Augmented ODV: Web-Driven Annotation and Interactivity Enhancement of 360 Degree Video in Both 2D and 3D

Maarten Wijnants(✉), Kris Van Erum, Peter Quax, and Wim Lamotte

Hasselt University – tUL – iMinds, Expertise Centre for Digital Media,
Wetenschapspark 2, 3590 Diepenbeek, Belgium
maarten.wijnants@uhasselt.be

Abstract. Despite recent technological innovations in the media authoring ecosystem, one example being the ability to video record a scene with a 360 degree or so-called omni-directional field of view, video consumption to date largely remains a lean-back type of endeavor. This paper fuses the *Augmented Video Viewing* paradigm with 360 degree video technology to give rise to *augmented Omni-Directional Video* (ODV), a novel content format that holds promise to more deeply engage viewers and that unlocks more active ways of interacting with omni-directional video footage. The augmented ODV principle is exposed through a collection of standards-compliant Web interfaces to ease adoption by developers. At the same time, its Web-driven design maximizes the applicability of the technology, both in terms of consumption platforms and usage domains. Two use case prototypes showcase the artistic expressiveness of the augmented ODV methodology, while performance evaluation results establish the modest computational footprint of its implementation.

Keywords: Augmented Video Viewing (AVV) · Omni-Directional Video (ODV) · 360 degree video · Interactive video · Augmented video · Hypervideo · Web technology · WebGL · JavaScript

1 Introduction

Recent advancements in the capturing and authoring process, both in terms of hardware and software, have paved the way for technological innovations in the media landscape. One such innovation is *Omni-Directional Video* (ODV) or so-called 360 degree video. As its name implies, ODV content refers to video footage that is recorded with a 360 degree (i.e., cylindrical or spherical) Field of View. Typical ODV player implementations allow viewers to freely adapt their viewing angle inside the omni-directionally captured video scene.

Technical evolutions like the ODV concept unfortunately cannot conceal the fact that typical media consumption environments continue to deliver largely passive, static and non-interactive experiences, just like they did upon their inception at the beginning of the 20th century. For sure, by affording the ability to spatially navigate through a 360 degree video scene in real-time and in an unconstrained fashion, ODV technology holds important promises in terms of granting viewers an enhanced feeling of belonging and immersion when compared to traditional video. Unfortunately, besides perspective personalization, no additional advanced interaction options are scaffolded by typical ODV installations. Interactivity has nonetheless proven to be a vital tool to attract the attention of video consumers and to maximize viewer retention (e.g., [1]).

To mitigate the laid-back characteristics of video as a medium, we have previously proposed the *Augmented Video Viewing* (AVV) paradigm and its associated Web-compliant codebase [2]. The AVV mindset aims to factor in interactivity and dynamism in the core fabric of the video consumption process. In effect, typical AVV experiences offer users interaction possibilities that go well beyond basic playback control. This is accomplished in a pragmatic fashion, by embellishing traditionally produced (passive) video material with interactive constructs that are exclusively realized using Web technologies. In particular, out-of-the-box HTML5, CSS and JavaScript features are leveraged to respectively define the structure and substance of the interactive assets, their styling, and their dynamic behavior (e.g., their spatio-temporal constraints). The interactive constructs are encoded as *hotspots* that are superimposed on top of the video playback in order to closely integrate the video content with its interaction provisions.

In this article, we give rise to the *augmented ODV* concept by mapping the AVV methodology to ODV data, this way effectively converting the latter from a passive into an interactive content format. All technical measures that had to be taken to reconcile the two constituting technologies will be described in detail and, in this process, our solution's far-reaching integration with contemporary Web standards will be emphasized. At the same time, the computationally lightweight nature of the augmented ODV paradigm will be underscored by presenting the outcome of an extensive performance benchmark. These prime achievements are accompanied by a total of three peripheral scientific contributions. First, through the presentation of two representative prototype realizations, we provide a hint of the advantageous traits of augmented ODV with regards to end-user envelopment, engagement in and participation with (omni-directional) video content. Secondly, the showcased prototypes offer a glimpse of the creative and artistic design options that are unlocked by the augmented ODV concept. In effect, augmented ODV is expected to pave the way for the delivery of a whole new breed of highly interactive, compelling and engaging video sensations in a myriad of application domains. As such, the augmented ODV approach holds important business opportunities for content producers by allowing them to cater to and capitalize on new consumer profiles. Finally, the third contribution of this article takes the form of a Web interface that offers video artists a graphical toolkit to facilitate the authoring and editing of rich AVV-based interactive (omni-directional) video experiences.

It is explicitly stressed here that the focus of this article is purely on the technological solutions that substantiate the augmented ODV paradigm and that jointly constitute a completely interactive and accessible system for the consumption and creation of augmented ODV experiences. As such, formal user experience validation is out of scope with regards to this publication. It is advocated to qualitatively assess user-oriented metrics on a case-by-case basis anyway, as they tend to depend largely on the content at hand and on the actual augmented ODV application scenario.

2 Related Work

The elementary notion of extending video content with interactive traits has already been studied and approached from a number of different perspectives. Meixner et al. have published an impressive and fairly up-to-date reference work pertaining to this subject [3]. In particular, they have devised an extensive taxonomy of solutions described in the academic literature by classifying them into four categories: *interactive video*, *annotated video*, *non-linear video* and *hypervideo*. Instead of needlessly recapitulating their work here, this section will deliberately concentrate on a critical selection of the most relevant scientific efforts only.

Given its Web-focus, the AVV paradigm is most akin to the *hypervideo* category of related systems identified in Meixner et al.'s taxonomy. The basic hypervideo objective consists of translating the hyperlink concept that we have become familiar with via the Web (mostly in textual form) to the video medium. As such, a hypervideo can be defined as a video document in which one or more user-clickable anchors are embedded. Two pioneering contributions in this research domain were delivered by the *HyperCafe* [4] and *HyperSoap* [5] experiments. Other notable academic hypervideo contributions include *Hyper-Hitchcock* and its *detail-on-demand* video concept [6], the *non-linear video* approach proposed by Fraunhofer Institute FOKUS [7], *HyLive* due to its emphasis on live broadcast scenarios [8], *SIVA Producer* [3], the *Component-based Hypervideo Model* (CHM) by Sadallah et al. [9], and the *360° hypervideo* solution proposed by Neng and Chambel [10].

Hypervideo-inspired systems have also been developed outside of the academic world, either in the form of freeware frameworks or commercialized offerings. Examples include cacophony.js (http://www.cacophonyjs.com), Mozilla's Popcorn.js HTML5 media framework (http://popcornjs.org), Gravidi (http://www.gravidi.com), and YouTube Video Annotations (http://www.youtube.com/t/annotations_about).

A somewhat different cluster of related work is that of Web-compliant multimedia presentation technologies. Systems in this solution category aim to deliver online interactive experiences involving a mixture of media types (i.e., they are not necessarily video-centric). A representative example of such a technology is *SMIL State* [11]. Unfortunately, native support for SMIL State is still largely lacking in contemporary Web browsers.

This article describes a pragmatic, Web-compliant approach to attach interactive features to ODV content. With the exception of the *360° hypervideo* solution by Neng and Chambel, none of the technologies cited above have explicitly considered ODV content. As a result, it is highly questionable whether they will be directly applicable to ODV contexts.

