DASH pour la Transmission d'Environnements Virtuels 3D

T. Forgione[1] A. Carlier[1] G. Morin[1] W. T. Ooi[2] V. Charvillat[1] P. K. Yadav[2]

[1] Institut de Recherche en Informatique de Toulouse - INP–IRIT
[2] National University of Singapore

Résumé

DASH est un standard largement utilisé pour la transmission de contenu vidéo grâce à sa simplicité, sa capacité à monter en charge, et sa facilité de déploiement. Dans ce papier, nous explorons l'utilisation de DASH pour un type de contenu multimédia différent – les environnements virtuels distribués (NVE), avec des propriétés et des contraintes différentes. Nous organisons une soupe de polygones texturés dans une structure compatible avec DASH MPD (Media Presentation Description), en ajoutant un ensemble de métadonnées destinées à aider le client à prendre des décisions intelligentes quant aux données à télécharger, ainsi qu'à leur résolution. Nous présentons également un client permettant la navigation dans un NVE basé sur DASH qui utilise des métriques d'utilité dépendant du point de vue et du réseau afin de déterminer ce qui doit être téléchargé, en exploitant seulement l'information fournie dans le MPD.

Mots Clef

Multimédia, Streaming, 3D, Environnements Virtuels.

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.

Keywords

Multimedia, Streaming, 3D, Virtual Environments.

1 Introduction

Avec la standardisation du format *Extensible 3D* (X3D) du Web3D Consortium et le support de WebGL par les navigateurs modernes, il est désormais possible de visualiser et d'interagir avec des environnements virtuels distribués (*Networked Virtual Environments, NVE*), tels que des modèles 3D de villes, de bâtiments, de sites archéologiques, avec des applications allant de l'aménagement urbain au tourisme. Ces scènes 3D consistent en des données de géométrie (typiquement représentées sous forme de maillage 3D) et de texture (représentées sous forme d'images), qui peuvent représenter jusqu'à plusieurs giga-octets.

De nombreux travaux ont été effectués pour transmettre des modèles 3D de grande échelle à travers le réseau pour, afin de les rendre disponibles à la visualisation et aux interactions d'utilisateurs (par exemple, [8, 4, 9]). Cependant, il n'existe pas actuellement de protocole standard largement utilisé pour la transmission de contenu 3D, et, en particulier, qui supporte l'adaptation à des conditions de réseau variables. Nous pensons que ce manque fait obstacle à un plus large développement des environnements virtuels accessibles par les navigateurs Web.

D'un autre côté, *Dynamic Adaptive Streaming over HTTP (DASH)* [16, 15], est un standard largement déployé pour la transmission de contenu vidéo sur le Web [10]. DASH fonctionne avec des serveurs HTTP standards, desquels il hérite la simplicité de déploiement et les performances. Il supporte de multiples représentations pour le même contenu vidéo à différentes résolutions et laisse le client choisir et adapter la qualité en fonction des évolutions des conditions du réseau. Puisque l'adaptation est réalisée du côté du client, le serveur est sans état, et se contente simplement de servir des fichiers en réponse aux requêtes. De tels serveurs HTTP ont l'avantage de pouvoir servir des quantités massives d'utilisateurs.

En raison des bonnes propriétés de DASH et du besoin d'un protocole aux propriétés similaires pour la transmission de scènes 3D, une question qui émerge naturellement est de savoir si DASH peut être utilisé pour la transmission adaptative de contenu 3D. DASH

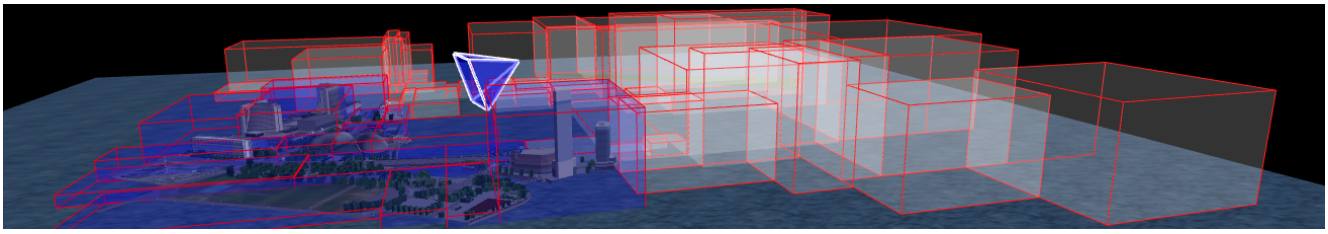

FIGURE 1 – Une scène 3D subdivisée en régions délimitées par des traits rouges, et un point de vue. En blanc, les régions à l'extérieur du champ de vision de la caméra, en bleu, les régions à l'intérieur.

