Multimed Tools Appl (2018) 77:125–148
DOI 10.1007/s11042-016-4255-8

Adaptive streaming of complex Web 3D scenes based on the MPEG-DASH standard

Markos Zampoglou[1] · **Kostas Kapetanakis**[1] · **Andreas Stamoulias**[1] · **Athanasios G. Malamos**[1] · **Spyros Panagiotakis**[1]

Received: 25 June 2015 / Revised: 1 December 2016 / Accepted: 8 December 2016 /
Published online: 16 December 2016

Abstract Modern Web 3D technologies allow us to display complex interactive 3D content, including models, textures, sounds and animations, using any HTML-enabled web browser. Thus, due to the device-independent nature of HTML5, the same content might have to be displayed on a wide range of different devices and environments. This means that the display of Web 3D content is faced with the same Quality of Experience (QoE) issues as other multimedia types, concerning bandwidth, computational capabilities of the end device, and content quality. In this paper, we present a framework for adaptive streaming of interactive Web 3D scenes to web clients using the MPEG-DASH standard. We offer an analysis of how the standard's Media Presentation Description schema can be used to describe adaptive Web 3D scenes for streaming, and explore the types of metrics that can be used to maximize the user's QoE. Then, we present a prototype client we have developed, and demonstrate how the 3D streaming process can take place over such a client. Finally, we discuss how the client framework can be used to design adaptive streaming policies that correspond to real-world scenarios.

✉ Spyros Panagiotakis
spanag@ie.teicrete.gr

Markos Zampoglou
markzampoglou@yahoo.gr

Kostas Kapetanakis
kapekost@gmail.com

Andreas Stamoulias
stamoulias@gmail.com

Athanasios G. Malamos
amalamos@ie.teicrete.gr

[1] Multimedia Content Lab, Department of Informatics Engineering, TEI of Crete, Estavromenos, Iraklio 71004 Crete, Greece

Keywords 3D streaming · MPEG-DASH · Quality of Experience · Web 3D · X3D · HTML5

1 Introduction

The advent and wide spread of HTML5 has brought radical changes to the way we deal with multimedia data, by solving a large number of issues on device-dependence and compatibility. It is now possible to play multimedia material such as video, audio and recently -using the WebGL JavaScript API- Virtual Reality (VR, i.e. interactive 3D) scenes within web pages, without the need to install specific software or plugins. All that is required is a modern, HTML5-enabled browser.

Within this context, Web 3D is finally attracting an increasing level of attention from industry and the research community. In the past, the Virtual Reality Modelling Language (VRML) [26, 27] and its XML-based successor, X3D [9, 10], were proposed as mid-level languages for declarative interactive Web 3D applications. However, up to the advent of WebGL, these standards could only work with the use of plugins or specialized browsers. Recently, the X3DOM [3] framework was proposed which, using WebGL, allows us to directly integrate X3D XML code in HTML, and directly display its content within the Web page.

These leaps in Web 3D technologies bring a new ubiquity to 3D content. Since a wide range of devices besides computers, such as mobile phones or smart TVs can now render VR scenes simply by browsing into the web pages that contain them, the same scene might end up being displayed on devices and environments with widely ranging characteristics, such as available bandwidth, computational power (including the presence or absence of a dedicated Graphical Processing Unit, or GPU), and screen size and resolution.

3D scenes can take up very large volumes of data, and can place a potentially very high computational burden on the end device. Similar to video, there have been efforts on making the downloading of 3D scenes progressive, so as to begin by displaying a rough, low-quality or partial version of a scene to the user, and progressively improve or complete it, until the entire scene is displayed in the best quality possible. In 3D graphics, the multiple versions of the same content in different qualities are known as Levels of Detail (LoD). If the partial scene allows navigation and interactions from the beginning, this approach can significantly smoothen the user experience, and partially solve issues of low throughput. However, it fails to tackle the problems of computational and display limitations, which can greatly compromise the user's Quality of Experience (QoE). The term QoE refers to a broad spectrum of attempts to measure and optimize the entire user experience and satisfaction with a product or service, as an extension of previous Quality-of-Service approaches [4, 6, 23]. In the case of Web 3D QoE, in addition to progressive streaming, another level of content adaptation would be to constantly monitor various, QoE-related device and environment performance metrics, which reflect on user experience, and adapt the scene content Level of Detail (LoD) [5, 19] to allow for a smoother user experience (for example, reduce model/texture quality if the user opens multiple applications and CPU/GPU demands go up).

This process of constant monitoring and re-adaptation is similar to adaptive video streaming, where video quality is reduced when bandwidth goes down, and increased again when it goes up. However, dealing with VR data is significantly more complex, as we are simultaneously monitoring both computational load and network throughput. More specifically, 3D scenes are produced as a convolution of spatial (geometry), image, video and audio

information. Controlling Quality of Experience in such heterogeneous information is a complex task and is unrealistic to assume that might be treated as a whole. It is rather feasible to assume that QoE-aware policies should be independently applied to each media type. In case of QoE in video and audio delivery many algorithms and platforms have been proposed over the last decades. On the other hand QoE in the case of spatial 3D information is rather considered as a Level of Detail issue where geometry adjusts to different numbers of vertices and faces according to throughput and processing constraints.

In this work, we present an approach to the adaptive, QoE-aware, streaming of Web 3D content, based on the MPEG-DASH standard, that attempts to tackle these problems simultaneously. This approach creates new opportunities in graphic applications and games. The join of Web 3D content and MPEG-DASH introduces a novel framework for progressive applications, fully functional in different QoE levels, that may potentially take advantage of the flexibility of HTTP, the power of the multicore-GPUs and the multithreading of the web tools that appear in the market. Thus, following our approach it is possible to develop a new generation of streaming games based on progressive inline objects (3D models, textures, videos) that are adapted to the user end-system and throughput capabilities. The rest of the paper is organized as follows: Section 2 presents the relevant research background in the field. Section 3 presents our adaptation of the MPEG-DASH standard for QoE-aware Web 3D streaming, while Section 4 describes the requirements for a Web 3D MPEG-DASH client, and presents our prototype implementation. Finally, Section 4 concludes our work and explores the new research directions that our work opens.

2 Background and related work

Progressive streaming of 3D models can be traced back to the Progressive Mesh approach by Hoppe [12]. In this approach, a model is encoded in a series of consecutive interrelated representations, starting from a very low LoD and gradualy refining them, up to the highest LoD corresponding to the original model. When a user requests a model, the lowest LoD is initially sent, and, consecutively, each next level is then transmitted to the client. The end result is a smooth refinement process which leads up to the actual model the user requested. Thus, progressively streaming models allows users to begin experiencing them long before they fully arrive, and thus reduces waiting times and increases smoothness and the overall QoE.

A large body of research has since been devoted to 3D streaming [1, 11, 22, 24, 28], taking advantage of different approaches with respect to model encoding, the degree and form of interdependence between consecutive levels (since commonly, a refined level is not autonomous, but exists as a function of the immediately lower one), and the type of compression. Recently, a fast and visually appealing method has been proposed [16]. The approach incorporates the state-of-the-art in 3D model progressive transmission and HTML5 technologies, including a novel algorithm for color information compression. However, the proprietary nature of some of the steps in the proposed streaming pipeline greatly reduce applicability for our needs.

The X3DOM framework also includes a number of approaches for progressive streaming [2, 17, 18]. These include algorithms which take X3D models stored as Indexed Face Sets, binarize them, and progressively transmit them. These methods are the X3DOM Binary Geometries (BG), Sequential Image Geometries (SIG) binary Level of Detail (bitLOD) geometries, and the Progressively Ordered Primitive (POP) buffer.

All the aforementioned methods are based on data transfer over simple HTTP requests. In our own previous work [15], we explored the possibility of partially streaming 3D models to a client using more advanced frameworks, such as Message Brokers, Websockets and Simple/Streaming Text Oriented Message Protocol (commonly known as STOMP), and demonstrated the overall advantages of certain combinations of these methods in performance, both in terms of network and computational demands.

The aforementioned methods all deal with single models, which are progressively streamed to the client, regardless of their device and environment status, until the highest LoD model has arrived. The reality of the present and future of Web 3D, however, is quite broader than that. For one, in the vast majority of cases we are not dealing with isolated models but with complex scenes, consisting of models, textures, sounds and other scene information (such as interactivity, or light). This data has to be synchronized to create a full user experience. Secondly, progressively pushing information from the server to the client may not be enough. In the Web 3D context, where we are faced with different devices, we also have to take computational limitations into account. Thus, adaptive streaming comes into play.