The proposed augmented ODV methodology and the *360° hypervideo* system are similar in the sense that they are both Web-driven. However, it is unclear how expressive the latter solution is in terms of overlay element specification. Our approach imposes no creative boundaries whatsoever in this regard. Also, the solution by Neng and Chambel appears to lack a visual authoring environment. Finally, the computational overhead imposed by the *360° hypervideo* system is unknown. In contrast, the performance analysis presented in Sect. 8 will establish that augmented ODV applications are readily consumable on commodity hardware, including tablet devices.

3 ODV Web Player

The prototypes that will be demonstrated later on in this paper have all been realized on top of an in-house developed ODV player for the Web. Although not the focus of this work, we will briefly touch on the player's functionality, design and implementation in this section to grant readers a comprehensive insight into our contributions. It is important to note however that the applicability of the AVV paradigm is by no means confined to this particular ODV player implementation; instead, the AVV codebase can readily be ingested in any Web-compliant ODV setup.

3.1 Functionality

The ODV player presents users a spatially restricted viewport into the omnidirectional video scene that is controllable via the Pan-Tilt-Zoom (PTZ) principle. This implies that viewers are granted directional control in two dimensions with regard to the positioning of the view window. At the same time, users can zoom in and out in order to narrow or widen the spatial spread of the viewport, respectively. On the horizontal axis, the player imposes no navigational restrictions. Stated differently, viewers can perform seamless 360 degree panning as they see fit. In the vertical direction on the other hand, the tilt movement is confined to 180 degrees in order to reduce the cognitive load on the viewer and to anticipate potential motion sickness. Intuitively, this means that users can freely look left and right, but cannot loop in their tilt movement (see Fig. 1).

3.2 Implementation

The ODV player is intended to be embedded in a HTML page. It is completely Web-compatible in the sense that it exclusively leverages standardized HTML5 technologies instead of resorting to the use of third-party plug-ins like Adobe Flash or Microsoft SilverLight. Input-wise, the player expects the ODV content to have undergone an equirectangular projection (see again Fig. 1).

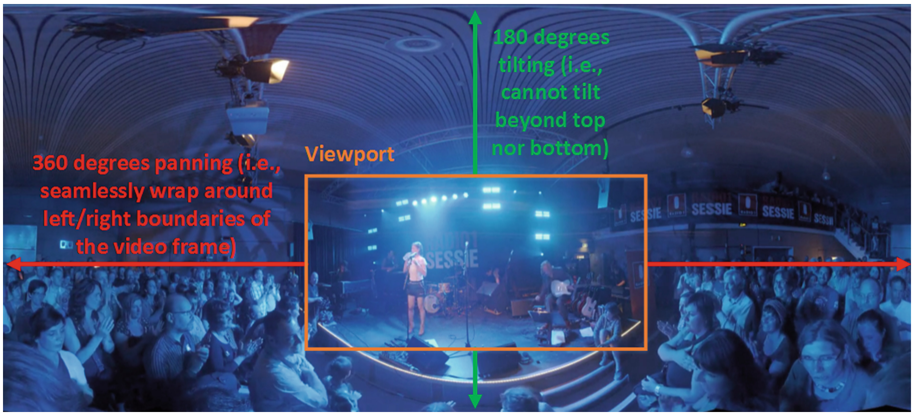

180 degrees tilting (i.e., cannot tilt beyond top nor bottom)
360 degrees panning (i.e., seamlessly wrap around left/right boundaries of the video frame)
Viewport

Fig. 1. Equirectangular projection of a single full frame of example ODV content, with an indication of the pan and tilt limitations imposed by the ODV player.

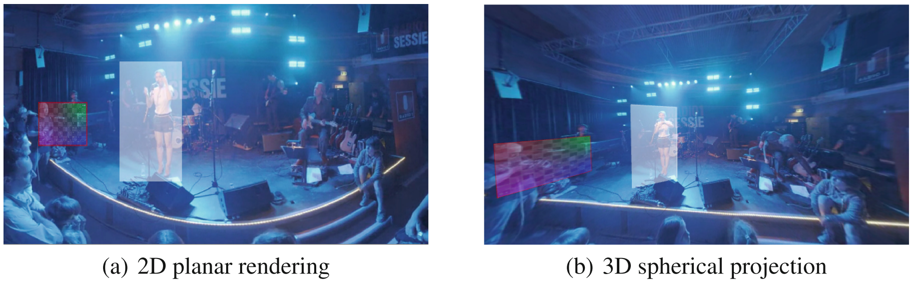


Fig. 2. The two instantiations of the employed ODV Web player. The 2D planar implementation introduces noticeable visual distortions which are induced by the equirectangular representation of the input ODV content.

Two alternative versions of the player exist that differ in the way ODV content is rendered and presented to the viewer. The first implementation adopts a two-dimensional (i.e., planar) rendering approach in which the currently active viewport is directly cropped from the equirectangular projection and subsequently rendered inside a standard HTML5 `<canvas>` element. The second version relies on WebGL (Web Graphics Library), a JavaScript API for plug-in-less rendering of hardware-accelerated 3D graphics inside Web browser instances [12]. Here, the ODV frames are textured onto the interior of a 3D sphere rendered through WebGL. Users control a virtual camera that is positioned inside this sphere in order to define their viewport into the ODV footage.

Both instantiations of the ODV player have their merits and shortcomings. First, as will be examined in Sect. 8, the planar visualization scheme generally has a lower computational expense compared to the 3D WebGL solution. Secondly, the 2D implementation only leverages basic HTML5 and JavaScript functionality, which maximizes portability. In comparison, its 3D counterpart requires

WebGL API support, which unfortunately is not yet universally available. As an example, while the majority of desktop Web browsers were pretty eager to adopt WebGL, one of the major mobile platforms (i.e., iOS) has only done so in the latest revision of its operating system (which was released on September 17th, 2014). Third, informally conducted qualitative experiments involving the ODV player have indicated that viewers generally appreciate the ability to zoom out the viewport to a global overview level (akin to the full frame visualization illustrated in Fig. 1). While the 2D instantiation can readily support this kind of behavior, the WebGL variant cannot (due to the fact that the movement of the virtual camera in this case is bounded by the interior of a sphere). Finally, concerning graphical fidelity, the WebGL-based ODV implementation outperforms its planar peer. This is evidenced in Fig. 2, which correlates the way the two player implementations visualize an ODV recording made during a small-scale concert that took place on a rectangular stage. The 2D planar visualization causes visual deformations to arise that are not present in the 3D spherical projection. This is best witnessed in the ill-shaped visualization of the podium in the 2D implementation.

4 Augmented Video Viewing

The basic premise of the AVV mindset consists of transforming video consumption from a purely passive, laid-back activity into a much more dynamic and (inter)active pursuit. This philosophy has bearing on a plethora of application contexts and use cases, including entertainment, online video-driven advertising and video-assisted remote tutoring. For example, in the video entertainment industry, the ultimate objective of content authors consists of producing thrilling and engaging drama that succeeds in not only captivating but also retaining the attention of the audience over time. However, due to the linear and non-interactive nature of classic video content, its initial appeal and viewer retention success largely depends on the skills of the different types of human operators who are involved in the content production pipeline (e.g., visual designers, camera operators, director). In case the presented content at some point fails to capture and engage the viewer, the risk arises that the viewer starts multitasking or even completely cancels the playback of the video. From the perspective of the content producer and distributor, this of course is highly unwanted behavior. Integrating interactive features in the video playback (e.g., in the form of gamification constructs [13]) holds promise to prevent the consumer from losing interest in and focus on the content.

As stated in the introduction, the AVV principle and its underlying implementation has previously been introduced [2]. Since then, the AVV codebase has been actively maintained and extended with additional functionality. In this section, the basic concepts of the AVV methodology will first concisely be recapitulated. Next, the newly developed AVV features will be described.

