VIA: Visibility-aware Web-based Virtual Reality

Carter Slocum
University of California, Riverside
Riverside, CA, USA
csloc001@ucr.edu

Jingwen Huang
University of California, Riverside
Riverside, CA, USA
jhuan333@ucr.edu

Jiasi Chen
University of California, Riverside
Riverside, CA, USA
jiasi@cs.ucr.edu

ABSTRACT

New standards such as WebXR enable cross-platform VR experiences, relying on the ubiquity of the modern web browser. However, upon measuring performance of WebXR scenes, we found users can suffer from high latency while waiting for all 3D objects appear in their field-of-view. This is because storage and fetching of 3D objects in WebXR (and its underlying WebGL libraries) are agnostic to the user's orientation and location, leading to latency issues. Specifically, fetching of texture files in arbitrary order results in 3D objects waiting on their texture dependencies, and the storage of all objects' geometry data in one large file blocks individual objects from rendering even if their texture dependencies are satisfied. To address these issues, we propose a systematic prioritization of which 3D objects and their dependencies should be fetched first, based on the user's position and orientation in the VR scene. To improve efficiency, the geometry data belonging to each 3D object are optimally grouped together to minimize the average latency. Our experiments with various WebXR scenes under different network conditions show that our scheme can significantly reduce the time to all 3D objects appearing in the user's field-of-view, by up to 50%, compared the default WebXR behavior.

CCS CONCEPTS

• **Networks**; • **Human-centered computing** → **Ubiquitous and mobile computing**;

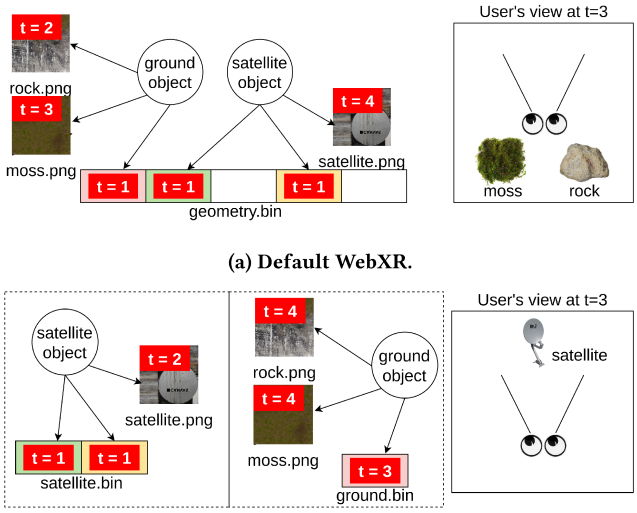
KEYWORDS

Virtual reality, page load time, WebXR

ACM Reference Format:
Carter Slocum, Jingwen Huang, and Jiasi Chen. 2021. VIA: Visibility-aware Web-based Virtual Reality. In *The 26th International Conference on 3D Web Technology (Web3D '21), November 8–12, 2021, Pisa, Italy.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3485444.3487641

1 INTRODUCTION

New standards such as WebXR [W3C 2021] enable cross-platform augmented and virtual reality (AR/VR) experiences by processing and displaying content through web browsers. This enables developers to write a single WebXR experience and have it work across multiple AR/VR devices, such as Oculus Quest and HTC Vive. Under the hood, WebXR works by calling WebGL Javascript libraries and retrieving the relevant assets from a remote web server. Once an object has all its dependent assets (geometry and texture data), the object is rendered on the VR display. WebXR is supported by most modern web browsers including Chrome and Edge and is standardized by the W3C.



(a) Default WebXR.

(b) Our approach, VIA.

Figure 1: By default, WebXR downloads the objects and their dependencies in arbitrary order (a), while VIA prioritizes downloading objects within the user's FoV so they can appear sooner (b).

Motivated by the current support and active development of WebXR, we experimented with various WebXR sample scenes, and observed significant performance issues in terms of latency. Users loading a WebXR scene had to wait up to 9.86 seconds under a 69 Mbps connection until all objects were fully visible in the user's field-of-view (FoV), hurting user experience. The reason for these high latencies is that WebXR is agnostic to the user's current FoV, and retrieves dependencies in an arbitrary order determined simply by how they are listed in a metadata file. In the worst case, objects that are behind a user might be downloaded and rendered first, while objects that are directly in front of the user might be download and rendered last, leaving the user to view a blank screen in the meantime. Based on these observations, in this work we propose reducing the load times of WebXR scenes, by optimizing how objects are stored and retrieved from a server, as illustrated in Fig. 1.

However, this is not a straightforward problem to solve for the following reasons. Firstly, it is not clear in what order objects should be retrieved, as some objects are entirely contained within the FoV,

Web3D '21, November 8–12, 2021, Pisa, Italy

ACM ISBN 978-1-4503-9095-8/21/11.
https://doi.org/10.1145/3485444.3487641

while other objects may span the entire FoV, or even be out of the FoV entirely. Secondly, the geometry data associated with WebXR objects is stored in very coarse-grained format (one large binary file), preventing fine-grained resource requests. To address the first challenge, we propose a new scoring function that prioritizes which objects to download, based on whether objects are potentially visible and their orientation from the center of the FoV. To address the second challenge, we propose grouping geometry data together into logically-meaningful chunks that minimize the average download time of any object in a scene. Implementing these techniques requires only changes to the server-side code, and works with un-modified clients and browsers.

This project is related to research in viewport optimization for web pages [Butkiewicz et al. 2015; Netravali et al. 2016] and 360° videos [Corbillon et al. 2017; Guan et al. 2019; Qian et al. 2018; Zhou et al. 2018] but differs in several major respects. Firstly, web page optimizations that prioritize resources "above the fold" rely on the DOM tree, which does not contain information about WebXR objects and their dependencies; furthermore, the implementation is very different as WebXR involves working with WebGL and Javascript, rather than mainly HTML/Javascript/CSS. Secondly, 360° video optimizations prioritize 2D tiles in the user's FoV from a single user location, whereas our solution prioritizes 3D objects and their dependencies, for any user location and orientation.

