3D Res (2015) 6:8
DOI 10.1007/s13319-015-0040-0

3DR REVIEW

Optimization Techniques for 3D Graphics Deployment on Mobile Devices

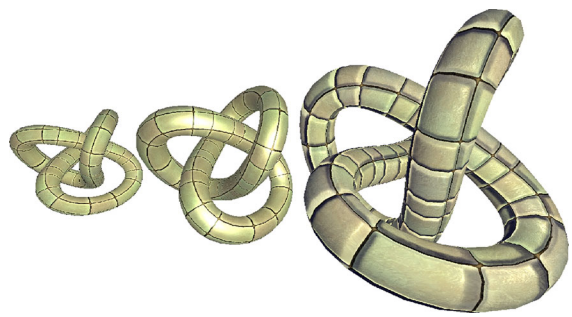
Timo Koskela · Jarkko Vatjus-Anttila

Received: 20 October 2014 / Revised: 19 January 2015 / Accepted: 23 January 2015 / Published online: 7 February 2015

Abstract 3D Internet technologies are becoming essential enablers in many application areas including games, education, collaboration, navigation and social networking. The use of 3D Internet applications with mobile devices provides location-independent access and richer use context, but also performance issues. Therefore, one of the important challenges facing 3D Internet applications is the deployment of 3D graphics on mobile devices. In this article, we present an extensive survey on optimization techniques for 3D graphics deployment on mobile devices and qualitatively analyze the applicability of each technique from the standpoints of visual quality, performance and energy consumption. The analysis focuses on optimization techniques related to data-driven 3D graphics deployment, because it supports off-line use, multi-user interaction, user-created 3D graphics and creation of arbitrary 3D graphics. The outcome of the analysis facilitates the development and deployment of 3D Internet applications on mobile devices and provides guidelines for future research.

T. Koskela (✉) · J. Vatjus-Anttila
Center for Internet Excellence, University of Oulu, Oulu, Finland
e-mail: timo.koskela@cie.fi

Graphical Abstract



Keywords 3D content · 3D Internet · Virtual environment · Content simplification · Mobile networking · Energy consumption

1 Introduction

3D graphics started to emerge as a separate branch in the field of computer graphics in the 1970s, when the first pieces of 3D graphics were introduced to both movie industry and computer gaming. In those days, expensive gaming consoles were needed to enjoy the 3D graphics, whereas today, even regular mobile devices have enough power to run applications enriched with 3D content. Along with the development of 3D graphics, the growth of the Internet and its mobilization has allowed seamless networking and multi-user applications for mobile users. The combination of 3D content and the Internet is sometimes called *the 3D Internet*, which is concerned with

creation, *modification*, *storage and transmission of 3D content and its presentation to the user*. 3D content can include 3D graphics, 3D video and 3D audio, but in this article, we solely focus on 3D graphics. Particularly, our focus is on polygon meshes that are also commonly known as surface meshes, since they only present the surface of a 3D model using vertices, edges and faces.

Commercially, the most successful 3D Internet applications have focused on entertainment. For example, massive multiplayer online (MMO) games that are capable of supporting a large number of users simultaneously have seen growth rise to 12 billion dollars in 2012 [1]. However, there are a growing number of new enablers that will speed up the introduction of 3D Internet applications in the future. First, wireless radio technologies such as LTE and WiFi make it possible to transmit large volumes of 3D graphics quickly. Second, WebGL supports viewing 3D graphics in a web browser, which provides a cross-platform access to 3D graphics without the need for installing any additional applications [2]. Third, technologies for creating 3D graphics, such as depth cameras, are becoming cheaper and accessible to a wider range of users [3]. Fourth, many applications with a strong 3D Internet base such as augmented reality are seeing new growth [4]. Supported by these new enablers, the 3D Internet will eventually emerge in many application areas including, for instance, education [5, 6], collaboration [7], navigation [8] and social networking [9].

Today, high-end mobile devices have the same computing power, storage and communication capabilities as powerful desktop computers from 10 years ago [10]. As a result, a variety of applications that were previously only available in the fixed network environment have been introduced to the mobile realm. This trend can be seen with social networking applications, for example, Facebook is now primarily accessed using mobile devices [11]. The use of mobile devices presents benefits for 3D Internet applications through wireless network access and richer context due to multiple embedded sensors such as a GPS/GLONASS, an accelerometer, a microphone and a camera [12]. These features of mobile devices enable creation of 3D Internet applications that are not tied to any fixed location, while providing richer interaction with the 3D Internet application itself as well as with the surrounding physical environment [4, 13].

Despite recent developments, the deployment of 3D graphics is still a demanding task for mobile devices which typically are very heterogeneous in terms of performance [14, 15]. First, the rendering of 3D models is a heavy process for both the graphical processing unit (GPU) and the central processing unit (CPU), and furthermore, 3D rendering requires a lot of memory to run fluently [16]. Second, the size of the 3D models that need to be delivered is large, which sets demands for the wireless bandwidth [17]. Third, the intensive use of hardware resources has a significant effect on the limited battery life of a mobile device [18].

In the literature, a significant number of techniques have already been introduced that enable the optimization of 3D graphics deployment according to the heterogeneous capabilities of end-user devices. The prior research altogether covers the whole delivery chain of 3D graphics including the 3D graphics creation, remote 3D graphics simplification, 3D graphics delivery, on-device 3D graphics optimization, and the 3D graphics rendering on the end-user device [19]. In this article, the focus is on remote 3D graphics simplification, 3D graphics delivery and on-device 3D graphics optimization, which are the phases, where the application/system developer (hence called the developer) has the best opportunity to influence the characteristics of 3D graphics. Furthermore, the article focuses only on data-driven 3D graphics deployment, where the 3D graphics received from the server drives the client's generic engine to reproduce the intended 3D space. The data-driven 3D graphics deployment provides good support for off-line use, multi-user interaction, creation and delivery of user-created 3D graphics as well as creation of arbitrary 3D graphics.

Although the optimization techniques for 3D graphics deployment have been widely examined by the research community, the applicability of these techniques for mobile devices has been rarely discussed. Furthermore, there exist no comprehensive studies that categorize and analyze the optimization techniques particularly from the standpoint of mobile devices. Capin et al. [14] present some optimization techniques for mobile devices, but their focus is more on 3D displays and 3D user interfaces. Gotchev et al. [20] provide an extensive survey on 3D content optimization for mobile devices, but their focus is solely on 3D video and the related optimization techniques. Evans et al. [21], in turn, introduce technologies for presenting 3D

graphics on a web browser without any particular focus on mobile use. In this article, we aim to fulfil this gap by providing (1) an extensive literature survey on the existing optimization techniques applicable for mobile devices, and (2) a qualitative analysis of their advantages and disadvantages from the standpoints of visual quality, performance and energy consumption. The contents of this article facilitate the selection of appropriate optimization techniques when implementing 3D Internet applications for mobile devices in the further research and development.

The rest of this article is organized as follows: the analysis framework is presented in Sect. 2, where Sect. 2.1 introduces different approaches for deploying 3D graphics on mobile devices, Sect. 2.2 illustrates the general delivery chain of 3D graphics and the categorization of the optimization techniques, and Sect. 2.3 presents the optimization targets for 3D graphics deployment on mobile devices. Based on Sects. 2, 3 analyzes the advantages and disadvantages of different optimization techniques developed for data-driven 3D graphics deployment. Finally, Sect. 4 provides some guidelines for selecting suitable optimization techniques and concludes the article.

2 Analysis Framework

2.1 Different Approaches for Deploying 3D Graphics on Mobile Devices

In this section, five different technical approaches for deploying 3D graphics on mobile devices are presented. The technical approaches are called (1) pre-packaged, (2) image-based, (3) streaming-based, (4) ontological, and (5) data-driven. Before going into details, six attributes are presented in Table 1 for comparing these different technical approaches. The attributes were selected in a way that they describe the general characteristics of each technical approach for deploying 3D graphics on mobile devices and they are consistent regardless of the chosen use case.

The first attribute in Table 1, *3D rendering author*, refers to the entity that produces the 3D graphics. It can be either the server or the client. Second, *offline 3D rendering* indicates whether after the first download of data, the client is able to simulate and redraw the 3D space even if there is no active network connection available. Third, *interaction with 3D objects* in the 3D space refers to (1) *offline usage*, i.e., is the client able to interact with the 3D space once there is no network connection available, and (2) *multi-user usage*, i.e., is the client able to interact with other users in the same 3D space. Fourth, *user-created 3D graphics* indicates whether the user is able to locally create and share new 3D graphics in the 3D space. Fifth, *3D graphics complexity limitations* refer to technology and how its data types are able to handle any arbitrary piece of 3D graphics. Finally, *3D graphics distribution* indicates the type of 3D graphics the server sends to its clients for 3D graphics reproduction.

Next, the basic operational principle of each technical approach is explained and their attributes based on Table 1 are elaborated. Henceforth, the term 3D scene is used for referring to a data structure that describes how the 3D graphics are arranged in a particular 3D space. The purpose of the 3D scene is to define the types of 3D objects and where those are placed in a 3D space.

2.1.1 Pre-packaged 3D Graphics Deployment

Pre-packaged 3D graphics deployment means that all 3D graphics are packaged and delivered to the client

Table 1 Differentiating properties of the five approaches for deploying 3D graphics on mobile devices

	3D rendering author	Offline 3D rendering	Interaction with 3D objects		User-created 3D graphics	3D graphics complexity not limited	3D graphics distribution
			Offline	Multi-user			
Pre-packaged	Client	Yes	Yes	Yes	No	Yes	None
Image-based	Server	Yes	No	No	No	Yes	Texture data
Streaming-based	Server	No	No	Yes	No	Yes	Video stream
Ontological	Client	Yes	Yes	Yes	Yes	No	Ontological data
Data-driven	Client	Yes	Yes	Yes	Yes	Yes	Mesh and texture data

device prior to the use of the 3D Internet application in question, and thus, all 3D rendering is executed by the client. This approach supports both off-line and multi-user usage. In case of multi-user usage, only the state information of 3D objects involved in the 3D scene needs to be continuously updated between the participating clients [22, 23]. State information can be, for instance, a position of a 3D object. MMOs are a good example, where pre-packaged 3D graphics deployment is widely used. This approach allows full control of the 3D graphics by the service provider, and also makes it difficult or impossible for users to add their own. Distribution of the 3D graphics before the use also allows the network traffic to be optimized to state changes only, since it is known that the client always has all the 3D graphics in place prior using the 3D Internet application.

2.1.2 Image-Based 3D Graphics Deployment

In image-based 3D graphics deployment, the actual 3D rendering is executed by the server, whereas the client merely reconstructs the 3D scene based on a set of images transmitted by the server [15, 17, 24, 25]. For instance, in [17], the server records six panorama images in the client's current position, generates a G-buffer cube map and transmits the cube map as a texture data to the client. Finally, the client reconstructs the 3D scene based on the transmitted panorama images using image-based rendering. The image-based 3D graphics deployment allows the client to present very complex 3D scenes, as all the complexity of the 3D scene is handled by the server [17]. However, as the G-buffer cube maps are pre-rendered from a single camera position, the client can reproduce the view only when its position remains unchanged. Whenever the client changes its position, or interacts with the 3D scene in a way that the 3D scene is changed, the view must be re-rendered and re-transmitted by the server. Thus, off-line usage with the image-based 3D graphics deployment is not possible. Multi-user usage is very challenging to implement, because the 3D scene is not continuously updated.

2.1.3 Streaming-Based 3D Graphics Deployment

In streaming-based 3D graphics deployment, the server does continuous rendering of the 3D scene for the client, keeping the client camera position and orientation updated [26–28]. The continuous rendering is encoded into a video stream and is then distributed to the client. Off-line usage is not possible. However, the server can control the video stream resolution and bitrate in order to adapt to the available bandwidth and the capabilities of the client [26, 28]. In general, streaming-based 3D graphics deployment requires a constant available bitrate from the network, as well as relatively low latency. This is due to the chain of continuously performed real-time activities, which are (1) decoding of user input into camera position/orientation changes by the client, (2) submitting new camera orientation to the server, (3) applying new camera orientation and re-rendering the 3D scene, (4) performing the video encoding, (5) sending the video stream to the client, and finally, (6) decoding the video stream at the client. In case of multi-user usage, interaction with 3D objects can be implemented by the server based on users' actions in a certain camera position by recording pairs of X, Y-coordinates pointed out by the user from the video presentation of a 3D scene [26].

