother looking edges.	Boolean	`true`
`clearColor`	Sets the background color of the viewport.	Number - or - String	`0x000000`
`fogColor`	Sets the distant 'fog' color of the viewport. Objects will fade into this color before disappearing into the distance. Typically, it would match `clearColor`	Number - or - String	`0x000000`
`fogFar`	Distance from the camera where fog effect will end and objects disappear completely	Number	`20`
`fogNear`	Distance from the camera where fog effect will start	Number	`10`
`forceAntialias`	On devices that do not support WebGL antialiasing (e.g. iOS devices), this creates an antialiasing effect by rendering to a larger canvas and scaling down. The value specifies how much larger than native resolution the canvas will be. Try values between 1.1 – 1.5. • Sets `pixelRatio` to `'auto'` • `antialias` must be true • Performance penalty for using this feature	Number	`0`
`pixelRatio`	Specifies the pixel ratio. `'auto'` sets it to the same as the device's `window.devicePixelRatio` (WDPR) for maximum clarity by rendering at native resolution instead of CSS resolution (e.g. retina screens). A value of one would match CSS resolution for better performance but less clarity. Values are capped at WDPR.	Number - or - String	`'auto'`
`wakeTime`	Number of seconds that a viewport will remain 'awake' after the user has stopped interacting with it. After the specified time, the viewport enters a battery saving mode with lower CPU and GPU use. Any interaction wakes it up again.	Number	`5`

Bling3D.Camera

```
<span data-b3d-camera="..."></span>
```

Adds named cameras to the scene. Other modules can reference these cameras to change the view depending on various circumstances such as clicking a button or selecting a slider.

Properties

Property	Description	Type	Default Value
`name`	A unique identifier for a camera. Other modules can refer to the camera by this name. Names only need to be unique per viewport	String	`'defaultCamera'`
`posTarg`	An array of six numbers that define the position and target of the camera. The first three numbers represent the position of the camera in world coordinates. The second set of three numbers represent the target position in world coordinates that the camera looks at. The viewport will a camera called 'defaultCamera' by default.	Array	`[ 0,0,0, 0,0,-10 ]`

Bling3D.MirrorFloor

```
<span data-b3d-mirror-floor = "..." ></span>
<img data-b3d-mirror-floor = "..." src = "path/to/floor-image.jpg" />
```

Adds a reflective plane to the scene to create a shiny floor effect. The effect is created in two layers; the bottom layer is the reflection; the top layer is a texture or color overlay. The reflection can be turned off if desired.

This 'mirror' effect is achieved by rendering the scene twice, so there can be performance implications on lower powered devices – use with caution. Only one plane with reflection can be used in the scene. Multiple non-reflective planes can be used.

Using Images

See: *Using Images with Modules*

Selective Object Reflections

Sometimes it will be of no benefit to reflect certain objects in the scene as they will just slow down rendering for no noticeable aesthetic benefit, for example, color domes over the scene. To prevent an object being reflected, add the `data-b3d-no-reflection` attribute to the object:

```
<span data-b3d-color-dome = "..." data-b3d-no-reflection></span>
```

Properties

Property	Description	Type	Default Value
`color`	Applies a color tint to the overlay. If using an image overlay, the default of white will preserve the original appearance. By using images that are gray-scale, you can easily control their color.	Number - or - String	`0xffffff`
`material`	The material used for rendering the plane. See **Error! Reference source not found.**	String	`'basic'`

Property	Description	Type	Default Value
`opacity`	The opacity of the overlay, 0 (min) to 1 (max). A value of one means no reflection will be seen, a value of zero means no overlay will be seen, just the reflection.	String	`0.5`

Bling3D.ImageTile

`<img data-b3d-image-tile = "..." src = "path/to/image.jpg" />`

Places an image tile inside the 3D scene. Tiles are double-sided and an optional depth and frame edge color can be specified.

Using Images

See: *Using Images with Modules*

Setting Position and Rotation

See: *Setting Position and Rotation of Objects*

Text Labels

A simple plain-text label can be applied to the top or bottom of the image tile, using a specified HTML attribute as the text content source.