To the best of our knowledge, this is the first work to combine page load reduction time reduction techniques with VR applications, improving the user experience for browser-based VR. We call our system VIA, short for VIsibility-Aware Web-based VR. The contributions of this paper can be summarized as:

- We showcase the latency issues of web-based VR by measuring the page load times of several WebXR sample scenes, and find that inefficient object download ordering is the root cause of the observed high latencies. For example, the time until all content is rendered in the user's FoV for a Shack scene (details in §7.1) is 41.26 seconds under a 10 Mbps connection. However, the objects in the FoV consumed only 64% of the total objects requested, indicating there are opportunities for significant savings.
- We propose a scoring method to determine which objects should receive priority downloads. The score is a combination of visibility and orientation from the center of the user's FoV. Given these scores, the WebGL metadata, geometry data, and and Javascript code are minimally modified to request the objects and their dependencies in the appropriate order. To enable fine-grained retrieval of object dependencies, we group relevant geometry data together on the server, enabling efficient download any combination of objects.
- To implement the above techniques, we develop a parser to determine WebXR dependencies. We experiment with various WebXR scenes under different network conditions. Our results show that our method reduces page load times by up to 50.3%, compared to the default WebXR implementation. Furthermore, our method works for different user FoVs and is robust to misestimation of the user FoV.

Next, we discuss related work (§2), a brief background on WebXR (§3), and the motivating measurements for our work (§4). Our problem setup and solutions are presented in §5, experimental results in §7, and conclusions in §8. The technical report and open-source code are provided on a website [Slocum and Huang 2021].

2 RELATED WORK

360° videos: Multiple papers study how to efficiently download pixels within the FoV for 360° videos (*e.g.*, [Corbillon et al. 2017; Guan et al. 2019; Qian et al. 2018; Zhou et al. 2018]). However, 360° videos are different from the true 3D scenes we consider in this work, as 360° videos only consist of 2D video data, and the decision are which tiles to request, unlike the objects, images, data buffers, and their dependencies that we have to consider in WebXR. Furthermore, typically 360° video scenes can only be viewed from a single position, whereas our method works for any initial viewing positions and orientations.

Other VR FoV optimizations: FlashBack [Boos et al. 2016] prerenders 3D scenes to reduce latency in a thin-client design, whereas this work follows the WebXR architecture where the client browser performs rendering. Vivo [Han et al. 2020] optimizes fetching of point cloud data based on visibility, whereas WebXR 3D objects we consider in this work are typically stored as meshes plus textures and normal maps that cover the meshes. WebXR objects thus require different splitting techniques to enable visibility awareness. [Hu et al. 2017] has a similar approach of prioritising the download of parts of a scene, using a different scoring heuristic and a more coarse-grained object grouping. Our approach works with the recent WebXR standard, is tested on more than one indoor scene, and uses the standard HTTP client-server architecture rather than relying on P2P torrents. Other space partitioning structures such as k-d trees [Assarsson and Moller 2000] could produce alternative object orderings than VIA's scoring method; however, we believe these gains would be incremental, as the majority of the latency savings come simply doing some form of intelligent ordering.

Page load time: Klotski [Butkiewicz et al. 2015] and Polaris [Netravali et al. 2016], among others, perform dependency analysis for webpages in order re-order content delivery for faster rendering "above the fold", but cannot parse WebXR metadata for the unique dependency structure of 3D models. Their proxy-based implementation could be adapted for use in VIA, rather than the server-based mechanisms we propose. Tools such as Lighthouse [goo 2021] can record various metrics related to page load time, but cannot accurately capture when all 3D objects are visible in WebXR in our experience (discussed in Section 7.1), and do not provide WebXR-specific suggestions for page load time reduction.

3D scenes and models: Previous work has been done to optimise page load times for streaming specific types of 3D models using the glTF [Schilling et al. 2016]; however, our solution is more general as it applies to arbitrary scenes and not just cities. 3D Tiles [Cesium 2017] has similar goals and methods as this project, but requires converting 3D data to the 3D Tiles file format and using a custom Cesium viewer. VIA can be considered a more "lightweight" version of 3D Tiles that works directly with WebXR. Multiple authors [Lavoué et al. 2013; Limper et al. 2013] suggest 3d graphic data compression and streaming standards, reducing the amount of bytes sent over the wire while requiring the browser to decompress

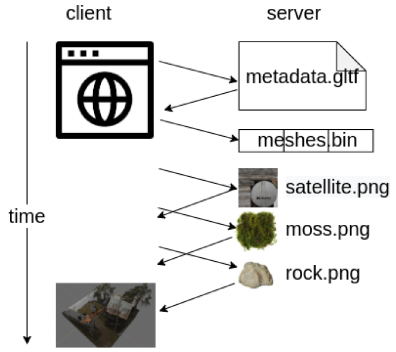


Figure 2: WebXR retrieves metadata, objects, and their data buffer and image dependencies from a remote server in order to render a 3D scene.

the data before rendering, which may be complementary in addition to VIA. Several works consider 3D object streaming by modifying the underlying content, such as using "geometry images" [Hu et al. 2008] or texture/mesh compression [Forgione et al. 2018; Hristova et al. 2020], which are complementary to this work, which focuses on the order in which to fetch those objects.

3 BACKGROUND

We first provide a brief background on WebXR. WebXR is an API for web-based AR/VR that enables cross-platform support for different hardware. It is the successor to WebVR, with contributors including Google and Mozilla, and the latest W3C working draft was published in July 2020 [W3C 2021]. WebXR operates as shown in Fig. 2. On the client, a WebXR scene is loaded like a regular webpage, and includes Javascript control code that handles frame updates, rendering, input devices, etc. with the help with WebGL libraries. The Javascript code first loads a metadata file (.glTF) that provides information on what objects are present in the scene, their locations and orientations, and their dependencies. These dependencies include the data buffers and image URIs to be rendered, which are stored on a remote server. In this paper, we call the geometry data and its associated information (*e.g.*, texture mappings, vertex positions, etc.) as a "data buffer" (corresponding to a bufferView and its corresponding accessor in the glTF standard). Once the Javascript has retrieved the objects and their dependencies, they can be rendered on the display. On the server, the data buffers for multiple objects are stored by default in one large (.bin) binary file, and the texture data are stored as individual image files (typically .png or .jpg).

4 MOTIVATION: HIGH LATENCY IN DEFAULT WEBXR

In this section, we highlight the problems with the default loading process of WebXR scenes and its root causes, based on our measurements and analysis. We conducted experiments with the default WebXR implementation in Google Chrome, viewing several sample scenes and recording the latency until all the objects within the FoV appeared on the screen (further details in Section 7). Our main observations are that startup latency is 44.26 seconds on average on a 10 Mbps connection, and the root causes of this latency are (a) the initial view-agnostic object fetching order and (b) the coarse-grained storage of data buffers on the server, as further described below.

