DASH for 3D Networked Virtual Environment

Thomas Forgione
Université de Toulouse - IRIT
thomas.forgione@irit.fr

Axel Carlier
Université de Toulouse - IRIT
axel.carlier@enseeiht.fr

Géraldine Morin
Université de Toulouse - IRIT
morin@enseeiht.fr

Wei Tsang Ooi
National Univ. of Singapore
weitsang@nus.edu.sg

Vincent Charvillat
Université de Toulouse - IRIT
charvi@enseeiht.fr

Praveen Kumar Yadav
National Univ. of Singapore
praveen@comp.nus.edu.sg

ABSTRACT

DASH is now a widely deployed standard for streaming video content due to its simplicity, scalability, and ease of deployment. In this paper, we explore the use of DASH for a different type of media content – networked virtual environment (NVE), with different properties and requirements. We organize a polygon soup with textures into a structure that is compatible with DASH MPD (Media Presentation Description), with a minimal set of view-independent metadata for the client to make intelligent decisions about what data to download at which resolution. We also present a DASH-based NVE client that uses a view-dependent and network dependent utility metric to decide what to download, based only on the information in the MPD file. We show that DASH can be used on NVE for 3D content streaming. Our work opens up the possibility of using DASH for highly interactive applications, beyond its current use in video streaming.

CCS CONCEPTS

• **Information systems → Multimedia streaming**; • **Human-centered computing → Virtual reality**;

ACM Reference Format:
Thomas Forgione, Axel Carlier, Géraldine Morin, Wei Tsang Ooi, Vincent Charvillat, and Praveen Kumar Yadav. 2018. DASH for 3D Networked Virtual Environment. In *2018 ACM Multimedia Conference (MM '18), October 22–26, 2018, Seoul, Republic of Korea.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3240508.3240701

1 INTRODUCTION

With the standardization of Extensible 3D (X3D) format from the Web3D Consortium and supports for WebGL in modern browsers, we can now remotely visualize and interact with complex 3D networked virtual environments (NVEs), such as 3D models of cities, buildings, and archaeological sites, with applications ranging from urban planning to tourism. These 3D scenes consist of the geometry data (typically represented as 3D meshes) and textures (represented as images), that can be up to order of hundreds of megabytes or gigabytes in size. There have been a series of research work on how to stream large-scale 3D models over the network for display and interaction with the viewers (e.g., [4, 8, 9]). Yet there is currently no widely adopted standard for a 3D streaming protocol, particularly, one that supports adaptation to dynamic network conditions. We believe this lack of widely accepted standard protocol is hindering a broader deployment of Web-based NVEs.

On the other hand, Dynamic Adaptive Streaming over HTTP (DASH), or MPEG-DASH [16, 17], is now a widely deployed standard for streaming adaptive video content on the Web [10]. DASH is simple and scalable. It uses the existing World Wide Web infrastructure and protocols. Hence it is easy to deploy in any network that supports HTTP. DASH works with a standard HTTP server and HTTP caching proxies without modification. It supports multiple representations of the same video content at different bitrates and lets the clients choose and adapt the quality in response to changing network conditions. As the main adaptation logic lies on the client, the server is stateless and simply acts as a server that responds to HTTP requests. As the Web has demonstrated, such stateless HTTP server can scale to a massive number of users.

Due to the strengths of DASH and a need for a protocol with similar properties for streaming 3D scene, a natural question that arises is whether DASH can be used for adaptive streaming of 3D content for NVE. DASH, however, is designed for video streaming, and adapting it for NVE is non-trivial. Unlike video, where viewing progresses linearly (unless the user seeks), an NVE user can freely navigate within the 3D scene. A typical NVE client would need to determine, and possibly predict, what are the geometry data and textures that fall into the viewing frustum (region defined by its camera position, viewing angle, and look-at point). This should be done purely in the client with only precomputed, view-independent metadata from the server. Thus, the first challenge we face is *what are the view-independent metadata that the server needs to provide along with 3D data to allow the client to make an intelligent view-dependent decision on what to download*? Clearly, the more metadata the server provides along with the 3D data (for DASH, in a Media Presentation Description (MPD) file), the better the decision that the client can make, at the expense of increasing the size of the MPD file. Better decision leads to a more efficient use of the bandwidth, with the aim of being able to download all the visible 3D data content, at the most appropriate resolution, without fetching data that is outside the field of view or is occluded.

For NVE, the content to stream consists of geometry data (vertices, faces, texture coordinates) and texture images. These data are often provided as a polygon soup without semantic information or a scene graph. The textures can be of different resolution levels. The area of the faces can vary significantly as well, implying that the decision as to which triangle to download can have a significant impact on the viewing quality. As such, the second challenge we face is *how to organize the soup of 3D data into DASH adaptation*

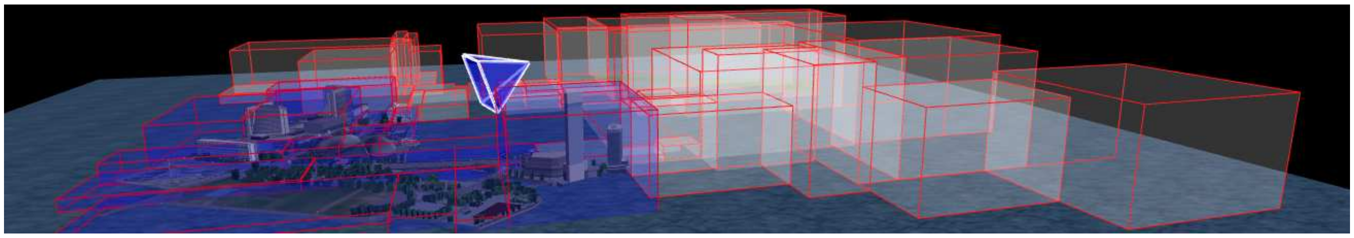

Figure 1: A subdivided 3D scene with a viewport, with regions delimited with red edges. In white, the regions that are outside the field of view of the camera; in blue, the regions inside the field of view of the camera

sets, representation, and segments to facilitate decision making to the DASH client, *without semantic information.*

In DASH video streaming, the bitrate of the segment correlates with the quality of the video. A DASH client can maximize the average bitrate (and hence the size) of the segments downloaded to improve the quality of experience. In NVE streaming, the quality of experience is determined by the rendered scene in the image space (view dependent), whereas the size of the textures determines only its quality in 3D space (view independent). A texture can be mapped onto multiple faces and appear in different regions of the 3D scene. The relative impact of geometry data and textures on the rendering quality also needs to be quantified. Hence, the third challenge we face is *what metadata to provide to support client's rate adaptation decisions, particularly the decisions of what to download (geometry data or textures?), and at which resolution*?

