POWER MANAGEMENT FOR INTERACTIVE 3D GAMES

YAN GU

M.Eng.(Computer Science & Engineering), Zhejiang University, China

A THESIS SUBMITTED

FOR THE REQUIREMENT OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

SCHOOL OF COMPUTING

NATIONAL UNIVERSITY OF SINGAPORE

2008

List of Publications

1. Y. Gu and S. Chakraborty. Control theory-based DVS for interactive 3D games. In *Proc. 2008 Design Automation Conference (DAC)*, Anaheim, CA, USA, 8-13 June, 2008.

2. Y. Gu and S. Chakraborty. A hybrid DVS scheme for interactive 3D games. In *Proc. 14th IEEE Real-Time Technology and Applications Symposium (RTAS)*, St. Louis, MO, USA, 22-24 April, 2008. IEEE Press.

3. Y. Gu and S. Chakraborty. Power management of interactive 3D games using frame structures. In *Proc. 2008 International Conference on VLSI Design (VLSID)*, pages 679-684, HICC, Hyderabad, India, 4-8 January, 2008. IEEE Press.

4. Y. Gu. Power-aware gaming on portable devices. In *SIGDA Ph.D. Forum at Design Automation Conference (DAC)*, San Diego, CA, USA, 4-8 June, 2007.

5. Y. Gu, S. Chakraborty and W. T. Ooi. Games are up for DVFS. In *Proc. 2006 Design Automation Conference (DAC)*, pages 598-603, San Francisco, CA, USA, 24-28 July, 2006. ACM Press.

Acknowledgments

As I stand at the threshold of earning my doctorate, I am overwhelmed when I recall all the people who have helped me get this far.

First and foremost, I would like to thank my Ph.D advisor, Dr. Samarjit Chakraborty, for his constant support, guidance, and inspiration that a graduate student can expect from his advisor. Samarjit is a truly remarkable advisor who grants students a lot of freedom to explore new ideas, but at the same time advises on them. I myself have gained more confidence on my research ability from such role-model as a researcher. I look forward to continuing my association with him in the future. I am indebted to Professor Wynne Hsu, my primary advisor, for offering me the chance to go that far in my research career. Thanks to her, I have never been ended up without my doctorate.

There are two professors whom I would specifically like to thank for their valuable advices throughout the hard time in my PhD study: Professor Beng Chin Ooi – his continuously recommendation for my graduate study in NUS, Dr. Zhiyong Huang – his generously personal help for my first year in Singapore.

I have had very productive collaborations with Professor Akkihebbal L. Ananda, who also recommended me working as a research assistant at Department of Computer Science, NUS, Dr. Mun Choon Chan, Dr. Rajesh Krishna Balan and Anand Bhojan. I

have also had valuable discussions on various aspects of the thesis and the life with Dr. Holun Cheng, Dr. Ye Wang and Professor Roger Zimmermann.

I would like to thank my thesis committee, for providing insightful comments and constructive criticisms on the ideas presented in this thesis.

Thanks also are due to Ying Chee Woo and Chandra Mukaya from Department of Electrical and Computer Engineering, NUS, for their help in the initial set up for power measurement of laptops in the thesis work. As well, for Yong Jun Aw's help in the power measurement of PDAs.

My graduate student colleagues have made my stay at NUS a truly enjoyable one. I would like to thank Binbin Chen, Wendong Huang, Yicheng Huang, William Ku, Lin Ma, Yuan Ni, Xiuchao Wu, Hang Yu and Jie Yu. My graduate student career has also been enriched by interactions with several labmates, including Unmesh Dutta Bordoloi, Jimin Feng, Ramkumar Jayaseelan, Lei Ju, Yun Liang and Balaji Raman.

My gratitude goes out to all the staffs at school's workshop, graduate office, finance and human resource offices, especially to Madam Line Fong Loo, Madam Tse Wei Hee, Madam Hui Chu Lou, Madam Siew Foong Ho and Madam Michelle Yeo.

Moving towards more personal acknowledgements, I am, of course, particularly indebted to my husband, Luke, for his monumental, unwavering spiritual and material support and encouragement. He has truly always been there for me, and without him none of this would have been even possible.

Last, but definitely not the least, I would like to express my gratitude to my parents for being an unstinting source of support and encouragement. My parents have taught me through their courage in overcoming the challenges of life and have worked hard to provide me the very best of it. They have always been there when I have needed them.

Contents

Summary

Graphics-intensive computer games are now widely available on a variety of portable devices ranging from laptops to PDAs and mobile phones. Battery life has been a major concern in the design of both the hardware and the software for such devices. Towards this, dynamic voltage scaling (DVS) has emerged as a powerful technique. However, the showcase applications for DVS algorithms so far have largely been video decoding where the workload associated with processing different frames can vary significantly. It is unclear if DVS algorithms can be applied to games due to their interactive (and hence highly unpredictable) nature. Motivated by the existing work in video decoding applications and the increasing availability of game applications on portable devices, this thesis addresses the problem of power-aware gaming on portable devices, which to the best of our knowledge has not been studied before.

In this thesis, we investigate the workload characteristics of game applications and observe that interactive game applications exhibit sufficient workload variations, thereby, are highly amenable to DVS techniques. Specifically, we have two key observations for game applications, as illustrated in the following.

- Unlike video frames, game frames **cannot be buffered** due to the interactive nature, while buffering is exploited in many known DVS algorithms.

- Game frames offer more "**structure**" information than video frames (which only contain the I, B, or P frame-type information). More specifically, the workload associated with processing a game frame depends on the contents of the frame, or the constituent *objects*, which can be easily determined by parsing the frame.

Based on the above observations, we study several issues regarding the power-aware gaming on portable devices in the thesis. The relevant contributions are listed below.

1. Whereas video frames can be buffered, buffering is not possible in game applications. As a result, many control-theoretic mechanisms designed for video decoding applications by employing queue capabilities as the feedback in their control systems are not applicable to game applications. We design a DVS scheme by exploiting control-theoretic feedback mechanisms, which have not yet been explored in the context of games. In our *control theory-based* DVS scheme, the prediction error between the predicted and the actual game workload is fed back to the controller and used to regulate the workload prediction for next frame. This control theory-based DVS scheme performs better in terms of power saving and output quality than the known history-based schemes for game applications.

2. As we observe that the workload prediction for game applications should *not* merely rely on the processing time of previous frames. More specifically, the ”structure” information of constituting objects in game frames can be exploited to predict their workload. Towards this, we design a novel *frame structure-based* DVS scheme for game applications by parsing a frame, prior to it being actually processed. The obtained structure of the frame is then used to estimate the frame's processing workload.

3. Furthermore, we observe that the game workload exhibits different degrees of variabilities. For game plays where the frame workload exhibits sufficient variability, our frame structure-based prediction scheme works well (and outperforms control-theoretic prediction schemes). However, for the frames with relatively constant rendering workload, the proposed control-theoretic prediction schemes

happen to perform better. To take advantage of both these schemes, we propose a *hybrid* DVS scheme by switching between the two schemes based on their relative performance.

In summary, the above issues are concerned with three general problems related to power management for interactive 3D games on portable devices. Is the workload associated with game applications sufficiently variable so that DVS algorithms achieve significant power savings? How can the workload of game applications be predicted accurately so that they become amenable to DVS? How to design efficient DVS algorithms that can offer sufficient control over energy savings versus game quality tradeoffs? The results corresponding to these problems that are presented in this thesis demonstrate that our proposed schemes provide effective power management techniques for graphics-intensive 3D game applications on portable devices.

List of Tables

List of Figures

Chapter 1

Introduction

Computer games have recently experienced a sharp increase in popularity and have attracted considerable attention in both the industry and academia. They are driving a number of innovations in areas ranging from graphics hardware and high performance computer architecture to networking and software engineering. Although most of the graphics-rich games are still largely played on high-performance desktops, over the last couple of years, a number of games are also available on portable devices such as Personal Digital Assistants (PDA) (e.g www.doompda.com) and cellular phones. Playing games on portable devices running on battery brings more mobility to life. Since such devices are becoming increasingly popular and powerful, we believe that this trend will certainly continue in the coming years.

Energy efficiency is one of the most critical issues in the design of such battery-powered portable devices. These portable devices are usually facilitated with dynamic voltage and frequency-scalable processors. The availability of such processors on portable devices has led to power management schemes based on DVS algorithms. Since the power dissipated per cycle with CMOS circuity scales quadratically to the supply voltage and linearly to the frequency ($P \propto f \cdot V^2$), DVS can potentially provide a very

large power saving through voltage and frequency scaling. DVS algorithms have been shown to be useful to reduce power consumption for a variety of application scenarios, such as audio [7] and digital signal processing applications [8].

Specifically, over the last few years, such algorithms have been very successfully applied to video encoding/decoding applications which are also computationally expensive and where the workload associated with processing different frames can vary significantly (for example, see [1, 11, 26, 56]). The basic principle behind most of these algorithms is to predict the workload associated with processing a video frame from the workloads of the previously decoded frames. The voltage/frequency of the underlying processor is then scaled based on such history-based workload predictions. This basic scheme has also been refined using control-theoretic feedback mechanisms, where previous prediction errors are taken into account while estimating the workload of a current frame [32, 41, 53, 54].

The main differences between games and video decoding applications stem from (i) the ***interactive*** nature of games, (ii) unlike video frames, game frames **cannot be buffered** (buffering is exploited in many DVS algorithms [27, 53, 54]), (iii) game frames are more "**structured**" than video frames (which only contain the I, B, or P frame-type information). More specifically, the workload associated with processing a game frame depends on the contents of the frame, or the constituent *objects*, which can be easily determined by parsing the frame. Similar conclusions were also arrived at [36, 37] for workload characterization of a 3D graphics processing pipeline.

Although DVS algorithms have been extensively applied to video encoding/decoding applications (which have almost attained the status of the *dining philosophers problem* in this domain) [38, 55, 56], their use in graphics-intensive games has not been suffi-

ciently explored so far. Motivated by the abovementioned line of work and the increasing availability of game applications on portable devices, this thesis addresses the issue of power management for interactive games. In the thesis, we investigate the workload characteristics of game applications. The sufficient workload variations indicate that the interactive game applications are highly amenable to DVS techniques. However, existing control theory-based DVS schemes from video applications employ queue capabilities as the feedback in their control systems. As we know, there is no buffering in game applications. Therefore, these control-theoretic feedback schemes exploiting queue capabilities are not applicable to game applications. In this thesis, we design a novel control theory-based workload predictor in DVS scheme for game applications. Further, based on one of our key observations that game frames offer more ”structures” than video frames, we present an innovative DVS scheme for game applications by exploiting frame structure. The emergence of different degrees of variabilities in game workload motivates our hybrid DVS scheme by combining the frame structure and the control theory techniques for game applications. All the above mentioned DVS schemes are evaluated on a laptop and a PDA, with simulation setting, real platforms such as Windows and Windows Mobile.

Before elaborating on our work in the thesis, we would like to introduce the design of a game engine, which is the reusable core of a game application. By adding details (which are often referred to as ”assets”) like models, animation, sound and story to a game engine, a (concrete) game is derived.

1.1 Anatomy of a Game Engine

A game engine runs in an infinite loop, where the body of this loop consists of tasks responsible for processing a single frame. This loop body is shown in Figure 1.1. Here *Event* denotes the user inputs or interactions with the game, which along with the current state of the game is used to generate the next frame to be displayed. This involves two sequential steps—computing and rendering—which we describe below. A more detailed discussion may be found in [6, 50].

The computing step comprises tasks such as collision detection, AI, simulation of game physics and particle systems. Collision detection includes algorithms for checking collisions between the different objects and characters in the game. Such algorithms compute intersections between two given solids, their trajectories as they move, impact times during a collision and their impact points. In some engines, the AI tasks determine the movement of the characters in the game. Game physics incorporates physical laws into the game engine so that different effects (e.g. collisions) appear more realistic to a player. Typically, simulation physics is only a close approximation of real physics, and computation is performed using discrete rather than continuous values. Finally, a particle system model allows a variety of other physical phenomenon to be simulated. These include smoke, moving water, blood, explosions and gun fires. The number of particles that may be simulated is typically restricted by the computing power of the machine on which the game is being played.

The rendering step involves algorithms to generate an image (or a frame) from a model, which is then displayed as shown in Figure 1.1. In this case, the model is typically a description of several three dimensional objects using a predefined language or data structure. It consists of geometry, viewpoint, texture and lighting information.

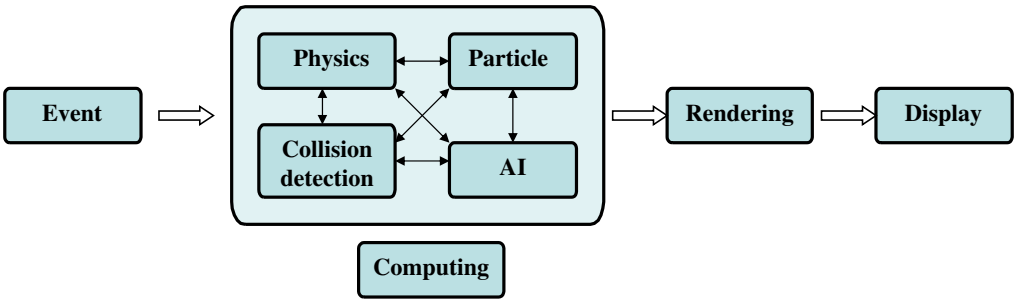


Figure 1.1: Frame *processing* in a game application.

In the case of 3D graphics, rendering may be done offline, as in pre-rendering, or in real time. Pre-rendering is a computationally intensive process that is typically used for movie creation, while real-time rendering is commonly done in 3D computer games, which often relies on the use of a specialized processor called a Graphics Processing Unit (GPU).

The rendering step involves two stages: *geometry stage* and *rasterization stage*. Each stage is pipelined as shown in Figure 1.2. The geometry stage performs per-vertex operations such as vertices transformation of solid objects to the screen space, lighting, texture coordinates generation and deletion of invisible pixels by clipping. The processed vertices are assembled into primitives and sent to the rasterization stage. The rasterization performs per-pixel operations, from simple operations such as writing color values into the frame buffer, to more complex operations such as texture mapping, depth buffering and alpha blending. The outcome of these steps is the transformation of 3D data onto 2D screen.

The first step in the rasterization is to decide whether a pixel is to be rendered or not. To determine whether a pixel is within a triangle, the most popular of algorithm is the *scanline algorithm* [1]. Scanline rendering is an algorithm for visible surface determination, which works on a row-by-row basis. Firstly, all of the polygons to be rendered

[1] http://en.wikipedia.org/wiki/Scanline_algorithm

are sorted according to their top y coordinates. Secondly, by using the intersection of a scan line with the polygon on the top of the sorted list, each row of the polygon is computed.

To determine whether a pixel is occluded or blocked by another pixel, a z *buffer* is used to ensure that the pixels close to the viewer are not overwritten by pixels far away. The z buffer is a 2D array corresponding to the image plane which stores a depth value for each pixel. Whenever a pixel is drawn, it updates the z buffer with its depth value. Any new pixel need check its depth value against the z buffer value before it is drawn.

Next, the rasterization need find out a pixel's color, texture and shading information. A *texture map* is a bitmap that is applied to a triangle to define its look. Each vertex of a triangle is associated with a texture color information and a texture coordinate (u, v) in 2D space. Whenever a pixel on a triangle is rendered, the corresponding *texel* [2] (the texture is represented by arrays of texels) in the texture must be found. This is extrapolated from the distance between the triangle's vertices and the rendered pixel.

Lighting effect on the pixel is taken into account to determine its resultant color. Generally, there are three types of lighting effects [2], i.e. *directional lights*, *point lights* and *spotlights*. Directional lights come from a single direction and have the same intensity throughout the entire scene. Point lights are the lights with a definite position in space and radiate light evenly in all directions. The point lights in real life experience quadratic attenuation in the intensity of light incident on objects farther away. Spotlights are the lights with a definite point in space, a direction, and an angle defining the cone of the spotlight.

The last step in the rasterization is *shading* [2]. The shading algorithm accounts for the distance from light and the normal vector of the shaded object with respect to the in-

[2] http://en.wikipedia.org/wiki/Texel_(graphics)

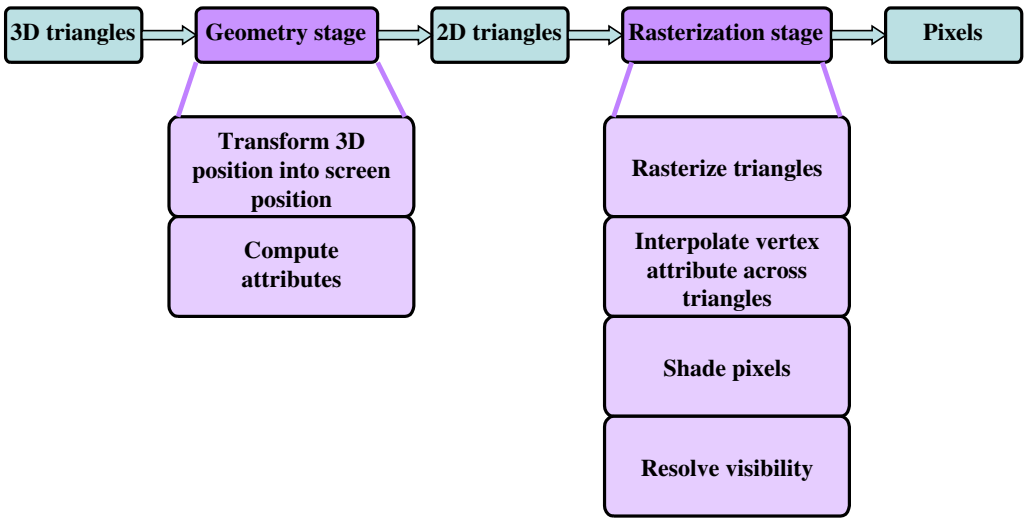


Figure 1.2: The diagram of rendering pipeline.

cident direction of light. The fastest algorithm is *flat shading*, in which all pixels on any given triangles are assigned with a constant lighting value. There is other algorithm – *Gouraud shading*, which separately shades vertices and interpolates the lighting values for the rendered pixels. The slowest and most realistic approach is *Phong shading*, in which the lighting value for each pixel is computed individually, by performing bilinear interpolation of the normal vectors.

Rendering is computationally expensive and occupies a significant fraction of the total processing time of a frame. The most significant component of the rendering task involves the rasterization stage.

In the next section, we explore the possibility of lowering the frame-rate of a game, thereby reducing its processing workload. This reduction would enable the game application to run at a constant, but lower processor frequency, thereby reducing power consumption. Although this approach would be a competing approach to DVS, we discuss what are its disadvantages and why dynamically changing the processor's voltage/frequency might be better in the case of games.

1.2 A First Cut: Reducing Frame Rates

A rule of thumb in game design is that users prefer high frame rates. As a result, most game applications are designed to maximize frame rates without any consideration towards resource usage or power consumption. The loop described in Section 1.1 therefore runs at the maximum possible rate and fully utilizes the available CPU bandwidth. We measure the CPU usage of Quake II running on an IBM laptop using Intel VTune Analyzer 7.2 [3] and notice that it occupies 95% of the CPU bandwidth on an average as shown in Figure 1.3, all through 60 second. However, the frame rate varies over time and depends on the state of the game (e.g. the number of characters and the complexity of the scene).

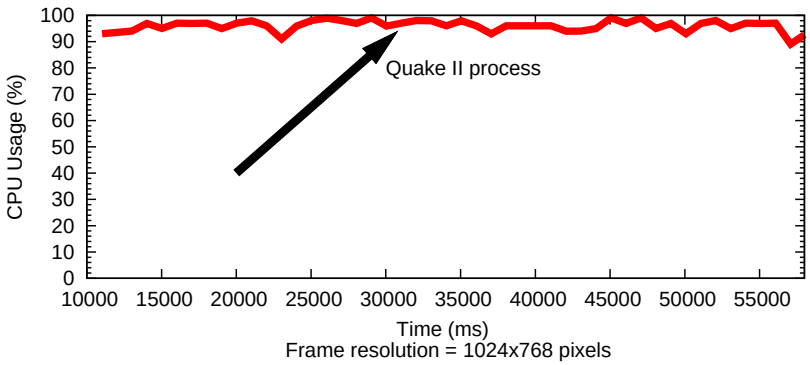


Figure 1.3: Quake II occupies 95% CPU bandwidth.

A recent study [12] on the effects of frame rates and resolution in First Person Shooter games concluded that although frame rates have a significant impact on the perceived quality-of-service, for most parts of a game very high frame rates are not required. More specifically, the resulting frame rate when a game application fully utilizes the CPU bandwidth might be unnecessarily high. As a result, a natural question that comes up is: Why not run the game at a *constant* (but lower) frequency?

It turns out that this is not a good strategy, because the variation in the number of

[3] http://www.intel.com/cd/software/products/asmo-na/eng/vtune/vpa/index.htm

processor cycles required to process different frames is considerably high, as we show in Chapter 4. While running the CPU at a constant but lower frequency would reduce the overall frame rate, the rate might drop below the tolerable range when rendering complex scenes. Before we present the results supporting this observation, let us briefly outline the experimental setup that we use throughout this thesis.

1.2.1 Experiments

We conducted all our experiments on an IBM laptop with a 1400 MHz Intel Mobile Processor built with Speedstep^{TM} technology, and an ATI Radeon^{TM} Mobility Video card. The CPU supports five different frequency operating points: 1400, 1200, 1000, 800 and 600 MHz. All our results are based on the ”vanilla” Quake II, version 3.21, whose source code is instrumented and compiled to run on Windows XP.

To ensure that the game is not preempted by other processes, we ran it with the highest priority and rendered the game with the ”software” option. The ”software” option disables the use of the GPU, causing the 3D functions to be executed on the CPU. This option uses *DirectDraw* to draw the pixels on the screen. Sounds are disabled during measurements, as our initial results show that the workload in loading and playing audio during games is negligible (approximately 1.8% of the total workload). All the processor cycle measurements are carried out using *RDTSC* (read time-stamp counter) instruction. We choose to use a software-only renderer as many battery-powered personal mobile devices such as (low-end) laptops, PDAs and mobile phones do not support GPUs yet. As investigated, a wide range of available PDAs do not provide hardware accelerators for graphical tasks [4], and we believe the market will continue for some time. In the future, it might be more PDAs and mobile phones facilitated with

[4] http://www.brighthand.com/best_pdas/default.asp?display=mostpopular

GPUs. However, the power management techniques that will be supported by GPUs is not clear at this stage. In conclusion, our proposed techniques hold today for most of PDAs and they could be applied to future portable devices as well.

To ensure reproducibility, instead of actually playing the game, we replayed pre-recorded demo files in Quake II. The game resolution is set to 1024×768, running in full-screen mode. While replaying demos allows us and the research community to repeat our experiments, the workload measured is slightly lower than the workload incurred by games played in real-time. The difference arises from the fact that, the demo has certain pre-recorded states (such as position of objects in each frame and input from users) and therefore these states are not computed again during playback. Our experiments suggest that this computation accounts for approximately 3% of the total workload of the game.

For power measurements, we removed the battery from the laptop and connected it to the external power supply using an AC power adapter. We then tapped the cable leading from the power adapter to the laptop using special probes connected to a National Instruments PXI-4071 $7\frac{1}{2}$-digit Digital Multimeter which measure the instantaneous current and voltage drawn by the laptop.

Figure 1.4 shows an excerpt of how the instantaneous frame rate varies with time for replaying the default Quake II demo with the processor frequency set to the five supportive levels. We measured the instantaneous frame rate as the reciprocal of the frame processing time. Note that with 1400 MHz, the frame rate varies between approximately 35 and 95 frames per second (fps). With 600 MHz, the frame rate varies roughly between 5 and 55 fps. With frequencies set to five levels between 600 and 1400 MHz, the frame rates are shown from lowest to highest respectively. A frame rate

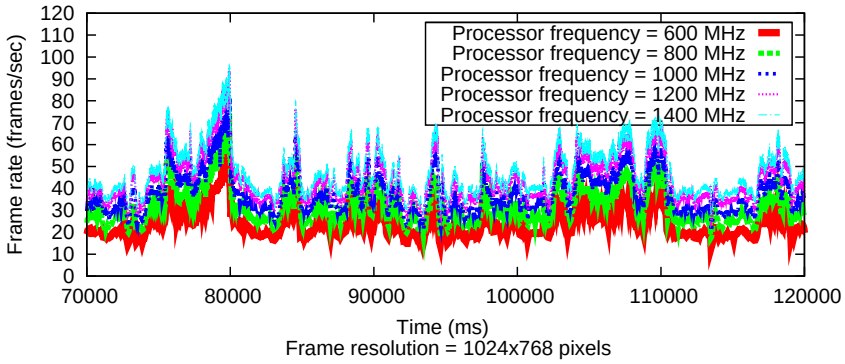


Figure 1.4: Resulting frame rates when the processor frequency is set to five supportive levels.

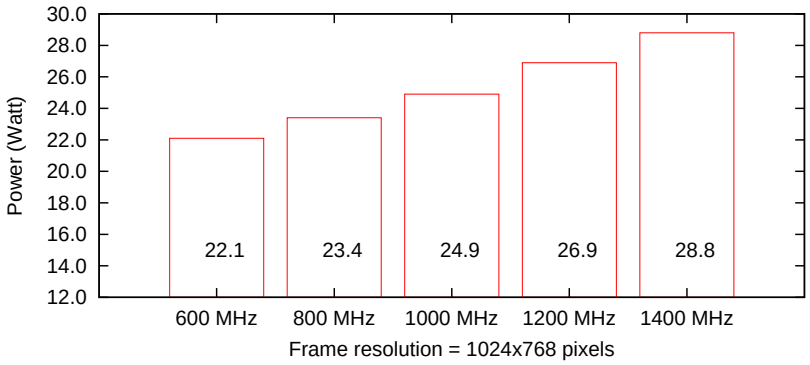


Figure 1.5: Average power consumption for different processor frequencies.

of 95 fps is much higher than necessary [12]. On the other hand, if we run the processor at a constant frequency of 600 MHz, we achieve undesirably low frame rates on certain frames exhibiting complex scenes. The average power consumptions corresponding to the five frequency values supported by our laptop with the game running on it are shown in Figure 1.5. We observe that the power consumption decreases correspondingly to the frequency. We computed these values by recording the instantaneous current $c(t)$ and voltage $v(t)$ drawn by the laptop every 5 ms, and calculating the power consumption over a duration of length T as $\sum_{t=0}^{T}(c(t)v(t)\delta t)/T$, where δt is the sampling interval (5 ms). Note that these values correspond to the total system power consumption and not the power consumed by the processor alone.

The first attempt to reduce frame rates by running a processor at a lower but constant

level leads to lower frame rates on certain frames with large game workload demands, albeit it reduces power consumption. On the other hand, this *constant* frequency scaling results in unnecessarily higher frame rates on certain frames with small workload demands. In contrast, dynamically scaling frequency to match required game workload could guarantee better frame rates with more power saving than the naive constant frequency scaling. In the thesis, we study the following three problems related to the issue of power management for interactive 3D games on portable devices.

- *Is the workload associated with game applications sufficiently variable so that DVS algorithms achieve significant power savings?*

The unpredictable interaction from game players incurs different game workload associated with variable constituent objects. From first point of view, it is unclear whether game applications are amenable to DVS or not. In this thesis, we show using detailed experiments that interactive games are highly amenable to DVS. We elaborate on this issue in Section 1.3.1.

- *How can the workload of game applications be predicted accurately so that they become amenable to DVS?*

As explained above, the nature of game applications is very different from video decoding applications, our finings of game workload in the first problem lead to a number of innovative DVS algorithms targeted towards game applications, exactly as video decoding applications have motivated a variety of schemes for DVS. In this thesis, we present three innovative DVS schemes towards interactive games. Section 1.3.2, 1.3.3 and 1.3.4 explain our proposed DVS algorithms in detail. To the best of our knowledge, it is the first time that DVS techniques have been applied to games.

- *How to design efficient DVS algorithms that can offer sufficient control over energy savings versus game quality tradeoffs?*

We are concerned with several critical issues regarding hardware and system, in the implementation of proposed DVS algorithms on multiple platforms. Section 1.3.5 elaborates our design of the mechanisms to address such issues and validates our design on multiple real platforms (e.g. laptops, PDAs).

1.3 Thesis Contributions

We designed power management techniques for graphics-interactive 3D games on portable devices. The results derived from different platforms show the consistently superior performance of our schemes, compared with known DVS algorithms designed for video decoding applications. Parts of work reported in the thesis have been published in [23, 19, 20, 22, 21].

1.3.1 DVS for Game Applications

We initiated a study of applying DVS technique for game applications in [23]. By carrying out detailed experiments using an open source, popular Fist Person Shooter game called Quake II, we observed that game applications exhibit sufficient variability in their workload to meaningfully exploit DVS schemes for power savings. Moreover, our investigation offers the possibility of developing DVS algorithms that better exploit the characteristics of game applications (compared to those that have been developed for video decoding applications).

1.3.2 A Control Theory-based DVS Scheme

One of the primary differences between video processing and game applications is the interactive nature of games. Whereas video frames can be buffered, buffering is not

possible in game applications where the content of a frame is dependent on the user input. As a result, many of the control-theoretic feedback mechanisms that were developed for predicting the workload of video processing applications (e.g. see [53, 54]) cannot be applied to games.

We investigated the use of such control-theoretic feedback mechanisms for dynamic voltage scaling for interactive 3D game applications in [20]. Such mechanisms have not yet been explored in the context of games, and more importantly, the buffer-centric approaches for workload prediction cannot be applied in this context.

We used a proportional-integral-derivative (PID) controller to predict the processing workload of a game frame. Following standard control theory terminology [28], the predicted processing workload of a frame was set as the *measured variable* and the actual workload (obtained after rendering the frame) was considered to be the *set point*. The resulting prediction error (i.e. the difference between the predicted and the actual workload) was fed back to the PID controller and was used for predicting the workload of the next frame. The predicted frame workload was taken to decide the voltage/frequency level of the processor.

The tunable parameters in the PID controller could be manually adjusted towards specific applications or automatically selected by available softwares. This scheme has negligible computational overhead, owing to the discrete formulation of the PID controller.

1.3.3 A DVS Scheme by Exploiting Frame Structure

Furthermore, we observed that the nature of game workload is very different from those arising from video decoding applications in [23], which motives the need for different

DVS schemes compared to the ones traditionally used for video decoding. In the case of game applications, the frames contain ”structure” which can be exploited to predict their workload or processor cycle requirements. While processing a frame, the workload depends heavily on the scene that the frame is depicting. More specifically, the workload depends on the content of the frame or the constituting *objects* that need to be processed.

Towards this, we designed a more efficient DVS algorithm for game applications by exploiting the ”structure” information (e.g. number of brush and alias models, textures and light maps information, number of particles) of game frames in [22]. By parsing a frame, prior to it being actually processed, the structure of the frame, or the constituting *objects* that need to be processed is efficiently obtained, which is then used to estimate the frame's processing workload. The predicted frame workload is used to decide whether the voltage/frequency of the processor should be scaled or not.

Compared with the control theory-based DVS scheme, this scheme incurs more computational overhead due to the parsing of game frames. However, this scheme could be extended and generalized to other game applications without losing the accuracy of workload prediction.

1.3.4 A Hybrid DVS Scheme

We observed that our frame structure-based prediction scheme works well (and outperforms control-theoretic prediction schemes) for game plays where the frame workload exhibits sufficient variability. However, for the frames with relatively constant rendering workload, the proposed control-theoretic prediction schemes happen to perform better. To take advantage of both these schemes, we proposed a *hybrid* workload prediction

scheme in [21], where we kept on switching between the two schemes based on their relative performance.

The hybrid prediction scheme combines two different techniques: (i) adjusting workload prediction by control-theoretical feedback mechanism, and (ii) analyzing the graphical objects in the current game scene by parsing the corresponding frame.

We evaluated the performance of the proposed control-theoretic DVS scheme, the frame structure-based DVS scheme and the hybrid DVS scheme by comparing with the known history-based DVS algorithms for interactive games. Our results derived from different platforms consistently show that there are significant improvements of our proposed DVS schemes, based on the data from the full-blown Quake games. The hybrid DVS scheme achieves the best performance in power saving and output quality; and its prediction overhead is within a feasible region.

1.3.5 Implementation on Multiple Platforms

In this thesis, we are concerned with frequency mapping and frequency transition overhead on the performance of DVS algorithms.

A number of previously-proposed algorithms for DVS have assumed the processor's frequency range to be continuous (e.g. see [32]). However, most voltage/frequency-scalable processors only support a fixed number of discrete frequency levels. Hence, in the thesis we assume that only a fixed number of frequency levels are available and the computed optimum frequency is mapped onto the next available higher frequency level. Such a conservative mapping satisfies the workload demands of the game application, at the cost of less than ideal energy savings. However, we also conduct simulations where we assume that the processor's frequency is continuously scalable.

Frequency switching of a processor is associated with an overhead which depends

on the processor's microarchitecture as well as the OS running on top of it. Our experimental results suggest that for the same processor, this overhead is higher in Windows XP compared to Linux. The average transition overhead in Windows XP running on an Intel Pentium Mobile processor is 20 million cycles, i.e., the overhead is 14 milliseconds with the operating frequency set to 1400 MHz.

Hence, to skip unnecessary frequency switches, we use a *lazy* transition mechanism. Instead of immediately switching the processor frequency whenever the predicted workload of a game frame changes, we defer the switch to the immediate next frame.

Apart from the evaluation of our proposed DVS schemes on a configurable simulation platform, we conduct the experiments on two heterogenous platforms: a laptop with Intel Pentium Mobile processor facilitated with SpeedStep™, running Windows and a PDA with Intel XScale processor, running Window Mobile. Their consistent results derived from the platforms enforce that our DVS schemes are applicable to different configurations, regardless of the underlying hardware and the operating system.

The measurements on the laptop are conducted based on Quake II engine for several reasons. It is a representative game that can be played on current, general purpose portable devices without hardware accelerators, such as low-end laptops, PDAs and mobile phones. The game engine of Quake II is the basis of other popular First Person Shooter games. Here, we would like to clarify that a game engine is the reusable core of a game applications. By adding details (which are often referred to as "assets") like models, animation, sound and story to a game engine, a (concrete) game is derived. Since our experimental results are based on Quake II, they immediately extend to other First Person Shooter games (e.g., Heretic II (1998), SiN (1998), Kingpin: Life of Crime (1999)) derived from the same game engine.

In addition, the results and conclusions of Quake II on the laptop are in line with Quake[5] on the PDA. Quake is an earlier version of Quake II. The structure of Quake engine involves the same game objects and the processing of game tasks follows the same logic as Quake II. Unfortunately, the high computational workload by Quake II results in unacceptable low frame rates on a PDA (less than 5 frames per second), thereby deteriorated the game quality. Thus, in the thesis, we adopt portable Quake instead of Quake II on the PDA without tampering comparability of results.