4.1 Interactive Video Overlays

The AVV framework offers a JavaScript API (the so-called JAVV API [2]) that facilitates the superimposing of interactive *video overlay elements* (VOEs) or so-called *hotspots* over a HTML5 video player. Overlay elements have spatio-temporal constraints that respectively define at which location and at what point during video playback they must be rendered. Spatial constraints are hereby not expressed as absolute screen coordinates but instead in terms of video coordinates (i.e., relative to the position of the video element in the surrounding HTML page). Furthermore, both static and animated positioning of video overlay elements is supported. Finally, overlay elements can have arbitrary programmatic logic associated with them. Typically, the execution of such logic is triggered by end-user interaction with the element or is timer-based.

Implementation-wise, each overlay element is represented by a `<div>` node in the HTML DOM. By assigning these dedicated nodes an elevated value for their CSS `z-index` property, it is guaranteed that they are always visibly overlaid on top of the video playback elements. HTML is exploited as markup language for the content that is embedded in overlay elements. This implies that hotspots can communicate a wide spectrum of information, as semantic information can be included through the application of appropriate HTML constructs. Analogously, the visual appearance and styling of overlay elements is controlled through the CSS standard. Finally, and again in line with the Web-focused design of the AVV framework, an overlay element's programmatic logic is expressed in JavaScript, with JavaScript's event-driven scripting model being exploited to couple dedicated interaction handlers to different types of viewer actions. In a desktop environment, for example, it will in many cases make sense to respond to `click`, `mouseover` and `mouseout` interactions that occur on the DOM representations of video overlay elements.

4.2 Motion-Tracked Video Overlays

The AVV framework comprises animation facilities for overlay elements. Looking at it from a software engineering perspective, the JAVV API codebase adopts the *strategy* design pattern to abstract overlay animation handling, this way introducing a certain level of flexibility in the animation subsystem. In particular, the software architecture defines an extensible family of interchangeable animation engines (each implementing a concrete animation style or technique), any of which can be attached at run-time to an overlay element in order to determine its spatial behavior over time. The initial version of the framework included only a single such animation engine, in particular one that implements a keyframing-like solution by performing linear interpolation between an overlay element's begin and end location over the course of the element's visible state [2]. Basically speaking, this animation engine causes overlay elements to follow a linear path.

Empirical insights obtained from developing concrete AVV-based test cases revealed that many scenarios would benefit from the ability to conceptually attach an overlay element to an in-scene object in the underlying video. In other

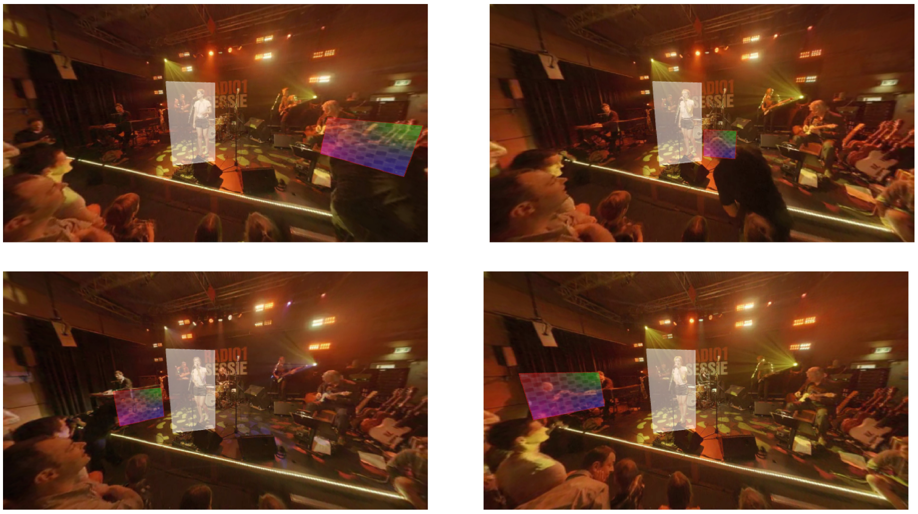


Fig. 3. Illustration of a *motion-tracked video overlay.* In this specific case, (the head of) an in-video actor was tracked so that an overlay element could be imposed on top of it.

words, we identified the need for an animation engine that enables overlay elements to follow the movement of subjects that appear in the video scene. To this end, we integrated a motion tracking animation engine in the JAVV API software architecture. This animation engine needs to be fed with cornerpin-based 2D tracking data as generated by off-the-shelf video editing software (e.g., Adobe After Effects or likewise) and subsequently applies the captured movement information to control both the position and spatial extent of the overlay element. As can be seen in Fig. 3, this animation engine is ideally suited to render overlay elements on top of (mobile) in-scene video items. Note that the visual accuracy of this animation engine largely depends on the performance and efficacy of the external algorithm that is responsible for implementing the motion tracking. Also note that the 2D motion tracking needs to be performed offline, as a pre-processing step.

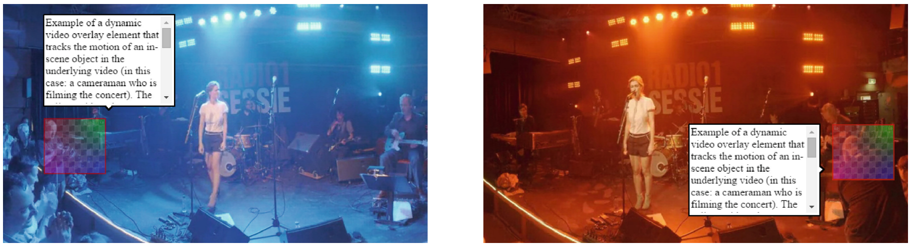
4.3 Reaction Video Overlays

Another desirable feature that emerged from practical experimentation with the JAVV API, was support for the specification of some form of parent/child relationship amongst individual overlay elements. Therefore, a specialized type of overlay element was implemented whose spatial positioning is defined relative to that of another hotspot (i.e., its parent). Since these types of overlays are typically visualized in response to some form of user interaction with the parent hotspot, they will in the remainder of this article be referred to as *reaction overlay elements.*

Three configuration settings control the spatial behavior of reaction overlay elements. The first configurable parameter is the absolute spacing that needs to be enforced between the reaction element and its parent. The reaction hotspot will always be offset the specified amount of pixels against the nearest edge of its parent. Secondly, experience designers can specify positioning preferences that define the order in which potential placement locations for the reaction overlay element are considered. These placement location alternatives are expressed relative to the parent item, at a high level of abstraction (i.e., to the left or right of the parent, above or below it, and so on). Reaction overlay elements are only rendered at a potential placement location in case sufficient screen real estate is available to host the element there. Since reaction items are never rendered partially, the following variables play a role in determining the eligibility of a placement location: the coordinates of the nearest border of the parent, the to-be-enforced spacing between the parent and the reaction element, the spatial dimensions of the reaction element and the spatial extent of the video player (or the view window in case of the ODV player, see Sect. 3.1). The third configuration setting is a Boolean flag indicating whether the reaction element must automatically be repositioned in case a placement location alternative that has precedence over the currently active one becomes available (e.g., due to changes to the viewport state in augmented ODV setups).

An advantageous trait of the reaction overlay element technology is that it interplays well with both animated parent items and viewport modifications in the augmented ODV player. In effect, reaction elements will follow the motion of their parent item and, in the course of the parent's animation, be dynamically repositioned as needed. This kind of behavior is illustrated in Fig. 4. Equivalent behavior is exhibited in the augmented ODV player when the user modifies his viewport into the ODV content.