Contributions. Our contributions for this paper are three-fold. First, we propose an organization of a polygon soup into a DASH-compliant format that facilitates selective and adaptive downloading of 3D content by the client. We do not assume the availability of semantic information nor structured scene graphs. Motivated by the need to view a smaller subset of a large 3D scene most of the time during navigation, we partition the 3D space into spatially coherent cells using an axis-aligned K-d tree. Faces that fall into a cell are assigned to a single DASH `adaptation set`. Each cell contains approximately the same number of faces (up to N_a faces). Within each adaptation set, the faces are grouped into N_s segments. To manage the heterogeneity of the faces surface size, we treat faces that are especially large (e.g., the ground or the sky) separately, grouping them into a special adaptation set, allowing the client to download these segments quickly.

The materials are declared in a single file that gives photometric properties and textures of the geometry. This file is described in its own `adaptation set`, in a single `segment`. The textures are placed into separate `adaptation sets`, with different `representations` for different texture resolutions. Separating the geometry and textures into different `adaptation sets` allows the DASH client to trade off between the importance of the geometry and textures accordingly. The latter is not possible using the scheme proposed previously in [20]. Note that in this work, we do not implement multi-resolution geometry (for reasons we will explain in Section 3.3), even though our content preparation would allow it.

Second, we propose precomputation of a small set of metadata that are helpful to the client in making rate adaptation and download decisions. In particular, for each adaptation set of geometry data, we include its bounding box; for adaptation set of texture, we include its average color. For each segment of geometry data, we include (i) the total area of the faces in 3D space; and (ii) the size in bytes of the segment. For each segment of texture data, we include (i) the mean square error of the image, relative to the highest resolution of the texture, and (ii) the size in bytes of the segment. We show that this minimal set of view-independent information is sufficient for complex decision making at the client.

Finally, we present a DASH-based NVE client that is adaptive to network RTT, bandwidth, and navigational pattern. Using only the metadata in the MPD file, the client selectively downloads adaptation sets that intersect with its viewing frustum and prioritizes segments that are likely to contribute more to the viewing quality. We also propose a utility metric that integrates the contribution of geometry data and textures in the image space. This utility metric is, independently, a useful metric that could be useful for other applications that require a comparable objective metric to measure the contribution of geometry data vs. texture data in a 3D scene.

Note that while our proposal is DASH-compliant, the contribution is not limited to DASH. It allows a stateless server to provide only view-independent information and to organize a polygon soup that is friendly to adaptation and selection by the client offline, in a way that any client can derive on his own an efficient, view-dependent, adaptive to network conditions streaming strategy.

Paper Organization. The rest of this paper is structured as follows: Section 2 reviews the related work, and Section 3 describes our proposal for adapting DASH for 3D content. In Section 4 we detail the proposed DASH 3D client. Then, Section 5 presents the experiments and results highlighting the impact of different design parameters for DASH. Finally, we conclude in Section 6.

2 RELATED WORK

2.1 Streaming 3D Content

Accessing and interacting with 3D content over the web have been explored in previous work. Behr et al. structure the 3D content to allow access from a web browser [1]. In a recent standardization effort, the Khronos group has proposed a generic format called glTF [15] to handle all types of 3D content representations: point clouds, meshes, animated model, etc. Although relevant for compression, transmission and in particular streaming, this standard does not yet consider view-dependent streaming.

Some work also focus on compression algorithms; Potenziani et al. describe a system that progressively streams 3D content in the context of cultural heritage [13]. Google Draco [6] is an open-source effort to provide an efficient compression and decompression system that can efficiently run on web browsers. Their work is applied to an attributed geometry modeling a single object, but the compression techniques are not suitable for a partitioned large Networked Virtual Environments (NVE).

The work by Limper et al. on Shape Resource Container [11] focuses more on the streaming aspect, aiming at rendering the model progressively and adapting the content to fit GPU structures to improve the rendering, but independently of the viewport.

The above does not consider view-dependent 3D streaming. View-dependent 3D streaming has been investigated in the work of Forgione et al. [4], where the authors use frustum and backface culling on the server to determine a set of polygons to be streamed by the client. Bookmarks provide a way to ease user navigation and improve streaming efficiency. The drawback of this method is that it does not scale well with the number of users due to the computation load on the server. Cheng and Ooi [2] proposed a view-dependent progressive mesh streaming system that is receiver-driven. Their technique leads to a stateless server and is scalable. Their work, however, focuses on a single progressive mesh structure and uses a customized application-level protocol.

Balancing between streaming of geometry and texture data are considered by Tian et al. [18], Guo et al. [7], and Yang et. al. [19]. All three work considered a single, manifold textured mesh model with progressive meshes. Their approach is to combine the distortion caused by having lower resolution meshes and textures into a single view independent metric. Our work focuses on a whole, heterogeneous 3D scene where separate object coding is not applicable, thus their proposed metric is not a good fit.

Zampoglou et al. are the first to propose DASH to stream 3D content [20]. In their work, the authors describe a system that allows users to access 3D content at multiple resolutions. They organize the content, following DASH terminology, into `periods`, `adaptation sets`, `representations`. Their first `adaptation set` codes the tree structure of an X3DOM. Each further `adaptation set` contains both geometry and texture information and is available at different resolutions defined in a corresponding `representation`. To avoid requests that would take too long and thus introduce latency, the `representations` are split into `segments`. The authors discuss the optimal number of polygons that should be stored in a single `segment`. On the one hand, using `segments` containing very few faces will induce many HTTP requests from the client, and will lead to poor streaming efficiency. On the other hand, if `segments` contain too many faces, the time to load the `segment` will be long and the system loses adaptability. The authors conclude that using `segments` of 5000 faces is a good compromise. This approach works well for several objects, but does not handle view-dependent streaming, which is desirable in the use case of large NVEs.

