Energy-Budget-Compliant Adaptive 3D Texture Streaming in Mobile Games

Mohammad Hosseini
Simon Fraser University
mohammad_hosseini@sfu.ca

Joseph Peters
Simon Fraser University
peters@cs.sfu.ca

Shervin Shirmohammadi
University of Ottawa
shervin@eecs.uottawa.ca

ABSTRACT

Advances in computing hardware and novel multimedia applications have urged the development of handheld mobile devices such as smartphones and PDAs. Amongst the most used applications on handheld devices are mobile 3D graphics such as 3D games and 3D virtual environments. With this significant increase of mobile applications and games, one of the challenges is how to efficiently transmit the bulky 3D information to resource-constrained mobile devices. Despite the many attractive features, 3D graphics impose significant demands on the limited battery capacity of mobile devices. Thus the development of 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 paper, we design and implement an adaptive priority-based context-aware 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.

Categories and Subject Descriptors

K.8.0 [**Personal Computing**]: General - Games ; I.3.7 [**Computer Graphics**]: Three-dimensional graphics and realism

General Terms

Design, Algorithms, Experimentation, Measurement

Keywords

Mobile 3D Games, Budget-based Gaming, Context-aware 3D Streaming, Energy-efficient Mobile Gaming, 3D Texture Streaming

1. INTRODUCTION

Efficient energy consumption is very important for battery-operated devices such as smartphones. The increasing power requirements of new smartphones and tablets are far outpacing improvements in battery technology. Switching over to superfast 4G networks, as many will in 2013, 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 [2, 4, 20]. A typical modern smartphone does not last more than about 24 hours without having to be recharged. 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 [8, 14]. 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 [9, 15]. 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 [16].

Rahmati, *et al* [19] 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

$$E = Ee + nEt \qquad (1)$$

where Ee is the energy cost for connection establishment and Et the energy per MByte of data transfer. Also in [17] and [6], 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). Figure 1 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 [5]. The high data rates and resulting high energy demands of modern networked multimedia systems

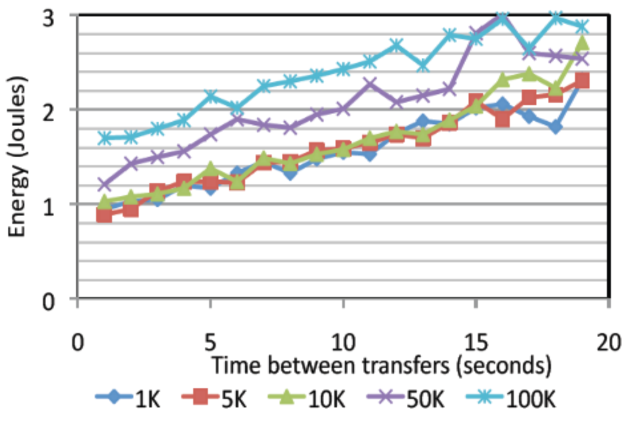


Figure 1: Average energy consumption for downloading data of different sizes over varying inter-transfer times for WiFi [6].

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. It is estimated that 78.6 million people in the U.S. alone played mobile games in 2009, downloads of mobile games increased tenfold compared to 2003, and mobile games generated more than $1.5 billion annually in revenue [12]. 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.

Due to the limitations in network bandwidth, memory size, battery life and computation resources of 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 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 paper, we propose an adaptive framework to efficiently stream bulky 3D textures to mobile devices. 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 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.

The paper is organized as follows: in Section 2, we discuss relevant background and previous work on textures, texture compression, and 3D texture streaming along with adaptive multimedia streaming. Section 3 explains our methodology for adaptive texture streaming. Section 4 presents our evaluation and experimental results, and Section 5 contains our conclusions and a discussion of future work.

2. BACKGROUND

In this section 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. 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.

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

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) [11] and Ericsson Texture Compression (ETC) [23]. 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) [24]. 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 decreasing the size of textures 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 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. This adaptation of quality is controlled by decision modules on either the client or the server. The adaptation may be the result of adjusting 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 [10].

2.4 Related work

In large virtual environments, users only interact with a subset of the objects that are visible to them at a specific given time. Work for 3D streaming that takes advantage of this fact are classified into two major classes: region-based and interest-based streaming. Region-based approaches only stream the geometry information for the player's region of activity, while interest-based techniques use an Area of Interest (AOI) to determine object visibility.