1.4 Organization of Thesis

The rest of the thesis is organized as follows. Chapter 2 reviews some prior work of graphics workload characterization and DVS algorithms in video decoding applications. Chapter 3 presents a DVS scheme by using control theory. Chapter 4 introduces a framework of workload characterization for game applications, followed by a DVS scheme exploiting frame structure information. The exhibition of game frames with a large degree of workload variability leads to a hybrid DVS scheme in Chapter 5. Chapter 6 shows the results of our proposed DVS algorithms on different platforms – a laptop and a PDA. Finally, some potential directions of this study are discussed in Chapter 7.

[5] http://quake.pocketmatrix.com/

Chapter 2

Previous Work

In this chapter, we discuss some prior work on workload characterization of 3D graphics and introduce the existing work of DVS techniques, mostly towards video decoding applications, finally, the latest work on power management for 3D graphics.

2.1 Workload Characterization of 3D Graphics

Mitra and Chiueh [34] discussed the bandwidth and memory requirements of rasterization workload in graphics hardware. First, they considered the bandwidth requirement of geometry information transferred between the CPU and the graphics hardware, over a high-speed system bus such as PCI. By demonstrating the variable requirements of *triangles*, *pixels*, *spans* and *pixelstamps* in each frame in different stages of rasterization processing, they suggested sufficient FIFO buffers between different stages of the pipeline are used to absorb the variation without introducing stalls. Second, they discussed the behavior of memory access in rasterization and proposed to improve the locality of frame buffer access by changing the pixel generation order during scan conversion. For texture, they investigated the effect of texel block and caching to the efficiency of texture memory access.

Their work mainly investigates the bandwidth and memory requirements of rasterization on system architecture, thereby, presents some implications for graphics pipeline, frame buffer design, texture memory management and system bus design. However, they did not discuss the workload of game applications on the processor. Their implications are hardly applicable for DVS algorithms.

Wimmer and Wonka [52] considered graphics pipeline as a parallelled rendering process, in which the CPU and the GPU perform tasks in parallel. Since the rendering tasks on the GPU constitute the most important factors for the rendering time, they proposed several heuristics to calculate the rendering time estimation functions.

The *view-cell sampling* method works for a view-cell based system, where a potentially visible set (PVS) is stored for each view cell. For each view cell, they discretized the set of view directions, randomly generated n views around each discretized direction and measured the rendering time for each view. The maximum rendering time of the n sample views is used as an estimation for the total rendering time of the direction and the view cell under consideration.

Another *per-object sampling* method estimates the rendering time of a set of objects by adding the rendering time estimations of the individual objects. To estimate the rendering time of a single object, they parameterized the rendering time estimation function by three angles. The first angle is the angle between the two supporting lines on the bounding sphere. This angle (which is related to the solid angle) is an estimate for the size of the screen projection. The other two angles (for elevation) describe from which direction the object is viewed. In a preprocess, they sampled this function using a regular sampling scheme and stored the values in a lookup table together with the object.

Their sampling for each object rendered on the GPU incurs huge computational overhead in their proposed *per-object sampling* method. The situation is deteriorated when varying objects occur in the scene. Such overhead is prohibitive in interactive applications such as games, as usually there are thousands of visible objects in game scenes.

The most recent work of workload characterization was done by Mochocki et al. [36, 37]. They discussed that quality factors of 3D graphics, such as resolution, LOD, lighting model and texture models affect power consumption on portable devices. They simplified graphics pipeline with a simulator and developed a *signature buffer* in the simulator. From the signature buffer, they obtained specific parameters which imply graphics workload: (i) average triangle area, (ii) triangle count, (iii) average triangle height, and (iv) vertex count. These parameters are concatenated as a signature and used to find a closest matching signature from a signature table. The corresponding workload is read from the table as the predicted workload.

2.2 Dynamic Voltage and Frequency Scaling for Video Applications

Recently, power management techniques with DVS for mobile devices have received increasing research attention [31, 35, 40, 46, 55]. Those techniques exploit an important characteristic of CMOS-based processors: the frequency scales almost linearly to the voltage, and the power dissipated per cycle scales quadratically to the supply voltage and linearly to the frequency. DVS algorithms have been applied for a variety of application scenarios, such as audio [7], digital signal processing applications [8].

Specifically, lately there is a large body of existing work devoted to DVS algorithms for video decoding applications. These work can be broadly classified into (i) history-based approaches, by exploiting historical information of applications, (ii) control theory-based approaches, in which previous output of control system is fed back to the system to adjust its performance, (iii) off-line meta-data approaches, by analyzing pre-recorded sequences of video applications.

2.2.1 History-based Approaches

The basic history-based prediction is to monitor the system utilization of previous uniform-length intervals, which is extrapolated as the voltage level for the next uniform-length interval. This idea can be extended by weightedly averaging actual workload of previously processed frames [15, 39].

Weighted Averaging

Weiser et al. [51] and Govil et al. [15] are among the first researchers who proposed DVS algorithms in operating systems.

In 1994, Weiser et al. first proposed an interval-based DVS algorithm to monitor CPU utilization constantly on a general purpose operating system. Processor frequency and voltage are adjusted at the beginning of each interval according to the CPU utilization of previous execution traces.

Govil et al. compared five DVS algorithms by assigning weights in different manners in [15]. For instance, in their *LONG_SHORT* algorithm, more weight is assigned to short-term historical intervals than long-term historical intervals. Therefore, it attempts to smooth behavior to a global average, but shows more concern for local peaks.

Aydin et al. discussed a series of algorithms in [3, 4], which anticipated early completions of future execution by using the average-case workload information.

Pering et al. proposed a weighted averaging scheme in [39]. In their scheme, a weighting factor α is used to filter out severe fluctuations of workload and achieve smaller average prediction error. The estimated workload of next interval $t+1$ is defined as $Workload_{avg}(t+1)$, which is predicted by extrapolating from actual and estimated workload of interval t, shown as follows.

$$Workload_{avg}(0) = Workload(0) \tag{2.1}$$

$$Workload(t+1) = \alpha \cdot Workload(t) + (1-\alpha) \cdot Workload_{avg}(t) \tag{2.2}$$

$$= \alpha \cdot \sum_{\tau=0}^{t} (1-\alpha)^{\tau} \cdot Workload(t-\tau) \tag{2.3}$$

As Pering et al. examined DVS algorithms through trace-driven simulation. Grunwald et al. further evaluated DVS policies through physical measurements [18].

Chandrasena et al. [9] incorporated the strengths of the conventional workload averaging technique with the *rate selection algorithm*. System workloads are buffered to estimate the CPU rate until the scaling factor matches the system quantized rates.

Averaging by Exploiting Frame Information

As introduced above, the weighted DVS algorithms smooth workload prediction depending on global and local behaviors of applications. They could be further refined by exploiting the frame information (e.g. frame type) of video decoding applications [5, 11].

Bavier et al. proposed a workload predictor by concerning with frame type and size of MPEG data in [5]. In their empirical study of MPEG data, they observed that there

is a linear model with least square algorithm ($R^2 = 0.97$) between actual and predicted workload. However, it is computational expensive to apply least square algorithm incrementally for the applications. Therefore, they proposed a *FRAME_TYPE_LEN* predictor to approximate the model. This predictor estimates processing frame time by averaging frame time of previously processed frames with same type. At the same time, it adjusts its prediction by taking into account the linear increment of cycles with number of bytes. Such foreknowledge about the linear model is used to eliminate those misleadingly predicted points.

Choi et al. proposed a frame type-based workload averaging scheme in [11]. They categorized processing steps of each type of video frames (i.e. I, P or B frame) into two phases. The first phase includes IDCT and reconstruction steps, whose processing time varies with frame type. Therefore, this phase is defined as *frame-dependent* (FD). The second phase mainly involves dithering, whose processing time is constant regardless of frame type as discussed in [42]. Hence, this phase is defined as *frame-independent* (FI). In their DVS algorithm, the workload of FD phase for a specific frame type is estimated by averaging actual workload of previously processed FD phases with same type. The misprediction by averaging for FD phase is compensated by taking FI phase into appropriate level of frequency.

Stochastic Model of Workload Distribution

There are some other work in DVS for video decoding applications, in which they profile workload for a much longer duration, thereby, provide more insights on workload distribution than the above introduced averaging approaches [31, 16, 17, 13].

Lorch and Smith estimated workload distribution in parametric and nonparametric

methods in [31]. Based on the probability distribution of workload requirements, they proposed an optimal scheduling for video decoding applications by assuming continuous frequency scaling.

Gruian [16, 17] combined off-line and on-line scheduling at both task level and task-set level. Stochastic data derived from previous task execution traces are used to produce energy-efficient schedules. Multiple frequency levels may be assigned to a single task.

Liu et al. [30] defined the *available cycle function (ACF)* and the *required cycle function (RCF)* to capture the CPU workload of the real-time tasks. The values of these two functions are derived from the parameters of the task set and the scheduling policy used to dispatch the tasks of the system. In their experiments, the execution time of each job is generated using a normal distribution. The functions capture the workload envelope, thereby, derive the optimal clock speed between an upper bound and a lower bound of processor cycles.

The history-based DVS algorithms estimate workload by exploiting historical information (short-term or long-term). They are effective in applications such as video decoding or audio, where workload exhibits rather less and small variations.

However, the underlying principle of these algorithms makes them inapplicable for workload with a large degree of variability, such as game applications. Because these algorithms cannot track workload variations, output quality of game applications with fluctuant workload will be severely deteriorated.

2.2.2 Control Theory Approaches

The control theory approaches bring in another element – control theory to DVS algorithms for video decoding applications.

Feedback Control

The history-based approaches have been refined using control-theoretic feedback mechanisms. Some of control-theoretic mechanisms use the buffer occupancies as the primary input to the feedback controller [32, 33, 53, 54]. Some of work predict workload by exploiting proportional-integral-derivative (PID) controller for DVS scheduling [47, 49, 58].

Lu et al. presented a feedback-control scheduling algorithm for DVS in [32, 33]. The decoding application is modelled as $M/M/1$ queue system, where the following equation holds:

$$T = \frac{1}{\mu - \lambda} \tag{2.4}$$

where T is the average delay for the frame, μ is the frame decoding rate and λ is the frame arrival rate. The controller system is modelled by using first order Taylor expansion about the desired decoding rate μ:

$$T(k+1) = T(k) - T_0^2 * (\mu * \gamma(k+1) - \mu * \gamma(k)) \tag{2.5}$$

where T_0 is the user-specified delay and $\gamma(k+1)$ is the frequency factor (a value between 0 and 1) at time $k+1$. Based on the difference between the user-specified delay and the actual average delay from the previous frames, the controller will produce a new frequency factor, which is used to scale voltage/frequency.

Wu et al. modelled multiple-clock-domain processors as queue system, in which queue occupancies are used to trigger DVS algorithm [53, 54]. To be more specific, their controller monitors two queue signals: the relative queue occupancy $(q_i - q_{ref})$ and the relative queue difference $(q_i - q_{i-1})$, where q_i represents the queue occupancy at the i-th sampling point, and q_{ref} is the target queue occupancy. The continuous-time model of their DVS control is expressed as follows:

$$\dot{f}(t) = \frac{m}{h(f)} \frac{step}{T_{m0}} (q(t) - q_{ref}) + \frac{l}{h(f)} \frac{step}{T_{l0}} \dot{q}(t) \tag{2.6}$$

where $\dot{f}$ is the time derivative of normalized frequency f (i.e. $\partial f / \partial t$); $\dot{q}$ is the time derivative of queue occupancy; $step$ is the step size of the frequency change triggered by the queue signals; T_{m0} and T_{l0} are the basic time delays for the queue signals $(q_i - q_{ref})$ and $(q_i - q_{i-1})$ respectively; m and l are constants; $h(f)$ is a function of f which is used to take account of possible affects of f on the effective time delay.

Some researchers proposed PID controller for DVS scheduling in video decoding applications. A PID controller consists of three different elements, namely, proportional control, integral control and derivative control. The typical form of a PID implementation is expressed as follows:

$$y(t) = K_P \cdot x + K_I \cdot \int_0^t (x - y) dt + K_D \cdot \frac{dx}{dt} \tag{2.7}$$

in which y is the output of the controller at time t, and x is the input at time t.

Stankovic et al. presented a scheduling paradigm by using feedback control in [47]. They chose PID controller as the basic feedback control technique in the scheduling. The system deadline miss ratio $MR(t)$, i.e. the percentage of tasks that miss their dead-

lines at time t, was defined as the *measured variable*. $MR_s(t) = 0$ was defined as the *set point*, then $error = MR_s(t) - MR(t) = -MR(t)$. The PID controller computes the control $\Delta CPU(t)$ in requested CPU utilization with the following formula:

$$\Delta CPU(t) = -K_P \cdot MR(t) - \frac{1}{K_I} \cdot \sum_{IW} MR(t) + K_D \cdot \frac{MR(t) - MR(t - DW)}{DW} \quad (2.8)$$

$\Delta CPU(t) > 0$ means that the requested utilization should be increased, otherwise the requested utilization should be decreased.

Varma et al. proposed a modified PID controller upon known DVS scheduling algorithms in [49]. In their work, they defined the actual and predicted workload at n-th interval as x_n and y_n. By applying the modified PID controller, they obtained the predicted workload y_{n+1} by using the previous m values of workload as follows:

$$y_{n+1} = K_P \cdot x_n + K_I \cdot \sum_{n-m+1}^{n} \frac{x_i}{m} + K_D \cdot (x_n - x_{n-1}) \quad (2.9)$$

where K_P, K_I and K_D are constants. Compared with Formula 2.7, the modified PID controller is converted from continuous-time formulation to discrete formulation, by replacing the integral operation with a summation, replacing the derivative operation with the difference between the current and the previous actual workload and eliminating the feedback term y in the loop.

Zhu et al. proposed a EDF scheduling algorithm based on a PID controller in [58]. In their work, they predicted worse-case execution cycles $C_{A_{ij}}$ for the j-th job of a task

T_i using the following discrete PID control formula:

$$\Delta C_{A_{ij}} = K_P \cdot \epsilon(t) + \frac{1}{K_I} \cdot \sum_{IW} \epsilon(t) + K_D \cdot \frac{\epsilon(t) - \epsilon(t - DW)}{DW} \quad (2.10)$$

$$C_{A_{i(j+1)}} = C_{A_{ij}} + \Delta C_{A_{ij}} \quad (2.11)$$

where K_P, K_I and K_D are proportional, integral and derivative coefficients, respectively. $\epsilon(t)$ is the monitored error. IW and DW are tunable window sizes. The output $\Delta C_{A_{ij}}$ is fed back to the system and is used to regulate the next anticipated value for execution cycles.

Usually, the PID controller-based approaches adjust the system performance by periodically feeding the output back to the system. In order to achieve rapid and accurate response to the input, it is critical to select the appropriate set point and return the feedback to the system. To our knowledge, no existing PID controller-based DVS scheme has been proposed targeted towards game applications. This thesis is the first time to introduce the feedback control mechanism in the context of interactive games.

Adaptive Control

As more and more DVS algorithms are proposed to save energy, some people suggested DVS algorithms should be integrated with hardware adaptation. As we know, the exe-

cution time and energy equations are:

$$Execution \quad time = Instruction \quad count \times \frac{1}{IPC} \times \frac{1}{f} \tag{2.12}$$

$$Energy = Power \times Execution \quad time \tag{2.13}$$

$$= CV^2 f \frac{Instruction \quad count}{IPC \times f} \tag{2.14}$$

$$= CV^2 \frac{Instruction \quad count}{IPC} \tag{2.15}$$

IPC is instruction per cycle, f is frequency, V is the supply voltage and C is the effective capacitance. These equations suggest at least tow forms of hardware adaptation. The first is DVS, the other is to adapt the architecture to reduce the effective capacitance.

In [25, 26], they proposed two forms of adaptations – architectural adaptation and DVS to reduce energy. Their algorithm 1 begins with by predicting the energy of each instruction EPI of all possible hardware configurations (architecture and frequency) using profiles of a single frame for a subset of the configures. Subsequently, before the execution of each frame, the algorithm predicts the number of instructions with history-based schemes. With the instruction count and EPI estimates, the algorithm chooses the architecture and frequency combination that will consume the least energy and meet the deadline for the next frame.

The integrated adaptive control algorithm has more efficiency than the feedback control algorithm which applies DVS to save energy alone. However, their work assumed that IPC was almost constant for different frames of the same type at a given frequency. IPC of a frame is almost independent of clock frequency and the instruction count for a given frame type varies slowly frame frame to frame, due to mostly smooth changes

Algorithm 1 Adaptive Control

Profiling

1: For each architectural configuration and frame type: Measure IPC_A and power P_A.
2: For each hardware configuration define: $I_{max} = Deadline \times frequency \times IPC$.
3: For each frame type, order the hardware configurations in increasing order of $\frac{P_A \cdot V_H^2}{IPC_A}$.

Adaptation

1: Predict instruction count for the next frame of the same type using a history-based predictor.
2: Choose the lowest EPI architecture that has I_{max} higher than the predicted instruction count.

in the amount of work per frame. Obviously, due to their assumption of smooth change in application workload in control theory approach, the algorithms are inapplicable in the fast-pace application with large fluctuations. This approach is hardly applicable in interactive applications such as games, where the processing workload is unpredictable because of the user behavior.

Optimal Control

Zhang et al. proposed an algorithm in [57], where the information about the task execution time distribution obtained by profiling similar recently executed tasks was used to optimally procrastinate voltage increase as much as possible to minimize unnecessary energy expenditure. Assuming the workload distribution of a task S has been binned in ascending order in term of the number of clock cycles $\{c_1, c_2, c_3, ..., c_k\}$ and their associated probabilities $\{p_1, p_2, p_3, ..., p_k\}$. To calculate a set of scheduling voltage $V(S) = \{V_1, V_2, ..., V_k\}$ based on the workload distribution of S and deadline T by

solving constrained optimization problem, they obtain:

$$V_1 = \frac{c_1 + (c_2 - c_1)(1 - p_1)^{1/3} + ... + (c_k - c_{k-1})(1 - \sum_{i=1}^{k-1} p_i)^{1/3}}{KT} \quad (2.16)$$

$$V_j = V_1 \cdot \left(\frac{1}{1 - \sum_{i=1}^{j-1} p_i} \right)^{1/3} \qquad j = 2, ..., k \quad (2.17)$$

All calculations are done offline. At runtime, if the task finishes before its worst-case execution time, a low power mode for the rest of period is set to minimize the consumption. Following is their proposed algorithm 2 in the paper, where K is a system constant.

Algorithm 2 Optimal Control

Offline

1: Given task S, deadline T, workload distribution $\{c_1, c_2, c_3, ..., c_k\}$ and corresponding probability $\{p_1, p_2, p_3, ..., p_k\}$.
2: Calculate optimal schedule $V(S) = V_1, V_2, ..., V_k$ using equations 2.16 and 2.17.

Online

1: Intial_voltage_frequency(S): $V = V_1$, $f = KV_1$.
2: On number of clock cycles finished equal to c_{i-1}, change voltage and frequency to: $V = V_i$, $f = KV_i$, $i = 2, ..., k$.
3: Upon task_finish: set processor to low power mode until T.
4: Back to Step 1 for next task

Their assumption that the probability distribution of their cycle demands is stable or changes slowly and smoothly does not hold for interactive applications as games.

2.2.3 Offline Approaches

Instead of directly estimating decoding workload of video frame during decoding phase, Huang et al. proposed an idea to insert meta-data information into MPEG-2 bitstream

during video encoding phase [24]. These meta-data is parsed while decoding, so that the voltage/frequency is scaled accordingly to its pre-specified information.

To be more specific, in video decoding applications, three tasks predominately occupy most of processing workload: variable length decoding (VLD), inverse discrete cosine transform (IDCT) and motion compensation (MC). In their paper, they identified that the workload of these tasks could be parameterized with known information. As the VLD task involves Huffman decoding followed by run-length decoding, whose workload is estimated by Huffman codes with the number of non-zero IDCT coefficients. It is calculated as follows:

$$n_{VLD} = a \cdot n_{coeff} + b \tag{2.18}$$

where n_{coeff} is the number of non-zero IDCT coefficients, a and b are constants determined by processor microarchitecture and the VLD code.

The MC task performs functions with different input parameters, therefore, its workload is determined by the specific input parameters. These parameters include whether: (i) Y[1] component's x-dimension is HALF-PIXEL, (ii) Y component's y-dimension is HALF-PIXEL, (iii) U or V component's x-dimension is HALF-PIXEL, (iv) U or V component's y-dimension is HALF-PIXEL, (v) forward or backward motion compensation is required, and (vi) the motion compensation window size is 16×8 or 16×16. By counting the number of functions with specific input parameters, the workload of the MC can be accurately predicted.

The workload of IDCT could be approximated as a constant in an optimized algorithm, as shown in [10, 14].

[1]Each frame in MPEG-2 is represented in *YUV* color space. See ISO MPEG-2 standard for details.

By offline parsing video clips and obtaining the coefficients for each task (i.e. VLD and MC), they estimated workload for each video microblock and stored in meta-data.

Such meta-data approaches analyze pre-stored video clips offline and insert estimated workload into video frame. Obviously, it is impossible to extend these approach to realtime interactive applications such as games.

2.3 Power Management for 3D Graphics

Mochocki et al. [36, 37] proposed a signature-based estimation for predicting 3D graphics workload. In their simplified graphics simulator, they obtained specific parameters which imply graphics workload: (i) average triangle area, (ii) triangle count, (iii) average triangle height, and (iv) vertex count. By table looking-up techniques, the workload with the closest signature could be estimated.

Although their work investigates the application of DVS techniques for graphics applications, their graphics pipeline simulator used in the paper is customized for their experiments. It could hardly represent the typical graphics applications, especially a full-blown 3D game engine. The usage of signature tables would incur computational cost and impact the prediction accuracy, especially for the applications where the signatures exhibit big variability.

As discussed, a large number of paper on DVS algorithms for video decoding application works on the assumption that the application workload exhibits larger similarities. However, our results in the following chapters show that game workload has different characteristics from video decoding workload. Furthermore, we observe that game frames involve different types of *objects*, which contribute to the corresponding

processing workload of frames.

To exploit the observed natures of game workload, in the thesis, we propose three novel DVS schemes towards interactive gam applications. The first DVS scheme is designed based on a PID controller, which periodically adjusts workload prediction by responding to recent prediction errors. The second DVS scheme provides workload prediction by parsing constituent objects in game frames, without the foreknowledge of their historical behaviors. The above two DVS schemes work well for respective frame sequences exhibiting different degrees of variability. To take advantage of both, we propose a hybrid DVS scheme by combining two techniques. Our experiments show that the hybrid DVS scheme achieves significant power saving with good output quality.

Chapter 3

A Control Theory-based DVS Scheme

In this chapter, we propose a control theory-based DVS algorithm for interactive 3D game applications running on battery-powered portable devices. Using this scheme, we periodically adjust the game workload prediction based on the feedback from recent prediction errors. Although such control-theoretic feedback mechanisms have been widely applied to predict the workload of video decoding applications, they heavily rely on estimating the queue lengths of video frame buffers. Given the interactive nature of games – where game frames cannot be buffered – the control-theoretic DVS schemes for video applications can no longer be applied. Our main contribution is to suitably adapt these schemes for interactive games.

3.1 Introduction

Graphics-intensive game applications are now increasingly spilling over from high-end desktops to mobile devices (e.g. PDAs, cell phones and portable game consoles) for which battery-life is a major concern. This has resulted in a growing interest in power management schemes specifically directed towards 3D graphics and game applications

[23, 36, 37]. It is now well-established that such applications exhibit sufficient variability in their workload for dynamic voltage scaling (DVS) algorithms to be meaningfully applied [23, 29, 48]. Over the last few years, such algorithms have been very successfully applied to video encoding/decoding applications which are also computationally expensive and where the workload associated with processing different frames can vary significantly (for example, see [1, 11, 26, 56]). The basic principle behind most of these algorithms is to predict the workload associated with processing a video frame from the workloads of the previously decoded frames. The voltage/frequency of the underlying processor is then scaled based on such history-based workload predictions. This basic scheme has also been refined using control-theoretic feedback mechanisms, where previous prediction errors are taken into account while estimating the workload of a current frame [32, 41, 53, 54].

One of the primary differences between video processing and game applications is the interactive nature of games. Whereas video frames can be buffered, buffering is not possible in game applications where the content of a frame is dependent on the user input. As a result, many of the control-theoretic feedback mechanisms that were developed for predicting the workload of video processing applications cannot be applied to games. Such mechanisms use the various buffer occupancies as the primary input to the feedback controller (e.g. see [53, 54]). Here, the buffer fill levels are used as clues on the speed-balance between various processors or the processor and the output device. The goal is to maintain such buffer fill levels at some predetermined constant level, while running the processor at the lowest-possible frequency at all time. To achieve this goal, the processor's frequency has to be scaled in response to a time-varying workload to ensure that the buffers in question do not underflow or overflow.

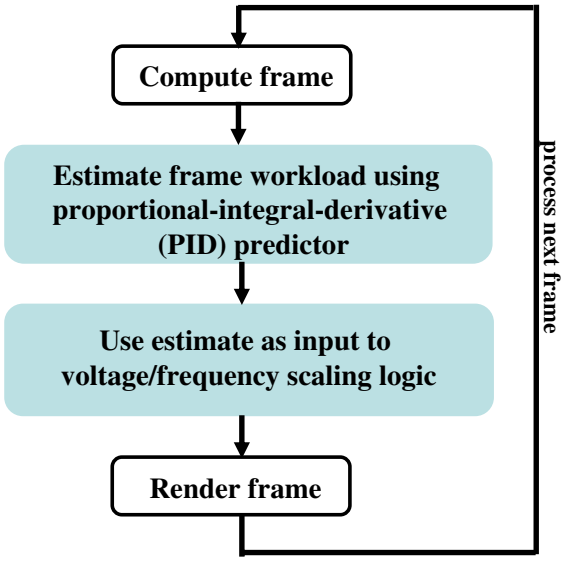


Figure 3.1: Integrating DVS in a game loop.

In this chapter, we investigate the use of such control-theoretic feedback mechanisms for dynamic voltage scaling for interactive 3D game applications. Such mechanisms have not yet been explored in the context of games, and more importantly, the buffer-centric approaches for workload prediction cannot be applied in this context. A high-level overview of our proposed scheme is shown in Figure 3.1. We use a proportional-integral-derivative (PID) controller to predict the processing workload of a game frame. Following standard control theory terminology [28], the predicted processing workload of a frame is set as the *measured variable* and the actual workload (obtained after rendering the frame) is considered to be the *set point*. The resulting prediction error (i.e. the difference between the predicted and the actual workload) is fed back to the PID controller and is used for predicting the workload of the next frame. The second component of our scheme is the *voltage/frequency scaling logic* which takes the predicted frame workload as input and decides whether the voltage/frequency of the processor should be changed from its current level. Since scaling a processor's voltage/frequency is associated with a certain overhead (which depends on the processor's

architecture, as well as the operating system running on top of it), very frequent voltage or frequency changes might not lead to optimal energy savings. Hence, the *voltage/frequency scaling logic* does not trigger a voltage/frequency switch in response to *every* workload change, but does so when such a change appears to last over a slightly longer duration. This is explained in further detail later in Section 6.1.

The rest of this chapter is organized as follows. Next section reviews the existing work of control theory mechanisms in video decoding applications, followed by the basics of PID controllers for close-loop systems in Section 3.3. In Section 3.4 we design our PID controller for game applications and apply the PID controller into the DVS scheme in Section 3.5. Finally we summarize this chapter in Section 3.6.

3.2 Control Theory in Video Applications

In [53, 54], Wu et al. modelled the buffers among multiple clock domains with queue theory, the queue occupancy was used to regulate the processor frequency in their DVS algorithm. Lu et al. presented a feedback-control scheduling algorithm for DVS in [32, 33]. The video decoding applications is modelled with $M/M/1$ queue system. The difference between the user-specified delay and the actual average delay is used to balance the processor frequency.

However, there is no buffer in game applications as video applications. Therefore, the control theory mechanisms which use the various buffer occupancies as the primary input to the feedback controller are not applicable to game applications.

3.3 PID Controller Basics

In this section, we describe a general proportional-integral-derivative (PID) controller for close-loop systems with an emphasis on the aspects that are important for understanding our scheme.

Feedback control is one of the fundamental mechanisms for dynamic systems to achieve equilibrium. In a feedback system, some variables – *measured process variables*, are monitored and measured by the feedback controller and compared to their desired values – *set points*. The differences (*errors*) between the controlled variable and the set point are fed back to the controller repeatedly. Corresponding system states are usually adjusted according to the differences to let the system variables approximate the set points as closely as possible.

A PID controller is a generic control loop feedback mechanism that is used to adjust system parameters based on the feedback from the recent error between a *measured process variable* and a desired *set point*. The measured variable usually reaches its set point and stabilizes within a short period. It involves three separate components – *Proportional Control*, *Integral Control* and *Derivative Control*. The proportional control determines the speed of the system in reacting to errors. The integral control is used to determine the accuracy of the system based on recent errors. Finally, the derivative control determines the system reaction based on the rate at which the error changes.

The PID feedback controller can be described in three major forms: the ideal form, the discrete form and the parallel form. Although the discrete form is often used in digital algorithms to keep tuning similar to electronic controllers, the parallel form is the simplest one where each control element is given the same error input in parallel.

The output of the controller is given by

$$Output(t) = P_{contrib} + I_{contrib} + D_{contrib} \tag{3.1}$$

where $P_{contrib}$, $I_{contrib}$ and $D_{contrib}$ are the feedback contributors of the PID controller.

$$P_{contrib} = K_p \cdot \epsilon(t) \tag{3.2}$$

$$I_{contrib} = \frac{1}{I} \cdot \int \epsilon(t)dt \tag{3.3}$$

$$D_{contrib} = D \cdot \frac{d\epsilon(t)}{dt} \tag{3.4}$$

where $\epsilon(t)$ is defined as the difference between a measured variable and a desired set point. K_p, I and D are the constants, which are defined as *proportional gain*, *integral gain* and *derivative gain* respectively.

By tuning the values of these gains, the PID controller can provide individualized control specific to process requirements involving error responsiveness, overshoot of the set point and system oscillation.

A high proportional gain results in a large change in the controller's output, given a changed system error. In contrast, a small proportional gain results in a small system response to a large system error, i.e., a less sensitive controller.

The integral contributor accelerates the movement of process towards set point and eliminates the residual steady-state error that occurs with a proportional-alone controller. However, since the integral contributor is responding to accumulated errors from the past, it can cause the present value to overshoot the set point (i.e. cross over the set point and then create a deviation in the other direction).

The derivative controller slows the rate at which the system output changes. Hence,

it is used to reduce the magnitude of the overshoot produced by the integral contributor and improve the controller stability. However, differentiation of an input error amplifies noise in the error, and thus the derivative controller is highly sensitive to noise in the error and can cause a process to become unstable, if the noise and the derivative gain are sufficiently large.

3.4 PID Controller Design

Here, the goal is to *predict* the workload of a game frame before the frame is processed/rendered. This is estimated to be the sum of the predicted workload of the previous frame and the output of the PID controller, which takes as input the prediction errors of a certain number of previously processed frames. Towards this we select the predicted workload $\bar{\omega}_i$ as the *measured variable* and the actual workload ω_i as the *set point*. The resulting error is periodically measured by the PID controller and is given by $\epsilon(t) = \omega_i - \bar{\omega}_i$, where t is the time stamp of the i-th frame. The following discrete PID controller formulation is used in our DVS scheme:

$$\Delta\bar{\omega}_i = K_p \cdot \epsilon(t) + \frac{1}{I} \cdot \sum_{T_I} \epsilon(t) + D \cdot \frac{\epsilon(t) - \epsilon(t - T_D)}{T_D} \tag{3.5}$$

$$\bar{\omega}_{i+1} = \bar{\omega}_i + \Delta\bar{\omega}_i \tag{3.6}$$

As explained in Section 3.3, here K_p, I and D are the proportional, integral and derivative coefficients respectively. T_I and T_D are the tunable parameters of the controller. In our scheme, T_I is set to the frame interval, and hence, $\sum_{T_I} \epsilon(t)$ is the sum of the prediction errors from frame $(i - T_I)$ to frame i. T_I is set to be 5 frames; hence, only the errors from previous 5 frames contribute to the integral element. T_D is set to be

equal to the frame execution time and hence $\epsilon(t - T_D)$ is the prediction error at time $t - T_D$. We restrict T_D to be the execution time of the last one frame to ensure that multiple feedback corrections do not affect one another. The output $\Delta\bar{\omega}_i$ is fed back to the controller and regulates the next estimated frame workload $\bar{\omega}_{i+1}$.

3.4.1 Tuning PID Parameters

In our experiments, we observe that the selection of values impacts the prediction results significantly. By manually tuning the parameters, we obtain the best prediction when $K_p = 0.5$, $I = 28$ and $D = 0.00001$ for entire game play, however, the optimal values of these parameters might vary from one game engine to the next (depending on the variability of the workload).

Proportional Gain

Figure 3.2, 3.3, 3.4 and 3.5 show the workload prediction using the proposed PID controller-based scheme with unchanged I and D, while K_p is set to 1, 0.7, 0.3 and 0.1 respectively.

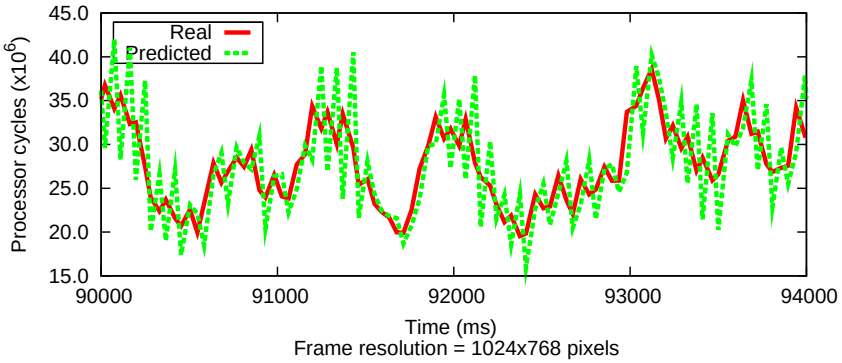


Figure 3.2: $K_p = 1$, $I = 28$, and $D = 0.00001$. The mean absolute prediction error is 4×10^6 cycles with standard deviation 2.5×10^6.

Note that if the proportional parameter is too large, the controller becomes unstable

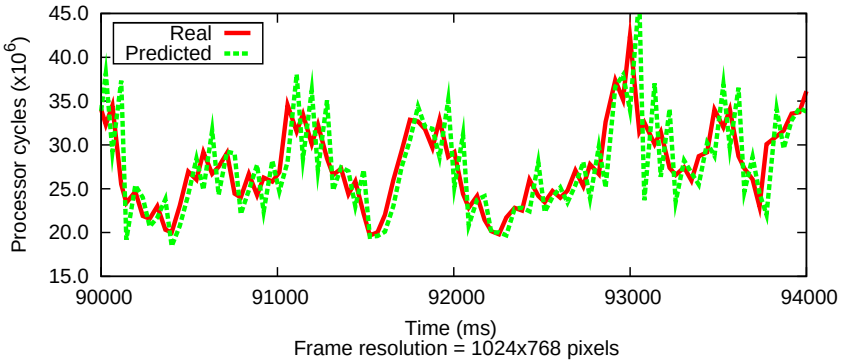


Figure 3.3: $K_p = 0.7$, $I = 28$, and $D = 0.00001$. The mean absolute prediction error is 3.4×10^6 cycles with standard deviation 2.7×10^6.