étant conçu pour la transmission vidéo, l'adapter aux environnements virtuels est non trivial. Contrairement à la vidéo, où la visualisation progresse linéairement (jusqu'à un clic de l'utilisateur), l'éventail d'actions possibles dans un environnement virtuel est beaucoup plus vaste. Un client typique de NVE devra déterminer, et si possible prédire, quelles sont les données de géométrie et de textures qui sont dans le champ de vision (région définie par la position de la caméra et la direction du regard). Ceci doit être fait par le client et uniquement grâce à des métadonnées (en DASH, elles figurent dans un fichier *Media Presentation Description, MPD*), provenant du serveur, précalculées, et donc indépendantes de la vue. En conséquence, le premier défi auquel nous devons faire face est de déterminer *quelles métadonnées indépendantes de la vue le serveur a besoin de fournir avec les données 3D afin de permettre au client de prendre une décision intelligente et dépendante de la vue sur le contenu à télécharger.*

Pour une scène 3D, le contenu à transmettre comprend des données de géométrie (sommets, coordonnées de texture, faces) et des images. Ces données sont fournies sous la forme d'une soupe de polygones sans information sémantique ni graphe de scène. Les textures peuvent être à différents niveaux de résolutions. De même, les aires des faces peuvent varier significativement, ce qui implique que le choix des polygones à télécharger peut avoir un impact significatif sur la qualité du rendu. Ainsi, le deuxième défi est de savoir *comment organiser le contenu 3D dans le formalisme de DASH, pour faciliter la prise de décision du client.*

Dans la transmission vidéo DASH, la taille d'une seconde de vidéo est corrélée avec la qualité de la vidéo. Un client DASH peut maximiser le débit moyen (et donc la taille) des fragments téléchargés pour améliorer la qualité de l'expérience. Dans la transmission de contenu 3D, la qualité d'expérience est déterminée par la scène rendue (et donc dépendante du point de vue), alors que la taille des textures determine seulement la qualité dans le monde 3D (indépendante de la vue). Une texture peut être appliquée à plusieurs faces et peut apparaître dans différentes régions de la scène. L'impact relatif de la géométrie et des textures sur le rendu doit être quantifié. Le troisième défi est donc de trouver comment *un client peut prendre des décisions qui s'adaptent au réseau, en particulier de quoi télécharger (géométrie ou textures) et à quelle résolution.*

Contributions. Nous défendons trois contributions dans cet article. Premièrement, nous proposons une organisation des polygones dans un format compatible avec DASH qui facilite le téléchargement de contenu 3D par le client. Ensuite, nous proposons le précalcul d'un ensemble de métadonnées qui sont utiles au client pour s'adapter à la bande-passante et pour prendre les décisions de téléchargement. Finalement, nous présentons un client basé sur DASH pour transmettre des scènes 3D, qui s'adapte à la bande passante et à la navigation de l'utilisateur.

Organisation de l'article. Le reste du papier est structuré de la façon suivante : la section 2 présente l'état de l'art, et la section 3 décrit notre méthode pour adapter DASH au contenu 3D. Dans la section 4, nous détaillons un client DASH 3D. Ensuite, la section 5 présente les expériences et les résultats, en mettant en valeur l'impact des différents paramètres. Enfin, nous concluons dans la section 6.

2 État de l'Art

2.1 Transmission de contenu 3D

L'accès et l'interaction avec le contenu 3D sur le web ont été largement explorés dans des travaux précedents. Behr et al. ont présenté une structuration du contenu 3D pour permettre l'accès depuis un navigateur web [1].

Dans un récent effort de standardisation, le groupe Khronos a proposé un format générique nommé glTF [14] pour gérer tous les types de représentations de contenu 3D : nuages de points, maillages, modèles animés. Certains travaux se focalisent aussi sur les algorithmes de compression : Potenziani et al. décrivent un système qui télécharge progressivement du contenu 3D dans le contexte du patrimoine culturel [12]. Google Draco [6] est un effort open-source pour permettre une compression et décompression qui soient efficaces sur les navigateurs web. Leurs travaux sont appliqués dans le cadre d'objets isolés, mais leurs techniques de compression ne sont pas adaptées à de vastes environnements virtuels.

La transmission dépendante de la vue a été étudiée

dans nos travaux précédents [4], où nous calculons côté serveur les polygones à l'extérieur du champ de vision pour ne transmettre que les polygones nécessaires au client. La contrepartie de cette méthode est qu'elle passe mal à l'échelle en raison de la complexité des calculs effectués par le serveur. Chen et Ooi [2] ont proposé un streaming de maillage progressif adapté au point de vue basé client. Leur technique conduit à un serveur sans état, et supporte donc la montée en charge. En revanche, leurs travaux se focalisent sur des maillages et utilisent un protocole dédié.

L'équilibre entre le streaming de la géométrie et des textures a été étudié par Tian et al. [17], Guo et al. [7], et Yang et al. [18]. Tous ces travaux ne considèrent qu'un maillage progressif, *manifold* texturé. Leurs approches combinent la distortion causée par la basse résolution de la géométrie et des textures dans une métrique globale. Nos travaux se focalisent sur une scène complète et hétérogène, dans laquelle un codage par objet n'est pas applicable, et pour laquelle leurs métriques ne seraient donc pas adaptées.

Zampoglou et al. sont les premiers à proposer l'utilisation de DASH pour transmettre du contenu 3D [19]. Dans leurs travaux, les auteurs décrivent un système qui permet aux utilisateurs d'accéder à un contenu 3D disponible à différents niveaux de résolution en organisant le contenu en suivant la terminologie DASH. Leur approche fonctionne pour un ensemble d'objets isolés, mais ne tient pas compte du point de vue, ce qui est indispensable dans le cadre d'environnements virtuels distribués.

