Impact of 3D Bookmarks on Navigation and Streaming in a Networked Virtual Environment

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

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

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

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

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

ABSTRACT

A 3D bookmark in a networked virtual environment (NVE) provides a navigation aid, allowing the user to move quickly from its current viewpoint to a bookmarked viewpoint by simply clicking on the bookmark. In this paper, we first validate the positive impact that 3D bookmarks have in easing navigation in a 3D scene. Then, we show that, in the context of a NVE that streams content on demand from server to client, navigating with bookmarks leads to lower rendering quality at the bookmarked viewpoint, due to lower locality of data. We then investigate into how prefetching the 3D data at the bookmarks and precomputation of visible faces at the bookmarks help to improve the rendering quality.

CCS Concepts

•Information systems → Multimedia streaming; •Computing methodologies → Virtual reality;

Keywords

3D Bookmarks, 3D Navigation Aid, Networked Virtual Environment, Prefetching, 3D Streaming

1. INTRODUCTION

With the progress in data acquisition and modeling techniques, networked virtual environments, or NVE, are increasing in scale. For instance, Gaillard et al. [12] reported that the 3D scene for the city of Lyon takes more than 30 GB of data. It has become impractical to download the whole 3D scene before the user begins to navigate in the scene. A more common approach is to stream the required 3D content (models and textures) on demand, as the user moves around the scene. Downloading the required 3D content the moment the user demands it, however, leads to "popping effect" where 3D objects materialize suddenly in the view of the user, due to the latency between requesting for and receiving the 3D content from the server [26]. Such latency can be quite high – Varvello et al. reported a median of about 30 seconds for all 3D data in an avatar's surrounding to be loaded in high density Second Life regions under their experimental network conditions, due to a bottleneck at the server [28].

For a smoother user experience, NVE typically prefetch 3D content, so that a 3D object is readily available for rendering when the object falls into the view of the user. Efficient prefetching, however, requires the client or the server to predict where the user would navigate to in the future and retrieve the corresponding 3D content before the user reaches there. In a typical scenario, users navigate along a continuous path in a NVE, leading to a significant overlap between the 3D content visible from the user's known current position and possible next positions (i.e., *spatial data locality*). Furthermore, there is a significant overlap between the 3D content visible from the current point in time to the next point in time (i.e., *temporal data locality*). Both forms of locality lead to content overlaps, thus making a correct prediction easier and a wrong prediction less costly. 3D content overlaps are particularly common in a NVE with open space, such as a 3D archaeological site or a 3D city.

Navigating in NVE with a large virtual space (most times through a 2D interface) is sometimes cumbersome. In particular, a user may have difficulties reaching the right place to find information. The content provider of the NVE may want to highlight certain interesting features for the users to view and experience, such as a vantage point in a city, an excavation at an archaeological site, or an exhibit in a museum. To allow users to easily find these interesting locations within the NVE, *3D bookmarks* or *bookmarks* for short, can be provided. A bookmark is simply a 3D virtual camera (with position and camera parameters) predefined by the content provider, and can be presented to users in different ways, including as a text link (URL), a thumbnail image, or a 3D object embedded within the NVE itself.

When users click on a bookmark, NVEs commonly provide a "fly-to" animation to transit the camera from the current viewpoint to the destination [21, 24] to help orient the users within the 3D space. Clicking on a bookmark to fly to another viewpoint leads to reduced data locality. The 3D content at the bookmarked destination viewpoint may overlap less with the current viewpoint. In the worst case, the 3D objects corresponding to the current and destination viewpoints can be completely disjoint. Such movement to a bookmark may lead to a *discovery latency* [28], in which users have to wait for the 3D content for the new viewpoint to be loaded and displayed. An analogy for this situation, in the context of video streaming, is seeking into a segment of video that has not been prefetched yet.

MMSys'16, May 10-13, 2016, Klagenfurt, Austria
© 2016 ACM. ISBN 978-1-4503-4297-1/16/05. . . $15.00
DOI: http://dx.doi.org/10.1145/2910017.2910607

In this paper, we explore the impact of bookmarks on NVE navigation and streaming, and make several contributions. First, we conducted a crowdsourcing experiment where 51 participants navigated in 3 virtual scenes to complete a task. This experiment serves two purposes: (i) it validates our intuition that bookmarks significantly reduce the number of interactions and navigation time (in average the time needed to complete the task for users with bookmarks is half the time for users without bookmarks); (ii) it produces a set of user interaction traces that we use for subsequent simulation experiments. Second, we quantified the effect of bookmarking on prefetching and visual quality in our experiments. We showed that, without prefetching, the number of correctly rendered pixels right after clicking on bookmarks can drop up to 10% on average. If we prefetch the 3D content from the bookmarks according to the probability of access, we do not limit this drop by more than 5%. Finally, we proposed a method to improve the visual quality after clicking on bookmarks, by exploiting the fact that the visible faces at the bookmark can be precomputed, and by fetching the visible faces only after a bookmark is clicked. We showed that, if the fetching is done during the 1-2 seconds of the "fly-to" camera movement from the current viewpoint to the bookmarked viewpoint, it suffices to increase the number of correctly rendered pixels to more than 20%, without wasting bandwidth on prefetching. Our key message is that, *in addition to easing navigation, bookmarking allows precomputation of visible faces and can significantly reduce interaction latency, without resorting to prefetching*, which may waste bandwidth by prefetching 3D data that will not be needed.

The rest of the papers consists of the following sections. Section 2 discusses the related work in 3D navigation and prefetching. This is followed by Section 3, which describes the 3D bookmarks that we use in our work, along with our experiments to validate the usefulness of bookmarking. Section 4 describes the streaming and prefetching mechanisms that we used to simulate our experiments as well as our main findings. Finally, we conclude in Section 5.

2. RELATED WORK

2.1 3D Bookmarks and Navigation Aids

Devising an ergonomic technique for browsing 3D environments through a 2D interface is difficult. Controlling the viewpoint in 3D (6 DOFs) with 2D devices is not only inherently challenging but also strongly task-dependent. In their recent review [16], Jankowski and Hachet distinguish between several types of camera movements: general movements for exploration (e.g., navigation with no explicit target), targeted movements (e.g., searching and/or examining a model in detail), specified trajectory (e.g., a cinematographic camera path), etc. For each type of movement, specialized 3D interaction techniques can be designed. In most cases, rotating, panning, and zooming movements are required, and users are consequently forced to switch back and forth among several navigation modes, leading to interactions that are too complicated overall for a layperson. Navigation aids and smart widgets are required and subject to research efforts both in 3D companies (see sketchfab.com, cl3ver.com among others) and in academia, as reported below.

