VRinVET

Co-funded by the European Union

Crafting VR Videos: Do You Need to Code?

Virtual reality video creation sits at a fascinating crossroads between filmmaking, software development, and interactive design. Whether you're a seasoned developer or a creative professional with no coding background, the world of VR content has a path for you. This presentation unpacks the tools, techniques, and trade-offs — so you can make an informed decision about where to begin.

IMMERSIVE MEDIA GUIDE

BEGINNER TO ADVANCED

Funded by the European Union. Views and opinions expressed are however those of the author(s) only, and do not necessarily reflect those of the European Union or European Education and Culture Executive Agency (EACEA). Neither the European Union nor the granting authority can be held responsible for them.

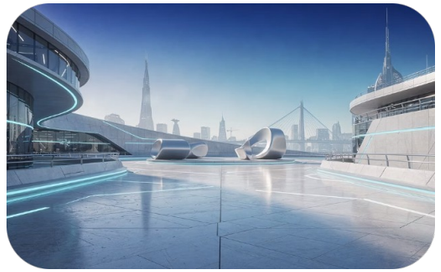
CHAPTER 1

The Immersive Landscape

Before diving into tools and code, it's essential to understand what VR video creation actually means — and why it's become one of the most compelling frontiers in modern digital media. This chapter sets the scene.

What is VR Video Creation?

VR video creation is far more nuanced than simply recording footage with a 360° camera. It involves a deliberate blend of real-world capture, digital environment design, and interactive experience architecture. Understanding these components helps clarify why different projects demand different skill sets.



Blending Real & Virtual

VR video creation often involves merging live-action footage captured with omnidirectional cameras with digitally constructed virtual environments. The result is a seamless hybrid space where viewers feel genuinely present — whether they're exploring a real location or a fully imagined world. Stitching and post-processing software bridge the gap between capture and experience.

Interactive 360° Experiences

Unlike passive video, VR experiences invite the viewer to look in any direction, interact with objects, and influence the narrative. Creating interactivity requires an understanding of spatial audio, gaze-based triggers, and controller input — elements that go well beyond traditional video editing workflows.

Delivery Through Headsets & AR

VR content is consumed through a growing ecosystem of devices — from standalone headsets like the Meta Quest to tethered PC VR systems and AR glasses. Each platform has its own technical requirements, resolution targets, and rendering constraints, all of which shape how you design and optimise your VR video content.

The Power of Immersive Content

The shift from flat-screen video to immersive VR experiences isn't merely cosmetic — it fundamentally changes how people receive, process, and remember content. Research consistently demonstrates that immersive media produces measurably stronger outcomes across education, marketing, training, and entertainment.

🎯 Engagement

VR videos captivate audiences in ways that flat media simply cannot replicate. By placing the viewer inside the scene rather than in front of it, immersive content commands full attention. Studies show that VR content produces up to **4× higher engagement rates** than traditional video formats, reducing distraction and increasing time-on-content significantly.

💡 Impact

The sense of physical presence that VR creates fosters a much deeper emotional connection to the subject matter. Whether a viewer is walking through a war-torn landscape for a documentary or rehearsing a high-stakes sales pitch, the emotional authenticity of VR accelerates understanding, empathy, and retention — outcomes that traditional training and storytelling struggle to match.

🚀 Innovation

VR opens entirely new creative and commercial avenues. From virtual field trips in schools and remote surgical training in medicine, to architectural walkthroughs and live event broadcasting — the applications are vast and rapidly expanding. Brands, educators, and storytellers are only beginning to explore the medium's full potential.

CHAPTER 2

The Developer's Toolkit

To build truly powerful VR video experiences, developers rely on a layered stack of technologies — each serving a distinct purpose. This chapter introduces the three pillars of web-based VR development: WebXR, Three.js, and A-Frame.

WebXR: The Foundation for Web-Based VR

WebXR is the bedrock upon which modern browser-based virtual and augmented reality experiences are built. Introduced as the successor to the now-deprecated WebVR API, WebXR was designed from the ground up to handle the full spectrum of immersive experiences — from simple 360° video playback to fully interactive mixed-reality environments.