2.2 SRD-DASH

Bien que DASH soit conçu pour la lecture de vidéo avec un seul degré de liberté (le temps), de récents travaux ont étendu la lecture vidéo pour permettre d'autres interactions, comme les rotations (par exemple, dans les vidéos à 360°) et le zoom [13]. Ces méthodes partitionnent chaque image de vidéo en blocs qui peuvent être décodés indépendamment par le client. Dans ce contexte, Niamut et al. ajoutent une description de la représentation spatiale (*Spatial Representation Description, SRD*[11]) aux métadonnées. Ceci permet d'extraire les parties du contenu nécessaires et de les afficher, et donc aide à zoomer et à naviguer à travers le contenu [3].

DASH pour les environnements virtuels distribués partage plusieurs points communs avec SRD-DASH : par exemple, un utilisateur peut à chaque instant ne voir qu'une partie du contenu. SRD-DASH travaille par bloc, en permettant au client de ne télécharger que les blocs utiles. De la même façon, dans notre méthode, nous partitionnons la géométrie en cellules ("blocs 3D"). Contrairement à SRD-DASH, il n'est pas trivial d'assigner une face à une cellule, puisqu'une grande face (par exemple, le ciel ou le sol) sera probablement étendue sur plusieurs cellules.

3 Représentation d'une scène 3D dans le format DASH

Dans cette section, nous décrivons comment nous préparons et stockons les données 3D de notre environnement virtuel dans un format qui soit compatible avec DASH. Dans nos travaux, nous utilisons les formats WaveFront OBJ pour les polygones et PNG pour les textures. Cependant, le processus s'applique aussi à d'autres formats.

3.1 Le MPD

Dans DASH, les informations telles que les URL des fichiers, leur résolution ou leur taille sont extraites par le client d'un fichier appelé *Media Presentation Description, MPD*. Le client ne se base que sur ces informations pour décider quel fichier télécharger et à quel niveau de résolution.

Le MPD est un fichier XML organisé hiérarchiquement en différentes sections. Les *periods* sont le premier niveau, qui dans le cas de la vidéo, indiquent le début et la durée d'un chapitre. Cet élément ne s'applique pas dans le cas d'un environnement virtuel statique, et nous utilisons donc une seule *period* qui contiendra toute la scène.

Chaque *period* contient un ou plusieurs *adaptation sets* qui décrivent des versions alternatives, formats et types de contenu (images, sons, sous-titres). Nous utilisons les *adaptation sets* pour organiser la géométrie et les textures de la scène.

3.2 Adaptation Sets

Quand un utilisateur navigue librement dans une scène 3D, le champ de vision à un moment donné ne contient qu'une partie limitée de la scène. De la même façon que DASH-vidéo partitionne une vidéo en blocs temporels, nous partitionnons les polygones en blocs spatiaux, de sorte que notre client puisse ne télécharger que les blocs nécessaires.

Gestion de la géométrie. Nous utilisons un arbre de partitionnement de l'espace pour organiser les faces en cellules. Une face appartient à une cellule si son barycentre est à l'intérieur de la boîte englobante correspondante. Chaque cellule correspond à un *adaptation set*. Ainsi, l'information géométrique est étalée dans les *adaptation sets* en fonction de leur cohérence spatiale, permettant au client de choisir les faces pertinentes à télécharger. Une cellule est pertinente si son intersection avec le champ de vision de l'utilisateur est non vide. Dans la figure 1, les cellules pertinentes sont représentées en bleu.

Puisque notre scène 3D est principalement étalée le long d'un plan horizontal, nous séparons alternativement le modèle dans les deux directions de ce plan.

Nous créons un *adaptation set* séparé pour les grandes faces (par exemple, le ciel ou le sol) puisqu'elles sont essentielles au modèle 3D et qu'elles ne rentrent pas dans les cellules. Nous considérons une face comme grande si son aire est supérieure à $a+3\sigma$ où a et σ sont respectivement la moyenne et l'écart-type des aires des faces. Dans notre exemple, ceci correspond aux 5 plus grandes faces, qui représentent 15% de l'aire totale. Nous obtenons ainsi une décomposition du NVE en *adaptation sets* qui partitionne la géométrie de la scène en (i) un *adaptation set* contenant les faces les plus grandes, et (ii) d'autres qui contiennent les faces restantes.

Nous enregistrons la position de chaque *adaptation set*, caractérisée par les coordonnées de sa boîte englobante, dans le MPD comme une propriété supplémentaire de l'*adaptation set*, sous la forme "x_{min}, *largeur*, y_{min}, *hauteur*, z_{min}, *profondeur*" (comme indiqué dans le listing 1). Ces informations sont utilisées par le client pour implémenter un *streaming* dépendant de la vue (section 4).

Gestion des textures. Avec les données de géométrie, nous gérons les textures en utilisant différents *adaptation sets*, indépendants de ceux de la géométrie. Chaque fichier de texture correspond à un *adaptation set* différent, avec différentes *representations* (voir section 3.3) qui fournissent des résolutions différentes des images. Nous ajoutons à chaque *adaptation set* de texture un attribut qui décrit la couleur moyenne de la texture. Le client peut se servir de cet attribut pour dessiner une face dont la texture n'a pas encore été téléchargée avec une couleur uniforme naturelle (voir la figure 2).