One example of an interesting application area for the reaction overlay technology is the implementation of call-out widget-like functionality. As is demonstrated in Fig. 4, a reaction hotspot could present additional information about an in-scene object that is marked with a (potentially transparent) overlay element. In addition, by attaching some basic programmatic logic to the parent


Example of a dynamic video overlay element that tracks the motion of an in-scene object in the underlying video (in this case: a cameraman who is filming the concert). The

Example of a dynamic video overlay element that tracks the motion of an in-scene object in the underlying video (in this case: a cameraman who is filming the concert). The

Fig. 4. Parent object motion causes the relative location of its associated reaction element to be on-the-fly modified in accordance with the latter's positioning preferences (in this example, the preferred positioning order equals: above, right, below, left).

object, the visibility of the call-out box could dynamically be controlled on the basis of end-user interaction with the parent. As such, viewers can at run-time prevent the reaction overlay from cluttering the video playback at times when it is undesirable to do so. Please note that in this example, some simple HTML chrome was added to the appearance of the reaction element in order to give it an archetypical call-out look-and-feel.

5 Augmented ODV

This section will address the fundamental contribution of this article, namely the coupling of the JAVV API to ODV setups in general and to the ODV Web player that was introduced in Sect. 3 in particular. The incorporation in both instantiations of the ODV Web player will be discussed disjointly, due to the largely divergent technological requirements that were posed by each integration.

5.1 2D Planar Augmented ODV

Like any traditional video player, the 2D planar ODV renderer in essence presents two-dimensional images to the user. Due to this conceptual analogy, porting the AVV framework to this ODV setup turned out to be relatively straightforward. To be more precise, it mostly sufficed to correctly cover a number of "boundary cases" which all originate from the observation that, in contrast to a traditional video application, the 2D ODV implementation not necessarily displays complete video frames. In effect, depending on the ODV player's zoom level, the video footage might be cropped to fit the current viewport.

Conceptual sketches of each boundary case that had to be addressed are shown in Fig. 5. First of all, video overlay elements that integrally fall outside of the current viewport must be completely excluded from the rendering pipeline. Secondly, hotspots that are only partially contained in the viewport need to be clipped appropriately. This is achieved by modifying the value of the CSS `clip` property of the DOM representation of the involved overlay element. Finally, overlay elements with a large horizontal extent might need to be broken up into two discrete `<div>` representations for them to be rendered correctly.

Besides the boundary cases, the ODV player's support for zooming operations also required special attention. When zooming in or out, care has to be taken that overlay elements retain their logical positioning in the video scene. On the other hand, hotspots need to scale proportionally to the applied zoom level (in both the horizontal and vertical dimension). This latter requirement is fulfilled by applying the `scale` function of the CSS `transform` property to the concerned DOM element.

5.2 3D Spherical Augmented ODV

Whereas the JAVV API software architecture could quite readily be reconciled with the 2D ODV player implementation, this was not the case for its WebGL-based counterpart. In effect, in this latter instantiation, there is no longer a

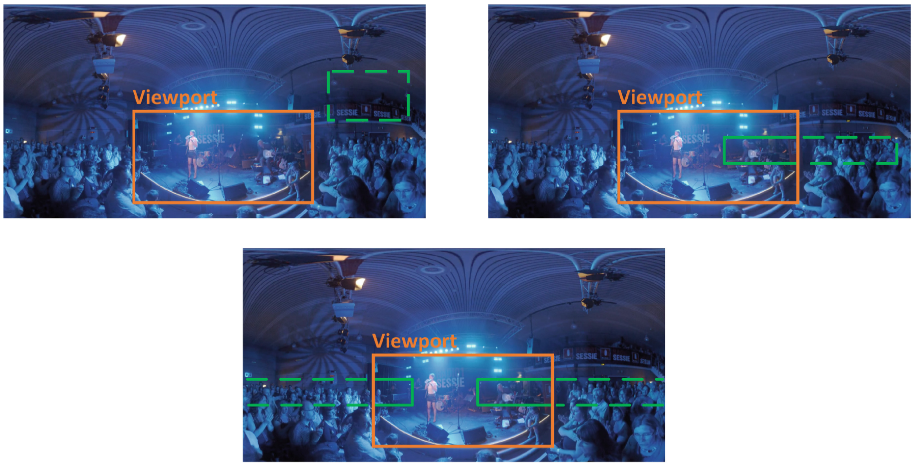


Fig. 5. Conceptual drawings of boundary conditions that had to be dealt with when porting the JAVV API to the 2D planar ODV setup. Dashed lines represent (portions of) hotspots that are not actually visualized.

notion of a planar presentation of video frames. Instead, the video content is applied to a curved surface (i.e., a sphere). If one would draw (flat) DOM elements as an exogenous layer on top of this three-dimensional scene, their graphical appearance would not necessarily blend in nicely with the ODV footage. Two alternative solutions were developed to remedy this problem.

CSS 3D Transformations. In the first approach, overlays remain to be represented by traditional DOM items, yet they are manipulated using CSS 3D transformations in order to correctly position them on top of the WebGL scene. In short, CSS defines a number of properties that allow DOM elements to be translated, rotated and scaled in a three-dimensional space [14].

We exploited the three.js JavaScript graphics library (http://threejs.org/), a convenient wrapper for WebGL, to mix the CSS 3D transformed DOM items with the WebGL scene. In particular, the three.js library allows for the addition of a CSS 3D renderer to an HTML page. This renderer is installed in such a manner that it spatially coincides with the WebGL renderer, yet in a superimposed manner.

Recall from Sect. 3.2 that the 3D ODV player applies the ODV footage to a curved surface. This causes the video content to be non-uniformly deformed (i.e., the deformation increases towards the edges of the viewport). The overlay elements must undergo an identical distortion for their visualization in the 3D space to be visually convincing. To this end, the two-dimensional video coordinates of each of the vertices that outline an overlay element, combined with the element's depth information (as specified by its CSS `z-index` value), are first expressed in a spherical coordinate system. The spherical coordinates (r, θ, ϕ)

are subsequently mapped to Cartesian positions (x, y, z) in 3D space [15] and are then fed to the CSS 3D renderer.

Representing Overlays as WebGL Objects. The second solution embraces the three-dimensional context not only for the visualization of the ODV material, but also for the rendering of the overlay elements. Like in the CSS 3D transformations scheme, the vertex outline of an overlay element is first spherically projected. Instead of providing the calculated 3D coordinates to the CSS 3D renderer, they are now directly interconnected in the 3D scene by means of WebGL-rendered lines. As a next step, the resulting shape is filled with a low-polygon 3D mesh. This approach enables the 3D representation of video overlay elements to display a background image (by attaching it as a texture to the mesh). Finally, the constructed mesh is equipped with a WebGL material so that, for example, a background color can be set for the overlay element.

The fact that video overlay elements are now rendered as 3D objects in a WebGL scene, as opposed to being represented as items in the DOM of the encapsulating webpage, has two important implications. First, it is no longer feasible to directly stylize hotspots via CSS. Therefore, CSS properties defining the visual appearance of overlay elements are parsed and (a subset of them) are translated to corresponding settings in WebGL. Examples of recognized CSS properties include border width, border color, background color and transparency level. A similar remark applies to the use of HTML to define hotspot content (e.g., an `<img>` tag is correctly translated to a WebGL texture, yet plain text in the HTML markup is simply discarded due to WebGL's poor text rendering support). Secondly, it invalidates the approach of handling end-user interaction with hotspots by listening for DOM events. As a workaround, a raycasting-based picking solution was adopted to determine which object(s) in the 3D scene the user is pointing at. In case the casted ray would intersect with multiple overlays, the one nearest to the virtual camera will be returned.