2.1.4 Ontological 3D Graphics Deployment

Ontological 3D graphics deployment means that the structure of the 3D scene is described in an abstract manner, and then it is left for the client to re-create the 3D scene. The server informs the client about the 3D objects such as buildings, trees and terrain that exist in a 3D scene [29, 30]. However, the actual 3D graphics are not delivered. Based on the provided ontological description of the 3D scene, the client creates the 3D scene by implementing a set of algorithms required for generating 3D objects and applies the parameters given by the server. As the 3D scene is created locally by the client, this approach supports off-line usage. In multi-user interaction, the state information of 3D objects involved in the same 3D scene needs to be continuously updated between the participating clients. In addition, ontological descriptions of new 3D objects need to be distributed in real-time between the participating clients. However, it should be noted that the visual appearance of a 3D scene depends on the used set of algorithms, and thus, the same 3D scene may have varying looks for different clients. Second Life primitive system [31] is an example of how this approach works. Their system is able to distribute information about primitive shapes (boxes, spheres,

etc.), which are used to build more complex 3D objects. Ontological 3D graphics deployment does not allow building of any arbitrary shapes, since it is restricted to the "ontological toolbox" defined by the server. Also, describing more complex shapes, like buildings, may lead into overly complex algorithms and description languages, which are not implementable, especially on mobile clients.

2.1.5 Data-Driven 3D Graphics Deployment

In data-driven 3D graphics deployment, the client software is built to be as generic as possible in terms of how to re-create a 3D scene. The data and the implementation are separate from each other, meaning that the implementation is generic and can be driven with any dataset compatible with the given implementation. A PNG picture format and its codec present an example. The PNG codec is able to accept all PNG compatible datasets and decode this data into viewable images. Hence, the PNG codec is said to be generic for arbitrary PNG data and to be data-driven. As summarized by Alatalo [32], the same analogy can also be applied to 3D graphics. In data-driven 3D graphics deployment, the pieces of 3D graphics received from the server acts as "data", or "attributes", which drive the client's generic 3D engine to reproduce the intended 3D scene. As the 3D scene is created locally by the client, this approach supports off-line usage. In multi-user usage, the state information of 3D objects involved in the same 3D scene needs to be continuously updated between the participating clients. In addition, new pieces of 3D graphics need to be distributed in real-time between the participating clients. For describing the surface geometry of 3D objects, a commonly used method is a generic geometry mesh type [33]. Generic mesh is defined by three dimensional points (vertices) and by information which points are connected to each other. Various connection types are possible, but typically three individual vertices are connected to form a triangle, and hence an array of triangles will form a 3D mesh surface. Vertices can store other types of information as well in addition to their position (X, Y and Z coordinates). Such information can be, but is not limited to, per-vertex normals, texture coordinates and color information. This kind of approach gives maximum freedom for defining the characteristics of 3D graphics, which is also important from the standpoint of user-created content. However, this freedom also brings forth the challenge to understand the complexity of 3D graphics and its effect on various types of devices. Mobile devices, for instance, may not be able to handle the 3D graphics in a similar way to powerful workstations [8].

The rest of this article concentrates especially on the optimization techniques related to the data-driven approach for deploying 3D graphics on mobile devices. This focus was chosen based on the assumptions that (1) in the future Internet, the creation of 3D graphics becomes much easier and faster, for instance, through the introduction of mobile devices embedded with depth cameras, (2) most of the 3D graphics are created, manipulated, re-used and distributed by end-users in a dynamic manner, following the trend that we have already witnessed with 2D content managed with Web 2.0 technologies [20], and (3) mobile devices are starting to dominate the use of many highly popular social networking applications (e.g., Facebook [11]). By taking these assumptions into account, it is important that the technical approach for deploying 3D graphics on mobile devices enables (1) offline use because a stable wireless network connection is not always available, (2) multi-user interaction to support social networking aspects, (3) easy creation and distribution of user-created 3D graphics to speed up the 3D graphics life-cycle, and (4) creation of arbitrary and artistic 3D graphics of which complexity is not limited by any technical means as in the ontological approach.

2.2 A General Delivery Chain of 3D Graphics

The delivery of 3D graphics from the source to the sink contains up to five identifiable phases, which are 3D graphics creation, remote 3D graphics simplification, 3D graphics delivery, on-device 3D graphics optimization and 3D graphics rendering. In this article, we will focus on the remote simplification, the delivery and the on-device optimization as illustrated in Fig. 1.

2.2.1 Remote Simplification Phase

The remote simplification phase includes the use of various methods for reducing the complexity and the size of 3D graphics prior to their delivery to a mobile device. The optimization techniques in the simplification phase fall into the following categories: (1) level-of-detail (LOD) techniques, where lower quality

Fig. 1 A general delivery chain of 3D graphics with the categorization of optimization techniques for deploying 3D graphics on mobile devices

versions of the original 3D models are created using various methods for geometry simplification [33], texture simplification [34] and surface material simplification [35], (2) data type transformations, because some 3D graphics formats are more efficient to render than others, especially if those formats are directly supported by the hardware of the mobile device [36], and (3) compression, because some types of 3D graphics are very suitable for additional lossless compression prior to the delivery [37].

2.2.2 *Delivery Phase*

The delivery phase covers various optimization methods which aim at making the delivery of 3D graphics as effective as possible without altering the actual content of the 3D deliverable. The optimization methods in this phase can be categorized into the following approaches: (1) remote caching, where all previously simplified 3D graphics are stored in a remote cache and re-used if similar requests appear [38], (2) traffic shaping, as combining multiple downloadable items into a single bundle will enable sending of larger data blocks instead of smaller ones hence favoring the wireless transmission channel [39], and (3) progressive downloading, in which a low quality version of the 3D graphics are first sent followed by incremental data that increases the visual quality of the 3D graphics [40].

2.2.3 *On-device Optimization Phase*

The on-device optimization phase includes methods that can be used to optimize the downloaded 3D graphics further by taking advantage of the possibly unique features of the mobile device in question. The optimization methods can be divided into the following categories: (1) GPU transcoding, in which the low level data types (e.g., floating point numbers) are transformed into a format which can be efficiently handled by the mobile GPU [41], (2) surface material simplification, in which surface materials are locally simplified to achieve better rendering performance at the expense of visual quality [42], (3) batch optimization, where rendering performance is improved by submitting properly organized batches, i.e., groups of 3D objects sharing the same properties, to the mobile GPU instead of single 3D objects [43, 44], and (4) local caching, in which the 3D graphics optimized for mobile devices are cached to avoid new network requests and re-translations of the previously downloaded 3D graphics [8].

The scientific publications presenting different optimization techniques for the deployment of 3D graphics on mobile devices are listed in Table 2.

2.3 Optimization Targets for 3D Graphics Deployment on Mobile Devices

In this section, three different optimization targets for 3D graphics deployment on mobile devices are described. In addition, the characteristics of mobile hardware are presented in the context of 3D graphics deployment and the optimization targets.

In the deployment of 3D graphics on mobile devices, three different optimization targets can be taken. Based on these targets, the optimization of 3D graphics deployment can be conducted from the standpoint of (1) *visual quality*, (2) *performance*, and (3) *energy*

Table 2 Classification of optimization techniques for deploying 3D graphics on mobile devices

Publication	Remote simplification	Delivery	On-device optimization
[8]		x	
[33]	x	x	
[34]	x		
[35]	x		
[36]	x		
[37]	x		
[38]	x	x	
[39]		x	
[40]	x	x	
[41]			x
[42]	x		
[43]	x	x	
[44]			x
[53]	x		
[54]	x		x
[55]	x		x
[56]	x		x
[57]		x	x
[58]	x		
[60]	x		
[61]	x		
[62]	x	x	
[63]	x	x	
[64]	x	x	
[65]	x		
[66]	x		
[67]	x		
[68]	x		
[69]	x		
[70]	x		
[71]	x		
[72]	x		
[73]	x		
[74]	x		x
[76]	x	x	
[77]	x		
[78]	x		
[79]	x		
[80]	x		
[81]	x		
[82]	x		
[83]	x		
[84]	x		
[85]	x		
[86]	x		
[88]	x		
[89]	x		
[93]	x		
[94]		x	
[95]		x	
[96]		x	
[97]		x	
[99]	x		
[98]	x		
[100]		x	
[101]		x	
[102]		x	
[103]		x	
[104]		x	
[105]		x	
[106]		x	
[108]		x	
[109]		x	
[110]	x	x	
[111]			x

consumption. Optimization from the standpoint of (1) visual quality stands for providing a high number of vertices, high resolution of textures and advanced surface materials (e.g., dynamic lighting), (2) performance stands for providing a high frame rate, a responsive user interface, and when the 3D graphics are transferred over a wireless transmission channel, also the effectiveness of how the available wireless bandwidth is used, and (3) energy consumption stands for providing a long battery life of a mobile device. Figure 2 presents the three optimization targets as a triangle, inside of which each point illustrates the chosen balance between the different optimization targets. When each optimization target is equally emphasized (i.e., there is no actual emphasis), the point is in the center of the triangle as shown in Fig. 2. It should be noted that one can only emphasize one or two optimization targets at a time, while the rest are unavoidably somewhat neglected.

As earlier mentioned, the capabilities of mobile hardware for 3D graphics re-production grow

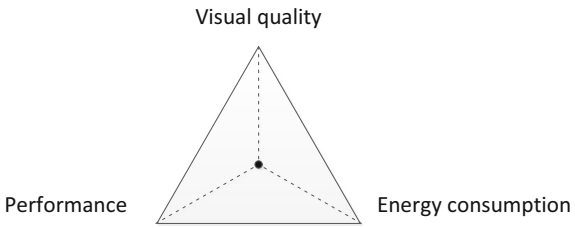


Fig. 2 Different optimization targets for 3D graphics deployment on mobile devices

extremely fast. For instance, 3D rendering performance of Tegra4 is roughly three–four times higher when compared to the previous generation [45]. From the standpoint of mobile 3D Internet applications, this results in increased performance and also in an opportunity for improved visual quality. However, at the same time, the battery capacity is growing very slowly, at only 4 % annually [46]. This emphasizes the importance of energy preservation when deploying 3D graphics on mobile devices.

Xiao et al. [18] present a system level model for estimating the total energy consumption of a mobile device. They split the device hardware into three major blocks, which are: (1) *the chipset*, (2) *the wireless radio interface*, and (3) *the display*. Based on the study conducted by Xiao et al. [18], it can be stated that each of the three blocks accounts roughly for one-third of the overall energy consumption. In their model, Xiao et al. [18] used a backlit display technology that is today being overcome by an OLED display technology. From the energy consumption standpoint, backlit displays contribute by their discrete backlight levels, whereas in OLED displays, each pixel contributes individually. Due to the characteristics of OLED displays [47], their overall energy consumption is greatly influenced by the actual content shown [48, 49]. To reduce the energy consumption of OLED displays [48, 49] propose contrast and color manipulation methods for user interface layouts. However, these methods are not directly applicable to 3D graphics without changing the intended characteristics of the 3D graphics in question. Therefore, the methods for addressing the energy consumption of the backlit and OLED displays are left out from the scope of this article. Next, the characteristics of the mobile chipset and the wireless radio interfaces are presented in the context of 3D graphics deployment and the optimization targets.

2.3.1 The Chipset (CPU, GPU and Memory)

The energy consumption in the chipset is dominated by the CPU and GPU duty cycles, as well as the memory accesses [18]. From the perspective of the CPU, 3D graphics deployment is a very special use case. The CPU needs to (1) manage the downloading of 3D graphics through a wireless radio interface, (2) optimize the 3D graphics based on the capabilities of the mobile device in question, (3) manage the whole 3D scene and its 3D objects, (4) determine the list of visible 3D objects in the user's current view, and (5) issue draw calls to the GPU in each frame update [50, 51]. In order to maintain a high frame rate and the responsiveness of the user interface, attention should be paid on the overall workload of the CPU and the workload split between the CPU and the GPU.

The GPU manages all graphics re-production. For each frame to be drawn, the CPU loads all attributes (e.g., vertices) to the GPU memory and issues all draw calls to the GPU's command queue. The GPU will draw every piece of 3D graphics that is issued in each frame. Hence, it is easy for the developer to unintentionally overload the GPU capacity. Therefore, the best frame rate can be achieved by making sure that (1) the 3D Internet application issues the smallest possible number of individual draw calls to the GPU, (2) the attributes, which are referenced by the draw calls are stored in their simplest possible data type, (3) the draw order of the calls is optimal for the GPU pipeline, and (4) the attributes are cached in the GPU memory whenever possible to minimize their access time [43].