Properties

Property	Description	Type	Default Value
`depth`	Depth or 'thickness' of the tile. Can be used to give images more 'substance' rather than being wafer-thin planes.	Number	`0`
`frameColor`	Color of the frame edge. Only relevant if a depth is specified.	Number - or - String	`0x000000`
`maxHeight`	The maximum height in world space that the tile will occupy. Image aspect ratio will be maintained, image will be scaled to fit inside size defined by `maxWidth` and `maxHeight`.	Number	`1`
`maxWidth`	The maximum width in world space that the tile will occupy. Image aspect ratio will be maintained, image will be scaled to fit inside size defined by `maxWidth` and `maxHeight`.	Number	`1`
`originY`	The vertical origin of the tile: 1 = top 0 = middle -1 = bottom So, to place an image flush with the floor, specify -1.	Number	`0`
`shadowPath`	Path to an image to be used as a shadow underneath the image tile. Works best with a subtle gradient image in PNG format.	String	`''`

Property	Description	Type	Default Value
`textAttribute`	HTML attribute to use as text source e.g. 'title'.	String	`''`
`textCanvas`	The text canvas properties. See **Error! Reference source not found.**	Object	-
`textVerticalPos`	Vertical position of text on image tile, top or bottom.	String	`'bottom'`

Bling3D.CanvasTile

Creates a 3D tile (similar to Bling3D.ImageTile) from a `<canvas>` element in the webpage. This allows you to draw on the canvas using JavaScript code or plugins as normal, but you will see the output in the 3D scene. This is great for charts, animations and other effects. You can also tile the canvas across the tile surface.

```
<canvas
    data-b3d-canvas-tile
    width="512" height="512"
    data-b3d-store-in-dom
>
</canvas>
```

The `data-b3d-store-in-dom` attribute makes Bling3D stores a reference to the `Bling3D.CanvasTile` object inside the canvas element. This is required so you can 'refresh' the canvas after it has been changed e.g. during animation.

Updating the Canvas

Here is an example snippet that shows how to refresh the canvas after it has changed:

```
// At this point the canvas has been drawn on and is ready to be refreshed.
canvas = document.getElementById("my-canvas");
if (canvas.bling3d) canvas.bling3d.refresh();
```

Notice the reference to the `Bling3D.CanvasTile` object is stored in a property called `bling3d` inside the actual canvas element.

Repeating the Canvas

See: *Specifying Image Repeat*

Setting Position and Rotation

See: *Setting Position and Rotation of Objects*

Bling3D.Node

`Bling3D.Node` acts as an empty parent node to which other objects can be attached. Think of it as analogous to a `<div>` element in HTML. Here's an example of usage:

```
<div data-b3d-node="px: 5">
    <img data-b3d-image-tile src="..." />
    <img data-b3d-image-tile="px:3" src="..." />
</div>
```

In this example, the first image would assume a world coordinate x-position of 5, the second image would assume an x-position of 8.

Setting Position and Rotation

See: *Setting Position and Rotation of Objects*

Bling3D.FlatText

Provides basic text handling using flat planes of bitmap text. Planes are double-sided so text can be read from both sides.

```
<p data-b3d-flat-text>This is some text</p>
```

Properties

Property	Description	Type	Default Value
`textCanvas`	The text canvas properties. See **Error! Reference source not found.**	Object	-
`textCanvasScale`	When `width` is `'auto'`, the internal text canvas width is multiplied by `textCanvasScale` to provide the world space size. Note that you can use simple calculations for the sake of clarity e.g. 1/200 instead of 0.02.	Number	`0.01`
`width`	Width of the text in world coordinates. If `'auto'` is specified, width will scale the internal text canvas by the value used in `textCanvasScale`.	Number - or - String	`'auto'`

Setting Position and Rotation

See: *Setting Position and Rotation of Objects*

Bling3D.PathSlider

Path sliders take control of groups of child objects (image tiles, flat text, canvas tiles etc.) to create straight sliders, carousel, cover-flow and other related effects. By using spline-based curves, the paths can be far more versatile than is typically seen in CSS-based sliders. Because the paths are defined in 3D world space, they can also slope and undulate up and down.

Path Types

Path sliders offer four types of path:

- **Line** – Classic linear sliders
- **Oval** – Closed circles and ellipses for classic carousels

- **Open curve** – Used for more elaborate linear sliders
- **Closed curve** – For more complex closed curves than ovals