Translating and rotating the camera can be simply specified by a *lookat* point. This is often known as point-of-interest movement (or *go-to*, *fly-to* interactions) [21]. Given such a point, the camera automatically animates from its current position to a new position that looks at the specified point. One key issue of these techniques is to correctly orient the camera at destination. In Unicam [29], the so-called click-to-focus strategy automatically chooses the destination viewpoint depending on 3D orientations around the contact point. The recent Drag'n Go interaction [22] also hits a destination point while offering control on speed and position along the camera path. This 3D interaction is designed in the screen space (it is typically a mouse-based camera control), where cursor's movements are mapped to camera movements following the same direction as the on-screen optical-flow.

Some 3D browsers provide a viewpoint menu offering a choice of viewpoints [27], [3]. Authors of 3D scenes can place several viewpoints (typically for each POI) in order to allow easy navigation for users, who can then easily navigate from viewpoint to viewpoint just by selecting a menu item. Such viewpoints can be either static, or dynamically adapted: the authors from [15] report that users clearly prefer navigating in 3D using a menu with animated viewpoints than with static ones.

Early 3D VRML environments [24] offer 3D bookmarks with animated transitions between bookmarked views. These transitions prevent disorientation since users see how they got there. Hyperlinks can also ease rapid movements between distant viewpoints and naturally support non-linear and non-continuous access to 3D content. Navigating with 3D hyperlinks is potentially faster, but is likely to cause disorientation, as shown by the work of Ruddle et al. [25]. Eno et al. [11] examine explicit landmark links as well as implicit avatar-chosen links in Second Life. These authors point out that linking is appreciated by users and that easing linking would likely result in a richer user experience. In [15], the Dual-Mode User Interface (DMUI) coordinates and links hypertext to 3D graphics in order to access information in a 3D space. Our results are consistent with the results on 3D hyperlinks, as we showed that in our NVE 3D bookmarks also improve users performance.

The use of in-scene 3D navigation widgets can also facilitate 3D navigation tasks. Chittaro and Venkataraman [10] propose and evaluate 2D and 3D maps as navigation aids for complex virtual buildings and find that the 2D navigation aid outperforms the 3D one for searching tasks. The ViewCube widget [17] serves as a proxy for the 3D scene and offers viewpoint switching between 26 views while clearly indicating associated 3D orientations. Interactive 3D arrows that point to objects of interest have also been proposed as navigation aids by Chittaro and Burigat [9, 2]: when clicked, the arrows transfer the viewpoint to the destination through a simulated walk or a faster flight. The 3D arrows update and reorientate as the user moves around. We use a similar technique in this paper.

2.2 Prefetching in NVE

We now present several related work on prefetching of 3D content in NVE. The general prefetching problem can be described as follows: what are the data most likely to be accessed by the user in the near future, and in what order do we download the data?

The simplest answer to the first question assumes that the user would likely access content close to the current position, thus would retrieve the 3D content within a given radius of the user (also known as the *area of interest*, or AoI). This approach, implemented in Second Life and several other NVEs (e.g., [20]), only depends on the location of the avatar, not on its viewing direction. It exploits spatial locality and works well for any continuous movement of the user, including turning. Once the set of objects that are likely to be accessed by the user is determined, the next question is in what order should these objects be retrieved. A simple approach is to retrieve the objects based on distance: the spatial distance from the user's virtual location and rotational distance from the user's view.

Other approaches consider the movement of the user and attempt to predict where the user will move to in the future. Chan et al. [6] and Li et al. [19] predict the direction of movement from the user's mouse input pattern. The predicted mouse movement direction is then mapped to the navigation path in the NVE. Objects that fall in the predicted path are then prefetched. CyberWalk [8] uses an exponentially weighted moving average of past movement vectors, adjusted with the residual of prediction, to predict the next location of the user.

Hung et al. [14] cluster the navigation paths of users and use them to predict the future navigation paths. Objects that fall within the predicted navigation path are prefetched. All these approaches work well for a navigation path that is continuous – once the user clicks on a bookmark and jumps to a new location, the path is no longer continuous and the prediction becomes wrong.

Moving beyond ordering objects to prefetch based on distance only, Park et al. [23] propose to predict the user's interest in an object as well. Objects within AoI are then retrieved in decreasing order of predicted interest value to the user. Zhou et al. [31] use another approach. Instead of predicting the location or movement of the user, they directly predict which objects will be accessed, through learning from the object access patterns.

2.3 Prefetching in Other Interactive Media

We briefly survey other research on prefetching that focuses on non-continuous interaction in other types of media.

In the context of navigating in a video, a recent work by Carlier et al. [4] prefetches video chunks located after bookmarks along the video timeline. Their work, however, focuses on changing the user behavior to improve the prefetching hit rate, by depicting the state of the prefetched buffer to the user. Carlier et al. also consider prefetching in the context of zoomable videos in an earlier work [5], and showed that predicting which region of videos the user will zoom into or pan to by analyzing interaction traces from users is difficult.

Prefetching for navigation through a sequence of short online videos is considered by Khemmarat et al in [18]. Each switch from the current video to the next can be treated as a non-continuous interaction. The authors proposed recommendation-aware prefetching – to prefetch the prefix of videos from the search result list and related video list, as these videos are likely to be of interest to the user and other users from the same community.

Grigoras et al. [13] consider the problem of prefetching in the context of a hypervideo; non-continuous interaction happens when users click on a hyperlink in the video. They propose a formal framework that captures the click probability, the bandwidth, and the bit rate of videos as a Markov decision problem, and derive an optimal prefetching policy.

Zhao and Ooi [30] propose Joserlin, a generic framework for prefetching that applies to any non-continuous media, but focuses on peer-to-peer streaming applications. They do not predict which item to prefetch, but rather focus on how to schedule the prefetch request and response.

There is a huge body of work on prefetching Web objects in the context of the World Wide Web. Interested readers can refer to numerous surveys written on this topic (e.g., [1]).

3. IMPACT OF 3D BOOKMARKS ON NAVIGATION

We now describe an experiment that we conducted on 51 participants, with two goals in mind. First, we want to measure the impact of 3D bookmarks on navigation within an NVE. Second, we want to collect traces from the users so that we can replay them for reproducible experiments for comparing streaming strategies in Section 4.