High latency: We first show that the time to first correct frame (*i.e.*, when all the objects appear in the FoV) which we hereafter refer to as the latency, is high. Fig. 3a shows the average latency for each test case, under a 10 Mbps or 3G connection. The latency is up to 51 seconds for 10 Mbps and 350 seconds for 3G. Similar multi-second load times have also been observed in 4G and 5G networks [Nam et al. 2019]. Next, we unpack the root causes of these high latencies.

User-agnostic object fetching order: We observed that when a WebXR scene is loaded, all the object dependencies are downloaded according to an arbitrary fixed order (based on the order they are listed in the .glTF metadata). This causes problems because the download order is agnostic to the user's FoV. For example, as shown in the network request trace in Fig. 3b, the texture files belonging to objects behind the user (*e.g.*, roof tiles) are fetched earlier, while the texture files belonging to objects in front of the user (*e.g.*, satellite dish) are fetched later. This delays the rendering of objects in front of the user. Such observations motivated our object scoring strategy, which determines what objects to prioritize in the user's FoV and fetches their dependencies first, decreasing the latency (see Section 5.2).

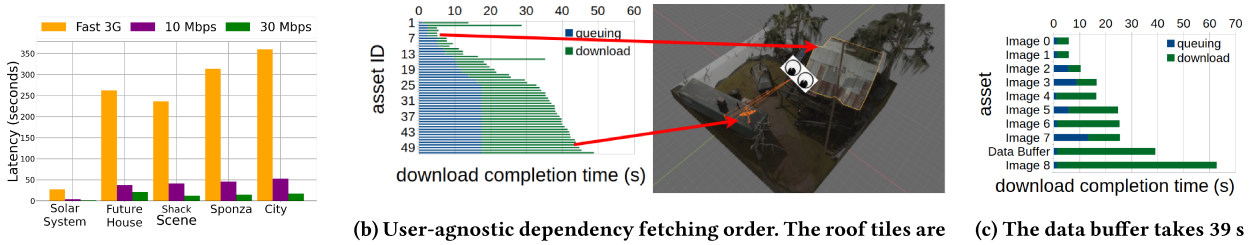
Coarse-grained data buffer storage: Besides image (texture and normal) data, objects also require data buffers containing geometry meshes and other related information in order to render the object correctly. However, we observed that the data buffers are typically stored in one large .bin file, causing issues, because a single object cannot be rendered until the entire file has been downloaded. For example, in Fig. 6c, the data buffer asset is the second-slowest file to download under a 10 Mbps connection due to its size; no objects can render before the data buffer finishes downloading at 39 seconds. These observations motivated us to consider splitting the binary appropriately; however, the challenge is to determine the split granularity – a fine granularity allows object fetching flexibility but incurs an extra RTT for each request (see Section 5.3).

5 PROBLEM AND SOLUTIONS

5.1 Overview

To quickly download and render the objects within the user's FoV, we have to solve the two aforementioned problems of object dependency fetching order, and coarse-grained asset storage. Our system system, has two modules:

- **Object scoring (Section 5.2):** The Object Scoring module's task is to determine which objects are within the user's FoV, and assign scores to the objects and their dependencies for later request order optimization. Here, the problem is to determine in what order to request the objects, based on where they are in the scene. The main idea is that objects within the user's FoV and directly centered in front should have higher priority. This information needed to compute this can be obtained from the glTF metadata for the scene.



(a) Default WebXR load latency of 38-51 seconds under 10Mbps.

(b) User-agnostic dependency fetching order. The roof tiles are fetched earlier despite being behind the user, and satellite dish is fetched later despite being within the FoV.

(c) The data buffer takes 39 s to load, blocking all objects from rendering.

Figure 3: High latency in default WebXR (a) is due to user-agnostic dependency fetching order (b) and coarse-grained data buffer storage (c).

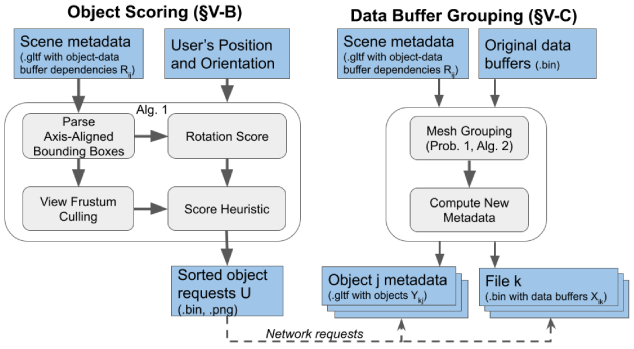


Figure 4: Overview of VIA.

Table 1: Table of notation.

Symbol	Definition
A_j	axis aligned bounding box of object j
B	Bandwidth
C_P, C_O, C_F	camera position, orientation, and FoV parameters
$\mathcal{O} \subseteq \mathcal{U}$	subset of objects in the user's current FoV
R_{ij}	Whether asset i is needed by object j
$\bar{s}_i$	size of asset i
s_k	size of chunk k
T	RTT/2
$\mathcal{U}$	complete set of objects in the scene
U	sorted list of objects in the scene
X_{ik}	Whether asset i is in chunk k
Y_{ij}	Whether object j requests chunk k

- **Data buffer grouping (Section 5.3):** The Data Buffer Grouping module stores the data buffer dependencies in a finer-grained fashion so that individual dependencies can be fetched in the order prescribed by the Object Scoring module. The problem here is to determine which data buffers to group together, to enable fine-grained object requests, while preserving sufficient aggregation for efficient downloads. The main idea is to define an optimization problem to group data buffers in order to minimize the average download time per object, and show that our proposed solution is optimal.

The output of the Object Scoring module are the order in which to request objects; these requests are sent to the server to retrieve the relevant objects and their dependencies, as stored by the data buffer Grouping module. A summary of the inputs, outputs, and interactions between the modules are shown in Fig. 4, and their details are provided next.

5.2 Object Scoring

The main intuitions behind our object scoring method are to deprioritize: (a) objects that have no vertices within the FoV and (b) objects that are far from the center of the FoV in terms of angle. This requires computing two weights in our method in Alg. 1: a visibility weight, and an angle weight. Note that sometimes these two objectives may be in conflict with each other. For instance, there may be an object that is centered behind the user, but some parts of the object are visible within the FoV (*e.g.*, a ground object); in such cases, our method returns a moderately high score due to its visibility and because the bulk of the object is behind the user.