Understanding what WebXR does — and crucially, what it *doesn't* do — is essential for anyone entering VR development. It is not a rendering engine. It won't load your 3D models, draw your scene, or manage your assets. Instead, it performs a highly specialised role: it communicates with the underlying hardware to manage the timing, pose estimation, and session lifecycle of an XR experience.

WebXR replaced the legacy WebVR API and is now the W3C-recommended standard for immersive web experiences across all major browsers and devices.

Core Responsibilities of WebXR

- **Session Management:** Initiates and terminates VR/AR sessions, handling device handshaking and permissions gracefully.
- **Frame Timing:** Provides precise per-frame timing data to synchronise rendering with the headset's refresh cycle — crucial for avoiding motion sickness.
- **Viewpoint Tracking:** Delivers head pose and eye position data so the scene can be rendered correctly from the viewer's perspective.
- **Controller Input:** Supports a broad range of input sources — gamepads, hand tracking, gaze — with a unified input model.
- **AR Support:** Unlike WebVR, WebXR natively supports augmented reality sessions, enabling mixed-reality content in the same API.

Three.js: Bringing 3D to Life

If WebXR is the foundation, Three.js is the engine room. It is one of the most widely used JavaScript libraries in existence, abstracting the notoriously complex WebGL API into an approachable and expressive toolkit for 3D graphics in the browser.

What Three.js Handles

Rendering Pipeline

Three.js manages the entire WebGL rendering pipeline, from setting up the canvas context to issuing draw calls. The `WebGLRenderer` handles shadows, anti-aliasing, and tone mapping out of the box.

Scene Graph & Cameras

Organise objects in a hierarchical scene graph and control them via a variety of camera types — perspective, orthographic, and stereo — essential for VR's split-screen rendering.

Materials & Lighting

A rich system of physically based materials, dynamic lighting, and shadow maps allows for realistic or stylised visuals depending on your creative direction.

VR with Three.js: A Quick Example

Creating a basic AR scene in Three.js involves a few key steps. You initialise a `WebGLRenderer`, enable its XR mode with `renderer.xr.enabled = true`, and attach a `VRButton` to enter the session. A simple coloured box can then be placed in 3D space:

```
const geometry = new THREE.BoxGeometry(0.1, 0.1, 0.1);
const material = new THREE.MeshBasicMaterial({
  color: 0x00aaff
});
const mesh = new THREE.Mesh(geometry, material);
scene.add(mesh);
```

This minimal snippet illustrates how Three.js reduces complex WebGL boilerplate into readable, maintainable code — making it an ideal stepping stone toward full VR development.

A-Frame: The Declarative VR Framework

A-Frame was created by Mozilla to democratise VR development on the web. By sitting on top of Three.js and using a familiar HTML-like syntax, A-Frame allows designers, educators, and developers with modest coding experience to build fully functional VR scenes in a matter of hours — not weeks.

1

HTML-Like Syntax

A-Frame scenes are written as standard HTML documents. You declare a `<a-scene>` tag, then nest primitives like `<a-box>`, `<a-sphere>`, and `<a-videosphere>` within it. No JavaScript is required to get started, dramatically lowering the barrier to entry for non-developers.

2

Entity-Component Framework

Beneath A-Frame's simple surface lies a powerful **Entity-Component-System (ECS)** architecture — the same pattern used by professional game engines like Unity. Every element in a scene is an entity, and behaviour is attached by composing components. This makes A-Frame both beginner-friendly and architecturally sound for complex projects.

3

Extensibility & Community

A-Frame has a vibrant ecosystem of community-built components available via the **A-Frame Registry**. Need particle effects, physics, or network multiplayer? Chances are there's already a component for it. For those who need even more power, A-Frame exposes the underlying Three.js API directly, making it a genuine hybrid between no-code and pro-code workflows.