3.1 Our NVE

To ease the deployment of our experiments to users in distributed locations on a crowdsourcing platform, we implement a simple Web-based NVE client using Three.js[1]. The NVE server is implemented with node.js[2]. The NVE server streams a 3D scene to the client; the client renders the scene as the 3D content are received.

The user can navigate within the NVE in the following way; he/she can translate the camera using the arrow keys along four directions: forward, backward, to the left, and to the right. Alternatively, the keys W, A, S and D can also be used for the same actions. This choice was inspired by 3D video games, which often use these keys in conjunction with the mouse to move an avatar. The virtual camera can rotate in four different directions using the keys I, K, J and L. The user can also rotate the camera by dragging the mouse in the desired direction. Finally, following the UI of popular 3D games, we also give users the possibility to lock their pointer and use their mouse as a virtual camera. The mouse movement controls the camera rotation. The user can always choose to lock the pointer, or unlock it using the escape key. The interface also includes a button to reset the camera back to the starting position in the scene.

3.2 3D Bookmarks

Our NVE supports 3D bookmarks. A 3D bookmark, or bookmark for short, is simply a fixed camera location (in 3D space), a view direction, and a focal. Bookmarks visible from the user's current viewpoint are shown as 3D objects in the scene. Figure 1 depicts some bookmarks from our NVE.

The user can click on a bookmark object to automatically move and align its viewpoint to that of the bookmark. The movement follows a Hermite curve joining the current viewpoint to the viewpoint of the bookmark. The tangent of the curve is the view direction. The user can hover the mouse pointer over a bookmark object to see a thumbnail view of the 3D scene as seen from the bookmark. (Figure 1, bottom left).

In our work, we consider two different possibilities for displaying bookmarks: viewports (Figure 1 top left) and arrows (Figure 1 top right). A viewport is displayed as a pyramid where the top corresponds to the optical center of its viewpoint and the base corresponds to its image plane. The arrows are view dependent. The bottom of the arrow turns towards the current position, to better visualize the relative position of the bookmark.

Bookmarks allow the user to achieve a large movement within the 3D environment using a single action (a mouse click). As bookmarks are part of the scene, they are visible only when not hidden by other objects from the scene. We chose size and colors that are salient enough to be easily seen, but not too large to limit the occlusion of regions within the scene. When reaching the bookmark, the corresponding arrow or viewport is not visible anymore, and subsequently will appear in a different color, to indicate that it has been clicked (similar to Web links).

3.3 User Study

We now describe in details our experimental setup and the user study that we conducted on 3D navigation.

Models. We use four 3D scenes (one for the tutorial and three for the actual experiments) that represent recreated scenes from a famous video game. Those models are light (a few thousand of triangles per model) and are sent before the experiment starts. We keep the models small so that users can perform the task with ac-

[1]http://threejs.org
[2]http://nodejs.org

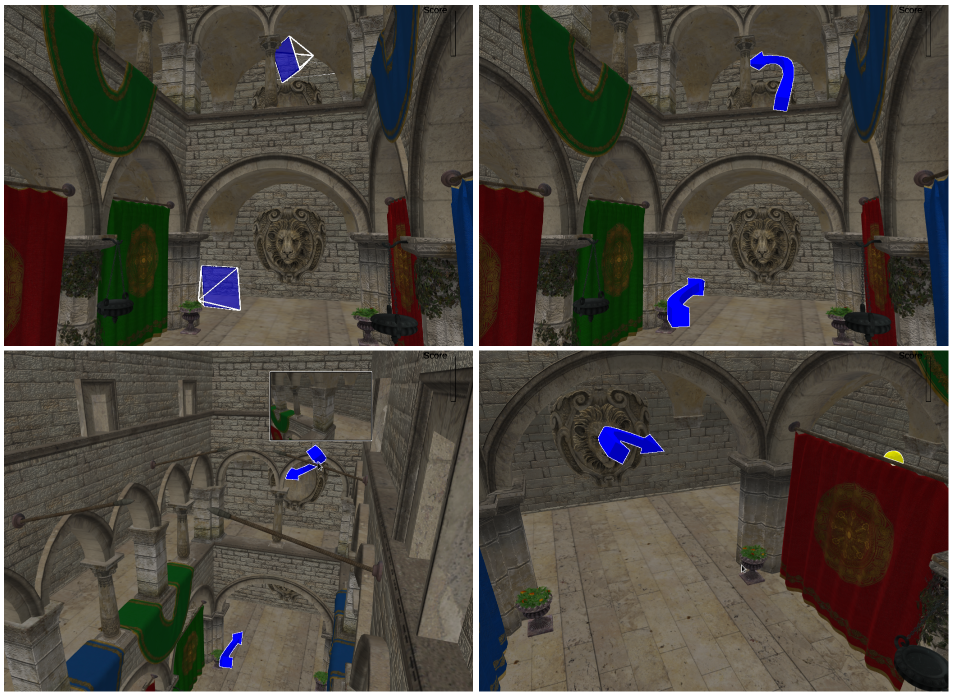

Figure 1: 3D bookmarks propose to move to a new viewpoint. A 3D bookmark may be displayed as a viewport (top-left) or as an arrow (top-right). Bottom-Left: a preview is shown when the user places the mouse on the bookmark; when the user clicks on the bookmark, his viewpoint moves to the indicated viewpoint. Bottom-Right: a global view with a bookmark and a coin hidden behind the curtain.

ceptable latency from any country using a decent Internet connection. Our NVE does not actually stream the 3D content for these experiments, in order to avoid unreliable conditions caused by the network bandwidth variation –which might affect how the users interact.

Task design. Since we are interested in studying how efficiently users navigate in the 3D scene, we ask our participants to complete a task that forces them to visit, at least partially, various regions in the scene. To this end, we hide a set of 8 coins on the scene: participants are asked to collect the coins by clicking on them. In order to avoid any bias due to the coins position, we predefined 50 possible coin locations all around the scene, and randomly select 8 out of these 50 positions each time a new participant starts the experiment.

Experiment. Participants are first presented with an initial screen to collect some preliminary information: age, gender, the last time they played 3D video games, and self-rated 3D gaming skills. We ask those questions because we believe that someone who is used to play 3D video games should browse the scene more easily, and thus, may not need to use our bookmarks.

Then, the participants go through a tutorial to learn how the UI works, and how to complete the task. The different interactions (keyboard navigation, mouse navigation, bookmarks interaction) are progressively introduced to participants, and the tutorial finishes once the participant completes an easy version of the task. The tutorial is always performed on the same scene.