2.2 SRD-DASH

While typical DASH for video is designed for video playback with a single dimension of freedom along the time axis, there are recent work that have extended video playback to allow higher dimension of freedom in interacting with the video, including panning (e.g., 360-degree videos) and zooming (e.g., zoomable video [14]). These schemes partition each video frame into tiles which can be decoded independently by the client. In this context, Niamut et al. proposed a modified version of traditional Media Presentation Description (MPD) files using Spatial Representation Description (SRD) [12]. Such partial content retrieval and rendering help in zooming and navigating through the content [3].

DASH for NVE shares many similarity with SRD-DASH: a user may view only a subset of the content. SRD-DASH puts each tile into a different adaptation set, and the client can only request for the adaptation sets (the tiles) that fall within the current view. Similarly, in our proposal, we partition 3D geometry and textures into cells ("3D tiles") and each cell forms an adaptation set. Unlike SRD-DASH for video however, it is not straightforward to assign a 3D face to a cell, since a large face (e.g., the sky or the ground) may intersect with multiple cells.

3 MAPPING NVE INTO DASH

In this section, we describe how we preprocess and store the 3D data of the NVE, consisting of a polygon soup, textures, and material information into a DASH-compliant Media Presentation Description (MPD) file. In our work, we use the `obj` file format for the polygons, `png` for textures, and `mtl` format for material information. The process, however, applies to other formats as well.

3.1 The MPD File

In DASH, the information about content storage and characteristics, such as location, resolution, or size, are extracted from an MPD file by the client. The client relies only on these information to decide which chunk to request and at which quality level.

The MPD file is an XML file that is organized into different sections hierarchically. The `period` element is a top-level element, which for the case of video, indicates the start time and length of a video chapter. This element does not apply to NVE, and we use a single `period` for the whole scene, as the scene is static.

Each `period` element contains one or more adaptation sets, which describe the alternate versions, formats, and types of media. We utilize adaptation sets to organize a 3D scene's material, geometry, and texture.

3.2 Adaptation Sets

When the user navigates freely within an NVE, the frustum at given time almost always contains a limited part of the 3D scene. Similar to how DASH for video streaming partitions a video clip into temporal chunks, we segment the polygons into spatial chunks, such that the DASH client can request only the relevant chunks.

Geometry Management. We use a space partitioning tree to organize the faces into cells. A face belongs to a cell if its barycenter falls inside the corresponding bounding box. Each cell corresponds to an adaptation set. Thus, geometry information is spread on adaptation sets based on spatial coherence, allowing the client to download the relevant faces selectively. A cell is relevant if it intersects the frustum of the client's current viewpoint. Figure 1 shows the relevant cells in blue. As our 3D content, a virtual environment, is biased to spread the most along the horizontal plane, we alternate between splitting between the two horizontal directions.

We create a separate adaptation set for large faces (e.g., the sky or ground) because they are essential to the 3D model and do not fit into cells. We consider a face to be large if its area in 3D is more than $a + 3\sigma$, where a and σ are the average and the standard deviation of 3D area of faces respectively. In our example, it selects the 5 largest faces that represent 15% of the total face area. We thus obtain a decomposition of the NVE into adaptation sets that partitions the geometry of the scene into a small adaptation set containing the larger faces of the model, and smaller adaptation sets containing the remaining faces.

We store the spatial location of each adaptation set, characterized by the coordinates of its bounding box, in the MPD file as the supplementary property of the adaptation set in the form of "x_{min}, *width*, y_{min}, *height*, z_{min}, *depth*" (as shown in Listing 1). This information is used by the client to implement a view-dependent streaming (Section 4).

Texture Management. As with geometry data, we handle textures using adaptation sets but separate from geometry. Each texture file is contained in a different adaptation set, with multiple representations providing different image resolutions (see Section 3.3). We add an attribute to each adaptation set that contains texture, describing the average color of the texture. The client can use this attribute to render a face for which the corresponding texture has not been loaded yet, so that most objects appear, at least, with a uniform natural color (see Figure 2).

Material Management. The material `.mtl` file is a text file that describes all materials used in the `.obj` files for the entire 3D model. A material has a name, properties such as specular parameters, and, most importantly, a path to a texture file. The `.mtl` file maps each face of the `.obj` to a material. As the `.mtl` file is a different type of media than geometry and texture, we define a particular adaptation set for this file, with a single representation.

3.3 Representations

Each adaptation set can contain one or more representations of the geometry or texture data, at different levels of detail (e.g., a different number of faces). For geometry, the resolution (i.e., 3D areas of faces) is heterogeneous, thus applying a sensible multi-resolution representation is cumbersome: the 3D area of faces varies from 0.01 to more than $10K$, disregarding the outliers. For textured scenes, it is common to have such heterogeneous geometry size since information can be stored either in geometry or texture. Thus, handling the streaming compromise between geometry and texture is more adaptive than handling separately multi-resolution geometry. Moreover, as our faces are partitioned into independent cells, multi-resolution would cause difficult stitching issues such as topological gaps between the cells.

For an adaptation set containing texture, each representation contains a single segment where the image file is stored at the chosen resolution. In our example, from the full-size image, we generate successive resolutions by dividing both height and width by 2, stopping when the image size is less or equal to 64×64. Figure 2 illustrates the use of the textures against the rendering using a single, average color per face.