Visibility Check. The visibility check returns whether (any part of) an object j is within the FoV, and penalizes any object that is guaranteed not to be within the camera's initial FoV by adding π to the object's score. Specifically, the visibility check relies on computing the viewing frustum (line 5), which is the area in the 3D scene that will be projected onto the user's 2D screen. Then, view frustum culling [Assarsson and Moller 2000] (line 7) is performed to see if an object is within the viewing frustum, by computing the six planes of the viewing frustum from the initial camera view, and checking whether the axis-aligned bounding boxes (A_j intersect with the area in between the planes. The bounding box centers can be computed from the minimum and maximum vertex position co-ordinate elements given by the glTF metadata.

Angle Check. The angle weight determines how far off an object is from the center of the user's FoV. This is done by calculating the angle between the camera's default forward vector and the "look at" vector (the vector from the camera's position to the center of an object). This is performed by the `LookAtAngle` function in line 13. The weights from the visiblity check and the angle check are then added together to compute the total score for each object. Finally, the objects are sorted by their scores in ascending order (line 16). The overall algorithm has complexity $O(J \log(J))$ due to the sorting step.

Algorithm 1 Object Scoring

Inputs: Set of objects in the scene $\mathcal{U}$, axis aligned bounding box A_j for each object $j \in \mathcal{U}$, camera position C_P, orientation C_O, and FoV parameters C_F.
Variables: score $score_j$ of object j, view frustum F
Outputs: Sorted list of objects U
$O \leftarrow \emptyset$
$F \leftarrow$ `ViewFrustum`(C_P, C_O, C_F)
for all j in $\mathcal{U}$ **do**
  if A_j intersects F **then** ▷ visibility check
    $score_j \leftarrow 0$
    $O \leftarrow O \cup j$
  else
    $score_j \leftarrow \pi$
  end if
  $\theta \leftarrow$ `LookAtAngle`$(C_P, C_O, A_j.center)$ ▷ angle check
  $score_j \leftarrow score_j + \theta$
end for
$U \leftarrow$ sorted list of $\{score_j\}$

5.3 Data Buffer Grouping

Setup. The Data Buffer Grouping module breaks down the single data buffer file from a WebXR scene into its constituent data buffers, so that the Object Scoring module can request object dependencies at a finer granularity. However, deciding which data buffers to group together is non-trivial because there are trade-offs between RTT and propagation time. For example, grouping all data buffers into a single file would cost only one RTT to retrieve from the server; however, this would result in a longer propagation time (due to constrained bandwidth) before rendering of any object could start (as in the default WebXR). On the other hand, creating J files from J data buffers would require a fresh RTT to retrieve each data buffer, but each file would have a short propagation latency, ensuring that objects could be independently downloaded and rendered.

Problem Formulation. We formalize this problem as follows. There are N data buffers and J objects in the scene. We seek to group data buffers into files. The scene construction from the WebXR metadata tells us $R_{ij} \in \{0, 1\}$, whether data buffer i is needed for object j. The problem is to determine the integer variables X_{ik} (whether data buffer i should be included in file k) and Y_{kj} (whether object j requires file k). The objective is to minimize the download time of the set of objects O visible within the FoV:

$$\sum_k \max_{j \in O} Y_{kj} \left(\frac{s_k}{B} + T \right) \tag{1}$$

where s_k is the size of file k, B is the current bandwidth, and T is the current RTT/2. The second term $\left(\frac{s_k}{B} + T\right)$ is the download time of file k, so $Y_{kj} \left(\frac{s_k}{B} + T\right)$ is non-zero only if object j is dependent on file k. The $\max_{j \in O}$ term accounts for browser caching within the same session; *i.e.*, we only need to count the latency of one download of file k, if file k needed by more than one object j.

If we knew O in advance, then we could optimize the data buffer grouping for those objects within the FoV. However, in reality, O could be anything, since the user's initial position and orientation could be set arbitrarily. It wouldn't be scalable to create and store

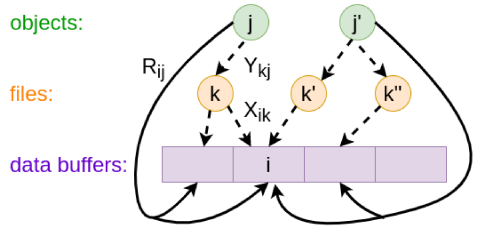


Figure 5: Problem 1 can be viewed as a series of set cover (Y_{kj}, mapping objects to files) and set membership (X_{ik}, mapping files to data buffers) problems.

files for every possible user position/orientation. Instead, we can try to optimize for a typical case, by minimizing the average time of all files, so that no matter the user's position/orientation, the relevant files can be retrieved quickly. Mathematically, we write:

$$\sum_k \max_{j \in O} Y_{kj} \left(\frac{s_k}{B} + T \right) \leq \sum_k \sum_{j \in \mathcal{U}} Y_{kj} \left(\frac{s_k}{B} + T \right) \tag{2}$$

where the LHS is (1) and the RHS is an upper bound where the maximum is replaced with a summation based on the intuition above. Using the RHS as our objective, the optimization problem is as follows.

Problem 1. *Data buffer grouping*

$$\underset{X_{ik}, Y_{kj}}{\textit{minimize}} \quad \sum_{j \in \mathcal{U}} t_j \tag{3}$$

$$\textit{subject to} \quad t_j = \sum_k Y_{kj} \left(\frac{s_k}{B} + T \right) \quad \forall j \tag{4}$$

$$s_k = \sum_i \bar{s}_i X_{ik} \quad \forall k \tag{5}$$

$$\sum_k X_{ik} Y_{kj} \geq R_{ij} \quad \forall i, j \tag{6}$$

$$X_{ik}, Y_{kj} \in \{0, 1\} \quad \forall i, j, k \tag{7}$$

The objective (3) is equivalent to the RHS of (2), and minimizes the average time of retrieving all objects. Constraint (4) defines an object's download time as the sum of the transmission delay and the RTT. Constraint (5) defines the file size as the sum of its constituent data buffers. Constraint (6) states that every object must receive all its required data buffers. X_{ik} and Y_{kj} are the integer decision variables.

Solution. This is an integer linear program, which is generally NP-hard. The problem can also be thought of as a variant of set cover, where both the subset membership and multiple set covers have to be determined. Namely, we have to determine the subset memberships (X_{ik} (which data buffers i should be included in subset k), and then solve J set cover problems, one for each object j (Y_{kj} (which subsets object j needs to cover all the data buffers in its own universe, $\{R_{ij}\}_i$). This is illustrated in Fig. 5.