Then, each participant has to complete the task three times. Each task is performed on a different scene, with a different interface. Three interfaces are used. A NoBM interface lets the participant navigates without any bookmarks. The other two interfaces allow a participant to move using bookmarks displayed as viewports (denoted as VP) and arrows (denoted as Ar) respectively.

The coins are chosen randomly, based on the coin configurations that were used by previous participants: if another participant has done an experiment with a certain set of coins, on a certain scene, with a certain type of bookmarks, the current participant will do the experiment with the same set of coins, on the same scene, but with a different type of bookmarks. This policy allows us to limit the bias that could be caused by coin locations.

Once a participant has found all coins, a button is shown on the interface to let the participant move to the next step. Alternatively, this button may appear one minute after the sixth coin was found. This means that a user is authorized to move on without completing the task, in order to avoid potential frustration caused by not finding the remaining two coins.

After completing the three tasks, the participants have to answer a set of questions about their experience with the bookmarks (we refer to the bookmarks as *recommendations* in the experiments). Table 1 shows the list of questions.

Participants. The participants were recruited on microworkers.com, a crowdsourcing website. There were 51 participants (36 men and 15 women), who are in average 30.44 years old.

3.4 Experimental Results

We now present the results from our user study, focusing on whether bookmarks help users navigating the 3D scene.

3.4.1 Questionnaire

We had 51 responses to the Questionnaire. The answers are summarized in Table 1. Note that not all questions were answered by all participants.

	Questions	Answers
1	What was the difficulty level WITHOUT recommendation?	3.04 / 5 ±0.31 (99% confidence interval)
2	What was the difficulty level WITH recommendation?	2.15 / 5 ±0.30 (99% confidence interval)
3	Did the recommendations help you to find the coins?	42 Yes, 5 No
4	Did the recommendations help you to browse the scene?	49 Yes, 2 No
5	Do you think recommendations can be helpful?	49 Yes, 2 No
6	Which recommendation style do you prefer and why?	32 Ar , 7 VP
7	Did you enjoy this ?	36 Yes, 3 No

Table 1: List of questions in the questionnaire and summary of answers.

The participants seem to find the task to be of average difficulty (3.04/5) when they have no bookmarks to help their navigation. They judge the task to be easier in average (2.15/5) with bookmarks, which indicates that bookmarks ease the completion of the task.

Almost all users (49 out of 51) think the bookmarks are useful for browsing the scene, and most users (42 out of 51) think bookmarks are also useful to complete the given task. This is slightly in contradiction with our setup; even if coins may appear in some bookmarked viewpoints (which is normal since the viewpoints have been chosen to get the most complete coverage of the scene), most of the time no coin is visible in a given bookmark, and there are always coins that are invisible from all bookmarks.

The strongest result is that almost all users (49 out of 51) find bookmarks to be helpful. In addition, users seem to have a preference for Ar against VP (32 against 7).

3.4.2 Analysis of Interactions

BM type	#Exp	Mean # coins	# completed	Mean time
NoBM	51	7.08	18	4:16 min
Ar	51	7.39	27	2:33 min
VP	51	7.51	30	2:16 min

Table 2: Analysis of the sessions length and users success by type of bookmarks

Table 2 shows basic statistics on task completion given the type of bookmarks that were provided to the participants.

First, we can see that without bookmarks, only a little bit more than a third of the users are able to complete the task, i.e. find all 8 coins. In average, these users find just above 7 coins, and spend 4 minutes and 16 seconds to do it.

Interestingly, and regardless of the bookmark type, users who have bookmarks complete the task more than half of the time, and spend in average significantly less time to complete the task: 2 minutes and 16 seconds using VP and 2 minutes and 33 seconds using Ar . Although VP seem to help users a little bit more in completing the task than Ar , the performance difference between both types of bookmarks is not significant enough to conclude on which type of bookmarks is best.

The difference between an interface with bookmarks and without bookmarks, however, is very clear. Users tend to complete the task more efficiently using bookmarks: more users actually finish the task, and it takes them half the time to do so. We computed 99% confidence intervals on the results introduced in Table 2. We found that the difference in mean number of coins collected with and without bookmarks is not high enough to be statistically significant: we would need more experiments to reach the significance. The mean time spent on the task however is statistically significant.

BM type	Total distance	Distance to a bookmark	Ratio
NoBM	610.80	0	0%
Ar	586.30	369.77	63%
VP	546.96	332.72	61 %

Table 3: Analysis of the length of the paths by type of bookmarks

Table 3 presents the length of the paths traveled by users in the scenes. Although users tend to spend less time on the tasks when they do not have bookmarks, they travel pretty much the same distance as without bookmarks. As a consequence, they visit the scene faster in average with bookmarks, than without bookmarks. The table shows that this higher speed is due to the bookmarks, as more than 60% of the distance traveled by users with bookmarks happens when users click on bookmarks and fly to the destination.

3.4.3 Discussion

In the previous paragraphs, we have shown how bookmarks are well perceived by users (looking at the questionnaire answers). We also showed that users tend to be more efficient in completing the task when they have bookmarks than when they do not.

We can say that bookmarks have a positive effect on navigation within the 3D scene, but since users move, on average, twice as fast, it might have a negative impact on the streaming of objects to the client.

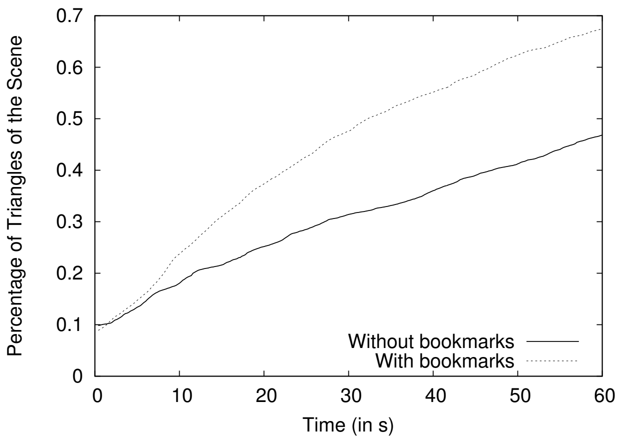

Figure 2: Comparison of the triangles queried after a certain time

Figure 2 shows a CDF of the percentage of 3D mesh triangles in the scene that have been queried by users after a certain time. We plotted this same curve for users with and without bookmarks. As expected, the fact that the users can browse the scene significantly quicker with bookmarks reflects on the demand on the 3D content. Users need more triangles more quickly, which either leads to more demand on network bandwidth, or if the bandwidth is kept constant, leads to fewer objects being displayed. In the next section, we introduce experiments based on our user study traces that show how the rendering is affected by the presence of bookmarks and how to improve it.