The available memory storage of a mobile device can be a constraint for a 3D Internet application. When a 3D scene is downloaded, the CPU must make sure that all 3D graphics can fit into the mobile device's memory space, this applies both to RAM (for runtime usage) and to permanent memory (for offline usage and local cache). The smaller the pieces of 3D graphics are, the less they reserve memory space but more importantly, the less they contribute to the overall energy consumption when the 3D graphics are accessed [18, 52].

2.3.2 Wireless Radio Interfaces

Mobile devices are connected to the Internet via a wireless radio interface that is usually a WCDMA, an LTE, or a WiFi radio. In a wireless transmission channel, the data transmission is the most efficient when

large data blocks are transmitted at a time [39, 53]. In contrary, continuous transmission of small and fragmented data blocks has a negative impact on the performance, as well as on the energy consumption of a wireless radio interface. This is based on two facts: (1) the performance of the wireless radio interface is at its best when the available wireless bandwidth is maximally utilized, and (2) wireless radio interfaces have in-built energy saving features that set the radio into a sleep state after a certain period of time has passed after the termination of the data transmission. If the time interval between the transmissions of data blocks is very short, the wireless radio cannot enter the sleep state at all, and therefore, consumes significantly more energy.

3 Analysis of Optimization Techniques for Data-Driven 3D Graphics Deployment on Mobile Devices

In this section, we present and analyze an extensive selection of optimization techniques for data-driven 3D graphics deployment on mobile devices. The presentation order of the optimization techniques follows the categorization given in Sect. 2.2. The advantages and disadvantages of each optimization technique are analyzed in the context of the defined optimization targets and the characteristics of mobile hardware described in Sect. 2.3.

3.1 Remote Simplification

In the remote simplification phase, the aim is to simplify the original 3D graphics for a heterogeneous group of mobile devices based on their known or estimated capacities. The influence of the remote simplification phase on the overall performance and energy consumption of a mobile 3D Internet application is crucial. First, the larger the pieces of 3D graphics are transmitted in the delivery phase, the more energy is consumed. As the use of wireless radio interfaces corresponds roughly to one-third of the total energy consumption, it is important to seek an appropriate balance between the level of visual quality and the amount of bytes transmitted. After the delivery phase, 3D graphics can still be simplified in the on-device optimization phase, but at this point, the processing power of the mobile CPU sets limitations to the scale of the optimizations. Second, the simpler the 3D graphics, the less strain is inflicted on the mobile chipset in the rendering phase. Also the mobile chipset accounts roughly for one-third of the total energy consumption, and therefore, conducting the majority of the 3D graphics simplifications already in the remote simplification phase can result in huge performance improvements and energy savings in the rendering phase. The component performing the 3D graphics simplifications can be implemented, for instance, in a network proxy [38]. The optimization techniques in the remote simplification phase include creation of LODs, data type transformations and compression.

3.1.1 Level-of-Detail

Creation of so-called LODs for 3D graphics means that in addition to the original (high quality) version of 3D graphics, there also exists a collection of lower quality variants of the same content. The lower quality variants are smaller in file size, which typically results in decreased processing load on the mobile chipset and in reduced use of the wireless radio interface. In the context of the defined optimization targets, LODs can be used for improving the performance and for lowering the energy consumption at the expense of the visual quality.

In general, the LODs can be categorized into *discrete* and *progressive* counterparts. Discrete LODs have a known number of low quality variants (e.g., five discrete levels) for each data type including geometry, textures and surface materials. For geometry, this stands for groups of geometry with less geometric detail computed from the original 3D model [54]. For textures, this stands for groups of images down scaled from the original image [34]. For surface materials, this stands for a reduced number of parameters which force simplified calculations of the surface shading effects [55]. Progressive LODs, in turn, provide additional details as a continuous stream instead of discrete levels [33]. Progressive LODs enable incremental reconstruction of 3D graphics in a way that each new increment enhances the quality of the 3D graphics. However, methods for creating progressive LODs can also be used for producing discrete LODs. In this case, certain fixed levels are chosen; say 25, 50 and 75 % of the detail in the original piece of 3D graphics and the simplification results are stored as such instead of the information used for reconstructing the 3D graphics progressively.

When comparing the advantages and disadvantages of discrete and progressive LODs from the standpoint

of mobile devices, the following four aspects need to be emphasized: (1) discrete LODs do not require any decoding time on the CPU, however, they may require some processing on the GPU [56]. Progressive LODs, in turn, typically require substantial amount of time for decoding which burdens the mobile CPU, (2) as each discrete LOD is an independent piece of 3D graphics, their use increases the amount of redundant information transmitted. With progressive LODs, the amount of redundant information transmitted is negligible typically resulting in lower use of wireless bandwidth and local storage on the mobile device, (3) with discrete LODs, the overall processing burden on the mobile can be more easily estimated than with progressive LODs. This enables tailoring the discrete LODs based on the capabilities of the target mobile device if its capabilities are known, and (4) multiple discrete LODs can be used in parallel, for instance, to enable distance-based adjustment of the rendering detail, but a progressive LOD is a single entity containing 1–100 % of the detail in the original piece of 3D graphics. It is possible to continuously decode and encode the progressive 3D graphics in real-time to achieve different LODs on the need basis, however, this may excessively burden the mobile chipset [57]. Alternatively, a mobile device can derive multiple LODs from the progressive 3D graphics and cache them in the memory.

When evaluating the suitability of various simplification methods for creating LODs, firstly, it is the most important that the methods do not require significant amounts of processing on the mobile device itself. Instead, the majority of the processing should already be conducted remotely on the network, where also more complex simplification methods can be used. Secondly, the simplification methods should provide a compact presentation of 3D graphics to lower the requirements for the wireless bandwidth and the local storage on the mobile device [56]. Thirdly, the simplification methods should preserve the vertex attributes such as per-vertex normals, texture coordinates and color information to retain adequate visual quality of the 3D graphics [58]. Finally, it is important to emphasize that this article focuses only on the simplification methods that are potentially suitable for deploying 3D graphics on mobile devices.

3.1.1.1 Geometry Simplification When the original geometry is computationally reduced, there is always damage done to the original 3D model as illustrated in Fig. 3. Geometry simplification can focus on (1) reduction of geometry data (i.e., vertices) or connectivity data (i.e., triangles), or (2) compression of geometry data (i.e., vertices) without altering the connectivity data [60]. Hence, the geometry simplification methods strive to fulfil the following three goals: (1) to minimize the local error, which is caused by individual vertex/edge/triangle removals or compression, (2) to preserve the global characteristics of the simplified 3D model to maintain its shape, and (3) to avoid causing discontinuities (e.g., holes or orphan edges) to the simplified 3D model. For simplifying geometry, the literature introduces various methods that focus on simplifying different features of geometry. These methods include, but are not limited to, edge collapse [33, 61], geometry quantization [40, 62, 63], geometry decimation [64], triangle decimation and cleansing conquest [40, 62, 65, 66], mesh partitioning [56] and vertex clustering [54]. It should be noted that this article does not focus on the algorithmic details of each individual method, since this information is already comprehensively summarized, for instance, in recent surveys [60, 67].

Fig. 3 An image presenting three LODs of geometry simplification [59] (reprinted with permission)

From standpoint of LODs, the methods presented in the following paragraphs can be split into three groups: (1) the methods which produce progressive LODs by storing the reconstruction information to enable progressive decoding of the geometry, (2) the methods which produce discrete LODs without storing any reconstruction information, and (3) the methods, which are not fully neither of the two above, but instead are implementations of special geometric cases.

Table 3 The comparison of different geometry simplification methods

Sources	D/P LOD	Encoding: triangles/s	Decoding: triangles/s	Compression: bits/vertex	Testing hardware
Method by Willmott [54]	D	160k	None	*N/A*	Intel Core i7 CPU
Method by Lee et al. [56]	D	*N/A*	None	>24	*N/A*
QEM-original [61]	D	49.4k	None	*N/A*	2.5 GHz Intel Duo, 2 GB RAM
QEM-improved [61]	D	25.8k	None	*N/A*	2.5 GHz Intel Duo, 2 GB RAM
QEM-original [65]	D	4.8k	None	*N/A*	195 MHz, SGI Indigo2, 128 MB RAM
Edge collapse [57]	P	~86k	100k	~487	200 MHz Pentium Pro CPU
Method by Lee et al. [62]	P	32k	23k	15–20	2.8 GHz dual-core CPU, 4 GB RAM
Method by Lavoué et al. [63]	P	*N/A*	27k	56	2.0 GHz laptop
Method by Maglo et al. [64]	P	93k	122k	~16	2.8 GHz Intel Core i7 CPU
POP buffers [40]	D/P	5.889k	None	*N/A*	2.4 GHz Intel Core i7, 4 GB RAM
Batched dynamic adaptive meshes [68]	D	5.2k	*N/A*	*N/A*	1.6 GHz AMD Athlon MP, 2 GB RAM

Table 3 summarizes the benchmark results from the literature related to the geometry simplification. The results related to encoding and decoding are normalized to triangles/s to allow easier comparison. It is important to note that the performance of the same method [e.g., quadric error metric (QEM)-original] can vary significantly when benchmarked at a different year and with a different computer. In the following paragraphs, different methods for geometry simplification are presented in more detail.

Hoppe [33, 57] introduced a well-known progressive method called edge collapse, where edges can be individually removed from the geometry until there is nothing to remove. The original method is rather fast, being able to collapse and re-construct approximately 100k vertices/s (Pentium Pro 200 MHz CPU) which equals to triangles/s since a single collapse operation removes one triangle. However, the compression ratio of 487 bits/vertex (bpv) on average is poor compared to the more recent methods, and hence, use of edge collapse may become overwhelming for larger pieces of geometry both in terms of the required wireless bandwidth and the memory. However, based on the pioneering work by Hoppe [33], many more mobile-friendly methods have been later introduced.

Lee et al. [62] introduced a progressive method based on geometry quantization. Their method is lossless, which means that the original geometry can be progressively decoded with 100 % accuracy. Their semi-accurate version of the method, which uses quantization bit width estimation to potentially skip the burdensome triangle decimation conquest step, is relatively fast, being able to encode 32k triangles/s and decode 23k triangles/s with 2.8 GHz dual-core CPU and 4 GB of RAM. When the geometry distortion is kept low, their charts indicate compression rates of 15–20 bpv. However, only color information is preserved for the vertices. Maglo et al. [64] present a progressive method based on geometry decimation and re-encoding of the decimated surfaces based on the new edge information. The method is able to perform encoding at a rate of 93k triangles/s and decoding 122k triangles/s when using Intel Core i7 CPU. The average compression rate achieved with their test triangle meshes is around 16 bpv. The method is very fast, but does not retain any vertex attributes. The progressive method presented by Lavoué et al. [63] is based on hierarchically applied geometry quantization. Lavoué et al. [63] do not present any estimates for the encoding time, but emphasize that their method is able decode data (tested with a 2 GHz laptop) at an average speed of 27k triangles/s. Their method achieves compression rate of 56 bpv, but as Lavoué et al. [63] state, they have given up some compression performance to enable faster decoding time. The method, however, preserves only color information for the vertices. Progressively ordered primitive (POP) buffers [40] represents a state-of-the-art method for creating progressive LODs. The method is based on geometry decimation and triangle conquest. From the standpoint of mobile devices, it should be emphasized that the method does not require any decoding on the mobile device, since the 3D graphics provided are suitable for the mobile

GPU rendering without any modifications. Limper et al. [40] do not provide any comparable compression rates, but they do state that 5 bits for the quantization resolution of a single vertex is enough to bring Hausdorff error (RMS) below 1 % using the Stanford bunny test mesh as input. Furthermore, the method preserves all vertex attributes. The method may or may not be treated as a pure progressive method, since the data is not decoded one vertex/edge/triangle at a time, but instead as groups of additional geometry which continuously refines the downloaded geometry. Therefore, each new increment of geometry acts also as a new discrete LOD. However, since every new group of data is concatenated to the previous data, we classify the POP buffers method as a progressive method.