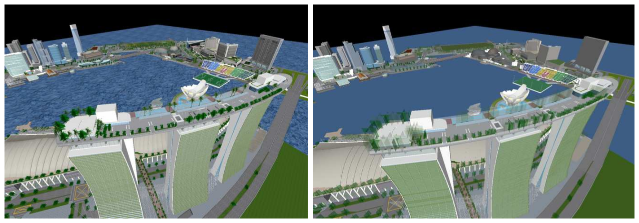
Figure 2: Rendering of the model with full resolution texture (left), and faces with average default color (right).

```
<AdaptationSet>
    <SupplementalProperty value="-8834.11230,2201.58853,
        -0.16950, 174.81540,-1344.47740,4767.83367" />
 <BaseURL>as1/</BaseURL>
 <Representation>
  <BaseURL>repr1/</BaseURL>
  <SegmentList>
   <SegmentURL area="2540342.3" size="120K" media="s0.obj" />
   <SegmentURL area="1124.4" size="162K" media="s1.obj" />
   <SegmentURL area="412.6" size="173K" media="s2.obj" />
   <SegmentURL area="270.3" size="147K" media="s3.obj" />
  </SegmentList>
 </Representation>
</AdaptationSet>

<AdaptationSet area="198632.73912" average="178,176,173"
      mimeType="image/png">
    <BaseURL>textures/MFLOOR07.PNG/</BaseURL>
    <Representation>
        <BaseURL>64x64/</BaseURL>
        <SegmentList>
            <SegmentURL size="7K" mse="57.6" media="t.png" />
        </SegmentList>
    </Representation>
    <Representation>
        <BaseURL>128x128/</BaseURL>
        <SegmentList>
            <SegmentURL size="27K" mse="0.0" media="t.png" />
        </SegmentList>
    </Representation>
</AdaptationSet>
```

Listing 1: MPD description of a geometry adaptation set, and a texture adaptation set.

3.4 Segments

To allow random access to the content within an adaptation set storing geometry data, we group the faces into segments. Each segment is then stored as a `.obj` file that can be individually requested by the client. For geometry, we partition the faces in an adaptation set into sets of N_s faces, by first sorting the faces by their area in 3D space in descending order, and then place each successive N_s faces into a segment. Thus, the first segment contains the biggest faces and the last one the smallest. In addition to the selected faces, a segment stores all face vertices and attributes so that each segment is independent. For textures, each representation contains a single segment.

4 DASH 3D CLIENT

In this section, we specify a DASH NVE client that exploits the preparation of the 3D content in an NVE for streaming.

The generated MPD file describes the content organization so that the client gets all the necessary information to make educated decisions and query the 3D content it needs according to the available resources and current viewpoint. A camera path generated by a particular user is a set of viewpoint $v(t_i)$ indexed by a continuous time interval $t_i \in [t_1, t_{end}]$.

The DASH client first downloads the MPD file to get the material (.mtl) file containing information about all the geometry and textures available for the entire 3D model. At time instance t_i, the DASH client decides to download the appropriate segments containing the geometry and the texture to generate the viewpoint $v(t_{i+1})$ for the time instance t_{i+1}.

Starting from t_1, the camera continuously follows a camera path $C = \{v(t_i), t_i \in [t_1, t_{end}]\}$, along which downloading opportunities are strategically exploited to sequentially query the most useful segments.

4.1 Segment Utility

Unlike video streaming, where the bitrate of each segment correlates with the quality of the video received, for 3D content, the size (in bytes) of the content does not necessarily correlate well to its contribution to visual quality. A large polygon with huge visual impact takes the same number of bytes as a tiny polygon. Further, the visual impact is *view dependent* – a large object that is far away or out of view does not contribute to the visual quality as much as a smaller object that is closer to the user. As such, it is important for a DASH-based NVE client to estimate the usefulness of a given segment to download, so that it can make good decisions about what to download. We call this usefulness the *utility* of the segment.

The utility is a function of a segment, either geometry or texture, and the current viewpoint (camera location, view angle, and look-at point), and is therefore dynamically computed online by the client from parameters in the MPD file.

Offline parameters. Let us detail first, all parameters available from the offline/static preparation of the 3D NVE. These parameters are stored in the MPD file. First, for each geometry segment s^G there is a predetermined 3D area $\mathcal{A}_{3D}(s^G)$, equal to the sum of all triangle areas in this segment (in 3D); it is computed as the segments are created. Note that the texture segments will have similar information, but computed at *navigation time t_i*. The second information stored in the MPD for all segments, geometry, and texture, is the size of the segment (in kB). Indeed, geometry segments have close to a similar number of faces; their size is almost uniform. For texture segments, the size is usually much smaller than the geometry segments but also varies a lot, as between two successive resolutions the number of pixels is divided by 4.

Finally, for each texture segment s^T, the MPD stores the *MSE* (mean square error) of the image and resolution, relative to the highest resolution (by default, triangles are filled with its average color). Offline parameters are stored in the MPD as shown in Listing 1.

Online parameters. In addition to the offline parameters stored in the MPD file for each segment, view-dependent parameters are computed at navigation time. First, a measure of 3D area is computed for texture segments. As a texture maps on a set of triangles, we account for the area in 3D of all these triangles. We could consider such an offline measure (attached to the adaptation set containing the texture), but we prefer to only account for the triangles that have been already downloaded by the client. We call the set of triangles colored by a texture $T : \Delta(s^T) = \Delta(T)$ (depending only on T and equal for any representation/segment s^T in this texture adaptation set). At each time t_i, a subset of $\Delta(T)$ has been downloaded; we denote it $\Delta(T, t_i)$.

Moreover, each geometry segment belongs to a geometry adaptation set AS^G whose bounding box coordinates are stored in the MPD. Given the coordinates of the bounding box $\mathcal{BB}(AS^G)$ and the viewpoint $v(t_i)$ at time t_i, the client computes the distance $\mathcal{D}(v(t_i), AS^G)$ of the bounding box $\mathcal{BB}(AS^G)$ as the distance from the center of $\mathcal{BB}(AS^G)$ to the principal point of the camera, given in $v(t_i)$.

Utility for geometry segments. We now have all parameters to derive a utility measure of a geometry segment. Utility for texture segments will follow from the geometric utility.

The utility of a geometric segment s^G for a viewpoint $v(t_i)$ is:

$$\mathcal{U}\Big(s^G, v(t_i)\Big) = \frac{\mathcal{A}_{3D}(s^G)}{\mathcal{D}(v(t_i), AS^G)^2}$$

where AS^G is the adaptation set containing s^G.