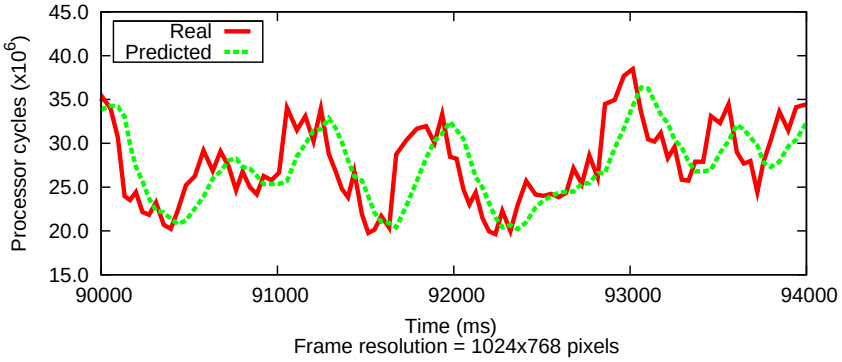


Figure 3.4: $K_p = 0.3$, $I = 28$, and $D = 0.00001$. The mean absolute prediction error is 3×10^6 cycles with standard deviation 2.1×10^6.

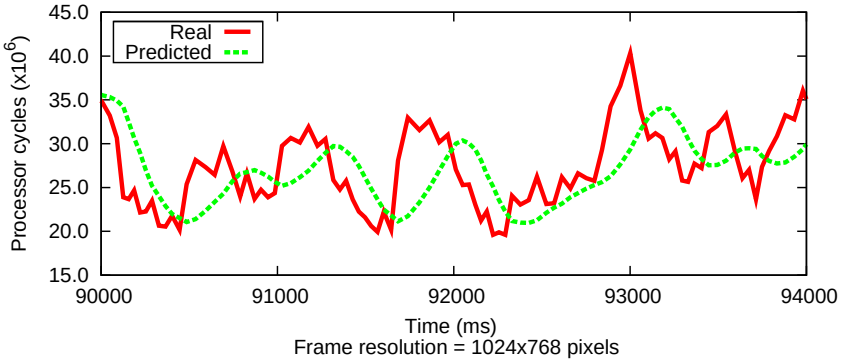


Figure 3.5: $K_p = 0.1$, $I = 28$, and $D = 0.00001$. The mean absolute prediction error is 3.1×10^6 cycles with standard deviation 2.1×10^6.

(see Figure 3.3 and Figure 3.2). In Figure 3.3, the mean prediction error is 3.4×10^6 cycles with standard deviation 2.7×10^6. While in Figure 3.2, the mean absolute prediction error goes up to 4×10^6 with 2.5×10^6 standard deviation.

In contrast, a smaller proportional parameter results in a slower response to input errors in the controller as shown in Figure 3.5 and Figure 3.4, where the mean prediction errors are 3×10^6 cycles with standard deviation 2.1×10^6. By compared with the result shown in Figure 3.6 when K_p is set to 0.5, where the mean absolute prediction error is 2.4×10^6 cycles with standard deviation 1.8×10^6, we notice the big improvement when K_p is set to the optimal value.

Integral and Derivative Gains

Moreover, we compare the results of the PID controller-based predictors with fixed $K_p = 0.5$ and $D = 0.00001$, while the integral parameter is set to different values such as 25, 26, 27, 28, 29 and 30. We notice that the predictor gives the best result when the integral parameter is set to 28. In the same manner, we manually tune the derivative parameter to different values such as 0.000001, 0.00001, 0.0001, 0.001, 0.01 and 0.1, coupled with $K_p = 0.5$ and $I = 28$. With the derivative parameter set to 0.00001, the predictor provides better result than the others. Furthermore, we notice that there are huge deviations from set points when the derivative parameter is sufficiently large (e.g. 0.1).

Impact of Proportional, Integral and Derivative Gains

Figure 3.7, 3.8 and 3.9 compare the workload prediction errors in cycles with the above tested values of three parameters, respectively. Note that the PID controller produces the best prediction when $K_p = 0.5$, $I = 28$ and $D = 0.00001$. Figure 3.7 shows that

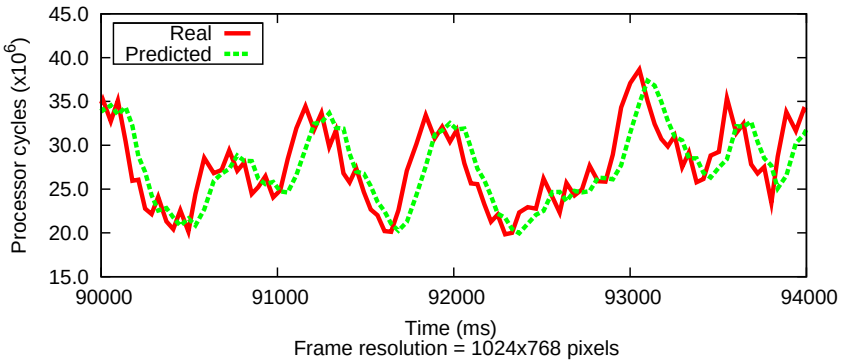


Figure 3.6: $K_p = 0.5$, $I = 28$, and $D = 0.00001$. The mean absolute prediction error is 2.4×10^6 cycles with standard deviation 1.8×10^6.

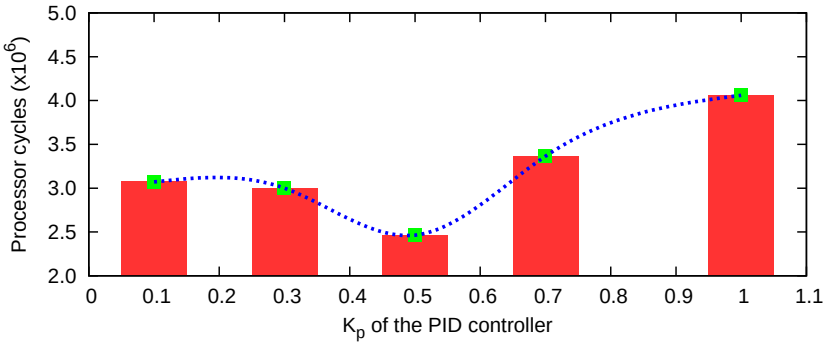


Figure 3.7: Impact of the proportional parameter K_p on frame workload prediction (errors in processor cycles), using the PID controller-based scheme.

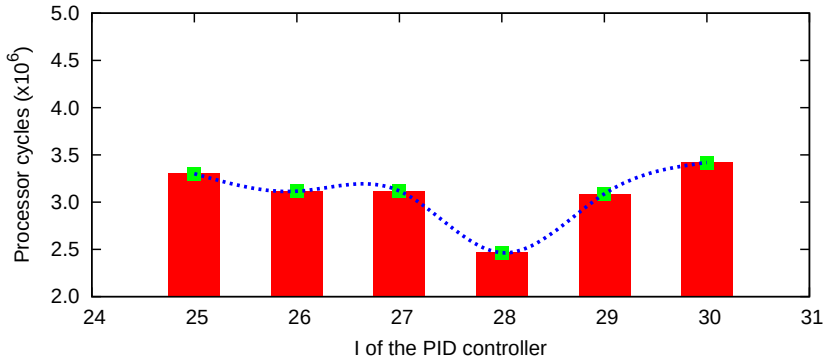


Figure 3.8: Impact of the integral parameter I on frame workload prediction (errors in processor cycles), using the PID controller-based scheme.

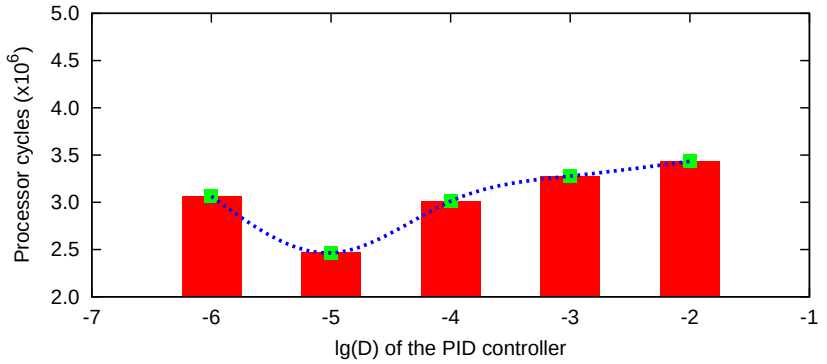


Figure 3.9: Impact of the derivative parameter D on frame workload prediction (errors in processor cycles), using the PID controller-based scheme.

the absolute prediction error is as high as 4×10^6 cycles when K_p is set to 1, while its error drops to 2.4×10^6 with $k_p = 0.5$, $I = 28$ and $D = 0.00001$. Figure 3.8 shows that the prediction error drops from 3.4×10^6 cycles with $I = 30$ to 2.4×10^6 with $k_p = 0.5$, $I = 28$ and $D = 0.00001$. In Figure 3.9, the prediction error comes up to 3.4×10^6 cycles when $D = 0.01$, compared with prediction error in 2.4×10^6 cycles with the optimal values.

3.4.2 Applying to a Different Demo File

We investigate that the optimal values selected based on one demo file could be applied to a different demo file from the same game engine, as they exhibit the similar degree of workload variability. In this section, we observe that the PID controller-based scheme

with the same optimal values of these parameters (i.e. $K_p = 0.5$, $I = 28$ and $D = 0.00001$) consistently achieves the best prediction, when applied to a different Quake II demo file from the above.

Proportional Gain

Figure 3.11, 3.12, 3.13 and 3.14 show the workload prediction using the proposed PID controller-based predictors with unchanged I and D, while K_p is set to 1, 0.7, 0.3 and 0.1 respectively. Figure 3.10 shows the best prediction when K_p is set to 0.5. where the mean absolute prediction error is 2.6×10^6 cycles with standard deviation 2.3×10^6.

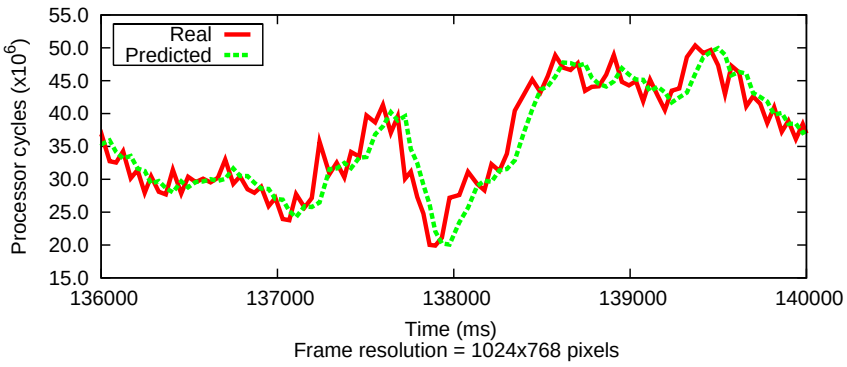


Figure 3.10: Apply $K_p = 0.5$, $I = 28$, and $D = 0.00001$ to a different demo file. The mean absolute prediction error is 2.6×10^6 cycles with standard deviation 2.3×10^6.

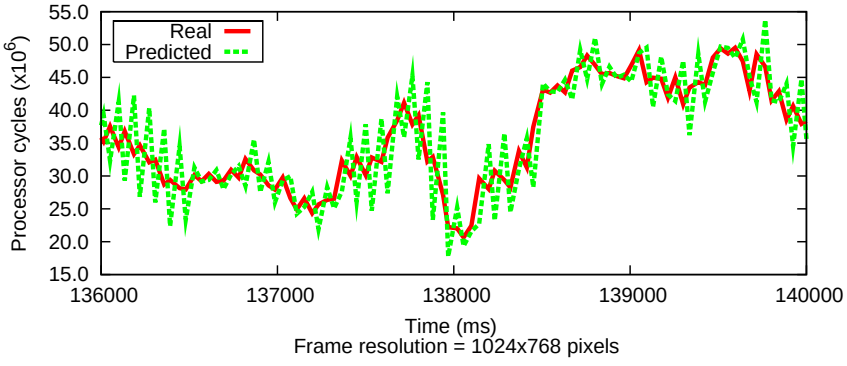


Figure 3.11: Apply $K_p = 1$, $I = 28$, and $D = 0.00001$ to a different demo file. The mean absolute prediction error is 4.2×10^6 cycles with standard deviation 2.9×10^6.

Notice that the controller becomes unstable (see Figure 3.12 and Figure 3.11), if the proportional parameter is too large ($K_p = 0.7$ and $K_p = 1$). In Figure 3.12, the mean prediction error comes up to 3.3×10^6 cycles. While it becomes even worse when $K_p = 1$ as shown in Figure 3.11.

While the controller becomes less sensitive to input error with a smaller proportional parameter ($K_p = 0.1$ and $K_p = 0.3$), as shown in Figure 3.14 and Figure 3.13, where the mean prediction errors are 3.8×10^6 and 2.7×10^6 cycles.

Integral and Derivative Gains

We apply the optimal value of the integral parameter ($I = 28$) to the different demo file, with fixed $K_p = 0.5$ and $D = 0.00001$. By compared with other settings such as 25, 26, 27, 29 and 30, we notice that the predictor gives the best result when the integral parameter is set to 28. Similarly, we observe that the predictor provides better result than the others, when the derivative parameter is set to 0.00001, coupled with $K_p = 0.5$ and $I = 28$.

Impact of Proportional, Integral and Derivative Gains

Figure 3.15, 3.16 and 3.17 compare the workload prediction errors in cycles, when K_p, I and D are set to different values respectively. Note that the PID controller produces the best prediction when $K_p = 0.5$, $I = 28$ and $D = 0.00001$. Figure 3.7 shows that the prediction errors is up to 4.2×10^6 when $K_p = 1$, compared with 2.6×10^6 when $k_p = 0.5$, $I = 28$ and $D = 0.00001$. Figure 3.8 shows that the prediction error is 3.2×10^6 when $I = 25$, while it drops to 2.6×10^6 with the optimal values. In Figure 3.9, the prediction error goes up to 3.9×10^6 with $D = 0.01$, compared with 2.6×10^6 with $k_p = 0.5$, $I = 28$ and $D = 0.00001$.

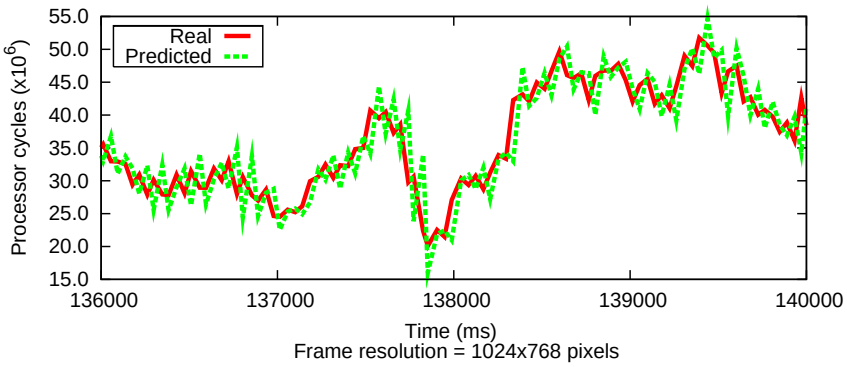


Figure 3.12: Apply $K_p = 0.7$, $I = 28$, and $D = 0.00001$ to a different demo file. The mean absolute prediction error is 3.3×10^6 cycles with standard deviation 2.2×10^6.

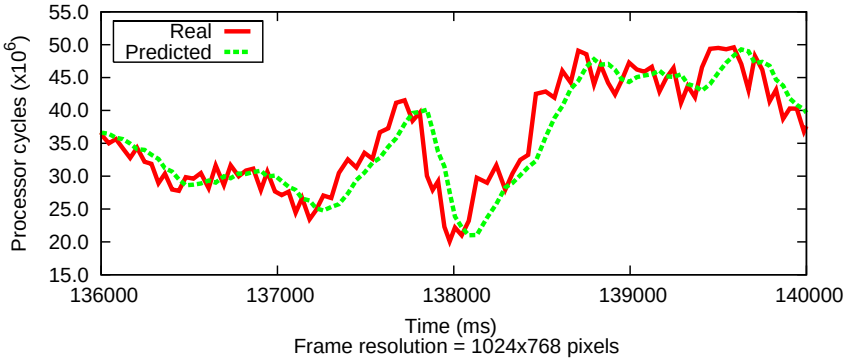


Figure 3.13: Apply $K_p = 0.3$, $I = 28$, and $D = 0.00001$ to a different demo file.The mean absolute prediction error is 2.7×10^6 cycles with standard deviation 2.3×10^6.

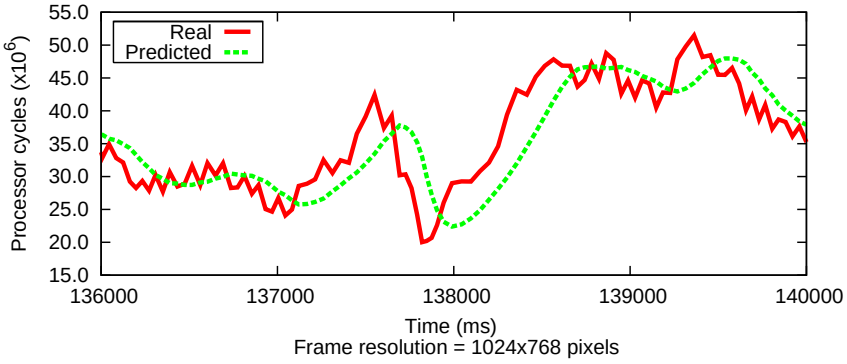


Figure 3.14: Apply $K_p = 0.1$, $I = 28$, and $D = 0.00001$ to a different demo file. The mean absolute prediction error is 3.8×10^6 cycles with standard deviation 2.8×10^6.

3.5 Workload Prediction

Figure 3.18 shows the various components of our DVS scheme. The value of the error $\epsilon(t)$ (obtained from previous frames) is used to compute the proportional, integral and derivative elements. Finally, the sum of these elements is used to estimate the workload of the current frame. This is followed by computing the voltage/frequency of the processor based on the predicted frame workload and the target frame rate. The processor's voltage/frequency is then scaled based on a *scaling logic*, followed by the rendering steps.

3.6 Summary

Although the control-theoretic feedback mechanisms have been previously applied to video encoding/decoding applications, their use in games have not been explored before. In this chapter we proposed the PID controller-based DVS scheme for interactive 3D game applications. In this DVS scheme, the resulting prediction error between the predicted and the actual workload is fed back to the PID controller and used to regulate the workload prediction of next frame.

In Chapter 5 and Chapter 6, this scheme is compared to other workload prediction schemes on different platforms (e.g. laptops and PDAs). It yields improvements in terms of power saving as well as output quality, compared to history-based workload prediction schemes – where the workload of a game frame is predicted by averaging the workload of previously rendered frames.

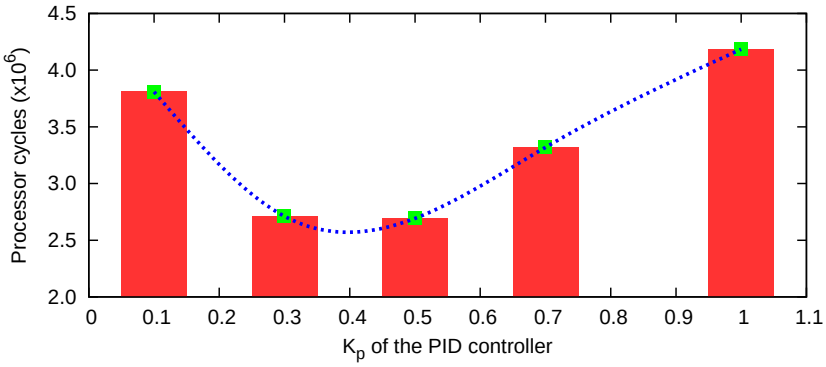


Figure 3.15: Impact of the proportional parameter K_p on frame workload prediction (errors in processor cycles), when applied to a different demo file.

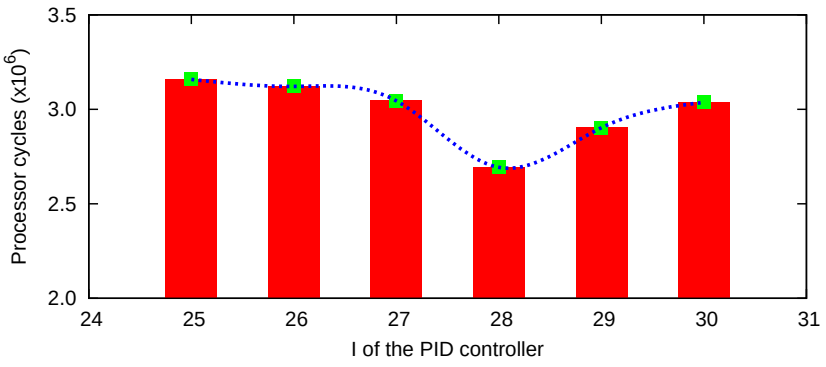


Figure 3.16: Impact of the integral parameter I on frame workload prediction (errors in processor cycles), when applied to a different demo file.

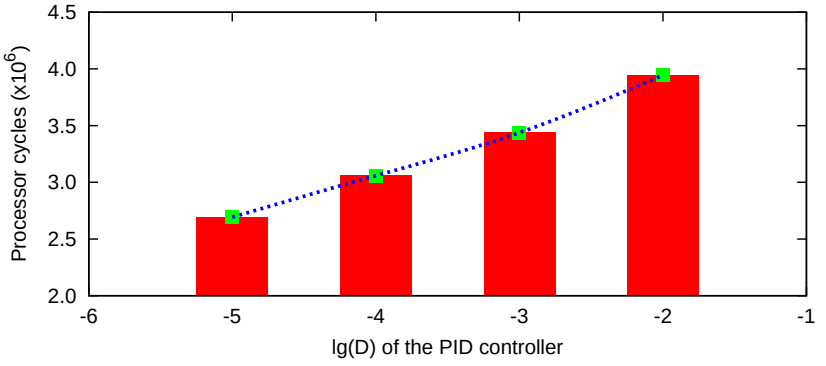


Figure 3.17: Impact of the derivative parameter D on frame workload prediction (errors in processor cycles), when applied to a different demo file.

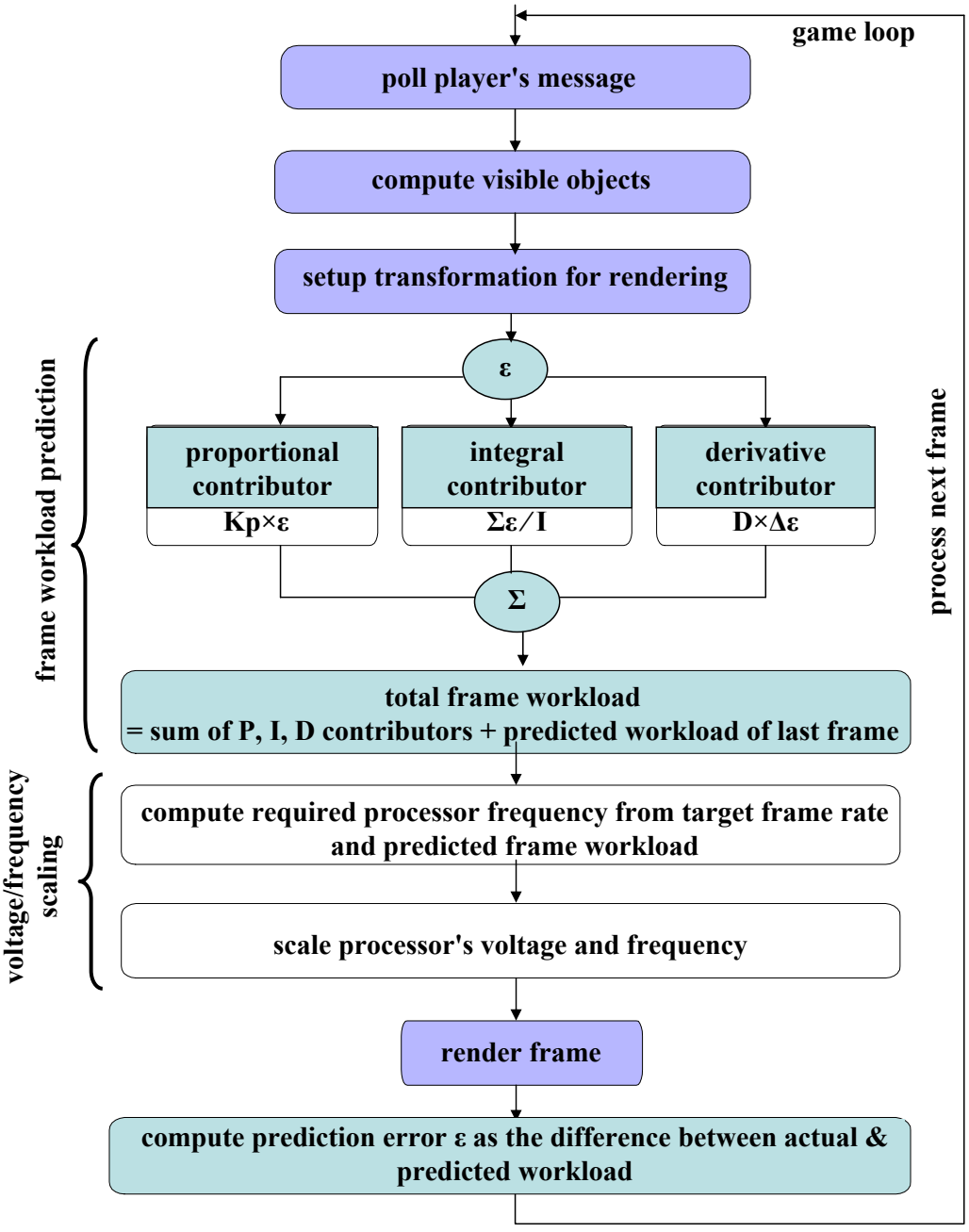


Figure 3.18: Overview of the PID-based DVS scheme.

Chapter 4

A DVS Scheme by Exploiting Frame Structure

In this chapter, we propose a novel DVS scheme that is specifically directed towards 3D graphics-intensive interactive game applications running on battery-operated portable devices. The key to this DVS scheme lies in parsing each game frame to estimate its rendering workload and then using such an estimate to scale the voltage/frequency of the underlying processor. The main novelty of this scheme stems from the fact that game frames offer a rich variety of ”structural” information (e.g. number of brush and alias models, texture information and light maps) which can be exploited to estimate their processing workload. Although DVS has been extensively applied to video decoding applications, compressed video frames do not offer any information (beyond the frame types – I, B or P) that can be used in a similar manner to estimate their processing workload. As a result, DVS algorithms designed for video decoding mostly rely on history-based mechanisms, where the workload of a frame is predicted from the workloads of the previously-rendered frames.

4.1 Introduction

Although DVS algorithms have been extensively applied to video encoding/decoding applications (which have almost attained the status of the *dining philosophers problem* in this domain) [38, 55, 56], their use in graphics-intensive games has not been sufficiently explored so far. We initiate a study of this problem in [23] and observe that game applications exhibit sufficient variability in their workload to meaningfully exploit DVS schemes for power savings. Moreover, they offer the possibility of developing DVS algorithms that better exploit the characteristics of game applications (compared to those that have been developed for video decoding).

One can envisage partitioning a game play into multiple *scenarios*. Each scenario can then be associated with a different power management policy – or in the simplest case a fixed operating frequency level. However, our experiments suggest that it is not possible to distinguish between workload variations across different scenarios, especially with reasonably simple scenario definitions (e.g. rooms). Figure 4.1 shows the workload variation when a scenario is defined as a room. The segment from 5000 to 65000 millisecond demonstrates the workload distribution when the player was in *Room 1*. The player kept changing his directions to face the different objects within this room, thereby, the workload varies from 14 to 29 million cycles per frame. The player moved from *Room 1* and passed a short passage before he entered *Room 2*. While the segment from 75000 to 135000 millisecond illustrates the workload variation for a different *Room 2*. Note that the game workload generated in *Room 2* varies from 14 to 28 million cycles. The workload variations from *Room 1* and *Room 2* exhibit considerable similarity. One could also define a scenario as a room viewed from a certain direction. However, we observe that even with such scenario definitions, it is still infeasible to

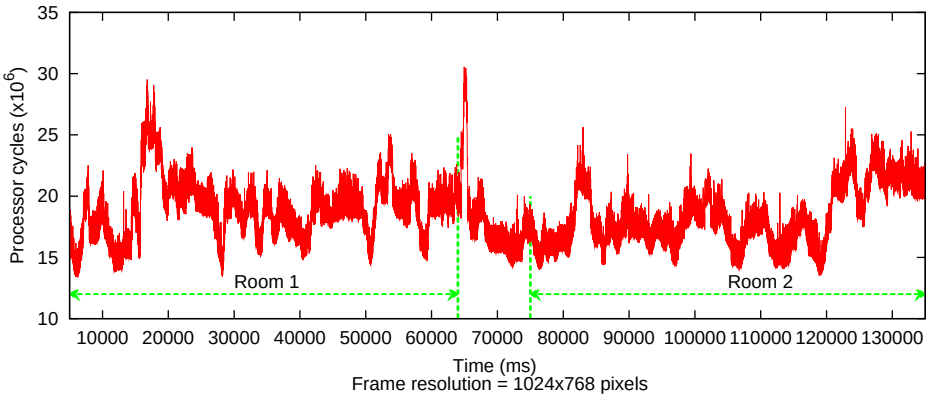


Figure 4.1: Workload in different game scenarios exhibits considerable similarity.

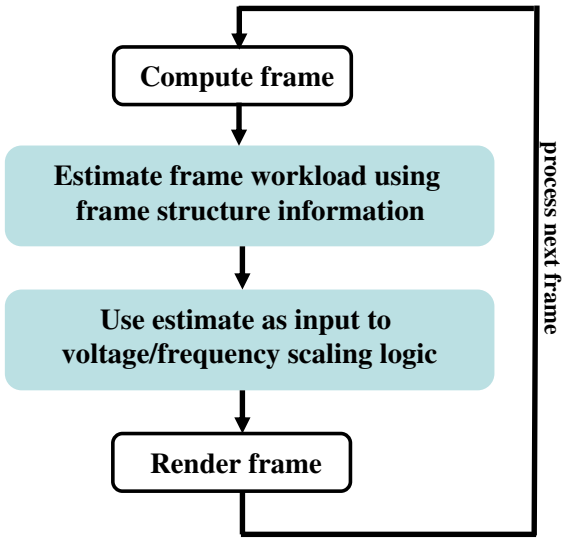


Figure 4.2: DVS in a game loop.

distinguish them by their workload variations. In fact, we observe that game scenario is constituted of multiple objects. More specifically, game frames are more ”structured” than video frames (which only contain the I, B, or P frame-type information). The workload associated with processing a game frame depends on the contents of the frame, or the constituent *objects*, which can be easily determined by parsing the frame. Similar conclusions were also arrived at in [36, 37] for workload characterization of a 3D graphics processing pipeline.

Hence, in this chapter, we propose a workload prediction and DVS scheme by exploiting such frame structure. Instead of predicting the workload associated with pro-

cessing a game frame from the history of previously-processed frames – as done with video decoding applications – we estimate this workload by *parsing* the game frame. The main novelty of our scheme stems from the observation that game frames offer a rich variety of *structural* information which can be used to predict their workload or processor cycle requirements.

An overview of this scheme is shown in Figure 4.2. The ”voltage/frequency scaling logic” is used to decide whether the voltage/frequency of the processor is to be changed from its current level based on the workload estimation. Since scaling the voltage/frequency of a processor involves a certain overhead – which depends on the processor's architecture as well as the underlying operating system – it might not be meaningful to switch the voltage and frequency in response to every workload change. This is explained in further detail later in Section 6.1. It may be noted that this scheme of parsing a frame to estimate its processing workload cannot be applied to video frames, which offer no structural information beyond the frame type (i.e. I, B or P).

The rest of this chapter is organized as follows. In the next section we give a brief overview of game workload and game maps. In Section 4.3, we introduce our framework of game workload characterization, followed by our workload prediction scheme using frame structures in Section 4.4. We summarize this chapter in Section 4.5.

4.2 Preliminaries

In this section, we introduce the corresponding processing workload for the important steps in a game engine (shown in Figure 1.1) and game ”assets” – game maps.

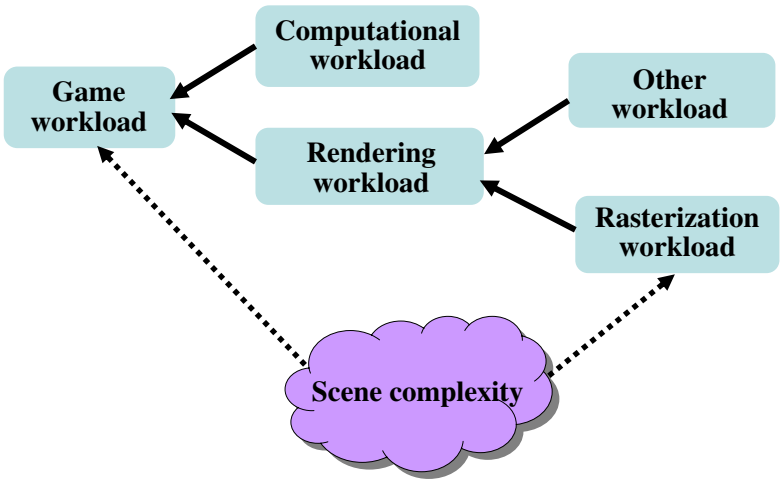


Figure 4.3: Corresponding workload associated with steps in processing a game frame.

4.2.1 Game Workload

As mentioned in Section 1.1, a game engine is designed to sequentially execute the computing and rendering tasks. For each frame, the engine polls the user's input and passes it over to the computing subsystems responsible for collision detection, AI, particle simulation etc. These subsystems compute new locations and appearances of the visible objects based on the user input. We refer to the resulting workload as the *computation workload*. The results of these computations are passed to the rendering task, which renders all the visible objects in the current frame and displays them on the screen. A significant component of this rendering task involves *rasterizing* objects on the screen. From this point on, we are primarily concerned with this rasterization component of the rendering task. Henceforth, we define the workload resulting from the rasterization task as the *rasterization workload*. Figure 4.3 illustrates the workload towards different tasks in a frame. As most of available mobile devices (e.g. low-end laptops, PDAs) do not support hardware accelerators, all tasks – including geometry processing, rasterization and texture processing – are performed on the CPU.