However, it turns out that $X = R, Y = I$ is an optimal solution to the problem (where X, Y, R are the matrix versions of X_{ik}, Y_{kj}, R_{ij}, and I is the identity matrix). This solution corresponds to an easy-to-implement grouping of placing all data buffers of an object into a single file. Intuitively, it is possible to find a solution because

the X_{ik} variable gives the ability to determine subset membership, actually making the set cover problem easier. The main idea behind the proof is that by requesting only one copy of each data buffer, and incurring as few RTTs as possible (one request per object), then the average latency per object is minimized. The proof is provided in the technical report [Slocum and Huang 2021].

Proposition 1. *$X = R, Y = I$ is an optimal solution to Problem 1.*

The algorithm is shown in Alg. 2, and runs in $O(JN)$.

Algorithm 2 Data Buffer Grouping

Inputs: Whether object i needs data buffer j R_{ij}
Outputs: Whether file k includes data buffer i X_{ik}, whether object j requests file k Y_{kj}
for all $j < J$ **do**
  for all $i < N$ **do**
    if $R_{ij} == 1$ **then** ▷ object j needs data buffer i
      $X_{ij} \leftarrow 1$ ▷ store data buffer i in file j
      $Y_{jj} \leftarrow 1$ ▷ object j requests file j
    end if
  end for
end for

Image re-ordering only. We also developed a simplified version of VIA based on observations from certain test scenes where the total data buffer size was much less than the total size of the textures and images in the scene. In such cases, the data buffers did not impact the latency much, since the images consumed most of the network time. Therefore, we introduce a simplified version of VIA where only the images are re-ordered according to the object scores, but not the data buffers (essentially, running Alg. 1 but not Alg. 2). We call this method VIA-Image.

6 VIA'S IMPLEMENTATION

VIA is implemented in approximately 700 lines of Python3 and runs once per scene, when it is first placed on the server. It can be run multiple times for different initial FoV's expect from client devices. The output of the script is the new data buffer files, new metadata files. The WebGL library and a customized Javascript file (containing the new request order) are also stored on the server. A user wishing to reduce WebXR latency simply requests the new Javascript file with an unmodified web browser, without needing to make any other changes on the client side. Below, we briefly overview the key implementation steps.

Parsing object dependencies. We developed a custom parser for glTFs, which is a JSON-like object, and recorded the objects and their dependent textures, normal maps, and data buffers into a dictionary data structure. Parsing the axis-aligned bounding-boxes for Alg. 1 required finding all bufferViews associated with an object, and finding the minimum and maximum values of the vertex coordinates along each axis. If any transformations are performed on the geometry data in the glTF, this also needs to be taken into account before computing the axis-aligned bounding boxes. All glTF parsing and alteration was performed based on the glTF 2.0 standard [The Chronos Group 2021].

Creating new (.glTF) metadata. Alg. 1 is run to score the 3D objects, and using the scores, the images are re-ordered within the dictionary. For VIA-Image, this new dictionary is immediately written to file as a new .gltf. For the full VIA, Alg. 2 is also run and the original dictionary is split into multiple dictionaries, one for each object. Then new .gltf metadata files are written, one for each object. The glTF files are given a suffix in the file name to denote what order they should be requested in. The buffer attribute in each new glTF points to the URI of the relevant data buffer file (.bin).

We also experimented with creating one combined metadata file for all the re-ordered objects (instead of one metadata file for each object), but this resulted in all the binaries being requested first followed by all the images, due to the default behavior of the glTF loader, thus destroying our object ordering. Therefore we settled one glTF per object; however, a disadvantage was this resulted in a "flattening" of the object hierarchy stored in the original metadata file, which we plan to address in future work.

Creating new data buffer (.bin) files. The byte ranges for the data buffer associated with each object are parsed from the original glTF, and then copied over and combined into the new .bin files in order of their parent object score. Special care must be taken to maintain byte alignment (4, 8, or 16 byte-aligned) in order to allow for efficient processing of the contained data.

WebXR scene alterations: For VIA-Image, no additional changes to the Javascript code are needed. For the full VIA, a short loop in the Javascript code must be added to request each glTF in the score order. This is currently done manually in 4 lines of code, and is potentially automatable in the future.

Priority hints. An initial problem existed even with the re-ordered image requests, by default, the Chrome browser automatically tagged images with "low" priority and the .bin files containing the data buffers as "high" priority [Google Developers 2019], thereby over-writing our careful ordering. To overcome this, we had to explicitly tag all the glTFs, data buffer binaries, and images with the same "low" priority by altering 2 lines of Javascript in the glTF loader. Note that setting these priorities alone would not enable implementation of our scheme, as there are only two priority levels, dis-allowing fine-grained re-ordering.

7 EXPERIMENTS

We performed experiments to show the performance of VIA. The latency improvement depends on the user's viewpoint, network conditions, and characteristics of the scene itself. We also show that our methods are robust to small changes in initial orientation without needing to re-compute the object request order.

7.1 Setup

Experiments were performed on Google Chrome, version 91.0.4472.124, for different algorithms, test scenes, and network conditions. The three algorithms were tested were:

- **Control**: The default operation of WebXR.
- **VIA-Image**: VIA with Alg. 1 only, so only image requests were re-ordered, not data buffers.

- **VIA**: VIA with Algs. 1 and 2, where the scene objects and their image and data buffer dependencies were scored, sorted and requested in order.

Our initial experiments were performed on four complex scenes with varying sizes, number of objects, and sizes of binaries and image files. We created three of the scenes based on publicly available 3D models, and the fourth scene is a WebXR test case.

- **Solar System** [Sol 2018]: A simple, open space with an extremely high image to .bin size (15.5:1 ratio). The scene contains 29 objects with a total size of 6.72MB.
- **Future House** [Fut 2016]: An enclosed indoor area with a moderate image to .bin size (2.9:1 ratio). The scene contains 27 objects with a total size of 55MB.
- **Bayou Shack** [Bay 2021]: An outdoor scene with high image to .bin ratio (6:1). The scene contains 425 objects with a total size of 38.8MB.
- **Sponza** [spo 2019]: A walled-in area with a medium image to .bin ratio (4.39:1). This is a sample WebXR scene. The scene contains 103 objects with a total size of 50.2MB.
- **City** [Cit 2021]: An outdoor cityscape without any images, only binaries. The scene contains 615 objects with total size 56MB.

We tested on three simulated network conditions indicative of mobile networks: 30 Mbps with a 30ms RTT, 10 Mbps with a 60 ms RTT, and the "Fast3G" preset in Google Chrome DevTools, which roughly corresponds to 1.44 Mbps and 12 ms RTT based on an Internet speed test. All results were averaged across 3 trials for each Network-Scene-Algorithm combination, for a total of 135 experiments for the main results. The standard deviation in terms of latency across trials was less than 1%, and so are not shown.

