ENERGY-BUDGET-COMPLIANT ADAPTIVE 3D TEXTURE STREAMING IN MOBILE GAMES

by

Seyyed Mohammad Hosseini

B.Sc., Sharif Univeristy of Technology, 2011

a Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science

in the
School of Computing Science
Faculty of Applied Sciences

APPROVAL

Name:	Seyyed Mohammad Hosseini
Degree:	Master of Science
Title of Thesis:	Energy-Budget-Compliant Adaptive 3D Texture Streaming in Mobile Games

Examining Committee: Dr. Ze-Nian Li, Professor, Computing Science
Chair

Dr. Joseph Peters, Professor, Computing Science
Simon Fraser University
Senior Supervisor

Dr. Shervin Shirmohammadi, Professor, Computing Science
University of Ottawa
Senior Supervisor

Dr. Jiangchuan Liu, Associate Professor, Computing Science
Simon Fraser University
SFU Examiner

Date Approved:

Abstract

Advances in computing hardware and multimedia applications have spurred the development of mobile devices such as smartphones and PDAs. Amongst the most used applications on handheld devices are mobile 3D graphics such as 3D games and virtual environments. With this significant increase of mobile applications, one of the challenges is how to efficiently transmit the bulky 3D information to resource-constrained mobile devices. They impose significant demands on the limited battery capacity of mobile devices. Thus deploying efficient approaches to decrease the amount of streamed data with the aim of increasing the battery lifetime has become a key research topic.

In this study, we design and implement an adaptive priority-based framework for efficiently streaming 3D textures to mobile devices with limited energy budget over wireless networks. Our results show that using our proposed adaptations significantly improves the gameplay quality per unit of energy consumed to download 3D textures in mobile games.

Keywords: Mobile 3D Games, Budget-based Gaming, Context-aware 3D Streaming, Energy-efficient Mobile Gaming, 3D Texture Streaming, Dynamic and Static Gaming Scenes

To my parents, family, supervisors, and beloved friends for their support.

— Happiness is motivation!

Find out where joy resides. For to miss the joy is to miss all,

Mohammad Hosseini, *2013*

Acknowledgments

Hereby, I would like to express my deep appreciations to my competent supervisors, Prof. Joseph Peters and Prof. Shervin Shirmohammadi for their profound influence on my research and studies. They introduced me to the field of Mobile Multimedia and Games, and taught me a great deal of valuable research skills. In the light of their helps, I quickly developed my skills, grew towards many achievements, and got involved in the world's premier conferences related to my research, which really helped me achieve many academic accomplishments within my master's studies.

I also want to thank the faculty members, staff, and friends in the School of Computing Science at Simon Fraser University for providing such a nice and academic environment. My graduate studies would not have been interesting without them. I have enjoyed the time with the members in the Network Systems Lab.

I also give appreciation to the School of Graduate Studies and the Faculty of Applied Science at Simon Fraser University, for scholarship funding that helped me to focus full time on my thesis. My research was also supported by a Natural Sciences and Engineering Research Council of Canada (NSERC) Grant.

Finally, I would like to thank my family for their love and support over the years. Clearly I was not able to finish my thesis without their kind helps. And the last, but not the least, I would like to thank my homemates for providing a friendly and convenient environment.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Nowadays with the growing demands of mobile applications together with improvements in handheld hardware technologies, one can undoubtedly say that mobile computing will soon replace PC-based computing for the average consumer. Already, more than half of the world population uses cell phones or other mobile devices [13], more mobile phones are used to browse the Internet than PCs, and the average person is more likely to use a handheld device than a PC for a variety of applications [12]. For computing, this is a natural evolution as we move from the traditional device-centric era (PC) to a new user-centric era, i.e. ubiquitous or mobile computing. For mobile devices, this trend is the result of the fast progress made in two areas: capabilities and features of mobile devices, which now have integrated 3D hardware accelerators, fast CPUs, large memory, Bluetooth and WiFi, as well as the rapid advancement in wireless networking technologies such as 3G and 4G networks. As a result of this growth, universities across the globe are including mobile applications design and development as part of their curricula, and major projects around the world are focusing on investment in the research and development of applications on the mobile handheld devices working on the next-generation wireless systems and services [42].

At the same time, we are witnessing a trend towards the mass consumption of new media. While traditional applications such as email, web browsing, music, and video are still popular, there is a significant rise in the next generation multimedia and graphics applications such as online 3D games and 3D virtual environments. Many online game genres such as Massively Multiplayer Online Games (MMOGs) make it possible for millions of players to interact with each other simultaneously. MMOGs are not only used for gaming, but also used for other purposes such as socializing, business, and even academics and

Figure 1.1: Quake 3, an FPS mobile game played on an Android smartphone [2].

scientific experiments. Currently many real companies such as IBM, Intel, CNN have opened virtual branches in these online games and online virtual environments like in Second Life, and massive number of players and users spend virtual monies in exchange for real-world money, or for trading virtual goods. As a popular example, World of Warcraft has gained over 11.5 million subscribers with a peak of 500,000 players online at a given time [49].

Mobile gaming is also growing with a high rate. There are impressive statistics showing the significance of mobile gaming: It is estimated that 78.6 million people in the United States alone played mobile games in 2009, downloads of mobile games increased tenfold compared to 2003 [43], and mobile games generated more than $1.5 billion annually in revenue [27]. Newzoo's mobile gaming trend report [7] shows that the total number of Americans that play games on their smartphone, tablet or iPod Touch has now surpassed the 100 million mark, a year-on-year increase of 35%. Europe shows a growth of 15%, totaling 70 million gamers for seven key territories. Men slightly outnumber women in the US (52%) as well as in key European countries (55%). Growth rate in terms of time and money spent is significantly higher. In 2011, mobile gaming took 13% of all time spent on games worldwide, totaling more than 130 million hours a day, and 9% of total money spent on games, grossing $5.8bn.

After the integration of 3D APIs into mobile platforms, the mobile gaming world started to launch its own brand games. There are many online mobile stores such as the popular Android Market [1] which offers applications and games on Google Play for Android-based smartphones and tablets. Same with other online mobile stores such as Apple Store [9] and Nokia Store [8]. With no doubt the growth rate of mobile games is so high that many observers believe it can beat the computer gaming industry in the near future. Many popular games like First Person Shooter (FPS) Games or Role Playing Games (RPG) which once were played on PCs are now available for mobile devices, enabled with the features for multiplayer modes. Figure 1.1 shows a screenshot of the popular Quake 3 FPS game on a smartphone that has support for online multiplayer. For Canada, it is estimated that the percentages are about the same if not higher, due to very high rate of home and mobile Internet penetration in Canada, and the fact that Canada is consistently one of the top 5 countries in terms of spending on computer games per capita; in fact it is often ranked number one [43].

The new interest in 3D media is not just limited to games and virtual environments; it impacts the whole Web experience as well. Browsers have started to incorporate 3D displays and processing capabilities as shown by initiatives such as WebGL and O3D, while social networks, traditionally static 2D environments, are now moving to offer users a rich 3D experience. Despite the promising nature of these new media, significant challenges remain to be solved in order to bring them to the mobile devices, the next generation computing platforms. Millions of people spend their time and money on online games and in online virtual/social environments, especially using their mobile handheld devices. This has led researchers to consider mobile 3D gaming as an avenue for research to further manage and support this emerging massive industry as well as its traffic on the network and user's quality of experience [42].

1.1 Motivation

Smartphones and other mobile handheld devices such as PDAs are ubiquitous. From this, we can easily derive all the other facts about mobile gaming, which is advancing with a high pace. As hardware prices are continuously falling down and new cellphones have ever-increasing computational power, they also become ideal gaming devices. Mobile phones seems to be must-have nowadays, therefore their market is huge. We are witnessing that cell service providers and companies are offering options to exchange the old-style and classic mobile phones with the new generation of smartphones with new and incredible options available in the form of a wide range of applications. In the past years people had to buy a video game system or a gaming PC in order to play video games. But nowadays they can get similar functionality from their mobile devices and smartphones, with no additional cost. The mobile device is a full functional package; it removes the need to carry a second dedicated system because everything is integrated in a single package.

Besides the benefit of having to carry only a single device for all our typical daily uses such as telephony, internet access, and gaming and entertaining needs, another factor which makes gaming on mobile devices incredibly accessible to much larger audience is the ease of usability. One can simply choose a game he or she likes, and immediately start to play. Clearly there is no need to go to a shopping store or download via PC, and furthermore for example, the USB cable might be lost to transfer the game to the smartphone. All the processes are done on the device itself maybe simply by pushing several buttons.

The increased processing power and advanced functionality of the current-generation mobile devices also has a big impression on the game developers. Even the ordinary class of devices is capable of generating 3D gaming experiences similar to games being played on the gaming consoles such as Xbox and PS2. Given these capable hardware platforms, we can also start experimenting with more elaborate games and showcasing our desired testing benchmarks which is area with so much potential for innovation and research.

Smartphones bought from the service providers are usually accompanied with data plans. They are not only used for pure telephony anymore but actually is used more for surfing the Internet. A user having a smartphone is very likely to be connected to the web at any time. Permanent connectivity opens up a completely new world for mobile gaming: Online gaming. People can challenge other people in a multiplayer 3D game, or maybe explore virtual 3D worlds together; and all of this probably occurs on a bus or train!

There are many challenges associated with the mobile online gaming. Due to the limitations in network bandwidth, memory size, battery life and computation resources of the mobile devices, a good 3D mobile game must balance view, energy, and performance. One of the challenges to achieving this balance is to efficiently transmit and render bulky 3D information, such as object textures. These textures are highly important as they are what the players will see. The failure to receive a texture can have a significant negative impact on the visual quality of the game and on the user's experience.

In this thesis, we are trying to tackle problems associated with insufficient available budgets on the mobile devices by proposing an adaptive framework to efficiently stream bulky 3D textures. Our goal is to maximize the quality of the 3D textures that are streamed within an energy budget specified by the user. We assume that the relationship between energy consumption and the amount of data downloaded is known as in Figure 1.2 so that a download budget that achieves the energy budget can be estimated. Our approach is to selectively reduce the sizes of the textures so that the total amount of data transmitted to a mobile device satisfies the download budget. In the following section we explain the research problem more in depth.

1.2 Proposed Research

As previously mentioned, the mobile handheld devices such as smartphones and PDAs are providing increasing functionality due to rapid improvements in processing power, storage capacities, graphics, high-speed connectivity, etc. Particularly, the advanced computation and communication technologies is leading to effectively integrate multimedia and graphics functions into these battery-powered devices. Understanding power consumption in mobile multimedia is the key for efficient power management of the next-generation mobile multimedia. Besides that, the increasing power requirements of the new smartphones and tablets are far outpacing improvements in battery technology. Due to these reasons the topic of *energy efficiency* has become very important for mobile battery-operated devices. Therefore the main problem faced by these devices is energy management, since battery capacities are not experiencing the same exponential growth as other technologies such as processing power and storage. While there is ongoing research in discovering and exploiting ambient energy sources, it is highly likely that energy will remain the key bottleneck for mobile devices in the near future.

Due to the slow development of battery technologies and limited battery capacities, the gap between power consumption of mobile multimedia and the limited power source has been widened. A typical modern smartphone does not last more than about 24 hours without having to be recharged [28]. However, a number of factors have conspired to change the rate of battery consumption. Firstly, wireless interfaces such as Bluetooth and 802.11 have become ubiquitous, and during data transfer (though not during idle state), they are relatively hungry consumers of energy. Switching over to superfast 4G networks, is only going to exacerbate the problem because 4G radio chipsets require a lot more processing power than current chipsets to decode far greater amounts of data encoded in the LTE wireless spectrum [6, 14, 41]. Secondly, mobile devices are getting into more multi-functional computing devices with the always-on expectation of phones, and at the same time, developers are producing increasingly more sophisticated and power-hungry applications. Key hardware components draining energy in smartphones are the display, the radio, and audio [18, 31]. While the contribution of each component varies according to how they are used by applications, all of those components significantly affect the energy budget.

WiFi devices need more power to generate a stronger signal compared to other radio-based interface cards such as Bluetooth and 3G [24, 32]. A recent study precisely measured energy consumption for different parts of a mobile phone mainly for wireless communication and other services. The results showed that the WiFi IEEE 802.11 Network Interface Card uses as much as 24 times more power while downloading data compared to the idle mode [35].

From 802.11 protocols, each frame length is described in detail. It would help calculating power consumption of single frame or specific data length (e.g., 20MB) exactly working under specific working modes. This theoretical value of energy consumption can be calculated using vendor specification of a card working with 802.11 protocols. One can obtain and compute the energy value from data obtained from real devices. Rahmati, *et al* [38] investigated a simple linear-cost energy model for wireless data transfers, assuming constant network conditions throughout a single transfer. They modeled the energy cost for establishing a connection and transferring n megabytes of data as

$$E = Ee + nEt \tag{1.1}$$

where Ee is the energy cost for connection establishment and Et the energy per MByte of data transfer. Also in [36] and [16], the authors argue that frame length details for 802.11 protocols can be used to calculate power consumption for specific data lengths. Their results also show that per-packet energy consumption of network interfaces can be modeled using Equation (1.1). Figure 1.2 shows the average energy consumed for downloading data of different sizes against varying inter-transfer times in WiFi.

While modern cellular standards highlight low client energy consumption, existing methods do not explicitly emphasize reducing power that is consumed when a client is actively communicating with the network and receiving large amounts of data [15]. The high data rates and resulting high energy demands of modern networked multimedia systems make energy-aware adaptations for these widely used mobile applications an important consideration. 3D mobile applications such as games and 3D virtual environments are amongst the most used multimedia applications. In addition, due to progress in the hardware of mobile devices, these devices can now support 3D graphics; many in fact use hardware acceleration and provide GPU-based support of popular 3D formats.

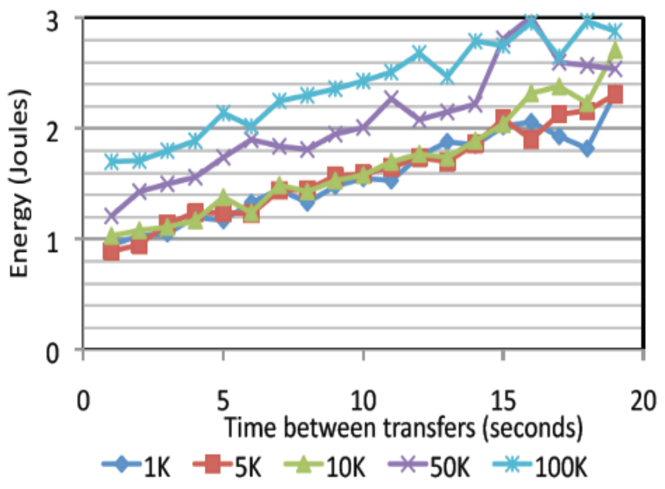


Figure 1.2: Average energy consumption for downloading data of different sizes over varying inter-transfer times for WiFi [16].

Anyways, in spite of the promising nature of 3D multimedia and graphics, significant challenges still remain to be solved in order to bring them into next generation computing platforms, including mobile devices. As each mobile handheld device has limited capabilities and resources, such as limited data plan needed for downloading, limited battery life, limited memory, higher network latency due to the nature of wireless networks, and so on, the usage of 3D games on mobile platforms is restricted. In addition to this, using more data for streaming, even if it is available, will contribute to more energy consumption by any wireless interfaces (e.g. WiFi, 3G, 4G, etc.), and thus faster battery drainage. Therefore, even with 4G wireless networks that provide bandwidths of up to 100 Mbps, we are not able to translate the availability of higher bandwidth into the continuous consumption of it.

The focus of this work lies in the streaming aspect of online mobile 3D games. The users must receive the 3D textures dynamically from the server, based on the importance of the content and the available energy budget. In our approach, we provide such features based on which texture is more important for the player in the game, which means our approach is context-based.

In the case of the mobile online 3D gaming experience, the goal of our proposed framework is for a mobile user to be able to receive the textures that are fit for the defined limited energy budget. Therefore, it is crucial for our framework to provide a prioritized and content-aware approach in such a way that it can adapt the 3D textures to the available budget, and the same time by providing the maximum possible quality.

Specifically, the goal is to take advantage of both the user's specific contexts (by taking into account the importance of data) and limitations (available energy budget), so to offer a new approach towards the delivery of live 3D game textures for mobile games over the wireless network. By doing so, we are trying to reduce the amount of streamed textures, and as a result we can address the challenges of limited download and energy budget that the mobile handheld devices are faced with. In other words, our research focuses on reducing the total size of data needed to be streamed from the server and be downloaded by the clients. Therefore, decreasing the amount of battery usage, memory usage, etc. with aim of maximizing the total quality is a consequence of our method.

1.3 Research Contributions

As the main contribution of this thesis, we propose an adaptive framework to efficiently stream bulky 3D textures to mobile devices with limited resources. Our goal is to maximize the quality of the 3D textures that are streamed within an energy budget specified by the user. We assume that the relationship between energy consumption and the amount of data downloaded is known as in Figure 1.2 so that a download budget that achieves the energy budget can be estimated. Our approach is to selectively reduce the sizes of the textures so that the total amount of data transmitted to a mobile device satisfies the download budget. But in short, the thesis consisting of the following novelties:

- We used the concept of Multiple Choice Knapsack Problem in our context to select all game textures, instead of selecting a part of textures which uses the 0-1 knapsack problem.

- We classify the game textures into different classes and prioritize them in our studies, and define a heuristic to make the best use of the available budget to stream the textures with the aim of maximizing the total quality per unit of consumed energy.

- We define a simple light-weight sampling method similar to a down-sampling approach to selectively choose texels in a texture to resize textures.

- We compare three different algorithms namely Non-adaptive, Semi-online, and Online versions for study of dynamic game scenes, with the aim of improving the game quality while keeping the available budget constant.

- Simulations and implementation of a proof of concept testbed for validating our design.

1.4 Scholastic Output and Achievements

In addition to meeting its objectives as described above, this research undertaking has also lead to a few of scholastic achievements and publications, as listed below.

Publications

- "Energy-budget-compliant adaptive 3D texture streaming in mobile games", Mohammad Hosseini, Joseph Peters, and Shervin Shirmohammadi. In Proceedings of the 4th ACM Multimedia Systems Conference (MMSys '13), Oslo, Norway, 2013, p 1-11.
- "Energy-aware adaptations in mobile 3D graphics", Mohammad Hosseini, Alexandra Fedorova, Joseph Peters, Shervin Shirmohammadi. In Proceedings of the 20th ACM international conference on Multimedia (MM '12). Nara, Japan, p 1017-1020.
- "Adaptive 3D Texture Streaming in M3G-based Mobile Games", Mohammad Hosseini, Dewan T. Ahmed, Shervin Shirmohammadi. In proceedings of ACM Multimedia Systems 2012 (MMSys '12), Chapel Hill, North Carolina, USA, p 143-148.

Awards

- ACM SIGMM Student Travel Award, ACM Multimedia, Nara, Japan, 2012.
- Finalist (among top 10 out of 490 submissions) in Lockheed Martin's *Innovate the Future Challenge*, 2012.
- Travel & Research Award, School of Computing Science, Simon Fraser University, 2012.

1.5 Thesis Outline

The thesis is organized as follows: In chapter 2, we discuss relevant background and previous work on textures, texture compression, and 3D texture streaming along with adaptive media streaming. Chapter 3 explains our initial study of adaptive texture streaming for *static scenes*. In this chapter we describe our texture adaptation framework, the texture compression module, our communication model, along with talking about our 3D proof-of-the-concept game and the evaluation part. Similarly, chapter 4 explains our detailed study for *dynamic scenes*, in which we thoroughly describe our simulation methodology to study dynamic gaming scenes, along with comparison of three different algorithms, evaluation and experimental results. Chapter 5 summarizes and concludes the thesis while outlining venues for future research.

Chapter 2

Background and Related Work

In this chapter we present some relevant background and different categories of state-of-the-art approaches associated with parts of our proposed framework, and we describe how our work is related to these approaches.

2.1 Textures

Texturing is a vital part of the visual experience of any type of 3D graphics and games. It is applicable to First Person Shooter (FPS) games as much as it is to Massive Multi-player Online (MMO) games, Role Playing Games (RPG), and 3D virtual worlds. A texture is an image that is used to provide surface covering for a 3D model. 3D textures are a logical extension of the traditional 2D textures, and have been used in high-end graphic systems to generate a three-dimensional image map. Figure 2.1 shows two examples of textures used in a mobile game, a texture for palm tree (left), and a texture for a column (right).

Generally, textures are bitmaps packed into an array or a matrix parallelepiped, with each dimension constrained to a power of two (64×64, 128×128, 256×256, etc.), and each cell representing a texture pixel, called a texel, which contains a color value. The power-of-2 rule for game textures is based on memory buffer sizes of the graphics card, to get the maximum memory efficiency out of the graphics card. Textures are characterized by two parameters: the number of texels, and the information content (color depth) per texel. There are other attributes that are applied to bitmaps but they are derived from these two fundamental parameters. Texels in a bitmap are RGB formatted, and contain certain color information. The information content is always the same for all of the texels

Figure 2.1: Two examples of textures used in a mobile game.

in a particular bitmap. Textures are loaded into RAM for 3D environments such as virtual reality applications and 3D games.

The most straightforward way of storing textures is simply to list the bitmap information. In this case, the amount of space required for any texture is easy to calculate given the texture dimensions ($N \times M$) and color depth in bits (B). The equation for the file size in KBytes is simply

$$\textit{(KByte)} = \frac{(N \times M \times B)}{(8 \times 1024)} \tag{2.1}$$

where N and M are the number of horizontal and vertical texels, and B is the number of bits per texel. As an example, a 256×256 texture with 24-bit color depth would have a file size of 196608 Bytes, or 196 KB.

2.2 Texture Compression and Resolution

To reduce the amount of texture data that needs to be transmitted, one can use texture compression, which is a specialized form of image compression designed for storing texture maps in 3D rendering systems. It is a method of reducing the size, memory, and memory bandwidth required for textures with a small reduction in visual quality. In certain games, where a low-resolution texture is used for a large surface (like a sky image), significant color banding can be seen if texture compression is enabled. A combination of enabling texture compression and high texture detail results in a good balance of quality and power saving in many games.

Currently there are a few texture compression techniques such as S3 Texture Compression (S3TC) [26] and Ericsson Texture Compression (ETC) [47]. S3TC is a group of related lossy texture compression algorithms used to compress textures in special hardware-accelerated 3D computer graphics. ETC enables compression and decompression of textures so that they can be used with ETC-capable handsets, such as Android-based handsets. The first version of the ETC compression algorithm, ETC1, does not support transparency. ETC2 is still under development, and is not available in any tools or hardware as of yet (2012) [48]. It should be noted that all of these texture compression techniques are hardware-based compressions and not all mobile devices have the hardware to support them. Another disadvantage is that they involve much decompression overhead at the client side due to the use of heavy and complicated arithmetics, which makes them unsuitable for battery-operated devices running applications that use huge numbers of textures such as 3D games. Furthermore, these texture compression techniques can lead to artifacts in low-resolution textures which are commonly used in mobile games.

As part of our system, we propose a simple approach for texture compression with negligible overhead. Unlike the texture compression approaches described above, our approach does not introduce discoloring artifacts, does not require the client device to have special hardware, and involves no decompression overhead at the client side.

Another significant feature of textures is resolution, which refers to how large the textures are. Using larger textures not only increases the streaming delay, but also uses more energy by requiring the network card to be in the active mode longer, requires the CPU to render more data, and uses more GPU memory due to the increased memory bandwidth needed. In some cases, the result is a choppier performance. Although this can be somewhat alleviated by using texture compression, texture compression itself can exhaust the hardware, and has the other disadvantages that were mentioned above.