Adaptive streaming refers to approaches that constantly monitor the user's bandwidth and computational resources, and adapt the stream quality accordingly in order to maximize the user's overall Quality of Experience. The MPEG-DASH (Moving Pictures Expert Group - Dynamic Adaptive Streaming over HTTP) standard [8], is a relatively young standard in the field, which is already attracting a lot of attention from both the research community and industry, based on its simple yet powerful nature.

MPEG-DASH (Dynamic Adaptive Streaming over HTTP) is based on the concept of the Media Presentation Description (MPD). An MPD, stored in an XML-based.mpd file, contains a detailed description of an entire media stream, its sources, their characteristics (such as content or quality) and their location, as well as their interrelations. It also includes recommendations on the proposed client behavior with respect to the stream in various environments.

Essentially, the standard does not include specific instructions for client development. The whole development approach, display method, and resource priorities (CPU, bandwidth etc.) that will maximize the user's QoE, are out of scope of the standard. The standard is furthermore, content-agnostic, in that the actual media data are of no concern to the MPD. Instead, only the stream organization is defined in MPEG-DASH.

A Media Presentation Description is organized in four levels of abstraction (Fig. 1): Periods, Adaptation Sets, Representations and Segments. A Period is a small temporal part of the entire stream, and contains all the information necessary for independently displaying the corresponding temporal segment of the stream. Within it, multiple Adaptation Sets refer to various media modalities, e.g. a video track, multiple audio tracks in different languages, and text tracks (such as subtitles) would each be described in a different adaptation set each. Within an Adaptation Set, multiple Representations contain the same medium in different qualities (e.g. various bitrates for video/audio). A Representation is then split into Segments, which refer to the actual media files to be transmitted, while a Segment can consist of multiple Sub-Segments. The corresponding, synchronized Sub-Segments of a Representation can form a Sub-Representation. At each level of organization, XML attributes provide information with respect to both the actual media files to be transmitted, and the media author's recommendations concerning the proposed adaptation choices a client should make while streaming –for example, how much bandwidth would be required to transmit a certain Representation without lagging. While MPEG-DASH does include recommendations on client adaptation policies, the philosophy behind the standard is that clients should be allowed to read a stream's MPD and make their own adaptation choices, which can be specific to

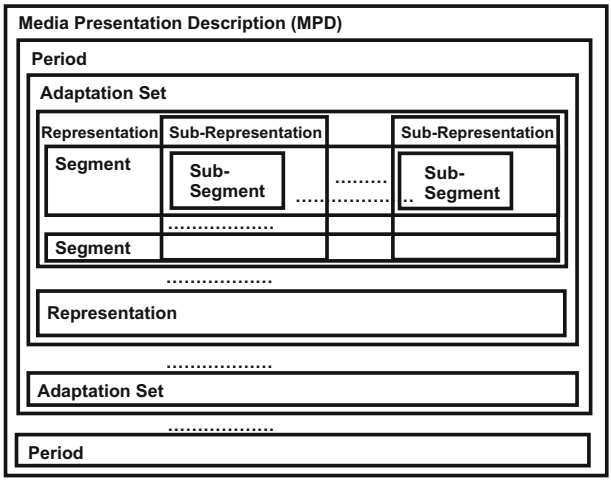


Fig. 1 The organization of a complex 3D scene in a media presentation description

their environment or the device on which they are running. Thus, MPEG-DASH essentially offers a way of describing media streams, and providing recommendations on client behavior.

In our previous work [14], we proposed to use the MPEG-DASH standard to adaptively stream a single X3D model. In this work, we present an exhaustive analysis of the MPEG-DASH standard, its XML structure and its attributes, vis-à-vis its potential for complex X3D scene's streaming. It is an attempt to tackle the problem of organizing X3D scenes comprising multiple geometries, image/video textures, and audio into MPD representations for adaptive streaming, while also developing a prototype JavaScript client implementation that can deliver and display such complex scenes for testing and evaluating the feasibility of our model. In contrast to our previous exploration of more advanced data transmission frameworks [15], and despite the potential advantages of using such technologies, in the work presented here we return to simple HTTP, so as to take advantage of its ubiquity and to ensure compliance with the MPEG-DASH standard.

3 MPEG-DASH framework for web 3D streaming

3.1 Introduction

MPEG-DASH is essentially usable with any multimedia type, and also content-agnostic by definition. The client is expected to acquire all information about the multimedia item from the MPD, combine it with information acquired from the environment such as bandwidth availability, and act accordingly. Thus, all the implementation, decision-making and actual content streaming and display falls on the client. The only requirement is that the multimedia type being streamed is a registered MIME type. In our case, as we explained in [14], *model/x3d+xml* is indeed registered, which means it is compatible with the standard. However, in our case, where we are attempting to apply MPEG-DASH to a type of media for the first time, we must give all efforts to ensure that the modeling of content in the MPD framework is made in a way that is consistent with the standard, open to further modifications, and easy to use with any conforming client implementations, while still providing an intuitive description of the content.

3.2 Organization of 3D scene media modalities in the media presentation description

The first step in the organization of an X3D scene in a Media Presentation Description is the separation of the multiple information modalities contained therein. An X3D scene is structured as a Scene Tree, which provides a framework of a grouping hierarchy, including spatial transformations. Within its nodes, the actual multimedia items are contained. These can be separated into a) the Scene Tree Code, that is, the scene hierarchy, as well as any additional code such as scripting, b) models, including additional information such as colors or normals (per vertex or per face), c) texture images, d) texture videos and e) audio files. Each of those media types requires different treatment to ensure parallel adaptive streaming of all the content of a scene. Table 1 lists the approach we follow towards each information type, the level of the MPD we structure it in, and whether and how we can stream it using a partial or progressive representation scheme.

The Scene Tree is encoded in XML text format, and contains the scene structure as well as any additional code needed for the scene to play: interactivity options, animations, lighting, world/background information and metadata. Essentially, it is all that remains of the scene if we remove models, videos, images and sound. Syntactically, it is written in X3D, which is very similar to XMT-A, the MPEG-4 Scene Description format [7]. In terms of data volume, this information takes a very small part of the scene (for complex scenes, less than 5%). All this information is stored in one Adaptation Set, which is expected to be transmitted immediately when display starts. The size of the Scene Tree is very unlikely to be large, so it can arrive first and initiate display. Every other type of information (models, images etc.) can then begin transmission –the Scene Tree Code will contain appropriate interlinked placeholders for each of them, and put them in position as they arrive. This means that from the first moment, the scene "skeleton" is placed, and the user can begin navigation and interactions, as the actual visual data are being transmitted.

Table 1 The organization of X3D scene components in a media presentation description

X3D component media	Basic structuring unit	Partial representation
Scene tree code (structure, animations, interactions etc.)	Contained in an undivided *Adaptation Set*	None
Models	Each model in a separate *Adaptation Set,* along with its textures, with a different *Representation* for each Level of Detail	Model separated into *Segments* for streaming
Texture images	Images accompany a model within the same *Adaptation Set*. Each *Representation* of decreasing LoD can contain an image texture of similarly decreasing resolution	None
Texture videos	Videos accompany a model within the same *Adaptation Set*. Each *Representation* of decreasing LoD can contain a video texture of similarly decreasing resolution	None, or segmented and streamed through existing MPEG-DASH techniques
Audio files	Each audio file in a separate *Adaptation Set* with different *Representations* for different bitrates	None, or segmented and streamed through existing MPEG-DASH techniques (X3DOM extension)

Complex models in X3D (and X3DOM) are stored, in most cases, as Indexed Face Sets, that is a list of 3-dimensional vertex points, accompanied by an index table linking them into faces. Although multiple other model types exist (such as Line Sets, of Triangle Sets), Indexed Face Sets are the most common case. Such models can be split into multiple parts, interconnected or independent, and streamed progressively. Each different LoD of the model takes up a different Representation, within the Adaptation Set corresponding to the particular model. The methodology under which a model can be split into Segments and transmitted adaptively according to the MPEG-DASH standard is described in [14].

Image textures are organized in a similar manner, alongside the models they are supposed to cover. Each Representation of the model, corresponding to a different LoD, can be accompanied by a different image, of correspondingly varying quality. This means that, whenever the client chooses a model LoD as appropriate for the situation, the corresponding texture image is automatically chosen and downloaded as well.

Video texture files essentially function similarly to image texture files, in that they are videos that X3DOM lays over the corresponding surfaces. Thus, in the proposed integrated framework, video textures could be treated exactly the same as image textures. However, video is a highly streamable medium, and the MPEG-DASH standard is already used widely for video streaming. Recently, we proposed an approach with which X3D scenes can adaptively stream MPEG DASH-based video textures [13]. In that work, we presented a modification to the X3DOM <*Video*> node to allow it to play MPEG DASH-based video streams as model textures –essentially extending the X3DOM framework to play, besides the typical video formats (MPEG-4, WebM) also MPEG DASH-based video streams. Our method has been incorporated in the X3DOM framework distribution, and is now an integral part of the framework from version 1.6 onwards.