The methods for creating discrete LODs are many, and thus, only a small number can be presented in this article. In QEM-based methods, the error caused by the edge removal is minimized. The original QEM-based method was introduced by Garland and Heckbert [65]. Garland and Heckbert [66] have also presented an improved version of the original method that preserves all vertex attributes, and since then, several further improved versions have been proposed. For example, Wei and Lou [61] added a feature sensitive metric to the original method, which utilizes surface normal deviation in addition to geometric distance of the removed vertices for error minimization. Their version of the method produces simplified 3D models with a lower error, but at the expense of additional computational complexity approximately doubling the required encoding time when compared to "Qslim" tool which is commonly used for creating QEM-based geometry simplification. With 2.5 GHz Intel CPU, the method is able to encode approximately 50k triangles/s. The method presented by Wei and Lou [61] does not preserve vertex attributes but they introduce several ways how this could be achieved as well. Lee et al. [56] present a method based on geometry quantization and mesh partitioning. They do not report any required encoding time, but instead, they emphasize that their method is suitable for mobile devices due to its simple de-quantization requirement. The de-quantization operation is managed via a single matrix multiplication, which can be effortlessly implemented by a mobile GPU. The compression rate provided by the method is at its best 24 bpv. The method does not preserve any additional vertex attributes. Willmott [54] presents a method based on vertex clustering. The method is very fast and is also able to preserve texture coordinates and per-vertex normals. The method is able to simplify geometry at a rate of 160k triangles in 40 ms on average when using Intel Core i7 CPU. Although Willmott [54] has not run the method directly on any mobile device, the execution speed gives a hint that the method could be runnable directly even on a mobile device when simplifying smaller pieces of geometry.

Batched dynamic adaptive meshes [68] presents a specialized geometry simplification method based on modified edge collapse to favor the characteristics of terrain geometry. The method is able to adaptively reduce the geometric complexity of varying planar surfaces, like terrain, which have less variation in their content. The method may also be suitable for mobile devices when creating discrete LODs for planar surfaces, but it is not generic enough to be used universally for any arbitrary geometry. Cignoni et al. [68] do not present any exact times for the geometry encoding and decoding, but instead, for the complete execution time which also includes the creation of the terrain texture. This total processing time yields a figure of 5,200 triangles/s, but Cignoni et al. [68] state that in the total processing time the texture generation is actually dominating.

3.1.1.2 Texture Simplification For textures used in the surface modelling, the LOD creation is done by creating a series of scaled down variants of the original image (i.e., MipMaps [34]). In addition, there are intermediate methods for enhancing the LOD creation process. These methods can be used to enhance the image in a way that the image degradation during the LOD creation is minimized.

Sloan et al. [69] present a texture coordinate optimization method for enhancing specific areas of the image data based on their importance. The method warps the texture to allocate more spatial space to the regions of the image containing more detail. Warping the image also causes that the original texture coordinates of the geometry need to be recalculated. Sloan et al. [69] use image decomposition methods to conduct the recalculation of the texture coordinates and show examples of the warped textures. Pre-processing the image with this kind of method produces textures with better visual quality, however, with the same resolution (i.e., the same texture size) as the original texture. There are also other methods

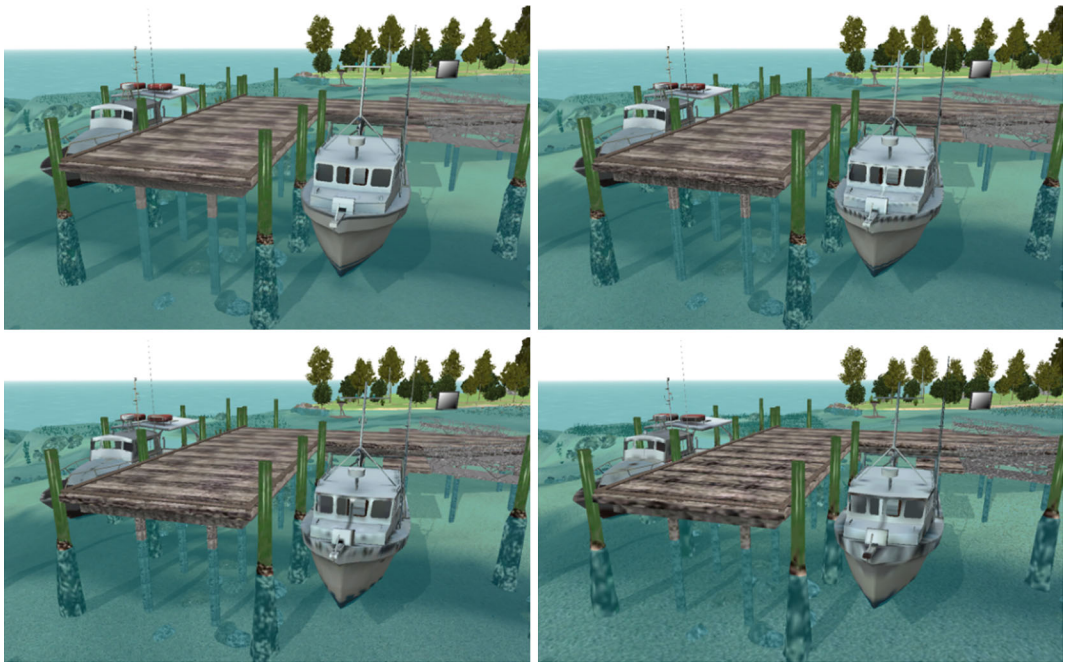


Fig. 4 *Top left* image presents the original version of the 3D graphics: the *top right*, the *bottom left* and the *bottom right* images present LOD levels, where all texture files have been reduced to 25, 13 and 6 % in size, respectively

which are similar in principle, but use different algorithms to perform the texture coordinate optimization. The method introduced by Hunter and Cohen [70] is based on the modification of the image frequency space. Their approach removes high and low frequency detail to harmonize the overall image frequency range and hence allow compression (scaling down) of the original image data. Balmelli et al. [71] also use frequency space analysis to determine the regions with high frequency content. In their work, the texture is warped to allocate more space for the identified regions of high frequency content. Balmelli et al. [71] show examples in which the original image is optimized and then scaled down to 80 % of the original size. The examples illustrate that the down scaled optimized image provides better visual quality than the down scaled original image. Likewise, Sander et al. [72] allocate more texture space for those regions of the image requiring more detail perseverance. Their method is based on analyzing the geometric surface details of the original 3D graphics, where the texture will be used using the derived signal-stretch metric computed by the Taylor expansion.

For creating MipMaps [34], a series of down scaled variants of the original texture, a single mobile GPU API call can be used if the mobile device is compatible with the OpenGL ES specification. This makes use of MipMaps very convenient for the developer. Texture coordinate optimization methods can also be used before conducting MipMapping, however, this requires additional effort from the developer. As the screen size of a typical mobile device is rather small, it should be first determined whether the better visual quality is actually recognizable [73]. Figure 4 illustrates four LODs from the same 3D scene, where the texture files have been reduced to 25, 13 and 6 % in size. The boat starts to show artifacts especially in its surface when using the 6 % LOD size (lower right corner).

When creating LODs for textures, some mobile GPU limitations should to be kept in mind. First, textures which are not power of two in terms of size in pixels are inefficiently rendered or are not rendered at all with a mobile GPU. This is as specified by Khronos in their OpenGL ES 2.0 specification [74]. Second, the use of an unnecessary large texture file together with its corresponding MipMap levels may lead in some cases to a situation that the original texture is not used at all [34]. As a consequence, a large texture file may remain allocated in the memory of the mobile GPU

needlessly, yet consuming the limited memory capacity. The principle of progressiveness applies also to textures. For example, JPEG compression format contains information indicating whether an image is encoded in layers, which can be downloaded and decoded progressively on-the-need basis [75, 76].

3.1.1.3 Surface Material Simplification In a typical case, there are a large number of surface materials that describe all the surface characteristics of a 3D scene. They describe, for instance, how the lighting works [77], how the reflectance works for each 3D model [78, 79], how the shadows should be calculated [42, 80] or whether the surfaces have details such as bumps in them or not [81].

Lighting and reflectance are computationally complex due to the mathematics involved in solving the lighting equation for each pixel drawn on the display. Since the lighting equation is solved by a GPU fragment processor, the bottleneck comes from the speed of a single fragment unit, and from how many parallel units are available for the computations. The complexity accumulates with the number of light sources and becomes easily intolerable for a mobile GPU which does not possess the same level of parallelism compared to a desktop GPU. The complexity of real-time shadow computation varies with the number of light sources, but also according to the type of light sources of which the point lights are particularly difficult to handle in a robust way. Shadows caused by directional light sources can be estimated with orthographic projections of the shadow volumes, since the light is assumed to traverse to a single direction only. For the point lights, this is not the case, and more complex methods have to be used.

Without modifying the underlying geometry, the details of the surface can also be artificially enhanced using various methods including (1) normal mapping [82], (2) parallax mapping [83], (3) relief mapping [81], and (4) displacement mapping [82]. The additional surface details include, but are not limit to, surface bumps, cracks, light refraction, scattering effects and shadows. For instance, Policarpo and Oliveira [81] create artificial bumps by using a relief map algorithm that relies on intensive sampling of bump texture in the fragment shader. Dmitriev and Makarov [82] show how vertex displacement maps can be computed directly from normal maps to improve the geometric details of the surfaces. Kaneko et al. [83], in turn, illustrate how parallax mapping can enhance the impression of depth on the surface of a 3D object.

All these enhancements to surface details could be computed in the rendering phase, however, as the methods are mathematically very complex, the mobile GPU could be potentially overwhelmed. This issue can be overcome by pre-computing the surface detail effects already in the remote simplification phase and by storing the effect-specific values in a texture map. As a result, the computational complexity is reduced down to a single value fetch from the given texture map. This process in general is called mapping. However, it should be noted that the use of each these methods generates an additional texture map file, which needs to be transmitted to the mobile device prior rendering. In this case, the enhanced visual quality comes with the cost of increased amount of data transmitted. A mobile GPU being often limited by the number of hardware texture samplers finds these methods challenging due to lack of available parallelism. All surface definitions are eventually derived into shader programs, which are executed by the mobile GPU. As a rule of thumb, the more complex the surface definitions, the more complex will the shader programs be. Consequently, this inflicts higher burden on the mobile GPU, and thus, increases the energy consumption during the rendering process.

The surface descriptions can be redefined in the remote simplification phase to reduce computational complexity of the given surfaces. Hosseini et al. [55] have presented how some of the surface material simplification methods (forcefully simplified lighting calculations and reduced texture brightness) can lower smartphone energy consumption by 20–33 %. In example images, presented by Hosseini et al. [55], there are noticeable degradations in the visual quality, but if used in conjunction with LODs, it can be argued whether those degradations are noticeable from a distance. Likewise Mochocki et al. [35] show that by manipulating the low level GPU parameters (such as the GPU texture addressing modes and the lighting types) up to 50 % of energy savings can be achieved. This kind of forceful surface material simplification can be considered as creation of LODs, since the process results in a varying set of surface materials with different levels of complexity. It should be noted that 3D objects further away can be easily rendered without complex surface materials without the user

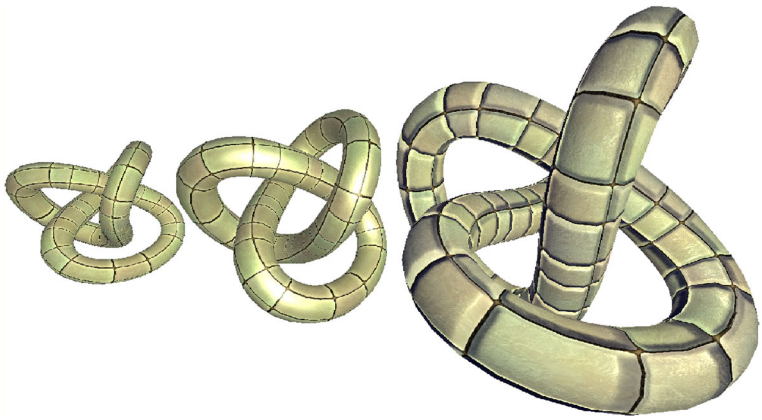


Fig. 5 (*Left*) a 3D object with only surface color texture and basic illumination, (*middle*) the same 3D object with surface color texture and per-fragment specular highlight computed based on the illumination parameters and (*right*) the same 3D object with surface color texture, detailed per-fragment normal and parallax bump maps, as well as Cook–Torrance illumination model

actually noticing the change [55]. Alternatively, Koulieris et al. [84] propose that LODs for surface materials can also be created based on context in which the 3D objects appear. This approach is based on the assumption that users have higher tendency to recognize the lack of detail in individual and isolated 3D objects than in compact groups of 3D objects. In Fig. 5, an example 3D object is rendered three times with different distances and surface material LODs.