2.3 Adaptive Media Streaming

One of the main approaches for energy saving on mobile devices is adaptive streaming. Adaptive streaming is a process whereby the quality of a multimedia stream is altered in real time while it is being sent from server to client. Figure 2.2 shows an overview of how multimedia adaptive streaming works. This adaptation of quality is controlled by decision modules on either the client or the server. The adaptation may be the result of adjusting

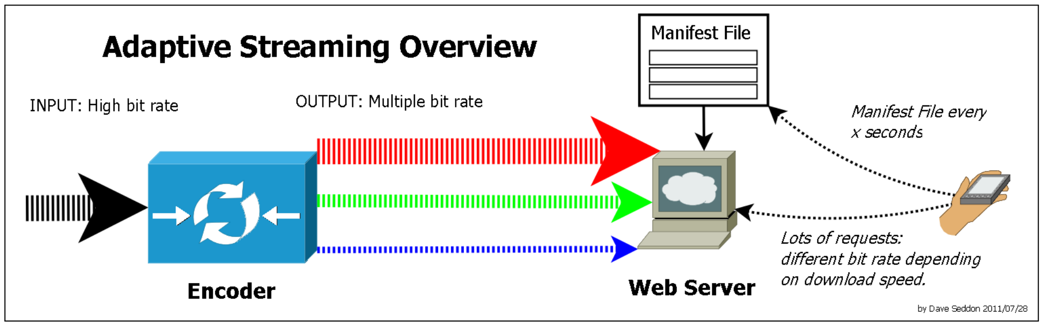

Adaptive Streaming Overview
INPUT: High bit rate
OUTPUT: Multiple bit rate
Manifest File
Manifest File every x seconds
Encoder
Web Server
Lots of requests: different bit rate depending on download speed.
by Dave Seddon 2011/07/28

Figure 2.2: An overview of multimedia adaptive streaming [11].

various network or device metrics. For example, with a decrease in network throughput, adaptation to a lower video bitrate may reduce video packet loss and improve the user's experience. Similarly, adaptations that reduce the amount of data being received over the Wireless Network Interface Controller (WNIC) can save energy. Additionally this allows the WNIC to be put into sleep mode more frequently, similar to the work in [25].

2.4 Related Work

In large virtual environments, users only interact with a subset of the objects that are visible to them at a given time. Work for 3D content streaming that takes advantage of this fact can be classified into two major classes: region-based and interest-based streaming.

Region-based approaches only stream the geometry information for the player's specific region. Examples include DIVE [23], CALVIN [34], Spline [50], and VIRTUS [40]. In such systems, the environment is divided into a number of "pre-defined" regions and before the user interacts with a given region, the full content of the region must be downloaded.

In contrast, interest-based techniques use an Area of Interest (AOI) to determine object visibility. NetEffect [20] is among such systems. These approaches however do not provide any mechanism to control the visual quality. They may reduce the amount of game content for downloading, but the download time might still be too long since in recent high-quality games, there might be huge numbers of objects inside the AOI.

Another shortcoming of the existing approaches is that they do not prioritize the objects according to whether the objects are important for a player, and whether a player is actually interested to receive them. Additionally, existing approaches do not consider the receiver's resource restrictions and hence are less suitable for mobile devices.

Finally, although it is not done for 3D games, Kennedy *et al* [30] proposed and developed an interesting approach for adaptive video streaming which analyses the remaining stream-duration and the remaining battery-life in order to decide whether or not to send an adaptation order to the dynamic streaming server. When the remaining stream duration exceeds the remaining battery life, the video quality is adapted. However, unlike video streaming, game streaming is not deterministic since the actions being taken by the users are not pre-defined. Also, purely from a gaming perspective, context-aware approaches are needed since different objects/textures need to have different priority levels. This is very much an open problem and still needs considerable research work.

In fact, there is currently no work that takes into account adaptive streaming of 3D graphics for battery-operated devices based on an available energy budget. To the best of our knowledge, the only research that has used game context as a parameter for object selection and prioritization in 3D streaming is [37] in which Rahimi *et al* presented an activity-centric context-aware object streaming approach for mobile games as a solution to maximizing the player's experience in the face of a mandatory reduction of the amount of data streamed to the mobile device due to download/network limitations. They introduced the idea of *prioritized activity-centric streaming* for mobile gaming. In their approach, they considered the activity of the player to decide which objects are more important for the accomplishment of that activity. In order to achieve this goal, the importance of each object for each specific activity in the game is determined a priori by the game designers. A list of different objects and activities for the current game scene would be provided prior to running the game and less relevant objects will not be streamed, freeing resources for objects that are more relevant. While interesting, this work does not distinguish between objects and textures. The main bottleneck for 3D game streaming is the bulky textures, not the wireframes of the objects. In our work, we specifically address textures.

In our previous work [27], we studied how to efficiently transmit bulky 3D information to bandwidth- and computationally-limited mobile devices, by proposing two methods for improving the transmission delay of 3D content over unreliable and congested networks. We introduced *Object Mesh Similarity* as a server-side approach which replaces an original object by a similar alternative object with less complexity which is then transmitted to the client side, as well as *Texture Stretching* as a client-side approach, that leads to the efficient receipt of textures. However, the goal in [27] was to improve the response time.

For this study, we build on concepts from [37] and [27] to implement an adaptive priority-based context-aware framework for efficiently streaming 3D textures to mobile devices with a limited energy budget over wireless networks.

Chapter 3

Proposed Approach: Static Scenes

In this chapter, we explain our methodology regarding the implementation of different parts of the adaptive texture streaming system for static scenes. Our design allows for efficient streaming of 3D textures to mobile devices while satisfying a download budget which is estimated based on an energy budget as explained in the introduction section, with the aim of decreasing power consumption.

Figure 3.1 shows a detailed overview of different processes in our framework. As can be seen in the figure, the system consists of two parts: client-side and server-side. Prior to the gameplay, the client device sends the current available budget to the server, which the server uses to optimize the textures that will be streamed based on how important they are in a given scene of the virtual environment. To achieve this, the system uses a classification list to prioritize the currently required textures acquired from a texture database. Then, based on the budget constraint received from the user and the prioritized list, the textures are selectively compressed, serialized and streamed to the target mobile device. The serializer serializes various structures describing the graphics state to a buffer. Serializer's additional function is to fill the buffers until certain criteria is met (theoretically it can pass the buffer to compressor after each command which, of course, would not be efficient for networking).

Progressive streaming is used as a complementary technique to send the 3D textures and the corresponding objects over the network. Finally, the client receives the textures and in parallel, as a part of the client-side 3D streaming, the 3D renderer adds the newly received textures and objects to the graphics-layer and continues to render the game.

Receiving an optimum and efficient size of textures during a gameplay experience reduces the network bandwidth and thus the energy consumption of the handheld device that

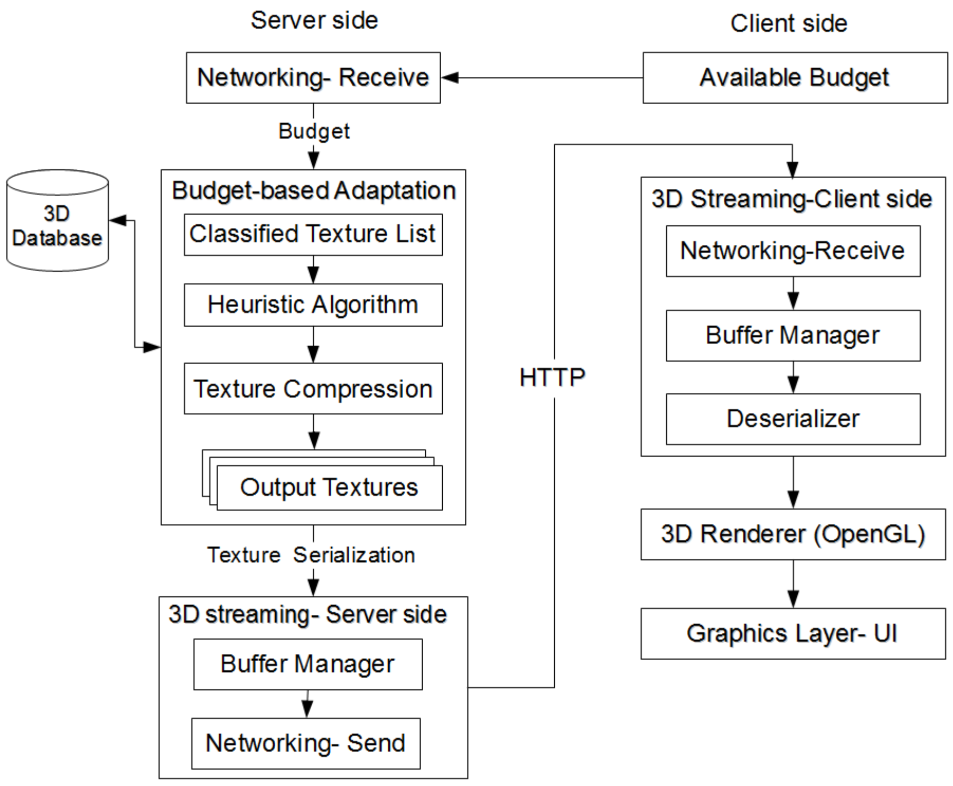


Figure 3.1: Outline of the proposed framework

receives the data as well as other limited resources (such as memory) for less-important parts of the 3D world, while the streamed textures still fulfill their role in the game.

3.1 Problem Definition

For texture selection and streaming, the most significant factor in both battery and bandwidth usage is the amount of data downloaded by the mobile device. As discussed in Chapter 1, we suppose that a user specifies an energy budget, and a download budget that achieves the energy budget is estimated. If the total size of the 3D objects and textures to be streamed does not exceed the download budget, then all of them can be streamed. If the download budget is insufficient to stream all of the objects and textures, then the total size must be reduced. In this case, we stream all of the objects leaving a download budget D for the textures. Every 3D texture has a specific size, and we must decide how to stream them within the budget D.

One approach to reducing the total size of the textures that are streamed is to transmit a subset of them. Let $\mathcal{T} = (\tau_1, \tau_2, \tau_3, \ldots)$ be the set of textures. Each texture τ_i has a size s_{τ_i} and a associated value v_{τ_i} which is based on the priority of the corresponding object, or how important the object is. We just use the term *value* to represent the importance value. The goal is to stream a subset of textures $\mathcal{T}' \subseteq \mathcal{T}$ that maximizes the total value of the streamed textures without exceeding the download budget D. In other words

$$\text{Maximize}_{\{\mathcal{T}' \subseteq \mathcal{T}\}} \sum_{\tau_i \in \mathcal{T}'} v_{\tau_i} \quad \text{subject to} \sum_{\tau_i \in \mathcal{T}'} s_{\tau_i} \leq D. \tag{3.1}$$

This selection scheme is the well-known 0-1 Knapsack optimization problem. The 0-1 Knapsack problem is NP-hard but there are good, efficient, approximation algorithms (fully polynomial approximation schemes), so this approach is computationally feasible. However, by using this method, only a subset of the textures would be selected and streamed to the client, and the visual impact could result in an unsatisfactory gaming experience. For example, compare Figure 3.14 (I) and Figure 3.14 (II) at the end of Section 3.5 on page 35.

To overcome the shortcomings of the 0-1 Knapsack approach, we propose a heuristic algorithm that sends all textures, but with different resolutions according to their priorities. This is the multiple-choice knapsack problem, in which the items (in this terminology, textures) are subdivided into k different groups, and exactly one item must be taken from each group. In other words, we are creating groups of textures where the textures in a group all correspond to the same object but have different resolutions and we choose exactly one texture from each group.

The idea of multiple-choice knapsack problem has been applied to certain contexts recently. Y. Song *et al* in their paper [44] investigated the multiple multi-dimensional knapsack problem and its applications in cognitive radio networks. In their paper, a centralized spectrum allocation in cognitive radio networks has been formulated as a multiple knapsack problem. They proposed a heuristic algorithm with guaranteed performance. Lamani *et al* [33] also proposed an end-to-end quality of service in pseudo-wire networks to tackle the problem of setting end-to-end connections across heterogeneous domains using a multiple choice knapsack problem. As another previous work, J. Chen *et al* also in their work [19] borrowed the idea of multiple-choice knapsack problem for video stream selection used to propose an online video adaptation system with the aim of reducing the users' pricing.

In general the idea here is to apply textural compression (described in detail in section

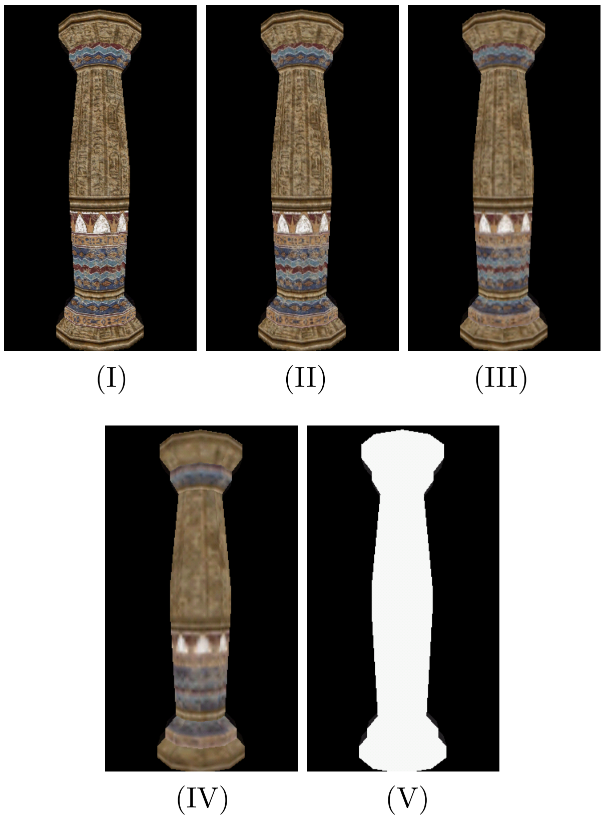


Figure 3.2: Visual view of applying a texture with different resolutions to a 3D column object. (I to IV): the texture resolutions are 512×512, 256×256, 128×128 and 64×64 with the same depth color. (V): no texture.

3.3) to reduce the resolutions, and hence the sizes, of some of the textures so that a texture for every object can be streamed within the download budget. Our heuristic algorithm to choose which textures to compress, and by how much, is described in section 3.2.

Figure 3.2 shows the visual impact of reducing the resolution of a texture which is applied to a 3D column object. The texture resolutions in Figures 3.2 (I) through (IV) are 512×512, 256×256, 128×128, and 64×64, respectively (all with 8-bit color depth). Figure 3.2 (V) shows the object with no texture applied. The reduction in quality in Figures 3.2 (I) through (IV) is noticeable, but the impact of even the greatest reduction (Figure 3.2 (IV)) is much less than the impact of not applying a texture. So sending a low quality texture (around only 4KB for the 64×64 case) is far more acceptable than sending nothing.

3.2 Adaptive Texture Streaming

The size of textures plays an important role in resource usage on mobile devices. If textures are large, but account for less important objects (such as sky background during a fight with the enemy) or if they are always rendered at a small size in the scene (such as a house in the far distance), then the mobile device is wasting a lot of resources not only to receive them via WNIC, but also to render them using valuable CPU/GPU resources. Thus, we must take into consideration how important textures will be when displayed in the scene.

Streaming data to mobile players in real time is expensive both in terms of bandwidth and computation. Besides streaming the bulky textures and objects, it requires passing data related to dynamism regarding the inter-relationship of 3D objects, view change detection, frustum culling, motion interpolation and extrapolation, and so on. Therefore as a key part of streaming for networked games, especially for MMOGs, due to the high number of textures in a 3D scene, the required set of textures is streamed in advance to be stored locally [21, 39, 45].

In gameplay, some textures are more important than others purely from the gaming context. As an example, the walls, floor, and some environmental textures in a typical game are not as important as the players' avatars, the enemies, and the goal objects. Therefore the first step in our method is to establish the importance of each texture within a scene of gameplay. We do this by allowing a designer to tag textures into two different importance classes: Less-Important (C_1) and Important (C_2). Currently, some 3D game engines (such as Unity3D) [10] support multiple levels of tagging (e.g. Player or Enemy tags) to identify the gaming objects for scripting purposes. Thus, adding a feature of two-level tagging for textures is not a considerable overhead for game design. In our thorough study in the next chapter, we will use multiple level of tagging for textures. We have only used two levels in this initial study for simplicity.

First, a list is created containing all textures classified by their importance class. Each τ_{ij} in the list represents a single texture where i signifies the importance class, and j is the index of the texture within that specific class, as shown in Figure 3.3.

Figure 3.4 is a hierarchical pyramid that shows the possible texture resolutions used in this study. As can be seen in Figure 3.4, the size of a 128×128 texture compared to a 256×256 is 1 to 4, if both have the same color depth. Thus for every index i and j, the

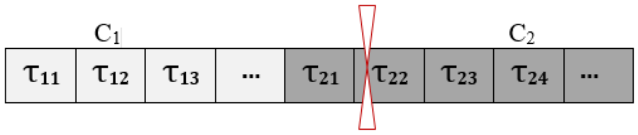
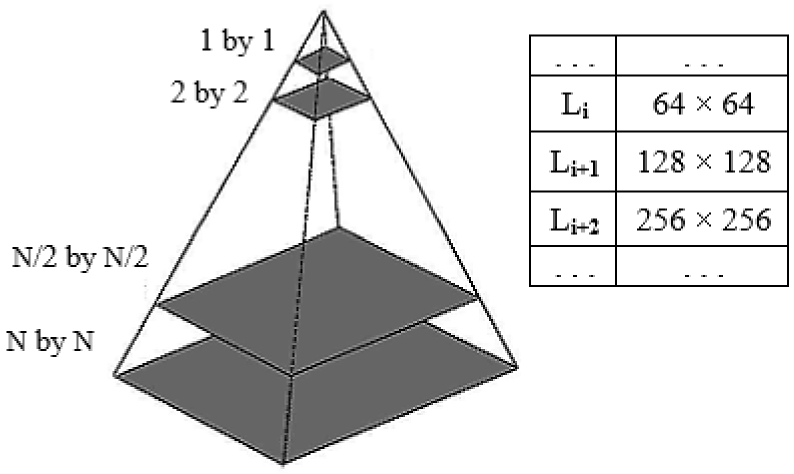


Figure 3.3: An overview of the texture list, showing two different priority classes, C_1 and C_2, along with the corresponding textures. The red sign represents a hypothetical available budget, shown as a cutoff.

following equation can be written:

$$Size(L_i) = 4^{(i-j)} \times Size(L_j) \qquad i \geq j > 0 \tag{3.2}$$



...	...
L_i	64×64
L_{i+1}	128×128
L_{i+2}	256×256
...	...

Figure 3.4: Different texture resolutions represented as various levels, along with a hierarchical view.

In this initial study, we have used a simple pyramid downsampling approach to compressing the textures. More sophisticated compression methods are available and the compression can be done in advance. Our approach can be used in these more general settings with only minor adjustments as long as the relationships among the compression levels are known.

Now we describe how our heuristic algorithm works.

Our problem is to stream textures to a mobile device in a way that maximizes the total quality of the streamed textures within an energy budget that is specified by the user of the mobile device. We cannot know the energy consumption associated with textures in advance, but as argued in Chapter 2, it is closely related to the sizes of the textures needed to be streamed to the client. So we estimate a download limit D based on the energy budget.

The size of a texture can be reduced by reducing its resolution, but this also reduces the quality. To take this into account, we set a user-defined maximum level of compression. Let R_{max} be the maximum reduction in resolution that is acceptable to the mobile user. In this study, we assume that $R_{max} = 4^i$ for some $i \geq 0$.

As an example, if the original size of a specific texture x is s_x and $R_{max} = 4^i$ with $i = 2$, then the size of the smallest acceptable compressed texture will be $s_x/R_{max} = s_x/16$.

The textures are classified into two importance classes, less-important (C_1) and important (C_2), and each class has an associated relative value (v_1 and v_2). The assignments to classes and the associated values are decided by the game designer. We normalize the values by setting $v_1 = 1$. Let S_1 be the total size of textures in C_1 before compression, S_2 the total size of textures in C_2 before compression, and $S{=}S_1{+}S_2$.

We assume that the quality of a streamed texture is a function of its size (with maximum quality corresponding to minimum compression) and its relative value. Our approach is general and, with minor adjustments, will work with any function that can be effectively computed. In this initial study, we use the simplest of these functions - the product of size and value. For example, a texture τ_i of original size s_{τ_i} with compression or scaling factor r_i and value v_{τ_i} has quality $\frac{s_{\tau_i} \times v_{\tau_i}}{r_i}$. Our goal is to compress textures in a way that maximizes the total quality subject to the constraints of the download budget D and maximum reduction R_{max}.

Algorithm

Calculate S_1 (Total size of all textures in C_1), S_2 (Total size of all textures in C_2), S ($S_1{+}S_2$), D (Available budget).

- If $S \leq D$, then no compression is needed.
- If $\frac{S}{R_{max}} > D$ then the problem cannot be solved within the constraints D and R_{max}.
- Otherwise, we solve one of the following subproblems.

Subproblem 1:

If $S_2 + \frac{S_1}{R_{max}} \leq D$ then all textures in C_2 can be sent uncompressed and the problem is to compress the textures in C_1 in a way that maximizes the quality of the compressed textures in C_1 within the download budget $D_1 = D - S_2$.

Subproblem 2:

If $S_2 + \frac{S_1}{R_{max}} > D$ then we compress all textures in C_1 by R_{max} and the problem is to compress the textures in C_2 in a way that maximizes the quality of the compressed textures in C_2 within the download budget $D_2 = D - \frac{S_1}{R_{max}}$.

Algorithm for subproblem 1:

Calculate i_1 such that:

$$\frac{S_1}{4^{i_1}} \leq D_1 < \frac{S_1}{4^{(i_1-1)}}. \tag{3.3}$$

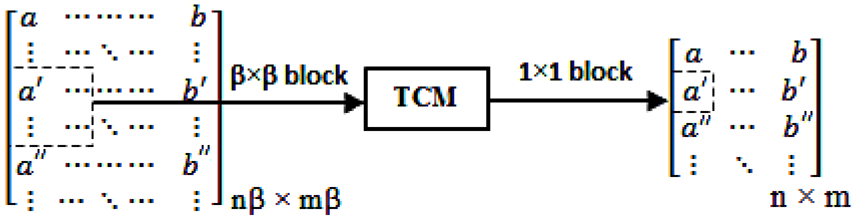
In other words, find the minimum i_1 such that all textures in S_1 can be streamed within the budget D_1 when they are resized by the factor 4^{i_1}.

Our goal is to maximize the total size of the textures sent within the budget D_1. To achieve this, we compress the first texture in C_2 by 4^{i_1}. Suppose that after compression, it has size x. This leaves a budget of $D_1 - x$ for the remaining textures. We then calculate a new i_2 for the remaining textures using budget $D_1 - x$ and compress the second texture in C_2 by 4^{i_2}. This is repeated until all textures have been processed.

Algorithm for subproblem 2:

The algorithm is the same as for subproblem 1, except that we are processing textures in C_2 with an available budget $D_2 = D - \frac{S_1}{R_{max}}$.

The algorithm iterates over the list of all textures in $O(n)$ time, supposing n is the number of all textures, and for each of the textures it finds a suitable scaling factor (i_1, i_2, etc.) in $O(c)$ time in which c is constant. These two cases make the complexity of the algorithm $O(nc) = O(n)$, which is a linear processing time.



```
int[][] TCM(int[][] src, int n, int m, int ratio){
    int[][] resized = new int[n/ratio][m/ratio];
    for(int i=0; i<m/ratio; i++)
        for(int j=0; j<n/ratio; j++){
        resized[i][j]=src[i*ratio][j*ratio];
        }
        return resized;
    }
```

Figure 3.5: TCM overall process along with the pseudo-code

3.3 Texture Compression

To make the quality of textures smaller, we designed a fast and efficient Texture Compression Module (TCM), which given a resizing ratio, β, makes the texture smaller by modifying the corresponding texture matrix. To achieve this, we simply choose a representative texel within each $[\beta \times \beta]$ block in the original texture. This representative texel could be any of the texels in the window block, or a mathematical mixture of them. In our experiments, we chose this representative texel to be the most top-left texel of each block. Figure 3.5 illustrates the transformation process along with the TCM pseudo-code. The right matrix represents a new compressed texture in which each texel is a representative chosen from each $[\beta \times \beta]$ block in the original texture. Figure 3.6 shows how TCM provides a 4×4 block when applied to a 16×16 texture with $\beta = 2$. Choosing the representative texels is a one-time operation, and our framework caches the produced textures for future uses.