Finally, audio files are easier to manipulate, as, in X3D, they are not inherently interlinked with any model, but are simply placed in the scene. X3DOM supports most audio file types. Thus, an audio source is placed in a separate Adaptation Set, wherein multiple versions of the same audio file can be placed in corresponding Representations at different qualities (bitrates). Alternatively, we could resort to a similar approach to video [13], since the MPEG-DASH standard is already well-suited for adaptive audio streaming.

3.3 The media presentation description of web 3D scenes

While MPEG-DASH clearly states that its existing schema is designed to cover adaptive streaming for all information types, in practice the standard is rather focused on video and audio, and its structure is not necessarily appropriate for streaming 3D scenes. As a result, we reviewed the entire standard and explored the ways in which the existing elements and attributes comprising a complete and valid MPD description apply to our case. We consider it very important to point out that no actual modifications to the MPEG-DASH schema were done. Instead, the proposed approach produces valid MPD files that conform to the standard schema.

Tables 2 and 3 provide a condensed overview of the usage of all MPD elements and attributes in our proposed approach, starting from the main *MPD* Element down to the Segment organization. In both tables, XML Elements are written in italics and attributes are preceded by the symbol "@". In the rest of this section, we will provide a detailed description of the role these XML Elements and Attributes play in the adaptive streaming of Web 3D

Table 2 The MPEG-DASH "MPD" and "Period" elements, their child nodes and their usage in our approach

Element/Attribute name	General intended use	X3D streaming use (if different)
MPD element	Multimedia Media Presentation Description of a stream.	
@id	Unique MPD identifier. Also, Periods, AdaptationSets and Representations have their own IDs.	
@profiles	Defines profiles (Interoperability Points), according to which additional schema constraints are added. Examples include "Live", "Full", and "On Demand".	X3D streaming is based on the MPEG-DASH "Full" profile.
@type	Can be "Static", for on-demand streams, or "Dynamic" for live streaming.	In most cases "Static" is used, but there could be cases (e.g. an online multiplayer game) where data would have to be streamed live.
@availabilityStartTime	Time the stream will begin. Mandatory for "Dynamic" type.	Ignored, unless a dynamic application requires it.
@mediaPresentationDuration	Duration of the stream. Mandatory for "Static" type.	Zero-value. Zero signifies infinite duration.
@minBufferTime	Mandatory. Time length of the video buffer.	Ignored by the client.
ProgramInformation	Element used for encoding meta-information on the stream content	
BaseURL	Base URL for the stream. BaseURLs of child elements are appended.	
Location	The URI for the MPD file location.	
Period	Period elements corresponding to the consecutive stream time pieces.	Only use a single Period element, containing the entire scene.
Metrics	Available for any application-specific user reporting mechanism.	
Period element	A section of the stream corresponding to a time interval	The Period element containing the entire scene
BaseURL	Base URL for the Period. BaseURLs of child elements are appended.	
AdaptationSet	Adaptation Sets contain various modalities, such as video, audio and text.	Each Adaptation Set corresponds to a single model or audio file, with the addition of one more for the base X3D code.

scenes, and explain how a player application should interpret this information in order to achieve adaptive streaming of complex Web 3D material. Figure 2 presents a short, indicative MPD structure modeling an X3D scene with two models.

The Element hierarchy we will present will be focused on four element levels: the *MPD* Element, the *Period* Element, the *AdaptationSet* Element, and the *Representation* Element. Besides those, there are also a number of attributes that appear on multiple Elements, known as the *commonAttributesElements* attributes. The tables list all MPD elements and attributes that we currently deem relevant to our streaming approach. Thus, we list not only the aspects that we have repurposed or interpret differently due to the different medium, but also those that are necessary for our adaptive streaming model while being used in the same manner both in video and 3D scenes.

The central attribute of an MPD document is the "Profile". Profiles add constraints to the schema according to their specific definition. In our case, since these profiles are generally

Table 3 The MPEG-DASH "AdaptationSet" and "Representation" elements, their child nodes and their usage

Element/Attribute Name	General intended use	X3D streaming use (if different)
AdaptationSet Element	Can contain any media modality, such as video, audio and text.	A single model, the base X3D code or an audio stream.
@minBandwidth	A guideline concerning the minimum bandwidths for the children Representations.	
@maxBandwidth	A guideline concerning the maximum bandwidths for the children Representations.	
@segmentAlignment	Define (True/False) whether segments overlap (T) or not (F) between different representations.	Segments for different representations could be built correspondingly, in order to cover the exact same model area.
@bitstreamSwitching	Whether we can switch mid-segment without waiting for the period to change.	Would be feasible for segmentAlignment="True".
commonAttributesElements*	A group of attributes used in both AdaptationSets and Representations. See below (*).	
BaseURL	The URL corresponding to the Adaptation Set. Appended to the URL of the Period.	For the X3D base code, this is the URL to the containing file (or the service transmitting it).
Representation Element		
@qualityRanking	A designer-imposed quality evaluation ranking to allow clients to know which of multiple Representation "looks & feels" better.	A sequential ranking of the various LoDs within an Adaptation Set, from the best (finest) to the poorest (coarsest).
@dependencyId	If Representations each depend on one another as sequential refinements, this attribute contains the ID of the dependency Representation.	For progressive refinement, higher representations can depend on the lower ones, without having to repeat the streaming process.
@bandwidth	Indicative estimate of the bandwidth necessary for smooth play. For dependent Representations this is the minimum bandwidth as defined above.	Indicative estimate of the minimum bandwidth necessary to play this Representation (Level of Detail).
CommonAttributesElements*	See below (*).	
BaseURL	The URL corresponding to the Representation. Appended to the URL of the Adaptation Set.	The URL corresponding to the Representation. If on-the-fly segmentation occurs, this is the final URL of the segmentation service.
SegmentBase, SegmentList, SegmentTemplate	Used to describe Segments. Segments described here override all previous ones.	Each Segment corresponds to a part of the entire model, either encoded as an independent X3D IndexedFaceSet, or as a function of the previously downloaded Segments.
*commonAttributesElements group of attributes		
@codecs	Mandatory field describing the codec.	Takes the value "none" for Model and Scene Tree Adaptation Sets. For Audio files, contains the name of their codec.
@mimeType	MIME Type of the contained media.	"model/x3d + xml"
ContentProtection	Information on protection schemes.	

```
<?xml version="1.0" encoding="UTF-8"?>
<MPD xmlns="urn:mpeg:dash:schema:mpd:2011"                    [MPD declaration and schema specification]
  xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
  profiles="urn:mpeg:dash:profile:full:2011" minBufferTime="PT0S"
  mediaPresentationDuration="P0D" type="static"
  xsi:schemaLocation="urn:mpeg:DASH:schema:MPD:2011
  DASH-MPD.xsd">
  <BaseURL>http://mclab1.medialab.teicrete.gr:8081</BaseURL>  [The URLs of the HTTP servers where the
  <BaseURL>http://testHost.medialab.teicrete.gr:8081</BaseURL> Segment files are located]
  <BaseURL>http://alternativeHost.host-services.gr:8081</BaseURL>
  <Period id="demo_scene">                                    [A single Period element containing the entire scene]
    <AdaptationSet id="SceneTree" mimeType="model/x3d+xml"    [The Adaptation Set containing the 3D
    codecs="none"    minWidth="280" minHeight="500">          Scene Tree]
        <Representation id="ST1" bandwidth="0">               [The Scene Tree only contains one Representation,
                                                              as there are no quality levels for scene structure]
          <BaseURL>SceneTree.x3d</BaseURL>                    [BaseUrl paths are appended to the low-level ones]
        </Representation>
    </AdaptationSet>
    <AdaptationSet id="Model1" mimeType="model/x3d+xml"       [An Adaptation Set containing a 3D model in three

    codecs="none" minFrameRate="10">                          different quality levels]
      <Representation id="M1_1" bandwidth="500000"            [Low quality version, requires the less bandwidth]
      qualityRanking="3">
        <BaseURL>Model1_LoD3.x3d</BaseURL>                    [The URL of the X3D file corresponding to the model.
                                                              Appended to the root BaseURLs to locate the file]
      </Representation>
      <Representation id="M1_2" bandwidth="1500000"           [Medium quality version]
      qualityRanking="2">
        <BaseURL>Model1_LoD2.x3d</BaseURL>
      </Representation>
      <Representation id="M1_3" bandwidth="3500000"           [High quality version, requires the highest bandwidth]
      qualityRanking="1">
        <BaseURL>Model1Full.x3d</BaseURL>
      </Representation>
    </AdaptationSet>
    <AdaptationSet id="Model2" mimeType="model/x3d+xml"       [A second Adaptation Set containing another 3D model]
      codecs="none">
      <Representation id="M2_1" bandwidth="500000"            [Only one version of the model is given]
      qualityRanking="1">
        <BaseURL>Model2_Full.x3d</BaseURL>
      </Representation>
    </AdaptationSet>
  </Period>
</MPD>
```