As the main flaw, 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 and textures 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* [13] 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 [18] 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 [12], 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 [12] was to improve the response time. In this paper, we build on concepts from [18] and [12] 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.

3. PROPOSED FRAMEWORK

In this section, we explain our methodology regarding the implementation of different parts of the adaptive texture streaming system. 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 Section 1, with the aim of decreasing power consumption.

Figure 2 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

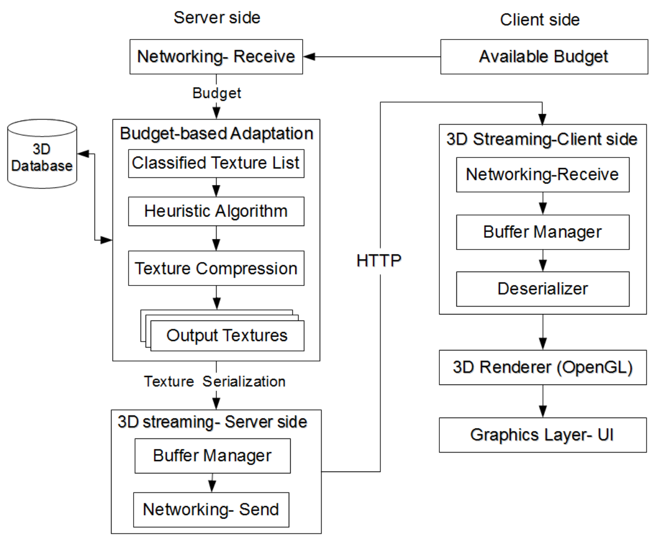


Figure 2: Outline of the proposed framework

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. 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 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 Section 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 W for the textures. Every 3D texture has a specific size, and we must decide how to stream them within the budget W.

One approach to reducing the total size of the textures that are streamed is to transmit a subset of them. Let $T = \{T1, T2, T3, ...\}$ be the set of textures. Each texture Ti has a size w_i and a value v_i which is based on the importance of the corresponding object. The goal is to stream a subset of textures $T' \subseteq T$ that maximizes the total value of the streamed textures without exceeding the download budget W. In other words

$$\text{Maximize}_{\{T' \subseteq T\}} \sum_{Ti \in T'} v_i \;\text{ subject to }\; \sum_{Ti \in T'} w_i \leq W. \qquad (2)$$

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 15 (I) and Figure 15 (II).

To overcome the shortcomings of the 0-1 Knapsack approach, we propose a heuristic algorithm that sends all textures, but with different qualities according to their priorities. The general idea is to apply textural compression using a sub-sampling approach (described in detail in subsection 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 subsection 3.2.

Figure 3 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 (I) through (IV) are 512×512, 256×256, 128×128, and 64×64, respectively (all with 8-bit color depth). Figure 3 (V) shows the object with no texture applied. The reduction in quality in Figures 3 (I) through (IV) is noticeable, but the impact of even the greatest reduction (Figure 3 (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 is 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 netwroked 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].

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

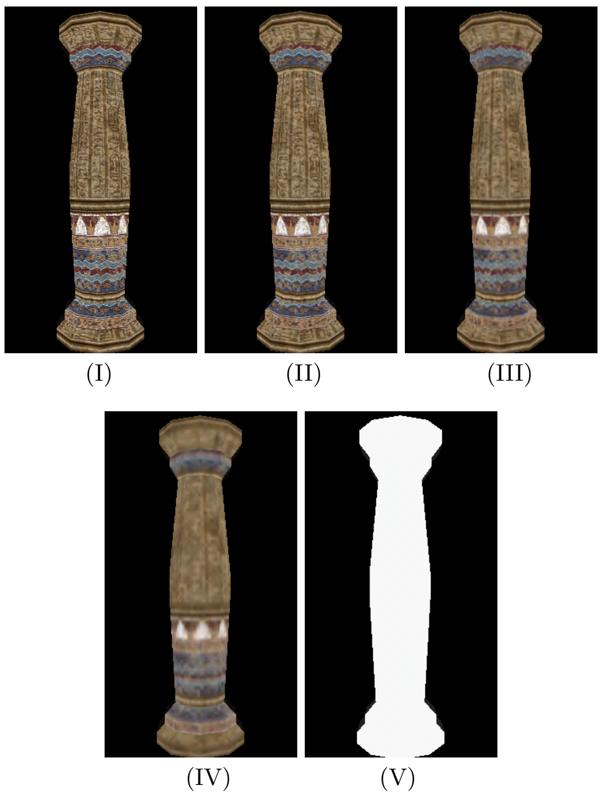


Figure 3: Visual view of applying a texture with different resolutions to a 3D column object. (I to IV): the texture resolutions are 512×512, 256×256, 128×128 and 64×64 with the same depth color. (V): no texture.