Comparison. The WebGL-integrated solution exhibits the detrimental characteristic that out-of-the-box support for the complete spectrum of existing CSS and HTML features is lost. Likewise, this implementation suffers from maintainability issues (if a new desirable CSS or HTML feature would be released, additional code would have to be written to explicitly support it). In contrast, the CSS 3D transformations-based scheme maximally retains compliance with the original AVV modus operandi. This implies that it inherits not just all AVV functionality that has previously been developed for non-ODV setups, but also all the off-the-shelf HTML and CSS constructs around which the AVV framework has been designed. For example, the layout and style of overlay elements remain controllable via CSS, the full HTML syntax remains exploitable to describe their contents, and interaction handling can still occur on the basis of DOM events. In terms of visual realism however, the WebGL-integrated approach will typically intertwine the overlay elements and the 3D scene in a visually somewhat more convincing fashion than the CSS 3D transformations-powered implementation.

This is due to the fact that the latter uses dedicated renderers for respectively the 3D scene and the overlay elements, which prohibits them from being tightly integrated. Finally, it is also important to note that the WebGL-integrated solution outperforms its CSS 3D transformations-based counterpart in terms of computational complexity (see Sect. 8 for concrete performance figures).

All reaction overlay elements that appear in a 3D ODV player screenshot in this paper were rendered using CSS 3D transformations (i.e., see Fig. 8). This is motivated by the fact that this class of hotspots typically relies extensively on CSS style features and HTML constructs (e.g., to implement the call-out chrome, see Sect. 4.3). Furthermore, they often carry text-based content. Based on the just presented comparison, it should be apparent that implementing such hotspots as WebGL objects would be cumbersome. On the other hand, the WebGL-integrated approach is the preferred solution for less complex overlay elements, due to its improved efficacy with regard to visual fidelity as well as its smaller computational footprint. Consequently, this technique was applied to visualize the non-reaction overlays in the 3D ODV player screenshots (i.e., see Figs. 2(b), 3 and 8).

6 Authoring

The targeted authoring audience of the AVV framework encompasses not only Web developers, but also video enthusiasts (both amateurs and professional practitioners). The technological expertise of these two user categories likely diverges largely. As an example, it is fairly safe to assume that members of the former have profound knowledge of prevailing Web practices. As a result, they might be sufficiently versed to define overlay elements directly in JavaScript. It is however highly improbable that the same is true for people with a background in visual design or video production. Such users would therefore benefit from the ability to construct AVV experiences from a more high-level point of view, using a graphical user interface. In this section, the alternative authoring solutions that were developed to cater to the desires and competences of both user bases will be presented. Of course, AVV experience designers are by no means confined to a single editing method and are free to fuse the different approaches as they see fit. For example, it could make perfect sense to draft a rough version of the envisioned AVV setup using the graphical editor, and then to refine the result by manually tweaking some settings directly in JavaScript.

6.1 Direct Video Overlay Instantiation

At the lowest end of the abstraction scale, authors can define video overlays directly in JavaScript, by instantiating their corresponding representation in the JAVV library. This approach basically boils down to the writing of JavaScript code, and will therefore probably only be viable for users that exhibit at least elementary Web development skills.

6.2 JSON Specification

The JAVV software architecture applies the *factory* design pattern to allow for the construction of overlay elements on the basis of JavaScript Object Notation (JSON) input. The code listing below provides an example of a JSON-encoded video overlay element representation. Since in this approach it suffices for authors to draft JSON documents instead of doing actual JavaScript coding, this solution is situated somewhat higher up the abstraction ladder than the previous one.

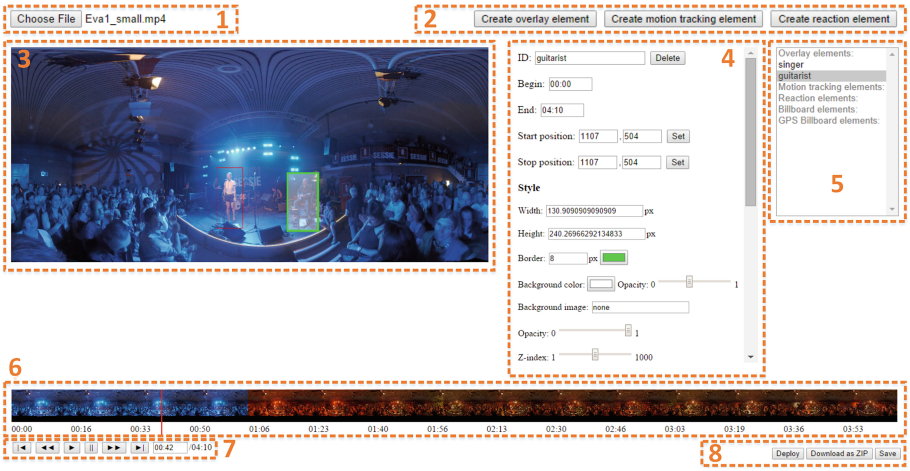
```
{
  "id": "MyId",             // Unique VOE identifier
  "timeStart": 100,         // Visibility start time
  "timeStop": 200,          // Visibility end time
  "positionStart": {        // 2D position at time "timeStart"
    "x": 10,
    "y": 10
  },
  "positionStop": {         // 2D position at time "timeStop"
    "x": 20,
    "y": 30
  },
  "htmlContent": /* Arbitrary HTML markup content goes here */,
  "interactionHandler": {
    "click": function(e, overlay) { ... },
    "mouseover": function(e, overlay) { ... },
    "mouseout": function(e, overlay) { ... }
  },
  "htmlStyle": /* CSS customization instructions go here */
}
```

Supporting indirect video overlay element instantiation via an open and language-independent standard for data interchange like JSON entails clear benefits in terms of flexibility, portability and interoperability. An additional advantage of embracing a widely accepted standard is that it is typically blessed with a wealth of facilitating tools. As an example, the JAVV library exploits JSON Schema [16] to validate the syntactical structure and semantics of JSON documents describing video overlay elements.

6.3 Graphical Editor

The most high-level AVV authoring solution is provided by a graphical editing interface that is accessible via a standard Web browser. An annotated screenshot of this Web service is shown in Fig. 6. The editor supports the augmentation of both ODV and traditional video content.

Without going into too much detail, the editor's design is centered around important HCI guidelines such as providing direct feedback (visual or otherwise) and supporting direct manipulation. The former design principle is, for instance, applied by ensuring that the actions performed by users are, whenever possible,



Eva1_small.mp4

ID: guitarist
Begin: 00:00
End: 04:10
Start position: 1107 , 504
Stop position: 1107 , 504
Style
Width: 130.9090909090909 px
Height: 240.26966292134833 px
Border: 8 px
Background color: Opacity: 0 1
Background image: none
Opacity: 0 1
Z-index: 1 1000

Overlay elements:
singer
guitarist
Motion tracking elements:
Reaction elements:
Billboard elements:
GPS Billboard elements:

00:00 00:16 00:33 00:50 01:06 01:23 01:40 01:56 02:13 02:30 02:46 03:03 03:19 03:36 03:53