Fig. 2 *Left* A sample MPD structure in XML. *Right* a brief description of the role and purpose of certain key MPD elements

designed to optimize video description and delivery, these have little value for us, so we recommend the use of the "Full" profile for 3D data. Another important feature is *type* attribute of the MPD element. The values it can take are "*Static*" or "*Dynamic*". The attribute aims at making the distinction between on-demand and live streaming sources, and is accompanied by the @*availabilityStartTime* for dynamic streams, to inform players when the stream will start, and @*mediaPresentationDuration* for static streams, to inform players of the full duration. This information is so narrowly video-specific that, while mandatory to fill in order to conform to the standard, has little to contribute to an X3D stream. We consider all cases we have dealt with as Static, although it is conceivable that an interactive application, such as an online game, might be drawing live information, and the stream might classify as Dynamic. In our case, when giving the *Static* value to the attribute, the accompanying @*mediaPresentationDuration* attribute takes the value "0", which is allowed by the standard and corresponds to streams of infinite duration.

Similarly, a number of video-specific attributes exist in the MPEG-DASH standard schema that are mandatory and have to take values, but have no meaning for the 3D streaming case. Such attributes are @*availabilityStartTime*, @*minBufferTime* and @*codecs*. The two first can take any value, since an X3D MPEG-DASH streaming client would ignore them as irrelevant to our case. The value of @*codecs* varies with the Adaptation Set type: for audio and video

streams, it takes the value of the codec used to encode the stream, while for models and the Scene Tree it takes the value "none", since such data are not encoded using any standard media codec. Finally, since a declarative X3D scene has no temporal axis through which to segment it, the entire *Period* element is not particularly useful, and in our proposed method, it is only used once, to enclose the entire scene in a single *Period* element.

On the other hand, the rest of the MPD structure has an important role to play in adaptive 3D streaming. Following the scene organization described in Section 3.B., the MPD attributes and elements serve to set up the adaptation scenario that a player can follow to create the desired user experience. A fundamental piece of information is contained in the *BaseURL* element, which is present at each level of the MPD hierarchy. At each level, multiple *BaseURL* elements may exist, providing alternative server sources for the media files. As we move down the hierarchy, the *BaseURL* values are concatenated, so that at the lowest level (that of the Representation or Segment) we have a final, complete URL attribute. The attribute specifies the HTTP location of the media file that the player must download. With respect to the adaptation process, the MPD structure contains certain attributes *(@minBandwidth* and *@maxBandwidth* for Adaptation Sets, and *@bandwidth* for Representations) that are aimed to serve as guidelines for the client to perform adaptation. We use these fields to declare the recommended bandwidth range within which the corresponding Representation can be streamed smoothly, but have to keep in mind that the values given can only serve as recommendations. In our case, in which we have more complex considerations than simply bandwidth, such as the availability of computational or memory resources, adaptation should take place with the help of performance metrics covering more aspects than simple bandwidth. However, we do take advantage of these two fields, as well as the *@qualityRanking* attribute, which serves to rank representations from the best to the poorest, so that the player can have an overview of the quality levels available.

Finally, a group of attributes exist which are particularly relevant to specific scenarios in streaming Web 3D scenes. A pair of potentially very powerful such attributes are *@segmentAlignment* and *@bitstreamSwitching* in the AdaptationSet level. In our case, *@segmentAlignment*="*true*" would suggest that the corresponding Segments in different Representations contain the exact same model parts, which means that a Segment from one Level of Detail could "fit" with a neighboring one, from another LoD. In this case, *@bitstreamSwitching* would also be true, since we could switch Representation mid-model. Finally, another attribute related to Web 3D is *@dependencyId*, at the Representation level. This allows us to structure Representations as progressive sequences over one another, in that the more high-quality ones require the coarser ones to have been downloaded first. This allows smooth integration of Progressive Mesh –like encodings, such as the ones described in Section 2.

4 Prototype X3D MPEG-DASH client

Like the original MPEG-DASH standard, the heart of our proposed framework is the Media Presentation Description, and, through it, the way that a complex Web 3D scene can be streamed to a client (player) with sensitivity to the user's QoE. Having established a standardized MPD representation for Web 3D scenes, the development of a client able to play DASH-encoded media can take place as a separate task. That is, while the development of an MPEG-DASH client is closely knit to the MPD it is expected to translate, anyone having access to the

standard should be able to independently develop its own client, and each client should be able to feature different design choices as to how to deliver content to the users given the MPD of a Web 3D scene, for example concerning resource allocation or speed/quality tradeoffs. As a prototype for evaluating the feasibility of our approach and as a test-bed for evaluations, we are also developing our own 3D streaming client. We have based our client on HTML5 and JavaScript, inspired by DASH-JS, the reference JavaScript client for MPEG-DASH video currently supported by the DASH Industry Forum.[1] Our main motivation for this choice was the fact that HTML5 inherently solves the issue of device-independence and allows the same code to run on any device with a modern browser.

The client we have designed is described in Fig. 3. The process begins by the client receiving an.mpd file from a web server. Initially, the client places an empty X3DOM scene in the page. The client then scans the Adaptation Sets contained in the MPD, locates the Scene Tree Adaptation Set and places it in the X3DOM scene as an empty structure. Consecutively, the client begins downloading and placing data from the other adaptation sets within the Scene Tree. Using the X3DOM framework, the visual and audio content begins being rendered the instant it arrives.

In our implementation, the client begins by downloading very low-quality data, and at the same time begins sampling on a series of local metrics concerning the status of the device and its environment, with respect to its features (e.g., screen resolution) and available resources (e.g. computational power or available bandwidth). If the client deduces that enough resources are available in order to play a more detailed Representation, it seeks higher LoD Representations from the Media Presentation Description, and begins to download them and display them immediately. When streaming a model, the client requests each segment via HTTP. The moment the segment arrives, its rendering begins, while the client simultaneously requests the next one, thus pipelining the streaming process by simultaneously rendering the delivered segments in the GPU and downloading the next ones through the network connection.

As HTML5 continues to evolve, JavaScript offers more ways to interact with the hardware and the environment without compromising security. Currently, HTML5 APIs enable the monitoring of battery status and memory state, simple timing measurements for performance evaluations or network traffic estimations, while the X3DOM framework itself offers display metrics, such as the achieved framerate (measured in Frames per Second, FPS). This capability, which is always enriched with new features as the web technologies evolve, is a powerful arsenal of utilities upon which we can base our adaptation policy on the part of the client. However, it must always remain clear that MPEG-DASH standardization does not deal with detailed client policies. Instead, the behavior of a client in the face of an.mpd document depends on the aims of the application.

The current publicly available implementation of our prototype client[2] can handle adaptive streaming of X3D scenes either using on-the-fly segmentation or using pre-segmented models. To test that, we have also implemented a streaming server that can provide both forms of content, either by storing scenes with pre-segmented models, or by segmenting them on the fly and distributing them as a REST service. We are working to extend our implementation to include sound and more complex phenomena in scenes, but it should be kept in mind overall that, any attempt to build a client for adaptive Web 3D streaming will have to include certain decisions on system and environment capabilities that may be controversial: for example, a conceivable use-case is when both the GPU and the network connection are loaded. In that case, the client might opt to replace the scene currently being played with a lower quality one,

[1] https://github.com/Dash-Industry-Forum/dash.js
[2] http://www.medialab.teicrete.gr/minipages/MpdStreaming/index.html