4

Ideal Entry Point

For educators building virtual field trips, journalists crafting immersive documentaries, or designers prototyping spatial interfaces, A-Frame is the ideal first tool. Its shallow learning curve means you can produce a compelling, shareable VR experience — hosted as a simple webpage — without touching a rendering pipeline or shader language.

CHAPTER 3

Coding vs. No-Code: What's Your Path?

One of the most common questions newcomers ask is whether they need to learn to code before they can make meaningful VR content. The honest answer is nuanced — and that's exactly what this chapter explores.

The "No-Code" Approach: A-Frame's Simplicity

A-Frame's primitives make it genuinely possible to create compelling VR video experiences with very little — and in some cases, *zero* — JavaScript. For creators who want results quickly, the following primitives are your most powerful allies.

`<a-video>` — Embedding 2D Video

The `<a-video>` primitive allows you to place a flat video screen anywhere within your 3D VR environment. Think of it as placing a cinema screen inside a virtual room. You simply reference a video asset and set position, rotation, and size attributes — no rendering knowledge required. This is ideal for hybrid experiences where you want to combine spatial environments with traditional video content.

```
<a-assets>
  <video id="clip" src="video.mp4"
         autoplay loop></video>
</a-assets>
<a-video src="#clip"
         width="3" height="1.7"
         position="0 1.5 -3">
</a-video>
```

`<a-videosphere>` — Full 360° Immersion

The `<a-videosphere>` primitive is where A-Frame truly shines for VR video creators. It maps a 360° equirectangular video onto the interior of a sphere, placing the viewer at the centre. The result is a fully immersive environment where the viewer can look in any direction. Adding this to a scene takes a single line of HTML:

```
<a-videosphere
  src="360-video.mp4"
  rotation="0 -90 0">
</a-videosphere>
```

With autoplay policies, basic event listeners (just a few lines of JavaScript) can handle play/pause on user interaction — keeping the code footprint minimal while delivering a polished outcome.

A fully functional immersive 360° video player can be built with A-Frame in under 20 lines of HTML — no prior coding experience required.

When Coding Becomes Essential

While A-Frame's no-code approach is genuinely powerful for video-centric VR, there are clear thresholds beyond which deeper programming knowledge becomes not just helpful, but **necessary**. Understanding these thresholds helps you plan your learning journey and set realistic expectations for what you can build at each stage.

Advanced Interactivity

Complex game mechanics — physics simulations, collision detection, procedural animation, AI-driven NPCs — require JavaScript or a dedicated game engine. Simple gaze triggers are achievable without code; branching narratives with state management are not.

Custom Shaders & Effects

Achieving a distinctive visual aesthetic — chromatic aberration, volumetric fog, stylised cel-shading, or custom post-processing — demands GLSL shader programming. These are GPU-executed programs that sit outside the reach of any declarative framework.

Performance Optimisation

At scale — high-resolution 360° video, dense interactive environments, or mobile VR targeting low-end hardware — you'll need to manually manage draw calls, implement level-of-detail (LOD) systems, and profile GPU memory usage. These tasks require intimate familiarity with the rendering pipeline.

External Integration

Connecting your VR experience to live data feeds, REST APIs, WebSocket streams, or user authentication systems requires server-side knowledge and careful client-side programming. Real-time data overlays in VR — stock tickers, live sports stats, IoT sensor readings — fall firmly into the coding-required category.

The Hybrid Approach: Leveraging Frameworks

The most practical path for many VR creators lies between the extremes of pure no-code and full custom development. A hybrid approach — using A-Frame's accessible syntax while selectively dipping into Three.js or raw WebXR — gives you the best of both worlds: rapid development for most features, with the power to go deeper when needed.

A-Frame + Three.js: Best of Both Worlds

Every A-Frame scene exposes the underlying Three.js scene, camera, and renderer. This means you can use A-Frame's HTML syntax to scaffold your environment and then access `el.sceneEl.object3D` to manipulate Three.js objects directly. Advanced lighting rigs, custom geometry, and particle systems can all be injected into an A-Frame scene this way — without abandoning the declarative structure you're comfortable with.