Measuring latency (time to first correct frame): The main evaluation metric was the latency from the page reload time to when all objects in FoV were loaded. Caching was disabled across trials. To measure this, we used the load time of the last object to show up in the FoV. By experimentally comparing with screen recordings we captured, we verified that the last object's load time corresponds very closely to its actual display time, with the rendering latency being negligible on the order of a few ms. We developed this methodology because existing page load time tools such as Lighthouse [goo 2021] do not capture the desired latency. For example, for the Sponza scene, Lighthouse reported a Largest Contentful Paint of 0.7 s, which only corresponded a system menu appearing, while in reality the entire scene was not view-able for 9.86 s.

7.2 Overall performance

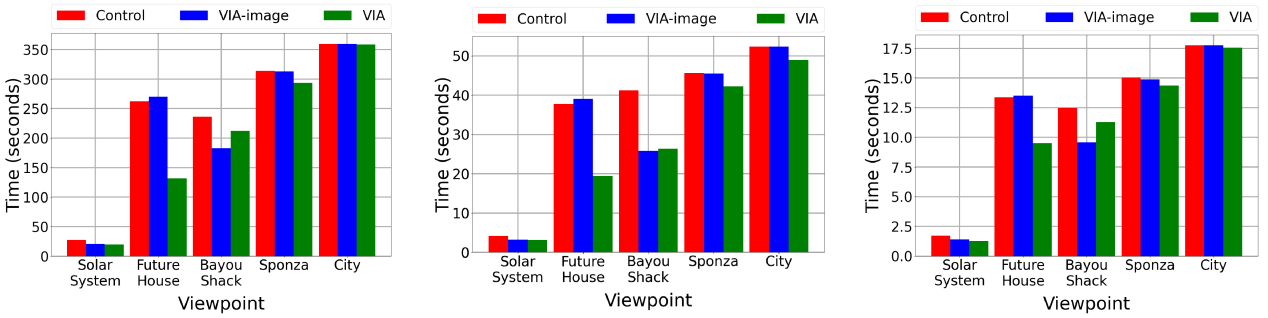
Performance of VIA. Figure 6b shows the latency of each scene with a 10 Mbps connection. While VIA had strictly lower latency than the control in all scenarios, the largest gains occurred in scene with the fewest bytes, with Solar System, Future House and Shack saving 26.08%, 48.52% and 36.17% of time compared to Control, respectively. Sponza and City saw the least improvement due to a large number of visible objects in the FoV, as well as the floor beneath the camera having a moderate object score and thus loading near the end of the trace, delaying the time to correct frame. In fact, Sponza, due to its scene structure, required 88% of all bytes to be loaded before a correct frame could be rendered, and thus is a challenging test case.

Performance of VIA-*Image.* VIA-Image showed some gains as well, significantly beating the control (going from ~40 to ~25 seconds) for Bayou Shack , the scene with the second highest image size to binary size ratio. This is because fetching Bayou Shack's large images in the correct order saved significant amounts of time. However, for all scenes besides Bayou Shack, the last resource to finish downloading was typically the large binary file (for Control and VIA-Image), meaning that all the objects had to wait for the binary to finish downloading before they were rendered all at once. In other words, the binary was the bottleneck, and hence the full VIA with data buffer grouping was needed. The City scene was particularly challenging, because the majority of the objects were within view (only 50 objects were culled by the visibility check).

Example screenshots. Figure 7 shows screenshots of the Sponza scene loading for the 10Mbps results discussed above. VIA is able to render in the first 3D objects at the five second mark, while the Control and VIA-Image are unable to render a single 3D object until the 45 second mark. VIA was able to render a partial scene after 20 s. This is due to the binary file splitting allowing rendering of individual objects earlier in VIA, rather than waiting the original single, large binary file. Control finishes a fraction of a second later than VIA-Image (bottom row), due to it downloading the lion's head textures (at the end of the corridor) last, whereas VIA-Image downloads it early on and thus is able to render a complete frame as soon as the binary finishes downloading.

Large objects. One particularly challenging type of scene is those with large objects whose bounding boxes cover nearly the entirety of the scene. In such cases, poor object scoring may cause bottlenecks in the latency for VIA, such as in Sponza or City. For example, objects may pass the view frustum check yet the center of their bounding boxes are located behind the user (*e.g.*, floors or building walls), resulting in a low to medium request priority from the Object Scoring module, despite them being visible in the FoV. On the other hand, other objects may pass the view frustum check with their centers directly in front of the user, thus receiving a higher priority from the Object Scoring module, despite them not being visible in the FoV because they are discontinuous (*e.g.*, two windows on both sides of a hallway). Scenes without large overlapping bounding boxes, such as Solar System, do not have this issue and avoid these potential bottlenecks.

System overheads. Here we briefly discuss the system overhead of running VIA. One overhead is in terms of storage – a side effect of splitting the large .bin of the original scene into many smaller .bins results in a small storage overhead. For the four test scenes, the average space overhead was an additional 3.3%, which is minimal. In terms of runtime, the main Python script executed in at most 16 seconds on an Intel i7-10700K CPU @ 3.80 Ghz, with over 15 seconds of that consumed by file I/O (*e.g.*, reading and writing the new data buffer files). For extremely large scenes in the future, view frustum culling may be accelerated with common space partitioning structures like octrees [Wilhelms and Van Gelder 1992] or K-d trees. Since the script only has to run once per WebXR scene, when it is



(a) Time to first correct frame under Fast 3G network conditions.

(b) Time to first correct frame under 10 Mbps and 60 ms RTT.

(c) Time to first correct frame under 30 Mbps and 30 ms RTT network conditions.

Figure 6: Time to first correct frame for different scenes under different network conditions. VIA improves load time by up to 50%, with greater improvement in scenes where there are fewer objects in the FoV.