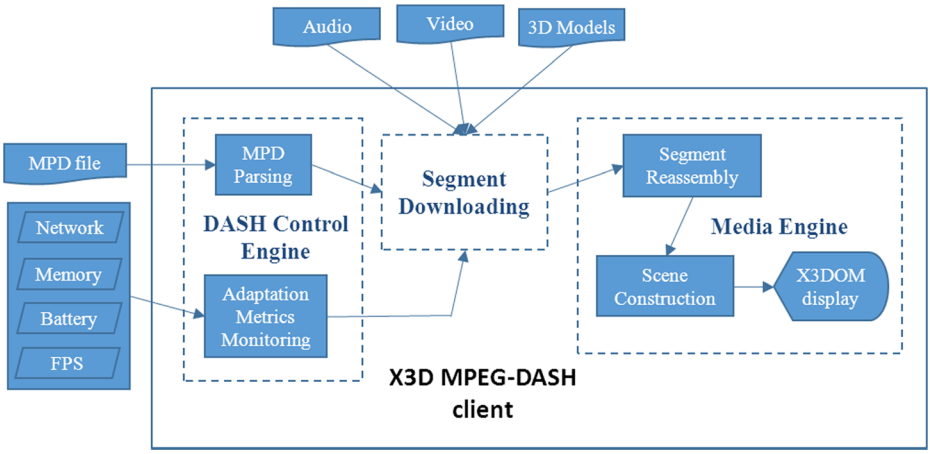


Fig. 3 Functional diagram of the prototype X3D MPEG-DASH client

using the lower LoD Representations provided by the MPD to replace the existing ones. This would lighten the load on the GPU and might sound like a reasonable decision in terms of computational cost. However, if the lower LoD Representations were not already stored locally, for example because the player did not maintain a cache of all possible models at all times, or because playback had begun at a mid-quality LoD and the lower qualities were never downloaded in the first place –which may be the case in dynamic streams, such as VR games-, such a decision would mean we would have to download all the low LoD models necessary for the shift in quality. This might increase traffic and, in the presence of limited bandwidth, might once again lead to poor QoE.

This example demonstrates how the prioritization philosophy for different aspects of user experience, using different metrics, will necessarily vary depending on the implementation. In the prototype we are developing, for demonstration purposes, we are maintaining a debugging approach, by making available all the QoE-related metrics we are collecting, regardless of which we choose to use for adaptation (Fig. 4), and the client always begins by downloading the lowest-quality Representations while evaluating the system and environment capabilities for downloading and rendering higher ones.

5 Evaluation of the system

In the process of evaluating and adapting the behavior of our streaming framework we have performed a series of evaluation runs over our prototype client. A scene featuring one very large model (~360,000 faces) was streamed to a PC, and the segment and model arrival times were recorded. The choice of a scene containing a single model may appear simple, but it is a setup that allows us to fully control the client behavior for our evaluations. The benchmark computer's characteristics were: CPU: Intel Core i7-3630QM 2.40GHz, GPU: AMD Radeon HD 7600 M Series 2GB, RAM: 4GB, OS: Windows 8.1, and the browser used was Google Chrome. The streaming process we implemented for the models was the same as in [15] with the difference that now we pre-segmented the streamed models into independent face sets, and sent them to the client upon its request for display. In the context of MPEG-DASH adaptation, multiple versions of the same model were created, using the Quadratic Edge Collapse

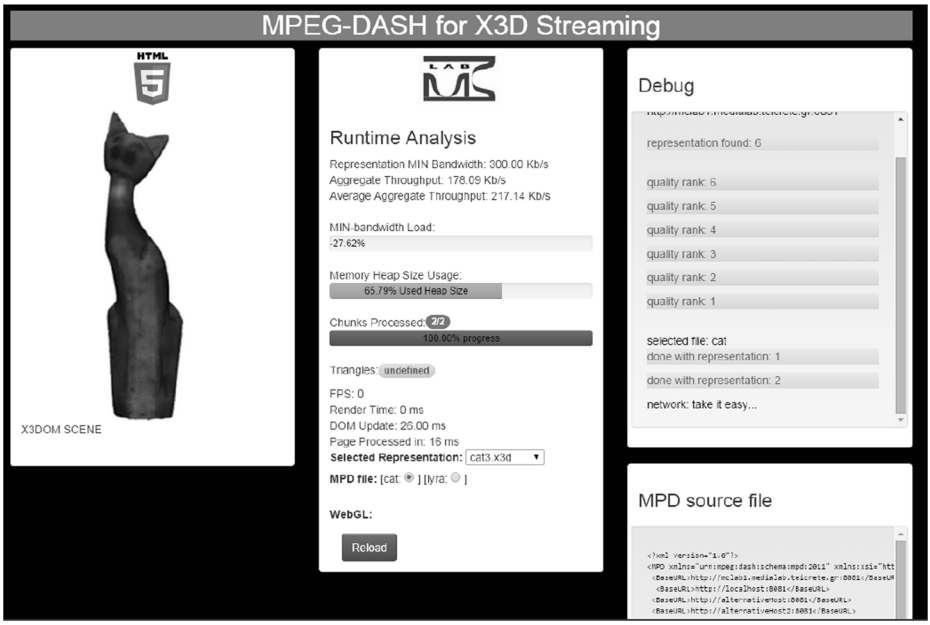


Fig. 4 The Prototype X3D MPEG-DASH client interface

algorithm. The original ("LoD 4") version of 360,000 faces was reduced to 100,000 faces ("LoD 3"), 40,000 faces ("LoD 2") and finally 8000 faces ("LoD 1"). The resulting Representations can be seen in Fig. 5. The essential aim of an adaptive MPEG-DASH streaming framework is to balance the tradeoff between model quality and computational/network load, so as to provide the user with the best possible experience by delivering the best possible model that the existing device and network infrastructure can provide.

One extremely important aspect of the system we were interested in exploring was the segment size. In essence, the segment size is what defines a streaming, QoE-aware, framework as such: this is due to the fact that, if the segment size is too large (e.g., as large as the model), we have no streaming, but instead the user waits in front of a blank screen for the model to download entirely and then appear. In contrast, reducing the segment size and correspondingly increasing the number of segments may lead to increased traffic overhead, as each segment is transferred using a separate HTTP request, and increased computational load as the client tries to seam the individual segments together. We thus proceeded to investigate the impact of segment size on performance.

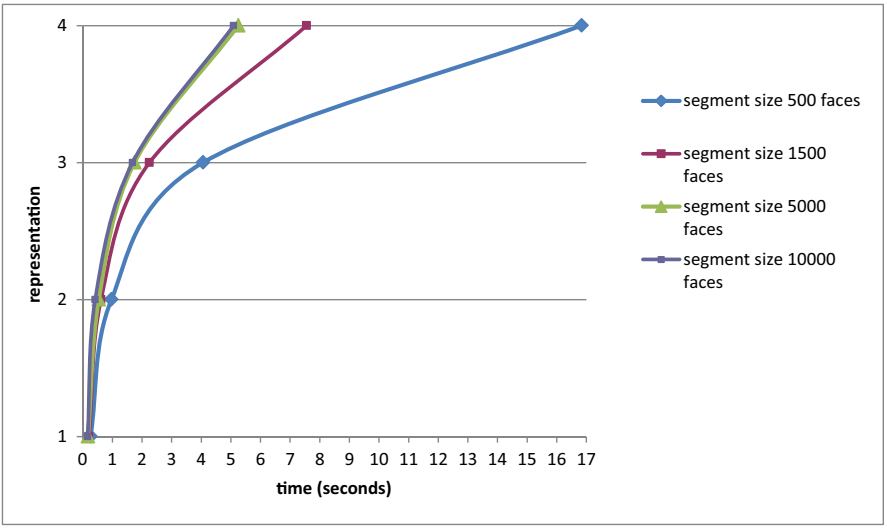
In Fig. 6, we present how our prototype client can be used for parameter evaluation in the streaming process. In order to explore the impact of segment size on the system performance, we performed multiple streaming runs for four different segment sizes (500, 1500, 5000 and

Fig. 5 The four model Representations used in our streaming evaluations. From left to right: Representation 1: 8000 faces, Representation 2: 40,000 faces, Representation 3: 100,000 faces, Representation 4: 360,000 faces

10,000 faces, respectively), and recorded the time that each Representation requires in order to be downloaded and rendered.

One major observation we can make is the very long waiting times for displaying the highest LoD and the smallest segment sizes. This suggests that segment sizes of around 500 faces are inefficient due to the heavy transmission and processing load that induces their large number. On the other hand, too large segment sizes negate the entire concept of progressive transmission; too large a segment size leads to the model being transmitted in a few big parts, thus failing to provide the user with the sense of smooth streaming. However, we notice that the larger segment sizes do not affect equally the streaming duration. Thus, segment sizes of 5000 faces achieve almost equal streaming delays with the ones of 10,000 faces, without affecting the smoothness of the streaming procedure. Based on the results presented in Fig. 6, for the second round of evaluations, we chose a segment size of 5000 faces. This value has the advantage of visually giving a smooth user experience, in the sense that the segments are small enough to allow the model to appear progressively starting from early on, but not so small as to make streaming forbiddingly slow.