textures into two different classes: Important (C1) and Less-important (C2). Currently, some 3D game engines (such as Unity3D) [3] 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 priority tagging for textures is not a considerable overhead for game design. In future work, we plan to use multiple priorities 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. Each Tij in the list represents a single texture where i signifies the priority class, and j is the index of the texture within that specific class, as shown in Figure 4.

Figure 5 is a hierarchical pyramid that shows the possible texture resolutions used in this paper. As can be seen in Figure 5, 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 following equation can be written:

$$Size(Li) = 4^{(j-i)} \times Size(Lj) \qquad j \geq i > 0 \qquad (3)$$

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.

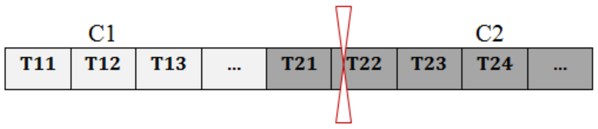


Figure 4: An overview of the texture list, showing two different priority classes, C1 and C2, along with the corresponding textures. The red sign represents a hypothetical available budget, shown as a cutoff.

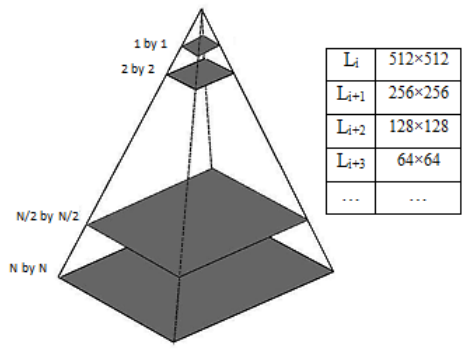


Figure 5: Different texture resolutions represented as various levels, along with a hierarchical view.

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 Section 1, it is closely related to the sizes of the textures needed to be streamed to the client. So we estimate a download limit W 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 reduction of resolution. Let R_max be the maximum reduction in resolution that is acceptable to the mobile user. In this paper, we assume that $R\_max = 4^i$ for some $i \geq 0$.

As an example, if the original size of a texture is W and $R\_max = 4^i$ with $i = 2$, then the size of the smallest acceptable compressed texture will be $W/R\_max = W/16$.

The textures are classified into two categories, important (C1) and less-important (C2), and each category has an associated relative value (V1 and V2). The assignments to categories and the associated values are decided by the game designer. We normalize the values by setting $V2 = 1$. Let $S1$ be the total size of textures in C1 before resizing, $S2$ the total size of textures in C2 before resizing, and $S = S1 + S2$.

We assume that the quality of a streamed texture is a function of its size (with maximum quality corresponding to minimum resizing) and its 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 of original size W with resizing factor R and value V has quality $\frac{W \times V}{R}$. Our goal is to compress textures in a way that maximizes the total quality subject to the constraints of the download budget W and maximum resolution R_max.

Algorithm

Calculate $S1$ (Total size of all textures in C1), $S2$ (Total size of all textures in C2), S (S1+S2), W (Available budget).

- If $S \leq W$, then no resizing is needed.
- If $\frac{S}{R\_max} > W$ then the problem cannot be solved within the constraints W and R_max.
- Otherwise, we solve one of the following subproblems.

Subproblem 1:

If $S1 + \frac{S2}{R\_max} \leq W$ then all textures in C1 can be sent uncompressed and the problem is to compress the textures in C2 in a way that maximizes the value of the compressed textures in C2 within the download budget $W2 = W - S1$.

Subproblem 2:

If $S1 + \frac{S2}{R\_max} > W$ then we compress all textures in C2 by R_max and the problem is to compress the textures in C1 in a way that maximizes the value of the compressed textures in C1 within the download budget $W1 = W - \frac{S2}{R\_max}$.