00:42 /04:10

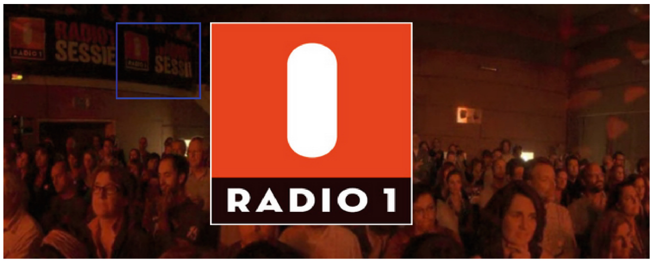
Fig. 6. The graphical interface of the editor Web service: (1) Video content selection; (2) Overlay element instantiation; (3) Preview widget (visualizes the currently active video frame and the overlay elements that apply to it); (4) Management form for overlay element properties; (5) Enumeration of defined overlay elements (i.e., selection pane); (6) Video playback timeline (including thumbnails of individual video frames to facilitate temporal navigation); (7) Video playback controls; (8) Export and deployment options.

directly reflected in the preview widget. Another illustration of the adoption of this principle is given by the WYSIWYG text editor that is included in the Web service (not shown in Fig. 6). This tool allows users to specify the textual contents of overlay elements without mandating them to be familiar with the HTML syntax. On the other hand, an example of direct manipulation support can be found in the fact that users are able to apply drag-and-drop interaction to intuitively reposition already defined overlay elements in the preview window.

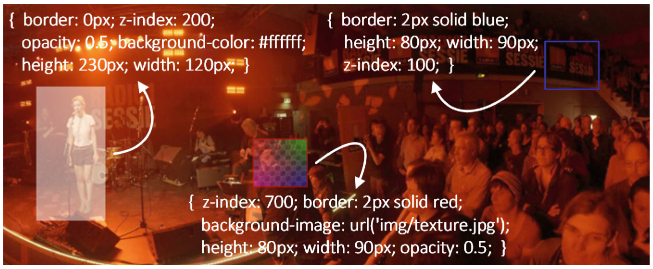
The result of the editing process can be exported to an intermediary format that in turn can be applied by the JAVV API in order to incorporate the authored AVV experience inside a HTML page. The Web service even includes the option to publish the editing outcome (i.e., the combination of the involved video clip and the overlay elements that were defined for it) in a directly consumable manner to an HTTP server.

7 Showcases

In this section, we will present two augmented ODV prototypes. The first prototype chiefly acts as a technological proof-of-concept and is hence intended to showcase the feasibility as well as the functional features of the proposed technology. The second demonstrator on the other hand involves a real-life use case and consequently provides a hint of the valorization potential of our work.



(a)



```
{ border: 0px; z-index: 200;
  opacity: 0.5; background-color: #ffffff;
  height: 230px; width: 120px; }
```

```
{ border: 2px solid blue;
  height: 80px; width: 90px;
  z-index: 100; }
```

```
{ z-index: 700; border: 2px solid red;
  background-image: url('img/texture.jpg');
  height: 80px; width: 90px; opacity: 0.5; }
```

(b)

Fig. 7. Additional screenshots of the augmented concert showcase: (a) Reaction video overlay carrying image content; (b) Overview of the three non-reaction elements, each annotated with its respective CSS style string.

7.1 Augmented Concert Capture

The first demonstrator is built around ODV content that was captured during a small-scale musical performance involving an audience of approximately 150 people. A stationary ODV camera was installed in front of the stage, amidst the audience, to record the concert. The screenshots that have been included in the paper up to this point all originate from this demonstrator.

The demonstrator itself is implemented as two distinct HTML pages which respectively host the 2D planar and the WebGL-based ODV player. Via a HTML link embedded in the pages, one can switch between the two ODV player instantiations. The video footage is augmented with a total of five overlay elements. Two of these are statically positioned, one is animated on the basis of motion tracking data, and the final two are reaction overlays. The statically positioned overlay elements mark the singer of the band and a segment of a promotional banner, respectively. Both are rectangularly shaped and their styling respectively consists of a semi-transparent fill color and colored edges (see Fig. 7(b)). Interacting with the singer's hotspot causes a short biography of the band to be displayed in a dedicated portion of the webpage, external to the ODV player. On the other hand, selecting the overlay element on top of the banner results in a reaction element popping up which holds an image of the radio station that organized the concert. This effect is shown in Fig. 7(a). The motion-tracked overlay element follows a camera operator as he moves around in front of the stage. Figure 4 illustrates that this particular element is visualized as a semi-transparent image with a border colored in red and that interacting with it causes a reaction video overlay acting as a call-out box holding textual information to appear on top of the video footage.

Please remark that the styling solutions and content presentation methods that are showcased in this prototype are mere illustrations and are in fact rather simplistic. For the sake of comprehensiveness, the exact style settings that apply to the three non-reaction hotspots are communicated in Fig. 7(b). It is explicitly repeated here that, courtesy of the extensiveness of both the CSS and HTML specifications, the artistic options for overlay element design available to visual artists are nearly limitless.

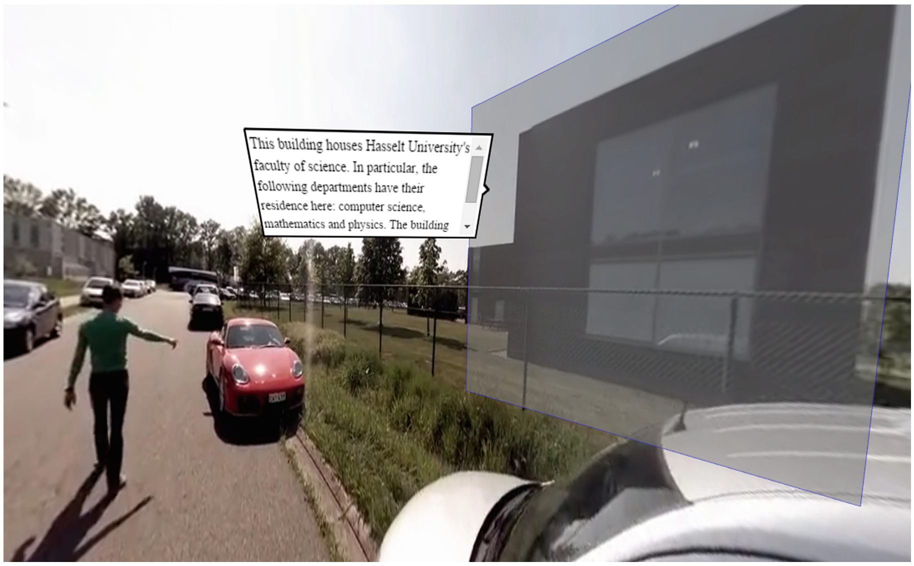
7.2 Virtual Walkthrough

The second demonstrator exerts the JAVV API to incorporate interactive features in a practical scenario, namely a virtual walkthrough use case.

Use Case. At Hasselt University, a Web application has been developed that offers students and employees a virtual walkthrough of the campus grounds in order to familiarize them with the spatial layout of the site. The application's media content consists of ODV recordings of a human guide as he walked around the campus territory. Along the way, the guide pointed out various salient features in his vicinity (e.g., university buildings or services). In a post-production phase, the ODV footage was temporally segmented on the basis of physical junction points (e.g., a crossroad) that were encountered during ODV capture.

The prototype can in a sense be regarded as an ODV-powered counterpart of the Street View functionality included in Google Maps. In effect, the typical usage scenario involves users virtually moving along the roadways and choosing their desired traveling direction at subsequent intersections in order to get a feel of the layout of the university grounds. While navigating the streets, users can play/pause the video sequences, freely change their viewing angle, and zoom in and out at their personal discretion. Please note that no specific effort was needed to realize this functionality, as the prototype directly inherits these functions from the incorporated ODV Web player. Also note that the Web application is meant for internal use only and is hence not publicly available.