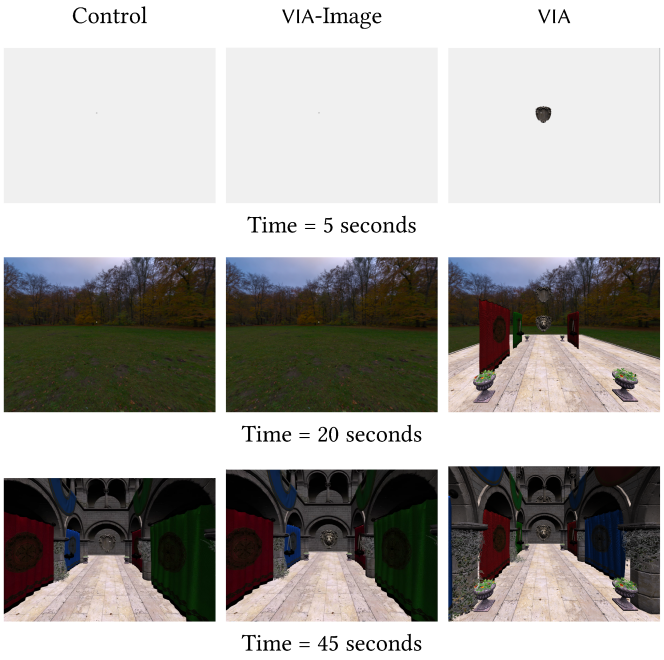


Figure 7: Screenshots of the Sponza WebXR scene at three different time instances, for the Control, VIA-Image, and VIA methods. VIA loads objects into the user FoV the fastest, while VIA-Image provides some benefits over Control.

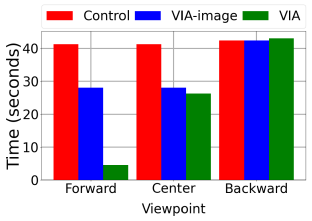


Figure 8: Time to first correct frame from different viewpoints, for Bayou Shack. The improvement of VIA over Control depends on the viewpoint.

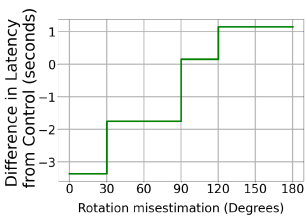


Figure 9: VIA Improvement of VIA over Control for the Sponza scene, if the system mis-estimates the user's head orientation.

first saved to the server, we consider this runtime acceptable and did not implement acceleration structures.

7.3 Impact of different viewpoints

The above experiments were conducted at the default initial user viewpoint. In this set of experiments, we examined the impact of different user viewpoints on performance. In particular, we studied the Bayou Shack scene at 10 Mbps in greater detail to show the extreme impact that different initial viewpoints can have on time to first correct frame. Three viewpoints were chosen: a viewpoint at the edge of the scene with only two objects in view (labeled as "forward"), a viewpoint in the center of the scene, similar to the earlier experiments (labeled as "center"), and a viewpoint near the edge of the scene looking inwards with nearly all objects visible (labeled as "backward").

Figure 8 show the latency of Bayou Shack at these viewpoints. The greatest gains are achieved by VIA in the "forward" viewpoint, because there are only two objects in the FoV that need to be downloaded, and thus the remaining 423 objects can be loaded afterwards. The "backward" viewpoint is the worst case scenario, as very little latency savings are possible due to all objects in the scene being visible. The results show that if all the resources are needed to render a frame correctly, VIA can actually perform slightly worse than Control, due to the overhead incurred by extra RTTs to the server for each additional object (metadata and binary file) request. However, when few resources are needed, the gains can be substantial, around a 90% reduction in latency.

7.4 Impact of network conditions

The latency savings with the various methods depends on the network conditions, as shown in Figures 6b, 6a, and 6. When increasing the bandwidth from Fast 3G to 10Mbps to 30Mbps, VIA and VIA-Image were able to outperform Control, with the magnitude of those improvements varying greatly; for example, going from approximately 125 to 20 to 9 seconds as network speeds increased for the Future House scene.

In general, the latency improvements of VIA over Control are inversely proportional to the network bandwidth (*i.e.*, greater improvement at slower speeds, which is exactly where we need the most improvement). This is because as network bandwidth increases, the overhead from extra RTTs (from the individual object requests) gradually dominates the total latency, while the propagation time shrinks. Note that as network speeds change, the relative improvement of VIA over Control may change (such as in Future House from 10Mbps to 30Mbps). Based on our observation that the last object to appear in the FOV depends on the network conditions, we hypothesize this is because modern browsers typically make parallel resource requests, so while the total latency tends to reflect the network speed, it may not be perfectly proportional due to changes in download completion order and other overheads.

We also examined performance under a fast wired network connection (250 Mbps with 30ms RTT). In those experiments, the fast data transfer speeds resulted in the scenes loading very quickly (2.5 s on average for Control), so the extra RTTs from VIA method dwarfed the overall download time for the scene (approximately 4 times as long on average). Consequently, we do not recommend using VIA for extremely high speed connections (250Mbps+), as the gains from the algorithms are minimized and the extra overhead from the additional object requests could result in worse performance than Control. However, in practical use cases, the VR devices are wireless, and so the network speeds would not be as fast.

7.5 Impact of FoV orientation mis-estimate

Our last set of experiments examined the impact of user orientation changes. Due to user movement during the page load time or incorrect orientation estimates from the VR device, it is possible for the viewpoint at the time of first correct frame to be different from the original viewpoint input to VIA. We performed experiments to show that our methods are robust to slight changes in the user's viewpoint. We loaded the Sponza scene and recorded the time to first correct frame with a user making a yaw rotation (turning left to right) from 0° to 180°, at intervals of 15°. However, the objects downloaded by VIA still used the object scores from the mis-estimated 0° rotation.

On the x-axis of Fig. 9, we plot the amount of yaw rotation, and on the y-axis, we plot the latency differences between VIA and Control (a negative number indicates an improvement over Control). The resulting plot has discrete jumps because the latency only changes when the user has rotated enough that an object that was not previously visible comes into view, or vice versa. The main observation is that the VIA method continues to outperform the Control, unless the user looks more than 90° away during the page load time. This indicates that slight deviations from the user's initial viewpoint don't significantly affect the performance of VIA.

8 CONCLUSIONS

This paper is the first study page load times for WebXR-based VR scenes. Upon measuring the performance of the default WebXR, we observed that the startup latency until all the 3D objects was quite high, around 10 seconds on a 60 Mbps connection. Motivated by this, we developed methods of fine-grained splitting and requests of the objects within the user's FoV. Experiments performed on a working prototype indicated savings of more than 90% on some test scenes, depending on the scene structure (number of objects in the FoV). Future work includes generalizing our techniques to other WebGL-based applications, and working with scenes with more complex object hierarchies.

ACKNOWLEDGMENTS

This work has been supported in part by NSF CAREER 1942700 and a Google exploreCSR grant.