It should be noted that currently there are several image resizing algorithms, such as Lanczos, bilinear, trilinear, and bicubic algorithm [17]. However, these algorithms are complex and applying them to all textures could cause too much overhead for the CPU and decrease the system response time. The compression method that we use in this study imposes very little demand on the CPU, but could result in lower quality than other methods depending on how quality is measured. Figure 3.8 shows a comparison of relative execution

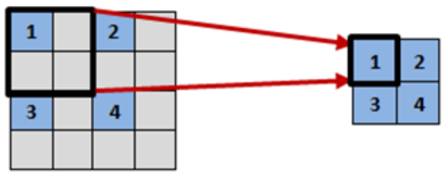


Figure 3.6: An example overview of the TCM procedure when applied to a 16 × 16 sample with β =2. The top-left most texel of each [2 × 2] window block would be chosen as the representative texel during each iteration.

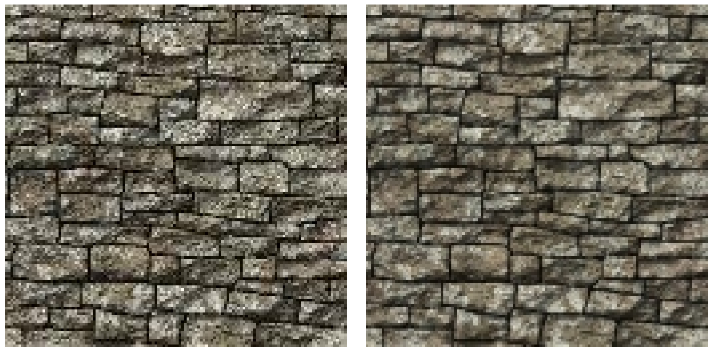


Figure 3.7: A 128 × 128 stone wall texture produced by TCM (left) compared with the bicubic resampling algorithm (right).

times for different resizing methods, and Figure 3.7 compares an output texture produced by TCM with a complex bicubic resizing algorithm. This is an interesting trade-off that merits further study.

3.4 Communication Channel

HTTP-based progressive streaming has become a de facto standard for web-based data delivery. Streaming over HTTP also allows multiple clients/devices to receive many possible streams simultaneously. In addition, HTTP does not cause any NAT/firewall issues as is the case with other media transport protocols like RTP/RTSP [4].

To take advantage of the above gains, we used HTTP 1.1 as the protocol for the communication channel. Unlike HTTP 1.0 in which a separate connection to the same server is made for every resource request, HTTP 1.1 can reuse a connection multiple times to download the required resources. HTTP 1.1 communications therefore experience less latency as the establishment of TCP connections presents considerable overhead [22].

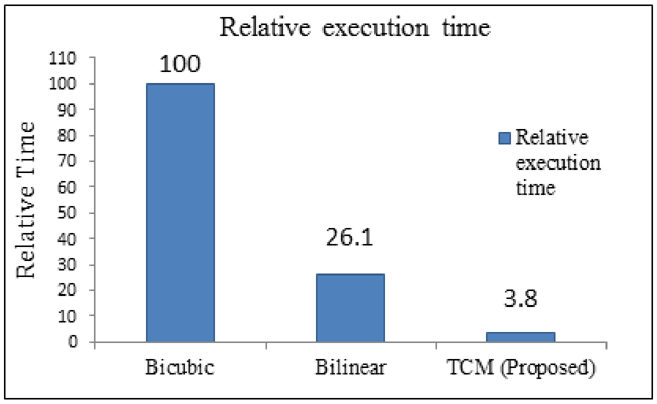


Figure 3.8: A comparison of relative execution time needed for running bicubic, bilinear and our proposed algorithm to resize a 512×512 texture by a ratio of 2 (β =2).

The concept being used in our framework is similar to Dynamic Adaptive Streaming over HTTP (DASH), which is an adaptive bitrate streaming technology developed under MPEG, where a multimedia file is partitioned into one or more pieces and delivered to a client using HTTP. One or more representations (i.e., versions at different resolutions or bit rates) of multimedia data are typically available, and selection can be made based on network conditions, device capabilities and user preferences, which enable adaptive bitrate streaming [46]. In our work we used a DASH-style approach for adaptive and progressive streaming of textures over HTTP. To the best of our knowledge, no previous work has used the idea of DASH for streaming of 3D graphics to mobile devices.

3.5 Evaluation

We used the Android port of the free jPCT 3D engine to evaluate our work. We prepared a benchmark for our experiments, called *Ninja Camp*, and ran it as a self-runner demo so that the tests were deterministic and not dependent on different gameplays.

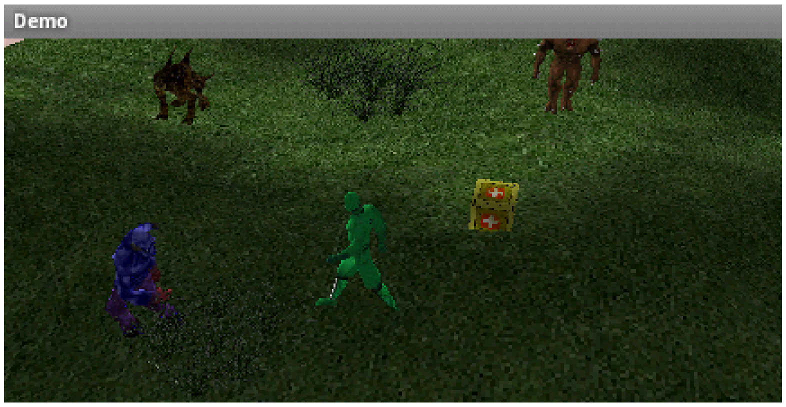
Ninja Camp is a third person streaming-based client/server 3D demo with high polygon and object count enabled with skeletal animation, consisting of 97 different textures. Based on the gaming context, the important and less-important textures account for 37.2% and 62.8% of total texture size, respectively.

To make the progressive streaming work, we used a multi-threaded implementation, so in parallel with streaming the objects/textures from the server, the previously streamed objects are built, rendered and added to the 3D world. We used HTTP as a communication protocol for progressive streaming of 3D textures and objects. 3D objects are not loss-tolerant and can therefore benefit from the reliable service of HTTP.

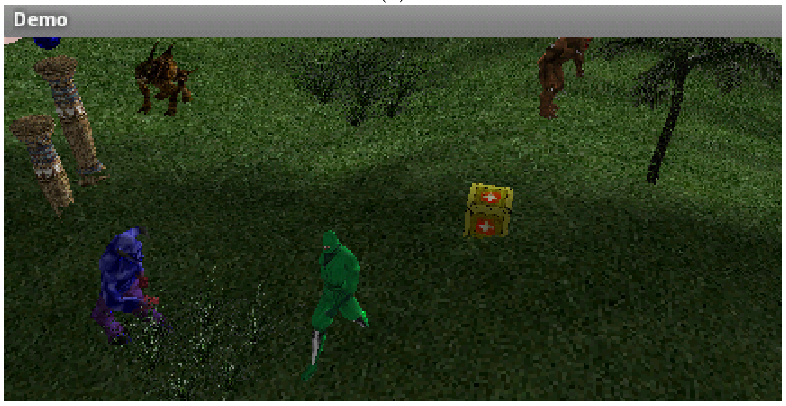
Our method uses a prioritization scheme based on the relative values of objects in the context of the current game scene. As an example, in our demo, the enemy or a health kit have a higher value than the trees and surrounding plants, so they should be shown with higher quality in terms of texture resolution, and they should be streamed before lower value objects. This is different than distance-based approaches that render with higher quality whatever is closer to the player regardless of their semantic relevance in the current game context. Figures 3.9 (I) to (III) show screenshots of different stages in the streaming of a particular scene based on the prioritization streaming approach. Figure 3.9 (III) shows the scene after all of the objects and textures currently in that scene have been streamed.

In our experiments to evaluate our proposed energy-efficient texture adaptation algorithm, the streaming server was an Intel 3GHz dual core machine running Java 1.7.0 standard edition. The server was an on-demand HTTP media server responding to HTTP requests from mobile access. During our experiments, the distance of the client device with the 802.11g WiFi router was 5 meters, receiving a signal strength of -60 dBmW. Our client device was an HTC 3D EVO smartphone which has a Snapdragon S3 chipset with a dual core 1.2 GHz processor. Appendix A provides more detailed information about this development smartphone. To calculate the amount of consumed energy, we used PowerTutor [51], a profiler which measures the power consumption of various hardware components using a device's built-in battery voltage sensors. By applying our proposed adaptations, we did not notice any decrease in the maximum frame rates, and NinjaCamp was able to run having 33 frames per second, which was actually no different than the run before adaptations. Therefore it is expected that our adaptations can be implemented in real-time at the maximum frame rate that cloud gaming services provide these days.

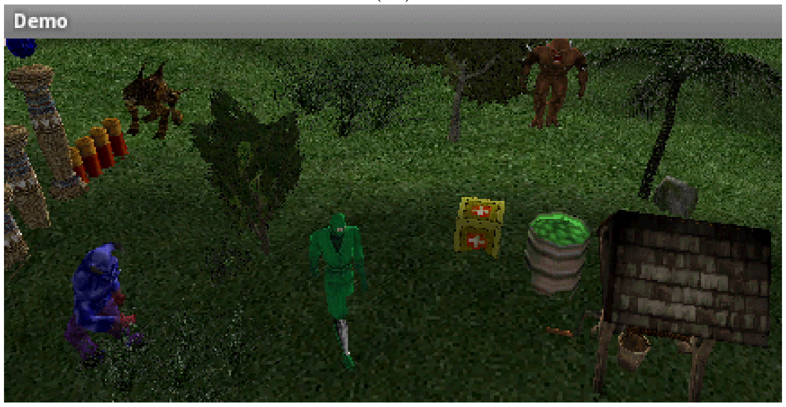
To evaluate our proposed texture adaptation algorithm, we ran our ninja benchmark with the available budget D set to be different percentages of S (total size of all textures). In particular, we set D to $0.1S$, $0.2S$, $0.3S$, ..., $1.0S$ corresponding to 10%, 20%, 30%, ..., 100% of the total size of textures. We chose two different values for R_{max} (i.e. the maximum acceptable compression scaling) that resulted in some textures in C_2 being compressed for most values of D. In practice, a user will choose R_{max} based on the perceived quality of the textures in the gaming environment. We also repeated all the experiments with the textures



(I)



(II)



(III)

Figure 3.9: (I) to (III): Different stages using prioritized progressive game streaming in the initial demo benchmark. Based on the gaming context, the player's avatar, the enemies and goal targets are more important as opposed to trees or grasses, so they are streamed in earlier stages.

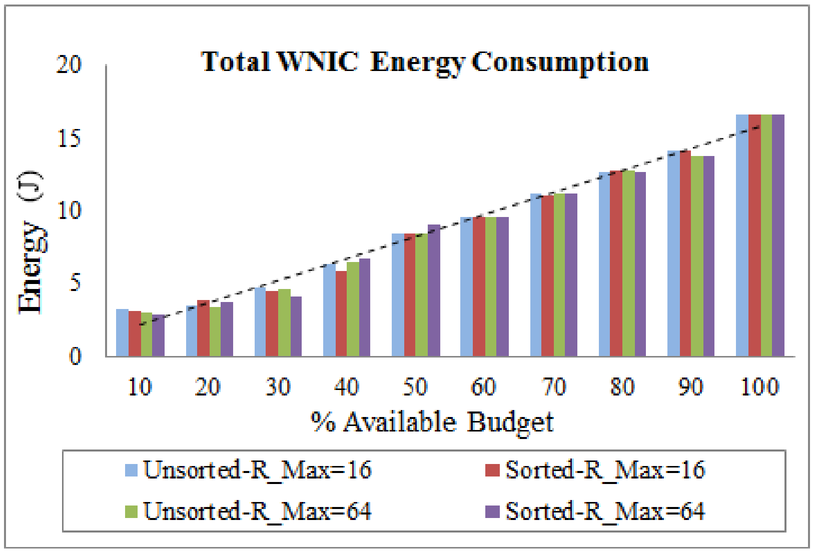
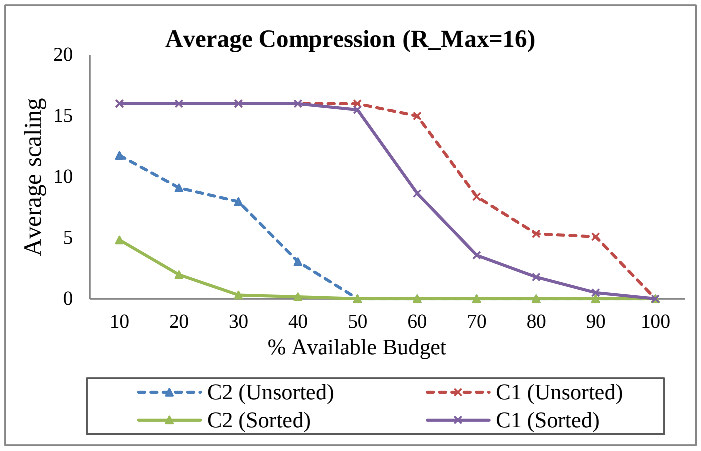


Figure 3.10: Total energy consumption for WiFi 802.11g Network Interface Controller.

in each importance class sorted by decreasing size. Each trial of our experiment was run until all objects and textures were fully streamed, and we repeated each test several times to ensure that the standard deviation of the measurements was within acceptable limits. Using PowerTutor measurements, Figure 3.10 shows the average energy consumption of the WiFi 802.11 WNIC in *Joules* with a precision of 0.1J, and showed how it changes as we apply our texture adaptation approach for ten different values of D (as a percentage of S). We measured the total energy for both sorted and unsorted texture lists, and for $R_{max} = 16$ and $R_{max} = 64$. As can be seen, the increase in total energy consumption grows roughly linearly with the amount of data being received at the client side in agreement with Equation (1.1) on page 6 and Figure 1.2 on page 7. The dashed line shows the linear trend-line for the energy consumption of the sorted texture list with R_{max} set to 16 (red bar).

Figure 3.11 show the results for the average compression in terms of average scaling ratio, measured for all textures in C_1 and C_2, for both sorted and unsorted texture lists. In Figure 3.11 (top) $R_{max} = 16$, while in Figure 3.11 (bottom), $R_{max} = 64$. In both graphs it can be seen that sorting improves the average quality by providing a lower average scaling ratio. Also, the larger value of R_{max} results in more scaling, and thus quality sacrifices, for textures in C_1 (i.e. textures tagged as *less-important*), while preserving more of the original textures in C_2 (i.e. *important* textures). Also it should be noted that with small download



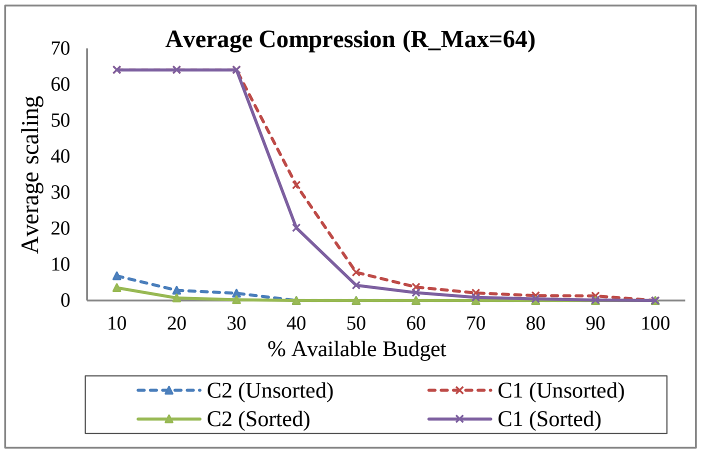


Figure 3.11: Average compression in terms of average scales, accounted for C_1 (i.e. Less-Important textures) and C_2 (i.e. Important textures), both for sorted and unsorted texture lists. (Top): R_{max} set as 16 (Bottom): R_{max} set as 64.

budgets, the larger value of R_{max} (R_{max}=64) works better since it brings more reduction in size, and thus can make it more possible to achieve the available budget.

As discussed previously, we assume that the quality of a streamed object is a function of its size and a relative value assigned by the game designer to its class. If S_1' and S_2' are the total sizes of the streamed textures in classes C_1 and C_2, respectively, that are received by the client, and v_1 and v_2 are the associated relative values, then we define the *total quality* of a scene to be

$$\text{Total Quality} = \frac{v_1.S_1' + v_2.S_2'}{S} \tag{3.4}$$

where S is the total size of the uncompressed textures. We normalized the relative values by setting v_1=1.

The total quality is a measure of the effectiveness of an approach to maximizing the total amount of data based on prioritizations, with larger values being more effective.

Figure 3.12 shows our experimental results for total quality per available budget for sorted and unsorted texture lists and two different values of R_{max} (16 and 64). We used three different pairs (v_1=1, v_2) of relative values, (1,1), (1,2) and (1,3), to differentiate the priorities of textures in C_1 and C_2. As can be seen in all four graphs, the total quality increases as the ratio $\frac{v_2}{v_1}$ increases, confirming that our proposed approach noticeably distinguishes the important textures from the less important ones. Interestingly, the maximum gap for the growth of the total quality is in the range of 30% to 50% of the available budget in all four graphs. Based on these figures, it can be concluded that ratios between the smallest and largest relative values that are larger than 1:3 are not likely to be interesting.

Figure 3.13 shows our results for quality per unit of energy for the same experiments as in Figure 3.12. As can be seen, in all four graphs, larger $\frac{v_2}{v_1}$ ratios result in more quality per unit of energy consumption. In all of the graphs, the maximum gain in quality is in the range of 30% to 40% of the available budget suggesting that this the range where our approach is most effective. This range is related to the ratio of important to less-important textures in our proof-of-concept game in which 37.2% of all textures are considered important. Our approach brings maximum in quality when all the less-important-tagged textures (textures in C_1) are compressed by R_{max}, and none of the important-tagged textures (textures in C_2) are modified. This specific point is considered as the peak of quality. As we go ahead with compressing the textures in C_2, the gain in quality brought by our approach is being decreased; a fact which is confirmed by tracking the textures being compressed in C_1 and C_2 and along with the achieved quality.

Figure 3.14 shows a visual comparison of a scene with no adaptation and the same scene using our approach. In both scenarios the available budget of the client device is 50% of the total size of all textures *before* compression. In Figure 3.14 (I), due to the budget limit, some textures have not been transferred, while in Figure 3.14 (II), our proposed texture adaptation streams all of the textures, but reduces the resolution of less-important textures (i.e. plane, columns, well, dragon monster, etc.). As can be seen in the screenshots, Figure 3.14 (I) does not provide a pleasing or acceptable game experience. Compared with the original demo in Figure 3.9 (III) in which the download budget is unlimited, our method does produce noticeable differences in quality. However, considering the power savings achieved using our

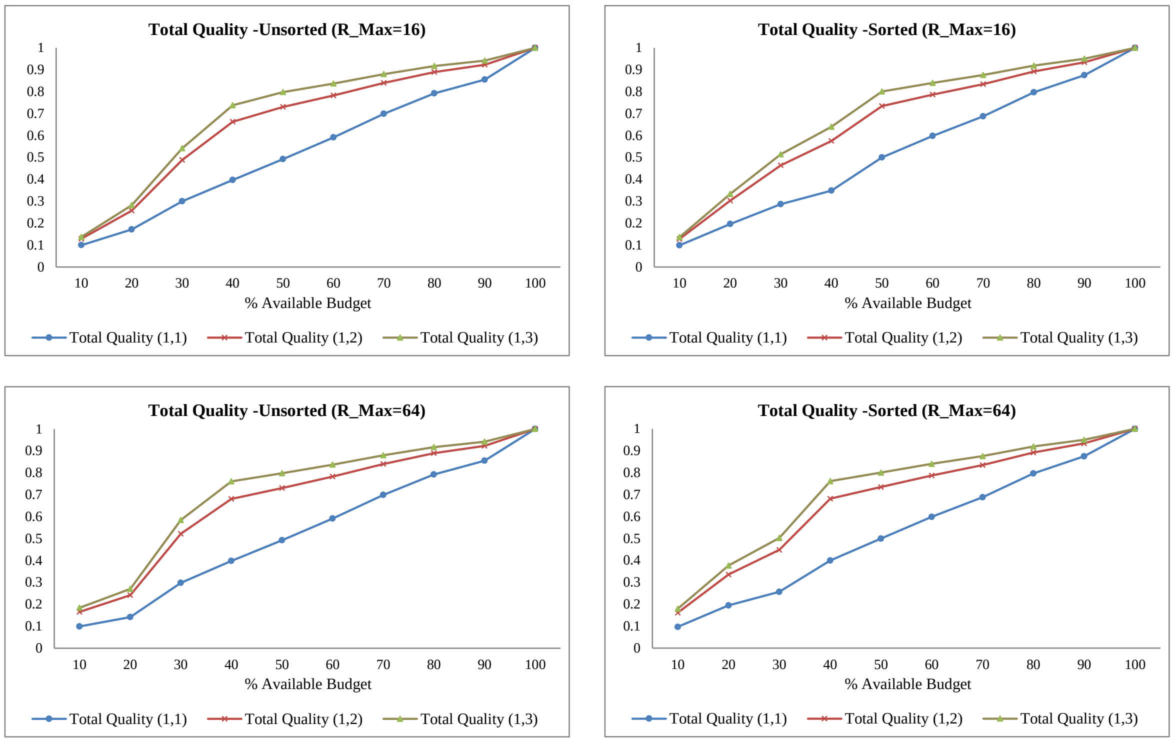


Figure 3.12: Comparison of the total quality measured for four situations and three relative value pairs (v_1=1,v_2). (Top-Left) Unsorted texture list, R_{max}=16. (Top-Right) Sorted texture list, R_{max}=16. (Bottom-Left) Unsorted texture list, R_{max}=64. (Bottom-Right) Sorted texture list, R_{max}=64.