Algorithm for subproblem 1:

Calculate $i1$ such that:

$$\frac{S2}{4^{i1}} \leq W2 < \frac{S2}{4^{(i1-1)}}. \tag{4}$$

In other words, find the minimum $i1$ such that all textures in S2 can be streamed within the budget $W2$ when they are resized by the factor 4^{i1}.

Our goal is to maximize the total size of the textures sent within the budget $W2$. To achieve this, we compress the first texture in C2 by 4^{i1}. Suppose that after resizing, it has size X. This leaves a budget of $W2 - X$ for the remaining textures. We then calculate a new $i2$ for the remaining textures using budget $W2 - X$ and compress the second texture in C2 by 4^{i2}. 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 C1 with an available budget $W1 = W - \frac{S2}{R\_max}$.

3.3 Texture Compression

To make the size of less-important textures smaller using the concept of down-sampling, 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.

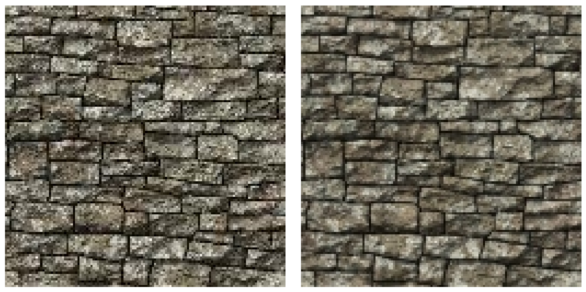


Figure 6: A 128 × 128 stone wall texture produced by TCM (left) compared with the bicubic resampling algorithm (right).

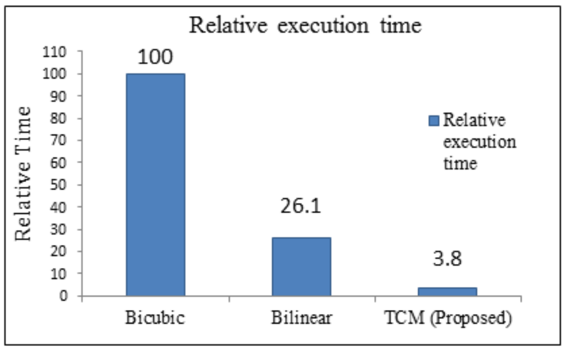


Figure 7: A comparison of relative execution time needed for running bicubic, bilinear and our proposed algorithm to resize a 512 × 512 texture by a ratio of 2.

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 [7]. The resizing method that we use in this paper imposes very little demand on the CPU, but could result in lower quality than other methods depending on how quality is measured. Figure 7 shows a comparison of relative execution times for different resizing methods, and Figure 6 compares an output texture produced by TCM with a complex bicubic resizing algorithm. However, our general approach is independent of the resizing method. Even if the resizing algorithms have much overhead, it is not a big issue since the resizing is done on the server, not on the mobile client. The resized textures can also be precomputed and stored for future uses. We can use the concept of mipmaps to store textures in multiple resolutions. 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 [1]. Therefore, to take advantage of the above gains, we used HTTP 1.1 as the protocol for the communication channel.

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 [22]. 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.

4. 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 importance 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 priority 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 priority 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 8 (I) to (III) show screenshots of different stages in the streaming of a particular scene based on the prioritization streaming approach. Figure 8 (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. To calculate the amount of consumed energy, we used PowerTutor [25], a profiler which measures the power consumption of various hardware components using a device's built-in battery voltage sensors.

To evaluate our proposed texture adaptation algorithm, we ran our ninja benchmark with W set to be different percentages of S (total size of all textures). In particular, we set W 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 C1 being compressed for most values of W. 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 in each priority 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 9 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 W (as a percentage of S). We measured the