The traces are online at https://github.com/tforgione/3dinterface/releases/tag/V1 in the `microTestFinal.pgsql` script which is a dump of the database at the end of the experiment.

4. IMPACT OF 3D BOOKMARKS ON STREAMING

4.1 3D Model Streaming

In this section, we describe our implementation of a 3D model streaming policy in our simulation. Note that the policy is different from that we used for the crowdsourcing experiments. Recall that in the crowdsourcing experiments, we load all the 3D content before the participants begin to navigate to remove bias due to different network conditions. Here, we implemented a streaming version, which we expect an actual NVE will use.

The 3D content we used are textured mesh – coded in `obj` file format. As such, the data we used in our experiments are made of several components. The geometry consists of (i) a list of vertices and (ii) a list of faces, and the texture consists of (i) a list of materials, (ii) a list of texture coordinates, and (iii) a set of texture images. In the crowdsourcing experiment, we keep the model small since the goal is to study the user interaction. To increase the size of the model, while keeping the same 3D scene, we subdivide each triangle three times, successively, thereby multiplying the total number of triangles in the scene by 64. We do this to simulate a reasonable use case with large 3D scenes. Table 4 shows that material and texture amount at most for 3.6% of the geometry, which justifies this choice.

When a client starts loading the Web page containing the 3D model, the server first sends the list of materials and the texture files. Then, the server periodically sends a fixed size chunk that indifferently encapsulates vertices, texture coordinates, or faces. A *vertex* is coded with three floats and an integer (x, y, and z coordinates and the index of the vertex), a *texture coordinate* with two floats and an integer (the x and y coordinates on the image and the index of the texture coordinate), and a face with eight integers (the index of each vertex, the index of each texture coordinate, the index of the face and the number of the corresponding material). Consequently, given the Javascript implementation of integers and floats, we approximate each vertex and each texture coordinate to take up 32 bytes, and each face takes up 96 bytes.

	Material	Images	Geometry
Scene 1	8 KB	72 KB	8.48 MB
Scene 2	302 KB	8 KB	8.54 MB
Scene 3	16 KB	92 KB	5.85 MB

Table 4: Respective sizes of materials, textures (images) and geometries for the three scenes used in the user study.

During playback, the client periodically (every 200 ms in our implementation) sends to the server its current position and camera orientation. The server computes a sorted list of relevant faces: first the server performs frustum culling to compute the list of faces that intersect with the client's viewing frustum. Then, it performs back-face culling to discard the faces whose normals point towards the same direction as the client's camera orientation. The server then sorts the filtered faces according to their distance to the camera. Finally, the server incrementally fills in chunks with these ordered faces. If a face depends on a vertex or a texture coordinate that has not yet been sent, the vertex or the texture coordinate is added to the chunk as well. When the chunk is full, the server sends it. Both client and server algorithms are detailed in algorithms 1 and 2. The chunk size is set according to the bandwidth limit of the server. Note that the server may send faces that are occluded and not visible to the client, since determining visibility requires additional computation.

```
while streaming is not finished do
    Receive chunk from the server;
    Add the faces from the chunk to the model;
    Update the camera (by 200ms);
    Compute the rendering and evaluate the quality;
    Send the position of the camera to the server;
end
```

Algorithm 1: Client slide algorithm

```
while streaming is not finished do
    Receive position of the camera from the client;
    Compute the list of triangles to send and sort them;
    Send a chunk of a certain amount of triangles;
end
```

Algorithm 2: Server side algorithm

In the following, we shall denote this streaming policy culling; in Figures 6 and 7 streaming using culling only is denoted C-only.

4.2 3D Bookmarks

We have seen (Figure 2) that navigation with bookmarks is more demanding on the bandwidth. We want to exploit bookmarks to improve the user's quality of experience. For this purpose, we propose two streaming policies based on offline computation of the relevance of 3D content to bookmarked viewpoints.

4.2.1 Visibility Determination for 3D Bookmarks

A bookmarked viewpoint is more likely to be accessed, compared to other arbitrary viewpoint in the 3D scene. We exploit this fact to perform some pre-computation on the 3D content visible from the bookmarked viewpoint.

Recall that culling does not consider occlusion of the faces. Furthermore, it prioritizes the faces according to distance from the camera, and does not consider the actual contribution of the faces to the rendered 2D images. Ideally, we should prioritize the faces that occupy a bigger area in the 2D rendered images. Computing this, however, requires rendering the scene at the server, and measuring the area of each face. It is not scalable to compute this for every viewpoint requested by the client.

However, we can pre-render the bookmarked viewpoints, since the number of bookmarks is limited, their viewpoints are known in advance, and they are likely to be accessed. For each bookmark, we render offline the scene using a single color per triangle. Once rendered, we scan the output image to find the visible triangles (based on the color) and sort them by decreasing projected area.

This technique is also used by [7]. Thus, when the user clicks on a 3D bookmark, this pre-computed list of faces is used by the server, and only visible faces are sent in decreasing order of contributions to the rendered image.

For the three scenes that we used in the experiment, we can reduce the number of triangles sent by 60% (over all bookmarks). This reduction is as high as 85.7% for one particular bookmark (from 26,886 culled triangles to 3,853 culled and visible triangles).

To illustrate the impact of sorting by projected area of faces, Figure 3 shows the quality improvement gained by sending the precomputed visible triangles prioritized by projected areas, compared to using culling only prioritized by distance. The curve shows the average quality over all bookmarks over all scenes, for a given number of triangles received. The quality is measured by the ratio of correctly rendered pixels, comparing the fully and correctly rendered image (when all 3D content is available) and the rendered image (when content is partially available). We sample one pixel every 100 rows and every 100 columns to compute this value. The figure shows that, to obtain 90% of correctly displayed samples, we require 1904 triangles instead of 5752 triangles, about 1/3 savings.

In what follows, we will refer to this streaming policy as visible.