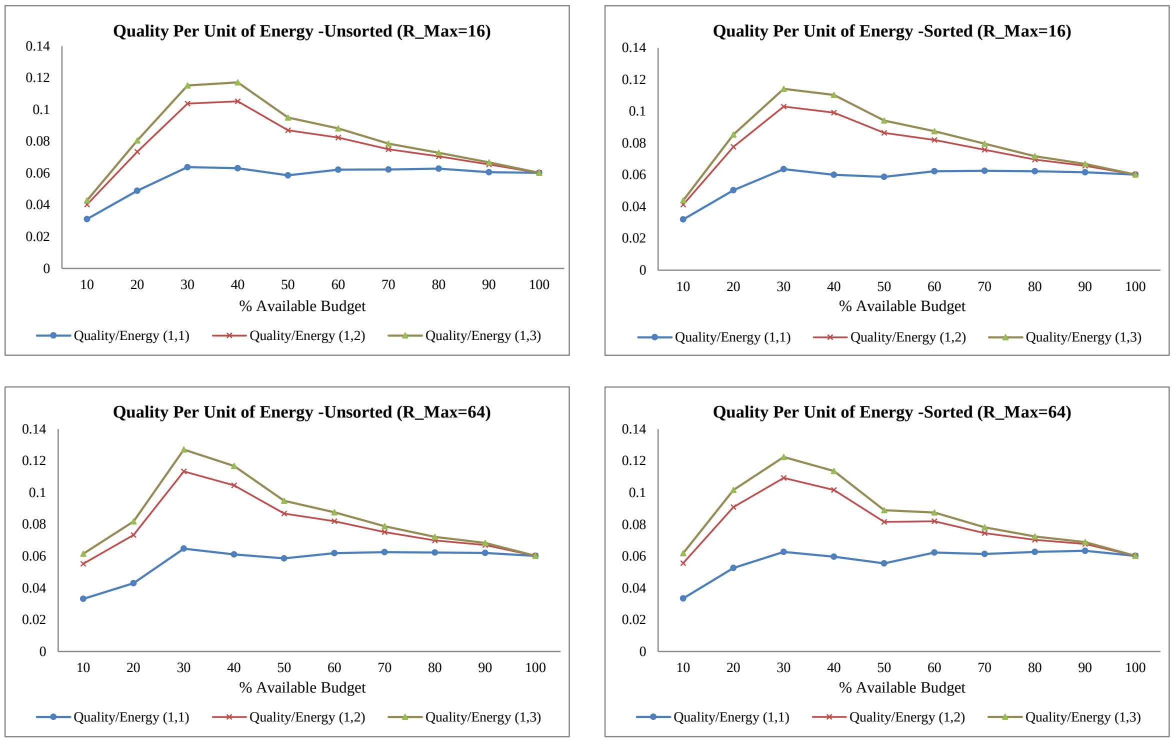
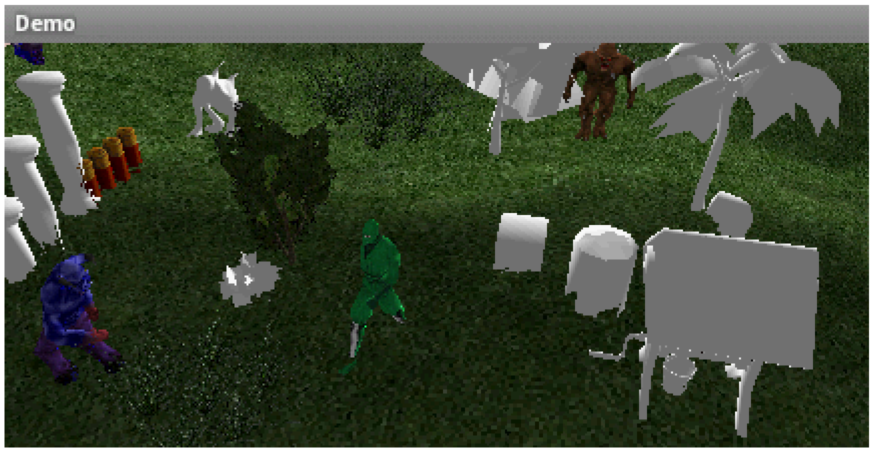
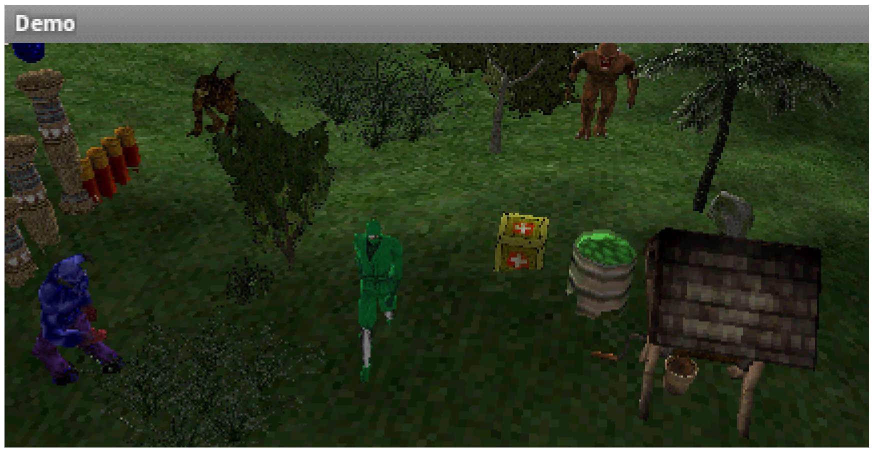


Figure 3.13: Quality per unit of energy for four situations and three relative value pairs (v_1=1,v_2). (Top-Left) Unsorted texture list, R_{max}=16. (Top-Right) Sorted texture list, R_{max}=16. (Bottom-Left) Unsorted texture list, R_{max}=64. (Bottom-Right) Sorted texture list, R_{max}=64.



(I)



(II)

Figure 3.14: Visual comparison of a gaming scene with a 50% download budget with (I) No adaptation and (II) after applying our proposed adaptation method. R_{max} was set to 16 and the textures list was sorted.

method, it is reasonable to believe that many players would make this sacrifice in quality in exchange for respecting their available download and/or energy budgets.

3.6 Shortcomings

In this initial study, we made some simplifying assumptions that are not very realistic. The main simplifications are the following.

- There is a single static scene and everything is known in advance. All textures are streamed at the beginning of the game and nothing changes.
- Our algorithms are strictly offline.
- Our study did not include a benchmark.

In the next chapter, we will run a detailed and thorough experimental design that addresses the above-mentioned shortcomings, focusing on studying mobile games with dynamic scenes. The idea is to introduce random deviations of the actual energy used to stream a texture compared to the predicted energy based on the size of the texture. We will introduce three algorithms, an offline version considered as our near-optimum benchmark, and semi-online and online versions with the aim of reacting to the deviations.

Chapter 4

Proposed Approach: Dynamic Scenes

In this chapter we present our study for adaptive 3D texture streaming for dynamic scenes.

4.1 Motivation

In our initial study, we studied the streaming of 3D data for an online game scene. The data consists of textures that are overlayed on the 3D objects. The game is played on a mobile device with a limited amount of energy in its battery. The energy usage depends on the amount of textural information that is streamed to the device. If the energy is insufficient to stream all of the textures, then our approach is to selectively reduce the sizes of some of the textures by reducing their resolutions. Some textures are more important than others and there are several choices of resolution. The user specifies an energy budget and a minimum acceptable resolution, and the game designer assigns relative values to the textures according to their importance. The value of a streamed texture is its relative value times its size. This results in a multiple-choice knapsack problem with the goal of maximizing quality, measured as the total value of the streamed textures, within a specified energy budget. Our experimental results using simple greedy heuristics showed significant improvements in quality per unit of energy consumed.

As argued before, the problem that we studied is static in the sense that one scene is streamed and all information is known in advance. A more realistic scenario is for dynamic scenes, in which the scene changes as the player moves around in the game space, so some objects disappear and new ones appear as the game progresses. The goal is now to achieve a target game session time within a download budget (which depends on the energy budget), with the best possible quality averaged over the entire game session.

4.2 Definitions

There are n textures $\mathcal{T} = \{\tau_1, \tau_2, \ldots, \tau_n\}$, and each $\tau_h \in \mathcal{T}$ has a full-resolution size s_{τ_h}, and a relative value or importance v_{τ_h}. The quality of texture τ_h is $q_{\tau_h} = v_{\tau_h} \times s_{\tau_h}$.

The duration of the game session is T, and the energy budget limits the total size of the textures that can be downloaded to D.

Let $X = \{x_1, x_2, \ldots, x_n\}$, be the set of textures that are streamed to the mobile device. Each $x_i \in X$ corresponds to an original texture $\tau_i \in \mathcal{T}$. Each x_i has an arrival time a_{x_i} when it becomes visible, a departure time d_{x_i} after which the texture can be deleted, and value $v_{x_i} = v_{\tau_i}$.

The size of a texture x_i might be smaller than the size of τ_i because the download constraint, D, might force the scaling (reducing the resolution) of some textures.

The size of a streamed texture $x_i \in X$ is the full-resolution size of the texture $\tau_i \in \mathcal{T}$ scaled by a factor of c^{r_i} for some $0 \leq r_i \leq k$ and constant c where $R_{max} = c^k$ is specified by the user as the maximum reduction of resolution that can be tolerated. So, the size of x_i is $s_{x_i} = \frac{s_{\tau_i}}{c^{r_i}}$. The quality of the streamed texture x_i is $q_{x_i} = v_{x_i} \times s_{x_i} = \frac{v_{\tau_i} \times s_{\tau_i}}{c^{r_i}}$.

We can divide the game time T into p periods where a period is a maximal contiguous period of time during which there are no changes to the textures that are visible. Let t_ℓ be the time that the ℓ^{th} period ends and let $t_0 = 0$. Then the ℓ^{th} period is $(t_{\ell-1}, t_\ell]$, and $0 = t_0 < t_1 < t_2 < \cdots < t_p = T$. Each t_ℓ will be a_{x_i} or d_{x_i} for at least one $x_i \in X$. (Note that multiple events can happen at each t_ℓ including the possibility that the entire scene changes.)

Let $X_\ell \subseteq X$ be the set of textures visible during the ℓ^{th} period and let $X_0 = \emptyset$. The total quality of X_ℓ is $Q_\ell = \sum\limits_{x_i \in X_\ell} q_{x_i}$ and the optimization problem is

$$\begin{aligned}
&\text{maximize} && \sum_{\ell=1}^{p} Q_\ell \times (t_\ell - t_{\ell-1}) = \sum_{\ell=1}^{p} \left(\sum_{x_i \in X_\ell} v_{x_i} \times \frac{s_{\tau_i}}{c^{r_i}} \times (t_\ell - t_{\ell-1}) \right) \\
&\text{subject to} && \sum_{\ell=1}^{p} \left(\sum_{x_i \in X_\ell \setminus X_{\ell-1}} \frac{s_{\tau_i}}{c^{r_i}} \right) \le D \quad \text{and} \quad r_i \le \log_c (R_{max})
\end{aligned}$$

These equations can be simplified because most of the values are known in advance. Furthermore, by taking an amortized view of the problem we can eliminate the double summations. The result of these simplifications is

$$\begin{aligned}
&\text{maximize} && \sum_{x_i \in X} q_{x_i} \times (d_{x_i} - a_{x_i}) = \sum_{i=1}^{n} \frac{v_{x_i} \times s_{\tau_i} \times (d_{x_i} - a_{x_i})}{c^{r_i}} \\
&\text{subject to} && \sum_{i=1}^{n} s_{x_i} = \sum_{i=1}^{n} \frac{s_{\tau_i}}{c^{r_i}} \le D \quad \text{and} \quad r_i \le \log_c (R_{max})
\end{aligned}$$

Noting that the quantities $v_{x_i}, s_{\tau_i}, d_{x_i}, \quad \text{and} \quad a_{x_i}$ are all known in advance, we can define $z_i = v_{x_i} \times s_{\tau_i} \times (d_{x_i} - a_{x_i})$, $1 \le i \le n$ to get

$$\begin{aligned}
&\text{maximize} && \sum_{i=1}^{n} \frac{z_i}{c^{r_i}} \\
&\text{subject to} && \sum_{i=1}^{n} \frac{s_{\tau_i}}{c^{r_i}} \le D \quad \text{and} \quad r_i \le \log_c (R_{max})
\end{aligned}$$

Note that c is a constant ($c = 4$ in our study), so the only variables are the scaling factors $r_1, r_2, \ldots, r_n$. These are the values that the algorithm has to determine.

4.3 Simulation Basics

In this section we talk about simulation basics, seed selection, and random number generation needed for simulating a system.

4.3.1 Random Number Generation

As it is known, one of the key concepts in running simulations is to have functions to generate random values for variables with a specified statistical distribution. There are two steps for doing so: First, a sequence of random numbers distributed uniformly between 0 and 1 is obtained, which we call random number generation. For the second step, the generated sequence is transformed into another sequence with the aim of generating random values satisfying the desired distribution. This is called random-variate generation [29].

4.3.2 Seed Selection

One of the important issues regarding random generations is the generator's period, which actually shows *how much randomness* the generator can generate. But a more important issue is how we seed or initialize a random number generator.

Seeds are used to initialize a random number generator. They should not affect the results of the simulation. However, a wrong combination of a seed and a random generator may lead to flawed conclusions.

As an example to understand the concept, let's consider Java's standard generator *java.util.Random*. This class uses a 48-bit seed, which causes a period of 2^{48}. We can imagine this random generation is a huge wheel with 2^{48} random cuts [5]. Whenever we create an instance of *java.util.Random*, the wheel starts in a *random* place, and moves round by one cut every time we generate a number from that instance. Wherever we start from, we will end up back at the same place after generating the 2^{48} numbers. If the place where we begin is *truly random*, then a sequence length of 2^{48} is probably sufficient to use for our application. But how do we pick a random place on the wheel to start from? The random place that we start from is in effect the seed (i.e. the initial state) of the random generator. The solution which *java.lang.Random* offers is to use one of the system clocks [5].

In our simulation, for each of our experiments, we generate 5 different random integer numbers, and use them as our seeds for the random generator functions. For each random sequence $p_1, p_2, p_3, \ldots$ that we need to generate, we use *Random* class to get a seed p_0. We print p_0, and store it in a file and then use p_0 as the seed to generate the sequence p_1, p_2, p_3, and so on. We can use p_0 again later if necessary to generate exactly the same sequence. This way, our experiments are repeatable.

4.3.3 Statistical Distributions

A statistical distribution explains the numbers of times each possible outcome occurs in a sample. For instance, assume we have 10 test scores with 4 possible outcomes of A, B, C, or D, a statistical distribution describes the relative number of times an A,B,C, or D occurs. For example, 2 A's, 4 B's, 4 C's, 0 D's.

In the following we enumerate a few specific statistical distributions that we used in our simulation system:

- **Uniform Distribution:**

This is one of the simplest distributions to use. A uniform distribution is commonly used if a random variable is bounded and no extra information is available. The discrete uniform distribution is used when it is believed that the value is equally likely over a bounded interval.

- **Poisson Distribution:**

  If the interarrival times are exponentially distributed with mean 1, the number of arrivals n over a given period T has a Poisson distribution with parameter T. Therefore, a Poisson variate can be obtained by continuously generating exponential variates until their sum exceeds T and returning the number of variates generated as the Poisson variate.

- **Exponential Distribution:**

  It describes the time between events in a Poisson process, i.e. a process in which events occur continuously and independently at a constant average rate. In order to generate exponentially distributed random numbers with a specific mean, we use equation 4.1. *Ln* function returns the natural logarithm (base e) of a double value, and *rnd* variable is the generated random float number ranging from 0 to 1.

$$\text{Exponential}(mean) : -mean \times Ln(1 \text{ - } rnd) \tag{4.1}$$

4.3.4 Standard Deviation Estimation

One of the important parts that we should take into account is regarding the length of the simulations. We should assure that the length of the simulation experiments is properly chosen. Two situations may occur: If the simulation is too short, the results might be highly variable. Conversely, if the simulation is too long, the available computation resources might be unnecessarily wasted. It follows that the simulation should be run until the confidence interval for the mean response narrows to a desired width [29].

In our simulations, each of the experiments was run for 10 times, and we calculated the average and the standard deviation of the sample mean of our 10 independent observations. Based on the results, we noticed that the standard deviation for all the different sets of 10 experiments is within 2% of the averages, which is an acceptable value. This is valid only if the observations are independent. That's why we use different seeds to generate random numbers, so to make sure all the results are independent.

In the next section we explain how to generate different test sequences for our simulation.

4.4 Test Sequences

As argued before, we have a set $\mathcal{T} = (\tau_1, \tau_2, \tau_3, \ldots, \tau_{97})$ of full-resolution textures (in some arbitrary order) and each τ_h has a full-resolution size s_{τ_h}. We already have this data from our initial study. From these, we will generate a set of test sequences for our experiments. This process only has to be done once.

We used two relative values in our initial study for static scenes, while we use three in this study. Based on our study for static scenes in Chapter 3, ratios between the smallest and largest relative values that are larger than 1:3 are not likely to be interesting, so each texture τ_h should have a relative value $v_{\tau_h} \in \{1, 2, 3\}$.

Assume that the game is to be played for time T. We need to generate the arrival time a_{x_i} and departure time d_{x_i} for each streamed texture x_i. We also need to generate deviations from the energy estimates.

We need to generate several sequences of random numbers to create the test sequences. Most random number generators produce uniformly distributed random numbers in the interval (0,1). It is very important to use different seeds for the sequences. As argued before, using the same seed, for example for both the sequence of arrival times and the sequence of relative values, will introduce correlations that will invalidate the results [29].

1. *Generating 10 random permutations of $\mathcal{T}$: $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_{10}$:*
   For each $\mathcal{T}_j$, we generate a sequence of random numbers and associate each random number with a texture. We then sort the (random number, texture) pairs according to the values of the random numbers. This will produce a random permutation of the textures.

2. *Generating a sequence of relative values for each $\mathcal{T}_j$:*
   For each $\mathcal{T}_j$, we generate a sequence of uniformly distributed random numbers in (0,1). Each value will be multiplied by 3, and round up to the closest integer to get the relative value v_{τ_i} for each texture τ_i.

3. *Generating the times that the textures will be visible for each $\mathcal{T}_j$:*
   We will add these times to the arrival times to get the departure times. These times are exponentially distributed, with a mean value of $T/5$ which based on the experiments can

be a reasonable choice. While generating these numbers, we throw away any values that are less than $T/10$ or greater than $T/2$ and keep generating values until there are 97 values between $T/10$ and $T/2$. Assume calling this sequence of numbers $f_1, f_2, \ldots, f_{97}$.

4. *Generating the arrival and departure times of the textures in each $\mathcal{T}_j$:*
   These are the times that the textures arrive in the game scene. The usual assumption is a Poisson arrival process. As said in previous section, this means that the interarrival times are exponentially distributed (an interarrival time is the time between two consecutive arrivals). To do this, we generate exponentially distributed random numbers with mean 1: $g_2, g_3, \ldots, g_{97}$. Next, we calculate the sequence $h_1 = 0, h_2 = h_1 + g_2, h_3 = h_2 + g_3, \ldots, h_{97} = h_{96} + g_{97}$. We then scale the values $h_1, h_2, \ldots, h_{97}$ as follows to get a sequence of arrival times in the range $[0, T - T/10]$. Let $H = \frac{9T}{10h_{97}}$. The sequence of arrival times is $a_1 = h_1 \times H, a_2 = h_2 \times H, \ldots, a_{97} = h_{97} \times H$ and the corresponding departure times are $d_i = \min\{a_i + f_i, T\}$. It should be noted that each departure time is no larger than T which is the end of the game.

5. *Generating random deviations of energy consumption for the textures in each $\mathcal{T}_j$:*
   For each $\mathcal{T}_j$, we first generate a sequence of uniformly distributed random numbers in (0,1), subtract 0.5 from each number, and then multiple each number by .02. This gives a sequence of numbers $e_1, e_2, \ldots, e_{97}$ in the range (-.01,.01). In the experiments, we will multiply these numbers by $\delta = 5$, $\delta = 10$, $\delta = 15$, or $\delta = 0$ to get random deviations of 5%, 10%, 15%, or 0% (no deviations).

4.5 Algorithms

Let T be the duration of the game and let D be the estimated download budget. The maximum compression that the user will tolerate is $R_{\max} = c^k$. In our initial study for static scenes in the last chapter, $c = 4$. Similarly, let C_1 be the class of textures with relative value 1 in a test sequence showing the least important class of textures, and let S_1 be the total full-resolution size of the textures in C_1. Likewise for C_2, S_2, C_3, and S_3. Let $S = S_1 + S_2 + S_3$ be the total size of all textures. Note that S is the same for all test sequences, but S_1, S_2, and S_3 will be different. So it should be noted that similar to our initial study, C_1 shows the least important class.

In our experimental setup, we used the same 97 textures as our initial study. We assumed

Size	**number of textures**
128	23
256	52
512	16
1024	6

Table 4.1: The number of different textures used in our studies

the game is running for 600 seconds (T=600), and we have a variable budget D starting from 0.1 to 1 (full) share of the total size of the textures. Table 4.1 shows the number of textures with different sizes.

4.5.1 Non-adaptive Benchmark Algorithm

This algorithm determines the maximum possible quality that can be achieved for a test sequence if the algorithm knows everything in advance. It provides an upper bound.

For each texture τ_i in the list, we calculate $z_i = v_{\tau_i} \times s_{\tau_i} \times (d_{\tau_i} - a_{\tau_i})$. This is the contribution that τ_i would make to the average quality of the game if it were sent at full resolution. We also calculate $s'_{\tau_i} = s_{\tau_i} \times (1+\delta e_{\tau_i})$. This value is the amount of the download budget that the texture would use if streamed at full resolution. It should be noted that s'_{τ_i} takes into account the actual amount of energy that the texture will use. We then calculate $S' = \sum_{i=1}^{97} s'_{\tau_i}$ and $D_{\min} = S'/R_{\max}$ which is the minimum download budget that is needed to stream all textures. Same as our initial study, if $D_{\min} > D$ then the problem cannot be solved. In the following assume that $D_{\min} \leq D$ so the unused download budget is $D_0 = D - D_{\min}$.

To determine the compression for each texture, we sort the list by z_i/s'_{τ_i} ratio from largest to smallest. For ease of notation in the following, suppose that the textures are re-indexed so that the sorted list of textures is $\tau_1, \tau_2, \ldots, \tau_{97}$. If $s'_{\tau_1} \times (1 - 1/R_{\max}) \leq D_0$ then there is enough unused budget to stream τ_1 at full-resolution, so the streamed texture x_1 has $s_{x_1} = s_{\tau_1}$ and contributes z_1 to the average quality. This leaves an unused download budget of $D_1 = D_0 - s'_{\tau_1} \times (1 - 1/R_{\max})$ for the remaining textures after x_1 has been streamed. We repeat for $\tau_2, \tau_3, \ldots$ until some texture τ_ℓ cannot be streamed at full resolution within the remaining budget $D_{\ell-1}$. We then determine the maximum resolution at which it can be streamed by calculating the minimum compression r_ℓ which similar to R_{max} is a power of 4 such that $s'_{\tau_\ell}/r_\ell \leq D_{\ell-1} + s'_{\tau_\ell}/R_{\max}$. The streamed texture x_ℓ will have size $s_{x_\ell} = s_{\tau_\ell}/r_\ell$ and will contribute $z'_\ell = v_{\tau_i} \times s_{x_\ell} \times (d_{\tau_i} - a_{\tau_i})$ to the average quality of the game. The remaining download budget after streaming x_ℓ will be $D_\ell = D_{\ell-1} - (s_{\tau_\ell}/r_\ell) \times (1 + \delta e_{\tau_\ell})$. We repeat this process to determine the size, amount of download budget, and quality contribution for each of the remaining streamed textures $x_{\ell+1}, x_{\ell+2}, \ldots, x_{97}$. Finally the total quality and other statistics are calculated.

The algorithm sorts the list of all textures in $O(n \cdot lg(n))$ time supposing n is the number of all textures, iterates over the list of all textures in $O(n)$, and for each of the textures tries to maximize the resolution which is done in $O(c)$ in which c is constant. These cases make the total complexity of the algorithm $O(n \cdot lg(n) + n \cdot c) = O(n \cdot lg(n))$. The memory complexity of the algorithm is $O(n \cdot c) = O(n)$, in which c is the constant number of the possible resolution levels for the textures. Also it should be noted that similar to our previous texture adaptation approach proposed in Chapter 3, firstly the algorithm needs a one-time implementation prior to the gameplay. Secondly, all the implementations are done on the cloud, or the server-side. We assume the server has enough available resources which is not a concern for our study. Therefore the algorithm does not provide any additional overhead during the game-play, and it is expected to be implemented in real-time at the maximum frame rate that cloud gaming services provide these days.

Margin Error

Our non-adaptive benchmark algorithm is pretty close to the optimal solution. In the non-adaptive benchmark algorithm, all information about textures is known in advance, and therefore the quality contribution of each texture can be calculated, the textures can be sorted based on their quality contribution, and thus the best solution can be applied to satisfy the total budget D. Clearly the difference of the non-adaptive benchmark algorithm with the optimal solution will be caused by the first texture which cannot be streamed at full resolution. This particular texture is partially rescaled, which causes the non-adaptive benchmark algorithm not to achieve the maximum capacity of the available budget. Based on the description of the non-adaptive benchmark algorithm, the process of finding the maximum resolution for $\tau_2, \tau_3, \ldots$ is repeated until some texture τ_ℓ cannot be streamed at full resolution within the remaining budget $D_{\ell-1}$. The algorithm then tries to find the maximum possible resolution that can be gained within $D_{\ell-1}$.

The optimum algorithm provides an upper bound solution for the non-adaptive benchmark algorithm. Due to the fact that the compression ratios are discrete, and not continuous (i.e. all scalings are based on a power of 4), the solution of the non-adaptive benchmark algorithm remains within a specific range from the optimum solution. This is exactly the cause of approximation or margin error. Assume s_{ℓ_1} is the current size of τ_ℓ (i.e. after considering the deviation), and s_{ℓ_2} is the best resolution of that which the algorithm achieves, so not to exceed the available budget $D_{\ell-1}$. Imagine the optimum solution for the best resolution of texture τ_ℓ is $s_{\ell'}$; then absolutely there is a larger resolution s_{ℓ_3} equal to or less than the maximum achievable resolution (i.e. the original resolution) where $\tau_{\ell_2}<\tau_{\ell'}<\tau_{\ell_3}$; then the algorithm brings an error of ($s_{\ell'}$ - s_{ℓ_2}), which is an upper bound on the error (i.e. the difference of the non-adaptive benchmark algorithm with the optimum solution) simply due to the reason that some of the unused capacity ($s_{\ell'}$ - s_{ℓ_2}) could be used by later textures. In section 4.6 we provide quantitative results for these margin errors by defining approximation error and approximation ratio.