As observed, the total processing workload has a clear correspondence with the

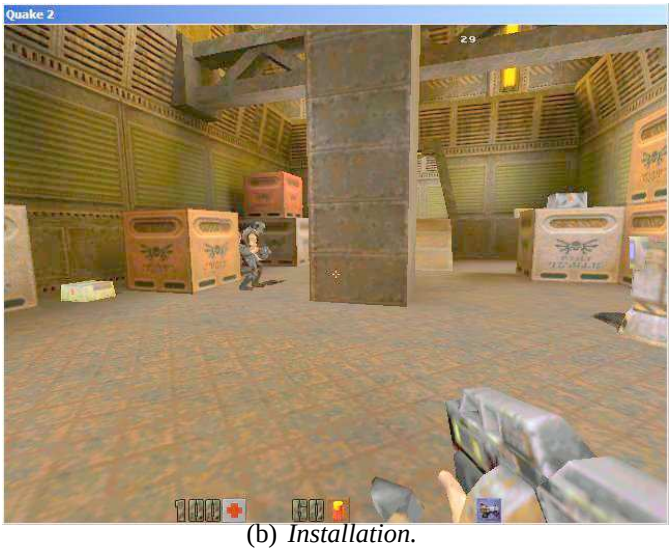
complexity of game scene. Moreover, we notice that the rasterization workload of a frame clearly has a direct correspondence with the *objects* that are contained in the frame. In other words, it depends on the ”complexity” of the scene to be rendered.

4.2.2 Game Maps

Before proceeding further, we will need to understand what a *game map* (also referred to as a *level*) is. The storyline of a game can be considered to progress from one location (or level) to the next, where each of these locations is represented using a game map. Examples of game maps might be cities, buildings, rooms and corridors. Intuitively, a game map may be considered to be a data structure which stores all the objects and characters in the scenario represented by the map. Snapshots of three different game maps from Quake II, called *Outer Base*, *Installation* and *Command Center* are shown in Figures 4.4(a), 4.4(b) and 4.4(c) respectively. The game map *Installation* is used in the default demo.

A commonly used data structure to represent a game map is a Binary Space Partition (BSP) tree [50]. A BSP tree represents a recursive, hierarchical partitioning or subdivision of space into convex subspaces. The BSP tree is constructed by partitioning a space using a hyperplane, with the resulting partitions being further partitioned by recursively applying the same procedure. For each leaf in the BSP tree, a set of leaves that are *visible* from this leaf are calculated and updated as the game is played. This set is referred to as the Potentially Visible Set (PVS). In addition, the BSP tree also records information related to texture and lighting. Both the computation and the rendering steps in processing a game frame (Figure 1.1) involve traversing and manipulating the BSP tree.

(a) *Outer Base.*



(b) *Installation.*

(c) *Command Center.*

Figure 4.4: Game maps.

In Quake-serial engines, a game map is divided into convex regions, forming the leaves of the BSP tree. To render a game map, the BSP tree is traversed to determine the leaf in which the *camera* is located. Once this leaf is identified, the PVS associated with this leaf lists the potentially visible leaves from this camera location[1]. The bounding boxes of these leaves are then used to quickly *cull* leaves from the PVS that are not within the viewing frustum. The remaining leaves are then passed to the subsequent rendering tasks, which include matrix transformations on the data and the rasterizing of a frame as 2D image onto the screen.

4.3 Workload Characterization

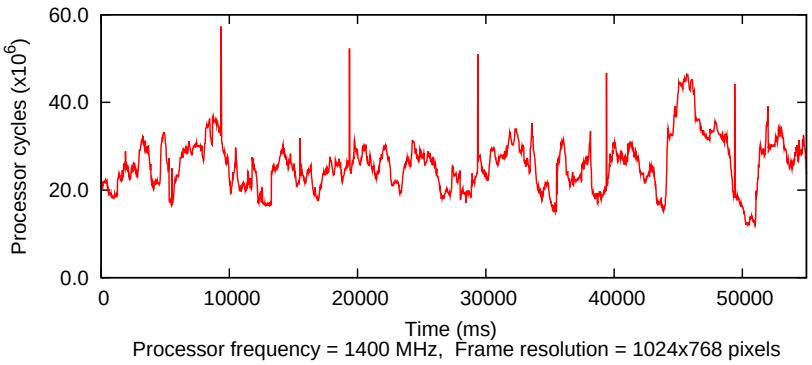


Figure 4.5: Rasterization workload per frame.

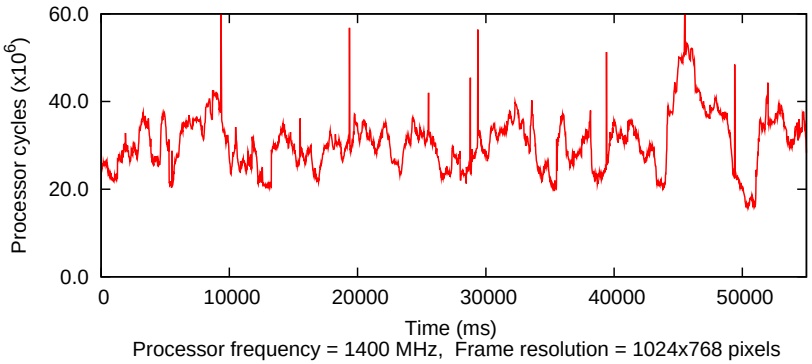


Figure 4.6: Total processing workload per frame.

[1]http://www.flipcode.com/articles/article_q2bsp.shtml"

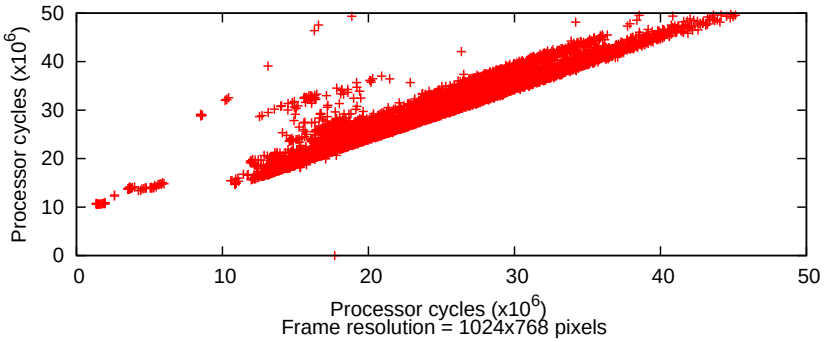
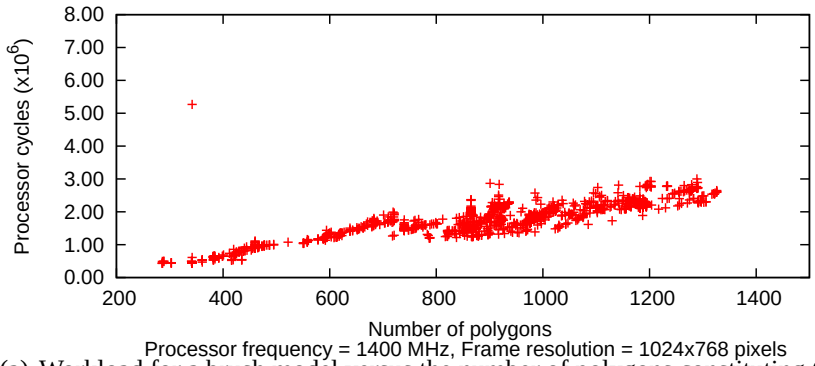


Figure 4.7: Linear correlation between rasterization and total processing workload of a frame.

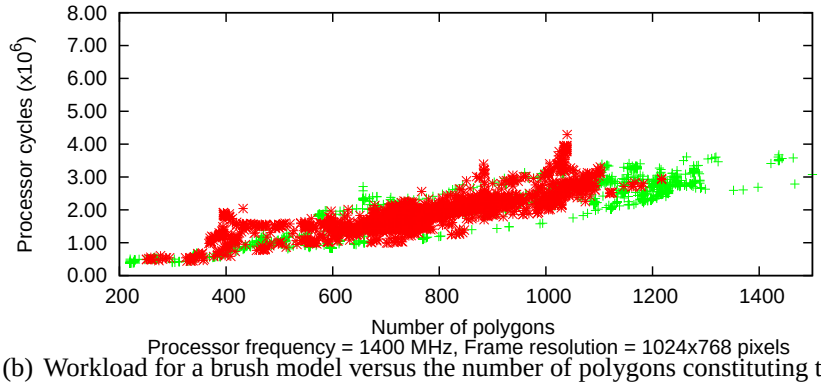
As mentioned in Section 4.2.1, both the rasterization workload and the total game workload of a frame clearly have direct correspondences with the *objects* that are contained in the frame. Figure 4.5 and 4.6 show how these two workload changes with time. Note that the fluctuations in the processor cycle demands in Figures 4.5 and 4.6 are highly correlated. Figure 4.7 shows the correlation between these two workloads, with the horizontal axis denoting the rasterization workload. Further, our measurements show that the rasterization workload constitutes approximately 75% of the total workload generated in processing a frame. From such observations, we believe that one can predict the total processing workload to reasonable accuracy if one can estimate the rasterization workload.

The rest of this section shows how the rasterization workload can be predicted. We propose a workload characterization in which the workload associated with rasterizing a frame depends on the constituting objects – which is specified as *primitives*. Our experimental results show that for Quake II, the types of primitives that predominately contribute to this workload are *brush model*, *alias model*, *texture*, *light map* and *particles*.

4.3.1 Brush Model



(a) Workload for a brush model versus the number of polygons constituting the brush model (Game Map: *Command Center*).



(b) Workload for a brush model versus the number of polygons constituting the brush model (Game Maps: *Outer Base* and *Installation*).

Figure 4.8: Brush model.

Brush models are 3D convex solids composed of polygons. Brush models are used to construct the geometry of a game map and they define the ”world space” in which players can move around. The workload resulting from rasterizing a frame will depend on the *number* of brush models in the frame and also the *types* of these models. We therefore parameterize a brush model using the number of *polygons* constituting it. To identify the workload involved in rasterizing a brush model with a specified number of polygons, we collect the number of polygons constituting each brush model and the number of processor cycles involved in rasterizing them.

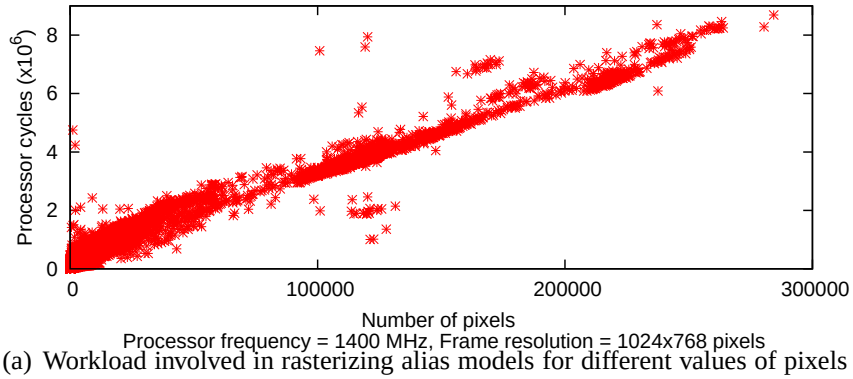
Our results obtained from the replay of a demo using the game map *Command*

Center are shown in Figure 4.4(c). Figure 4.8(a) shows that the number of processor cycles required to rasterize a brush model increases almost linearly with the number of polygons in it, which indicates that the rasterization workload of brush model can be predicted based on the relationship with the number of constituting polygons. Let us assume that $c(n)$ is the number of processor cycles requires to rasterize any *one* brush model with n polygons. Then $c(n)$ is calculated by the above-mentioned correlation. To compute the rasterization workload for *all* brush models in a frame, let us assume that $B(n)$ is the number of brush models in this frame with n polygons. Then the total rasterization workload for all the brush models in the frame is equal to $\sum_{n=1,\ldots,\infty} c(n) \times B(n)$ processor cycles (where ∞ is the maximum possible number of polygons in any brush model).

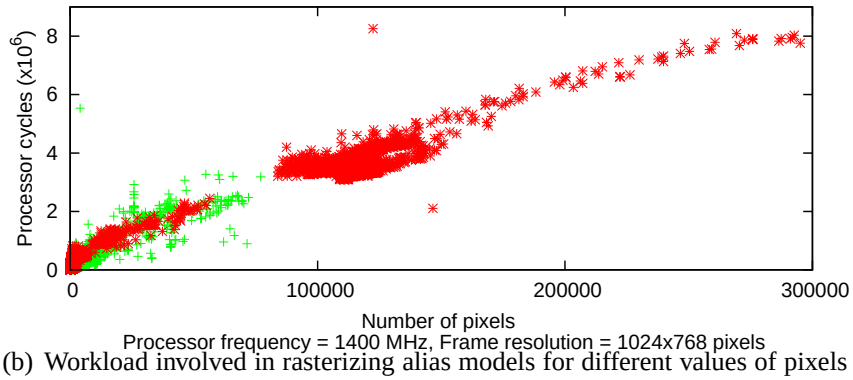
To see if the relationship between the rasterization workload and the number of polygons in a brush model holds for different game maps, we repeat the experiments using other game maps as well. Figure 4.8(b) shows the workload involved in rasterizing brush models with different number of polygons from two other game maps, *Outer Base* and *Installation* (see Figures 4.4(a) and 4.4(b)). Note that the number of processing cycles for each polygon is consistent across the three game maps and the workload involved in rasterizing brush models also increases linearly with the number of polygons for the latter two game maps. It is known that different game maps may include different brush models, however, different brush models all consist of polygons. Therefore, the rasterization workload of brush models in different game maps is totally determined by the rasterization of constituting polygons. Our results confirm that the relationship between the rasterization workload of brush models and the number of polygons holds across different game maps. Thus, we can predict the rasterization workload of brush

models based on the relationship with the number of polygons from a different game map.

4.3.2 Alias Model



(a) Workload involved in rasterizing alias models for different values of pixels (Game Map: *Installation*).



(b) Workload involved in rasterizing alias models for different values of pixels (Game Maps: *Outer Base* and *Command Center*).

Figure 4.9: Alias model.

Alias models are used to represent different entities in Quake II (such as monsters, soldiers and weapons). Usually an alias model is composed of *triangles*. Since the rasterization of triangles is done on the CPU instead of a graphics hardware, the number of *pixels* constituting each triangle affects the CPU workload.

We characterize the rasterization workload of an alias model by the total number of pixels rendered. This number can be obtained by summing up the area of triangles constituting an alias model. To compute the rasterization workload of alias models with

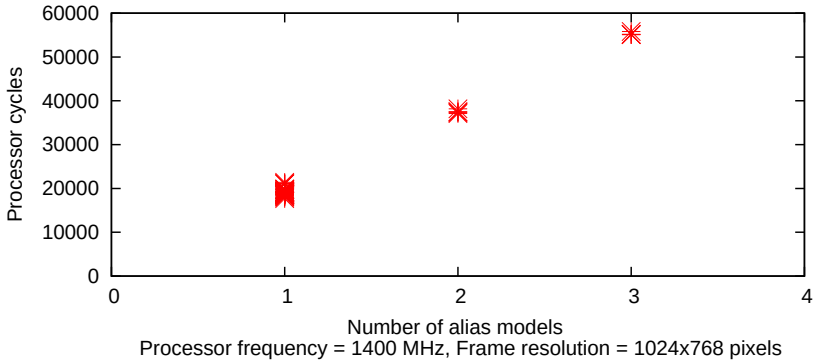


Figure 4.10: Rasterization workload for alias models linearly scales to number of alias models (Game Map: *Installation*).

different values of constituting pixels, we capture all models arising in different frames along with their rasterization workloads.

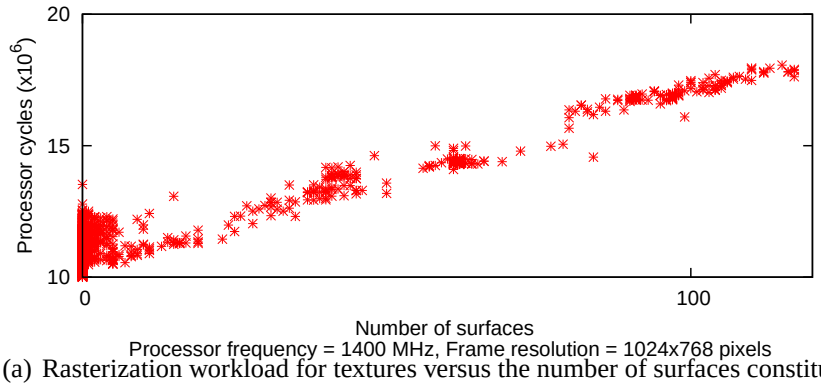
Figure 4.9(a) shows the results for alias models with different pixels. This figure shows that the rasterization workload of an alias model scales almost linearly with the number of pixels. Similar to brush models, these results suggest that the rasterization workload of alias models can be predicted based on its relationship with the number of pixels of alias models.

It seems that these relationships are also consistent across game maps. Figure 4.9(b) shows the workload involved in rasterizing alias models with different number of pixels from two different game maps, *Outer Base* and *Command Center*. Note that the linear relationships are consistent across the different game maps, which suggests that the rasterization workload of alias models can be predicted based on the relationship with the number of pixels from a different game map. It can be explained that the rasterization of constituting pixels determines the rasterization of alias models regardless of game maps, although different game maps may contain different alias models.

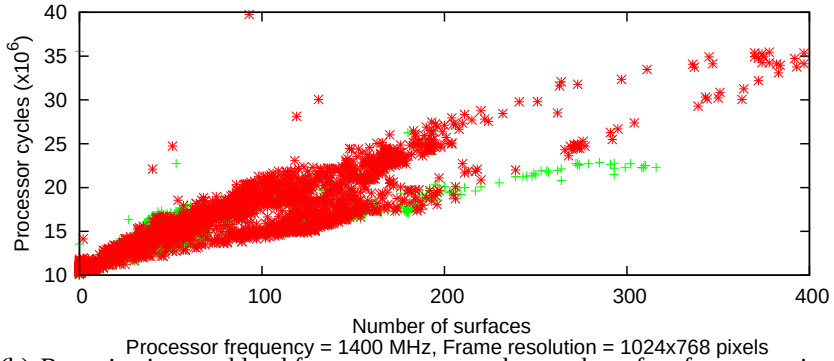
From Figure 4.10, note that when multiple alias models with these pixels occurred in a frame, their rasterization workload almost linearly scales with the number of alias

models. The same linear relationship also holds for alias models with different number of pixels and for different game maps. This case is therefore similar to brush models whose workload is parameterized using the number of polygons. The workload involved in rasterizing all alias models in a frame can be therefore calculated from the workloads of different alias models (i.e. models having different number of pixels) occurring in the frame and multiplying them by the number of times a particular model occurs.

4.3.3 Texture



(a) Rasterization workload for textures versus the number of surfaces constituting the textures (Game Map: *Command Center*).



(b) Rasterization workload for textures versus the number of surfaces constituting the textures (Game Maps: *Outer Base* and *Installation*).

Figure 4.11: Texture.

Textures are the 2D images applied to the face of brush models to give them the appearance of real surfaces. For instance, concrete slabs, brick walls and metal plates.

Textures are typically composed of multiple *surfaces*. We therefore characterize the rasterization workload of textures in terms of the number of surfaces constituting them.

As in the case of brush models, we capture the textures arising from a sample game play and plot their rasterization workload versus their number of surfaces in Figure 4.11(a). In this case we find that the rasterization workload increases almost linearly with the number of surfaces in textures. This observation implies that the rasterization workload of textures could be predicted from the number of constituting surfaces.

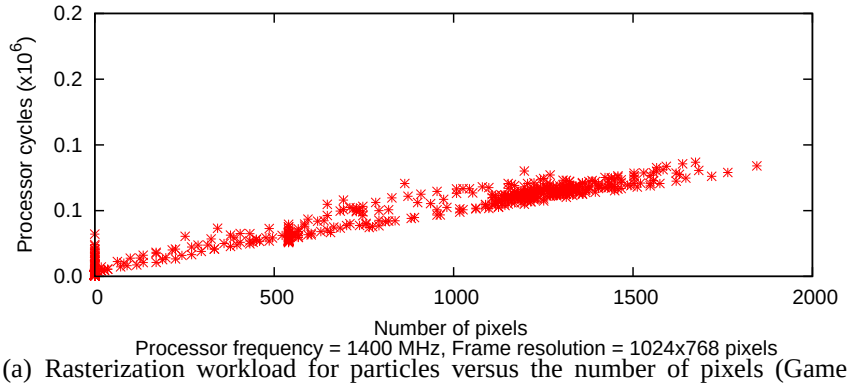
Again, Figure 4.11(b) shows the rasterization workload of textures against the number of surfaces in two different game maps, *Outer Base* and *Installation*. Note that the correlations are consistent across different game maps. This implies that the correlation from a game map could be applied to predict the rasterization workload of textures in a different game map.

4.3.4 Light Map

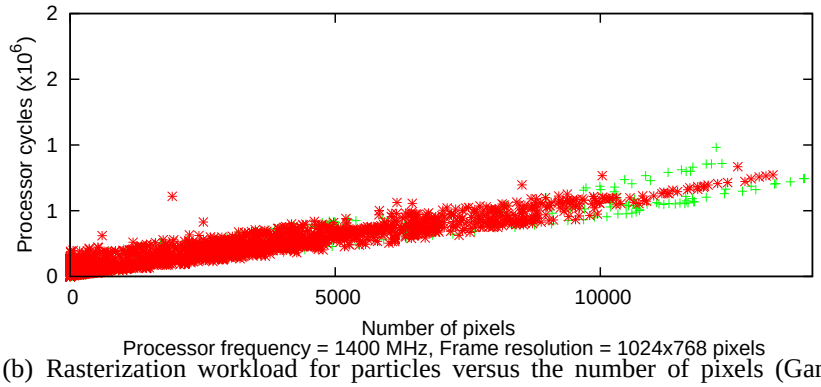
Light maps are used to store pre-calculated lighting information for different scenes in a game. Static light maps in Quake II are low resolution bitmaps which are rendered as multiple *surfaces*. Hence, the workload involved in rasterizing light maps is already included in the workload resulting from rasterizing textures. Therefore it need not be accounted for separately.

4.3.5 Particles

Particles are often used to model small debris resulting from gun shots hitting a target. They are usually generated as a set of 3D *points*. Thus, the number of *pixels* of the points is used to parameterize the rasterization workload of particles. This work-



(a) Rasterization workload for particles versus the number of pixels (Game Map: *Command Center*).



(b) Rasterization workload for particles versus the number of pixels (Game Maps: *Outer Base* and *Installation*).

Figure 4.12: Particles.

load scales almost linearly with the number of pixels as shown in Figure 4.12(a), and the scaling factor again remains consistent across game maps (*Outer Base* and *Installation*), as shown in Figure 4.12(b). These observations imply that the rasterization workload of particles could be predicted by the number of constituting pixels. The correlation between the rasterization workload and the number of pixels of particles could be applied across the different game maps.

The contributions of the abovementioned five types of primitives to the rasterization workload are summarized in Figure 4.13 (the workload resulting from light maps is not shows for reasons already described). Rasterizing textures and light maps is clearly the most computational expensive. Lastly, note that apart from these five primitives, *sprite*

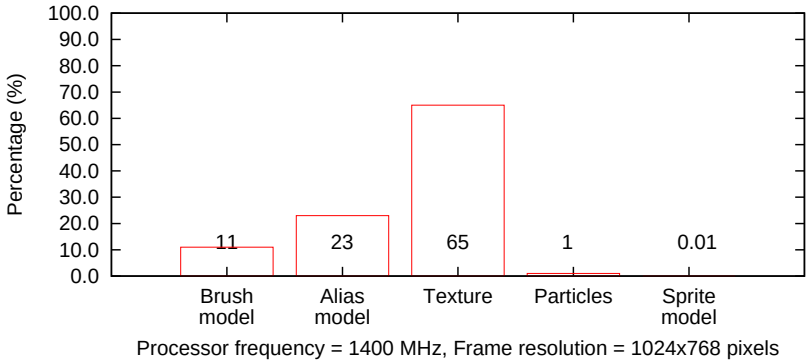


Figure 4.13: Contributions of the different objects in a frame towards the rasterization workload.

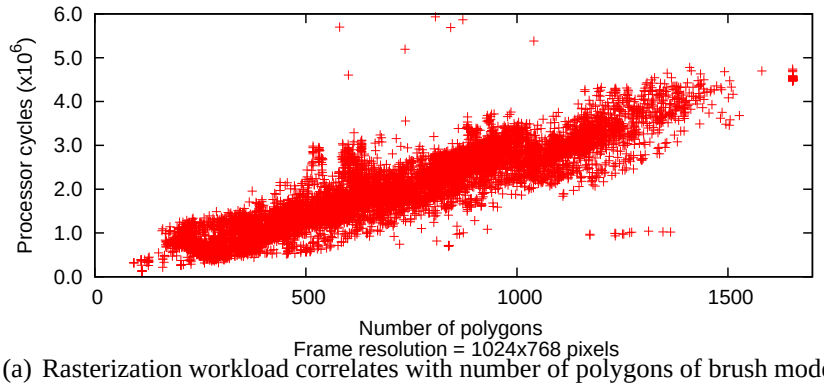
models are also responsible for a small fraction (almost negligible) of the rasterization workload. These models are often used to represent dust particles or special effects like sparkles.

4.3.6 Correlation Functions

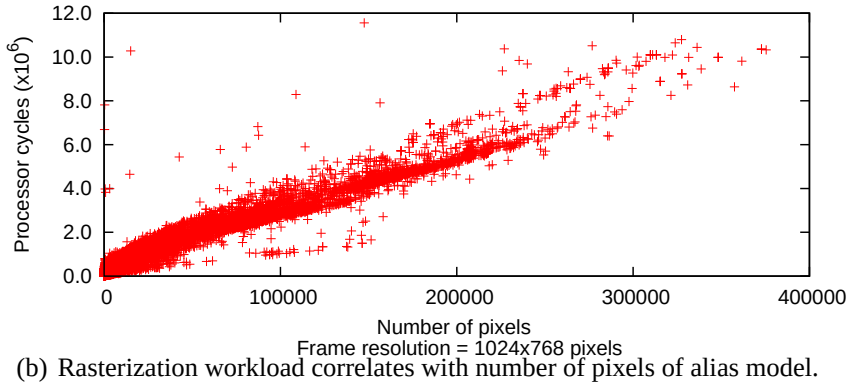
While computing, or rather predicting, the rasterization workload of the different primitives constituting a frame, several correlations between the rasterization workload of each type of primitives and their detailed constitution need to be computed. We observe that a (nearly) linear correlation for most of the primitives such as brush model, alias model, texture and particles, as shown in Figure 4.14, where the workload increases along with the detailed constitution of each type of primitives.

We generate the correlation coefficients with *linear regression models* in Matlab fitting toolkit [2]. Table 4.1 summarizes the coefficients of each linear function $f(x) = x \cdot p_1 + p_2$ for brush model, alias model, texture, particles and game frame, where x is the detailed constitution and $f(x)$ is the corresponding workload in processing cycles, p_1 and p_2 are two coefficients of the linear function. In the proposed DVS scheme by exploiting frame structure, the acquired function coefficients are employed

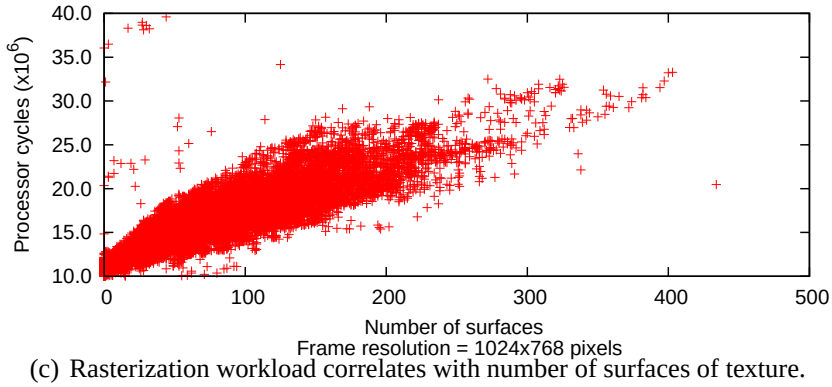
[2] http://www.mathworks.com/access/helpdesk/help/toolbox/curvefit/



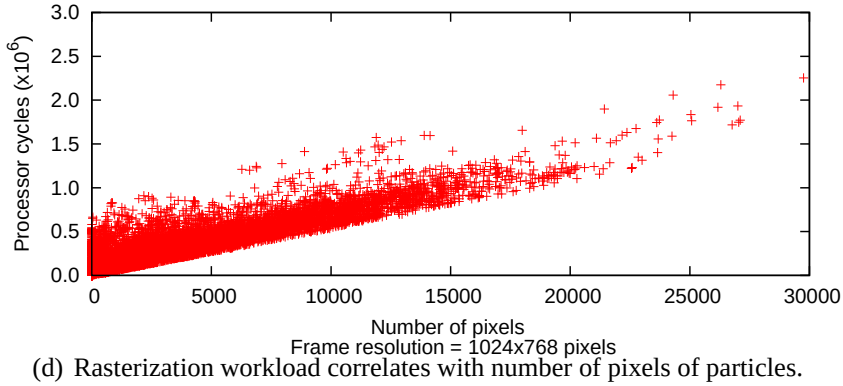
(a) Rasterization workload correlates with number of polygons of brush model.



(b) Rasterization workload correlates with number of pixels of alias model.



(c) Rasterization workload correlates with number of surfaces of texture.



(d) Rasterization workload correlates with number of pixels of particles.

Figure 4.14: Linear correlations of individual primitives - brush model, alias model, texture and particles.

to compute the corresponding workload once the detailed constitution of each primitive is obtained. Furthermore, the correlation functions generated from one game map could be extended to compute the corresponding workload in other game maps, on account of consistently-consumed workload for detailed constitution of each type of primitives, as aforementioned.

	$\mathbf{p_1}$	$\mathbf{p_2}$
Brush model	2763	5.887×10^4
Alias model	28.78	1.507×10^5
Texture	5.848×10^4	1.157×10^7
Particles	60.2	1.136×10^4
Game frame	1.024	4.714×10^6

Table 4.1: Coefficients in the linear functions for Quake II (demo file: crusher.dm2).

4.4 Workload Prediction

In this section we describe the frame structure-based workload prediction scheme that forms the basis of our DVS algorithm. It may be noted that a significant component of the rendering task involves *rasterizing* objects on the screen. Our experimental results suggest that the total workload generated from processing a frame is almost linearly correlated with its rasterization workload, as shown in Figure 4.7. Hence, we predict the total workload by estimating the rasterization workload of a frame.

4.4.1 Exploiting the Frame Structure

Figure 4.15 shows an overview of the proposed frame structure-based workload prediction scheme. Note that it primarily consists of estimating the rasterization workload for each frame. Towards this, we compute the number of occurrences of the different primitives in a frame (e.g. brush models, alias models and particles) and multiply these with

the workload involved in processing each of these primitives. This is possible because, once again, the workload involved in processing all primitives of the same type almost linearly scales with the number of primitives occurring in the frame. The workload corresponding to a single primitive of any given type is computed in an offline fashion. We have experimentally verified this (nearly) linear correlation for most of the primitives such as brush models, alias models, textures and particles. It may also be noted that the workload corresponding to each of these primitives exhibits sufficient short-term variability, as shown in Figure 4.16. Hence, our proposed scheme performs significantly better than history-based predictors even if they are applied individually to the different primitive-types. In what follows, we describe our rasterization workload estimation in further details.

For each frame, once the current *view frustum* is computed based on the user input, the number of occurrences of the different primitives in the frame is estimated (e.g. the number of brush models, alias models, etc.). Further, for each of these primitives, its detailed constitution is also computed. For each brush model, this amounts to computing its number of constituent polygons. For each alias model it amounts to computing its number of pixels, for each texture model its number of surfaces, and for each particle its number of pixels. Based on offline simulation, the workload associated with each of the different primitives is parameterized by their constitution as elaborated in Section 4.3.6. For example, these correlations include the workload associated with processing an alias model with m pixels for different values of m. Let us assume that $c(m)$ is the number of processor cycles requires to rasterize any *one* alias model with m polygons. Then $c(m)$ is calculated by the above-mentioned correlation. To compute the rasterization workload for *all* alias models in a frame, let us assume that $B(m)$ is

the number of alias models in this frame with m pixels. Then the total rasterization workload for all the alias models in the frame is equal to $\sum_{m=1,\ldots,\infty} c(m) \times B(m)$ processor cycles (where ∞ is the maximum possible number of pixels in any alias model). This procedure is followed for all the different primitives, with the exception of texture models (which is explained below).

The abovementioned estimation process clearly has a computational overhead, which turns out to be prohibitive in the case of textures. Textures are drawn on brush models and hence their number of constituent surfaces cannot be determined unless the associated brush models are rasterized. However, rasterizing brush models solely for workload estimation purposes is prohibitively expensive. Hence, as shown in Figure 4.15, the workload associated with rasterizing texture models in a frame is estimated using a history-based prediction scheme. Although this is not as accurate as frame structure-based predictions, this loss of accuracy is unavoidable. However, using frame structure-based estimation, at least for the other primitives like brush models, alias models and particles, reduces the overall error compared to using history-based predictors for all the primitives. Further, this mix of two different estimation schemes results in a good tradeoff between prediction accuracy and computational overhead.

4.5 Summary

In this chapter we proposed the frame structure-based DVS scheme targeted towards graphics-intensive game applications. Using the proposed framework of workload characterization, the rasterization workload of a frame is computed as the sum of the rasterization workload of its constituent primitives. The computed workload is then appro-

priately scaled to *predict* the total processing workload of the frame, which is used to adjust the processor's voltage and frequency.

We implemented our scheme on a Intel Pentium based laptop running Windows XP and a Intel XScale PDA running Window Mobile. Chapter 6 shows the attractive results in energy saving and output quality, compared to known DVS algorithm that were developed for video decoding applications.

Further, we explore the possibilities of designing hybrid workload predictors that combine the PID controller-based (which has lower prediction overhead) and the proposed frame structure-based predictors in the following chapter.