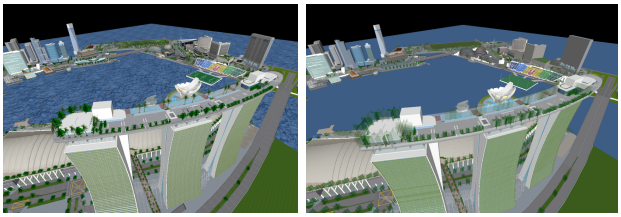

FIGURE 2 – Rendu du modèle avec les textures à la résolution maximale (gauche), et avec une couleur moyenne par face (droite).

```
<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" />
  </SegmentList>
 </Representation>
</AdaptationSet>

<AdaptationSet area="198632.73912" average="178,176,173">
    <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 – Description d'un *adaptation set* de géométrie, et d'un autre de texture, dans le MPD.

3.3 Représentations

Chaque *adaptation set* peut contenir une ou plusieurs *representations* de la géométrie ou des textures, à différent niveaux de détail (par exemple, avec un nombre différent de faces). Pour la géométrie, la résolution est hétérogène, et appliquer une représentation multi-résolution est pénible : l'aire des faces varie de $0.01m^2$ à plus de $10000m^2$, sans tenir compte des faces extrêmes. Pour les scènes texturées, il est commun d'avoir des données hétérogènes puisque l'information peut être stockée soit sous forme de géométrie, soit sous forme de texture. Anisi, gérer un compromis entre géométrie et textures est plus adaptable que gérer une combinaison multi-résolution. Pour chaque texture, nous générons des résolutions successives en divisant par 2 la hauteur et la largeur, et en s'arrêtant lorsque l'image a une taille inférieure à 64×64. La figure 2 montre l'utilisation des textures comparée à l'affichage avec une seule couleur par face.

3.4 Segments

Pour éviter la perte d'adaptabilité due au temps de transmission d'un trop gros fichier, nous groupons les faces d'un *adaptation set* en *segments*. Chaque *segment* est ensuite encodé en un fichier OBJ, qui peut être requêté individuellement par le client. Nous partitionnons les faces d'un *adaptation set* en ensembles de N_s faces, en triant les faces par aires décroissantes, et en plaçant ensuite les N_s faces successives dans un *segment*. Ainsi, le premier *segment* contient les faces les plus grandes et le dernier les faces les plus petites. Pour les textures, chaque *representation* contient un *segment* unique.

4 Client DASH 3D

Dans cette section, nous détaillons un client DASH NVE qui exploite la préparation du contenu 3D.

Le client DASH commence par télécharger le MPD, avant de commencer à télécharger les segments. Quand un segment arrive, le client prend la décision du prochain segment à télécharger pour l'afficher quand il arrivera.

Nous considérons une caméra virtuelle qui suit continuellement un chemin $C = \{v(t_i), t_i \in [t_1, t_{final}]\}$, où

t_i est l'instant où le segment i est requêté, et au cours duquel les opportunités de téléchargement sont stratégiquement exploitées pour télécharger séquentiellement les segments les plus utiles.

4.1 Utilité des Segments

Contrairement au *streaming* vidéo, où la taille (en octets) de chaque segment est corrélée à la qualité de la vidéo reçue, pour le contenu 3D, la taille du contenu n'est pas nécessairement corrélée à sa contribution en terme de qualité du rendu. Un grand polygone qui aura un grand impact visuel occupera à peu près autant d'octets qu'un petit polygone. De plus, l'impact visuel dépend du point de vue – un grand polygone lointain ne contribuera pas autant qu'un petit polygone plus près de l'utilisateur. Ainsi, il est important pour un client DASH-NVE d'estimer ce que nous appelons *l'utilité d'un segment*, de sorte à prendre les bonnes décisions de téléchargement.

L'utilité est une fonction d'un segment, qu'il soit de géométrie ou de texture, et du point de vue courant (position de la caméra et direction du regard), et est donc calculé dynamiquement par le client à partir des métadonnées du MPD.

Paramètres statiques. Dans un premier temps, nous allons détailler les paramètres calculés lors de la préparation du contenu, stockés dans le MPD.

Tout d'abord, pour chaque segment de géométrie s^G, nous calculons une aire $\mathcal{A}(s^G)$, égale à la somme des aires des polygones du segment. Ensuite, pour chaque segment de texture s^T, le MPD enregistre l'erreur quadratique moyenne (EQM) entre l'image courante et l'image à la plus haute résolution disponible. Enfin, pour tous les segments, nous stockons la taille du fichier (en octets). En effet, les segments de géométrie ont un nombre similaire de faces, et leurs tailles sont donc à peu près les mêmes. En ce qui concerne les textures, les tailles sont généralement bien plus faibles que celles des segments de géométrie, mais sont aussi très variables, puisqu'entre deux résolutions, le nombre de pixels est multiplié par 4.