4.5.2 Semi-online Adaptive Algorithm

This algorithm determines the maximum possible quality that can be achieved for a test sequence if the algorithm knows everything in advance except the deviations in energy consumption which it has to handle online. The goal is to study the effects of the deviations on quality.

Here we do the same calculations as for the Non-adaptive Benchmark Algorithm except that we substitute s_{τ_i} for s'_{τ_i} in the calculations because the Semi-online Adaptive Algorithm does not know the energy deviations in advance. This gives the same sequence of streamed textures $\tau_1, \tau_2, \tau_3, \ldots, \tau_{97}$ with the same sizes as the Non-adaptive Benchmark Algorithm with $\delta = 0$.

Next, we adaptively stream the textures in the correct order according to arrival time. To do this, we first compute a sequence of target values. Suppose that $\tau_1, \tau_2, \ldots, \tau_{97}$ is the list of textures to be streamed, but this time ordered by arrival time. We compute the sequence of target values $W_1, W_2, \ldots, W_{97}$ where $W_i = \sum_{j=1}^{i} s_{\tau_j}$ is the amount of the download budget that would be used after the first i textures in the list have been streamed, assuming that no energy deviations occur.

We then stream τ_1 using $s'_{\tau_1} = s_{\tau_1} \times (1 + \delta e_{\tau_1})$ of the download budget. Note that this takes into account the actual energy that is used including any deviation. We suppose that the user tells the server the amount of energy that was actually used so that the server can adapt. If $s'_{\tau_1} > W_1$ then we have used more energy than the target, so we have to adapt by reducing the size of next texture, which supposedly is the upcoming texture. If $s'_{\tau_1} < W_1$ then we increase the size of next texture. In general, we try to keep $\sum_{j=1}^{i} s'_{\tau_j}$ which we denote as W'_i, as close to the target W_i as possible. Based on comparison of W'_i (used budget after deviations) and W_i (the target value), we decide whether to reduce or increase the size of the next texture τ_{i+1}. The average quality and other statistics are computed in a similar way to the Non-adaptive Benchmark Algorithm using the size and energy values that were actually used when the textures were streamed by the Semi-online Adaptive Algorithm. Algorithm 1 describes our heuristic for the semi-online algorithm.

Regarding the time complexity of this algorithm, same as the previous algorithm, it sorts the list of all textures in $O(n \cdot lg(n))$ time supposing n is the number of all textures, iterates over the list of all textures in $O(n)$, and for each of the textures tries to decide whether to increase or decrease the resolution which is done in $O(c)$ in which c is constant. Thus similarly these cases make the total time complexity of the algorithm $O(n \cdot lg(n) + n \cdot c) = O(n \cdot lg(n))$. The memory complexity of the semi-online algorithm is also $O(n)$. For the implementation of this algorithm, the only overhead in addition to the non-adaptive benchmark algorithm is regarding not knowing the deviations in advance. The only extra workload the client-side is responsible for doing is comparing the $W'[i]$ with the target value $W[i]$, and notifying the server about the result using two extra bits. Since the connection has been already established, the comparison does not complicate the runtime overhead at the client-side.

4.5.3 Online Adaptive Algorithm

This algorithm only knows the sizes $s_{\tau_1}, s_{\tau_2}, \ldots, s_{\tau_{97}}$ and relative values $v_{\tau_1}, v_{\tau_2}, \ldots, v_{\tau_{97}}$ of the full-resolution textures $\tau_1, \tau_2, \tau_3, \ldots, \tau_{97}$, the game time T, the download budget D, and the maximum compression $R_{\max} = c^k$. Of course, it can also calculate S, S_1, S_2, S_3. It does not know the arrival and departure times so it cannot calculate $z_i = v_{\tau_i} \times s_{\tau_i} \times (d_{\tau_i} - a_{\tau_i})$, and it does not know the deviations in the energy consumption. It is not completely online because is knows the sizes and values of all of the textures in advance. The goal is to study the effects on quality of not knowing the arrival and departure times in advance.

Algorithm 1 Our defined heuristic

```
    ▷ %comment: Textures_scaled[ ] represents the list of textures after scaling based on
the benchmark algorithm, and Textures_original[ ] represents the list of original sizes of
textures before scaling %

Textures_scaled[ ]=$\mathcal{T}$ [1 .. numberOfTextures]
Textures_original[ ]=$\mathcal{T}_{original}$ [1 .. numberOfTextures]
W[i]: Target values: Total size of $i$ transmitted textures
W'[i]: $\sum_{j=1}^{i} s'_{\tau_j}$ (Total sizes of $i$ transmitted textures, after deviations)
$R_{max}$: Maximum reduction in resolution that is acceptable
$W'[0] \leftarrow 0$

for $i = 1$ to $Textures[\ ].length$ do
        ▷ %comment: We deal with X, the difference of target value W[i], and W'[i]%
    $X \leftarrow W'[i] - W[i]$
    next_original ← Textures_original[i]
    next ← Textures_scaled[i]

▷ %comment: If W'[i] is bigger that W[i], then make the upcoming texture (i) smaller%
    if (X > 0) then
            ▷ %comment: If not at the minimum size (i.e. previously scaled by $R_{max}$)%
        if (next != next_original ÷ power(4, $R_{max}$)) then
                ▷ %comment: decrease the size of upcoming texture by factor of 4%
            $next \leftarrow next \div 4$
        end if
        break
    end if

▷ %comment: If W'[i] is smaller that W[i], then make the upcoming texture (i) larger%
    if (X < 0) then
                                        ▷ %comment: If not at the original size %
        if (next != next_original) then
                ▷ %comment: increase the size of upcoming texture by factor of 4%
            $next \leftarrow next \times 4$
        end if
    end if
    $W'[i] \leftarrow W'[i] + next \times (1 + deviation(next))$

end for
```

Our approach here is to adapt the idea of target values from the Semi-online Adaptive Algorithm. If we have a download budget D spread over the game time T, then the download budget per unit of time is D/T. Suppose that $\tau_1, \tau_2, \tau_3, \ldots, \tau_{97}$ is the list of textures ordered by arrival time. As with the Semi-online Adaptive Algorithm, we suppose that the user tells the server the amount of energy that was actually used after it receives each texture so that the server can adapt. If texture τ_i arrives at time t, then we compare the download budget that we have already used for the first $i-1$ textures with the target usage tD/T and choose the compression ratio for τ_i using the same heuristic as we used for the Semi-online Adaptive Algorithm. Due to the lack of not knowing both the arrival and departure times, and deviations, there is no sorting part being done in advance, which makes the time complexity of the online algorithm to be $O(n \cdot c) = O(n)$ in which similar to the semi-online algorithm c is a constant (refer to Algorithm 1). The memory complexity of the online algorithm is also $O(n)$ required at the server-side. Note that these calculations involving download budgets also account for observed energy deviations, similar to the previous two algorithms. With the same reasons as in the semi-online algorithm, the only extra workload the client-side is responsible for doing is comparing the W_i' with the target value W_i, which does not complicate the runtime overhead and thus the framerate at the client-side.

4.6 Results and Analysis

After performing the random generation and getting the proper values, we got ten test sequences. Each sequence consists of a randomly permuted list of the 97 textures, and associated with each texture τ_i in the list are a relative value v_{τ_i}, an arrival time a_{τ_i}, a departure time d_{τ_i}, and an energy deviation value e_{τ_i}. Each experiment consists of running these ten test sequences and collecting statistics, the average and the standard deviation as said before. We also run four experiments for each algorithm, one with each of the four different values of δ. For our simulation we used the same machine that we used for our initial study, an Intel 3GHz dual core machine running Java 1.7.0 standard edition. We ran the algorithms with D set to be different percentages of S (total size of all textures). Similar to our previous study in Chapter 3, we set D to $0.1S$, $0.2S$, $0.3S$, ..., $1.0S$ corresponding to 10%, 20%, 30%, ..., 100% of the total size of textures. We chose two different values for R_{max}.

Figure 4.1 to 4.4 show our experimental results for total quality and quality per unit of energy measured for two different values of R_{max} (16 and 64), each associated with one of the four values of δ (0, 5, 10, and 15). In order to reduce the scale of the diagrams and show smaller values on the diagrams, we divided the total qualities by a factor of 1,000,000. It should be noted that in our simulations, as concluded in our initial study, we assume the consumed *energy* is in a linear relation with *size*, therefore they can easily be converted to each other. We compute the total quality and the quality per unit of energy according to Equation 4.2 and Equation 4.3 respectively (for each texture τ_i in the texture list $\mathcal{T}$, as introduced earlier in the definitions, v_{τ_i}, s_{τ_i}, d_{τ_i}, and a_{τ_i} stand for the value, size, the departure time, and the arrival time of texture τ_i). Clearly $(d_{\tau_i} - a_{\tau_i})$ is the period of time in which τ_i is visible during gameplay.

$$\text{Total Quality: } \sum_{\tau_i \in T} (v_{\tau_i} \times (d_{\tau_i} - a_{\tau_i}) \times s_{\tau_i}) \tag{4.2}$$

$$\text{Quality per Unit of Energy: } \frac{\sum_{\tau_i \in \mathcal{T}} (v_{\tau_i} \times (d_{\tau_i} - a_{\tau_i}) \times s_{\tau_i})}{\sum_{\tau_i \in \mathcal{T}} s_{\tau_i}} \tag{4.3}$$

In the following we run a discussion about the diagrams.

As can be seen in all graphs, the total quality achieved by all the proposed three algorithms increases as the available budget increases. Clearly that's because of the contribution of size in our defined factor of quality that a specific texture τ_i contributes which is derived in equation 4.2. As the available budget increases, there is more space to be filled by textures, so the total size, and thus, the corresponding total quality will also increase.

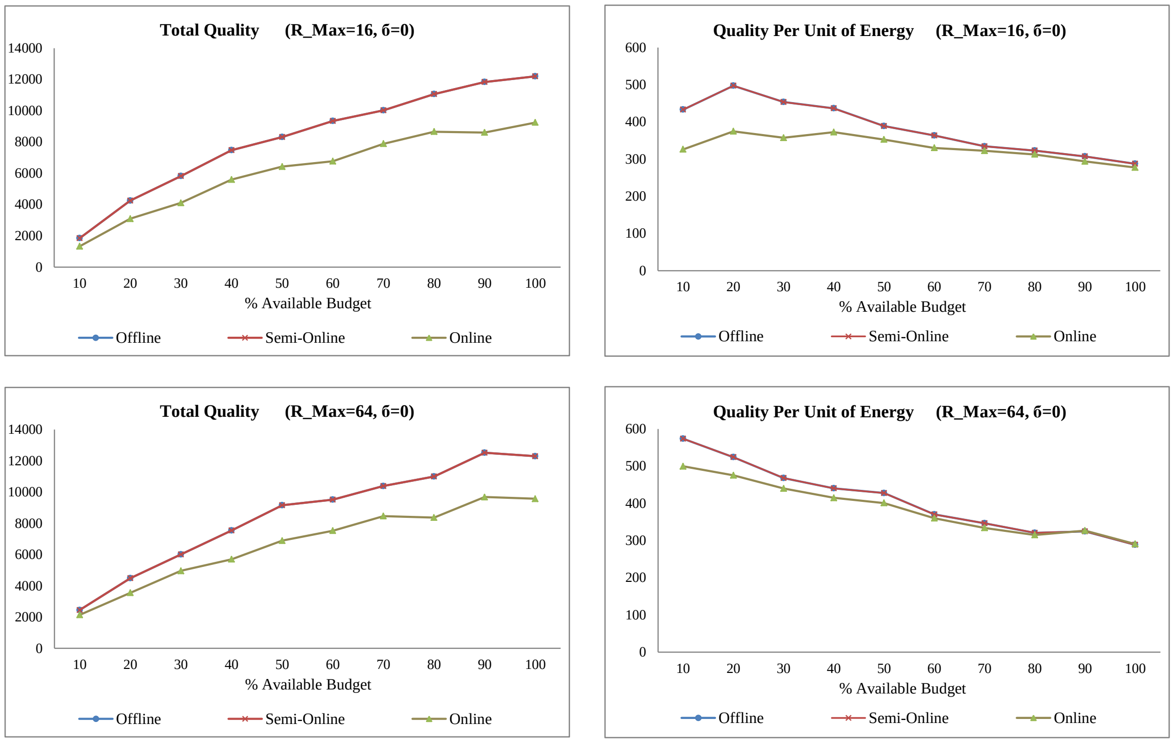


Figure 4.1: Total quality and quality per unit of energy measured for four situations when $\delta = 0$. (Top-Left) Total quality, R_{max}=16. (Bottom-Left) Total quality, R_{max}=64. (Top-Right) Quality per unit of energy, R_{max}=16. (Bottom-Right) Quality per unit of energy, R_{max}=64.

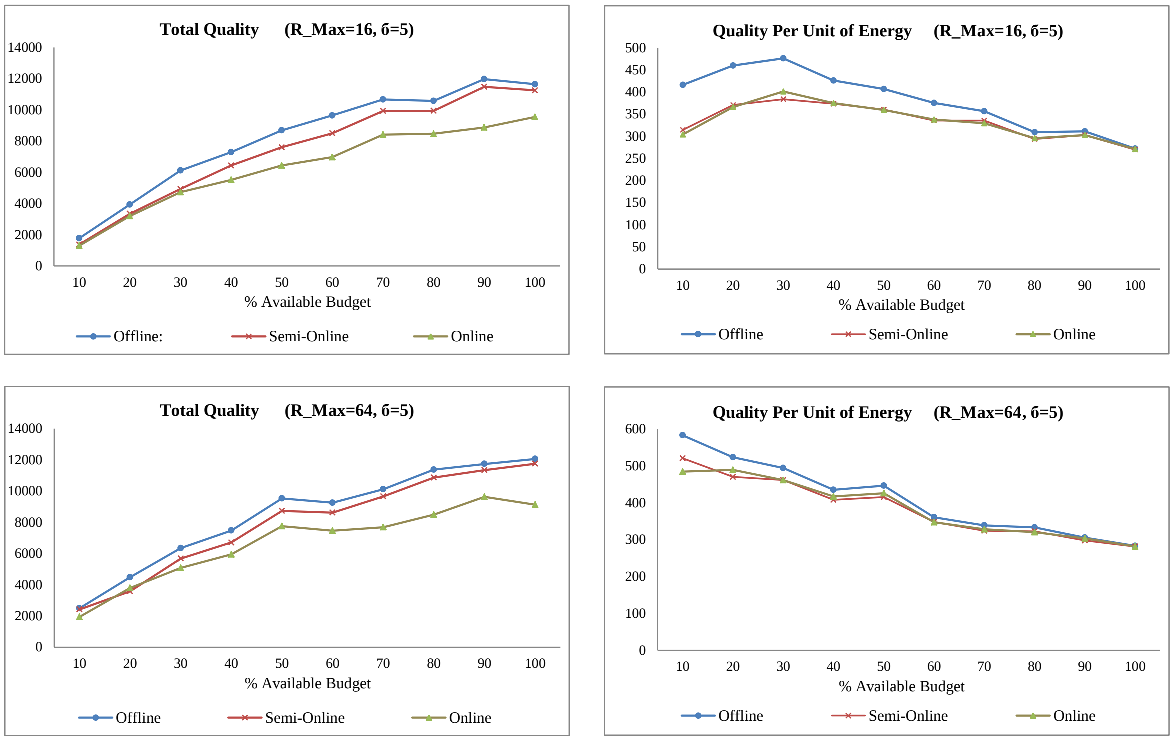
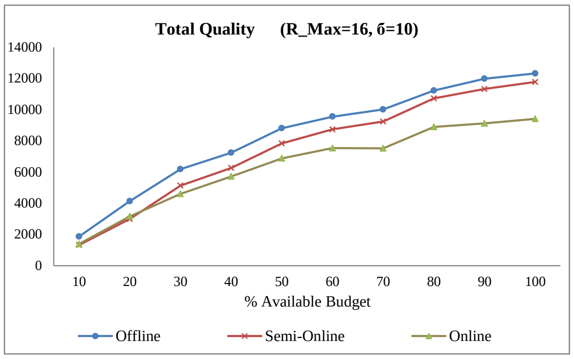
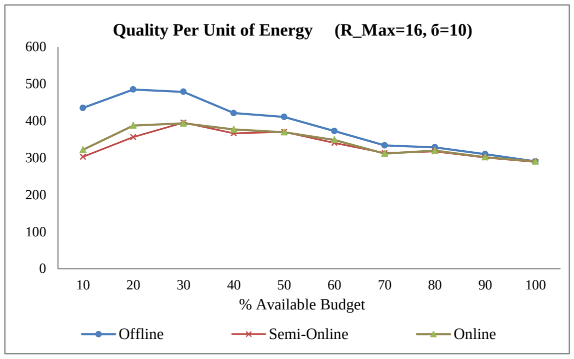
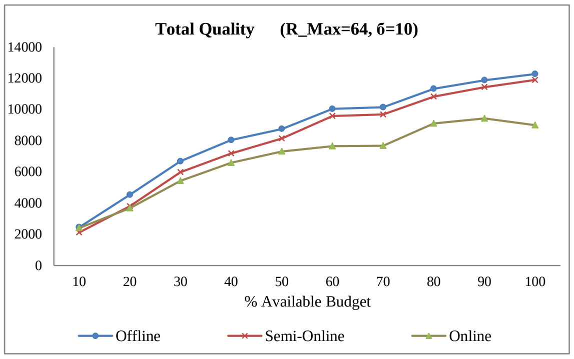


Figure 4.2: Total quality and quality per unit of energy measured for four situations when $\delta = 5$. (Top-Left) Total quality, R_{max}=16. (Bottom-Left) Total quality, R_{max}=64. (Top-Right) Quality per unit of energy, R_{max}=16. (Bottom-Right) Quality per unit of energy, R_{max}=64.







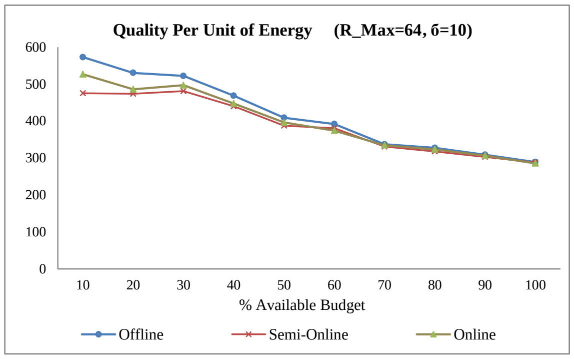


Figure 4.3: Total quality and quality per unit of energy measured for four situations when $\delta = 10$. (Top-Left) Total quality, R_{max}=16. (Bottom-Left) Total quality, R_{max}=64. (Top-Right) Quality per unit of energy, R_{max}=16. (Bottom-Right) Quality per unit of energy, R_{max}=64.

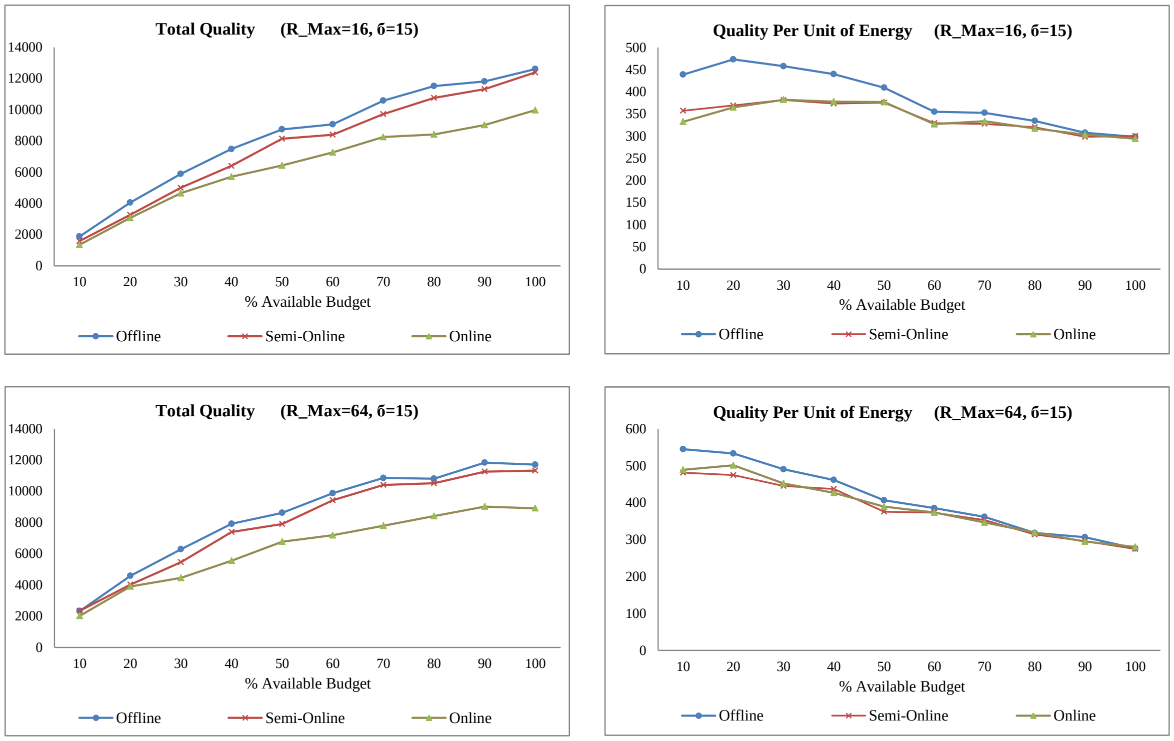


Figure 4.4: Total quality and quality per unit of energy measured for four situations when $\delta = 15$. (Top-Left) Total quality, R_{max}=16. (Bottom-Left) Total quality, R_{max}=64. (Top-Right) Quality per unit of energy, R_{max}=16. (Bottom-Right) Quality per unit of energy, R_{max}=64.

As argued before, we also notice here that the non-adaptive benchmark algorithm is closer to the optimum solution, and performs better and brings more total quality compared to the other two algorithms. It is more efficient than the semi-online algorithm since it knows all the information about the textures in advance, including the deviations and the arrival/departure times. Obviously to have a knowledge of the deviations of all textures a priori, it can provide a better adaptation for the best use of the available budget. However in the semi-online algorithm, the algorithm has to adapt itself with the deviations in an online way.

Similarly, the online algorithm not only does not know the deviations, but also it does not have any information about the arrival and departure time of the textures, so it has to adapt as they arrive. There is no information on the order of arrival time of the upcoming textures. We compute the share of budget over time, and use them as our target values, which of course does not represent the exact target values as they are in the semi-online version. This difference brings an error, which causes the online algorithm to bring less total size, and thus less total quality compared to the semi-online algorithm. Additionally, as available budget increases, there will be more growth in this error, which causes the online algorithm to get behind the other two algorithms. That's why we notice a bigger gap between the online and the semi-online algorithms as the available budget increases. The *maximum* gaps between the total quality derived by the online and the semi-online algorithms for pairs of ($R_{max} = 16$, $R_{max} = 64$) are (27.2%, 23.8%) for $\delta = 0$, (22.6%, 22.2%) for $\delta = 5$, (20.0%, 24.4%) for $\delta = 10$, and (20.2%, 25.0%) for $\delta = 15$. It should be noted that these percentage values are showing the *maximum*, not the *average*. The aim of calculating the maximum gaps was to only show the maximum possible difference that the three proposed algorithms might lead to.

In the same way, we also computed the average *competitive ratios* and *competitive errors*, which places the online algorithm in competition with the non-adaptive benchmark algorithm that receives more information. In other words, assume Q_{semi} and Q_{bench} show the total qualities achieved by the semi-online and the non-adaptive benchmark algorithms. Then the competitive ratio is simply computed using the ratio $\frac{Q_{semi}-Q_{bench}}{Q_{bench}}$. The difference of the non-adaptive benchmark algorithm and the optimum solution itself (approximation error) was also being calculated.