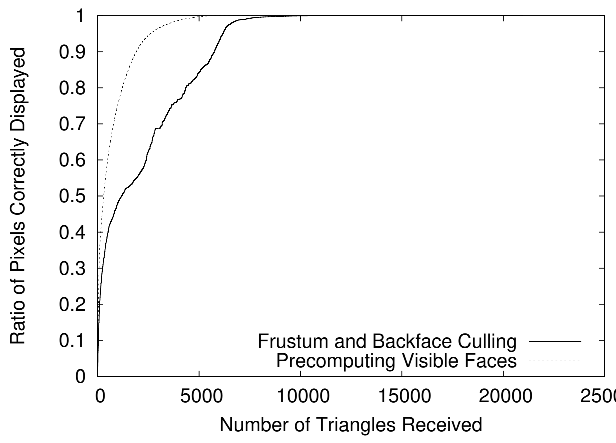

Figure 3: Comparison of rendered image quality (average on all bookmarks and starting position): the triangles are sorted offline (dotted curve), or sorted online by distance to the viewpoint (solid curve).

4.2.2 *Prefetching by Predicting the Next Bookmark Clicked*

We can now use the precomputed, visibility-based streaming of 3D content for the bookmarks to reduce the amount of traffic needed. Next, we propose to prefetch the 3D content from the bookmarks. Any efficient prefetching policy needs to accurately predict users' actions.

As shown, users tend to visit the bookmarked viewpoints more often than others, except the initial viewpoint. It is thus natural to try to prefetch the 3D content of the bookmarks.

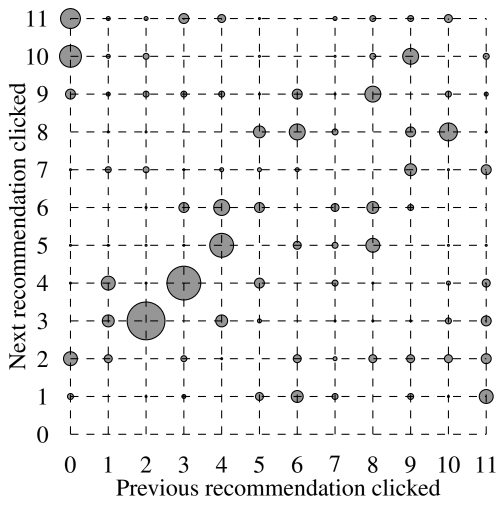

Figure 4: Probability distribution of 'next clicked bookmark' for Scene 1 (computed from the 33 users with bookmarks). Numbering corresponds to 0 for initial viewport and 11 bookmarks; the size of the disk at (i, j) is proportional to the probability of clicking bookmark j after i.

Figure 4 shows the probability of visiting a bookmark (vertical axis) given that another bookmark has been visited (horizontal axis). This figure shows that users tend to follow similar paths when consuming bookmarks. Thus, we hypothesize that prefetching along those paths would lead to better image quality and lower discovery latency.

We use the following prefetching policy in this paper. We divide each chunk sent by the server into two parts. The first part is used to fetch the content from the current viewpoint, using the culling streaming policy. The second part is used to prefetch content from the bookmarks, according to their likelihood of being clicked next. We use the probabilities displayed in Figure 4 to determine the size of each part. Each bookmark B has a probability $p(B|B_{prev})$ of being clicked next, considering that B_{prev} was the last clicked bookmark. We assign to each bookmark $p(B|B_{prev})/2$ of the chunk to prefetch the corresponding data. We use the visible policy to determine which data should be sent for a bookmark.

We denote this combination as V-PP, for Prefetching based on Prediction using visible policy.

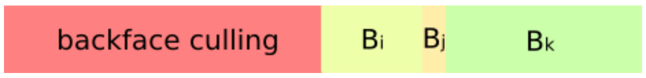

Figure 5: Example of how a chunk can be divided into fetching what is needed to display the current viewport (culling), and prefetching three recommendations according to their probability of being visited next.

4.2.3 *Fetching Destination Bookmark*

An alternate method to benefit from the precomputing visible triangles at the bookmark, is to fetch 3D content during the "fly-to" transition to reach the destination. Indeed, as specified in Section 3, moving to a bookmarked viewpoint is not instantaneous, but rather takes a small amount of time to smoothly move the user camera from its initial position towards the bookmark. This transition usually takes from 1 to 2 seconds, depending on how far the current user camera position is from the bookmark.

When the user clicks on the bookmark, the client fetches the visible vertices from the destination viewpoint, with all the available bandwidth. So, during the transition time, the server no longer does culling, but the whole chunk is used for fetching following visible policy.

The immediate drawback of this policy is that on the way to the bookmark, the user perception of the scene will be degraded because of the lack of data for the viewpoints in transition. On the bright side, no time is lost to prefetch bookmarks that will never be consumed, because we fetch only when we are sure that the user has clicked on a bookmark. This way, when the user is not clicking on bookmarks, we can use the entire bandwidth for the current viewpoint and get as many triangles as possible to improve the current viewpoint. We call this method V-FD, since we are Fetching the 3D data from the Destination using visible policy.

4.3 Comparing Streaming Policies

In order to determine which policy to use, we replay the traces from the user study while simulating different streaming policies. The first point we are interested in is which streaming policy leads to the lower discovery latency and better image quality for the user: culling (no prefetching), V-PP (prefetching based on probability of accessing bookmarks), or V-FD (no prefetching, but fetch the destination during fly-to transition) or combining both V-PP and V-FD (V-PP+FD).

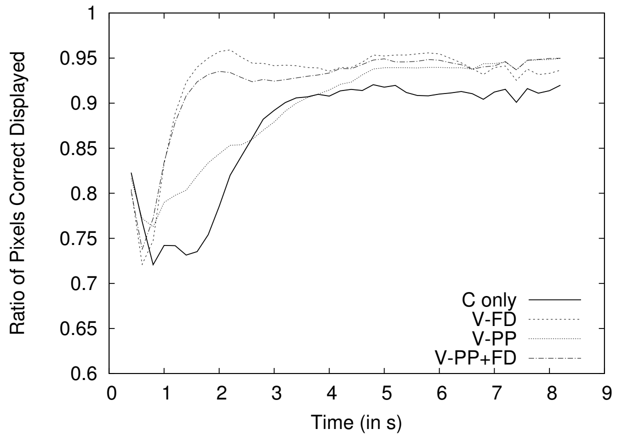

Figure 6: Average percentage of the image pixels that are correctly rendered against time –for all users with bookmarks, and using a bandwidth (BW) of 1 Mbps. The origin, $t = 0$, is the time of the first click on a bookmark. Each curve corresponds to a streaming policy.