AVV Integration. The AVV framework was integrated into this Web application so that relevant physical elements in the captured scenery could be tagged by means of interactive overlays. Given the mobility of the capture camera, all hotspots in this prototype are of the motion-tracked type. When selected, the hotspots display textual descriptions (encoded as reaction overlays) of the underlying object in the video footage. The AVV integration into this use case hence served a dual purpose: complementarily highlight the real-world items pointed



This building houses Hasselt University's faculty of science. In particular, the following departments have their residence here: computer science, mathematics and physics. The building

Fig. 8. Screenshot of the augmented version of the virtual walkthrough prototype (visualized using the 3D ODV player).

out by the human guide in the ODV footage, and appropriately present informative call-outs to users. A screenshot that illustrates the outcome of the AVV integration can be found in Fig. 8.

8 Performance Evaluation

To analyze the computational complexity of the augmented ODV solution, a performance benchmark was conducted. In particular, the augmented concert showcase presented in Sect. 7.1 was profiled with regard to computational resource usage on two distinct platforms: a mid-range desktop PC equipped with GPU hardware acceleration, and a low-cost laptop. The exact hardware specifications of the desktop PC were as follows: Intel Xeon W3505 CPU running at 2.53 GHz, Nvidia GeForce GTX 760 GPU, 4 GB RAM. The laptop was a DELL Latitude E6510 housing an Intel Core i3 M370 CPU clocked at 2.40 GHz, an Nvidia NVS 3100M graphics card, and 4 GB RAM. Software-wise, the desktop and laptop test platforms ran Windows 8.1 and Windows 7 SP1, respectively. On both machines, the audit was executed using Google Chrome version 39.0.2171.95 m (32-bit).

The benchmark test case was encoded as a script and subsequently applied a number of times under variable configuration settings and conditions. The script first made sure that the viewport of the ODV Web player was positioned in such a way that the band and the rectangular stage were in view. From that point on, no viewport modifications whatsoever occurred during the remainder of the test scenario. Once the viewport was appropriately positioned, the benchmark actually commenced. Approximately 1 second after the start, the overlay element that is associated with the singer of the band was selected (i.e., clicked on). After a time interval of about 1 second, the camera operator's motion-tracked video overlay was then selected. This action caused the corresponding reaction overlay element to be rendered in the viewport (see Fig. 4). Finally, again approximately 1 second later, the test case was concluded. Each benchmark run consequently consumed about 3 seconds in total.

The configuration settings that were varied across tests include (i) the resolution of the input ODV material, (ii) the output resolution of the ODV Web player (i.e., the viewport resolution), and (iii) using the 2D planar versus the 3D spherical augmented ODV Web player implementation.

The performance results were collected using the Chrome Developer Tools [17] and are summarized in Tables 1 and 2. In both tables, the figures communicate CPU consumption (i.e., method execution time), expressed as a percentage of the total running time of the involved benchmark test. As an example, a value of 5 would indicate that 5 percent of the benchmark's running time was spent processing the corresponding JavaScript function. The percentual time

Table 1. Benchmark results (% of total time) on a hardware accelerated desktop; 1920x1080 versus 960x540 video input, static viewport.

	2D Planar Augmented ODV				3D Spherical Augmented ODV				
Output	**draw**	**anim**	**clip**	**Idle**	**render**	**anim**	**uCSS**	**uWGL**	**Idle**
640x360	11.3	8.7	3.0	72.8	11.1	4.3	1.6	0.1	71.2
	1.6	8.5	2.9	82.1	2.8	4.3	1.8	0.1	79.1
1280x720	10.1	8.6	2.9	72.9	11.1	4.2	1.6	0.1	70.2
	1.6	8.2	2.9	83.0	2.8	4.7	1.8	0.1	78.4
1920x1080	9.7	8.1	2.7	72.2	10.2	4.4	1.7	0.2	70.9
	1.3	8.0	2.8	81.8	2.7	4.6	1.9	0.1	78.6

Table 2. Benchmark results (% of total time) on a commodity laptop; 1920x1080 versus 960x540 video input, static viewport.

	2D Planar Augmented ODV				3D Spherical Augmented ODV				
Output	**draw**	**anim**	**clip**	**Idle**	**render**	**anim**	**uCSS**	**uWGL**	**Idle**
640x360	10.1	12.4	4.0	65.9	11.8	6.5	2.7	0.1	62.1
	2.4	11.2	3.6	74.5	4.5	5.9	2.4	0.1	70.7
1280x720	9.5	12.4	3.9	63.7	12.1	6.8	2.7	0.1	60.2
	2.3	10.9	3.6	73.3	4.4	6.2	2.4	0.1	69.1
1920x1080	9.8	12.7	4.1	65.2	12.6	6.7	2.7	0.1	58.3
	2.3	11.8	3.8	72.3	5.2	6.4	2.5	0.1	67.9

the CPU was idle during the different audit configurations is likewise included in the tables. Each reported figure corresponds with the average of the results from 5 independent runs of the respective test, to moderate the impact of outliers. The semantics of the method names appearing in the benchmark results are as follows:

`draw` (abbreviation for `drawImage`) The default HTML5 method to draw to a `<canvas>` element; is used to render the currently active ODV viewport to the screen in the 2D planar ODV player

`anim` The JAVV API function that implements the positioning, animation and rendering of overlay elements; is separately implemented for both the 2D planar and 3D spherical augmented ODV alternatives

`clip` A subroutine of the 2D planar `anim` method that is exclusively responsible for overlay element clipping (see Sect. 5.1)

`render` The three.js function to render a WebGL scene

`uCSS` A subroutine of the 3D spherical `anim` method that manages the animation of overlay elements using CSS 3D transformations (see Sect. 5.2)

`uWGL` A subroutine of the 3D spherical `anim` method that manages the WebGL-integrated animation of overlay elements (see Sect. 5.2)

Space limitations unfortunately refrain us from delving exhaustively into the benchmark outcomes. Therefore, we will limit ourselves to enumerating four of the more salient findings. The first observation is that the 3D spherical augmented ODV renderer is more efficient at drawing and animating the overlay elements compared to its 2D planar counterpart (see the values of the respective `anim` columns in Tables 1 and 2). This finding holds true across all tested input and output resolutions, on both test platforms. Second, by comparing the `uCSS` and `uWGL` columns, it becomes apparent that updating and rendering video overlay elements using the CSS 3D transformations scheme is computationally considerably more expensive than via the WebGL-integrated approach. Recall from Sect. 5.2 that the former solution is exploited to display reaction hotspots. Exactly one such element became visible in the course of the benchmark test scenario. If the test case would not include a reaction hotspot, the difference in overlay element rendering time between the 2D and 3D implementations (as identified in the first finding) would be even larger. Thirdly, in the 2D planar augmented ODV implementation, the clipping of overlay elements consumes a considerable slice of the CPU budget (i.e., approximately one third of the animation processing is devoted to clipping). This is caused by the relatively large number of position and offset calculations that the clipping operation requires. The performance of this animation phase could potentially be improved by resorting to a clipping scheme that is based on the CSS `overflow` property. An `overflow`-based solution would transform the clipping from a manual to an automated process, which is expected to have a positive influence on performance. Investigating the validity of this hypothesis is an important subject of future work. The final finding pertains to the impact of the input and output video resolutions. In hardware accelerated settings, the input video resolution does not appear to

Table 3. Benchmark idle times (% of total time) on a Nexus 7 tablet; 1920x1080 versus 960x540 video input, static viewport.