In average, the differences of semi-online algorithm with the benchmark algorithm (relative competitive error) for pairs of ($R_{max} = 16$, $R_{max} = 64$) are (11.3%, 7.4%) for $\delta = 5$,

δ	$R_{max} = 16$									
$\delta = 0$	0.26	0.08	0.09	0.04	0.06	0.03	0.03	0.04	0.03	0.85
$\delta = 5$	0.23	0.10	0.04	0.08	0.23	0.07	0.12	0.18	0.16	0.07
$\delta = 10$	0.06	0.42	0.54	0.39	0.08	0.20	0.13	0.24	0.24	0.98
$\delta = 15$	0.03	0.14	0.14	0.68	0.29	0.69	0.10	0.61	0.31	0.12

Table 4.2: Approximation errors measured for the four different values of δ for $R_{max} = 16$.

δ	$R_{max} = 64$									
$\delta = 0$	0.04	0.02	0.02	0.02	0.01	0.01	0.01	0.01	0.02	0.52
$\delta = 5$	0.11	0.10	0.02	0.37	0.23	0.04	0.28	0.27	0.25	0.52
$\delta = 10$	0.14	0.21	0.39	0.15	0.02	0.30	0.35	0.96	0.27	0.86
$\delta = 15$	0.28	0.33	0.24	0.21	0.10	0.23	0.02	0.82	0.19	0.98

Table 4.3: Approximation errors measured for the four different values of δ for $R_{max} = 64$.

(12.8%, 7.8%) for $\delta = 10$, and (9.9%, 5.8%) for $\delta = 15$. Obviously the pairs of competitive ratios are (88.7%, 92.6%), (87.3%, 92.2%), (90.1%, 94.2%) respectively. As we will talk later in this section, clearly there is no difference when $\delta = 0$ since both algorithms perform identically. Also in average, the differences of online algorithm with the benchmark algorithm for pairs of ($R_{max} = 16$, $R_{max} = 64$) are (25.4%, 20.8%) for $\delta = 0$, (23.0%, 20.7%) for $\delta = 5$, (23.2%, 18.9%) for $\delta = 10$, and (23.7%, 23.4%) for $\delta = 15$. This results in the competitive ratios of (74.6%, 79.2%), (77.0%, 79.3%), (76.8%, 81.1%), (76.3%, 76.6%). However these percentages do not follow a specific additive or subtractive pattern, but it can obviously be seen that with $R_{max} = 16$, the differences are larger. The possible reason is that bigger values of R_{max} causes smaller reserved sizes for textures compared to their original sizes. That means there are more possible choices for the textures to be increased in size ($log_4 R_{max}$ brings a larger value), and therefore the rescaling can more precisely be done.

Table 4.2 and table 4.3 show the differences of the non-adaptive benchmark algorithm with the optimum solution (which is a margin error or approximation error, representing a relative error measure) calculated for both $R_{max} = 16$ and $R_{max} = 64$, measured for different δs. Based on these results, in average, the approximation errors for pairs of ($R_{max} = 16$, $R_{max} = 64$) are (0.15%, 0.06%) for $\delta = 0$, (0.10%, 0.10%) for $\delta = 5$, (0.04%, 0.07%) for $\delta =$

Effect of:	$(R_{max} = 16)$	$(R_{max} = 64)$
Not knowing times (Benchmark & Online)	24.42	22.17
Not knowing deviations (Benchmark & Semi-Online)	11.67	7.33
Not knowing times and deviations (Semi-Online & Online)	23.63	21.84
Interaction of factors	12.46	7.66

Table 4.4: Average effect of not knowing each, or both of the two factors (the arrival and departure times, and the deviations) for $\delta = 0$ and $\delta = 10$, measured for both $R_{max} = 16$ and $R_{max} = 64$. The values are in percentage (ranging from 0 to 100), and relative to the non-adaptive benchmark algorithm.

10, and (0.25%, 0.27%) for $\delta = 15$. It results to the *approximation ratios* of (0.9985,0.9994), (0.9990,0.9990), (0.9996,0.9993), (0.9975,0.9973) for the previous pairs respectively. As can seen in the results, the approximation errors are always within 1% of the optimum solution, which clearly shows and confirms how close the non-adaptive benchmark algorithm is to the optimum solution. Also it should be noted that unlike the two other algorithms, the non-adaptive benchmark algorithm does not exceed D, the available budget (compare the diagram of the non-adaptive benchmark algorithm with the diagrams of semi-online and online algorithms). The approximation ratio is the ratio between the result obtained by the near-optimum benchmark algorithm and the optimal version. These percentages do not follow a specific pattern, so possibly the different values of δ do not bring much difference in the total quality. This is mainly because the algorithms do not use δ as a factor for decision. The deviations are applied to the sizes, and then the algorithms make decisions based on the sizes.

As another part of the analysis of the three proposed algorithms, we should notice that there are two factors affecting the performance of the online algorithm: deviations, and arrival and departure times. We can evaluate the effect of each of these two factors simply by separating. The effect of arrival and departure times can be determined by setting δ=0 (so there are no deviations) and comparing the online algorithm with the non-adaptive benchmark algorithm. In the same way, we can also determine the effect of deviations by comparing the semi-online algorithm and the non-adaptive benchmark algorithm. As a result, we can determine the effect of the interaction of these two factors by comparing the online algorithm with the non-adaptive benchmark algorithm for various values of δ and subtracting the individual effects for deviations and for arrival and departure times, which then only leaves the interaction of the factors.

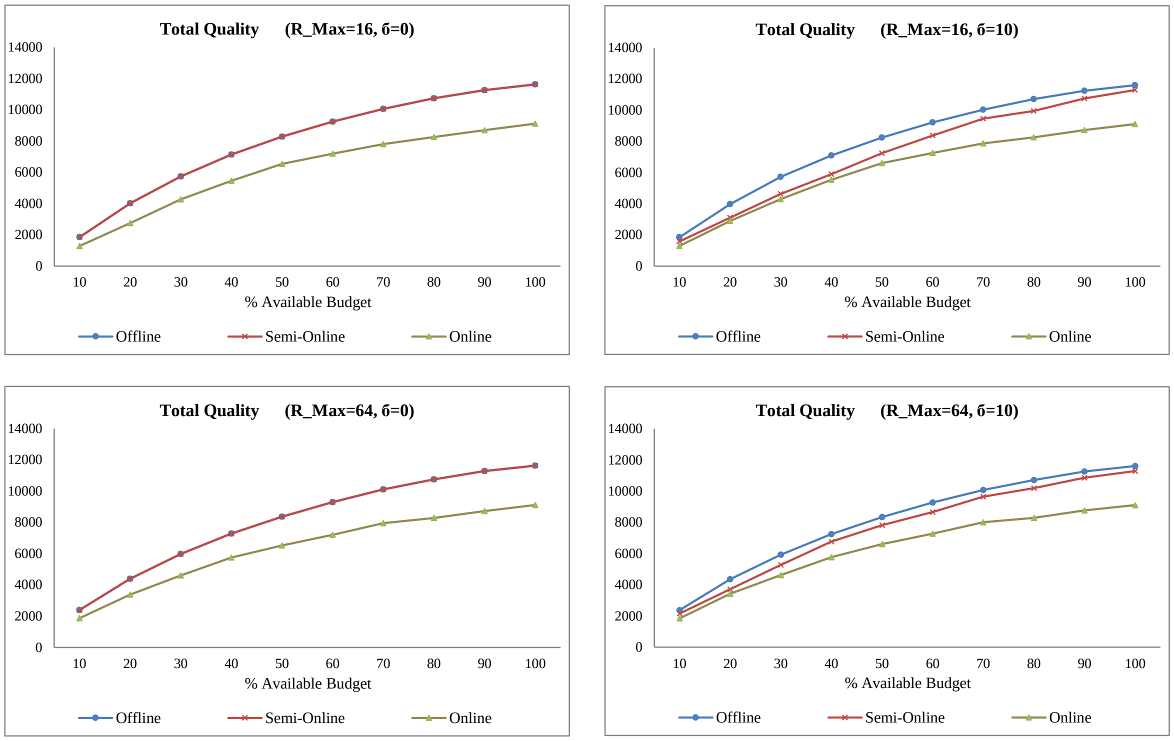


Figure 4.5: Total quality measured using constant seeds for $\delta = 0$ and $\delta = 10$. (Top-Left) Total quality, $\delta = 0$ and R_{max}=16. (Bottom-Left) Total quality, $\delta = 0$ and R_{max}=64. (Top-Right) Total quality, $\delta = 10$ and R_{max}=16. (Bottom-Right) Total quality, $\delta = 10$ and R_{max}=64.

In order to determine the effect of each of these two factors, we have run our experiments for δ=0 and a non-zero value of δ (we have chosen δ=10 as an instance) for constant values of seeds. As argued previously, using a constant seed for random generation always results to the same random number. This way we can make sure the experimental setup always remains identical for all experiments, thus the experimental results can be comparable. Similarly we repeated each experiments for 10 times, and calculated the average so to be able to obtain a concrete conclusion. Figure 4.5 show the resulting diagrams of total qualities, for two values of δ (δ=0 and δ=10), and tested for both R_{max}=16 and R_{max}=64. Table 4.4 also shows the average effect of not knowing each of the above mentioned factors (the arrival and departure times, and the deviations), the effect of not knowing both of them, and finally the effect of interaction of the two factors. The values are resulted from the total qualities achieved by our three proposed algorithms for constant seeds measured for both R_{max}=16 and R_{max}=64. Note that the percentages are relative to the non-adaptive benchmark algorithm. Based on the interesting results coming from Table 4.4, we can clearly see that the effect of not knowing the arrival and departure times are relatively more compared to the effect of not knowing the deviations.

For the quality per unit of energy, we can show interesting conclusion. To achieve the quality per unit of energy, we assumed energy is in a linear fashion with the size, so the factor of energy is considered linearly related to the numerator which is a factor of size, which leads to Equation 4.3. This is the total quality per unit of energy. Unlike the total quality, we can see that the online algorithm is well tracking the semi-online algorithm, noting that in some cases it even brings larger values. This shows the effectiveness of the proposed online algorithm, confirming the fact that it can be used and deployed in our framework to stream textures in online mobile games.

Another interesting fact regarding the trend of the quality per unit of energy diagrams is that in all diagrams with $R_{max} = 16$, we are witnessing that the slope gets smaller as the available budget increases compared to the slope seen in the diagrams associated with $R_{max} = 64$. That is mainly due to the reason that with $R_{max} = 16$, averagely the amount of texture compression is not as much compared to $R_{max} = 64$. In other words, with $R_{max} = 16$ it makes less difference for the compression process before and afterwards. Also for the same reason the gap between the non-adaptive benchmark version and the other two algorithms gets bigger with the smaller values of R_{max}, which therefore causes the semi-online and the online algorithms to work better with larger values of R_{max}. In addition, in should be noted that all the three algorithms bring more quality per unit of energy with smaller available budgets when $R_{max} = 64$. However, with $R_{max} = 16$ the later statement is not always true. As can be seen in the diagrams, the reduction in the quality per unit of energy mostly starts when the available budget is around 20%. Clearly the three proposed algorithms can not do much in reduction $R_{max} = 16$ with small portions of budget, something almost around 10%. Therefore with $R_{max} = 16$, the largest quality per unit of energy is mostly gained in the interval of 20% to 30% of the available budget. Although it is interesting to see that for smaller portions of available budget, larger quality per unit of energy can be achieved, which is actually the main aim of our research.

Another point which is worth mentioning is that for $\delta = 0$, the non-adaptive benchmark and the semi-online algorithms perform identically since there is no deviation in the streamed textures, and thus no need for the semi-online algorithm to adapt. This can easily be concluded by replacing δ with value of 0 according to Algorithm 1 described in the semi-online algorithm. In other words, for situations in which there is a small energy deviation during the process of texture streaming (δ is close to 0), we can assure that our semi-online algorithm will perform fairly similar to the non-adaptive benchmark algorithm, and thus, similar to the optimum solution.

Figure 4.6 to 4.17 show different experimental results for total quality and quality per unit of energy with four different values of δ ($\delta = 0$, 5, 10, and 15), measured for C_1 (the less-important textures, which are the textures with value 1, shown in Figures 4.6, 4.9, 4.12, and 4.15), C_2 (the mid-important class which are the textures with value 2, shown in Figures 4.7, 4.10, 4.13, and 4.16), and C_3 (the important class or the textures with value 3, as shown in Figures 4.8, 4.11, 4.14, and 4.17). We used the same classification naming as in our previous study. The aim of these particular measurements was to study how the three proposed algorithms affect the prioritization of the textures. The results bring some interesting conclusions. Firstly, regarding the diagrams showing the total quality, as we go ahead from C_1 to C_3 (from least important to the most important class of textures), the gaps between the three algorithms increase. In other words, the diagrams for C_3 show larger differences in the three algorithms compared to C_1. This is mainly due to the value associated with each texture. Rescaling a texture with larger values simply has more effects on the quality contribution of this particular texture (refer to equation 4.2) is more compared to a situation in which a texture with smaller value is rescaled. The possibility of modification and rescaling textures in the online algorithm is higher than the semi-online, and that is more than the non-adaptive benchmark algorithm which provides a close-to-optimal solution. For the same reason, the diagrams in C_3 show higher values compared to C_2 and C_1.

It is also worth mentioning that mostly for the total qualities, as the δ increases, the diagrams generally get more saw-like. That's mainly due to the fact that with larger values of δ, the deviations in energy are more, and thus the possibility of texture scaling (i.e. increments and decrements) gets higher, which therefore, more affects the resulted corresponding quality.

Regarding the quality per unit of energy, it is interesting to see that the corresponding diagrams do not simply follow a specific pattern. Referring to Equation 4.3, this is mainly due to the factor of energy on the denominator (which is actually size) and the factor of size on the numerator. This leaves two factor with type of $(d - a) \times v$ (period of visibility, and values). However, this is not what our algorithms are optimizing. For generality of our conclusion, if we assume the period of visibility $(d - a)$ to be constant for all textures, the diagrams accounting for C_3 show higher values compared to C_2 and C_1 because of larger relative values (for instance compare 4.14 with 4.13 and 4.12 for $\delta = 10$; note that the scales on the Y-axis of quality per unit of energy diagrams are not identical). This is actually the main cause of the remaining factor of v in $(d - a) \times v$.

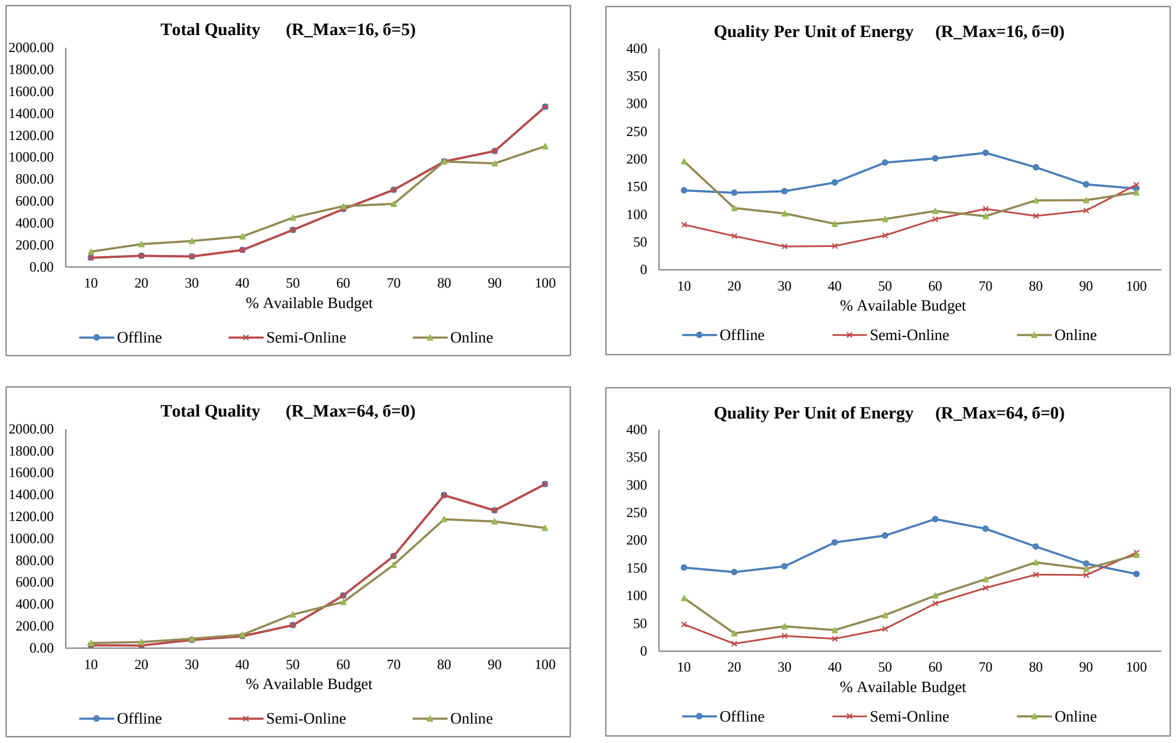


Figure 4.6: Total quality and quality per unit of energy of C_1 class of textures (textures with value 1) measured for $\delta = 0$. (Top-Left) Total quality, R_{max}=16. (Bottom-Left) Total quality, R_{max}=64. (Top-Right) Quality per unit of energy, R_{max}=16. (Bottom-Right) Quality per unit of energy, R_{max}=64.

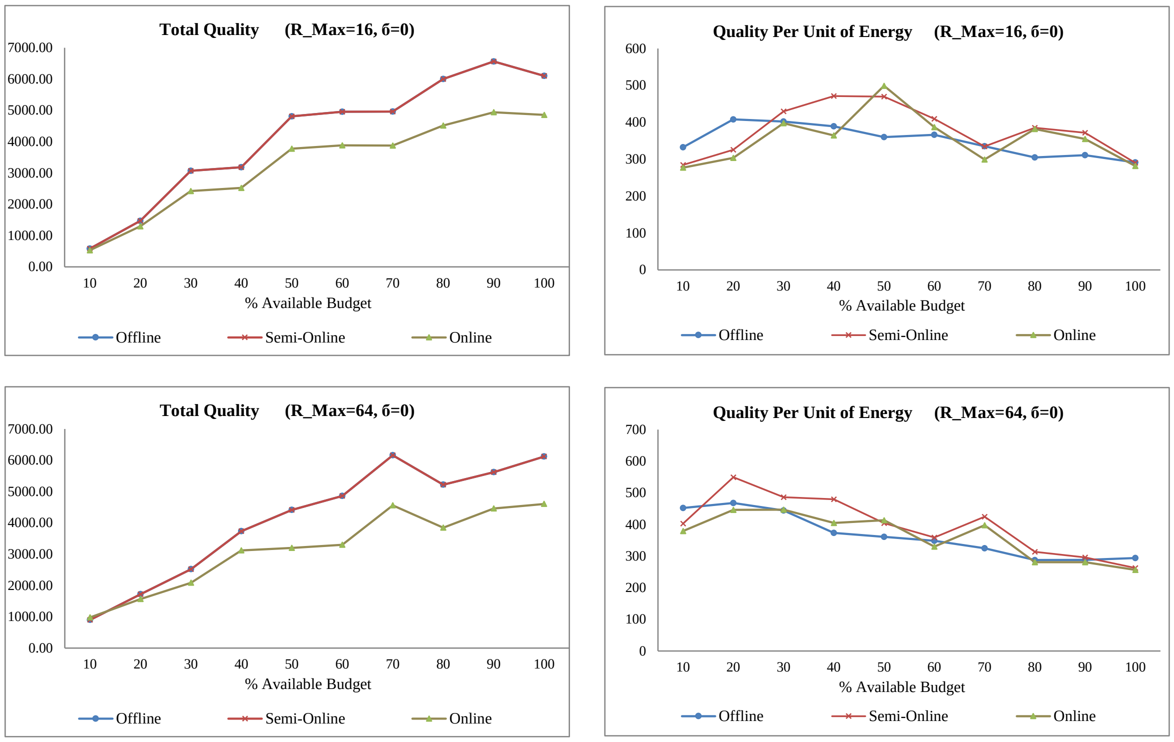


Figure 4.7: Total quality and quality per unit of energy of C_2 class of textures (textures with value 2) measured for $\delta = 0$. (Top-Left) Total quality, R_{max}=16. (Bottom-Left) Total quality, R_{max}=64. (Top-Right) Quality per unit of energy, R_{max}=16. (Bottom-Right) Quality per unit of energy, R_{max}=64.

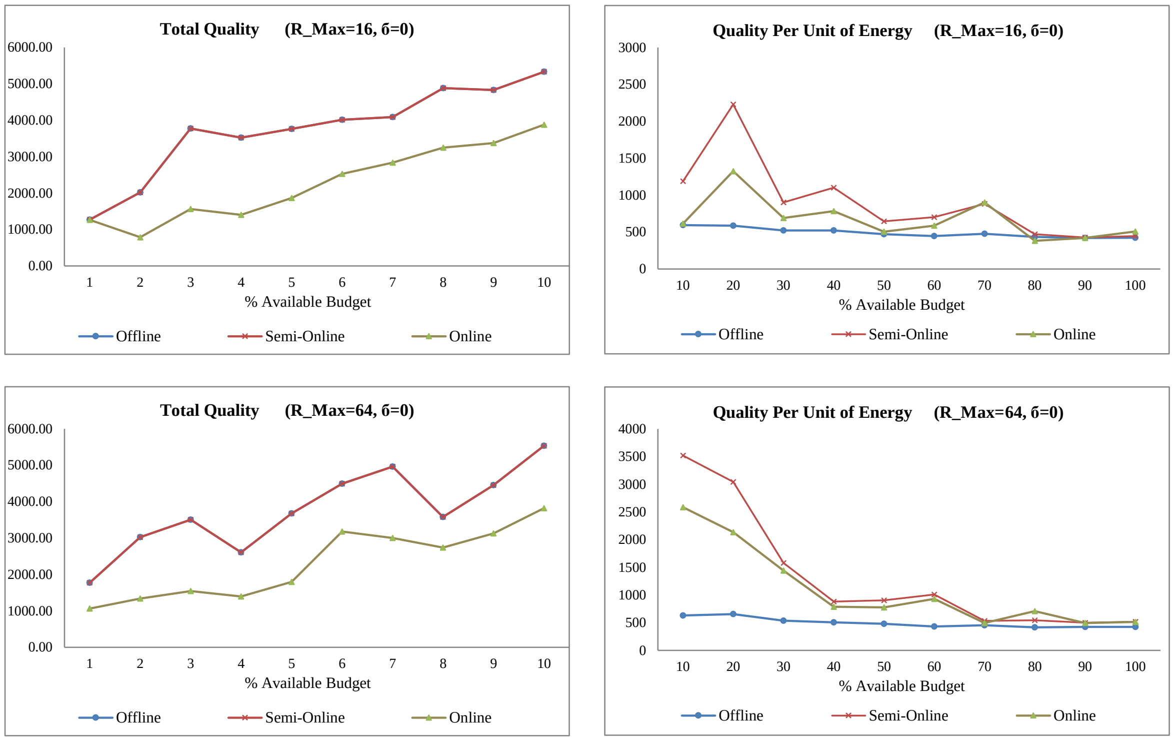


Figure 4.8: Total quality and quality per unit of energy of C_3 class of textures (textures with value 3) measured for $\delta = 0$. (Top-Left) Total quality, R_{max}=16. (Bottom-Left) Total quality, R_{max}=64. (Top-Right) Quality per unit of energy, R_{max}=16. (Bottom-Right) Quality per unit of energy, R_{max}=64.