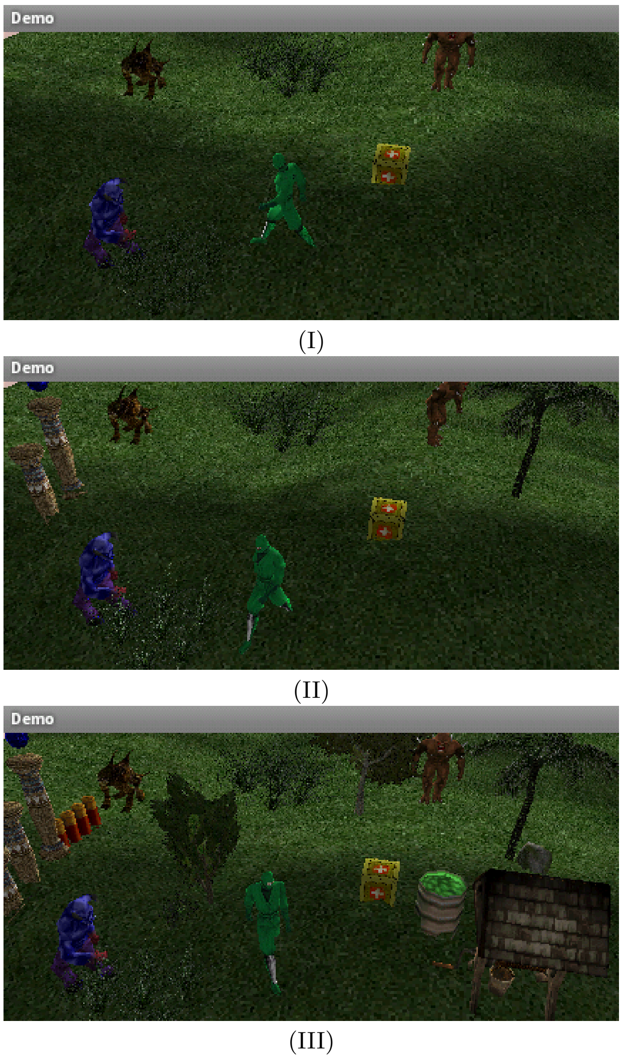


Figure 8: (I) to (III): Different stages using prioritized progressive game streaming in the initial demo benchmark. Based on the gaming context, the player's avatar, the enemies and goal targets are more important as opposed to trees or grasses, so they are streamed in earlier stages.

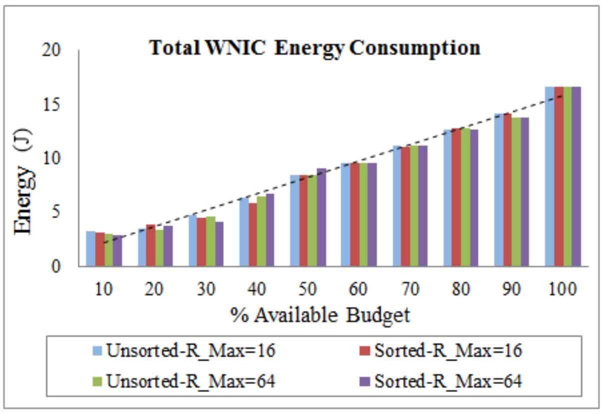


Figure 9: Total energy consumption for WiFi 802.11g Network Interface Controller.

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) and Figure 1. 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 10 show the results for the average compression in terms of average scaling ratio, measured for all textures in C1 and C2, for both sorted and unsorted texture lists. In Figure 10 (top) $R\_max = 16$, while in Figure 10 (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 C2 (i.e. textures tagged as *less important*), while preserving more of the original textures in C1 (i.e. *important*-tagged textures). Also it should be noted that the larger value of R_max works better for lower values of the available budget.

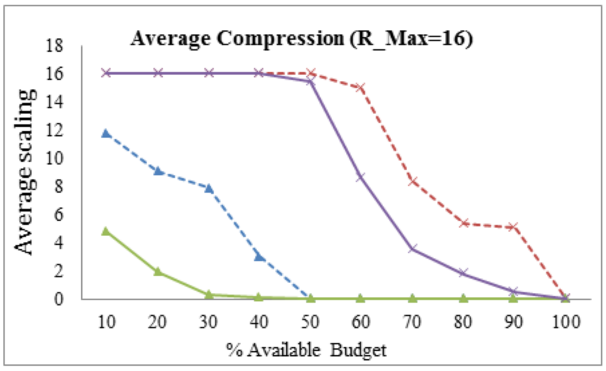
As discussed in Subsection 3.2, we assume that the quality of a streamed object is a function of its size and a relative value assigned by the game designer to its class. If $S1'$ and $S2'$ are the total sizes of the streamed textures in classes C1 and C2, respectively, that are received by the client, and V1 and V2 are the associated relative values, then we define the *value factor* for a scene to be

$$\text{Value Factor} = \frac{V1.S1' + V2.S2'}{S} \tag{5}$$

where S is the total size of the uncompressed textures. We normalized the relative values by setting V2=1.