Paramètres dynamiques. En plus des paramètres statiques stockés dans le MPD pour chaque segment, des paramètres dépendants du point de vue sont calculés pendant la navigation. Premièrement, une mesure d'aire est calculée pour les segments de texture. Puisqu'une texture est peinte sur un ensemble de polygones, on compte pour l'aire de la texture la somme des aires de ces polygones. Nous pourrions calculer cette information de manière statique et la stocker dans le MPD, mais faire ce calcul dynamiquement nous permet de ne prendre en compte que les polygones déjà reçus par le client. Pour une texture T, nous notons l'ensemble des polygones peints par cette texture $\Delta(s^T) = \Delta(T)$ (qui ne dépend que de la texture T et qui est donc constante pour chaque représentation de la texture). À chaque instant t_i, un sous ensemble de $\Delta(T)$ a été téléchargé, nous le notons $\Delta(T, t_i)$.

De plus, chaque segment de géométrie appartient à un *adaptation set* AS^G dont les coordonnées de la boîte englobante sont stockées dans le MPD. Étant donné la boîte englobante $\mathcal{BB}(AS^G)$ et le point de vue $v(t_i)$ à l'instant t_i, le client calcule la distance $\mathcal{D}(v(t_i), AS^G)$ à $\mathcal{BB}(AS^G)$ comme la distance du centre de $\mathcal{BB}(AS^G)$ au point principal de la caméra.

Utilité des segments de géométrie. Nous disposons maintenant de tous les paramètres pour déduire une mesure d'utilité d'un segment de géométrie. L'utilité des segments de textures est déduite des utilités des segments de géométrie.

L'utilité d'un segment de géométrie s^G pour un point de vue $v(t_i)$ est

$$\mathcal{U}\Big(s^G, v(t_i)\Big) = \frac{\mathcal{A}(s^G)}{\mathcal{D}(v(t_i), AS^G)^2} \tag{1}$$

où AS^G est l'*adaptation set* qui contient s^G.

Concrètement, l'utilité d'un segment est proportionnelle à l'aire couverte par ce segment, et inversement proportionnelle au carré de la distance entre la caméra et la boîte englobante de son *adaptation set*. De cette manière, nous favorisons les segments contenant des grandes faces, et qui sont proches de la caméra.

Utilité des segments de texture. Pour une texture T, les polygones de $\Delta(T)$ peuvent être dans plusieurs segments de géométrie. Ainsi, pour chaque segment de géométrie déjà téléchargé $s^G_k \in K$, on compte les polygones de $\Delta(T, t_i)$ dans s^G_k, et on calcule ainsi le ratio qu'occupe T dans $\mathcal{A}(s^G_k)$. Nous définissons donc l'utilité d'un segment de texture par

$$\mathcal{U}\Big(s^T, v(t_i)\Big) = psnr(s^T) \sum_{k \in K} \frac{\mathcal{A}_{3D}(s^G_k \cap \Delta(T, t_i))}{\mathcal{A}_{3D}(s^G_k)} \mathcal{U}\Big(s^G_k, v(t_i)\Big) \tag{2}$$

Concrètement, cette formule définit l'utilité d'un segment de texture grâce à la combinaison linéaire des utilités des segments de géométrie qui utilisent cette texture, pondérées par la proportion occupée par la texture dans le segment. On calcule ensuite un PSNR en utilisant l'erreur quadratique moyenne du MPD et on le note $psnr(s^T)$, de sorte à donner une utilité plus grande à des textures de plus haute résolution.

Le client peut ainsi utiliser les utilités définies sur les segments de géométrie et de textures pour sa stratégie de chargement.

4.2 Logique d'adaptation de DASH

Le long du chemin de caméra $C = \{v(t_i)\}$, les points de vue sont indexés par un intervalle de temps continu $t_i \in [t_1, t_{final}]$. Par contraste, la logique d'adaptation

de DASH procède séquentiellement le long d'une suite discrète d'instants. La première requête HTTP faite par le client DASH à l'instant t_1 choisit le segment le plus utile s_1^*, qui sera suivi des décisions suivantes aux instants $t_2, t_3, \ldots$. Pour choisir le segment s_i^* , le client doit faire un compromis entre la géométrie et les différentes résolutions de texture en tenant compte du débit, des mouvements de la caméra, et des utilités des segments. La différence entre t_{i+1} et t_i correspond au temps que va mettre s_i^* à arriver. Cette durée peut varier en fonction de la taille du segment et des conditions du réseau. L'algorithme 1 explique comment notre client DASH prend ses décisions.

entrée : Indice courant i, instant t_i, point de vue $v(t_i)$, ensemble des segments déjà téléchargés $\mathcal{B}_i$, MPD
sortie : Prochain segment s_i^* à télécharger