Figure 6 compares the quality of the view of a user after his/her first click on a bookmark. The ratio of pixels correctly displayed is computed in the client algorithm, see also algorithm 1. In this figure we use a bandwidth of 1 Mbps. The solid curve corresponds to the culling policy. Clicking on a bookmark generates a user path with less spatial locality, causing a large drop in visual quality that is only compensated after 4 seconds. During the first second, the camera moves from the current viewport to the bookmarked viewport.

When the data has been prefetched according to the probability of the bookmark to be clicked, the drop in quality is less visible (V-PP curve). However, by benefiting from the precomputation of visible triangles and ordering of the important triangles in a bookmark (V-FD) the drop in quality is still there, but is very short (approximately four times shorter than for culling). This drop in quality is happening during the transition on the path. More quantitatively, with a 1 Mbps bandwidth, 3 seconds are necessary after the click to recover 90% of correct pixels.

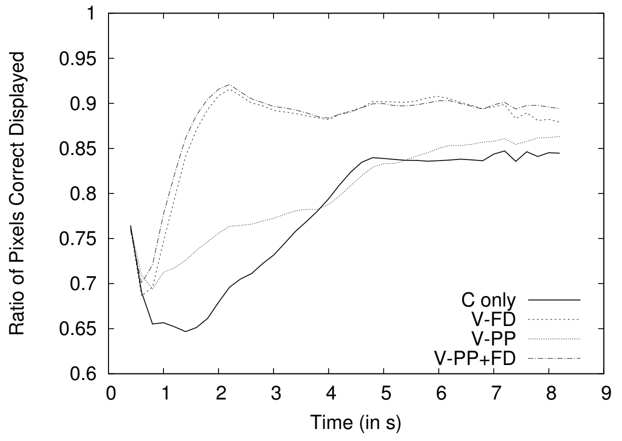

Figure 7: Average percentage of the image pixels that are correctly rendered against time –for all users with bookmarks, and using a bandwidth (BW) of 0.5 Mbps. The origin, $t = 0$, is the time of the first click on a bookmark. Each curve corresponds to a streaming policy.

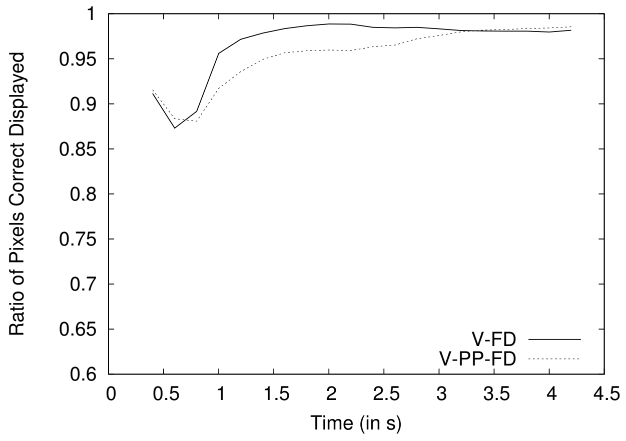

Figure 8: Same curve as Figures 6 and 7, for comparing streaming policies V-FD alone and V-PP+FD. BW=2Mbps

Figure 7 showed the results of the same experiment with 0.5 Mbps bandwidth. Here, it takes 4-5 seconds to recover 85% of the pixels with culling and V-PP, against 1.5 second for recovering 90% with V-FD. Combining both strategies (V-PP+FD leads to the best quality.

At 1 Mbps bandwidth, V-PP penalizes the quality, as the curve V-PP-FD) leads to a lower quality image than V-FD alone. This effect is even stronger when the bandwidth is set to 2 Mbps (Figure

8). Both streaming strategies based on the pre-computation of the ordering improves the image quality. We see here, that V-FD has a greater impact than V-PP. Here, V-PP may prefetch content that eventually may not be used, whereas V-FD only sends relevant 3D content (knowing which bookmark has been just clicked).

We present only the results after the first click. For subsequent clicks, we found that other factors came into play and thus, it is hard to analyze the impact of the various streaming policies. For instance, a user may revisit a previously visited bookmark, or the bookmarks may overlap. If the users click on a subsequent bookmark after a long period, then more content would have been fetched for this user, making comparisons difficult.

To summarize, we found that exploiting the fact that bookmarked viewpoints are frequently visited to precompute the visible faces and sort them according to projected areas can lead to significant improvement in image quality after a user interaction (clicking on a bookmark). This alone can lead to 60% less triangles being sent, with 1/3 of the triangles sufficient to ensure 90% of pixels correctly rendered, compared to doing frustum/backface culling. If we fetch these precomputed faces of the destination viewpoint this way immediately after the click, during the “fly-to” transition, then we can already significantly improve the quality without any prefetching. Prefetching helps if the bandwidth is low, and fewer triangles can be downloaded during this transition. The network conditions play a minimum role in this key message – bookmarking allows precomputation of an ordered list of visible faces, and this holds regardless of the underlying network condition (except for non-interesting extreme cases, such as negligible bandwidth or abundance of bandwidth).

5. CONCLUSION AND FUTURE WORK

In this work, we have shown the usefulness of 3D bookmarks for easing navigation in a networked virtual environment. First, users who benefit from the bookmarks took, on average, half the time of user without bookmarks to complete the task of collecting coins around the scene, even when the bookmarks were placed independently of the coins. A large majority of users also expressed their preference for the UIs with bookmarks. However, as the speed and task efficiency of the user increase with bookmarks, clicking on a bookmark decreases 3D data locality and the quality of the rendered images lowers as well. We propose two streaming policies that can benefit from the bookmarks by precomputing an ordering of visible triangles for bookmarked viewpoints based on projected area: one that prefetches based on previous navigation pattern, the other fetches content during the fly-to transition to the bookmark after a user click. The proposed streaming outperforms a policy based only on frustum/backface culling computed during navigation.

One future work could be to optimize the chunk size allocated to prefetching for higher bandwidth. A more challenging future work could be to adapt the navigation to system conditions, e.g., slowing down the fly-to to a bookmark when the bandwidth is lower, to improve the rendering at the destination bookmark. Another adaptation could be to zoom out on the path to increase the perceptual quality during transitions.

6. REFERENCES

[1] W. Ali, S. M. Shamsuddin, and A. S. Ismail. A survey of web caching and prefetching. *International Journal of Advances in Soft Computing and its Application*, 3(1):18–44, 2011.

[2] S. Burigat and L. Chittaro. Navigation in 3d virtual environments: Effects of user experience and location-pointing navigation aids. *International Journal of Man-Machine Studies*, 65(11):945–958, 2007.