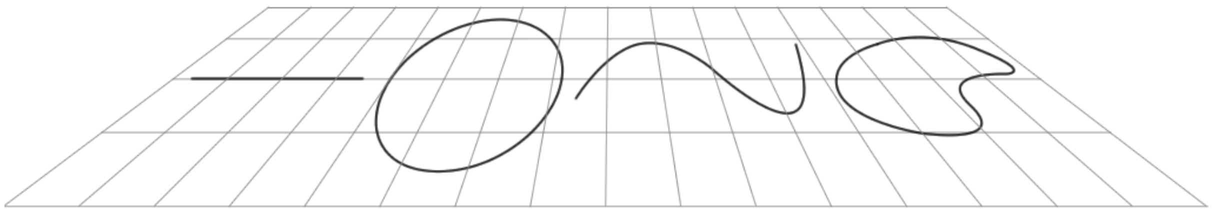

Figure 4Path types from left to right: line, oval, open curve, closed curve

The paths are defined by specifying values in a `points` array property. The values can represent either groups of world coordinates (x,y,z), or in the case of oval paths, two radii. A smooth path using those values will then be calculated. The child objects will move along that path, turning and tilting to match the curves and undulations.

Each path type requires two or more points/values to to be defined:

- **Line** – Two points (start and end of line), e.g:
  `points:[-5,0,0, 5,0,0]` a straight line, 10 units wide
- **Oval** – Two values representing the radii of the circle or ellipse, e.g:
  `points: [3,3]` a circle, radius 3 units
- **Open curve** – Multiple points (start to end), e.g:
  `points: [-5,0,0, 0,0,1, 5,0,0]` a curve that bulges out in the z-axis in the middle
- **Closed curve** – Multiple points (start to end, end of will join to start)
  `points: [-3,0,-3, 3,0,-3, 3,0,3, -3,0,3]` a square with rounded corners

Note: The positions defined will be relative to the path slider position.

Looping and Non-Looping Paths

Line and open curve type paths can set one of two types of looping action as image tiles move along them:

1. Non-looping – Tiles will disappear off the ends of the path
2. Looping – Tiles disappear off the ends of the path, but reappear at the other end

Note: Oval and closed curve type paths are always looping regardless of settings.

Offset along Path

Internally, a child object's offset along the path is represented by a single value, 0 – 1, where zero represents the beginning of the path and one represents the end of the path. A value of 0.5 would therefore be halfway along the path.

The child object's offset along the path will then be converted to world space coordinates for display purposes.

Setting Position and Rotation

See: *Setting Position and Rotation of Objects*

Properties

Property	Description	Type	Default Value
`debug`	This displays the path as a green line which is useful for debugging the points array.	Boolean	`false`
`moveToPrimary`	If true, when an image tile is clicked, it will automatically scroll to the `primaryOffset` in the path.	Boolean	`false`
`path`	The path type, one of: • `'oval'` • `'line'` • `'openCurve'` • `'closedCurve'`	String	`'oval'`
`points`	Values for the path.	Array	`[3,3]`
`primaryOffset`	This defines the path offset of the 'hot' image tile – the one that is the focus of attention that will be used by other modules to extract information from such as text descriptions. The first image tile will appear at this offset when the slider initializes.	Number	`0.5`
`spacing`	Defines the spacing between the image tiles along the path. When `'auto'`, the items will be spaced evenly along the entire length of the path. So, it there were 10 items, the spacing would automatically set to 0.1, where 1 represents the entire length of the path. If specifying a manual spacing on open paths (line or open curves) you can use a spacing that will force some image tiles to 'overspill' out of the path dimensions. Then the `looping` status will come into play.	Number - or - String	`'auto'`
`swipeDirection`	Defines the screen swiping direction that affects the path slider: • `'horizontal'` • `'vertical'`	String	`'horizontal'`
`swipeMomentumSpeed`	Defines the deceleration speed when dragging has stopped. The value is multiplied by the final dragging velocity to give a deceleration velocity.	Number	`0.4`

Property	Description	Type	Default Value
`swipeSpeed`	Defines the speed of movement when dragging a path slider with either touch or mouse. Larger values are faster.	Number	`0.2`
`looping`	Defines the looping state of the image tiles along line or open curve paths. Closed curve or oval will always loop.	Boolean	`true`
`cameraName`	The name of the camera that the path slider will use. For scenes that contain multiple path sliders, you can define different cameras for each path slider.	String	`'defaultCamera'`
`itemRotation`	When path sliders take control of image tiles, the default behavior is for the image tiles to line up along the path like dominoes waiting to be toppled. By using `itemRotation`, you can make the image tiles face in other directions. The rotations are specified as degrees around the x, y or z axis. If you wanted the image tiles to face out of the path instead of along it, you could specify and rotation around the y-axis of -90 degrees.	Object	`{x:0, y:0, z:0}`
`firstBound`	Sets the path offset beyond which the first image tile will not pass when moving forwards through the path. (Use with non-looping paths)	Number	`1e99`
`lastBound`	Sets the path offset beyond which the last image item will not pass when moving backwards through the path. (Use with non-looping paths)	Number	`-1e99`

Setting Position and Rotation

See: *Setting Position and Rotation of Objects*

Bling3D.ColorDome

`Bling3D.ColorDome` allows you to easily create a 'dome' of gradient color over the scene. It is an effective way of adding atmosphere to a scene with minimal effort.