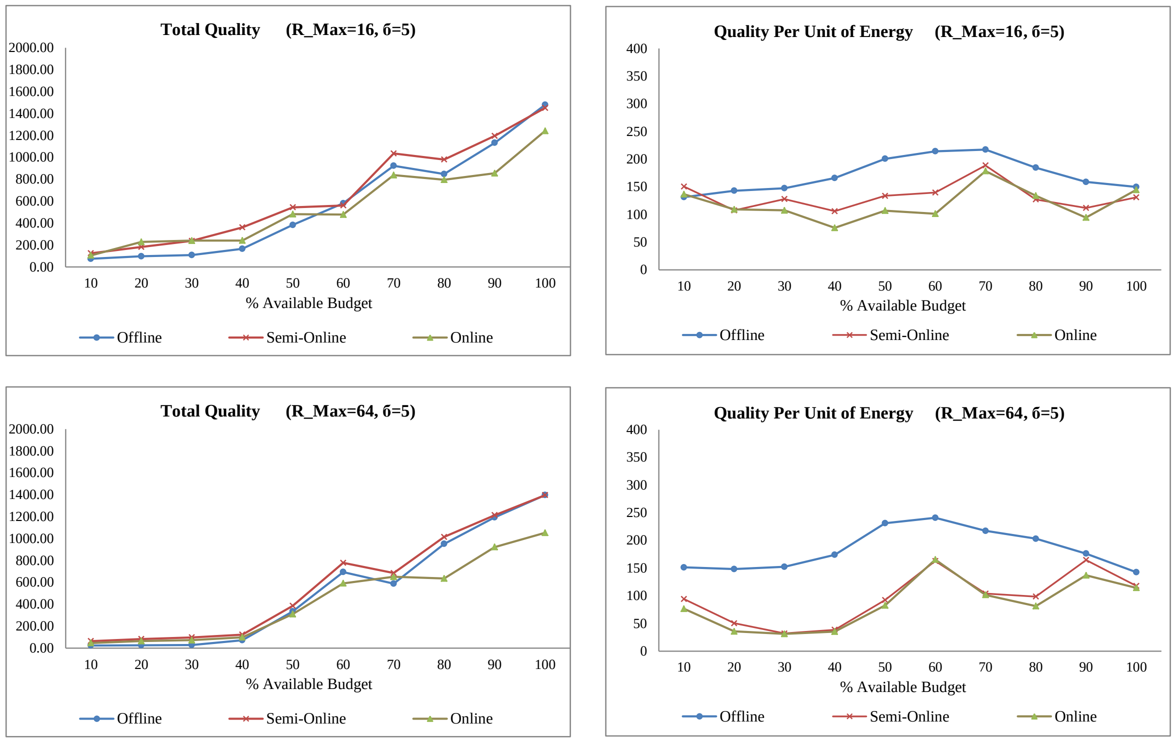


Figure 4.9: Total quality and quality per unit of energy of C_1 class of textures (textures with value 1) measured for $\delta = 5$. (Top-Left) Total quality, R_{max}=16. (Bottom-Left) Total quality, R_{max}=64. (Top-Right) Quality per unit of energy, R_{max}=16. (Bottom-Right) Quality per unit of energy, R_{max}=64.

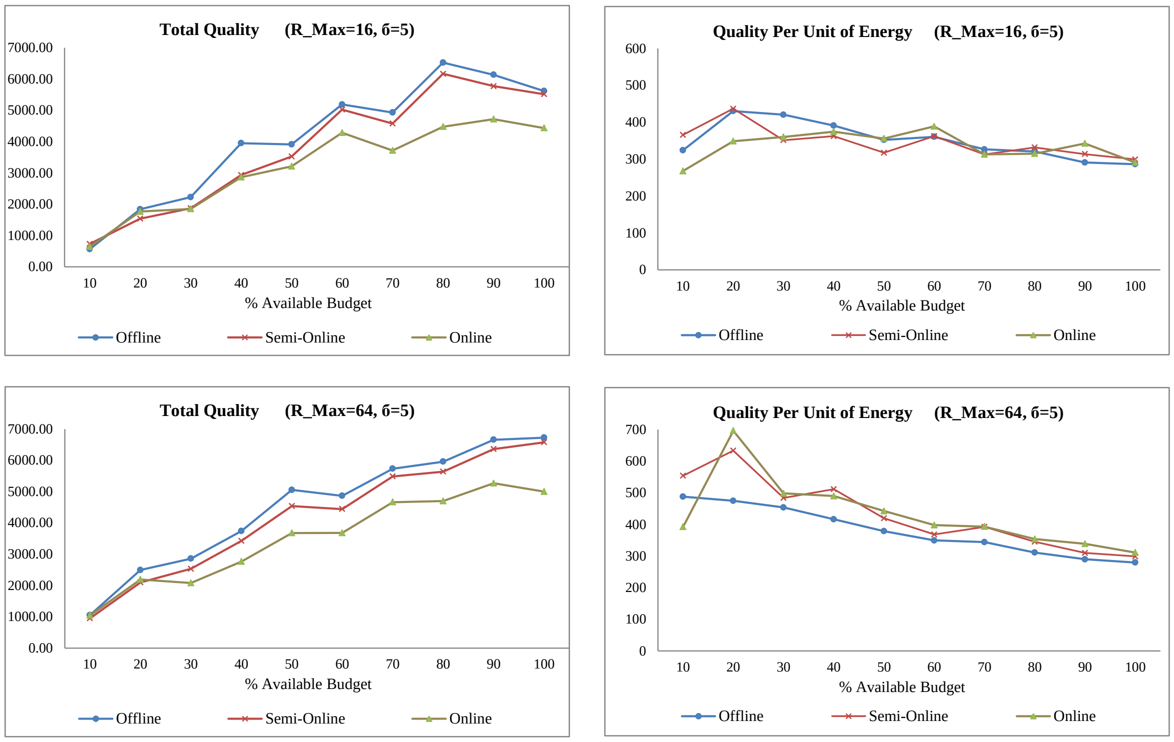


Figure 4.10: Total quality and quality per unit of energy of C_2 class of textures (textures with value 2) measured for $\delta = 5$. (Top-Left) Total quality, R_{max}=16. (Bottom-Left) Total quality, R_{max}=64. (Top-Right) Quality per unit of energy, R_{max}=16. (Bottom-Right) Quality per unit of energy, R_{max}=64.

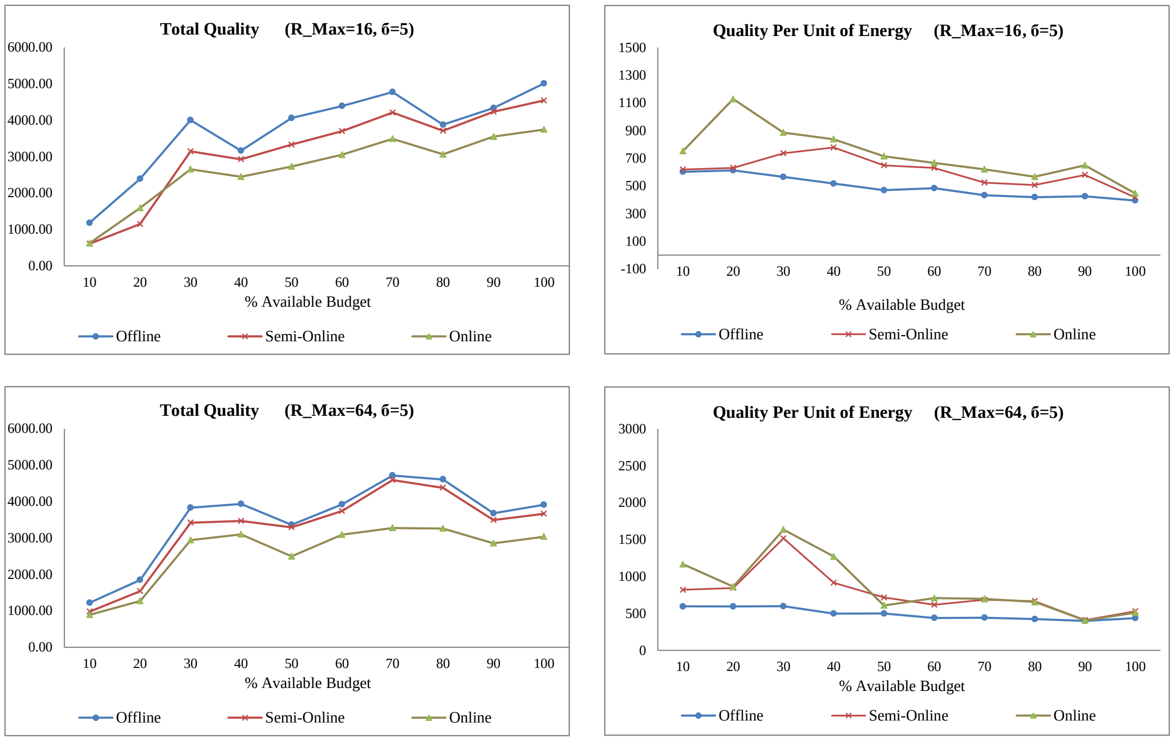


Figure 4.11: Total quality and quality per unit of energy of C_3 class of textures (textures with value 3) measured for $\delta = 5$. (Top-Left) Total quality, R_{max}=16. (Bottom-Left) Total quality, R_{max}=64. (Top-Right) Quality per unit of energy, R_{max}=16. (Bottom-Right) Quality per unit of energy, R_{max}=64.

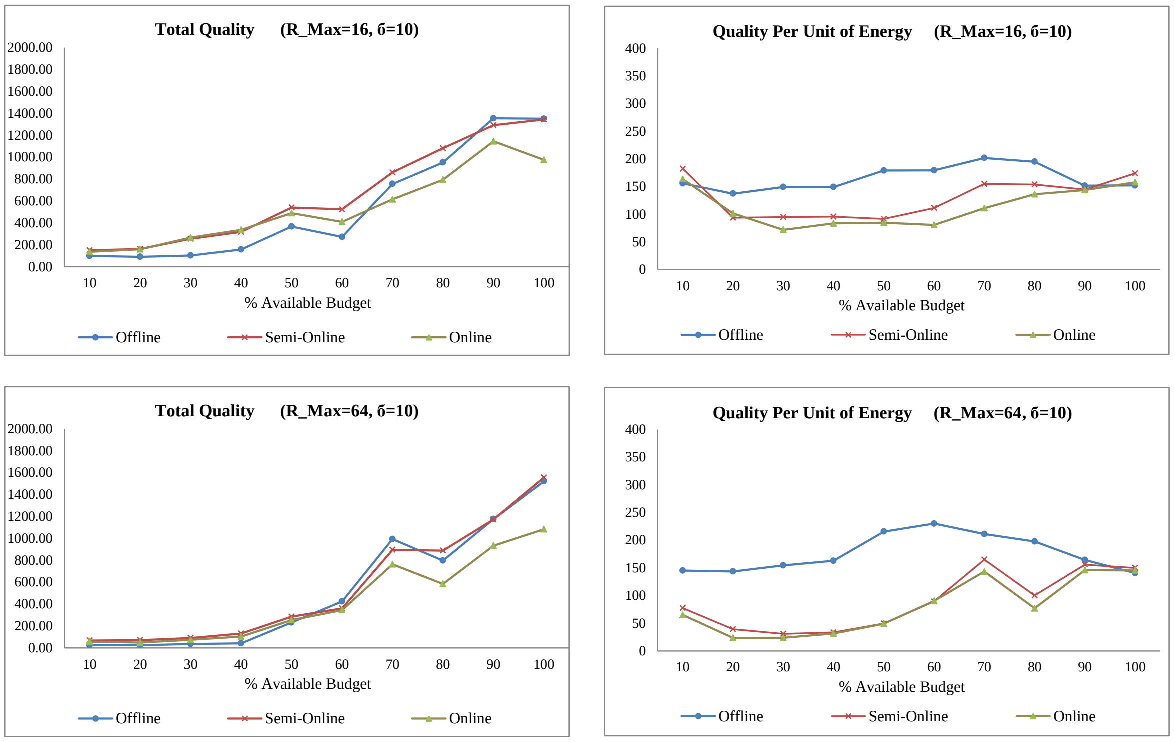


Figure 4.12: Total quality and quality per unit of energy of C_1 class of textures (textures with value 1) measured for $\delta = 10$. (Top-Left) Total quality, R_{max}=16. (Bottom-Left) Total quality, R_{max}=64. (Top-Right) Quality per unit of energy, R_{max}=16. (Bottom-Right) Quality per unit of energy, R_{max}=64.

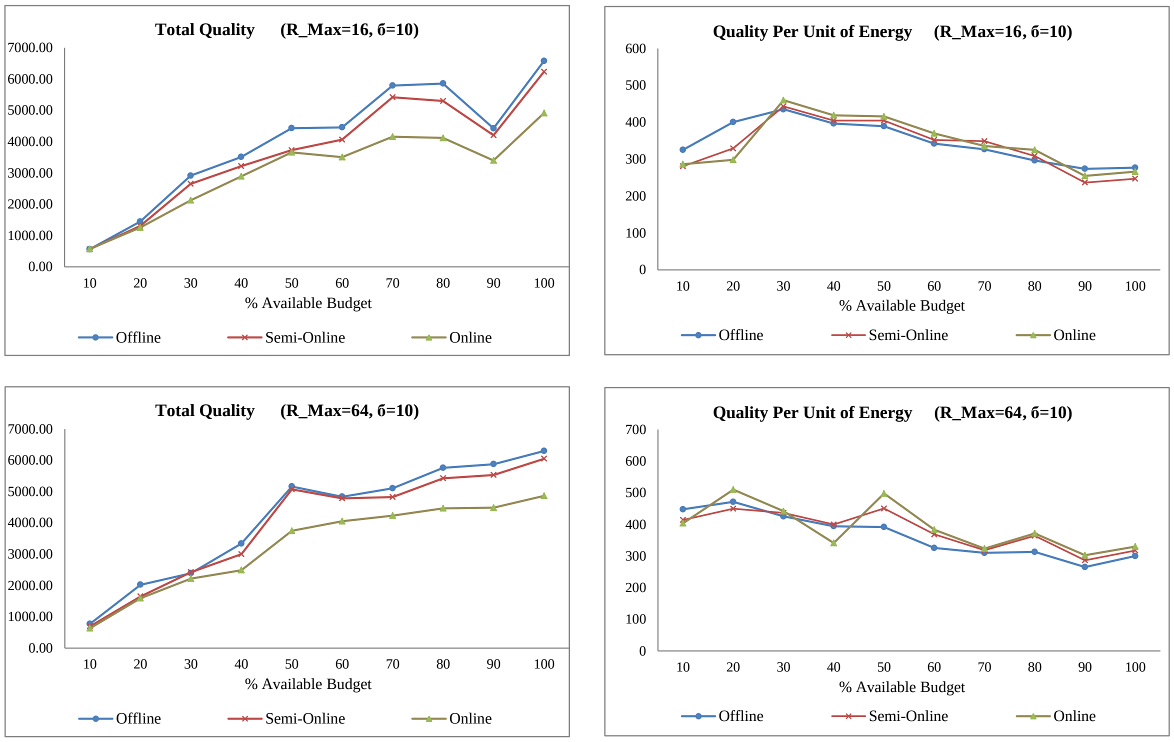


Figure 4.13: Total quality and quality per unit of energy of C_2 class of textures (textures with value 2) measured for $\delta = 10$. (Top-Left) Total quality, R_{max}=16. (Bottom-Left) Total quality, R_{max}=64. (Top-Right) Quality per unit of energy, R_{max}=16. (Bottom-Right) Quality per unit of energy, R_{max}=64.

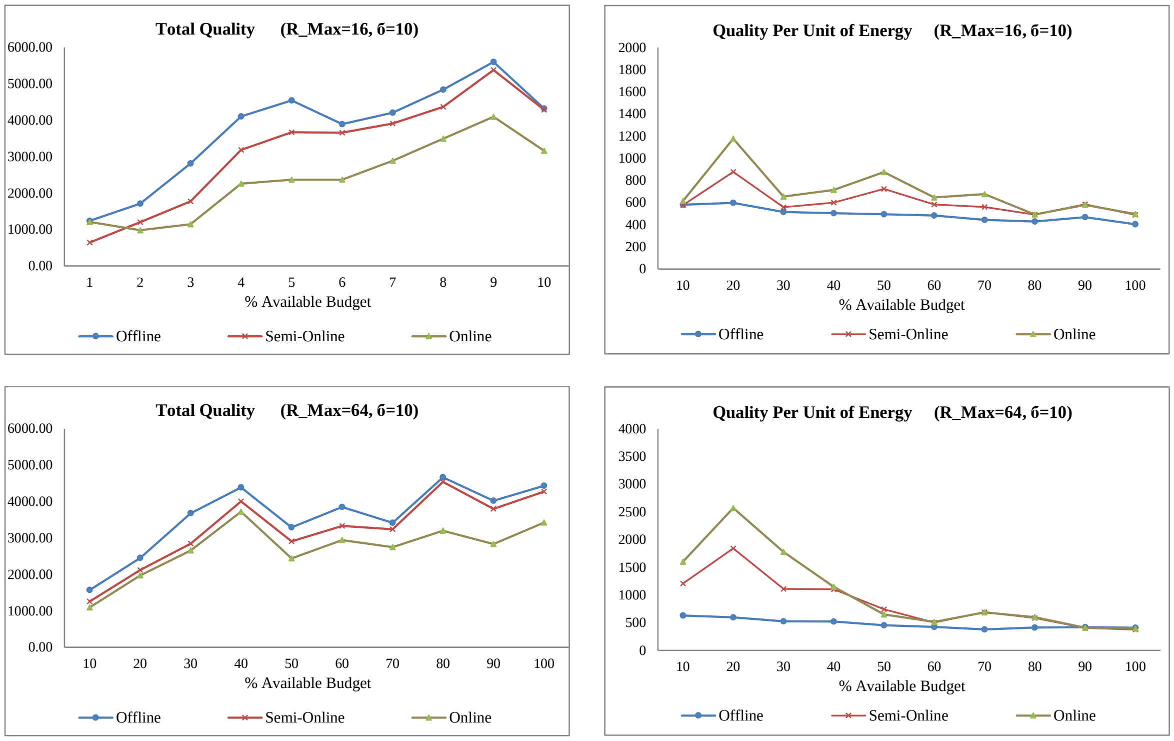


Figure 4.14: Total quality and quality per unit of energy of C_3 class of textures (textures with value 3) measured for $\delta = 10$. (Top-Left) Total quality, R_{max}=16. (Bottom-Left) Total quality, R_{max}=64. (Top-Right) Quality per unit of energy, R_{max}=16. (Bottom-Right) Quality per unit of energy, R_{max}=64.

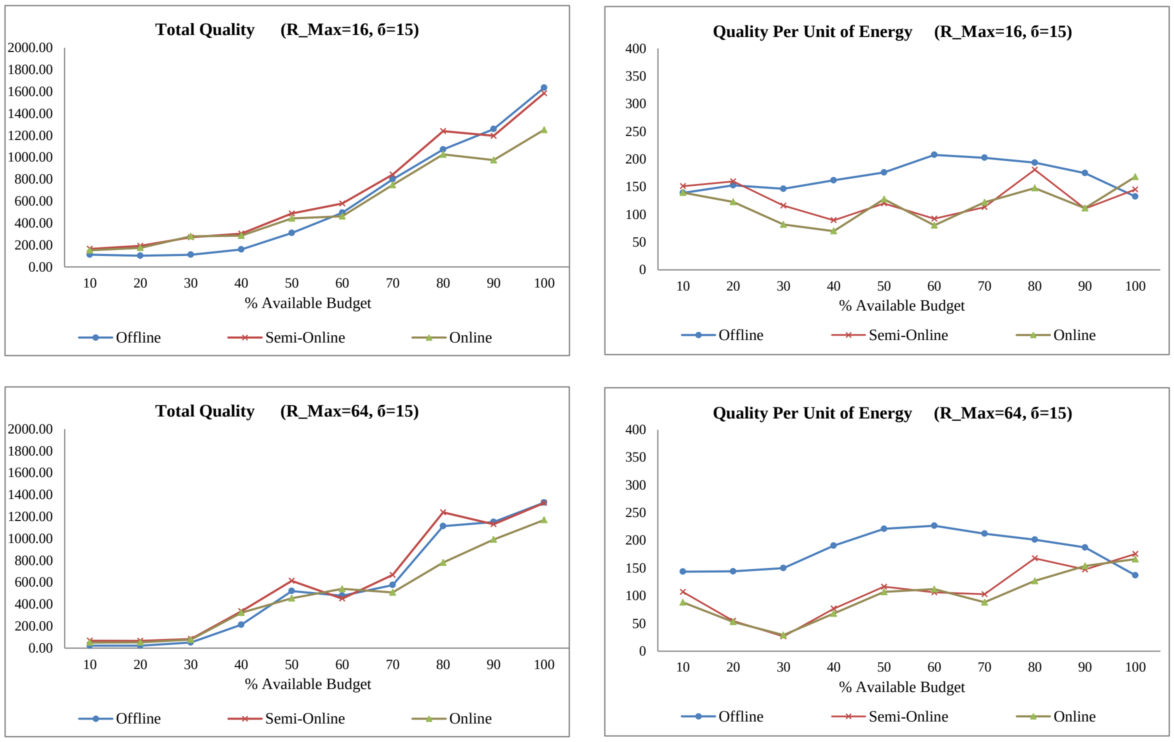


Figure 4.15: Total quality and quality per unit of energy of C_1 class of textures (textures with value 1) measured for $\delta = 15$. (Top-Left) Total quality, R_{max}=16. (Bottom-Left) Total quality, R_{max}=64. (Top-Right) Quality per unit of energy, R_{max}=16. (Bottom-Right) Quality per unit of energy, R_{max}=64.

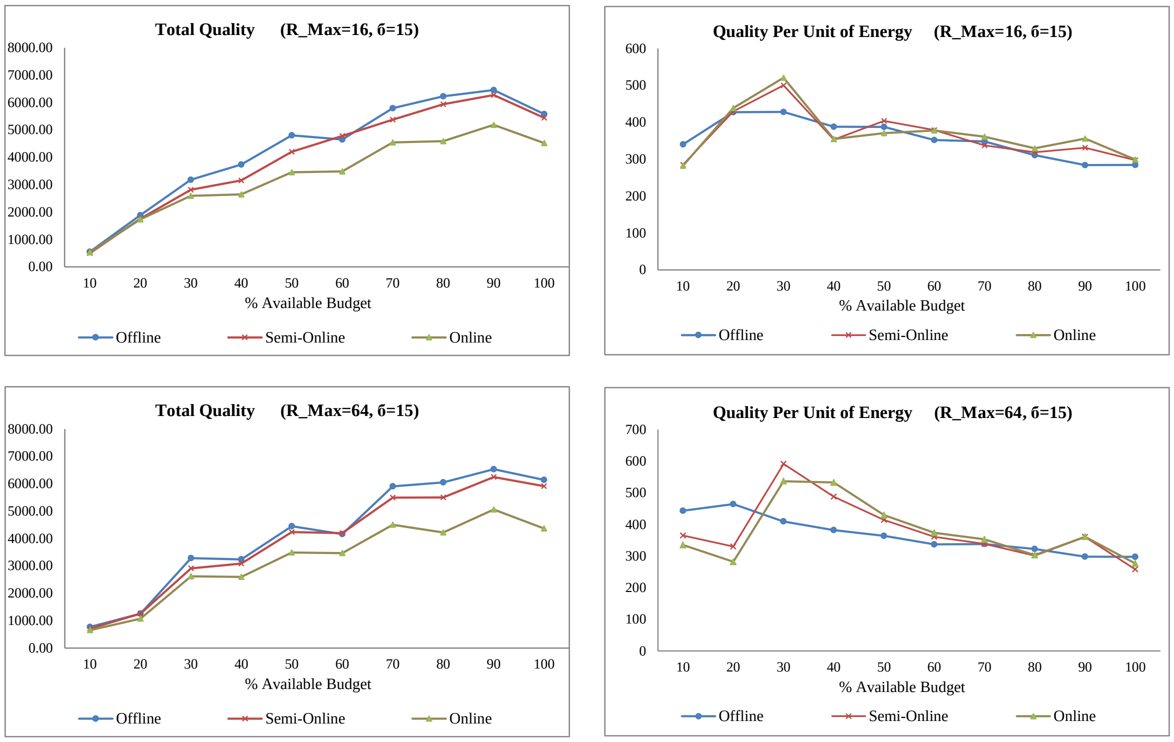


Figure 4.16: Total quality and quality per unit of energy of C_2 class of textures (textures with value 2) measured for $\delta = 15$. (Top-Left) Total quality, R_{max}=16. (Bottom-Left) Total quality, R_{max}=64. (Top-Right) Quality per unit of energy, R_{max}=16. (Bottom-Right) Quality per unit of energy, R_{max}=64.