Basically, the utility of a segment is proportional to the area that its faces cover, and inversely proportional to the square of the distance between the camera and the center of the bounding box of the adaptation set containing the segment. That way, we favor segments with big faces that are close to the camera.

Utility for texture segments. For a texture T stored in a segment s^T, the triangles in $\Delta(T)$ are stored in arbitrary geometry segments, that is, they do not have spatial coherence. Thus, for each k^{th} downloaded geometry segment s_k^G, and total downloaded segment K at time t_i, we collect the triangles of $\Delta(T, t_i)$ in s_k^G, and compute the ratio of $\mathcal{A}_{3D}(s_k^G)$ covered by these triangles. So, we define the utility:

$$\mathcal{U}\Big(s^T, v(t_i)\Big) = psnr(s^T) \sum_{k \in K} \frac{\mathcal{A}_{3D}(s_k^G \cap \Delta(T, t_i))}{\mathcal{A}_{3D}(s_k^G)} \mathcal{U}\Big(s_k^G, v(t_i)\Big)$$

where we sum over all geometry segments received before time t_i that intersect $\Delta(T, t_i)$ and such that the adaptation set it belongs to is in the frustum. This formula defines the utility of a texture segment by computing the linear combination of the utility of the geometry segments that use this texture, weighted by the proportion of area covered by the texture in the segment. We compute the PSNR by using the MSE in the MPD and denote it $psnr(s^T)$. We do this to acknowledge the fact that a texture at a greater resolution will have a higher utility than a lower resolution texture. The equivalent term for geometry is 1 (and does not appear).

Having defined a utility on both geometry and texture segments, the client uses it next for its streaming strategy.

4.2 DASH Adaptation Logic

Along the camera path $C = \{v(t_i)\}$, viewpoints are indexed by a continuous time interval $t_i \in [t_1, t_{end}]$. Contrastingly, the DASH adaptation logic proceeds sequentially along a discrete time line.

The first request (`HTTP request`) made by the DASH client at time t_1 selects the most useful segment s_1^* to download and will be followed by subsequent decisions at $t_2, t_3, \ldots$. While selecting s_i^*, the i-th best segment to request, the adaptation logic compromises between geometry, texture, and the available `representations` given the current bandwidth, camera dynamics, and the previously described utility scores. The difference between t_{i+1} and t_i is the s_i^* delivery delay. It varies with the segment size and network conditions. Algorithm 1 details how our DASH client makes decisions.

```
input  : Current index $i$, time $t_i$, viewpoint $v(t_i)$, buffer of
         already downloaded segments $\mathcal{B}_i$, MPD
output : Next segment $s_i^*$ to request, updated buffer $\mathcal{B}_{i+1}$
- Estimate the bandwidth $\widehat{BW_i}$ and RTT $\widehat{\tau_i}$ ;
- Among all segments that are not already downloaded
  $s \in \mathcal{S} \backslash \mathcal{B}_i$, keep the ones inside the upcoming viewing
  frustums $\mathcal{FC} = \mathbb{FC}(\widehat{v}(t_i)), t \in [t_i, t_i + \chi]$ thanks to a
  viewpoint predictor $t_i \rightarrow \hat{v}(t_i)$, a temporal horizon $\chi$ and a
  frustum culling operator $\mathbb{FC}$ ;
- Optimize a criterion $\Omega$ based on $\mathcal{U}$ values and well chosen
  viewpoint(s) $v(t_i)$ to select the next segment to query
      $s_i^* = \underset{s \in \mathcal{S} \backslash \mathcal{B}_i \cap \mathcal{FC}}{\operatorname{argmax}} \Omega_{\theta_i}\Big(\mathcal{U}(s, v(t_i)\Big)$
  given parameters $\theta_i$ that gathers both online parameters
  $(i, t_i, v(t_i), \widehat{BW_i}, \widehat{\tau_i}, \mathcal{B}_i)$ and offline metadata;
- Update the buffer $\mathcal{B}_{i+1}$ for the next decision: $s_i^*$ and lowest
  representations of $s_i^*$ are considered downloaded;
- return segment $s_i^*$, buffer $\mathcal{B}_{i+1}$;
```

Algorithm 1: Algorithm to identify the next segment to query

The most naive way to sequentially optimize the $\mathcal{U}$ is to limit the decision-making to the current viewpoint $v(t_i)$. In that case, the best segment s to request would be the one maximizing $\mathcal{U}(s, v(t_i))$ to simply make a better rendering from the current viewpoint $v(t_i)$. Due to transmission delay however, this segment will be only delivered at time $t_{i+1} = t_{i+1}(s)$ depending on the segment size and network conditions:

$$t_{i+1}(s) = t_i + \frac{\texttt{size}(s)}{\widehat{BW_i}} + \widehat{\tau_i}$$

In consequence, the most useful segment from $v(t_i)$ at decision time t_i might be less useful at delivery time from $v(t_{i+1})$.

A better solution is to download a segment that is expected to be the most useful in the future. With a temporal horizon χ, we can optimize the cumulated $\mathcal{U}$ over $[t_{i+1}(s), t_i + \chi]$:

$$s_i^* = \underset{s \in \mathcal{S} \backslash \mathcal{B}_i \cap \mathcal{FC}}{\operatorname{argmax}} \int_{t_{i+1}(s)}^{t_i+\chi} \mathcal{U}(s, \hat{v}(t_i))dt \qquad (1)$$

In our experiments, we typically use $\chi = 2s$ and estimate the (1) integral by a Riemann sum where the $[t_{i+1}(s), t_i + \chi]$ interval is divided in 4 subintervals of equal size. For each subinterval extremity, an order 1 predictor $\hat{v}(t_i)$ linearly estimates the viewpoint based on $v(t_i)$ and speed estimation (discrete derivative at t_i).