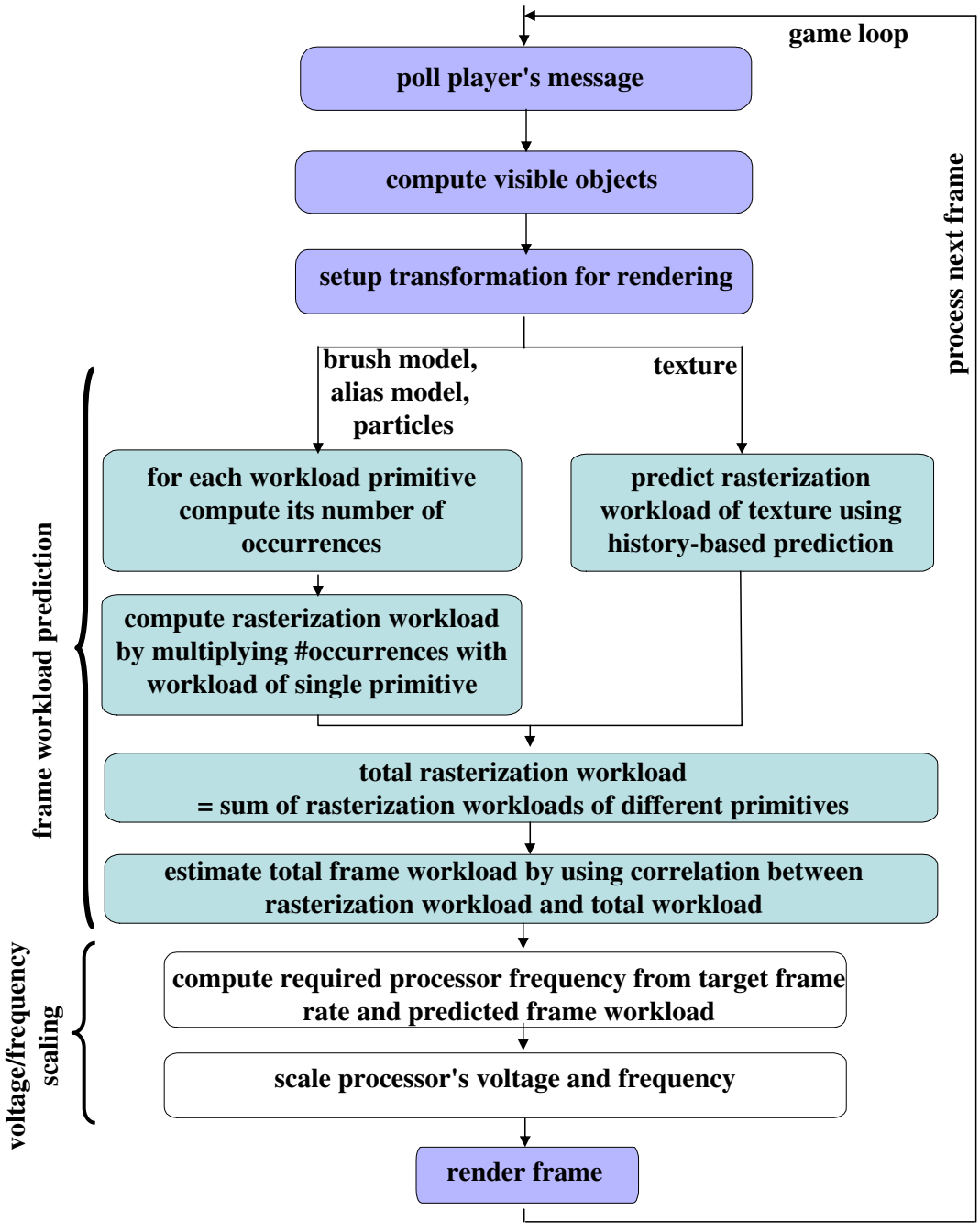
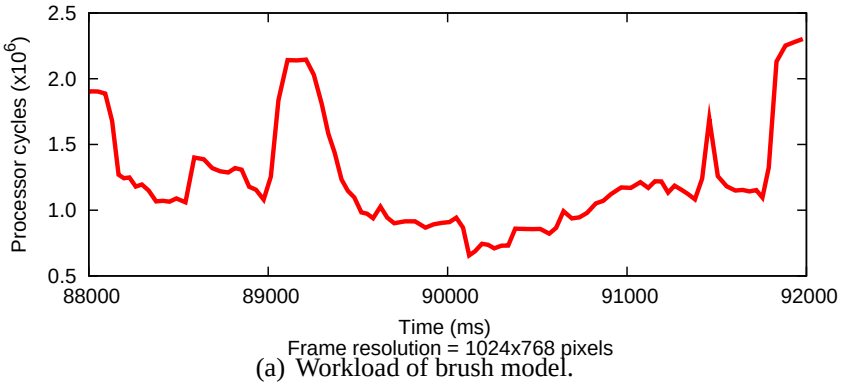
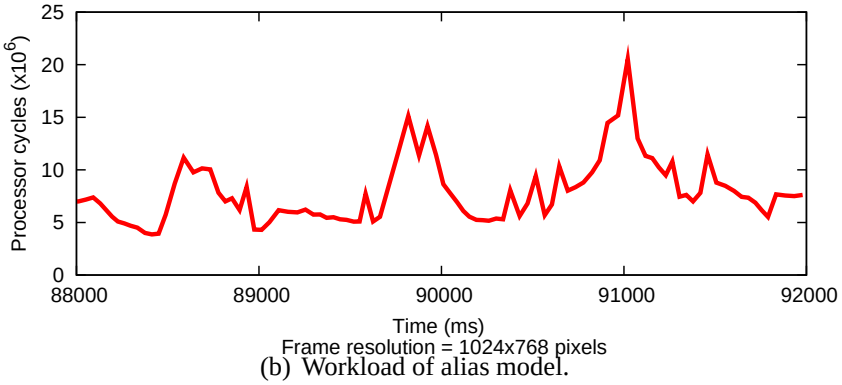


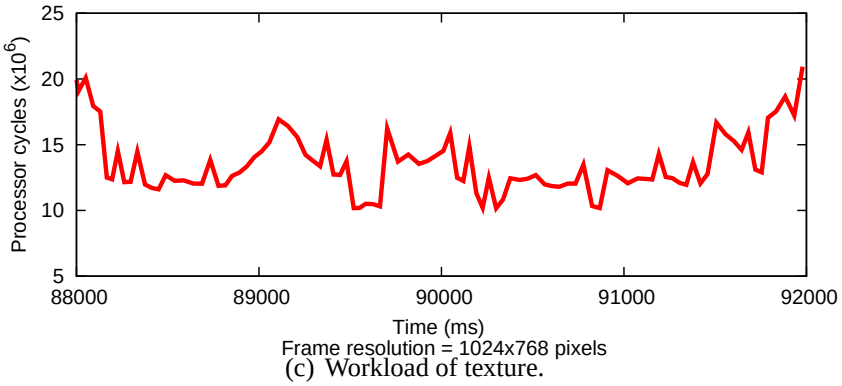
Figure 4.15: Overview of the frame structure based workload prediction scheme.



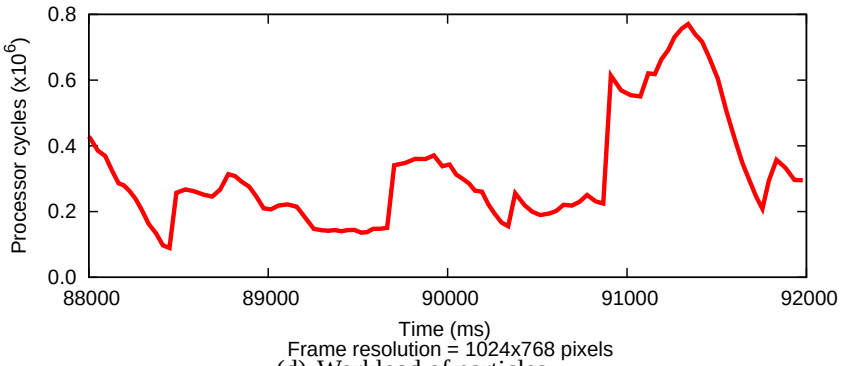
(a) Workload of brush model.



(b) Workload of alias model.



(c) Workload of texture.



(d) Workload of particles.

Figure 4.16: Rasterization workload variations for individual primitives – brush model, alias model, texture and particles.

Chapter 5

A Hybrid DVS Scheme

In this chapter, we propose a novel DVS scheme based on an accurate prediction of the rendering workload of a current game scene. Based on the observation that there exist two types of workload variations in First Person Shooter games such as Quake II, we compute the voltage/frequency setting for any game scene using a hybrid combination of two different techniques: (i) adjusting the workload prediction using a control-theoretical feedback mechanism, and (ii) analyzing the graphical objects in the current game scene by parsing the corresponding frame. In our experiments, we obtain significant power savings while maintaining high frame rates.

5.1 Introduction

A game application, similar to a video decoder, runs in an infinite loop and processes a sequence of game frames that have to be rendered and displayed on the screen. Our proposed DVS algorithm for such applications is integrated into this game loop. The critical step is to accurately predict the workload of a frame, which is followed by a ”voltage/frequency scaling logic” (that we describe in more detail later in this paper).

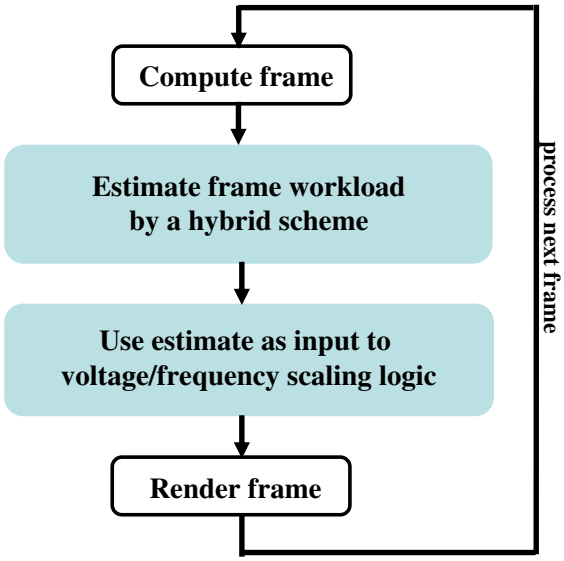


Figure 5.1: DVS in a game loop.

An overview of this DVS algorithm is shown in Figure 5.1.

As introduced in Chapter 4, we develop the frame structure-based DVS algorithm in which each game frame is parsed to identify its constituent objects, based on which the workload associated with rendering the frame is estimated. This estimate, along with the target frame rate is used to determine the frequency (and voltage) of the processor running the game application.

Note that although our *frame structure-based prediction* works well (and outperforms control-theoretic prediction schemes) for game plays where the frame workload exhibits sufficient variability, often there are sequences of frames with relatively constant rendering workload. For such frames, control-theoretic prediction schemes happen to perform better. To take advantage of both these schemes, in this chapter we propose a *hybrid* workload prediction scheme, where we keep on switching between the two schemes based on their relative performance.

In the hybrid workload prediction scheme, the workload associated with rendering a game frame is roughly the sum of the workloads generated by processing the dif-

ferent objects (e.g. *brush models, alias models,* etc.) constituting the frame. Each of these workload components is estimated separately and then summed up to compute the processing workload of a frame. Whenever the estimation error for any object type is beyond a certain level, the scheme switches to a different prediction mode for that object type. In other words, our hybrid scheme is applied at an *object* level rather than at the frame level. Naturally, this leads to a more accurate workload prediction (albeit at the cost of slightly higher computational overhead) compared to applying this scheme at the frame level.

An illustrative example: Figure 5.2 shows the workload variation for the *particles* object type in an excerpt from a game play, and the corresponding run of our workload prediction scheme. The horizontal axis shows the time stamp of game frame in millisecond. The workload is measured in terms of number of processor cycles. Note that the first switch from the control-theoretic scheme to the frame structure-based prediction scheme occurs at 92142 millisecond, when the real frame workload exhibits a significant increase from a flat profile. Once the profile again becomes relatively flat from 92585 millisecond, the relative performance of the frame structure-based scheme degrades and the scheme switches back to the control-theoretic scheme. The final change from the control-theoretic to the frame structure-based scheme (at 92765 millisecond) again occurs when the workload profile shows a relatively large dip. In summary, this scheme keeps on switching between the two workload prediction schemes based on their relative prediction errors. For a relatively flat workload profile, as seen in Figure 5.2, the prediction accuracy of the control-theoretic scheme dominates. Whenever the workload profile exhibits a significant variation, the frame structure-based prediction scheme takes over.

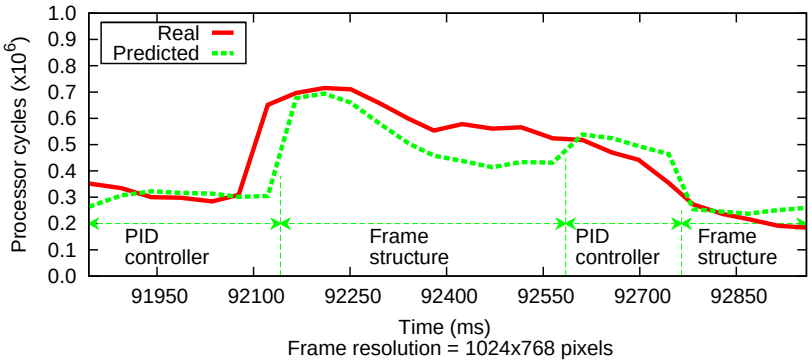


Figure 5.2: Sample run of the hybrid scheme.

The rest of this chapter is organized as follows. In Section 5.2 we describe our hybrid workload prediction scheme, followed by an optimal control method in the hybrid workload prediction in Section 5.3. Section 5.4 briefly explains other possible hybrid schemes, apart from our hybrid workload prediction scheme in the thesis. We evaluate the performance of our hybrid DVS scheme in Section 5.5 and finally we conclude in Section 5.6.

5.2 Workload Prediction

In this section, we present our hybrid frame workload prediction which combines the two techniques described in Chapter 3 and 4, i.e. (i) adjusting the workload prediction using a control-theoretic feedback mechanism (viz. PID controller), and (ii) analyzing the graphical objects in the current game scene by parsing the corresponding frame (viz. the frame structure scheme).

5.2.1 Workload Variation

As we already discussed, game workload exhibits a large degree of variability. We have experimented with various game plays and workload prediction schemes. In particular,

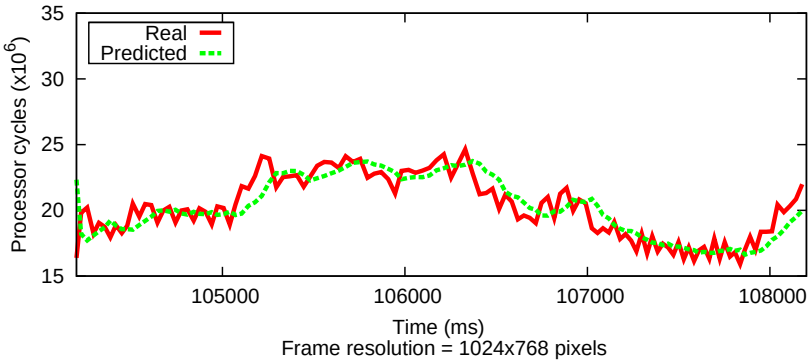


Figure 5.3: Workload prediction using a history-based predictor for a frame sequence with relatively low workload variability.

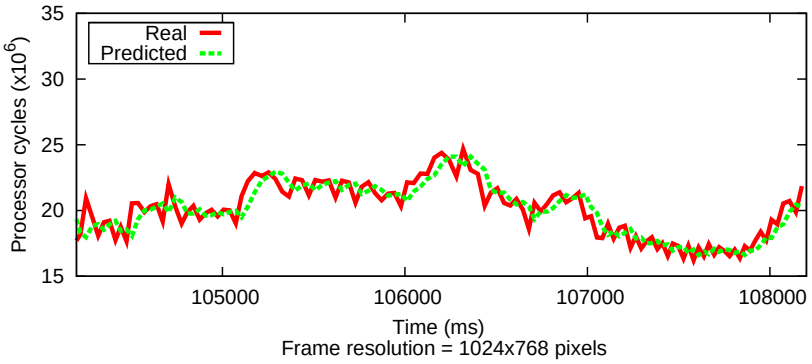


Figure 5.4: Workload prediction using a PID controller-based predictor for a frame sequence with relatively low workload variability.

we have investigated three different prediction schemes: (i) history-based predictors, where the workload of a game frame is estimated to be the average of the workloads of a certain number of previous frames, (ii) PID controller-based predictors (as described in Chapter 3), and (iii) frame structure-based predictors (as described in Chapter 4). Our results show that for frame sequences which exhibit relatively low workload variation, the first two schemes outperform (iii). However, there also exists frame sequences with high workload variability, and for such sequences a frame structure-based predictor is certainly better. In fact, this observation is the main motivation behind devising a hybrid workload prediction scheme.

Figure 5.3, 5.4 and 5.5 show comparisons of the three abovementioned schemes for a sequence of game frames with relatively low variability respectively. Clearly, the PID

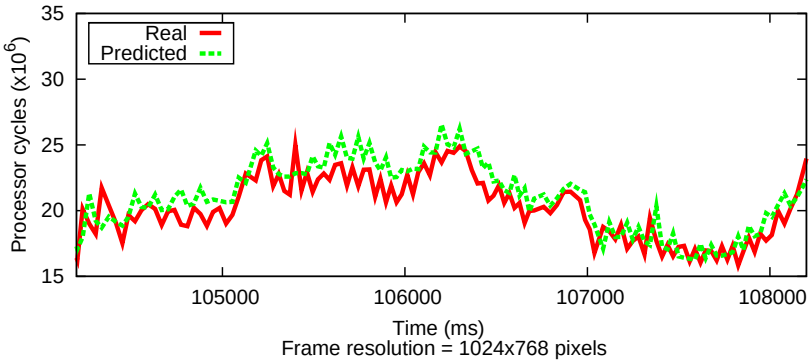


Figure 5.5: Workload prediction using a frame structure-based predictor for a frame sequence with relatively low workload variability.

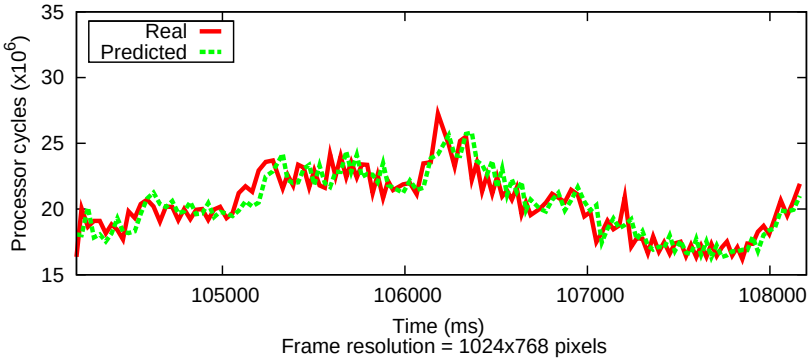


Figure 5.6: Workload prediction using a history + frame structure-based predictor for a frame sequence with relatively low workload variability.

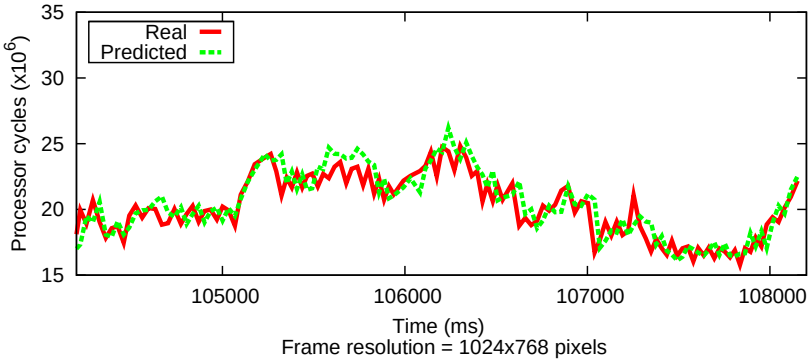


Figure 5.7: Workload prediction using a PID controller + frame structure-based hybrid predictor for a frame sequence with relatively low workload variability.

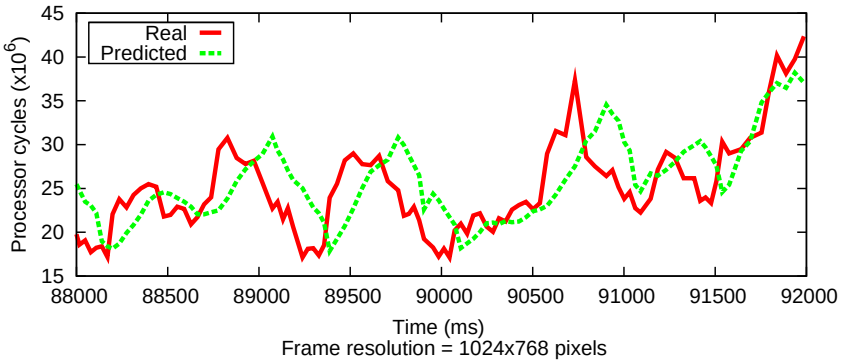


Figure 5.8: Workload prediction using a history-based predictor for a frame sequence exhibiting high workload variability.

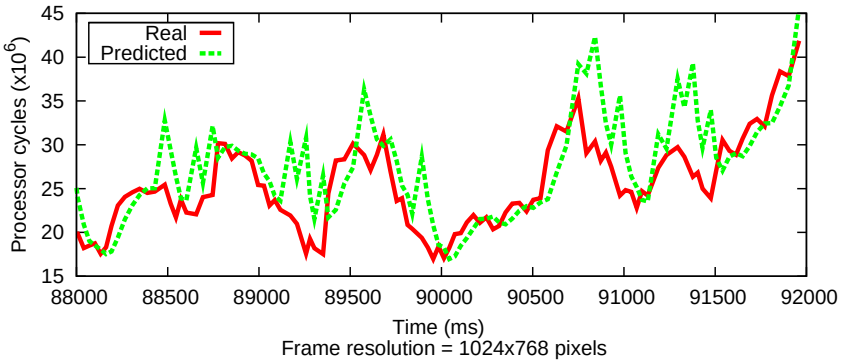


Figure 5.9: Workload prediction using a PID controller-based predictor for a frame sequence exhibiting high workload variability.

controller-based prediction scheme outperforms the other two, with the frame structure-based predictor giving the worst result. Figure 5.6 and 5.7 show the performance of two hybrid predictors, which are explained in the followings. Obviously, the two hybrid schemes outperform the history-based, the PID controller and the frame structure-based schemes for the sequence of frames with low variability.

Figure 5.8, 5.9 and 5.10 show the same comparisons of the history, PID controller, and frame structure-based predictors for a sequence of frames exhibiting a high variability in their workload. It is easy to see that for this sequence of frames, the frame structure-based predictor outperforms the first two. Figures 5.11 and 5.12 show the performance of two hybrid predictors, which perform even better.

To be more specific, Figure 5.13, 5.14 and 5.15 show the respective prediction

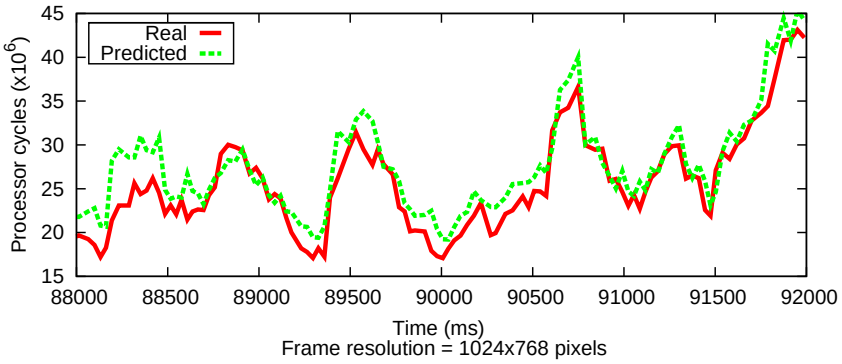


Figure 5.10: Workload prediction using a frame structure-based predictor for a frame sequence exhibiting high workload variability.

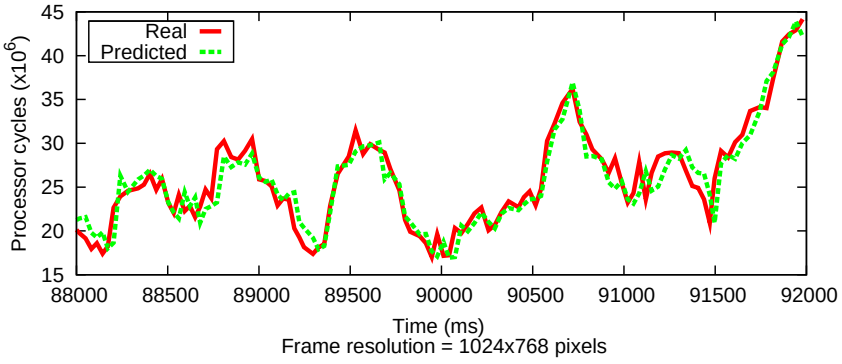


Figure 5.11: Workload prediction using a history + frame structure-based hybrid predictor for a frame sequence exhibiting high workload variability.

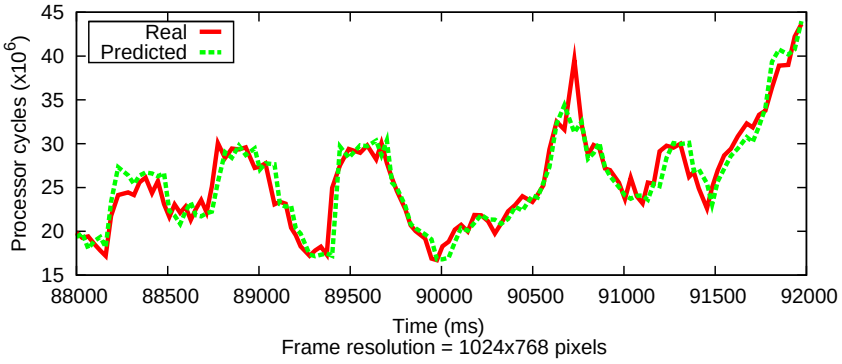


Figure 5.12: Workload prediction using a PID controller + frame structure-based hybrid predictor for a frame sequence exhibiting high workload variability.

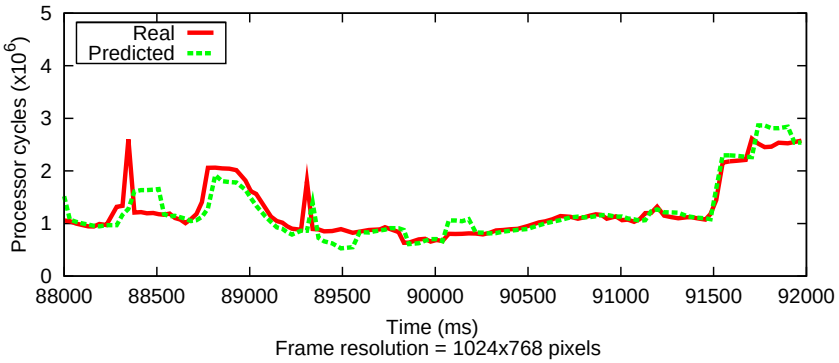


Figure 5.13: Workload prediction using a PID controller + frame structure-based hybrid predictor for brush models.

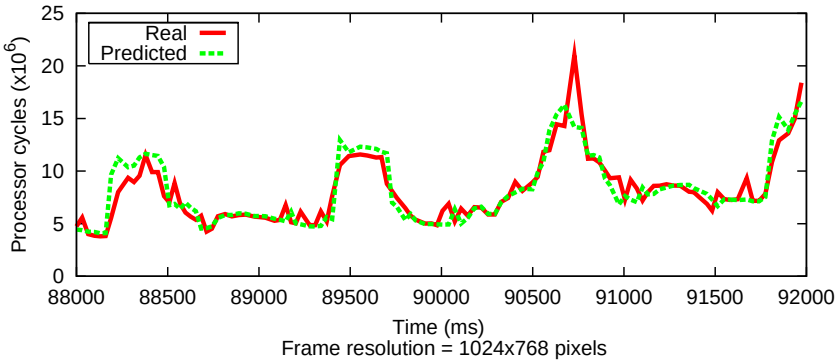


Figure 5.14: Workload prediction using a PID controller + frame structure-based hybrid predictor for alias models.

for brush models, alias models and particles with the hybrid scheme by combining the PID controller-based and the frame structure-based techniques. Taking particles as an example, we compare the prediction results with the history and the frame structure-based prediction schemes (as shown in Figures 5.16 and 5.17), the proposed hybrid scheme provides much better prediction for particles in Figure 5.15.

We describe the details of these two predictors in the following sections. Figure 5.18 summarizes the above observations. It shows how the relative prediction errors of the different predictors change as the workload variability of a sequence of frames increases. When the frame workload approaches to a relatively high variability, any predictors appeal to relatively larger workload prediction errors than their errors when the frame workload exhibiting relatively low variability. Further, in the case of frame work-

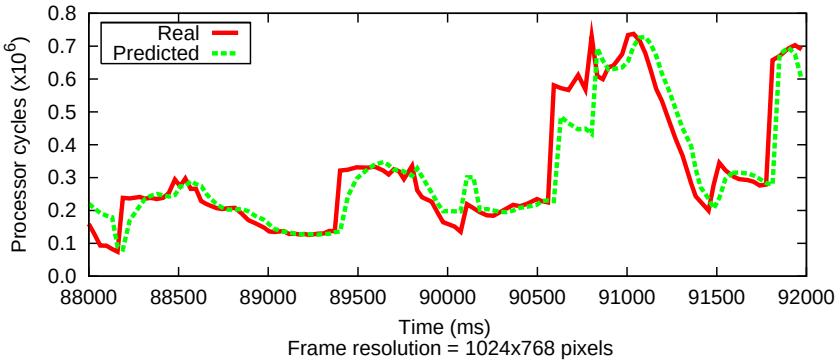


Figure 5.15: Workload prediction using a PID controller + frame structure-based hybrid predictor for particles.

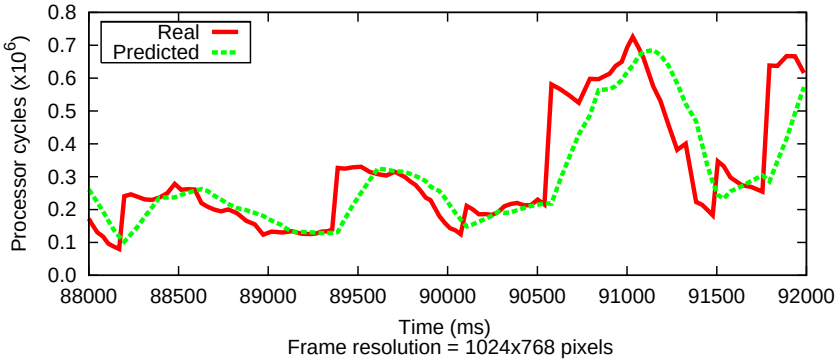


Figure 5.16: Workload prediction using a history-based predictor for particles.

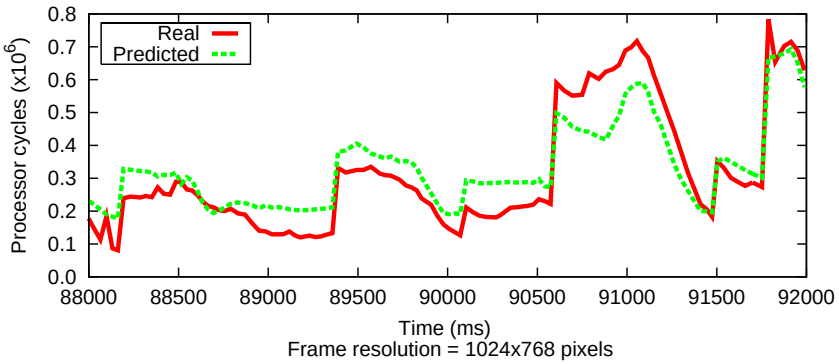


Figure 5.17: Workload prediction using a frame structure-based predictor for particles.

load with a high variability, usually the frame structure-based predictor generates less prediction error than the history/PID controller-based predictor. In contrast, in the case of frame workload with a low variability, the history/PID controller-based predictor has less prediction error than the frame structure-based predictor.

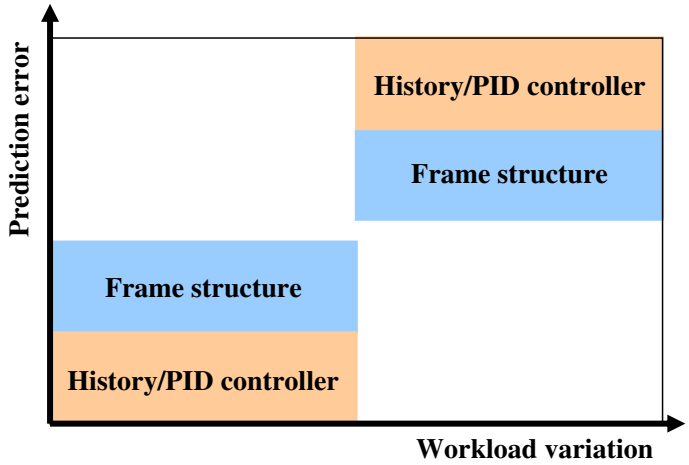


Figure 5.18: Workload prediction error versus variability.

5.2.2 Prediction Mode Switching

Figure 5.19 shows an overview of our proposed hybrid workload prediction scheme. As mentioned before, the total workload involved in processing a frame is made up of the sum of the rasterization workloads of the different primitives that constitute the frame and the workload associated with tasks such as collision detection, AI, simulation of game physics and particle systems. Our prediction scheme estimates the rasterization workload and scales it appropriately to obtain the total frame workload (which is almost linearly proportional to the rasterization workload; see Section 4.4). Figure 5.19 shows the workload estimation scheme for a single primitive type (e.g. brush models). The same scheme is followed for other types of primitives (e.g. alias models, particles), with the exception of textures. As explained in Section 4.4, the number of constituent

surfaces of textures cannot be determined unless the associated brush models are rasterized. Hence, the estimation process by exploiting the frame structure turns out to be prohibitive in the case of textures. In order to reduce the estimation overhead of textures, we avoid repetitively rasterizing associated brush models by using a history-based prediction scheme. Such mix of two different estimation schemes results in a good tradeoff between prediction accuracy and computational overhead.

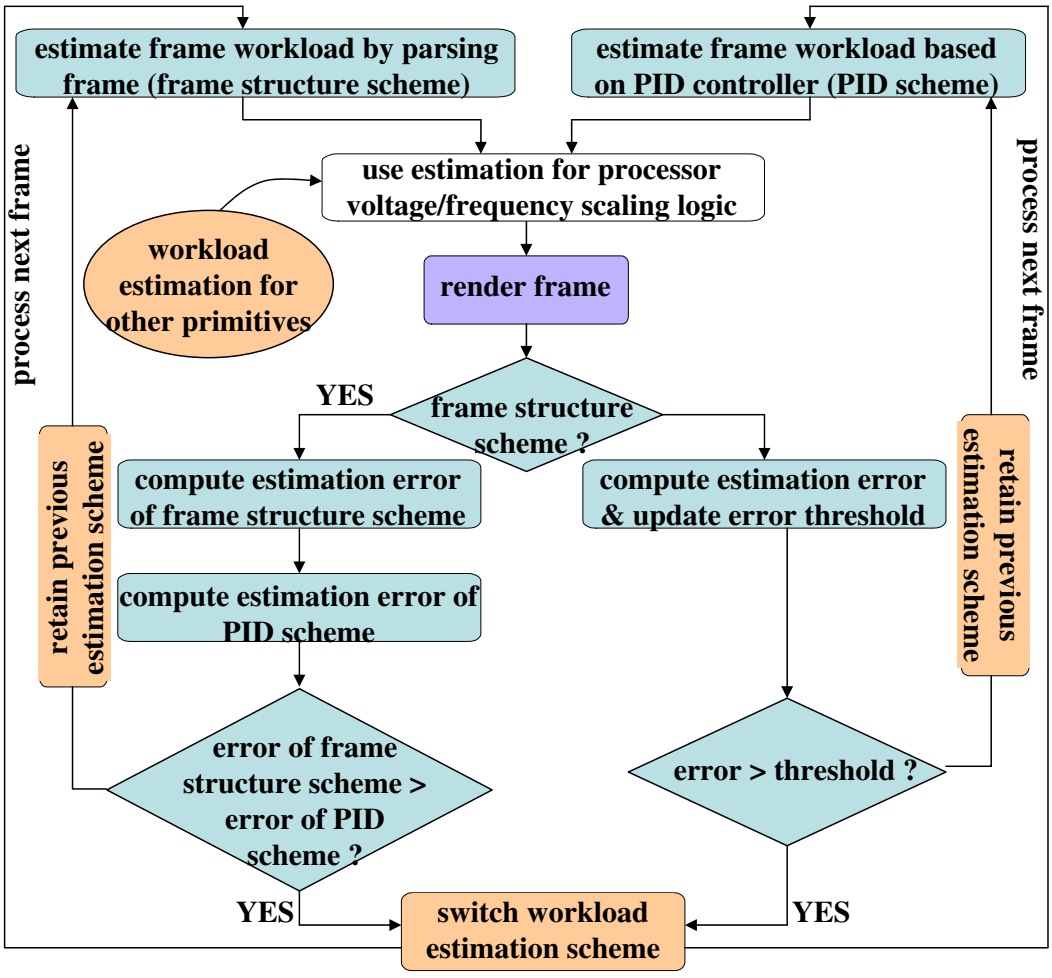


Figure 5.19: Overview of the hybrid DVS algorithm.