- Estimer la bande passante $\widehat{BW_i}$ et le temps d'aller-retour $\widehat{\tau_i}$;
- Parmi les segments non téléchargés $s \in \mathcal{S} \backslash \mathcal{B}_i$, conserver ceux qui sont dans le champ de vision à venir $\mathcal{CV}$ grâce à un prédicteur de point de vue $t_i \rightarrow \hat{v}(t_i)$;
- Optimiser un critère Ω basé sur les utilités $\mathcal{U}$ et des points de vue bien choisis $v(t_i)$ pour choisir le prochain segment à télécharger

$$s_i^* = \underset{s \in \mathcal{S} \backslash \mathcal{B}_i \cap \mathcal{CV}}{\operatorname{argmax}} \Omega_{\theta_i}\Big(\mathcal{U}(s, v(t_i)\Big) \tag{3}$$

à l'aide de paramètre θ_i qui contient les paramètres dynamiques $(i, t_i, v(t_i), \widehat{BW_i}, \widehat{\tau_i}, \mathcal{B}_i)$ et les métadonnées du MPD ;
- **return** *segment* s_i^*;

Algorithme 1 : Sélection du prochain segment

La façon la plus naïve de séquentiellement optimiser $\mathcal{U}$ est de limiter la décision au point de vue courant $v(t_i)$. Dans ce cas, le meilleur segment s à télécharger sera celui qui maximisera $\mathcal{U}(s, v(t_i))$ pour simplement avoir un meilleur rendu au point de vue courant $v(t_i)$. À cause des délais de transmission, ce segment n'arrivera qu'à l'instant $t_{i+1} = t_{i+1}(s)$ qui dépendra des conditions du réseau et de la taille du segment

$$t_{i+1}(s) = t_i + \frac{\texttt{size}(s)}{\widehat{BW_i}} + \widehat{\tau_i} \tag{4}$$

En conséquence, le segment le plus utile depuis $v(t_i)$ à l'instant t_i sera peut-être moins utile au moment où il arrivera, à l'instant t_{i+1}.

Une meilleure solution est de télécharger un segment qui devrait être plus utile dans le futur. Avec un horizon temporel χ, nous pouvons optimiser le cumul de $\mathcal{U}$ pendant $[t_{i+1}(s), t_i + \chi]$:

$$s_i^* = \underset{s \in \mathcal{S} \backslash \mathcal{B}_i \cap \mathcal{FC}}{\operatorname{argmax}} \int_{t_{i+1}(s)}^{t_i+\chi} \mathcal{U}(s, \hat{v}(t_i))dt \tag{5}$$

Nous avons aussi testé une approche gloutonne qui optimise l'utilité à l'arrivée du segment :

$$s_i^{\texttt{GLOUTON}} = \underset{s \in \mathcal{S} \backslash \mathcal{B}_i \cap \mathcal{FC}}{\operatorname{argmax}} \frac{\mathcal{U}\left(s, \hat{v}\left(t_{i+1}(s)\right)\right)}{t_{i+1}(s) - t_i} \tag{6}$$

5 Évaluation

Nous décrivons maintenant notre protocole expérimental et les données que nous utilisons. Nous présentons une évaluation de notre système et une comparaison de l'impact des choix de conception que nous avons introduit dans les sections précédentes.

5.1 Protocole expérimental

Modèle. Dans nos expériences, nous utilisons un modèle du quartier de Marina Bay à Singapour. Le modèle contient 387.551 sommets et 552.118 faces. La géométrie occupe 62 MO et les textures en occupent 167. Nous partitionnons la géométrie dans un *k-d tree* jusqu'à ce que les feuilles contiennent moins de 10.000 faces, ce qui nous donne 64 *adaptation sets*, plus un pour les grandes faces.

Navigation des utilisateurs. Pour évaluer notre système, nous avons collecté des traces d'utilisateurs réalistes que nous pouvons rejouer.

Nous avons présenté notre interface web à six utilisateurs, sur laquelle le modèle se chargeait progressivement pendant que l'utilisateur pouvait naviguer. Les interactions disponibles sont inspirées des jeux vidéos à la première personne (le clavier pour se déplacer et la souris pour tourner). Nous avons demandé aux utilisateurs de naviguer et d'explorer la scène jusqu'à ce qu'ils estiment avoir visité les régions les plus importantes. Nous leur avons ensuite demandé d'enregistrer un chemin qui donnerait une bonne présentation de la scène à un utilisateur qui voudrait la découvrir. Toutes les 100 ms, la position et l'angle de la caméra sont enregistrées dans un tableau qui sera ensuite exporté au format JSON. Les traces enregistrées nous permettent de rejouer chaque enregistrement et d'effectuer les simulations et évaluations de notre système. Nous avons ainsi collecté 13 enregistrements.

Configuration du réseau. Nous avons testé notre implémentation sous trois débits de 2.5 Mbps, 5 Mbps et 10 Mbps avec un temps aller-retour de 76 ms, en suivant les paramètres de DASH-IF [5]. Les valeurs restent constantes pendant toute la durée de la session pour analyser les variations de performance en fonction du débit.

Dans nos expériences, nous créons une caméra virtuelle qui suit un enregistrement, et notre système télécharge les segments en temps réel selon l'algorithme 1.

Paramètres	Valeurs