The simplification of surface materials always has more global effect on the visual quality of the 3D scene compared to the texture and especially geometry simplification. The reason is that usually surface materials (i.e., shadows and lighting) have influence on more than one 3D model at a time, and hence the simplification effects may be cumulative [38, 85]. In overall, the simplification methods used in the creation of LODs are very efficient, but not perfect. It should be noted that sometimes the methods can fail especially if there are discontinuities in the geometry of the original 3D model (e.g., holes or orphan edges). These failures can appear as misaligned texture coordinates causing incorrect results in the surface shading, or they can result in incorrect mesh topologies. Vatjus-Anttila et al. [38] and Pool et al. [85] show visual examples of when the simplification process causes loss of the 3D model integrity. They also emphasize that the creation of LODs for textures may result in the disappearance of important details in the 3D models.

3.1.2 Data Type Transformation

For 3D graphics, the data type transformation enables use of data types which are the most appropriate according to the available wireless bandwidth and the processing capacity of the mobile chipset. In the context of the defined optimization targets, data type transformation can be used for adjusting the balance between the performance and the energy consumption.

When conducting data type transformations, the following two aspects need to be examined [37]: (1) the load caused on the mobile chipset, and (2) the required wireless bandwidth. As the data type transformation is conducted remotely on the network, it is not typically crucial (except in collaborative 3D Internet applications [40]) even if the data type transformation is computationally a demanding task. Instead, the more important is the amount of processing that is required to be done by the mobile chipset.

In case of textures, transforming hundreds of JPEG images into compressed texture formats such DXT or ETC may take some time on the network, but if the mobile GPU supports these compressed textures, zero processing time is required from the mobile CPU. In case of JPEGs, the mobile CPU always needs to decompress the JPEG images into RGB data before the mobile GPU is capable of rendering the textures [37]. In addition, compressed textures also require less RAM memory, and consequently, require a smaller number of memory accesses in the mobile chipset. According to

Vatjus-Anttila et al. [86], use of compressed textures typically results in four–six times lesser use of RAM memory. They concluded that from the standpoint of the mobile chipset, the use of JPEGs (i.e., RGB data) resulted approximately in 5–15 % higher energy consumption than the use of compressed textures.

The OpenGL ES 2 specification [36, 74] defines that a compatible mobile device may optionally support one or multiple compressed texture formats. Although the majority of today's mobile devices are OpenGL ES 2 compatible (e.g., 100 % of Android devices [87]), support for compressed textured is very limited, and primarily only the mobile devices equipped with NVIDIA graphics hardware support DXT. Due to this trend, the OpenGL ES 3 specification defines ETC2 [88] as the sole compressed texture format that must be supported by all compatible mobile devices. Currently, 22.5 % of Android devices, for example, are OpenGL ES 3 compatible [87], but in the future, the number will evidently rise.

From the standpoint of wireless transmission channel, compressed texture files require more bandwidth than, for example, JPEG files. Ström and Wennersten [37] estimate that the size of a JPEG file is typically 60 % smaller than the size of a compressed texture file with the same visual quality. Therefore, the delivery of compressed texture files introduces a negative effect both on the performance and the energy consumption. However, by further using domain-specific lossless data compression for the compressed texture files, the difference in file size can be eliminated [37].

It should be noted that textures are not the only types of 3D graphics that may benefit from the data type transformations. Other practical examples include hierarchical structures of geometry and animation data. Gil and Trezentos [89] evaluated the impact of XML, JSON and a binary format (protocol buffers) on the energy consumption and performance of smart phones. They concluded that the most efficient format is JSON in terms of file size, parsing time and energy consumption if it is used with additional lossless compression (e.g., gzip). When transmitting large amounts of data and additional lossless compression is too heavy or even impossible to implement, the binary format proved to be the most suitable option.

3.1.3 *Compression*

In case of 3D graphics, the need for the wireless bandwidth can be further lowered using lossless compression algorithms. These algorithms are general purpose and can be applied on any kind of 3D graphics prior transmission. In the context of the defined optimization targets, compression can be used for improving the performance and the energy consumption.

In lossless compression, the original data can be retrieved from the compressed data exactly, i.e., none of the original bits are lost in the process [90]. Use of compression, however, increases the complexity of the data, since the receiving mobile device needs to decompress the data before using it. This can result in increased load on the mobile CPU, and as a consequence, in higher energy consumption. However, Gil and Trezentos [89] show that even with the additional CPU load, the overall energy consumption of the data processing can be decreased up to 30 % when lossless compression is used. This emphasizes the importance of the selection for the lossless compression algorithm. In case of mobile devices, the algorithms should (1) provide good compression ratio to minimize the use of wireless bandwidth, and (2) be computationally simple not to exhaust the mobile CPU by the decompression effort. From the standpoint of energy consumption, the setup is optimal when the energy savings achieved through the decreased use of wireless bandwidth are higher than the energy spent in the decompression process. From the standpoint of performance, lossless compression decreases the time needed for wireless data transmission, but increases the number of the CPU duty cycles, which may have impact on the frame rate and on the responsiveness of the user interface.

gzip [90] is a widely used lossless compression algorithm due to the availability of a stable implementation on multiple platforms. It is based on DEFLATE algorithm, which is eventually based on LZ77 [91] and Huffman coding. gzip is among one of the original accepted HTTP content encoding schemes in RFC2616 [92]. Inclusion in RFC2616 has significantly increased the adoption of gzip, and today, majority of modern mobile devices implement the gzip algorithm. There are also several other lossless compression algorithms available as de facto standards, but either their complexity [93] or incompatibility with gzip has limited their adoption. Other lossless compression algorithms include, for instance, LZ78 [94] and Lempel–Ziv–Welch (LZW) [95] that are also successors of the original LZ77 algorithm, Burrows–Wheeler-Transform (BWT) [96], LZMA, prediction by partial matching (PPM) [97, 98] and Google's Zopfli [93].

Table 4 The comparison of different lossless compression methods in terms of performance and energy consumption

	LZ77	LZW	BWT	PPM	gzip	Zopfli	Median
(1) Compression ratio	0.46	0.39	0.24	0.23	0.29	0.27[a]	0.28
(2) Compression time (s)	0.45	2.0	11.75	4.75	6.3	504[a]	5.53
(3) Decompression time (s)	0.25	0.7	2.8	5.35	0.2	0.2[a]	0.48
(4) Energy consumption for data compression and transmission (J)	4.2	3.8	7.9	7.3	7.2	N/A	7.2
(5) Energy consumption for data reception and decompression (J)	3.3	2.9	3.5	7.7	2.2	N/A	3.3

[a] The estimates were calculated based on the reported performance difference between gzip and Zopfli [93]

Barr and Asanović [99] have compared LZ77, LZW, BWT, and PPM in terms of compression, transmission and decompression performance. They also examined how the algorithms cumulatively affect the total energy consumption of a mobile device. When comparing gzip with the other algorithms (see Table 4), its (1) compression ratio is average, (2) compression time is one of the longest and (3) decompression time is the shortest. Finally, from the energy consumption perspective, gzip provides (4) average energy efficiency for data compression and transmission, but more importantly, (5) the best energy efficiency for data reception and decompression.

Especially from the energy consumption standpoint, these results make gzip a very suitable lossless compression algorithm for mobile devices. However, if the wireless bandwidth is very scarce, use of BWT or PPM should also be considered, since they provide better compression ratio.

When compared to gzip, Google's Zopfli [93] algorithm provides an equivalent decompression performance and 6 % better compression ratio on average, but at the same time, requires roughly 80 times longer compression time (see Table 4). For the advantage of Zopfli, the algorithm is compatible with gzip. Very good compression ratio makes Zopfli suitable for 3D Internet applications, where the 3D graphics are static, i.e., mobile clients do not create and modify the 3D graphics on-the-fly [40]. This is due to the fact that the CPU load caused by the compression process is too high for a mobile device. However, a 3D Internet application could be built in a way that the mobile device uses, for example, gzip for the compression prior to sending the 3D graphics. In this case, the decompression using gzip and the re-compression with Zopfli would be handled by the server afterwards.

3.2 Delivery

In the delivery phase, the aim is to save energy and minimize download latencies by enabling efficient transmission of 3D graphics and optimal use of the wireless transmission channel based on its characteristics. 3D Internet applications introduce quite different requirements for the wireless network when compared to traditional multimedia applications (e.g., video streaming). In many cases, a relatively large amount of 3D graphics needs to be first transmitted before the user can see a meaningful portion of the 3D scene. Therefore, high bandwidth connection is typically required. In multi-user applications, also low latencies are required to enable fast propagation of users' activities [38]. Furthermore, delivery of 3D graphics is also vulnerable to packet loss, because all the lost packets need to be retransmitted or a 3D object cannot be correctly reconstructed. As a comparison with a video stream, a packet loss may merely result in a few lost frames, which does not severely affect the decoding of other incoming packets. To enable reliable and ordered delivery, TCP should be primarily used as a transport protocol for 3D graphics. For propagating short-lived status updates in multi-user applications, also UDP can be used. The optimization techniques in the delivery phase include remote caching, traffic shaping and progressive transmission.

3.2.1 Remote Caching

In general, the objective of caching is to store data that the incoming requests can be served faster in the future. This results from the fact that a cache fetch typically occurs in a fraction of a second, whereas processing large datasets can take several seconds or

tens of seconds of time. In the context of the defined optimization targets, remote caching can be used for improving the performance.

The simplification of the original 3D graphics may take time and it may not always be possible to store all the simplified versions of 3D graphics on the servers. In these cases, use of remote caching may significantly decrease the processing delays. Furthermore, the connection between the servers providing the 3D graphics and the mobile devices may suffer from high latencies and/or low available bandwidth. If the remote cache is implemented in the mobile operator networks, significantly lower latencies and higher available bandwidth can be achieved between the remote cache and the mobile devices. As a consequence, remote caching may also significantly decrease the time needed for acquiring the first meaningful portion of the 3D scene when starting a 3D Internet application. In overall, the benefits of remote caching are especially notable when dealing with slow wireless network connections [38, 100, 101].

A widely used method for building the remote cache is to use a proxy server [101] which has a cache memory associated with it to store all the passing data. Caching of 3D graphics is a special case for a proxy server, since there are multiple instances of the same 3D graphics separated only by their LODs. The challenge is how to efficiently index this information and how to reliably provide the most appropriate piece of 3D graphics to a requesting mobile device based on its capabilities. An extensive survey of different types of caching strategies has been presented, for instance, by Podlipnig and Böszörmenyi [102].

3.2.2 *Traffic Shaping*

As explained in Sect. 2.3, the data transmission in a wireless medium is the most efficient when large data blocks are transmitted at a time [53]. Instead, frequently sent small data blocks do not allow the wireless radio to enter the sleep state at all, which has a significant negative effect on the battery life. Although the characteristics of wireless radio interfaces such as WCDMA, LTE and WLAN are rather similar from the application viewpoint, there exist some differences that should be taken into account when optimizing the use of the wireless transmission channel [103, 104]. In the context of the defined optimization targets, traffic shaping can be used for lowering the energy consumption at the expense of some performance.

WCDMA has four different operational states that are DCH, FACH, PCH and IDLE sorted from the highest to the lowest power drain [105]. In DCH, FACH and PCH states, the energy consumption stays relatively constant. WCDMA also possess specific inactivity timers controlling the transitions between the operational states. The values of inactivity timers are set by the operators, and are typically less than 10 s [104]. With WCDMA, the inactivity timers are rather long, which results in unnecessary lingering in high-power states after a completed data transfer. As a consequence, additional energy (typically called tail energy) is consumed which may comprise nearly 60 % of the total energy consumed [106]. In order to decrease the amount of tail energy, an additional mechanism has also been implemented to allow moving from the DCH state to the FACH state, when there is only low level of traffic [105]. According to measurements conducted in real-life WCDMA networks, the latency was 180 ms and the downlink throughput 0.763 Mbps on average [107] representing a minimum requirement for 3D Internet applications.

The LTE also has four different operational states that are ACTIVE, Short DRX, Long DRX and IDLE sorted from the highest to the lowest power drain [108]. In ACTIVE state, the energy consumption stays relatively constant, but in both DRX states, the wireless radio is periodically turned on and off to save energy and to check for new incoming traffic. Although LTE has rather similar kind of timers as WCDMA, the DRX power saving mechanism is more sophisticated due to the intelligence added to the base stations [103]. For instance, the operation of DRX (i.e., timer values) can be adjusted according to the pattern of the incoming traffic. Typically, the transition from the ACTIVE state to the first power saving state (i.e., Short DRX) occurs in less than a second [104]. This significantly improves the energy consumption of LTE especially with bursty traffic. However, it should also be noted that the use of the LTE radio interface consumes more energy than the WCDMA radio interface [108]. According to measurements conducted in real-life LTE networks, the latency was 70 ms and the downlink throughput 9.185 Mbps on average [107] making LTE well suited for 3D Internet applications.