In what follows, we describe how to switch between the two prediction schemes (viz. the PID controller-based scheme and the frame structure-based scheme). First, it should be noted that when the frame structure-based scheme is in use, the PID controller-based scheme is also kept active, but its estimation result is not used for

voltage/frequency scaling. This incurs a certain (negligible) computational overhead. However, when the PID controller-based scheme is in use, the frame structure-based predictor is switched off because it is computationally expensive. If the workload of the last frame is estimated using the frame structure-based predictor, then our proposed algorithm computes two *prediction errors*; one incurred by the frame structure-based predictor and the other incurred by the PID controller. If the former error is larger than the latter then a prediction mode switch is enabled (i.e. the scheme switches to the PID controller-based predictor), otherwise the same predictor is retained.

On the other hand, if the PID controller is currently in use, the mode switching decision is based on whether the prediction error for the last frame is greater than a certain *threshold error*. However, this threshold error is not statically predefined, but is constantly computed (or updated) at runtime. This threshold error depends on the frames whose workload was last predicted using the frame structure-based predictor. Let these be the α-th to the β-th frames. In other words, α and β are such that the workload of the $(\alpha - 1)$-th and $(\beta + 1)$-th frames were predicted using the PID controller-based predictor, and the workload of the α-th to the β-th frames were predicted using the frame structure-based predictor. The *threshold error* is equal to $\min(\frac{\sum_{i=\alpha}^{\beta} \epsilon_{i,o}}{\beta-\alpha+1}, \frac{\sum_{i=\alpha}^{\beta} \epsilon_{i,c}}{\beta-\alpha+1}) + \tau|\frac{\sum_{i=\alpha}^{\beta} (\epsilon_{i,o}-\epsilon_{i,c})}{\beta-\alpha+1}|$, where $\epsilon_{i,o}$ and $\epsilon_{i,c}$ denote the prediction errors of the i-th frame as incurred by the frame structure-based and the PID controller-based schemes respectively. The value of $0 < \tau \leq 1$ has to be appropriately chosen. With the PID controller currently in use, if the prediction error of a frame exceeds the *threshold error*, then a mode switch is enabled.

Once again, note that this scheme is individually applied to all the different primitives constituting a frame, except for textures, whose workload is always estimated

using a PID controller-based scheme. The total estimated workload of a frame, along with the desired frame rate is used to compute the target processor frequency, which is then mapped onto a discrete voltage/frequency level that is supported by the processor. In summary, our hybrid predictor switches between two prediction schemes based on their relative performance. Finally, it may be noted that the PID controller-based predictor may be replaced by a simple history-based predictor (that is described in Section 5.2.1) with everything else remaining the same. This would result in a history + frame structure-based hybrid predictor.

5.3 Optimal PID Controller

As discussed above, the PID controller-based mode is evoked for frame workload with a relatively low degree of variability. Notice that even in this mode, it is possible to categorize the frames into different groups based on their workload variation (measured by the standard deviation). In our experiments, whenever the workload prediction is switched to the PID controller-based mode, we compute the standard deviations of real workload in previous certain frames for each type of primitives. Their workload is compared with the *standard deviation thresholds* and categorized into either *medium* workload variation or *flat* workload variation.

Notice that the same process is individually executed for each type of primitives except for textures, i.e., brush models, alias models and particles. For different categories of workload variation, different sets of the proportional, integral and derivative parameters are adaptively applied to the predictor. The total estimated workload of a frame, along with the desired frame rate is used to compute the target processor frequency, which is then mapped onto a discrete voltage/frequency level that is supported by the processor.

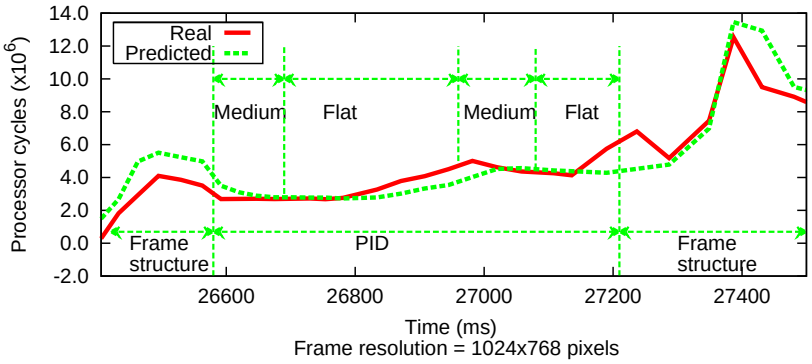


Figure 5.20: Workload transition for alias models in the optimal PID controller.

An illustrative example: Figure 5.20 shows the workload variation for the *alias models* in an excerpt from a game play, and the corresponding run of the optimal PID controller in the hybrid workload prediction scheme. The horizontal axis shows the time stamp of game frame in millisecond. The workload is measured in terms of number of processor cycles. Note that the hybrid prediction first switches from the frame structure-based to the PID controller-based scheme, when the real frame workload exhibits a decrease in variation from a fluctuant profile (at 26592 millisecond). Once the profile again becomes relatively fluctuant from 27237 millisecond to the end, the relative performance of the PID controller-based scheme degrades and the scheme switches back to the frame structure-based scheme.

Within the first phase with the PID controller-based scheme, we further differentiate the frames into different workload variations which are indicated by either *flat* or *medium*, according to their standard deviations. From 26592 to 26675 millisecond and from 26982 to 27058 millisecond, the frame workload exhibits medium variation. While from 26702 to 26945 and from 27108 to 27190 millisecond, the frame workload exhibits flat variation as shown.

	***Flat* variation ($\times 10^6$ cycles)**	***Medium* variation ($\times 10^6$ cycles)**
Brush model	$[0, 0.1)$	$[0.1, 0.3)$
Alias model	$[0, 0.5)$	$[0.5, 0.8)$
Particles	$[0, 0.02)$	$[0.02, 0.07)$

Table 5.1: Standard deviation thresholds for different groups of workload variations.

5.3.1 Parameters

Table 5.1 shows the specified standard deviation thresholds of *flat* and *medium* workload variations for brush models, alias models and particles. For brush models, the workload with standard deviation below 0.1 million cycles is defined as *flat* and from 0.1 up to 0.3 million cycles is defined as *medium*. For alias models, the workload with standard deviation below 0.5 million cycles is defined as *flat* and from 0.5 up to 0.8 million cycles is defined as *medium*. While for particles, the workload with standard deviation below 0.02 million cycles is defined as *flat* and from 0.02 up to 0.07 million cycles is defined as *medium*.

In order to obtain the optimal parameters of PID controller predictors for the specified *medium* and *flat* workload, we segment the long demo file and conduct the prediction process for each segment. It is found that for a sequence of frames with *medium* workload variation, the values of $K_p = 0.5$, $I = 28$ and 0.00001 work well. For a sequence of frames with *flat* workload variation, the values change to $K_p = 0.3$, $I = 28$ and 0.01, by giving a less sensitive PID controller.

5.3.2 Results

Figure 5.21 shows the comparison of workload prediction by using an optimal PID controller + frame structure-based hybrid predictor for a 4-second demo file. Note that

the prediction with optimal PID controller in the hybrid scheme (Figure 5.21) does not improve the result too much, compared with the hybrid without optimal controller (Figure 5.12), as the workload shown in Figure 5.21 is rather fluctuant so that there are few frames in PID controller mode.

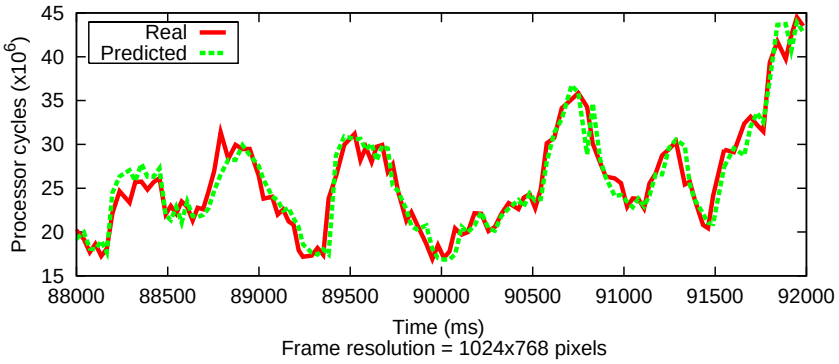


Figure 5.21: Workload prediction using an optimal PID controller + frame structure-based hybrid predictor for a frame sequence exhibiting high workload variability.

In the case when there are more frames with a relative low workload variability, i.e. as shown in Figure 5.22, the optimal PID controller in the hybrid prediction scheme improves the prediction, compared with the result in Figure 5.7.

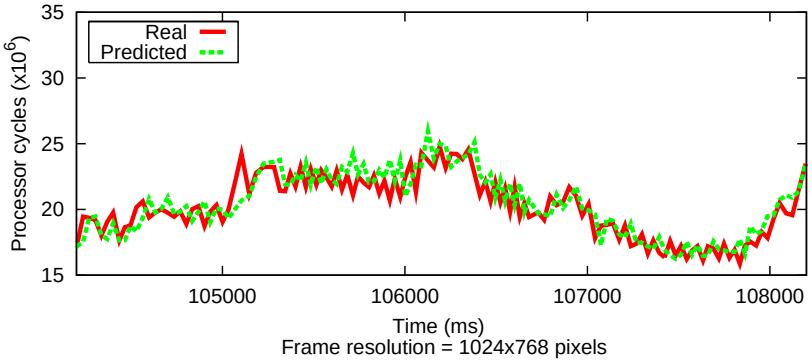


Figure 5.22: Workload prediction using an optimal PID controller + frame structure-based hybrid predictor for a frame sequence with relatively low workload variability.

5.4 Discussion

In order to apply the optimal PID controller for each type of primitive objects, the differentiation of workload variation is conducted based on the historical standard deviation, which does not capture the variation very well. In our experiments, we notice that the impacts of different sets of PID parameters on each type of primitive objects are not as noticeable as their impacts on total frame workload (as shown in Chapter 3), as each type of objects contributes to part of the total workload. Although such optimal PID controller provides better results for the frames with a low workload variability, the 3D First Person Shooter games usually generate rather fluctuant workload. Therefore, with such concerns and observations, in our following experiments, we keep using the original hybrid scheme by combining the frame structure-based and the static PID controller-based techniques.

As discussed so far, in our work, we apply the hybrid scheme to each type of primitive objects. In other words, either the frame structure-based or the PID controller-based mode is applied to each type of objects respectively, based on its relative prediction error. However, as shown in Figure 5.7 and Figure 5.12, the total workload of game frame exhibits a large degree of variability. Instead of applying such sophisticated hybrid DVS scheme to each type of objects, it is possible to take the game frame as a whole and apply either the frame structure-based or the PID controller-based mode to the entire game frame, without explicitly taking into account of the workload variation of each type of objects. In such hybrid scheme, the prediction overhead in the frame structure-based mode will not be reduced, since this scheme still obtains the occurrences of each representative objects in the game frame as usual.

5.5 Prediction Accuracy and Overheads

In this section, we compare the performance of the following predictors in terms of the overhead incurred and the prediction accuracy.

- `History`: The history-based predictor that estimates the workload of a frame by averaging the actual workload of a certain number of previously processed frames. Let ω_i be actual workload of frame i, then the estimated workload of frame $i+1$ by the history-based predictor is defined as $\bar{\omega}_{i+1} = \frac{\sum_{k=i-l+1}^{i} \omega_k}{l}$, where l is the number of previous frames.

- `PID controller`: The PID controller-based predictor described in Chapter 3.

- `Frame structure`: The frame structure-based predictor described in Chapter 4.

- `Hybrid(history)`: The hybrid predictor that switches between `History` and `Frame structure` predictors.

- `Hybrid(control)`: The hybrid predictor that switches between `PID controller` and `Frame structure` predictors.

5.5.1 Prediction Overhead

As one would expect, both `History` and `PID controller` incur negligible computational overheads. The `Frame structure` workload predictor is computationally more expensive and incurs, on an average, 1.7 million processor cycles per frame on a laptop with an Intel Pentium Mobile processor running Windows XP. Note from Figure 5.8 that the workload generated by processing a frame varies between 15 to 45 mil-

lion cycles. Hence, the computational overhead incurred by the frame structure-based predictor and its hybrid combinations is certainly within the feasible region.

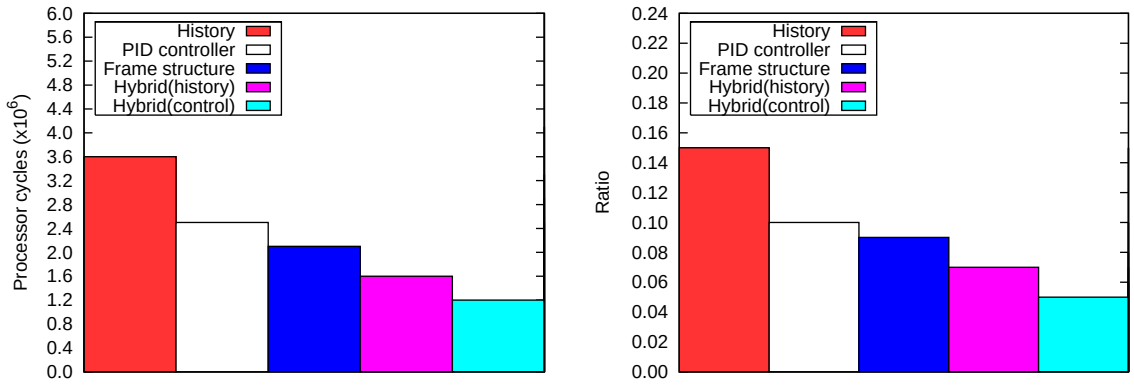
5.5.2 Prediction Accuracy

Figure 5.8, 5.9, 5.10, 5.11 and 5.12 compare the different workload predictors against the actual workload of a sequence of game frames. The excerpt shown in the figures is generated from a 4 second demo file of Quake II (massive1.dm2[1]) running on a laptop with Windows XP. Each point in the figures corresponds to a frame and the horizontal axis refers to the time stamp (in milliseconds) associated with each frame. The vertical axis refers to the total processing workload of a frame in terms of the number of processor cycles. It may be noted that the workload varies between 15 million cycles to 45 million cycles per frame (and therefore offers the possibility of dynamic voltage/frequency scaling).

It it clear from this figure that our proposed hybrid schemes match the profile of the actual frame workload more closely than the history, PID controller, or frame structure-based predictors. To measure the incurred prediction error, we used two metrics: (i) the *absolute prediction error* which is defined as the absolute difference in processor cycles between the actual and predicted workloads, and (ii) the *relative prediction error* which is defined as the ratio between the absolute prediction error and the actual workload. Note that the errors in Figure 5.8 (i.e. using the `History` predictor) turn out to be 3.6 million cycles and 0.15 respectively. These errors drop to 1.2 million cycles and 0.05 respectively using our proposed `Hybrid(control)` scheme (see Figure 5.12).

Figure 5.23 summarizes the absolute and relative prediction errors with different

[1]http://cure.gamepoint.net/files/massive1.zip



(a) Absolute prediction errors.

(b) Relative prediction errors.

Figure 5.23: Comparison of prediction errors with different predictors.

predictors on the laptop. Note that our proposed `Hybrid(control)` predictor results in more than 60% improvement in prediction accuracy over a simple history-based workload predictor.

Furthermore, our proposed `Hybrid(control)` predictor consistently achieves the best prediction even for a longer demo file (160 second), when compared with `History` and `Frame structure` predictors. Figure 5.24 and 5.25 compare the cumulative distribution of absolute and relative prediction errors using different predictors. Note that in Figure 5.24, 90% frames with `Hybrid(control)` have 2.7 million cycles in absolute prediction error, while the absolute errors increase up to 5.5 and 4 million cycles respectively for 90% frames with `History` and `Frame structure` predictors. Hence, `Hybrid(control)` provides much better prediction for the 160 second demo file, compared with `History` and `Frame structure`. While `History` and `Frame structure` have the comparable prediction errors, on account of the exhibition of a large degree of variability in the long-duration game play.

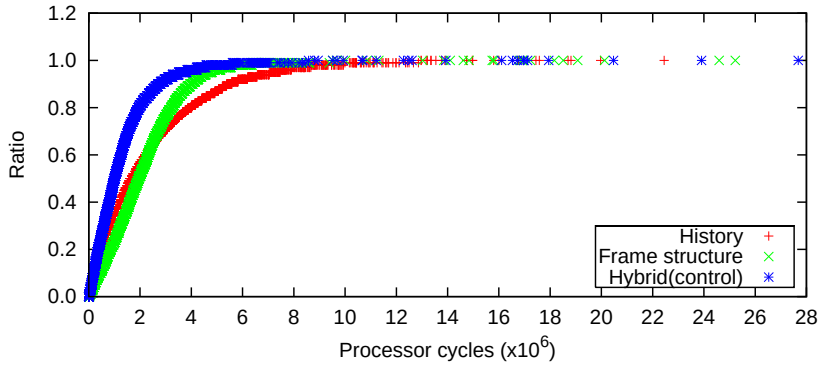


Figure 5.24: Distribution of absolute prediction errors for a 160-second demo file.

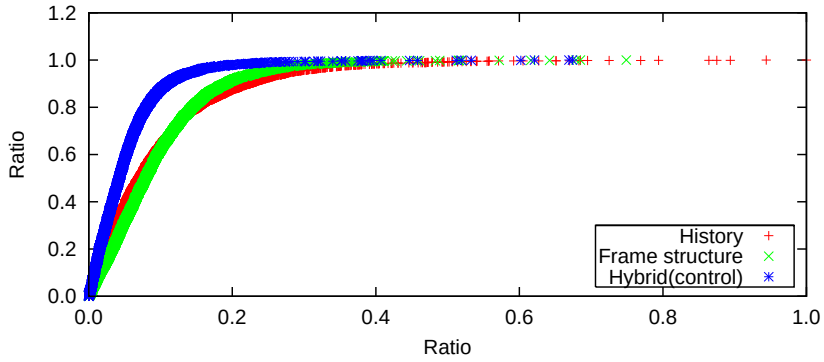


Figure 5.25: Distribution of relative prediction errors for a 160-second demo file.

5.6 Summary

We observed that the frame workload exhibits a larger degree of variability in game applications. Furthermore, we investigated that the frame structure-based DVS scheme works better than the history-based DVS scheme for frames with relatively high workload variability, while the latter works better for frames with relatively low workload variability. Such observations motivate the hybrid DVS scheme for game applications in this chapter.

The proposed hybrid DVS scheme switches prediction between the frame structure-based and the PID controller-based modes based on their relative performance, for each type of workload primitives (i.e. brush model, alias model, particles). Our evaluation shows that this hybrid DVS scheme achieves significant improvement in prediction accuracy over the other predictors, for a 4-second and a 160-second demo file.

Chapter 6

Experimental Evaluation

In this chapter, we introduce our design accounting for several systems and hardware issues, in the implementation of our DVS schemes on multiple real platforms. Next, we evaluate the performance of our proposed DVS schemes on simulation and real platforms. Our results show that the hybrid DVS schemes achieves significant improvements in terms of game quality and power saving.

6.1 Implementation Issues

As discussed before, the predicted workload for each frame is fed into a voltage/frequency scaling logic, which takes into account several hardware and systems issues to decide the voltage/frequency level of a processor for a current frame. In this section, we describe the voltage/frequency logic in detail.

Most processors support a fixed number of discrete operating frequency (and associated voltage) levels. From predicted workload of a game frame and the target frame/display rate, the optimum operating frequency of the processor may be calculated. This calculated frequency needs to be mapped onto the discrete frequency levels

available on the processor in a conservative manner. Further, since scaling a processor's voltage/frequency is associated with a certain overhead – which depends on the processor's microarchitecture and the OS running on top of it – it might not be meaningful to switch the clock frequency at every possible game frame or workload change. Below we address these issues in detail.

6.1.1 Frequency Mapping

A number of previously-proposed algorithms for DVS have assumed the processors were facilitated with fine-granularity frequencies (e.g. [32, 45]). Lu etc. evaluated their system using real workload of video decoding applications on a Compaq iPAQ with a StrongArm SA-1100 processor in [32]. They assumed the processor supported 32 discrete levels in their simulation and claimed that the quantizing frequency scaling factors into discrete levels had a negligible effect on performance since the transition time was small compared to frame decoding time (40 millisecond). In [45], the authors modelled the frequency scaling of XScale processor as 320 steps and modelled that of Transmeta processor as 32 steps.

However, most voltage/frequency-scalable processors only support a fixed number of discrete frequency levels. Hence, we have assumed that only a fixed number of frequency levels are available and the computed optimum frequency is mapped onto the next available higher frequency level. Such a conservative mapping satisfies the workload demands of the game application, at the cost of less than ideal energy savings. However, we have also conducted simulations where we assumed that the processor's frequency is continuously scalable. In Chapter 6 we present a comparison of the energy savings obtained with such ideal settings and where the frequency can only be set to discrete levels.

6.1.2 Frequency Transition

As mentioned before, switching the frequency of a processor is associated with an overhead which depends on the processor's microarchitecture as well as the OS running on top of it. Our experimental results suggest that for the same processor, this overhead is higher in Windows XP compared to Linux. The average transition overhead in Windows XP running on an Intel Pentium Mobile processor is 20 million cycles, i.e., the overhead is 14 milliseconds with the operating frequency set to 1400 MHz.

Hence, to skip unnecessary frequency switches, we have used a *lazy* transition mechanism. Instead of immediately switching the processor frequency whenever the predicted workload of a game frame changes, we defer the switch to the immediate next frame. For instance, if the estimated quantized frequency (i.e. the computed frequency mapped to the frequency level available on the processor) for the current frame i is different from the frequency associated with the previous frame $i-1$, then the switching decision is deferred to the next frame (i.e. frame $i+1$). If the computed quantized frequency of the $(i+1)$-th frame is also different from the frequency of the $(i-1)$-th frame, then the frequency of the processor is changed to the frequency computed for frame $i+1$, otherwise the operating frequency is kept unchanged. Such a *lazy* frequency scaling is resilient to frequent frequency adaptations which might be unnecessary and expensive.

Note that we defer the frequency scaling decision by only one frame. Our experimental results suggest that for our setup this provides satisfactory results. However, if in a different setting, the switching overhead is even higher, then it might be meaningful to defer the switching decision by multiple frames. Finally, note that we switch the operating frequency of the processor with the assumption that the voltage is automati-

cally scaled accordingly (i.e. we do not explicitly control the operating voltage of the processor).

6.2 Settings

We evaluate our proposed DVS scheme by integrating it with the Quake/Quake II game engines running on a number of different platform settings: (i) on a laptop with an Intel Pentium Mobile processor running Windows XP, (ii) on a PDA with an Intel XScale processor running Window Mobile 5, (iii) using a discrete event simulator where the processor has the same power consumption characteristics as in the laptop/PDA, but its frequency transition overhead is assumed to be zero, and (iv) same as (iii) with the additional assumption that the processor's frequency is continuously scalable. Settings (iii) and (iv) are referred to as *simu-disc* (i.e. simulation with discrete frequency levels) and *simu-cont* (simulation with continuous frequency levels) respectively. These two settings represent ideal cases and the results obtained using them give an upper bound on the energy savings that can be obtained using our scheme.

Our motivation behind using the Quake-serial game engines primarily stems from the fact that it is a popular game that can be played on a variety of mobile devices such as PDAs, mobile phones and laptops without additional graphics hardware. Further, this game engine forms the core of a number of other First Person Shooter games (e.g. Hexen II) and its software architecture is representative of those in many other commercially-available games. Finally, the source codes of Quake and Quake II are freely available, which allows for experimentation and appropriate modification.

To ensure reproducibility, we use pre-recorded demo files. Since these demo files keep pre-recorded states and therefore they are not computed during playback, there is

some difference in workload when compared to a real-time game play. However, we verify that these differences are negligible and do not affect the conclusions derived from this study. Finally, the game resolution on the laptop is set to 1024×768 pixels, running in full-screen mode. Again, the conclusions derived from this setting also hold for other resolutions, as verified by our experiments with running Quake at 240×320 pixels on the PDA. To ensure that the game process is not preempted by other processes, it is set to the highest priority.

6.2.1 Laptop Settings

The laptop used for our experiments (with the 1400 MHz Intel Pentium Mobile processor) is equipped with Speedstep™ technology and had an ATI Radeon™ Mobility Video card. The processor supported five different operating points with clock frequencies of 1400, 1200, 1000, 800 and 600 MHz. For Pentium processors, the *RDTSC*[1] (read time-stamp counter) instruction is an excellent high-resolution, low-overhead mechanism to collect execution requirements of tasks in terms of processor cycles. All processor cycle counts on the laptop are measured using the *RDTSC* instruction that is inserted into the Quake II source code.

We set the processor frequency by calling Windows APIs. *ReadProcessorPwrScheme* and *WriteProcessorPwrScheme* are a pair of APIs to retrieve and write processor power policy settings for the specified power scheme. The change of frequency does not affect the current system power policy until *SetActivePwrScheme* is called with the index of this power scheme.

All the power measurements are conducted by connecting this laptop to a National

[1]Intel 64 and IA-32 Software Developer's Manual Vol 2B

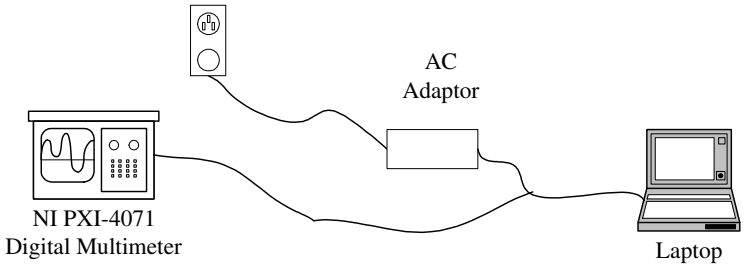


Figure 6.1: Power measurement on a laptop.

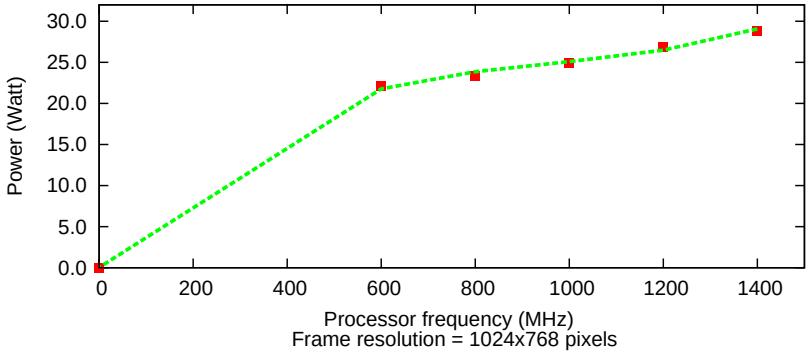


Figure 6.2: Processor frequency versus total system power consumption of the laptop.

Instruments PXI-4071 $7\frac{1}{2}$-digit Digital Multimeter. Figure 6.1 shows the setup used. The laptop battery is removed and the system runs on the external DC power adapter. The power cable of laptop is connected to the Multimeter. We collect the instantaneous current $c(t)$ and voltage $v(t)$ supplied every 5 ms. The average power consumptions over a duration of length T are calculated as $\sum_{t=0}^{T}(c(t)v(t)\delta t)/T$, where δt is the sampling interval (5 ms). Our estimated power consumptions therefore refer to the full system power and not that of the processor alone. Figure 6.2 shows the total system power consumption for the five different processor frequency levels on the laptop. Note that this varies between 28.8 Watts and 22.1 Watts, which correspond to the processor frequencies of 1400 MHz and 600 MHz respectively. Hence, the maximum possible reduction in power consumption is upper bounded by 23%.

6.2.2 PDA Settings

The PDA used for our experiments is a Dell Axim X51 with a 520 MHz Intel XScale PXA270 processor and 64MB SDRAM. The processor supports six different operating frequency points: 520, 416, 312, 208, 156 and 104 MHz. Unfortunately, the *RDTSC* instruction is not supported by XScale processors and processor cycle counts cannot be read by application programs. Hence, we use high-resolution and low-overhead Windows APIs such as the *QueryPerformanceCounter* to retrieve the processor time associated with different tasks. However, the results returned by these APIs become unreliable especially when the operating frequency of the processor is changed at run-time. To avoid these problems we, in addition, conduct experiments using a discrete event simulator with the power characteristics measured from the PDA.

To estimate the power characteristics of the PDA, we measure the power consumption of its CPU-core by connecting an iWave prototype PDA board[2] to the National Instruments PXI-4071 $7\frac{1}{2}$-digit Digital Multimeter (as we did with the laptop). The iWave prototype board has the same processor as many regular PDAs (i.e. Intel XScale PXA270) (see Figure 6.3). Furthermore, each component on the board (e.g. CPU-core, LCD, wireless interface, etc.) can be hooked up to measuring instruments. Hence, the measured CPU-core power consumption of the iWave board is used to estimate the power characteristics of a regular PDA. Figure 6.5 shows the CPU-core power consumptions for the six different frequencies on the PDA. Note that the power consumption of the CPU-core varies between 0.4 to 0.13 Watt, corresponding to the frequency range 520 - 104 MHz. Therefore, the maximum possible reduction in power consumption is upper bounded by 68%. This clearly shows that DVS can achieve much better

[2] http://www.iwavesystems.com/

power savings when the CPU-core of the PDA is considered in isolation, compared to the maximum system-wide power savings that can be achieved for the laptop we experiment with (which, as mentioned above, is 23%).

Figure 6.3: iWave prototype PDA board.

6.3 Results on the Laptop

We define two quality metrics that have been motivated by a study in [12]. This study concluded that while frame rates higher than a pre-defined constant target frame rate do not improve the overall gaming experience, lower than target frame rates severely degrade the game quality. In our work, we define the target frame rate as Γ. Hence, each frame has to be processed within $1/\Gamma$th of a second, which is set as the *frame deadline*. The *actual workload* and *predicted workload* of frame i are defined as ω_i and

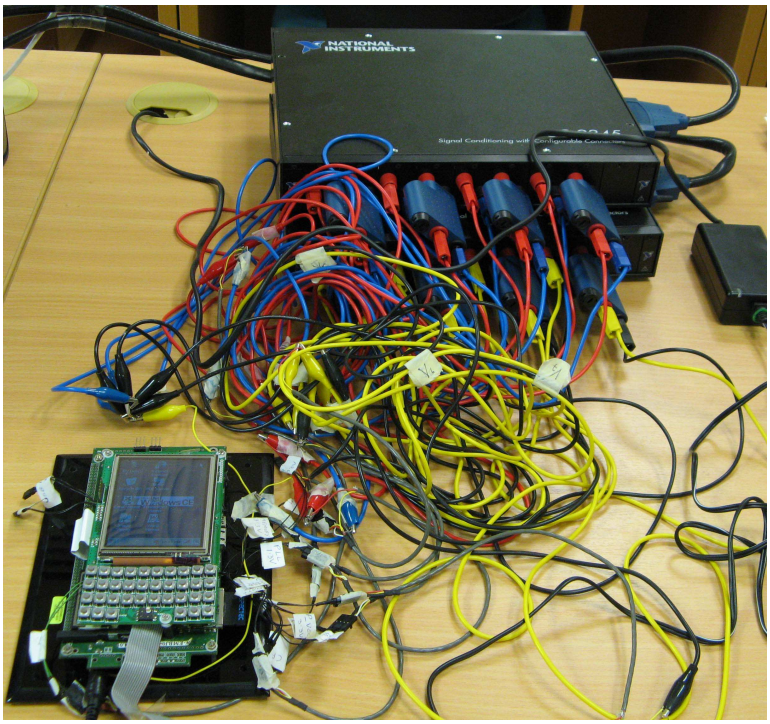


Figure 6.4: Power measurement on the iWave prototype PDA board.

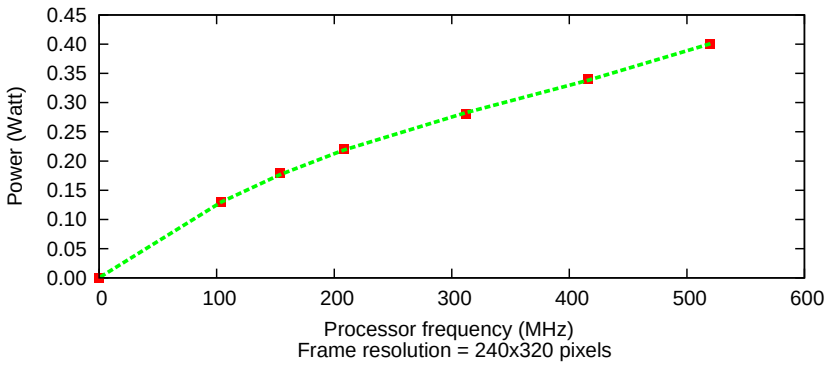


Figure 6.5: Processor frequency versus total system power consumption of the PDA.

$\bar{\omega_i}$ respectively. The required continuous processor frequency ζ_i is calculated by the formula $\zeta_i = \bar{\omega_i} \times \Gamma$. In the *simu-cont* setting, the resultant frame rate γ_i is calculated by the formula $\gamma_i = \frac{\zeta_i}{\omega_i}$. While in the *simu-disc* setting and the real platforms, the continuous frequency has to be mapped to the next available discrete frequency $\bar{\zeta_i}$. By scaling the processor frequency to the mapped discrete frequency $\bar{\zeta_i}$, the resultant frame rate γ_i is calculated by the formula $\gamma_i = \frac{\bar{\zeta_i}}{\omega_i}$.