The notice that segment sizes of 5000 faces achieve almost equal streaming delays with the ones of 10,000 faces underlines that for segment sizes close to 5000 faces, or larger, their transmission delay is not as important as their processing delay. This means that in the time our client takes to process a segment of this size and put it in the scene, the next segment has arrived for processing. So, the measured delay is mostly the time our client needs to seam the segments and render the full model rather than downloading them. This time inevitably cannot be avoided or reduced since it is, to its great extent, GPU-dependent and more specifically

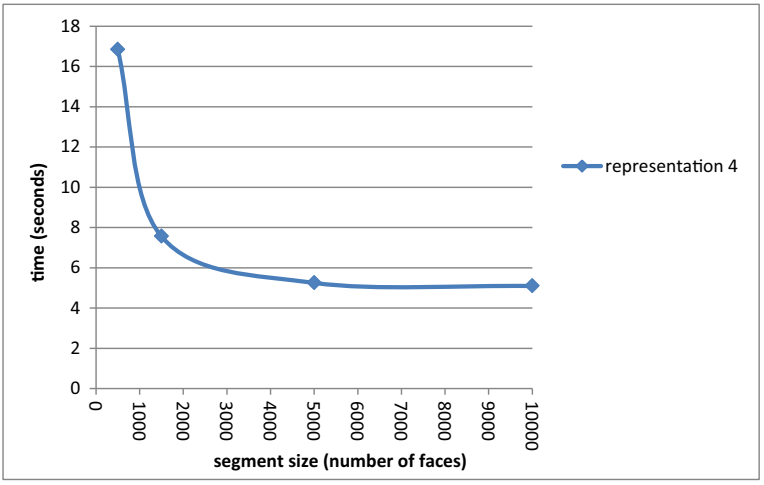


representation	segment size 500 faces	segment size 1500 faces	segment size 5000 faces	segment size 10000 faces
1	0,2636 sec	0,2112 sec	0,1764 sec	0,1644 sec
2	0,97008 sec	0,6174 sec	0,542 sec	0,4208 sec
3	4,0588 sec	2,2532 sec	1,7464 sec	1,6818 sec
4	16,837 sec	7,5576 sec	5,249 sec	5,0924 sec

Fig. 6 Evaluation of the time of delivery of 3D model faces over our MPEG-DASH streaming framework for different segment sizes (bandwidth: 100 Mbps)

dependent on how WebGL communicates with the underlying GPU for rendering graphics. We mention here that WebGL data must first be uploaded from general memory to special WebGL data structures before being rendered in the GPU. These special WebGL data structures are called WebGL textures (bitmap images) and WebGL buffers (generic byte arrays). Once data is uploaded, rendering is really fast, but uploading is generally slow. This creates a bottleneck in the rendering of web graphics, in general.

Taking into account that MPEG-DASH is a web technology, we understand that there is a performance limit, here, we cannot overcome with our browser-based MPEG-DASH client. Figure 7 illustrates exactly this limitation since it visualizes that models comprising segments of sizes larger than 3000 faces take almost the same time to be rendered. For this curve we have used the same measurements with Fig. 6 but now we depict the time Representation 4 takes to be downloaded in the client as a function of the segment size. The same curve form is taken for all Representations but for simplicity we illustrate in Fig. 7 just the heaviest Representation of them. The experiment depicts that our client cannot fall below the limit of 5 sec for this model, whatever the segment size will be, unless a totally different software engineering approach for our DASH client is adopted. The latter could include the development of our client as a native application, so it avoids the limitation of WebGL data structures and exploits optimally the multi-core capabilities of the underlying GPU hardware. Alternatively, we could try to use SIMD.js, a promising experimental technology that wishes to bring data parallelization into web applications, but is not currently available for commercial applications. However, this outcome is a reality for all WebGL-based applications, since full access to the multiple cores of a GPU system is currently restricted from within a browser. As mentioned above, GPU memory

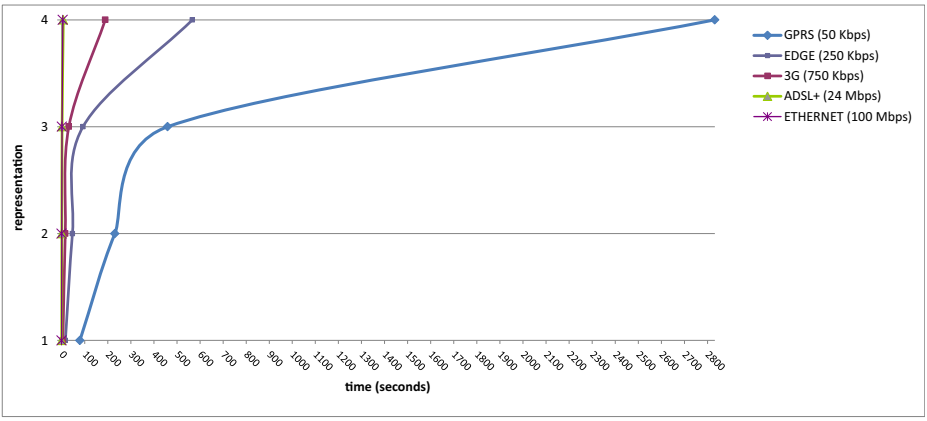


segment size	representation 4
500 faces	16,837 sec
1500 faces	7,5576 sec
5000 faces	5,249 sec
10000 faces	5,0924 sec

Fig. 7 Evaluation of the time of delivery of a 3D model over our MPEG-DASH streaming framework for different segment sizes (bandwidth: 100 Mbps)

is abstracted by WebGL data structures such as textures and buffers. Data is best uploaded once to WebGL and then used many times by the GPU. The uploading is slow by itself, and by uploading data right before rendering with it, the GPU has to wait for the data to upload before it can proceed with rendering. This explains why using small segment size in our experiments achieves so large times. GPUs are intended to be used to draw large batches of faces at once. When we have for example 5000 faces to draw, doing it in one single operation will be much faster than doing 5000 separate draw operations of one face each.

Similarly, using a fixed segment size, we can use our framework to design the Media Presentation Description with respect to the different networking environments a scene might be streamed over. By keeping the segment size fixed at 5000 faces, we simulated a number of bandwidth and network constraints, using Google Chrome's native interface. The five scenarios we evaluated were: "GPRS - 50 Kbps", "EDGE - 250 Kbps", "3G - 750 Kbps", "ADSL High - 24 Mbps", and "ETHERNET – 100 Mbps". Figure 8 presents the evaluation results of our streaming framework performance under these five scenarios. It is clear that, for GPRS and EDGE there is little point in going beyond Representation 1, as the time demands are extremely high. In fact, in the case of GPRS, even receiving Representation 1 might be an issue. With respect to the high-quality representations, on the other hand, while 3G can support up to Representation 2 without prohibitive delays, Representations 3 and 4 seem to require an ETHERNET or ADSL connection to display within a reasonable time frame. The outcome from Fig. 8 that ADSL and ETHERNET achieve similar streaming times despite their great bandwidth difference, assures our previous result that for segment sizes larger than 3000 faces the streaming delay is mostly processing than transmission based. The above observations can be used to determine the behavior of the stream through its Media Presentation Description, for example by appropriately setting the corresponding *@bandwidth* attribute for each representation.

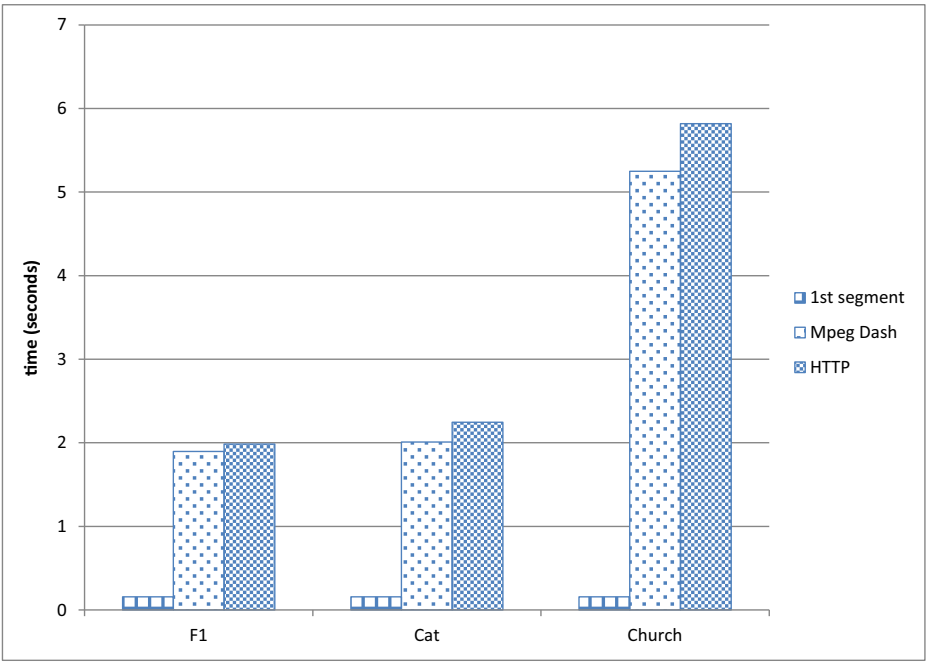


representation	GPRS (50 Kbps)	EDGE (250 Kbps)	3G (750 Kbps)	ADSL+ (24 Mbps)	ETHERNET (100 Mbps)
1	79,167 sec	16,167 sec	5,5075 sec	0,285 sec	0,1764 sec
2	230,969 sec	46,794 sec	15,636 sec	0,853 sec	0,542 sec
3	459,668 sec	92,669 sec	31,25 sec	1,965 sec	1,7464 sec
4	2831,827 sec	567,046 sec	189,2175 sec	7,2525 sec	5,249 sec