WLAN has two different operational states that are awake mode (CAM) and power save mode (PSM). WLAN does not have any inactivity timers, and hence, it switches to PSM right after the completion of the data transfer. With two modern mobile devices, Vergara and Nadjm-Tehrani [105] measured that switching to PSM took only 70 and 220 ms. This makes WLAN very efficient for data transfer in terms of energy consumption. However, the overhead caused by the association with a WLAN access point may be comparable to the tail energy consumption of WCDMA. In terms of performance, WLAN is somewhat comparable with LTE. However, the coverage of LTE is much wider and will continue to grow in the upcoming years.

For shaping the traffic, both the client and the server should implement send buffers. At the client side, the requests, and at the server side, the responses are bundled together to form larger packets that occupy the wireless transmission channel more efficiently. For their 3D Internet application, Nurminen [8] organized the send buffers as tail queues that allow re-organization of the buffers before the transmission via TCP. Vatjus-Anttila et al. [38] presented an implementation for a virtual world client, where all sent and received TCP/UDP packets were bundled and compressed aiming for maximum MTU transmission. The experiment lowered both packet rate ($\sim$88 % uplink, $\sim$61 % downlink) and wireless bandwidth requirements ($\sim$76 % uplink, $\sim$70 % downlink). The savings are clear, but as Vatjus-Anttila et al. [38] indicate, forceful manipulation of the packet flow may cause other problems, for instance, in the prediction algorithms which estimate movements of 3D objects in a 3D scene (dead reckoning [109]). By bundling (and hence delaying) the sent packets broke the dead reckoning logic, which assumed an update interval of 50 ms in this particular example. However, for the less important status updates in a 3D scene the effect was not so obvious, hence, indicating that for some types of traffic bundling and compression can be feasible.

3.2.3 Progressive Downloading

When the 3D graphics are simplified using progressive methods (no discrete LODs are created), the 3D graphics also need to be delivered in a progressive manner. In the context of the defined optimization targets, progressive downloading can be used for adjusting the balance between the performance and the energy consumption.

Use of progressive downloading enables a mobile device to quickly display a meaningful proportion of a 3D scene, however, typically with a low visual quality [110]. Therefore, progressive LODs are generally more suitable than discrete LODs for the delivery of 3D graphics when available wireless bandwidth is very scarce. As the 3D graphics are transmitted as a stream of detail, progressive downloading is not so vulnerable to packet losses compared to downloading discrete LODs. All lost packets must be resend, but the previously received increments of detail can already be rendered. With discrete LODs, the complete file must be successfully received before the rendering can start. It should also be noticed that typically less wireless bandwidth is required when the complete piece of 3D graphics is progressively downloaded compared to separately downloading multiple discrete LODs. However, by downloading only one discrete LOD with the highest visual quality, less bandwidth is typically required compared to progressively downloading the complete piece of 3D graphics.

3.3 On-device Optimization

In the on-device optimization phase, a 3D Internet application still has an opportunity to optimize the 3D graphics based on the known capabilities of the mobile chipset and/or the underlying rendering engine to facilitate the upcoming rendering phase. As the processing power of mobile CPUs is limited and the optimizations are typically run in real-time, the changes to the 3D graphics need to be significantly smaller in scale compared to the remote simplification phase. Therefore, their effect on the overall performance and energy consumption of a 3D Internet application is typically smaller. However, it should be noted that the optimizations done in the on-device optimization phase can be fine-tuned more precisely according to the specific hardware features of the mobile GPU in question. The optimization techniques in the on-device optimization phase include GPU transcoding, surface material simplification, batch optimization and local caching.

3.3.1 GPU Transcoding

In the GPU transcoding, primitive data types of the attributes (e.g., vertices) are transcoded into data types

that are more favorable to handle by the mobile GPU. In the context of the defined optimization targets, GPU transcoding can be used for improving the performance and for lowering the energy consumption potentially at the expense of minor decrease in the visual quality.

Geometry data is often delivered in floating point numbers, which usually require storage space of 32-bits per floating point (single precision IEEE floating point number). Several mobile GPUs that support OpenGL ES specification implement a programming extension "OES_vertex_half_float", which allows using 16-bit floating point numbers instead of 32-bit [74]. In general, 32-bit floating point numbers can be converted into 16-bit equivalents without straining the mobile CPU and without compromising the visual quality of the 3D graphics. From the standpoint of the mobile chipset, this result in notable savings in the use of internal bandwidth and memory, which further contributes to better performance and lower energy consumption [41]. It should be noted that the same principle can be applied to all data types (i.e., 8-, 16- or 32-bit triangle indices as required, or fixed point 8- or 16-bit vertex data [56]).

3.3.2 Surface Material Simplification

In addition to the remote simplification phase, surface materials can also be simplified later in the on-device optimization phase by the mobile device itself. In the context of the defined optimization targets, surface material simplification can be used for increasing the performance and for lowering the energy consumption at the expense of the visual quality.

If the capacity of a mobile device is either identified or estimated incorrectly, the remotely conducted surface material simplification may result in unnecessarily complex surface materials. In addition, the user of a 3D Internet application may prefer performance over visual quality. In these cases, the surface materials should still be further simplified in the on-device optimization phase. For example, the mobile device can ignore a large number of light sources and combine them into one directional light, and hence simplify the lighting calculations [111]. The surface material definitions directly affect the complexity of the locally generated shader programs, and hence, the rendering performance increases at the expense of visual quality [55]. As a consequence, the GPU duty cycle shortens, and thus, the rendering of a single frame consumes less energy.

3.3.3 Batch Optimization

In the batch optimization, the features of the modern mobile GPUs are taken advantage of to enable more efficient rendering process. In the context of the defined optimization targets, batch optimization can be used for increasing the performance and for lowering the energy consumption.

A batch is a group of geometry which has common surface material information, and hence can be rendered with a single shader program and a draw call [111]. The rule of the thumb is that the more geometry can be packed into single batch (i.e., single draw call) the better will be the rendering performance. To accelerate the rendering process, batches can be formulated either (1) by organizing vertex attributes into their own vectors separately, or (2) by interleaving vertex attributes into a single array where the attributes of a single vertex are packed into a group. Based on the OpenGL ES 3.0 specification, the use of a single interleaved array is recommended, since the mobile GPU cache usage can be optimized when all attributes of a single vertex reside in nearby memory locations [44]. However, if the rendered 3D graphics are dynamic by nature, it may not be efficient to store the vertex attributes in a single interleaved array. This is because of the fact that updating individual vertex attributes (e.g., vertex positions) is not efficient when the vertex attributes are not stored in consecutive memory locations. In this case, having vertex attributes stored in individual arrays may be more efficient. In overall, the less time is used in the rendering process, the less energy is also consumed. This should be noted by the developer when selecting the batch optimization approach.

Finally, regardless of how the batches are formed, vertex attributes should be packed into a vertex buffer object [40, 73] which is a cached copy of the geometry. This enables the mobile GPU to organize the vertex attributes as efficiently as possible, because the characteristics of the input data are already known before the rendering process [41, 44, 111].

3.3.4 Local Caching

If the downloaded 3D graphics do not completely fit into the memory space of a mobile device, available

permanent memory can be used for local caching to avoid the need for re-transmissions of 3D graphics from the network. In the context of the defined optimization targets, local caching can be used for enhancing performance and for lowering the energy consumption.

Use of local caching results in lesser use of a wireless radio interface that has a significant positive effect on both the performance and the energy consumption. As the permanent memory on the mobile device is limited, some caching policies are needed. In general, local caching should always store the 3D graphics that is predictably needed in the near future. An example implementation for local caching on mobile devices is provided by Nurminen [8] who apply a method similar to the least recently used algorithm. When the local cache becomes full, the 3D graphics that are outside the current view are released either in reverse distance (for geometry) or in a LOD (for textures) order. This caching mechanism also favors small pieces of 3D graphics instead of few large ones for minimizing access latencies.

4 Discussion and Conclusions

Commonly, 3D Internet applications have been designed for desktop computers which have greater hardware capabilities than mobile devices. In this article, we have (1) identified different technological approaches for deploying 3D graphics on mobile devices, (2) defined a general delivery chain of 3D graphics for mobile devices, (3) identified three optimization targets (visual quality, performance and energy consumption) for 3D graphics deployment on mobile devices based on the special characteristics of mobile hardware, and finally (4) introduced and analyzed an extensive number of optimization techniques for data-driven 3D graphics deployment on mobile devices. The data-driven approach was chosen for closer examination, because it supports off-line use, multi-user interaction, creation and distribution of user-created 3D graphics as well as creation of arbitrary 3D graphics.

Based on the analysis conducted in Sect. 3, some recommendations are next provided for selecting appropriate optimization techniques according to the defined optimization targets. It should be noted that the right balance between the three optimization targets should always emerge from the requirements of the 3D Internet application itself and the characteristics of the predicted usage environment including the available wireless bandwidth and the capacity of mobile device. Table 5 summarizes the impact of the wireless bandwidth and the capacity of the mobile device on the implementation of the three optimization targets.

When *visual quality* is optimized without completely neglecting performance and energy consumption, it is beneficial to use discrete LODs for geometry with as high quality as enabled by the wireless bandwidth. For texture LODs, texture coordinate optimization enables high quality textures even with lower resolution images. When wireless bandwidth is scarce, RGBA images provide a compact presentation for textures. Otherwise, compressed textures can be used. The higher the capacity of the mobile device, the more advanced surface materials should be used to provide the best visual quality possible.

When *performance* is optimized without completely neglecting visual quality and energy consumption, low quality discrete or progressive LODs for geometry, low quality textures and simple surface materials should be used. This is to make sure that (1) all 3D graphics can be quickly transferred over the wireless medium, (2) all 3D graphics fits into the available memory of the mobile chipset, and (3) the 3D rendering process is as efficient as possible to provide a high frame rate and a responsive user interface. It should be noted that progressive LODs are an option when the capacity of the mobile device is high due to the potential decoding overhead inflicted on the mobile CPU. When wireless bandwidth is scarce, it is important to choose a progressive method that provides a very low bpv ratio (e.g., [64]) to enable as efficient use of the wireless bandwidth as possible. When wireless bandwidth is not a limiting factor, a hybrid combination of discrete and progressive methods presented by Limper et al. [40] is an ideal choice, because it does not require any decoding on the mobile CPU. However, the method requires more wireless bandwidth due to the low level of geometry compression. For textures, either RGBA or compressed textures with lossless compression should be used when the wireless bandwidth is scarce. When the capacity of the mobile device is low, GPU transcoding increases the performance of the mobile GPU. Batch optimization also increases performance when implemented appropriately based on the static or dynamic nature of the 3D

Table 5 The impact of the wireless bandwidth and the capacity of the mobile device on the implementation of the three optimization targets

	Visual quality		Performance		Energy consumption	
Wireless bandwidth						
Low	* Few medium quality discrete LODs for geometry with all vertex attributes * High quality RGBA textures, texture coordinate optimization * Simple surface materials	* Few medium quality discrete LODs for geometry with all vertex attributes * High quality RGBA textures, texture coordinate optimization * Advanced surface materials	* Few low quality discrete LODs for geometry * Low quality RGBA textures * Simple surface materials * GPU transcoding * Remote caching * Batch optimization * Local caching	* Few low quality discrete or progressive LODs for geometry * Low quality compressed textures + lossless compression * Simple surface materials * Remote caching * Batch optimization * Local caching	* Few low quality discrete LODs for geometry * Low quality compressed textures + lossless compression * Simple surface materials * GPU transcoding * Traffic shaping * Batch optimization * Local caching	* Few low quality discrete or progressive LODs for geometry * Low quality compressed textures + lossless compression * Simple surface materials * GPU transcoding * Traffic shaping * Batch optimization * Local caching
High	* Several medium quality discrete LODs for geometry with all vertex attributes * High quality compressed textures, texture coordinate optimization * Simple surface materials	* Several high quality discrete LODs for geometry with all vertex attributes * High quality compressed textures, texture coordinate optimization * Advanced surface materials	* Several low quality discrete LODs for geometry * Low quality compressed textures * Simple surface materials * GPU transcoding * Remote caching * Batch optimization * Local caching	* Several low quality discrete or progressive LODs for geometry * Low quality compressed textures * Simple surface materials * Remote caching * Batch optimization * Local caching	* Several low quality discrete geometry LODs * Low quality compressed textures + lossless compression * Simple surface materials * GPU transcoding * Traffic shaping * Batch optimization * Local caching	* Several low quality discrete or progressive LODs for geometry * Low quality compressed textures + lossless compression * Simple surface materials * GPU transcoding * Traffic shaping * Batch optimization * Local caching
Capacity of the mobile device	Low	High	Low	High	Low	High

scene. Finally, remote caching enables faster responses to 3D graphics queries over the wireless medium and local caching decreases the number of required queries if all 3D graphics do not fit into the memory on the mobile chipset.