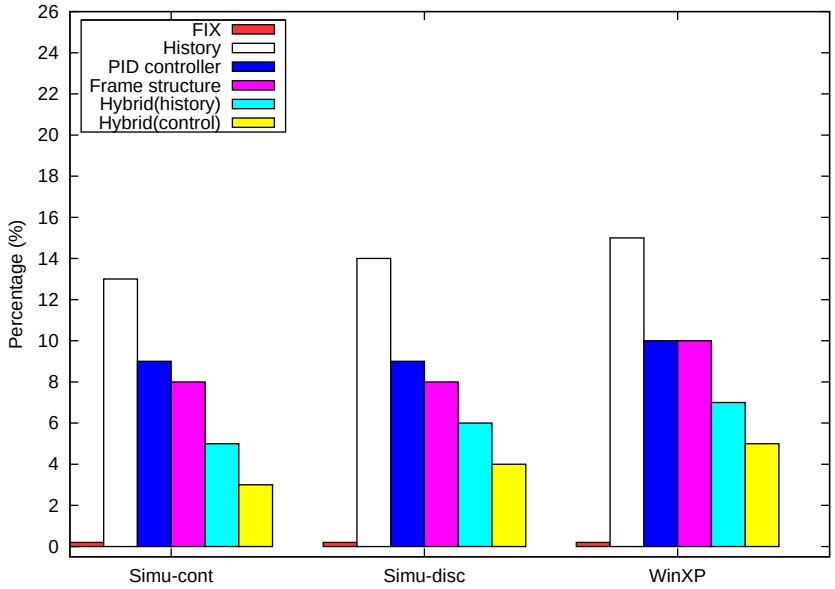
Our first metric of game quality only measures the percentage of frames that miss

their deadlines by the formula $perc = 100 \times \frac{\kappa}{N}$, where γ_1, γ_2, ... γ_κ are less than the target frame rate Γ; N is the total number of frames. The second metric also takes into account the magnitude of the missed deadlines (or the *tardiness*). It is computed by the formula $tardiness = 100 \times \frac{\sum |\Gamma - \gamma_i|/\Gamma}{N}$, where $i \in [1, \kappa]$.

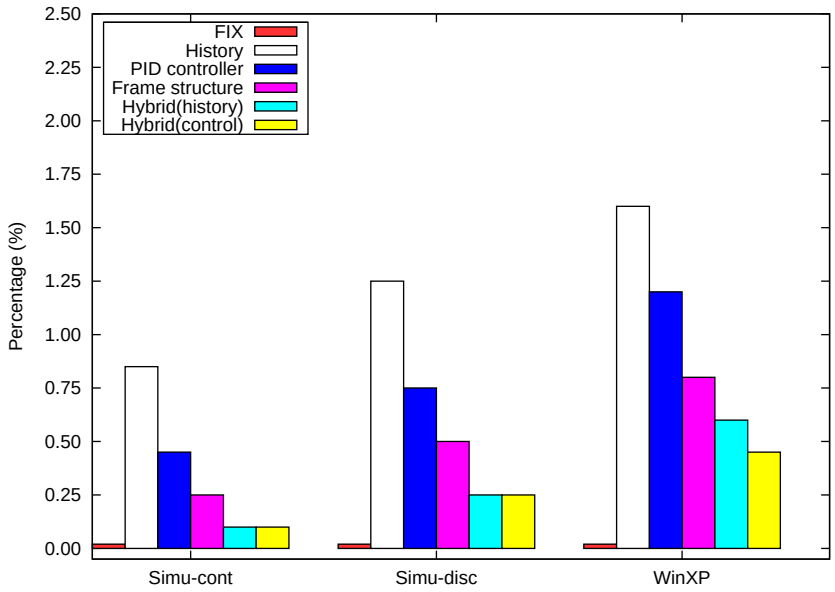
We compare the performance of different DVS schemes: `FIX` (where the processor is run at a constant frequency of 1400 MHz, i.e. no frequency scaling), `History` (DVS with a history-based predictor), `PID controller` (DVS with the PID controller-based predictor), `Frame structure` (DVS using the frame structure-based workload prediction scheme), `Hybrid(history)` (DVS with a hybrid combination of history-based and the frame structure-based predictors), and `Hybrid(control)` (DVS with our proposed hybrid combination of a PID controller-based predictor and a frame structure-based predictor). For all our experiments on the laptop, we set the target frame rate to 20 frames/second. Hence, each frame has to be processed within $1/20$th of a second, which is set as the *frame deadline*. We manually tune the PID controller parameters in `PID controller` and `Hybrid(control)` and obtain the best results with $K_p = 0.5$, $I = 28$, and $D = 0.00001$ on the laptop.

Clearly, the energy consumption during a game play and the quality of the game are mutually dependent on each other. Hence, to compare the different DVS schemes (i) we fix the energy consumption and then measure the output quality resulting from the different schemes, and (ii) we fix the output quality (e.g. *all* frames have to be processed within their pre-specified deadline, which is equal to $1/20$th of a second when the target frame rate is 20 frames/sec) and measure the energy consumption resulting from the different schemes.

In order to achieve the same energy consumption for each scheme, we selected the



(a) Percentage of frames with missed deadlines.



(b) Average tardiness of frames.

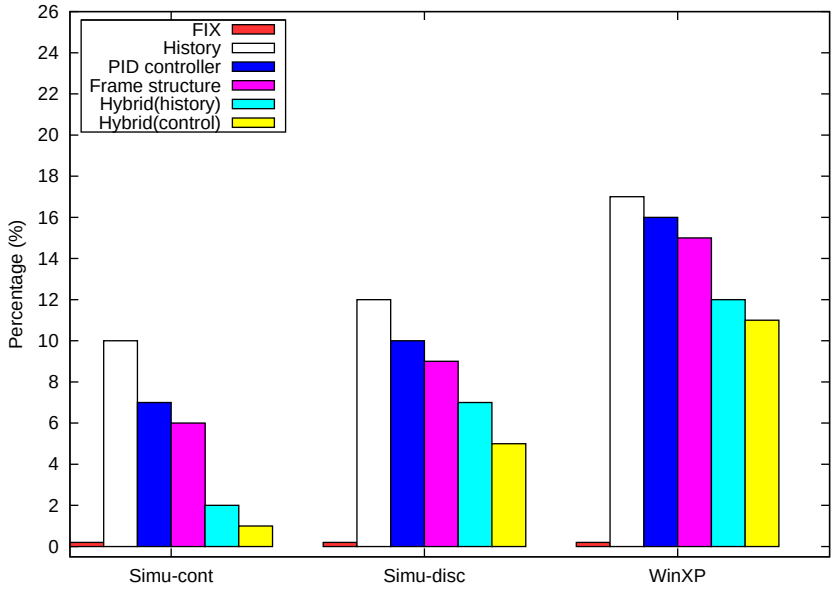
Figure 6.6: Comparison of game quality using different prediction schemes on a laptop running WinXP (with the target frame deadline set to $1/20$th of a second). The results were collected for a 4 second game play (88000 to 92000 millisecond), which was excerpted from a demo file in [43].

power of `History` as the baseline in each setting. If the power consumption of other scheme is higher than the baseline value, we tuned the pre-specified frame deadline to slow down the processor frequency. The lower frequency consumes the less power consumption, while the playing time for the same demo file keeps the same in game application. Therefore, the energy of playing the same demo file with other scheme is reduced. By this approach, we could guarantee the same power consumptions with different predictors in each setting.

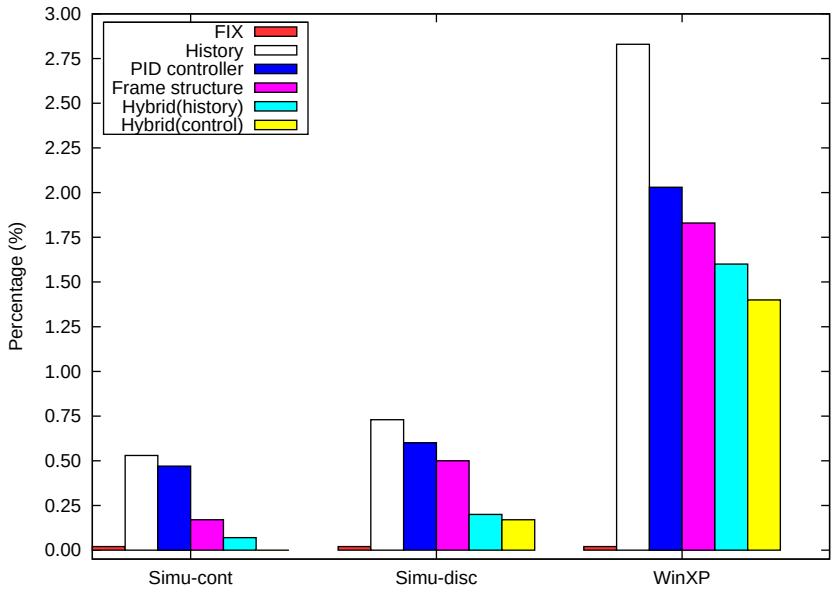
Similarly, in order to ensure none of frames misses its deadline, we tuned the pre-specified frame deadline in the experiments. If the percentage of frames missing their deadline is not zero, we tuned the pre-specified frame deadline to speed up the processor frequency. The higher frequency provides more processing cycles for game workload, thereby, reduces the percentage of frames missing their deadline.

Figure 6.6 shows the game quality for the different DVS schemes under the two metrics outlined above. From this figure, it may be noted that under the average tardiness metric, our proposed `Hybrid(control)` scheme results in more than 72% improvement over `History` on a laptop running Windows XP, for the same amount of energy consumption. The results under the simulation setting (with the PDA power characteristics) are even more attractive. In terms of power savings, compared to the `FIX` scheme, our proposed `Hybrid(control)` scheme achieves up to 22% power savings, where the upper bound on the savings, as mentioned before, is 23% on the laptop. Note that to match the target frame deadline, most of the frequencies computed for the estimated frame workload approach the lowest possible frequency (i.e. 600 MHz) on the laptop.

When the target frame deadline is reduced to $1/30$th of a second (i.e. 30 frames/sec),



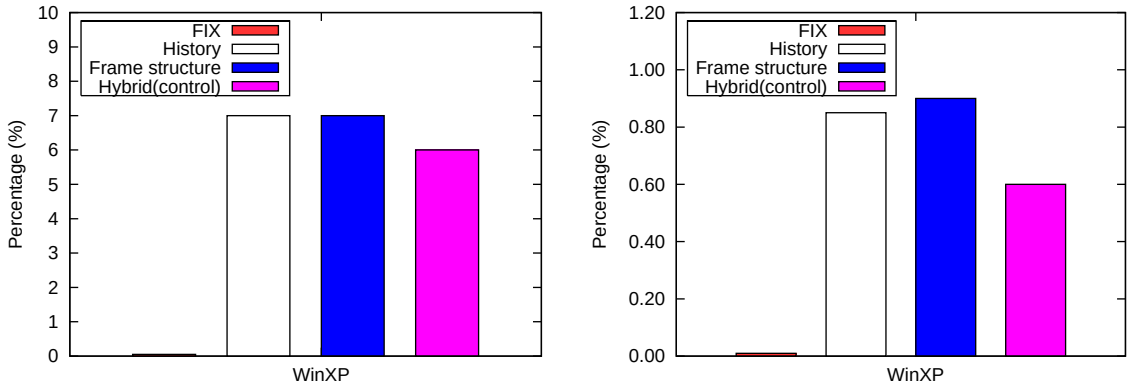
(a) Percentage of frames with missed deadlines.



(b) Average tardiness of frames.

Figure 6.7: Comparison of game quality using the different prediction schemes on a laptop running WinXP (with the target frame deadline set to $1/30$th of a second). The results were collected for a 4 second game play (88000 to 92000 millisecond), which was excerpted from a demo file in [43].

more processor cycles are required to speed up the game play. Therefore, the power savings from `Hybrid(control)` drops to 13% comparing with `FIX`. Finally, as shown in Figure 6.7, the game quality obtained using the `Hybrid(control)` is consistently better than that obtained using `History`.



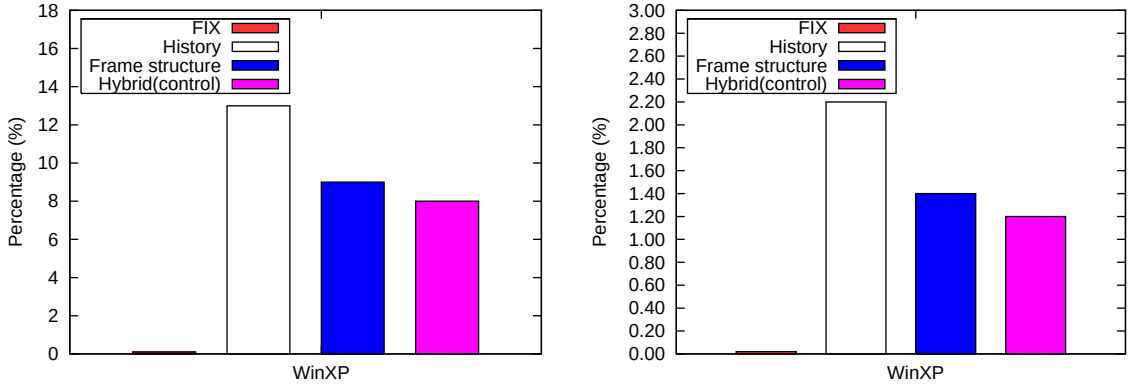
(a) Percentage of frames with missed deadlines.

(b) Average tardiness of frames.

Figure 6.8: Comparison of game quality for a 160 second demo file in [43] on a laptop running WinXP (with the target frame deadline set to $1/20$th of a second).

We also evaluate the power consumptions resulting from the different DVS algorithms on the laptop, when *all* the frames are required to meet their deadlines. We observe that the power savings of `Hybrid(control)` over `History` on the laptop are not as obvious as the improvements in quality as discussed above. This is primarily because we consider the power consumption of the entire laptop and not the CPU alone. As shown in Figure 6.2, only around 6% more power is consumed, even when the frequency is scaled to one higher level on the laptop.

Figure 6.8 shows the game quality for the 160 second demo file using `History` scheme, `Frame structure` scheme and `Hybrid(control)` scheme under the two metrics outlined above. From this figure, it may be noted that under the average tardiness metric, our proposed `Hybrid(control)` scheme results in more than 29% improvement over `History` on a laptop running Windows XP, for the same amount



(a) Percentage of frames with missed deadlines.

(b) Average tardiness of frames.

Figure 6.9: Comparison of game quality for a 160 second demo file in [43] on a laptop running WinXP (with the target frame deadline set to $1/30$th of a second).

of energy consumption. In terms of power savings, compared to the `FIX` scheme, our proposed `Hybrid(control)` scheme achieves up to 21% power savings, where the upper bound on the savings, as mentioned before, is 23% on the laptop.

When the target frame deadline is reduced to $1/30$th of a second (i.e. 30 frames/sec), more processor cycles are required to speed up the game play. Therefore, the power savings from `Hybrid(control)` drops to 12% comparing with `FIX`. Finally, as shown in Figure 6.9, the game quality obtained using the `Hybrid(control)` results in more than 46% improvement under the average tardiness metric, compared with `History`.

6.4 Results on the PDA

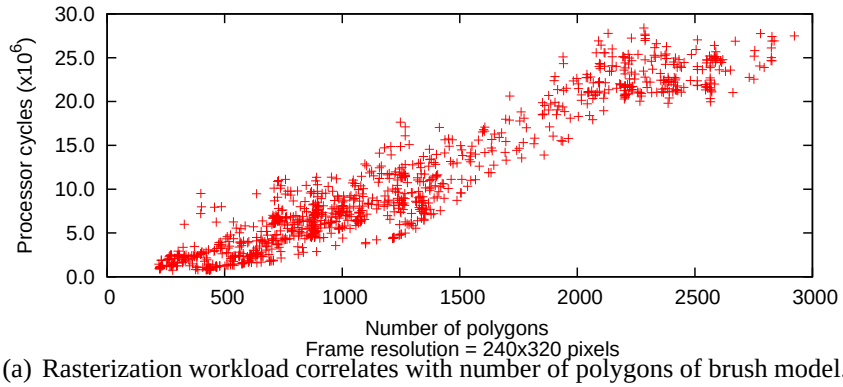
In this section, we discuss the applicability of proposed schemes on a different platform – a PDA. We investigate the framework of workload characterization (proposed in Chapter 4) on the PDA. The results imply that the proposed frame structure-based DVS scheme is extensible on the PDA. Furthermore, our experiments on the PDA show

that game frames exhibit a large degree of workload variability, which indicate that the proposed hybrid DVS scheme is also applicable on the PDA. By compared with the other predictors in prediction accuracy on the PDA, the hybrid DVS scheme achieves the best prediction. Finally, the results on the PDA demonstrate that our hybrid DVS scheme outperforms the other DVS schemes in terms of the game quality and the power saving.

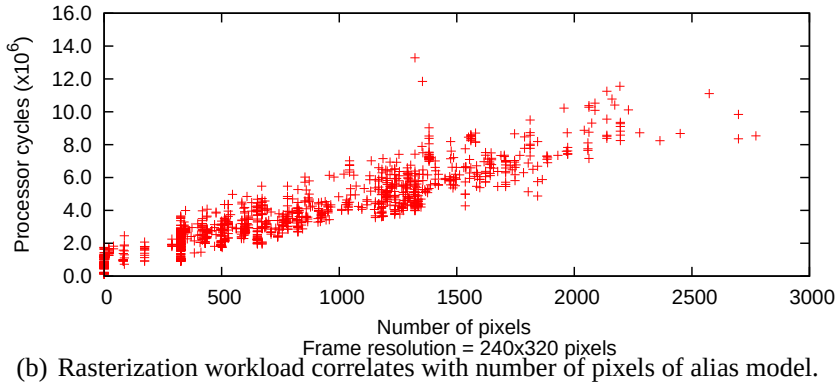
6.4.1 Workload Characterization

In the experiments on the PDA, we study Quake engine instead of Quake II. The reason is that the high computational workload generated by Quake II results in unacceptably low frame rates (around 5 frames per second) on the PDA, thereby deteriorating the game quality.

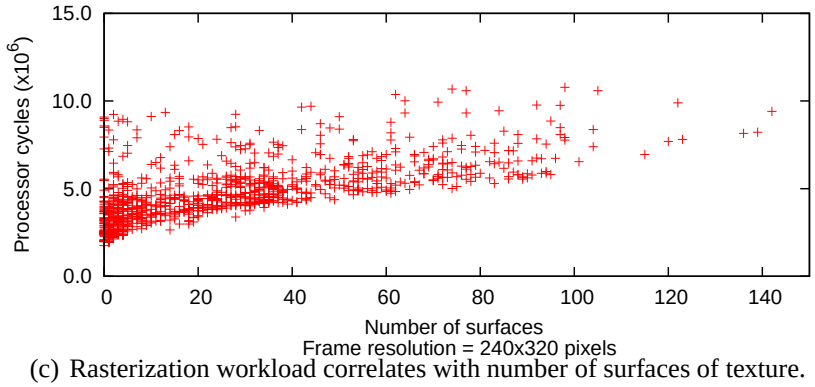
Fortunately, Quake engine is designed with similar architecture as Quake II, namely, it processes the representative primitives in the same approach. Therefore, we characterize the rasterization workload of individual primitives – brush model, alias model, texture and particles with their detailed constitution and observe the correlations between the constitution and the corresponding rasterization workload for each type of primitives, as shown in Figure 6.10. Figure 6.10(a) shows that the rasterization workload of brush model almost linearly scales to the number of constituent polygons. Figure 6.10(b) shows that the rasterization workload of alias model linearly scales to the number of pixels. Figure 6.10(c) shows that the rasterization workload of texture almost linearly scales to the number of constituent surfaces. Figure 6.10(d) shows that the rasterization workload of particles linearly scales to the number of pixels. Hence, the respective rasterization workload of each type of primitives could be derived from the involved detailed constitution.



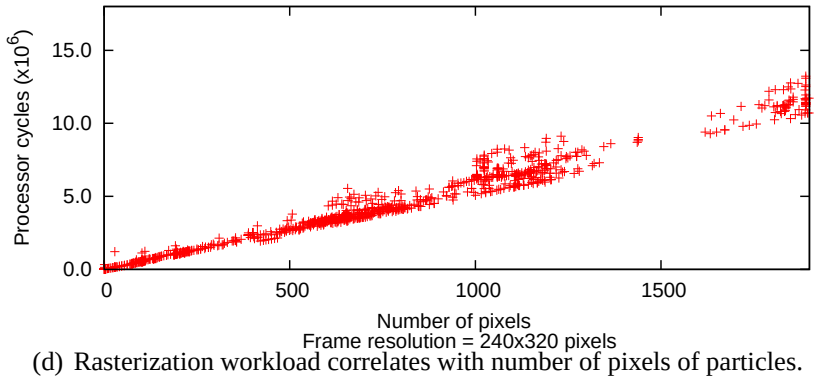
(a) Rasterization workload correlates with number of polygons of brush model.



(b) Rasterization workload correlates with number of pixels of alias model.



(c) Rasterization workload correlates with number of surfaces of texture.



(d) Rasterization workload correlates with number of pixels of particles.

Figure 6.10: Linear correlations of individual primitives - brush model, alias model, texture and particles on the PDA.

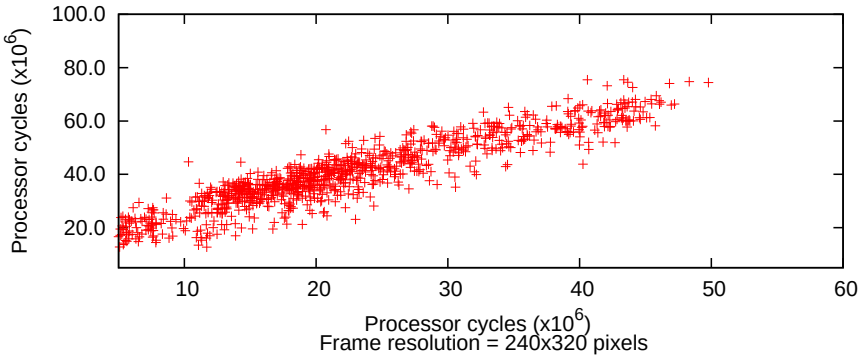


Figure 6.11: Linear correlation between rasterization and total processing workload on the PDA.

	$\mathbf{p_1}$	$\mathbf{p_2}$
Brush model	9016	0
Alias model	3686	7.334×10^5
Texture	4.291×10^4	3.522×10^6
Particles	6123	0
Game frame	1.154	1.451×10^7

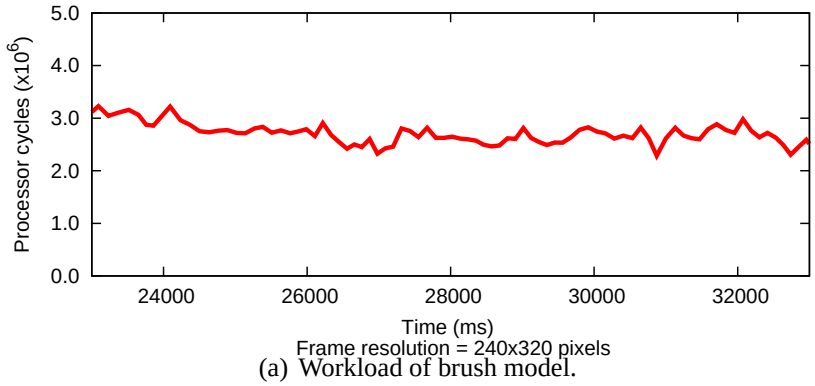
Table 6.1: Coefficients in the linear functions for Quake on the PDA.

Moreover, our experimental results suggest that the total workload generated from processing a frame is almost linearly correlated with its rasterization workload. Hence, we predict the total workload by estimating the rasterization workload of a frame. Figure 6.11 show the correlation between these two workload, with the horizontal axis denoting the rasterization workload.

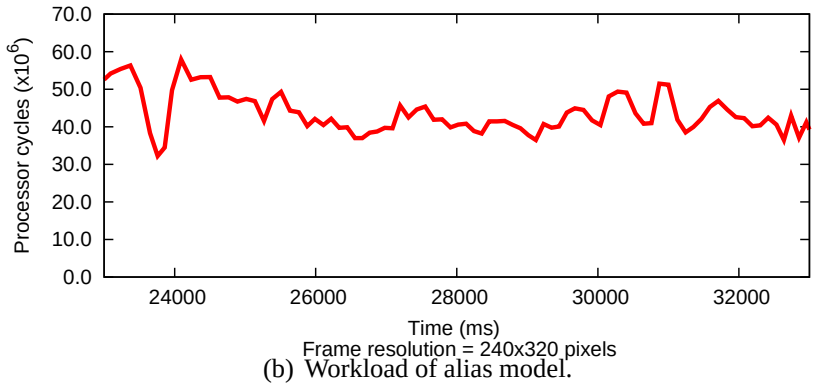
Similarly, we generate the coefficients of those linear correlations with linear regression models in Table 6.1.

6.4.2 Workload Variations

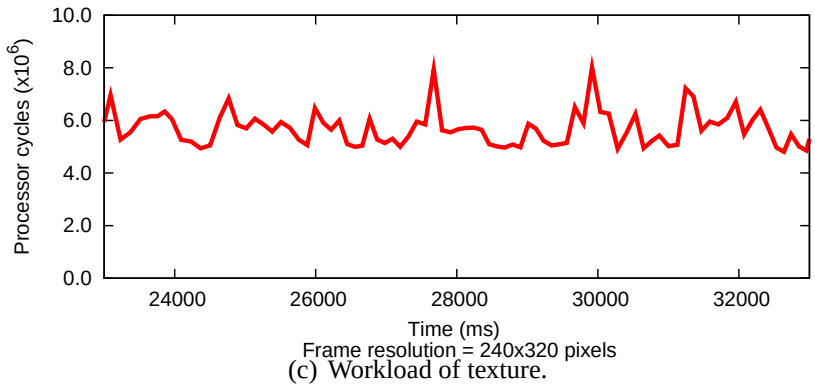
The game workload exhibits a large degree of variability on the laptop, as shown in Section 5.2. We observe the similar results on the PDA. Figure 6.12 illustrates the rasterization workload for each type of primitives exhibiting low variability. Figure 6.13



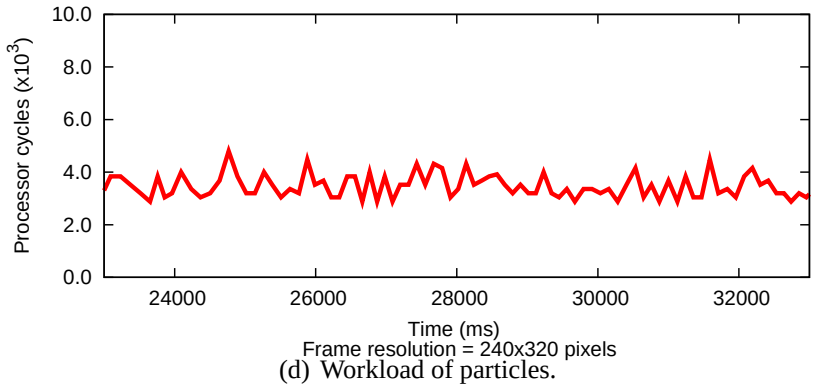
(a) Workload of brush model.



(b) Workload of alias model.

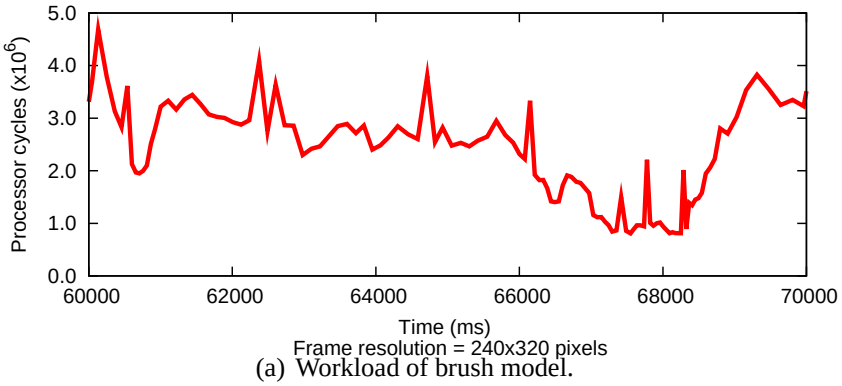


(c) Workload of texture.

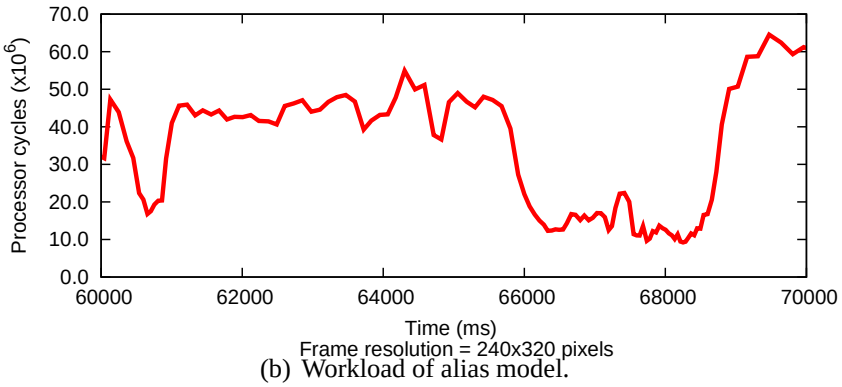


(d) Workload of particles.

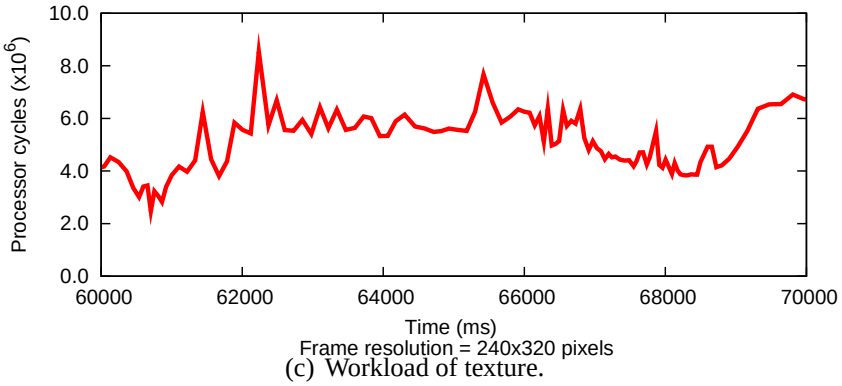
Figure 6.12: Rasterization workload exhibiting low variability for individual primitives - brush model, alias model, texture, particles on the PDA.



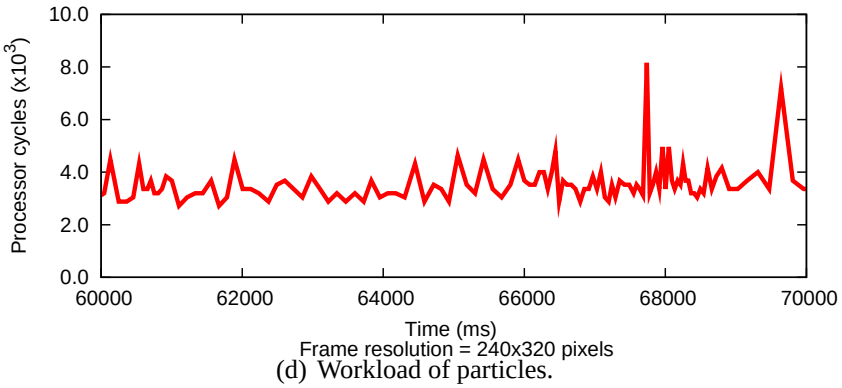
(a) Workload of brush model.



(b) Workload of alias model.



(c) Workload of texture.



(d) Workload of particles.

Figure 6.13: Rasterization workload exhibiting high variability for individual primitives - brush model, alias model, texture, particles on the PDA.

illustrates the rasterization workload for each type of primitives exhibiting high variability.

Figure 6.14 shows game workload varying from 50 to 75 million cycles per frame. In contrast, Figure 6.15 shows the fluctuant game workload varying from 20 to 90 million cycles per frame.

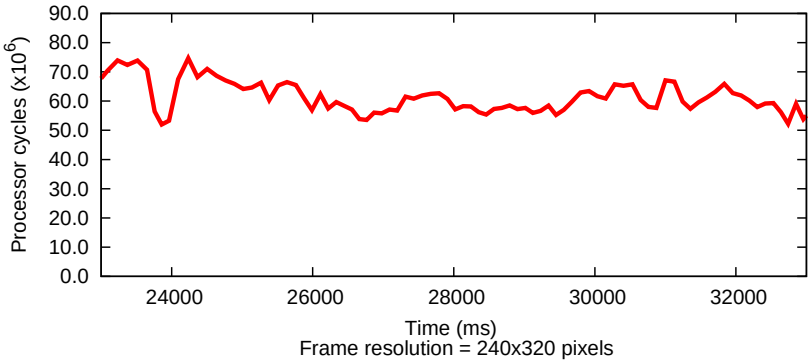


Figure 6.14: Processing workload exhibiting low variability on the PDA.

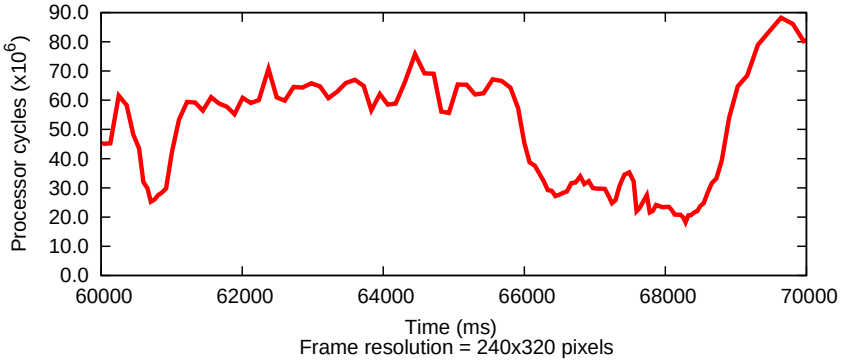


Figure 6.15: Processing workload exhibiting high variability on the PDA.

Such observations on the PDA are in line with those on the laptop, which motivates the hybrid DVS scheme.

6.4.3 Prediction Accuracy

In Section 5.2, we have conducted the experiments to compare the performance of prediction schemes on the laptop. The results show that for frame sequences which exhibit

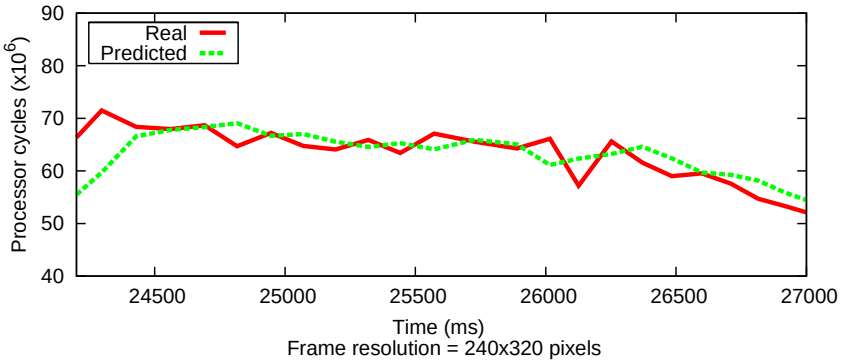


Figure 6.16: Workload prediction using `PID controller` scheme on the PDA, for a frame sequence exhibiting low workload variability.

low workload variation, the history-based, the PID controller-based predictors outperform the frame structure-based predictor. Our experiments on the PDA demonstrate the same results. Figure 6.16 and 6.17 show the comparisons of the PID controller-based and the frame structure-based predictors for a sequence of game frames with low variability respectively. Note that the PID controller-based prediction scheme surpasses the frame structure-based predictor.