REFERENCES

2016. Future House. https://xeogl.org/examples/#importing_gltf_BranchHouse. Accessed: 2021-06-27.
2018. Solar System. https://sketchfab.com/3d-models/solar-system-b6b69a95a6f0426bb8bbc2e8cb7ff46a. Accessed: 2021-08-01.
2019. Sponza. https://immersive-web.github.io/webxr-samples/tests/sponza.html.
2021. City Grid Block. https://sketchfab.com/3d-models/city-grid-block-3488e40ceca846bb9023f894a749c398. Accessed: 2021-06-27.
2021. Forest Loner. https://sketchfab.com/3d-models/forest-loner-b914786647d54b979bc7021e69c0db2e. Accessed: 2021-06-27.
2021. Google Lighthouse. https://developers.google.com/web/tools/lighthouse.
Ulf Assarsson and Tomas Moller. 2000. Optimized View Frustum Culling Algorithms for Bounding Boxes. *Journal of Graphics Tools* 5 (07 2000). https://doi.org/10.1080/10867651.2000.10487517
Kevin Boos, David Chu, and Eduardo Cuervo. 2016. Flashback: Immersive virtual reality on mobile devices via rendering memoization. *ACM MobiSys* (2016).
Michael Butkiewicz, Daimeng Wang, Zhe Wu, Harsha V Madhyastha, and Vyas Sekar. 2015. Klotski: Reprioritizing web content to improve user experience on mobile devices. In *USENIX NSDI*. 439–453.
Cesium. 2017. *Cesium 3D Tileset.* https://github.com/CesiumGS/3d-tiles
Xavier Corbillon, Gwendal Simon, Alisa Devlic, and Jacob Chakareski. 2017. Viewport-adaptive navigable 360-degree video delivery. In *IEEE ICC*. IEEE, 1–7.
Thomas Forgione, Axel Carlier, Géraldine Morin, Wei Tsang Ooi, Vincent Charvillat, and Praveen Kumar Yadav. 2018. DASH for 3D networked virtual environment. In *ACM Multimedia*. 1910–1918.
Google Developers. 2019. Get Ready for Priority Hints. https://developers.google.com/web/updates/2019/02/priority-hints.
Yu Guan, Chengyuan Zheng, Xinggong Zhang, Zongming Guo, and Junchen Jiang. 2019. Pano: Optimizing 360 video streaming with a better understanding of quality perception. In *ACM SIGCOMM*. 394–407.
Bo Han, Yu Liu, and Feng Qian. 2020. ViVo: Visibility-aware mobile volumetric video streaming. In *ACM MobiCom*. 1–13.
Hristina Hristova, Gwendal Simon, Viswanathan Swaminathan, and Stefano Petrangeli. 2020. 3CPS: a novel supercompression for the delivery of 3D object textures. In *ACM MMSys*. 66–76.
S-Y Hu, T-H Huang, S-C Chang, W-L Sung, J-R Jiang, and B-Y Chen. 2008. Flod: A framework for peer-to-peer 3d streaming. In *IEEE INFOCOM*. IEEE, 1373–1381.
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 *ACM Web3D*.
Guillaume Lavoué, Laurent Chevalier, and Florent Dupont. 2013. Streaming Compressed 3D Data on the Web Using JavaScript and WebGL. In *ACM Web3D*.
Max Limper, Stefan Wagner, Christian Stein, Yvonne Jung, and André Stork. 2013. Fast delivery of 3D Web content: A case study. *ACM Web3D*, 11–17.
Duckkyoun Nam, Daehyeon Lee, Seunghyun Lee, and Soonchul Kwon. 2019. A Comparative Study on 3D Data Performance in Mobile Web Browsers in 4G and 5G Environments. *International Journal of Internet, Broadcasting and Communication* 11, 3 (2019), 8–19.
Ravi Netravali, Ameesh Goyal, James Mickens, and Hari Balakrishnan. 2016. Polaris: Faster Page Loads Using Fine-grained Dependency Tracking. *USENIX NSDI* (2016).
Feng Qian, Bo Han, Qingyang Xiao, and Vijay Gopalakrishnan. 2018. Flare: Practical viewport-adaptive 360-degree video streaming for mobile devices. In *ACM MobiCom*. 99–114.
Arne Schilling, Jannes Bolling, and Claus Nagel. 2016. Using glTF for streaming CityGML 3D city models. *ACM Web3D*, 109–116.
Carter Slocum and Jingwen Huang. 2021. VIA GitHub Repository. https://github.com/mavens-lab/webxr_latency.
The Chronos Group. 2021. The glTF standard. https://www.khronos.org/gltf/.
W3C. 2021. WebXR Device API. https://www.w3.org/TR/webxr/.
Jane Wilhelms and Allen Van Gelder. 1992. Octrees for Faster Isosurface Generation. *ACM Trans. Graph.* 11, 3 (July 1992), 201–227. https://doi.org/10.1145/130881.130882
Chao Zhou, Mengbai Xiao, and Yao Liu. 2018. Clustile: Toward minimizing bandwidth in 360-degree video streaming. In *IEEE INFOCOM*. IEEE, 962–970.

APPENDIX

Proposition 1

Proof. The objective function (3) can be written as

$$\frac{1}{B}\sum_i\sum_j\sum_k X_{ik}Y_{kj}\bar{s}_i + T\sum_j\sum_k Y_{kj} \tag{8}$$

We will show that with $X = R$ and $Y = I$, the first and second terms are individually minimized, and hence their sum is also minimized.

First term in in (3): Using constraint (6), we can write:

$$\frac{1}{B}\sum_i\sum_j\sum_k X_{ik}Y_{kj}\bar{s}_i \geq \frac{1}{B}\sum_i\sum_j R_{ij}\bar{s}_i \tag{9}$$

where the LHS is the first term in (3) and the RHS is a tight lower bound (otherwise the solution would be infeasible). If $X = R, Y = I$, then the first term is $\frac{1}{B}\sum_i\sum_j\sum_k X_{ik}Y_{kj}\bar{s}_i = \frac{1}{B}\sum_i\sum_j\sum_k R_{ik}Y_{kj}\bar{s}_i = \frac{1}{B}\sum_i\sum_j R_{ij}\bar{s}_i$, achieving that lower bound.

Second term in (3): We claim that $T\sum_j\sum_k Y_{kj} \geq TJ$ is a tight lower bound on the second term in (3), and since $Y = I$ satisfies this lower bound, it minimizes the second term. To see the bound, assume that $\sum_j\sum_k Y_{kj} < J$. This implies $\exists\ j$ such that:

$$\sum_k Y_{kj} = 0 \tag{10}$$

$$\sum_i\sum_k X_{ik}Y_{kj} = 0 \tag{11}$$

$$\sum_i R_{ij} = 0 \tag{12}$$

which contradicts the assumption that an object requires at least one asset (otherwise we could just remove it from the problem). □