Last updated: June 16, 2018

Technical integration guide

Overview

Cimmerse uses the WebGL API [1] and javascript to render 3D and instant AR real time in all modern web browsers, mobile and desktop. It uses the WebVR API [2] to render a Virtual Reality scene directly to supported VR hardware in supported browsers. And the WebXR API [7] is used to render a full, 6DOF AR experience in supported browsers.

The main components of the Cimmerse platform is cloud-based backend where digitized 3D models are uploaded and hosted (the “model repository”), and a client-side application that renders the 3D scene on the screen and in VR (the “viewer”).

Workflow

The workflow to add a 3D model to Cimmerse is:

1. The physical product is digitized, typically using photogrammetry. Existing 3D models or hand-made models are also supported.
2. The 3D model is uploaded to the model repository using Cimmerse’s web-based admin tool.
3. Lighting, positioning, maps and other render settings are adjusted in the admin tool to give product optimal render conditions.
4. The client receives a unique embed script url to add to their site in pages where 3D models should be appear. The script can be added using Google Tag Manager if available, through a content management system if one is used, or by a developer with access to update the code.
5. Depending on the case, the script can either use the url or information on the page to figure out what 3D to render, or a developer can control the specific model along with other configuration through adding additional code.

Cimmerse staff will typically handle step 1-3.

Client-side integration

The viewer can be embedded into any web site by inserting a script tag referencing a javascript hosted on the Cimmerse server. The script typically inserts an iframe into an HTML container element existing on the site, overriding the previous content (assumed to be an image placeholder while the 3D loads), but it can be customized to work differently. The inserted iframe will attempt to always be 100% width and height of the container. This gives the host site full control over where and at what size the viewer should be rendered.

The script tag to insert is:

```
<script async src="https://embed.cimmerce.com/[your-url]"></script>
```

The script tag is loaded asynchronously after the rest of page has loaded to avoid any impact on initial page load time. During the testing phase, the script will typically be configured to only render if a certain querystring such as “?cimmerse=enabled” is present in the url. This way, the script can safely be inserted into a production site without having any visible impact until the testing phase is finished and the client is ready to go live. Due to the “async” attribute the script can be inserted anywhere on the page without affecting initial page load performance.

We highly recommend that the container element contains a product image. This will be displayed while the script and iframe is loading, and if the script detects that the browser is not supported, the product image is still there and the page is fully functional.

Once the script is loaded and the iframe is initialized, the 3D model will render and blend seamlessly into the website, replacing the product image in the container. Technically this happens by replacing the target element, so the recommended html structure is something similar to this:

```
<div id="MyContainer">
  <div id="CimmerseTarget">
    <img src="my_preview_image.jpg" />
  </div>
</div>
```

Continuing the example above, once the 3D model has loaded, the “CimmerseTarget” element will be replaced, and the final DOM will look something like this:

```
<div id="MyContainer">
  <div id="CimmersePlaceholder">
    <iframe src="..." />
  </div>
</div>
```

The “CimmersePlaceholder” element is generally always styled to scale to a 100% width and height of it’s parent element, but this can be changed or even configured dynamically at run time. Alternatively, it can be unstyled from our side and left to be styled by the host site. Please

note the “CimmersePlaceholder” element must have some width and height defined to be visible on the page.

Tracking

If Google Analytics are found on the host page, a number of events are automatically pushed to Google Analytics using the event category “Cimmerse”. The event action is one of:

- model-rendered
- enter-vr
- exit-vr
- enter-fullscreen
- exit-fullscreen
- exit-fullscreen
- toggle-menu
- start-rotating
- start-zooming
- start-panning

By default, no label is applied to the event, but one can be specified using the trackingLabel configuration.

Configuration

We support plug-and-play integration where just the script need to be inserted and it handles the rest, but we also allow the host site a lot of control in cases where that is required.