VRButton.js: Simplified Session Entry

Mozilla's `VRButton.js` utility (part of the Three.js examples library) adds a single-click VR session entry button to any Three.js project. It handles browser compatibility checks, device detection, and session initiation automatically — abstracting away some of the more tedious WebXR session management code and letting you focus on your experience.

Animation Loops in VR

Standard `requestAnimationFrame()` does not work correctly in VR contexts. For Three.js VR projects, `renderer.setAnimationLoop(callback)` is the required replacement. It integrates with the WebXR session's frame timing, ensuring your animations stay synchronised with the headset's render cycle and preventing the juddering that breaks immersion.

Hybrid Stack at a Glance

A typical hybrid VR video project might look like this:

- **Scene Structure:** A-Frame HTML primitives
- **Video Playback:** `<a-videosphere>`
- **Custom UI:** A-Frame components in JavaScript
- **Special Effects:** Three.js injected via `object3D`
- **Session Entry:** `VRButton.js`
- **Animation:** `renderer.setAnimationLoop()`

This stack lets a single developer move from prototype to polished product without switching frameworks mid-project.

CHAPTER 4

The Future of VR Video Creation

The tools and techniques of today are only the beginning. As hardware becomes more accessible, AI more capable, and streaming infrastructure more robust, the ceiling for what VR video can achieve continues to rise dramatically.

Beyond Basic Video: Interactive Narratives

The next frontier of VR video isn't just higher resolution or wider field of view — it's the fusion of pre-recorded footage with live data, AI-generated content, and genuinely branching narratives. These emerging capabilities are transforming VR from a viewing medium into a participatory one.

Adaptive Streaming (DASH & HLS)

Delivering high-quality 360° video over the internet presents significant bandwidth challenges. Adaptive streaming protocols — **DASH (Dynamic Adaptive Streaming over HTTP)** and **HLS (HTTP Live Streaming)** — solve this by dynamically adjusting video quality based on the viewer's connection speed. Google's VR SDK for Unity integrates DASH support natively, enabling smooth spatial video delivery even on variable connections. Tiled streaming further optimises bandwidth by only delivering the highest resolution to the viewer's current field of view.

Real-Time Data Integration

Overlaying live information within a VR environment opens remarkable possibilities. Imagine a virtual stadium experience where player statistics float above athletes in real time, a surgical training simulation fed by live patient data, or a financial trading floor where market tickers exist as spatial objects around you. WebSockets and REST APIs bridge the gap between the live web and the VR scene, though building robust real-time integrations requires solid JavaScript fundamentals.

AI-Powered VR Content

Generative AI is beginning to reshape VR content creation at every level. Large language models can script branching narratives and NPC dialogue. Diffusion models can generate skyboxes, textures, and environment assets on demand. AI-driven motion capture can animate characters from minimal input. In the near future, entire virtual environments may be generated procedurally from a text prompt — radically reducing the cost and time required to produce rich, varied VR experiences.

Your VR Video Journey Starts Now

The question of whether you need to code to create VR video has a satisfying, empowering answer: **not necessarily — but the more you learn, the more you can create.** The ecosystem is deliberately tiered to welcome creators at every level, with clear pathways to grow as your ambitions expand.

Beginner — Start with A-Frame

Use `<a-videosphere>` and `<a-video>` primitives to publish your first 360° video experience as a webpage. No build tools, no complex setup — just HTML and creativity.

Intermediate — Explore Three.js

Layer custom JavaScript into your A-Frame projects, access the Three.js API for advanced visual effects, and begin managing scene state programmatically. Build interactive narratives and custom UI components.

Advanced — Dive into WebXR

Work directly with the WebXR Device API for full control over session management, custom input handling, GLSL shaders, and performance optimisation. Integrate streaming, live data, and AI-driven content for professional-grade immersive experiences.

The bottom line: You don't always need extensive coding knowledge to create VR video — but each level of skill you acquire unlocks dramatically greater creative potential. Start where you are, ship something real, then keep climbing.