ISSN : 2278 – 1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 1, Issue 6, August 2012

Frame Based Prediction for Games in Mobile Devices Using Dynamic Voltage Scaling

[1]A.Sanjeevi Kumar, [2]Dr.A.Rajalingam and [3]B.Kokila

[1] Research Scholar CMJ University, Shillong, India.

[2]Associate Professor, Saveetha Engineering College, Chennai, India.

[3]Assistant Professor, Velammal Engineering College, Chennai, India.

ABSTRACT: Mobile devices have major problems with battery life. Dynamic voltage scaling technique is used to solve this problem with low power utilization .Most of the video games are implemented by higher video decoding .In this paper we are implementing frame based prediction algorithm for mobile devices games with different methods of DVS .The workload of application is divided and predicted by past history and every frames are executed with processor without affecting the quality of decoding and power consumption is also achieved.

Keywords: Low power VLSI, Dynamic Voltage Scaling, Computer Games, Workload Characterization, Power-aware Design, Prediction Algorithm.

I. INTRODUCTION

Computer games have recently experienced a sharp increase in popularity and have attracted considerable attention in both the industry and the 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 PDAs. Since such devices are becoming increasingly popular and powerful, this trend will certainly continue. Energy efficiency is one of the most critical issues in the design of such battery-powered portable devices. The availability of dynamic voltage scalable processors has lead to power management schemes for portable devices that are based on dynamic voltage scaling (DVS) algorithms. Since the power consumed by a processor depends linearly on its frequency and on the square of its operating voltage, DVS algorithms scale the operating frequency and voltage of the processor to match a varying computational workload as closely as possible. The showcase application for DVS algorithms so far has largely been video decoding [2],[4],[5],[6],[7] primarily because of two reasons: (i) video decoding applications are computationally expensive, and (ii) their workload exhibits high variability. These reasons make video decoding applications ideal candidates to illustrate the potential energy savings that may be achieved by DVS algorithms. A number of innovative DVS schemes have indeed been motivated by video decoding applications.

II. PROBLEM STATEMENT

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.

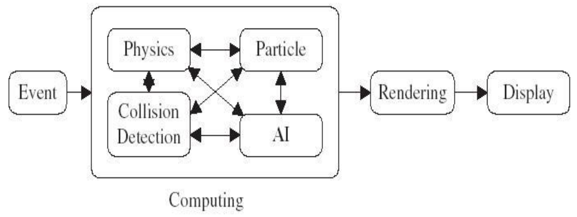


Figure 1: Frame *processing* in a game application.

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 step computing and rendering which we describe below. A more detailed discussion may be found in [1]. The computing step comprises tasks such as collision detection, AI, simulation of game physics and particle systems. Collision detection includes algorithms for checking

ISSN : 2278 – 1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 1, Issue 6, August 2012

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 are 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. 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. 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 rely on the use of a specialized processor called a Graphics Processing Unit (GPU). The rendering steps include the transformation of the vertices of solid objects to the screen space, deletion of invisible pixels by clipping, rasterization, deletion of occluded pixels, and interpolation of various parameters. The outcome of these steps is the transformation of the 3D data onto the 2D screen. Rendering is computationally expensive and occupies a significant fraction of the total processing time of a frame.

III. USING DVS ALGORITHM

DVS clearly indicates that by using DVS, the fluctuations in the frame rate can be reduced, thereby resulting in an acceptable perceptual quality and at the same time it reduces energy consumption. We construct the case for such a DVS scheme. Towards this, we start by presenting a framework for characterizing the workload of game applications. We then discuss how such a framework might be used to design DVS algorithms for games. Finally, we outline how such DVS algorithms would differ from those used for video decoding applications.

IV. DEAD BY DAWN GAME CHARACTERIZATION OF WORKLOAD

Figure 2: Dead by dawn game Frames

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 will primarily be concerned with this rasterization component of the rendering task, for reasons which we explain later in this section. Henceforth, we call the workload resulting from the rasterization task as the *rasterization workload.* When Dean By dawn uses its software renderer, all tasks including geometry processing, rasterization and texture processing are performed on the CPU. 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. Frames of game are shown in Figures 2. 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 [10]. 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 shown in Figure 1 involve traversing and manipulating the BSP tree. A game map is divided into convex regions, forming the leaves of the BSP tree. To render a game map, first 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 [7]. The bounding box of these leaves is 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 includes matrix transformations on the data and the rasterizing of a frame as 2D image onto the screen.

V. WORKLOAD AS A FUNCTION OF SCENE COMPLEXITY

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. The total workload involved in processing a frame also has a correspondence with the complexity of the frame. This correspondence can show how the total workload changes with time. Further, our measurements show that the rasterization workload constitutes approximately 38% of the total workload generated in processing a frame. From these two 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 paper 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 *objects* constituting the frame.