Figure 6.18, 6.19 and 6.20 show the comparisons of the history, PID controller and frame structure-based predictors for a sequence of frames exhibiting high workload variability. It is easy to see that the frame structure-based predictor exceeds the first two. Figure 6.21 and 6.22 show the performance of two hybrid predictors, which match the profile of the actual frame workload more closely than the history, PID controller, or frame structure-based predictors. Here, the workload varies between 20 million processor cycles and 90 million cycles per frame. The absolute and relative errors in Figure 6.18 (i.e. using the `History` predictor) are 6.2 million cycles and 0.15 respectively. These errors drop to 2.5 million cycles and 0.06 respectively for the `Hybrid(control)` predictor (see Figure 6.22).

Figure 6.23 summarizes the absolute and relative prediction errors with different

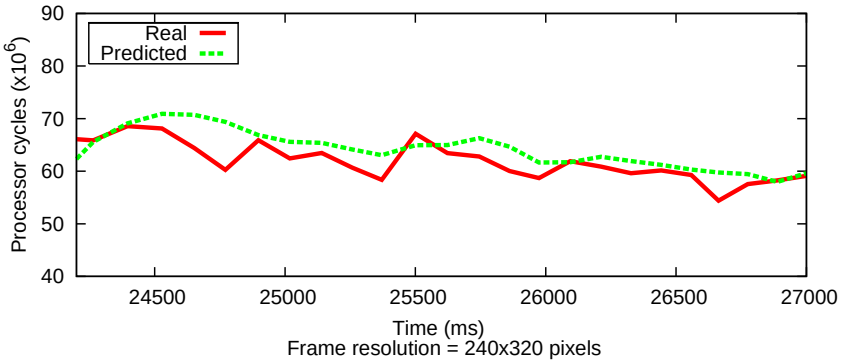


Figure 6.17: Workload prediction using `Frame structure` scheme on the PDA, for a frame sequence exhibiting low workload variability.

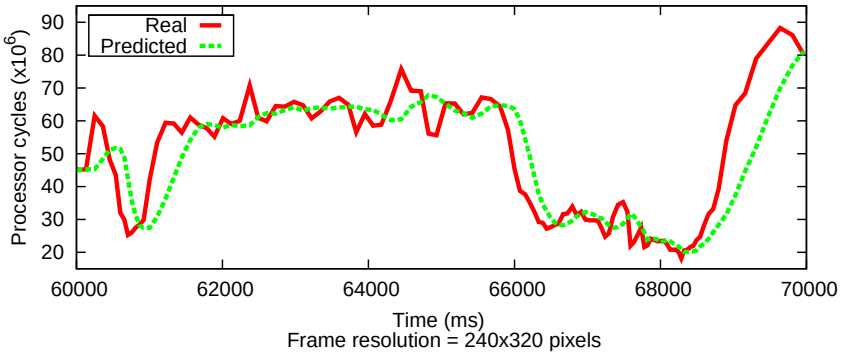


Figure 6.18: Workload prediction using `History` scheme on the PDA, for a frame sequence exhibiting high workload variability.

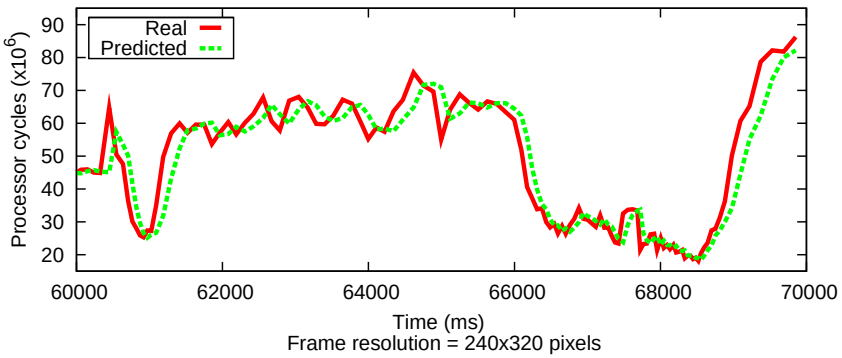


Figure 6.19: Workload prediction using `PID controller` scheme on the PDA, for a frame sequence exhibiting high workload variability.

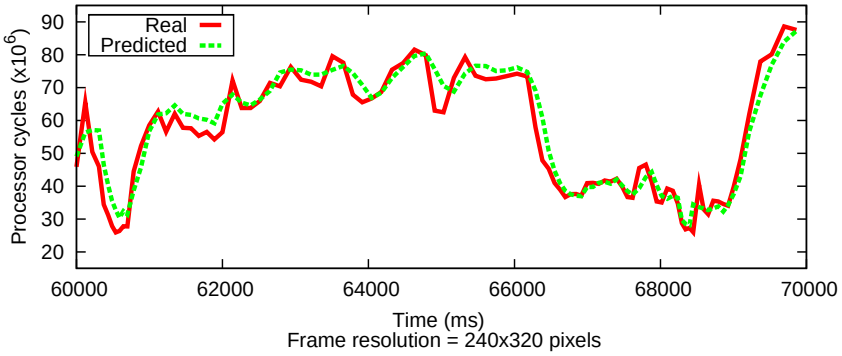


Figure 6.20: Workload prediction using `Frame structure` scheme on the PDA, for a frame sequence exhibiting high workload variability.

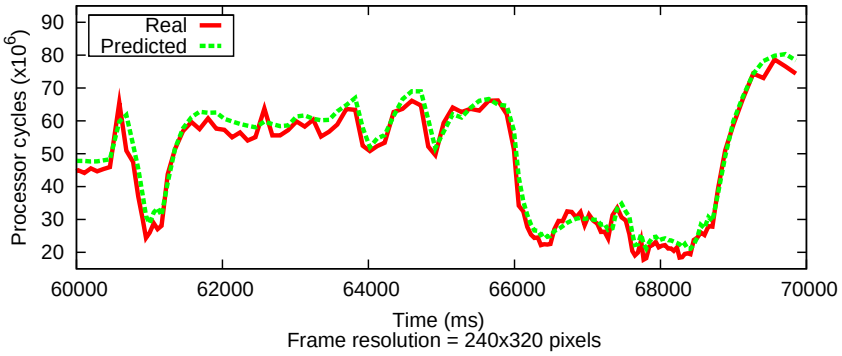


Figure 6.21: Workload prediction using `Hybrid(history)` scheme on the PDA, for a frame sequence exhibiting high workload variability.

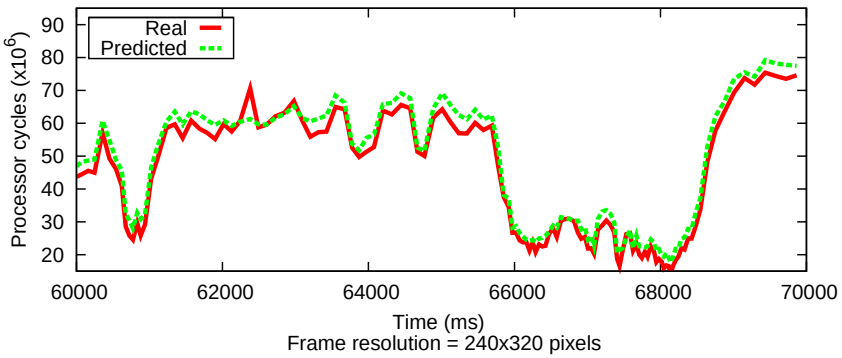
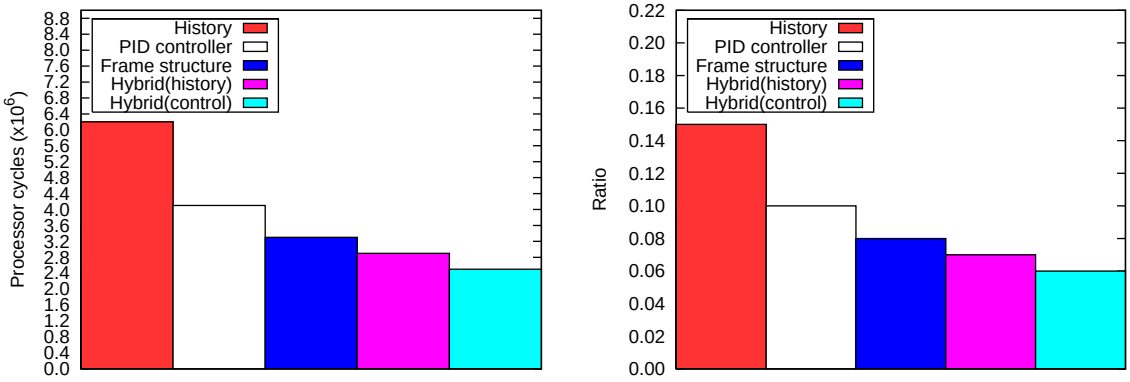


Figure 6.22: Workload prediction using `Hybrid(control)` scheme on the PDA, for a frame sequence exhibiting high workload variability.



(a) Absolute prediction errors.

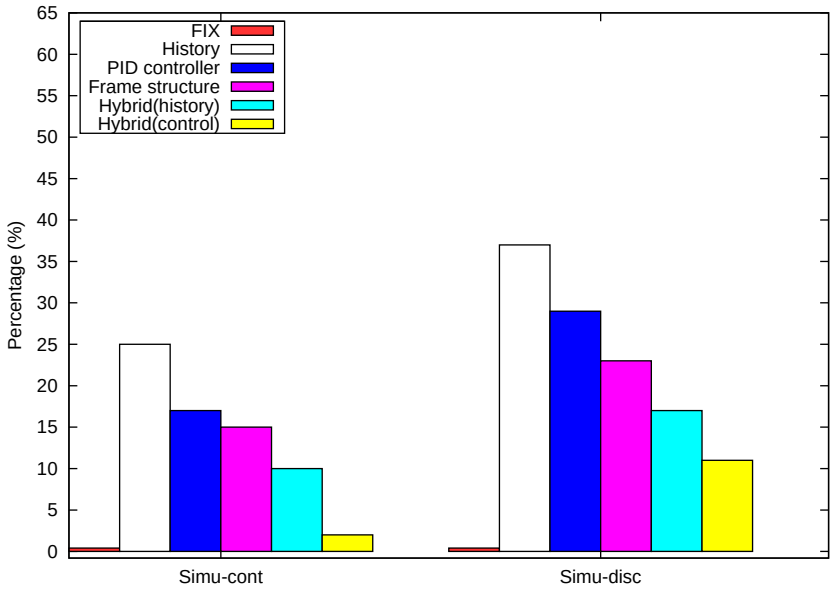
(b) Relative prediction errors.

Figure 6.23: Comparison of prediction errors with different predictors on the PDA. The results were collected for a 10 second game play, which was excerpted from a demo file in [44].

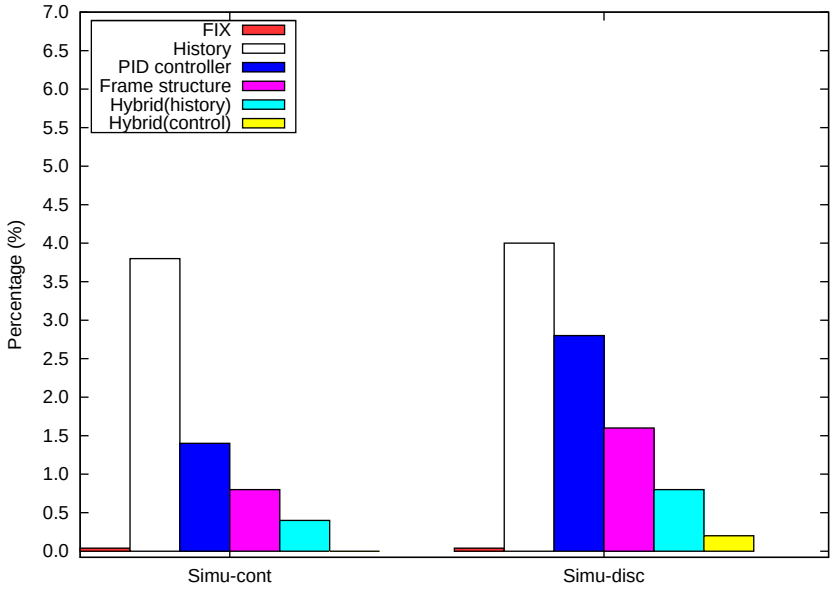
predictors on the PDA. Notice that on the PDA, our proposed `Hybrid(control)` predictor results in more than 60% improvement in prediction accuracy over a simple history-based workload predictor.

6.4.4 Performance of DVS Schemes

On the PDA, the best results are obtained with $K_p = 0.7$, $I = 50$, and $D = 0.00001$ for the target frame rate set to 5 frames/second. For the `FIX` scheme, the XScale processor is run at a constant frequency of 520 MHz. Figure 6.24 shows the game quality under the two metrics for the different DVS schemes on the PDA. Note that under the average tardiness metric, our proposed `Hybrid(control)` scheme leads to more than 100% quality improvement over `History` with the *simu-cont* setting for the same power consumption. With the *simu-disc* setting, this drops to 95% improvement. Finally, compared to `FIX`, our scheme yields more than 35% and 25% improvements in power savings with the *simu-cont* and *simu-disc* settings, where the upper bound on the savings, as mentioned before, is 68%.



(a) Percentage of frames which missed their deadlines.



(b) Average tardiness of frames.

Figure 6.24: Comparison of game quality using different prediction schemes on a PDA (with the target frame deadline set to $1/5$th of a second). The results were collected for a 10 second game play, which was excerpted from a demo file in [44].

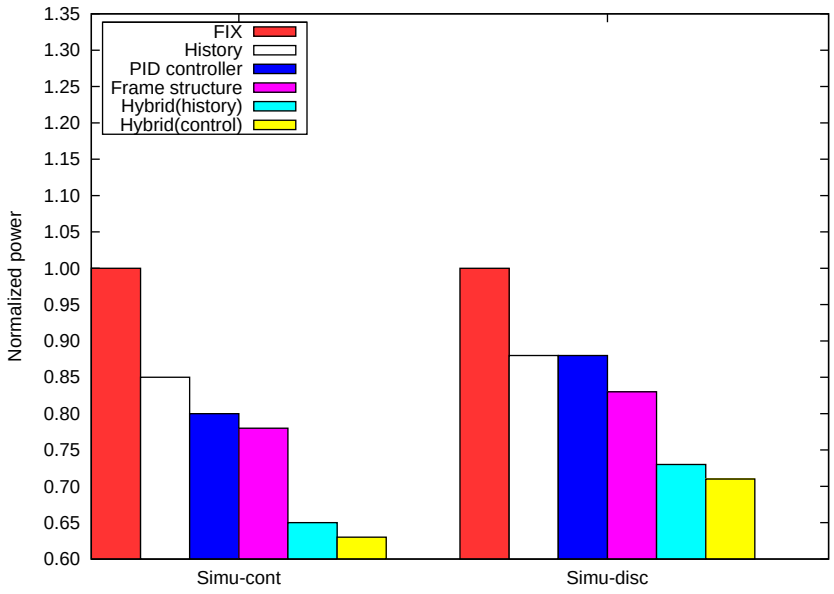


Figure 6.25: Normalized power consumption using the different prediction schemes against `FIX` as a baseline on the PDA. The results were collected for a 10 second game play, which was excerpted from a demo file in [44].

As shown in Figure 6.5, the CPU-core power consumption of each scheme is normalized against `FIX` as a baseline. It exhibits a relatively large change as the processor frequency is scaled. Therefore, the power savings of `Hybrid(control)` over `History` on the PDA are more significant than those on the laptop, when no frames are allowed to miss their predefined deadlines. Figure 6.25 shows the CPU-core power consumptions resulting from the different DVS algorithms using the simulation platform with PDA power characteristics. Here, `Hybrid(control)` achieves 26% and 19% more CPU-core power savings than `History` under the *simu-cont* and *simu-disc* settings respectively. *simu-cont* has more savings than *simu-disc* since we assume continuous frequency scaling and no frequency transition overhead (i.e. it represents an ideal case, which results in an upper bound on the energy savings).

6.5 Summary

In this chapter, we compared the performance of our proposed DVS schemes with known history-based algorithms developed for video decoding applications, on the simulators and the real platforms. Our results indicate that there are significant improvements of our proposed hybrid DVS scheme over the known algorithms in the game quality. At the same time, it saves considerable power consumption, compared with a scheme running the applications at a constant and full frequency.

Along with the evaluation on the laptop, we conducted the experiments on the different platform – the PDA. We characterized the workload primitives with their detail constitution, i.e. the number of constituting polygons of brush model, the number of constituting pixels of alias model, the number of pixels of texture, the number of constituting pixels of particles. We observed that there are linear correlations between workload primitives and their constitution on the PDA, even the resolution of the PDA is different from the laptop. Therefore, those correlations are exploited to predict the rasterization workload from the detailed constitution. By parsing game frame, we obtained the constituent primitives, thereby, determined frame workload on the PDA.

Moreover, we observed the frame workload on the PDA exhibits a large degree of variability, which explains why the proposed hybrid DVS scheme provides the best prediction, compared with other predictors.

On the PDA facilitated with the different microarchitecture, the different operating system and the different game resolution, we observed the consistent results of the proposed DVS schemes on the PDA. Next, we further evaluate their performance in game quality and power saving on the PDA. Even further improvement in the power saving is achieved on the PDA, as we considered the power consumption of the CPU-

core of the PDA in isolation, compared with the power consumption of the entire system of the laptop.

Chapter 7

Concluding Remarks

In this thesis, we looked into several issues related to power management for interactive 3D games on portable devices. The power management techniques have already been widely studied in the context of video decoding applications, which are computationally expensive. However, these studies have mostly focused on scaling voltage/frequency of a processor by predicting the workload associated with processing a video frame from the workload of the previously decoded frames. The power management techniques for interactive 3D games, on the other hand, exploit the unique natures of game applications and prolong game play with guaranteed game quality on portable devices. In this context, the main results that we have obtained in this thesis can be summarized as follows.

- We presented experimental results to identify the workload primitives, e.g. brush model, alias model, texture and particles, which predominantly contribute to the rasterization workload of the First-Person-Shooter game – Quake. Since Quake belongs to the genre of fast-pace action game without much artificial intelligence, the computation workload is relatively smaller than the rasterization workload.

We observed that the rasterization workload of each primitive varies accordingly to the detailed constitution, i.e. the constituent polygons of brush model, the constituent pixels of alias model, the constituent surfaces of texture and the constituent pixels of particles. Therefore, we identified that there is correlation between the detailed constitution and the rasterization workload of each type of primitives. Those correlations are exploited to predict the rasterization workload from the detailed constitution in our proposed DVS schemes. By parsing game frame, we obtained the constituent primitives, thereby, determined frame workload.

- For the frame sequences exhibiting fluctuant workload, we proposed the *frame structure-based DVS scheme* by exploiting the observed "structure" information of game frames. In contrast, we introduced the *PID controller-based DVS scheme* for the frame sequences exhibiting flat workload, where this DVS scheme achieves better prediction than the frame structure-based DVS scheme.

Motivated by the observation that game workload exhibits a large degree of variability, i.e. frame sequences with flat workload and frame sequences with fluctuant workload, we presented the *hybrid DVS scheme* by switching between two techniques: (i) adjusting workload prediction by the PID controller-based mechanism (viz. PID controller), and (ii) analyzing the graphics objects in current game scene (viz. the frame structure scheme).

- To investigate the influences of frequency mapping and frequency transition overhead to the DVS schemes, we designed two simulators. The performance of our proposed DVS schemes were evaluated on the simulators and the real platforms. From the experiments, we observed that the hybrid DVS achieves significant improvement in

the game quality, compared with known history-based DVS algorithms. At the same time, it saves considerable power consumption, compared with a scheme running the applications at a constant and full frequency.

The proposed DVS schemes were extended from the laptop to the different platform – the PDA. With the different microarchitecture of underlying processor and the different operating system on the PDA, we observed that the results on the PDA are consistent with those obtained on the laptop. Moreover, we observed further improvement in the power saving on the PDA, as the CPU-core of the PDA is considered in isolation, compared to the maximum system-wide power savings that can be achieved for the laptop.

All of the above results are derived from the data on the full-blown game engines on the real platforms, which are interesting to system-level design of power management techniques on mobile devices.

7.1 Future Work

Our work also gives rise to several open issues. A few of these have been listed below.

- The DVS schemes we proposed in the thesis consider power consumption of a processor alone. As we know, there are other components (e.g. wireless interface, LCD, backlight, etc.), which drain the battery largely on a portable device. For example, in the context of multi-player online games, the wireless interface keeps signalling communication even when no real game data is transmitted. In such scenarios, it is interesting to design intelligent power management techniques for the wireless interface. Further work could be extended to integrate the power management techniques of the processor

with the wireless interface or other components, such that the entire system consumes as less power as possible for interactive game applications.

- To provide content-rich graphics applications like 3D games on portable devices, the design community is moving towards multiprocessors to leverage application parallelism for higher performance. In [36, 37], they simulated the graphics rendering pipeline with three processor elements and discussed the quality factors such as level of detail and the resolution in graphics application. However, to date, no power management techniques have been proposed for game application on multiprocessor architecture. The possible future work in power-aware gaming on multiprocessor architecture should partition whole game pipeline into different parts according to different workload characteristics of each partition. Each partition should be mapped to individual processor and be executed in parallel or in sequence due to the data dependency. As the consequence of partitioning games on multiple processors, it will be important to design power management techniques for individual processor and to adapt bus frequency based on partition traffic so that the total on-chip power consumption is minimized with guaranteed game quality.

Bibliography

[1] A. Acquaviva, L. Benini, and B. Riccó. An adaptive algorithm for low-power streaming multimedia processing. In *Proc. 2001 Design, Automation and Test in Europe (DATE)*, pages 273 – 279, Munich, Germany, March 2001. ACM Press.

[2] T. Akenine-Moller and E. Haines. *Real-time Rendering*. A K Peters, Ltd, 2002.

[3] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez. Dynamic and agressive scheduling techniques for power-aware real-time systems. In *Proc. 22nd IEEE Real-Time Systems Symposium (RTSS)*, pages 95–105, London, UK, December 2001. IEEE Press.

[4] H. Aydin, R. Melhem, D. Mosse, and P. Mejia-Alvarez. Power-aware scheduling for periodic real-time tasks. *IEEE Trans. Comput.*, 53(5):584–600, 2004.

[5] A. C. Bavier, A. B. Montz, and L. L. Peterson. Predicting MPEG execution times. In *Proc. 1998 SIGMETRICS*, pages 131 – 140, Madison, Wisconsin, USA, June 1998. ACM Press.

[6] L. Bishop, D. Eberly, T. Whitted, M. Finch, and M. Shantz. Designing a PC game engine. *IEEE Computer Graphics and Applications*, 18(1), 1998.

[7] T. D. Burd, T. Pering, A. Stratakos, and R. W. Brodersen. A dynamic voltage scaled microprocessor system. *IEEE Journal of Solid-State Circuit*, 35(11):1571.

[8] A. Chandrakasan, V. Gutnik, and T. Xanthopoulos. Data driven signal processing: an approach for energy efficient computing. In *Proc. 1996 International Symposium on Low Power Electronics and Design (ISLPED)*, pages 347 – 352, Monterey, CA, USA, August 1996. IEEE Press.

[9] L. H. Chandrasena, P. Chandrasena, and M. J. Liebelt. An energy efficient rate selection algorithm for voltage quantized dynamic voltage scaling. In *Proc. 14th International Symposium on Systems Synthesis (ISSS)*, pages 124 – 129, Montreal, Canada, October 2001. IEEE Press.

[10] W.-H. Chen, C. Smith, and S. Fralick. A fast computational algorithm for the discrete cosine transform. *IEEE Transactions on Communications*, 25(9):1004–1009, 1977.

[11] K. Choi, K. Dantu, W.-C. Cheng, and M. Pedram. Frame-based dynamic voltage and frequency scaling for a MPEG decoder. In *Proc. 2002 International Conference on Computer-Aided Design (ICCAD)*, pages 732 – 737, San Jose, California, USA, November 2002. ACM Press.

[12] M. Claypool, K. Claypool, and F. Damaa. The effects of frame rate and resolution on users playing First Person Shooter games. In *Proc. 2006 Multimedia Computing and Networking Conference (MMCN)*, San Jose, CA, USA, June 2006. ACM Press.

[13] A. Dudani, F. Mueller, and Y. Zhu. Energy-conserving feedback EDF scheduling for embedded systems with real-time constraints. In *Proc. 2002 Joint Con-*

ference on Languages, Compilers, and Tools for Embedded Systems and Software and Compilers for Embedded Systems (LCTES-SCOPES), pages 213 – 222, Berlin,Germany, June 2002. ACM Press.

[14] E. Feig and S.Winograd. Fast algorithms for the discrete cosine transform. *IEEE Transactions on Signal Processing*, 40(9):2174–2193, 1992.

[15] K. Govil, E. Chan, and H. Wasserman. Comparing algorithms for dynamic speed-setting of a low power CPU. In *Mobile Computing and Networking (MobiCom)*, pages 13 – 25, Berkeley, CA, USA, November 1995. ACM Press.

[16] F. Gruian. Hard real-time scheduling for low energy using stochastic data and DVS processors. In *Proc. 2001 International Symposium on Low-Power Electronics and Design (ISLPED)*, pages 46 – 51, Huntington Beach, CA, USA, August 2001. ACM Press.

[17] F. Gruian and K. Kuchcinski. LEneS: task scheduling for low-energy systems using variable voltage processors. In *Proc. 2001 Asia and South Pacific Design Automation Conference ((ASP-DAC)*, pages 449 – 455, Yokohama, Japan, January 2001. ACM Press.

[18] D. Grunwald, C. B. Morrey, P. Levis, M. Neufeld, and K. Farkas. Policies for dynamic clock scheduling. In *Proc. 4th Symposium on Operating Systems Design and Implementation (OSDI)*, pages 73–86, San Diego, CA, USA, October 2000. USENIX Association.

[19] Y. Gu. Power-aware gaming on portable devices. In *SIGDA Ph.D. Forum at Design Automation Conference (DAC)*, San Diego, CA, USA, June 2007.

[20] Y. Gu and S. Chakraborty. Control theory-based DVS for interactive 3D games. In *Proc. 2008 Design Automation Conference (DAC)*, Anaheim, CA, USA, June 2008.

[21] Y. Gu and S. Chakraborty. A hybrid DVS scheme for interactive 3D games. In *Proc. 14th IEEE Real-Time Technology and Applications Symposium (RTAS)*, St. Louis, MO, USA, April 2008. IEEE Press.

[22] Y. Gu and S. Chakraborty. Power management of interactive 3D games using frame structures. In *Proc. 21st International Conference on VLSI Design (VLSID)*, pages 679–684, HICC, Hyderabad, India, Janauary 2008. IEEE Press.

[23] Y. Gu, S. Chakraborty, and W. T. Ooi. Games are up for DVFS. In *Proc. 2006 Design Automation Conference (DAC)*, pages 598 – 603, San Francisco, CA, USA, July 2006. ACM Press.

[24] Y. Huang, S. Chakraborty, and Y. Wang. Using offline bitstream analysis for power-aware video decoding in portable devices. In *Proc. 2005 ACM Multimedia (MM)*, pages 299 – 302, Singapore, November 2005. ACM Press.

[25] C. Huges, J. Srinivasan, and S. Adve. Saving energy with architectural and frequency adaptations for multimedia applications. In *Proc. 34th International Symposium on Microarchitecture*, pages 250 – 261, Austin, TX, USA, December 2001. IEEE Press.

[26] C. J. Hughes and S. V. Adve. A formal approach to frequent energy adaptations for multimedia applications. In *Proc. 31st International Symposium on Computer Architecture (ISCA)*, pages 138– 149, Munich, Germany, June 2004. IEEE Press.

[27] C. Im, S. Ha, and H. Kim. Dynamic voltage scheduling with buffers in low-power multimedia applications. *ACM Transactions on Embedded Computing Systems (TECS)*, 3(4):686–705, 2004.

[28] B. C. Kuo and F. Golnaraghi. *Automatic Control Systems*. Wiley, 2002.

[29] G. Lafruit, L. Nachtergaele, K. Denolf, and J. Bormans. 3D computational graceful degradation. In *Proc. 2000 International Symposium on Circuits and Systems (ISCAS)*, pages 547 – 550, Geneva, Switzerland, May 2000. IEEE Press.

[30] Y. Liu and A. K. Mok. An integrated approach for applying dynamic voltage scaling to hard real-time systems. In *Proc. 9th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 116 – 123, Toronto, Canada, May 2003. IEEE Press.

[31] J. R. Lorch and A. J. Smith. Improving dynamic voltage scaling algorithms with PACE. In *Proc. 2001 SIGMETRICS*, pages 50 – 61, Cambridge, Massachusetts, United States, June 2001. ACM Press.

[32] Z. Lu, J. Hein, M. Humphrey, M. Stan, J. Lach, and K. Skadron. Control-theoretic dynamic frequency and voltage scaling for multimedia workloads. In *Proc. 2002 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, pages 156 – 163, Greenoble, France, October 2002. ACM Press.

[33] Z. Lu, J. Lach, M. R. Stan, and K. Skadron. Reducing multimedia decode power using feedback control. In *Proc. 2003 International Conference on Computer Design (ICCD)*, pages 489– 496, San Jose, CA, USA, October 2003. IEEE Press.

[34] T. Mitra and T. Z. Chiueh. Dynamic 3D graphics workload characterization and

the architectural implications. In *Proc. 32nd International Symposium on Microarchitecture*, pages 62 – 71, Haifa, Israel, 1999. IEEE Press.

[35] A. Miyoshi, C. Lefurgy, E. V. Hensbergen, R. Rajamony, and R. Rajkumar. Critical power slope: Understanding the runtime effects of frequency scaling. In *Proc. 2002 International Conference on Supercomputing (ICS)*, pages 35 – 44, New York City, NY, USA, June 2002. ACM Press.

[36] B. Mochocki, K. Lahiri, and S. Cadambi. Power analysis of mobile 3D graphics. In *Proc. 2006 Design, Automation, and Test in Europe (DATE)*, pages 502 – 507, Leuven, Belgium, March 2006. European Design and Automation Association.

[37] B. Mochocki, K. Lahiri, S. Cadambi, and X. S. Hu. Signature-based workload estimation for mobile 3D graphics. In *Proc. 2006 Design Automation Conference (DAC)*, pages 592 – 597, San Francisco, CA, USA, July 2006. ACM Press.

[38] S. Mohapatra, R. Cornea, N. Dutt, A. Nicolau, and N. Venkatasubramanian. Integrated power management for video streaming to mobile handheld devices. In *Proc. 2003 ACM Multimedia (MM)*, pages 582 – 591, Berkeley, CA, USA, November 2003. ACM Press.

[39] T. Pering, T. Burd, and R. Brodersen. The simulation and evaluation of dynamic voltage scaling algorithms. In *Proc. 1998 International Symposium on Low Power Electronics and Design (ISLPED)*, pages 76 – 81, Monterey, CA, USA, August 1998. ACM Press.

[40] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proc. 2001 ACM Symposium on Operating Systems*

Principles (SOSP), pages 89 – 102, Chateau Lake Louise, Banff, Alberta, Canada, October 2001. ACM Press.

[41] C. Poellabauer, L. Singleton, and K. Schwan. Feedback-based dynamic frequency scaling for memory-bound real-time applications. In *Proc. 11th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 234 – 243, San Francisco, CA, USA, March 2005. IEEE Press.

[42] J. Pouwelse, K. Langendoen, R. Lagendijk, and H. Sips. Power-aware video decoding. In *Picture Coding Symposium*, 2001.

[43] Quake II demo file
http://www.comp.nus.edu.sg/∼guyan/massive1.1.dm2.

[44] Quake demo file
http://www.comp.nus.edu.sg/∼guyan/test.dem.

[45] G. Semeraro, G. Magklis, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas, and M. L. Scott. Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling. In *Proc. 2002 High-Performance Computer Architecture (HPCA)*, pages 29– 40, Boston, Massachusettes, USA, February 2002. IEEE Press.

[46] T. Simunic, L. Benini, A. Acquaviva, P. Glynn, and G. D. Micheli. Dynamic voltage scaling and power management for portable systems. In *Proc. 2001 Design Automation Conference (DAC)*, pages 524 – 529, Las Vegas, NV, USA, June 2001. ACM Press.

[47] J. A. Stankovic, C. Lu, S. H. Son, and G. Tao. The case for feedback control

real-time scheduling. In *Proc. 1999 Euromicro Conference on Real-Time Systems (ECRTS)*, pages 11 – 20, University of York, York, UK, June 1999. IEEE Press.

[48] N. Tack, F. Morán, G. Lafruit, and R. Lauwereins. 3D graphics rendering time modeling and control for mobile terminals. In *International Conference on 3D Web Technology*, 2004.

[49] A. Varma, B. Ganesh, M. Sen, S. R. Choudhury, L. Srinivasan, and J. Bruce. A control-theoretic approach to dynamic voltage scheduling. In *Proc. 2003 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, pages 255 – 266, San Jose, California, USA, October 2003. ACM Press.

[50] A. Watt and F. Policarpo. *3D Games: Real-time Rendering and Software Technology, Volume 1*. Addison-Wesley, 2001.

[51] M. Weiser, B. Welch, A. Demers, and S. Shenker. Scheduling for reduced CPU energy. In *Proc. 1994 Operating Systems Design Implementation (OSDI)*, pages 13–23, Monterey, CA, USA, November 1994. USENIX Association.

[52] M. Wimmer and P. Wonka. Rendering time estimation for real-time rendering. In *Eurographics Workshop on Rendering (EGWR)*, 2003.

[53] Q. Wu, P. Juang, M. Martonosi, and D. W. Clark. Formal online methods for voltage/frequency control in multiple clock domain microprocessors. In *Proc. 2004 International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 248 – 259, Boston, MA, USA, October 2004. ACM Press.

[54] Q. Wu, P. Juang, M. Martonosi, L.-S. Peh, and D. W. Clark. Formal control techniques for power-performance management. *IEEE Micro*, 25(5):52–62, 2005.

[55] W. Yuan and K. Nahrstedt. Energy-efficient soft real-time CPU scheduling for mobile multimedia systems. In *Proc. 2003 ACM Symposium on Operating Systems Principles (SOSP)*, pages 149 – 163, Bolton Landing, NY, USA, October 2003. ACM Press.

[56] W. Yuan and K. Nahrstedt. Practical voltage scaling for mobile multimedia devices. In *Proc. 2004 ACM Multimedia (MM)*, pages 924 – 931, New York, NY, USA, October 2004. ACM Press.

[57] Y. Zhang, Z. Lu, J. Lach, K. Skadron, and M. R. Stan. Optimal procrastinating voltage scheduling for hard real-time systems. In *Proc. 2005 Design Automation Conference (DAC)*, pages 905 – 908, San Diego, CA, USA, June 2005. ACM Press.

[58] Y. Zhu and F. Mueller. Feedback EDF scheduling exploiting dynamic voltage scaling. In *Proc. 10th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 84– 93, Toronto, Canada, May 2004. IEEE Press.