The value factor 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 11 shows our experimental results for value factor per available budget for sorted and unsorted texture lists and two different values of R_max (16 and 64). We used three different pairs (V1,V2=1) of relative values, (1,1), (2,1) and (3,1), to differentiate the priorities of textures in C1 and C2. As can be seen in all four graphs, the value factor increases as the ratio $\frac{V1}{V2}$ increases, confirming that our proposed approach noticeably distinguishes the important textures from the less important ones. Interestingly, the maximum gap for



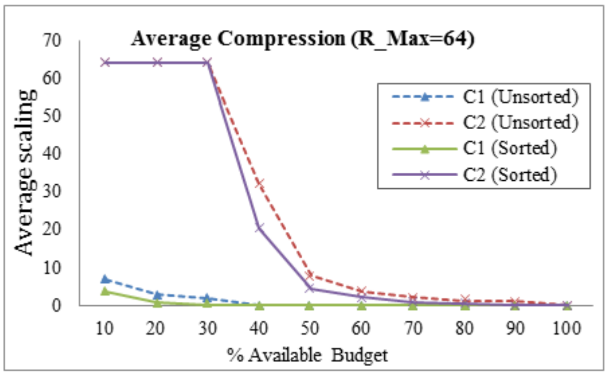


Figure 10: Average compression in terms of average scales, accounted for C1 (i.e. Important textures) and C2 (i.e. Less-Important textures), both for sorted and unsorted texture lists. (Top): R_max set as 16 (Bottom): R_max set as 64.

the growth of the value factor is in the range of 30% to 50% of the available budget in all four graphs.

Figure 12 shows our results for value per unit of energy for the same experiments as in Figure 11. As can be seen, in all four graphs, larger $\frac{V1}{V2}$ ratios result in more value per unit of energy consumption. In all of the graphs, the maximum gain in value is in the range of 30% to 40% of the available budget suggesting that this the range where our approach is most effective.

Figure 13 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* resizing. In Figure 13 (I), due to the budget limit, some textures have not been transferred, while in Figure 13 (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 13 (I) does not provide a pleasing or acceptable game experience. Compared with the original demo in Figure 8 (III) in which the download budget is unlimited, our method does produce noticeable differences in quality. However, considering the power savings achieved using our 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.