Utilité	Statique, Dynamique, Combinée
Stratégie	Gloutonne, Proposée
Segments	Triés par aire, non triés
Bande passante	2.5 Mbps, 5 Mbps, 10 Mbps

TABLE 1 – Paramètres de nos expériences

Nous enregistrons dans un fichier JSON les moments où les segments sont requêtés et reçus. En faisant ainsi, nous évitons de gaspiller le temps et les ressources nécessaires à l'évaluation du système pendant que les segments sont en train d'être téléchargés et au stockage des informations nécessaires pour tracer les courbes présentées dans les prochaines sections.

Machines et logiciels. Les expériences ont été lancées sur un Acer Aspire V3, avec un processeur Intel Core i7-3632QM et une carte graphique NVIDIA GeForce GT 740M. Le client DASH est écrit en Rust, et utilise Glium pour le rendu et reqwest pour le téléchargement des segments.

Métriques. Pour mesurer objectivement la qualité du rendu, nous utilisons le PSNR. La scène rendue a posteriori en utilisant les mêmes chemins mais en ayant téléchargé toute la géométrie et les textures est utilisée comme vérité terrain. Dans notre cas, une erreur de pixel ne peut arriver que lorsqu'une face est manquante ou quand une texture est manquante ou à une résolution trop faible.

Expériences. Nous présentons des expériences qui valident nos choix d'implémentation à chaque étape de notre système. Nous rejouons les chemins créés par les utilisateurs avec différentes conditions de débit tout en variant les composants clés de notre système.

Nous considérons deux stratégies de chargement appliquées à notre client, proposées dans la section 4. La stratégie gloutonne détermine, à chaque décision, le segment qui maximise l'utilité prédite du segment au moment de son arrivée, ce qui correspond à l'équation (6). La deuxième stratégie de chargement que nous testons est celle proposée dans l'équation (5). Nous avons aussi analysé l'impact du groupement des faces dans les segments de géométrie en fonction de leur aire. Enfin, nous testons différents paramètres de bande-passante pour étudier comment notre système s'adapte à des conditions de réseau différentes.

5.2 Résultats expérimentaux

La figure 3 montre comment la métrique d'utilité peut exploiter les paramètres statiques et dynamiques. Les expériences utilisent un *k-d tree* et la politique de chargement proposée, sur tous les chemins. On observe

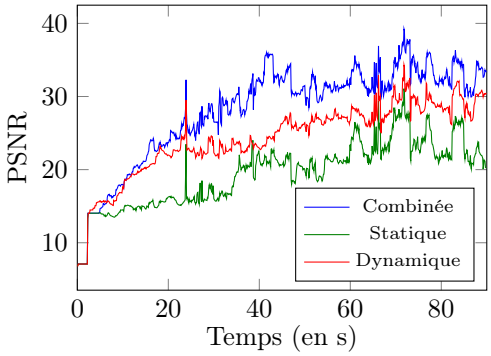

FIGURE 3 – Impact de l'utilité des segments de géométrie sur le rendu à une débit de 5 Mbps.

qu'une métrique d'utilité purement statique donne des mauvais PSNR. Une utilité purement dynamique donne des résultats légèrement meilleurs, notamment grâce a l'élimination des parties à l'extérieur du champ de vision, mais la version combinée décrite dans la section 4.1 donne les meilleurs résultats.

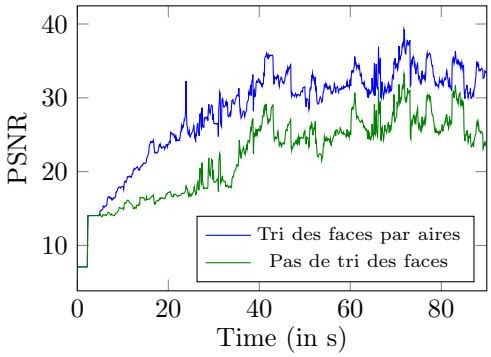

FIGURE 4 – Impact du tri des faces dans les segments à un débit de 5 Mbps

La figure 4 montre l'impact de l'arrangement des polygones dans les segments en fonction de leur aire. Il est clair que le PSNR augmente considérablement lorsque l'aire des faces est prise en compte lors de la création des segments. Puisque les segments ont tous la même taille (en octets), trier les faces par aire avant de les ranger dans les segments introduit une asymétrie dans la distribution des aires. Cette asymétrie permet au client de prendre des décisions (télécharger les segments avec la plus grande utilité) et peut créer une grande différence en terme de qualité de rendu.

Nous avons aussi comparé l'approche gloutonne et celle proposée (voir figure 5) pour une bande-passante limitée (5 Mbps). La méthode proposée est meilleure sur les 30 premières secondes et fait mieux en moyenne. La table 2 montre le PSNR moyen pour les deux méthodes pour différentes bandes-passantes. C'est sur les 30 premières secondes que les décisions sont cruciales puisqu'elles correspondent aux moments où peu de

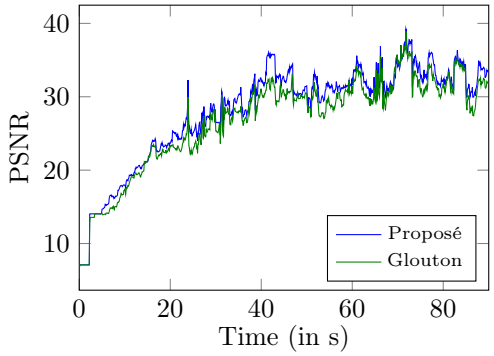

FIGURE 5 – Impact de la politique de chargement (glouton vs proposé) à un débit de 5 Mbps

	Premières 30s			**Globalement**		
BP (Mbps)	2.5	5	10	2.5	5	10
Glouton	14.4	19.4	22.1	19.8	26.9	29.7
Proposé	16.3	20.4	23.2	23.8	28.2	31.1

TABLE 2 – PSNR moyens, Glouton vs Proposé

contenu a été téléchargé. Nous observons que notre méthode augmente la qualité du rendu entre 1 et 1.9 dB par rapport à l'approche gloutonne.