When *energy consumption* is optimized without completely neglecting visual quality and performance, low quality discrete or progressive LODs for geometry, low quality textures and simple surface materials should be used. This way, the energy consumption can be decreased (1) by using the wireless radio interface as little as possible, and (2) by minimizing the load on the mobile chipset caused by the 3D rendering phase. When wireless bandwidth is very scarce, it may be beneficial to use a progressive method that provides a low bpv ratio (e.g., [64]) to enable as efficient use of the wireless bandwidth as possible. When wireless bandwidth is not a limiting factor, a hybrid combination of discrete and progressive methods presented by Limper et al. [40] is also an option, because the method does not require any decoding on the mobile CPU and consumes less memory on the mobile chipset compared to having several discrete LODs. For textures, it is the most energy efficient to use compressed textures with lossless compression. In addition, it is beneficial to use GPU transcoding to decrease the load on the mobile chipset through lesser use of the internal bandwidth and the memory. Batch optimization also decreases energy consumption when implemented appropriately based on the static or dynamic nature of the 3D scene. Traffic shaping should also be used to minimize the use of the wireless radio interface. As a result, 3D graphics are bundled, i.e., packaged together and sent to the mobile device in a single package, instead of multiple small packages. It should be noted that this approach may not be suitable for 3D Internet applications with strict real-time requirements. However, traffic shaping enables significant energy savings. Finally, local caching should be used to decrease the use of the wireless radio interface if all 3D graphics do not fit into the memory on the mobile chipset.

In addition to the limitations set by the wireless bandwidth and the capacity of the mobile device, also collaborative nature of a 3D Internet application may introduce real-time requirements that must be taken into account when selecting appropriate optimization techniques. When 3D graphics are created and modified by mobile users, their devices need to be capable of updating new pieces of 3D graphics to the 3D Internet application (and its other users) within an acceptable timeframe. From the standpoint of geometry simplification, the encoding speed of a simplification method becomes an important selection criterion. However, simplification methods providing fast encoding times do not typically provide very good compression rates, which may result in delayed interaction when dealing with limited wireless bandwidth. It may be beneficial to use progressive methods in these cases, because incremental addition of detail enables having a low quality version of 3D graphics very quickly available.

Research on the deployment of 3D graphics for mobile devices is a wide area and is continuously affected by the delivery of new mobile devices and their features to the market. The optimization techniques presented in this article are valuable, but to gain even further benefit of their use, a deeper understanding of the target mobile hardware is of the essence. Such knowledge together with the information provided in this article will help in deploying robust and mobile friendly 3D Internet applications in the future.

Acknowledgments The authors would like to thank the staff at Center for Internet Excellence for their help and assistance. This work has been carried out in the CADIST3D Project funded by the Academy of Finland and the Chiru Project funded by the Finnish Funding Agency for Technology and Innovation (Tekes).

References

1. Superdata. *Superdata*. http://www.superdataresearch.com/global-mmo-games-spending-exceeds-12bn/.
2. Ortiz, S. (2010). Is 3D finally ready for the web? *Computer, 43*(1), 14–16.
3. Google. *The project Tango*. https://www.google.com/atap/projecttango/#project.
4. You, Y., & Murphy, D. (2012). From 2D web map to mobile 3D mirrorworld: A live virtual advertising use case. In *6th International conference on next generation mobile applications, services, and technologies, NGMAST 2012* (pp. 42–47).
5. Boulos, M. N. K., Hetherington, L., & Wheeler, S. (2007). Second Life: An overview of the potential of 3-D virtual worlds in medical and health education. *Health Information and Libraries Journal, 24*, 233–245.
6. Hew, K. F., & Cheung, W. S. (2010). Use of three-dimensional (3-D) immersive virtual worlds in K-12 and higher education settings: A review of the research. *British Journal of Educational Technology, 41*, 33–55.

7. Lou Maher, M., Liew, P.-S., Gu, N., & Ding, L. (2005). An agent approach to supporting collaborative design in 3D virtual worlds. *Automation in Construction, 14*(2), 189–195.
8. Nurminen, A. (2007). Mobile, hardware-accelerated urban 3D maps in 3G networks. In *12th International conference on 3D web technology*, *proceedings* (pp. 7–16).
9. Hendaoui, A., Limayem, M., & Thompson, C. W. (2008). 3D social virtual worlds: Research issues and challenges. *IEEE Internet Computing, 12*, 88–92.
10. Intel. *Introducing the Intel Science and Technology Center for Secure Computing*. http://download.intel.com/newsroom/kits/research/2011/pdfs/ISTC-SC_WhitePaper.pdf.
11. SocialBakers. *SocialBakers*. http://www.socialbakers.com/blog/554-facebook-hits-488-million-mobile-users-infographic.
12. Kanjo, E., Benford, S., Paxton, M., Chamberlain, A., Fraser, D. S., Woodgate, D., et al. (2008). MobGeoSen: Facilitating personal geosensor data collection and visualization using mobile phones. *Personal and Ubiquitous Computing, 12*, 599–607.
13. Back, M., Childs, T., Dunnigan, A., Foote, J., Gattepally, S., Liew, B., Shingu, J., & Vaughan, J. (2010). The Virtual Factory: Exploring 3D worlds as industrial collaboration and control environments. *Proceedings*: *IEEE virtual reality* (pp. 257–258).
14. Capin, T., Pulli, K., & Akenine-Möller, T. (2008). The state of the art in mobile graphics research. *IEEE Computer Graphics and Applications, 28*, 74–84.
15. Hildebrandt, D., Klimke, J., Hagedorn, B., & Döllner, J. (2011). Service-oriented interactive 3D visualization of massive 3D city models on thin clients. In *2nd International conference on computing for geospatial research and applications, Washington, USA* (p. 1). New York: ACM.
16. Nadalutti, D., Chittaro, L., & Buttussi, F. (2006). Rendering of X3D content on mobile devices with OpenGL ES. In *11th ACM international conference on 3D web technology* (pp. 19–26).
17. Döllner, J., Hagedorn, B., & Klimke, J. (2012). Server-based rendering of large 3D scenes for mobile devices using G-buffer cube maps. In *17th ACM international conference on 3D web technology* (Vol. 1, pp. 97–100).
18. Xiao, Y., Bhaumik, R., Yang, Z., Siekkinen, M., Savolainen, P., & Ylä-Jääski, A. (2010). A system-level model for runtime power estimation on mobile devices. In *IEEE/ACM international conference on cyber*, *physical and social computing* (pp. 27–34).
19. Preda, M., Villegas, P., & Morán, F. (2008). A model for adapting 3D graphics based on scalable coding, real-time simplification and remote rendering. *Visual Computer, 24*, 881–888.
20. Gotchev, A., Akar, G. B., Capin, T., Strohmeier, D., & Boev, A. (2011). Three-dimensional media for mobile devices. *Proceedings of the IEEE, 99*(4), 708–741.
21. Evans, A., Romeo, M., Bahrehmand, A., Agenjo, J., & Blat, J. (2014). 3D graphics on the web: A survey. *Computational Graphics, 41*, 43–61.
22. Kim, J., Choi, J., Chang, D., Kwon, T., Choi, Y., & Yuk, E. (2005). Traffic characteristics of a massively multi-player online role playing game. In *4th ACM SIGCOMM workshop on Network and system support for games*: *NetGames'05* (pp. 1–8). New York: ACM Press.
23. Wang, X., Kim, H., Vasilakos, A., Kwon, T., Choi, Y., Choi, S., & Jang, H. (2009). Measurement and analysis of world of warcraft in mobile WiMAX networks. In *8th Annual workshop on network and systems support for games*.
24. Chang, C.-F., & Ger, S.-H. (2002). Enhancing 3D graphics on mobile devices by image-based rendering. In *Advances in multimedia information processing—PCM 2002* (pp. 1–17). Berlin: Springer.
25. Schinko, C., Berndt, R., Eggeling, E., & Fellnet, D. (2014). A scalable rendering framework for generative 3D content. In *Proceedings of the nineteenth international ACM conference on 3D web technologies* (pp. 81–87). Vancouver, BC: ACM.
26. Lamberti, F., & Sanna, A. (2007). A streaming-based solution for remote visualization of 3D graphics on mobile devices. *IEEE Transactions on Visualization and Computer Graphics, 13*, 247–260.
27. Merkle, P., Wang, Y., Müller, K., Smolic, A., & Wiegand, T. (2009). Video plus depth compression for mobile 3D services. In *3DTV-CON 2009*: *3rd 3DTV-conference*: *The true vision—Capture*, *transmission and display of 3D video*, *proceedings* (pp. 1–4).
28. Paravati, G., Sanna, A., Lamberti, F., & Ciminiera, L. (2011). An adaptive control system to deliver interactive virtual environment content to handheld devices. *Mobile Networks and Applications, 16*, 385–393.
29. Siltanen, P., Karhela, T., Woodward, C., & Savioja, P. (2007). Augmented reality for plant lifecycle management. In 13th International conference on concurrent enterprising, Sofia Antipolis (pp. 407–414).
30. Over, M., Schilling, A., Neubauer, S., & Zipf, A. (2010). Generating web-based 3D City Models from OpenStreetMap: The current situation in Germany. *Computers, Environment and Urban Systems, 34*, 496–507.
31. SecondLife. *SecondLife*. http://wiki.secondlife.com/wiki/Primitive.
32. Alatalo, T. (2011). An entity-component model for extensible virtual worlds. *IEEE Internet Computing, 15*, 30–37.
33. Hoppe, H. (1996). Progressive meshes. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*: *SIGGRAPH'96* (pp. 99–108). New York: ACM Press.
34. Cantlay, I. (2005). MipMap-level measurements. In M. Pharr & R. Fernando (Eds.), *GPU Gems 2*: *Programming techniques for high-performance graphics and general-purpose computation*. Boston, USA: Addison-Wesley Professionals.
35. Mochocki, B., Lahiri, K., & Cadambi, S. (2006). Power analysis of mobile 3D graphics. In *Design automation and test in Europe conference* (pp. 107–112).
36. Ström, J., & Akenine-Möller, T. (2005). i PACKMAN: High-quality, low-complexity texture compression for mobile phones. In *ACM SIGGRAPH/EUROGRAPHICS conference on graphics hardware*, Los Angeles (pp. 177–182).
37. Ström, J., & Wennersten, P. (2011). Lossless compression of already compressed textures. In *ACM SIGGRAPH symposium on high performance graphics*, New York (pp. 177–182).