It allows you to specify two colors, one for the top or 'sky' color, and one color for the bottom or 'horizon'. A smooth gradient of color will be created between the top and the bottom.

The color dome is always positioned at the same position as the camera, so the camera will never appear to get closer or further away from the dome. This gives the impression of an all-encompassing panorama of color.

Typically, you would want the viewport's fog color to match the bottom color so objects appear to fade into the distance.

Properties

Property	Description	Type	Default Value
`topColor`	Color of the upper part of the dome.	Number - or - String	`'white'`
`bottomColor`	Color of the lower part of the dome.	Number - or - String	`'black'`

Module Boosters

Module boosters add extra functionality or features to modules. Boosters are added in the same way as modules, by adding the appropriated `data-` attribute to the HTML element. The booster attribute is always added in addition to the module attribute itself, for example:

```
<img data-b3d-image-tile data-b3d-some-booster src= "image.jpg" >
```

In this example, the first `data-` attribute is for the module, the second is for the booster. Boosters may or may not be specific to a certain module.

In the sections below, the modules that a specific booster works on are shown in parenthesis in the heading.

Bling3D.StoreInDom (all modules)

Stores a reference to a module in the HTML element that triggered the module. This is useful if you need to gain access to the JavaScript module object to perform API calls. The reference will be stored in a property called `bling3d`.

```
<img id="my-image" data-b3d-image-tile data-b3d-store-in-dom src= "image.jpg" >
```

Get the reference back from the element like this:

```
var element = document.getElementById("my-image");
if (!element.bling3d) return;    // Could be undefined if module not initialized.
var module = element.bling3d;
```

If you are subsequently deleting elements from the DOM that have the `bling3d` property set, it would be prudent to delete the property or set it to `null` to ensure memory is freed.

Bling3D.CoverFlow (Bling3D.PathSlider)

Adds ‘cover-flow’ style animation to path sliders. Works best with linear-style sliders.

Bling3D.ViewportInput (Bling3D.Viewport)

Adds default mouse, wheel and touch behaviour to viewports. Handles swiping, clicking scrolling and other user-input actions.

Properties

Property	Description	Type	Default Value
`composer`	Working out the positions and angles of great looking cameras views can be tricky to do by trial and error. By setting `composer:true`, the viewport will enter a mode where the camera can easily be manipulated with mouse, touch and keyboard. When you are happy with the camera angle and positions, just copy the six numbers displayed in the top left of the screen into the camera setup.	Boolean	`false`

Bling3D.AsDefault (all modules)

Sets the last set of module properties as the default properties for that module. This allows for more concise HTML setup:

```
<div data-b3d-path-slider >
    <img data-b3d-image-tile = "frameColor: 'red', depth:0.1"
         data-b3d-as-default
         src='...'
    >
    <img src='...' data-b3d-image-tile>
    <img src='...' data-b3d-image-tile>
    <img src='...' data-b3d-image-tile>
    <img src='...' data-b3d-image-tile>
</div>
```

In the above example, only the first image tile needs to have a `frameColor` and `depth` set. Because `data-b3d-as-default` is being used, all subsequent image tiles will assume the same values for `frameColor` and `depth`.

Bling3D.ChangeCamera (all modules)

Add camera changing ability to HTML elements. When a user clicks the element, Bling3D will switch to the chosen camera. If the HTML element triggers a module (e.g. `data-b3d-image-tile`), then the 3D object can also be clicked to change the Camera.

```
// Single button camera.
<button data-camerachange="...">Camera 1</button>

// Select camera.
```

```
<select data-camerachange="...">
    <option value="camera1">Camera 1</option>
    <option value="camera2">Camera 2</option>
</select>
```

Properties

Property	Description	Type	Default Value
`name`	The unique identifier of the camera to change to. 'defaultCamera', will switch to the viewport’s default camera.	String	`''`
`triggerEvent`	Space–separated list of JavaScript events that trigger the camera change. If using input type elements like `<select>`, then use use `triggerEvent: 'change'`	String	`'click'`