Files	Size	Files	Size
3DS Max	55 MB	OBJ file	62 MB
Textures (high res)	167 MB	MTL file	0.27Mb
Textures (low res)	11 MB		

Table 1: Sizes of the different files of the model

We also tested an alternative greedy heuristic selecting the segment that optimizes an utility variation during downloading (between t_i and t_{i+1}): ty

$$s_i^{\texttt{GREEDY}} = \underset{s \in \mathcal{S} \backslash \mathcal{B}_i \cap \mathcal{FC}}{\operatorname{argmax}} \frac{\mathcal{U}\Big(s, \hat{v}(t_{i+1}(s))\Big)}{t_{i+1}(s) - t_i} \qquad (2)$$

5 EVALUATION

We now describe our setup and the data we use in our experiments. We present an evaluation of our system and a comparison of the impact of the design choices we introduced in the previous sections.

5.1 Experimental Setup

Model. We use a city model of the Marina Bay area in Singapore in our experiments. The model came in 3DS Max format and has been converted into Wavefront OBJ format before the processing described in Section 3. The converted model has 387,551 vertices and 552,118 faces. Table 1 gives some general information about the model. We partition the geometry into a k-*d* tree until the leafs have less than 10000 faces, which gives us 64 adaptation sets, plus one containing the large faces.

User Navigations. To evaluate our system, we collected realistic user navigation traces that we can replay in our experiments. We presented six users with a web interface, on which the model was loaded progressively as the user could interact with it. The available interactions were inspired by traditional first-person interactions in video games, i.e., W, A, S, and D keys to translate the camera, and mouse to rotate the camera. We asked users to browse and explore the scene until they felt they had visited all important regions. We then asked them to produce camera navigation paths that would best present the 3D scene to a user that would discover it. To record a path, the users first place their camera to their preferred starting point, then click on a button to start recording. Every 100ms, the position, viewing angle of the camera and look-at point are saved into an array that will then be exported into JSON format. The recorded camera trace allows us to replay each camera path to perform our simulations and evaluate our system. We collected 13 camera paths this way.

Network Setup. We tested our implementation under three network bandwidth of 2.5 Mbps, 5 Mbps, and 10 Mbps with an RTT of 38 ms, following the settings from DASH-IF [5]. The values are kept constant during the entire client session to analyze the difference in magnitude of performance by increasing the bandwidth.

In our experiments, we set up a virtual camera that moves along a navigation path, and our access engine downloads segments in real time according to Algorithm 1. We log in a JSON file the time

Parameters	Values
Content preparation	Octree, k-d tree
Utility	Offline, Online, Proposed
Streaming policy	Greedy, Proposed
Grouping of Segments	Sorted based on area, Unsorted
Bandwidth	2.5 Mbps, 5 Mbps, 10 Mbps

Table 2: Different parameters in our experiments

when a segment is requested and when it is received. By doing so, we avoid wasting time and resources to evaluate our system while downloading segments and store all the information necessary to plot the figures introduced in the subsequent sections.

Hardware and Software. The experiments were run on an Acer Aspire V3 with an Intel Core i7-3632QM processor and an NVIDIA GeForce GT 740M graphics card. The DASH client is written in Rust[1], using Glium[2] for rendering, and reqwest[3] to load the segments.

Metrics. To objectively evaluate the quality of the resulting rendering, we use PSNR. The scene as rendered offline using the same camera path with all the geometry and texture data available is used as ground truth. Note that a pixel error can occur in our case only in two situations: (i) when a face is missing, in which case the color of the background object is shown, and (ii) when a texture is either missing or downsampled. We do not have pixel error due to compression.

Experiments. We present experiments to validate our implementation choices at every step of our system. We replay the user-generated camera paths with various bandwidth conditions while varying key components of our system.

Table 2 sums up all the components we varied in our experiments. We compare the impact of two space-partitioning trees, a k-d tree and an Octree, on content preparation. We also try several utility metrics for geometry segments: an offline one, which assigns to each geometry segment s^G the cumulated 3D area of its belonging faces $\mathcal{A}_{3D}(s^G)$; an online one, which assigns to each geometry segment the inverse of its distance to the camera position; and finally our proposed method, as described in Section 4.1 ($\mathcal{A}_{3D}(s^G)/\mathcal{D}(v(t_i), AS^G)^2$). We consider two streaming policies to be applied by the client, proposed in Section 4. The greedy strategy determines, at each decision time, the segment that maximizes its predicted utility at arrival divided by its predicted delivery delay, which corresponds to equation (2). The second streaming policy that we run is the one we proposed in equation (1). We have also analyzed the effect of grouping the faces in geometry segments of an adaptation set based on their 3D area. Finally, we try several bandwidth parameters to study how our system can adapt to varying network conditions.

[1]https://www.rust-lang.org/
[2]https://github.com/glium/glium
[3]https://github.com/seanmonstar/reqwest/

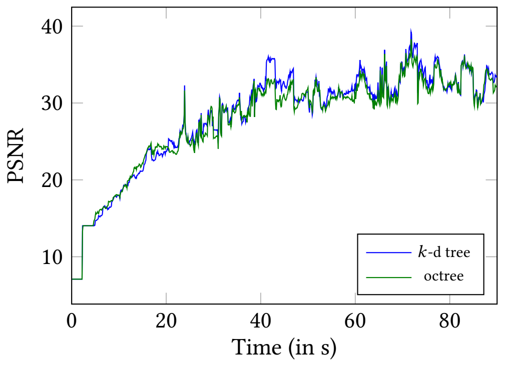

Figure 3: Impact of the space-partitioning tree on the rendering quality with a 5Mbps bandwidth.

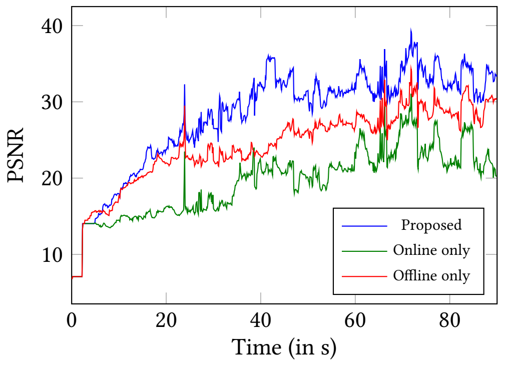

Figure 4: Impact of the segment utility metric on the rendering qualit with a 5Mbps bandwidth.