[3] N. Burtnyk, A. Khan, G. Fitzmaurice, and G. Kurtenbach. Showmotion: camera motion based 3d design review. In *Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 167–174. ACM, 2006.

[4] A. Carlier, V. Charvillat, and W. T. Ooi. A video timeline with bookmarks and prefetch state for faster video browsing. In *Proceedings of the 23rd Annual ACM Conference on Multimedia Conference*, pages 967–970, Brisbane, Australia, Oct. 2015. ACM.

[5] A. Carlier, G. Ravindra, and W. T. Ooi. Towards characterizing users’ interaction with zoomable video. In *Proceedings of the 2010 ACM Workshop on Social, Adaptive and Personalized Multimedia Interaction and Access*, SAPMIA ’10, pages 21–24, Firenze, Italy, 2010.

[6] A. Chan, R. W. H. Lau, and B. Ng. A hybrid motion prediction method for caching and prefetching in distributed virtual environments. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, VRST ’01, pages 135–142, Baniff, Alberta, Canada, 2001. ACM.

[7] W. Cheng and W. T. Ooi. Receiver-driven view-dependent streaming of progressive mesh. In *Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 9–14. ACM, 2008.

[8] J. Chim, R. W. Lau, H. V. Leong, and A. Si. CyberWalk: a Web-based distributed virtual walkthrough environment. *Multimedia, IEEE Transactions on*, 5(4):503–515, 2003.

[9] L. Chittaro and S. Burigat. 3d location-pointing as a navigation aid in virtual environments. In *Proceedings of the working conference on Advanced visual interfaces, AVI 2004, Gallipoli, Italy, May 25-28, 2004*, pages 267–274, 2004.

[10] L. Chittaro and S. Venkataraman. Navigation aids for multi-floor virtual buildings: A comparative evaluation of two approaches. In *Proceedings of the ACM symposium on Virtual Reality Software and Technology*, pages 227–235. ACM, 2006.

[11] J. Eno, S. Gauch, and C. W. Thompson. Linking behavior in a virtual world environment. In *Proceedings of the 15th International Conference on Web 3D Technology*, pages 157–164. ACM, 2010.

[12] J. Gaillard, A. Vienne, R. Baume, F. Pedrinis, A. Peytavie, and G. Gesquière. Urban data visualisation in a web browser. In *Proceedings of the 20th International Conference on 3D Web Technology*, Web3D ’15, pages 81–88, Heraklion, Crete, Greece, 2015. ACM.

[13] R. Grigoras, V. Charvillat, and M. Douze. Optimizing hypervideo navigation using a Markov decision process approach. In *Proceedings of the 10th ACM International Conference on Multimedia*, pages 39–48, Juan les Pins, France, 2002.

[14] S.-S. Hung and D. S.-M. Liu. Using prefetching to improve walkthrough latency: Research articles. *Comput. Animat. Virtual Worlds*, 17(3-4):469–478, July 2006.

[15] J. Jankowski and S. Decker. A dual-mode user interface for accessing 3d content on the world wide web. In *Proceedings of the 21st international conference on World Wide Web*, pages 1047–1056. ACM, 2012.

[16] J. Jankowski and M. Hachet. Advances in interaction with 3d environments. *Comput. Graph. Forum*, 34(1):152–190, 2015.

[17] A. Khan, I. Mordatch, G. Fitzmaurice, J. Matejka, and G. Kurtenbach. Viewcube: A 3d orientation indicator and controller. In *Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games*, I3D '08, pages 17–25. ACM, 2008.

[18] S. Khemmarat, R. Zhou, D. K. Krishnappa, L. Gao, and M. Zink. Watching user generated videos with prefetching. *Signal Processing: Image Communication*, 27(4):343–359, 2012.

[19] T.-Y. Li and W.-H. Hsu. A data management scheme for effective walkthrough in large-scale virtual environments. *The Visual Computer*, 20(10):624–634, 2004.

[20] K. Liang, R. Zimmermann, and W. T. Ooi. Peer-assisted texture streaming in metaverses. In *Proceedings of the 19th ACM International Conference on Multimedia*, pages 203–212, Scottsdale, AZ, 2011. ACM.

[21] J. D. Mackinlay, S. K. Card, and G. G. Robertson. Rapid controlled movement through a virtual 3d workspace. In *ACM SIGGRAPH Computer Graphics*, volume 24, pages 171–176. ACM, 1990.

[22] C. Moerman, D. Marchal, and L. Grisoni. Drag'n go: Simple and fast navigation in virtual environment. In *3D User Interfaces (3DUI), 2012 IEEE Symposium on*, pages 15–18. IEEE, 2012.

[23] S. Park, D. Lee, M. Lim, and C. Yu. Scalable data management using user-based caching and prefetching in distributed virtual environments. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, pages 121–126, Banff, Canada, 11 2001.

[24] S. Rezzonico and D. Thalmann. Browsing 3D bookmarks in BED. In *Proceedings of WebNet 96 - World Conference of the Web Society*, San Francisco, California, USA, October 1996.

[25] R. A. Ruddle, A. Howes, S. J. Payne, and D. M. Jones. The effects of hyperlinks on navigation in virtual environments. *International Journal of Human-Computer Studies*, 53(4):551–581, 2000.

[26] B. Seo and R. Zimmermann. Quantitative analysis of visibility determinations for networked virtual environments. *Journal of Visual Communication and Image Representation*, 23(5):705–718, 2012.

[27] J. T. Todd. The visual perception of 3d shape. *Trends in cognitive sciences*, 8(3):115–121, 2004.

[28] M. Varvello, S. Ferrari, E. Biersack, and C. Diot. Exploring Second Life. *IEEE/ACM Transactions on Networking (TON)*, 19(1):80–91, 2011.

[29] R. C. Zeleznik, A. S. Forsberg, and P. S. Strauss. Two pointer input for 3d interaction. In *Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 115–120. ACM, 1997.

[30] Z. W. Zhao and W. T. Ooi. Joserlin: Joint request and service scheduling for peer-to-peer non-linear media access. In *Proceedings of the 21st ACM International Conference on Multimedia*, MM '13, pages 303–312, Barcelona, Spain, Oct. 2013.

[31] Z. Zhou, K. Chen, and J. Zhang. Efficient 3-D scene prefetching from learning user access patterns. *IEEE Transactions on Multimedia*, 17(7):1081–1095, July 2015.