38. Vatjus-Anttila, J., Hickey, S., & Kuusela, E. (2011). Methods and network architecture for modifying extensible virtual environment to support mobility. In *15th MindTrek conference*, Tampere (pp. 45–51).
39. Hoque, M. A., Siekkinen, M., & Nurminen, J. K. (2011). On the energy efficiency of proxy-based traffic shaping for mobile audio streaming. In *2011 IEEE consumer communications and networking conference*, *CCNC'2011* (pp. 891–895).
40. Limper, M., Jung, Y., Behr, J., & Alexa, M. (2013). The POP buffer: Rapid progressive clustering by geometry quantization. *Computer Graphics Forum, 21*, 197–206.
41. Munshi, A., Ginsburg, D., & Shreiner, D. (2008). *OpenGL ES 2.0 programming guide*. Boston: Addison-Wesley.
42. Stamminger, M., & Drettakis, G. (2002). Perspective shadow maps. *ACM Transactions on Graphics, 21*, 557–562.
43. Kuehne, B., True, T., Commike, A., & Shreiner, D. (2005). Performance OpenGL: Platform independent techniques. In *ACM SIGGRAPH conference on computer graphics and interactive techniques*.
44. Ginsburg, D., & Purnomo, B. (2014). *OpenGL ES 3.0 programming guide*. Boston: Addison-Wesley Professionals.
45. NVIDIA. *NVIDIA Tegra4 family GPU architecture*. http://www.nvidia.com/docs/IO/116757/Tegra_4_GPU_Whitepaper_FINALv2.pdf.
46. Belleville, M., Cantatore, E., Fanet, H., Fiorini, P., Nicole, P., Pelgrom, M., et al. (2009). *Energy autonomous systems: Future trends in devices, technology, and systems*. Paris: CATRENE.
47. Kim, D., Jung, W., & Cha, H. (2013). Runtime power estimation of mobile AMOLED displays. In *Design, automation test in Europe conference exhibition* (*DATE*), 2013 (pp. 61–64).
48. Dong, M., Choi, Y.-S. K., & Zhong, L. (2009). Power modeling of graphical user interfaces on OLED displays. In *46th Annual design automation conference on ZZZ*: *DAC'09* (pp 652–657). New York: ACM Press.
49. Dong, M., Choi, Y.-S. K., & Zhong, L. (2009). Power-saving color transformation of mobile graphical user interfaces on OLED-based displays. In *Proceedings of ACM/IEEE ISLPED* (pp. 339–342).
50. Samet, H. (1989). *Applications of spatial data structures*: *Computer graphics*, *image processing, and GIS*. Boston, USA: Addison-Wesley Longman Publishing Co., Inc.
51. Schmalstieg, D., & Tobler, R. F. (1999). Fast projected area computation for three-dimensional bounding boxes. *Journal of Graphics Tools, 4*, 37–43.
52. Schmidt, D., & Wehn, N. (2009). DRAM power management and energy consumption: A critical assessment. In *22nd Annual symposium on integrated circuits and system design*: *Chip on the dunes* (pp. 32:1–32:5). New York: ACM.
53. Harjula, E., Kassinen, O., & Ylianttila, M. (2012). Energy consumption model for mobile devices in 3G and WLAN networks. In *Consumer communications and networking conference* (pp. 532–537).
54. Willmott, A. (2011). Rapid simplification of multi-attribute meshes. In *High-performance graphics*, Lyon (pp. 151–158).
55. Hosseini, M., Fedorova, A., Peters, J., & Shirmohammadi, S. (2012). Energy-aware adaptations in mobile 3D graphics. In *20th ACM international conference on multimedia*, Nara (pp. 1017–1020).
56. Lee, J., Choe, S., & Lee, S. (2010). Mesh geometry compression for mobile graphics. In *7th IEEE consumer communications and networking conference*, *CCNC 2010* (pp. 301–305).
57. Hoppe, H. (1998). Efficient implementation of progressive meshes. *Computational Graphics, 22*, 27–36.
58. Hussain, M. (2009). Efficient simplification methods for generating high quality LODs of 3D meshes. *The Journal of Computer Science and Technology, 24*, 604–613.
59. Cacciola, F. Triangulated surface mesh simplification. http://doc.cgal.org/latest/Surface_mesh_simplification/index.html.
60. Peng, J., Kim, C. S., & Kuo, C. C. J. (2005). Technologies for 3D mesh compression: A survey. *Journal of Visual Communication and Image Representation, 16*, 688–733.
61. Wei, J., & Lou, Y. (2010). Feature preserving mesh simplification using feature sensitive metric. *The Journal of Computer Science and Technology, 25*, 595–605.
62. Lee, H., Lavoué, G., & Dupont, F. (2012). Rate-distortion optimization for progressive compression of 3D mesh with color attributes. *Visual Computer, 28*, 137–153.
63. Lavoué, G., Chevalier, L., & Dupont, F. (2013). Streaming compressed 3D data on the web using JavaScript and WebGL. In *18th International conference on 3D web technology*, San Sebastian (pp. 19–27).
64. Maglo, A., Courbet, C., Alliez, P., & Hudelot, C. (2012). Progressive compression of manifold polygon meshes. In *Computers and graphics* (pp. 349–359). Amsterdam, The Netherlands: Elsevier.
65. Garland, M., & Heckbert, P. S. (1997). Surface simplification using quadric error metrics. In *24th Annual conference on Computer graphics and interactive techniques—SIGGRAPH'97* (pp. 209–216). New York: ACM Press.
66. Garland, M., & Heckbert, P. S. (1998). Simplifying surfaces with color and texture using quadric error metrics. In *Visualization'98* (cat. no. 98CB36276, pp. 263–269).
67. Alliez, P., & Gotsman, C. (2005). Recent advances in compression of 3D meshes. In *Advances in multiresolution for geometric modelling* (pp. 1–25). Berlin Heidelberg: Springer.
68. Cignoni, P., Ganovelli, F., Gobbetti, E., Marton, F., Ponchio, F., & Scopigno, R. (2003). BDAM—Batched dynamic adaptive meshes for high performance terrain visualization. *Computer Graphics Forum, 22*(3), 505–514.
69. Sloan, P.-P. J., Weinstein, D. M., & Brederson, D. J. (1998). Importance driven texture coordinate optimization. *Computer Graphics Forum, 17*, 97.
70. Hunter, A., & Cohen, J. D. (2000). Uniform frequency images: Adding geometry to images to produce space-efficient textures. In *Proceedings visualization 2000*, *VIS 2000* (cat. no. 00CH37145, pp. 243–250).
71. Balmelli, L., Taubin, G., & Bernardini, F. (2002). Space-optimized texture maps. *Computer Graphics Forum, 21*(3), 411–420.
72. Sander, P., Gortler, S., Snyder, J., & Hoppe, H. (2002). Signal-specialized parametrization. In *Thirteenth eurographics workshop on rendering*, 2002 (pp. 87–89).
73. Vatjus-Anttila, J., Hickey, S., & Koskela, T. (2013). Adaptive content management for collaborative 3D virtual

spaces. In *13th Conference of FRUCT Association*, Petrozavodsk (pp. 132–142).
74. Khronos. *OpenGL ES version 3.0*. http://www.khronos.org/registry/gles/specs/3.0/es_spec_3.0.0.pdf.
75. Skodras, A., Christopoulos, C., & Ebrahimi, T. (2001). The JPEG 2000 still image compression standard. *IEEE Signal Processing Magazine, 18*, 36–58.
76. Cheng, A. M. K., & Shang, F. (2005). Priority-driven coding of progressive JPEG images for transmission in real-time applications. In *11th IEEE international conference on embedded and real-time computing systems and applications* (pp. 129–134).
77. Sun, B., Ramamoorthi, R., Narasimhan, S. G., & Nayar, S. K. (2005). A practical analytic single scattering model for real time rendering. In *ACM SIGGRAPH 2005 papers of SIGGRAPH 05* (Vol. 24, pp. 1040–1049).
78. Cook, R. L., & Torrance, K. E. (1981). A reflectance model for computer graphics. *ACM SIGGRAPH Computer Graphics, 15*, 307–316.
79. Walter, B., Marschner, S., Li, H., & Torrance, K. (2007). Microfacet models for refraction through rough surfaces. In 18th eurographics conference on rendering techniques (pp. 195–206). Grenoble: Eurographics Association.
80. Lefohn, A. E., Sengupta, S., & Owens, J. D. (2007). Resolution-matched shadow maps. *ACM Transactions on Graphics, 26*, 1–17.
81. Policarpo, F., & Oliveira, M. M. (2006). Relief mapping of non-height-field surface details. In *Symposium on interactive 3D graphics and games—SI3D'06* (Vol. 1, p. 55).
82. Dmitriev, K., & Makarov, E. (2011). Generating displacement from normal map for use in 3D games. In *Proceedings of ACM SIGGRAPH 2011 talks*. Vancouver, BC: ACM.
83. Kaneko, T., Takahei, T., Inami, M., Kawakami, N., Yanagida, Y., Maeda, T., & Tachi, S. (2001). Detailed shape representation with parallax mapping. *Proceedings of ICAT, 2001*, 205–208.
84. Koulieris, G. A., Drettakis, G., Cunningham, D., & Mania, K. (2014). C-LOD: Context-aware material level-of-detail applied to mobile graphics. *Computer Graphics Forum, 33*(4), 41–49.
85. Pool, J., Lastra, A., & Singh, M. (2011). Precision selection for energy-efficient pixel shaders. In *ACM SIGGRAPH symposium on high performance graphics*, Vancouver (pp. 159–168).
86. Vatjus-Anttila, J., Koskela, T., & Hickey, S. (2013). Power consumption model of a mobile GPU based on rendering complexity. In *7th International conference on next generation mobile apps, services and technologies*, Prague (pp. 210–215).
87. Android. *Android developers*. http://developer.android.com/about/dashboards/index.html.
88. Ström, J., & Pettersson, M. (2007). ETC2: Texture compression using invalid combinations. In *22nd ACM SIGGRAPH/EUROGRAPHICS symposium on graphics hardware*, Sarajevo (pp. 63–70).
89. Gil, B., & Trezentos, P. (2011). Impacts of data interchange formats on energy consumption and performance in smartphones. In *Workshop on open source and design of communication—OSDOC'11* (pp. 1–6). New York: ACM Press.
90. Deutsch, P. (1996). *GZIP file format specification version 4.3—IETF RFC1952*.
91. Ziv, J., & Lempel, A. (1977). A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory, 23*, 337–343.
92. Fielding, R. T., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., & Berners-Lee, T. (1999). Hypertext—HTTP/1.1. http://www.ietf.org/rfc/rfc2616.txt.
93. Alakuijala, J., & Vendevenne, L. Data compression using Zopfli. https://zopfli.googlecode.com/files/Data_compression_using_Zopfli.pdf.
94. Ziv, J., & Lempel, A. (1978). Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory, 24*, 530–536.
95. Welch, T. (1984). A technique for high-performance data compression. *Computer (Long Beach, California), 17*, 8–19.
96. Burrows, M., & Wheeler, D. (1994). A block-sorting lossless data compression algorithm. Technical Report 124. Palo Alto: Digital Equipment Corporation.
97. Cleary, J. G., & Witten, I. H. (1984). Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications, COM-32*, 396–402.
98. Shkarin, D. (2002). PPM: One step to practicality. In *Proceedings of DCC 2002. Data compression conference* (pp. 202–211).
99. Barr, K. C., & Asanović, K. (2006). Energy-aware lossless data compression. *ACM Transactions on Computer Systems, 24*, 250–291.
100. Rauschenbach, U., & Schumann, H. (1999). Demand-driven image transmission with levels of detail and regions of interest. *Computational Graphics, 23*, 857–866.
101. Mauve, M., Fischer, S., & Widmer, J. (2002). A generic proxy system for networked computer games. In *1st Workshop on Network and system support for games—NETGAMES'02* (pp. 25–28). New York: ACM Press.
102. Podlipnig, S., & Böszörmenyi, L. (2003). A survey of Web cache replacement strategies. *ACM Computing Surveys, 35*, 374–398.
103. Bontu, C. S., & Illidge, E. (2009). DRX mechanism for power saving in LTE—[topics in radio communications]. *IEEE Communications Magazine, 47*, 48–55.
104. Siekkinen, M., Hoque, M., Nurminen, J., & Aalto, M. (2013). Streaming over 3G and LTE: How to save smartphone energy in radio access network-friendly way. In *5th Workshop on mobile video*, Oslo (pp. 13–18).
105. Vergara, E. J., & Nadjm-Tehrani, S. (2013). EnergyBox: A trace-driven tool for data transmission energy consumption studies. In *Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics*, pp. 19–34). Berlin: Springer
106. Balasubramanian, N., Balasubramanian, A., & Venkataramani, A. (2009). Energy consumption in mobile phones. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference—IMC'09* (p. 14). New York: ACM Press.
107. Huang, J., Quian, F., Guo, Y., Zhou, Y., Xu, Q., Mao, Z. M., Sen, S., & Spatscheck, O. (2013). An in-depth study of LTE: Effect of network protocol and application behavior on performance. In *SIGCOMM* (pp. 363–374).

108. Grigorik, I. (2013). *High performance browser networking*. Sebastopol, CA: O'Reilly Media.
109. Harvey, R. C., Hamza, A., Ly, C., & Hefeeda, M. (2010). Energy-efficient gaming on mobile devices using dead reckoning-based power management. In *2010 9th Annual workshop on network and systems support for games, NetGames 2010*.
110. Alliez, P., & Desbrun, M. (2001). Progressive compression for lossless transmission of triangle meshes. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (pp. 195–202). New York: ACM Press.
111. Möller, T., Haines, E., & Hoffman, N. (2008). *Real-time rendering*. Wellesley, MA: A K Peters/CRC Press.