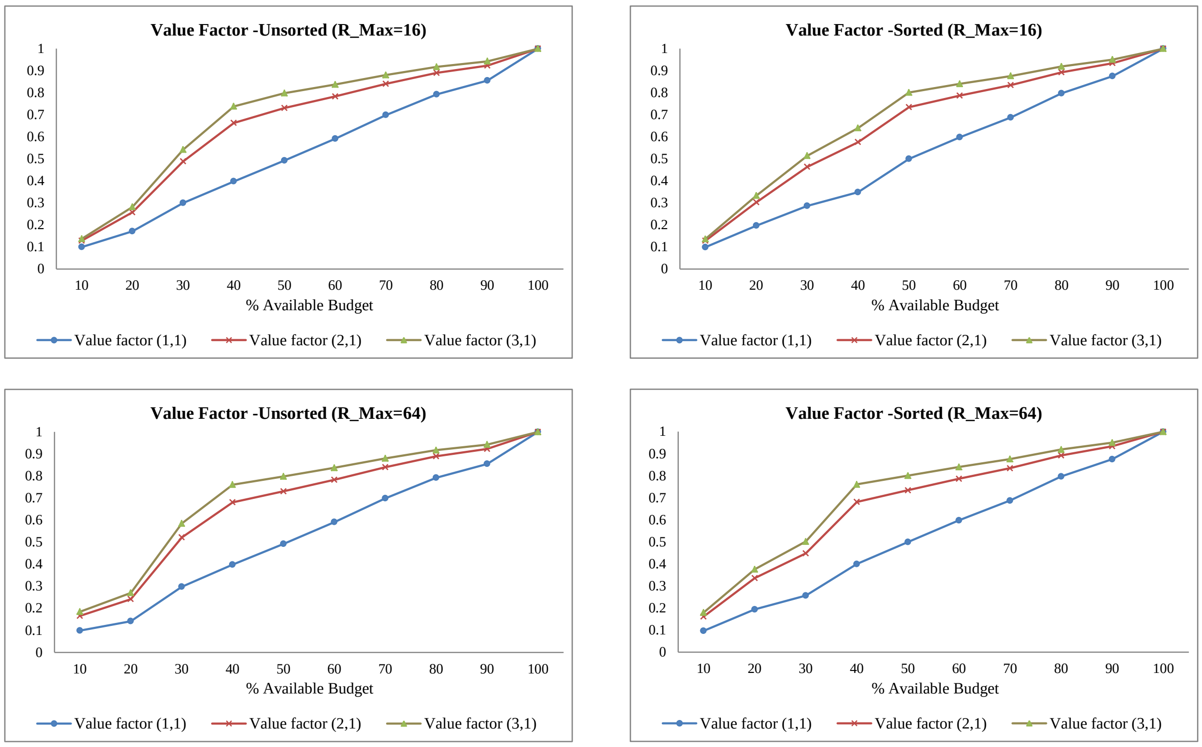


Figure 11: Comparison of the value factor measured for four situations and three relative value pairs (V1,V2=1). (Top-Left) Unsorted texture list, $R\_max = 16$. (Top-Right) Sorted texture list, $R\_max = 16$. (Bottom-Left) Unsorted texture list, $R\_max = 64$. (Bottom-Right) Sorted texture list, $R\_max = 64$.

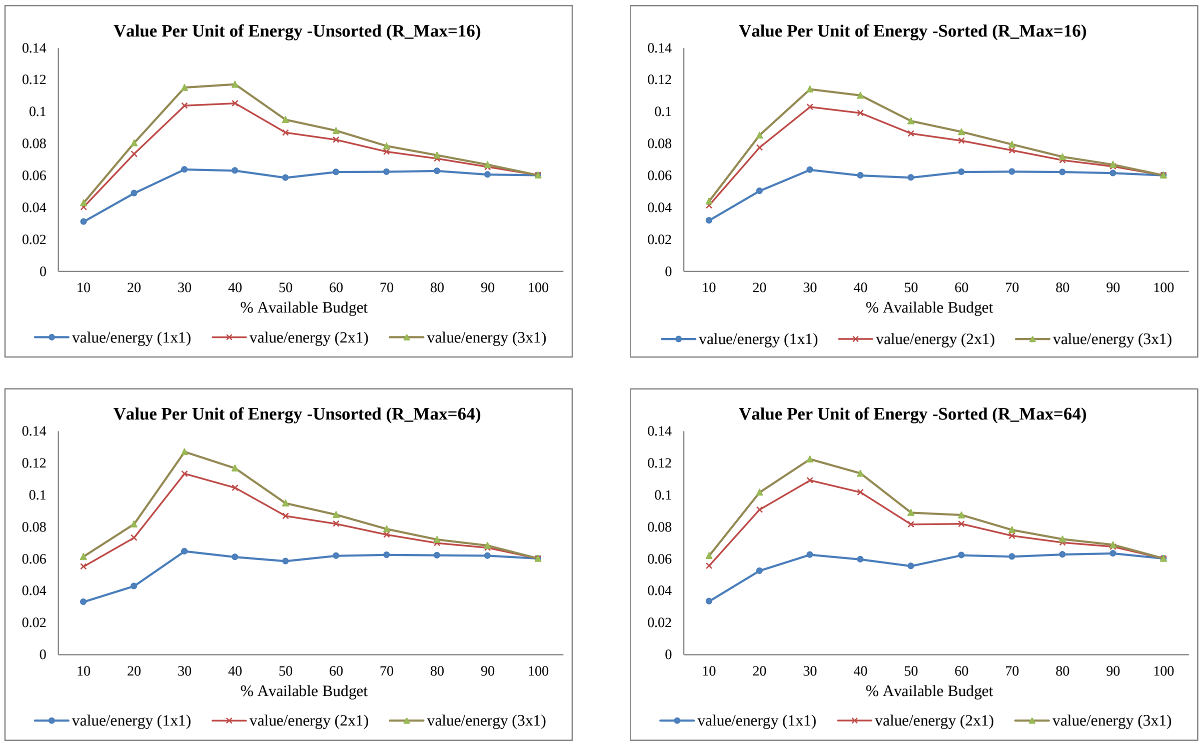


Figure 12: Value of data per unit of energy for four situations and three relative value pairs (V1,V2=1). (Top-Left) Unsorted texture list, $R\_max = 16$. (Top-Right) Sorted texture list, $R\_max = 16$. (Bottom-Left) Unsorted texture list, $R\_max = 64$. (Bottom-Right) Sorted texture list, $R\_max = 64$.