5.2 Experimental Results

Figure 3 shows how the space partition can affect the rendering quality. We use our proposed utility metrics (see Section 4.1) and streaming policy from Equation (1), on content divided into adaptation sets obtained either using a k-d tree or an Octree and run experiments on all camera paths at 5 Mbps. The Octree partitions content into non-homogeneous adaptation sets; as a result, some adaptation sets may contain smaller segments, which contain both important (large) and non-important polygons. For the k-d tree, we create cells containing the same number of faces N_a (here, we take $N_a = 10k$). Figure 3 shows that the system seems to be slightly less efficient with an Octree than with a k-d tree based partition, but this result is not significant. For the remaining experiments, partitioning is based on a k-d tree.

Figure 4 displays how a utility metric should take advantage of both offline and online features. The experiments consider k-d tree cell for adaptation sets and the proposed streaming policy, on all camera paths. We observe that a purely offline utility metric leads to poor PSNR results. An online-only utility improves the results, as it takes the user viewing frustum into consideration, but still, the proposed utility (in Section 4.1) performs better.

Figure 5 shows the effect of grouping the segments in an adaptation set based on their area in 3D. Clearly, the PSNR significantly

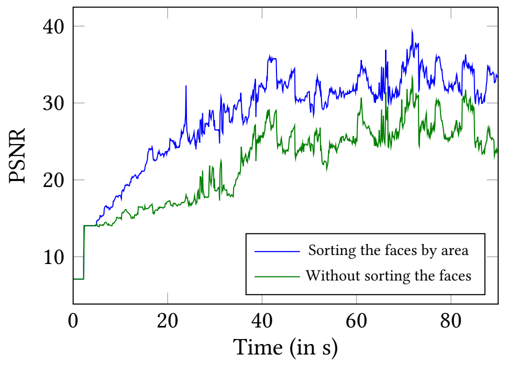

Figure 5: Impact of creating the segments of an adaptation set based on decreasing 3D area of faces with a 5Mbps bandwidth.

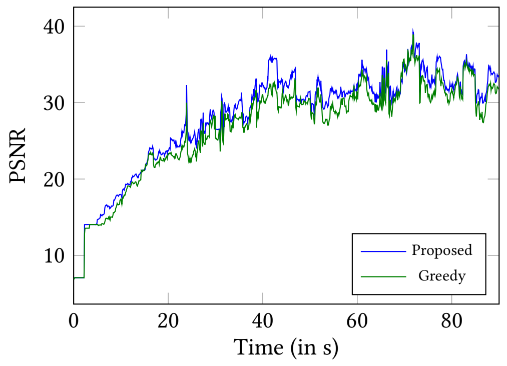

Figure 6: Impact of the streaming policy (greedy vs. proposed) with a 5 Mbps bandwidth.

	First 30 Sec			Overall		
BW (in Mbps)	2.5	5	10	2.5	5	10
Greedy	14.4	19.4	22.1	19.8	26.9	29.7
Proposed	16.3	20.4	23.2	23.8	28.2	31.1

Table 3: Average PSNR, Greedy vs. Proposed

Resolutions	2.5 Mbps	5 Mbps	10 Mbps
1	5.7% vs 1.4%	6.3% vs 1.4%	6.17% vs 1.4%
2	10.9% vs 8.6%	13.3% vs 7.8%	14.0% vs 8.3%
3	15.3% vs 28.6%	20.1% vs 24.3%	20.9% vs 22.5%
4	14.6% vs 18.4%	14.4% vs 25.2%	14.2% vs 24.1%
5	11.4% vs 0.3%	11.1% vs 5.9%	11.5% vs 13.9%

Table 4: Percentages of downloaded bytes for textures from each resolution, for the greedy streaming policy (left) and for our proposed scheme (right)

improves when the 3D area of faces is considered for creating the segments. Since all segments are of the same size, sorting the faces by area before grouping them into segments leads to a skew distribution of how useful the segments are. This skewness means that the decision that the client makes (to download those with the largest utility first) can make a bigger difference in the quality.

We also compared the greedy vs. proposed streaming policy (as shown in Figure 6 for limited bandwidth (5 Mbps). The proposed scheme outperforms the greedy during the first 30s and does a better job overall. Table 3 shows the average PSNR for the proposed method and the greedy method for different downloading bandwidth. In the first 30 sec, since there are relatively few 3D contents downloaded, making a better decision at what to download matters more: we observe during that time that the proposed method leads to 1 - 1.9 dB better in quality terms of PSNR compared to Greedy.

Table 4 shows the distribution of texture resolutions that are downloaded by greedy and our Proposed scheme, at different bandwidths. Resolution 5 is the highest and 1 is the lowest. The table clearly shows a weakness of the greedy policy: as the bandwidth increases, the distribution of downloaded textures resolution stays more or less the same. In contrast, our proposed streaming policy adapts to an increasing bandwidth by downloading higher resolution textures (13.9% at 10 Mbps, vs. 0.3% at 2.5 Mbps). In fact, an interesting feature of our proposed streaming policy is that it adapts the geometry-texture compromise to the bandwidth. The textures represent 57.3% of the total amount of downloaded bytes at 2.5 Mbps, and 70.2% at 10 Mbps. In other words, our system tends to favor geometry segments when the bandwidth is low, and favor texture segments when the bandwidth increases.

6 CONCLUSION AND FUTURE WORK

Our work in this paper started with the question: can DASH be used for NVE? The answer is *yes*. In answering this question, we contributed by showing how to organize a polygon soup and its textures into a DASH-compliant format that (i) includes a minimal amount of metadata that is useful for the client, (ii) organizes the data to allow the client to get the most useful content first. We further show that these metadata that is precomputed offline is sufficient to design and build a DASH client that is adaptive – it can selectively download segments within its view, make intelligent decisions about what to download, balancing between geometry and texture while being adaptive to network bandwidth. We believe our proposed DASH for NVE is flexible enough for the community to start the simplicity and ease of deployment of DASH for NVE and to start investigating different streaming strategies to improve the quality of experience of NVE users.

In the future, we will try to take into account any semantic information that would be available to avoid splitting buildings in different adaptation sets for example. With semantic information, we will also try to embed multi-resolution geometry support, and define the utility metrics accordingly.