	2D Planar Augmented ODV			3D Spherical Augmented ODV		
Output:	**640x360**	**1280x720**	**1920x1080**	**640x360**	**1280x720**	**1920x1080**
Idle time:	35.22 47.25	7.71 9.82	3.54 4.52	28.92 23.51	2.02 4.67	0.1 0.9

play an appreciable role with respect to overlay element rendering and animation performance, for either the 2D or the 3D Web player implementation. In the absence of decent GPU acceleration, a small impact on performance is however noticeable (the computational overhead of animating hotspots rises as the input video resolution increases, see the `anim` columns in Table 2). The tested ODV viewport resolutions on the other hand seem to only marginally affect the CPU workload induced by the JAVV library.

It is important to mention that, under every considered test condition, ample idle CPU cycles were available (as is evidenced by the "Idle" columns in Tables 1 and 2), which resulted in all experiments running smoothly at comfortable frame rates. This observation unfortunately only partly holds true on tablet devices. As an example, Table 3 elaborates the idle times on a Nexus 7 tablet (2012 edition) running Android 5.0 and Google Chrome 38.0.2125.509 (identical benchmark configuration permutations and test case as before, idle time expressed as a percentage of the total running time of the benchmark test, results were averaged out over 5 independent test runs). As can be derived from this table, the benchmark test conditions that yield the highest visual quality failed to render smoothly. On the other hand, the tested mid-range and low-end quality settings produced acceptable frame rates (i.e., above 25 FPS). As an example, when fed with 960x540 video input and producing 1280x720 video output, the 2D planar implementation witnessed 9.82 % idle time, which resulted in an average frame rate of approximately 25 FPS. Please note that, since the augmented ODV codebase is currently not optimized for mobile platforms, the figures reported in Table 3 definitely leave room for improvement.

In all, the presented audit findings lead us to conclude that the augmented ODV solution is computationally compatible with commodity hardware, although deployment on mobile devices might mandate quality sacrifices depending on terminal capabilities.

9 Conclusions

Over the years, the video medium has witnessed substantial technological innovations. This paper has focused on one such fairly recent innovation, namely the ability to record scenes with a spherical Field of View. Somewhat surprisingly, the medium has not seen the same degree of evolution when it comes to the types of experiences it is able to deliver to viewers. In particular, video consumption to date largely remains a passive matter, with little thought for end-user interactivity. This paper has proposed a pragmatic solution to battle this stagnation in the form of the augmented Omni-Directional Video concept. The rationale behind the concept consists of overlaying ODV footage with interactive elements that are represented and rendered in a Web standards-compliant manner, in this way effectively "activating" ODV consumption. The various building blocks required to realize the augmented ODV methodology have all been addressed.

Interactive elements to be integrated into the ODV content can be presented through divergent mechanisms (each with specific advantages and drawbacks) depending on imposed technical restrictions and the context of use. Nevertheless, all proposed mechanisms build upon well-established Web standards such as HTML5, CSS, JavaScript and WebGL. This deliberate design decision benefits the authoring process, as it allows content producers to engineer novel experiences without them facing the steep learning curve that is typically associated with the adoption of a new technology. To further improve the content creation workflow, a graphical authoring interface has been proposed that enables point-and-click interaction for tasks that would otherwise require repetitive (manual) editing.

The practical applicability and valorization potential of the augmented ODV concept has been showcased by two representative use case descriptions, which were implemented using a mixture of technologies presented in this paper. Finally, performance benchmark figures have revealed our implementation to be computationally economical.

Acknowledgements. The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 610370, ICoSOLE ("Immersive Coverage of Spatially Outspread Live Events", http://www.icosole.eu). The authors would like to explicitly thank Jori Liesenborgs for developing the ODV Web player, Jeroen Leën for porting the AVV codebase to the 2D planar ODV implementation, and prof. Philippe Bekaert for the ODV recording of the concert which served as video material in the technological demonstrator described in Sect. 7.1.

References

1. Raney, A.A., Arpan, L.M., Pashupati, K., Brill, D.A.: At the movies, on the Web: An investigation of the effects of entertaining and interactive Web content on site and brand evaluations. J. Interact. Mark. **17**, 38–53 (2003)
2. Wijnants, M., Leën, J., Quax, P., Lamotte, W.: Augmented video viewing: transforming video consumption into an active experience. In: Proceedings of the 5th Multimedia Systems Conference, MMSys 2014, Singapore, Singapore, pp. 164–167. ACM (2014)
3. Meixner, B., Matusik, K., Grill, C., Kosch, H.: Towards an easy to use authoring tool for interactive non-linear video. Multimedia Tools Appl. **70**, 1251–1276 (2014)

4. Sawhney, N., Balcom, D., Smith, I.: Authoring and navigating video in space and time. IEEE MultiMedia **4**, 30–39 (1997)
5. Dakss, J., Agamanolis, S., Chalom, E., Michael Bove Jr., V.: Hyperlinked Video. In: Proceedings of SPIE Multimedia Systems and Applications, vol. 3528, pp. 2–10 (1999)
6. Shipman, F., Girgensohn, A., Wilcox, L.: Combining spatial and navigational structure in the hyper-hitchcock hypervideo editor. In: Proceedings of the 14th Conference on Hypertext and Hypermedia, Hypertext 2003, Nottingham, UK, pp. 124–125. ACM (2003)
7. Seeliger, R., Räck, C., Arbanowski, S.: Non-linear video - A cross-platform interactive video experience. In: Proceedings of the 2nd International Conference on Creative Content Technologies, CONTENT 2010, Lisbon, Portugal, pp. 34–38 (2010)
8. Hoffmann, P., Kochems, T., Herczeg, M.: HyLive: Hypervideo-Authoring for Live Television. In: Tscheligi, M., Obrist, M., Lugmayr, A. (eds.) EuroITV 2008. LNCS, vol. 5066, pp. 51–60. Springer, Heidelberg (2008)
9. Sadallah, M., Aubert, O., Prié, Y.: CHM: an annotation- and component-based hypervideo model for the web. Multimedia Tools Appl. **70**, 1–35 (2012)
10. Neng, L.A.R., Chambel, T.: Get around 360° hypervideo. In: Proceedings of the 14th International Academic MindTrek Conference: Envisioning Future Media Environments, MindTrek 2010, Tampere, Finland, pp. 119–122. ACM (2010)
11. Jansen, J., Bulterman, D.C.: SMIL State: an architecture and implementation for adaptive time-based web applications. Multimedia Tools Appl. **43**, 203–224 (2009)
12. Khronos Group: WebGL Specification Version 1.0.2 (2013). https://www.khronos.org/registry/webgl/specs/1.0/
13. Hamari, J., Koivisto, J., Sarsa, H.: Does gamification work? - A literature review of empirical studies on gamification. In: Proceedings of the 47th Hawaii International Conference on System Sciences, HICSS-47, Hawaii, USA, pp. 3025–3034 (2014)
14. W3C: CSS Transforms Module Level 1 (2014). http://dev.w3.org/csswg/css-transforms/
15. Weisstein, E.W.: Spherical Coordinates. From MathWorld - A Wolfram Web Resource (2014). http://mathworld.wolfram.com/SphericalCoordinates.html. Online
16. Galiegue, F., Zyp, K., Court, G.: JSON Schema: interactive and non interactive validation. Internet-Draft, Internet Engineering Task Force (2013). http://tools.ietf.org/html/draft-fge-json-schema-validation-00
17. Google: Chrome DevTools Overview (2015). https://developer.chrome.com/devtools