Brush Model:

A brush model is a 3D convex solid composed of polygons. Brush models are used to construct the geometry of a game map and they define the “world space” in which the player 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 collected the number of polygons constituting each brush model and the number of processor cycles involved in rasterizing them.

Alias Model:

Alias models are used to represent the different entities in Games (such as monsters, soldiers and weapons). Usually an Alias model consists of the geometry and the skin texture of the entity being modeled. The geometry in turn is composed of *triangles*. Since the rasterization of the triangles is done on the CPU instead of a graphics hardware, the number of *pixels* constituting each triangle affects the CPU workload. The software renderer renders the skin texture of an Alias model with two rendering *modes* called *opaque* and *alpha blend*. These modes call different functions and therefore incur different rasterization workloads For each mode, 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 the triangles constituting an Alias model. Let x denote the total number of pixels of an Alias model. Let $t = 0$ denote the case where Alias models with alpha blended texture are being used, and $t = 1$ denote the case where models with opaque texture are being used. To compute the rasterization workload of Alias models with different values of x and t, we capture all the models arising in different frames along with their rasterization workloads.

Texture:

Texture is the 2D image applied to the face of a brush model to give it the appearance of a real surface, examples of which are concrete slabs, brick walls and metal plates. A texture is typically composed of multiple *surfaces*. We therefore characterize the rasterization workload of a texture in terms of the number of surfaces constituting it. 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 this case we found that the rasterization workload increases almost linearly with the number of surfaces in a texture. Again, this function remains consistent across different game maps.

Light Map:

Light maps are used to store pre-calculated lighting information for different scenes in a game. Static light maps in Games 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.

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*. The number of *pixels* of the points generated in rasterization is used to parameterize the rasterization workload of particles.

This workload scales almost linearly with the number of pixels, as expected, and the scaling factor again remains consistent across game maps. The contributions of the abovementioned five types of objects to the rasterization workload are summarized in Figure 3 (the workload

ISSN : 2278 – 1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 1, Issue 6, August 2012

resulting from light maps is not shows for reasons already described).

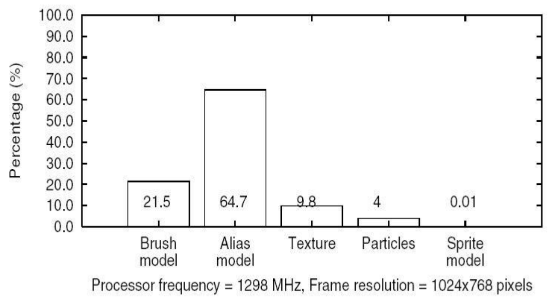


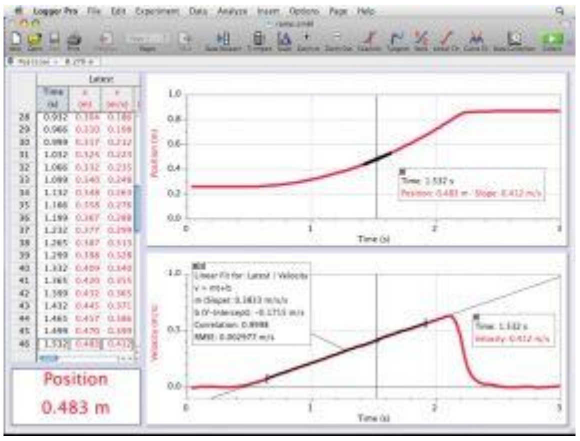
Figure 3: Contributions of the different objects in a frame towards the rasterization workload.

Rasterizing Alias models is clearly the most expensive. Lastly, note that apart from these five objects, *sprite 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.

VI. PREDICTING FRAME WORKLOAD FOR DVS

Most DVS algorithms targeted towards video decoding applications rely on predicting the processing workload of future frames or macroblocks and then adjusting the processor's operating voltage and frequency to match this workload as closely as possible. Such predictions are often based on the decoding times (or equivalently, processor cycle requirements) of previously decoded frames. We believe that DVS schemes for game applications would require fundamentally different approaches. More specifically, the workload prediction for a frame should *not* rely on the processing times of previous frames. Instead, the "structure" in the frame should be exploited to predict its workload. The framework for workload characterization that we presented in the previous subsection can be used towards this. Using this framework, the rasterization workload of a frame can be computed as the sum of the rasterization workloads of its constituent objects.