Fig. 8 Evaluation of the rate of delivery of 3D model faces over our MPEG-DASH streaming framework for different bandwidths (segment size: 5000 faces)

For reasons of comparison, we also ran a set of comparisons between our MPEG-DASH streaming approach and model transfer via direct HTTP request (i.e. without streaming). For MPEG-DASH streaming, we measured the time until the first segment was displayed, thus initiating the user experience, and the time to completion of the stream. Obviously for the case of direct HTTP request, the time the full model is displayed coincides with the time the user starts to view something in his browser. Figure 9 shows the results of our measurements. We tested our platform for three different 3D models of different complexity, namely the model of a F1 racing vehicle shown in Fig. 10 ("F1", ~40,000 faces), the 3D scan of a prehistoric cat idol shown in Fig. 5 ("Cat", ~80,000 faces) and the 3D scan of a historic church shown in Fig. 4 ("Church" ~350,000 faces). The segment size was set at 5000 faces, the bandwidth at 100 Mbps and the times reported were measured from the moment the user requested a model to the moment the model (or segment) was displayed on the user's screen, thus including delays due to the HTTP request-response round-trip times, and delays due to GPU processing via WebGL.

As shown in Fig. 9 the main goal of the streaming approach, that is to provide the user with some content as soon as possible, is indeed achieved with our framework. Indeed, in all cases the display experience with MPEG-DASH streaming begins significantly earlier than with direct HTTP transfer,



	F1	Cat	Church
1st segment	0,1584 sec	0,1592 sec	0,1598 sec
MPEG -DASH	1,8974 sec	2,0104 sec	5,249 sec
HTTP	1,9866 sec	2,2486 sec	5,8194 sec

Fig. 9 Comparison of the time (in ms) between request and display, for three different models: model of an F1 racing vehicle shown in Fig. 9 (F1), the 3D scan of a prehistoric cat idol shown in Fig. 5 (Cat) and the 3D scan of a historic church shown in Fig. 4 (Church)

as indicated by the first column ("Time for first X3D segment"). However, with respect to the transmission of the entire model, Fig. 9 demonstrates that our streaming approach achieves slightly faster times than pure HTTP downloading. Obviously this is due to the outcome that performance of our client cannot fall below a certain limit that has to do with processing of the X3D segments (sewing of the segments and rendering of the scene) at the client's side by WebGL. So, segmented or not, the reconstruction of a 3D scene with a certain number of faces by WebGL requires an inevitable amount of time. However, this performance limitation will be common for all browser-based solutions, as we explained, since full access to the multiple cores of a GPU system is currently prohibited from within a browser. As soon as it is a reality, our DASH-based streaming of 3D models will be faster than simple HTTP streaming, since the exploitation of the underlying hardware cores will enable parallelism of the Media Engine accelerating the rendering of web 3D scenes.

At this stage of work, our client serves as a proof of concept for our 3D streaming framework. It is able to handle the increased complexity of streaming large, multimodal scenes. Figure 10 shows a snapshot of our client, featuring a scene with multiple high-quality 3D models and a video stream in parallel. In more specific, the prototype client supports the presence and synchronization of multiple X3DOM models with MPEG-DASH video textures. The latter means that we can stream MPEG-DASH videos from within an X3DOM scene, as it is illustrated in Fig. 10, where the virtual world includes a cinema screen behind the F1model for the projection of a DASH-based streaming video. This way, the entire 3D world is at first streamed following the MPEG-DASH paradigm, and then the video is also streamed within the world using DASH. Thus an MPEG-DASH-within-MPEG-DASH approach is achieved [13]. However, within the current framework, although we have incorporated MPEG-DASH video as a native aspect of the X3DOM framework, we are still handling audio information in separate adaptation sets containing multiple qualities managed by the overall client. Taking into account that in the X3DOM framework, audio is very similar to video in the way it is handled, it would be straightforward to extend the X3DOM framework to play.mpd audio streams, just as we have done with videos. Then, adaptive audio streams would be played using their own client within the X3DOM player. The management of such multiple sources of information in the same scene will allow uniform control of all media resources opening the path towards optimization of user experience.

6 Conclusions and future directions

The work we presented here advances the current state of the art, by introducing adaptive Web 3D scene streaming that follows the rules of the MPEG-DASH standard. We placed our focus primarily on the structuring of the Media Presentation Description for describing complex Web 3D scenes, and the ways that the MPEG-DASH standard could be repurposed for our needs.

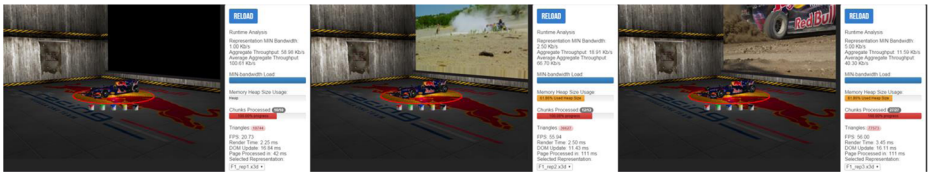


Fig. 10 A sample complex MPEG-DASH scene containing both X3D models and streaming video. The scene is shown at three consecutive time instances, with the model and the video being streamed consecutively

We proposed an adaptation of the standard that does not violate the MPEG-DASH schema, does not require additional elements or attributes, and strictly follows the MPD structure.

The proposed framework smoothly fuses multiple geometries, scene information, textures, sound and video. Our prototype client already implements a number of these aspects, and we are in the process of refining and extending it to fully cover all proposed framework specifications. Other clients could be easily developed, either for the Web, using X3DOM or other WebGL-based frameworks, or as stand-alone applications -still aimed at displaying Web 3D graphics, however, similar to classic X3D stand-alone players.

Our work, besides its tangible contribution of an adaptive streaming framework, also opens a number of future research directions with respect to Web 3D multimedia. Essentially, we are proposing an integrated framework which could apply ubiquitously to all devices, environments and situations for Web 3D display. However, a necessary step towards this goal would be the implementation of a system for the automatic generation of Media Presentation Descriptions from X3D/X3DOM scenes, including the generation of different LoD models.

By streamlining the adaptive streaming process, based on the proposed framework, the way we view Web 3D data can be radically altered. Recently, we proposed a framework for integrating physics in the X3DOM framework [25] by implementing the rigid body component of the X3D specification. Our contribution has been integrated in X3DOM, similarly to our MPEG-DASH video texture component. These extensions tend to make X3DOM a self-contained, fully functional environment in which scenes can automatically be reproduced and generate natural animations and interactivity functionalities. We intend to further work on the integration between these Web 3D methodologies (physics, MPEG-DASH adaptive streaming) in order to provide a seamless, adaptive, QoE-aware environment that can provide the best interactive, immersive experience for any user, in any device and at any environment.

Furthermore, a new generation of streaming games may be implemented, based on progressive inline distributed objects, either synthetic (graphics) or natural (audio-visual), that are fully functional into different QoE-levels and the objects (and consequently the game's Level of Detail and resolution) may be adapted to the user throughput and end-system requirements.

Finally, our methodology for adaptive streaming of entire 3D scenes could ideally fit the requirements for virtual city modeling, where 3D models of entire cities need to be integrated with various GIS data [21]. The latter can also concern real-time data produced from Internet of Things establishments [20]. In such cases, the heavy load of the transmitted data makes critical the mediation of a streaming system that will adapt the streamed content according to the network conditions. Hence, a system similar to the one introduced in this paper, could perfectly integrate IoT data into X3Dom-based city scenes and adaptively stream them.