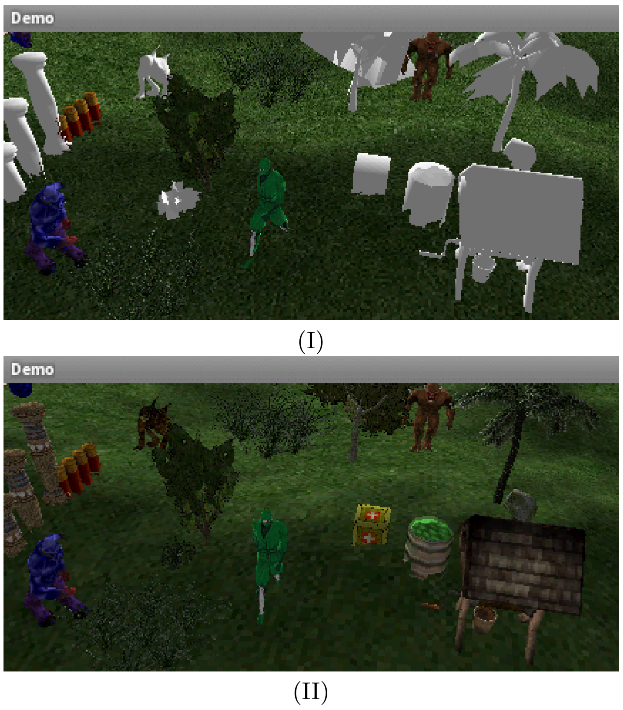


Figure 13: Visual comparison of a gaming scene with a 50% download budget with (I) No adaptation and (II) after applying our proposed adaptation method. *R_max* was set to 16 and the textures list was sorted.

5. CONCLUSIONS

In this paper 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. The 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.

In future work, we plan to use multiple priorities for textures and extend our approach to dynamic scenes that change over time. We also plan to more precisely estimate a download limit W based on the energy budget.

6. REFERENCES

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

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

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

[4] LTE advanced: The global 4G solution. http://www.qualcomm.com, Qualcomm, 2012.

[5] 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.

[6] N. Balasubramanian, A. Balasubramanian, and A. 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.

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

[8] A. Carroll and G. 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.

[9] D. Halperin, B. Greenstein, A. Sheth, and D. 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.

[10] R. C. Harvey, A. Hamza, C. Ly, and M. 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.

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

[12] M. Hosseini, D. T. Ahmed, and S. 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.

[13] M. Kennedy, H. Venkataraman, and G.-M. 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.

[14] J. Kim and M. Lee. *Green IT: Technologies and Applications*. Springer. ISBN-10: 3642221785, 1st edition, Jul 2011.

[15] K.-H. Kim, A. W. Min, D. Gupta, P. Mohapatra, and J. P. Singh. Improving energy efficiency of wi-fi sensing on smartphones. In *IEEE International Conference on Computer Communications (INFOCOM)*, pages 2930–2938. IEEE, July 2011.

[16] G. Perrucci, F. 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.

[17] H. 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.

[18] H. Rahimi, 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.

[19] A. Rahmati and L. 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.

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

[21] A. Steed and M. F. Oliveira. *Networked Graphics: Building Networked Games and Virtual Environments*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2009.

[22] T. 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.

[23] J. Ström and T. Akenine-Möller. PACKMAN: texture compression for mobile phones. In *ACM SIGGRAPH 2004 Sketches*, SIGGRAPH '04, pages 66–, New York, NY, USA, 2004. ACM.

[24] J. Ström and M. 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.

[25] L. Zhang, B. Tiwana, R. Dick, Z. Qian, Z. Mao, Z. Wang, and L. 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.