There are two sets of configurations that can be provided by making them available as callable functions on the window object before the embed script tag:

```
<script>
window.getCimmerseTenantConfig = function() {
 return {
   modelId: '[string]',
   targetSelector: '[string]',
   trackingLabel: '[string]',
   visibleWhileLoading: [true / false],
   placeholderStyles: '[string]',
   limitInteractionAreaOnMobile: [true / false]
 }
}
```

```
window.getCimmerseViewerConfig = function() {
 return {
   transparentMode: [true / false],
   autorotateOverride: [number],
   interactionTip: ['standard', 'discrete', 'hidden'],
   materialmaps: {
     '[map name]': '[map url]'
   },
   onLoadCallback: [function],
   onUserInteractionCallback: [function]
 }
}
</script>
```

getCimmerseTenantConfig can return these options. If supplied, a few are required:

Name	**Description**	**Required**	**Example**
modelId	The unique identifier of the 3D model	Yes	“8b1d41b3-220f-4ffc-811e-cb43c1b0b61a”
targetSelector	A DOM selector that locates the container element for the iframe when passed to document.querySelector	Yes	“#modelViewer”
trackingLabel	Used as label when tracking events into Google Analytics.	No	“landingpage”
visibleWhile-Loading	Show preview image and load progress immediately	No	true
placeholder-Styles	Styling to apply to the CimmersePlaceholder element at runtime.	No	width: 500px;
limitInteraction AreaOnMobile	Narrows the part of the viewer that can be interacted with for easier scrolling on mobile.	No	true

getCimmerseViewerConfig can return these options:

Name	**Description**	**Default**	**Example**
transparent-Mode	Disables floor, skybox etc.	false	true
autorotate-Override	A float (positive or negative) which overrides the default autorotation for the model	-	0.5 -3
interactionTip	Type of help to indicate that the 3D model can be interacted with.	“standard”	“standard” “discrete” “hidden”

materialmaps	Specifies one or more materials that have their map replaced with the given url. The url must support CORS from the host site.	-	{
			
			'ContentPlaceholder': 'https://example.com/maps/1.jpg'
			}

onload	Callback invoked when model is fully loaded and rendered	-	function() { console.log(‘Done!’) }
oninteraction	Callback invoked with one argument when user interacts with the model. The argument is one of: ● enter-vr ● exit-vr ● enter-fullscreen ● exit-fullscreen ● exit-fullscreen ● toggle-menu ● start-rotating ● start-zooming ● start-panning	-	function(event) { console.log('User did this', event) }

Supported hardware and browsers

We support 3D rendering in the newest versions of all major browsers on all major platforms and devices. We support instant AR (augmented reality with 3 degrees of freedom) in all mobile browsers. We support full AR (augmented reality with 6 degrees of freedom) only on iOS currently, with Android support coming soon. We support VR in any browser where WebVR is enabled.

OS	**Browser**	**3D**	**Instant AR**	**WebXR**	**VR**
iOS	Safari 8+	Yes	Yes	No	No
	WebXR Viewer	Yes	No	Yes	
Android	Chrome 40+	Yes	Yes	No	No
	Oculus Browser	Yes	Yes	No	Gear VR [6]
	Samsung Internet				
Windows	Chrome 40+	Yes	No	No	Oculus Rift [4] HTC Vive [5]
	Firefox 35+ (55+ for VR)	Yes			
	Edge 12+	Yes	No	No	No
	IE 11	Yes	No	No	No
MacOS	Chrome/Safari	Yes	No	No	No
	Firefox 35+ (55+ for VR)	Yes	No	No	Oculus Rift [4] HTC Vive [5]

Notice: WebVR is still an experimental feature in some browsers. Firefox 55+ supports WebVR on Windows. For Chrome on Windows, a flag need to enabled by the user for VR support. The mobile browsers already have WebVR enabled where applicable.

References

1. WebGL documentation: https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API
2. WebVR introduction: https://webvr.rocks/

3. Google Cardboard: https://vr.google.com/cardboard/
4. Oculus Rift: https://www.oculus.com/rift/
5. HTC Vive: https://www.vive.com/
6. Samsung Gear VR: https://www.oculus.com/gear-vr/
7. WebXR API: https://immersive-web.github.io/webxr/