The computed workload can then be appropriately scaled to *predict* the total processing workload of the frame, which can be used to adjust the processor's voltage and frequency. While computing, or rather predicting, the rasterization workload of the different objects constituting a frame, several data structures or tables need to be created, as discussed in the previous subsection. An example of such a table is the workload of each (single) Alias model for different values of the parameters (*x, t*). Exactly how these tables are created, and more importantly how they are maintained or updated would depend on the specifics of the DVS scheme.



	Time (s)	x (m)	v (m/s)
28	0.932	0.304	0.185
29	0.966	0.310	0.198
30	0.999	0.317	0.212
31	1.032	0.324	0.223
32	1.066	0.332	0.235
33	1.099	0.340	0.248
34	1.132	0.348	0.263
35	1.166	0.358	0.276
36	1.199	0.367	0.288
37	1.232	0.377	0.299
38	1.265	0.387	0.315
39	1.299	0.398	0.328
40	1.332	0.409	0.340
41	1.365	0.420	0.355
42	1.399	0.432	0.365
43	1.432	0.445	0.371
44	1.465	0.457	0.386
45	1.498	0.470	0.389
46	1.532	0.483	0.412

Time: 1.532 s
Position: 0.483 m Slope: 0.412 m/s

Linear Fit for: Latest | Velocity
v = mt+b
m (Slope): 0.1613 m/s/s
b (Y-Intercept): -0.1715 m/s
Correlation: 0.8988
RMSE: 0.002977 m/s

Time: 1.532 s
Velocity: 0.412 m/s

Figure: 4 Prediction frames DVS Output

In contrast to such schemes, the only "structure" information that DVS algorithms for video decoding applications can use is whether the frame is of type I, B or P. Figure 4 observation points to the potentially greater energy savings that can be achieved in the case of game applications. Finally, we would like to conclude by pointing out that buffering technique to smooth out the variations in the decoding times of frames, which are widely used in video decoding applications, cannot be used for games due to their interactive nature.

V. CONCLUSION

This paper was concerned with building a case for DVS algorithms specifically targeted towards interactive graphics games. Our main contribution was a framework for characterizing the workload of game applications. We also outlined how the proposed workload characterization framework may be used to design concrete DVS algorithms and how such algorithms might differ from those used for video decoding applications. In this paper is to use the proposed framework to *predict* the processor cycle requirements of frames and use this prediction to scale the processor's voltage and frequency. However, more work needs to be done to *efficiently* maintain the various tables and compute/predict the frame workloads The workload of application is divided and predicted by past history and every frames are executed with processor without affecting the quality of decoding and power consumption is also achieved.

ISSN : 2278 – 1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 1, Issue 6, August 2012

REFERENCES

[1] L. Bishop, D. Eberly, T. Whitted, M. Finch, and M. Shantz. Designing a PC game engine. *IEEE Computer Graphics and Applications*, 18(1):46–53, 1998.

[2] K. Choi, K. Dantu, W.-C. Cheng, and M. Pedram, Frame-based dynamic voltage and frequency scaling for a MPEG decoder. In IEEE/ACM International Conference on Computer-aided Design (ICCAD), pages 732–737, San Jose, California, 2002.

[3] M. Claypool, K. Claypool, and F. Dama. The effects of frame rate and resolution on users playing First Person Shooter games. In *Multimedia Computing and Networking (MMCN) Conference*, San Jose, California, 2006.

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

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

[6] Z. Lu, J. Lach, M. R. Stan, and K. Skadron. Reducing multimedia decode power using feedback control. In *International Conference on Computer Design (ICCD)*, pages 489–497, San Jose, California, 2003.

[7] Yan Gu Samarjit Chakraborty Wei Tsang Ooi ,"Games are Up for DVFS" 2006

[8] http://www.lucidvisiongames.com

[9] Intel VTune Performance Analyzer http://www.intel.com/cd/software/products/asmona/ eng/vtune/vpa/index.htm.

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

Biography

Mr. A. SANJEEVI KUMAR working as Assistant Professor in department of Electronics and Communication Engineering Meenakshi Academy Of Higher Education And Research, Chennai, India. His current research activities pertain to design VLSI Signal processing.

Dr.A.Rajalingam received the Bachelor of Computer Science and Engineering degree from Anna University, Chennai, India. And he received Master of VLSI Design Engineering degree from Anna University, Chennai, India and he completed his Doctoral degree from the Bharath University Chennai, India. His current research activities pertain to design low power and high speed VLSI technologies. Presently he is working as Associate Professor in department of Electronics and Communication Engineering at Saveetha Engineering, Chennai, India.

Mrs.B.Kokila working as Assistant Professor in department of Computer science and Engineering, Velammal Engineering College, Chennai, India. Her current research activities pertain to design processor scheduling and Prediction algorithms.