REFERENCES

[1] J. Behr, Y. Jung, J. Keil, T. Drevensek, M. Zoellner, P. Eschler, and D. Fellner. 2010. A scalable architecture for the HTML5/X3D integration model X3DOM. In *Proceedings of the 15th International Conference on Web 3D Technology - Web3D '10*. ACM, Los Angeles, California, 185–194. https://doi.org/10.1145/1836049.1836077

[2] Wei Cheng and Wei Tsang Ooi. 2008. Receiver-driven View-dependent Streaming of Progressive Mesh. In *Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV '08)*. ACM, Braunschweig, Germany, 9–14. https://doi.org/10.1145/1496046.1496049

[3] Lucia D'Acunto, Jorrit van den Berg, Emmanuel Thomas, and Omar Niamut. 2016. Using MPEG DASH SRD for zoomable and navigable video. In *Proceedings of the 7th International Conference on Multimedia Systems (MMSys '16)*. ACM, Klagenfurt, Austria, 34–37.

[4] Thomas Forgione, Axel Carlier, Géraldine Morin, Wei Tsang Ooi, and Vincent Charvillat. 2016. Impact of 3D Bookmarks on Navigation and Streaming in a Networked Virtual Environment. In *Proceedings of the 7th International Conference on Multimedia Systems (MMSys '16)*. ACM, Klagenfurt, Austria, Article 9, 10 pages. https://doi.org/10.1145/2910017.2910607

[5] DASH Industry Forum. 2014. Guidelines for implementation: DASH-AVC/264 test cases and vectors. http://dashif.org/guidelines/.

[6] Google. 2017. Draco. https://github.com/google/draco.

[7] Jinjiang Guo, Vincent Vidal, Irene Cheng, Anup Basu, Atilla Baskurt, and Guillaume Lavoue. 2017. Subjective and objective visual quality assessment of textured 3D meshes. *ACM Transactions on Applied Perception (TAP)* 14, 2 (2017), 11.

[8] Shun-Yun Hu, Ting-Hao Huang, Shao-Chen Chang, Wei-Lun Sung, Jehn-Ruey Jiang, and Bing-Yu Chen. 2008. FLoD: A framework for peer-to-peer 3D streaming. In *Proceedings of the 27th IEEE Conference on Computer Communications (INFOCOM '08)*. IEEE, IEEE, Phoenix, AZ, USA, 1373–1381.

[9] Yonghao Hu, Zhaohui Chen, Xiaojun Liu, Fei Huang, and Jinyuan Jia. 2017. WebTorrent based fine-grained P2P transmission of large-scale WebVR indoor scenes. In *Proceedings of the 22nd International Conference on 3D Web Technology*. ACM, 7.

[10] ISO/IEC 23009-1:2014 2014. *Information technology – Dynamic adaptive streaming over HTTP (DASH) – Part 1: Media presentation description and segment formats*. Standard.

[11] Max Limper, Maik Thöner, Johannes Behr, and Dieter W. Fellner. 2014. SRC - a Streamable Format for Generalized Web-based 3D Data Transmission. In *Proceedings of the 19th International ACM Conference on 3D Web Technologies (Web3D '14)*. ACM, Vancouver, British Columbia, Canada, 35–43. https://doi.org/10.1145/2628588.2628589

[12] Omar A. Niamut, Emmanuel Thomas, Lucia D'Acunto, Cyril Concolato, Franck Denoual, and Seong Yong Lim. 2016. MPEG DASH SRD: Spatial Relationship Description. In *Proceedings of the 7th International Conference on Multimedia Systems (MMSys '16)*. ACM, Klagenfurt, Austria, Article 5, 8 pages. https://doi.org/10.1145/2910017.2910606

[13] Marco Potenziani, Marco Callieri, Matteo Dellepiane, Massimiliano Corsini, Federico Ponchio, and Roberto Scopigno. 2015. 3DHOP: 3D heritage online presenter. *Computers & Graphics* 52 (2015), 129–141.

[14] Ngo Quang Minh Khiem, Guntur Ravindra, Axel Carlier, and Wei Tsang Ooi. 2010. Supporting Zoomable Video Streams with Dynamic Region-of-interest Cropping. In *Proceedings of the First Annual ACM SIGMM Conference on Multimedia Systems (MMSys '10)*. ACM, New York, NY, USA, 259–270. https://doi.org/10.1145/1730836.1730868

[15] Fabrice Robinet and Cesium Patrick Cozzi. 2013. glTF - Runtime asset format for WebGL, OpenGL ES, and OpenGL.

[16] Iraj Sodagar. 2011. The MPEG-DASH Standard for Multimedia Streaming Over the Internet. *IEEE Multimedia* 18, 4 (apr 2011), 62–67. https://doi.org/10.1109/MMUL.2011.71

[17] Thomas Stockhammer. 2011. Dynamic Adaptive Streaming over HTTP –: Standards and Design Principles. In *Proceedings of the Second Annual ACM Conference on Multimedia Systems (MMSys '11)*. ACM, San Jose, CA, USA, 133–144. https://doi.org/10.1145/1943552.1943572

[18] Dihong Tian and Ghassan AlRegib. 2008. Batex3: Bit allocation for progressive transmission of textured 3-d models. *IEEE Transactions on Circuits and Systems for Video Technology* 18, 1 (2008), 23–35.

[19] Sheng Yang, Chao-Hua Lee, and C.-C. Jay Kuo. 2004. Optimized Mesh and Texture Multiplexing for Progressive Textured Model Transmission. In *Proceedings of the 12th Annual ACM International Conference on Multimedia (MULTIMEDIA '04)*. ACM, New York, NY, USA, 676–683. https://doi.org/10.1145/1027527.1027683

[20] Markos Zampoglou, Kostas Kapetanakis, Andreas Stamoulias, Athanasios G. Malamos, and Spyros Panagiotakis. 2016. Adaptive streaming of complex Web 3D scenes based on the MPEG-DASH standard. *Multimedia Tools and Applications* (Dec 2016), 1–24. https://doi.org/10.1007/s11042-016-4255-8