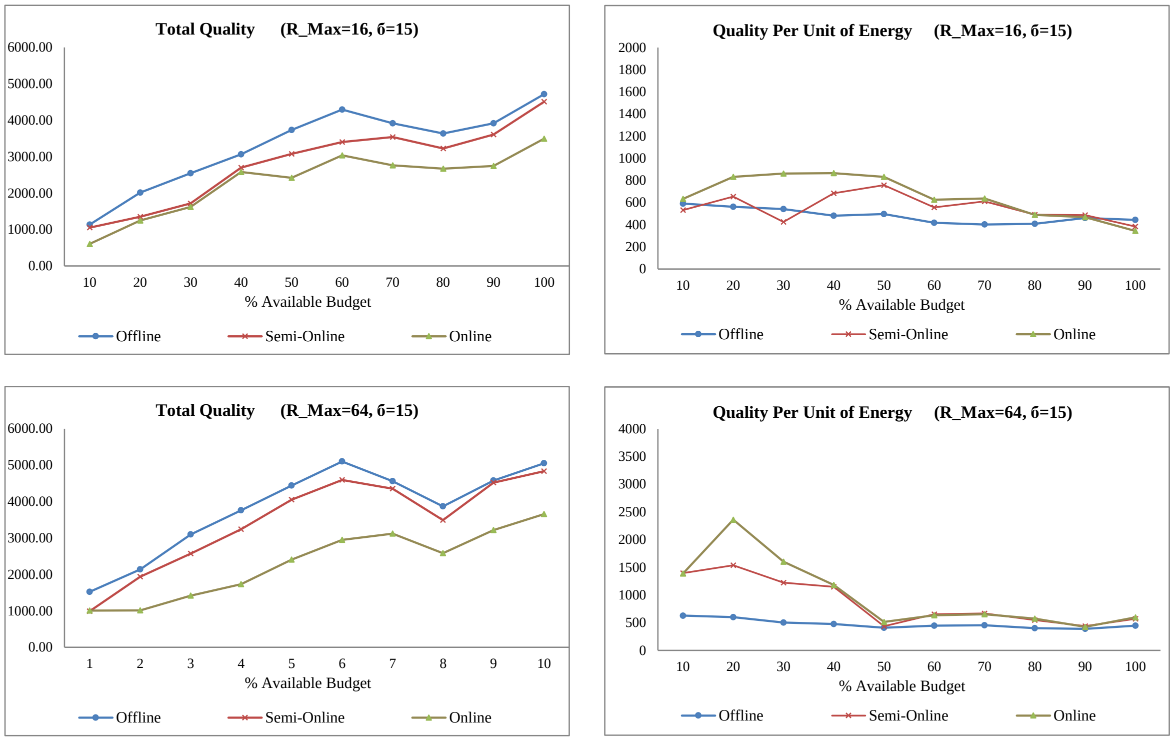


Figure 4.17: Total quality and quality per unit of energy of C_3 class of textures (textures with value 3) measured for $\delta = 15$. (Top-Left) Total quality, R_{max}=16. (Bottom-Left) Total quality, R_{max}=64. (Top-Right) Quality per unit of energy, R_{max}=16. (Bottom-Right) Quality per unit of energy, R_{max}=64.

An interesting point regarding our results which is worth analyzing more in depth is the inexplicable behaviors for a few of the resulting diagrams, especially for the detailed measurements associated with the subclasses (C_1, C_2, and C_3 classes). As shown in Figure 4.18 (top-left and top-right), we have chosen two of these instances, which accounts for the resulting total quality separately measured for two of odd-behaving diagrams. These two diagrams represents a detailed measurements of total quality for C_3 class of textures (textures with value 3) for two different δs ($\delta = 0$ and $\delta = 10$). As can be seen there are unexpected valleys and peaks in these two diagrams for which there is no logical justifications. We believe this behavior is possibly due to the randomness caused by the relatively small size of resulting texture dataset for each class. It should also be noted that due to the classification of textures (there are three different classes, namely C_1 to C_3), in average, statistically the total number of textures belonging to a specific class is one third of the total number of textures in our original experimental texture dataset, which makes it even harder to run a firm conclusion derived from the behavior of our proposed algorithms. To tackle this problem, we enlarged our experimental texture dataset, and repeated our experiments with the larger dataset, with a size of 10 times bigger than the original experimental texture dataset. Figure 4.18 bottom-left and bottom-right show the results of total quality achieved from the larger dataset. As can clearly be seen, the new diagrams do not show any strange peaks or valleys. The corresponding trendlines are still following the trendlines for the original dataset, which is actually a confirmation of our expectations.

In order to verify our approach and thus, the results derived from choosing a larger dataset, we computed the linear regression of our data points resulted from each of the three algorithms for different percentages of available budgets. The dashed linear trendlines in Figure 4.18 represent the calculated linear regressions. Given a set of data points, linear regression gives a formula to compute a linear line which is most closely matching those points. We also computed the resulting R^2 (i.e. R-Squared) value, which is interpreted as how well the resulting trendline matches the original data points, or how well the linear line is a good prediction. More generally, a higher value of R-Squared means the trendline can better predict one point of data from another. As can be seen for both $\delta = 0$ and $\delta = 10$ the trendlines in the diagrams resulted from the larger dataset show higher values of R^2 compared to the results achieved from the original dataset. For $\delta = 0$, the value of R^2 increased from 0.71 to 0.87, and from 0.87 to 0.96 for the non-adaptive benchmark algorithm and the online algorithm respectively (note that when $\delta = 0$, there is no deviations, and

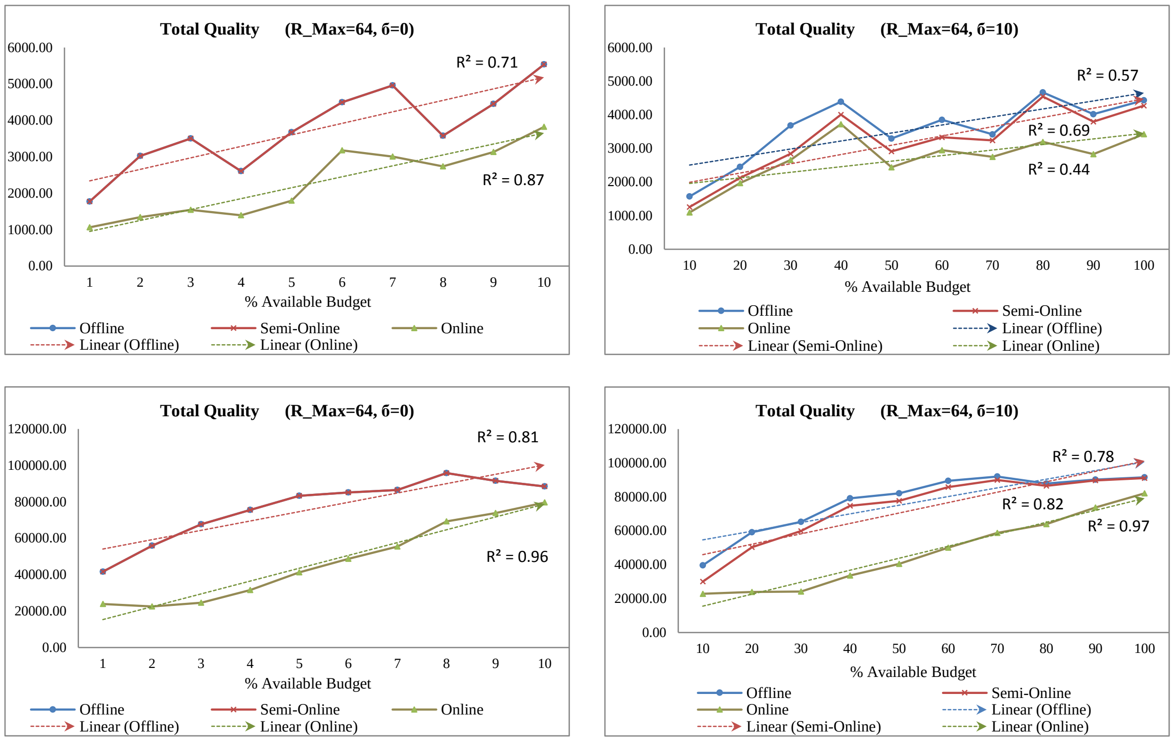


Figure 4.18: Total quality separately measured for two instances of C_3 class of textures (textures with value 3) for two different δs, along with the corresponding linear trend-lines (dashed lines) and R^2 values. As can be seen the diagrams behave smoothly when a larger experimental dataset is adopted. (Top-Left) Total quality associated with the original dataset, for R_{max}=64 and $\delta = 0$. (Bottom-Left) Total quality associated with the large dataset, for R_{max}=64 and $\delta = 0$. (Top-Right) Total quality associated with the original dataset, for R_{max}=64 and $\delta = 10$. (Bottom-Right) Total quality associated with the large dataset, for R_{max}=64 and $\delta = 10$.

thus both of non-adaptive benchmark algorithm and semi-online algorithm cause identical results). Similarly, for $\delta = 10$, the value of R^2 increased from 0.57, 0.69, and 0.44 to 0.78, 0.82, and 0.97 respectively for the non-adaptive benchmark, semi-online, and the online algorithms.

The results brought from Figure 4.18 also confirms our claim regarding the effect of randomness in the resulting odd-behaving diagrams, and that applying our proposed algorithms to larger datasets brings more validity for the resulting total qualities.

The last, but not the least, similar to the results and our corresponding argument in Chapter 3, we can clearly see that our adaptations results in minor visually perceptible changes to the scene, yet at the same time they maintain enough detail to ensure a satisfactory user gaming experience. Considering the power savings and increased playing time achieved by our adaptations, it is reasonable to believe that many users would be happy to make some minor sacrifice in exchange for longer battery life. On the other hand, our adaptations will also bring enough incentive for the game providers to implement these algorithms in their gaming system and integrate these approaches in existing gaming services because of the following two main reasons: Firstly, the increased users' satisfaction and better gaming experience causes more marketing revenue and absorbs more customers especially with the increasing demands of mobile gaming industries. Secondly the proposed adaptations also brings less traffic, less congestion and more power saving for the cloud gaming service providers since the corresponding servers are also transmitting less data.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

Nowadays with the increasing popularity of mobile 3D graphics applications, online mobile gaming and the potential application of mobile 3D virtual environments in a variety of scenarios, it would be advantageous to have these environments work smoothly and quickly, providing high levels of interactivity for their users. Therefore, the domain of online mobile gaming is fertile ground for novel approaches and new technologies.

On the other hand, with the improved processing power, graphic quality and high-speed wireless connection in the state-of-the-art mobile devices and smartphones, it looks more attractive than ever to introduce networked games on these devices. However, these games consume higher levels of energy. While device features and application resource requirements are rapidly growing, the battery technologies are not growing at the same pace. Thus, the main concern is the limited energy of these handheld devices to support a longer game play.

In this thesis we introduced an adaptive context-aware priority-based framework to manage progressive streaming of bulky 3D textures to mobile devices based on an available energy budget. Our approach is to selectively reduce the sizes of the textures so that the overall amount of data transferred to a mobile device does not exceed a download budget, with the aim of decreasing the amount of energy needed to download the 3D textures. We ran two separate studies for both static scenes and dynamic scenes. We tested our proposed methodology by implementing a proof-of-concept benchmark game and evaluating it through different metrics. Our evaluation results show that our game-context-based texture adaptation method improves the quality of textures in a gaming experience by making best use of the limited resources of mobile devices. Our approach not only reduces resource utilization at the mobile client, but also increases the quality of streamed textures and thus, the gameplay as experienced by the player. Our framework can be deployed on any types of online mobile gaming architecture.

5.2 Future Work

While our presented design accomplishes context-aware 3D texture streaming for mobile games, assumptions have been made with regard to having a complete game context model with all the importance factors for all the objects being set in advance by game designers. These two pre-requisites are outside the scope of this particular work, but in order to conduct our research and evaluate our proposed texture streaming technique, we implemented a proof-of-concept game benchmark with importance factors being set and explained how game designers could use a similar approach to their games to design them in a way that meets the requirements of our framework.

Therefore, one avenue that can be explored in the future is developing a mechanism that will allow different textures to have their importance factors set automatically during the game, which can be called a self-prioritization approach of the 3D objects, or the corresponding textures. In this approach, during initial gameplay, different 3D objects will learn how important they are, and their importance factor will then be set automatically by the framework and not by game designers in advance. This can be for instance as a history-based approach in which a one-time prioritization can be done, and then used for future gameplays. This requires applying a specific method in order for objects to become intelligent in such a way that they learn from gameplay.

Moreover, as another possible avenue of future work, instead of our gameplay simulations in chapter 4, we study the exact amount of streaming deviations and thus, battery saving of the mobile client. This can be done exactly the same as our initial study for the real game power in an experimental way, by running the game several times and measuring how much energy deviations will be associated with each texture.

Another possible research could be searching for more complicated texture compression approaches such as PNG or JPEG compressions instead of a down-sampling approach, which makes the textures to be compressed by any possible amount. This will remove the necessity of sampling by the sole factor of 4.

As another avenue for future work, we are going to look into more complicated heuristics for use of our semi-online and online algorithms defined in chapter 4. Our future heuristics will take into account the value of upcoming texture to make decisions.

And last, but not least, we also plan to more precisely estimate a download limit D based on the energy budget. This can also be measured in an experimental way, by running the game several times and measuring how much deviation will be caused in average after downloading of that specific texture.

Bibliography

[1] Android market - android apps on google play. http://play.google.com.

[2] Droid does: Quake 3. also quake 2. and they're available for almost all android phones. http://blog.gsmarena.com/droid-does-quake-3-also-quake-2-and-theyre-available-for-almost-all-android-phones/ [With permission].

[3] HTC EVO 3D - HTC support. https://support.htc.com/en-uk/HTC_EVO_3D [With permission].

[4] Hypertext transfer protocol- HTTP/1.1, RFC 2616. http://www.ietf.org.

[5] java.util.Random class- Oracle documentation. http://docs.oracle.com/javase/6/docs /api/java/util/Random.html.

[6] LTE: 4G long term evolution. http://www.3gpp.org/LTE.

[7] Newzoo's mobile games trend report. http://www.newzoo.com/trend-reports/mobile-games-trend-report.

[8] Nokia store. store.ovi.com.

[9] Official apple store. store.apple.com.

[10] Unity 3D game engine. http://www.unity3d.com.

[11] Adaptive bitrate streaming. http://en.wikipedia.org/wiki/Adaptive_bitrate_streaming, July 28, 2011 [With permission].

[12] Mobile phones used more than pcs to browse the internet. Softpedia News, March 21, 2009.

[13] Global cell phone use at 50 percent. Reuters, November 29, 2007.

[14] LTE Advanced: The Global 4G Solution. http://www.qualcomm.com, Qualcomm, 2012.

[15] S. Andreev, Y. Koucheryavy, N. Himayat, P. Gonchukov, and A. Turlikov. Active-mode power optimization in OFDMA-based wireless networks. In *GLOBECOM Workshops (GC Wkshps), 2010 IEEE*, pages 799 –803, dec. 2010.

[16] Niranjan Balasubramanian, Aruna Balasubramanian, and Arun Venkataramani. Energy consumption in mobile phones: a measurement study and implications for network applications. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, IMC '09, pages 280–293, New York, NY, USA, 2009. ACM [With permission].

[17] Wilhelm Burger and Mark J. Burge. *Principles of Digital Image Processing: Core Algorithms.* Springer. ISBN-10: 1848001940, 1st edition, March 2009.

[18] Aaron Carroll and Gernot Heiser. An analysis of power consumption in a smartphone. In *Proceedings of the 2010 USENIX conference on USENIX annual technical conference*, USENIXATC'10, pages 21–21, Berkeley, CA, USA, 2010. USENIX Association.

[19] Jiasi Chen, Amitabha Ghosh, Josphat Magutt, and Mung Chiang. Qava: quota aware video adaptation. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, CoNEXT '12, pages 121–132, New York, NY, USA, 2012. ACM.

[20] Tapas K. Das, Gurminder Singh, Alex Mitchell, P. Senthil Kumar, and Kevin McGee. Neteffect: a network architecture for large-scale multi-user virtual worlds. In *Proceedings of the ACM symposium on Virtual reality software and technology*, VRST '97, pages 157–163, New York, NY, USA, 1997. ACM.

[21] U. Farooq and J. Glauert. Time management for virtual worlds based on constrained communication model. In *Network and Systems Support for Games (NetGames), 2010 9th Annual Workshop on*, pages 1 –6, nov. 2010.

[22] Stephane Gouache, Guillaume Bichot, Amine Bsila, and Christopher Howson. Distributed and adaptive HTTP streaming. In *Multimedia and Expo (ICME), 2011 IEEE International Conference on*, pages 1 –6, july 2011.

[23] O. Hagsand. Interactive multiuser VEs in the DIVE system. *MultiMedia, IEEE*, 3(1):30 –39, spring 1996.

[24] Daniel Halperin, Ben Greenstein, Anmol Sheth, and David Wetherall. Demystifying 802.11n power consumption. In *Proceedings of the 2010 international conference on Power aware computing and systems*, HotPower'10, pages 1–, Berkeley, CA, USA, 2010. USENIX Association.

[25] R. Cameron Harvey, Ahmed Hamza, Cong Ly, and Mohamed Hefeeda. Energy-efficient gaming on mobile devices using dead reckoning-based power management. In *Proceedings of the 9th Annual Workshop on Network and Systems Support for Games*, NetGames '10, pages 4:1–4:6, Piscataway, NJ, USA, 2010. IEEE Press.

[26] Iourcha Konstantine I. Hong, Zhou and Krishna S. Nayak. System and method for fixed-rate block-based image compression with inferred pixel values. (US Patent no 6775417), August 2004.

[27] Mohammad Hosseini, Dewan T. Ahmed, and Shervin Shirmohammadi. Adaptive 3D texture streaming in M3G-based mobile games. In *Proceedings of the 3rd Multimedia Systems Conference*, MMSys '12, New York, NY, USA, 2012. ACM.

[28] Mohammad Hosseini, Joseph Peters, and Shervin Shirmohammadi. Energy-budget-compliant adaptive 3d texture streaming in mobile games. In *Proceedings of the 4th ACM Multimedia Systems Conference*, MMSys '13, pages 1–11, New York, NY, USA, 2013. ACM.

[29] Raj K. Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling.* Wiley. ISBN-10: 0471503363, 1st edition, 1991.

[30] Martin Kennedy, Hrishikesh Venkataraman, and Gabriel-Miro Muntean. Battery and stream-aware adaptive multimedia delivery for wireless devices. In *Proceedings of the 2010 IEEE 35th Conference on Local Computer Networks*, LCN '10, pages 843–846, Washington, DC, USA, 2010. IEEE Computer Society.

[31] Jae Kim and Myung Lee. *Green IT: Technologies and Applications.* Springer. ISBN-10: 3642221785, 1st edition, Jul 2011.

[32] Kyu-Han Kim, Alexander W. Min, Dhruv Gupta, Prasant Mohapatra, and Jatinder Pal Singh. Improving energy efficiency of wi-fi sensing on smartphones. In *IEEE International Conference on Computer Communications (INFOCOM)*, pages 2930–2938. IEEE, July 2011.

[33] Mohamed Lamine Lamali, Helia Pouyllau, and Dominique Barth. End-to-end quality of service in pseudo-wire networks. In *Proceedings of The ACM CoNEXT Student Workshop*, CoNEXT '11 Student, pages 21:1–21:2, New York, NY, USA, 2011. ACM.

[34] J. Leigh, A.E. Johnson, C.A. Vasilakis, and T.A. DeFanti. Multi-perspective collaborative design in persistent networked virtual environments. In *Virtual Reality Annual International Symposium, 1996., Proceedings of the IEEE 1996*, pages 253 –260, 271–2, mar-3 apr 1996.

[35] G.P. Perrucci, F.H.P. Fitzek, and J. Widmer. Survey on energy consumption entities on the smartphone platform. In *Vehicular Technology Conference (VTC Spring), 2011 IEEE 73rd*, pages 1 –6, may 2011.

[36] Henrik Petander. Energy-aware network selection using traffic estimation. In *Proceedings of the 1st ACM workshop on Mobile internet through cellular networks*, MICNET '09, pages 55–60, New York, NY, USA, 2009. ACM.

[37] H. Rahimi, A.A. Nazari Shirehjini, and S. Shirmohammadi. Activity-centric streaming of virtual environments and games to mobile devices. In *Proc. IEEE Symposium on Haptic Audio Visual Environments and Games*, pages 45 –50, Qinhuangdao, Hebei, China, Oct. 2011.

[38] Ahmad Rahmati and Lin Zhong. Context-for-wireless: context-sensitive energy-efficient wireless data transfer. In *Proceedings of the 5th international conference on Mobile systems, applications and services*, MobiSys '07, pages 165–178, New York, NY, USA, 2007. ACM.

[39] J. Royan, P. Gioia, R. Cavagna, and C. Bouville. Network-based visualization of 3D landscapes and city models. *Computer Graphics and Applications, IEEE*, 27(6):70 –79, Dec. 2007.

[40] Kurt Saar. Virtus: a collaborative multi-user platform. In *Proceedings of the fourth symposium on Virtual reality modeling language*, VRML '99, pages 141–152, New York, NY, USA, 1999. ACM.

[41] J.J. Sanchez, D. Morales-Jimenez, G. Gomez, and J.T. Enbrambasaguas. Physical layer performance of long term evolution cellular technology. In *Mobile and Wireless Communications Summit, 2007. 16th IST*, pages 1 –5, july 2007.

[42] Shervin Shirmohammadi, Mohamed Hefeeda, Wei Tsang Ooi, and Romulus Grigoras. Introduction to special section on 3d mobile multimedia. *ACM Trans. Multimedia Comput. Commun. Appl.*, 8(3s):41:1–41:3, October 2012.

[43] J. Soh and B. Tan. Context-for-wireless: context-sensitive energy-efficient wireless data transfer. In *Proceedings of the 5th international conference on Mobile systems, applications and services*, MobiSys '07, March 2008.

[44] Yang Song, Chi Zhang, and Yuguang Fang. Multiple multidimensional knapsack problem and its applications in cognitive radio networks. In *Military Communications Conference, 2008. MILCOM 2008. IEEE*, pages 1–7, 2008.

[45] Anthony Steed and Manuel Fradinho Oliveira. *Networked Graphics: Building Networked Games and Virtual Environments.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2009.

[46] Thomas Stockhammer. Dynamic adaptive streaming over HTTP: standards and design principles. In *Proceedings of the second annual ACM conference on Multimedia systems*, MMSys '11, pages 133–144, New York, NY, USA, 2011. ACM.

[47] Jacob Ström and Tomas Akenine-Möller. PACKMAN: texture compression for mobile phones. In *ACM SIGGRAPH 2004 Sketches*, SIGGRAPH '04, pages 66–, New York, NY, USA, 2004. ACM.

[48] Jacob Ström and Martin Pettersson. ETC2: texture compression using invalid combinations. In *Proceedings of the 22nd ACM SIGGRAPH symposium on Graphics hardware*, GH '07, pages 49–54, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[49] Douglas Thomas and Jagan Nemani. The collaboration curve: Exponential performance improvement in world of warcraft. http://www.deloitte.com, October 2009.

[50] R. Waters, D. Anderson, J. Barrus, D. Brogan, M. Casey, S. McKeown, T. Nitta, I. Sterns, and W. Yerazunis. Diamond park and spline: A social virtual reality system with 3D animation, spoken interaction, and runtime modifiability. Presence, vol. 6, no. 4, pp. 461-480.

[51] Lide Zhang, B. Tiwana, R.P. Dick, Zhiyun Qian, Z.M. Mao, Zhaoguang Wang, and Lei Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2010 IEEE/ACM/IFIP International Conference on*, pages 105 –114, oct. 2010.

Appendix A

HTC Evo 3D Smartphone

HTC Evo 3D smartphone is an advanced smartphone which provides a platform to develop high-end applications targeting demanding mobile consumers. This smartphone is a proper tool to develop, test, and to showcase innovative applications which feature rich graphics, high-performance multimedia, and unparalleled user experience.

The HTC EVO 3D is an Android smartphone developed by HTC which was released exclusively in the United States, and was first pre-released in May 2012 as the name HTC Evo V 4G. As a special feature, this is a good device to use for 3D showcases and proof-of-concept testing benchmarks since it can be used to take photos or video in stereographic 3D, which can be viewed on its autostereoscopic display without the need for 3D glasses. Reference [3] provides a detailed explanation on the features and the technical specification of this smartphone.

Figure A.1: An overview picture of a HTC EVO 3D [3]