Acknowledgements The research of this paper is granted by the European Union and the Hellenic General Secretary of Research and Technology under the "COOPERATION 2009/09SYN-72-956" Framework.

References

1. Alliez P, Desbrun M (2001) Progressive compression for lossless transmission of triangle meshes. Computer Graphics (SIGGRAPH-01), Proceedings of the Annual Conference on, 195–202, ACM
2. Behr J, Jung Y, Franke T, Sturm T, (2012) Using images and explicit binary container for efficient and incremental delivery of declarative 3D scenes on the web. Web3D Technology (Web3D), 17th International Symposium on, 17–25, ACM

3. Behr J, Jung Y, Keil J, Drevensek T, Zollner M, Eschler P, Fellner D (2010) A scalable architecture for the HTML/X3D integration model X3DOM. Web 3D Technology (Web3D), 15th International Conference on, 185–194, ACM
4. Bourdena A, Kormentzas G, Pallis E, Mastorakis G (2013) Radio resource management algorithms for efficient QoS provisioning over cognitive radio networks. Communications (ICC), 2013 I.E. International Conference on, 2415–2420
5. Brooks RJ, Tobias AM (1996) Choosing the best model: level of detail, complexity, and model performance. Math Comput Model 24(4):1–14
6. Carmona-Murillo J, Gonzalez-Sanchez JL, Cortes-Polo D, Rodriguez-Perez FJ (2014) QoS in Next generation mobile networks: an analytical study. In Mavromoustakis CX, Pallis E, Mastorakis G (eds) Resource management in mobile computing environments. Springer International Publishing, pp. 25–41
7. Coding of audio-visual objects - Part 11: scene description and application engine, ISO/IEC 14496–11:2014
8. Dynamic adaptive streaming over HTTP (DASH) — Part 1: media presentation description and segment formats, ISO/IEC 23009–1:2014
9. Extensible 3D (X3D), ISO/IEC 19775:2004/Am1:2006
10. Extensible 3D (X3D) encodings, ISO/IEC 19776:2005
11. Gandoin PM, Devillers O (2002) Progressive lossless compression of arbitrary simplicial complexes. Computer Graphics and Interactive Techniques (SIGGRAPH-02), Proceedings of the ACM Conference on, 372–379, ACM
12. Hoppe H (1996) Progressive meshes. Computer graphics (SIGGRAPH-96), Proceedings of the ACM Conference on, 99–108, ACM
13. Kapetanakis K, Panagiotakis S, Malamos AG, Zampoglou M (2014) Adaptive Video Streaming on top of Web3D: A bridging technology between X3DOM and MPEG-DASH. Telecommunications and Multimedia (TEMU), Proceedings of the 6th IEEE International Conference on, 226–231
14. Kapetanakis K, Zampoglou M, Malamos AG, Panagiotakis S, Maravelakis E (2014) An MPEG-DASH framework for QoE-aware Web3D streaming. International Journal of Wireless Networks and Broadband Technologies (IJWNBT) 3(4):1–20. doi:10.4018/ijwnbt.2014100101, **IGI Global**
15. Kapetanakis K, Zampoglou M, Milionis F, Malamos AG, Panagiotakis S, Maravelakis E (2014) State-of-the-art web technologies for progressive presentation of synthetic cultural heritage scenes, Information, Intelligence, Systems and Applications (IISA 2014), Proceedings of the 5th IEEE International Conference on, 211–216
16. Lavoue G, Chevalier L, Dupont F (2013) Streaming compressed 3D data on the web using javascript and webGL, Web3D Technology (Web3D) 18th International Conference on, 19–27, ACM
17. Limper M, Jung Y, Behr J, Alexa M (2013) The POP buffer: rapid progressive clustering by geometry quantization. Comput Graph Forum 32(7):197–206
18. Limper M, Wagner S, Stein C, Jung Y, Stork A (2013) Fast delivery of 3D web content: a case study. Web 3D Technology (Web3D), In Proceedings of the 18th International Conference on, 11–17, ACM
19. Luebke DP (2003) Level of detail for 3D graphics. Morgan Kaufmann
20. Lv Z, Li X, Zhang B, Wang W, Zhu Y, Hu J, Feng S (2016) Managing big city information based on WebVRGIS. IEEE Access 4:407–415
21. Lv Z, Réhman SU, Chen G (2013). Webvrgis: A p2p network engine for vr data and gis analysis. In International Conference on Neural Information Processing (pp. 503–510). Springer Berlin Heidelberg
22. Mamou K, Dehais C, Chaieb F, Ghorbel F (2010) Shape approximation for efficient progressive mesh compression. Image Processing (ICIP), Proceedings of the IEEE International Conference on, pp. 3425–3428
23. Narayanan RGL (2014) Mobile video streaming resource management. In Mavromoustakis CX, Pallis E, Mastorakis G (eds) Resource management in mobile computing environments, Springer International Publishing, pp. 461–480
24. Peng J, Kuo CCJ (2005) Geometry-guided progressive lossless 3D mesh coding with octree (OT) decomposition. ACM Trans Graph 24(3):609–616
25. Stamoulias A, Malamos AG, Zampoglou M, Brutzman D (2014) Enhancing X3DOM declarative 3D with rigid body physics support. 3D Web Technologies (Web3D), Proceedings of the Nineteenth International ACM Conference on, 99–107, ACM
26. The Virtual Reality Modeling Language (VRML) – Part 2: external authoring interface (EAI), ISO/IEC 14772–2:2004
27. The Virtual Reality Modeling Language – part 1: functional specification and UTF-8 encoding, ISO/IEC 14772–1:1997
28. Valette S, Chaine R, Prost R (2009) Progressive lossless mesh compression via incremental parametric refinement. Comput Graph Forum 28(5):1301–1310

Markos Zampoglou was born in Thessaloniki, Greece, in 1981. He received a BSc in Applied Informatics from the University of Macedonia, Thessaloniki, in 2004, an MSc in Artificial Intelligence from the University of Edinburgh, in 2005, and a PhD from the University of Macedonia, in 2011. Since 2012, he has been a Postdoctoral Researcher and Research Project Coordinator at the Multimedia Content Laboratory, dept. of Informatics Engineering, Technological Educational Institute of Crete. Dr Zampoglou's previous publications include peer-reviewed papers in books, journals, and conferences on the areas of behavioral robotics, machine vision, unsupervised machine learning, semantic multimedia retrieval and Web 3D technologies.

Kostas Kapetanakis was born in Rethymno, Greece, in 1988. He received a B.Sc. in Applied Informatics and Multimedia in 2011 and a M.Sc. in Informatics Engineering, in 2014, both at the Technological Educational Institute of Crete. Since 2008, he has been a Researcher at the Multimedia Content Laboratory, dept. of Informatics Engineering, T.E.I. of Crete, with main interests in mobile and multimedia web applications. Mr Kapetanakis has authored multiple publications on ubiquitous devices, energy consumption, content adaptation, user experience and multimedia content streaming.

Andreas Stamoulias was born in Athens, Greece, in 1987. He received a B.Sc. in Applied Informatics and Multimedia in 2011 and a M.Sc. in Informatics Engineering, in 2014, both at the Technological Educational Institute of Crete. Since 2011, he has been a Researcher at the Multimedia Content Laboratory, dept. of Informatics Engineering, T.E.I. of Crete. His main interests are 3D graphics and multimedia web applications

Athanasios G. Malamos (M'12) was born in 1969. He received his BSc degree in Physics from the University of Crete (1992) and his PhD from the Technical University of Crete in 2000. From 1997 to 2002 he was a Research Assistant and a researcher in the ICCS National Technical University of Athens. Since 2002 he is with the Technological Educational Institute of Crete at the Department of Informatics Engineering as an Assistant Professor (2002–2006) and as an Associate Professor (2006 until present). He is the coordinator of the Multimedia Content Lab, and has supervised many research projects granted with EU and National research funds. He has served as program committee member and reviewer for several international conferences and workshops. Dr. Malamos is a reviewer for IEEE, Springer as other international journals. He is member of the IEEE Computer Society, ACM SIGGRAPH and the WEB3D consortium. His research interests include multimedia semantics, AI and graphics.

Spyros Panagiotakis was born in Iraklio, Crete, in 1973. He received a BSc in Physics from the University of Athens (1997), an MSc in Electronic Automation in 2001 and a PhD in Communication Networks from the Department of Informatics and Telecommunications of the University of Athens in Greece in 2007. He is currently an Assistant Professor at the Department of Informatics Engineering of the Technological Educational Institution of Crete in Greece. He is author of over than 40 publications. His research interests focus on mobile multimedia technologies, communications and networking, web engineering, mobile applications, pervasive computing, and sensor networks.