La table 3 montre la distribution des textures téléchargées par les deux approches, à différents débits. La résolution 5 est la plus détaillée, et la résolution 1 la plus grossière. Cette table met en évidence une faiblesse de la politique gloutonne : quand le débit augmente, la distribution des résolutions téléchargées reste plus ou moins la même. En revanche, notre politique s'adapte en téléchargeant des plus hautes résolutions quand le débit est meilleur (13.9% à 10 Mbps contre 0.3% à 2.5 Mbps). En fait, une propriété intéressante de la politique proposée est qu'elle adapte le compromis géométrie-texture à la bande-passante. Les textures représentent 57.3% des octets téléchargés à 2.5 Mbps, et 70.2% à 10 Mbps. En d'autres termes, notre système tend à favoriser la géométrie quand le débit est faible, et favoriser les textures quand le débit augmente.

6 Conclusion

Notre travail sur ce papier a commencé avec la question : peut-on utiliser DASH pour la transmission des NVE ? La réponse est *oui*. Pour répondre à cette question, nous avons montré comment organiser une soupe de polygones et ses textures dans un format compatible avec DASH qui inclut un minimum de métadonnées utiles au client, et organise le contenu pour permettre au client de télécharger le contenu le plus utile

R	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 3 – Pourcentage d'octets téléchargés pour chaque résolution de texture, pour la politique gloutonne (gauche) et pour celle proposée (droite)

en premier. Nous avons ensuite montré que ces métadonnées précalculées sont suffisantes pour concevoir et implémenter un client DASH adaptable – il peut choisir les segments dans son champ de vision, en prenant des décisions intelligentes, et en faisant un compromis entre la géométrie et les textures tout en s'adaptant aux conditions du réseau.

Références

[1] J. Behr, Y. Jung, J. Keil, T. Drevensek, M. Zoellner, P. Eschler, and D. Fellner. A scalable architecture for the HTML5/X3D integration model X3DOM. ACM, 2010.

[2] W. Cheng and W. T. Ooi. Receiver-driven view-dependent streaming of progressive mesh. NOSSDAV. ACM, 2008.

[3] L. D'Acunto, J. van den Berg, E. Thomas, and O. Niamut. Using MPEG DASH SRD for zoomable and navigable video. MMSys. ACM, 2016.

[4] T. Forgione, A. Carlier, G. Morin, W. T. Ooi, and V. Charvillat. Impact of 3d bookmarks on navigation and streaming in a networked virtual environment. MMSys. ACM, 2016.

[5] D. I. Forum. Guidelines for implementation : DASH-AVC/264 test cases and vectors, 2014.

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

[7] J. Guo, V. Vidal, I. Cheng, A. Basu, A. Baskurt, and G. Lavoue. Subjective and objective visual quality assessment of textured 3d meshes. *ACM Transactions on Applied Perception (TAP)*, 2017.

[8] S.-Y. Hu, T.-H. Huang, S.-C. Chang, W.-L. Sung, J.-R. Jiang, and B.-Y. Chen. Flod : A framework for peer-to-peer 3D streaming. INFOCOM. IEEE, 2008.

[9] Y. Hu, Z. Chen, X. Liu, F. Huang, and J. Jia. WebTorrent based fine-grained P2P transmission of large-scale WebVR indoor scenes. ACM, 2017.

[10] Information technology – Dynamic adaptive streaming over HTTP (DASH) – Part 1 : Media presentation description and segment formats. Standard, 2014.

[11] O. A. Niamut, E. Thomas, L. D'Acunto, C. Concolato, F. Denoual, and S. Y. Lim. Mpeg dash srd : Spatial relationship description. MMSys. ACM, 2016.

[12] M. Potenziani, M. Callieri, M. Dellepiane, M. Corsini, F. Ponchio, and R. Scopigno. 3dhop : 3d heritage online presenter. *Computers & Graphics*, 2015.

[13] N. Quang Minh Khiem, G. Ravindra, A. Carlier, and W. T. Ooi. Supporting zoomable video streams with dynamic region-of-interest cropping. MMSys. ACM, 2010.

[14] F. Robinet and C. Patrick Cozzi. gltf - runtime asset format for webgl, opengl es, and opengl, 2013.

[15] I. Sodagar. The MPEG-DASH Standard for Multimedia Streaming Over the Internet. *IEEE Multimedia*, 2011.

[16] T. Stockhammer. Dynamic adaptive streaming over http – : Standards and design principles. MMSys. ACM, 2011.

[17] D. Tian and G. AlRegib. Batex3 : Bit aloptlocation for progressive transmission of textured 3-d models. *IEEE Transactions on Circuits and Systems for Video Technology*, 2008.

[18] S. Yang, C.-H. Lee, and C.-C. J. Kuo. Optimized mesh and texture multiplexing for progressive textured model transmission. MULTIMEDIA '04. ACM, 2004.

[19] M. Zampoglou, K. Kapetanakis, A. Stamoulias, A. G. Malamos, and S. Panagiotakis. Adaptive streaming of complex Web 3D scenes based on the MPEG-DASH standard. *Multimedia Tools and Applications*, 2016.